Sławomir SAMOLEJ, Wojciech RZĄSA
Dariusz RZOŃCA, Jan SADOLEWSKI

Wprowadzenie do informatyki II – bezpieczeństwo systemów informatycznych, sieci komputerowe, systemy operacyjne i bazy danych

skrypt dla studentów
kierunków nieinformatycznych
na uczelniach technicznych

OFICYNA
WYDAWNICZA
POLITECHNIKI RZESZOWSKIEJ

Wydano za zgodą Rektora

O p i n i o d a w c a
dr hab. Marcin SZPYRKA, prof. AGH

R e d a k t o r
Marta JAGIEŁOWICZ

P r o j e k t o k ł a d k i
Joanna MIKUŁA

Matryce zostały przygotowane przez Autorów w programie LaTeX

Skrypt przeznaczony dla studentów kierunków nieinformatycznych
na uczelniach technicznych

bezpieczeństwo systemów informatycznych
sieci komputerowe
systemy operacyjne
bazy danych

p-ISBN 978-83-7199-984-4
e-ISBN 978-83-7934-755-1

Oficyna Wydawnicza Politechniki Rzeszowskiej
al. Powstańców Warszawy 12, 35-959 Rzeszów

Nakład 100 egz. Ark. wyd. 10,49. Ark. druk. 13,25. Papier offset. 70g B1.
Oddano do druku w grudniu 2016 r. Wydrukowano w grudniu 2016 r. Dodruk.
Drukarnia Oficyny Wydawniczej, al. Powstańców Warszawy 12, 35-959 Rzeszów
Zam. nr 130/16

Spis treści

Rozdział 1.

Wstęp

Oddajemy w ręce Czytelników drugi tom skryptu, w którym kontynuujemy przegląd głównych dziedzin współczesnej informatyki. Kolejnymi zagadnieniami, które zdecydowaliśmy się przybliżyć, są bezpieczeństwo systemów informatycznych, sieci komputerowe, systemy operacyjne oraz bazy danych. W naszym odczuciu są to obszary wiedzy, których znajomość jest kluczowa dla poznania pełnego ekosystemu, jaki tworzą składniki współczesnych systemów informatycznych. Programy przygotowywane na podstawie algorytmów w różnych językach programowania sterujące pracą systemów mikroprocesorowych (por. pierwszą część skryptu [40]) są najczęściej nadzorowane przez *systemy operacyjne*. Za komunikację pomiędzy systemami mikroprocesorowymi odpowiadają *sieci komputerowe*. Z kolei coraz powszechniejszymi składnikami programów i usług sieciowych są *bazy danych*. Tematyką łączącą wymienione działy współczesnej informatyki jest *bezpieczeństwo systemów informatycznych*. Techniki zapewnienia bezpieczeństwa w dostępie do zasobów systemu operacyjnego, bazy danych czy wskazania bezpiecznego sposobu wymiany informacji przez sieć komputerową stanowią jedną z najintensywniej rozwijanych gałęzi informatyki.

Adresatami niniejszego skryptu są wszystkie osoby pragnące usystematyzować swoją wiedzę z podstawowych działów informatyki na takim poziomie, aby mogły swobodnie komunikować się ze specjalistami informatykami. W kursie przyjęliśmy zasadę wyjaśniania poszczególnych zagadnień na przykładach, celowo pomijając omówienie bardziej ścisłych zasad matematycznych leżących u podstawy działania omawianej techniki komputerowej. Dobór materiału zawartego w skrypcie jest wynikiem naszych kilkunastoletnich doświadczeń w prowadzeniu wykładów, ćwiczeń audytoryjnych i laboratoryjnych z przedmiotu „informatyka" dla nieinformatycznych kierunków studiów technicznych.

Drugi tom skryptu podzieliliśmy na 6 rozdziałów. W rozdziale 2. wprowadziliśmy podstawowe pojęcia z dziedziny bezpieczeństwa systemów informatycznych. Wyjaśniliśmy w nim główne techniki kryptograficzne oraz ich wybrane zastosowania. W rozdziale 3. omówiliśmy zasady działania i techniki transmisji informacji z zastosowaniem sieci komputerowych. Jako szkielet wywodu przyjęliśmy war-

stwowy model sieci OSI. Opis technik komunikacyjnych uzupełniliśmy o zagadnienia bezpieczeństwa przesyłu danych przez sieć. Kluczowe zagadnienia dotyczące systemów operacyjnych przedstawiliśmy w rozdziale 4. Wprowadziliśmy pojęcia procesów, zasobów oraz technik ich zarządzania. Omówiliśmy również zasady zarządzania pamięcią operacyjną, systemem plików oraz podsystem zapewniający bezpieczeństwo systemu operacyjnego. Rozdział 5. poświęciliśmy technikom projektowania, programowania oraz zarządzania bazą danych. Wskazaliśmy tam również metody zapewnienia bezpieczeństwa baz. Rozdział 6. stanowi krótkie podsumowanie, w którym odsyłamy Czytelnika do innych publikacji zawierających poszerzone wiadomości z zagadnień przedstawionych w skrypcie.

Rozdział 2.

Bezpieczeństwo systemów informatycznych

Wojciech Rząsa, Dariusz Rzońca

Whatever you do will be insignificant, but it is very important that you do it.[1]
Mahatma Gandhi

2.1. Wprowadzenie

Zagadnienie zapewniania bezpieczeństwa informacji jest dość specyficzne. Od wieków podejmowano różne próby opracowania skutecznych mechanizmów zabezpieczających, ale większość z nich nie wytrzymała próby czasu. W kryptografii niewiele jest algorytmów, których bezpieczeństwa można dowieść (jak np. *szyfr z kluczem jednorazowym*), większość z nich jest uważana za bezpieczne, póki nie zostanie znaleziona luka. Rozwój technologiczny sprawiał, że wykrywano słabości wielu zabezpieczeń uważanych swego czasu za genialne i niemożliwe do złamania, od antycznego *szyfru Cezara*, przez wykorzystywaną podczas II wojny światowej Enigmę [13], do współczesnego algorytmu DES [8]. Wykrywanie wad poszczególnych rozwiązań stymulowało opracowanie nowych, skuteczniejszych metod. Mimo że obecny rozwój komputerów kwantowych może za pomocą algorytmu Shora służącego do faktoryzacji dużych liczb pierwszych znacznie osłabić obecne mechanizmy kryptografii asymetrycznej, przyczynia się także do rozwoju nowej dziedziny o ogromnych możliwościach, tj. kryptografii kwantowej. Parafrazując cytat będący mottem niniejszego rozdziału, można stwierdzić, że każde z zabezpieczeń, jakie zastosujemy, może się okazać niewystarczające, ale ważne

[1] *Cokolwiek zrobisz będzie nieznaczące, ale jest bardzo ważne, abyś to zrobił.*

jest, abyśmy je jednak zastosowali. Niniejszy rozdział ma na celu pobieżne przedstawienie wybranych elementarnych zagadnień związanych z bezpieczeństwem systemów informatycznych wspólnych dla różnych zastosowań. Aspekty ściśle związane z sieciami komputerowymi czy systemami operacyjnymi zostały opisane w odpowiednich podrozdziałach w dalszej części pracy.

2.2. Podstawowe pojęcia

Problematyka zapewnienia bezpieczeństwa systemów informatycznych jest złożonym i obszernym zagadnieniem. W zależności od docelowego obszaru zastosowań inne aspekty stają się szczególnie istotne. Można jednak wyróżnić pewne podstawowe pojęcia adekwatne w różnych zastosowaniach. Pierwszym z nich jest *uwierzytelnianie* rozumiane jako sposób potwierdzenia tożsamości stron. Oczywiście inne mechanizmy uwierzytelniania będą stosowane np. w ramach kontroli dostępu do systemu operacyjnego, a inne podczas transmisji danych, co zostanie szczegółowo wyjaśnione w dalszej części pracy. Z uwierzytelnianiem blisko związana jest *autoryzacja* do wykonania pewnej czynności, oznaczająca mechanizm weryfikacji, czy zamierzone działanie mieści się w ramach przyjętych uprawnień. Podczas wymiany danych ważnym aspektem bezpieczeństwa jest zapewnienie *poufności* transmisji, a więc zagwarantowanie, że dane nie mogą zostać odczytane przez osoby niepowołane. Podobnie istotna jest *integralność* przesyłanych danych, rozumiana jako pewność, że wiadomość dotarła od nadawcy w niezmienionej formie, a więc nie została podczas transmisji celowo zmodyfikowana przez osoby trzecie. *Niezaprzeczalność* oznacza z kolei brak możliwości wyparcia się przez jedną ze stron uczestnictwa w wymianie danych.

2.3. Elementy współczesnej kryptografii

2.3.1. Podstawy

Spełnienie tak sformułowanych wymagań niesie konieczność zastosowania specjalizowanych mechanizmów. Jednym z możliwych do zastosowania narzędzi jest kryptografia dostarczająca mechanizmów, które dobrze wykorzystane mogą zapewnić spełnienie wymagań związanych z bezpieczeństwem. Za pomocą narzędzi kryptograficznych można zaszyfrować przekazywaną informację, można wygenerować skrót informacji, żeby sprawdzić poprawność transmisji, można informację podpisać, można też zweryfikować tożsamość osoby, z którą się kontaktujemy.

Kryptografia posługuje się wieloma pojęciami, które często są używane przez laików, ale nie zawsze są poprawnie rozumiane. Dlatego też wyjaśniamy te, które są najistotniejsze w tej pracy. *Algorytmem szyfrującym* albo po prostu *szyfrem*

będziemy nazywać funkcję przetwarzającą *tekst jawny* w jego zakodowaną postać zwaną *kryptogramem*. Analogicznie funkcję odtwarzającą tekst jawny na podstawie kryptogramu nazwiemy *deszyfrującą*. Współczesne szyfry są z reguły funkcjami dwóch zmiennych, rolę jednej zmiennej pełni oczywiście tekst jawny, a drugą *klucz szyfrujący*.

Warto podkreślić, że obecnie szyfry są tak konstruowane, by klucz stanowił jedyny element, którego poufność musi być zapewniona dla bezpieczeństwa informacji, to znaczy dane o całym systemie wraz ze specyfikacją użytego algorytmu szyfrowania bez znajomości klucza nie pozwalają intruzowi na złamanie zabezpieczeń (zasada Kerckhoffsa). Konieczność przyjęcia założenia, że „intruz zna system" zostało potwierdzone przez Shannona w pracy, która jest uważana za leżącą u podwalin współczesnej kryptografii [41]. Obecnie do dobrych praktyk należy ujawnienie pełnej specyfikacji każdego nowego algorytmu szyfrującego, co pozwala na znalezienie i załatanie potencjalnych luk zanim szyfr zostanie powszechnie zastosowany.

Współczesną kryptografię można podzielić na *symetryczną* i *asymetryczną*. Dodatkowo szyfry symetryczne są dzielone na *strumieniowe* i *blokowe*. Od rodzaju algorytmu szyfrowania zależy sposób jego użycia i to, do jakich zastosowań się nadaje.

2.3.2. Kryptografia symetryczna

W kryptografii symetrycznej podczas szyfrowania i deszyfracji jest wykorzystywany ten sam klucz. W czasie transmisji danych musi on być więc znany obu stronom, co komplikuje jego poufne ustalenie i przekazanie bezpiecznym kanałem. Najprostsze szyfry tego typu stosowano już w starożytności. Przykładem może być antyczny szyfr, prawdopodobnie używany przez Juliusza Cezara, zwany *szyfrem Cezara*. W szyfrze tym każdej literze alfabetu przyporządkowujemy inną literę przesuniętą w alfabecie o ustaloną liczbę miejsc[2]. Dla przykładowego przesunięcia o jedno miejsce w alfabecie łacińskim litera `A` po zaszyfrowaniu byłaby zapisywana jako `B`, odpowiednio `B` jako `C`, finalnie `Z` jako `A`. Kluczem będzie więc ustalona liczba miejsc do przesunięcia, funkcją szyfrującą odpowiednie przesunięcie w prawo, a deszyfrującą – w lewo. Przykładowy tekst jawny `TEKST` po zaszyfrowaniu w opisany sposób miałby postać `UFLTU`.

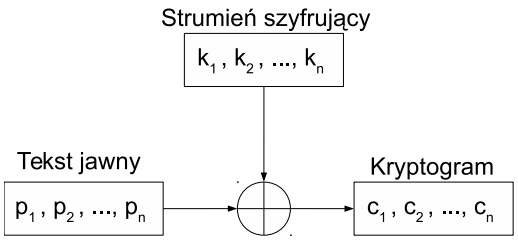
Zauważmy, że szyfr Cezara nie zapewnia elementarnego bezpieczeństwa. Niewielki zakres zmian wartości klucza (możliwego przesunięcia w alfabecie) powoduje, że bez znajomości klucza można szybko odtworzyć tekst jawny na podstawie kryptogramu, sprawdzając wszystkie możliwe przesunięcia. Taką metodę nazywamy *atakiem brute-force*.

[2] Cezar często stosował przesunięcie o trzy miejsca.

Szyfr Cezara każdej literze alfabetu przyporządkowuje inną literę. Szyfry o takiej właściwości nazywamy *podstawieniowymi*. Sposób dokonywania podstawienia w szyfrze Cezara był bardzo prosty. Możliwe jest jednak zastosowanie dużo bardziej skomplikowanych funkcji, aby zmniejszyć podatność na atak *brute-force*. Zauważmy jednak, że w tego typu szyfrach takie same litery tekstu jawnego są szyfrowane na identyczne litery kryptogramu. Właściwość ta powoduje, że szyfry podstawieniowe są podatne na *kryptoanalizę statystyczną* i obecnie niestosowane w praktyce. W językach naturalnych zazwyczaj pewne litery występują dużo częściej niż inne, przykładowo w języku polskim znacznie częściej występują litery `a` czy `i` niż np. `f`. Zakładając pewien rozkład częstości występowania poszczególnych znaków w tekście jawnym i porównując go z rozkładem obliczonym dla odpowiednio długiego kryptogramu (lub zestawu kryptogramów szyfrowanych tym samym kluczem), można odtworzyć przyporządkowanie wynikające z zastosowanej funkcji szyfrującej i klucza.

Zauważmy, że podobną kryptoanalizę statystyczną można zastosować także wówczas, gdy sposób dokonywania podstawienia zależy od miejsca w tekście jawnym. Rozważmy szyfr, w którym są używane różne podstawienia dla parzystych i nieparzystych znaków tekstu jawnego, wynikające z różnych funkcji szyfrujących lub różnych kluczy. Przykładowo, załóżmy, że dla nieparzystych pozycji tekstu jawnego jest stosowany szyfr Cezara z przesunięciem o jeden znak, a dla parzystych również szyfr Cezara, ale z przesunięciem o dwa znaki. Litera `A` występująca na miejscach nieparzystych będzie więc szyfrowana jako `B`, a na miejscach parzystych jako `C`. Podejście takie zwiększy wprawdzie odporność na atak *brute-force*, ale niewiele na kryptoanalizę statystyczną. Analiza częstości występowania poszczególnych znaków w kryptogramie poprowadzona osobno dla parzystych i nieparzystych pozycji pozwoli na łatwe odtworzenie klucza.

Podatność na kryptoanalizę statystyczną jest spowodowana powtarzalnością podstawień. Jeśli innym kluczem zaszyfrujemy parzyste, a innym nieparzyste znaki tekstu jawnego, to intruz może osobno obliczyć statystykę przy jednym i drugim kluczu. Jeśli dodamy trzeci element klucza i będziemy w jednakowy sposób szyfrować co trzeci znak tekstu jawnego, to intruz analogicznie może obliczać częstości dla co trzeciego znaku kryptogramu. Zwiększanie klucza o kolejne elementy zwiększa liczbę statystyk, która musi zostać obliczona przez intruza. Gdybyśmy jednak dysponowali kluczem o długości równej długości tekstu jawnego, to kryptoanaliza statystyczna takiego kryptogramu nie dałaby rezultatu, gdyż każdy znak byłby zaszyfrowany innym elementem klucza. Taka metoda zwana *szyfrem z kluczem jednorazowym* jest bezpieczna, pod warunkiem że dany klucz jest stosowany tylko raz, czyli dla wygenerowania jednego kryptogramu, a poszczególne elementy klucza nie zależą od siebie (np. klucz jest wygenerowany losowo). Wielokrotne użycie tego samego klucza do zaszyfrowania różnych tekstów jawnych jest nie-



Rys. 2.1. Zasada działania szyfru strumieniowego

dopuszczalne. Niestety praktyczne wykorzystanie takiej metody jest ograniczone ze względu na znaczny rozmiar jednorazowych kluczy, które muszą być losowe i wcześniej uzgodnione pomiędzy komunikującymi się stronami.

Współcześnie stosowane symetryczne algorytmy szyfrujące można podzielić na *blokowe* i *strumieniowe*. Szyfry blokowe operują na blokach o ustalonej długości, podczas gdy strumieniowe mogą przetwarzać bloki o dowolnej długości, bit po bicie. Generują one *strumień szyfrujący*, który jest używany jako jednorazowy klucz dodawany do tekstu jawnego zazwyczaj operacją XOR. Zasadę działania szyfru strumieniowego zilustrowano na rys. 2.1, przyjmując oznaczenia c_i – i-ty bit kryptogramu, p_i – i-ty bit tekstu jawnego, k_i – i-ty bit strumienia szyfrującego.

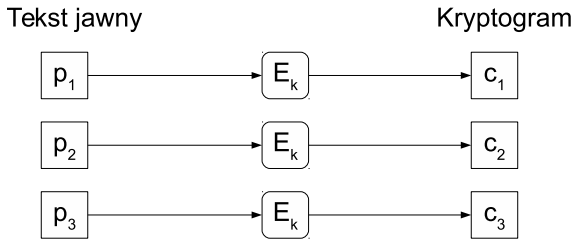
W odróżnieniu od szyfrów strumieniowych szyfry blokowe operują na blokach o ustalonej długości. Przykładowo, AES [1] wymaga bloków o rozmiarze 128 bitów. Matematycznie taki szyfr można rozumieć jako funkcję wzajemnie jednoznaczną (bijekcję), która przy danej wartości klucza każdemu n-bitowemu słowu na wejściu przyporządkowuje n-bitowe słowo na wyjściu. Funkcja odwrotna do niej pozwala na deszyfrację kryptogramu, to jest na odtworzenie początkowego tekstu jawnego (słowa podanego na wejście funkcji szyfrującej) na podstawie słowa wyjściowego i klucza. W praktyce zastosowanie szyfrów blokowych do kodowania wiadomości o dowolnym rozmiarze wymaga przyjęcia pewnego sposobu postępowania zwanego *trybem działania szyfru blokowego*. Podstawowe tryby omówiono w kolejnej części skryptu.

Tryby działania szyfrów blokowych

Skrótowo opisano podstawowe tryby działania szyfrów blokowych możliwe do zastosowania podczas szyfrowania tekstu jawnego o dowolnym rozmiarze. Opis ten nie wyczerpuje całości zagadnienia, więcej informacji dociekliwy Czytelnik może znaleźć np. w pracach [6, 22].

Początkowo tekst jawny jest dzielony na bloki o rozmiarze wejścia danego algorytmu szyfrującego. W razie potrzeby ostatni blok jest uzupełniany do ustalonej

długości[3]. W najbardziej elementarnym trybie, zwanym trybem elektronicznej książki kodowej ECB (ang. *Electronic Codebook*), bloki tekstu jawnego są kolejno szyfrowane, tworząc poszczególne bloki kryptogramu. Proces ten przedstawiono na rys. 2.2.



Rys. 2.2. Tryb *Electronic Codebook*

Matematycznie proces szyfrowania można zapisać jako:

$$c_i = E_k(p_i)$$

gdzie c_i oznacza i-ty blok kryptogramu (ang. *ciphertext*), p_i – i-ty blok tekstu jawnego (ang. *plaintext*), E_k – funkcję szyfrującą (ang. *encrypt*) przy kluczu k. Analogicznie proces deszyfracji można zapisać jako:

$$p_i = D_k(c_i)$$

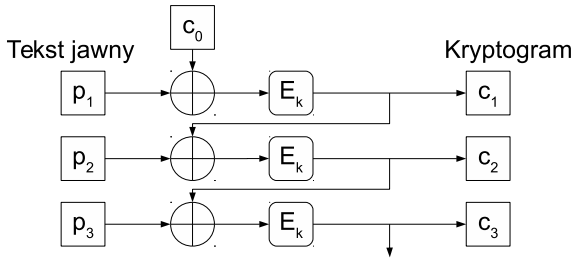
gdzie D_k oznacza funkcję deszyfrującą (ang. *decrypt*) przy kluczu k.

Należy zaznaczyć, że tryb ten nie powinien być stosowany w praktyce, gdyż nie zapewnia wystarczającego poziomu bezpieczeństwa. W trybie ECB identyczne bloki tekstu jawnego są szyfrowane na identyczne bloki kryptogramu, co znacznie ułatwia potencjalne ataki.

W kolejnym z trybów, zwanym CBC (ang. *Cipher Block Chaining*), przed zaszyfrowaniem danego bloku tekstu jawnego obliczany jest jego XOR ($\oplus$) z poprzednim blokiem kryptogramu. Aby zróżnicować kryptogramy identycznych wiadomości, dla pierwszego bloku tekstu jawnego rolę poprzedniego bloku kryptogramu pełni losowy *wektor inicjalizacyjny* IV, który może być jawnie przekazany wraz z kryptogramem[4]. Proces ten pokazano na rys. 2.3.

[3] Można przyjąć różne sposoby uzupełnienia, np. pojedynczym bajtem o wartości 80 szesnastkowo, po którym następuje ciąg bajtów o wartości równej zeru, albo ciągiem bajtów o wartości równej długości wypełnienia. Przy tych metodach, aby możliwe było odtworzenie oryginalnego tekstu jawnego, musi wystąpić uzupełnienie o przynajmniej jeden bajt, a więc tekst jawny o długości równej wielokrotności rozmiaru bloku będzie uzupełniony całym blokiem. Brak deterministycznego wypełnienia może skutkować poważnymi atakami, jak np. atak POODLE (ang. *Padding Oracle On Downgraded Legacy Encryption*) na SSL 3.0 [23].

[4] Należy podkreślić konieczność losowego wyboru IV, determinizm może prowadzić do poważnych luk bezpieczeństwa, jak np. luki w SSL 3.0 i TLS 1.0 wykorzystywane w ataku BEAST (ang. *Browser Exploit Against SSL/TLS*) [9].

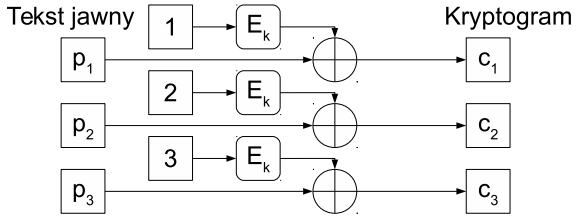


Rys. 2.3. Tryb *Cipher Block Chaining*

Procesy szyfrowania i deszyfrowania można tu zapisać odpowiednio jako:

$$c_0 = IV$$
$$c_i = E_k(c_{i-1} \oplus p_i)$$
$$p_i = c_{i-1} \oplus D_k(c_i)$$

Warto zauważyć, że w trybie tym szyfrowanie musi być wykonywane sekwencyjnie i nie można tego procesu zrównoleglić, gdyż każdy z bloków kryptogramu zależy od poprzedniego bloku. Deszyfracja natomiast może być prowadzona równolegle, gdyż na podstawie dwóch sąsiednich bloków kryptogramu można odtworzyć blok tekstu jawnego. Jeżeli zastosowanie wymaga także możliwości zrównoleglenia szyfrowania, to muszą być wykorzystywane inne tryby. Najprostszy z nich to tak zwany tryb licznikowy CTR (ang. *Counter*). W trybie tym są szyfrowane kolejne wartości licznika, a następnie jest obliczany XOR z odpowiednimi blokami tekstu jawnego. Podobnie jak poprzednio, aby zróżnicować kryptogramy identycznych wiadomości, początkowa wartość licznika jest zazwyczaj losowa, przesyłana z wiadomością jako wektor inicjalizacyjny. Tryb ten pokazano na rys. 2.4.

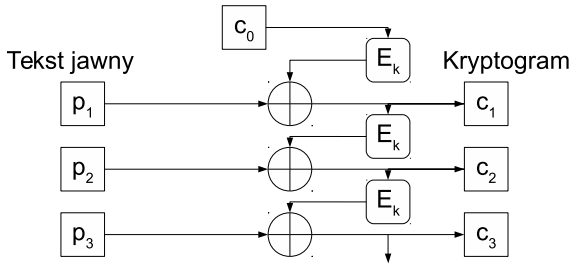


Rys. 2.4. Tryb *Counter*

Szyfracja i deszyfracja przebiega według wzorów:

$$c_i = E_k(IV + i) \oplus p_i$$
$$p_i = E_k(IV + i) \oplus c_i$$

Tryb taki pozwala na pełne zrównoleglenie procesów szyfrowania i deszyfrowania. Warto także zauważyć, że szyfrowanie kolejnych wartości licznika *de facto* tworzy strumień szyfrujący, z którym następnie jest obliczany XOR kolejnych bloków wiadomości. Dzięki temu można łatwo uniknąć konieczności rozszerzenia ostatniego bloku do rozmiaru obsługiwanego przez użyty algorytm szyfrujący. Kolejną cechą jest użycie funkcji szyfrującej (E_k) także podczas deszyfracji. Zrezygnowanie z konieczności implementacji funkcji D_k stanowi pewne ułatwienie. Podobnie jest w trybie CFB (ang. *Cipher Feedback*). W trybie tym poprzedni blok kryptogramu jest szyfrowany, tworząc strumień szyfrujący do obliczenia XOR z kolejnym blokiem tekstu jawnego. Analogicznie do trybu CBC początkowy blok c_0 jest inicjalizowany losowym wektorem, by zapewnić zróżnicowanie kryptogramów identycznych wiadomości. Tryb ten pokazano na rys. 2.5.



Rys. 2.5. Tryb *Cipher Feedback*

Symbolicznie procesy szyfrowania i deszyfrowania w tym trybie można zapisać jako:

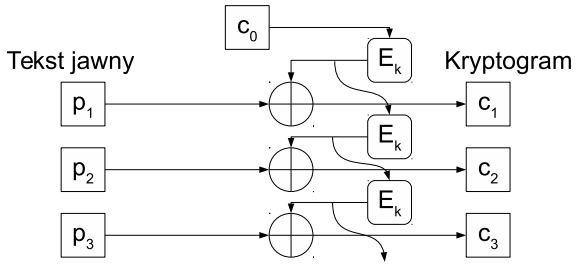
$$\begin{aligned} c_0 &= IV \\ c_i &= E_k(c_{i-1}) \oplus p_i \\ p_i &= E_k(c_{i-1}) \oplus c_i \end{aligned}$$

W ostatnim z omówionych trybów, zwanym OFB (ang. *Output Feedback*), strumień szyfrujący jest tworzony przez wielokrotne szyfrowanie początkowego wektora inicjalizacyjnego. Proces ten pokazano na rys. 2.6.

Działanie tego trybu można zapisać następująco:

$$\begin{aligned} c_0 &= IV \\ c_i &= E_k^i(c_0) \oplus p_i \\ p_i &= E_k^i(c_0) \oplus c_i \end{aligned}$$

Opisane tryby działania szyfrów blokowych należą do najbardziej podstawowych. Oprócz wymienionych istnieje także wiele innych trybów, w tym bardziej złożonych, zapewniających jednocześnie kontrolę integralności wiadomości, jak



Rys. 2.6. Tryb *Output Feedback*

np. CCM (ang. *Counter with CBC-MAC*) [10] czy OCB (ang. *Offset Codebook*) [39]. Jednak omówienie tych trybów znacznie wykracza poza ramy niniejszej pracy.

2.3.3. Kryptografia asymetryczna

Algorytmy kryptografii asymetrycznej są dość złożone i ich przedstawienie wykracza poza ramy niniejszej pracy. Szczegółowe opisy dociekliwy Czytelnik znajdzie np. w pracach [20, 22]. W rozdziale tym będą przedstawione jedynie ogólne zasady i wskazane obszary zastosowań.

Kryptografia asymetryczna wykorzystuje parę kluczy: *klucz publiczny* i *prywatny*. Oba klucze są generowane jednocześnie. Klucze te wzajemnie się uzupełniają. Wiadomości zaszyfrowane kluczem publicznym mogą być odszyfrowane jedynie kluczem prywatnym. Analogicznie wiadomości zaszyfrowane kluczem prywatnym mogą być odszyfrowane kluczem publicznym z danej pary kluczy. Klucz publiczny jest jawny, klucz prywatny poufny. Należy zauważyć, że w zależności od użytego algorytmu może nie być obojętne, który z kluczy potraktujemy jako prywatny, a który jako publiczny. Istotne jest, by na podstawie klucza publicznego nie było możliwe odtworzenie klucza prywatnego.

Należy podkreślić, że klucze są tworzone poprzez złożone obliczenia i nie można ich utożsamiać z hasłem definiowanym przez użytkownika. Jeżeli oprogramowanie w trakcie generowania pary kluczy żąda podania hasła do ochrony klucza prywatnego, to oznacza jedynie, że utworzony klucz prywatny dla większego bezpieczeństwa nie będzie przechowywany w formie jawnej, ale zaszyfrowany z użyciem podanego hasła pewnym algorytmem symetrycznym.

Kryptografia asymetryczna jest powszechnie wykorzystywana w procesie uwierzytelniania stron i podczas składania podpisów elektronicznych, jak to szerzej omówiono w podrozdziale 2.4. Rzadko natomiast stosuje się ją do szyfrowania dużych ilości danych. Z powodu dużej złożoności obliczeniowej wykorzystanie algorytmów asymetrycznych w trakcie transmisji w praktyce sprowadza się do

bezpiecznego ustalenia jednorazowego klucza sesji, którym za pomocą algorytmu symetrycznego są następnie szyfrowane wymieniane informacje.

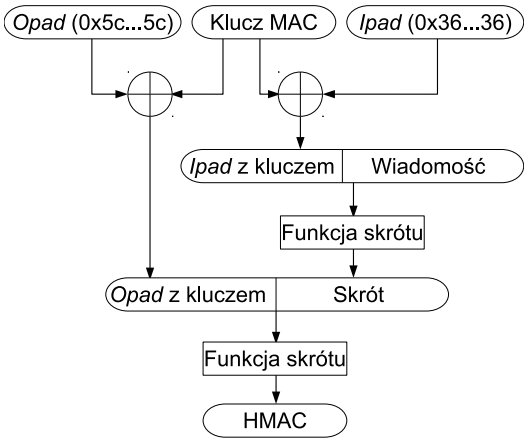
2.3.4. Funkcje skrótu

Często zachodzi konieczność obliczenia na podstawie dowolnie długiej wiadomości jej *skrótu*, czyli przyporządkowania jej wartości z pewnego ograniczonego zbioru. Taką prostą funkcją skrótu mogłaby być przykładowo suma modulo 256 (tj. z odrzuceniem przeniesień na dziewiąty bit) poszczególnych bajtów wiadomości. W ten sposób każdej wiadomości można przyporządkować wartość z zakresu 0–255. Zauważmy jednak, że taka prosta funkcja skrótu ma poważne wady. Niewielka moc zbioru możliwych skrótów powoduje, że nie można zaniedbać prawdopodobieństwa, że obliczony przez nas skrót dwóch różnych wiadomości będzie identyczny. Taką parę różnych wiadomości o jednakowym skrócie nazywamy *kolizją*. Opisana funkcja jest wyjątkowo podatna na kolizje. Dodawanie jest przemienne, więc modyfikacja wiadomości polegająca jedynie na zmianie kolejności liter skutkowałaby identycznym skrótem. Co więcej, intruz łatwo mógłby przygotować zupełnie inną wiadomość, ale o tym samym skrócie.

Funkcje skrótu używane w kryptografii nie mogą mieć opisanych wad. *Kryptograficznie silną funkcją skrótu* będziemy nazywać funkcję przyporządkowującą długiej wiadomości wartość z pewnego ograniczonego zbioru, tak że brak będzie praktycznej możliwości odtworzenia wiadomości na podstawie danego skrótu, znalezienia dwóch różnych wiadomości o identycznym skrócie w rozsądnym czasie, jak również znalezienia innej wiadomości o danym skrócie.

Zauważmy, że znalezienie dwóch różnych wiadomości o jednakowym skrócie jest na ogół łatwiejszym zadaniem niż spreparowanie wiadomości o ustalonym skrócie. Ma to związek z tak zwanym *paradoksem urodzinowym*. Łatwo zauważyć, że w grupie 367 osób muszą znaleźć się przynajmniej dwie osoby obchodzące urodziny tego samego dnia, niekoniecznie jednak znajdzie się tam choć jedna osoba mająca urodziny w przyjętym przez nas *a priori* dniu (np. 1 stycznia). W praktyce już dla niewielkiej grupy dwudziestu trzech losowo wybranych osób prawdopodobieństwo, że są tam dwie osoby mające urodziny tego samego dnia jest większe niż 50%.

Opierając się na tym fakcie, jest możliwy *atak urodzinowy* na funkcję skrótu. Załóżmy, że osoby A i B mają zawrzeć umowę, która będzie podpisana przez każdą z osób podpisem elektronicznym. Szerzej mechanizm składania podpisu elektronicznego wyjaśniono w podrozdziale 2.4.4, tu ważny jest jedynie fakt, że podpis w istocie jest składany nie pod pełnym dokumentem, a pod jego skrótem. Przyjmijmy, że osoba A przygotowuje tekst umowy do podpisu i chce oszukać osobę B. Zamierza ona przygotować dwie umowy różniące się treścią, ale o identycznym skrócie. Jedną z nich przedstawi osobie B do podpisu. Identyczny skrót obu umów



Rys. 2.7. Obliczanie HMAC

spowoduje, że podpis osoby B pod oryginalną umową będzie także prawidłowym podpisem dla drugiej niekorzystnej dla niej umowy. Osobie A byłoby trudno dla konkretnej umowy z danym skrótem znaleźć drugą o identycznym skrócie. Jeżeli jednak może modyfikować jednocześnie treści obu umów (np. przez zmianę formatowania, wstawianie nadmiarowych spacji), to łatwiej znajdzie szukaną parę umów o identycznym skrócie.

Kryptograficznie silne funkcje skrótu mają wiele zastosowań. Jak wspomniano, są wykorzystywane np. przy podpisach elektronicznych. Innym zastosowaniem jest metoda zagwarantowania integralności transmisji, oparta na kodzie uwierzytelniającym obliczanym z wykorzystaniem kryptograficznie silnej funkcji skrótu HMAC (ang. *Hash-based Message Authentication Code* [49]). Sposób obliczania pokazano na rys. 2.7. Początkowo obliczany jest XOR klucza MAC i wewnętrznego (ang. *Inner padding – Ipad*) oraz zewnętrznego wypełnienia (ang. *Outer padding – Opad*). Do wewnętrznego wypełnienia z kluczem jest doklejana wiadomość, a następnie jest obliczany ich skrót. Skrót ten jest łączony z kluczowanym zewnętrznym wypełnieniem i ponownie poddany działaniu funkcji skrótu, tworząc kod uwierzytelniający HMAC wiadomości.

2.4. Zastosowanie kryptografii

2.4.1. Poufność danych

Dynamicznie rozwijająca się przez ostatnie lata kryptografia jest jednym ze sposobów zapewnienia bezpieczeństwa przechowywanych i przesyłanych danych, a także systemów informatycznych. Opisujemy, jak można wykorzystać mechani-

zmy oferowane przez kryptografię do rozwiązania typowych problemów bezpieczeństwa.

Zapewnienie poufności danych jest jednym z najbardziej znanych zastosowań kryptografii. Można ją zrealizować poprzez zaszyfrowanie danych – ich odczyt będzie wtedy możliwy jedynie za pomocą odpowiedniego klucza.

Do zaszyfrowania danych można użyć algorytmów symetrycznych, a do ich odszyfrowania będzie potrzebny ten sam klucz, którego użyto do zaszyfrowania. Algorytmy symetryczne wykorzystują stosunkowo proste operacje matematyczne, tak więc szyfrowanie i deszyfrowanie są szybkimi operacjami.

Bardziej skomplikowanym rozwiązaniem jest zastosowanie algorytmów asymetrycznych oraz kluczy publicznego i prywatnego. Dane zaszyfrowane jednym z kluczy można odszyfrować jedynie drugim kluczem. Algorytmy asymetryczne znacznie ułatwiają zrealizowanie zaszyfrowanej komunikacji, gdyż strony tej komunikacji nie muszą współdzielić tego samego klucza. Nie jest to jednak ich jedyne zastosowanie. Przykładowo, jeśli Jan Kowalski wygeneruje dla siebie parę kluczy (publiczny i prywatny), klucz publiczny może z definicji udostępnić każdemu bez ograniczeń. Każdy kto będzie chciał przesłać tajną wiadomość do Jana Kowalskiego, będzie mógł ją zaszyfrować jego ogólnie znanym kluczem publicznym i przesłać. Taką wiadomość można odszyfrować jedynie kluczem prywatnym. Mimo że wiele osób zna klucz publiczny Jana Kowalskiego, przesyłana wiadomość jest bezpieczna. Jedyna trudność polega na zweryfikowaniu, że posiadany przez wysyłającego klucz publiczny rzeczywiście należy do Jana Kowalskiego – do tego służą opisane dalej rozwiązania PGP (zob. podrozdział 2.4.5) i PKI (zob. podrozdział 2.4.6). Algorytmy asymetryczne wymagają zastosowania dłuższych kluczy i bardziej skomplikowanych operacji matematycznych, są więc wolniejsze od symetrycznych.

Szczególnym przypadkiem, w którym konieczne jest zapewnienie poufności, jest przechowywanie w systemach komputerowych haseł użytkowników potrzebnych do zalogowania. Aby zapewnić ich zarówno wymagany poziom bezpieczeństwa, jak i szybkość działania, nie używa się w tym wypadku ani algorytmów symetrycznych ani asymetrycznych. Wymagałoby to bowiem przechowywania w tym samym systemie informatycznym kluczy umożliwiających odszyfrowanie listy haseł. Ewentualny intruz miałby więc tylko nieco bardziej utrudnione zadanie – po włamaniu się do systemu oprócz zaszyfrowanej listy haseł musiałby zabrać także klucze szyfrujące. Ten szczególny przypadek zapewnienia poufności nie wymaga nigdy odszyfrowania raz zaszyfrowanego hasła. W tym wypadku stosuje się tzw. funkcje skrótu (zob. podrozdział 2.3.4). Aby sprawdzić, czy logujący się do systemu użytkownik podał poprawne hasło, można obliczyć skrót podanego przez niego hasła i porównać ze skrótem zapisanym w systemie. W razie wycieku listy skrótów haseł bardzo trudno jest odtworzyć pierwotne hasła. Szczegóły zapewnie-

nia bezpiecznego uwierzytelniania podczas dostępu do systemów informatycznych opisano w podrozdz. 4.10.1.

2.4.2. Integralność danych

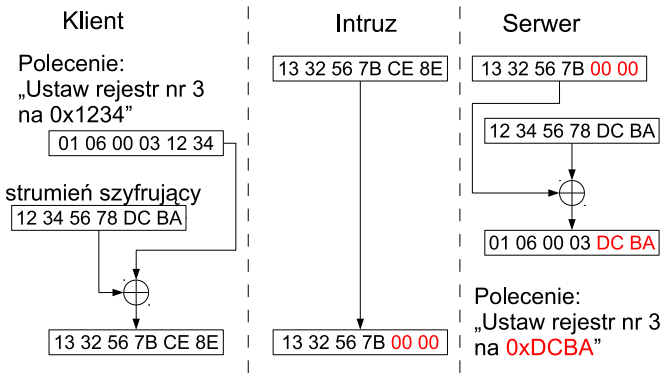
Ważnym aspektem bezpieczeństwa danych jest ich integralność – zapewnienie, że dane nie zostały zmodyfikowane przypadkowo bądź celowo. Najczęściej przywoływanym przykładem, w którym zapewnienie integralności jest potrzebne, jest transmisja danych – w takim wypadku należy zapewnić, że odebrane dane są takie same, jak dane wysyłane. Podobna sytuacja może mieć miejsce w przypadku przechowywania danych przez jakiś czas na nośnikach – integralność oznacza, że przechowywane dane nie zostały zmodyfikowane. Ochrona przed przypadkową modyfikacją, będącą skutkiem błędu w transmisji danych czy uszkodzenia nośnika, może być dość skutecznie zrealizowana za pomocą prostej sumy kontrolnej. Zabezpieczenie przed umyślną modyfikacją wymaga bardziej skomplikowanych metod – należy założyć, że osoba zmieniająca dane będzie w stanie dołączyć do nich także odpowiednio zmodyfikowaną sumę kontrolną.

Należy zaznaczyć, że gwarancja poufności danych nie zapewnia automatycznie ich integralności. Atakujący może nie być w stanie przewidzieć, co dokładnie zmienia albo jaki dokładnie będzie skutek modyfikacji, może jednak – działając na oślep – zmienić przesyłany kryptogram. Jeśli nie zastosowano metod zapewnienia poufności, po odszyfrowaniu nie będzie możliwości stwierdzenia, czy nastąpiła modyfikacja danych czy też nie. Błędne jest założenie, że skoro odbiorca był w stanie prawidłowo odszyfrować transmisję, to nie mogła ona zostać zmodyfikowana przez intruza nieznającego klucza.

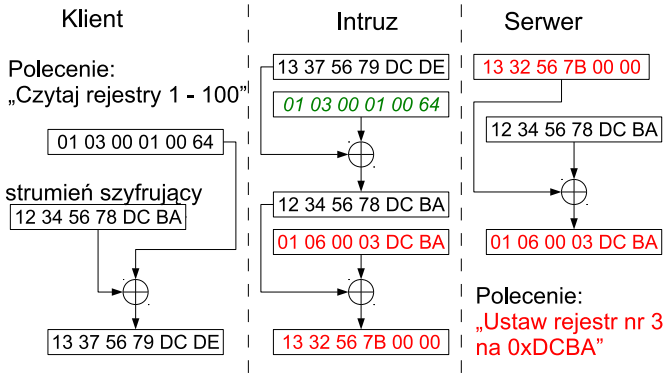
Przykładowo, rozważmy transmisję, do której użyto szyfru strumieniowego, jak pokazano na rys. 2.8. Podczas ataku *man-in-the-middle* intruz może w losowych miejscach modyfikować przekaz, zmieniając znaczenie przesyłanych danych. Dodatkowo znajomość specyfikacji protokołu (zazwyczaj jawnej) pozwala na modyfikację jedynie pól wybranych przez intruza.

Częściowa bądź pełna znajomość tekstu jawnego pozwala intruzowi na odtworzenie fragmentu klucza szyfrującego i wstrzyknięcie do transmisji swoich danych, podszywając się pod zaufanego nadawcę. Należy podkreślić, że sytuacja gdy osoby trzecie są w stanie z dużym prawdopodobieństwem odtworzyć fragment przesyłanego tekstu jawnego często występuje w praktyce. Dane zazwyczaj zawierają standardowe fragmenty (np. nagłówki plików pewnych typów) bądź aplikacja cyklicznie ponawia transmisję identycznego polecenia (np. odczyt konfiguracji bezpośrednio po nawiązaniu połączenia). Przebieg ataku w takiej sytuacji pokazano na rys. 2.9.

Zapewnienie integralności danych może być zrealizowane poprzez dołączenie do transmitowanych danych kodu uwierzytelniającego HMAC wygenerowanego za



Rys. 2.8. Przykład ataku *man-in-the-middle* (modyfikacja nieznanej treści)



Rys. 2.9. Przykład ataku *man-in-the-middle*
(modyfikacja treści przy znanym tekście jawnym)

pomocą kryptograficznie silnej funkcji skrótu (zob. 2.3.4). HMAC jest wyznaczany na podstawie osobnego klucza z kryptogramu lub z tekstu jawnego (przed ewentualnym zaszyfrowaniem)[5]. Atakujący, który – działając na oślep – zmodyfikuje kryptogram, nie będzie wiedział, jaki dokładnie jest skutek tej modyfikacji, nie będzie więc w stanie odpowiednio zmodyfikować kodu uwierzytelniającego. Po odszyfrowaniu wiadomości łatwo stwierdzić, czy dołączony do niej kod uwierzytelniający pasuje do jej treści.

Integralność danych za pomocą HMAC można zapewnić niezależnie od tego, czy dane są szyfrowane czy nie. Jest to istotne w przypadkach, w których poufność

[5] Możliwe są trzy podejścia: 1) wyznaczenie HMAC z tekstu jawnego i zaszyfrowanie wraz z nim (ang. *MAC-then-encrypt*), 2) wyznaczenie HMAC z tekstu jawnego i przesłanie niezaszyfrowanego razem z kryptogramem (ang. *encrypt-and-MAC*), 3) wyznaczenie HMAC z kryptogramu (ang. *encrypt-then-MAC*). Trzecie podejście jest najbezpieczniejsze, pozostałe są podatne na ataki [4, 18].

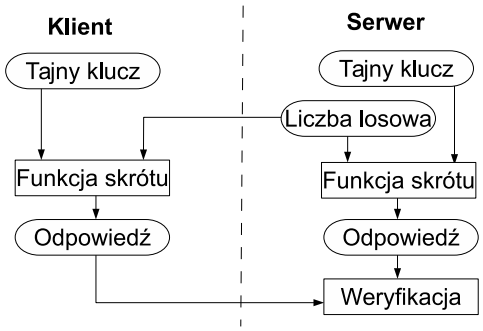
nie jest konieczna, a szyfrowanie powoduje zbyt duże narzuty. Wymaga to jednak przechowywania dodatkowej informacji. Przykładowo, aby zapewnić integralność przesyłanej informacji, dwie strony biorące udział w komunikacji muszą współdzielić hasło – fragment informacji, który zostanie dołączony do przesyłanych jawnym tekstem danych przed zastosowaniem funkcji skrótu. Bez tego hasła nie będzie możliwe obliczenie nowego poprawnego kodu uwierzytelniającego. Atakujący, nie znając tajnego hasła, nie będzie w stanie niezauważenie zmodyfikować przesyłanych danych.

2.4.3. Uwierzytelnianie

Uwierzytelnianie polega na zweryfikowaniu tożsamości. W systemach informatycznych uwierzytelnianie może być wymagane w różnych sytuacjach, np. na początku zdalnej komunikacji – przesyłania danych przez sieć komputerową, albo przed rozpoczęciem pracy z systemem informatycznym – w tym przypadku odbywa się zwykle tzw. logowanie do systemu. Do uwierzytelniania w różnych sytuacjach można wykorzystać mechanizmy kryptograficzne, choć często stosowane są prostsze metody (np. login i hasło).

Mechanizm, który jest często stosowany do uwierzytelniania kryptograficznego, zakłada wykorzystanie specyfiki algorytmów asymetrycznych. Zasadniczą ich cechą jest to, że wiadomość zaszyfrowaną za pomocą klucza publicznego można odszyfrować tylko kluczem prywatnym i odwrotnie. Klucz prywatny powinien być tajny – znany tylko właścicielowi. Przykładowo, będąc w posiadaniu klucza publicznego Jana Kowalskiego (a więc takiego, dla którego odpowiedni klucz prywatny posiada tylko Jan Kowalski), można zweryfikować, czy komunikująca się z nami osoba jest Janem Kowalskim. Aby udowodnić swoją tożsamość, osoba musi wykazać, że jest w posiadaniu odpowiedniego klucza prywatnego, który jak wiemy należy do Jana Kowalskiego. Należy to oczywiście zrobić bez ujawniania tego klucza – ma nadal pozostać znany tylko Janowi Kowalskiemu. Można tego dokonać, przekazując osobie wiadomość, którą należy zaszyfrować kluczem prywatnym Jana Kowalskiego. Jeśli zaszyfrowaną przez sprawdzaną osobę wiadomość da się odszyfrować kluczem publicznym Jana Kowalskiego i uzyskać wiadomość oryginalną, będzie wiadomo, że zaszyfrował ją Jan Kowalski swoim kluczem prywatnym.

Wiarygodna weryfikacja tożsamości osoby w opisany sposób wymaga pewności co do prawdziwego właściciela klucza publicznego – pewność, kto jest jedynym posiadaczem odpowiedniego klucza prywatnego. Do rozwiązania tych problemów stosuje się mechanizmy oferowane przez takie rozwiązania, jak PGP (zob. podrozdział 2.4.5) bądź PKI (zob. podrozdział 2.4.6). Należy też zadbać, aby osoba, której tożsamość jest weryfikowana, nie mogła odgadnąć bądź wpłynąć na to, jaką wiadomość otrzyma do zaszyfrowania. Jest bowiem możliwe, że intruz jest



Rys. 2.10. Protokół *challenge-response*

w posiadaniu jakiejś wiadomości wcześniej zaszyfrowanej kluczem prywatnym Jana Kowalskiego, którą spróbuje przedstawić jako dowód swojej tożsamości.

Możliwe jest też wykonanie uwierzytelniania za pomocą kryptografii symetrycznej. Wymaga to jednak współdzielenia przez obie strony tego samego tajnego klucza. Wtedy jedna ze stron może zażądać od drugiej zaszyfrowania pewnej wiadomości tajnym kluczem i przekazania szyfrogramu z powrotem. Jeśli szyfrogram po odszyfrowaniu będzie pasował do oryginalnej wiadomości, jest to dowód na posiadanie przez drugą stronę poprawnego tajnego klucza. Rozwiązanie to pozostawia problem ustalenia tajnego klucza w bezpieczny sposób oraz wygenerowania odpowiedniej, nieprzewidywalnej z góry wiadomości służącej do uwierzytelniania, co opisano już wcześniej.

Innym rozwiązaniem jest oparcie uwierzytelniania na kryptograficznie silnej funkcji skrótu w protokole *challenge-response* pokazanym na rys. 2.10. Klient i serwer współdzielą tajny klucz. Serwer, żądając uwierzytelnienia klienta, przesyła do niego losowo wygenerowaną liczbę. Klient oblicza skrót z połączenia hasła z otrzymaną liczbą i przesyła go na serwer. Serwer, porównując otrzymany skrót z niezależnie wygenerowanym przez siebie, weryfikuje tożsamość klienta. Warto zauważyć, że losowy wybór liczby przez serwer minimalizuje możliwość podszycia się intruza pod klienta przy kolejnym uwierzytelnieniu na podstawie podsłuchanych poprzednio danych.

2.4.4. Podpis elektroniczny

Kryptografia asymetryczna i mechanizmy podobne do używanych przy uwierzytelnianiu kryptograficznym (opisanym w podrozdziale 2.4.3) są także wykorzystywane do składania *podpisów elektronicznych*. Podpis elektroniczny służy bowiem stwierdzeniu tożsamości osoby i integralności dokumentu.

Jeśli jesteśmy w posiadaniu klucza publicznego i wiemy, że ten klucz należy do Jana Kowalskiego (tylko Jan Kowalski jest posiadaczem odpowiedniego klucza

prywatnego), wtedy Jan Kowalski może użyć tych faktów, żeby upewnić nas, że jest autorem wiadomości. Jeśli otrzymamy wiadomość, którą będzie można odszyfrować jego kluczem publicznym, będziemy wiedzieli, że zaszyfrowano ją jego kluczem prywatnym, a więc będziemy mieli pewność, że zaszyfrował ją Jan Kowalski. Kowalski może więc wysyłać wiadomość otwartym tekstem (dla każdego, kto chce ją szybko i wygodnie przeczytać) wraz z jej zaszyfrowaną kopią (dla tych, którzy chcą zweryfikować, od kogo pochodzi wiadomość). Aby zweryfikować pochodzenie wiadomości, można spróbować odszyfrować ją kluczem publicznym Jana Kowalskiego i wynik porównać z oryginalną wiadomością. Oczywiście wysyłanie dwóch kopii wiadomości byłoby niewygodne – wymaga bowiem wysłania dwukrotnie większej liczby informacji. Dlatego zamiast szyfrować całą wiadomość oblicza się najpierw jej kryptograficznie silny skrót, który następnie szyfruje się kluczem prywatnym podpisującego. Ten zaszyfrowany skrót jest podpisem elektronicznym wiadomości. Aby zweryfikować podpis, należy odszyfrować go za pomocą klucza publicznego osoby podpisującej, obliczyć skrót oryginalnej wiadomości i porównać, czy obliczony skrót jest taki sam, jak odszyfrowany. Poprawna weryfikacja potwierdza, że osoba, która złożyła podpis jest posiadaczem klucza prywatnego z pary.

Podpisy elektroniczne stanowią jedno z głównych zastosowań kryptografii asymetrycznej. Są wykorzystywane nie tylko do podpisywania dokumentów (co w Polsce reguluje w ostatnich latach odpowiednia ustawa [52], zob. podrozdział 2.4.6), ale też podczas komunikacji sieciowej realizowanej przez różnego rodzaju oprogramowanie, podczas wymiany maili itp. Często użytkownik nie ma świadomości, że używane przez niego oprogramowanie wykorzystuje podpis cyfrowy do realizacji swoich zadań.

2.4.5. PGP

PGP (ang. *Pretty Good Privacy*) to nazwa pierwszego powszechnie dostępnego oprogramowania, które umożliwiało wykorzystanie zalet kryptografii symetrycznej i asymetrycznej. Projekt został zapoczątkowany w 1991 roku. Zaowocował on między innymi powstaniem standardu OpenPGP definiowanego przez dokument RFC 4880 [37]. Ze względu na zmianę licencji w kolejnych wersjach PGP środowisko wolnego oprogramowania stworzyło darmowy, otwarty projekt zgodny ze standardem OpenPGP pod nazwą GPG (ang. *GNU Privacy Guard*). Jest to oprogramowanie otwarte, darmowe, dostępne dla różnych platform, które pozwala wykorzystać w dowolnym celu zalety kryptografii.

GPG pozwala generować klucze kryptografii asymetrycznej i wykorzystywać je do szyfrowania, deszyfrowania i podpisywania wiadomości. W tym celu bardzo często jest stosowane do bezpiecznej komunikacji mailowej. Do kluczy publicznych można dołączać dodatkowe informacje o tożsamości ich właściciela (np. imię,

nazwisko, adres email). Klucze te można opublikować za pomocą dostępnych publicznie tzw. serwerów kluczy – każdy może umieścić swój klucz na jednym z serwerów i każdy może poszukiwać kluczy osób, z którymi chce nawiązać bezpieczną komunikację.

Bezpieczna komunikacja za pomocą kryptografii asymetrycznej poza kluczami wymaga jeszcze potwierdzenia tożsamości właściciela klucza. Ktoś musi zaświadczyć, że klucz pobrany z serwera rzeczywiście należy do konkretnej osoby. W przypadku GPG zastosowano rozwiązanie nazywane *web of trust* (sieć zaufania). Nie istnieją centralne instytucje wydające klucze czy potwierdzające ich prawdziwość, czynią to wzajemnie użytkownicy. Osoba generująca dla siebie nową parę kluczy GPG może poprosić inną osobę, aby podpisała jej klucz publiczny swoim kluczem. Podpisanie klucza publicznego Jana Kowalskiego za pomocą klucza Józefa Nowaka oznacza, że Józef Nowak zaświadcza o autentyczności klucza Jana Kowalskiego (że rzeczywiście należy do tej osoby). W tej sytuacji każdy kto ufa prawdomówności Józefa Nowaka może też zaufać autentyczności klucza publicznego Jana Kowalskiego. Pojedynczy klucz publiczny GPG może być podpisany przez wiele osób, zyskując zaufanie szerszego kręgu ludzi. Takie rozwiązanie nie wymaga utrzymywania kosztownej infrastruktury, równocześnie oferując bardzo wysoki poziom bezpieczeństwa. Pozwala też bardzo szybko tworzyć bezpieczne sieci zaufania, gdy tylko stają się potrzebne. Dzięki temu zapewnienie wysokiego poziomu bezpieczeństwa stało się proste i tanie, dostępne nawet dla bardzo prostych zastosowań.

GPG pozwala wygenerować tzw. *fingerprint* dla każdego klucza. Jest to skrót kryptograficzny (zob. podrozdział 2.3.4) klucza, a więc informacja znacznie krótsza niż sam klucz, ale mogąca potwierdzić jego autentyczność. Czasami istnieje konieczność zweryfikowania autentyczności klucza publicznego pobranego z serwera czy przesłanego mailem w inny sposób niż poprzez podpis kogoś znanego. W takiej sytuacji innym kanałem komunikacji (np. SMSem czy podczas rozmowy telefonicznej) można przekazać jego *fingerprint*, który jest znacznie krótszy niż sam klucz.

Oprogramowanie GPG oferuje też możliwość szyfrowania i deszyfrowania algorytmami symetrycznymi. W tym celu należy podać klucz szyfrowania. Rozwiązanie to jest szybsze niż zastosowanie kryptografii asymetrycznej, może więc być z powodzeniem zastosowane do dużych zbiorów danych. W razie potrzeby klucze symetryczne można bezpiecznie ustalić za pomocą komunikacji zabezpieczonej algorytmami asymetrycznymi zaimplementowanymi w GPG.

2.4.6. Infrastruktura klucza publicznego (PKI)

W podrozdziale 2.4.4 opisano możliwe sposoby wykorzystania kryptografii asymetrycznej do wygenerowania podpisu elektronicznego. Prawidłowość takiego

podpisu może zostać zweryfikowana przez każdą osobę dysponującą odpowiednim kluczem publicznym. Podstawowy problem polega jednak na uzyskaniu pewności, że klucz publiczny, którym dysponujemy, jest w istocie kluczem publicznym osoby, której podpis chcemy zweryfikować, nie zaś kluczem podrobionym przez osobę trzecią. Podobnie, chcąc wysłać poufną wiadomość zaszyfrowaną kluczem publicznym odbiorcy, musimy mieć pewność, że jest to prawidłowy klucz publiczny tej osoby, a nie podsunięty przez intruza podczas ataku *man-in-the-middle*. W niektórych przypadkach możliwe jest zweryfikowanie prawdziwości klucza przez porównanie jego skrótu obliczonego lokalnie ze skrótem oryginalnego, prawidłowego klucza (tzw. *fingerprint* klucza). *Fingerprint* musi zostać przekazany innym, bezpiecznym kanałem komunikacyjnym (przykład opisano dla PGP w podrozdziale 2.4.5). Istotną wadą tej metody jest konieczność wcześniejszego istnienia osobnego bezpiecznego kanału komunikacji, co w praktyce znacznie ogranicza jej stosowalność.

Istnieje inne rozwiązanie tego problemu. Wykorzystuje ono takie same mechanizmy, jak opisane wcześniej PGP (podrozdział 2.4.5), ale w inny sposób. Zamiast potwierdzenia autentyczności klucza publicznego poprzez różne osoby i przekazywania w ten sposób zaufania tworzy się instytucje odpowiedzialne za autoryzowanie informacji dołączonych do klucza publicznego.

Rozważmy następujący przypadek. Osoba A pragnie potwierdzić autentyczność klucza publicznego osoby C. Załóżmy, że z pewnych względów bezpośrednie sprawdzenie w opisany poprzednio sposób jest niemożliwe do zrealizowania. Załóżmy także, że osoba A dysponuje kluczem publicznym instytucji B i ma pewność co do autentyczności tego klucza. Jeżeli teraz instytucja B dysponowałaby autentycznym kluczem publicznym osoby C, to mogłaby podpisać ten klucz swoim kluczem prywatnym i przesłać do A. Taki klucz publiczny wzbogacony o informację o tożsamości właściciela i podpisany przez zaufaną osobę trzecią nazywamy *certyfikatem*. Osoba A jest w stanie zweryfikować prawdziwość podpisu instytucji B i – ufając B – potwierdzić prawdziwość klucza C. Opisany przypadek można znacznie rozszerzać, możliwy jest ciąg wielu podpisów, z których każdy uwiarygadnia kolejny klucz, aż do klucza docelowego. Przyjęcie wiarygodności finalnego klucza opiera się na posiadaniu autentycznego klucza pierwszej osoby, istnieniu nieprzerwanego łańcucha podpisów i zaufaniu do wszystkich podpisujących.

Do wykonania prawnie akceptowanego podpisu elektronicznego potrzebne są certyfikaty, których autentyczność została formalnie poświadczona przez powołaną do tego instytucję. Dlatego zamiast zdecentralizowanej sieci poświadczeń (jak w PGP) w takich celach jest budowana hierarchiczna *infrastruktura klucza publicznego* PKI (ang. *Public Key Infrastructure*). Infrastruktura taka jest rozumiana jako zbiór jednostek (osób, urzędów, instytucji), procedur, polityk bezpieczeństwa i mechanizmów pozwalających na wystawianie certyfikatów poświadczających

tożsamość osób czy też firm. Użytkownik, chcąc poświadczyć autentyczność swojego klucza publicznego, przesyła go do właściwej instytucji, która po sprawdzeniu tożsamości (w sposób prawny, np. przez weryfikację dokumentów) wystawia odpowiedni certyfikat składający się z przesłanego klucza publicznego oraz potwierdzonych danych osoby czy instytucji. Całość jest podpisana kluczem prywatnym urzędu, co potwierdza autentyczność i integralność certyfikatu – jeśli jakaś część zawartych w nim danych zostanie zmieniona, podpis urzędu przestanie „pasować" do całości i certyfikat stanie się nieważny.

Zazwyczaj można wyodrębnić osobne organy certyfikacyjne (CA, ang. *Certification Authority*) i rejestracyjne (RA, ang. *Registration Authority*). CA zajmuje się wystawianiem certyfikatów, jak też ich unieważnianiem (dawniej publikując tzw. listy unieważnionych certyfikatów CRL (ang. *Certificate Revocation List*), obecnie częściej umożliwiając ich sprawdzenie on-line poprzez OCSP (ang. *Online Certificate Status Protocol*), np. w razie ujawnienia klucza prywatnego, co pozwalałoby na podszycie się pod inną osobę. RA rejestruje wnioski o wydanie certyfikatu i sprawdza tożsamość wnioskodawcy.

Oczywiście PKI można stworzyć na swój prywatny użytek bądź dla firmy czy innej organizacji także za pomocą darmowego oprogramowania, do celów niezwiązanych z prawnie regulowanym podpisem elektronicznym. Jest to jednak nieco bardziej skomplikowane niż zdecentralizowana sieć PGP.

Warto podkreślić, że w wielu krajach prawodawstwo zrównuje podpis elektroniczny z podpisem odręcznym, wówczas także kształt odpowiedniej infrastruktury PKI poświadczającej autentyczność certyfikatu danej osoby jest uregulowany prawnie. W Polsce ustawa z dnia 18 września 2001 r. o podpisie elektronicznym (Dz.U. 2001 Nr 130, poz. 1450 wraz z późn. zm. [52]) wprowadziła pojęcie *podpisu kwalifikowanego* równoważnego z podpisem własnoręcznym. Podpis taki musi być składany za pomocą bezpiecznego urządzenia (aplikacji podpisującej i czytnika kart mikroprocesorowych), w którym znajduje się karta kryptograficzna z kluczem prywatnym danej osoby, a klucz publiczny jest potwierdzony ważnym certyfikatem kwalifikowanym.

2.4.7. Bezpieczna komunikacja

Bezpieczna komunikacja, czyli zapewniająca poufność i integralność danych oraz weryfikację tożsamości komunikujących się stron, możliwa jest do realizowania za pomocą metod kryptograficznych. Najczęściej dodatkowym wymaganiem w takiej sytuacji jest minimalizacja narzutu, jaki powoduje zapewnienie bezpieczeństwa.

Jak opisano we wcześniejszej części tego rozdziału, każde z wymagań bezpiecznej komunikacji może zostać spełnione przez rozwiązania oferowane przez kryptografię. Należy jedynie połączyć opisane mechanizmy, tak aby zestawić bez-

pieczny kanał komunikacyjny niegenerujący zbyt dużego narzutu. Narzut może dotyczyć dwóch istotnych parametrów – ilości przesyłanych danych oraz liczby wykonywanych obliczeń (na ogół czasu procesora).

Kryptografia asymetryczna wraz z dodatkowym rozwiązaniem w postaci PKI czy PGP zapewnia wygodny i bezpieczny sposób ustalenia tożsamości komunikujących się stron (co opisano w podrozdz. 2.4.3). Może też zapewnić bezpieczną komunikację – przesyłane dane można szyfrować kluczem publicznym odbiorcy i mieć pewność, że tylko właściwy adresat będzie mógł je odszyfrować. Niestety rozwiązanie to powoduje duży narzut obliczeniowy – kryptografia asymetryczna wykorzystuje bardziej złożone operacje matematyczne. Szyfrowanie symetryczne jest znacznie szybsze, jego wadą jest trudność ustalenia współdzielonego klucza szyfrowania. Można więc połączyć zalety obu rozwiązań – za pomocą kryptografii asymetrycznej zweryfikować tożsamość komunikujących się stron oraz ustalić i przesłać tajny klucz służący do szyfrowania symetrycznego. Klucza tego można używać do zapewnienia efektywnej i poufnej wymiany nawet dużej ilości danych. Oczywiście na tym etapie należy zastosować mechanizmy opisane w podrozdz. 2.4.2, aby zapewnić integralność danych podczas ich transmisji.

Taka koncepcja może być zrealizowana „ręcznie" w postaci komunikacji mailowej i narzędzia GPG. Jest ona jednak implementowana w całości w protokołach, takich jak TLS [38] (szerzej opisany w podrozdziale 3.8.3), powszechnie stosowanych np. do zapewnienia bezpiecznej komunikacji przez www. Protokół ten wykorzystuje większość opisanych mechanizmów, gdyż poza wspomnianymi w poprzednim akapicie korzysta także z PKI, aby zweryfikować poprawność certyfikatów prezentowanych podczas rozpoczynania komunikacji. Zarówno serwery www, jak i przeglądarki przechowują listy certyfikatów zaufanych instytucji (*Certificate Authorities*) i ufają jedynie certyfikatom, których tożsamość potwierdza jedna z tych instytucji. W przypadku pojawienia się podczas nawiązywania połączenia niezaufanego certyfikatu wyświetlane jest ostrzeżenie. Niezaufany certyfikat może oznaczać, że administrator serwera nie wykupił certyfikatu u zaufanej instytucji. Może to jednakże być skutek próby przeprowadzenia przez kogoś ataku *man-in-the-middle*.

2.5. Podsumowanie

Jak opisano w tym rozdziale, metody kryptograficzne mogą zapewnić bardzo wysoki poziom bezpieczeństwa. Są one też powszechnie dostępne dzięki bardzo dobremu darmowemu i komercyjnemu oprogramowaniu. Jak wszystko co dotyczy bezpieczeństwa, metody te także należy stosować świadomie, gdyż niewielkie odstępstwo od określonych reguł może spowodować dramatyczny spadek poziomu bezpieczeństwa. Wystarczy zignorować ostrzeżenie przeglądarki o nieważnym

certyfikacie podczas wchodzenia na stronę banku internetowego, aby paść ofiarą ataku pozwalającego intruzowi odczytać całą transmisję danych.

Pojęcie bezpieczeństwa jest ściśle związane z zaufaniem. Trudno zrealizować politykę bezpieczeństwa opartą na założeniu „nie ufam nikomu". PKI wymaga zaufania do instytucji poświadczających autentyczność certyfikatu, PGP wymaga zaś zaufania do osób podpisujących klucze publiczne i poświadczających ich poprawność. Warto zauważyć, że w obu tych rozwiązaniach przyjęcie autentyczności klucza opiera się na zaufaniu do wszystkich podpisujących. Niekiedy rodzi to problemy – fakt, że prawdziwość pewnego klucza poświadcza osoba, której autentyczność poświadczenia możemy zweryfikować, jeszcze nie znaczy że jej ufamy, że poświadczyła zgodnie z prawdą. Dodatkowo relacja zaufania między ludźmi nie jest przechodnia – jeśli X ufa Y, a Y ufa Z, to niekoniecznie implikuje, że X ufa Z, a konieczny jest wymóg zaufania do wszystkich podpisujących w łańcuchu. Problem ten jest wspólny dla wielu rozwiązań potwierdzających autentyczność klucza.

Podobnie, dopuszczenie użytkownika do systemu informatycznego (zezwolenie na zalogowanie się) wymaga zaufania do tego użytkownika, tj. że nie nadużyje udostępnionych mu funkcji systemu. Poziom tego zaufania objawia się w postaci nadanych mu uprawnień. Nawet jeśli działania tego użytkownika są przez kogoś kontrolowane, konieczne jest zaufanie do kontrolującego. Należy więc pamiętać, że zdefiniowanie dobrej *polityki bezpieczeństwa* wymaga określenia właściwej *polityki zaufania*.

Na ogół zapewnienie wysokiego poziomu bezpieczeństwa skutkuje utrudnieniami. Nie tylko w przypadku systemów informatycznych występuje ścisła zależność pomiędzy wygodą użytkowania a zapewnionym poziomem bezpieczeństwa. Do pewnego poziomu zależność ta może być niezauważana albo akceptowana, w pewnym jednak momencie może się okazać, że podniesienie poziomu bezpieczeństwa skutkuje istotnym ograniczeniem wygody. Podobnie jest z kosztami zabezpieczeń. Należy pamiętać, że w ostatecznych sytuacjach skutkiem tych zależności jest to, że nie istnieje „absolutne bezpieczeństwo". Bezpieczeństwo zawsze będzie kwestią porównania kosztów zabezpieczeń z zyskiem, jaki ktoś może osiągnąć, przełamując te zabezpieczenia. Dlatego proste systemy o niewielkim zakresie odpowiedzialności będą zabezpieczane znacznie prostszymi, tańszymi metodami, co będzie też skutkować wygodą użytkowania.

Należy także pamiętać, że często najsłabszym ogniwem jest człowiek. Zbyt duże ograniczenie wygody użytkownika przez paranoiczną politykę bezpieczeństwa może paradoksalnie skutkować zmniejszeniem bezpieczeństwa. Przykładowo, jeśli administrator narzuci hasła nie krótsze niż 30 znaków, które muszą zawierać znaki specjalne, cyfry, małe i duże litery i które muszą być zmieniane co tydzień i nie może się powtórzyć żadne z haseł z ostatnich pięciu lat, to przypuszczalnie

większość użytkowników będzie zapisywać aktualne hasło w mało bezpieczny sposób, np. na kartce przyklejonej do monitora i widocznej dla wszystkich wokół. W ten sposób wysiłki zmierzające do zapewnienia wysokiego poziomu bezpieczeństwa zostaną skutecznie zniweczone.

Specyficzne wymagania różnych systemów informatycznych oraz zastosowania w szczególnych rozwiązaniach wymagają odpowiedniego podejścia i wykorzystania odpowiednich mechanizmów zapewniających bezpieczeństwo. Z tego względu kolejne rozdziały tej pracy zawierają dyskusję aspektów bezpieczeństwa odpowiednią dla ich specyfiki. Część z przytoczonych zagadnień w jakimś stopniu została omówiona w tym rozdziale, ale część z nich ze względu na swoją specyfikę nie została tutaj ujęta.

2.6. Zadania

1. Zdefiniuj krótko poufność, integralność, uwierzytelnianie, autoryzację.
2. Jaka jest różnica między kryptografią symetryczną a asymetryczną?
3. Porównaj szyfry strumieniowe i blokowe.
4. Scharakteryzuj wybrane tryby działania szyfrów blokowych.
5. Dlaczego hasła nie powinny być przechowywane w sposób jawny?
6. Dlaczego prosta suma kontrolna nie jest wystarczającym sposobem zabezpieczenia integralności przesyłanych danych przed celową modyfikacją przez osoby trzecie?
7. Co to jest podpis elektroniczny?
8. Zdefiniuj krótko pojęcie certyfikatu.
9. Co to jest infrastruktura klucza publicznego?
10. Jak zrealizować bezpieczną i efektywną transmisję dużej liczby danych za pomocą metod kryptograficznych?
11. W jaki sposób jest potwierdzana tożsamość właściciele klucza publicznego w PGP, a jak w PKI? Wskaż różnice.
12. Jak zrealizować uwierzytelnianie za pomocą kryptografii asymetrycznej?
13. Jak zrealizować uwierzytelnianie za pomocą kryptografii symetrycznej?
14. Jakie mechanizmy kryptograficzne pozwalają zapewnić integralność danych?

Rozdział 3.

Sieci komputerowe

Sławomir Samolej, Wojciech Rząsa, Dariusz Rzońca

3.1. Wprowadzenie

Sieć komputerowa to zbiór niezależnych komputerów, które mogą się ze sobą komunikować [5, 48]. Medium komunikacyjnym mogą być przewody miedziane, światłowody lub fale elektromagnetyczne (radio, mikrofale, podczerwień, łączność satelitarna).

Jednym z zastosowań sieci komputerowej jest *współużytkowanie zasobów*. Komputery połączone w sieć mogą korzystać z uwspólnionych drukarek, skanerów, plików oraz innych urządzeń. Fundamentem sieci komputerowych jest również tak zwany układ *klient-serwer*. W sieci wyróżnia się komputer-serwer (na którym pracuje np. oprogramowanie bazodanowe) mogący dostarczyć komputerom-klientom odpowiednie dane lub usługi. Zaletą takiego scentralizowanego podejścia jest łatwość utrzymania spójności danych po stronie serwera. Stacje klienckie nie muszą również być zdolne do przeprowadzania wydajnych obliczeń. Alternatywą dla rozwiązania klient-serwer są techniki dostępu do informacji typu *„każdy z każdym”* (ang. *peer-to-peer*). W takiej równorzędnej sieci dane są rozproszone i dostarczane członkom sieci w razie potrzeby. Wprowadzenie technologii sieciowej pozwala również na konstruowanie rozproszonych centrów obliczeniowych. Odpowiednie oprogramowanie komunikacyjne umożliwia przekształcenie zbioru komputerów połączonych za pomocą sieci w *system do prowadzenia obliczeń współbieżnych*.

W miarę rozwoju technologii sieć komputerowa stała się *globalnym medium komunikacji*. Lokalne sieci komputerowe zostały połączone w *Internet* – sieć złożoną z sieci. Opierając się na podstawowych mechanizmach sieciowego przesyłania strumieni danych, plików lub komunikatów, opracowano *pocztę elektroniczną*, *komunikatory*, *strony WWW* (ang. *World Wide Web*), a później szereg aplikacji, np. *bankowość elektroniczna, sieci społecznościowe, handel elektroniczny, systemy gier sieciowych, przesyłanie głosu przez Internet – VoIP* (ang. *Voice over IP*). Obecnym trendem jest włączanie w globalną sieć różnych urządzeń przemysłowych

i domowego użytku, a także całych domów, kompleksów mieszkaniowych czy miast. Urzeczywistnia się koncepcja integracji jak największej liczby urządzeń połączonych w globalną sieć komputerową tworzących nowy technologiczny ekosystem, który zapewnia optymalizację zużycia energii i zaawansowane zarządzanie ośrodkami miejskimi, a nawet krajami.

W niniejszym rozdziale zostanie dokonany przegląd kluczowych zagadnień związanych z sieciami komputerowymi. Po przeprowadzeniu klasyfikacji sieci zostanie wprowadzony tak zwany warstwowy model sieci formułujący zasady konstruowania urządzeń i protokołów sieciowych. Opierając się na modelu, zostaną omówione kolejno wybrane media, techniki, protokoły oraz aplikacje służące do efektywnego przesyłania informacji w sieciach komputerowych. W ostatniej części zostaną przedstawione wybrane zagadnienia bezpieczeństwa odnoszące się do sieci komputerowych.

W trakcie wprowadzania kolejnych treści Czytelnik znajdzie również odwołania do dwu typów dokumentów normalizujących techniki konstruowania urządzeń sieciowych oraz metod komunikacji. Pierwszymi z nich są normy IEEE (ang. *Institute of Electrical and Electronics Engineers*), drugimi – standardy RFC (ang. *Request For Comments*). Wszystkie dokumenty IEEE i RFC, na które powołano się w tym rozdziale, są dostępne nieodpłatnie w zasobach Internetu.

3.2. Klasyfikacja sieci

Jak dotąd nie udało się wprowadzić jednolitej klasyfikacji sieci komputerowych. Zwykle dokonuje się ich podziału ze względu na technologię, w której następuje przesył informacji lub na odległość pomiędzy połączonymi komputerami. W sieciach komputerowych stosuje się dwie podstawowe technologie przesyłu: *rozgłoszeniową* i *dwupunktową*. W technologii dwupunktowej (ang. *unicasting*) przesył informacji następuje dokładnie pomiędzy jedną parą komputerów. W przesyle może pośredniczyć wiele innych komputerów pośrednich, a jego trasa nie musi być zawsze taka sama. Kluczową rolę odgrywają tutaj algorytmy wyszukiwania odpowiedniej ścieżki przesyłu danych. W technologii rozgłoszeniowej, stosowanej między innymi w sieciach bezprzewodowych, pakiety danych są wysyłane do wszystkich potencjalnych odbiorców. Jeśli odbiorca stwierdzi, że dane są zaadresowane do niego, to je odbiera, w przeciwnym razie ignoruje je. W systemach rozgłoszeniowych istnieje również możliwość przesłania danych do wszystkich węzłów sieci (ang. *broadcasting*) lub do grupy komputerów (ang. *multicasting*).

Dobrym kryterium podziału sieci komputerowych może być również odległość, w jakiej znajdują się węzły sieci. Parametr ten rzutuje zwykle również na techniki przesyłania danych w takiej sieci. W tablicy 3.1 podzielono sieci ze względu

Tablica 3.1. Klasyfikacja sieci według skali

Odległość między węzłami	*Węzły położone w tym samym*	*Przykład*
1 m	metrze kwadratowym	sieć osobista
10 m	pomieszczeniu	sieć lokalna
100 m	budynku	
1 km	grupie budynków	
10 km	mieście	sieć miejska
100 km	kraju	sieć rozległa
1000 km	kontynencie	
10 000 km	planecie	Internet

na rząd odległości pomiędzy węzłami oraz podano przykłady rodzajów sieci, do których takie grupy węzłów mogą należeć.

Sieci osobiste (PAN - ang. *Private Area Network*) łączą wiele urządzeń używanych przez jedną osobę. Dobrym przykładem sieci pozwalającej na taką integrację jest *Bluetooth*[1], który może posłużyć do połączenia mobilnych urządzeń komputerowych używanych przez daną osobę (np. smartfona ze słuchawkami, systemem obsługi telefonu w samochodzie czy urządzeniami technologii ubieranej) lub do „podłączenia" urządzeń wejścia-wyjścia do zestawu mikrokomputerowego (np. drukarka, klawiatura, mysz).

Sieć lokalna (LAN - ang. *Local Area Network*) łączy komputery znajdujące się w jednym lub kilku sąsiadujących budynkach. Może to być sieć urządzeń należących do jednego gospodarstwa domowego, sieć jednego przedsiębiorstwa lub sieć jednego z jego działów. Sieć taka służy do współdzielenia zasobów (np. drukarek, przestrzeni dyskowej, skanerów). Komputery są w niej połączone zwykle z zastosowaniem kabli miedzianych, światłowodów lub łączności bezprzewodowej.

Sieć miejska (MAN - ang. *Metropolitan Area Network*) łączy komputery znajdujące się w jednym mieście. Budowa sieci miejskiej jest oparta na miejskiej infrastrukturze telekomunikacyjnej, integrując sieci stosujące wyodrębnione pasma w łączach telefonicznych, telewizji kablowej czy radiowej.

Sieci rozległe (WAN - ang. *Wide Area Network*) obejmują swym obszarem cały kraj, a nawet kontynent. Wymagają one zwykle innych technik przesyłania sygnałów niż wcześniej omawiane sieci. W transmisji muszą uczestniczyć specjalne urządzenia (np. routery) pozwalające na przesłanie pakietów z danymi pomiędzy szeregiem komputerów pośrednich znajdujących się w różnych punktach kraju czy kontynentu. Bezpośrednią komunikację pomiędzy dwoma komputerami w sieciach rozległych można uzyskać poprzez zainwestowanie we własne linie teletransmi-

[1] `http://www.bluetooth.com`

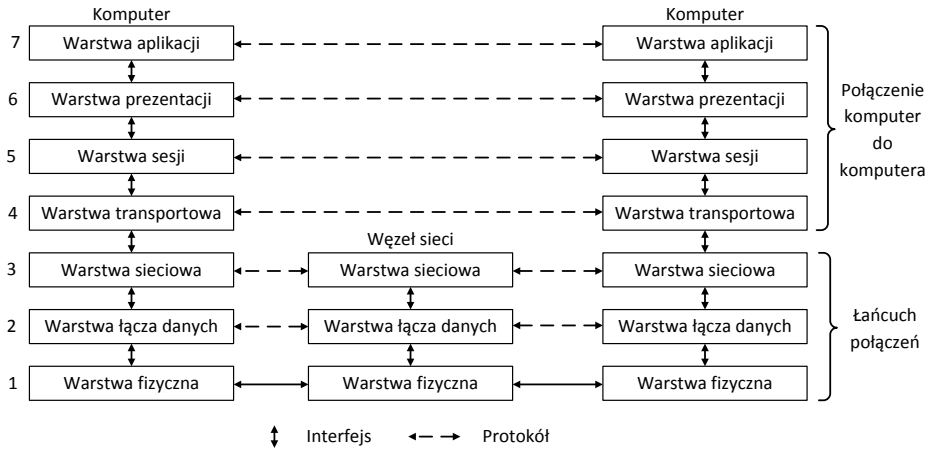
syjne, opłacenie dzierżawy łącz od jednego dostawcy dostępu do Internetu lub zastosowanie oprogramowania tworzącego wirtualną sieć prywatną (VPN - ang. *Virtual Private Network*).

Internet jest siecią złożoną z wielu sieci. Integruje on wszystkie sieci komputerowe, które chcą być do niego podłączone w jeden ogólnoświatowy organizm sieciowy. Umożliwia on przesyłanie informacji pomiędzy dowolnymi dwoma komputerami na Ziemi i w przestrzeni okołoziemskiej.

3.3. Model warstwowy sieci OSI

3.3.1. Zasada działania modelu warstwowego

W 1983 roku Międzynarodowa Organizacja Normalizacyjna (ISO - ang. *International Organization for Standardization*) zaproponowała wzorzec łączenia systemów otwartych (OSI - ang. *Open Systems Interconnection*), a więc gotowych na komunikację. Technologię wymiany informacji z zastosowaniem oprogramowania i urządzeń sieciowych podzielono na siedem warstw. Rysunek 3.1 prezentuje schemat opracowanego modelu.



Rys. 3.1. Model odniesienia OSI

Komunikacja w sieci jest zorganizowana w postaci stosu warstw. Zadaniem kolejnej warstwy jest dostarczenie warstwom wyższym określonych *usług* bez podawania informacji, jak dane usługi zostały zaimplementowane. Każda warstwa może być rozumiana jako swego rodzaju maszyna wirtualna oferująca usługi (por. rys. 3.1). Zmiana implementacji jednej z warstw (np. wymiana kart sieciowych z obsługujących połączenia z zastosowaniem kabli miedzianych na takie, które

obsługują połączenia światłowodowe) nie powinna mieć żadnego wpływu na pracę pozostałych warstw. Pozwala to na ewolucyjną wymianę oprogramowania i sprzętu przy zachowaniu ciągłości działania infrastruktury komunikacji sieciowej jako całości. Istotną właściwością takiego podejścia jest to, że podczas komunikacji wymiana informacji następuje pomiędzy odpowiadającymi sobie warstwami. Przykładowo, rozpoczęcie nadawania na poziomie warstwy transportowej strumienia danych z jednego komputera powinno skutkować przejęciem tego strumienia danych na poziomie warstwy transportowej drugiego komputera. Reguły wymiany informacji na poziomie danej warstwy noszą nazwę *protokołów* (por. rys. 3.1). Na rysunku 3.1 pokazano, że logiczna komunikacja odbywa się pomiędzy odpowiadającymi sobie warstwami (partnerami). Mogą to być programy, urządzenia lub nawet ludzie. Fizycznie dane nie są jednak przenoszone pomiędzy równorzędnymi warstwami. W rzeczywistości każda warstwa przesyła informację do warstwy znajdującej się poniżej, uzupełniając je odpowiednimi informacjami sterującymi. Dane z warstwy najniższej są przekazywane za pomocą nośnika fizycznego (kabla, światłowodu lub fal elektromagnetycznych) do warstwy najniższej innego składnika sieci, a tam z kolei są propagowane do kolejnych warstw wyższych. W niektórych urządzeniach następuje tylko wzmocnienie sygnału na poziomie warstwy fizycznej, inne analizują częściowo przesyłany strumień danych, aby go skutecznie skierować do adresata. Na rysunku 3.1 wirtualne przekazywanie informacji pomiędzy warstwami przedstawiono za pomocą linii przerywanych, natomiast realne zaznaczono liniami ciągłymi. Pomiędzy poszczególnymi warstwami są zdefiniowane szczegółowe *interfejsy* (por. rys. 3.1) opisujące, które operacje i usługi są dostarczane przez warstwę niższą na rzecz warstwy wyższej.

Warstwa fizyczna nazywana warstwą sprzętową zajmuje się fizycznym przesyłaniem bitów w kanale informacyjnym. Jej zadaniem jest przesłanie kolejnych *bitów* danych pomiędzy komputerami tworzącymi sieć lokalną. Na poziomie tej warstwy ustala się poziomy napięć w kablu miedzianym, kolory i intensywność światła w światłowodzie, częstotliwości i metody modulacji fal elektromagnetycznych w łączności bezprzewodowej, a także parametry czasowe przesyłanych sygnałów. Warstwa ta definiuje również standardy złączy, interfejsy mechaniczne i elektryczne, zgodnie z którymi mają pracować fizyczne urządzenia sieciowe.

Warstwa łącza danych odpowiada za *niezawodne* przesłanie danych pomiędzy komputerami tworzącymi sieć lokalną. W tym celu dane są dzielone na fragmenty zwane *ramkami*. Każda ramka jest przesyłana osobno. W przesyłaniu ramek stosuje się matematyczne metody wykrywania błędów (np. sumy kontrolne), techniki potwierdzania, że ramka dotarła oraz retransmisję uszkodzonych ramek. W tej warstwie mogą zostać również zdefiniowane techniki *kontroli przepływu*, czyli metody powiadamiania nadawcy, że nadaje zbyt szybko, aby można było bez utraty odbierać jego dane.

Warstwa sieciowa jest odpowiedzialna za przesłanie *pakietów* danych pomiędzy dwoma komputerami, które potencjalnie mogą się znaleźć w dowolnych miejscach na Ziemi. Na poziomie tej warstwy są definiowane techniki wyznaczania tras oraz kontroli przepływu w sieciach rozległych. Na tym poziomie muszą być również rozstrzygnięte problemy przesłania pakietu danych z zastosowaniem sieci o różnej architekturze, formacie i szybkości przesyłania informacji. Warstwa *nie zapewnia niezawodnego przesyłania danych*.

Na poziomie *warstwy transportowej* dane otrzymywane z warstwy wyższej są formułowane w pakiety. Warstwa ta jest odpowiedzialna za *niezawodny* przesył tych pakietów pomiędzy dwoma komputerami znajdującymi się w dowolnym miejscu na Ziemi. W tym celu są sformułowane specjalne protokoły wymiany danych pomiędzy komputerami znajdującymi się potencjalnie w dwu różnych sieciach komputerowych.

Warstwa sesji pozwala na ustanowienie połączenia pomiędzy aplikacjami pracującymi na komunikujących się komputerach. Zapewnia utrzymanie połączenia lub jego wznowienie, jeśli zostało przerwane.

Warstwa prezentacji odpowiada za konwersję (jeśli jest potrzebna) sposobu reprezentacji danych. Okazuje się, że różne architektury komputerów w różny sposób reprezentują znaki oraz liczby całkowite i zmiennopozycyjne. Warstwa udostępnia mechanizmy uniwersalnego kodowania danych dla transportu w sieci oraz możliwość „powracania" do kodowania charakterystycznego dla danej architektury.

Na poziomie *warstwy aplikacji* są zdefiniowane protokoły sieciowe, którymi posługują się programy użytkowników. Przykładami takich protokołów są HTTP (ang. *HyperText Transfer Protocol*) do przesyłania zawartości stron WWW (ang. *World Wide Web*) czy SMTP (ang. *Simple Mail Transfer Protocol*) do przesyłania wiadomości pocztowych. Inne protokoły umożliwiają przesyłanie plików, list dyskusyjnych itp. Za tą warstwą kryją się również po prostu programy użytkowników korzystające z łączności sieciowej.

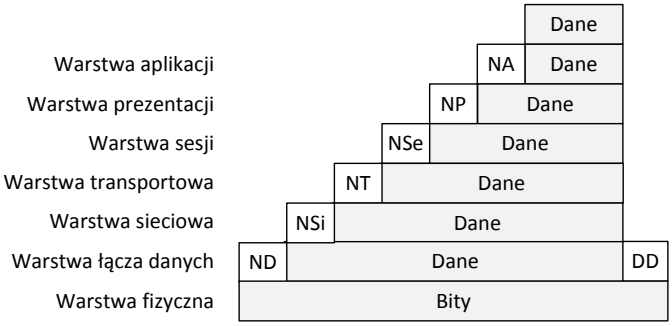
W niniejszej pracy przyjęto, że poszczególne urządzenia i protokoły sieciowe będą omawiane zgodnie z modelem ISO/OSI. Ze względu na ograniczony rozmiar pracy zostaną przedstawione jedynie najważniejsze z nich.

3.3.2. Opakowywanie danych

Dane przechodząc w dół stosu warstw, są opakowywane (ang. *encapsulated*) w dodatkowe struktury danych po to, aby ułatwić ich zrozumienie dla warstw niższych oraz by można je było efektywnie przekazać do warstw wyższych u odbiorcy. Ta modyfikacja polega zwykle na:

- dodaniu nagłówka lub zakończenia do porcji przesyłanych danych,

- przekodowaniu wybranych sekwencji bitów, które mogłyby być inaczej zrozumiane,
- przestawieniu kolejności przesyłania danych.



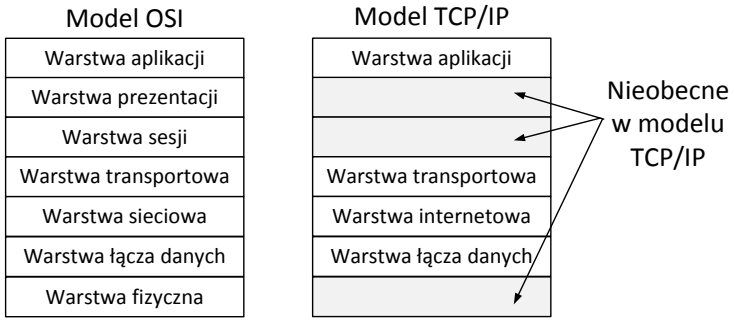
Rys. 3.2. Możliwy scenariusz opakowywania danych

Na rysunku 3.2 przedstawiono możliwy scenariusz opakowywania (enkapsulacji) danych. Przykładowo, aplikacja pocztowa uzupełnia treść naszego listu odpowiednim nagłówkiem (NA - nagłówek warstwy aplikacji) wskazującym adresata oraz nadawcę. Warstwa prezentacji dodaje swój nagłówek (NP - nagłówek warstwy prezentacji) i dokonuje (jeśli potrzeba) odpowiednich konwersji sposobu reprezentacji danych. Z kolei warstwa sesji dokłada kolejny nagłówek (NSe - nagłówek warstwy sesji) i utrzymuje połączenie z odpowiednim programem pocztowym na czas przesyłania danych. Warstwa transportowa dodaje swój nagłówek (NT - nagłówek warstwy transportowej) i zapewnia bezbłędny przesył danych pomiędzy komputerem łączącym się do systemu pocztowego a serwerem usług pocztowych. Warstwa sieciowa uzupełnia dane z wyższej warstwy o swój nagłówek (NSi - nagłówek warstwy sieciowej) i pozwala na efektywne przesłanie pakietów w sieci globalnej. Warstwa łącza danych kontroluje prawidłowy przesył ramek w danej podsieci lokalnej. Dołącza do danych nagłówek (ND) pozwalający na przemieszczenie informacji w danej sieci oraz dodatkowe dane (DD) na końcu, umożliwiające weryfikację poprawności transmisji w danej podsieci. Tak „opakowane" dane są przenoszone do warstwy transportowej, która przekazuje je jako strumień bajtów do łącza fizycznego.

3.3.3. Model warstwowy ISO a model TCP/IP

Model warstwowy OSI był istotnym wyznacznikiem zasad tworzenia sieci komputerowych. Wprowadzanie nowych rozwiązań technicznych w sieciach komputerowych nadal odnosi się do tego modelu. W praktyce jednak dominującą

rolę odgrywa model odniesienia TCP/IP. Na rysunku 3.3 porównano wymienione modele odniesienia.



Rys. 3.3. Porównanie modeli warstwowych ISO i TCP/IP

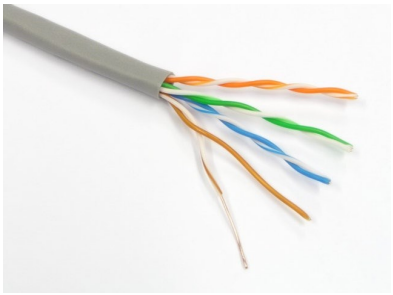
Warstwa *łącza danych* nie jest szczegółowo opisywana. Jej zadaniem jest umożliwienie przesyłania danych w pojedynczym łączu sieci. *Warstwa internetowa* stanowiąca odpowiednik warstwy sieciowej OSI ma za zadanie wygenerowanie w dowolnej sieci zbioru pakietów z ustaloną informacją i zapewnienie, że większa część z nich dotrze do adresata – innego komputera usytuowanego w dowolnym miejscu na Ziemi lub przestrzeni okołoziemskiej. Zdefiniowany tu jest oficjalny format pakietu *IP* (ang. *Internet Protocol*) oraz protokół pomocniczy *ICMP* (ang. *Internet Control Message Protocol*). *Warstwa transportowa* odpowiadająca warstwie transportowej OSI zapewnia bądź niezawodny mechanizm przesyłania strumienia bajtów pomiędzy komputerami z zastosowaniem protokołu *TCP* (ang. *Transmission Control Protocol*), bądź zawodny protokół *UDP* (ang. *User Datagram Protocol*) dla aplikacji w jakiś sposób gotowych na utratę części nadsyłanych danych (np. odbieranie strumieni audio i video). W modelu TCP/IP zrezygnowano z warstw sesji i prezentacji, sugerując, że ich role powinny spełniać już same aplikacje. Bezpośrednio na warstwie transportowej zaproponowano *warstwę aplikacji*. W warstwie tej początkowo zaproponowano protokół realizujący wirtualny terminal (telnet), protokół umożliwiający transfer plików (FTP – ang. *File Transfer Protocol*) oraz pocztę elektroniczną (SMTP – ang. *Simple Mail Transfer Protocol*). Z czasem ten zbiór protokołów został rozszerzony o usługę odwzorowywania nazw serwerów na ich adresy w sieci (DNS – ang. *Domain Name System*), protokół pobierania stron internetowych (HTTP – ang. *Hypertext Transfer Protocol*), protokół dostarczania mediów (głos, wideo) w czasie rzeczywistym (RTP – ang. *Real-time Transport Protocol*) i wiele innych.

3.4. Warstwa fizyczna i warstwa łącza danych

3.4.1. Nośniki informacji

We współczesnych sieciach komputerowych z powodzeniem koegzystuje wiele nośników informacji. W sieciach lokalnych i metropolitalnych stosuje się odpowiednio skonstruowane kable miedziane. W sieciach lokalnych dominującym rozwiązaniem jest tak zwana skrętka. Istnieją również rozwiązania stosujące kable koncentryczne, techniki przesyłania danych oparte na okablowaniu sieci energetycznych oraz okablowaniu miedzianych systemów telekomunikacji głosowej. Innym coraz częściej stosowanym nośnikiem, zwłaszcza w transmisjach na dalsze odległości, jest światłowód. Coraz większe znaczenie odgrywa także tak zwana łączność bezprzewodowa stosująca wiele technik modulacji fal elektromagnetycznych do przesyłania informacji. W dalszej części podrozdziału szczegółowo omówiono wybrane nośniki.

Skrętka jest nadal jednym z najczęściej stosowanych nośników transmisji, zwłaszcza w sieciach lokalnych. Składa się z dwu izolowanych przewodów lub drutów miedzianych o średnicy ok. 1 mm skręconych ze sobą w helisę, podobnie jak cząsteczka DNA. Skręcanie przewodów jest niezbędne, ponieważ wygłusza fale elektromagnetyczne wytwarzane oraz odbierane przez parę kabli. Sygnał przesyłany przez parę kabli jest zwykle kodowany przez zmianę różnicy napięć pomiędzy przewodami. Obecnie najczęściej stosowanym typem skrętki jest kabel kategorii 5. (CAT 5). Składa się on z dwu izolowanych przewodów lekko skręconych, zwykle grupowanych po cztery pary we wspólnej powłoce. Na rysunku 3.4 pokazano przykładowy kabel z czterema parami skręconych przewodów. Kable

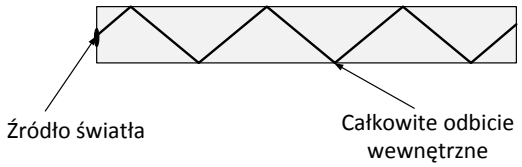


Rys. 3.4. Skrętka z czterema parami przewodów

takie są różnie stosowane w rozwiązaniach sieci lokalnych. Sieci Ethernet o przepustowości 100 Mb/s stosują dwie pary (z czterech) skrętki do transmisji danych, po jednej w każdym kierunku. Standard Ethernet 1 Gb/s wykorzystuje wszystkie cztery pary skrętek, każda do transmisji w obie strony. Nowsze typy skrętki to

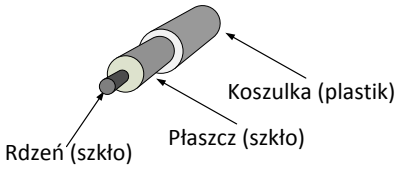
kategorie 6. i 7. Kategoria 6. ma budowę podobną do kategorii 5., ale zastosowane materiały i jakość wykonania zapewniają kablowi większą przepustowość i możliwość transmisji z prędkością 10 Gb/s. Wykonanie kabla w kategorii 7. zakłada dodatkowe ekranowanie każdej z par przewodów oraz całego kabla, co zwiększa odporność na przesłuchy i zakłócenia zewnętrzne.

Światłowody stosuje się do długodystansowych transmisji w sieciach rozległych, w szybkich lub niepodatnych na zakłócenia elektromagnetyczne sieciach lokalnych oraz w rozwiązaniach szybkiego dostępu do Internetu (tzw. światłowód do domu, FttH – ang. *Fiber to the Home*). System transmisji światłowodowej składa się z trzech podstawowych składników: źródła światła, medium transmisyjnego oraz detektora. Impuls światła oznacza sygnał o wartości „1”, brak światła to bit „0”. Jeżeli medium transmisyjnym jest odpowiednie ultracienkie przejrzyste włókno szklane i promień światła wpada do niego pod odpowiednim kątem, to następuje w nim zjawisko całkowitego odbicia wewnętrznego (por. rys. 3.5). Promień światła



Rys. 3.5. Zjawisko całkowitego wewnętrznego odbicia

w takim włóknie może być przenoszony na odległość wielu kilometrów praktycznie bez strat. Jeśli włókno światłowodowe umożliwia wprowadzanie promieni pod wieloma kątami, mówi się, że jest to światłowód wielomodowy. Inny rodzaj światłowodu o grubości zredukowanej do kilku długości fali świetlnej zachowuje się jak falowód. Umożliwia on wprowadzenie jednego promienia świetlnego poruszającego się w linii prostej. Taki światłowód jest nazywany jednomodowym. Współczesne włókna jednomodowe umożliwiają transfer danych z prędkością 100 Gb/s na odległość 100 km bez wzmacniania.

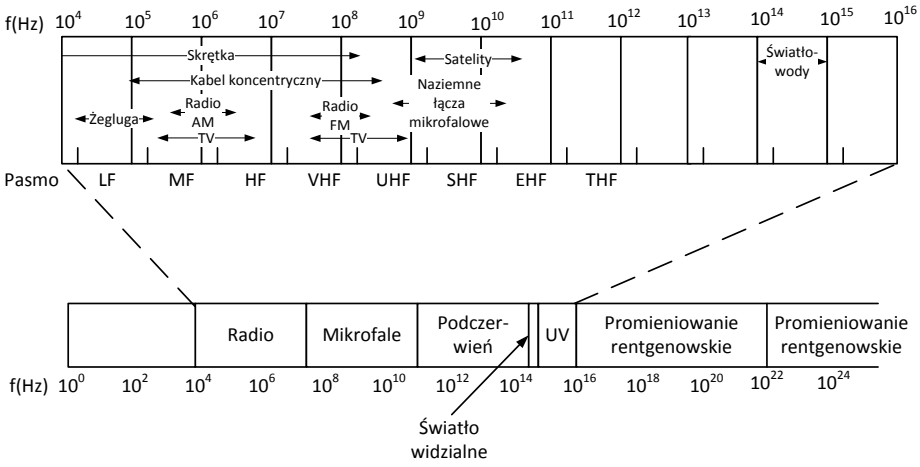


Rys. 3.6. Konstrukcja włókna światłowodowego

Kabel światłowodowy (por. rys. 3.6) składa się z rdzenia wykonanego ze szkła otoczonego przez płaszcz ze szkła o innym współczynniku załamania niż w rdzeniu, co powoduje, że światło pozostaje w rdzeniu. Koszulka z tworzywa sztucznego chroni płaszcz. Włókna światłowodowe łączy się w wiązki chronione dodatkową otuliną zewnętrzną. Światłowody wielomodowe mają rdzenie zwykle o przekroju 50 mikronów, a jednomodowe – od 8 do 10 mikronów. Emiterami światła dla światłowodów są diody świecące lub lasery, odbiornikami – fotodiody.

Wydaje się, że światłowody w najbliższej przyszłości staną się dominującymi trwałymi nośnikami transmisji. Mogą przesyłać dane bez wzmacniania na odległości dziesięciokrotnie dalsze niż kable miedziane, są niewrażliwe na przepięcia czy zakłócenia elektromagnetyczne, nie ulegają korozji, są lżejsze od kabli, a także trudne do „podsłuchania". Barierami do ich powszechnego zastosowania są relatywnie wyższa cena oraz konieczność posiadania specjalistycznego sprzętu i umiejętności do łączenia instalacji światłowodowych.

Fale elektromagnetyczne od bardzo dawna służą człowiekowi do przekazywania informacji. Sygnały świetlne w postaci zapalanych na wzgórzach ognisk służyły do powiadamiania o nadchodzącym zagrożeniu w wielu starożytnych cywilizacjach. W końcu XIX wieku przeprowadzono pierwszą udaną transmisję radiową. Od tego czasu opanowano technikę przekazywania informacji z zastosowaniem fal elektromagnetycznych o różnych częstotliwościach. Na rysunku 3.7 pokazano widmo fal elektromagnetycznych z zaznaczonymi przedziałami częstotliwości stosowanymi obecnie w różnych technikach przenoszenia informacji.



Rys. 3.7. Zastosowania widma elektromagnetycznego w komunikacji

W większości przypadków transmisji dane nadaje się w ustalonym wąskim zakresie pasma częstotliwości. Pozwala to na osiągnięcie wysokiej efektywności przesyłu w stosunku do zastosowanej mocy nadawczej. Istnieją również rozwiązania stosujące szersze pasma, w których przekazuje się na raz większą liczbę informacji (por. technologie Bluetooth lub CDMA). Jak wynika z rys. 3.7, większość bezprzewodowych transmisji cyfrowych odbywa się w paśmie mikrofalowym. Okazuje się, że fale elektromagnetyczne powyżej częstotliwości 100 MHz rozchodzą się niemal w linii prostej, co pozwala je łatwo kierować, a nawet skonstruować systemy nadawczo-odbiorcze złożone z kilku nadajników/odbiorników ustawionych w rzędzie. Problemem jest ograniczona przenikliwość tych fal przez budynki, a od częstotliwości powyżej 4 GHz - ich pochłanianie w środowisku wodnym. W zakresie mikrofalowym działają systemy telefonii cyfrowej, systemy komunikacji satelitarnej oraz lokalne sieci bezprzewodowe. Jako medium komunikacyjne stosuje się również sygnały świetlne wysyłane przez lasery oraz fale podczerwone (stanowiące powszechny nośnik wymiany informacji pomiędzy pilotami a telewizorami i odtwarzaczami), służące do integracji urządzeń znajdujących się blisko stanowiska komputerowego (por. standard IrDA).

Częstotliwości fal elektromagnetycznych, z jakich korzystają cyfrowe urządzenia komunikacyjne, w większości państw są ustalane na drodze prawnej. Przykładowo, dużym sukcesem międzynarodowym było ustalenie częstotliwości 900 MHz, 1800 MHz oraz 1900 MHz dla komunikacji w sieciach GSM oraz przyjęcie pasm 2,4 GHz oraz 5 GHz jako pasm do swobodnego użytku na potrzeby lokalnych sieci bezprzewodowych i innych urządzeń o niewielkiej mocy nadawczej.

3.4.2. Ethernet

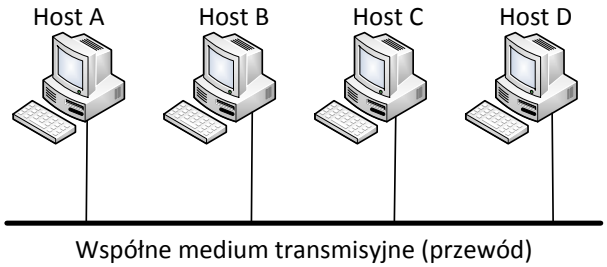
Nośniki informacji są w stanie przekazywać dane zakodowane w postaci odpowiednich fal. Warstwa fizyczna sieci komputerowych zajmuje się formułowaniem tych fal w celu efektywnego przesyłania kolejnych bitów informacji. Warstwa łącza danych organizuje ciągi bitów w ramki, umożliwiając zorganizowaną i niezawodną transmisję porcji danych w segmencie sieci działającym w jednolitej technologii. Dla trwałych nośników informacji na poziomie łącza danych współistnieje wiele protokołów. Najczęściej stosowane to ADSL (ang. *Asymmetric Digital Subscriber Line*) umożliwiający komunikację sieciową z zastosowaniem łączy telefonicznych, SONET (ang. *Synchronous Optical NETwork*) stosowany w rozległych sieciach z zastosowaniem okablowania światłowodowego oraz Ethernet, który zdominował rynek sieci lokalnych. Wraz z protokołami ADSL oraz SONET współpracuje nadrzędny dla nich protokół PPP (ang. *Point-to-Point Protocol*), który pozwala na efektywne przesyłanie pakietów protokołu IP. Natomiast technologia Ethernet sama w sobie oferuje możliwość transferu pakietów IP i nie wymaga żadnych nad-

rzędnych protokołów. Protokół Ethernet bardziej szczegółowo opisano w dalszej części podrozdziału.

Warto zwrócić uwagę, że obecnie są używane dwie odmiany sieci Ethernet operującej na trwałych nośnikach informacji. W wersji klasycznej zakłada się, że komputery dysponują wspólnym medium i same rozwiązują problem dostępu do niego. *Klasyczny Ethernet* wychodzi z użycia, chociaż techniki dostępu do łącza danych są ponownie wdrażane w jego wersji bezprzewodowej, która zostanie omówiona w następnym podrozdziale. Umożliwia on komunikację z prędkościami od 3 do 10 Mb/s. Drugim typem, dominującym obecnie w sieciach lokalnych, jest *Ethernet przełączany* (ang. *swithed Ethernet*). Do połączeń między komputerami stosuje się w nim przełączniki (ang. *switch*), które zarządzają ruchem w sieci, eliminując powszechne w klasycznym rozwiązaniu kolizje. Różne typy przełączanego Ethernetu oferują transmisję 100, 1000, a nawet 10000 Mb/s.

Klasyczny standard Ethernet został opracowany w 1982 roku jako przedsięwzięcie konsorcjum firm DEC, Intel i Xerox. Po kilku latach został opublikowany standard IEEE 802.3 [15] porządkujący założenia Ethernet. Jest on systematycznie rozwijany wraz z postępem techniki komunikacyjnej i uwzględnia rozszerzenia związane z technikami przełączanymi. Ethernet i IEEE 802.3 różnią się w niektórych szczegółach, ale w niniejszym opracowaniu będą traktowane jednakowo. Podstawowymi technikami klasycznego Ethernet są metoda *nasłuchu łącza* (ang. *carrier sense*) oraz *dostępu wielokrotnego* (ang. *multiple access*) z *detekcją kolizji* (ang. *collision detection*) – CSMA/CD. Standard definiuje również postać ramki danych – odpowiedniej sekwencji bitów umożliwiającej efektywną komunikację pomiędzy komputerami połączonymi łączem zgodnym z IEEE 802.3.

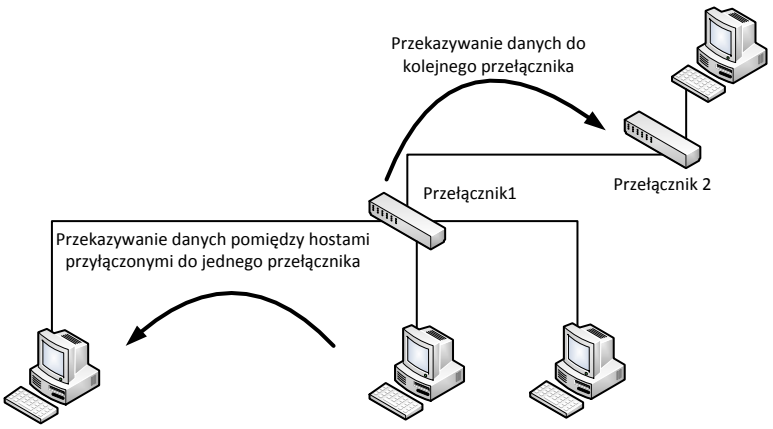
Metoda CSMA/CD zakłada, że wszystkie węzły sieci są połączone za pomocą tego samego medium, w którym z chwilą rozpoczęcia nadawania sygnału następuje jego bardzo szybkie rozchodzenie. Jeśli rozważy się zbiór komputerów (hostów) podłączonych do pojedynczego medium, jak na rys. 3.8, to metodę CSMA/CD można wyjaśnić na przykładzie w następujący sposób: W przypadku gdy komputer



Rys. 3.8. Klasyczna sieć Ethernet

A zamierza wysłać ramkę danych do komputera B, ale okazuje się, że w danej chwili komputer C wysyła dane do komputera D, to komputer A odkłada transmisję. Dzieje się tak, ponieważ komputer zamierzający rozpocząć nadawanie *nasłuchuje łącze*. Jeśli komputer nasłuchujący (A) stwierdzi, że łącze jest wolne, to rozpoczyna nadawanie. Może się jednak zdarzyć, że inny komputer (np. B) również w tym samym czasie rozpocznie swoją transmisję. Ponieważ podczas wysyłania pakietów komputery nadal nasłuchują łącze, wykryją one *kolizję* i przerwą swoje nadawanie, a następnie każdy z nich po odczekaniu losowego czasu podejmie ponowną próbę dostępu do łącza. Jak się okazuje, taka prosta metoda polegająca na nasłuchiwaniu, wykrywaniu kolizji oraz ponownych próbach nadawania danych (w przypadku niepowodzenia transmisji po upłynięciu losowego odcinka czasu) skutecznie rozwiązuje problem zarządzania przepływem danych w sieciach lokalnych o niezbyt dużym natężeniu ruchu.

Sieciowe urządzenia dla klasycznego Ethernetu są nazywane *koncentratorami* lub *hubami*. Są one w zasadzie jedynie wzmacniaczami sygnałów elektrycznych rozchodzących się we współdzielonym medium komunikacyjnym. Zostały opracowane koncentratory dla klasycznych sieci Ethernet, w których medium były kable koncentryczne lub skrętka. Urządzenia te nie są już wprowadzane w nowych rozwiązaniach sieciowych, ponieważ nie nadają się do konstrukcji sieci dla Ethernetu przełączanego. Ethernet przełączany nie wykorzystuje metody CSMA/CD,

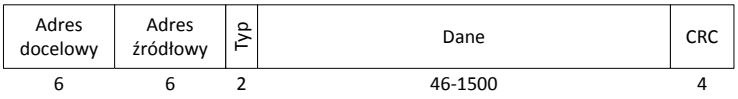


Rys. 3.9. Kierowanie ruchem w sieci przez przełączniki

ponieważ w strukturach sieci komputerowych opartych na przełącznikach nie następują kolizje. Ruchem ramek w sieci zarządzają *przełączniki*, które identyfikują nadawcę oraz odbiorcę i przesyłają strumień bitów przez odpowiedni kanał informacyjny. W odróżnieniu od koncentratorów przełączniki sprawdzają, do jakiego

komputera mają być przesłane dane. Dysponują również skonfigurowaną przez użytkownika lub automatycznie wygenerowaną tablicą adresów Ethernet. W konsekwencji, jeśli nadawca i adresat są przyłączeni do tego samego przełącznika, to ruch w sieci ograniczany jest tylko do przekazywania ramek w jednym przełączniku. Jeśli przełącznik „stwierdzi", że dane nie należą do żadnego z komputerów do niego przyłączonych, wyśle je do kolejnego przełącznika. Na rysunku 3.9 pokazano zasadę kierowania ruchem ramek w sieciach opartych na przełącznikach. Ruch w sieci o podobnej konfiguracji, ale stosującej koncentratory powodowałby rozprzestrzenianie się nadawanych ramek do wszystkich segmentów sieci.

Podstawowe składniki ramki Ethernet przedstawiono na rys. 3.10. Pierwsze dwa pola ramki o długości 6 bajtów każde zawierają odpowiednio jej *adres docelowy* i *adres źródłowy*. Każda karta sieciowa pracująca w standardzie Ethernet ma swój unikalny 48-bitowy (6-bajtowy) adres w skali całego świata (adres MAC). Dzięki temu przesłanie ramki danych pomiędzy dwoma urządzeniami w takim



Rys. 3.10. Ramka Ethernet

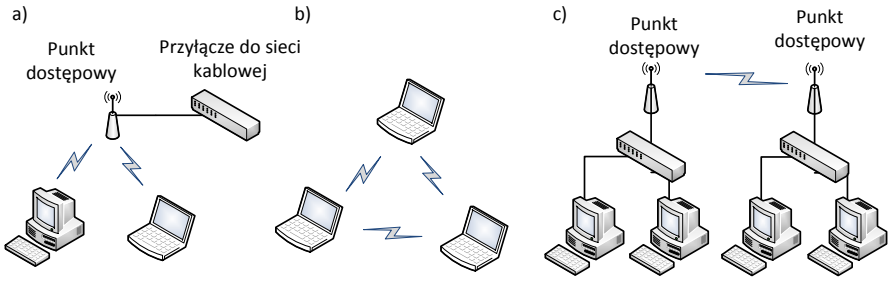
standardzie wymaga jedynie wskazania, kto dane chce przesłać i do kogo dane mają być wysłane. Dwubajtowe pole *typu* zawiera unikalną liczbę określającą *rodzaj przesyłanych danych* („stary" standard Ethernet) lub *długość danych* (standard 802.3). Kolejne pole ramki zawiera faktycznie przesyłane *dane* o długości od 48 do 1500 bajtów. Ostatnie czterobajtowe pole zawiera obliczoną z ramki tak zwaną *sumę kontrolną* (CRC - ang. *Cyclic Redundancy Check*). Jest to funkcja matematyczna przekształcająca dowolny bazowy ciąg bitów w ciąg o ustalonej długości, ale o takiej właściwości, że z dużym prawdopodobieństwem zmiana ciągu bazowego (np. w wyniku błędów transmisji) będzie skutkować zmianą ciągu o ustalonej długości. W konsekwencji, jeśli po przesłaniu ramki i obliczeniu z niej sumy kontrolnej okazuje się, że wartość otrzymanej sumy jest inna niż wartość sumy przesłanej, to można stwierdzić, że dane uległy przekłamaniu.

Jak już wcześniej zaznaczono, technologia Ethernet była konstruowana głównie z myślą o sieciach lokalnych. Rozwiązania techniczne pozwalają na przyłączanie za pomocą skrętki komputerów odległych od koncentratora czy przełącznika do 100 m. Zasięg przesyłu danych pomiędzy urządzeniami w standardzie Ethernet może sięgnąć nawet 40 km, jeśli jako medium zostanie zastosowany odpowiedni (jednomodowy) światłowód. Standard 802.3 obejmuje bardzo duży wachlarz rozwiązań umożliwiających przesył danych z prędkościami 10, 100, 1000, 10000, a nawet 40000 Mb/s. Dobranie właściwego rozwiązania do potrzeb danego przed-

siębiorstwa wymaga rozważenia wielu zagadnień technicznych i inwestycyjnych. Naturalną tendencją jest systematyczny spadek cen urządzeń i okablowania w miarę upowszechniania się danego standardu przesyłu danych. Wiele przełączników oferuje możliwość integracji urządzeń posiadających różne prędkości działania kart sieciowych. Do konstruowania tanich lokalnych sieci działających do prędkości 1 Gb/s nadal powszechnie stosuje się skrętkę. Do łączenia bardziej odległych budynków czy oddziałów firm są rozważane połączenia komunikacyjne z zastosowaniem światłowodów. Warto również zaznaczyć, że technologia Ethernet służy obecnie do tworzenia sieci urządzeń przemysłowych (por. standard EtherCAT lub PROFINET), a nawet jest medium komunikacyjnym na pokładach samolotów pasażerskich (por. standard ARINC 664 [3]).

3.4.3. Sieci bezprzewodowe

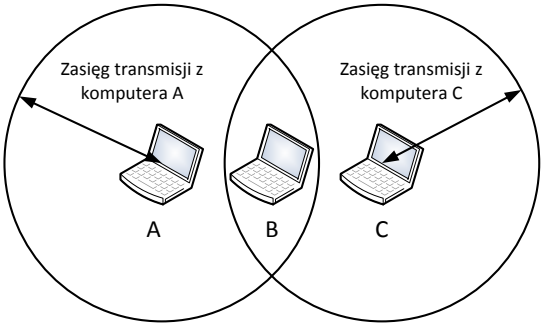
Lokalne sieci bezprzewodowe stały się składnikiem naszego życia codziennego. Korzystamy z nich w biurach, restauracjach i domach. Wydają się skutecznie wypierać sieci z trwałymi nośnikami informacji. Najczęściej stosowanym standardem takich sieci jest 802.11, nazywany też bezprzewodowym Ethernetem. Typowa konfiguracja takich sieci to *punkt dostępowy* (ang. *access point*), nazywany też stacją bazową, który zarządza bezprzewodową komunikacją z klientami (komputerami personalnymi, laptopami, smartfonami itp.). Punkt dostępowy jest z kolei podłą-



Rys. 3.11. Konfiguracje sieci bezprzewodowych: a) sieć bezprzewodowa z punktem dostępowym, b) sieć ad hoc, c) bezprzewodowy odcinek sieci przewodowej

czony do sieci za pomocą połączenia kablowego. W standardzie IEEE 802.11 [16] dopuszcza się stworzenie infrastruktury połączonych punktów dostępowych i takie jej skonfigurowanie, aby klient – przemieszczając się z jednego pomieszczenia do drugiego – był „przyjmowany” przez kolejne urządzenia. Zasięg takich sieci to ok. 100 m w pomieszczeniach i 300 m w terenie otwartym. Standard nie przewiduje jednak takiej mobilności, jak sieci telefonii komórkowych, które umożliwiają

prowadzenie wymiany informacji przez mobilnego klienta poruszającego się samochodem. Rzadziej stosowaną formą działania sieci bezprzewodowych są tak zwane *sieci ad hoc* polegające na zestawieniu połączeń bezprzewodowych pomiędzy kilkoma klientami w celu bezpośredniej wymiany informacji. Współczesne punkty dostępowe można również skonfigurować jako *klienty* innych punktów dostępowych i w ten sposób tworzyć np. *bezprzewodowe odcinki sieci komputerowych*. Na rysunku 3.11 pokazano typowe konfiguracje stosujące połączenia bezprzewodowe. Łączność pomiędzy współczesnymi urządzeniami sieci bezprzewodowych odbywa się w pasmach 2,4 GHz (standardy 802.11b – prędkość przesyłu do 11 Mbs oraz 802.11g – prędkość przesyłu do 56 Mb/s) oraz 5 GHz (standard 802.11a – prędkość przesyłu do 56 Mb/s), uznanych przez społeczność międzynarodową za tak zwane „pasma uwolnione", czyli niezarezerwowane prawnie do jakichś szczególnych zastosowań (por. podrozdz. 3.4.1). Przesyłanie danych odbywa się z zastosowaniem kilku technik tak zwanego rozpraszania widma. Osiąga się wtedy możliwość przesyłu współbieżnie w jednej szczelinie czasowej od kilku do kilkudziesięciu bitów. Kanały przesyłu radiowego zyskują wtedy cechy komunikacji równoległej. W 2009 roku został opracowany nowy standard o identyfikatorze 802.11n, w którym zakłada się transfer do 600 Mb/s, przy zastosowaniu nowych urządzeń z czterema antenami każde.



Rys. 3.12. Ograniczona „widoczność" elementów sieci bezprzewodowej

W konstruowaniu sieci bezprzewodowych występuje wiele nowych problemów niespotykanych w przypadku sieci z trwałymi nośnikami informacji. Najważniejsze z nich to:

- wysoki współczynnik występowania błędów spowodowanych szumami elektromagnetycznymi występującymi w środowisku,
- realna szerokość pasma zależna od otoczenia i odległości między hostami,
- wzajemne zakłócanie się sieci w wyniku interferencji,

- niewielka moc transmisji ograniczona prawnie, co skutkuje również ograniczonym zasięgiem,
- niepełna wzajemna widoczność wszystkich uczestników ruchu w sieci (por. rys. 3.12, np. jeśli stacja A „widzi" stację B i stacja B „widzi" stację C, to wcale nie oznacza, że stacja A „widzi" stację C, co może być źródłem kolizji),
- pakiety w sieciach radiowych są z założenia niezabezpieczone i rozgłaszane w eterze, wówczas należy mieć na uwadze problemy z osiągnięciem bezpieczeństwa i uwierzytelnienia tak transmitowanych danych.

Dostęp do łącza w sieciach klasy 802.11* odbywa się zgodnie z protokołem CSMA/CA. Następuje w nim *wykrywanie fali nośnej* (ang. *Carrier Sense*) i próba *wielokrotnego dostępu* (ang. *Multiple Access*) oraz *zapobieganie kolizjom* (ang. *Collision Avoidance*). Próba podjęcia przesyłu danych przez hosta rozpoczyna się od nasłuchiwania stanu łącza. Jeśli łącze jest wolne, zostaje wysłany pakiet, jeśli nie, to host czeka na zakończenie aktualnej transmisji, a następnie oczekuje jeszcze przez losowo wybrany przedział czasu. W przypadku gdy łącze jest nadal wolne następuje wysłanie pakietu. W przeciwnym wypadku host rozpoczyna nasłuchiwanie łącza i podejmuje próbę ponownego wysłania pakietu.

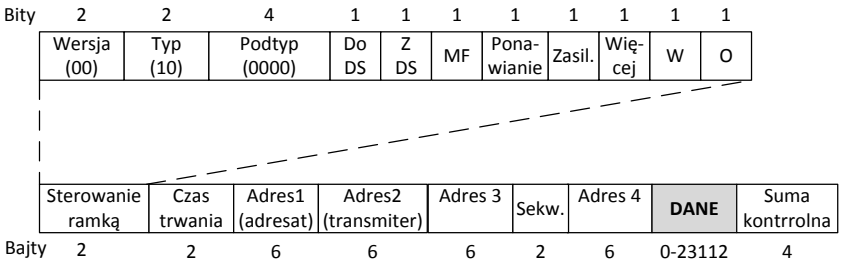
Dla usprawnienia działania podstawowego protokołu CSMA/CA w sieciach bezprzewodowych wprowadzono dodatkowe właściwości. Pierwsze rozszerzenie zakłada, że po odebraniu danych przez węzeł docelowy przesyła on pakiet potwierdzenia ACK (ang. *ACKnowledgment*). Jeśli nadawca nie otrzyma potwierdzenia,

Rys. 3.13. Transfer danych z zastosowaniem CSMA/CA wraz z rozszerzeniami

uznaje, że jego dane nie dotarły i ponawia ich transmisję. Problemy z ograniczoną widocznością rozwiązuje się drugim rozszerzeniem polegającym na wprowadzeniu do protokołu dodatkowych ramek potwierdzeń RTS/CTS (ang. *Request To Send/Clear To Send*). Kiedy nadawca chce rozpocząć przekazywanie danych, wysyła do odbiorcy ramkę z prośbą o zezwolenie na wysyłanie – RTS. Jeśli odbiorca jest gotowy do odbioru danych, potwierdza to wysłaniem ramki CTS. Taka inicjalizacja transmisji wymianą krótkich ramek informuje wszystkie hosty w zasięgu

nadawcy i odbiorcy, że nastąpi między nimi komunikacja, co powoduje odroczenie wysyłania ramek przez innych nadawców. Dodatkowo wszystkie rodzaje ramek zawierają informację o długości danych, które będą przesyłane, co pozwala na ustalenie przez inne hosty wstępnego czasu oczekiwania zanim rozpoczną procedurę dostępu do łącza. Na rysunku 3.13 pokazano przykładowy scenariusz przesyłania danych z zastosowaniem protokołu CSMA/CA z uwzględnieniem omówionych rozszerzeń. Wymiana danych następuje pomiędzy hostami A i B. Hosty C i D zawieszają próby podjęcia dostępu do łącza na czas zgłoszonej przez komputery A i B sesji. W środowiskach, w których następują silne zakłócenia sygnałów, przewidziano możliwość automatycznej zmiany prędkości nadawania sygnałów. Realna prędkość nadawania bitów jest taka sama, natomiast wydłużane jest ich kodowanie, co zwiększa odporność na zakłócenia.

W standardzie 802.11 zdefiniowano trzy klasy ramek: ramki danych, ramki sterujące oraz ramki zarządzania. Na rysunku 3.14 pokazano ramkę danych. Pierw-



Rys. 3.14. Format ramki danych w standardzie 802.11

szym polem ramki jest *sterowanie ramką* składające się z 11 podrzędnych pól. Pierwsze pole podrzędne zawiera *wersję protokołu* (ustawianą obecnie na 00). Pole podrzędne *typ* wskazuje, czy przesyłaną ramką jest ramka danych, ramka sterująca lub ramka zarządzająca. Pole podrzędne *podtyp* pozwala na doprecyzowanie typu ramki (CTS czy RTS). W przypadku przesyłania zwykłych danych pole *typ* przyjmuje wartość 10, a pole *podtyp* – 0000 (zapisane w systemie dwójkowym). Bity pól podrzędnych *Do DS* i *Z DS* wskazują, czy ramka przychodzi z sieci, do której jest przyłączony punkt dostępowy (zwanej punktem dystrybucyjnym), czy do niej podąża. Bit *MF* (ang. *More Fragments*) wskazuje, że za bieżącym fragmentem będą nadawane kolejne. Ustawienie bitu *ponowienie* wskazuje, że dana ramka jest retransmitowana. Jeśli ramka zawiera ustawiony bit *zarządzanie zasilaniem*, to nadawca przechodzi w tryb oszczędzania energii. Przesłanie do niego danych będzie wymagało dodatkowego oczekiwania na „obudzenie". Jeśli bit *Więcej* jest ustawiony, to nadajnik chce wysłać jeszcze kolejne ramki do odbiornika. Bit *W* przekazuje informację, że przesyłane dane są zaszyfrowane. Z kolei

bit *O* przekazuje informację, że wyższa warstwa sieci spodziewa się przesłania uporządkowanej sekwencji ramek.

Zawartość drugiego pola ramki o nazwie *czas trwania* informuje, ile czasu zajmie przesył danej ramki wraz z potwierdzeniem (w mikrosekundach). Jak już wspomniano, pole to jest podstawą do obliczania dodatkowych okresów wyczekiwana na dostęp do łącza.

Kolejne trzy pola to adresy. *Adres1* jest adresem odbiorcy. Jeśli bit *Do DS* jest ustawiony, jest to adres punktu dostępowego. Jeśli jest ustawiony bit *Z DS*, to jest to adres hosta. *Adres2* jest adresem transmitera. Jeśli bit *Z DS* jest ustawiony, jest to adres punktu dostępowego. Jeśli jest ustawiony bit *Do DS*, to jest to adres hosta. Jeśli *adres1* zawiera adres docelowego hosta, to *adres3* zawiera adres nadawcy. Podobnie, jeśli *adres2* zawiera adres źródłowy, to *adres3* zawiera adres docelowy. Interpretacja pola *adres4* jest następująca: Jeśli sieć zawiera bezprzewodowe połączenie punktu dostępowego z innym punktem dostępowym, który z kolei świadczy usługi dostępowe dla hostów, to *adres1* zawiera adres odbierającego punktu dostępowego, *adres2* – adres transmitera, *adres3* jest adresem docelowego hosta, a *adres4* – adresem hosta źródłowego.

Pole *sekw.* pozwala na numerowanie ramek i ułożenie ich w sekwencję. Pozwala także eliminować duplikaty ramek. Cztery bity tego pola identyfikują tak zwany fragment, a pozostałe 12 numer, który zwiększa się w każdej kolejnej transmisji.

Ustalono również, że pierwszy bajt pola danych ma format *LLC* (ang. *Logical Link Control*). Pozwala on na zidentyfikowanie, do jakiego protokołu warstwy wyższej (np. IP) należą dane przesyłane w ramce. Na końcu ramki znajduje się pole sumy kontrolnej (*CRC*), której rolę wyjaśniono w podrozdz. 3.4.2.

Ramki zarządzające mają postać identyczną z ramkami danych. Zawierają jednak odpowiednio sformatowane dane w polu danych. Z kolei ramki sterujące nie zawierają danych. Ich interpretacja odbywa się na podstawie pól *podtyp*.

Ze względu na specyfikę przesyłania informacji w sieciach bezprzewodowych standard 802.11 (dokładnie 802.11i) definiuje schemat zabezpieczeń transmisji. Po pierwsze, w komunikacji z punktem dostępowym można ustalić metodę autentykacji klienta, dopuszczając do wymiany danych tylko te urządzenia, które znają odpowiedni identyfikator i hasło. Po drugie, komunikacja pomiędzy węzłami sieci bezprzewodowej może być szyfrowana. Jednym z zalecanych schematów zabezpieczeń jest *WPA2* (ang. *WiFi Protected Access*), oferujący możliwość zarówno autentykacji klientów, jak i szyfrowania zgodnego ze standardem *AES* (ang. *Advanced Encryption Standard*). Poprzedni standard WEP (ang. *Wired Equivalent Privacy*) nie zapewnia elementarnego poziomu bezpieczeństwa i nie powinien być obecnie stosowany. Powszechnie dostępne oprogramowanie pozwala na złamanie klucza WEP w ciągu kilku minut.

Przesyłanie danych w formie bezprzewodowej na większe odległości niż w sieciach 802.11 wymaga zmiany jakościowej. Pierwszą barierą jest uzyskanie uniwersalnego pasma komunikacji na poziomie metropolii czy kraju, a więc pasma licencjonowanego. Pozwoli to na odpowiednie wzmocnienie sygnału i objęcie sieci większym obszarem. Jednym ze standardów, który definiuje technikę transmisji bezprzewodowej w sieciach metropolitalnych, jest IEEE 802.16 (*Wi-MAX*, ang. *Worldwide Interoperability for Microwave Access*). Warto również zwrócić uwagę, że obecnie są wdrażane standardy pozwalające tworzyć bezprzewodowe sieci metropolitalne, a w przyszłości krajowe, czy globalne. Jednym z nich jest standard telefonii komórkowej czwartej generacji LTE (ang. *Long Term Evolution*), która jest nastawiona na transmisję danych, a połączenia głosowe są jedną z usług.

Równolegle z lokalnymi sieciami bezprzewodowymi skutecznie koegzystują technologie tworzenia bezprzewodowych sieci osobistych, a więc rozciągniętych na odległość do kilku metrów. Podstawowymi standardami przemysłowymi tworzenia takich systemów, które zostały uznane przez rynek, są Bluetooth i RFID (ang. *Radio Frequency IDentification*). Standard Bluetooth umożliwia zbudowanie osobistej sieci komputerowej złożonej z bezprzewodowych urządzeń peryferyjnych komputera personalnego: klawiatury, myszy, drukarek czy aparatu fotograficznego. Jest on również stosowany do komunikacji pomiędzy smartfonami a urządzeniami tak zwanej elektroniki ubieranej (np. zegarki, okulary, obuwie i ubrania wyposażone w odpowiednie moduły elektroniczne) oraz do włączania smartfonów w systemy komputerowe pojazdów samochodowych (systemy głośnomówiące itp.). Standard RFID operuje na dwu typach urządzeń – znacznikach, które w niektórych wypadkach nie muszą być zasilane, oraz czytnikach, które z pewnej odległości są w stanie zinwentaryzować stan pobliskich znaczników. Znaczniki mają 96-bitowe identyfikatory oraz pewien niewielki obszar pamięci, którą można zapisać lub odczytać. Okazuje się, że na podstawie miniaturowych znaczników, które mogą być wtapiane w dokumenty, wszczepiane zwierzętom, naklejane na paczki czy produkty handlowe, oraz odpowiednich czytników można skutecznie zorganizować m. in. bezprzewodowe systemy zarządzania przedsięwzięciami logistycznymi, zarządzania dostępem do pomieszczeń, ochrony własności, identyfikacji dokumentów, zwierząt i przedmiotów.

3.5. Warstwa sieciowa

3.5.1. Projektowanie warstwy sieciowej

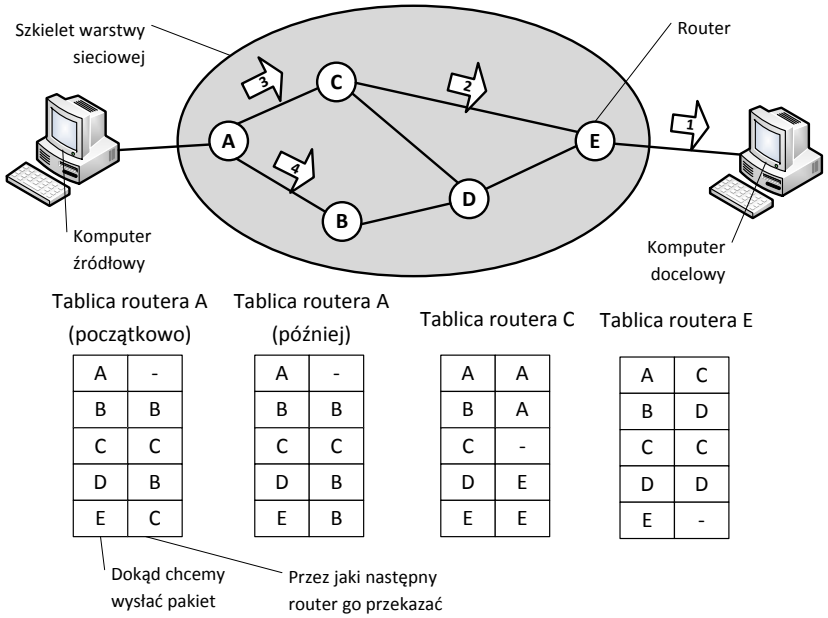
Gdyby standard Ethernet był jedynym obowiązującym na świecie, stworzenie globalnej sieci komputerowej nie stanowiłoby problemu. Unikalne adresy MAC urządzeń w skali całego świata znacznie by to ułatwiły. Jednak został on opracowany w czasie, gdy koncepcja i protokoły Internetu były na tyle dojrzałe, że

nie dały się zepchnąć na margines. Ponadto do czasu wdrożenia techniki światłowodowej Ethernet oferował metodę komunikacji na niewielkie odległości. Istotą projektowania warstwy sieciowej stało się opracowanie mechanizmu do globalnej komunikacji komputerów, przy założeniu że są one połączone za pomocą różnych standardów komunikacyjnych, nazywanych w modelu odniesienia TCP/IP warstwą łącza danych (por. podrozdz. 3.3.3). Przełomem było zaproponowanie systemu niescentralizowanego. Do tamtej pory użytkownicy komputerów łączyli się na odległość z pojedynczymi komputerami, a nie z ich siecią. Opracowanie warstwy sieciowej zostało powiązane z wprowadzeniem na rynek nowych komputerów komunikacyjnych – *routerów*. Miały to być urządzenia, które po przesłaniu do nich odpowiedniego pakietu z danymi umiałyby go między sobą przekazywać, aż do „znalezienia" podsieci, w której znajduje się komputer docelowy – adresat pakietu. Kluczowym mechanizmem warstwy sieciowej stały się tak zwane *algorytmy routingu* – techniki generowania tabel wskazujących, do jakiego kolejnego routera ma zostać przekazany pakiet, aby osiągnął komputer docelowy. Istotnym składnikiem warstwy sieciowej było również opracowanie nowego *ogólnoświatowego mechanizmu adresacji komputerów* oraz *formatu pakietów do przesyłania danych* w takiej niejednorodnej sieci. Założono przy tym, że pakiety mogą być *utracone* oraz że mogą zostać *pofragmentowane* na mniejsze jednostki, jeśli dana podsieć nie będzie w stanie przesłać ich w całości.

Dominującym rozwiązaniem warstwy sieciowej we współczesnych sieciach globalnych (i lokalnych) jest protokół *IP* (ang. *Internet Protocol*). Pozwala on na uniwersalne przekazywanie danych w ogólnoświatowej sieci Internet, z którą mogą się łączyć wszelkiego rodzaju systemy mikroprocesorowe, poczynając od komputerów personalnych, poprzez urządzenia mobilne, a kończąc na urządzeniach przemysłowych i domowego użytku.

3.5.2. Routing

Pakiety przesyłane w warstwie sieciowej posiadają jedynie adresy nadawcy i odbiorcy oraz „wskazówki" co do ważności przesyłanych danych. Nie zawierają jednak żadnych informacji, którędy miałyby być transportowane. Routery – podstawowe urządzenia komunikacyjne w warstwie sieciowej – „potrafią" na podstawie tych skąpych informacji wskazać trasę dla pakietów. Dodatkowo strumień danych podzielony na pakiety może w warstwie sieciowej wędrować różnymi trasami, jeśli routery uznają to za „korzystne". Na rysunku 3.15 przedstawiono historię przepływu czterech pakietów pomiędzy dwoma komputerami sterowaną przez routery *A*, *B*, *C*, *D* i *E*. W każdym routerze znajduje się tak zwana tablica routingu informująca, gdzie należy przesłać pakiet, jeśli ma dotrzeć do routera o określonym adresie. Przykładowo, w tablicy początkowej routera A widać, że pakiety adresowane do routera *E* mają zostać przekazane do routera *C*. Pokazany na rys. 3.15



A	-
B	B
C	C
D	B
E	C

A	-
B	B
C	C
D	B
E	B

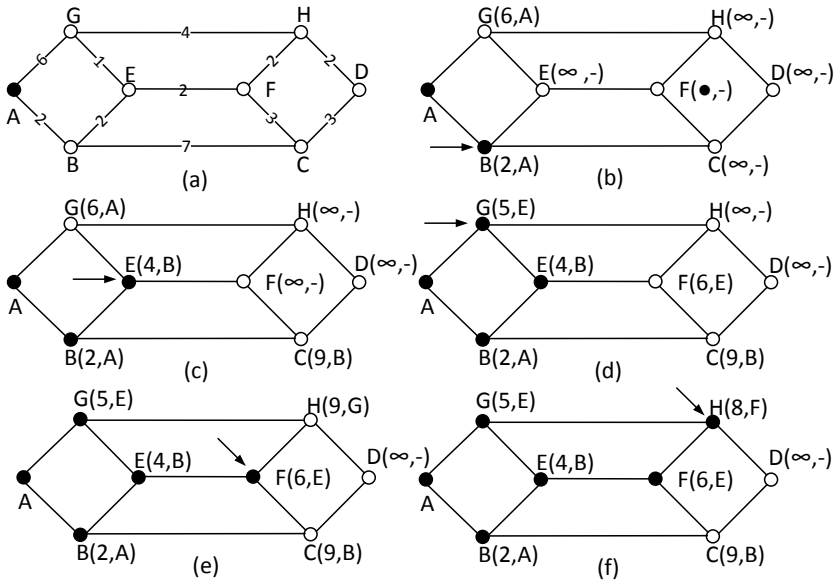
A	A
B	A
C	-
D	E
E	E

A	C
B	D
C	C
D	D
E	-

Rys. 3.15. Routing w warstwie sieciowej

scenariusz przepływu można zinterpretować następująco: dla pierwszych trzech pakietów routery „uznały", że najkorzystniejszą trasą będzie *A*, *C*, *E*. Ostatni pakiet na skutek zmiany w tablicy routingu routera *A* został skierowany do routera *B*. Przyczyną zmiany w tablicy mogło być wykrycie uszkodzenia lub przeciążenia routera *C*.

Największym wyzwaniem w projektowaniu warstwy sieciowej jest opracowywanie algorytmów wypełniania tablic routingu. Na wczesnych etapach rozwoju Internetu takie tablice można było wypełnić „ręcznie". Obecnie wypełnianiem tablic zajmuje się oprogramowanie routerów. Oprócz przekazywania pakietów od użytkowników routery rozsyłają własne pakiety, za pomocą których wymieniają się informacjami o swoich tablicach routingu oraz „poszukują" kolejnych ścieżek przekazywania pakietów. Istnieje wiele algorytmów wyznaczania trasy dla pakietów. Najważniejsze z nich to algorytm z wyborem najkrótszej ścieżki, routing rozpływowy, routing z użyciem wektorów odległości (stosowany w sieci ARPANET) i routing z użyciem stanów połączeń (stosowany obecnie w wielu sieciach). Szkice rozwiązań tych algorytmów można znaleźć w pracy [48]. W dalszej części podrozdziału zostanie zaprezentowany szkic rozwiązania algorytmu z wyborem najkrótszej ścieżki (tzw. algorytm Dijkstry) dla segmentu sieci komputerowej o znanej budowie.



Rys. 3.16. Pierwsze sześć etapów szukania najkrótszej ścieżki od węzła *A* do *D* z zastosowaniem algorytmu Dijkstry; strzałka wskazuje węzeł roboczy

Początkowym etapem algorytmu Dijkstry jest zbudowanie grafu, w którym węzeł oznacza router, a krawędź – linię łączącą go z następnym routerem. Krawędzie etykietuje się odległościami pomiędzy węzłami. Wyznaczenie trasy pomiędzy dwoma routerami polega na wyszukaniu najkrótszej ścieżki. Należy zaznaczyć, że odległość pomiędzy węzłami nie jest zawsze miarą rzeczywistej odległości liczonej w kilometrach. Może to być średnie opóźnienie w przesyle pakietu czy liczba przeskoków pomiędzy routerami, które musi wykonać pakiet, aby dotrzeć do wskazanego routera. Na początku działania algorytmu wszystkie węzły są oznaczone symbolem nieskończoności. W miarę postępu działania algorytmu etykiety węzłów mogą się zmieniać, odwzorowując lepsze ścieżki. Wartości początkowe etykiet są tymczasowe. Jeśli algorytm wykaże, że dana etykieta reprezentuje najkrótszą ścieżkę, to staje się ona trwała.

Na rysunku 3.16 przedstawiono pierwsze sześć kroków algorytmu Dijkstry, który poszukuje najkrótszej ścieżki pomiędzy routerem *A* i pozostałymi routerami opisanymi grafem. W pierwszym kroku algorytmu węzeł roboczy *A* jest oznaczany jako trwały (węzły trwałe są wskazywane przez zamalowane kółko, por. rys. 3.16a). Następnie sprawdzana jest odległość od węzła *A* do każdego jego sąsiada. Nowo obliczonymi odległościami etykietuje się sąsiednie węzły grafów, dołączając do ich etykiet również oznaczenie węzła, od którego odległość była liczona. Przeszukuje się wszystkie węzły z etykietami tymczasowymi w grafie i jako trwały wybiera ten,

którego etykieta ma najniższą wartość. Wyznaczony węzeł zostaje nowym węzłem roboczym.

W rozważanym przykładzie nowym węzłem roboczym został węzeł *B* (por. rys. 3.16b). Ponownie następuje sprawdzenie odległości pomiędzy węzłem roboczym a jego sąsiadami z wyłączeniem węzłów trwałych. Odległość zapamiętaną w węźle roboczym sumuje się z odległością od niego do badanego węzła-sąsiada. Jeśli nowo obliczona odległość jest mniejsza od zapisanej w etykiecie sąsiada, to została znaleziona krótsza ścieżka i należy uaktualnić etykietę sąsiada. Ponownie są przeszukiwane węzły grafu z tymczasowymi oznaczeniami w celu znalezienia węzła o najniżej etykiecie.

Omówiona procedura wskazywania kolejnego węzła roboczego, obliczania odległości pomiędzy węzłem a jego sąsiadami z nietrwałymi etykietami, ewentualnego przeetykietowania węzłów jest prowadzona do wyznaczenia ostatniego węzła trwałego. W rezultacie otrzymuje się zbiór najkrótszych ścieżek od wskazanego węzła do wszystkich węzłów w grafie. Posługując się wynikiem działania algorytmu, można przygotować tablice routingu dla określonej struktury routerów. Ograniczeniem algorytmu jest to, że działa dobrze w z góry określonej, zamkniętej strukturze połączeń pomiędzy routerami. Wspiera więc tak zwane protokoły *routingu wewnątrzdomenowego* pracujące w strukturach sieci należących do jednej organizacji. Routing na poziomie sieci rozległych tworzy się przez połączenie takich zamkniętych struktur w klastry, pomiędzy którymi działają tak zwane *międzydomenowe algorytmy routingu*.

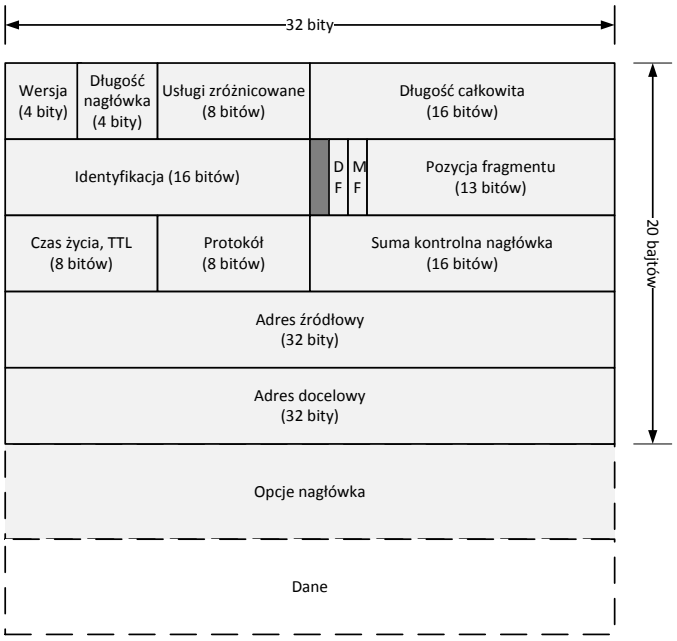
3.5.3. Protokół IPv4

Podstawowym protokołem komunikacyjnym warstwy sieciowej w większości sieci komputerowych jest *IP* (ang. *Internet Protocol*). Jak już wspomniano, umożliwia on przesłanie pakietu danych pomiędzy dwoma komputerami znajdującymi się w dowolnym położeniu strefy okołoziemskiej. W przesyłaniu danych w sieci Internet dominuje wersja formatu pakietu o numerze 4 opisana w dokumencie [27]. Nowa wersja o numerze 6, która jest stopniowo wdrażana, jest opisana w podrozdziale 3.5.4. Pakiety protokołu IPv4 (IP w wersji 4.) składają się z nagłówka pozwalającego na identyfikację oraz pola danych. Na rysunku 3.17 przedstawiono budowę takiego pakietu.

Pole *wersja* wskazuje, do jakiej wersji protokołu należy pakiet. Obecnie można się tam spodziewać wartości 4 lub 6 (por. podrozdział 3.5.4).

Pole *długość nagłówka* wskazuje, jak długi jest nagłówek pakietu w 32-bitowych słowach. Pole to wprowadzono, ponieważ nagłówek nie ma stałej długości i może zawierać do 40 bajtów tak zwanych wartości opcjonalnych.

Pole *usługi zróżnicowane* zostało przedefiniowane już w czasie stosowania pakietu IP i obecnie jest interpretowane w następujący sposób (por. [30]): Pierwsze



Rys. 3.17. Budowa pakietu IPv4

6 bitów pola wskazuje klasę obsługi. Przykładowo, *klasa przekazywania ekspresowego* wskazuje, że dany pakiet powinien przechodzić tak, żeby „wyprzedzać" pakiety zwykłe w każdym routerze. Natomiast *klasa przekazywania gwarantowanego* oferuje 4-priorytetową klasyfikację pakietów wraz z dodatkowymi mechanizmami odrzucania wybranych danych podczas przeciążeń sieci. Dwa ostatnie bity niosą jawną informacje, czy pakiet nie doznał przeciążenia. Pole *długość całkowita* zawiera informacje o liczbie bajtów całego pakietu (nagłówka i danych). Łatwo obliczyć, że maksymalny rozmiar pakietu może wynosić 65535 bajtów.

Pole *identyfikacja* zawiera unikalną liczbę całkowitą wskazującą bieżący pakiet. Jak już wcześniej wspomniano, pakiety w czasie ich transportu mogą być dzielone na mniejsze *fragmenty*. Wszystkie fragmenty danego pakietu są pakietami IP zawierającymi takie samo pole identyfikacji, co umożliwia ich ponowne scalenie.

Kolejny bit nagłówka jest niewykorzystywany. Następny bit nagłówka identyfikowany przez *DF*, stanowiący skrót od angielskiego *Don't Fragment*, informuje routery, czy dany pakiet może być fragmentowany. Ustawienie tego bitu oznacza, że routery mają poszukać takiej ścieżki przesyłu danego pakietu, w której nie nastąpi jego fragmentacja. Bit *MF*, którego nazwa jest skrótem od angielskiego wyrażenia *More Fragments*, jest ustawiany we wszystkich (za wyjątkiem ostatniego)

fragmentach danego pakietu. Umożliwia on sprawdzenie, czy do odbiorcy dotarły wszystkie fragmenty.

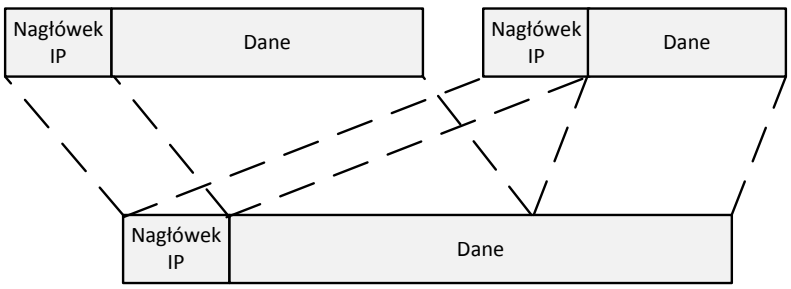
Pole *pozycja fragmentu* wskazuje, jaki odcinek informacji z pola danych jest przesyłany w danym fragmencie. Podczas fragmentacji pakietu następuje powielenie nagłówka pakietu oraz dołączenie do niego ustalonej części przesyłanych danych. Omawiane pole wskazuje, która część danych jest transportowana w danym fragmencie. Na rysunku 3.18 zilustrowano technikę fragmentacji pakietów. Połączenie informacji z pól *identyfikacja*, *MF* oraz *pozycja fragmentu* pozwala na efektywne zarządzanie fragmentowaniem pakietów.

Pole *czas życia* zawiera liczbę całkowitą decydującą o czasie życia pakietu w sieci. Dokumentacja protokołu IP zakłada, że pole powinno być zmniejszane o jeden co sekundę życia pakietu. W praktyce jest ono zmniejszane po każdym przeskoku pakietu przez router. Jeśli licznik osiągnie wartość równą zeru i pakiet nie dotrze do adresata, jest on niszczony, a jego nadawca jest powiadamiany specjalnym pakietem ICMP o niepowodzeniu transmisji. Zapobiega to krążeniu w nieskończoność w sieci niedostarczonych pakietów.

Pole *protokół* wskazuje proces transportowy, do którego w komputerze ma być dostarczony pakiet. Najczęściej stosowane procesy transportowe to TCP (ang. *Transport Control Protocol*) oraz UDP (ang. *User Datagram Protocol*).

Pole *suma kontrolna nagłówka* zawiera sumę kontrolną policzoną z nagłówka, co daje możliwość sprawdzenia spójności nagłówka po jego przesłaniu. Jest ona obliczana w czasie dokonywania każdego przeskoku pakietu, ponieważ zmianie ulega co najmniej pole *czas życia*.

Pola *adres źródłowy* i *adres docelowy* zawierają odpowiednio adres IPv4 nadawcy i odbiorcy pakietu. Adresy IPv4 mają 32 bity długości i w założeniu miały jednoznacznie określić unikalny adres komputera w skali całej sieci. Opcjonalne pola nagłówka nie będą omawiane.



Rys. 3.18. Zasada fragmentacji pakietów IPv4

Adresy IP są 32-bitowymi liczbami. Dla wygody przyjęto technikę ich zapisywania w tzw. notacji kropkowej. Adres jest wtedy podzielony na 4 pola, każde reprezentowane przez 8-bitową liczbę zapisaną w systemie dziesiętnym. Listing 3.1 pokazuje, jak należy rozumieć adres IP. Współczesna adresacja IP komputerów opiera się na schemacie *CIDR* (ang. *Classles InterDomain Routing*) – bezklasowym routingu międzydomenowym opisanym w dokumencie [36]. Oprócz samego adresu IP podaje się przypisaną mu tak zwaną maskę sieci. Jest to druga 32-bitowa liczba wskazująca, jaka część adresu jest adresem komputera, a jaka część – adresem sieci, w której komputer jest umieszczony. Część adresu, dla której w masce znajdują się wartości 1, jest adresem sieci. Bity adresu, dla których maska sieci

Listing 3.1

```
 1  Adres IP:                  218.64.116.75
 2  Maska sieci:               255.255.255.192
 3  Adres IP (binarnie):       11011010.10000001.11010000.01001011
 4  Maska (binarinie):         11111111.11111111.11111111.11000000
 5  Adres sieci (binarnie):    11011010.10000001.11010000.01000000
 6  Adres sieci:               218.64.116.64
 7  Adres rozgoszeniowy
 8  (binarnie):                11011010.10000001.11010000.01111111
 9  Adres rozgoszeniowy:       218.64.116.127
10  Adres pierwszego komputera
11  w sieci (binarnie):        11011010.10000001.11010000.01000001
12  Adres pierwszego komputera
13  w sieci:                   218.64.116.65
14  Adres ostatniego komputera
15  w sieci (binarnie):        11011010.10000001.11010000.01111110
16  Adres ostatniego komputera
17  w sieci:                   218.64.116.126
18  Dopuszczlna liczba
19  komputerow w podsieci:      61 (63-2)
```

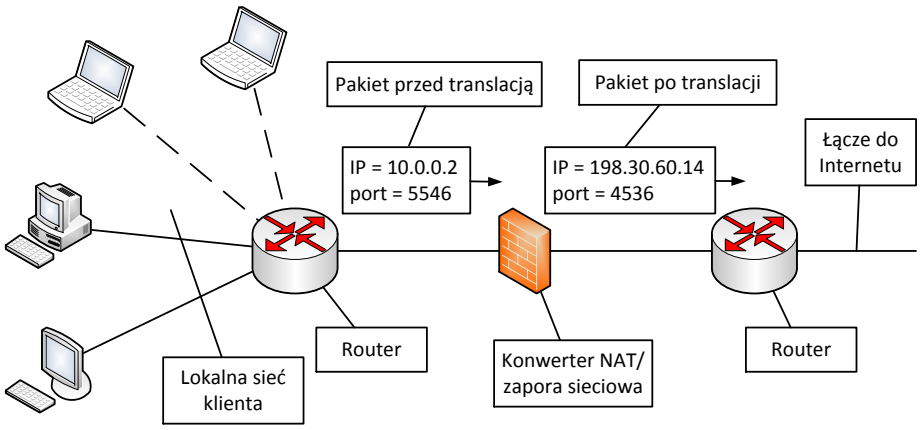
wynosi 0, są zarezerwowane na numery komputerów w danej sieci. Z puli adresów komputerów należących do danej sieci należy wyłączyć adres zawierający same wartości 0 lub 1 w polu przeznaczonym na adres komputera (por. listing 3.1). Pierwszy z nich jest rozumiany jako adres danej podsieci, drugi – jako adres rozgłoszeniowy, czyli adres do wszystkich komputerów w danej sieci. Adresy IP ustalone zgodnie z CIDR zapisuje się również w postaci *218.64.116.75/26*. Liczba 26 po ukośniku wskazuje długość maski sieci (liczba bitów w adresie przeznaczona na numer sieci).

Każdy komputer pracujący pod nadzorem protokołu IP powinien mieć swój niepowtarzalny adres w sieci. Wraz z rozwojem Internetu okazało się, że 32-bitowych adresów zaczęło brakować i część komputerów we współczesnej sieci globalnej ma takie same adresy. Powielanie adresów stało się możliwe dzięki wprowadze-

niu mechanizmu *NAT* (ang. *Network Address Translation*) – translacji adresów sieciowych opisanego w dokumencie [31]. Mechanizm translacji wykorzystuje informację, że trzy zakresy adresów IP zostały zdefiniowane jako tak zwane *adresy prywatne*. Na listingu 3.2 wymieniono wartości oraz zakresy tych adresów. Adresy prywatne nigdy nie zostaną przydzielone żadnemu komputerowi w publicznym Internecie. Pakiety o tych adresach mogą być wymieniane w obrębie sieci lokalnych (prywatnych) i nie są przepuszczane przez routery „na zewnątrz". Technika

Listing 3.2

```
1   10.  0.  0.  0   -    10.255.255.255/8  (16777216 adr. IP)
2  172. 16.  0.  0   -   172. 31.255.255/12 ( 1048576 adr. IP)
3  192.168.  0.  0   -   192.168.255.255/16 (   65536 adr. IP)
```



Rys. 3.19. Zasada działania konwertera NAT

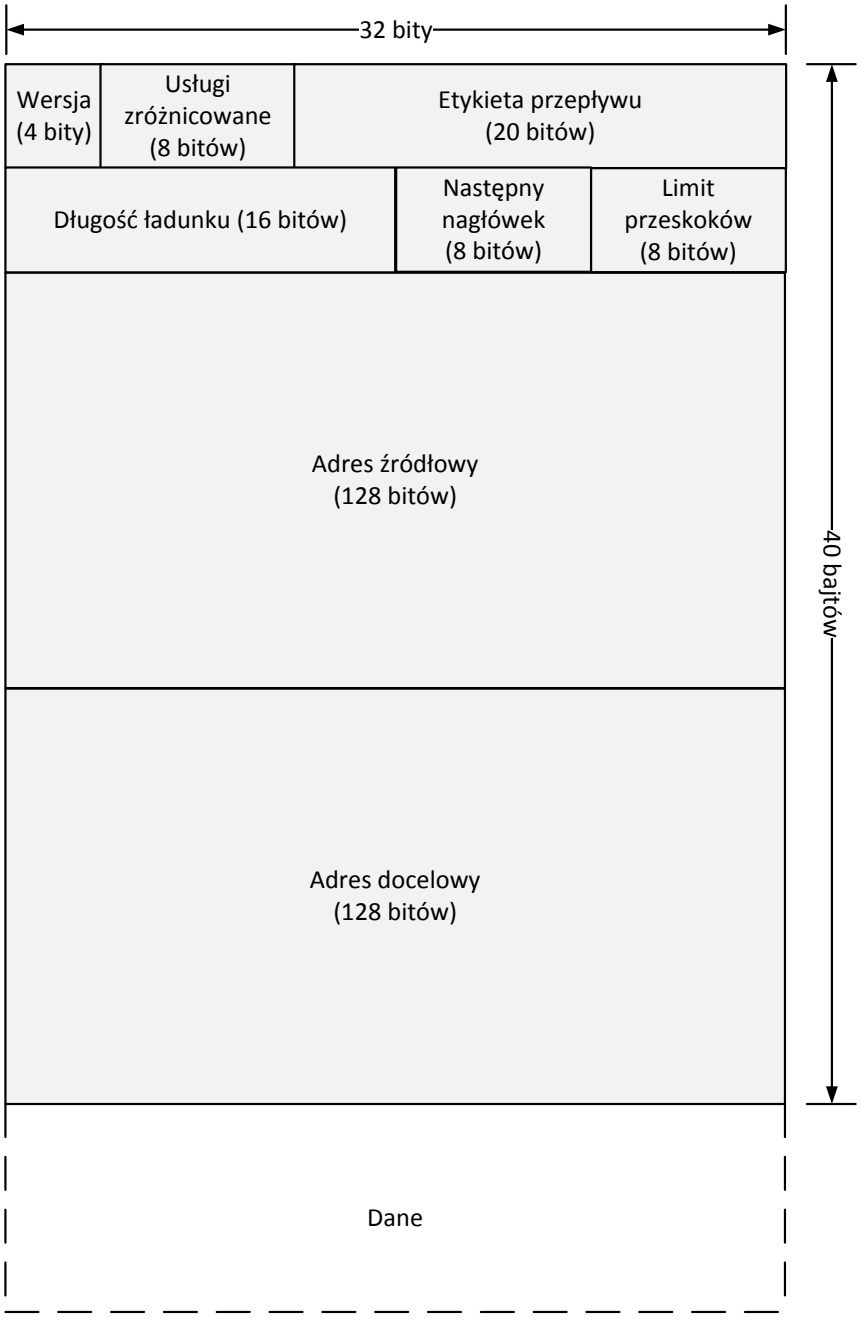
integracji sieci prywatnej z Internetem została przedstawiona na rys. 3.19. Pakiet z adresem klienta należącym do puli adresów prywatnych, który chce odwołać się do „zewnętrznego" adresu IP, jest w konwerterze NAT „przeadresowywany" (zmieniane jest pole adresu nadawcy). Po przejściu przez konwerter uzyskuje on inny adres pozwalający na swobodne przenoszenie pakietu w sieci nadrzędnej. Odpowiedź do klienta przychodząca z Internetu jest z kolei „przeadresowywana" przez konwerter z powrotem na jego adres w sieci lokalnej, co z kolei umożliwia efektywne odbieranie danych z Internetu. Zatem lokalna podsieć prywatna jest widoczna „na zewnątrz" konwertera NAT jako pojedynczy komputer z danym adresem IP. Technika NAT rozwiązała problem brakujących adresów IPv4. Instytucje, które chcą korzystać z zasobów Internetu, mogą „być widoczne" w postaci jednego komputera z pojedynczym adresem IP.

Oprócz adresów prywatnych wymienionych w listingu 3.2 w puli adresów Internetu istnieje jeszcze kilka zakresów adresów o specjalnym przeznaczeniu. Adres *0.0.0.0* jest rozumiany jako adres całego Internetu, a adres *255.255.255.255* jest teoretycznie adresem rozgłoszeniowym do wszystkich komputerów w Internecie (z oczywistych względów takie rozgłoszenie odbywa się realnie tylko w danej podsieci, oddzielonej od Internetu routerem). Specjalnym adresem jest również *127.0.0.1* nazywany adresem pętli. Jest to adres umożliwiający skonfigurowanie komunikacji poprzez interfejs sieciowy pomiędzy programami znajdującymi się na jednym komputerze. Jest to zawsze adres komputera, na którym właśnie wykonujemy pracę. Adresy od *127.0.0.2* do *127.0.0.255* także należą do wewnętrznych adresów komputera. Z kolei adresy posiadające pierwszy bajt z zakresu od 224 do 239 są zarezerwowane do prowadzenia tak zwanej komunikacji grupowej, adresy, których pierwszy bajt przekracza wartość 239, należą zaś do grupy adresów zarezerwowanych, których nie możemy w sieci Internet normalnie użytkować.

3.5.4. Protokół IPv6

Podstawowym problemem, z jakim się boryka warstwa sieciowa współczesnych sieci komputerowych, jest niedostatek adresów IPv4. W trakcie używania protokołu IPv4 okazało się również, że przesyłanie pakietów w strukturach współczesnych sieci komputerowych (znacznie mniej podatnych na awarie) można uprościć i przyspieszyć. Już w latach 90. XX wieku powstał nowy protokół komunikacji o nazwie IPv6 (ang. *Internet Protocol version 6*), opisany szczegółowo w dokumentach [29, 33]. Oferuje on przede wszystkim 128-bitowe pola na przechowywanie adresów komputerów, co oznacza, że na jednym metrze kwadratowym Ziemi może się znajdować $7 \cdot 10^{23}$ osobno zaadresowanych komputerów. Ciekawym zjawiskiem jest wciąż dość wolna migracja w sieci Internet z protokołu IPv4 do IPv6. Gdyby sięgnąć do specyfikacji współczesnych urządzeń zdolnych do komunikacji sieciowej, to większość z nich jest zgodna z nowym protokołem. Jednak obecnie w sieci istnieją tylko „wyspy" komputerów komunikujących się z zastosowaniem protokołu IPv6, które wymieniają się danymi poprzez „tunelowanie" pakietów IPv6 w pakietach IPv4.

Schemat pakietu IPv6 przedstawiono na rys. 3.20. W polu *wersja* zapisana jest wartość 6. Pole ma pozwalać routerom na rozpoznawanie pakietów IP w wersji 4 i 6, gdy będą one równocześnie obsługiwane. Pole *usługi zróżnicowane* jest interpretowane identycznie z polem o tej samej nazwie w pakiecie IPv4 (por. podrozdział 3.5.3). Pole *etykieta przepływu* umożliwia rozróżnienie strumienia pakietów biegnących od danego nadawcy do odbiorcy. Pakiety należące do jednego przepływu posiadają ten sam numer etykiety. Dzięki temu możliwe jest specyficzne traktowanie pakietów należących do jednego strumienia. Pole *długość ładunku* zawiera liczbę bajtów, które w danym pakiecie umieszczono po 40-bajtowym



Rys. 3.20. Pakiet IPv6

nagłówku. W odróżnieniu od protokołu IPv4 pole informuje tylko o długości danych. Pierwszy nagłówek protokołu IPv6 z założenia ma stałą wartość. Pole *następny nagłówek* wskazuje, który z dostępnych sześciu opcjonalnych nagłówków znajduje się po pierwszym nagłówku pakietu. Jeśli jest to ostatni nagłówek, to zawiera on kod protokołu transportowego, za pomocą którego są przenoszone dane w warstwie nadrzędnej. Pole *limit przeskoków* ma w praktyce takie samo znaczenie, jak pole *czas życia* w protokole IPv4. Po przejściu przez kolejny router wartość tego pola jest zmniejszana o jeden. W chwili osiągnięcia wartości 0 pakiet jest niszczony, a nadawca jest o tym powiadamiany przez wysłanie odpowiedniego zwrotnego pakietu serwisowego. Ostatnimi polami głównego nagłówka protokołu IPv6 są dwa 128-bitowe adresy, odpowiednio adres nadawcy i adres odbiorcy. Dla adresów IPv6 przyjęto inną niż kropkowa (por. podrozdział 3.5.3) metodę zapisu. Szesnastobajtowy adres dzieli się na osiem dwubajtowych słów oddzielonych dwukropkami:

8020:0000:0000:0000:0124:4556:8945:CDDA

Jeśli adres zawiera zera na początku opisu słów dwubajtowych, to można je pominąć, natomiast jedną lub kilka grup zerowych bitów można zastąpić parą dwukropków:

8020::124:4556:8945:CDDA

„Stare" 32-bitowe adresy zgodne z IPv4 można zapisać w postaci:

::192.31.17.22

Nagłówek protokołu IPv6 jest uproszczony w stosunku do jego odpowiednika w protokole IPv4. Zrezygnowano z pola *długości nagłówka*, ponieważ podstawowy nagłówek ma stałą długość. Rolę pola *protokół* przejęło pole *następny nagłówek*. Zrezygnowano z pól odpowiedzialnych za fragmentację. Rozmiar pakietu IPv6 ustala nadawca. Jeśli pakiet nie będzie mógł być przetransportowany w ustalonym rozmiarze, to router, który nie jest w stanie przenieść pakietu, odsyła specjalną informację nadawcy, aby rozmiar pakietu został zmniejszony. Zrezygnowano również z pola *suma kontrolna*, co zwiększa wydajność.

3.5.5. Protokoły sterujące warstwy sieciowej

W warstwie sieciowej oprócz protokołów IP istnieją tak zwane protokoły sterujące. Najważniejszymi z nich są *ICMP* (ang. *Internet Control Message Protocol*), *ARP* (ang. *Address Resolution Protocol*) oraz *DHCP* (ang. *Dynamic Host Configuration Protocol*).

Nieoczekiwane zjawiska w Internecie są raportowane nadawcy w postaci komunikatu ICMP. Komunikaty ICMP są przesyłane w ramkach IP. Najważniejszymi

błędami zgłaszanymi w komunikatach ICMP są niemożliwość dostarczenia pakietów, zniszczenie pakietu z powodu osiągnięcia przez pole *czas życia/limit przeskoków* wartości 0 czy błąd w polu nagłówka wskazujący na potencjalny błąd oprogramowania IP nadawcy lub jednego z routerów.

Protokół ARP służy do wiązania adresów IP z fizycznymi adresami (np. Ethernet) urządzeń sieciowych. Zanim nastąpi przesyłanie danych zgodnych z protokołem IP do sieci są wysyłane rozgłoszeniowe pakiety formatu ARP z zapytaniem, który z komputerów posiada dany adres IP. Po otrzymaniu odpowiedzi nadawca wie, do którego fizycznego urządzenia należy przesłać pakiet opakowany w ramkę. Komputer włączony w strukturę nowej sieci samodzielnie buduje mapę pomiędzy adresami IP i np. Ethernet.

Protokół DHCP „zwalnia" użytkownika z konieczności samodzielnego konfigurowania protokołu IP na swoim komputerze. Większość urządzeń podłączanych do sieci wymieniających pakiety zgodnych z protokołem IP jest tak skonfigurowana, że w chwili przyłączenia się do nowej sieci poszukuje w niej komputera z uruchomionym serwerem DHCP. Serwer, posługując się mechanizmami warstwy łącza danych, przekazuje nowemu klientowi adres IP z zarezerwowanej puli. Wdrożenie protokołu DHCP uprościło łączenie się z siecią zgodną z protokołem IP. Protokół pozwala również na racjonalne przydzielanie puli adresów, jeśli w podsieci znajduje się więcej komputerów niż dostępnych adresów.

3.6. Warstwa transportowa

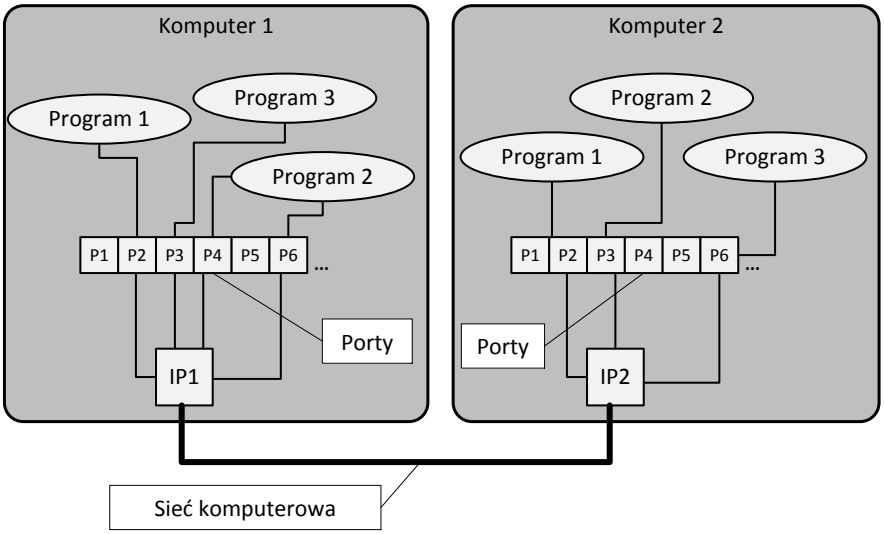
3.6.1. Zadania warstwy transportowej

Zadaniem warstwy transportowej jest przeniesienie danych pomiędzy procesami uruchomionymi na komunikujących się ze sobą komputerach. Dla komunikacji na tym poziomie należy dostarczyć możliwość niezawodnego dostarczenia danych. W rozwiązaniach dla Internetu w warstwie tej znajduje się również usługa zawodnego, ale szybkiego przesyłania pakietów. Warstwa transportowa Internetu jest także oferowana w postaci zestawu funkcji w językach programowania, dostarczając programistom abstrakcyjną perspektywę dostępu do sieci. Najważniejszymi protokołami warstwy są *UDP* (ang. *User Datagram Protocol*) oraz *TCP* (ang. *Transport Control Protocol*).

3.6.2. Porty

Każdy komputer w warstwie sieciowej posiada unikalny adres (IP). Zwykle jednak na danym komputerze koegzystuje wiele programów, które odwołują się do interfejsu sieciowego. Mechanizm portów pozwala na rozdzielenie strumieni

danych przychodzących i wychodzących przez interfejs sieciowy pomiędzy aplikacjami. Na rysunku 3.21 pokazano zasadę takiego „rozdziału ruchu". Aplikacje,



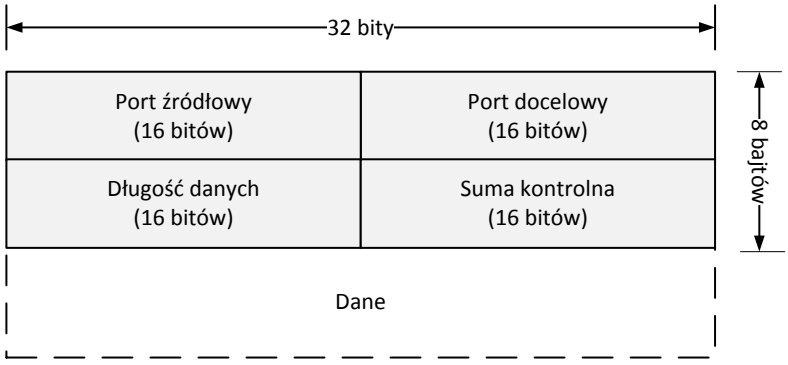
Rys. 3.21. Zasada funkcjonowania portów

które chcą wysłać jakieś dane do sieci, wysyłają swoje strumienie danych przez dodatkowe „bramki" nazywane portami. Interfejs sieciowy przekształca te strumienie w łańcuchy pakietów IP przesyłanych przez sieć komputerową. Aplikacje, które chcą odbierać dane, „nasłuchują" na umówionych portach. Jeśli znajdą się tam jakieś pakiety, to je odbierają, ponieważ uznają że są „zaadresowane" do nich. Porty są dwustronnym medium komunikacji i umożliwiają zarówno wysyłanie, jak i odbiór danych. Zestawienie port i adres IP nosi nazwę gniazda (ang. *socket*). Interfejs gniazd jest jednym z podstawowych zbiorów funkcji programistycznych do tworzenia aplikacji sieciowych.

W rodzinie protokołów TCP/IP przyjęto, że porty będą posiadały numery od 0 do 65535. Numery portów od 0 do 1024 są tak zwanymi zarezerwowanymi numerami portów. W tym zakresie portów są już wyznaczone numery, na których podejmują komunikację powszechnie znane usługi sieciowe, takie jak WWW, FTP, IMAP. Przykładowo, wszystkie zapytania, które wysyła się do serwera WWW, są z założenia przekazywane na port 80. Aktualną listę wszystkich zarezerwowanych portów można znaleźć na stronie WWW: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

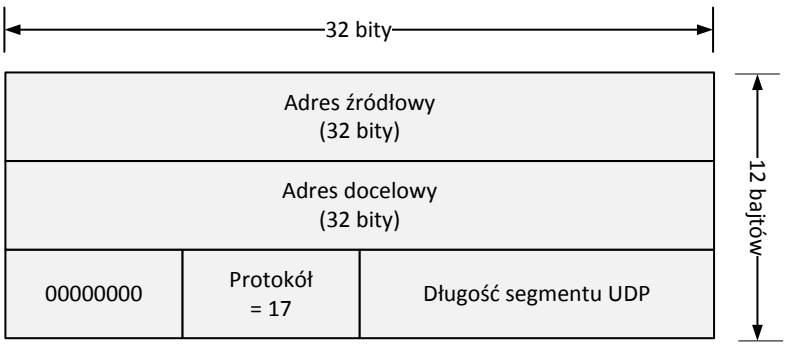
3.6.3. Protokół UDP

Protokół UDP (wyspecyfikowany w dokumencie [26]) można uznać za swego rodzaju prostą „nakładkę" na protokół IP, umożliwiającą zestawienie łącza pomiędzy komputerami w warstwie transportowej. Protokół jest bezpołączeniowy i zawodny, to znaczy nie sprawdza, czy w sieci jest odbiorca gotowy na przyjmowanie danych oraz nie posiada wbudowanych mechanizmów sprawdzania, czy dany pakiet został dostarczony. Jak się jednak okazuje, ma on duże zastosowanie w apli-



Rys. 3.22. Pakiet UDP

kacjach, gdzie nadrzędną potrzebą jest szybki przesył pakietów, z założeniem, że część z nich może nie dotrzeć. Takimi aplikacjami są: przesyłanie strumieni audio i wideo, komunikacja w grach sieciowych, dostarczanie informacji o nazwach domen (DNS - ang. *Domain Name System*), a nawet komunikacja w lokalnych sieciach komputerowych projektowanych na potrzeby przemysłu lotniczego (por. specyfikacja [3]). Na rysunku 3.22 przedstawiono pakiet zgodny z protokołem UDP. W polu danych protokołu IP znajduje się 8-bajtowy nagłówek, a następnie właściwe dane. Pierwsze dwie informacje zawarte w nagłówku to *numer portu źródłowego* i *numer portu docelowego*. Pakiet dostarczony do danego komputera jest kierowany do portu wskazywanego w numerze portu docelowego. Z tym portem powinna być skojarzona aplikacja chętna do przejęcia ładunku danych. Numer portu źródłowego może być zastosowany do skierowania odpowiedzi od adresata do nadawcy danych. Pole *długość danych* podaje długość pakietu UDP wraz z 8-bajtowym nagłówkiem. Dla aplikacji wymagających większej niezawodności w nagłówku UDP można zamieścić *sumę kontrolną*. Suma kontrolna jest obliczana z nagłówka, właściwych danych oraz *pseudonagłówka IP*. Jeśli pole danych ma nieparzystą długość bajtów, to jest ono dodatkowo wydłużane o osiem bitów o wartości równej zeru. Pseudonagłówek jest tworzony tymczasowo na podstawie danych z pakietu IP i zawiera dane przedstawione na rys. 3.23. Ostatecznie pseudonagłówek oraz ewentualne

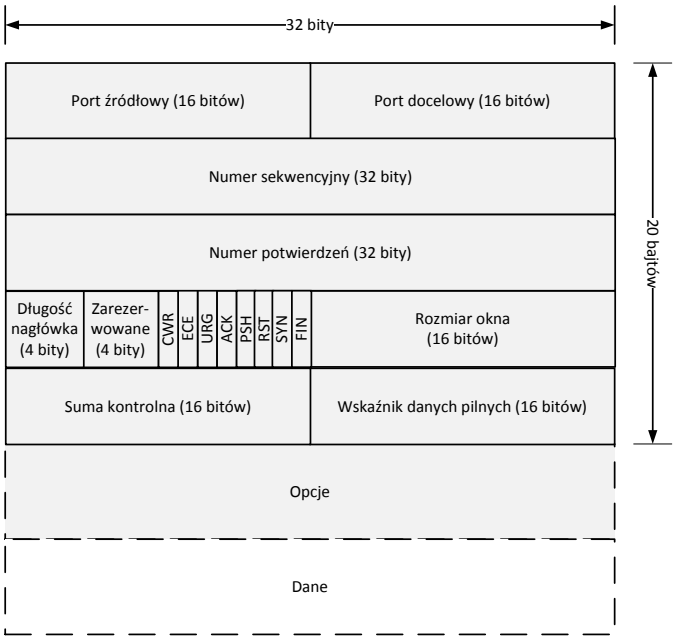


Rys. 3.23. Pseudonagłówek UDP

dopełnienie pola danych ośmioma zerami włączone do obliczania sumy kontrolnej nie są fizycznie przesyłane. Odtwarza się je u odbiorcy w celu zweryfikowania sumy kontrolnej.

3.6.4. Protokół TCP

Protokół TCP powstał w celu niezawodnego przesyłania danych na poziomie warstwy transportowej. Jest on połączeniowy, czyli przed rozpoczęciem transmisji następuje potwierdzenie, że w sieci istnieje komputer gotowy na wymianę danych. Głównym dokumentem opisującym protokół jest RFC 793 [28] z 1981 roku. Poprawki i uzupełnienia definicji protokołu zostały opisane w wielu innych dokumentach RFC. Spis tych dokumentów i zasadę posługiwania się nimi zebrano w RFC 4614 [35]. Głównym mechanizmem zapewniającym niezawodność w przesyłaniu danych zgodnie z protokołem TCP jest potwierdzanie odebranych porcji danych w postaci specjalnych pakietów ACK (ang. *ACKnowledgements*) odsyłanych do nadawcy. Jeśli komputer A wysyła dane do komputera B, to komputer B musi wysłać do komputera A potwierdzenie o ich otrzymaniu. Jeśli potwierdzenie nie nadchodzi w założonym przedziale czasowym, następuje retransmisja tej samej porcji danych. W ramach protokołu TCP automatycznie są obliczane: czas, jaki trzeba odczekać na odebranie potwierdzenia, limit liczby ponownych potwierdzeń, maksymalne opóźnienia po stronie odbiorcy na odesłanie potwierdzenia. Podczas przesyłania danych następuje również: rozpoznawanie potwierdzeń dla pakietów dochodzących w przypadkowej kolejności, porządkowanie danych przychodzących w przypadkowej kolejności, rozpoznawanie powielonych pakietów, regulacja szybkości przesyłanych pakietów. Na rysunku 3.24 przedstawiono pakiet zgodny z protokołem TCP. Dane przesyłane w pakiecie TCP są poprzedzane obowiązkowym 20-bajtowym nagłówkiem, który można rozszerzyć o następne 40



Rys. 3.24. Pakiet TCP

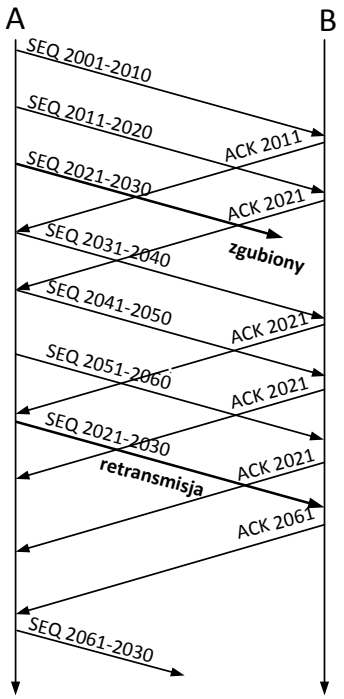
bajtów tak zwanych opcji. Dwa pierwsze pola nagłówka obowiązkowego zawierają odpowiednio *numer portu źródłowego* oraz *numer portu docelowego*. Pola te są interpretowane identycznie z protokołem UDP.

Podczas przesyłania danych zgodnie z protokołem TCP każdy przesłany bajt jest numerowany. Na początku transmisji jest generowana przypadkowa liczba 32-bitowa, która jest z kolei zwiększana o jeden (modulo 32) dla każdego przesłanego bajtu. Przykładowo, jeśli w danym pakiecie przesłano 10 bajtów danych, to wspomniana liczba zostanie zwiększona o 10 dla następnie nadawanej porcji danych. Aktualna wartość tej liczby jest wysyłana w pakiecie TCP w polu *numer sekwencyjny*.

Pole *numer potwierdzeń* jest wypełniane w pakietach potwierdzających (ACK) odebranie porcji danych. Podaje ono numer bajtu, na który teraz oczekuje odbiorca. Przykładowo, jeśli nadawca przesłał pakiet o numerze sekwencyjnym 2000 i przekazał w nim porcję danych o długości 10 bajtów, to pole *numer potwierdzeń* będzie miało wartość 2011, wskazując, że odbiorca odebrał wszystkie dane poprawnie i oczekuje teraz na bajt o numerze 2011.

Na rysunku 3.25 pokazano fragment sesji TCP, w której komputer *A* przesyła dane do komputera *B* zgodnie z protokołem TCP w porcjach po 10 bajtów. Segmenty zawierające dane bajty o numerach 2001-2010 oraz 2011-2020 zostały

przesłane poprawnie i ich otrzymanie przez komputer *B* zostało potwierdzone odpowiednimi pakietami ACK. Niestety pakiet zawierający bajty 2021-2030 nie został dostarczony komputerowi *B*. Kiedy komputer *B* otrzymuje kolejny pakiet, odsyła pakiet potwierdzenia z numerem 2021, informując komputer *A*, że wciąż oczekuje na pakiet zawierający bajty o numerze od 2021. Dzieje się tak za każdym razem, gdy komputer otrzymuje inny pakiet niż ten rozpoczynający się od bajtu 2021. Po retransmisji sekwencji bajtów 2021-2030 komputer *B* potwierdza odebranie bajtów 2021-2060 w jednym pakiecie potwierdzenia. Pomimo wykrycia utraty pakietu 2021-2030 pakiety nadsyłane z komputera *A* były zapamiętywane i zostały po odzyskaniu straconego pakietu odpowiednio „ułożone" w strumieniu odbieranych danych. O skutecznym odzyskaniu strumienia danych informuje komputer *B* pakietem potwierdzenia o numerze 2061.



Rys. 3.25. Potwierdzenia TCP

Pole *długość nagłówka* wskazuje liczbę 32-bitowych słów, z których składa się cały nagłówek. Jak już wspomniano, nagłówek TCP może zawierać pola opcjonalne, dlatego też taka informacja musi się w pakiecie znaleźć.

Po polu długości nagłówka znajdują się cztery bity zarezerwowane i dotąd niewykorzystane. Z kolei następne 6 bitów zawiera znaczniki (flagi) pozwalające na dodatkową identyfikację przesyłanego pakietu TCP. Bity *ECE* oraz *CWR* służą do sygnalizacji przeciążeń zgodnych z mechanizmem ECN (ang. *Explicit Congestion Notification*).

Ustawienie bitu *URG* powoduje, że w analizie danego pakietu brane jest pod uwagę pole *wskaźnik danych pilnych*. Pakiet z ustawionym bitem *URG* oznacza, że w strumieniu są przesyłane pilne dane, które należy osobno wyodrębnić. Pole *wskaźnik danych pilnych* informuje odbiorcę, o ile bajtów trzeba się przesunąć od bieżącego *numeru sekwencyjnego*, aby znaleźć początek pilnych danych.

Jeśli w nagłówku TCP jest ustawiony bit *ACK*, oznacza to, że dany pakiet zawiera potwierdzenie i należy w nim zinterpretować pole *numer potwierdzeń*. W pakietach oznaczonych na rys. 3.25 jako ACK wspomniany bit był ustawiony.

Jeśli w nagłówku jest ustawiony bit *PSH*, to oznacza, że dane zawarte w pakiecie muszą być natychmiast wysłane. Typowym „zachowaniem" protokołu TCP jest gromadzenie porcji danych i wysyłanie ich w „większych" pakietach, co obniża obciążenie sieci komputerowej. Niekiedy jednak oczekuje się natychmiastowego wysłania danych bez ich uprzedniego buforowania, wtedy jest ustawiany właśnie bit *PSH*.

Jeśli jakiś pakiet zawiera ustawiony znacznik *RST*, to oznacza, że jego nadawca żąda natychmiastowego zresetowania połączenia. Przyczyną wysłania takiego pakietu może być wykrycie różnych błędów w transmisji.

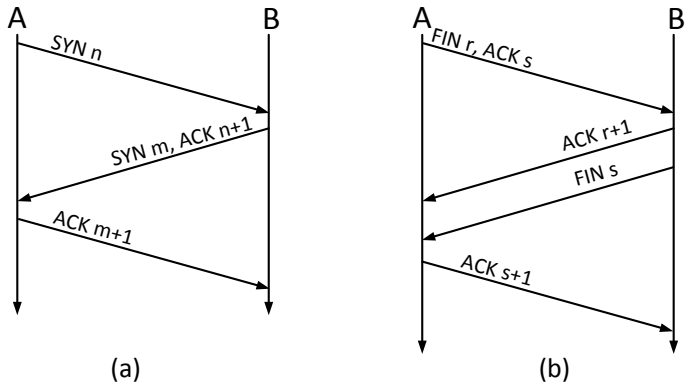
Flagi *SYN* oraz *FIN* są stosowane w procedurze nawiązywania połączenia pomiędzy komputerami, które chcą wymieniać dane zgodnie z protokołem TCP. Procedura ta zostanie przybliżona Czytelnikowi w dalszej części podrozdziału.

Pole *rozmiar okna* jest ustawiane przez odbiorcę i wskazuje, ile bajtów nadawca może jeszcze przesłać od bajtu wskazanego w polu *numer potwierdzenia*. Pole to jest mechanizmem kontroli przepływu na poziomie protokołu TCP. Odbiorca może przesłać nawet wartość 0 w tym polu, co jest interpretowane jako żądanie przerwania nadawania pakietów. Nadawanie pakietów rozpocznie się, gdy odbiorca prześle ponownie pakiet potwierdzenia z inną niż 0 wartością pole *rozmiar okna*.

Pole *suma kontrolna* zawiera sumę kontrolną obliczoną dla nagłówka uzupełnionego przez pseudonagłówek (identycznie jak w przypadku protokołu UDP, por. podrozdział 3.6.3). Jest to dodatkowy mechanizm zapewniający kontrolę poprawności transmisji.

Pole *opcje* może zawierać wiele dodatkowych informacji o danym pakiecie. Najczęściej definiowane pola opcji pozwalają na uzgodnienie pomiędzy nadawcą a odbiorcą maksymalnej wielkości pakietu lub na przenoszenie znacznika czasowego w celu precyzyjnego obliczania czasu przesyłu pakietu.

W najprostszym przypadku nawiązywanie połączenia pomiędzy dwoma komputerami w protokole TCP odbywa się następująco (por. rys. 3.26a): Komputer odbierający jest w stanie aktywnego oczekiwania. Inicjator połączenia ustala swój *numer sekwencyjny* i wysyła do komputera odbierającego pakiet zawierający ten numer wraz z ustawionym bitem *SYN*. Komputer odbierający odsyła wygenerowany przez siebie *numer sekwencyjny* wraz ze zwiększonym o jeden numerem sekwencyjnym inicjatora połączenia włączonym do pola *numer potwierdzeń*. Pakiet zawiera również ustawione bity *SYN* i *ACK*. Z kolei komputer inicjujący odsyła do odbierającego pakiet potwierdzający (z ustawionym bitem *ACK*), zawierający powiększoną o jeden wartość pola *numer sekwencyjny* komputera odbierającego w polu *numer potwierdzeń* oraz ze zwiększoną o jeden wartością własnego pola *numer sekwencyjny*.



Rys. 3.26. Nawiązanie (a) i rozłączenie (b) połączenia TCP

Scenariusz obustronnego zamknięcia połączenia TCP przedstawiono na rys. 3.26b. Komputer inicjalizujący zamknięcie wysyła do komputera odbierającego pakiet potwierdzenia z ustawionym bitem *FIN*. Komputer odbierający odsyła pakiet potwierdzenia, a następnie sam wysyła pakiet sygnalizujący zakończenie połączenia (z ustawionym bitem *FIN*). Całkowite zamknięcie połączenia odbywa się po otrzymaniu przez komputer odbierający pakietu potwierdzenia od komputera inicjalizującego zamknięcie.

3.7. Warstwa sesji, prezentacji i aplikacji

3.7.1. Zadania warstwy sesji

Warstwa sesji jest odpowiedzialna za nawiązanie i utrzymanie połączenia pomiędzy programami komunikującym się z zastosowaniem łącza sieciowego. Przy-

kładowo, kiedy przesył strumienia danych po łączu sieciowym zostaje z jakichś przyczyn przerwany, to warstwa może „samodzielnie" podjąć próbę ponownego zestawienia połączenia i kontynuacji komunikacji. W innym zastosowaniu warstwa transportowa „utrzymuje" jedno połączenie dla podtrzymania komunikacji pomiędzy klientem a serwerem. Klient i serwer prowadzą wtedy dialog bez konieczności nawiązywania i rozwiązywania sesji po przesłaniu pojedynczego komunikatu czy pliku. W praktycznych rozwiązaniach (por. rodzinę protokołów TCP/IP) warstwa sesji jest realizowana poprzez uruchomienie odpowiedniego podprogramu zapewniającego ciągłość połączenia.

3.7.2. Zadania warstwy prezentacji

Warstwa prezentacji w modelu OSI jest odpowiedzialna za konwersję danych. Z punktu widzenia logicznego te same dane przechowywane na różnych komputerach pracujących pod kontrolą różnych systemów operacyjnych, przetwarzane przez procesory o różnych architekturach mogą być reprezentowane w różny sposób. Jednym z przykładów niezgodności mogą być reprezentacje znaków. Wciąż działają komputery, które kodują litery za pomocą różnych kombinacji bitów. Standard kodowania znaków ASCII często stosowany w wielu komputerach nie jest jedyny. Co więcej, jest on stopniowo wypierany przez nowy standard kodowania znaków UCS (ang. *Universal Character Set*) umożliwiający jednolite zakodowanie znaków wszystkich języków na Ziemi. Podobnie rzecz ma się z np. zasadami kodowania liczb zmiennopozycyjnych. Procesory o różnych architekturach preferują różne metody kodowania liczb, które bardziej odpowiadają mikroprogramom wbudowanym w strukturę jednostek obliczeniowych. W środowisku informatycznym przyjęte są również dwa sposoby reprezentacji kolejności bajtów w długich słowach. Część komputerów (np. Sparc) zapamiętuje długie słowa bajtowe w konwencji *big endian*. Wtedy najbardziej znaczący bajt jest pamiętany pod najniższym adresem pamięci, a mniej znaczące pod kolejnymi. Z kolei inne komputery (np. Intel) stosują format zapisu *little endian*, gdzie najmniej znaczący bajt słowa jest zapamiętywany pod najniższym adresem, a bajty bardziej znaczące – pod kolejnymi adresami rosnącymi.

Warstwa prezentacji dostarcza mechanizmów pozwalających na prawidłową interpretację przesyłanych przez sieć strumieni danych. Problem niezgodności formatów danych rozwiązuje się często poprzez wcześniejsze uzgodnienie formatu, jaki powinny mieć dane przesyłane przez sieć. Na poziomie warstwy prezentacji następuje wtedy przekształcenie „lokalnych" reprezentacji liczb w tak zwany sieciowy format reprezentacji liczb. Po odebraniu danych zachodzi proces odwrotny, w którym dane z formatu sieciowego są przekształcane do formatu akceptowanego przez dany komputer. Warto zauważyć, że w modelu odniesienia TCP/IP nie ma

wyróżnionej warstwy prezentacji i twórcy oprogramowania sieciowego muszą odwoływać się do funkcji bezpośrednio dokonujących konwersji.

3.7.3. Zadania warstwy aplikacji

Warstwa aplikacji to zbiór protokołów i programów, które są w stanie stosować łącza sieciowe jako medium komunikacji. Podstawowe programy wymieniające dane z zastosowaniem łączy sieciowych można skonstruować za pomocą interfejsu gniazd. Pozwalają one na wysyłanie niepotwierdzanych datagramów lub potwierdzanych strumieni danych z zastosowaniem odpowiednio protokołów UDP i TCP. Kolejnym, naturalnym krokiem rozwoju aplikacji sieciowych było opracowanie protokołów przesyłania informacji dla podstawowych usług sieciowych: poczty elektronicznej, stron WWW, transferu plików czy transferu strumieni danych.

3.7.4. System nazw domen - DNS

Jedną z podstawowych usług warstwy aplikacji jest DNS (ang. *Domain Name System*). Serwery DNS dostarczają informacji o adresie IP posiadanym przez komputer identyfikowany za pomocą łańcucha tekstowego (np. www.rsa.com). Na początku działania Internetu uzgodniono konwencję tworzenia nazw komputerów. Przyjęto podział nazw na tak zwane domeny, które z kolei dzielą się na poddomeny itd. Internet został podzielony na 250 domen najwyższego poziomu. Domeny najwyższego poziomu podzielone są na dwie kategorie: rodzajowe (np. *gov, com*) i narodowe (np. *pl, uk*). W tablicy 3.2 pokazano kilka wybranych domen

Tablica 3.2. Wybrane domeny rodzajowe najwyższego poziomu

Domena	**Planowane przeznaczenie**	**Data utworzenia**
com	zastosowanie komercyjne	1985
edu	instytucje edukacyjne	1985
gov	instytucje rządowe	1985
int	organizacje międzynarodowe	1988
org	organizacje niekomercyjne	1985
aero	transport lotniczy	2001
mobi	urządzenia mobilne	2005

rodzajowych. Uzyskanie domeny drugiego lub trzeciego poziomu (np. *firma.com*, *prz.edu.pl*) wymaga zarejestrowania i wniesienia niewielkiej (ok. 100 zł) opłaty rocznej. Problemem staje się uzyskanie nazwy domeny już zarejestrowanej przez inną osobę czy instytucję.

Typowa konfiguracja komputera podłączonego do Internetu oprócz adresu IP, maski, adresu IP najbliższego routera (bramy) zawiera również wskazanie serwera DNS, który będzie dla danego komputera tłumaczył nazwy typu *firma.com.pl* na adres IP. Serwery DNS tworzą pewnego rodzaju rozproszoną bazę danych. Jeśli jeden z nich nie jest w stanie „przetłumaczyć" nazwy domenowej na adres IP, to sięga on po zasoby kolejnego serwera itd.

3.7.5. Wybrane protokoły i usługi warstwy aplikacji

Nowe aplikacje stosujące komunikację sieciową powstają codziennie, dlatego też próba ich usystematyzowania jest dosyć trudna. Wszystkie opierają się w istocie na możliwości przesyłania strumieni danych lub datagramów z zastosowaniem protokołów TCP/UDP/IP. Obecnie najczęściej używanymi usługami sieciowymi są w chwili obecnej WWW i poczta elektroniczna.

Do wymiany między komputerami informacji dotyczących zawartości stron *WWW* służy protokół HTTP (ang. *HyperText Transfer Protocol*). Omówienie tego protokołu wychodzi poza ramy niniejszego opracowania. W uproszczeniu klient chcący uzyskać dostęp do ustalonej strony internetowej wysyła na port 80 serwera WWW prośbę o przesłanie mu pliku o odpowiedniej nazwie sformatowanego w odpowiedni sposób. Plik jest zapisany w formacie HTML (ang. *HyperText Markup Language*). Jeśli klient dysponuje odpowiednim oprogramowaniem (przeglądarką internetową), to nadesłany plik może wyświetlić na ekranie komputera w formie graficznej. Pierwotną właściwością plików HTML była możliwość umieszczenia w nich odnośników do innych plików, a wybranie takiego odnośnika było równoważne z wysłaniem do odpowiedniego serwera WWW żądania przesłania kolejnego dokumentu HTML. Język HTML umożliwia również osadzanie w dokumentach obrazów, filmów, dźwięków czy grafiki komputerowej. Odpowiednie oprogramowanie dołączane do przeglądarek, tak zwane wtyczki, pozwalają na wyświetlanie danych zawartych w dokumentach internetowych wykraczających poza standard HTML. Kolejną funkcją stron jest możliwość bardziej zaawansowanej interakcji – przesyłanie tekstów, fotografii czy filmów od strony użytkownika. Zjawiskiem powszechnym stało się wypełnianie formularzy zamieszczonych na stronach WWW w celu przekazania lub uzyskania jakichś dóbr czy danych. Współczesne pliki stron WWW są generowane automatycznie na życzenie klienta, co umożliwia znaczącą personalizację wyglądu i funkcjonalności. Serwery WWW w tym celu współpracują ze sprzęgniętymi z nimi serwerami baz danych – baza danych dostarcza informacji, a serwer WWW konstruuje na ich podstawie stronę dla klienta. Jednym z istotnych współczesnych trendów tworzenia oprogramowania jest próba dostarczania wszelkich usług przez interfejs przeglądarki. Ostatecznym celem takiego rozwoju systemów informatycznych jest przeniesienie wszystkich danych do sieci komputerowej i umożliwienie ich przetwarzania za pomocą pro-

gramów, które również w sieci egzystują. Personalne urządzenie komputerowe mogłoby wtedy służyć tylko jako swego rodzaju wtyczka do sieci.

Poczta elektroniczna była pierwszą poważną usługą, która skłoniła użytkowników do korzystania z Internetu. Podobnie jak w przypadku protokołu pobierania stron WWW, protokół SMTP (ang. *Simple Mail Transfer Protocol*) za pomocą odpowiednich komend umożliwiał połączenie się z serwerem przechowującym wiadomości, ich tworzenie, odczyt, usuwanie czy przesłanie do innego adresata. Współczesne programy komunikujące się z serwerami poczty wciąż używają tego protokołu, tylko dla użytkowników jest on ukryty, a rezultaty jego „pracy" są widoczne w postaci graficznej na ekranie komputera. Pierwotnie wiadomości pocztowe zawierały tylko tekst. Po wprowadzeniu zasad kodowania zgodnych z MIME (ang. *Multipurpose Internet Mail Extension*) wiadomości pocztowe mogły przenosić dowolne rodzaje danych. Kodowanie MIME przekształca dowolne ciągi bajtów w taki sposób, że mogą być rozumiane jako znaki w kodowaniu ASCII. Dzięki temu oprogramowanie serwerów pocztowych było w stanie przenosić wiadomości z załączonymi zdjęciami, programami czy filmami, traktując je jako długie wiadomości tekstowe. Z kolei oprogramowanie klientów poczty było w stanie wyodrębnić z takich pseudoteksowych wiadomości treść samej wiadomości oraz załączniki.

Kolejnym powszechnie używanym protokołem sieciowym jest *FTP* (ang. *File Transfer Protocol*). Jest to protokół umożliwiający przesyłanie dowolnych plików pomiędzy komputerami. Ponownie odpowiednie komendy tekstowe umożliwiają przeglądanie katalogów plików na zdalnym komputerze oraz wysyłanie lub odbieranie plików do określonego katalogu. Użytkownicy współczesnych komputerów stosują ten protokół, kiedy kopiują pliki z Internetu na lokalny komputer lub kiedy wysyłają jakieś pliki poprzez odpowiednie formularze na stronach WWW.

Sieciową usługą zbliżoną funkcjonalnie do FTP jest *NFS* (ang. *Network File System*) – sieciowy system plików. Taka usługa sieciowa pozwala na „podłączenie" się do drzewa katalogów znajdującego się na innym komputerze i korzystania ze znajdujących się tam plików identycznie jak na lokalnym komputerze.

Inną współcześnie stosowaną usługą sieciową jest *RPC* (ang. *Remote Procedure Call*). Istnieje możliwość takiego skonstruowania oprogramowania komputerów, zgodnie z którym do wykonywania części obliczeń programu może zostać oddelegowany inny komputer. Łącze sieciowe służy wtedy za kanał przesyłający podprogram na inny komputer oraz za medium pozwalające na odesłanie otrzymanych wyników.

Zwiększenie pasma Internetu umożliwiło wprowadzenie do sieci nowych usług – *radia i telewizji internetowej*. Obie te usługi opierają się na przesyłaniu, dekodowaniu i odbieraniu w czasie rzeczywistym strumienia danych cyfrowych. Wysyłanie strumieni wideo stało się możliwe po opracowaniu standardów MPEG2

i MPEG4, pozwalających na kompresowanie danych wideo i ich odtwarzanie po stronie klienta, co z kolei umożliwia osiągnięcie kompromisu pomiędzy zajętością sieci a jakością obrazu.

Protokołem rzadziej używanym jest *telnet*. Umożliwia on połączenie się z konsolą zdalnego komputera i uruchamianie na nim programów czy administrowanie zasobami. Protokół ten nie oferuje żadnych zabezpieczeń. W podrozdziale 3.8.5 opisano protokół SSH (ang. *Secure Shell*) również oferujący funkcjonalność zdalnego terminala, ale przy użyciu bezpiecznego połączenia. Inne rozwiązania w tym obszarze usług oferują użytkownikom możliwość zarządzania komputerem znajdującym się w innym miejscu (kraju, państwie) z zastosowaniem mechanizmu *zdalnego pulpitu*. Pozwala to na wyświetlenie na swoim lokalnym komputerze pulpitu innego komputera i zarządzanie nim z zastosowaniem typowego interfejsu graficznego.

3.8. Wybrane zagadnienia bezpieczeństwa w sieciach komputerowych

3.8.1. Aspekty bezpieczeństwa

Problem bezpieczeństwa w sieciach komputerowych jest bardzo złożony. Z jednej strony zagadnienie to dotyczy bezpieczeństwa przesyłanych przez sieć danych. Protokoły sieciowe niewykorzystujące kryptografii są zawsze podatne na ataki, które sprawiają, że transmisja danych nie spełnia żadnego z aspektów bezpieczeństwa omówionych w podrozdziale 2.2. Z drugiej strony istotnym problemem jest zapewnienie bezpieczeństwa samym sieciom komputerowym podłączonym do Internetu oraz podłączonym do sieci systemom informatycznym. To zadanie dodatkowo utrudnia fakt, że obecne systemy informatyczne są systemami rozproszonymi i jako takie wymagają komunikacji sieciowej.

Ze względu na zakres zagadnienia rozwiązania zapewniające bezpieczeństwo w sieciach komputerowych są bardzo różnorodne. Bezpieczną transmisję danych (wraz z uwierzytelnianiem obu stron) można zrealizować za pomocą protokołów wykorzystujących kryptografię. Takie rozwiązania oferują między innymi protokoły TLS/SSL (ang. *Transport Layer Security/Secure Socket Layer*) oraz IPsec (ang. *Internet Protocol Security, IP Security*) opisane w dalszej części tego rozdziału. Mimo że są one często wykorzystywane, istnieje jednak wiele innych rozwiązań, które mogą być bardziej adekwatne do specyficznych zastosowań. Bezpieczeństwo sieci komputerowych oraz w pewnym stopniu podłączonych do nich systemów informatycznych można zwiększyć dzięki właściwej konfiguracji firewalla, co także opisano. Poprawna realizacja bezpiecznego rozproszonego systemu informatycznego jest znacznie bardziej złożonym zadaniem i wykracza poza ramy niniejszej

pracy. Bardzo często łączy ona zarówno opisane rozwiązania zabezpieczające transmitowane dane, jak i właściwą konfigurację sieci komputerowych.

3.8.2. Firewall i filtrowanie ruchu sieciowego

Działanie zapory sieciowej

Filtrowanie ruchu sieciowego jest jednym z istotnych mechanizmów służących do zabezpieczania sieci komputerowych, a w niektórych przypadkach także systemów operacyjnych. Oprogramowanie sieciowe oczekujące na nadejście połączeń TCP czy pakietów UDP często ma bardzo ograniczone możliwości filtrowania nadchodzących z sieci danych. Nawet jeśli konfiguracja takiego oprogramowania pozwala na odpowiednie jego zabezpieczenie, to na ogół jest ona wykonywana przez inne osoby niż administratorzy odpowiedzialni za bezpieczeństwo sieci. Dodatkowo, sieci wewnętrzne (domowe czy należące do instytucji) podłączone do Internetu powinny oferować swoim użytkownikom wygodę i większy zakres bezpieczeństwa niż sieć publiczna, muszą więc być od tej sieci oddzielone. Wymaga to wprowadzenia ograniczeń i ścisłego określenia, jakie pakiety z sieci publicznej mogą zostać przesłane do sieci wewnętrznej. Czasami filtruje się także ruch wychodzący z sieci wewnętrznej do sieci publicznej.

Do filtrowania ruchu sieciowego służy oprogramowanie nazywane *firewallem* (a czasami *zaporą sieciową*). Jest ono albo częścią systemu operacyjnego, albo ściśle współpracuje z systemem operacyjnym, aby zrealizować swoje zadania. Firewall najczęściej jest konfigurowany na routerze łączącym sieć wewnętrzną z Internetem bądź też różne sieci wewnętrzne pomiędzy sobą. W ostatnim czasie często wykorzystuje się go także na komputerach osobistych w celu dodatkowego ich zabezpieczenia.

Firewall pozwala na bardzo precyzyjną konfigurację filtrów decydujących o tym, jakie pakiety mogą być przekazywane pomiędzy sieciami czy też przyjmowane przez system operacyjny, na którym pracuje. Często jest możliwa także modyfikacja przesyłanych pakietów, np. w celu realizacji NAT (zob. podrozdział 3.5.3). Dzięki temu administrator sieci czy też administrator systemu operacyjnego może zdecydować, jakie pakiety z innej sieci w ogóle trafią do systemów w sieci przez niego zarządzanej czy też jakie zostaną wysłane na zewnątrz. Typową funkcją firewalla jest możliwość gromadzenia (logowania) informacji o przetwarzanych przez niego pakietach sieciowych, co jest bardzo pomocne w analizie udanych i nieudanych ataków sieciowych.

Najprostsze filtry, jakie można zrealizować na firewallu, pozwalają konstruować warunki na podstawie zawartości przechwyconego pakietu sieciowego. Warunki te mogą dotyczyć protokołów różnych warstw: sieciowej – IPv4/IPv6, transportowej – TCP/UDP, ale także warstw wyższych czy niższych. Możliwe jest filtrowanie

na podstawie adresów źródłowych, docelowych (warstwy sieciowej lub fizycznej) czy też numerów portów (TCP oraz UDP), ustawionych bądź nieustawionych flag protokołu TCP (SYN, ACK ...). Dzięki takim filtrom administrator może zadecydować, do jakich usług w zabezpieczanej przez niego sieci będzie w ogóle dostęp z Internetu oraz z jakimi usługami w Internecie będą się mogli połączyć użytkownicy z jego sieci. Ten rodzaj filtrowania nazywa się *bezstanowym* (ang. *stateless*) – każdy otrzymany pakiet jest analizowany oddzielnie bez powiązania z jego poprzednikami, a więc firewall nie zapamiętuje w żaden sposób np. aktualnego stanu połączeń TCP czy komunikacji UDP.

Bardziej zaawansowane możliwości oferuje tzw. filtrowanie *stanowe* (ang. *stateful*). Filtry stanowe zapamiętują stan komunikacji sieciowej, w której pośredniczą. Dzięki temu możliwe jest np. utworzenie filtra odrzucającego segmenty TCP nienależące do żadnego z aktualnie nawiązanych połączeń (prawdopodobnie błędne albo będące częścią potencjalnego ataku sieciowego). Realizacja NAT także wymaga firewalla stanowego – system musi przechowywać informację o mapowaniu adresów i portów dla aktualnie nawiązanych połączeń TCP czy komunikacji UDP. Firewall stanowy oferuje znacznie większe możliwości filtrowania i jest szczególnie użyteczny w przypadku istotnego zagrożenia zaawansowanymi atakami sieciowymi. Jest on jednak bardziej skomplikowany i wymaga większych zasobów – konieczne jest zapamiętywanie stanu oraz szybkie odnajdywanie zapamiętanej informacji.

Zastosowanie firewalla

W sieciach, dla których łącze internetowe służy jedynie ich użytkownikom do korzystania ze światowej sieci, konfiguracja jest stosunkowo prosta. Pojedynczy router jest podłączony jednym interfejsem (często nazywanym WAN) do Internetu i na tym interfejsie jest skonfigurowany jeden rutowalny adres IPv4 przydzielony przez dostawcę łącza. Do drugiego (wewnętrznego) interfejsu routera jest podłączany przełącznik (w najprostszych rozwiązaniach wbudowany w urządzenie routera) służący do podłączenia komputerów sieci lokalnej. Ten wewnętrzny interfejs ma przydzielony adres IP z puli prywatnej. Na routerze jest skonfigurowany NAT, dzięki czemu wszystkie komputery z sieci lokalnej są widziane w Internecie tak, jakby były pojedynczym komputerem korzystającym z pojedynczego adresu rutowalnego. Firewall routera na ogół blokuje ruch z Internetu do sieci lokalnej, chyba że należy on do komunikacji zainicjowanej przez jeden z lokalnych komputerów.

Nawet proste routery przeznaczone do użytku domowego umożliwiają udostępnienie portów routera i przeforwardowanie ich na porty wybranego komputera z sieci lokalnej. W domowych zastosowaniach jest to często wykorzystywane w grach komputerowych. Wiele routerów umożliwia też konfigurację powodującą

przekazywanie całej przychodzącej do nich komunikacji na jeden z komputerów lokalnych. W domowych zastosowaniach takie rozwiązanie może być dopuszczone.

Jeśli poza dostępem do Internetu łącze służy także do połączenia z Internetem serwerów, konfiguracja powinna być nieco bardziej złożona. Najprostsze rozwiązanie opisane w poprzednim akapicie ma istotną wadę: włamanie z zewnątrz na komputer w sieci wewnętrznej, na którym udostępniono jeden lub więcej portów, umożliwia intruzowi dostęp do całej sieci lokalnej, tak jakby był jej uprawnionym użytkownikiem. Aby uniknąć takiej sytuacji, należy zastosować rozwiązanie nazywane *strefą DMZ* (ang. *Demilitarized Zone* – strefa zdemilitaryzowana). Router łączący sieć lokalną z Internetem powinien mieć jeszcze jeden (trzeci) interfejs służący do podłączenia dodatkowej podsieci nazywanej właśnie strefą DMZ. Konfiguracja routera powinna pozwalać na połączenia z Internetu oraz z sieci lokalnej z serwerami w strefie DMZ w takim stopniu, jak to jest potrzebne (np. z usługami udostępnianymi przez te serwery). Połączenia z Internetu oraz ze strefy DMZ z komputerami w sieci lokalnej powinny być zablokowane. Dzięki takiemu rozwiązaniu jest możliwe zarówno odpowiednie zabezpieczenie lokalnych serwerów (znajdujących się za firewallem w strefie DMZ), jak i sieci lokalnej. Równocześnie w razie włamania na serwer w strefie DMZ sieć lokalna jest bezpieczna.

Opisane scenariusze to jedynie najprostsze, klasyczne rozwiązania wykorzystujące firewall do zapewnienia bezpieczeństwa sieci. Możliwości konfiguracyjne zapór sieciowych oraz różnorodne wymagania specyficznych sieci sprawiają, że istnieje bardzo wiele możliwych rozwiązań.

3.8.3. SSL i TLS

Bezpieczna transmisja danych w sieci Internet w większości jest oparta na protokołach SSL (ang. *Secure Sockets Layer*) i wywodzącym się z niego TLS (ang. *Transport Layer Security*) [38]. Protokoły te działają w warstwie prezentacji modelu OSI (pomiędzy warstwą aplikacji a transportową w modelu TCP/IP). Pozwalają one na zapewnienie *poufności*, *integralności* przesyłanych danych i *uwierzytelniania* stron (wyjaśnienie tych pojęć znajduje się w rozdziale 2.). Uwierzytelnienie serwera jest obligatoryjne na podstawie certyfikatu, natomiast klienta – opcjonalne. Komunikacja między klientem a serwerem jest szyfrowana algorytmem symetrycznym. W zależności od wersji protokołu możliwe jest użycie różnych algorytmów zarówno blokowych, jak i strumieniowych, m.in. AES, DES, IDEA, RC4. Komunikacja taka jest oczywiście w pełni dwukierunkowa, ciekawostką jest użycie różnych symetrycznych kluczy szyfrujących dla komunikacji prowadzonej w poszczególne strony. Symetryczne klucze szyfrujące są uzgadniane na etapie nawiązywania połączenia za pomocą kryptografii asymetrycznej. W tym przypadku również jest możliwe użycie różnych algorytmów, m.in. RSA czy algorytm Diffiego-Hellmana. Integralność przesyłanych danych jest zapewniana za

pomocą kodu HMAC (zob. podrozdział 2.3.4) mogącego wykorzystywać funkcje skrótu, takie jak MD5, SHA1 lub SHA256/384.

Jak widać, protokoły te są dość rozbudowane. W kolejnych wersjach są usuwane rozwiązania kryptograficzne uznawane za przestarzałe i niezapewniające wystarczającego poziomu bezpieczeństwa, a w zamian są dodawane nowe. Faktycznie wykorzystany podczas konkretnego połączenia szyfr jest uzgadniany przy jego nawiązywaniu, zależnie od wersji i konfiguracji serwera i klienta[2]. W istocie protokoły SSL/TLS składają się z czterech protokołów składowych: RLP (ang. *Record Layer Protocol*), HP (ang. *Handshake Protocol*), CCSP (ang. *Change Cipher Specification Protocol*) i AP (ang. *Alert Protocol*). RLP jest głównym protokołem pobierającym dane z warstwy aplikacji zabezpieczającym je i przekazującym do kolejnej warstwy. HP jest wykorzystywany do uzgodnienia wspólnego zestawu protokołów zabezpieczających (szyfr, funkcja skrótu itd.), algorytmu kompresji i kluczy do przygotowanego zestawu. CCSP kopiuje przygotowany zestaw do aktualnych parametrów roboczych. AP pozwala na sygnalizację problemów komunikacyjnych.

Protokoły SSL/TLS mogą służyć do zabezpieczania transmisji prowadzonej dowolnym protokołem aplikacyjnym, ale najczęściej są wykorzystywane do zabezpieczania protokołu HTTP, tworząc bezpieczny HTTPS (ang. *Hypertext Transfer Protocol Secure*). Jest on powszechnie wykorzystywany podczas połączenia z serwisami bankowymi, płatności w sklepach internetowych itp. Adres w pasku przeglądarki zaczyna się od prefiksu `https://` (a nie `http://`) i jest poprzedzony ikoną kłódki symbolizującej bezpieczne połączenie. Z reguły kliknięcie tej ikony pozwala na wyświetlenie szczegółowych informacji o połączeniu, takich jak tożsamość witryny czy użyte algorytmy kryptograficzne.

3.8.4. IPsec

IPsec [34] jest grupą protokołów realizujących zabezpieczenie transmisji w warstwie internetowej, tj. na poziomie datagramów IP. Zaletą działania w niskiej warstwie jest transparentność dla aplikacji, a więc brak konieczności wspierania takiego zabezpieczenia lub specjalnej konfiguracji. Głównym zastosowaniem IPsec jest tworzenie bezpiecznych *wirtualnych sieci prywatnych* (VPN – ang. *Virtual Private Network*), tj. logicznych struktur, które w sposób przezroczysty dla użytkownika łączą fizyczne fragmenty sieci prywatnych za pomocą sieci publicznej. Rozwiązanie takie pozwala np. pracownikom na zdalną pracę przez Internet na komputerze domowym, tak jakby był podłączony bezpośrednio do wewnętrznej sieci korporacyjnej.

[2] Powinien to być najmocniejszy zestaw mechanizmów kryptograficznych wspierany przez obie strony, niestety w praktyce możliwe są ataki *man-in-the-middle* typu *downgrade dance* wymuszające słabsze zabezpieczenia.

Możliwe są dwa tryby pracy IPsec – transportowy i tunelowy. W trybie transportowym dane ze źródłowego pakietu IP są zabezpieczane, a po oryginalnym nagłówku IP pakietu jest wstawiany nagłówek IPsec. W trybie tunelowym cały źródłowy pakiet IP jest zabezpieczany (wraz z oryginalnym nagłówkiem), a następnie (wraz z nagłówkiem IPsec) wstawiany do pola danych nowego pakietu IP. Taki tunel zestawiony pomiędzy dwoma routerami pozwala na logiczne połączenie dwóch osobnych fizycznych podsieci.

IPsec definiuje dwa stosunkowo proste protokoły zabezpieczeń *IP Authentication Header* (AH) i *IP Encapsulating Security Payload* (ESP). Pierwszy z nich zapewnia kontrolę *integralności* przesyłanych danych i *uwierzytelnianie* (zob. rozdział 2.), jak również ochronę przed atakami powtórzeniowymi, ale nie szyfruje pakietów. ESP dodatkowo szyfruje dane. Wykorzystywana jest tu kryptografia symetryczna (np. AES, TripleDES). Uzgodnienie parametrów kanału, takich jak metody szyfrowania i kontroli integralności, kluczy symetrycznych, ich cykliczna zmiana i uwierzytelnianie, odbywa się z wykorzystaniem kryptografii asymetrycznej protokołem ISAKMP (ang. *Internet Security Association and Key Management Protocol*).

3.8.5. SSH

W podrozdziale 3.7.5 wspomniano o protokole *telnet* pozwalającym na podłączenie się do konsoli zdalnego komputera. Protokół ten nie zapewnia żadnych mechanizmów bezpieczeństwa i w praktyce nie jest już stosowany. Bezpiecznym odpowiednikiem, ale o dużo większych możliwościach, jest zestaw protokołów SSH (ang. *Secure Shell*) [32]. Pozwala on nie tylko na uruchomienie zdalnego terminala tekstowego, ale także na bezpieczny transfer plików (poprzez SCP i SFTP), tworzenie tuneli, przekierowywanie portów i połączeń X11 (X Windows System).

SSH działa w warstwie aplikacji i realizuje bezpieczną transmisję dzięki szyfrowaniu wybranym algorytmem symetrycznym (zob. podrozdział 2.3.2). W zależności od wersji są obsługiwane m.in. szyfry AES, Blowfish, IDEA. Klucz sesji jest uzgadniany za pomocą kryptografii asymetrycznej. Uwierzytelnianie serwera następuje na podstawie klucza, natomiast klienta na podstawie hasła albo klucza (RSA, DSA, ECDSA). Autentyczność klucza publicznego nie jest weryfikowana przez infrastrukturę PKI, zaufane klucze są przechowywane lokalnie, odpowiednio po stronie klienta i serwera.

Przy pierwszym połączeniu do nieznanego serwera klient wyświetla *fingerprint* klucza, prosząc o potwierdzenie, czy ten klucz w istocie należy do serwera, z którym chcieliśmy się połączyć. Użytkownik powinien wówczas zweryfikować ten *fingerprint* z otrzymanym od administratora serwera bezpiecznym kanałem w celu zapobieżenia atakom *man-in-the-middle*. Przy kolejnych połączeniach otrzymany klucz jest porównywany z zapisanym lokalnie. Ewentualna niezgodność powo-

duje wyświetlenie komunikatu. Może być ona spowodowana zmianą kluczy przez administratora, ale także próbą ataku *man-in-the-middle*.

Uwierzytelnianie użytkownika może nastąpić na podstawie klucza publicznego zapisanego po stronie serwera, a gdy takiego klucza nie ma – na podstawie hasła (hasło to oczywiście jest transmitowane bezpiecznym kanałem). Użytkownik może wygenerować parę kluczy (publiczny i prywatny), prywatny pozostaje zapisany lokalnie po stronie klienta, a publiczny jest kopiowany na serwer (zazwyczaj podczas pierwszego połączenia autoryzowanego hasłem).

Przekierowanie portów pozwala na utworzenie bezpiecznych tuneli pomiędzy klientem a serwerem wykorzystywanych przez dowolną aplikację sieciową. Możliwe jest przekierowanie wskazanego portu lokalnego na port zdalny docelowej maszyny, tj. uruchomienie nasłuchu na wybranym porcie lokalnym, a po nawiązaniu połączenia z tym portem – przekierowanie komunikacji przez SSH i zainicjowanie z serwera połączenia na wybranym porcie zdalnym docelowej maszyny. SSH pozwala także na analogiczne przekierowanie wskazanego portu na serwerze na wybraną maszynę i port po stronie klienta, jak również przekierowania dynamiczne. Funkcjonalność taka pozwala na skonfigurowanie aplikacji sieciowych, aby działały jakby były uruchomione na komputerze na drugim końcu połączenia. Przykładowo, możliwe jest tunelowanie niezabezpieczonych protokołów pocztowych (POP3, SMTP) między uruchomionym lokalnie czytnikiem poczty a zdalnym serwerem pocztowym albo utworzenie proxy dla przeglądarki internetowej (dzięki dynamicznemu przekierowaniu i SOCKS), by uzyskać połączenie ze stroną WWW wychodzące z adresu IP serwera.

3.9. Podsumowanie

Sieci komputerowe, a zwłaszcza Internet są istotną technologią wpływającą na kształt naszego życia społecznego i metody wytwarzania dóbr konsumpcyjnych. Globalna sieć komputerowa staje się głównym źródłem pozyskiwania i wymiany wiedzy, platformą do wymiany handlowej, a także miejscem nawiązywania kontaktów interpersonalnych. Jedną z poważnych gałęzi światowej gospodarki jest wprowadzanie kolejnych usług dostępnych z poziomu przeglądarek internetowych. Integracja sieci telefonicznych i komputerowych pozwala na dostęp do Internetu z poziomu większości urządzeń przenośnych. Sieci komputerowe stają się podstawowym medium wymiany i gromadzenia informacji zarówno w instytucjach finansowych, edukacyjnych, rządowych, jak i jednostkach wytwarzających dobra konsumpcyjne. Co więcej, sieci komputerowe integrują podsystemy mikroprocesorowe w samochodach i innych środkach transportu, umożliwiają agregację danych pochodzących z tzw. elektroniki ubieranej, pozwalają w końcu na monitorowanie i zarządzenie kompleksami miejskimi, a także instytucjami dobra publicznego.

Sieci komputerowe istotnie wpływają na rozwój społeczności i społeczeństw. Serwisy społecznościowe stają się platformą do powiadamiania obywateli o poczynaniach polityków, ale także metodą szybkiego organizowania się nieformalnych grup sprzeciwu lub poparcia wobec pewnych idei. Dostęp do sieci pozwala na zdobycie wykształcenia oraz nawiązanie bezpośredniego kontaktu z wieloma osobami zamieszkującymi dowolne zakątki Ziemi. Globalna sieć komputerowa staje się coraz ważniejszym medium przekazywania bieżących wiadomości.

Zawarte w niniejszym rozdziale studium wiedzy z zakresu sieci komputerowych miało na celu wyjaśnienie podstawowych mechanizmów i technologii umożliwiających efektywną wymianę informacji pomiędzy komputerami. Wprowadzanie wiedzy odbywało się zgodnie ze zdefiniowanym na początku warstwowym modelem odniesienia OSI. Charakteryzując kolejne warstwy modelu, odnoszono się do istniejących praktycznych rozwiązań, które funkcjonują w świecie współczesnych sieci komputerowych. Podstawowymi technologiami, jakie omówiono, były: Ethernet, stos protokołów TCP/IP oraz wybrane aplikacje sieciowe. Zrozumienie tych zagadnień – zdaniem autorów – pozwala na podjęcie świadomego dialogu z osobami na co dzień zapewniającymi naszym urządzeniom dostęp do zasobów sieci komputerowych.

3.10. Zadania

1. Porównaj architekturę połączeń typu klient-serwer oraz każdy z każdym (peer-to-peer).
2. Podaj definicję sieci komputerowej.
3. Wyjaśnij, czym różnią się dwie podstawowe technologie przesyłu danych w sieci: rozgłoszeniowa i dwupunktowa.
4. Sklasyfikuj sieci komputerowe ze względu na odległość pomiędzy ich węzłami.
5. Omów warstwowy model odniesienia OSI.
6. W jakim celu dokonuje się opakowywania informacji przed przesłaniem ich do kolejnych niższych warstw sieci?
7. Porównaj model odniesienia ISO z modelem TCP/IP.
8. Wymień i omów trzy podstawowe nośniki informacji stosowane do konstruowania sieci komputerowych.
9. Wyjaśnij, czym różni się „klasyczny Ethernet” od „Ethernetu przełączanego”.

10. Wyjaśnij zasadę działania metody dostępu do łącza CSMA/CD.
11. Wyjaśnij, czym różnią się sieciowe koncentratory (huby) od przełączników.
12. Omów ramkę danych zgodnych ze standardem 802.3.
13. Podaj przynajmniej dwa standardy komunikacji przemysłowej bazujące na technologii Ethernet.
14. Omów, w jakich podstawowych konfiguracjach można konstruować sieci bezprzewodowe.
15. Wymień co najmniej pięć problemów spotykanych przy konstruowaniu sieci bezprzewodowych, które mają marginalne znaczenie w konstruowaniu sieci z trwałymi nośnikami informacji.
16. Wyjaśnij zasadę działania metody dostępu do łącza CSMA/CS.
17. Omów ramkę danych zgodnych ze standardem 802.11.
18. Wyjaśnij, co to są algorytmy routingu.
19. Omów na przykładzie algorytm Dijkstry.
20. Omów pakiet danych zgodny z protokołem IPv4.
21. Omów pakiet danych zgodny z protokołem IPv6.
22. Omów, na czym polega bezklasowy routing bezdomenowy.
23. Omów, na czym polega mechanizm NAT.
24. Omów adresy prywatne.
25. Scharakteryzuj zadania protokołów ICMP, ARP oraz DHCP.
26. Omów pakiet danych zgodny z protokołem UDP.
27. Omów pakiet danych zgodny z protokołem TCP.
28. Omów, na czym polega mechanizm DNS.
29. Omów cztery wybrane protokoły warstwy aplikacji.

Rozdział 4.

Systemy operacyjne

Sławomir Samolej, Jan Sadolewski, Wojciech Rząsa, Dariusz Rzońca

4.1. Wprowadzenie

Oprogramowanie systemu mikroprocesorowego można tworzyć dwoma podstawowymi sposobami. **Pierwszą metodą** jest napisanie własnego kompletnego programu, który samodzielnie będzie zarządzał podsystemami komputera i wykonywał ustalane zadanie przetwarzania. Wymaga to posiadania dobrej wiedzy na temat architektury danego mikroprocesora oraz dysponowania kompilatorem języka programowania dla tego urządzenia. Rozwiązanie takie jest nadal często stosowane podczas tworzenia niezbyt skomplikowanych urządzeń przeznaczonych do wykonywania relatywnie prostych zadań przetwarzania danych. Przykładami takich aplikacji mogą być sterowniki mikroprocesorowe urządzeń AGD, podstawowe telefony komórkowe, podsystemy ABS/ESP wbudowane w samochody, autopiloty dla prostych obiektów latających. Zaletą takiego podejścia są niewątpliwie niskie koszty wywarzania całego urządzenia (tanie mikroprocesory, darmowe kompilatory), natomiast mankamentem – ograniczona przenośność i skalowalność takiego rozwiązania sprzętowo-programowego. Często się okazuje, że przeniesienie oprogramowania z jednej na drugą platformę mikroprocesorową jest związane z poważnym nakładem pracy przy ponownym opracowaniu podprogramów zarządzających podsystemami mikroprocesora i zapewniających wymianę informacji z urządzeniami wejścia-wyjścia. Poważnym wyzwaniem przy zastosowaniu takiej metody tworzenia oprogramowania staje się również samodzielne skonstruowanie aplikacji wielozadaniowych oraz takich, które obsługują bardziej zaawansowane interfejsy komunikacyjne (np. Ethernet/TCP/IP). W konsekwencji wraz z rozwojem oprogramowania oraz sprzętu komputerowego zaproponowano **drugi sposób** tworzenia oprogramowania komputerów polegający na opracowywaniu aplikacji przeznaczonych na platformę systemu operacyjnego.

Z punktu widzenia programistów oprogramowanie przygotowywane dla danego systemu operacyjnego nie odwołuje się do konkretnego systemu mikroprocesoro-

wego, ale do usług systemowych. Programista zostaje więc zwolniony z obowiązku studiowania architektury mikroprocesora, a posługuje się zbiorem podprogramów (usług), które umożliwiają mu współpracę ze swego rodzaju wirtualnym systemem mikroprocesorowym. W konsekwencji aplikacja przygotowana na dany system operacyjny będzie na nim poprawnie działała pomimo przeniesienia jej na inny komputer z innym mikroprocesorem. Dostatecznie zaawansowany system operacyjny daje możliwość uruchamiania wielu aplikacji na raz, rozstrzygając problemy wynikające z zarządzania czasem procesora, pamięcią, dostępem do urządzeń wejścia-wyjścia bez udziału programisty. System operacyjny pozwala również na znacznie swobodniejszy dobór aplikacji i konfigurację pracy systemu mikroprocesorowego. Urządzenie z takim systemem dysponuje pewną skalowaną kolekcją aplikacji dostosowaną do potrzeb danego użytkownika. Wprowadzenie sytemu operacyjnego ma również cele marketingowe. Znacznie łatwiej jest nakłonić twórców oprogramowania do tworzenia aplikacji dla danego systemu operacyjnego niż dla danej platformy sprzętowej. Znakiem czasów są portale internetowe oferujące darmowe lub relatywnie tanie aplikacje przygotowane na systemy operacyjne. Aplikacja przygotowana dla systemu operacyjnego ma także szansę na „dłuższe życie" pomimo szybkich zmian dokonywanych w platformach sprzętowych.

Warto również wspomnieć o wadach wprowadzenia systemu operacyjnego. Po pierwsze wprowadzenie systemu operacyjnego wymaga od sytemu mikroprocesorowego dodatkowego nakładu obliczeniowego. Po drugie część systemów operacyjnych i kompilatorów przeznaczonych do tworzenia dla nich oprogramowania jest płatna.

Podsumowując, kiedy złożoność oprogramowania dla systemów mikroprocesorowych przekracza pewną barierę lub kiedy wymagana jest pewna uniwersalność w doborze oprogramowania dla danego urządzenia, warte rozważenia jest zastosowanie sytemu operacyjnego – programu, który oddzieli twórców oprogramowania od platformy sprzętowej i umożliwi użytkownikom znacznie swobodniejsze konfigurowanie pracy systemu mikroprocesorowego.

W rozdziale przedstawiono wybrane zagadnienia przybliżające specyfikę systemów operacyjnych. Zdefiniowano system operacyjny, a następnie opisano kluczowe etapy rozwoju tego typu oprogramowania. Wyróżniono także podstawowe zadania systemu operacyjnego oraz mechanizmy sprzętowe wspierające jego konstruowanie. Po wprowadzeniu pojęcia zasobu omówiono techniki zarządzania procesami oraz elementy przetwarzania współbieżnego. Problemy zarządzania pamięcią, systemem plików oraz urządzeniami wejścia-wyjścia stanowią treść kolejnych trzech podrozdziałów. Ostatni podrozdział poświęcono elementom zabezpieczeń systemów operacyjnych.

4.2. Podstawowe pojęcia i historia systemów operacyjnych

4.2.1. Definicja systemu operacyjnego

System operacyjny to oprogramowanie umożliwiające zarządzanie zasobami sprzętowymi komputera dostarczające środowiska do uruchamiania i zarządzania programami użytkownika. System operacyjny pośredniczy pomiędzy programami a sprzętem komputerowym. Udostępnia czas procesora, pamięć oraz urządzenia wejścia-wyjścia poszczególnym programom z zachowaniem uzgodnionych reguł. Umożliwia również zarządzanie danymi zgromadzonymi na komputerze (system plików) oraz komunikację pomiędzy uruchomionymi procesami obliczeniowymi. Jego zadaniem jest również wykrywanie i obsługa ustalonej grupy błędów.

4.2.2. Pojęcie zasobu

System operacyjny przydziela programom uruchomionym pod jego nadzorem tak zwane zasoby. *Zasobem* jest element sprzętowy lub programowy systemu komputerowego, którego brak może potencjalnie zablokować wykonywanie programu. Za podstawowe zasoby zarządzane przez system operacyjny można uznać procesor (czas procesora), pamięć, dane oraz urządzenia wejścia-wyjścia. Przykładowo, system operacyjny może odłożyć uruchomienie kolejnego programu, gdy „dojdzie do wniosku", że nie dysponuje wystarczającą ilością pamięci.

4.2.3. Pojęcie procesu

Proces (zadanie) jest w uproszczeniu *wykonywanym programem*. Podczas uruchamiania programu w systemie operacyjnym następuje przeniesienie kodu programu z jakiegoś nośnika do pamięci operacyjnej. Kod programu jest „otaczany" strukturami danych identyfikującymi go na czas wykonywania i zapewniającymi jego ochronę (tzw. blok kontrolny procesu). Następuje także powiązanie (planowanie przydziału) ze zbiorem zasobów systemu mikroprocesorowego, z których będzie korzystał. Zasobami przyszłego procesu są czas procesora, pamięć, system plików oraz urządzenia wejścia-wyjścia. Tak przygotowany program staje się procesem, który jest gotowy do wykonywania.

4.2.4. Historia systemów operacyjnych

Zalążkami systemów operacyjnych były biblioteki programów pozwalające na komunikację z urządzeniami wejścia-wyjścia dołączane do pierwszych programów. Programiści wywoływali wtedy umowne instrukcje zapisu lub odczytu do urządzenia bez wnikania w jego architekturę. Pierwsze systemy operacyjne,

tzw. *wsadowe systemy operacyjne*, pozwalały na wczytanie grupy zadań obliczeniowych (wsadu) i kolejne ich wykonywanie. Kiedy zauważono, że podczas realizacji operacji wejścia-wyjścia przez pojedynczy program procesor nie wykonuje żadnych obliczeń, oczekując na dane, opracowano *wieloprogramowe systemy operacyjne*. Podczas gdy jedno zadanie oczekiwało na komunikację z urządzeniami wejścia-wyjścia, zawieszane było jego działanie i rozpoczynane wykonywanie następnego zadania gotowego do przetwarzania. Kolejnym znaczącym etapem w rozwoju systemów operacyjnych było wprowadzenie *podziału czasu obliczeń*. W tych systemach przełączanie pomiędzy zadaniami następowało dodatkowo po upłynięciu odcinka czasu. Umożliwiło to przekształcenie komputera działającego pod kontrolą systemu operacyjnego w urządzenie interaktywne. Szybkie przełączenia pomiędzy wykonywanymi zadaniami pozwalały na bieżąco zarządzać pracą komputera i wchodzić z nim w interakcję przez bieżące uruchamianie i monitorowanie pracy programów. Systemy operacyjne przeżyły kolejną przemianę z chwilą opracowania komputerów wieloprocesorowych. *Wieloprocesorowe systemy operacyjne* umożliwiły znaczące zwiększenie wydajności przetwarzania, zwłaszcza gdy oprogramowanie składało się z wielu niezależnych, wykonywanych współbieżnie procesów. Kolejnym stadium rozwoju były *rozproszone systemy operacyjne* – oprogramowanie pozwalające na scalenie wielu komputerów w jeden wirtualny komputer wykonujący wskazane zadania przetwarzania i udostępniania danych. Poważnym etapem rozwoju systemów było wprowadzenie *wirtualizacji*. Dany system operacyjny oferuje wirtualny system mikroprocesorowy, na którym można zainstalować kolejny system operacyjny. Pierwotnie takie instalowanie wirtualnych systemów pozwalało na korzystanie z mocy obliczeniowych dużych komputerów przez indywidualnych użytkowników, którzy wykonywali przetwarzanie danych na zdalnych komputerach, posługując się wirtualną wersją typowego systemu operacyjnego komputerów personalnych. Obecnie wirtualizacja może być oparta na mocy obliczeniowej typowych komputerów personalnych oraz możliwości ich systemów operacyjnych.

Osobną gałęzią systemów operacyjnych są *systemy operacyjne czasu rzeczywistego*. Są to wielozadaniowe systemy operacyjne umożliwiające uruchamianie zbioru zadań, dla których można wyznaczyć precyzyjny czas wykonywania obliczeń. Systemy takie pozwalają na tworzenie aplikacji, które generują wyniki w przewidywalnych chwilach czasu, pozwalając na efektywne sterowanie lub zarządzanie.

Za ciekawostkę można uznać to, że omówiona historia rozwoju systemów operacyjnych odbyła się jak dotąd trzykrotnie. Pierwszy cykl obejmował lata 50 – 80. zeszłego wieku i dotyczył rozwoju oprogramowania komputerów *main frame* i minikomputerów. Systemy operacyjne komputerów personalnych również ewoluowały od od prostych jednozadaniowych do wielozadaniowych i sieciowych

(lata 80 – 90. ubiegłego wieku). W czasie tworzenia niniejszego opracowania ten swoisty cykl rozwoju kończą urządzenia mobilne (m.in. tablety i smartfony).

Nowym zjawiskiem powiązanym z problematyką systemów operacyjnych są obliczenia w chmurze. Oprogramowanie systemowe większości współczesnych komputerów pozwala na przechowywanie danych i ich przetwarzanie w wirtualnym komputerze dostępnym przez połączenie z Internetem, przy czym przeciętny użytkownik przestaje być świadomy, jakie komputery przetwarzają jego dane i gdzie fizycznie są zgromadzone dane.

4.3. Komponenty i warstwy systemów operacyjnych

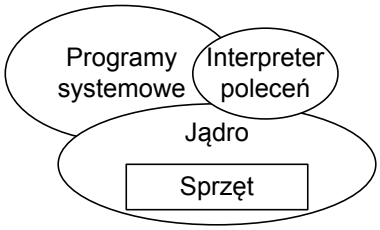
W większości systemów operacyjnych można wyróżnić komponenty odpowiedzialne za następujące zadania:

- zarządzanie procesami,
- zarządzanie pamięcią operacyjną,
- zarządzanie systemem plików,
- zarządzanie systemem wejścia-wyjścia,
- pracę sieciową,
- system ochrony,
- system interpretacji poleceń.

W dalszej części podrozdziału bardziej szczegółowo omówiono zadania wybranych komponentów.

Oprogramowanie systemów operacyjnych tworzy się zwykle z kilku warstw dostarczających coraz bardziej zaawansowanych mechanizmów (por. rys. 4.1). W uproszczeniu można powiedzieć, że najniższą warstwą systemu są *sterowniki urządzeń* wchodzących w skład systemu mikroprocesorowego. Ze sterowników urządzeń korzysta tzw. *jądro systemu*. Jest to główny program zarządzający systemu operacyjnego zajmujący się zarządzaniem procesami, pamięcią, urządzeniami wejścia-wyjścia, systemem plików oraz podstawową obsługą błędów. Wyróżnionym programem systemu jest *interpreter poleceń*. Przyjmuje on od programów lub od użytkownika określone komendy. Jeśli komenda jest poleceniem systemowym, to interpreter ją wykonuje, jeśli nie, to stara się znaleźć wśród tzw. programów systemowych (lub programów użytkownika) ten, który dane polecenie może wykonać. Ostatnią warstwą systemu są programy (systemowe). W tym miejscu przenikają się tak naprawdę programy stanowiące uzupełnienie jądra systemu oraz programy,

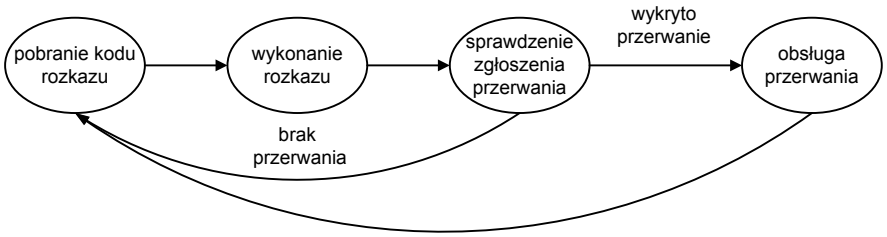
które użytkownik dołączył (zainstalował) do systemu. Z punktu widzenia ich uruchamiania na komputerze są to już zewnętrzne składniki systemu operacyjnego.



Rys. 4.1. Podstawowe warstwy systemu operacyjnego

4.4. Wybrane mechanizmy sprzętowe wspierające działanie systemu operacyjnego

W konstruowaniu systemów operacyjnych pomocne są wybrane rozwiązania sprzętowe wbudowane w mikroprocesory. Na rysunku 4.2 pokazano uproszczony schemat wykonywania pojedynczej instrukcji procesora.

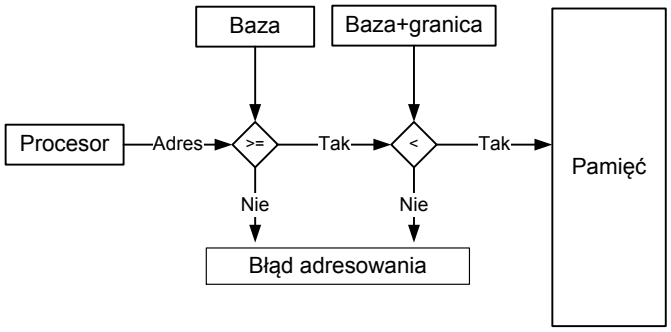


Rys. 4.2. Uproszczony schemat wykonywania instrukcji w procesorze

Po wykonaniu każdego z rozkazów procesor sprawdza, czy w systemie nie nastąpiło zgłoszenie *przerwania* – sygnału zewnętrznego świadczącego o zaistnieniu jakiegoś zdarzenia (np. urządzenie wejścia-wyjścia zakończyło działanie). Poprawnie skonstruowany system operacyjny przejmuje obsługę przerwań i dzięki temu może po wykonaniu niemal każdego rozkazu procesora przejąć kontrolę nad innymi programami. Szczególnym przerwaniem obsługiwanym przez systemy operacyjne jest *przerwanie czasowe*, którego obsługa może decydować, co jaki przedział czasu nastąpi przełączenie pomiędzy wykonywanymi programami w systemach z podziałem czasu.

Systemy operacyjne korzystają również z układów czasowych *watchdog* wbudowanych w mikroprocesory. Układy tego typu oczekują od systemu potwierdzenia poprawności działania co pewien określony przedział czasu. W przypadku braku takiego zgłoszenia istnieje duże prawdopodobieństwo, że system wszedł w tryb nieprzewidziany w jego działaniu i nastąpiła awaria. Wówczas może nastąpić ponowne uruchomienie sytemu operacyjnego.

Poprawnie skonstruowany system operacyjny powinien pracować przynajmniej w dwu podstawowych trybach: systemowym i użytkownika. W trybie systemowym dozwolone są wszystkie operacje zdefiniowane w systemie, podczas gdy w trybie użytkownika część z nich jest zabroniona w celu ochrony integralności systemu. Współczesne mikroprocesory posiadają specjalny *rejestr* wskazujący, w jakim trybie w danej chwili pracuje system operacyjny. W trybie systemowym odbywają się na przykład wszystkie polecenia obsługi wejścia-wyjścia. Użytkownik nie ma do nich bezpośredniego dostępu. Aby program użytkownika wykonał taką operację, musi zwrócić się do systemu operacyjnego (wywołać usługę).



Rys. 4.3. Podstawowa sprzętowa ochrona pamięci

Systemy mikroprocesorowe wspierają również *ochronę pamięci*. Oczekuje się, że każdy z programów wykonywanych w systemie operacyjnym będzie dysponował swoim własnym obszarem pamięci operacyjnej i inne programy nie będą mogły tego obszaru pamięci modyfikować. Takiego rodzaju ochronę można zapewnić przez zdefiniowanie dwu rejestrów: bazowego i granicznego. Rejestr bazowy przechowuje najniższy adres pamięci operacyjnej, z jakiego korzysta dany program, a graniczny – rozmiar pamięci, jaki programowi przydzielono. Na rysunku 4.3 pokazano zasadę wykorzystania tych rejestrów przez system mikroprocesorowy. Podczas wykonywania programu przy odwoływaniu się do pamięci sprzęt zawsze odpytuje system operacyjny, czy dany program odwołuje się do tego obszaru, który mu przydzielono.

4.5. Zarządzanie procesami

4.5.1. Blok kontrolny procesu

Pojęcie procesu wprowadzono w podrozdziale 4.2.3. Struktura danych „otaczająca" dany proces w systemie operacyjnym nazywa się *blokiem kontrolnym procesu* i zawiera między innymi następujące właściwości:

- *identyfikator* – unikalna liczba przydzielana procesowi w momencie jego tworzenia; dzięki niej możliwe jest wykonanie operacji zarządzania procesem przez system i operatora,
- *stan* – informacja, w jakim stanie jest w danej chwili proces („Nowy", „Gotowy" itp. -– stany procesu zostaną omówione w dalszej części rozdziału),
- *priorytet* – priorytet procesu w odniesieniu do innych procesów (priorytet jest wykorzystywany do ustalenia, kiedy dany proces ma prawo przejąć procesor),
- *licznik programu* – adres kolejnej instrukcji programu, która ma być teraz wykonywana,
- *wskaźniki pamięci* – informacja, gdzie znajduje się w pamięci kod, dane oraz wartości rejestrów granicznych,
- *rejestry* – zachowany stan rejestrów procesora z ostatniej partii obliczeń procesu,
- *informacja na temat urządzeń wejścia-wyjścia* – lista plików i urządzeń, z których korzysta proces,
- *informacje ewidencyjne* – dane służące systemowi operacyjnemu do opracowywania statystyk (np. czas używania procesora, opóźnienia wykonania, liczba operacji wejścia-wyjścia itp.).

Blok kontrolny procesu zawiera wystarczające informacje dla systemu operacyjnego, aby efektywnie można było wstrzymać, a następnie wznowić wykonywanie procesu.

4.5.2. Stany procesu

Można przyjąć, że współczesne systemy operacyjne składają się ze zbioru współbieżnie wykonywanych procesów. Część z nich jest wprowadzona do systemu przez użytkownika, inne zaś są procesami systemowymi. Wszystkie jednak na poziomie systemu operacyjnego są zarządzane zgodnie z ustalonymi ogólnymi

regułami. Na rysunku 4.4 pokazano podstawowy diagram stanów, w jakich znajduje się proces w systemie operacyjnym.

Rys. 4.4. Diagram stanów procesu w systemie operacyjnym

Po utworzeniu procesu jest on w stanie *nowy*. Kiedy nowy proces może być wykonywany, to przechodzi do stanu *gotowy*. Gotowy proces oczekuje na przydzielanie czasu procesora. Z chwilą udostępnienia mu procesora proces jest w stanie *uruchomiony*. Wykonywanie obliczeń może być przerwane trzema podstawowymi zdarzeniami. Po pierwsze proces może skończyć swoje obliczenia i przejść do stanu *zakończony*. Po drugie przerwa w wykonywaniu procesu (przejście pomiędzy stanami *uruchomiony* a *gotowy*) pojawia się zwykle, gdy upływa maksymalny dozwolony czas wykonywania dla danego procesu i system operacyjny przydziela czas kolejnemu z gotowych procesów. Inną przyczyną tego przejścia może być pojawienie się w systemie zadania o wyższym priorytecie, co w niektórych systemach powoduje natychmiastowe przełączenie pomiędzy zadaniami. Po trzecie proces zostaje *zablokowany*, jeśli żąda dostępu do zasobu, na który musi czekać lub inicjuje operację wejścia-wyjścia, na której obsługę prawie zawsze musi poczekać. Przejście pomiędzy stanami *zablokowany* a *gotowy* jest rezultatem zaistnienia zdarzenia, na które czekał proces (np. zasób jest zwolniony lub rezultaty operacji wejścia-wyjścia są dostępne).

System operacyjny przechowuje informacje o zbiorze wszystkich działających pod jego nadzorem procesów. W danej chwili działania systemu operacyjnego pewien zbiór procesów jest gotowy do wykonywania, pewien zbiór – zablokowany, jeden lub pewna liczba (w zależności, czy system komputerowy jest jedno- czy wieloprocesorowy) procesów jest wykonywana. Część procesów jest w stanie inicjalizacji, a część jest zakończonych. W typowym przypadku informacja ta służy do takiego manipulowania przydziałem zasobów dla poszczególnych procesów,

aby jak największa liczba zadań zakończyła swoje obliczenia w jak najkrótszym czasie.

4.5.3. Przełączanie i planowanie procesów

Wykonywanie zbioru procesów, z których składa się system operacyjny, wymaga wypracowania strategii zarządzania. Kluczowym mechanizmem staje się więc przełączanie procesów umożliwiające bardziej efektywne wykorzystanie procesora (np. gdy jakiś proces czeka na zasób, inny może w tym czasie wykonywać obliczenia) lub zapewnienie interaktywności (np. oprócz procesów obliczeniowych przydziela się pewien odcinek czasu programowi, z którym użytkownik pozostaje w interakcji). Problem przełączania musi być rozważony niezależnie od liczby procesorów czy rdzeni dostępnych w danym komputerze. W praktycznych rozwiązaniach rzadko zdarza się, że komputer dysponuje wystarczającą liczbą rdzeni, tak aby każdy z uruchomionych procesów dysponował co najmniej jednym.

Przełączenie procesu polega na zawieszeniu jego wykonania i przekazaniu możliwości wykonywania innemu procesowi. Z przełączaniem wiąże się szereg operacji związanych z zapamiętywaniem bieżącego stanu procesu, tak aby w chwili ponownego jego uruchomienia w dalszym ciągu program był wykonywany poprawnie. W chwili przełączenia:

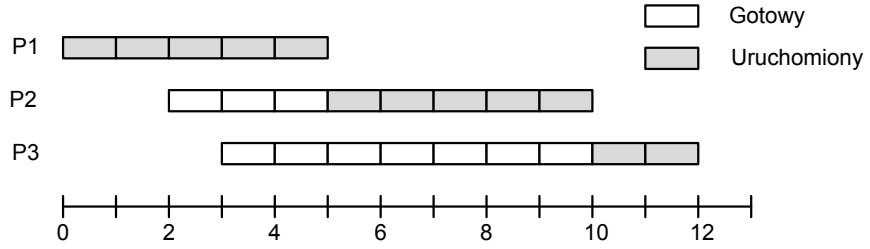
- jest zapisywany stan rejestrów procesora,
- jest aktualizowany blok kontrolny procesu (stan procesu, informacje ewidencyjne),
- jest wybierany inny proces do wykonywania,
- jest przywracany stan rejestrów procesora dla nowego procesu.

Decyzja o przełączeniu procesów jest podejmowana zwykle w następujących sytuacjach:

1. Proces przechodzi ze stanu *uruchomiony* do *zablokowany* (np. z powodu oczekiwania na zakończenie operacji wejścia-wyjścia).
2. Proces przechodzi ze stanu *uruchomiony* do *gotowy* (np. z powodu upłynięcia czasu procesora, który był mu przydzielony).
3. Proces przechodzi ze stanu *zablokowany* do *gotowy* (np. z powodu zakończenia operacji wejścia-wyjścia).
4. Proces kończy działanie (przechodzi ze stanu *uruchomiony* do *zakończony*).

W sytuacjach 1. i 4. system operacyjny musi przekazać sterowanie nowemu procesowi. Natomiast w sytuacjach 2. i 3. należy podjąć decyzję, czy kontynuować pracę poprzedniego procesu czy też należy wykonać przełączenie.

Jeżeli system operacyjny podejmuje decyzje o przekazaniu sterowania tylko w przypadkach 1. i 4., to posługuje się on *niewywłaszczeniowym* schematem przydziału czasu procesora. W takim schemacie dany proces wykonuje obliczenia do końca lub do chwili, gdy zostanie zablokowany. Bardziej zaawansowanym schematem planowania czasu procesora jest schemat *wywłaszczeniowy*, który zakłada, że wykonywanie bieżącego procesu może być w każdej chwili zawieszone na rzecz innego procesu. Przykładowo, wykonywany w danej chwili proces może zostać przełączony przed zakończeniem przez niego obliczeń, ponieważ w systemie „pojawił się" nowy proces uznany przez system operacyjny za ważniejszy.

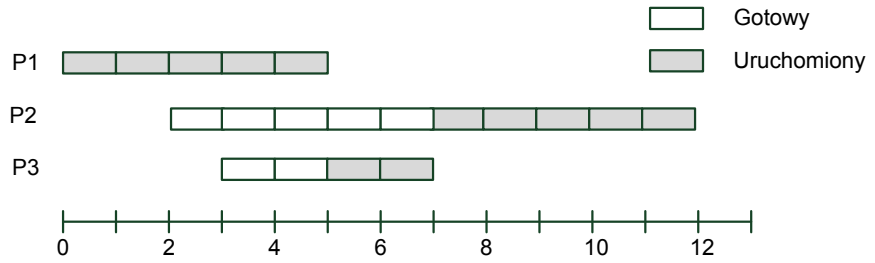


Rys. 4.5. Przykład planowania zgodnie z algorytmem FCFS

Wybór kolejnego procesu do wykonywania zależy od przyjętego algorytmu planowania. Najprostszym algorytmem planowania jest FCFS (ang. *First Come, First Served*) – „pierwszy zgłoszony – pierwszy obsłużony". W przypadku zastosowania tego algorytmu proces, który pierwszy zażąda dostępu do procesora, pierwszy go otrzymuje. Algorytm FCFS stosuje niewywłaszczeniowy schemat przydziału procesora. Zasadę działania tego algorytmu zilustrowano na rys. 4.5. Ponieważ proces P1 był jedynym procesem zgłoszonym do obsługi w chwili czasu 0, został przydzielony do wykonywania. W chwili czasu 2 gotowy do wykonywania był proces P2, jednak –ponieważ był przetwarzany proces P1 – rozpoczęcie jego wykonywania zostało odroczone do chwili czasu 5. Na podobnej zasadzie zostało odroczone wykonywanie procesu P3, gotowego do przetwarzania w chwili czasu 3, ostatecznie przeznaczonego do uruchomienia w chwili czasu 10.

Algorytmem planowania pozwalającym na zakończenie w ustalonym przedziale czasu większej liczby zadań obliczeniowych jest SJF (ang. *Shortest Job First*) – „najkrótsze zadanie pierwsze". Algorytmowi „dostarcza się" dodatkową informację wskazującą, ile czasu danemu procesowi zajmie wykonywanie swo-

ich obliczeń. W czasie selekcji kolejnego procesu do wykonywania wybiera się ten, który „deklaruje", że najszybciej skończy swoje obliczenia. W konsekwencji system komputerowy zwiększa swoją przepustowość. Algorytm planowania SJF może być stosowany zarówno z wywłaszczeniowym, jak i niewywałszczeniowym schematem przydziału czasu procesora.

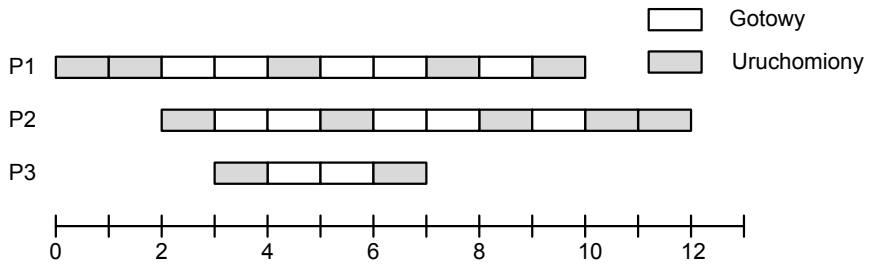


Rys. 4.6. Przykład planowania zgodnie z algorytmem SJF

Zasadę działania tego algorytmu przedstawiono na rys. 4.6. Ponieważ proces P1 był jedynym procesem zgłoszonym do obsługi w chwili czasu 0, został on przydzielony do wykonywania. W chwili czasu 2 gotowy do wykonywania był proces P2, a w chwili czasu 3 – proces P3. Kiedy proces P1 zakończył obliczenia, algorytm planowania wybrał do wykonywania ten proces, który „zadeklarował", że w krótszym czasie zakończy swoje obliczenia. W rezultacie w chwili czasu 7 już dwa procesy zakończyły swoje obliczenia.

Algorytmem planowania dostosowanym do wywłaszczeniowego schematu przydziału procesora i skierowanym szczególnie do systemów interaktywnych jest planowanie karuzelowe. W tym algorytmie wyznacza się pewien kwant czasu, który cyklicznie będzie przydzielany każdemu z gotowych procesów. W każdym kwancie czasu obliczenia prowadzi jeden z uprzednio gotowych procesów. Po upłynięciu wyznaczonego kwantu następuje przydzielenie czasu procesora kolejnemu z gotowych procesów. Dzięki takiej strategii nie uzyskuje się skrócenia czasu obliczeń, jednak użytkownik zyskuje wrażenie, że wszystkie procesy są ciągle aktywne, w szczególności proces, z którym pozostaje w bezpośredniej interakcji.

Zasadę działania tego algorytmu zilustrowano na rys. 4.7. Podobnie jak we wcześniejszych przykładach, ponieważ proces P1 był jedynym procesem zgłoszonym do obsługi w chwili czasu 0, został on przydzielony do wykonywania. Kiedy w chwili czasu 2 do puli gotowych do wykonywania procesów dołączył P2, otrzymał on kwant czasu procesora na przeprowadzenie części swoich obliczeń. Podobnie, w chwili czasu 3 do puli rotacyjnie wykonywanych procesów został włączony proces P3. W kolejnych chwilach czasu 4, 5, 6 czas procesora otrzy-



Rys. 4.7. Przykład planowania karuzelowego

mywały kolejno procesy P1, P2, P3. Po zakończeniu obliczeń przez proces P3 naprzemiennie przydzielano czas procesom P1 i P2.

Kluczowym problemem w implementacji tego algorytmu jest odpowiednie dobranie odcinka (kwantu) czasu pomiędzy kolejnymi przełączeniami. Zbyt długi odcinek może spowodować dyskomfort użytkownika, który dostrzeże, że jego aplikacja nie reaguje na jego sygnały z dostateczną szybkością. Z kolei ustalenie zbyt krótkiego czasu pomiędzy kolejnymi przełączeniami może spowodować zjawisko, w który system operacyjny „zajmuje się" głównie wywłaszczaniem procesów, pozostawiając im zbyt mało czasu na przeprowadzanie jakichkolwiek obliczeń.

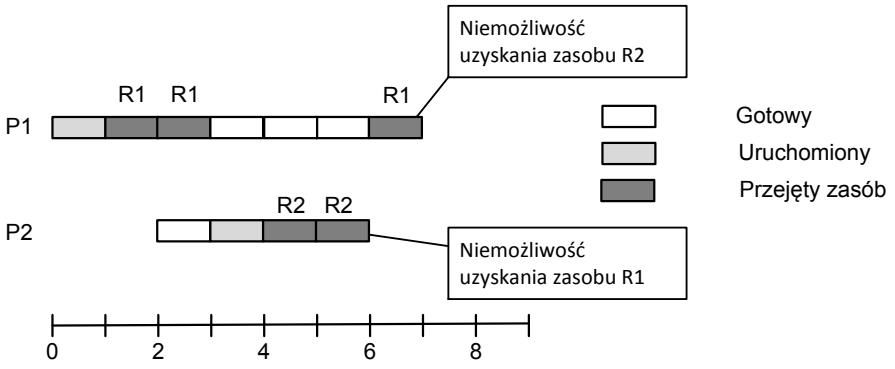
Na rysunkach 4.5–4.7 pokazano planowanie zadań do wykonywania w systemach z jednym mikroprocesorem. W systemach wieloprocesorowych system operacyjny dokonuje dodatkowego przydziału procesów do poszczególnych procesorów. W prostszych rozwiązaniach jeden z procesorów przydziela się jako nadrzędny i realizujący funkcje systemowe, pozwalając innym procesorom realizować tylko „zwykłe" zadania obliczeniowe. W systemach bardziej zaawansowanych wszystkie procesory są traktowane jako równorzędne i mogą wywoływać polecenia jądra systemu, co jednak wymaga bardziej zaawansowanego mechanizmu zarządzania zasobami komputera.

4.6. Współbieżność

Jeżeli system operacyjny składa się ze zbioru współbieżnie wykonywanych procesów, wcześniej czy później dochodzi do sytuacji, kiedy rywalizują one o pewne zasoby systemu mikroprocesorowego. Jeśli jeden z nich otrzyma dostęp do takiego zasobu, to inne muszą poczekać do chwili, gdy zasób zostanie zwolniony. Podejmując analizę takiego zjawiska, można dojść do wniosku, że podczas działania systemu mogą wystąpić trzy podstawowe problemy. Pierwszym z nich jest koniecz-

ność wprowadzenia mechanizmu *wzajemnego wykluczania*. Jeśli kilka procesów żąda naraz dostępu do jednego zasobu, to zasób staje się zasobem krytycznym, a część programu, która wymaga dostępu do takiego zasobu, nazywa się *sekcją krytyczną*. W systemie operacyjnym muszą się znaleźć mechanizmy zapewniające, że tylko jeden program wykonuje sekcję krytyczną.

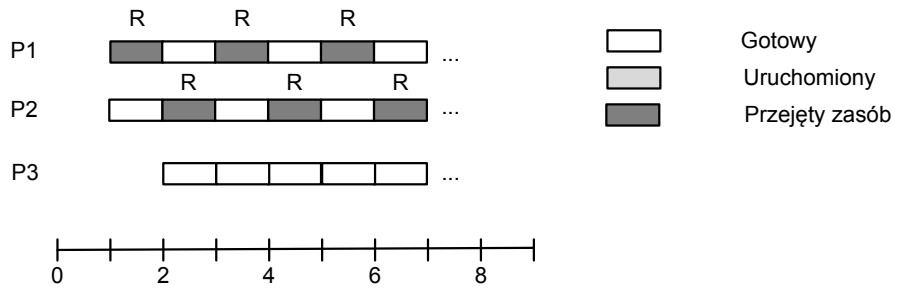
Konsekwencją istnienia sekcji krytycznej są dwa kolejne problemy: *zakleszczenie* oraz *zagłodzenie*. Przykładem zakleszczenia może być następujące zjawisko. Załóżmy, że w systemie istnieją dwa procesy P1 i P2 oraz dwa zasoby R1 i R2. Załóżmy również, że do wykonywania każdego z procesów są potrzebne oba zasoby. Proces P1 potrzebuje najpierw dostępu do zasobu R1, a następnie – przy utrzymaniu dostępu do zasobu R1 – również dostępu do zasobu R2. Z kolei proces P2 potrzebuje najpierw dostępu do zasobu R2, a następnie przy utrzymaniu dostępu do zasobu R2 – również dostępu do zasobu R1. Możliwą sekwencję wydarzeń przy przedstawionych założeniach ilustruje rys. 4.8.



Rys. 4.8. Przykład zakleszczenia

Proces P1 uzyskuje czas procesora i wykonuje obliczenia, a następnie przejmuje zasób R1. Decyzją programu planującego proces P1 zostaje wywłaszczony przez proces P2. Proces P2 po wykonaniu porcji obliczeń przejmuje zasób R2. Do dalszych obliczeń potrzebuje również zasobu R1. Ponieważ zasób R1 jest przejęty przez proces P1, to proces P2 zostaje zawieszony, a obliczenia wznawia proces P1. W chwili czasu 7 proces P1 oczekuje przydzielenia zasobu R2, który został już zajęty przez proces P2. W tej sytuacji doszło do zakleszczenia procesów P1 i P2, ponieważ dalsze ich wykonywanie nie jest możliwe ze względu na wzajemnie zablokowane zasoby.

Do zjawiska zagłodzenia dochodzi, kiedy pewien proces, pomimo że potencjalnie może uzyskać dostęp do zasobu, nigdy go nie otrzymuje. Przykładem zagłodzenia może być następujące zjawisko (por. rys. 4.9): W systemie istnieją



Rys. 4.9. Przykład zagłodzenia

trzy procesy P1, P2 i P3. Wszystkie wymagają okresowego dostępu do zasobu R. Proces P1 przejmuje zasób, procesy P2 i P3 oczekują na niego. Kiedy proces P1 zwalnia zasób, przejmuje go proces P2. Z kolei, gdy zasób jest zwolniony przez proces P2, jest on przejmowany ponownie przez proces P1. Może to np. wynikać z priorytetów przypisanych procesom P1 i P2. W konsekwencji proces P3 pozostaje „głodzony" i nie może prowadzić swoich obliczeń.

Podstawowym mechanizmem umożliwiającym rozwiązanie problemu wzajemnego wykluczania dostarczanym przez system operacyjny jest *semafor*. Ogólna definicja semafora mówi, że jest to pewna zmienna będąca liczbą całkowitą, na której można wykonać tylko operacje, takie jak:

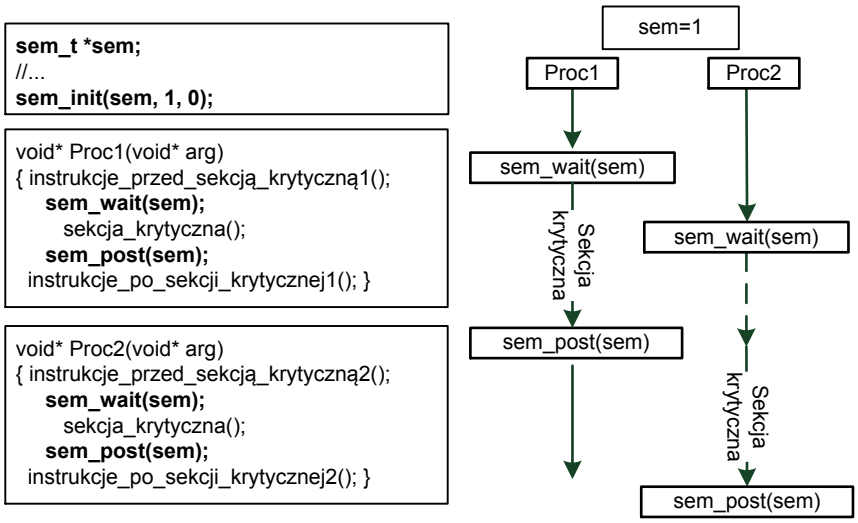
1. Inicjalizacja (*sem_init*) – nadanie wartości początkowej, większej lub równej 0.

2. Czekanie / opuszczenie (*sem_wait*) – jeśli wartość semafora jest większa od 0, to zmniejsza się go o 1; jeśli wartość semafora jest równa 0, to następuje zablokowanie procesu, który wykonuje tę operację.

3. Sygnalizuj / podnoszenie (*sem_post*) – jeśli istnieją jakieś procesy zablokowane na semaforze, to następuje odblokowanie jednego z nich; jeśli brak jest procesów zablokowanych na semaforze, to należy zwiększyć jego wartość o 1.

Zmiany wartości semafora muszą się odbywać w sposób *atomowy* – jeśli jakiś proces wykonuje pewną operację na semaforze, to wykonanie jakiejkolwiek operacji na tym semaforze przez inny proces jest zabronione. Atomowość operacji na semaforze zapewnia system operacyjny.

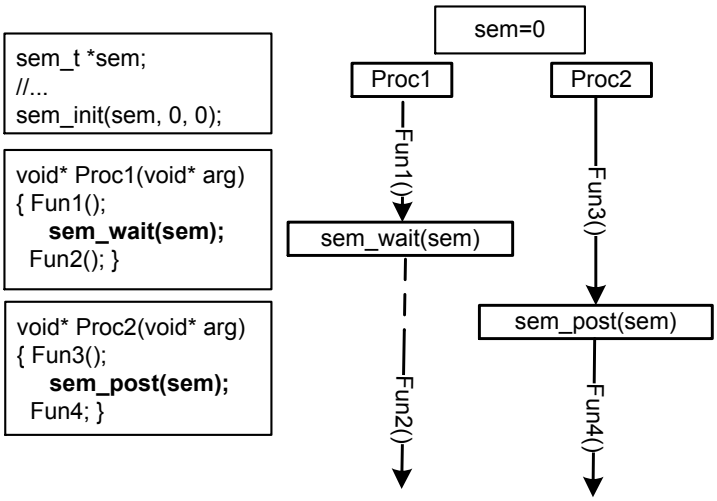
Rozwiązanie problemu sekcji krytycznej z zastosowaniem semafora zilustrowano na rys. 4.10. Przyjęto notację funkcji sterujących pracą semafora zgodnie ze



Rys. 4.10. Rozwiązanie problemu sekcji krytycznej

specyfikacjami POSIX [14] i ANSI C [2]. Funkcje Proc1 i Proc2 należy traktować jako kody źródłowe dwu współbieżnie wykonywanych procesów. W każdym z nich została wyróżniona funkcja o nazwie *sekcja_krytyczna()*, która w danej chwili może być wykonywana tylko przez jeden z procesów. Aby zapewnić wzajemne wykluczanie w dostępie do tego fragmentu programu, w systemie operacyjnym został powołany semafor o nazwie *sem*, który został zainicjalizowany początkową wartością 1. Jeśli proces Proc1 pierwszy wywoła operację *sem_wait*, to spowoduje, że wartość semafora zostanie zmniejszona o 1 i wyniesie 0. Proces Proc1 rozpocznie wykonywanie sekcji krytycznej. Jeśli w czasie wykonywania sekcji krytycznej przez proces Proc1 chęć wykonywania sekcji zgłosi proces Proc2, to podczas wywoływania operacji *sem_wait* zostanie zablokowany. Proces Proc1 po zakończeniu wykonywania sekcji krytycznej wywołuje operację *sem_post* odblokowującą proces Proc2, który teraz może „wejść” do uprzednio zablokowanej sekcji. Po zakończeniu wykonywania sekcji przez proces Proc2 ponownie jest wywoływana operacja *sem_post*, która zgodnie z podaną wcześniej definicją tym razem zwiększy wartość semafora o 1, przywracając jego stan początkowy.

Semafor w łatwy sposób pozwala również na synchronizację procesów, czyli ustalenie, że wskazane obliczenia w jednym procesie nastąpią na pewno po zakończeniu określonych obliczeń w innym procesie. Technikę synchronizacji dwu procesów z zastosowaniem semafora pokazano na rys. 4.11. Funkcje Proc1 i Proc2 należy traktować jako kody źródłowe dwu współbieżnie wykonywanych procesów.



Rys. 4.11. Przykład rozwiązania synchronizacji

Utworzony w systemie semafor o nazwie *sem* jest inicjalizowany wartością początkową 0. Wywołanie w procesie Proc1 operacji *sem_wait*, a w procesie Proc2 operacji *sem_post* gwarantuje, że wykonywanie funkcji *Fun2()* nie rozpocznie się przed zakończeniem wykonywania funkcji *Fun3()*. Jeśli proces Proc1 zakończy wykonywanie funkcji *Fun1()*, a proces Proc2 kontynuuje wykonywanie funkcji *Fun3()*, to proces Proc1 zostanie zablokowany na semaforze *sem*. Dopiero po zakończeniu wykonywania przez proces Proc2 funkcji *Fun3()* następuje wywołanie operacji *sem_post*, co umożliwia kontynuowanie obliczeń przez proces Proc1. Jeśli natomiast wykonywanie funkcji *Fun3()* zakończy się przed zakończeniem wykonywania funkcji *Fun1()*, to proces Proc1 może kontynuować obliczenia bez konieczności oczekiwania na proces Proc2.

Z zastosowaniem semaforów można również rozwiązać kolejne zagadnienia związane z komunikacją i synchronizacją procesów, a mianowicie problemy producent-konsument oraz czytelnicy i pisarze. Szczegółowe omówienie zasad konstruowania systemów współbieżnych rozwiązujących wspomniane zagadnienia można znaleźć między innymi w pracy [43].

Problem producent-konsument rozważa się w systemie operacyjnym, gdy zamierza się przekazywać dane pomiędzy działającymi procesami. Pewna grupa procesów wysyła dane, inna zaś – odbiera je. Przekazywanie danych odbywa się zwykle przez bufor, do którego należy zapewnić wzajemne wykluczanie. Rów-

nocześnie nie należy zapominać, że sam bufor może mieć w rzeczywistych rozwiązaniach ograniczoną pojemność. Należy również pamiętać, że próba zapisu do przepełnionego bufora może skutkować zablokowaniem procesu producenta. Na zablokowanie może być również narażony konsument, który próbuje odczytywać dane z pustego bufora.

Problem czytelników i pisarzy rozważa się w systemie operacyjnym, gdy ze współdzielonego zasobu (np. strony WWW) może na raz *odczytywać* dane wiele procesów (użytkowników, czytelników). Natomiast w czasie, gdy dane w zasobie są *zapisywane*, to dostęp do niego może mieć tylko jeden proces (użytkownik, pisarz). Czytelnicy mają więc prawo współbieżnie odczytywać dane, jednak w czasie pracy pisarza dostęp do czytelni musi być zablokowany.

Odpowiednie zastosowanie semaforów pozwala na zbudowanie gotowych mechanizmów rozwiązujących omówione problemy w postaci gotowych podprogramów. Efektywną komunikację pomiędzy procesami realizuje się z zastosowaniem *systemowych kolejek*, a także *interfejsy komunikacji sieciowej*.

Podsumowując, we współczesnych systemach operacyjnych istotnym zagadnieniem jest współbieżność – możliwość równoczesnego wykonywania procesów. Wraz z możliwością współbieżnego wykonywania programów system operacyjny powinien dostarczać mechanizmów do ich komunikacji i synchronizacji.

4.7. Zarządzanie pamięcią operacyjną

W komputerze zarządzanym systemem operacyjnym pamięć jest oprócz procesora drugim podstawowym zasobem. Podstawowymi zadaniami systemu operacyjnego w stosunku do pamięci są:

- przydział i odzyskiwanie,
- ochrona,
- udostępnianie w celu współdzielenia,
- transformacja adresów,
- transfer informacji pomiędzy różnymi rodzajami pamięci.

System operacyjny powinien umożliwić przydzielenie pamięci maksymalnej liczbie programów, które o nią aplikują. Kiedy dany proces kończy swoją pracę, system powinien również odnotować zwolnienie przez niego pamięci i dostarczyć pamięć następnemu procesowi.

Ochrona pamięci na poziomie systemu operacyjnego polega przede wszystkim na sprawdzeniu, czy dany proces nie odwołuje się do obszaru pamięci zarezerwowanego dla innego procesu lub dla systemu operacyjnego. Podstawową technikę

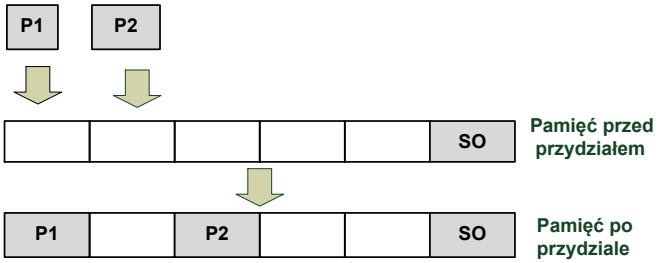
ochrony pamięci pokazano w podrozdziale 4.4. W czasie odwoływania się przez proces do pamięci następuje porównanie wartości adresu z rejestrem granicznym. System dopuści tylko do takich odwołań, których adres jest mniejszy od zapisanego w tym rejestrze.

Na potrzebę wymiany informacji pomiędzy procesami system powinien również dostarczyć mechanizmów wyznaczania obszarów pamięci, do których prawo zapisu i odczytu mają grupy procesów. Dzięki temu można zaproponować techniki komunikacji pomiędzy różnymi komponentami systemu.

Ponieważ podczas pracy typowego komputera w pamięci jest umieszczonych wiele programów, system operacyjny musi dostarczyć mechanizmów transformujących adresy zapisane w kodzie każdego programu (adresy logiczne) na adresy konkretnych komórek pamięci dostępnych w systemie mikroprocesorowym (adresy fizyczne). Podstawowa technika wspierająca przeadresowanie pamięci została pokazana w podrozdziale 4.4 i opiera się na zastosowaniu tak zwanego rejestru bazowego. Adres logiczny wystawiany przez program jest automatycznie zwiększany o wartość rejestru bazowego, co pozwala na umieszczenie programu w ustalonym przez system miejscu pamięci operacyjnej.

W wielu systemach operacyjnych dopuszcza się przenoszenie bloków pamięci operacyjnej do pamięci masowej (np. dysk twardy, dysk SSD, pamięć FLASH). Zadaniem systemu operacyjnego jest w tym wypadku efektywne zarządzanie procesem wymiany informacji pomiędzy pamięcią główną (RAM) a innymi rodzajami pamięci.

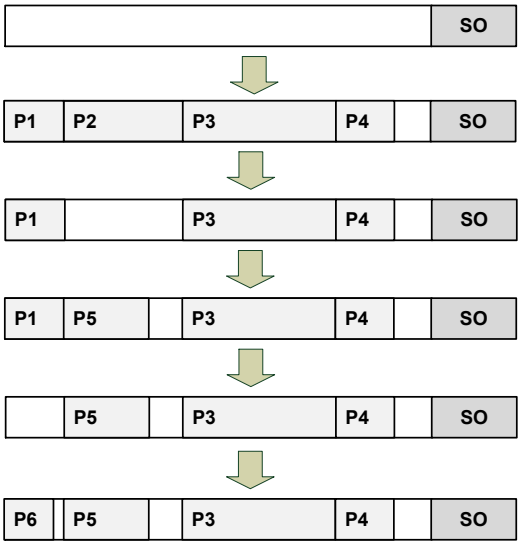
Jeśli w systemie operacyjnym ma koegzystować pewien (maksymalny) zbiór procesów, to należy się zastanowić, w jaki sposób dokonać dla niego przydziału pamięci. Najprostszym rozwiązaniem (stosowanym w pierwszych systemach operacyjnych) był podział pamięci na bloki o ustalonym z góry rozmiarze (partycje), przy założeniu, że rozmiar programu nie przekroczy rozmiaru partycji. Stosowano jednakowe rozmiary wszystkich partycji lub definiowano partycje o różnych wielkościach. Strategia przydziału pamięci polegała na przydzielaniu (lub zwalnianiu) całych partycji dla kolejnych programów. Taki rodzaj przydziału pamięci nosi nazwę *partycjonowania statycznego*. Na rysunku 4.12 pokazano rezultat przydziału pamięci dla dwu procesów, kiedy w systemie zastosowano partycjonowanie o jednakowych rozmiarach partycji. Niezależnie od rzeczywistych rozmiarów pamięci danych procesów nastąpiło przydzielenie im całego rozmiaru partycji. W związku z tym część pamięci operacyjnej została zmarnowana. Taki sposób straty pamięci nazywa się *fragmentacją wewnętrzną*. Istotną wadą takiego rozwiązania było także ograniczenie dopuszczalnego rozmiaru programu do rozmiaru największej partycji, zaletą zaś jego prostota i łatwość zarządzania. Współczesne systemy operacyjne, takie jak Android czy Windows, stosują zbliżone mechanizmy niepozwalające aplikacjom użytkownika na zajęcie więcej niż 32 MB pamięci operacyjnej.



Rys. 4.12. Przydział pamięci przy partycjach o równych rozmiarach

Próbą przezwyciężania wad partycjonowania statycznego było zaproponowanie *partycjonowania dynamicznego*. Rozmiar każdej partycji był ustalany w zależności od potrzeb danego procesu. Uzyskano możliwość elastycznego dopasowywania rozmiaru do potrzeb programów. Na rysunku 4.13 pokazano przebieg przydziału pamięci dla ustalonego zbioru procesów przy partycjonowaniu dynamicznym. W pierwszej fazie pracy system operacyjny przydzielił pamięć dla procesów P1, P2, P3 i P4 według zgłoszonych przez nie wymagań. Po zwolnieniu pamięci przez proces P2 w jego miejsce przydzielono pamięć „mniejszemu" procesowi P5. Z kolei obszar pamięci zwolniony przez proces P1 został częściowo przydzielony procesowi P6. Po kilkukrotnym zarezerwowaniu i zwolnieniu pamięci pojawiają się w niej tak zwane dziury, a pamięć z biegiem czasu może zostać znacząco sfragmentowana. Jest to tak zwana *zewnętrzna fragmentacja*. Redukcji fragmentacji dokonywano przez uruchamianie w systemie co pewien czas procesu *upakowania*, który przemieszczał bloki pamięci, „likwidując" powstałe dziury. Stosowanie partycjonowania dynamicznego okazało się kosztowne z punktu widzenia nakładów obliczeniowych i pamięciowych, które musiał ponosić system operacyjny na zarządzanie i utrzymanie informacji związanych z obsługą pamięci.

We współczesnych systemach operacyjnych stosuje się metodę zarządzania pamięcią, stanowiącą swego rodzaju „złoty środek" pomiędzy partycjonowaniem statycznym a dynamicznym – *stronicowanie*. W przypadku stronicowania pamięć operacyjna jest dzielona na niewielkie jednakowe bloki zwane *ramkami*. Proces także jest dzielony na bloki odpowiadające rozmiarom ramek zwane *stronami*. Jeśli dany proces ma być wykonywany, to jego strony są wprowadzane do dowolnych ramek pamięci operacyjnej. Na rysunku 4.14 pokazano przebieg przydziału pamięci dla ustalonego zbioru procesów przy stronicowaniu. W pierwszej fazie działania systemu pamięć przydzielono procesom P1, P2 i P3 zajmującym odpowiednio 3, 2 i 3 ramki. W miejsce pamięci po procesie P2 wprowadzono dwie strony procesu P4, strony pozostałe zaś w obszarze pamięci „za" obszarem przydzielonym procesowi

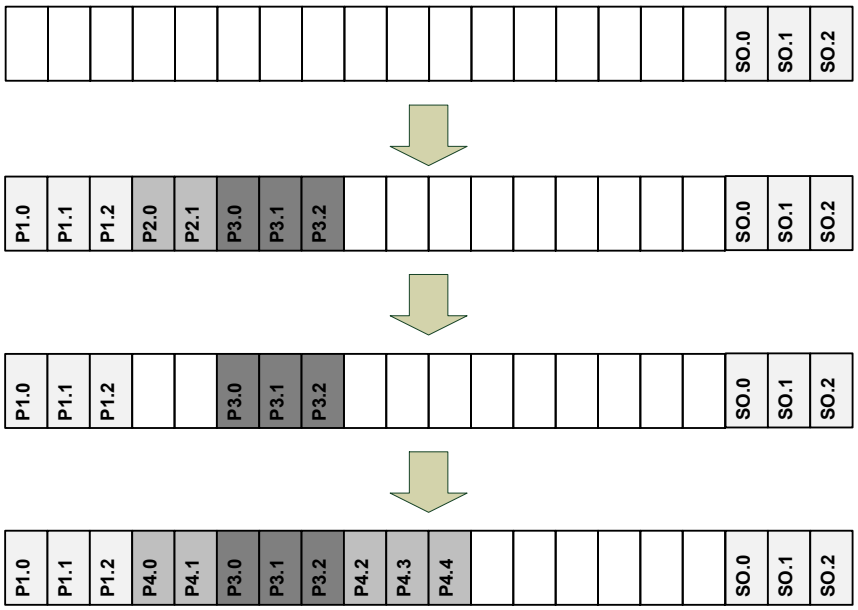


Rys. 4.13. Historia przydziału pamięci przy partycjonowaniu dynamicznym

P3. Stronicowanie umożliwia niemal taką samą elastyczność w dopasowywaniu rozmiaru pamięci operacyjnej do rozmiaru pamięci procesu jak dynamiczne partycjonowanie. Dodatkowo umożliwia rozlokowanie procesu w różnych fragmentach pamięci (por. rozlokowanie procesu P4 na rys. 4.14). Takie „rozdrobnienie" pamięci wymaga jednak odpowiednich struktur danych oraz zaawansowanych procedur do efektywnego zarządzania. Ponadto ostatnia strona procesu może być wewnętrznie sfragmentowana. W przypadku zastosowania stronicowania nie dochodzi do fragmentacji zewnętrznej.

Dysponując mechanizmem stronicowania, system obsługi pamięci można rozszerzyć o tak zwaną *pamięć wirtualną*. Część stron procesów można przechowywać w pamięci masowej (np. dysk twardy) i „sprowadzać" ja na żądanie do pamięci operacyjnej. W ten sposób system operacyjny oferuje programom przestrzeń adresową ograniczoną jedynie rozmiarem pamięci masowej. Możliwe jest wtedy również utrzymywanie w pamięci operacyjnej większej liczby programów, choć część stron ich pamięci będzie pozostawała w pamięci masowej.

Mechanizm pamięci wirtualnej na pierwszy rzut oka może spowodować zwolnienie pracy komputera, ponieważ do kontynuacji obliczeń przez procesy potrzeba sukcesywnie „sprowadzać" ich strony z pamięci masowej. Przeprowadzone analizy (por. [42, 43]) pokazały jednak, że całościowa efektywność systemu wzrasta. Okazuje się, że część stron programów jest bardzo rzadko sprowadzana do pamięci, bo



Rys. 4.14. Historia przydziału pamięci przy stronicowaniu

zawierają np. procedury obsługi wyjątków lub wyjątkowo używane procedury czy tablice zmiennych przewidziane przez programistów „na wyrost”.

Podsumowując, istotnym zadaniem systemu operacyjnego jest efektywne zarządzanie pamięcią operacyjną. Najchętniej stosowaną współczesną techniką przydziału pamięci jest stronicowanie, często uzupełnione pamięcią wirtualną. Techniki zarządzania pamięcią obejmują również ochronę obszarów pamięci, udostępnianie pamięci w celu współużytkowania oraz transformację adresów.

4.8. System plików

Na pewnym etapie rozwoju urządzeń mikroprocesorowych pojawia się znacząca presja na możliwość efektywnego przechowywania na nich nieulotnych informacji. W zależności od zastosowania mogą to być bufory przeznaczone do przechowywania danych pomiarowych, obszary pamięci służące do zapamiętywanie zdjęć, danych dźwiękowych czy dokumentów. Kolejnym naturalnym krokiem w rozwoju takich urządzeń jest wdrożenie systemu plików – mechanizmu, który pozwoli na efektywne zarządzanie informacją w systemie mikroprocesorowym. W urządzeniach pracujących pod nadzorem systemów operacyjnych system plików jest naturalnym składnikiem, chociaż w niektórych najnowszych komputerach

przenośnych jest on coraz głębiej ukrywany przed przeciętnym użytkownikiem. W typowym rozwiązaniu programy i dane są gromadzone w postaci tak zwanych plików w pamięci masowej (np. dysku twardym, dysku SSD) i sprowadzane do pamięci operacyjnej w celu uruchomienia bądź przetworzenia. Istotnym zadaniem systemu operacyjnego jest w tym wypadku dostarczenie logicznej reprezentacji danych, którą będzie rozumiał użytkownik, oraz powiązanie jej z reprezentacją fizyczną odpowiadającą technicznym możliwościom przechowywania informacji na urządzeniach elektronicznych czy elektromechanicznych. Warto zwrócić uwagę, że mechanizmy pamięci wirtualnej omawiane w podrozdziale 4.7 również opierają się na systemie plików.

Plik jest nazwanym zbiorem powiązanych ze sobą informacji zapisanym w pamięci pomocniczej (masowej) [42]. Jak już wspomniano, w plikach mogą znajdować się zarówno programy, jak i dane. Organizacja pliku zależy od jego twórcy i może to być ciąg bitów, bajtów czy wreszcie struktur danych.

System operacyjny powinien dostarczyć mechanizmów identyfikacji plików, czyli możliwość ich znalezienia. W dalszej kolejności od systemu operacyjnego oczekuje się zdolności do dostarczenia programom podstawowych operacji na plikach pozwalających na ich odczyt lub modyfikację. W systemach, z których korzysta wielu użytkowników, system plików musi być rozbudowany o moduł zapewnienia bezpieczeństwa w dostępie do współdzielonych plików.

Plik w systemie operacyjnym jest identyfikowany przez pewien zbiór atrybutów. Ich liczba i rodzaj zmienia się w zależności od twórcy sytemu. Do najpopularniejszych należą:

- *nazwa* – ciąg znaków jednoznacznie identyfikujący plik czytelny dla użytkownika,
- *typ* – informacja służąca do rozpoznawania zawartości pliku,
- *lokalizacja* – informacja wskazująca, gdzie w systemie komputerowym znajduje się plik,
- *wielkość* – informacja wskazująca na rozmiar pliku w ustalonych jednostkach,
- *ochrona* – informacja pozwalająca zidentyfikować, który użytkownik i na jakim poziomie może skorzystać z informacji zawartych w pliku,
- *czasy dostępu* – informacja o czasie i dacie wykonywania wybranych operacji na pliku.

Informacje o plikach są gromadzone w strukturze katalogowej zgromadzonej również w pamięci masowej.

Niezależnie od tego, na jakim urządzeniu fizycznym znajduje się plik, każdy system operacyjny powinien dostarczać podstawowy zestaw operacji, które można na takim zbiorze informacji wykonać:

- *tworzenie pliku* – znalezienie miejsca na utworzenie, a następnie dokonanie wpisu w katalogu plików z uwzględnieniem atrybutów pliku,
- *zapisywanie do pliku* – znalezienie pliku, a następnie wprowadzenie do niego ustalonej informacji we wskazanym miejscu,
- *odczytywanie z pliku* – znalezienie pliku, a następnie odczyt z niego ustalonej porcji informacji ze wskazanego miejsca,
- *zmiana pozycji w pliku* – zmiana miejsca w pliku, z którego będą odczytywane dane,
- *usuwanie pliku* – znalezienie wpisu o pliku w katalogu, usunięcie go oraz zwolnienie obszaru pamięci masowej, gdzie rezydował,
- *skracanie pliku* – usunięcie zawartości pliku, ale pozostawienie go w strukturze katalogowej.

Opierając się na wymienionych podstawowych operacjach na plikach, można opracować inne, np. *dopisywanie danych na końcu pliku* lub *zmiana nazwy pliku*. Niektóre nośniki plików (np. płyty CD/DVD tylko do odczytu) z uwagi na właściwości fizyczne przechowywania danych mogą reagować tylko na wybrane operacje plikowe.

Część plików z danymi jest zorganizowana w postaci sekwencji jednakowych bloków (np. rekordy bazy danych). Odczytywanie lub zapisywanie takich plików można zorganizować na dwa podstawowe sposoby. Pierwszy z nich, nazywany *dostępem sekwencyjnym*, zakłada, że w pliku jest ustalana pozycja zapisu lub odczytu, a następnie jest przetwarzany blok pamięci jeden po drugim. Drugi sposób, zwany *dostępem swobodnym*, pozwala na odczytywanie bloków pamięci w dowolnej kolejności. Odmianę techniki dostępu swobodnego stosuje się również do innych rodzajów plików. Pliki takie zawierają zwykle pewną pomocniczą strukturę danych (indeks) informującą, gdzie w danym pliku można pozyskać lub zapisać dane określonego rodzaju. Odczyt lub zapis do takiego pliku rozpoczyna się od przeglądnięcia indeksu, aby znaleźć poszukiwaną pozycję w pliku. Następnie w sposób swobodny dokonuje się „przeskoku" do znalezionej pozycji.

W dotychczasowych rozważaniach kilkukrotnie wspomniano o strukturze katalogowej plików. *Katalog plików* jest jedną z podstawowych struktur danych systemu operacyjnego i zawiera informacje o atrybutach, lokalizacji i właścicielu

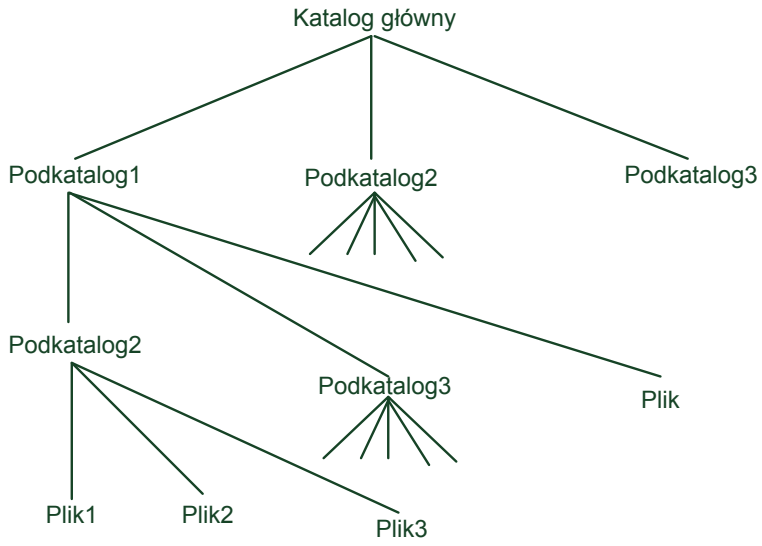
plików. Z punktu widzenia użytkownika katalog zawiera odwzorowanie pomiędzy nazwami plików znanymi przez użytkowników i aplikacje a samymi plikami. Podstawowymi mechanizmami, jakie powinny dostarczać usługi katalogowe, są:

- *odnajdywanie pliku* – możliwość przeszukania zawartości katalogu w celu uzyskania informacji, czy i gdzie plik o danej nazwie lub nazwie zawierającej pewien ciąg liter jest obecny,
- *tworzenie pliku* – możliwość utworzenia pliku, nadania mu nazwy oraz wpisu tej nazwy do struktury katalogowej,
- *usuwanie pliku* – możliwość usunięcia wpisu o pliku ze struktury katalogowej,
- *wypisanie zawartości katalogu* – możliwość uzyskania wykazu plików znajdujących się w katalogu,
- *zamiana nazwy pliku* – możliwość zmiany atrybutów pliku,
- *przegląd systemu plików* – możliwość przeszukania wszystkich katalogów i plików.

Najczęściej stosowaną strukturą katalogu plików jest drzewo (por. rys. 4.15). Każda struktura katalogowa rozpoczyna się od jednego głównego katalogu, a następnie rozgałęzia się na kolejne podkatalogi, które z kolei mogą zawierać inne podkatalogi i pliki. W danym katalogu nie powinny się znajdować dwa pliki lub katalogi o takiej samej nazwie. Pliki lub katalogi o takich samych nazwach mogą się znajdować natomiast w różnych katalogach. Dostęp do danego pliku czy katalogu uzyskuje się poprzez podanie „ścieżki" prowadzącej od katalogu głównego poprzez kolejne podkatalogi, aż do katalogu, w którym znajduje się dany plik lub katalog.

Ponadto w komputerach, z których korzysta wielu użytkowników, system operacyjny musi zapewnić ochronę danych, czyli udostępnić mechanizmy pozwalające na wskazanie, który użytkownik i na jakim poziomie może korzystać z informacji zawartych w danych plikach i katalogach. Z plikami i katalogami są powiązane ograniczenia dostępu. Najpopularniejsze z nich to:

- *czytanie* – możliwość odczytu zawartości pliku,
- *pisanie* – możliwość zapisywania danych do pliku,
- *wykonywanie* – możliwość uruchamiania pliku jako programu lub skryptu,
- *dopisywanie* – możliwość dopisywania danych do końca pliku,



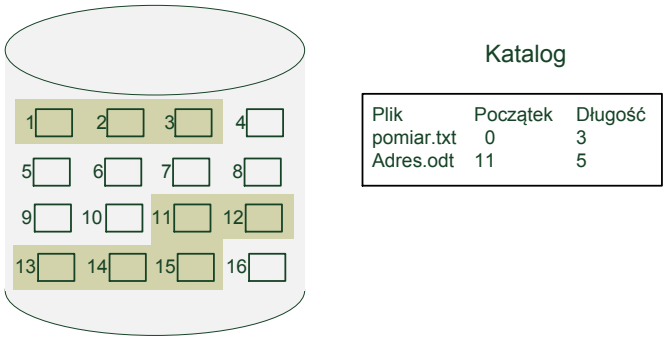
Rys. 4.15. Katalog ze strukturą drzewa

- *usuwanie* – możliwość usuwania pliku z drzewa katalogów,
- *opisywanie* – możliwość odczytu nazwy i atrybutów pliku.

Ograniczenia dostępu przypisuje się zwykle na trzech poziomach: *właściciela pliku*, *grupy* i *innych użytkowników*. Właściciel posiada zwykle wszystkie prawa do swoich plików. Może je więc tworzyć, usuwać, modyfikować i uruchamiać. Może on również przydzielić prawa do plików innym użytkownikom należącym do zdefiniowanych w systemie operacyjnym grup. Wtedy każdy członek grupy posiada takie same ograniczenia dostępu do wskazanych plików. Wybrane uprawnienia właściciel może przydzielić również wszystkim użytkownikom systemu.

Jeśli jeden plik jest dostępny dla wielu użytkowników, to system operacyjny musi przyjąć jakąś strategię jego współużytkowania. Najprostszym sposobem jest zablokowanie dostępu do pliku innym użytkownikom, gdy jest on już otwarty przez jednego użytkownika. Istnieją systemy operacyjne pozwalające na współużytkowanie plików i ich modyfikację przez wielu użytkowników w jednym czasie, przy czym arbitralnie rozstrzygają wtedy efekty modyfikacji plików. Modyfikacje pliku są zapisywane w pamięci podręcznej, a przy zamykaniu sesji z plikiem przez ostatniego użytkownika następuje sekwencyjne wprowadzenie kolejnych modyfikacji.

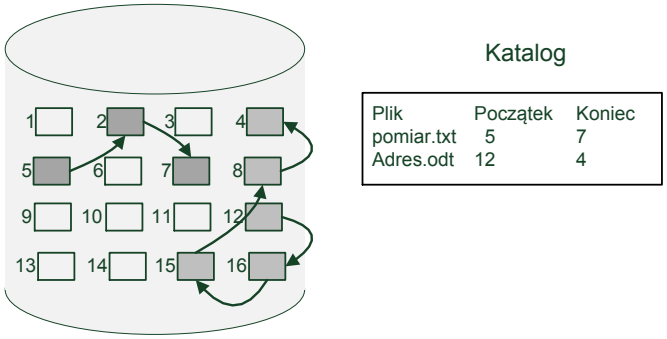
Logiczna struktura plików i katalogów jest ostatecznie odwzorowywana w pamięci masowej. Jednostką pamięci masowej jest zwykle blok danych, a system operacyjny przenosi zawartość pliku w postaci zbioru bloków danych. Zarządzanie blokami danych w pamięci masowej jest podobne do technik zarządzania pamięcią operacyjną (por. podrozdział 4.7). Zawartość plików gromadzi się w blokach pamięci masowej zwykle na trzy podstawowe sposoby. Pierwszy z nich polega na *ciągłym przydziale pamięci*. Oznacza to, że w katalogu wskazuje się pierwszy blok pamięci masowej, do którego będą wprowadzane dane z pliku oraz liczby bloków, jakie plik zajmuje. Przykład takiego przydziału pokazano na rys. 4.16. Przydział ciągły jest korzystny z punktu widzenia szybkości odczytu danych. Stwarza jednak problemy, gdy plik zmienia rozmiar. Jeśli kolejne bloki pamięci masowej nie są wolne, to dla pliku należy znaleźć inny wolny obszar.



Rys. 4.16. Ciągły przydział bloków danych w pamięci masowej

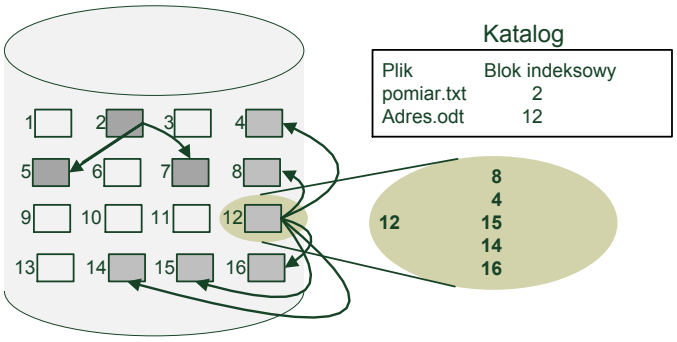
Drugim sposobem porządkowania danych w pamięci masowej jest *przydział listowy* (por. rys. 4.17). Bloki danych tworzą listę i mogą się znajdować gdziekolwiek na dysku. W katalogu w pamięci masowej wskazuje się pierwszy blok, do którego będą wprowadzane dane z pliku, oraz ostatni blok. Eliminuje to kłopoty ze znalezieniem miejsca dla pliku, który zmienił swój rozmiar, powoduje jednak wydłużony odczyt pliku. Aby odczytać wybrane dane z pliku, trzeba po kolei odczytać wszystkie bloki. Polepszenie pracy tak zorganizowanej struktury zapisywania danych na dysku polega na umieszczaniu danych w sąsiednich blokach, jeśli to możliwe, tworząc tak zwane grona lub klastry.

Trzecim sposobem zarządzania jednostkami pamięci masowej jest *przydział indeksowy* (por. rys. 4.18). Jeden z bloków w pamięci masowej zawiera spis (indeks) adresów wszystkich bloków, do których zostały zapisane kolejne fragmenty pliku. Katalog zawiera jedynie informacje, gdzie znajduje się blok pamięci masowej zawierającej indeks do pozostałych fragmentów pliku. Takie rozwiązanie pozwala



Rys. 4.17. Listowy przydział bloków danych w pamięci masowej

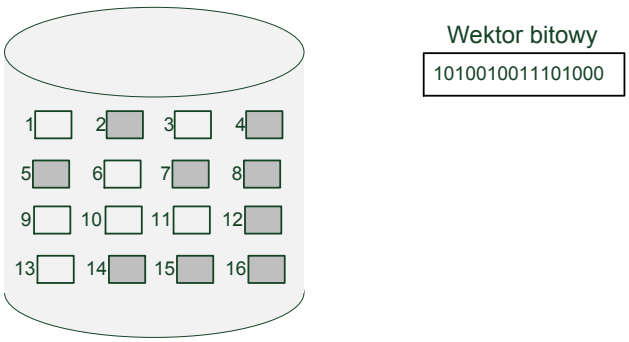
na szybkie wczytywanie wybranych bloków pamięci masowej, choć odbywa się to kosztem zwiększenia struktur danych zgromadzonych w pamięci masowej.



Rys. 4.18. Indeksowy przydział bloków danych w pamięci masowej

System operacyjny zarządzający dostępem do pamięci masowej musi również przechowywać informacje, które z bloków pamięci masowej są wolne, a które zajęte. Jednym ze sposobów przechowywania takiej informacji jest *wektor bitowy*. Kolejne bloki pamięci masowej są odwzorowywane w postaci stanu jednego bitu w ustalonym, odpowiednio długim słowie bitowym. Jeśli wartość bitu w słowie wynosi 1, to oznacza, że dany blok jest wolny. Rysunek 4.19 stanowi interpretację wektora bitowego dla wybranej pamięci masowej.

Podsumowując, podsystem zarządzania plikami stanowi istotny komponent systemu operacyjnego. Jego podstawowym zadaniem jest umożliwienie reprezen-



Rys. 4.19. Interpretacja wektora bitowego

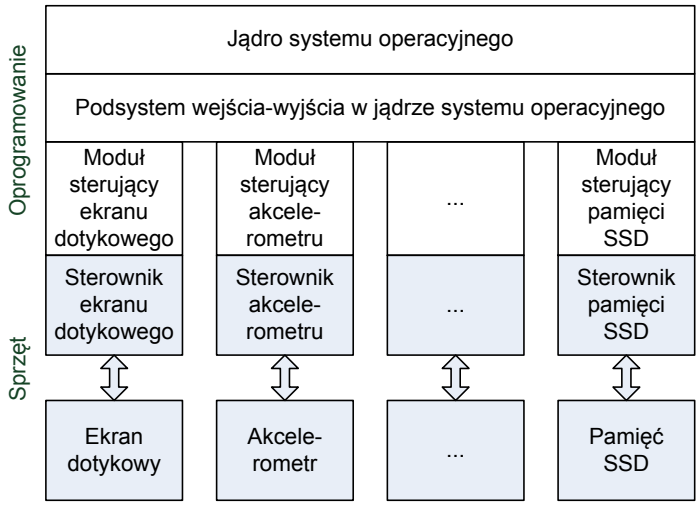
tacji stałej informacji przechowywanej na komputerze w postaci zbioru jednostek logicznych zwanych plikami. Pliki mogą zawierać zarówno dane, jak i programy. Są reprezentowane zwykle w strukturze katalogowej pozwalającej na przejrzyste odczytywanie plików przez użytkownika i programy. W systemach dla wielu użytkowników można zdefiniować ograniczenia w dostępie do prywatnych plików użytkowników czy systemu operacyjnego.

4.9. Obsługa urządzeń wejścia-wyjścia

Każdy system komputerowy dysponuje zbiorem urządzeń wejścia-wyjścia. Typowe komputery osobiste komunikują się z użytkownikiem z zastosowaniem urządzeń wskazujących (mysz, ekran dotykowy, touchpad), klawiatur, monitorów, kamer, mikrofonów, głośników czy czujników przyspieszenia. Zawierają także pamięci masowe (dyski twarde, dyski SSD, czytniki i nagrywarki CD/ DVD/ Blue Ray) oraz urządzenia komunikacji sieciowej, które z punktu widzenia systemu mikroprocesorowego są urządzeniami wejścia-wyjścia. Wreszcie architektura części komputerów osobistych pozwala na przyłączanie do nich kart rozszerzeń oraz urządzeń komunikujących się przez standardowe porty komunikacyjne. Komputery realizujące z kolei aplikacje wbudowane stosują do komunikacji ze swoim otoczeniem przetworniki sygnałów analogowo-cyfrowych oraz cyfrowo-analogowych, a także zestawy komunikacyjne oparte na interpretacji sygnałów binarnych.

Istotnym zadaniem sytemu operacyjnego jest możliwość efektywnego zintegrowania wymienionych urządzeń z systemem mikroprocesorowym i dostarczenie programistom w miarę jednolitego mechanizmu dostępu. Taką integrację ułatwiają wypracowane na przestrzeni lat techniki przyłączania urządzeń zewnętrznych. Zwykle „pomiędzy” systemem mikroprocesorowym a danym urządzeniem wejścia-wyjścia

znajduje się dodatkowy moduł elektroniczny zwany *sterownikiem urządzenia*. Sterownik jest w stanie zinterpretować sygnały elektryczne wysyłane do niego przez magistralę procesora oraz doprowadzić do wymiany informacji (przesłania odpowiedniej porcji danych) pomiędzy urządzeniem a systemem mikroprocesorowym. Po stronie systemu operacyjnego ze sterownikiem urządzenia współpracuje *moduł sterujący* – program przeznaczony do komunikacji z określonym rodzajem urządzeń wejścia-wyjścia. Komunikuje się on z kolei z *podsystemem wejścia-wyjścia w jądrze sytemu operacyjnego*, gdzie zróżnicowane techniki wymiany danych z urządzeniami zewnętrznymi są sprowadzane do ujednoliconego interfejsu: operacji otwarcia, zamknięcia, przeszukiwania, zapisu i odczytu. Na rysunku 4.20 pokazano część struktury sprzętowo-programowej podsystemu wejścia-wyjścia typowego urządzenia przenośnego.



Rys. 4.20. Sprzęt i oprogramowanie urządzeń wejścia-wyjścia w systemie operacyjnym

Sterowniki urządzeń wejścia-wyjścia są integrowane z systemem mikroprocesorowym na dwa podstawowe sposoby. Po pierwsze, mogą one być podłączone do magistral realizujących specjalne rozkazy wejścia-wyjścia mikroprocesora. Procesor, wykonując te specjalne rozkazy, zapisuje lub odczytuje wybrane pola bitowe w sterowniku urządzeń i w ten sposób pozyskuje lub przesyła dane. Po drugie, istnieje możliwość odwzorowywania komunikacji z urządzeniem zewnętrznym przez odwoływanie się do obszarów pamięci operacyjnej. Tutaj urządzenie zewnętrzne „jest widziane” jako zbiór komórek pamięci operacyjnej, z którymi można wymieniać dane.

Moduły sterujące komunikują się zwykle ze sterownikami urządzeń z zastosowaniem trzech mechanizmów. Pierwszy sposób komunikacji polega na ciągłej interakcji ze sterownikiem urządzenia. Jest on nazywany *odpytywaniem*. Moduł sterujący ustawia odpowiednie bity w słowie sterującym sterownika i cyklicznie sprawdza stan innych słów informujących o zmianach, jakie zaszły w urządzeniu wejścia-wyjścia. Jeśli otrzymuje potwierdzenie, że urządzenie wejścia-wyjścia jest gotowe do odbioru lub wysłania danych, następuje ich przesłanie. Ten sposób komunikacji jest czasochłonny z punktu widzenia systemu operacyjnego, bo musi co pewien czas dokonywać detekcji stanu urządzenia. Z drugiej strony konstrukcja oprogramowania do takiej obsługi jest prosta. Drugi sposób komunikacji polega na *zintegrowaniu urządzenia wejścia-wyjścia z systemem obsługi przerwań*. Moduł sterujący wysyła wtedy do urządzenia wejścia-wyjścia komunikat o chęci odczytu lub wysłania porcji danych. Sterownik urządzenia, kiedy jest gotowy na wymianę danych, zgłasza przerwanie. Odpowiednia procedura obsługi przerwania nadzoruje lub inicjuje proces przenoszenia danych pomiędzy urządzeniem a systemem operacyjnym. W takim rozwiązaniu system jest odciążony od odpytywania. Obsługa przerwań powoduje bardziej złożony kod systemu operacyjnego. Trzecim sposobem jest *włączenie urządzeń wejścia-wyjścia do współpracy z podsystemem DMA* (ang. *Direct Memory Access*) systemu mikroprocesorowego. Podsystem ten umożliwia kopiowanie danych z urządzeń wejścia-wyjścia bez udziału głównego procesora. DMA jest programowany na autonomiczne przesłanie odpowiedniej ilości danych z urządzenia do wskazanego obszaru pamięci. Po zakończeniu przesyłu danych system jest powiadamiany o zakończeniu operacji wejścia-wyjścia. Zastosowanie podsystemu DMA zwiększa przepustowość systemu operacyjnego. Wymaga jednak odpowiednio zaprojektowanych urządzeń wejścia-wyjścia.

Warto zwrócić uwagę, że urządzenia wejścia-wyjścia cechują się bardzo dużą różnorodnością pod względem szybkości komunikacji i sposobu przesyłania danych. Najszybciej są przesyłane sygnały z urządzeń pamięci masowej (twarde dyski i dyski SSD), najwolniejszych zdarzeń można się zaś spodziewać od urządzeń przesyłających sygnały od użytkownika – znaki klawiatury, zmiana pozycji i kliknięcia myszki. Pomiędzy nimi plasują się karty sieciowe, kamery, drukarki i inne. Część urządzeń przesyła dane sekwencyjnie (np. strumień danych z klawiatury), część z kolei może przesyłać porcje danych w blokach niekoniecznie uporządkowanych (np. dane z pamięci masowej). Z punktu widzenia efektywności działania systemu operacyjnego ważne jest takie dostosowanie metody dostępu do danego urządzenia, aby wymagała najmniejszego nakładu pracy. Okazuje się, że urządzenia generujące najrzadsze sygnały warto obsługiwać z zastosowaniem podsystemu przerwań. Z kolei odpytywanie urządzeń generujących odpowiedzi bardzo szybko może się okazać mniej obciążające z uwagi na brak przełączania procesów.

Kolejnym istotnym zagadnieniem związanym z efektywną wymianą danych pomiędzy urządzeniami wejścia-wyjścia a systemem operacyjnym jest buforowanie, czyli gromadzenie partii danych w odpowiednio zaplanowanych blokach pamięci. Dane buforuje się wtedy, gdy w komunikacji uczestniczą urządzenia o różnej prędkości (np. program generujący wydruki i drukarka). Dane wygenerowane szybko przez jeden składnik systemu można zgromadzić w pamięci, a następnie z odpowiednią prędkością przekazać składnikowi wolniejszemu. Buforowanie danych ma również istotne znaczenie, gdy informacje należy „złożyć" z mniejszych partii. Bufor służy wtedy za magazyn, w którym następuje zebranie porcji informacji nadchodzących w sposób niekoniecznie sekwencyjny, a następnie przesłanie dalej zebranej informacji. W taki sposób następuje na przykład kolekcjonowanie pakietów danych przychodzących przez sieć komputerową. W niektórych przypadkach korzystne jest stosowanie tak zwanego podwójnego buforowania. Komunikacja taka polega na zapisywaniu danych do jednego bufora, podczas gdy zawartość drugiego bufora jest już odczytywana. Po zakończeniu zapisu do właśnie zapełnianego bufora może nastąpić przełączenie pomiędzy buforami. Ostatnio zapisywany jest przeznaczony do odczytu, a poprzednio odczytywany – do zapisu. Na takiej zasadzie są wyświetlane kolejne klatki animacji na ekranie komputera bez zauważalnego migotania.

Podsumowując, kolejnym ważnym zadaniem systemu operacyjnego jest dostarczenie mechanizmów komunikacji programistów i użytkowników z urządzeniami wejścia-wyjścia. Komponentami systemu odpowiedzialnymi za komunikację z takimi urządzeniami są moduły sterujące, które interpretują sygnały wymieniane pomiędzy systemem mikroprocesorowym a sterownikami urządzeń. Urządzenia wejścia-wyjścia znacząco się różnią zarówno szybkością generowania sygnałów, jak i naturą przesyłanych danych. System operacyjny ostatecznie przekształca różne metody komunikacji z urządzeniami zewnętrznymi na standardowy zbiór funkcji otwierania, zamykania, przesunięcia, odczytu i zapisu.

4.10. Elementy zabezpieczeń w systemach operacyjnych

4.10.1. Logowanie użytkowników

W systemach operacyjnych często występuje konieczność implementacji mechanizmu logowania użytkowników. Taki proces potwierdzenia tożsamości jednej bądź obu ze stron nazywamy *uwierzytelnianiem*. W przypadku osób uwierzytelnianie może następować na podstawie różnych cech, np. biometrycznych (kontrola odcisku palca, skanowanie siatkówki oka), ale najczęściej jest spotykana weryfikacja hasła (klucza, PIN-u). Z pewnością Czytelnik wielokrotnie spotkał się z takim uwierzytelnieniem, np. logując się do komputera, systemu bankowego czy wprowadzając PIN w trakcie płatności kartą płatniczą. Warto poznać sposób

działania uwierzytelnienia opartego na haśle, choćby po to by móc ocenić, czy administrator systemu prawidłowo go zaimplementował, minimalizując ryzyko wycieku haseł czy też podszycia się pod inną osobę.

Implementując uwierzytelnianie na podstawie loginu i hasła, najprostszym rozwiązaniem wydaje się przechowywanie hasła użytkownika w systemie w jawnej postaci. Rozwiązanie takie nie zapewnia minimalnego poziomu bezpieczeństwa i nie powinno być w ogóle stosowane. W przypadku wycieku pliku czy też bazy danych z hasłami są ujawnione hasła wszystkich użytkowników danego systemu. Wielu użytkowników wykorzystuje identyczne hasła w różnych miejscach, co w przypadku takiego wycieku jest poważnym zagrożeniem. Niestety taki sposób przechowywania hasła jest nadal często spotykany. Taki system można rozpoznać na pierwszy rzut oka, jeżeli w razie utraty hasła przez użytkownika istnieje możliwość jego „przypomnienia", a nie wygenerowania nowego. Należy podkreślić , że ostatnio modne tzw. *hasło maskowane*, tj. uwierzytelnienie jedynie na podstawie kilku losowo wybranych znaków z hasła, a nie całości, wymaga przechowywania hasła w postaci jawnej! Taki sposób logowania często jest wprowadzany na żądanie nieświadomych użytkowników, którym wydaje się, że zwiększa to bezpieczeństwo, gdyż w przypadku logowania z komputera zainfekowanego złośliwym oprogramowaniem ujawniają jedynie fragment hasła. W praktyce zabezpieczenie takie jest pozorne, gdyż wystarczy kilkukrotne logowanie, by potencjalny trojan poznał większość hasła, z dużym prawdopodobieństwem wystarczającą do podszycia się pod użytkownika.

Lepszym, choć nadal niewystarczającym rozwiązaniem jest przechowywanie zamiast hasła użytkownika jego skrótu (tzw. *hash* hasła), tj. wyniku działania na niego kryptograficznie silną funkcją skrótu. W trakcie uwierzytelniania obliczany jest skrót hasła podanego przez użytkownika i porównywany z prawidłowym skrótem zapisanym w bazie. Niestety rozwiązanie takie ma istotne wady. Identyczne hasła generują identyczne skróty, więc w razie wycieku taka baza skrótów podatna jest na *atak słownikowy*, tj. porównanie skrótów zgromadzonych w bazie ze skrótami obliczonymi dla słownika zawierającego zbiór często stosowanych haseł. Należy zauważyć, że skróty takie mogą być obliczone wcześniej, co znacznie przyspiesza atak. Co więcej, są dostępne gotowe bazy takich skrótów popularnych haseł, przygotowane w formie tablic (*lookup table*) bądź w celu oszczędności miejsca jako tabele tęczowe (*rainbow table*). Dokładny opis tych mechanizmów wykracza jednak poza zakres niniejszej pracy. Innym atakiem możliwym do zastosowania jest *atak brute-force*, polegający na obliczaniu skrótów dla kolejno wygenerowanych haseł. Ataki te pozwalają na odtworzenie słabych haseł, tj. występujących w słowniku, bądź krótkich, niezawierających kombinacji małych i dużych liter, cyfr i znaków specjalnych.

Aby zmniejszyć podatność na wzmiankowany atak słownikowy stosuje się prostą modyfikację. Dla każdego użytkownika jest losowana liczba, tzw. *salt* zapisywany jawnie w bazie. Zamiast skrótu hasła jest przechowywany skrót obliczany z połączenia hasła z wygenerowaną liczbą. Powoduje to, że skróty jednakowych haseł dla różnych wartości *salt* są różne. Atak słownikowy wymagałyby więc przygotowania słownika skrótu każdego hasła dla każdej możliwej wartości *salt*, co z oczywistych względów jest niepraktyczne.

Warto zauważyć, że opisana poprzednio modyfikacja nieco zmniejsza podatność na atak *brute-force*. Wartość *salt* jest wprawdzie jawnie podana w bazie, może więc być wykorzystana przez intruza podczas generowania skrótu, ale intruz w tym przypadku musi osobno łamać każde z haseł, a nie zbiorczo wszystkie występujące w bazie. Niemniej jednak, jeśli celem intruza podczas ataku *brute-force* jest odtworzenie hasła konkretnego użytkownika, a nie złamanie jak największej liczby haseł dowolnych użytkowników, wymagany nakład obliczeniowy w obu przypadkach jest podobny. Aby zminimalizować prawdopodobieństwo pomyślnego ataku zazwyczaj stosuje się technikę zwaną *key stretching*, polegającą na wielokrotnym obliczaniu skrótu. W każdym kroku jest obliczany skrót z hasła połączonego z poprzednim skrótem. Proces ten jest powtarzany tysiące bądź nawet setki tysięcy razy. Po stronie systemu jest zapisywany finalny skrót po określonej liczbie iteracji. Rozwiązanie takie wprawdzie zwiększa czas potrzebny na weryfikację hasła wprowadzonego przez użytkownika, ale w praktyce opóźnienie to jest niezauważalne. Zaletą takiego podejścia jest znaczne zwiększenie nakładu obliczeń (a więc i czasu) potrzebnego na udany atak. Rozwiązanie takie jest rekomendowane do stosowania w praktyce (PBKDF2 [50]).

4.10.2. Hierarchia uprawnień

Logowanie użytkowników do systemu ma niewielki sens, jeśli po zalogowaniu wszyscy mają takie same uprawnienia. W systemach uniksowych (Linux, Unix) niemal wszystkie elementy systemu są reprezentowane w postaci plików. Niektóre z tych plików, tzw. pliki specjalne, reprezentują różnego rodzaju urządzenia, np. drukarki, dyski. Dlatego w tych systemach nadawanie uprawnień użytkownikom do różnych elementów systemu sprowadza się do nadawania im uprawnień do plików i katalogów.

Każdy z użytkowników systemów uniksowych posiada swoją nazwę (tzw. *login*). Poza tym każdy posiada także unikalny numer użytkownika, tzw. UID (ang. *User ID*), na podstawie którego jest identyfikowany w systemie. Narzędzia systemowe, wyświetlając informacje, często zamieniają numer UID na login. Szczególnym użytkownikiem systemów uniksowych jest użytkownik tradycyjnie nazywany *root*. UID tego użytkownika ma wartość równą zeru. Posiada on uprawnienia administratora całego systemu.

Aby usprawnić zarządzanie uprawnieniami, użytkownicy systemów uniksowych są łączeni w grupy. Jeden użytkownik może należeć do wielu grup, a grupy składają się z wielu użytkowników. Grupy, podobnie jak użytkownicy, mają swoje nazwy, a w systemie są reprezentowane za pomocą numerów nazywanych GID (ang. *Group ID*). Każdy użytkownik należy co najmniej do jednej grupy.

Każdy z plików na dysku w systemie uniksowym ma indywidualnego właściciela oraz grupę użytkowników, do której należy. Użytkownikowi można nadać trzy rodzaje uprawnień do plików:

- odczytu (oznaczane literą `r`),
- zapisu (oznaczane literą `w`),
- wykonywania (oznaczane literą `x`).

Uprawnienia do odczytu pozwalają odczytać zawartość pliku. Uprawnienia do zapisu pozwalają zapisać dane do pliku – zmienić jego zawartość. Uprawnienia do wykonywania umożliwiają uruchomienie pliku, co jest możliwe i ma sens tylko wtedy, gdy jest to plik, w którym zapisano program.

Nadawane uprawnienia mogą dotyczyć właściciela, całej grupy użytkowników, do której należy plik, bądź wszystkich innych użytkowników systemu. Do pliku jego właścicielowi można więc nadać wszystkie trzy uprawnienia (`rwx`), grupie tylko uprawnienia do odczytu i wykonania (`r-x`), a wszystkim pozostałym użytkownikom systemu nie nadawać żadnych uprawnień (`---`).

Taki sam zestaw uprawnień można nadać katalogom na dysku, ale ich znaczenie będzie nieco inne. Prawo odczytu katalogu pozwala sprawdzić, jakie pliki się w nim znajdują. Prawo zapisu pozwala tworzyć pliki w katalogu, prawo wykonania zaś na „wejście do katalogu", a więc uczynienie go swoim katalogiem bieżącym (co nie oznacza prawa do sprawdzenia, jakie plik znajdują się w tym katalogu).

Jeśli plik, w którym zapisano program, ma nadane uprawnienie do wykonywania dla wszystkich użytkowników systemu, każdy zalogowany użytkownik może ten program uruchomić. Uruchomiony program będzie miał w systemie dokładnie takie same uprawnienia, jak uruchamiający go użytkownik, niezależnie od tego, kto jest właścicielem pliku, w którym zapisano program.

Zmianę tego zachowania i dodatkowe możliwości sterowania uprawnieniami dają atrybuty SUID (czasami określany jako SetUID – ang. *Set User ID*) i SGID (czasami określany jako SetGID – ang. *Set Group ID*). Jeśli plik zawierający program poza uprawnieniami do wykonywania ma ustawiony także atrybut SUID, to po uruchomieniu zawsze będzie się wykonywał z uprawnieniami swojego właściciela (użytkownika, który jest właścicielem pliku programu), a nie z uprawnieniami użytkownika, który go uruchomił. Jeśli więc jakiś program należy do użytkownika *root* i ma ustawiony atrybut SUID, to niezależnie od tego, kto ten program uruchomi, program będzie miał pełne uprawnienia administratora systemu. Atrybut

SGID pozwala uruchamiać programy z uprawnieniami grupy, do której należą. Mechanizm ten pozwala delegować zwykłym użytkownikom pewną ściśle określoną część uprawnień administratora systemu – mały program wykonujący ściśle określone zadanie może wymagać do tego uprawnień administratora. Każdy kto ma prawo go uruchomić skorzysta z tych uprawnień, ale tylko w takim zakresie, w jakim pozwala na to program. Dla katalogów znaczenie ma ustawienie atrybutu SGID – powoduje ono, że właścicielem grupowym wszystkich plików tworzonych wewnątrz tego katalogu będzie grupowy właściciel katalogu.

Uniksowy system uprawnień oparty na grupach i nadawaniu uprawnień dla właściciela pliku, grupy i innych był często krytykowany jako zbyt prosty i niepozwalający na konfigurację dowolnie złożonej polityki bezpieczeństwa. Dla odmiany powszechny kiedyś system sieciowy firmy Novell miał znacznie bardziej skomplikowany system definiowania uprawnień. Narzędziem, które w systemach uniksowych od dawna pozwala jeszcze precyzyjniej zarządzać uprawnieniami do wykonywania poszczególnych zadań, jest program `sudo`. Pozwala on jednym użytkownikom wykonywać programy z uprawnieniami innych użytkowników, przy czym jest możliwe bardzo dokładne skonfigurowanie, jacy użytkownicy do jakich poleceń mają uprawnienia i jakie parametry mogą im podawać. Za pomocą tych narzędzi rozwiązuje się wszystkie praktyczne problemy z uprawnieniami w systemach uniksowych.

4.10.3. Szyfrowanie plików

Często występuje konieczność ochrony poufnych danych przed osobami niepowołanymi. Najprostszy poziom ochrony na podstawie uprawnień do pliku lub katalogu (szerzej opisany w podrozdziale 4.10.2) zazwyczaj jest niewystarczający, gdyż dane są przechowywane w postaci jawnej, dzięki czemu intruz, który uzyska prawa administratora lub fizyczny dostęp do urządzenia, może je odczytać. Konieczne jest szyfrowanie zapewniające poufność informacji w sensie wyjaśnionym w rozdziale 2. W wielu współczesnych systemach operacyjnych istnieje możliwość szyfrowania poszczególnych plików, katalogów bądź dysków. Mechanizmy takie mogą być wbudowane bezpośrednio w system operacyjny czy urządzenie fizyczne (np. dysk czy pendrive) lub dostarczane przez zewnętrzną aplikację.

Szczegółowe omówienie dostępnych narzędzi znacznie wykracza poza ramy niniejszej pracy. Jednak podczas dokonywania wyboru należy zwrócić uwagę na wybrane kluczowe aspekty. Dostępne są zarówno rozwiązania komercyjne o zamkniętym kodzie źródłowym, jak też darmowe otwarte. Należy podkreślić, że w zastosowaniach kryptograficznych rozwiązania charakteryzujące się w pełni jawnym kodem źródłowym są szczególnie polecane jako całkowicie zgodne z zasadą Kerckhoffsa, wzmiankowaną w podrozdziale 2.3. Rozwiązania zamknięte są bardziej narażone na występowanie luk bądź nawet celowo umieszczonych „tylnych

drzwi", pozwalających na dostęp do chronionej informacji osobom niepowołanym. W przypadku wolnego oprogramowania nadzorowanego przez społeczność ryzyko takie jest niewielkie.

Warto zauważyć, że współczesne rozwiązania kryptograficzne są na tyle silne, że najsłabszym ogniwem systemu pozostaje człowiek. Przestępcy, aby uzyskać dostęp do chronionych danych, mogą posunąć się do przemocy fizycznej, zmuszając użytkownika do podania hasła, co często będzie łatwiejsze niż inne metody łamania zabezpieczeń (tzw. *rubber-hose cryptanalysis*)[1]. Aby zminimalizować prawdopodobieństwo sukcesu takiego ataku, niektóre z rozwiązań oferują mechanizmy zapewniające możliwość *wiarygodnego zaprzeczenia* (ang. *plausible deniability*), że zaszyfrowane dane (plik, dysk) w ogóle istnieją. Kryptogram charakteryzuje się znaczną entropią (ciąg bajtów niosący zaszyfrowany przekaz przypomina ciąg losowy), co jest łatwo wykrywalne i nie pozwala użytkownikowi na zaprzeczenie ich istnienia. Dostępne są jednak narzędzia pozwalające ukryć długi łańcuch o znacznej entropii w miejscu, w którym prawdopodobne jest jego naturalne występowanie, np. w wolnej przestrzeni innego zaszyfrowanego dysku.

4.11. Podsumowanie

Kształcenie inżynierów na kierunkach nieinformatycznych obejmuje zwykle znaczące poszerzenie umiejętności obsługi systemów operacyjnych, a nawet programowania komputerów. Rozdział 4. miał na celu uzupełnienie tych umiejętności o znajomość kluczowych technik i mechanizmów działania systemów operacyjnych. Wiedza ta – jak się wydaje – pozwoli na bardziej świadome użytkowanie sprzętu komputerowego oraz zrozumienie pojęć, którymi posługują się inżynierowie systemów informatycznych.

Z uwagi na ograniczony rozmiar opracowania zdecydowano się na omówienie wybranych, reprezentatywnych mechanizmów zarządzania procesami, pamięcią, systemem plików czy urządzeniami wejścia-wyjścia. Nie dokonywano również szczegółowych analiz technik zarządzania zaimplementowanych we współczesnych systemach operacyjnych.

Śledząc historię rozwoju rozwiązań informatycznych, można zauważyć, że pojawianie się nowych urządzeń konstruowanych według technologii mikroprocesorowych powoduje po pewnym czasie tworzenie specjalizowanych systemów operacyjnych. Doświadczenia zdobyte podczas tworzenia pierwszych systemów operacyjnych przeznaczonych dla dużych komputerów zajmujących się przede wszystkim przetwarzaniem wsadowym zostały później twórczo rozwinięte w systemach operacyjnych dla mikrokomputerów, które nie były przeznaczone tylko do wykonywania obliczeń, ale do przyjaznej komunikacji z użytkownikiem. Z kolei

[1] `http://xkcd.com/538/`

techniki interaktywnej pracy z użytkownikiem rozwinięto w nowych systemach operacyjnych dla urządzeń mobilnych. Na rozwój systemów operacyjnych wpływają również nowe urządzenia wejścia-wyjścia. W najnowszych urządzeniach całkowicie zrezygnowano z podzespołów elektromechanicznych (dyski twarde) na rzecz rozwiązań elektronicznych (dyski SSD, ekrany dotykowe itp.). Innym znakiem czasów jest obecność w prawie każdym nowym systemie operacyjnym modułu do utrzymywania łączności z siecią komputerową.

Systemy operacyjne stanowią ugruntowaną i ciągle rozwijającą się gałąź współczesnej informatyki. Ułatwiają one tworzenie oprogramowania oraz prowadzą do standaryzacji obsługi urządzeń mikroprocesorowych. Można się spodziewać, że coraz więcej urządzeń mikroprocesorowych będzie nimi zarządzana.

4.12. Zadania

1. Wymień dwie zalety i dwie wady stosowania systemów operacyjnych.
2. Podaj definicję systemu operacyjnego.
3. Wyjaśnij, czym różnią się wieloprogramowe systemy operacyjne od systemów z podziałem obliczeń.
4. Wymień pięć podstawowych zadań systemu operacyjnego.
5. Omów dwa mechanizmy sprzętowe wbudowywane w systemy mikroprocesorowe wspierające konstruowanie systemów operacyjnych.
6. Zdefiniuj pojęcie zasobu.
7. Wymień i omów pięć składników bloku kontrolnego procesu.
8. Wymień i omów pięć podstawowych stanów procesu pracującego pod nadzorem systemu operacyjnego.
9. Podaj i omów cztery sytuacje, kiedy proces może być przełączany.
10. Podaj i omów trzy przykładowe algorytmy szeregowania procesów.
11. Wyjaśnij, na czym polegają zjawiska zakleszczenia i zagłodzenia.
12. Wyjaśnij, jak można zrealizować wzajemne wykluczanie w wykonywaniu sekcji krytycznej z zastosowaniem semafora.
13. Wymień i omów cztery zadania systemu operacyjnego związane z zarządzaniem pamięcią.

14. Wyjaśnij, na czym polega przydział pamięci zgodnie z zasadą partycjonowania statycznego.

15. Wyjaśnij, na czym polega przydział pamięci zgodnie z zasadą partycjonowania dynamicznego.

16. Wyjaśnij, na czym polega przydział pamięci zgodnie z zasadą stronicowania.

17. Zdefiniuj pamięć wirtualną oraz omów jej zastosowania.

18. Wymień i omów pięć przykładowych atrybutów plików.

19. Wymień i omów pięć przykładowych operacji plikowych.

20. Wymień i omów pięć przykładowych usług katalogowych.

21. Wymień i omów trzy sposoby zarządzania jednostkami fizycznej pamięci masowej.

22. Wskaż i omów trzy podstawowe komponenty sprzętowo-programowe podsystemu obsługi urządzeń wejścia-wyjścia w systemie operacyjnym.

23. Podaj i omów dwie techniki służące do komunikacji się z urządzeniami wejścia-wyjścia, którymi posługują się moduły sterujące.

Rozdział 5.

Bazy danych

Wojciech Rząsa

Data matures like wine,
applications like fish.[1]

5.1. Wprowadzenie

Zagadnienie projektowania i eksploatacji baz danych szeroko omawiane choćby w pracach [11, 12] jest bardzo istotną częścią nie tylko teorii informatyki, ale przede wszystkim praktycznego jej zastosowania. Okazuje się bowiem, że oprogramowanie zmienia się stosunkowo szybko, a niekiedy należałoby powiedzieć nawet bardzo szybko. Im bliżej końcowego użytkownika jest dany fragment programu, tym większa szansa, że konieczna będzie jego szybka aktualizacja, a nawet wymiana. Coraz nowsze, wygodniejsze, bardziej rozbudowane programy, przeznaczone dla nowych platform (ostatnio np. mobilnych) bardzo często korzystają z tych samych baz danych, z których korzystały ich dawno zapomniane poprzedniki. Doskonałym przykładem są systemy bankowe, które w ostatnich latach umożliwiają klientom wygodny dostęp do konta z telefonów komórkowych. Wiele z tych systemów nadal korzysta z baz danych, które powstawały w latach 80. czy 90. XX w., kiedy to nie tylko nie myślano o telefonii komórkowej, ale nawet powszechny dostęp do Internetu był akceptowany jako tematyka bliska *science-fiction*. Taka sama sytuacja dotyczy baz danych korporacji przechowujących informacje o ich działalności. Dzieje się tak również ze znacznie mniej skomplikowanymi systemami informatycznymi o znacznie mniejszym budżecie. Ostatecznie nawet zapis wydatków domowych wygodnie byłoby mieć na stałe, niezależnie od komputera czy systemu operacyjnego, z jakiego będziemy korzystać za kilka lat.

[1] *Dane starzeją się jak wino, aplikacje jak ryba*. Cytat jest zwięzłym podsumowaniem tekstu Jamesa Governora, który można znaleźć pod adresem: `http://redmonk.com/jgovernor/2007/04/05/why-applications-are-like-fish-and-data-is-like-wine/`

Stąd motto tego rozdziału doskonale wyjaśnione w artykule wymienionym w przypisie. Projektowanie i eksploatacja baz danych to na ogół znacznie bardziej istotne zagadnienie niż tworzenie ładnie wyglądającej aplikacji dla użytkownika końcowego w najnowszej technologii. Źle zaprojektowana aplikacja, której czas życia przewiduje się najwyżej na kilka lat, będzie prawdopodobnie znacznie mniej kłopotliwa, niż źle zaprojektowana baza danych i straty z tego tytułu będą znacznie mniejsze.

Należy także zwrócić uwagę, że bazy danych to bardzo dojrzały dział informatyki. Wiadomo jak należy je projektować, tworzyć i eksploatować, aby dobrze działały. Zasady te w przeciwieństwie do wielu technologii nie zmieniają się z roku na rok. Technologie i aplikacje przeznaczone bezpośrednio dla użytkowników – jak pisze autor wymienionego już artykułu – uczą się, amortyzują wstrząsy związane z rozwojem technologii, przyciągają uwagę – są podatne na zmiany spowodowane modą, zmianami technologicznymi itp. Im dalej od tej pierwszej warstwy, tym mniej gwałtownych zmian w systemach informatycznych, a tym więcej dojrzałości. Na samym dole tego „stosu" znajdują się bazy danych.

Baza danych to zbiór powiązanych ze sobą danych. Są one uporządkowane w określoną strukturę i na ogół ich ilość jest zbyt duża, żeby mogły się zmieścić w pamięci operacyjnej. Strukturę zapisanych danych opisuje *schemat bazy danych* i jest on konieczny do zinterpretowania zapisanych w bazie informacji.

Obecnie najczęściej są stosowane relacyjne bazy danych, choć coraz większą popularność w specyficznych zastosowaniach zyskują tzw. bazy nie-SQLowe (*no-SQL*) bądź bezschematowe. Ten rozdział skupia się głównie na bazach danych relacyjnych jako najpopularniejszych. Dodatkowo omówiono też krótko inne rozwiązaniach.

5.2. Problem trwałego przechowywania danych

Dane, na których bezpośrednio operują programy komputerowe, są przechowywane w pamięci operacyjnej. Struktury, w jakich są przechowywane dane, są dostosowywane do operacji wykonywanych przez konkretny program, zapewniając efektywność programu i wygodę programiście. Struktury danych opisano szerzej w rozdziale 4. publikacji [40].

Aby przechowywać dane pomiędzy kolejnymi uruchomieniami programu, zapisuje się je na dyskach przed zakończeniem działania i wczytuje po kolejnym uruchomieniu programu. Jeśli ilość przetwarzanych danych jest niewielka, a ich struktura jest niezbyt złożona, takie podejście dobrze się sprawdza. Problem pojawił się, gdy programy zaczęły operować na danych, których ilość wykluczała umieszczenie ich w pamięci w całości. System operacyjny odpowiada za zarządzanie plikami na dyskach (zob. rozdział 4.), ale zarządzanie zawartością danego pliku

należy do korzystającego z niego programu. Na programistę każdorazowo spadała więc odpowiedzialność za odszukanie właściwych danych w pliku na dysku, a po przetworzeniu zapisanie ich z powrotem we właściwym miejscu. Powodowało to znaczne komplikacje i zmuszało programistów do poświęcania znacznej ilości czasu operacjom wejścia-wyjścia, które nie były kluczowe dla tworzonego oprogramowania.

Programiści dość szybko zauważyli, że do wygodnego operowania na danych na dyskach wystarczyłaby dość ograniczona liczba operacji, które można na nich wykonać, jak odczyt, zapis, wyszukiwanie. Problem stanowiło zorganizowanie danych w sposób umożliwiający efektywne wykonanie tych operacji oraz takie ich zaimplementowanie, żeby działały efektywnie, niezależnie od tego, jaki zbiór danych jest przetwarzany. Zaczęły powstawać pierwsze rozwiązania umożliwiające zarządzanie bazami danych. Rozwiązania te dalekie były jednak od ogólności i schemat przechowywanych danych był dostosowany specyficznie do operacji wykonywanych przez dany program.

5.3. Systemy zarządzania bazą danych

System zarządzania bazą danych (ang. *Database Management System* – DBMS) to oprogramowanie odpowiadające za zarządzanie danymi przechowywanymi w pamięci trwałej (np. na dyskach) i udostępniające programom możliwości wykonywania na tych danych określonych operacji. Dzięki zastosowaniu takich systemów programiści nie muszą zajmować się niskopoziomowymi szczegółami związanymi z przechowywaniem danych. Oprogramowanie, które obecnie korzysta z baz danych, wykorzystuje w tym celu jeden z istniejących na rynku systemów zarządzania bazami danych. Więcej na temat dostępnych systemów tego typu można przeczytać w podrozdziale 5.6.8.

System zarządzania bazą danych pełni szereg zadań, które zapewniają programiście wygodę, a programom efektywność i bezpieczeństwo przechowywanych danych.

Interakcja z systemem plików. DBMS odpowiada za zapisywanie danych na dyskach, za odczyt zapisanych danych, a także za ich lokalizację w plikach.

Zapewnienie integralności (poprawności) danych pozwala zdefiniować warunki, jakie powinny spełniać dane zapisywane w bazie i wyegzekwować je np. poprzez uniemożliwienie zapisu niepoprawnych danych. Przykładem może być zapewnienie, że ocena studenta nie będzie mniejsza niż 2 i większa niż 5.

Bezpieczeństwo i autoryzacja dostępu zapewniają, że dostęp do określonych danych mają jedynie uprawnieni użytkownicy.

Kopie zapasowe i odtwarzanie danych. Funkcja ta zapewnia bezpieczeństwo danych w przypadku awarii sprzętowej, pojawienia się intruza bądź przypadkowego uszkodzenia danych.

Kontrola współbieżnego dostępu zapewnia, że równocześnie wielu użytkowników i wiele programów może korzystać z danych bez groźby ich uszkodzenia.

Przetwarzanie poleceń użytkownika zapewnia użytkownikowi czy programiście możliwość wykonywania operacji na danych.

Zarządzanie strukturą bazy danych zapewnia użytkownikowi bądź programiście możliwość definiowania i modyfikacji struktury bazy danych.

5.4. Korzyści z zastosowania baz danych

Krzysztof Świder i in. w pracy *Inżynieria systemów informatycznych* [46] opisuje charakterystykę baz danych i korzyści z ich zastosowania. Zastosowanie baz danych zapewnia *niezależność aplikacji i danych*. Dane można wprowadzać i modyfikować niezależnie od aplikacji. Podobnie aplikację można rozwijać i poprawiać niezależnie od przechowywanych danych. Baza danych zapewnia *abstrakcyjną reprezentację danych*. Dzięki temu programista może zdefiniować na wysokim poziomie warunki wyboru danych czy inne wykonywane na nich operacje. Określenie *co* należy zrobić z danymi, a nie *jak* dokładnie należy zrealizować te operacje pozwala na znacznie wygodniejsze operowanie na danych. Na ogół pojedynczy zbiór danych może być użyteczny dla wielu różnych programów. Przykładowo, z bazy danych studentów może korzystać system dziekanatowy, system obsługi praktyk studenckich, system studenckiej poczty elektronicznej itd. Dobrze zaprojektowana baza danych umożliwia *różne perspektywy patrzenia na dane*. Dzięki temu wielu użytkowników w różnych celach może korzystać z tego samego zbioru danych. System zarządzania bazą danych odpowiada za fizyczną reprezentację danych, zapewniając korzystającym z niego programom *fizyczną niezależność danych*. Dzięki temu w przypadku wymiany sprzętu czy oprogramowania odpowiadającego za zarządzanie zbiorem danych korzystające z tych danych aplikacje nie wymagają zmian. Zapewnia to trwałość bazy danych, ponieważ – jak już wspomniano – bazy danych bardzo często mają znacznie dłuższy czas życia niż korzystające z nich oprogramowanie. *Logiczna niezależność danych* zapewniana przez poprawnie zaprojektowaną bazę danych pozwala na wprowadzanie nowych danych bez dezaktualizacji starych oraz na niezależne aktualizowanie danych, które nie są ze sobą logicznie powiązane.

Zastosowanie baz danych zmniejsza więc kłopotliwą nadmiarowość danych. Skoro wiele niezależnych aplikacji może korzystać z tej samej bazy danych, to

nie ma konieczności przechowywania, utrzymywania i synchronizowania wielu kopii tych samych danych. Dane przechowywane w poprawnie zaprojektowanej bazie mogą być współdzielone nie tylko przez różne aplikacje, ale także przez wiele instytucji. Konieczna w takiej sytuacji autoryzacja dostępu także może być zapewniona przez DBMS. Te same dane mogą być wykorzystywane na różne sposoby w zależności od potrzeb. Różne aplikacje korzystające z tej samej bazy danych mogą zapewnić dostęp do nich za pomocą różnych, potrzebnych użytkownikom interfejsów. Dziekanat może mieć dostęp do danych o przedmiotach i ocenach studentów, dział stypendialny jedynie do wyliczonych w określony sposób średnich, studenci do własnych wyników, a administratorzy systemów poczty elektronicznej tylko do danych pozwalających na autoryzację użytkowników tych systemów. Na ogół dane opisujące fragment rzeczywistości są od siebie zależne. Oceny są powiązane ze studentami, prowadzącymi, którzy je wystawili, przedmiotami, z których zostały wystawione, przedmioty są powiązane z zajęciami, te znów z prowadzącymi te zajęcia pracownikami i studentami, którzy w nich uczestniczą. Bazy danych umożliwiają reprezentację tych złożonych związków znaczeniowych i ich poprawną interpretację oraz wykorzystanie w programach. Ograniczenia integralnościowe, o których już wspomniano, pozwalają zapewnić poprawność danych w bazie, niezależnie od tego, za pomocą jakiego oprogramowania zostały wprowadzone. Przechowywanie wszystkich istotnych danych w jednym miejscu ułatwia zorganizowanie ochrony przed awariami i ich utratą, np. poprzez wykonywanie kopii zapasowych.

Zarządzanie danymi jest złożonym problemem, szczególnie dla większych zbiorów danych. Bazy danych dają gotowe recepty na rozwiązanie tych problemów i zapewniają programistom wygodny dostęp do potrzebnych im danych. Dzięki zastosowaniu baz danych przechowywanie danych w programach stało się znacznie prostsze i coraz więcej programów korzysta z usług systemów zarządzania bazami danych. Typowymi przykładami są banki, sklepy internetowe czy biblioteki. Mniej oczywiste zastosowania, to np. systemy poczty elektronicznej, które przechowują w bazach danych informacje o użytkownikach. Nawet zwykłe programy okienkowe, jak choćby popularna przeglądarka www Firefox, korzystają z prostych relacyjnych baz danych do przechowywania informacji o konfiguracji czy danych użytkownika.

5.5. Modele danych

Jak zaznaczają autorzy wspomnianej pracy [46], podczas projektowania i implementacji baz danych wyróżnia się trzy modele danych. Służą one przeprowadzeniu analizy problemu, utworzeniu właściwego schematu bazy danych oraz przechowywaniu tych danych na nośnikach.

Model konceptualny powstaje w początkowej fazie projektowania bazy danych. Przedstawia związki pomiędzy danymi, które mają być przechowywane, które wynikają ze specyfiki analizowanego problemu. Model ten jest najczęściej przedstawiany za pomocą diagramu związków encji (ERD), który zostanie opisany dalej.

Model implementacyjny to reprezentacja modelu konceptualnego w postaci odpowiedniej dla wybranej bazy danych. Przykładowo, dla modelu relacyjnego (jak to zostanie opisane w dalszej części rozdziału) będzie to reprezentacja w postaci tabel.

Model fizyczny opisuje sposób zapisu danych na dyskach czy innych nośnikach. Za realizację tego modelu odpowiada system zarządzania bazą danych. Model ten nie musi być znany programistom wykorzystującym bazy danych, może się też zmieniać bez szkody dla programów korzystających z danych.

5.6. Relacyjne bazy danych

5.6.1. Powstanie

Za czas powstania relacyjnych baz danych uważa się 1970 rok, kiedy to Edgar F. Codd, będący wówczas pracownikiem firmy IBM, opublikował pracę pod tytułem *A Relational Model of Data for Large Shared Data Banks* [7]. Jakkolwiek koncepcja ta była obiecująca, to do końca lat 70. XX w. nie było dostępnych maszyn dość szybkich, aby możliwe było praktyczne wykorzystanie modelu relacyjnego. Pierwsze komercyjne systemy baz danych zaczęły pojawiać się pod koniec lat 70., wtedy też pojawił się język SQL służący do zarządzania danymi przechowywanymi w bazie. W tym czasie powstały następujące systemy bazodanowe:

- **1974/5 r. IBM System R**, na podstawie którego utworzono wciąż rozwijaną i sprzedawaną przez IBM bazę danych DB2.
- **1978 Oracle** do dzisiaj rozwijana, oferowana przez firmę Oracle.
- **1979 INGRES Berkeley** będąca między innymi podstawą bazy danych PostgreSQL – jednego z najlepszych darmowych systemów zarządzania bazą danych.

W latach 80. XX w. nastąpił rozwój maszyn cyfrowych pozwalający na praktyczne wykorzystanie modelu relacyjnego, w rezultacie czego model ten stawał się coraz bardziej popularny. W latach 90. model ten stał się powszechnie stosowanym rozwiązaniem bazodanowym. Obecnie, jakkolwiek stosuje się w specyficznych sytuacjach inne rozwiązania, to model relacyjny jest wszechobecny i jest rozwiązaniem niemal automatycznie narzucającym się w miejscach, w których potrzebne jest zastosowanie baz danych.

Tablica 5.1. Przykładowa relacja `studenci`

Kolumny

	imie	**imie2**	**nazwisko**	**numer indeksu**
Rekordy	Jan	Marek	Kowalski	73693
	Józef		Nowak	23098
	Krzysztof	Jan	Kwiatkowski	30432
	Anna	Agnieszka	Nowacka	49754
	Janina		Staniszewska	82345

5.6.2. Koncepcja

Przechowywanie danych

Koncepcja relacyjnej bazy danych zakłada, że dane są przechowywane w „tabelkach" nazywanych *relacjami*. Pojęć *tabela* oraz *relacja* używa się zamiennie. Relacja – jak każda tabela – składa się z kolumn i wierszy, przy czym wiersze są nazywane *rekordami*. Jeśli relacja miałaby służyć do przechowywania danych studentów, to kolejne rekordy (wiersze) reprezentowałyby kolejnych studentów, a poszczególne kolumny dane tych studentów (np. imię, drugie imię, nazwisko, numer indeksu, numer telefonu jak w tabl. 5.1). Liczba kolumn jest więc stała, definiowana w momencie tworzenia bazy danych. Na tym etapie należy zdecydować, jakie informacje będą przechowywane w danej relacji. Każda kolumna ma też na stałe zdefiniowany typ przechowywanych w niej danych (np. imię jest typu tekstowego, numer indeksu to liczba całkowita itd.). Liczba wierszy (rekordów) w danej relacji zmienia się, bo jest zależna od liczby studentów, których dane są w niej przechowywane. Wypełnienie kolumn danymi może być obowiązkowe bądź opcjonalne. Jeśli jest opcjonalne, to w relacji (tabeli) mogą znajdować się rekordy (wiersze), w których dana kolumna ma wartość pustą. Nie da się jednak dodać do relacji rekordu, w którym nie wypełniono danymi kolumn obowiązkowych.

Relacyjna baza danych składa się oczywiście z wielu relacji przechowujących różne dane. W bazie danych dziekanatu poza tabelą z danymi studentów mogą się znajdować także tabele z informacjami o prowadzących czy przedmiotach. Każda z tych relacji będzie miała inny zestaw kolumn o określonych typach.

Powiązania danych i klucze

Dane przechowywane w różnych relacjach mogą być ze sobą powiązane. Jako przykład możemy rozważyć relację `stypendia`, gdzie są przechowywane informacje o wysokości i rodzaju stypendiów poszczególnych studentów. Każdy student

Tablica 5.2. Przykładowa relacja `stypendia`

numer indeksu	kwota	rodzaj
73693	300,00	socjalne
73693	500,00	naukowe
23098	700,00	unijne
30432	100,00	socjalne
30432	200,00	naukowe
49754	400,00	unijne
82345	300,00	socjalne
49754	100,00	naukowe
82345	400,00	naukowe
30432	500,00	unijne

może mieć niezależnie kilka różnych stypendiów, np. naukowe, socjalne, unijne. Na podstawie kolumny `numer indeksu` można stwierdzić, któremu studentowi przyznano dane stypendium.

Wyszukiwanie w relacyjnej bazie danych odbywa się na podstawie danych przechowywany w relacjach. Poszczególne rekordy nie mają nadanych przez system bazodanowy adresów, wskaźników czy numerów. Jedyny sposób wskazania, o który rekord chodzi, polega na podaniu znajdujących się w nim danych (np. rekord, w którym imię to Jan, a nazwisko to Kowalski). Może się oczywiście zdarzyć, że więcej niż jeden rekord będzie spełniał tak określone kryterium.

Aby umożliwić jednoznaczną identyfikację rekordów w relacji, projektując bazę danych, należy dla każdej relacji znaleźć informację, która jest wystarczająca do jednoznacznej identyfikacji rekordu. Taką informację nazywa się *kluczem* danej relacji. Rolę klucza w relacji `studenci` z tabl. 5.1 pełni `numer indeksu`. Klucz może się składać z jednej lub wielu kolumn i w praktycznych zastosowaniach można znaleźć wiele różnych kolumn bądź zestawów kolumn, które mogą pełnić rolę kluczy. Jeśli klucz składa się z jednej kolumny, jest nazywany *kluczem prostym*. Klucz składający się z wielu kolumn jest nazywany *kluczem złożonym*.

Każdą kolumnę bądź zestaw kolumn, które spełniają kryterium klucza, nazywa się *kluczem kandydującym*. Spośród kluczy kandydujących wybierany jest jeden, który będzie używany w projektowanej bazie danych. Klucz kandydujący danej relacji wybrany do reprezentowania związków z innymi relacjami nazywa się *kluczem głównym* danej relacji. Kluczem głównym relacji `studenci` używanym w przykładzie przedstawionym w tabl. 5.1 i 5.2 jest `numer indeksu`, który z założenia jednoznacznie identyfikuje studenta.

Numer indeksu – klucz główny relacji `studenci` – jest umieszczany w relacji `stypendia`, aby zaznaczyć, do którego studenta należy dana ocena. Klucz

Tablica 5.3. Przykładowa relacja `przedmioty`

id	nazwa	wyklad	cwiczenia	lab	projekt
1	Informatyka	30	0	45	0
2	Algebra	30	30	0	0
5	Analiza matematyczna	45	30	0	0
7	Fizyka	30	30	30	0

główny relacji umieszczony w innej relacji w celu zaznaczenia związku pomiędzy poszczególnymi rekordami tych relacji jest nazywany *kluczem obcym*. Kolumna `numer indeksu` w relacji `stypendia` jest więc kluczem obcym łączącym tę relację z relacją `studenci`.

Czasami zdarza się, że w relacji trudno znaleźć odpowiedni klucz kandydujący. W takiej sytuacji można dodać do niej dodatkową kolumnę, która nie będzie reprezentowała rzeczywistego atrybutu danych przechowywanych w relacji, a system bazodanowy zadba, żeby jednoznaczne identyfikowała rekord. Taką kolumnę często nazywa się `id`. Systemy zarządzania bazą danych dostarczają narzędzi pozwalających automatycznie wypełniać taką kolumnę unikalnymi wartościami numerycznymi. Przykład relacji, w której dodano kolumnę `id`, aby stanowiła klucz główny, pokazano w tabl. 5.3. Jakkolwiek nazwy przedmiotów w tej relacji w podanym przykładzie są unikalne, może się jednak zdarzyć, że nie będą. Algebra – przedmiot o identyfikatorze 2 – może być prowadzony dla określonego kierunku studiów w wymiarze 30 godzin wykładu i 30 godzin ćwiczeń. Nic nie stoi jednak na przeszkodzie, aby na innym kierunku studiów także pojawił się przedmiot o nazwie Algebra, ale w wymiarze 45 godzin wykładu i 30 godzin ćwiczeń. W tej sytuacji w relacji z tabl. 5.3 będą się znajdować dwa rekordy zawierające przedmiot o nazwie Algebra. Rozróżnienie tych dwóch rekordów zapewni wartość z kolumny `id`.

W praktyce najczęściej do każdej relacji dodawana jest dodatkowa kolumna, która niezależnie od rodzaju i formatu przechowywanych danych będzie stanowić jej klucz główny. Robi się tak także dla relacji, w których można znaleźć naturalne klucze kandydujące. Dzięki takim rozwiązaniom niezależna od projektujących bazę danych zmiana w rodzaju czy formacie przechowywanych danych nie powoduje konieczności przeprojektowania całej bazy danych. W praktycznym rozwiązaniu przykład relacji `studenci` i `przedmioty` wyglądałby więc raczej tak, jak to przedstawiają tabl. 5.4 i tabl. 5.5, gdzie kluczem głównym relacji studenci jest dodatkowa kolumna o nazwie `id`. Dzięki temu, jeśli okaże się, że uczelnia zmieni format numeru indeksu albo na przykład zdecyduje się, że numer indeksu po jakimś czasie (np. 10 latach) może się jednak powtarzać, konieczne zmiany w schemacie

Tablica 5.4. Przykładowa relacja `studenci`, w której jako klucz główny dodano dodatkową kolumnę `id`

id	imie	imie2	nazwisko	numer indeksu
1	Jan	Marek	Kowalski	73693
3	Józef		Nowak	23098
4	Krzysztof	Jan	Kwiatkowski	30432
10	Anna	Agnieszka	Nowacka	49754
11	Janina		Staniszewska	82345

Tablica 5.5. Przykładowa relacja `stypendia`

student_id	kwota	rodzaj
1	300,00	socjalne
1	500,00	naukowe
3	700,00	unijne
4	100,00	socjalne
4	200,00	naukowe
1	400,00	unijne
1	300,00	socjalne
1	100,00	naukowe
1	400,00	naukowe
4	500,00	unijne

bazy danych i istniejących w niej danych będą znacznie mniejsze i mniej kłopotliwe. Do tego rozwiązania będziemy się więc odnosić w dalszej części rozdziału.

Krotności związków

Związki pomiędzy danymi przechowywanymi w różnych relacjach mogą mieć różną krotność. Związek pomiędzy relacją `studenci` i `stypendia` ma krotność *jeden do wielu*: jeden student ma wiele stypendiów (może też nie mieć żadnego), ale każde stypendium należy do jednego studenta.

Jeśli podczas projektowania bazy danych pojawiają się związki *jeden do jednego* pomiędzy dwoma relacjami, w większości przypadków oznacza to, że relacje te należy połączyć w jedną. Nowa relacja będzie składała się z kolumn pochodzących z obydwu łączonych relacji.

Podczas projektowania baz danych często zdarza się, że pojawiają się związki *wiele do wielu*. Przykładowo, jeden student uczęszcza na wiele przedmiotów, a na każdy przedmiot uczęszcza wielu studentów. Takiego związku nie da się

Tablica 5.6. Przykładowa relacja łącząca relacje `studenci` i `przedmioty` realizująca związek wiele do wielu pomiędzy nimi

przedmiot_id	**student_id**
1	1
1	3
2	3
5	11
5	10
1	10
7	4
2	11

wprost zrealizować za pomocą jedynie dwóch relacji. Potrzebna jest do tego relacja pośrednicząca. Taka relacja składa się z dwóch kolumn[2], każda z kolumn przechowuje klucze obce łączonych relacji.

Aby zrealizować związek wiele do wielu pomiędzy studentami (tabl. 5.4) a przedmiotami (tabl. 5.3), należy wprowadzić dodatkową relację. Taką relację przedstawia tabl. 5.6. Dane przechowywane w tej relacji pozwalają stwierdzić, że na przedmiot o identyfikatorze 1 uczęszczają studenci o identyfikatorach 1, 3 i 10, oraz że student o identyfikatorze 3 uczęszcza na przedmioty o identyfikatorach 1 i 2. Szczegółowe dane studentów można odczytać z relacji `studenci`, a dane o przedmiotach z relacji `przedmioty`.

W wielu przypadkach istnienie relacji pośredniczącej ma naturalną interpretację, wtedy nie ma kłopotu z nadaniem jej nazwy. W opisanym przykładzie związek pomiędzy studentami a przedmiotami można określić jako konieczność uzyskania z danego przedmiotu *zaliczenia* i tak zapewne należy nazwać relację z tabl. 5.6. W takich sytuacjach często pojawia się konieczność dodania do relacji pośredniczących dodatkowych kolumn. W tym przypadku relacja ta jest naturalnym miejscem, gdzie można zapisać ocenę studenta z zaliczenia danego przedmiotu tabl. 5.7.

Gdy trudno znaleźć naturalną nazwę dla relacji pośredniczącej, często nazywa się ją, łącząc nazwy relacji, których związek realizuje dana relacja za pomocą znaku podkreślenia. W tym przypadku relacja nazywałaby się `przedmioty_studenci` albo `studenci_przedmioty`. Jeśli to jednak możliwe, lepiej znaleźć naturalną nazwę dla takiej relacji.

[2] Zakładając, że kluczami głównymi łączonych relacji są klucze proste – składające się z pojedynczych kolumn. Jeśli nie, relacja pośrednicząca będzie składała się z większej liczby kolumn reprezentujących klucze złożone obydwu łączonych relacji.

Tablica 5.7. Przykładowa relacja `zaliczenia` łącząca relacje `studenci` i `przedmioty` realizująca związek wiele do wielu pomiędzy nimi i przechowująca dane o ocenach

przedmiot_id	student_id	ocena
1	1	4.0
1	3	
2	3	4.5
5	11	
5	10	
1	10	2.0
7	4	5.0
2	11	

Opcjonalność związków

Związki pomiędzy danymi w różnych relacjach mogą być opcjonalne bądź obowiązkowe. Przykładowo, student może mieć jedno bądź wiele stypendiów, ale może także nie mieć żadnego (tabl. 5.4 i 5.5). Związek ten jest więc opcjonalny po stronie studenta. Oznacza to, że w relacji `studenci` może istnieć rekord niepowiązany z żadnym rekordem relacji `stypendia`. Inaczej wygląda sytuacja w przypadku stypendiów. Każde ze stypendiów musi należeć do jakiegoś studenta, żeby było wiadomo komu przyznano dane stypendium i komu należy je wypłacać. Związek jest więc obowiązkowy po stronie relacji `stypendia`. Oznacza to, że każdy rekord z relacji `stypendia` musi mieć odpowiadający mu rekord z relacji `studenci`. Związek pomiędzy relacją `studenci` i `stypendia` jest więc typu jeden do wielu, opcjonalny po stronie „jeden” (po stronie studenta) i obowiązkowy po stronie „wiele” (po stronie stypendiów).

System zarządzania bazą danych dostarcza narzędzi pozwalających wymusić poprawność związków obowiązkowych. W poprawnie utworzonej bazie danych nie będzie możliwości dodania do relacji `stypendia` rekordu, w którym `student_id` nie będzie miał odpowiednika w relacji polu `id` `studenci`. Jeśli związek po stronie relacji `stypendia` także miałby być opcjonalny, wystarczyłoby zdefiniować, że do relacji `stypendia` można dodać rekord, w którym `student_id` ma wartość pustą.

5.6.3. Schemat relacyjnej bazy danych

Schemat relacyjnej bazy danych stanowią definicje tabel (z kolumnami i ich typami) wraz z powiązaniami i ograniczeniami integralnościowymi. Ograniczenia

te określają, kiedy dane są poprawne i jakie powiązania są obowiązkowe. W relacyjnych bazach danych schemat bazy jest stały – ustalany na początku podczas projektowania bazy – i musi umożliwiać przechowywanie wszystkich wymaganych danych. Możliwa jest oczywiście rozbudowa bazy danych wymagająca zmiany jej schematu, działania takie nie należą jednak do codziennych operacji wykonywanch na bazach danych. Wymagają one przeprojektowania bazy i mogą wymagać modyfikacji już istniejących danych, co zawsze może się okazać ryzykowne i powinno być podejmowane z odpowiednią dozą ostrożności.

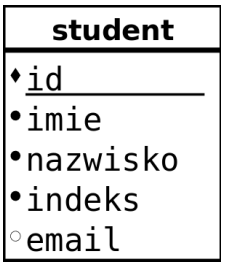
5.6.4. Projektowanie baz danych

Schematy rzeczywistych baz danych składają się z bardzo wielu powiązanych ze sobą relacji. Aby je poprawnie zaprojektować, wykonuje się najpierw model konceptualny (zob. 5.5) i poprawia go, usuwając błędy i anomalie. Następnie na jego podstawie tworzy się definicję schematu bazy danych dla wybranego DBMS.

ERD – diagramy związków encji

Do zapisania modelu konceptualnego najczęściej używa się ERD – ang. *Entity Relationship Diagram*, czyli diagramu związków encji. Diagram ten pełni podobną rolę jak rysunek techniczny – jest podstawą do dyskusji, analizy poprawności projektu i ostatecznie implementacji. Niestety istnieje wiele notacji przeznaczonych do tworzenia ERD. W niniejszej pracy zaprezentowano jedną z częściej stosowanych, używanych między innymi przez firmę Oracle.

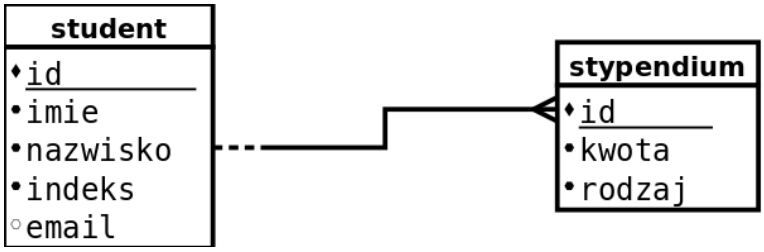
ERD jest diagramem składającym się z powiązanych ze sobą *encji*. Przykładowo, rys. 5.1 przedstawia encję o nazwie *student*, której kluczem głównym jest *id*, polami (atrybutami) obowiązkowymi *imię*, *nazwisko* oraz *indeks*, natomiast polem opcjonalnym *email*.



Rys. 5.1. Przykładowa reprezentacja *encji* na ERD

ERD reprezentuje nie tylko encje i ich zawartość, ale także związki pomiędzy nimi oraz ich charakter: krotność i opcjonalność. Przykład reprezentacji związku

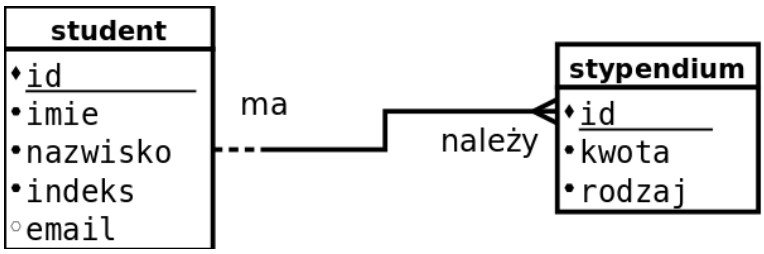
wiele do jednego przedstawia rys. 5.2. W tym przypadku jeden student jest powiązany z wieloma stypendiami, ale każde stypendium jest powiązane tylko z jednym studentem. Przerywana linia po stronie studenta oznacza, że związek po tej stronie jest opcjonalny – student *może mieć* wiele stypendiów (ale może też nie mieć żadnego). Linia po stronie stypendium jest ciągła, co oznacza, że po tej stronie związek jest obowiązkowy – stypendium *musi mieć* dokładnie jednego studenta.



Rys. 5.2. Przykład reprezentacji na ERD związku wiele do jednego wraz z opisem

Jak widać na przykładzie, odczytywanie charakteru związków pomiędzy encjami należy rozpocząć od odczytania nazwy encji, następnie (na podstawie tego, czy linia jest ciągła czy przerywana) opcjonalności, potem krotności, na końcu nazwy drugiej encji. Odczytując w ten sposób diagram z rys. 5.2, otrzymujemy dwa stwierdzenia: *student może mieć wiele stypendiów* oraz *stypendium musi mieć jednego studenta*.

Aby ułatwić interpretację związków pomiędzy encjami, stosuje się dodatkowe opisy, jak na rys. 5.3. Wtedy zdania opisujące przedstawiony związek będą miały następującą postać: *student może mieć wiele stypendiów* oraz *stypendium musi należeć do jednego studenta*. Ta z pozoru mało istotna informacja i niewielka zmiana w interpretacji związku staje się dużym ułatwieniem w interpretacji bardziej złożonych diagramów. Nie zawsze znaczenie przedstawionych na ERD związków jest tak oczywiste, jak w przedstawionym przykładzie.

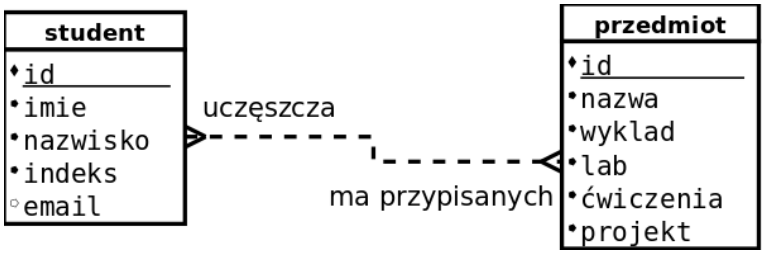


Rys. 5.3. Przykład reprezentacji na ERD związku wiele do jednego wraz z opisem

Oczywiście, jak wspomniano wcześniej, w bazach danych występują różnego rodzaju związki: o różnej krotności oraz o różnej opcjonalności. Krotność i opcjo-

nalność są niezależne od siebie – każdy związek może być obowiązkowy zarówno po stronie „jeden", jak i po stronie „wiele".

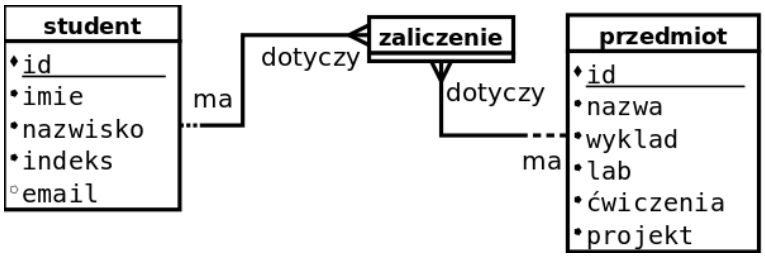
Poza związkami jeden do wielu występują także związki jeden do jednego oraz wiele do wielu. Związki te na ERD przedstawia się analogicznie do jeden do wielu. Przykład związku wiele do wielu przedstawia rys. 5.4. Dodatkowo związek ten po obu stronach jest opcjonalny: *student może uczęszczać na (jeden lub) wiele przedmiotów*, *przedmiot może mieć przypisanych (jednego lub) wielu studentów*.



Rys. 5.4. Przykład reprezentacji na ERD związku wiele do wielu

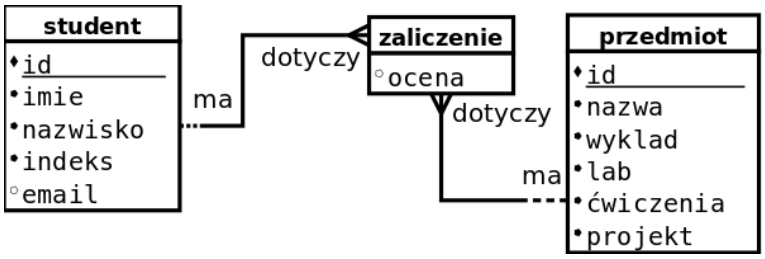
W trakcie pracy nad projektem bazy danych związki wiele do wielu pojawiają się wielokrotnie. W modelu konceptualnym są one dopuszczalne i ich dodawanie ułatwia pracę nad projektem. Reprezentacja takich związków nie jest jednak możliwa bezpośrednio w relacyjnej bazie danych. Dlatego na dalszym etapie projektowania bazy związki takie należy usunąć, zastępując je *encją pośredniczącą* oraz związkami jeden do wielu. Pojedynczy związek wiele do wielu zamienia się na dwa związki jeden do wielu za pomocą encji pośredniczącej zawsze w taki sam sposób. Diagram z przykładu na rys. 5.4 będzie miał postać taką, jak na rys. 5.5. W tym przypadku dla encji pośredniczącej udało się znaleźć naturalną nazwę – związek studenta z przedmiotem jest reprezentowany przez *zaliczenie* danego przedmiotu przez studenta (bądź interpretując to nieco inaczej – przez konieczność zdobycia *zaliczenia*). Należy także zwrócić uwagę, w jaki sposób opcjonalność relacji wiele do wielu została przeniesiona na nowe relacje. Gdyby na oryginalnym diagramie związek z którejkolwiek strony był obowiązkowy, należałoby usunąć opcjonalność po stronie „jeden" z odpowiedniego związku na nowym diagramie. Związki po stronie encji pośredniczącej powinny być obowiązkowe – nie ma sensu istnienie zaliczenia niepowiązanego z przedmiotem lub niepowiązanego z żadnym studentem.

Bardzo często po wprowadzeniu encji pośredniczącej okazuje się, że jej obecność w projekcie jest zupełnie naturalna i że istnieją dane, które zostały wcześniej przeoczone, a które powinny być przechowywane właśnie w tej encji. W przedstawionym przykładzie taką informacją jest ocena studenta z danego przedmiotu (rys. 5.6). Informacja ta nie może być przechowywana w encji *student*, gdyż wtedy student mógłby mieć tyko jedną ocenę i nie byłoby wiadomo jakiego przed-



Rys. 5.5. Przykład reprezentacji na ERD związku wiele do wielu za pomocą *encji pośredniczącej*

miotu ona dotyczy. Oceny nie można także umieścić w encji *przedmiot*, gdyż wtedy z danego przedmiotu można byłoby wystawić tylko jedną ocenę, na dodatek niepowiązaną z żadnym studentem. Poprawne jest umieszczenie oceny w encji reprezentującej związek studenta z przedmiotem.



Rys. 5.6. Encja pośrednicząca może zostać wykorzystana do przechowywania atrybutów

Czasami zdarza się, że na ERD pojawia się związek jeden do jednego. Niewiele jest sytuacji, w których da się uzasadnić pozostawienie takiego związku w bazie danych. Jednym z takich przypadków może być realizacja związku analogicznego do dziedziczenia z programowania obiektowego (zob. [40], rozdział 5.)[3], innym specyficzne kwestie związane z optymalizacją bazy danych[4]. Zwykle jed-

[3] Przykładowo, jeżeli baza danych ma przechowywać głównie dane o samochodach osobowych (marka, numer rejestracyjny), ale także o pewnej liczbie samochodów ciężarowych, dla których dodatkowo należy zapisać ładowność, można to zrealizować za pomocą związku jeden do jednego. Encja *samochody* będzie przechowywała dane podstawowe, natomiast encja *ciezarowe* dane o ładowności. Rekordy z encji *samochody* odpowiadające samochodom ciężarowym zostaną złączone związkiem jeden do jednego z odpowiednim rekordem encji *ciezarowe*. Dla uproszczenia przyjęto, że encja *ciezarowe* zawiera tylko jeden dodatkowy atrybut. W praktyce uzasadnienie takiego rozwiązania będzie wymagało przypadku, w którym encja przechowująca bardziej szczegółowe dane zawiera znacznie więcej atrybutów, przez co kłopotliwe i nieefektywne jest pozostawienie ich w encji *samochody*.

[4] Takie rozwiązanie można zastosować, gdy encja zawiera bardzo wiele atrybutów, większość z nich jest używana rzadko, a niewielka część bardzo często. Atrybuty, do których dostęp jest potrzebny często można wydzielić do osobnej encji. Ponieważ będzie ona znacznie mniejsza od ory-

nak w przypadku wystąpienia związku jeden do jednego należy połączyć encje, tworząc encję wynikową zawierającą atrybuty obydwu łączonych encji.

Realizacja projektu

Nietrudno zauważyć analogie pomiędzy ERD a relacjami i związkami pomiędzy nimi w relacyjnej bazie danych. Na początku projektowania modelu danych nie jest możliwe „przetłumaczenie" ERD na schemat bazy danych, choćby ze względu na występowanie związków wiele do wielu. W końcowej fazie projektowania bazy danych można (i należy) doprowadzić ERD do postaci, w której pojedyncza *encja* diagramu odpowiada niemal dokładnie *relacji* w bazie danych. Zasadnicza różnica polega na tym, że na ERD nie umieszcza się kluczy obcych, gdyż ich obecność jest reprezentowana przez zaznaczone graficznie związki pomiędzy encjami. Druga różnica polega na tym, że tradycyjnie relacje w bazie danych mają nazwy w liczbie mnogiej, natomiast encje w modelu konceptualnym, a więc także na ERD – w liczbie pojedynczej.

W relacyjnej bazie danych związki jeden do wielu są reprezentowane w naturalny sposób za pomocą kluczy obcych. Przykład stanowi porównanie ERD z rys. 5.3 z przykładem realizacji w postaci relacji przedstawionych na tabl. 5.4 i 5.5. Podobnie związki z rys. 5.6 są zrealizowane w przykładach relacji przedstawionych w tabl. 5.4, 5.7 oraz 5.3. Jak można zaobserwować na przykładach, aby zrealizować związek wiele do jednego, do relacji po stronie „wiele" należy dodać klucz obcy pochodzący z relacji po stronie „jeden". I tak: do relacji `stypendia` dodano klucz główny z relacji `studenci`, do relacji `zaliczenia` dodano klucz główny z relacji `studenci` oraz z `relacji` przedmioty.

Wymagania i anomalie

ERD stanowi narzędzie, które pozwala „zapisać" projekt bazy danych i konsultować go z innymi. Wykonanie projektu należy jednak do ludzi i dobry projekt musi spełniać określone wymagania. Relacyjna baza danych musi być zaprojektowana w taki sposób, aby było pewne, że nie będą w niej występować anomalie podczas wykonywania operacji na danych. Zmiany konieczne do wykonania przy ewentualnych przyszłych modyfikacjach powinna być minimalna, należy także zapewnić, że model danych będzie jasny i zrozumiały. Ostatnim, bardzo istotnym wymaganiem jest unikanie specjalizacji. Dalej dokładniej opisano poszczególne wymagania.

Anomalie. W błędnie zaprojektowanej bazie danych mogą wystąpić anomalie przy trzech rodzajach operacji wykonywanych na danych:

ginalnej, dostęp do niej będzie szybszy. Obydwie encje muszą być złączone związkiem jeden do jednego.

- *update anomaly* – anomalia przy aktualizacji,
- *insert anomaly* – anomalia przy dodawaniu rekordów,
- *deletion anomaly* – anomalia przy usuwaniu rekordów.

Anomalia przy aktualizacji występuje w sytuacji, w której jedna informacja jest przechowywana w wielu miejscach. Przykład takiej sytuacji przedstawia tabl. 5.8. Aby umożliwić przechowywanie wielu ocen dla każdego studenta, projektant bazy danych postanowił, że dodając nową ocenę, do relacji należy dodać nowy rekord, w którym będą umieszczone wszystkie dane studenta wraz z nowo dodawaną oceną. W ten sposób np. dane Krzysztofa Kwiatkowskiego są zapisane w trzech różnych rekordach, a dane Jana Kowalskiego w dwóch, za każdym razem z inną wartością w kolumnie `ocena`. Modyfikując dane osobowe studentów zapisanych w tej relacji, trzeba pamiętać, aby zmodyfikować wszystkie rekordy. Jeśli ktoś zaktualizuje adres email Krzysztofa Kwiatkowskiego tylko w jednym rekordzie, skutki będą następujące:

1. W bazie danych pojawi się nowy student – będzie dwóch Krzysztofów Kwiatkowskich różniących się adresami email.
2. Krzysztof Kwiatkowski, który dotąd miał zapisane trzy oceny, od tego momentu prawdopodobnie będzie miał tylko jedną (jeśli ktoś sprawdzający jego oceny, zgodnie z oczekiwaniami, poda jego dane wraz z nowym adresem email).
3. Pozostałe oceny należące dotąd do Krzysztofa Kwiatkowskiego będą należały do nieistniejącej osoby (Krzysztofa Kwiatkowskiego ze starym adresem email).

W źle zaprojektowanej bazie danych aktualizacja danych o adresie email spowodowała nieoczekiwane zmiany danych na temat liczby studentów, ocen studenta, właścicieli ocen. W przypadku tylko trochę bardziej złożonym niż przedstawiony bardzo trudno odnaleźć przyczynę takiego błędu i przywrócić poprawność danych – prawdopodobnie nikt nie będzie kojarzył modyfikacji adresu email z utraconymi przez studenta ocenami. Poprawnie zaprojektowana baza danych będzie się składała z dwóch relacji: jednej z danymi studentów, drugiej z danymi o ocenach połączonych związkiem jeden do wielu (jeden student może mieć wiele ocen), podobnie jak to ma miejsce w przykładzie ze studentami i stypendiami (tabl. 5.4, 5.5).

Anomalia przy dodawaniu rekordów powoduje, że niektórych informacji, które powinny się znaleźć w bazie danych, nie da się w niej w ogóle zapisać. Przykład relacji podatnej na tę anomalię prezentuje tabl. 5.9. Jest to relacja służąca firmie

Tablica 5.8. Przykład nieprawidłowo zaprojektowanej relacji `studenci`

imie	**nazwisko**	**indeks**	**email**	**ocena**
Jan	Kowalski	73693	jan@kowalski.net	5.0
Jan	Kowalski	73693	jan@kowalski.net	3.0
Józef	Nowak	23098	jozek@wp.pl	2.0
Krzysztof	Kwiatkowski	30432	krz@owq.pl	4.0
Krzysztof	Kwiatkowski	30432	krz@owq.pl	3.0
Krzysztof	Kwiatkowski	30432	krz@owq.pl	4.5
Anna	Nowacka	49754	ann@owq.pl	4.5
Janina	Staniszewska	82345	janina@stan.pl	5.0

Tablica 5.9. Przykład błędnie zaprojektowanej relacji `kursy`; kompletność danych zapewniono przez oznaczenie wszystkich kolumn jako obowiązkowych

imie	**nazwisko**	**temat**	**wykł.**	**lab.**
Jan	Kowalski	Systemy operacyjne	30	15
Józef	Nowak	Bazy danych	45	30
Janina	Kwiatkowska	Programowanie	30	30
Janina	Kwiatkowska	Algorytmy i struktury danych	30	30

prowadzącej kursy i szkolenia do zapisywania informacji o ofercie i osobach prowadzących poszczególne kursy. Obok danych kursu (temat, liczba godzin) znajduje się imię i nazwisko osoby prowadzącej dane zajęcia. Aby nie było możliwości umieszczenia w bazie błędnych danych (np. kursu bez tematu albo kursu, w którym dla osoby prowadzącej zapomniano podać nazwiska), wszystkie kolumny w tej relacji oznaczono jako obowiązkowe. Może się jednak zdarzyć, że firma zechce zatrudnić nowego pracownika, którego dane powinny się znaleźć w firmowej bazie danych, pomimo że jeszcze nie przypisano mu żadnego kursu do poprowadzenia. Ewentualnie, reagując na potrzebę rynkową, firma utworzy nowy kurs, będzie chciała umieścić go w bazie danych, aby automatycznie pojawiał się w ofercie, pomimo że jeszcze nie zapadła decyzja, który z pracowników poprowadzi ten kurs. W żadnym z wymienionych przypadków nie uda się dodać rekordów do relacji z tabl. 5.9. Nie pozwolą na to ograniczenia integralnościowe, definiujące wszystkie pola jako obowiązkowe. Usunięcie tych ograniczeń umożliwi natomiast dodanie niepoprawnych rekordów, jak to opisano wcześniej. Warto też zauważyć, że relacja tabl. 5.9 jest podatna na anomalię przy modyfikacji (np. zmiana nazwiska po ślubie p. Kwiatkowskiej).

Przedstawiona w tabl. 5.9 relacja `kursy` jest też podatna na *anomalię przy usuwaniu*. Załóżmy, że z pracy w rozważanej firmie zwolnił się p. Józef Nowak, który wg danych z tabl. 5.9 prowadził kurs o nazwie „Bazy danych". Skoro p. Nowak nie pracuje już w firmie, należy usunąć jego dane z bazy. Nie jest to jednak możliwe, ponieważ usunięcie rekordu z danymi p. Nowaka powoduje usunięcie z oferty kursu „Bazy danych". Nie można też zostawić rekordu bez imienia i nazwiska prowadzącego kurs, gdyż nie pozwalają na to opisane wcześniej ograniczenia integralnościowe.

Poprawne rozwiązanie tego problemu wymaga umieszczenia danych o kursach i danych o prowadzących kursy w osobnych relacjach. Ponieważ każdy prowadzący może prowadzić wiele kursów, a każdy kurs ma swojego prowadzącego, związek pomiędzy tymi relacjami będzie typu jeden do wielu. Związek ten powinien być opcjonalny zarówno po stronie prowadzącego, jak i po stronie kursów (prowadzący może jeszcze nie mieć przypisanego żadnego kursu, kurs może jeszcze nie mieć prowadzącego). Związek ten można zrealizować w bazie danych za pomocą klucza obcego w relacji z danymi kursów, dopuszczając dla niego wartość pustą. Każda z relacji może mieć niezależne ograniczenia integralnościowe: wymagane pola `imię` i `nazwisko` dla prowadzących, wymagane pola `temat` oraz liczba godzin dla kursów.

Zrozumiały model i łatwość modyfikacji. Aby zrealizować te dwa łączące się wymagania, należy zadbać, aby związki w bazie danych odpowiadały rzeczywistości. Tworzenie czy pozostawienie w bazie sztucznych powiązań pomiędzy danymi powoduje, że model danych jest trudny do zrozumienia, a jego modyfikacje wymagają na ogół trudnych do przewidzenia zmian w oprogramowaniu korzystającym z bazy danych. Dane, które w rzeczywistości są niezależne (np. email studenta i lista jego ocen), powinny być niezależne także w bazie danych.

Unikanie specjalizacji. Dane z dobrze zaprojektowanej bazy danych można wykorzystywać w różnych celach, patrzeć na nie z różnej perspektywy. Dobrze zaprojektowana baza pozwala wykorzystać dane w sposób, którego nie przewidywał ani zamawiający ją ani jej projektant. Jest to niezwykle istotne wymaganie, gdyż często bazy danych wykorzystywane są przez bardzo długi okres – jak już zaznaczono ich czas życia znacznie przekracza czas życia i eksploatacji korzystających z nich aplikacji. Wykonanie bazy danych zgodnie z regułami pozwala spełnić to wymaganie.

Przykład relacji, w której zapomniano o unikaniu specjalizacji przedstawia tabl. 5.10. Relacja ta odpowiada początkowemu wymaganiu klienta, które można wyrazić stwierdzeniem: *Chcę przechowywać dane studentów i ich oceny, żeby móc policzyć średnią ocen*. Projektant w tym przypadku spełnił dokładnie wymaganie klienta: dla każdego studenta można odczytać oddzieloną przecinkami listę ocen, następnie aplikacja może odczytać z tej listy oceny i policzyć średnią. Dopisanie

Tablica 5.10. Przykład błędnie zaprojektowanej relacji `studenci` do przechowywania danych studentów i ocen

imie	**nazwisko**	**indeks**	**oceny**
Jan	Kowalski	73693	5.0, 2.5, 3.0, 4.5
Józef	Nowak	23098	2.0, 3.5, 4.0, 5.0
Krzysztof	Kwiatkowski	30432	4.5, 5.0
Anna	Nowacka	49754	4.5, 5.0, 5.0
Janina	Staniszewska	82345	5.0, 4.5, 5.0

oceny wymaga dopisania w odpowiednim rekordzie na końcu przecinka i kolejnej oceny.

Można się jednak spodziewać, że klient, który zamówił taką bazę danych, w niedalekiej przyszłości wykorzystać zechce zgromadzone dane także do innych celów. Klient nie ma świadomości, jaka struktura została użyta do przechowywania danych, wie natomiast, że ma bazę danych z danymi studentów i ich ocenami. Dlaczego więc nie mógłby z niej uzyskać dodatkowej informacji: *Chcę mieć listę studentów, którzy mają co najmniej dwie oceny 5.0, żeby przyznać im stypendia za wybitne osiągnięcia*. Z punktu widzenia klienta problem wydaje się prosty: dane są, trzeba je tylko przejrzeć. Niestety realizacja tego zadania w przypadku projektu z tabl. 5.10 nie jest łatwa i dla większej ilości danych może się okazać, że będzie wymagała dużych zasobów obliczeniowych. Projektant bazy nie przewidział bowiem takiego jej zastosowania i baza danych jest specjalizowana do innych celów. Odnalezienie studentów z co najmniej dwoma ocenami 5.0 wymaga przejrzenia całej listy studentów, dla każdego z nich odczytania listy ocen w postaci tekstu, odnalezienia w niej ocen i policzenia liczby ocen 5.0. Dla większej liczby studentów zadanie jest czasochłonne i przy innej reprezentacji na pewno mogłoby być szybciej wykonane.

Tablica 5.11. Poprawne rozwiązanie przykładu z tabl. 5.10 – relacja `studenci`

id	**imie**	**nazwisko**	**indeks**
2	Jan	Kowalski	73693
3	Józef	Nowak	23098
5	Krzysztof	Kwiatkowski	30432
6	Anna	Nowacka	49754
7	Janina	Staniszewska	82345

Tablica 5.12. Poprawne rozwiązanie przykładu z tabl. 5.10 – relacja `oceny`

ocena	student_id
3.5	2
5.0	2
2.0	3
3.0	5
5.0	5
4.0	6
3.0	7
5.0	5

Poprawny projekt wymaga zapisania ocen w osobnej relacji `oceny` połączonej związkiem jeden do wielu z relacją `studenci` za pomocą klucza obcego, jak to pokazano w tabl. 5.11 i 5.12. W takiej sytuacji do wygenerowania listy, której wymaga klient, można wykorzystać system zarządzania bazą danych, wydając mu jedno polecenie. Przetłumaczony na język polski odpowiednik polecenia zwracającego identyfikatory studentów i liczbę ocen 5.0, ale tylko dla studentów, którzy mają co najmniej dwie oceny 5.0, wyglądałoby następująco:

```
1 WYBIERZ id, ilosc(ocena) Z TABELI oceny GDZIE ocena=5.0
2 GRUPUJ PO id MAJACE ilosc(ocena)>=2
```

DBMS wykona zadanie, przeglądając jedynie potrzebne rekordy. Rozbudowując nieco przedstawiony przykład, można łatwo uzyskać polecenie, które zwróci dane studentów zamiast ich identyfikatorów.

Dodatkowe kryteria. Poza wymienionymi wcześniej wymaganiami warto pamiętać o dwóch kryteriach, których spełnienie ułatwia realizację poprawnego projektu. Pierwsze kryterium dotyczy redundancji. Każda informacja zapisywana w bazie danych powinna być zapisana raz – w jednym miejscu. Należy unikać nie tylko przechowywania dokładnie tej samej informacji w wielu miejscach (np. imienia czy nazwiska studenta). Dotyczy to także informacji od siebie zależnych, np. takich, które można wywnioskować bądź wyliczyć z informacji zapisanych w bazie danych. Przykładowo, jeśli baza danych przechowuje wszystkie oceny studenta, to łatwo można z nich wyliczyć średnią ocen. Jeśli więc średnia ocen jest potrzebna korzystającym z bazy danych, zdecydowanie nie należy jej zapisywać w bazie, skoro można ją obliczyć. Powód unikania redundancji jest prosty – utrudnia ona zapewnienie spójności danych. Jeśli w bazie danych byłyby przechowywane zarówno oceny, jak i ich średnia, trzeba byłoby pamiętać o zaktualizowaniu średniej po aktualizacji czy dopisaniu nowych ocen. Baza danych, w której zapisana średnia

nie odpowiadałaby średniej wyliczonej z ocen, byłaby nie tylko bezużyteczna, ale mogłaby wprowadzać w błąd.

Drugie kryterium, które warto spełnić, projektując bazę danych, to zapewnienie niepodzielności przechowywanych w niej atrybutów. Powodem jest fakt, że wyszukiwanie danych na podstawie częściowej wartości atrybutu jest w relacyjnej bazie danych znacznie utrudnione. Przykładowo, jeśli tabela zawiera pole tekstowe, w którym jest przechowywany cały adres pocztowy wraz z kodem, nie będzie możliwe wyszukiwanie w niej danych np. za pomocą kodu pocztowego czy samej tylko ulicy, a w każdym razie będzie to znacznie mniej efektywne. Dlatego warto upewnić się, że w projektowanej bazie nie połączono w jeden atrybut danych, które mogą być traktowane oddzielnie.

Przykładowy projekt

Projektowanie dobrze skonstruowanych relacyjnych baz danych wymaga nieco praktyki. Doświadczenie pozwala unikać błędów, zauważać usterki i braki. W tym rozdziale przedstawiono przykład inżynierskiego podejścia do projektowania bazy danych.

Projektowana baza danych ma służyć do przechowywania informacji o tym, którzy studenci z którymi prowadzącymi mają zajęcia, w jakich terminach i gdzie się te zajęcia odbywają. Analizując problem, najwygodniej jest najpierw wypisać encje, jakie powinny się znaleźć na diagramie danych. Nie musi to być kompletna lista, ważne żeby pozwalała rozpocząć projektowanie. W tym przypadku z całą pewnością możemy stwierdzić, że w bazie należy przechowywać dane o studentach, przedmiotach, prowadzących i salach. Stąd na pierwszym diagramie można umieścić encje:

- *student*,
- *przedmiot*,
- *prowadzący*,
- *sala*.

Taki diagram przedstawia rys. 5.7.

Kolejnym krokiem jest zastanowienie się, jakie związki występują pomiędzy zaznaczonymi encjami i oznaczenie ich na diagramie. W tym wypadku należy stwierdzić, że:

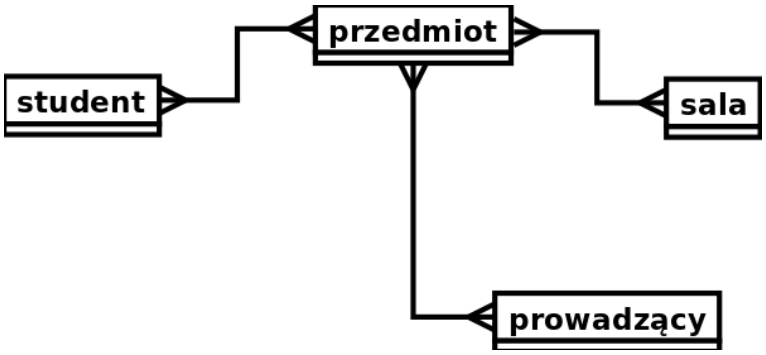
- każdy student uczęszcza na wiele przedmiotów oraz na każdy z przedmiotów uczęszcza wielu studentów,

Rys. 5.7. Pierwsza wersja ERD zawierająca jedynie encje

- prowadzący zajęcia mogą prowadzić wiele przedmiotów, każdy przedmiot jest prowadzony przez wielu prowadzących (wykłady, laboratoria, ćwiczenia, projekty),
- zajęcia z każdego przedmiotu odbywają się w wielu salach, a każda z sal może potencjalnie służyć do zajęć z wielu przedmiotów,
- sale są powiązane z prowadzącymi jedynie poprzez fakt prowadzenia przez daną osobę przedmiotu w danej sali, nie ma więc bezpośredniego powiązania pomiędzy prowadzącymi a salami,
- studenci i prowadzący są powiązani jedynie poprzez to, że studenci uczestniczą w przedmiocie prowadzonym przez danego pracownika, nie ma więc także bezpośredniego powiązania pomiędzy studentami a prowadzącymi.

Z tego wynika, że pomiędzy encjami z diagramu na rys. 5.7 występują trzy związki wiele do wielu przedstawione na diagramie z rys. 5.8. Ten etap projektowania zwykle realizuje się, analizując istniejący diagram. Krok po kroku zaznacza się związki pomiędzy encjami, bezpośrednio po zidentyfikowaniu, że dany związek ma miejsce.

Kolejnym krokiem, który można podjąć, jest usunięcie związków wiele do wielu. Pomaga to doprecyzować projekt i ewentualnie zidentyfikować kolejne encje. Aby usunąć związek wiele do wielu pomiędzy encjami *prowadzący* i *przedmiot*, należy – jak to opisano wcześniej – wprowadzić encję pośredniczącą. Aby określić nazwę dla tej encji, warto zastanowić się, jakie praktyczne znaczenie ma powiązanie prowadzącego z przedmiotem. Łatwo zauważyć, że w rzeczywistości powiązanie to oznacza, że dany prowadzący prowadzi *zajęcia* z danego przedmiotu, i tak zapewne należy nazwać dodawaną encję. Zajęcia dotyczą jednego konkretnego przedmiotu i są prowadzone przez konkretnego prowadzącego, ale zarówno jeden prowadzący może prowadzić wiele zajęć, jak i z jednego przedmiotu

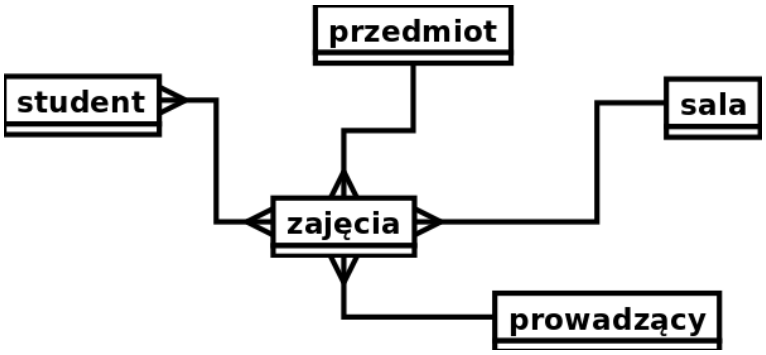


Rys. 5.8. Początkowa wersja ERD z zaznaczonymi związkami pomiędzy encjami

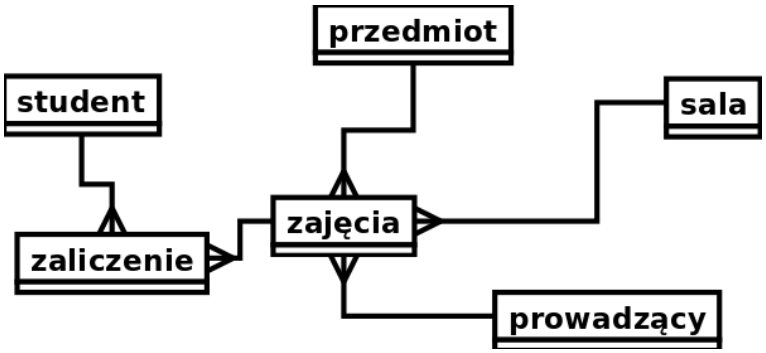
może się odbywać wiele zajęć. Dlatego też zajęcia będą powiązane związkami wiele do jednego z przedmiotem i z prowadzącym. Próbując poprawić diagram na tym etapie, szybko dochodzimy do wniosku, że zajęcia są powiązane nie tylko z prowadzącym i przedmiotem. Powiązanie studenta z przedmiotem w rzeczywistości oznacza, że dany student uczestniczy w *zajęciach* z przedmiotu. Należy więc przenieść związek spomiędzy encji *student* i *przedmiot* pomiędzy encję *student* i nową encję *zajęcia*. Jednak w tym przypadku nowy związek będzie miał nieco inny charakter niż w przypadku prowadzącego. Każdy student uczestniczy w wielu zajęciach i w jednych zajęciach uczestniczy wielu studentów. Związek ten będzie więc nadal miał krotność wiele do wielu. Sytuacja ze związkiem pomiędzy salą a przedmiotem jest podobna do tej, którą analizowano dla związku prowadzącego z przedmiotem. Ten związek także należy przenieść na związek jeden do wielu pomiędzy encją *sala* a encją *zajęcia* (konkretne zajęcia odbywają się w jednej sali). Zmodyfikowany diagram przedstawia rys. 5.9. Próba usunięcia jednego związku wiele do wielu doprowadziła do dodatkowych zmian, które pozwoliły zauważyć braki i doprecyzować model danych.

Kolejnym krokiem, jaki można wykonać, jest usunięcie z diagramu związku wiele do wielu występującego pomiędzy encjami *student* i *zajęcia*. Wprowadzona encja pośrednicząca będzie reprezentowała związek studenta z zajęciami. W rzeczywistości związek ten skutkuje koniecznością uzyskania przez studenta zaliczenia z określonych zajęć, dobrym rozwiązaniem wydaje się więc nazwanie encji pośredniczącej *zaliczenie*. Kolejna wersja diagramu znajduje się na rys. 5.10.

Diagram z rys. 5.10 nie zawiera już związków wiele do wielu. Na tym etapie wygodnie będzie się zastanowić nad opcjonalnością poszczególnych związków i zaktualizować ją na diagramie. Zacznijmy od lewej strony diagramu: student może mieć wiele zaliczeń, ale nie musi mieć żadnego – może nie być przypisany do żadnych zajęć, np. zaraz po przyjęciu na studia czy będąc na urlopie dziekańskim. Zaliczenie natomiast, jeśli tylko istnieje, musi być przypisane do studenta. Związek



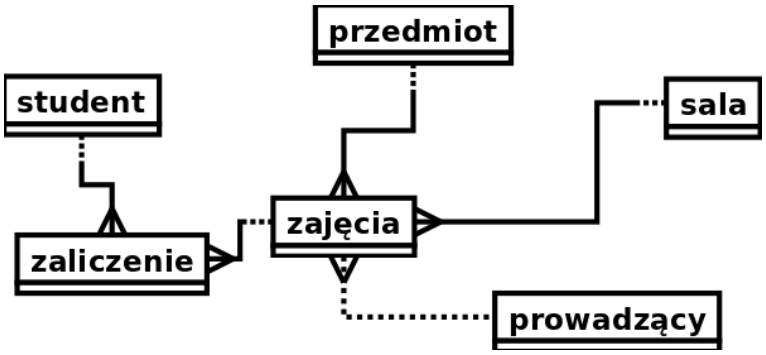
Rys. 5.9. ERD z zaznaczonymi związkami pomiędzy encjami oraz usuniętym związkiem wiele do wielu pomiędzy encjami *prowadzący* i *przedmiot*



Rys. 5.10. ERD, w którym usunięto wszystkie związki wiele do wielu

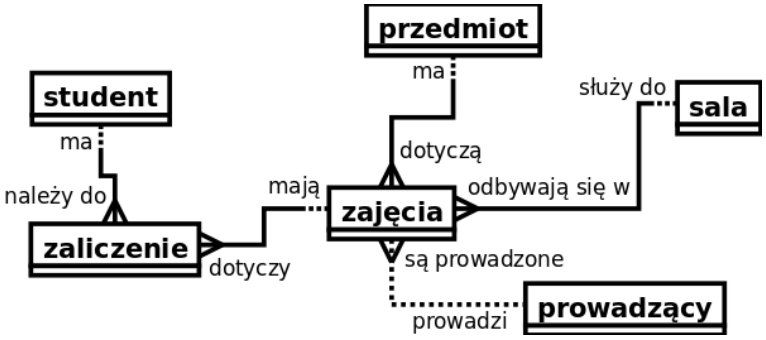
pomiędzy encjami *student* i *zaliczenie* będzie więc opcjonalny po stronie studenta (student może mieć zaliczenie) i obowiązkowy po stronie zaliczenia (zaliczenie musi należeć do studenta). Może się zdarzyć, że istnieją zajęcia, do których nie przypisano jeszcze żadnych studentów (żaden student nie ma obowiązku uzyskania zaliczenia z tych zajęć). Każde z istniejących zaliczeń musi jednak dotyczyć konkretnych zajęć. Związek encji *zaliczenie* i *zajęcia* będzie więc opcjonalny po stronie encji *zajęcia* i obowiązkowy po stronie encji *zaliczenie*. Zajęcia muszą dotyczyć konkretnego przedmiotu, może się jednak zdarzyć, że z jakiegoś przedmiotu nie ma jeszcze zdefiniowanych zajęć. Związek *zajęcia – przedmiot* będzie więc opcjonalny po stronie przedmiotu i obowiązkowy po stronie zajęć. Zajęcia powinny mieć zawsze przypisaną salę, w której się odbywają, może się jednak zdarzyć, że w jakiejś sali nie odbywają się żadne zajęcia. Związek pomiędzy encjami *sala* i *zajęcia* będzie więc opcjonalny po stronie encji *sala* i obowiązkowy po stronie encji *zajęcia*. Możliwe jest, że wystąpi konieczność zdefiniowania zajęć, do których nie przypisano jeszcze żadnego prowadzącego. Może też zaistnieć sytuacja, w której

prowadzący nie ma jeszcze przypisanych żadnych zajęć. Stąd związek *prowadzący – zajęcia* jest opcjonalny po obu stronach. Diagram uwzględniający opcjonalność związków przedstawiono na rys. 5.11.



Rys. 5.11. ERD, w którym uwzględniono opcjonalność związków

Na tym etapie wygodnie będzie zanotować znaczenie poszczególnych związków, dodając do nich opisy. Ułatwi to późniejszą interpretację związków pomiędzy encjami. Diagram z opisami przedstawiono na rys. 5.12

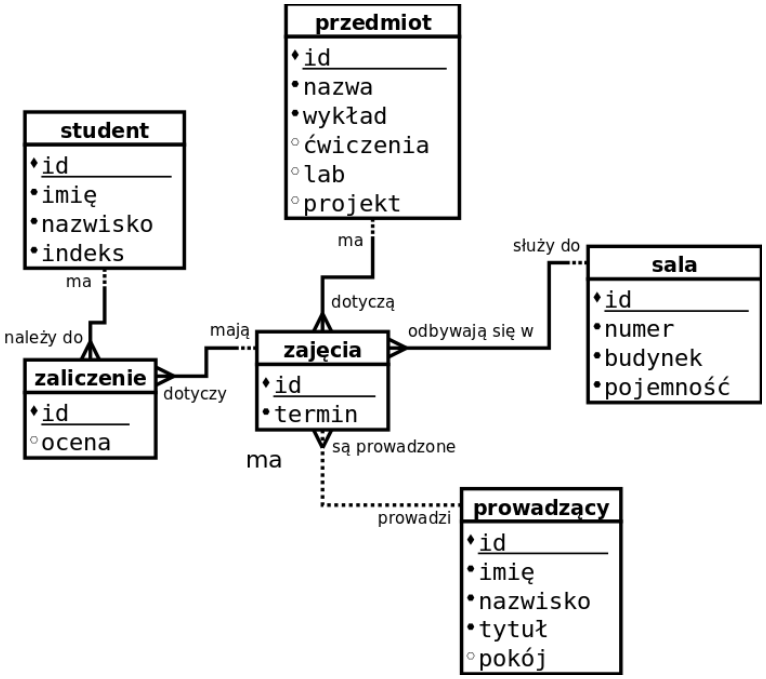


Rys. 5.12. ERD wraz z opisami związków

Aby zweryfikować, czy projektowana baza będzie mogła przechowywać wszystkie wymagane dane, konieczne jest dodanie atrybutów do poszczególnych encji. Diagram z atrybutami przedstawiono na rys. 5.13. Jak widać, encje dodane w czasie usuwania związków wiele do wielu okazały się użyteczne – znalazły swoją interpretację w modelowanej dziedzinie i umieszczono w nich atrybuty: *ocena* jest atrybutem *zaliczenia* uzyskanego przez danego *studenta* z konkretnych *zajęć*. *Termin* jest atrybutem *zajęć* prowadzonych z danego *przedmiotu* przez konkretnego *prowadzącego* w określonej *sali* dla grupy *studentów*. W encji *przedmiot* znalazły się atrybuty *wykład*, *ćwiczenia*, *lab* i *projekt* służące do zapisania liczby

godzin przeznaczonych na poszczególne formy zajęć z tego przedmiotu. Większość atrybutów na diagramie jest obowiązkowa. Wyjątek stanowią:

- ocena z zaliczenia – na początku student nie ma oceny z zaliczenia,
- liczba godzin ćwiczeń, laboratoriów oraz projektu dla przedmiotu – przedmiot może nie przewidywać danej formy zajęć,
- numer pokoju prowadzącego.

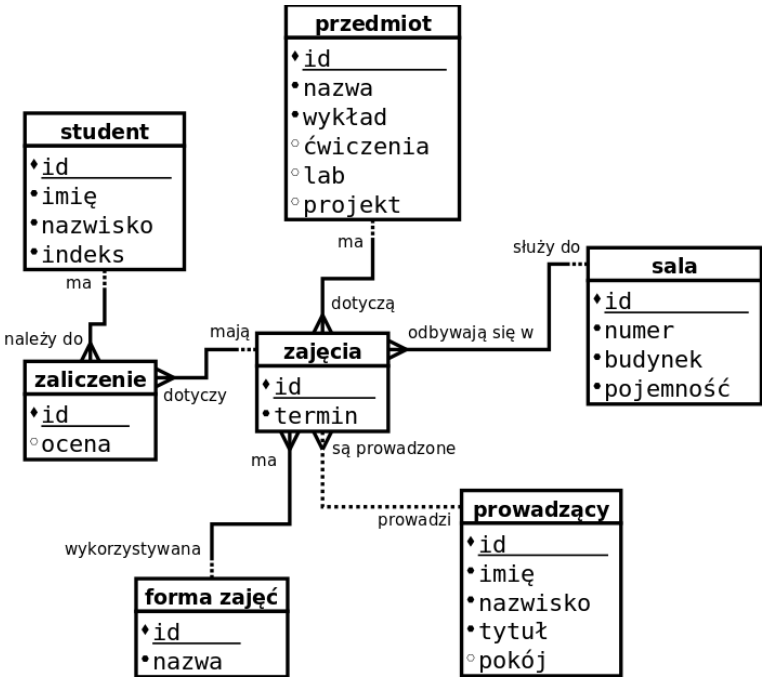


Rys. 5.13. ERD z atrybutami

Na tym etapie należy przeanalizować powstały projekt i sprawdzić, czy wszystkie wymagane dane znajdują w nim swoje miejsce. Jeśli jakichś danych brakuje, należy dodać kolejne atrybuty do istniejących encji bądź dodać nowe encje. Analiza taka zwykle wymaga konsultacji z klientem bądź analizy dokumentów utworzonych w trakcie rozmów z nim. Po uważnym przejrzeniu diagramu z rys. 5.13 można zauważyć, że nie ma na nim miejsca do przechowywania informacji o formie zajęć – nie wiadomo, czy dane zajęcia to wykład, ćwiczenia, laboratoria czy projekt[5].

[5] Atrybuty encji *przedmiot* mówią tylko o liczbie godzin z danego przedmiotu, z którego może być prowadzonych wiele form zajęć. Nadal jednak nie wiadomo nic na temat formy zajęć, której odpowiada zapis w encji *zajęcia*.

Konkretne zajęcia zawsze mają jedną formę, ale jedna forma zajęć jest realizowana w ramach różnych przedmiotów, związek pomiędzy formą zajęć a zajęciami jest więc typu jeden do wielu. Aby umożliwić przechowywanie tej informacji, należy dodać kolejną encję *forma zajęć* powiązaną związkiem jeden do wielu z encją *zajęcia*. Każde zajęcia muszą mieć określoną formę, ale może się zdarzyć, że jedna z form zajęć nie jest w ogóle wykorzystywana w ramach studiów (np. konwersatorium na studiach technicznych czy laboratoria na humanistycznych). Dlatego związek ten będzie opcjonalny po stronie encji *forma zajęć* i obowiązkowy po stronie encji *zajęcia*. Rozszerzony diagram przedstawiono na rys. 5.14.



Rys. 5.14. ERD z dodaną encją *forma zajęć*

Jeśli projekt przedstawiony na diagramie pozwala na przechowywanie wszystkich istotnych informacji, należy rozważyć jeszcze jedną kwestię. Wszystkie atrybuty encji muszą być niepodzielne. W przypadku pewnego rodzaju atrybutów na ogół ich niepodzielność nie ulega wątpliwości: nazwisko, imię, numer indeksu w większości przypadków można uznać za atrybuty niepodzielne. Natomiast różnie można interpretować atrybut służący do zapisania adresu zamieszkania. Często osobno zapisuje się każdą z linijek adresu umieszczanego na kopercie: ulica i numer domu, kod pocztowy i miasto. To wystarcza do wygodnego, automatycznego zaadresowania korespondencji. Czasami jednak sposób, w jaki są wykorzystywane dane, wymaga dodatkowego podziału. Kod pocztowy można zapisać w osobnym

polu, aby łatwo prowadzić statystyki regionów, z których pochodzą poszczególne osoby, numery budynków i mieszkań pozwalają wykorzystać automatyczną lokalizację za pomocą serwisów udostępniających mapy. Należy więc rozważyć i jeśli potrzeba skonsultować z klientem kwestię niepodzielności atrybutów.

W przedstawionym przykładzie znajduje się przynajmniej jeden atrybut, który powinien budzić wątpliwości. Tym atrybutem jest *termin* z encji *zajęcia*. Nie jest łatwo określić, w jakiej formie termin powinien być zapisywany:

- zajęcia mogą się odbywać regularnie co tydzień, wtedy wystarczy zapisać dzień tygodnia i godzinę,
- zajęcia mogą się odbywać co drugi tydzień, wtedy należy zapisać także informację, w których tygodniach się odbywają,
- zajęcia mogą się także odbywać nieregularnie, wtedy należałoby rozważyć możliwość zapisania listy terminów, w których mają się odbywać, prawdopodobnie wymagałoby to dodania kolejnych encji.

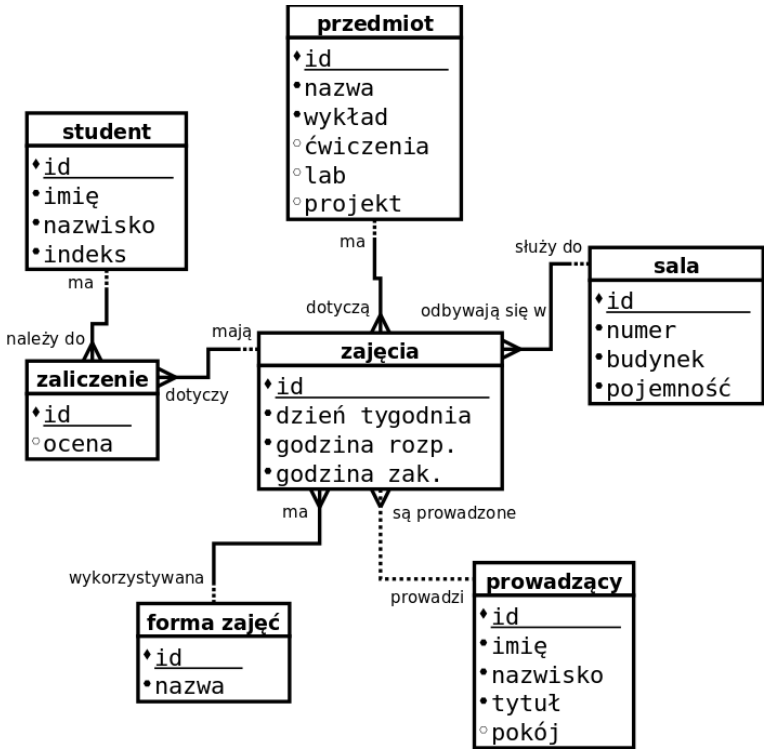
W analizowanym przypadku zdecydowano się na stosunkowo proste rozwiązanie pozwalające zapisać informacje o zajęciach, które odbywają się regularnie co tydzień. Poprawiony diagram przedstawiono na rys. 5.15.

Przedstawione w tym rozdziale kroki powtarza się do skutku, poprawiając i rozbudowując projekt tak długo, aż spełni on zarówno wymagania klienta jak i wymagania poprawnie zaprojektowanej bazy danych (podrozdział 5.6.4). Kroki te można wykonywać i powtarzać w różnej kolejności. Ostateczna decyzja o poprawności wykonanego projektu wymaga przemyślenia, a także nabytego z czasem doświadczenia. Często dobrą praktyką jest skonsultowanie wykonanego projektu z inną osobą, która ma możliwość odnalezienia przeoczonych przez projektanta usterek.

Na rynku są dostępne narzędzia, które pozwalają zaprojektować bazę danych w formie diagramu, a następnie automatycznie na jego podstawie utworzyć taką bazę na wyznaczonym serwerze bądź wygenerować skrypt, który ją utworzy. Ułatwia to znacząco pracę i uwalnia projektanta od konieczności żmudnego tłumaczenia diagramu na schemat bazy danych.

Normalizacja

Ogół działań, które wykonuje się na projekcie bazy danych, aby zapewnić, że nie będzie on powodował anomalii, nazywa się *normalizowaniem* bazy danych. Określenie to ma znaczenie formalne – można matematycznie dowieść, czy zaprojektowana baza danych spełnia kryteria wybranej *postaci normalnej*. Istnieje sześć postaci normalnych nazywanych kolejnymi numerami od 1NF (ang. *First*



Rys. 5.15. ERD z dodaną encją *forma zajęć*

Norlmal Form) do 6NF, poza nimi istnieją inne, dodatkowe, jak choćby BCNF (ang. *Boyce-Codd normal form*). W praktyce jednak w projektowaniu i normalizowaniu baz danych używa się reguł inżynierskich, które pozwalają zapewnić poprawność projektu i nie dowodzi się jej za pomocą metod formalnych.

5.6.5. Wykorzystanie baz danych i SQL

Zarządzaniem bazami danych zajmują się systemy zarządzania bazą danych opisane w podrozdziale 5.3. To z nimi komunikują się programy korzystające z danych zapisanych w bazie, wydając polecenia mówiące, co należy zrobić z danymi bądź całą bazą danych. Do wydawania tych poleceń służy SQL – język zarządzania bazą danych.

SQL – specyfika języka

Structured Query Language (SQL) to język, który pozwala na zarządzanie bazami danych oraz zapisanymi w nich danymi. Jest to język specjalizowany do tego celu i w przeciwieństwie do wielu innych języków programowania nie pozwala

np. na zapis algorytmów. Jest językiem deklaratywnym, co oznacza, że opisuje się nim oczekiwany rezultat (np. jakie dane, w jakim zestawieniu uzyskać), a nie sposób jego osiągnięcia (w jaki sposób odnaleźć i zwrócić oczekiwane zestawienie). Fakt ten sprawia, że operowanie na bazach danych jest bardzo wygodne i niezależne od szczegółów związanych z przechowywaniem danych (tj. od modelu fizycznego opisanego w podrozdziale 5.5). Za sposób realizacji zapytania SQL odpowiada DBMS, programista musi jedynie wyspecyfikować, jakie dane chce uzyskać.

SQL jest opisany standardem opublikowanym przez ISO pod numerem ISO/IEC 9075:2011 (ostatnia wersja pochodzi z 2011 roku) [17]. W większości aspektów producenci systemów zarządzania bazami danych dostosowują swoje produkty do tego standardu. Niestety pod pewnymi względami rozwiązania różnych producentów różnią się między sobą i nie odpowiadają standardom. Można się więc spotkać z pewnymi specyficznymi wyrażeniami SQL, które działają np. tylko na serwerach firmy Oracle. Niespójności te dotyczą pewnych szczegółów, w ogólności jednak należy uznać SQL za język dobrze ustandaryzowany także w praktyce. Przykłady zawarte w tym rozdziale były testowane na bazie PostgreSQL[6].

Zgodnie ze standardem wielkość liter w poleceniach SQL nie ma znaczenia. Niekiedy tylko w niektórych systemach zarządzania bazami danych zdarza się, że wielkość liter (np. w nazwach relacji czy kolumn) jest istotna. Taka sytuacja występuje w użytej tutaj bazie PostgreSQL, co opisano w podrozdziale 5.6.8. Z tego powodu wszystkie nazwy tabel i kolumn w przykładach SQLa będą pisane małymi literami.

SQL składa się z czterech części, każda z nich odpowiada za innego rodzaju operacje:

- DDL (ang. *Data Definition Language*) – pozwala manipulować bazą danych i jej schematem,
- DCL (ang. *Data Control Language*) – odpowiada za definicje związane z kontrolą dostępu do danych,
- DML (ang. *Data Manipulation Language*) – pozwala manipulować danymi zapisanymi w bazie,
- DQL (ang. *Data Query Language*) – służy do uzyskiwania danych z bazy.

W kolejnych akapitach opisano sposób wykorzystania poszczególnych elementów SQLa. Nie są one szczegółowym opisem języka, gdyż jest on bardzo rozbudowany. Składnię i znaczenie szczegółowo opisują dokumentacje poszczególnych DBMS.

[6] `http://www.postgresql.org/`

Tutaj można znaleźć wprowadzenie i przykłady pozwalające zrozumieć zasadę działania języka.

Z każdym z serwerów baz danych można połączyć się za pomocą tak zwanej „konsoli". Konsola jest programem tekstowym albo graficznym, który pozwala na bezpośrednie wykonywanie poleceń SQL. Na ogół polecenia wpisane w konsoli należy zakończyć znakiem średnika – samo naciśnięcie klawisza Enter nie powoduje uruchomienia polecenia, dzięki czemu możliwe jest wprowadzanie pojedynczego polecenia w wielu liniach. W SQL znak końca linii nie kończy więc polecenia. Stąd też w podanych przykładach polecenia będą w razie potrzeby dzielone na wiele linii i kończone znakiem średnika.

Do testowania zapytań SQL i nauki tego języka wystarczy zainstalowanie jednego z systemów zarządzania bazą danych (np. jednego z wymienionych w podrozdziale 5.6.8). Jego konsolę można wykorzystać do utworzenia testowej bazy danych i wydawania innych poleceń.

DDL – tworzenie schematu bazy danych

DDL (ang. *Data Definition Language*) jest częścią SQLa służącą do definiowania schematu bazy danych. Służy on między innymi do:

- tworzenia nowych baz danych,
- usuwania istniejących baz danych,
- tworzenia relacji (tabel) wraz z kolumnami,
- usuwania relacji,
- modyfikowania istniejących relacji (np. przez zmianę, usunięcie bądź dodanie kolumn),
- definiowania ograniczeń integralnościowych dla danych zapisywanych w relacji.

Polecenia tej części SQLa zaczynają się od jednego z trzech słów:

- `CREATE` – tworzenie struktur,
- `ALTER` – modyfikacja struktur,
- `DROP` – usuwanie struktur.

Kolejne słowo użyte po słowie `CREATE` decyduje o tym, co zostanie utworzone. Przykładowo, aby utworzyć nową bazę danych należy użyć polecenia zaczynającego się od `CREATE DATABASE`, jak w przykładzie z listingu 5.1. Polecenie to spowoduje utworzenie nowej bazy danych o nazwie `skrypt`.

Listing 5.1
```
1 CREATE DATABASE skrypt;
```

Przykład tworzenia nowej relacji (tabeli) w bazie danych przedstawia listing 5.2. Tworzona jest tabela o nazwie `studenci` odpowiadająca przykładowi z tabl. 5.4. Pierwszą kolumną w tej tabeli jest `id` będąca kluczem głównym (`PRIMARY KEY`). Użyty typ `SERIAL` powoduje, że przy dodawaniu nowego rekordu do tabeli, jeśli wartość pola `id` nie zostanie podana, zostanie domyślnie przyjęta kolejna wartość naturalna. Jest to wygodne rozwiązanie powalające łatwo zapewnić unikalne wartości dla klucza głównego. Tabela zawiera pola `imie` i `nazwisko` typu tekstowego o zmiennej długości (`VARCHAR`), które nie mogą być puste (`NOT NULL`), oraz pole `imie2` tego samego typu, które można pozostawić puste podczas dodawania nowych rekordów (brak deklaracji `NOT NULL`). Ostatnie pole to `indeks` typu całkowitego (`INT`) przeznaczone na numer indeksu studenta. To pole także nie może być puste. Definicja `UNIQUE` oznacza, że numer indeksu musi być unikalny. DBMS nie pozwoli dodać do tej tabeli nowego rekordu ani wykonać aktualizacji danych, jeśli numer indeksu w zapisywanym rekordzie będzie taki sam, jak w innym rekordzie w tej tabeli.

Listing 5.2
```
1 CREATE TABLE studenci (
2   id SERIAL PRIMARY KEY,
3   imie VARCHAR NOT NULL,
4   imie2 VARCHAR,
5   nazwisko VARCHAR NOT NULL,
6   indeks INT NOT NULL UNIQUE
7 );
```

Aby utworzyć tabelę `stypendia` z przykładu tabl. 5.5, należy użyć przykładu z listingu 5.3. Poza kolumnami z tabl. 5.5 tabela ta będzie zawierała także kolumnę `id` będącą jej kluczem głównym, co znacznie ułatwi operacje na rekordach tej tabeli. Kolumna `student_id` typu całkowitego (nazwy `INT` oraz `INTEGER` mogą być używane zamiennie) jest kluczem obcym pochodzącym z tabeli `studenci`. Informuje o tym deklaracja `REFERENCES studenci(id)` definiująca, jakiej kolumnie z jakiej tabeli powinna odpowiadać wartość w kolumnie `student_id`. Ponieważ pole to jest obowiązkowe (`NOT NULL`), nie będzie możliwe dodanie do tabeli informacji o stypendium bez powiązania jej ze studentem. Polem obowiązkowym tej tabeli jest też `kwota`. Typ tego pola służy do przechowywania wartości dziesiętnych. Wartości te będą przechowywane dokładnie, bez zaokrągleń, ze zdefiniowaną precyzją. W tym wypadku dokładność będzie wynosiła dwa miejsca po przecinku, przy czym liczba może mieć najwyżej 6 cyfr znaczących (wraz z cyframi po przecinku). Operacje liczbowe na liczbach tego typu są często znacznie wolniejsze niż dla typów całkowitych czy zmiennoprzecinkowych (typy

liczbowe omówiono w rozdziale 3. pozycji [40]), ale dzięki temu, że przechowywane wartości są dokładne typ ten nadaje się doskonale do przechowywania wartości monetarnych. Poprawna wartość w tej kolumnie musi spełniać warunek określony w nawiasach po słowie `CHECK`. Pozwala to zapewnić, że zapisana w bazie kwota stypendium będzie większa od zera. Niespełnienie warunku przy aktualizacji danych bądź dodawaniu nowego rekordu spowoduje wystąpienie błędu, w rezultacie czego nowe dane nie zostaną zapisane. Aby komunikat błędu łatwiej było rozszyfrować, warto nadać nazwę definiowanemu warunkowi, tak jak na listingu 5.4. Sprawdzane warunki można definiować dla różnych pól i mogą one być znacznie bardziej złożone. Ograniczenia mogą też być definiowane poza definicją kolumny i mogą obejmować wartości wielu kolumn. Przykładowo, można zapewnić, że zapisana w bazie płaca dodatkowa będzie mniejsza od zapisanej w niej płacy podstawowej. Opis składni i przykłady definiowania takich warunków można znaleźć w dokumentacji DBMS. Ostatnim polem jest `rodzaj` – obowiązkowe pole typu tekstowego.

Listing 5.3
```
1 CREATE TABLE stypendia (
2   id SERIAL PRIMARY KEY,
3   student_id INTEGER NOT NULL REFERENCES studenci(id),
4   kwota DECIMAL(6,2) NOT NULL CHECK(kwota > 0),
5   rodzaj VARCHAR NOT NULL
6 );
```

Listing 5.4
```
1 CREATE TABLE stypendia (
2   id SERIAL PRIMARY KEY,
3   student_id INTEGER NOT NULL REFERENCES studenci(id),
4   kwota DECIMAL(6,2) NOT NULL
5             CONSTRAINT kwota_dodatnia CHECK(kwota > 0),
6   rodzaj VARCHAR NOT NULL
7 );
```

DDL pozwala nie tylko tworzyć, ale także modyfikować istniejące struktury. Przykładowo, aby dodać pole `email` do tabeli studenci, należy użyć polecenia `ALTER` przedstawionego na listingu 5.5.

Listing 5.5
```
1 ALTER TABLE studenci ADD COLUMN email VARCHAR;
```

Polecenie `DROP` służy do usuwania struktur. Aby usunąć utworzoną wcześniej tabelę `stypendia`, należy użyć polecenia z listingu 5.6. Polecenie to spowoduje usunięcie tabeli wraz z zawartymi w niej danymi! Aby usunąć całą bazę danych (wraz ze wszystkimi tabelami i danymi), należy użyć polecenia z listingu 5.7.

Listing 5.6
```
1 DROP TABLE stypendia;
```

Listing 5.7
```
1 DROP DATABASE skrypt;
```

DDL jest używany na ogół do tworzenia bazy danych i jej schematu na samym początku. Czasami później używa się go do modyfikacji już istniejącej bazy, zwłaszcza gdy zaistnieje potrzeba jej rozbudowania. Ta część SQLa jest wykorzystywana głównie przez twórców bazy danych bądź osobę, która tę bazę danych instaluje. Polecenia te nie są wykorzystywane na co dzień przez użytkowników.

DCL – kontrola dostępu do danych

Serwery relacyjnych baz danych pozwalają bardzo szczegółowo zdefiniować uprawnienia poszczególnych użytkowników do wykonywania operacji na danych. Możliwe do nadania bądź odebrania uprawnienia to między innymi prawo do:

- łączenia się z bazą danych,
- tworzenia tabel w bazie danych,
- czytania rekordów,
- dodawania rekordów,
- aktualizacji istniejących rekordów,
- usuwania rekordów.

Uprawnienia można nadawać zarówno do baz danych, jak i do tabel, a nawet poszczególnych kolumn. Można je nadać użytkownikom, ale także rolom (grupom użytkowników).

Ta część SQLa składa się z poleceń rozpoczynających się od jednego z dwóch słów:

- `GRANT`
- `REVOKE`

Uprawnienia nadaje się za pomocą polecenia `GRANT`. Przykład nadający prawo odczytywania danych (`SELECT`) z tabeli `studenci` użytkownikowi `dziekanat` przedstawia listing 5.8.

Listing 5.8
```
1 GRANT SELECT ON studenci TO dziekanat;
```

Aby odebrać nadane uprawnienia należy użyć polecenia `REVOKE`. Przykład z listingu 5.9 odbiera uprawnienia do aktualizacji danych (`UPDATE`) w tabeli `stypendia` użytkownikowi `wykladowca`.

Listing 5.9
```
1 REVOKE UPDATE ON stypendia FROM wykladowca;
```

DCL używają głównie administratorzy baz danych. W najprostszych zastosowaniach relacyjnych baz danych w zupełności wystarczają ustawienia domyślne. W bardziej złożonych uprawnienia są nadawane w momencie instalowania oprogramowania i bazy danych. Są jednak instalacje, w których wymagania dotyczące bezpieczeństwa powodują konieczność bardzo precyzyjnego określenia uprawnień różnych użytkowników do różnych części bazy danych czy też różnych baz danych.

DML – manipulowanie danymi

DML jest częścią SQLa służącą do modyfikowania danych zapisanych w bazie. Operacje, jakie można wykonywać za pomocą poleceń DML, to:

- dodawanie nowych rekordów,
- usuwanie rekordów,
- modyfikacja danych w istniejących rekordach.

Składa się on z poleceń zaczynających się od jednego z trzech słów:

- `INSERT`
- `DELETE`
- `UPDATE`

Do dodawania rekordów służy polecenie `INSERT`. Pozwala ono dodać nowy rekord, wypełniając go podanymi danymi, tak jak na listingu 5.10. Po słowach `INSERT INTO` należy podać nazwę tabeli, do której ma być dodany rekord, a następnie w nawiasach listę kolumn, które mają być wypełnione. Po liście kolumn występuje słowo `VALUES` i w nawiasach lista wartości dla każdej z wyspecyfikowanych kolumn. Oczywiście podane wartości muszą odpowiadać typom poszczególnych kolumn. Wykonanie polecenia nie powiedzie się, jeśli w poleceniu `INSERT` zostaną pominięte kolumny, których wypełnienie jest obowiązkowe (oznaczone jako `NOT NULL` podczas tworzenia tabeli), bądź jeśli podane wartości nie będą spełniać ograniczeń integralnościowych (odpowiednich wartości kluczy obcych, warunków zdefiniowanych za pomocą `CHECK`).

Listing 5.10
```
1 INSERT INTO stypendia (student_id, kwota, rodzaj)
2   VALUES (10, 430.00, 'naukowe');
```

Przykład z listingu 5.10 dodaje do tabeli `stypendia` rekord, wypełniając pole `student_id` liczbą całkowitą `10`, pole `kwota` liczbą dziesiętną `430.00`, pole `rodzaj` tekstem `naukowe`. Polecenie nie ustawia wartości pola `id`, wartość ta zostanie więc ustawiona automatycznie, dzięki temu, że dla tej kolumny zadeklarowano typ `SERIAL` (zob. podrozdział 5.6.5). Pominięcie nieobowiązkowej kolumny innego typu spowoduje wpisanie w niej wartości `NULL`. Aby polecenie to powiodło się, w tabeli `student` musi istnieć rekord, w którym pole `id` ma wartość `10` (taką jak wartość pola `student_id` w dodawanym rekordzie) – dodawane stypendium musi być przypisane do istniejącego studenta.

Aby usunąć rekordy z tabeli `stypendia`, należy użyć polecenia `DELETE FROM`, jak na listingu 5.11. To polecenie usunie wszystkie rekordy z tabeli! Aby usunąć jedynie niektóre rekordy, należy na końcu polecenia dodać słowo `WHERE` i warunek, który muszą spełniać usuwane rekordy, co przedstawia listing 5.12. Polecenie to spowoduje usunięcie z tabeli wszystkich studentów o imieniu Jan i nazwisku Kowalski. Aby mieć pewność, że usunięty zostanie dokładnie jeden rekord, najlepiej użyć w warunku klucza głównego tabeli, tak jak na listingu 5.13.

Listing 5.11
```
1 DELETE FROM studenci;
```

Listing 5.12
```
1 DELETE FROM studenci
2   WHERE imie = 'Jan' AND nazwisko = 'Kowalski';
```

Listing 5.13
```
1 DELETE FROM studenci WHERE id = 1;
```

Do modyfikowania danych zapisanych w bazie służą polecenia zaczynające się od słowa `UPDATE`. Przykład z listingu 5.14 spowoduje ustawienie w tabeli `stypendia` kolumny `kwota` na wartość `4000` we wszystkich rekordach. Aby zmienić kwotę tylko w wybranych rekordach, na końcu polecenia należy dodać słowo `WHERE` i warunek jaki muszą spełniać modyfikowane rekordy, tak jak na listingu 5.15. Polecenie to spowoduje ustawienie kolumny `kwota` tylko w rekordach, w których pole `kwota` ma wartość `100`. Stypendium podniesione do kwoty `4000` będą mieć zatem studenci, którzy mają stypendium w wysokości `100`.

Listing 5.14
```
1 UPDATE stypendia SET kwota = 4000;
```

Listing 5.15
```
1 UPDATE stypendia SET kwota = 4000
2   WHERE kwota = 100;
```

Wartości w rekordach można ustawiać także w zależności od wartości tych lub innych pól w rekordach. Można na przykład dwukrotnie zwiększyć wybrane stypendia za pomocą polecenia z listingu 5.16. Zmodyfikowane zostaną rekordy spełniające warunek po słowie `WHERE`, a więc te, dla których kwota stypendium jest mniejsza niż 1000.

Listing 5.16
```
1 UPDATE stypendia SET kwota = kwota*2
2   WHERE kwota < 1000;
```

SQL pozwala na modyfikację wielu kolumn za pomocą pojedynczego polecenia. Przykład takiego polecenia pokazano na listingu 5.17. Polecenie to spowoduje w tabeli `studenci` ustawienie pola `imie2` na wartość `Krzysztof`, a pola `nazwisko` na wartość `Nowakowski`, ale tylko w rekordach, w których pole `imie` ma wartość `Józef`, a pole `nazwisko` wartość `Nowak`. Oczywiście może się zdarzyć, że rekordów spełniających taki warunek będzie wiele, wtedy zmodyfikowane zostaną wszystkie pasujące. Aby mieć pewność, że zmodyfikowany zostanie jeden wybrany rekord, w warunku najlepiej użyć klucza głównego, jak to pokazano na listingu 5.18.

Listing 5.17
```
1 UPDATE studenci SET imie2 = 'Krzysztof', nazwisko = 'Nowakowski'
2   WHERE imie = 'Józef' AND nazwisko = 'Nowak';
```

Listing 5.18
```
1 UPDATE studenci SET imie2 = 'Krzysztof', nazwisko = 'Nowakowski'
2   WHERE id = 3;
```

Oczywiście warunki określające, które rekordy powinny być zmodyfikowane, mogą być znacznie bardziej złożone niż w przedstawionych przykładach. Można w nich używać operatorów logicznych w postaci słów `AND`, `OR` oraz `NOT`, porównań oraz nawiasów. Dokładną specyfikację można znaleźć w dokumentacjach do serwerów baz danych.

DQL – zapytania

Wyrażenia tej części SQLa, nazywane często *zapytaniami*, zaczynają się od słowa `SELECT`. Jest to najbardziej złożona część SQLa i służy do generowania dowolnych zestawień danych znajdujących się w bazie. Dalej przedstawiono i omówiono niektóre przykłady użycia zapytań.

Proste wybieranie rekordów. Aby w wyniku otrzymać wszystkie pola i wszystkie rekordy z wybranej tabeli, należy użyć zapytania z listingu 5.19. Zwróci ono wszystkie pola i wszystkie rekordy z tabeli `studenci`.

Listing 5.19
```
1 SELECT * FROM studenci;
```

Na ogół jednak potrzebne jest wyświetlenie tylko wybranych pól z danej tabeli. Aby to zrobić, należy umieścić listę pól po słowie `SELECT`, jak na listingu 5.20. Zapytanie to zwróci wszystkie rekordy, ale tylko pola `imie` i `nazwisko`.

Listing 5.20
```
1 SELECT imie, nazwisko FROM studenci;
```

Aby dodatkowo ograniczyć listę zwracanych rekordów, należy użyć warunku umieszczonego na końcu zapytania po słowie `WHERE`, analogicznie do przypadku poleceń `DELETE` czy `UPDATE`. Przykład takiego zapytania przedstawia listing 5.21. Zapytanie to zwróci pola `imie` i `nazwisko` dla rekordów, w których pole `imie2` ma wartość `Jan`.

Listing 5.21
```
1 SELECT imie, nazwisko FROM studenci
2   WHERE imie2 = 'Jan';
```

Sortowanie rekordów. Rekordy zwracane przez zapytanie mogą być posortowane według wybranych kryteriów. Aby posortować zwracane rekordy według nazwisk studentów, na końcu zapytania należy dodać `ORDER BY nazwisko`. Kompletne zapytanie przedstawia listing 5.22. Aby posortować rekordy według więcej niż jednego pola, po `ORDER BY` należy umieścić ich listę, rozdzielając je przecinkami. Listing 5.23 przedstawia przykład sortowania rekordów najpierw według nazwiska, potem według imienia. Rekordy zostaną posortowane po nazwisku, a te, dla których nazwisko będzie takie samo, zostaną posortowane po imieniu.

Listing 5.22
```
1 SELECT imie, nazwisko FROM studenci
2   ORDER BY nazwisko;
```

Listing 5.23
```
1 SELECT imie, nazwisko FROM studenci
2   ORDER BY nazwisko, imie;
```

Domyślnie rekordy są sortowane w porządku rosnącym. Aby posortować je malejąco, należy użyć słowa `DESC`, tak jak w przykładzie z listingu 5.24. Nic nie stoi na przeszkodzie, żeby sortować według jednych kryteriów rosnąco, a według

Listing 5.24
```
1 SELECT imie, nazwisko FROM studenci
2   ORDER BY nazwisko DESC;
```

Listing 5.25
```
1 SELECT imie, nazwisko FROM studenci
2   ORDER BY nazwisko, imie DESC;
```

innych malejąco. Przykład na listingu 5.25 przedstawia sortowanie rekordów domyślnie (rosnąco) według nazwisk oraz malejąco według imion.

Można też wprost wskazać sortowanie rosnąco za pomocą słowa `ASC`. Można więc sortować zwracane rekordy według wielu różnych pól i różnej kolejności, także wskazując wprost kolejność rosnącą, jak na listingu 5.26.

Listing 5.26
```
1 SELECT imie, nazwisko FROM studenci
2   ORDER BY nazwisko DESC, imie DESC, indeks ASC;
```

Rekordy z wielu tabel. SQL pozwala robić zestawienia danych zapisanych w wielu tabelach. Po słowie `FROM` zamiast jednej nazwy tabeli można umieścić ich kilka, rozdzielając je przecinkami. Najprostszy przykład takiego zapytania znajduje się na listingu 5.27. Zapytanie to zwróci pełen iloczyn kartezjański rekordów z obydwu wymienionych w nim tabel. Każdy z rekordów z tabeli `studenci` zostanie więc połączony z każdym rekordem z tabeli `stypendia`, liczba zwróconych wierszy będzie równa iloczynowi liczby wierszy w tabeli stypendia i liczby wierszy w tabeli `oceny`. Nie jest to więc użyteczna informacja. Użyteczne jest połączenie danych studentów z tabeli `studenci` z odpowiadającymi im danymi o stypendiach z tabeli `stypendia`. Z pełnego iloczynu kartezjańskiego dwóch tabel należy więc wybrać rekordy, dla których `id` studenta z tablicy `studenci` odpowiada wartości pola `student_id` z tabeli `stypendia`. Zapytanie z listingu 5.27 wystarczy rozszerzyć o taki warunek, żeby uzyskać użyteczne zestawienie: listę studentów z ich stypendiami. Taki przykład przedstawia listing 5.28.

Listing 5.27
```
1 SELECT * FROM studenci, stypendia;
```

Listing 5.28
```
1 SELECT * FROM studenci, stypendia
2   WHERE studenci.id = stypendia.student_id;
```

Jak widać, w warunku z listingu 5.28 użyto nazw tabel oraz nazw pól z tych tabel, aby jednoznacznie określić, o które pola chodzi. Przy bardziej złożonych warunkach używanie dość długich nazw tabel może być niewygodne. W takiej sytuacji można wykorzystać tzw. aliasy, nadając tabeli tymczasowo krótszą nazwę

na potrzeby jednego zapytania. Takie zapytanie znajduje się na listingu 5.29 i jego znaczenie jest dokładnie takie samo, jak zapytania z listingu 5.28. Specyfikując nazwy tabel po słowie `FROM`, każdej z nich nadano alias, podając go po nazwie tabeli przed przecinkiem. Ten alias wykorzystano później w warunku.

Listing 5.29
```
1 SELECT * FROM studenci s, stypendia sty
2   WHERE s.id = sty.student_id;
```

Łącząc otrzymane wiadomości, można napisać zapytanie zwracające z dwóch tabel pasujące do siebie rekordy spełniające określony warunek i odpowiednio posortowane. Zapytanie takie przedstawia listing 5.30. Zwraca ono imiona i nazwiska oraz kwoty stypendiów, ale tylko dla studentów, dla których kwota stypendium jest mniejsza od 200. Wyniki są posortowane według kwoty stypendium – od największej, następnie według nazwiska, potem imienia rosnąco.

Listing 5.30
```
1 SELECT imie, nazwisko, kwota
2   FROM studenci s, stypendia sty
3   WHERE s.id = sty.student_id AND
4         sty.kwota < 200
5   ORDER BY kwota DESC, nazwisko, imie;
```

W zapytaniu z listingu 5.30 po słowie `SELECT` wymieniono trzy nazwy kolumn: dwie z tabeli `studenci` i jedną z tabeli `stypendia`. Nie wymagało to podania nazw tabel, z których są wybierane kolumny, gdyż użyte nazwy kolumn w tych tabelach się nie powtarzają. Jeśli w zapytaniu występowałyby dwie tabele z takimi samymi nazwami kolumn, po słowie `SELECT` trzeba byłoby podać nazwę tabeli wraz z nazwami kolumn (np. `studenci.imie` albo `s.imie`), podobnie jak są one podawane w warunkach po `WHERE`. Jeśli nazwy pól w tabelach nie powtarzają się, w warunkach można także pominąć nazwy tabel, pozostawiając same nazwy kolumn (`id = student_id AND kwota < 200`). Łatwiej jednak zinterpretować warunki, jeśli występują w nich nazwy tabel bądź ich aliasy, zwłaszcza jeśli warunki te dotyczą złączenia tabel.

Pojedyncze zapytanie pozwala wybierać rekordy z dowolnej liczby tabel, łącząc je ze sobą. Należy jednak pamiętać o wyspecyfikowaniu warunków łączących rekordy pomiędzy tabelami, aby uzyskać użyteczne informacje. Teoretycznie wybieranie rekordów z wielu tabel przedstawia się jako wygenerowanie pełnego iloczynu kartezjańskiego, a następnie wybranie z niego rekordów spełniających warunki (podane po `WHERE`). W rzeczywistości jednak systemy zarządzające bazami danych unikają realizacji zapytań w ten sposób. Iloczyny kartezjańskie nawet stosunkowo niewielkich tabel składają się z bardzo wielu rekordów i ich wygenerowanie jest bardzo kosztowne, zarówno pod względem obliczeniowym, jak i pamięciowym.

Systemy zarządzania bazami danych stosują skomplikowane algorytmy pozwalające realizować zapytania bez konieczności kosztownego generowania pełnych iloczynów kartezjańskich.

Zaawansowane łączenie tabel. Opisane wcześniej złączenie definiowane za pomocą warunku w podanego w klauzuli `WHERE` pozwala połączyć tabele tylko w jeden sposób. W wyniku zapytania z listingu 5.28 pojawią się jedynie odpowiadające sobie rekordy z obydwu tabel. Jeśli w tabeli `studenci` byłyby dane studentów, którzy nie mają przydzielonych stypendiów, to nie pojawią się one w wyniku. Czasami jednak konieczne jest wygenerowanie pełnej listy danych z jednej z tabel i dołączenie do nich danych z innej tabeli, jeśli takie istnieją. Przykładowo, można wygenerować pełną listę studentów i obok tych, którzy mają stypendia, umieścić też dodatkowo informację o stypendium. Do takiego łączenia tabel służy klauzula `JOIN` użyta w zapytaniu z listingu 5.31.

Listing 5.31
```
1  SELECT imie, nazwisko, kwota
2    FROM studenci s LEFT JOIN stypendia sty
3                    ON s.id = sty.student_id;
```

Zapytanie z listingu 5.31 spowoduje złączenie tabeli `studenci` z tabelą `stypendia`. Warunek złączenia podano po słowie `ON`. Jak poprzednio, zakłada on, że pole `student_id` z tabeli `stypendia` musi odpowiadać polu `id` z tabeli student. Ponieważ użyto klauzuli `LEFT JOIN`, to z tabeli po lewej stronie słowa `JOIN` (`studenci`) zostaną wzięte wszystkie rekordy i do nich zostaną dołączone rekordy z tabeli po prawej stronie słowa `JOIN` (`stypendia`), jeśli warunek złączenia będzie na to pozwalał. Na liście wynikowej pojawią się więc imiona i nazwiska wszystkich studentów. Studenci, którzy mają przydzielone stypendium, będą mieć jego wartość wpisaną w wyniku, w kolumnie `kwota`, pozostali kolumnę tę będą mieć pustą[7]. Do tego zapytania można dodać na końcu słowo `WHERE` i warunki ograniczające listę wyświetlanych rekordów, np. tak jak to pokazano na listingu 5.32.

Listing 5.32
```
1  SELECT imie, nazwisko, kwota
2    FROM studenci s LEFT JOIN stypendia sty
3                    ON s.id = sty.student_id
4    WHERE imie = 'Jan';
```

Jeśli w zapytaniu z listingu 5.31 zamienimy tabele lewą z prawą bądź zamiast `LEFT JOIN` użyjemy `RIGHT JOIN`, otrzymamy efekt odwrotny: wszystkie re-

[7] W SQLu wartość pustą oznacza się jako `NULL`. Taka też będzie wartość pola `kwota` dla tych studentów, którym nie przydzielono stypendiów

kordy zostaną wyświetlone z tabeli `stypendia`, a z tabeli `studenci` tylko te pasujące[8].

W przypadku łączenia tabel, w których związek jest opcjonalny po obu stronach, zamiast `LEFT` bądź `RIGHT` można użyć `FULL JOIN`. Spowoduje to umieszczenie w wyniku wszystkich rekordów z obydwu złączanych tabel. Te rekordy, które będą sobie odpowiadały, będą złączone, natomiast te, które nie mają odpowiedników w drugiej tabeli, będą miały wartości `NULL` w kolumnach odpowiadających drugiej tabeli.

Użycie samego tylko słowa `JOIN` (np. `studenci JOIN stypendia ON ...`) spowoduje działanie, takie jak w przypadku złączenia za pomocą warunku w `WHERE`. W wyniku znajdą się tylko odpowiadające sobie rekordy z obydwu tabel.

Złączeń `JOIN` można używać wielokrotnie w pojedynczym zapytaniu. Mają one też znacznie więcej opcji niż opisano. Warto zapoznać się z nimi za pomocą dokumentacji SQLa dołączanej do dokumentacji serwerów baz danych.

Funkcje agregujące. SQL pozwala nie tylko wykonywać zestawienia danych istniejących w tabelach, ale także grupować je. Najprostszą funkcją agregującą (grupującą) jest `count`, która służy do zliczenia liczby rekordów. Listing 5.33 przedstawia zapytanie, które zwróci liczbę rekordów w tabeli `studenci`.

Listing 5.33
```
1  SELECT count(*) FROM studenci;
```

Funkcji agregujących można używać w bardziej zaawansowanych zastosowaniach. Zapytanie 5.34 zwróci liczbę rekordów z tabeli `studenci` z podziałem według imion: dla każdego imienia podana będzie informacja, ilu jest studentów o tym imieniu. `count(*)` jest wywołaniem funkcji zliczającej rekordy. Umieszczenie kolumny `imie` na liście kolumn do wyświetlenia (po słowie `SELECT`) oraz klauzula `GROUP BY imie` powodują, że rekordy są grupowane według imion. SQL w przypadku użycia funkcji agregującej w zapytaniu wymaga, aby każda kolumna, która ma być wyświetlona (jest wymieniona po słowie `SELECT`), a nie jest argumentem funkcji agregującej, była wymieniona w klauzuli `GROUP BY`. Z tego względu zapytanie 5.35 jest niepoprawne i się nie wykona.

Listing 5.34
```
1  SELECT imie, count(*) FROM studenci
2    GROUP BY imie;
```

[8] W tym szczególnym przypadku nie ma to większego sensu – wszystkie rekordy z tabeli `stypendia` muszą mieć opowiadających im studentów. W praktyce jednak taka sytuacja nie zawsze występuje – można złączać tabele, dla których związek w obydwu kierunkach jest opcjonalny, jak np. związek *prowadzący – zajęcia* z diagramu na rys. 5.15.

Listing 5.35
```
1 -- Niepoprawne zapytanie -- brak grupowania po kolumnie imie!
2 SELECT imie, count(*) FROM studenci;
```

Inną bardzo użyteczną funkcją agregującą jest `avg`, która służy do obliczania średniej arytmetycznej. Do policzenia średniej wszystkich stypendiów z tabeli `stypendia` można użyć zapytania z listingu 5.36. Aby sprawdzić, jaka jest średnia kwota dla każdego rodzaju stypendium, wyniki należy pogrupować według kolumny `rodzaj`, tak jak w zapytaniu na listingu 5.37. Zapytanie to zwróci tabelę składającą się z kolumny `rodzaj` i kolumny zawierającej średnią wartość kwoty stypendiów danego rodzaju.

Listing 5.36
```
1 SELECT avg(kwota) FROM stypendia;
```

Listing 5.37
```
1 SELECT rodzaj, avg(kwota) FROM stypendia
2   GROUP BY rodzaj;
```

Sumę wszystkich stypendiów można obliczyć za pomocą funkcji `sum`. Jej użycie pokazuje listing 5.38. Bardziej użyteczne jest jednak obliczenie sumy stypendiów dla każdego studenta z osobna, co wymaga pogrupowania wyników według identyfikatora studenta (pola `student_id`), tak jak na listingu 5.39. W wyniku zapytanie zwróci tabelę składającą się z dwóch kolumn: `student_id` i sumy stypendiów dla studenta o tym `id`.

Listing 5.38
```
1 SELECT sum(kwota) FROM stypendia;
```

Listing 5.39
```
1 SELECT student_id, sum(kwota) FROM stypendia
2   GROUP BY student_id;
```

Wyniki można grupować według wielu różnych kryteriów, niekoniecznie według tylko jednej kolumny. Można też użyć więcej niż jednej funkcji agregującej, jak w zapytaniu z listingu 5.40, które zwróci w wyniku tabelę zawierającą identyfikator studenta, liczbę stypendiów, jakie posiada, oraz sumaryczną kwotę jego stypendiów. Aby kolumna z sumą stypendiów miała wygodną nazwę `suma`, można ją nadać, po `sum(kwota)` dodając `AS suma`, albo po prostu `suma`, tak jak na listingu 5.41.

Listing 5.40
```
1 SELECT student_id, count(*), sum(kwota) FROM stypendia
2   GROUP BY student_id;
```

Listing 5.41
```
1 SELECT student_id, count(*), sum(kwota) suma FROM stypendia
2   GROUP BY student_id;
```

Często używanymi i bardzo użytecznymi funkcjami agregującym są `min` i `max` zwracające odpowiednio najmniejszą i największą wartość dla danego pola. Za ich pomocą można np. wygenerować zestawienie pokazujące największą oraz najmniejszą kwotę stypendium każdego rodzaju, co pokazuje listing 5.42.

Listing 5.42
```
1 SELECT rodzaj, min(kwota), max(kwota) FROM stypendia
2   GROUP BY rodzaj;
```

Funkcji agregujących można używać nie tylko na wartościach liczbowych. Przykładowo, funkcje `min` i `max` mają bardzo istotne zastosowanie dla typów związanych z datami, a także do łańcuchów tekstowych. Wartości tych typów można sortować, można więc też wyznaczyć dla nich największą i najmniejszą wartość. Za pomocą tych funkcji można odnaleźć najwcześniejszą bądź najpóźniejszą datę, pierwsze bądź ostatnie nazwisko na liście (w kolejności alfanumerycznej). Szczegółowego opisu funkcji agregujących oraz typów danych, na których działają, należy szukać w dokumentacji do wykorzystywanego systemu zarządzania bazą danych.

Nic nie stoi na przeszkodzie, aby listę rekordów branych pod uwagę w takim zapytaniu zawęzić za pomocą `WHERE`. Zapytanie z listingu 5.43 uwzględni tylko rekordy, dla których `student_id` ma wartość 12.

Listing 5.43
```
1 SELECT rodzaj, min(kwota), max(kwota) FROM stypendia
2   WHERE student_id = 12
3   GROUP BY rodzaj;
```

Jeśli warunek ma zawierać wartość obliczoną za pomocą funkcji agregującej, nie można umieścić go w `WHERE`. Warunki umieszczone po słowie `WHERE` są brane pod uwagę przed obliczeniem wartości przez funkcję agregującą. Warunki zawierające wartości obliczone przez funkcje agregujące umieszcza się w klauzuli `HAVING`. Aby więc ograniczyć listę z zapytania 5.42, tak aby zawierała tylko przypadki, w których minimalna kwota dla danego rodzaju stypendiów będzie mniejsza od 500, należy użyć zapytania z listingu 5.44.

Listing 5.44
```
1 SELECT rodzaj, min(kwota), max(kwota) FROM stypendia
2   GROUP BY rodzaj
3   HAVING min(kwota) < 500;
```

Warunki umieszczane po słowie HAVING także mogą być złożone, można w nich używać operatorów logicznych w postaci słów AND, OR i NOT oraz nawiasów. Można oczywiście łączyć użycie WHERE i HAVING w jednym zapytaniu – warunek w WHERE ograniczy listę rekordów wziętych pod uwagę przed grupowaniem, a warunek w HAVING dodatkowo usunie z listy wynikowej rekordy, które po zastosowaniu funkcji agregujących i grupowania nie spełniają podanego warunku. Taki przykład przedstawia listing 5.45. Zapytanie to zwróci minimalną i maksymalną wartość stypendium każdego rodzaju. Obliczając te wartości, weźmie pod uwagę tylko studentów, których identyfikatory są pomiędzy 10 a 50, a wyświetlone zostaną tylko wyniki, dla których obliczona kwota minimalna jest mniejsza od 500.

Listing 5.45
```
1  SELECT rodzaj, min(kwota), max(kwota) FROM stypendia
2    WHERE student_id >= 10 AND student_id <= 50
3    GROUP BY rodzaj
4    HAVING min(kwota) < 500;
```

Podzapytania. Dotychczas opisane zapytania pozwalały tworzyć zestawienia danych zapisanych w tabelach bazy danych. Wyniki tych zapytań także były tabelami – składały się z kolumn i rekordów. Nie powinno więc dziwić, że SQL pozwala tworzyć zapytania do wyników innych zapytań.

Do tej pory w zapytaniach po słowie FROM były umieszczane nazwy tabel, z których należało uzyskać dane. Zamiast nazwy tabeli można tam umieścić w nawiasach inne zapytanie SQL, trzeba jednak jego wynikowi nadać alias (tak samo jak w zapytaniach aliasy były nadawane dla tabel). Można to wykorzystać do napisania zapytania, które wyświetli bardziej użyteczne informacje niż to z listingu 5.41. Tamto zapytanie wyświetla jedynie identyfikator studenta. Bardziej użyteczne będzie jednak wyświetlenie imienia i nazwiska studenta. Aby to zrobić, można potraktować wynik zapytania 5.41 tak, jakby to była tabela (nazwana kwoty), w której kluczem obcym jest student_id. Można wtedy napisać zapytanie, które połączy ją z tabelą studenci zawierającą imię i nazwisko studenta. Takie zapytanie przedstawia listing 5.46.

Listing 5.46
```
1  SELECT imie, nazwisko, suma FROM
2    studenci,
3    (SELECT student_id, count(*), sum(kwota) suma FROM stypendia
4      GROUP BY student_id) kwoty
5    WHERE studenci.id = kwoty.student_id;
```

Tabele z podzapytaniami można łączyć zarówno za pomocą warunków w WHERE, tak jak w zapytaniu z listingu 5.46, jak i za pomocą JOIN. Stosując

JOIN, w miejscu, gdzie w opisanych dotąd przykładach pojawiały się nazwy tabel, można umieścić w nawiasach podzapytanie. Ważne, aby wynikowi podzapytania nadać alias, który pozwoli używać go w dalszej części zapytania. Przykład takiego zapytania przedstawia listing 5.47. Oczywiście w miarę potrzeby można używać także LEFT, RIGHT i FULL JOIN.

Listing 5.47
```
1  SELECT imie, nazwisko, suma FROM
2    studenci
3    JOIN
4    (SELECT student_id, count(*), sum(kwota) suma FROM stypendia
5      GROUP BY student_id) kwoty
6    ON studenci.id = kwoty.student_id;
```

Podzapytań można także używać w warunkach umieszczanych po słowie WHERE. Aby wyświetlić dane studentów, którzy mają jakieś stypendium wyższe od 1000, można najpierw wybrać identyfikatory tych studentów z tabeli stypendia, tak jak na listingu 5.48, a następnie użyć tej listy, wybierając studentów, których dane należy wyświetlić. Kompletne zapytanie pokazuje listing 5.49. Słowo IN w warunku powoduje sprawdzenie, czy identyfikator studenta (po lewej stronie słowa IN) znajduje się na liście po prawej stronie (wynik zapytania). W tym wypadku ten sam efekt można osiągnąć, złączając tabele studenci i stypendia. Jednakże w bardziej skomplikowanych przykładach zawierających złożone złączenia takie podzapytanie jest bardzo użyteczne.

Listing 5.48
```
1  SELECT student_id FROM stypendia WHERE kwota > 1000;
```

Listing 5.49
```
1  SELECT imie, nazwisko FROM studenci
2    WHERE id IN
3      (SELECT student_id FROM stypendia WHERE kwota > 1000);
```

Łączenie wyników zapytań. SQL pozwala wykonywać na wynikach zapytań operacje mnogościowe – znane z matematycznych operacji na zbiorach. Jest to możliwe, jeśli zapytania zwracają wyniki z taką samą liczbą kolumn i typy kolumn są ze sobą kompatybilne.

Za pomocą polecenia zapytanie1 UNION zapytanie2 można połączyć wyniki dwóch zapytań, w rezultacie otrzymując sumę mnogościową rekordów obu zapytań – rekordy występujące przynajmniej w jednym z nich. Polecenie INTERSECT pozwala obliczyć część wspólną wyników dwóch zapytań – rekordy wstępujące w wynikach obydwu zapytań. Wreszcie polecenie EXCEPT zwraca

rekordy występujące w wynikach zapytania z lewej strony, a niewystępujące w wynikach zapytania z prawej strony polecenia. W przypadku wszystkich przedstawionych poleceń z wyników są usuwane duplikaty, chyba że do polecenia dołączono słowo `ALL` (np. `UNION ALL`).

Podsumowanie. SQL jest powszechnie używany do zarządzania danymi w relacyjnych bazach danych. Ponieważ jest to bardzo rozbudowane narzędzie, w rozdziale przedstawiono jedynie fragment jego możliwości, pozwalając jednak zrozumieć ideę tego języka. Producenci serwerów baz danych dbają o bardzo dokładną dokumentację do SQLa (oczywiście w odmianie obsługiwanej przez ich produkty), dlatego w poszukiwaniu dobrych materiałów dotyczących tego języka warto zajrzeć do dokumentacji wykorzystywanego DBMS.

5.6.6. Transakcje

Bezpieczne przetwarzanie danych

Z założenia baza danych jest przeznaczona do równoczesnego wykorzystywania przez wielu klientów: wiele komputerów, wiele działających programów, wielu użytkowników. Naturalnie w takiej sytuacji na pojedynczej bazie danych w tym samym czasie wykonywanych jest wiele operacji. Może się też zdarzyć, że pewne operacje są wykonywane w tym samym czasie na tych samych danych! Dodatkowo operacje wykonywane przez aplikacje na danych mogą być długotrwałe lub składać się z wielu kroków. W zasadzie jednak żadna, nawet najkrótsza z wykonywanych operacji nie jest aż tak krótka, aby nie mogła być przerwana awarią – brakiem prądu, awarią sprzętu, chwilową przerwą w łączności zdalnej itp.

System zarządzania bazą danych odpowiada za bezpieczeństwo i poprawność przechowywanych danych w obydwu opisanych przypadkach. Bez względu na zdarzenie dane w bazie powinny pozostać w stanie spójnym. Nie można pozwolić, aby któraś z operacji wykonała się jedynie częściowo. Nie można też pozwolić, żeby wykonywane równocześnie operacje spowodowały, że którakolwiek z nich otrzyma z bazy danych nieprawdziwe dane czy spowoduje, że dane zapisane w bazie będą niepoprawne.

Sztandarowym przykładem, w którym są potrzebne transakcje, aby zapewnić bezpieczeństwo danych w przypadku awarii, jest wykonywanie przelewu pieniężnego. Przelew 1000 zł z konta p. Kowalskiego na konto p. Kwiatkowskiego składa się z dwóch istotnych operacji przedstawionych w tabl. 5.13. Awaria powodująca przerwanie połączenia z bazą danych po wykonaniu operacji 1., a przed wykonaniem operacji 2. spowoduje zaginięcie 1000 zł, których nie będzie ani na koncie p. Kowalskiego (bo już je odjęto) ani na koncie p. Kwiatkowskiego (bo nie zdążono ich dodać przed awarią). Bezpieczne wykonanie operacji przelewu wymaga, aby wykonały się poprawnie i do końca wszystkie operacje albo aby nie wykonała się

Tablica 5.13. Operacja wykonania przelewu z konta

1. Odjęcie 1000 zł. od stanu konta p. Kowalskiego.
2. Dodanie 1000 zł. do stanu konta p. Kwiatkowskiego.

Tablica 5.14. Dwie przeplatające się operacje płatności wykonywane za pomocą środków z jednego konta bankowego

1. Klient 1.: Sprawdzenie, czy stan konta pozwala na wykonanie płatności.
2. Klient 2.: Sprawdzenie, czy stan konta pozwala na wykonanie płatności.
3. Klient 1.: Odjęcie kwoty z konta.
4. Klient 2.: Odjęcie kwoty z konta.

żadna. Jeśli przelew byłby wykonywany przez program za pomocą dwóch oddzielnych niezależnych poleceń SQL, zawsze istniałaby możliwość pozostawienia bazy danych w stanie niespójnym, przejawiającym się w tym przypadku „zagubionym” 1000 zł.

Przykładem, w którym w tym samym czasie, na tych samych danych wykonywane są dwie różne operacje jest równoległe wykonywanie dwóch płatności środkami z jednego konta bankowego. Nietrudno wyobrazić sobie przykład, w którym taka sytuacja może mieć miejsce, zwłaszcza w przypadku kont, do których dostęp ma więcej niż jedna osoba fizyczna (konta firmowe, konta wspólne). Operacja płatności wymaga wykonania dwóch kroków:

1. Sprawdzenia, czy stan konta pozwala na wykonanie płatności.
2. Odjęcia kwoty z konta.

Jeżeli dwa klienty (programy klienckie) próbują równocześnie wykonać operacje płatności, kolejne kroki mogą wykonać się w różnej kolejności. Jedno z pesymistycznych rozwiązań przedstawia tabl. 5.14. Każdy z dwóch klientów najpierw sprawdził stan konta, z którego ma być wykonana płatność, żaden z nich nie uwzględnił jednak drugiej płatności będącej w toku. Następnie każdy z klientów,

stwierdziwszy, że stan konta pozwala na wykonanie płatności, pomniejszył stan konta o kwotę płatności. Jeśli stan konta pozwalał na wykonanie tylko jednej płatności, to scenariusz ten spowoduje wykonanie nieuprawnionej płatności i pozostawienie stanu konta klienta mniejszego od zera, pomimo że każdy z klientów na początku sprawdził, czy środki na koncie są wystarczające! Taką sytuację we współbieżności nazywamy *wyścigiem*.

System zarządzania bazą danych musi zapewnić bezpieczeństwo danych w każdej z opisanych sytuacji. Jeżeli klient wykonujący pewne operacje na danych otrzyma od DBMS informację, że operacja przebiegła poprawnie, to zapisane w bazie dane muszą być bezpieczne i poprawne, niezależnie od tego, jakie operacje wykonywały się w tym samym czasie czy jaka awaria nastąpiła później. Jeśli nie jest możliwe poprawne wykonanie operacji, których zażądał klient (np. z powodu innych operacji trwających w tym samym czasie), klient musi zostać o tym poinformowany, a zapisane w bazie dane muszą pozostać poprawne.

Transakcje i reguła ACID

Aby zapewnić wymagane bezpieczeństwo danych, operacje wykonywane na relacyjnych bazach danych są grupowane w *transakcje*. Jest to bardzo ważny termin, transakcje stanowią bowiem podstawę poprawności działania relacyjnych baz danych i bezpieczeństwa zapisanych w nich danych.

Aby spełniać swoją rolę, transakcje muszą spełniać cztery kluczowe wymagania, nazywane regułą *ACID* (od angielskich nazw tych wymagań).

Atomicity (atomowość) – transakcja stanowi całość; jeśli nie można wykonać wszystkich jej kroków, baza danych musi pozostać w takim stanie, jakby żaden krok transakcji nigdy się nie wykonał.

Consistency (spójność) – dane zapisane w bazie po zakończeniu transakcji muszą być spójne i kompletne.

Isolation (izolacja) – każda z transakcji musi być wykonywana tak, jakby była jedyną wykonywaną w danym momencie (niezależnie od tego, ile transakcji wykonuje się w tym samym czasie).

Durability (trwałość) – awaria po zakończeniu transakcji nie może mieć wpływu na jej wynik – dane muszą być na trwałe zmienione.

Zapewnienie *atomowości* operacji rozwiązuje problem awarii połączenia z bazą danych opisywany w związku z przykładem z tabl. 5.13. Jeśli z jakiegokolwiek powodu system nie będzie w stanie wykonać drugiej operacji, pierwsza zostanie odwołana.

Izolacja zapewnia, że DBMS może równocześnie obsługiwać wielu klientów. Umieszczenie każdej płatności z przykładu tabl. 5.14 w osobnej transakcji zmusi system zarządzania bazą danych do wykonania operacji w kolejności zapewniającej spójność danych. W pesymistycznym przypadku z tabl. 5.14 transakcja wykonywana przez klienta 2. zostanie zerwana i wycofana, gdyż nie będzie możliwe jej poprawne zakończenie z powodu braku środków na koncie.

Spełnienie warunku *trwałości* zapewnia, że po poprawnym zakończeniu transakcji nie wystąpi problem, że „coś się nie zapisało...". Jeżeli operacje z przykładów zamieszczonych w tabl. 5.13 i 5.14 zostaną umieszczone w transakcjach, to klienty, które otrzymały od systemu zarządzania bazą danych informację o poprawnym zakończeniu transakcji, mogą mieć pewność, że skutki transakcji są trwałe. Awaria połączenia z bazą danych, awaria sprzętowa ani awaria zasilania nie może spowodować, że baza danych powróci do stanu sprzed transakcji.

Warunki spójności dla bazy danych są określane w momencie jej tworzenia. Żadna transakcja nie może powodować pozostawienia danych w stanie niespójnym. Przykładowo, każde stypendium zapisane w relacji `stypendia` (tabl. 5.5) musi mieć właściciela – studenta w relacji `studenci` (tabl. 5.4). Usuwając z bazy danych jednego ze studentów, należy też usunąć dane o jego stypendiach. Poprawne wykonanie transakcji i spełnienie warunku *spójności* nie pozwala przerwać operacji po usunięciu studenta i pominąć usuwanie jego stypendiów.

Transakcje w SQL

Zapewnienie poprawnej realizacji transakcji ze spełnieniem wszystkich wymagań jest zagadnieniem bardzo złożonym. Jest to jednakże zadanie realizowane przez system zarządzania bazą danych i to właśnie na twórców DBMS spada odpowiedzialność za poprawne rozwiązanie tego problemu. Korzystający z baz danych muszą jedynie zadecydować, które operacje powinny stanowić całość – pojedynczą transakcję. Należy jedynie zdecydować, kiedy transakcja się rozpoczyna oraz kiedy się kończy. Można także zażądać od systemu zarządzania bazą danych wycofania aktualnej transakcji.

Obsługa transakcji w SQL jest bardzo prosta i sprowadza się do świadomego używania trzech poleceń:

- `BEGIN`
- `COMMIT`
- `ROLLBACK`

Polecenie `BEGIN` oznacza rozpoczęcie transakcji. Wszystkie kolejne polecenia SQL będą traktowane jako pojedyncza transakcja, która musi spełniać wymagania

ACID. Po ostatnim poleceniu, które ma być częścią transakcji, należy wydać polecenie `COMMIT`. Oznacza ono żądanie zakończenia transakcji i uwidocznienia jej skutków w bazie danych. Poprawne zakończenie tego polecenia oznacza poprawne wykonanie całej transakcji i bezpieczne zapisanie jej wyniku w bazie danych. Jeśli z jakiegoś powodu w trakcie wykonywania transakcji klient stwierdzi, że chce się z niej wycofać, powinien poinformować o tym DBMS, wydając polecenie `ROLLBACK`. Skutkiem tego będzie wycofanie wszystkich wykonanych dotąd operacji.

Przykład wykonania w ramach transakcji bazodanowej przelewu 1000 zł z konta p. Kowalskiego na konto p. Kwiatkowskiego przedstawia listing 5.50[9]. Polecenie `BEGIN` w linii 1 rozpoczyna transakcję, polecenie z linii 2 pomniejsza stan konta p. Kowalskiego o 1000 zł, polecenie z linii 4 powiększa stan konta p. Kwiatkowskiego o 1000 zł, wreszcie polecenie z linii 6 zatwierdza transakcję. Jeśli polecenie `COMMIT` powiedzie się, skutki wszystkich poleceń tej transakcji zostaną zapisane w bazie – przelew będzie wykonany. Jeśli polecenie `COMMIT` zwróci błąd, będzie to oznaczało, że DBMS z jakichś powodów nie był w stanie zakończyć transakcji poprawnie, skutki żadnego z poleceń nie zostaną zapisane w bazie i pieniądze pozostaną na koncie p. Kowalskiego. W przypadku awarii (np. awarii sieci i przerwania komunikacji z bazą danych), jeśli przez zadany czas nie zostanie wykonane polecenie `COMMIT`, DBMS sam podejmie decyzję o wycofaniu transakcji – pieniądze pozostaną na koncie p. Kowalskiego.

Listing 5.50

```
1 BEGIN;
2 UPDATE konta SET stan = stan - 1000
3   WHERE nazwisko = 'Kowalski';
4 UPDATE konta SET stan = stan + 1000
5   WHERE nazwisko = 'Kwiatkowski';
6 COMMIT;
```

Przykład wykonania płatności w ramach transakcji bazodanowej przedstawia listing 5.51. Pierwsze polecenie rozpoczyna transakcję, kolejne sprawdza, czy stan konta pozwala na płatność w kwocie 2500. Jeśli polecenie z linii 2 zwróci prawdę, klient wykonuje polecenie z linii 3, pomniejszając stan konta o 2500. Ostatnie polecenie kończy i zatwierdza transakcję. Jeśli polecenie `COMMIT` wykona się poprawnie, klient ma pewność, że płatność została zakończona sukcesem. Jeśli po wykonaniu zapytania z linii 2 klient otrzyma wynik negatywny (stan konta nie pozwala na wykonanie płatności), powinien wykonać polecenie `COMMIT`, kończąc tym samym rozpoczętą transakcję bez modyfikowania stanu konta. DBMS zapewni, że dwie transakcje płatności z tego samego konta wykonywane równocześnie nie

[9] Dla uproszczenia przykładu zakładamy, że nazwisko jednoznacznie identyfikuje klienta.

spowodują przeplotu operacji z linii 2 i 3, który spowodowałby niespójność w bazie danych.

Listing 5.51
```
1 BEGIN;
2 SELECT stan > 2500 FROM konta WHERE id = 1234;
3 UPDATE konta SET stan = stan - 2500 WHERE id = 1234;
4 COMMIT;
```

Korzystanie z transakcji

Złożoność problemu polegającego na realizacji transakcji zgodnie z regułą ACID spoczywa w całości po stronie systemu zarządzania bazą danych. Korzystanie z transakcji po stronie użytkownika bazy danych (czy programisty, którego program korzysta z bazy danych) jest bardzo proste. Użytkownik taki powinien rozumieć do czego służą transakcje, co mogą mu zapewnić i w jakich sytuacjach ich stosowanie jest konieczne. Należy też pamiętać, że długo trwające transakcje stanowią narzut dla DBMS i utrudniają realizację innych transakcji. Z tego względu z mechanizmu tego należy korzystać rozważnie, ale zdecydowanie należy go stosować wszędzie tam, gdzie to jest potrzebne. Programiści systemów bazodanowych niekorzystający z transakcji, niemający świadomości konieczności ich stosowania narażają użytkowników swojego oprogramowania na poważne szkody, a siebie na konieczność poszukiwania trudnych do zlokalizowania błędów i usuwania skomplikowanych niespójności w bazie danych.

5.6.7. Indeksy

Efektywne wyszukiwanie danych

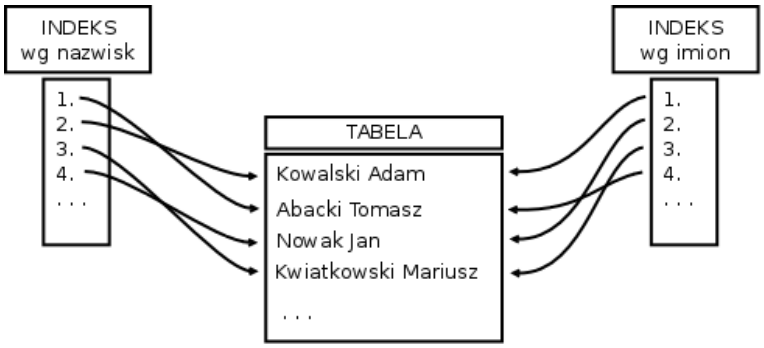
Wyszukiwanie danych według różnych kryteriów jest bardzo ważną funkcją relacyjnych baz danych. Bardzo często od wydajnego wyszukiwania danych zależy efektywność całych aplikacji bazodanowych.

Najszybszym algorytmem wyszukiwania jest wyszukiwanie binarne o czasowej złożoności obliczeniowej $O(log(n))$, jego zastosowanie wymaga jednak, aby przeszukiwane dane były posortowane. Sortowanie danych jest stosunkowo kosztowną operacją – najszybsze algorytmy sortujące dowolny zbiór danych mają czasową złożoność obliczeniową $O(nlog(n))$. Jeśli jednak wyszukiwanie ma być wykonywane wielokrotnie, sortowanie danych okazuje się opłacalne. Problem wyszukiwania w bazach danych jest jednak bardziej złożony – wyszukiwanie jest wykonywane według bardzo wielu różnych kryteriów, a zbiór danych nie może być posortowany równocześnie według różnych kryteriów. Jak łatwo stwierdzić, sortowanie danych

przy zmianie kryteriów wyszukiwania byłoby trudne do zrealizowania i rzadko opłacalne.

Rozwiązaniem problemu jest zastosowanie indeksów. Kosztem niewielkiego narzutu pozwalają one korzystać ze zbioru danych, tak jakby był posortowany. Idea tego rozwiązania jest stosunkowo prosta, co przedstawa rys. 5.16. Dane w tabelach bazy danych nie są sortowane. Aby zapewnić szybkie wyszukiwanie według jakiegoś kryterium, tworzy się dla nich indeks według tego kryterium. Aby w przykładzie z rys. 5.16 umożliwić szybkie wyszukiwanie według nazwisk, dla tabeli należy utworzyć indeks według nazwisk. Indeks ten określa, które rekordy należałoby kolejno odczytywać, aby uzyskać kolejność, jaką zapewnia sortowanie według nazwisk. Aby uzyskać dane sortowane po nazwiskach, system zarządzania bazą danych sprawdza najpierw w indeksie, który rekord odczytać, następnie odczytuje wskazany rekord z tabeli, w kolejnym kroku sprawdza w indeksie, który rekord powinien być drugi w kolejności i odczytuje go z tabeli itd. Takie postępowanie dla człowieka byłoby nużące i kłopotliwe, maszyny są jednak doskonałe do wykonywania tego typu zadań i narzut przy używaniu indeksu jest mało znaczący.

Zamiast więc sortować rekordy w tabeli według zadanego kryterium dla tego kryterium tworzy się indeks, zapamiętując, w jakiej kolejności byłyby zapisane rekordy, gdyby były posortowane według tego kryterium. Kluczową zaletą indeksów jest to, że (jak pokazuje przykład z rys. 5.16) dla jednej tabeli można utworzyć wiele, niezależnych od siebie indeksów. Efekt jest taki, jakby tabela była posortowana według różnych kryteriów równocześnie. Kryteria, według których są tworzone indeksy, mogą być bardzo złożone. Przykładowo, można utworzyć indeks, dla którego kryterium sortowania jest: według nazwiska rosnąco, imienia malejąco i wieku rosnąco.



Rys. 5.16. Indeksy bazodanowe

Poza niewielkim narzutem podczas dostępu do danych indeksy powodują także narzut podczas aktualizacji danych – konieczna jest wtedy aktualizacja wszystkich indeksów, które dotyczą aktualizowanych danych. Jeśli przykładowo w tabeli

z rys. 5.16 aktualizujemy imię Adama Kowalskiego, zmieniając je na Krzysztof, konieczna jest aktualizacja indeksu dla imion – p. Kowalski po aktualizacji danych powinien być umieszczony jako drugi na liście, a p. Nowak znajdzie się na miejscu pierwszym w tym indeksie. Narzut ten jest jednak stosunkowo niewielki i niewiele jest przypadków, w których dodanie indeksu jest niekorzystne dla wydajności bazy danych.

Indeksy w SQL

Tworzenie indeksów w SQLu jest wykonywane za pomocą polecenia `CREATE INDEX`. Prosty przykład tworzenia indeksu dla pola `nazwisko` z tabeli `studenci` znajduje się na listingu 5.52.

Listing 5.52
```
1  CREATE INDEX studenci_nazwisko_index ON studenci(nazwisko);
```

Od momentu utworzenia indeksu system zarządzania bazą danych dba o jego aktualizację i wykorzystuje go do wyszukiwania danych w tabeli, jeśli tylko jest to możliwe. Warto zajrzeć do dokumentacji stosowanego DBMS, gdyż różne systemy implementują różne rodzaje indeksów, umożliwiając dobranie właściwego algorytmu do rodzaju przechowywanych danych.

Stosowanie indeksów

Stosowanie indeksów znacząco przyspiesza działanie baz danych i obniża obciążenie serwerów, z których korzystają. Gdy to tylko możliwe, należy zapewnić, że baza danych będzie mogła użyć indeksu zamiast sekwencyjnego przeglądania całych tabel. O utworzenie indeksów powinien zadbać projektujący i tworzący bazę danych. Także programiści korzystający z tej bazy danych, gdy potrzebują wyszukiwania według jakiegoś kryterium, powinni upewnić się, że będzie się ono odbywało z użyciem indeksu. Należy zaznaczyć, że wyszukiwanie odbywa się nie tylko wtedy, gdy wprost żąda tego wydane polecenie. Utworzenie złączenia tabel czy aktualizacja danych spełniających zadane kryterium wymaga także wykonania wyszukiwania danych. Dużymi systemami bazodanowymi zawierającymi bardzo dużą liczbę rekordów (np. systemy bankowe) opiekują się specjalnie w tym celu zatrudnieni administratorzy – specjaliści od relacyjnych baz danych. Do ich obowiązków należy także analiza statystyk bazy danych i w razie potrzeby tworzenie indeksów.

Utworzenie indeksu nie jest ani trudne, ani czasochłonne, zarządzaniem utworzonymi indeksami zajmuje się DBMS. Niestety wielu twórców oprogramowania korzystającego z relacyjnych systemów baz danych nie dba o ich stosowanie.

Dzieje się tak na ogół w znacznie mniej zaawansowanych zastosowaniach niż złożone aplikacje bankowe, np. w aplikacjach typu CMS – zarządzających zawartością stron www. Wbrew pozorom rozumne użycie indeksów także w takich zastosowaniach jest bardzo ważne – tego typu systemy obsługujące większe serwisy www przechowują nieraz setki tysięcy rekordów. Wyszukiwanie w ich bazach danych za pomocą indeksów byłoby znacznie szybsze i znacznie mniej obciążałoby serwery. Niestety takimi aplikacjami na ogół nie zajmują się specjaliści od baz danych, a wiadomości ich twórców na temat działania relacyjnych baz danych pozostawiają wiele do życzenia.

5.6.8. Systemy zarządzania bazami danych

Systemy zarządzania relacyjnymi bazami danych zaczęły powstawać w latach 70. XX w. Od tego czasu rozwinęły się i zostały dopracowane. Dzięki temu obecnie na rynku są dostępne bardzo dobre, zarówno komercyjne, jak i darmowe rozwiązania. Do najczęściej stosowanych darmowych rozwiązań należą:

- **PostgreSQL**[10] – bardzo popularny serwer baz danych, powszechnie stosowany, spełniający wymagania bezpiecznego przechowywania danych.
- **MySQL**[11] – serwer relacyjnych baz danych wykorzystywany bardzo często przez serwisy www.
- **SQLite**[12] – prosta baza danych realizująca spory zestaw funkcji relacyjnego DBMS, przechowująca dane w pojedynczym pliku na dysku.
- **Firebird**[13] – prosta baza danych realizująca spory zestaw funkcji relacyjnego DBMS, przechowująca dane w pojedynczym pliku na dysku.

Do najpopularniejszych komercyjnych rozwiązań należą:

- **DB2** – baza danych firmy IBM, powstała dla komputerów typu Mainframe, obecnie jest dostępna też wersja na komputery PC.
- **Oracle** – popularna baza danych firmy Oracle.
- **Microsoft SQL Server**– serwer baz danych produkowany przez firmę Microsoft.

[10] `http://www.postgresql.org/`
[11] `http://www.mysql.com/`
[12] `http://www.sqlite.org/`
[13] `http://www.firebirdsql.org/`

Darmowe serwery relacyjnych baz danych są rozpowszechnione i wykorzystywane w bardzo wielu profesjonalnych zastosowaniach. Oprogramowanie to jest na tyle dojrzałe, że może zapewnić zarówno bezpieczeństwo danych, jak i wymaganą wydajność. Komercyjne rozwiązania na ogół oferują darmowe wersje produktów z ograniczeniami dotyczącymi możliwego sposobu ich wykorzystania. Ograniczenia te dotyczą wykorzystywania do celów komercyjnych bądź konfiguracji sprzętu, na którym dany produkt może być uruchamiany. Na ogół jednak bez przeszkód można otrzymać, zainstalować i używać danego rozwiązania do celów edukacyjnych.

Liczba i różnorodność dostępnych na rynku systemów zarządzania bazami danych sprawia, że wykorzystanie relacyjnej bazy do przechowywania danych w tworzonych aplikacjach nie stanowi problemu. Niezależnie od rodzaju tworzonej aplikacji czy dostępnego budżetu możliwe jest dobranie do niej gotowego rozwiązania obsługującego SQL i pozwalającego na wygodne przechowywanie danych. Większe rozwiązania mogą korzystać z większych serwerów baz danych, mniejsze, łącznie z systemami wbudowanymi, mogą zaś wykorzystać prostsze implementacje, jak SQLite czy Firebird.

Proste rozwiązania bazodanowe, jak SQLite czy Firebird, nie spełniają całej gamy wymagań stawianych przed DBMS. Przykładowo, SQLite pomija weryfikację spójności danych. Bardziej rozbudowane rozwiązania realizują wszystkie istotne wymagania. Dotyczy to nie tylko rozwiązań komercyjnych, ale także darmowych, jak choćby PostgreSQLa. Niestety każdy z tych systemów ma też swoje specyficzne rozwiązania i każdy z nich w nieco specyficzny sposób obsługuje SQLa. Na początkowym etapie różnice te są mało zauważalne, przy bardziej zaawansowanych zastosowaniach mogą jednak spowodować zamieszanie. Dlatego w razie wątpliwości warto doczytać szczegóły w dokumentacji używanego systemu. Szczegółowy opis SQLa jest częścią dokumentacji każdego z popularnych systemów zarządzania bazą danych i ten opis jest dostosowany do specyfiki danego produktu.

Przykłady przedstawione w niniejszym opracowaniu testowane były za pomocą serwera PostgreSQL, jednak powinny one działać także w innych systemach. Specyficznym wymaganiem PostgreSQLa jest nazywanie tabel bazodanowych i ich kolumn małymi literami. Ta część SQLa w PostgreSQLu nie jest niestety nieczuła na wielkość liter. Jeśli tabele czy kolumny są nazywane wielkimi literami, w poleceniach SQL konieczne jest umieszczanie ich nazw w cudzysłowach, co jest bardzo niewygodne i mało czytelne. Z tego względu we wszystkich przedstawionych przykładach nazwy tabel i kolumn składają się jedynie z małych liter i takie rozwiązanie polecamy wszystkim korzystającym z tego DBMS.

5.7. Nierelacyjne bazy danych

Relacyjne bazy danych nie są jedynym możliwym modelem baz danych. Zanim zaczęły się rozwijać, stosowano tzw. sieciowe bazy danych. Były one jednak niezbyt wygodne i mocno związane ze specyfiką oprogramowania, któremu służyły. Istnieje jednak więcej modeli baz danych.

Relacyjne bazy danych doskonale pasują do strukturalnego paradygmatu programowania opisanego w rozdziale 5. pracy [40]. Później, gdy coraz częściej zaczęto stosować programowanie obiektowe, zaczęto poszukiwać odpowiednich dla niego rozwiązań bazodanowych. W ten sposób zaczęły rozwijać się *obiektowe bazy danych*. Nie były one jednak w stanie wyprzeć baz danych relacyjnych – szeroko już wtedy stosowanych i posiadających solidne implementacje. Aby umożliwić wygodne operacje na danych przechowywanych w bazach relacyjnych z programów obiektowych, najczęściej stosuje się *mapowanie obiektowo-relacyjne* (tzw. ORM – ang. *Object-Relational Mapping*). W ten sposób programy obiektowe korzystają z relacyjnych baz danych.

W ostatnich latach pojawiło się nowe podejście do przechowywania danych, a wraz z nim nowe rodzaje baz danych. Ich rozwój jest mocno związany z rozwojem Internetu, sieci społecznościowych, z koniecznością przechowywania dużych ilości danych, szybkiego do nich dostępu, ale także z rozluźnieniem wymagań dotyczących ich spójności.

Współczesne aplikacje internetowe potrzebują wydajnych baz danych, które pozwalałyby przechowywać dane w bardzo prostych strukturach klucz – wartość, odpowiadających tablicom asocjacyjnym zaimplementowanym w niektórych językach programowania, ale dostępnych zdalnie (więcej o tablicach asocjacyjnych można znaleźć w rozdziale 5. pracy [40]). Są one wykorzystywane np. jako pamięć *cache* do tymczasowego przechowywania zawartości czy też do przechowywania danych o sesjach użytkowników. Specyficznym wymaganiem aplikacji, zwłaszcza tych, które obsługują sieci społecznościowe, jest bardzo wysoka skalowalność baz danych: powinny one być w stanie przechowywać bardzo duże ilości danych i bardzo szybko je dostarczać bardzo licznym klientom. Wymaga to rozproszenia bazy danych na wiele serwerów. Spełnienie tego wymagania możliwe jest dzięki równoczesnemu rozluźnieniu wymagań dotyczących spójności danych: dopuszczalne jest np. serwowanie przez jakiś czas nieaktualnych danych. Te wymagania są zdecydowanie różne od spotykanych np. w aplikacjach bankowych, gdzie przede wszystkim dane muszą być spójne, a ich struktura jest bardzo złożona.

Istnieje bardzo wiele gotowych implementacji nierelacyjnych systemów zarządzania bazą danych. Serwis `http://nosql-database.org/` w momencie pisania tego tekstu wymienia ich 150! Warto tam zajrzeć, gdyż serwis podejmuje próbę usystematyzowania prezentowanych rozwiązań oraz podaje wiele odnośników.

Wiele języków programowania udostępnia gotowe narzędzia (biblioteki) ułatwiające korzystanie z baz nierelacyjnych, ich użycie w oprogramowaniu jest więc stosunkowo proste. Jeśli więc wydajność SQLowej bazy danych w jakimś zastosowaniu okazuje się zdecydowanie niewystarczająca, struktura przechowywanych danych nie jest skomplikowana czy też więzy integralności nie są najistotniejsze, warto rozważyć zastosowanie bazy nierelacyjnej. Na ogół stosuje się ją obok bazy relacyjnej do przechowywania pewnej części danych, co do których wymagania na to pozwalają.

Różnorodność nierelacyjnych baz danych jest spora, ale ich zastosowania są bardziej specyficzne. Zapewne relacyjne bazy danych nie będą w stanie zająć ich miejsca w specyficznej domenie, dla której zostały stworzone. Równie trudno sobie wyobrazić wyparcie baz relacyjnych przez te rozwiązania. Wiele systemów informatycznych korzysta równocześnie w różnych celach z kilku rodzajów baz danych i w każdym z tych specyficznych zastosowań sprawdza się odpowiednie do nich rozwiązanie.

5.8. Bezpieczeństwo

5.8.1. Użytkownicy i uprawnienia

Jak opisano w podrozdziale 5.6.5, bazy danych oferują złożony system uprawnień. Pozwala on tworzyć użytkowników, definiować ich grupy nazywane *rolami* oraz bardzo dokładnie kontrolować, jakie operacje na jakich danych mogą oni wykonywać.

W najprostszych przypadkach dla pojedynczej bazy jest tworzony tylko jeden użytkownik z pełnymi do niej uprawnieniami. W tych prostych zastosowaniach takie rozwiązanie sprawdza się dobrze – aplikacja korzystająca z bazy danych zapewnia wymagany poziom bezpieczeństwa, autoryzuje operacje wykonywane przez użytkowników. W takich rozwiązaniach dane na ogół nie są krytycznie ważne i ewentualne błędy w aplikacjach służą intruzom raczej do przejęcia kontroli nad serwerem niż do wykradzenia czy uszkodzenia danych.

Poważne zastosowania, jak na przykład systemy bankowe, wymagają jednak bardziej złożonego systemu bezpieczeństwa. W takich rozwiązaniach dla jednej bazy danych tworzy się wielu użytkowników, nadając każdemu uprawnienia jedynie do wykonywania ściśle określonych operacji na części danych. Poszczególne moduły aplikacji korzystające z bazy danych łączą się z nią za pomocą uprawnień odpowiednich użytkowników. Jeśli któryś z modułów okaże się podatny na atak, intruz nie będzie w stanie dostać się do wszystkich danych.

Jeśli z jednej bazy danych korzysta wiele różnych aplikacji, dobrym rozwiązaniem jest utworzenie dla każdej z tych aplikacji osobnego użytkownika z odpowied-

nimi uprawnieniami. Dzięki temu dane będą bezpieczniejsze, a baza łatwiejsza do zarządzania.

5.8.2. SQL injection

Jednym z bardziej niebezpiecznych, a zarazem bardzo częstych ataków na aplikacje korzystające z relacyjnych baz danych i języka SQL jest atak *SQL injection*. Polega on na modyfikacji polecenia wysyłanego przez aplikację do serwera baz danych w taki sposób, aby została wykonana operacja zamierzona przez intruza.

Bardzo rzadko zdarza się, aby aplikacje wysyłały do bazy danych zawsze takie same zapytania. Na ogół treść zapytania jest przygotowywana przez aplikację: sklejana z fragmentów w zależności od potrzeby i gdy jest gotowa – wysyłana do bazy danych. Bardzo często treść takiego zapytania zależy wprost od danych, które podał użytkownik aplikacji. Typowym przykładem jest logowanie do programu. Użytkownik na ekranie logowania podaje swój login i hasło, i w zależności od tego co podał aplikacja przygotowuje zapytanie do bazy danych pozwalające sprawdzić, czy w bazie istnieje użytkownik z podanym loginem i hasłem. Celem jest „wpuszczenie" do aplikacji tylko uprawnionych użytkowników.

Najprostszym sposobem przygotowania opisanego zapytania jest „sklejenie" fragmentów zapytania SQL z danymi pochodzącymi od użytkownika – loginem i hasłem. Jeśli dane podane przez użytkownika byłyby umieszczone w zmiennych `login` i `haslo`, to taką operację można byłoby wykonać tak, jak na listingu 5.53. Jeśli użytkownik poda login `janek` i hasło `tajne_haslo`, zostanie utworzone zapytanie z listingu 5.54. Jest to poprawne zapytanie, które zwróci listę wszystkich użytkowników z podanym loginem i hasłem. Należy jedynie sprawdzić, czy zapytanie zwróciło co najmniej jeden wiersz. Jeśli tak, to znaczy, że użytkownik o podanym loginie i haśle istnieje i uwierzytelnianie się powiodło – użytkownik zalogował się poprawnie.

Listing 5.53
```
1  zapytanie = "SELECT login, haslo FROM uzytkownicy WHERE login='"
2              + login + "' AND haslo='" + haslo + "';"
```

Listing 5.54
```
1  SELECT login, haslo FROM uzytkownicy
2    WHERE login='janek' AND haslo='tajne_haslo';
```

Atakujący, wiedząc jak jest tworzone takie zapytanie, może to wykorzystać, aby zalogować się do systemu, nie znając poprawnego hasła. Jeśli atakujący poda dowolny login użytkownika, a zamiast hasła wpisze tekst „`' OR true --`", treść utworzonego przez aplikację zapytania do bazy danych będzie taka, jak na listingu

5.55. Pierwsza część warunku w tym zapytaniu nakazuje zwrócić wszystkich użytkowników z loginem `janek` i pustym hasłem. Dodanie na końcu `OR true` powoduje jednak, że warunek w `WHERE` będzie spełniony dla każdego rekordu, niezależnie więc od hasła rekordy te zostaną zwrócone w wyniku[14]. Aplikacja sprawdzi, że zapytanie zwróci co najmniej jeden rekord i uzna, że atakujący podał poprawne hasło![15]

Listing 5.55
```
1  SELECT login, haslo FROM uzytkownicy
2    WHERE login='janek' AND haslo='' OR true --';
```

Przedstawiono jedynie najprostszy przykład ataku. Podatność na *SQL injection* bardzo często pozwala na wykonanie w bazie danych dowolnej operacji z uprawnieniami, jakie posiada zaatakowana aplikacja. Można więc dowolnie modyfikować i usuwać dane, a często także je odczytywać.

Aby zabezpieczyć się przed tego typu atakiem, należy użyć rozwiązań oferowanych przez języki i biblioteki wykorzystywane podczas programowania aplikacji. Oferują one specjalne funkcje, które przeglądają dane dołączane do zapytania i w razie potrzeby modyfikują je, dzięki czemu nie są one interpretowane jako część zapytania. W żadnym wypadku nie należy „sklejać" zapytania jak prostego łańcucha tekstowego, tak jak na listingu 5.53, jeśli jest jakakolwiek szansa, że użyte w nim dane pochodzą od użytkownika, a najlepiej nie robić tego nigdy.

5.9. Podsumowanie

Rozwój baz danych znacznie ułatwił przechowywanie danych i wykonywanie na nich operacji. Dzięki temu tworzenie oprogramowania operującego na zbiorach danych, jak również gromadzenie i korzystanie z danych stało się znacznie łatwiejsze i tańsze. Spowodowało to znaczny wzrost ilości gromadzonych danych, zwłaszcza wraz ze spadkiem cen nośników. Dzięki temu obecnie za pośrednictwem Internetu niemal każdy ma łatwy i bardzo szeroki dostęp do różnorodnych informacji, których przygotowanie jeszcze niedawno wymagało wytężonej pracy wielu ludzi, a co za tym idzie było drogie. Bardzo wiele z tych informacji jest przechowywanych w relacyjnych bazach danych i z nich pobieranych przez opro-

[14] Znaki `--` umieszczone na końcu podanego przez atakującego „hasła" to SQLowy znak komentarza. Spowoduje on zignorowanie dalszych znaków, które na końcu zapytania doklei aplikacja (w tym wypadku jest to zamykający apostrof).

[15] Opisany przykład zakłada przechowywanie w bazie danych nieszyfrowanych haseł, co jest oczywistym błędem. Więcej na ten temat można przeczytać w podrozdziale 4.10.1. Modyfikacja zarówno aplikacji, jak i scenariusza ataku w przypadku przechowywania haseł w postaci zaszyfrowanej jest jednak niewielka i w żaden sposób nie zmienia podatności na opisywany atak.

gramowanie (np. obsługujące strony www, serwisy bankowe) i przesyłanych za pośrednictwem Internetu.

Można zatem stwierdzić, że bazy danych są działem informatyki, który osiągnął niebywały sukces. Jakkolwiek czterdzieści lat rozwoju baz danych wydaje się długim czasem w porównaniu z czasem rozwoju samej informatyki – od wynalezienia pierwszego programowalnego komputera minęło niecałe 80 lat – to w porównaniu z czasem, jakiego potrzebowały inne ludzkie osiągnięcia jest on niezmiernie krótki. Bazy danych są nie tylko ważnym działem informatyki, ale też pełnią niezmiernie istotną rolę w funkcjonowaniu współczesnego społeczeństwa, choć w większości przypadków nie jest to zauważane.

5.10. Zadania podstawowe

1. Jakie powinny być tabele w relacyjnej bazie danych i jakie powinny mieć kolumny, jeśli mają przechowywać dane o samochodach i ich właścicielach? Jeden samochód ma jednego właściciela, ale jedna osoba może posiadać wiele samochodów. Na temat samochodów należy przechowywać: markę, kolor, numer rejestracyjny, o właścicielach: imię, nazwisko, numer telefonu. Należy wyjaśnić znaczenie wszystkich innych kolumn występujących w zaprojektowanych tabelach.

2. W relacyjnej bazie danych jest tabela o nazwie `studenci` składająca się z następujących kolumn: `student_id` (liczba), `imie` (tekst), `nazwisko` (tekst), `numer_indeksu` (liczba), `data_urodzenia` (data) oraz tabela `oceny` zawierająca kolumny: `ocena` (liczba), `student_id` (liczba). Pole `student_id` łączy obie tabele. Napisz zapytanie SQL, które zwróci wszystkie rekordy i wszystkie pola z tabeli `studenci`.

3. Dla tabel z zadania 2. napisz zapytanie SQL, które zwróci jedynie nazwisko i datę urodzenia, ale tylko dla osób urodzonych przed 21 lipca 1989 r. (w zapytaniu można porównać pole daty urodzenia z tekstem zawierającym żądaną datę).

4. Dla tabel z zadania 2. napisz zapytanie SQL, które zwróci wszystkie oceny Jana Kowalskiego.

5. Dla tabel z zadania 2. napisz wyrażenie SQL, które usunie z tabeli `oceny` wszystkie oceny 2.0.

5.11. Zadania trudniejsze

1. Zaprojektuj bazę danych do przechowywania danych biblioteki. Baza danych ma przechowywać dane o książkach (autor, tytuł, ISBN), klientach – osobach, które wypożyczają książki (imię, nazwisko, email, telefon), wypożyczeniach książek – jaki pracownik jaką książkę jakiemu klientowi wypożyczył (data wypożyczenia, data zwrotu), pracownikach (imię, nazwisko, telefon służbowy). Baza danych musi umożliwiać uzyskanie informacji wymaganych w zadaniach 3-7. Projekt przedstaw w postaci ERD niezawierającego związków wiele do wielu.

2. Napisz wyrażenia DDL tworzące bazę danych z zadania 1.

3. Napisz zapytanie do bazy danych z zadania 1., które zwróci liczbę wszystkich książek w bazie.

4. Napisz zapytanie do bazy danych z zadania 1., które zwróci tylko imiona i nazwiska wszystkich osób, które wypożyczyły książki 12 stycznia 2004 r.

5. Napisz zapytanie do bazy danych z zadania 1., które zwróci tylko adresy email osób, które kiedykolwiek wypożyczyły książkę pod tytułem *Język C++* autorstwa Bjarne Stroustrupa.

6. Napisz zapytanie do bazy danych z zadania 1., które zwróci tylko imię i nazwisko klienta, imię i nazwisko pracownika, który wypożyczył książkę oraz tytuł książki dla wszystkich wypożyczeń pomiędzy 1 stycznia a 31 marca 2012 r. Wyniki powinny być posortowane według nazwiska, potem imienia klienta rosnąco, potem nazwiska pracownika malejąco.

7. Napisz zapytanie do bazy danych z zadania 1., które zwróci tylko imię i nazwisko klienta oraz średni czas przetrzymywania książek wypożyczonych pomiędzy 1 stycznia a 31 grudnia 2013 r., ale tylko dla przypadków, w których ten średni czas jest dłuższy niż 2 dni.

8. Zaprojektuj bazę danych do przechowywania danych serwisu komputerów. Każdy z komputerów (numer seryjny, model procesora, ilość pamięci RAM, główny system operacyjny) ma swojego właściciela – klienta serwisu (imię, nazwisko, telefon, email, adres). Każdy z komputerów może być wielokrotnie serwisowany, za każdym razem przez konkretnego serwisanta (imię, nazwisko, tel. służbowy, data zatrudnienia). Na temat każdej naprawy serwisat zapisuje: datę jej rozpoczęcia, datę zakończenia, notatkę. Baza danych musi umożliwiać uzyskanie informacji wymaganych w zadaniach 10-14. Projekt przedstaw w postaci ERD niezawierającego związków wiele do wielu.

9. Napisz wyrażenia DDL tworzące bazę danych z zadania 8.

10. Napisz zapytanie do bazy danych z zadania 8. wyświetlające, ile jest w bazie komputerów z głównym systemem operacyjnym o nazwie „Windows 7”.

11. Napisz zapytanie do bazy danych z zadania 8. wyświetlające tylko emaile klientów, którzy kiedykolwiek naprawiali komputery z procesorem „Intel Core i5”.

12. Napisz zapytanie do bazy danych z zadania 8. zwracające tylko nazwiska serwisantów, którzy serwisowali komputery wyposażone w 1024 MB pamięci RAM po 31 grudnia 2011 r.

13. Napisz zapytanie do bazy danych z zadania 8. zwracające dla każdego klienta serwisantów, którzy serwisowali komputery tego klienta. Należy wyświetlić tylko imię i nazwisko klienta oraz imię i nazwisko serwisanta. Wynik ma być posortowany rosnąco według nazwisk klientów, potem według nazwisk serwisantów.

14. Napisz zapytanie do bazy danych z zadania 8., które zwróci dla każdego klienta maksymalny czas trwania naprawy jego komputerów. Należy wyświetlić tylko imię i nazwisko klienta oraz maksymalny czas naprawy, uwzględniając tylko tych klientów, dla których maksymalny czas naprawy był dłuższy niż tydzień.

Podziękowania

Ten rodział nie mógłby powstać, gdyby nie pomoc Jacka Rząsy, mojego Taty, od którego uczyłem się baz danych i którego cenne uwagi pozwoliły ulepszyć tę pracę.

Rozdział 6.

Podsumowanie

Celem skryptu było systematyczne wprowadzenie do zagadnień związanych z bezpieczeństwem systemów informatycznych, sieciami komputerowymi, systemami operacyjnymi oraz bazami danych. Ze względu na ograniczoną objętość publikacji pewne zagadnienia zostały potraktowane w sposób selektywny. Mamy nadzieję, że praca ta stanowi dobre wyjście do podjęcia samodzielnych studiów nad wybranymi działami informatyki, które zainteresowały Czytelnika.

Kontynuację studiów z zakresu bezpieczeństwa systemów informatycznych Czytelnik może rozpocząć od publikacji [6, 20, 22]. Swoje zainteresowania na temat sieci komputerowych może z kolei rozwinąć poprzez studiowanie monografii [5, 19, 48]. Książka [24] zawiera systematyczny kurs omawiający zarówno warstwę sprzętową, jak i programową lokalnych sieci komputerowych wraz z gotowymi przepisami na konfigurację takich sieci z zastosowaniem komputerów działających pod kontrolą systemów operacyjnych Linux i Windows. Osoby zainteresowane samodzielnym konfigurowaniem serwerów internetowych pracujących pod kontrolą systemu Linux powinny sięgnąć do pracy [25]. Dobrym uzupełnieniem wiadomości z dziedziny sieci telefonów komórkowych może być publikacja [21]. Czytelnik zainteresowany poszerzeniem i ugruntowaniem swojej wiedzy na temat zagadnień związanych z systemami operacyjnymi powinien sięgnąć po publikacje [42, 43]. Dobrym rozszerzeniem wiedzy dotyczącej problematyki sieciowych i rozproszonych systemów operacyjnych jest praca [47]. Specjalistyczną wiedzę dotyczącą systemów operacyjnych czasu rzeczywistego oraz technik projektowania oprogramowania dla nich można znaleźć w opracowaniach [45, 44]. Problematykę projektowania, programowania oraz zarządzania bazami danych omawiają prace [12, 51].

Autorzy zachęcają również do sięgnięcia po pierwszy tom niniejszego skryptu obejmujący wprowadzenie do architektury komputerów, algorytmiki, paradygmatów i języków programowania [40].

Spis rysunków

Spis tablic

Bibliografia

[1] *Advanced Encryption Standard (AES)*, Nov. 2001. Federal Information Processing Standards Publications 197. National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[2] ANSI, *ANSI X3.159-1989 Programming Language C*, 1989.

[3] ARINC Specification 664P7. Aircraft Data Network. Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network, 2005.

[4] Bellare M., Namprempre C., *Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm*, [in:] Okamoto T., (ed.), *Advances in Cryptology — ASIACRYPT 2000*, vol. 1976, serii *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, 2000, s. 531–545.

[5] Bradford R., *Podstawy sieci komputerowych*. Wydawnictwa Komunikacji i Łączności, 2009.

[6] Buchmann J. A., *Wprowadzenie do kryptografii*. Wydawnictwo Naukowe PWN, 2006.

[7] Codd E. F., *A Relational Model of Data for Large Shared Data Banks*. Commun. ACM, 13(6):377–387, 1970.

[8] *Data Encryption Standard (DES)*, Oct. 1999. Federal Information Processing Standards Publications 46-3. National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[9] Duong T., Rizzo J., *Here Come The $\oplus$ Ninjas*, May 2011, http://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf.

[10] Dworkin M., *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, May 2004. NIST Special Publication 800-38C. National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf.

[11] Elmasri R., Navathe S. B., *Fundamentals of Database Systems*. Addison-Wesley, 2010.

[12] Garcia-Molina H., Ullman J. D., Widom J., *Systemy baz danych. Kompletny podręcznik*. Helion, 2011.

[13] Grajek M., *Enigma. Bliżej prawdy*. Dom Wydawniczy Rebis, Poznań 2007.

[14] IEEE Computer Society 1003.1-2008/Cor 1-2013 – IEEE Standard for Information Technology – Portable Operating System Interface (POSIX(R)), 2009.

[15] IEEE Computer Society Standard 802.3. IEEE Standard for Ethernet, 2012.

[16] IEEE Computer Societyc Standard 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2012.

[17] ISO/IEC 9075:2011. Information technology – Database languages – SQL, 2011.

[18] Krawczyk H., *The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)*, [in:] Kilian J., (ed.), *Advances in Cryptology — CRYPTO 2001*, vol. 2139, serii *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, 2001, s. 310–331.

[19] Krysiak K., *Sieci komputerowe kompendium*. Wydawnictwo Helion, 2005.

[20] Kutyłowski M., Strothmann W.-B., *Kryptografia. Teoria i praktyka zabezpieczania systemów komputerowych*. Oficyna Wydawnicza Read Me, 1999.

[21] Lal K., Rak T., *Systemy telefonii komórkowej. Wybrane zagadnienia*. Oficyna Wydawnicza Politechniki Rzeszowskiej, Rzeszów 2005.

[22] Menezes A. J., van Oorschot P. C., Vanstone S. A., *Kryptografia stosowana*. Wydawnictwa Naukowo-Techniczne, 2005.

[23] Möller B., Duong T., Kotowicz K., This POODLE Bites: Exploiting The SSL 3.0 Fallback Security Advisory, Sep. 2014. Google, https://www.openssl.org/~bodo/ssl-poodle.pdf.

[24] Rak T., *Budowa i obsługa domowych sieci komputerowych*. Wydawnictwo Helion, 2011.

[25] Rak T., Lal K., *Po prostu własny serwer internetowy*. Wydawnictwo Helion, 2002.

[26] RFC 768. User Datagram Protocol, 1980.

[27] RFC 791. Internet Protocol, 1981.

[28] RFC 793. Transmission Control Protocol DARPA Internet Program Protocol Specification, 1981.

[29] RFC 2460. Internet Protocol, Version 6 (IPv6) Specification, 1998.

[30] RFC 2474. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, 1998.

[31] RFC 3022. Traditional IP Network Address Translator (Traditional NAT), 2001.

[32] RFC 4250. The Secure Shell (SSH) Protocol Assigned Numbers, 2006.

[33] RFC 4291. IP Version 6 Addressing Architecture, 2006.

[34] RFC 4301. Security Architecture for the Internet Protocol, 2005.

[35] RFC 4614. A Roadmap for Transmission Control Protocol (TCP) Specification Documents, 2006.

[36] RFC 4632. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan, 2006.

[37] RFC 4880. OpenPGP Message Format, 2007.

[38] RFC 5246. The Transport Layer Security (TLS) Protocol, 2008.

[39] Rogaway P., Bellare M., Black J., *OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption*. ACM Transactions on Information and System Security (TISSEC), 6(3):365–403, 2003.

[40] Samolej S., Rząsa W., Rzońca D., Sadolewski J., Jędrzejec B., *Wprowadzenie do informatyki I – architektura komputerów, algorytmika, paradygmaty i języki programowania*. Oficyna Wydawnicza Politechniki Rzeszowskiej, Rzeszów 2014.

[41] Shannon C. E., *Communication Theory of Secrecy Systems*. The Bell System Technical Journal, 28(4):656–715, 1949.

[42] Silberschatz A., Galvin P. B., Cagne G., *Podstawy systemów operacyjnych*. Wydawnictwa Naukowo-Techniczne, 2006.

[43] Stallings W., *Systemy operacyjne. Struktura i zasady budowy*. Wydawnictwo Naukowe PWN, 2006.

[44] Szmuc T., Motet G., *Specyfikacja i projektowanie oprogramowania systemów czasu rzeczywistego*. Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH, 2000.

[45] Szymczyk P., *Systemy operacyjne czasu rzeczywistego*. Wydawnictwa AGH, 2003.

[46] Świder K., Dec G., Trybus B., *Inżynieria systemów informatycznych. Podstawy i praktyka budowy systemów oprogramowania*. Oficyna Wydawnicza Politechniki Rzeszowskiej, Rzeszów 2004.

[47] Tanenbaum A. S., van Steen M., *Systemy rozproszone. Zasady i paradygmaty*. Wydawnictwo Naukowe PWN, 2006.

[48] Tanenbaum A. S., Wetherall D. J., *Sieci komputerowe*. Wydawnictwo Helion, 2012.

[49] The Keyed-Hash Message Authentication Code (HMAC), July 2008. Federal Information Processing Standards Publications 198-1. National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

[50] Turan M. S., Barker E., Burr W., Chen L., *Recommendation for Password-Based Key Derivation. Part 1: Storage Applications*, Dec. 2010. NIST Special Publication 800-132. National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf.

[51] Ullman J. D., Widom J., *Podstawowy kurs systemów baz danych*. Helion, 2011.

[52] Ustawa z dnia 18 września 2001 r. o podpisie elektronicznym. Dziennik Ustaw 2001 Nr 130, poz. 1450 wraz z późniejszymi zmianami.

Skorowidz