Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Gniazda TCP i UDP”

laboratorium: 01
system operacyjny: Linux

Kraków, 2009

Spis treści

 Copyright © 2009 Politechnika Krakowska

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące programowania aplikacji klient-serwer, a w szczególności programowania z wykorzystaniem protokołów warstwy transportowej modelu ISO/OSI: **TCP i UDP**. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie gniazd w oparciu o **protokół TCP** oraz **protokół UDP**. Gniazda (ang. *sockets*) są najbardziej uniwersalnym sposobem komunikacji spośród mechanizmów IPC (ang. *Inter Process Communication* - komunikacja międzyprocesowa). Jeśli dwa procesy mają się między sobą komunikować, każdy z nich tworzy po swojej stronie jedno gniazdo. Parę takich gniazd (końcówek kanału komunikacyjnego) można określić mianem asocjacji. Gniazda reprezentują najczęściej dwukierunkowy punkt końcowy połączenia – dwukierunkowość oznacza możliwość wysyłania i przyjmowania danych.
Gniazd używa się głównie do komunikacji z odległym procesem za pośrednictwem sieci, jednak można je zastosować także w przypadku wymiany informacji między procesami działającymi w obrębie jednej maszyny. Ta uniwersalność zastosowań jest zapewniona dzięki istnieniu różnych odmian gniazd.

Gniazdo posiada następujące właściwości:

- domenę komunikacyjną (rodzinę protokołów), w obrębie której odbywa się komunikacja,
- typ gniazda (określający sposób transmisji danych),
- protokół właściwy dla danej domeny komunikacyjnej i typu gniazda,
- informacje adresowe (np.: adres, numer portu), aby mogła nastąpić komunikacja.

 Copyright © 2009 Politechnika Krakowska

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- budowy modelu ISO/OSI
- znajomości budowy nagłówka protokołu TCP [2]
- znajomości budowy nagłówka protokołu UDP [3]
- obsługi programów **tcpdump** oraz **netstat** [5, 6]
- porządku bajtów: big-endian, little-endian [1]
- protokołu HTTP/1.1 [4]

Należy także zapoznać się z zagadnieniami dotyczącymi **gniazd TCP i UDP**: [1]

- numeracja portów
- nawiązywanie i zakończenie połączenia TCP
- stany połączeń TCP
- funkcje gniazd TCP
- funkcje gniazd UDP
- operacje wejścia/wyjścia: funkcja `select()`

Literatura:

[1] W.R. Stevens, *„Programowanie Usług Sieciowych"*, *„API: gniazda i XTI"*
[2] IETF (http://www.ietf.org/), RFC 793, *„Transmission Control Protocol"*
[3] IETF, RFC 768, *„User Datagram Protocol"*
[4] IETF, RFC 2616, *„Hypertext Transfer Protocol -- HTTP/1.1"*
[5] MAN (8), http://linux.about.com/od/commands/l/blcmdl8_netstat.htm
[6] MAN (8), http://www.tcpdump.org/tcpdump_man.html

1.3. Opis laboratorium

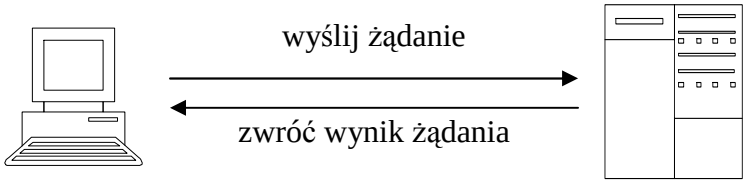
Laboratorium koncentruje się na czwartej warstwie modelu referencyjnego ISO/OSI - warstwie transportowej. Opierając się na pracujących w niej protokołach TCP i UDP przedstawione zostaną zagadnienia związane z gniazdami (punktami końcowymi), portami oraz z technologią klient-serwer. W celu wykonania ćwiczeń praktycznych przydatne mogą być przedstawione w tym podrozdziale informacje teoretyczne.

1.3.1. Klient-serwer

Architektura sieciowa, w której każdy komputer lub program jest klientem (jednostką pobierająca dane), albo serwerem (jednostką je udostępniającą). Serwer przechowuje informacje, przyjmuje zgłoszenia od klientów i świadczy im pewne usługi. W oparciu o model klient-serwer projektowana jest dziś większość sieci komputero-

 Copyright © 2009 Politechnika Krakowska

wych. Został on także wbudowany w grupę podstawowych protokołów Internetu TCP/IP.
Na poniższym rysunku został przedstawiony schemat typowej komunikacji pomiędzy klientem a serwerem TCP.



Rys. 1. Komunikacja klienta z serwerem.

Pierwszym etapem w obu procesach jest utworzenie gniazd (*socket*). Serwer musi dodatkowo przypisać gniazdu numer portu, na którym oczekiwać będzie na nadchodzące połączenia. Po otrzymaniu i zaakceptowaniu żądania serwer przetwarza je, a następnie odsyła odpowiedź. Wymiana informacji trwa do chwili, kiedy klient zamknie połączenie, wysyłając do serwera znacznik końca pliku. Następnie również serwer zamyka ze swojej strony połączenie i kończy działanie lub oczekuje na nowe zgłoszenie.

Port:

Adres wewnętrzny, który zapewnia interfejs pomiędzy aplikacją, a protokołem warstwy transportowej. Każde połączenie TCP lub UDP odbywa się przez port, reprezentowany jako 16-bitowa liczba z zakresu od 1 do 65 535.

Gniazdo:

Mechanizm komunikacyjny, umożliwiający transmisje danych pomiędzy urządzeniami w sieci.

GNIAZDO = ADRES IP + NUMER PORTU

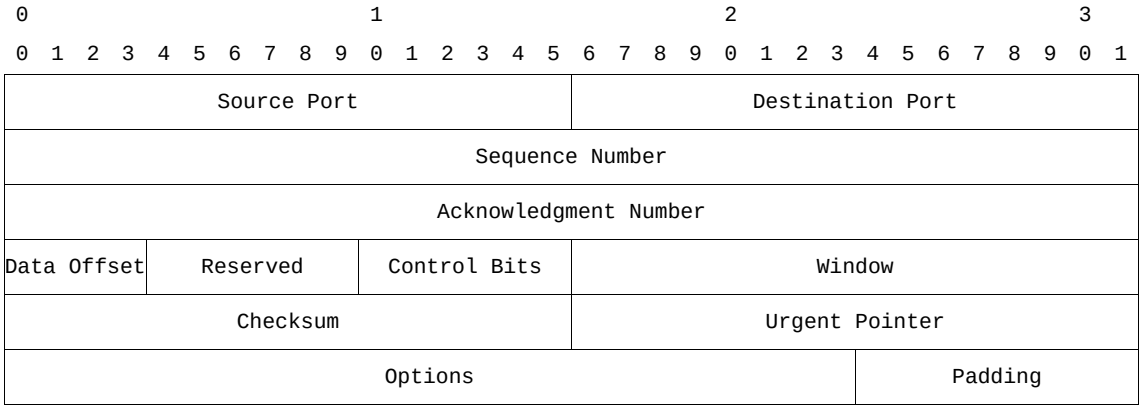
1.3.2. Podstawy protokołu TCP

TCP (ang. *Transmission Control Protocol*) jest zorientowanym połączeniowo protokołem warstwy transportowej. Umożliwia on zestawianie połączeń, w których efektywnie i niezawodnie przesyła się dane. Transmisja za pomocą TCP odbywa się z zachowaniem kolejności danych i jest kontrolowana przez mechanizmy przeprowadzania retransmisji i sterowania przepływem danych.

TCP organizuje dwukierunkową współpracę między warstwą sieciową, a warstwami wyższymi, uwzględniając przy tym wszystkie aspekty priorytetów i bezpieczeństwa. Protokół ten prawidłowo obsługuje niespodziewane zakończenie aplikacji, do której właśnie wędruje pakiet, jak również skutecznie izoluje warstwy wyższe - w szczególności aplikacje użytkownika - od skutków awarii w warstwie sieciowej modelu referencyjnego ISO/OSI. Scentralizowanie wszystkich tych aspektów w jednym protokole umożliwia znaczną oszczędność nakładów na projektowanie oprogramowania.

 Copyright © 2009 Politechnika Krakowska

Pomimo związku z protokołem IP, TCP jest protokołem w pełni niezależnym i może zostać zaadaptowany do wykorzystania z innymi systemami dostarczania danych.

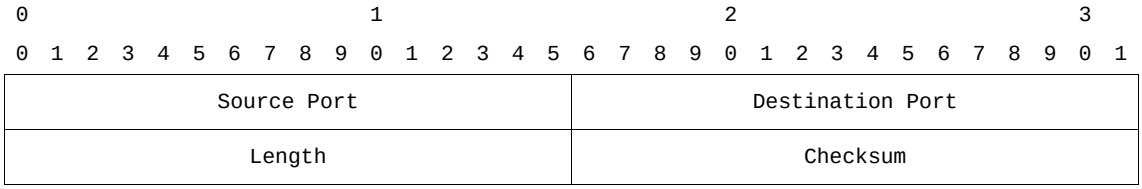


Rys. 2. Format nagłówka TCP[2].

1.3.3. Podstawy protokołu UDP

UDP (*ang. User Data Protocol*) jest bezpołączeniowym protokołem warstwy transportowej modelu referencyjnego ISO/OSI. Protokół ten nie ustanawia w żaden sposób połączenia i nie sprawdza gotowości odległego komputera do odebrania danych. W zamian za to zmniejszona została ilość informacji kontrolnych przesyłanych w sieci i złożoność samego protokołu. Protokół UDP stanowi prosty sposób wymiany datagramów - bez mechanizmu potwierdzeń i kontroli przepływu danych. Czasochłonne nawiązywanie połączenia i potwierdzanie odebranych pakietów nie zawsze są korzystne. Na przykład podczas transmisji danych multimedialnych, potwierdzenia zwrotne mogłyby pogorszyć jakość odtwarzania.

Protokół UDP nie wykorzystuje mechanizmów okienkowania ani potwierdzeń. Niezawodność - jeśli jest wymagana - musi być zapewniona przez protokoły warstw wyższych (aplikację). Program wykorzystujący do transportu protokół UDP musi opcjonalnie zapewnić: retransmisję zagubionych danych, kontrolę przepływu, unikanie przeciążeń w sieci, fragmentację i ponowne składanie strumienia danych. UDP zapewnia jedynie mechanizm prostej sumy kontrolnej, która jest obliczana na podstawie nagłówka i danych datagramu. Ze względu na prosty nagłówek, brak mechanizmów potwierdzeń i sterowania przepływem danych, stosowanie UDP zwiększa efektywność transmisji danych w sieci.



Rys. 3. Format nagłówka UDP[3].

 Copyright © 2009 Politechnika Krakowska

1.3.4. Podstawowe funkcje gniazd

socket

Funkcja `socket()` wywoływana jest w celu utworzenia gniazda komunikacyjnego. Zwraca liczbę całkowitą, za pomocą której można odwołać się do stworzonego gniazda (deskryptor) lub -1 w przypadku błędu (podobnie wszystkie inne funkcje operujące na gniazdach). Gniazda charakteryzują się trzema atrybutami: rodziną protokołów, typem i protokołem. Mają także adres, który używany jest jako ich nazwa. Adresy mają różne formaty, w zależności od rodziny protokołów (ang. *protocol family*).

```
#include <sys/socket.h>

int socket(int domain,  int type, int protocol);
```

Argument **domain** określa rodzinę protokołów, za pomocą której nastąpi komunikacja poprzez gniazdo. Najczęściej stosowane rodziny protokołów to:

Parametr	Opis
PF_INET	protokoły IPv4
PF_INET6	protokoły IPv6
PF_LOCAL	protokoły dziedziny Unix

Argument **type** określa typ gniazda. Najczęściej wykorzystywane typy to:

Parametr	Opis
SOCK_STREAM	gniazdo strumieniowe
SOCK_DGRAM	gniazdo datagramowe
SOCK_RAW	gniazdo surowe

Podstawowe typy gniazd:

- *SOCK_STREAM* - oferują połączenia w postaci uporządkowanego i niezawodnego strumienia bajtów. Oznacza to, iż mamy gwarancję, że przesyłane dane nie zostaną zagubione, zduplikowane albo przemieszczone bez ostrzeżenia, że wystąpił błąd.
- *SOCK_DGRAM* - nie nawiązują i nie podtrzymują połączenia. Istnieje także limit rozmiaru datagramu, który można przesłać. Datagram jest transmitowany jako pojedynczy komunikat, który może zostać zagubiony, zduplikowany, bądź też nadejść w niewłaściwej kolejności.

 Copyright © 2009 Politechnika Krakowska

Argument **protocol** określa protokół dla danej rodziny protokołów i typu gniazda (zero oznacza domyślny protokół dla danej rodziny i typu gniazda).

connect

W przypadku protokołu TCP, funkcja `connect()` ustanawia połączenie klienta z serwerem poprzez inicjację trójfazowego połączenia (ang. three-way handshake).
W przypadku protokołu UDP, funkcja pozwala na skojarzenie adresu zdalnego z gniazdem. Dzięki temu zamiast funkcji `sendto()` mogą zostać wykorzystane funkcje `send()` oraz `write()` (dane wysyłane są wówczas na adres określony przez funkcję `connect()`), a zamiast `recvfrom()` – funkcje `recv()` oraz `read()`.

```
#include <sys/socket.h>

int connect(int sockfd,const struct sockaddr *servaddr,socklen_t ddrlen);
```

Parametr	Opis
`sockfd`	deskryptor gniazda
`*servaddr`	wskaźnik do gniazdowej struktury adresowej
`addrlen`	rozmiar struktury gniazdowej

Funkcje operujące na gniazdach powstały przed opracowaniem standardu ANSI C, (typ `void` nie istniał). Struktura `sockaddr` jest ogólną strukturą gniazdową (wskaźnik do niej można traktować jak wskaźnik na typ `void`). Wskaźniki na struktury adresowe określonych protokołów (np. protokołów IPv4 i IPv6) rzutuje się na wskaźnik do struktury `sockaddr`. Dzięki zastosowaniu wskaźnika do struktury `sockaddr`, funkcje operujące na gniazdach posiadają jednolity interfejs umożliwiający przyjmowanie struktur adresowych różnych protokołów.

Poniżej przedstawiona jest struktura `sockaddr_in` przechowująca informacje adresowe dla protokołu IPv4:

```
#include <netinet/in.h>

struct sockaddr_in {
      short int       sin_family; /* AF_INET */
      unsigned short  sin_port;   /* Numer portu */
      struct in_addr  sin_addr;   /* Adres IP */
      unsigned char   sin_zero[8] /* Wyzerowane */
};
```

Liczba pól struktury może różnić się w zależności od systemu operacyjnego, ale zawsze występują 3 pola:

 Copyright © 2009 Politechnika Krakowska

`sin_family` – domena adresowa (`AF_INET` dla IPv4),
`sin_port` – numer portu przechowywany w porządku sieciowym (*big endian*),
`sin_addr` – adres IP w porządku sieciowym; struktura przedstawiona jest poniżej:

```
#include <netinet/in.h>

struct in_addr {
      unsigned int s_addr;
};
```

Konwencją jest wyzerowanie całej struktury `sockaddr_in` przed jej użyciem, np. za pomocą funkcji `memset()`.

bind

Funkcja `bind()` przypisuje gniazdu nazwę (dla protokołu IP jest to kombinacja adresu IP oraz 16 bitowego numeru portu).

```
#include <sys/socket.h>

int bind(int sockfd,  const struct sockaddr *addr, socklen_t addrlen);
```

Parametr	Opis
`sockfd`	deskryptor gniazda
`*addr`	wskaźnik do struktury adresowej właściwej dla danego protokołu
`addrlen`	rozmiar struktury gniazdowej

Struktura adresowa może określać:

1. konkretny numer portu - najczęściej w przypadku serwera,
2. numer portu równy 0 – system operacyjny wybierze port efemeryczny (z dostępnych portów),
3. konkretny adres,
4. adres nieokreślony (ang. *wildcard address*) – w przypadku IPv4 jest to `INADDR_ANY`.

listen

Funkcja `listen()` jest wywoływana tylko przez serwer. Przekształca gniazdo niepołączone w gniazdo bierne, które będzie użyte do akceptacji przychodzących połączeń za pomocą funkcji `accept()`.

```
#include <sys/socket.h>

int listen(int sockfd,  int backlog);
```

 Copyright © 2009 Politechnika Krakowska

Parametr	Opis
sockfd	deskryptor gniazda
backlog	długość kolejki – ogranicza liczbę połączeń oczekujących dla danego gniazda

accept

Funkcja accept() jest używana w przypadku protokołów zorientowanych połączeniowo. Pobiera pierwsze połączenie z kolejki połączeń oczekujących na zaakceptowanie, tworzy nowe gniazdo do celów komunikacji z klientem i zwraca deskryptor gniazda. Nowe gniazdo będzie miało ten sam typ, co gniazdo zdefiniowane w wywołaniu funkcji listen().

```
#include <sys/socket.h>

int accept(int sockfd,  struct sockaddr *cliaddr, socklen_t *addrlen);
```

Parametr	Opis
sockfd	deskryptor gniazda
*cliaddr	wskaźnik do struktury adresowej wypełnianej przez funkcję accept() adresem gniazda partnera (klienta)
*addrlen	określa rozmiar struktury wskazanej przez cliaddr

close

Wywołanie funkcji close() zmniejsza liczbę deskryptorów gniazda o jeden (deskryptor można traktować jak referencję do gniazda; może istnieć wiele deskryptorów dla tego samego gniazda, np. w procesie macierzystym i potomnym). Po wywołaniu funkcji, deskryptor gniazda nie może zostać użyty przez proces jako argument funkcji operujących na gniazdach, np.: write()/read(), send()/recv(), sendto()/recvfrom(). Jeżeli po wywołaniu funkcji close() liczba deskryptorów osiągnie zero, to gniazdo jest zamykane, a w przypadku protokołu TCP zostaje zainicjowane poprawne zamknięcie połączenia (przez wysłanie FIN).

```
#include <unistd.h>

int close(int sockfd);
```

sendto

Funkcja sendto() jest wykorzystywana do wysyłania danych na określony adres. Stosowana jest najczęściej w przypadku protokołów bezpołączeniowych (np. UDP) – pozwala określić adres na jaki ma być wysłany datagram.

 Copyright © 2009 Politechnika Krakowska

```
#include <sys/socket.h>

ssie_t sendto(int sockfd, const void *buff, size_t nbytes, int flags,
              const struct sockaddr *to, socklen_t addrlen);
```

Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, z którego wysyła się dane
nbytes	liczba wysyłanych bajtów
flags	opcjonalne flagi
*to	wskaźnik na strukturę zawierającą adres pod który mają być wysłane dane
addrlen	rozmiar struktury gniazdowej

send

Funkcja send() jest stosowana w przypadku gniazd skojarzonych z adresem zdalnym (za pomocą funkcji connect() lub acccept()).

```
send(sockfd, buff, nbytes, flags);
jest równoważne z wywołaniem:
sendto(sockfd, buff, nbytes, flags, NULL, 0);
```

recvfrom

Funkcja recvfrom() jest wykorzystywana do odbierania danych. Stosowana jest najczęściej w przypadku protokołów bezpołączeniowych – pozwala określić adres z jakiego otrzymany został datagram.

```
#include <sys/socket.h>

ssie_t recvfrom(int sockfd, void *buff, size_t nbytes, int flags,
                const struct sockaddr *from, socklen_t *addrlen);
```

Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, do którego pobiera się dane
nbytes	rozmiar bufora w bajtach
flags	opcjonalne flagi
*from	wskaźnik do struktury adresowej wypełnianej przez funkcję recfrom() adresem gniazda procesu partnera

 Copyright © 2009 Politechnika Krakowska

*addrlen	wskaźnik do rozmiaru struktury gniazdowej

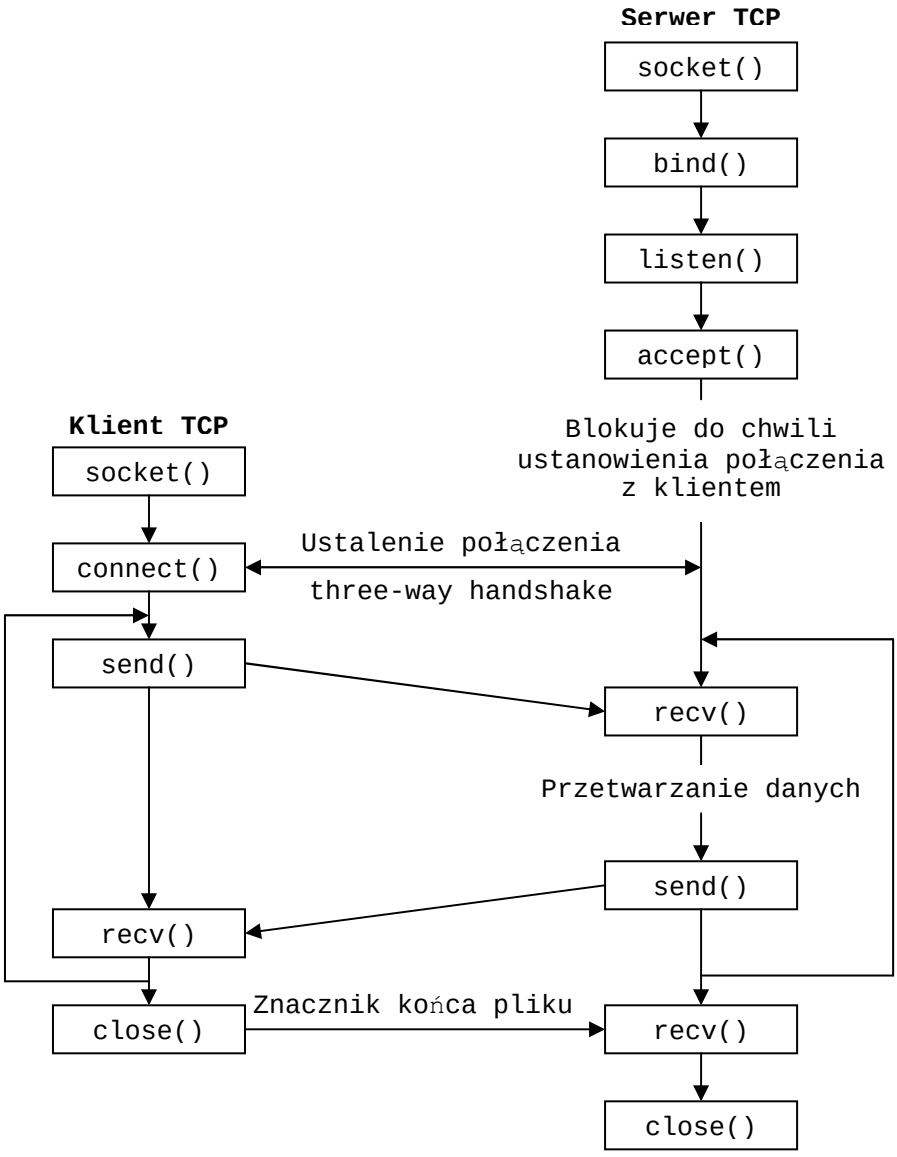
recv

Funkcja recv() jest stosowana w przypadku gniazd skojarzonych z adresem zdalnym (za p mocą funkcji connect() lub acccept()).

```
recv(sockfd, buff, nbytes, flags);
jest równoważne z wywołaniem:
recvfrom(sockfd, buff, nbytes, flags, NULL, NULL);
```

1.3.5. Typowe scenariusze komunikacji klient-serwer

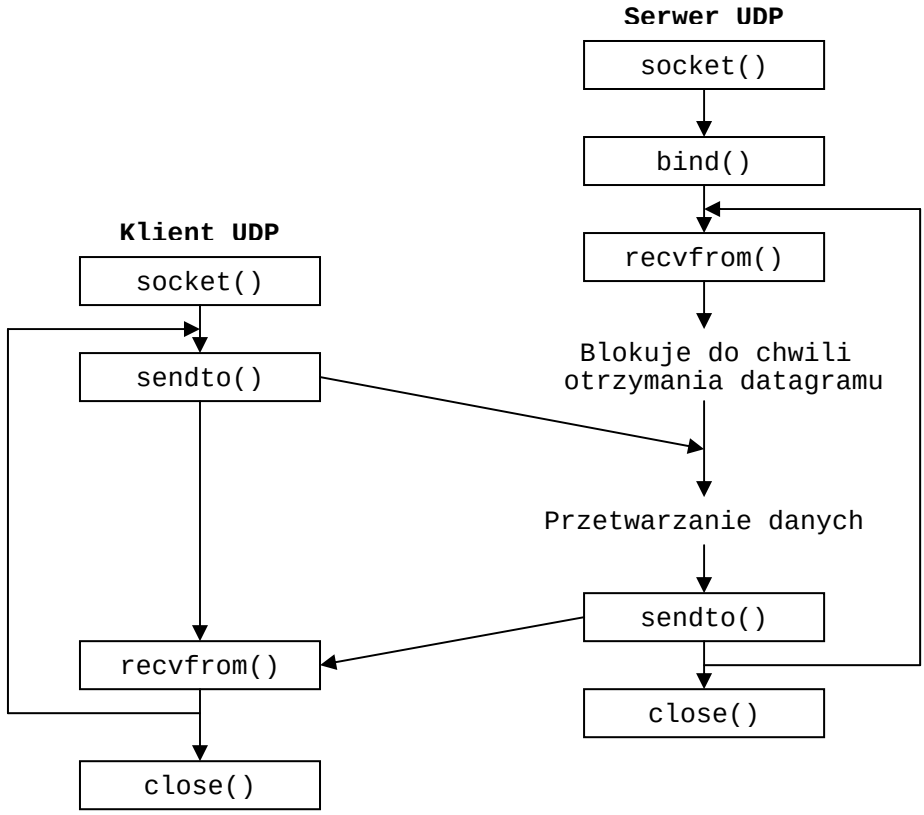
Przebieg typowej komunikacji między procesami klienta i serwera TCP:



Rys. 4. Przebieg typowej komunikacji między procesami klienta i serwera TCP.

 Copyright © 2009 Politechnika Krakowska

Przebieg typowej komunikacji między procesami klienta i serwera UDP:



Rys. 5. Przebieg typowej komunikacji między procesami klienta i serwera UDP.

1.4. Cel laboratorium

Celem laboratorium jest poznanie elementarnych technik przesyłania prostych komunikatów między stacjami komputerowymi w sieci IP. Techniki te są bazą w tworzeniu zaawansowanych i wydajnych aplikacji klient-serwer. Podczas realizacji tego laboratorium zapoznasz się z:

- programowaniem warstwy transportowej przy użyciu protokołów TCP oraz UDP,
- z możliwościami protokołów TCP i UDP,
- z zaletami oraz wadami tych protokołów,

Dzięki laboratorium nabędziesz praktycznych umiejętności w tworzeniu aplikacji klient-serwer za pomocą gniazd.

 Copyright © 2009 Politechnika Krakowska

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Zadanie 1. Komunikacja klient-serwer TCP

Zadanie polega na analizie przykładowych programów klienta i serwera (datetime) oraz zaobserwowaniu przebiegu komunikacji pomiędzy nimi. W tym celu na jednej ze stacji roboczych należy uruchomić **sniffer**. Sniffer jest programem służącym do przechwytywania i analizy danych przepływających w sieci. Za jego pomocą można wyświetlić nagłówki protokołów oraz dane pojawiające się na wybranym interfejsie sieciowym.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami (rozpakować je w razie konieczności).
2. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client1 client1.c
   $ gcc –o server1 server1.c
   ```
3. Uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).
4. Uruchomić serwer TCP podając wybrany przez siebie numer portu:
   ```
   $ ./server1 <numer portu>
   ```
5. Uruchomić klienta i przeanalizować działanie programów:
   ```
   $ ./client1 <adres IP> <numer portu>
   ```
6. Za pomocą sniffera zaobserwować trójfazowe połączenie TCP. Szczególną uwagę należy zwrócić na numery jakie przesyłane są w polach *Sequence Number* oraz *Acknowledgment Number* nagłówka TCP.
7. Za pomocą programu **netstat** zaobserwować stany w jakich znajdują się połączenia TCP w trakcie działania programów i tuż po ich zakończeniu (aktywne zamknięcie połączenia przez klienta i otrzymanie FIN od serwera powoduje przejście połączenia w stan TIME_WAIT). W tym celu należy wywołać program netstat z opcjami: `-nat`.

 Copyright © 2009 Politechnika Krakowska

2.2. Zadanie 2. Komunikacja klient-serwer UDP

Zadanie polega na analizie przykładowych programów klienta i serwera (echo) oraz zaobserwowaniu przebiegu komunikacji pomiędzy nimi. W tym celu na jednej ze stacji roboczych należy uruchomić sniffer.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client2 client2.c
   $ gcc –o server2 server2.c
   ```
3. Uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).
4. Uruchomić serwer UDP podając wybrany przez siebie numer portu:
   ```
   $ ./server2 <numer portu>
   ```
5. Przeanalizować wynik polecenia: `netstat –nau` dla programu serwera.
6. Uruchomić klienta i przeanalizować działanie programów:
   ```
   $ ./client2 <adres IP> <numer portu> <wiadomość>
   ```
7. Za pomocą sniffera zaobserwować przebieg komunikacji między klientem i serwerem.

2.3. Zadanie 3. Kient-serwer UDP: kojarzenie adresu zdalnego z gniazdem.

Celem zadania jest zaimplementowanie programów klienta i serwera UDP. Na przykładzie klienta należy zademonstrować możliwość kojarzenia adresu zdalnego z gniazdem (jest to możliwe przy użyciu funkcji `connect()`). Po skojarzeniu adresu zdalnego z gniazdem, odbieranie danych możliwe jest za pomocą funkcji `recv()`, a do wysyłania danych należy posłużyć się funkcją `send()`.

Zadaniem klienta jest przesyłanie danych wprowadzanych z klawiatury do serwera.
Program klienta należy zaimplementować w oparciu o poniższe założenia:

1. Adres IP i numeru portu serwera są argumentami wywołania programu.
2. Klient w pętli oczekuje na dane do przesłania.
3. Dane (ciąg znaków) pobierane są ze standardowego wejścia i przesyłane do serwera po wciśnięciu ENTER.
4. Wciśnięcie ENTER bez uprzedniego wprowadzenia danych powoduje wysłanie datagramu UDP bez danych (sam nagłówek) i przerwanie działania programu.
5. Po wysłaniu danych, klient oczekuje na odpowiedź serwera i wypisuje ją na standardowe wyjście.

Program serwera należy zaimplementować w oparciu o poniższe założenia:

1. Serwer może odebrać dane z dowolnego interfejsu sieciowego (`INADDR_ANY`).
2. Serwer w pętli oczekuje na dane od klienta (model iteracyjny).

 Copyright © 2009 Politechnika Krakowska

3. Po odebraniu datagramu UDP za pomocą funkcji `recvfrom()`, serwer wypisuje adres i numer portu klienta (można posłużyć się funkcją `inet_ntop()`).
4. W dalszej kolejności serwer sprawdza, czy otrzymany ciąg znaków można traktować jako liczbę całkowitą. Jeżeli tak, to sprawdza czy liczba jest palindromem i przesyła odpowiedź do klienta za pomocą funkcji `sendto()`.
5. Podczas sprawdzania czy otrzymane dane są palindromem liczbowym, należy wykorzystać funkcję `is_palindrome()` z pliku `libpalindrome.c`. Funkcja ignoruje białe znaki oraz wiodące zera. Przykładowy ciąg znaków: 000000 0123 5556 797 6555 321 jest traktowany jako poprawny palindrom.
6. Otrzymanie datagramu UDP bez danych kończy działanie serwera.

Do implementacji można posłużyć się kodem źródłowym programów z zadania 2.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client3 client3.c
   $ gcc –o server3 server3.c libpalindrome.c
   ```
2. Uruchomić serwer UDP podając wybrany przez siebie numer portu:
   ```
   $ ./server3 <numer portu>
   ```
3. Uruchomić klienta i przeanalizować działanie programów:
   ```
   $ ./client3 <adres IP> <numer portu>
   ```

2.4. Zadanie 4. Serwer chat TCP/IP

Celem zadania jest zaimplementowanie prostego serwera TCP, do którego można połączyć się za pomocą aplikacji **telnet**. Trudność polega na odpowiednim zorganizowaniu operacji wejścia/wyjścia w taki sposób, aby serwer umożliwiał obsługę wielu klientów. Dane odebrane od jednego klienta (telnet) mają być wysyłane do pozostałych.

Program serwera należy zaimplementować w oparciu o poniższe założenia:

1. Serwer może odebrać dane z dowolnego interfejsu sieciowego (`INADDR_ANY`).
2. W celu organizacji operacji wejścia/wyjścia należy posłużyć się funkcją `select()`.
3. Po odebraniu nowego połączenia TCP za pomocą funkcji `accept()`, serwer wypisuje adres i numer portu klienta (można posłużyć się funkcją `inet_ntop()`). Deskryptor utworzony przez funkcję `accept()` należy dodać do zbioru deskryptorów monitorowanych przez funkcję `select()`.
4. Zamknięcie połączenia przez aplikację telnet, ma powodować usunięcie odpowiedniego deskryptora ze zbioru deskryptorów monitorowanych.
5. Dane tekstowe otrzymane od jednego klienta mają być przesyłane do wszystkich pozostałych.

 Copyright © 2009 Politechnika Krakowska

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o server4 server4.c`
2. Uruchomić serwer TCP podając wybrany przez siebie numer portu:
   `$ ./server4 <numer portu>`
3. Uruchomić kilka instancji aplikacji telnet i przetestować komunikację:
   `$ telnet <adres IP serwera> <numer portu>`

2.5. Zadanie 5. Implementacja wielowątkowego serwera HTTP.

Zadanie polega na zaimplementowaniu prostego, wielowątkowego serwera HTTP. Zadaniem głównego wątku serwera jest akceptowanie nowych połączeń TCP (w tym celu wątek główny oczekuje w nieskończonej pętli na nowe połączenia). W wypadku nawiązania połączenia przez klienta, wątek główny ma utworzyć wątek roboczy odpowiedzialny za dalszą obsługę połączenia. Taka implementacja serwera jest często spotykana, gdy obsługa pojedynczego klienta jest bardzo długa i mogłaby uniemożliwiać akceptację nowych połączeń. Aby osiągnąć opisany powyżej efekt należy użyć funkcji `pthread_create()`, która po nawiązaniu połączenia z klientem utworzy wątek roboczy i przekaże mu zadanie obsługi klienta.

Wątki robocze mają być odpowiedzialne za obsługę zapytań HTTP. Należy w ten sposób zaimplementować funkcję wątku roboczego, aby po połączeniu przeglądarki internetowej z serwerem, w przeglądarce wyświetliła się strona HTML prezentująca wszystkie zdjęcia z katalogu *img*.

Wątek roboczy należy zaimplementować w oparciu o poniższe założenia:

1. Za pomocą funkcji `recv()` wątek powinien odbierać dane od klienta i sprawdzać czy dane te są poprawnym zapytaniem HTTP (metoda GET).
2. Jeżeli zapytanie metodą GET dotyczy pliku graficznego (.jpg, .jpeg, .png lub .gif) z katalogu *img*, to zadaniem wątku jest przesłanie pliku graficznego jako odpowiedzi HTTP. Szczególną uwagę należy zwrócić na określenie nagłówków HTTP: *Content-Length* oraz *Content-Type*.
3. W odpowiedzi na każde inne zapytanie HTTP, wątek powinien przesyłać odpowiedź HTTP zawierająca stronę HTML o treści:

```
<html><body><center>
<img src=”img/nazwa_pliku_graficznego1.jpg”></img><br />
<img src=”img/nazwa_pliku_graficznego2.png”></img><br />
.
.
.
</center></body></html>
```

 Copyright © 2009 Politechnika Krakowska

Znaczniki <img> mają być określone na podstawie zawartości katalogu *img* (wszystkie pliki graficzne .jpg, .jpeg, .png lub .gif). Strona HTML opisana w tym punkcie może być generowana raz, przy starcie serwera (na podstawie zawartości katalogu *img*) i przechowywana w pamięci serwera.

Uwagi:

1. Serwer może odebrać dane z dowolnego interfejsu sieciowego (`IN-ADDR_ANY`).
2. Nazwy plików w katalogu *img* powinny być zgodne z ASCII.
3. W celu określenia zawartości katalogu *img* można wykorzystać funkcje `opendir()` oraz `readdir()`.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
2. `$ gcc –o server5 server5.c -pthread`
3. Uruchomić serwer HTTP podając wybrany przez siebie numer portu:
   `$ ./server5 <numer portu>`
4. Za pomocą przeglądarki internetowej połączyć się z serwerem:
   `http://<adres IP>:<numer portu>`

 Copyright © 2009 Politechnika Krakowska

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Gniazda TCP i UDP"* polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego. Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

 Copyright © 2009 Politechnika Krakowska