

Implementacja aplikacji sieciowych z wykorzystaniem interfejsu gniazd BSD

1. Wprowadzenie

Wymagania wstępne: wykonanie ćwiczenia „Adresacja IP”.

Począwszy od wersji 4.1 systemu UNIX BSD, wprowadzono do niego obsługę protokołów TCP/IP wraz z interfejsem dostępu dla programistów o nazwie **gniazda** (ang. *sockets*). Interfejs ten jest także dostępny w systemach Linux, a jego zadaniem jest pośredniczenie pomiędzy programami użytkowników a implementacją protokołów TCP/IP.

Do dyspozycji programistów oddano zestaw funkcji bibliotecznych, które służą do obsługi komunikacji pomiędzy procesami działającymi na różnych węzłach w sieci (możliwa jest także komunikacja pomiędzy procesami w ramach jednej maszyny), które składają się na interfejs sieci. Funkcje te maskują przed programistą warstwę transportową i do zapewnienia komunikacji wymagają utworzenia specjalnego końcowego punktu kanału komunikacji, zwanego właśnie *gniazdem*. W procesie systemowym gniazdo jest traktowane jak plik specjalny (po jego utworzeniu programista uzyskuje odpowiedni deskryptor), co pozwala na realizację funkcji odczytu i zapisu analogicznie jak na pliku (np.: `read()` i `write()`).

1.1 Przyjęty model i tryby komunikacji

Nawiązanie połączenia i rozpoczęcie komunikacji wymaga aby jeden z procesów, w niej uczestniczący nasłuchiwał (oczekiwał) zgłoszeń; wówczas inny proces może nawiązać z nim połączenie w dowolnym, wybranym przez siebie momencie.

Proces, który oczekuje na połączenia nazywamy *serwerem*. Serwer może działać według dwóch schematów: (i) po odebraniu połączenia od innego procesu następuje przetwarzanie przyjętego zgłoszenia oraz (ewentualne) wysłanie wyników wykonanych operacji; dalej serwer podejmuje oczekiwanie na kolejne zgłoszenia – taki serwer nazywamy **iteracyjnym**; (ii) po odebraniu połączenia serwer uruchamia proces współbieżny, który jest odpowiedzialny za dalszą obsługę i komunikację z procesem, który nawiązał połączenie; serwer w tym czasie oczekuje na inne zgłoszenia – taki serwer nazywamy **współbieżnym**.

Proces, który nawiązuje połączenie z serwerem nazywamy *klientem*; proces ten musi posiadać informacje o adresie serwera aby móc nawiązać z nim połączenie; serwer dowiaduje się o adresie klienta dopiero po nawiązaniu przez niego komunikacji.

Interfejs gniazd obsługuje m.in. dwa tryby komunikacji pomiędzy serwerem i klientem: tryb połączeniowy – z wykorzystaniem protokołów TCP/IP – oraz tryb bezpołączeniowy – z wykorzystaniem protokołów UDP/IP.

1.2 Interfejs gniazd

Korzystanie z mechanizmu komunikacji TCP/IP wymaga utworzenia obiektu zwanego *gniazdem*, które jest traktowane przez system jak plik specjalny. Gniazdo jest tworzone przy pomocy funkcji bibliotecznej `socket()`, która w wyniku podaje deskryptor utworzonego gniazda. Oba komunikujące się procesy (klient i serwer) muszą posiadać gniazda; jedno gniazdo może służyć zarówno do odbioru jak i do wysyłania danych. Przed użyciem gniazda należy określić adres – podając adres IP oraz nr portu – dla komputera lokalnego lub dla komputera oddalonego lub dla obu komputerów jednocześnie oraz wskazać tryb komunikacji (np. połączeniowy lub bezpołączeniowy). Na jednym komputerze można utworzyć dwa różne gniazda z takim samym nr portu (i nr IP) ale dla różnych trybów komunikacji.

1.3 Główne funkcje interfejsu gniazd

Aby wykorzystać omawiane funkcje, należy skorzystać z następujących plików nagłówkowych: `<sys/types.h>`, `<sys/socket.h>`, oraz aby korzystać z dodatkowych funkcji – omawianych w dalszej części – należy również użyć plików: `<netinet/in.h>`, `<arpa/inet.h>`, `<netdb.h>`.

`int socket(int domain, int type, int protocol)` – funkcja tworzy nowe gniazdo do komunikacji sieciowej; funkcja w wyniku podaje deskryptor gniazda lub wartość -1 w przypadku błędu. Pierwszy argument funkcji określa domenę komunikacyjną (ang. *communication domain*) i służy do wyboru rodziny protokołów: m.in. `PF_UNIX` (lub `PF_LOCAL`) – protokoły wewnętrzne systemu, `PF_INET` – protokoły internetowe (TCP/IP) w wersji nr 4. Drugi argument określa tryb komunikacji: m.in. `SOCK_STREAM` – gniazdo strumieniowe, `SOCK_DGRAM` – gniazdo datagramowe, `SOCK_RAW` – gniazdo surowe (ang. *raw*). Trzeci parametr specyfikuje protokół, jeśli w danej domenie komunikacyjnej i w wybranym trybie komunikacji dostępnych jest kilka protokołów – najczęściej parametr ten przyjmuje wartość 0.

`int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)` – funkcja ta pozwala na związanie wcześniej utworzonego gniazda z adresem lokalnej maszyny i jest głównie używana przez serwery. Poprawnie wykonana funkcja podaje wartość 0, a błąd jest oznaczany wartością -1. Pierwszy parametr funkcji to deskryptor gniazda, drugi parametr to adres lokalnej maszyny (odpowiednia struktura została opisana poniżej), a trzeci parametr powinien zawierać rozmiar struktury adresu (drugiego argumentu).

`int listen(int sockfd, int backlog)` – funkcja pozwala na przygotowanie utworzonego już gniazda do odbierania zgłoszeń oraz umożliwia określenie rozmiaru kolejki żądań, oczekujących na wykonanie funkcji `accept()`. Wynikiem poprawnie wykonane funkcji jest wartość 0, błąd jest oznaczany wartością -1. Pierwszym parametrem funkcji jest deskryptor gniazda, a drugim rozmiar kolejki docierających żądań połączeń.

`int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)` – funkcja służy do pobrania zgłoszenia z kolejki lub oczekiwania na takie zgłoszenie (funkcja działa w sposób blokujący) i obsługuje tylko gniazda strumieniowe. W wyniku działania funkcji otrzymujemy nowy deskryptor gniazda, a w przypadku błędu otrzymujemy wartość -1. Parametrami tej funkcji są kolejno: deskryptor gniazda, wskaźnik na strukturę adresową (opisaną poniżej), która zostanie wypełniona danymi – adres i numer portu – maszyny, która nawiązała połączenie, wskaźnik na zmienną całkowitą, która będzie zawierała rozmiar struktury adresowej.

`int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen)` – funkcja umożliwia związanie wcześniej utworzonego gniazda z adresem odległego punktu końcowego; wynikiem poprawnego działania funkcji jest wartość 0, błąd jest oznaczany wartością -1. Jeśli gniazdo zostało utworzone w trybie połączeniowym, to funkcja ta nawiązuje połączenie z serwerem, jeśli natomiast gniazdo zostało utworzone w trybie bezpołączeniowym, to funkcja ta jedynie przypisuje gniazdu adres maszyny zdalnej. Pierwszym parametrem funkcji jest deskryptor gniazda, drugi parametr to adres maszyny odległej (odpowiednia struktura została opisana poniżej), a trzeci parametr powinien zawierać rozmiar struktury adresu (drugiego argumentu).

`int close(int fd)` – funkcja zamyka gniazdo i usuwa jego deskryptor; poprawnie wykonana funkcja przekazuje wartość 0, a błąd jest oznaczany wartością -1. Jako parametr należy podać deskryptor zamykanego gniazda.

Do odbierania i wysyłania danych należy używać funkcji `read()` oraz `write()` – posługując się deskryptorem gniazda. W przypadku komunikacji bezpołączeniowej należy używać funkcji `recv()/recvfrom()` oraz `send()/sendto()`, które pozwalają na określanie parametrów przesyłania komunikatu, a ponadto funkcja `sendto()` pozwala na podanie adresata komunikatu.

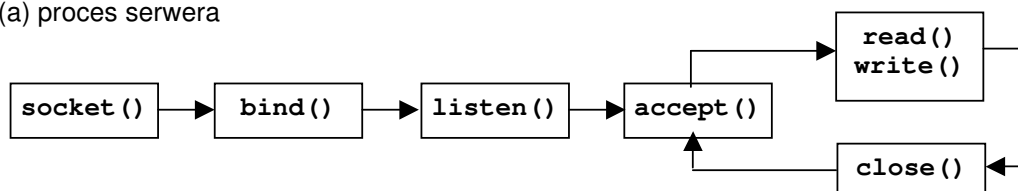
Poniżej przedstawiono struktury używane przez funkcje interfejsu gniazd (uwaga: w domenie protokołów internetowych używamy zamiast struktury `sockaddr` strukturę `sockaddr_in`, wykonując przy przekazywaniu wskaźnika rzutowanie na strukturę (`struct sockaddr*`):

```
struct sockaddr_in
{
    u_short sin_family;      // PF_INET
    u_short sin_port;       // numer portu
    struct in_addr sin_addr; // adres węzła
    char sin_zero[8];       // nie używane
}

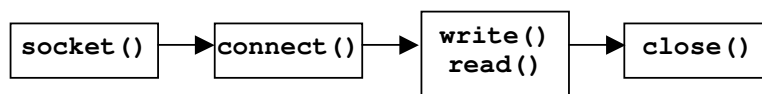
struct in_addr
{
    u_long s_addr;          // 32-bitowy adres
}
```

Rysunek nr 1 przedstawia kolejność wywołań kolejnych funkcji interfejsu gniazd dla obu stron: klienta i serwera.

(a) proces serwera



(b) proces klienta



Rysunek nr 1. Przykładowa kolejność wywołań funkcji interfejsu gniazd dla procesów (a) serwera, (b) klienta.

1.4 Funkcje pomocnicze

Dla protokołów TCP/IP przyjęto standard określający **sieciowy porządek bajtów** – kolejność bajtów reprezentujących liczbę, następuje od najbardziej do najmniej znaczącego. Niektóre parametry funkcji interfejsu gniazd należy podawać zgodnie z tym właśnie porządkiem, np. numer portu w strukturze `sockaddr_in`. Dostępne są zatem funkcje, które umożliwiają konwersję z porządku danego komputera do porządku sieciowego i odwrotnie (istnieją dwa zestawy tych funkcji: dla liczb 16-bitowych oraz dla liczb 32-bitowych).

`uint32_t htonl(uint32_t hostlong)` – funkcja konwersji z reprezentacji lokalnej do sieciowej dla liczb 32-bitowych.

`uint16_t htons(uint16_t hostshort)` – funkcja konwersji z reprezentacji lokalnej do sieciowej dla liczb 16-bitowych.

uint32_t ntohs(uint32_t netlong) – funkcja konwersji z reprezentacji sieciowej do lokalnej dla liczb 32-bitowych.

uint16_t ntohs(uint16_t netshort) – funkcja konwersji z reprezentacji sieciowej do lokalnej dla liczb 16-bitowych.

in_addr_t inet_addr(const char *cp) – funkcja zamienia adres zapisany w postaci XX.XX.XX.XX na jego reprezentację sieciową.

Pozostałe funkcje, operujące na adresach i nazwach, które są użyteczne dla aplikacji sieciowych:

struct hostent *gethostbyaddr(const char *addr, int len, int type) – funkcja w wyniku przekazuje informację na temat komputera o znanym adresie IP.

struct hostent *gethostbyname(const char *name) – funkcja pozwala na odwzorowanie nazwy domenowej komputera na jego adres IP.

int getpeername(int sockfd, struct sockaddr *name, socklen_t *namelen) – funkcja w wyniku przekazuje adres zdalnego komputera, z którym jest połączone wskazane gniazdo. Funkcja ta może być użyta przez serwer do rozpoznania adresu klienta, który nawiązał komunikację.

2 Organizacja, wymagany sprzęt, oprogramowanie

- Zadanie wykonywane jest przez parę studentów;
- sprzęt: 2 komputery;
- oprogramowanie: kompilator *cc* (lub *gcc*).

3 Zadania

1. Na jednym z komputerów należy uruchomić serwer *daytime*; zadanie polega na napisaniu programu sieciowego – klienta – który uruchomiony na drugim komputerze odczyta informacje z uruchomionego serwera. Należy użyć trybu połączeniowego.
2. Napisać dwa programy: klienta i serwera; program klient przesyła do serwera ciąg znaków, na który składa się nr indeksu studenta, na co program serwera odpowiada ciągiem znaków, będącym nazwiskiem studenta o wskazanym nr indeksu. Program należy napisać w dwóch wersjach połączeniowej i bezpołączeniowej.
3. Wzbogacić powyżej opisane programy o obsługę nazw domenowych.

4 Pytania sprawdzające

1. Wyjaśnić różnice pomiędzy serwerami: bezpołączeniowym-iteracyjnym, bezpołączeniowym-współbieżnym, połączeniowym-iteracyjnym i połączeniowym-współbieżnym.
2. W jakich sytuacjach może dojść do zakleszczenia (ang. *deadlock*) serwera?
3. Do czego można użyć funkcji `connect()`, w trybie bezpołączeniowym?
4. Jak może mieć zastosowanie drugi deskryptor gniazda uzyskany funkcją `accept()`?
5. Jak może być zastosowanie funkcji systemowej `select()` w implementacji serwera sieciowego?

5 Literatura

1. Comer D.E, Stevens D.L.: *Sieci komputerowe TCP/IP. Programowanie w trybie klient-serwer. Wersja BSD.* (Tom III), Wydawnictwa Naukowo-Techniczne, 1997.
2. A.S. Tanenbaum: *Computer Networks*, Prentice Hall PTR, 2003.
3. M. Gabassi, B. Dupouy: *Przetwarzanie rozproszone w systemie UNIX*, LUPUS, 1995.