programowanie usług sieciowych

„Rozgłaszanie i rozsyłanie grupowe”

laboratorium: 09

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **rozgłaszania i rozsyłania grupowego**. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer realizujących rozgłaszanie grupowe i rozsyłanie. Oba te zagadnienia realizowane są za pomocą protokołu warstwy transportowej UDP (ang. User Datagram Protocol). W laboratorium należy zwrócić szczególną uwagę na poniższe zagadnienia :

- dostępność rozsyłania grupowego jest nieobowiązkowa dla protokołu IPv4, ale obowiązkowa dla protokołu IPv6,
- w protokole IPv6 nie uwzględniono rozgłaszania, a zatem każdą usługę IPv4 używającą rozgłaszania trzeba zmodyfikować zastępując rozgłaszanie rozsyłaniem grupowym.

1.2. Zagadnienia do przygotowania

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

- Gniazda UDP [RFC 768],
- Adresami rozgłaszania [Stevens],
- Różnice pomiędzy broadcastem a unicastem [Stevens].

A także z zagadnieniami dotyczącymi **rozsyłania grupowego**:

- Standardem multitaskingu [RFC 1112],
- IPv6 multicast [RFC 2464],
- Porównanie unicastu i milticastu [Stevens].

Literatura:

[1] W.R. Stevens, *„Programowanie Usług Sieciowych”*, *„API: gniazda i XTI”*.
[2] IETF (http://www.ietf.org/), RFC 1112, RFC 768, RFC 2464

1.3. Cel laboratorium

Celem laboratorium jest poznanie różnic pomiędzy rozgłaszaniem, a rozsyłaniem grupowym, określenie ich wad i zalet oraz zastosowania. Podczas realizacji tego laboratorium zapoznasz się z:

- programowaniem warstwy transportowej przy użyciu protokołu UDP,
- dowiązaniami gniazd do numeru portu i adresu IP,
- konfiguracją interfejsu lokalnego, oraz adres rozgłaszania grupowego dla danej podsieci.

2. Przebieg laboratorium

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

2.1. Przygotowanie laboratorium

Przed przystąpieniem do laboratorium konieczne jest zaznajomienie się z programem ethereal. Aplikacja ta służy do przechwytywania danych wysyłanych i odbieranych przez kartę sieciową zainstalowaną w komputerze. Kolejny program który wykonuje zadanie analogiczne do poprzedniego to tcpdump, należy się także zapoznać z jego opcjami. Ostatni program z którego opcjami należy się zapoznać to netstat, służy on do obserwowania połączeń sieciowych a także stanu gniazd tworzonych przez nas. Katalog na programy celem przeprowadzenia laboratorium to:

```
~/pus/pus_09_unibroadmulti
```

2.2. Zadanie 1. Unicast

Zadaniem jest analiza przykładowego programu klienta i serwera, działających przy wykorzystaniu protokołu UDP. Przykład ten ilustruje prostą komunikację pomiędzy klientem, a serwerem.

Do tego ćwiczenia potrzebne będą źródła programu, które należy ściągnąć z katalogu:

```
$ cd  home/shared/pus/pus_09_unibroadmulti/zad1
```

Poniżej znajduje się kod serwera serwer_udp.c

```
1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
4. #include<sys/types.h>
5. #include<sys/socket.h>
6. #include<netinet/in.h>

7. int main(int argc, char **argv)
```



```
8. {
9.    struct sockaddr_in addr_s,addr_k,*tmp;
10.   int listenfd;
11.   char buf[51];
12.   int index,i,len;
13.   listenfd=socket(AF_INET,SOCK_DGRAM,0);
14.   bzero(&addr_s,sizeof(addr_s));
15.   addr_s.sin_family=AF_INET;
16.   addr_s.sin_port=htons(9000);
17.   addr_s.sin_addr.s_addr=htonl(INADDR_ANY);
18.   bind(listenfd,(struct sockaddr*)&addr_s,sizeof(addr_s));
19.   len=sizeof(addr_k);
20.   for(;;)
21.   {
22.         recvfrom(listenfd,buf,50,0,(struct                sock-
            addr*)&addr_k,&len);
23.         tmp=&addr_k;
24.         static char str[128];
25.         inet_ntop(AF_INET,&tmp->sin_addr,str,
            sizeof(str));
26.         printf("ip%sport%d\n",str,ntohs(tmp->sin_port));
27.         buf[index]=0;
28.         fputs(buf,stdout);
29.         sendto(listenfd,buf,strlen(buf),0,(struct
            sockaddr*)&addr_k,sizeof(addr_k));
30          for(i=0;i<51;i++)
31.         {
32.               buf[i]=0;
33.         }
34.   }
35.   return 0;
36. }
```

Teraz omówimy kolejno działanie tego programu. Linie 1-6 dołączenie potrzebnych plików nagłówkowych , linia 13 to utworzenie gniazda o odpowiednich parametrach, które są omówione powyżej przy opisie funkcji. Funkcja *socket()* zwraca deskryptor, który przechowywany jest w zmiennej int listenfd. Następnie punkty 14-17 najpierw wyzerowanie struktury zawierającej adres serwera a następnie przypisanie odpowiednich wartości polom struktury (**struct sockaddr**) ustawiamy tu miedzy innymi to iż , korzystamy z IPv4 (opcja **AF_INET**) oraz port na którym serwer będzie nasłuchiwał w tym przypadku 9000 . Używamy tu także funkcji *htons* oraz *htonl* które odpowiadają za konwersje liczb z reprezentacji danego systemu do reprezentacji sieciowej. Kolejna linia 18 to wywołanie funkcji *bind* która łączy gniazdo z adresem IP maszyny. Dalsza część to nieskończona pętla for w której wywołana jest funkcja *recvfor()* która odbiera dane wysyłane do serwera . Parametry tej funkcji opisane są w

powyżej. Dalej linie 23-26 służą konwersacji i wypisaniu adresu IP oraz portu z którego serwer otrzymał dane. Dalsza część programu to wypisanie odebranych danych oraz wysłanie ich ponownie do klienta , wykorzystany jest tu adres ,który mamy z funkcji recvfrom().

Kod klienta klient_udp.c

```
1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
4. #include<sys/types.h>
5. #include<sys/socket.h>
6. #include<netinet/in.h>

7. int main(int argc ,char **argv)
8. {
9.          struct sockaddr_in addr;
10.         int listenfd;
11.         int index;
12.         char buf[51],buf1[51];
13.         int len;
14.         listenfd=socket(AF_INET,SOCK_DGRAM,0);
15.         bzero(&addr,sizeof(addr));
16.         addr.sin_family=AF_INET;
17.         addr.sin_port=htons(9000);
18.         inet_pton(AF_INET,argv[1],&addr.sin_addr);
19.         len=sizeof(addr);
20.         fgets(buf,50,stdin);
21.         sendto(listenfd,buf,strlen(buf),0,(struct          sock-
            addr*)&addr,len);
22.         recvfrom(listenfd,buf1,50,0,NULL,NULL);
23.         buf1[index]=0;
24.         fputs(buf1,stdout);
25.         fgets(buf,50,stdin);
26.         return 0;
27.  }
```

Powyżej mamy kod programu będącego klientem. Omówimy go pod kątem występujących nowości w porównaniu z serwerem. W linii 18 mamy dodatkowo przypisany do struktury adres który jest podawany jako pierwszy argument programu. Dla gniazda którego będzie używał klient nie ustawiamy portu robi to jadro systemu przydzielając jeden z wolnych portów (z portów wysokich). Dalsza część programu to funkcja wysyłająca oraz pobierająca dane od serwera. Program ten wysyła najpierw dane podane na konsolę do serwera a następnie czeka na odpowiedź od serwera.

Kompilacja i uruchomienie
1. W celu uruchomienia powyższych programów dokonujemy najpierw ich kompilacji:

```
$ gcc serwer_udp.c –o serwer
$ gcc klient_udp.c –o klient
```

2. Teraz możemy już uruchomić powyższe programy:
Np. na stacji itilab01 uruchamiamy serwer poleceniem

```
$ ./serwer
```

Oraz np. na itilab02 uruchamiamy klienta (jako parametr podajemy IP serwera w tym wypadku 192.168.202.1)

```
$ ./klient 192.168.202.1
```

3. Poniżej umieszczam przykład działania klienta:

```
$  ./klient 192.168.202.1
programowanie
programowanie
usług
usług
sieciowych
sieciowych
```

oraz dane wypisywane na konsolę serwera:

```
      $ ./serwer
ip192.168.202.2port1036
programowanie
ip192.168.202.2port1036
usług
ip192.168.202.2port1036
sieciowych
```

Na powyższych wydrukach widać że klient odebrane dane od serwera (każde słowo występuje podwójnie) Natomiast na konsoli serwera mamy dodatkowo wiadomość o IP klienta i porcie.

2.3. Zadanie 2. Broadcast

Kolejny program to podobnie jak poprzednio komunikacja serwer klient ale wykorzystująca rozsyłanie typu broadcast. Implementacja serwera jest identyczna jak poprzednio natomiast zmieniony jest kod klienta, który może teraz pracować w trybie rozgłaszania.

Kod klienta broadcast klientb_udp.c

```
1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
4. #include<sys/types.h>
5. #include<sys/socket.h>
6. #include<netinet/in.h>

7. int main(int argc ,char **argv)
8. {
9.          struct sockaddr_in addr_s,addr_k,tmp,*tmp1;
10.         int listenfd;
11.         int index,i;
12.         int len,len1;
13.         listenfd=socket(AF_INET,SOCK_DGRAM,0);
14.         bzero(&addr_s,sizeof(addr_s));
15.         addr_s.sin_family=AF_INET;
16.         addr_s.sin_port=htons(9000);
17.         inet_pton(AF_INET,argv[1],&addr_s.sin_addr);
18.         char buf[51];
19.         char buf1[51];
20.         bind(listenfd,(struct                                   sock-
            addr*)&addr_k,sizeof(addr_k));
21.         const int value_of_opt=1;
22.         setsockopt(listenfd,SOL_SOCKET,SO_BROADCAST
            ,&value_of_opt,sizeof(value_of_opt));
23.         len=sizeof(addr_s);
24.         fgets(buf,50,stdin);
25.         static char str[128];
26.         sendto(listenfd,buf,strlen(buf),0,(struct          sock-
            addr*)&addr_s,len);
27.         len1=sizeof(tmp);
28.         for(;;)
29.         {
30.               recvfrom(listenfd,buf1,50,0,
                  (struct sockaddr*)&tmp,&len1);
31.               buf1[index]=0;
32.               tmp1=(struct sockaddr_in*)&tmp;
```



```
33.               inet_ntop(AF_INET,&tmp1
                  ->sin_addr,str,sizeof(str));
34.               printf("odebrano z ip%s,port %d",str,ntohs(tmp1-
                  >sin_port));
35.               fputs(buf1,stdout);
36.               for(i=0;i<51;i++)
37.               {
38.                     buf[i]=0;
39.                     buf1[i]=0;
40.               }
41.               fgets(buf,50,stdin);
42.         return 0;
43. }
```

Program ten opiszemy pod kątem różnic w stosunku do klienta który działał na zasadzie unicastu. Pierwsza różnica i w zasadzie jedyna to linijka 22 w której zmieniamy opcje gniazda tak aby mogło działać w trybie rozgłoszeniowym. Kolejna to, to, że klient także wypisuje na konsole IP i port serwera od którego odebrał dane. Różnica ta wynika tylko z tego ze klient odbiera dane od kilku serwerów i dlatego dodaliśmy tą funkcjonalność.

Program ten należy skompilować analogicznie tak jak w powyższym przykładzie, jedyna różnica to, to że podawany adres przy uruchomieniu klienta jest adresem broadcast dla sieci 192.168.202.0 maska 255.255.255.255 zatem adres rozgłoszeniowy to 192.168.202.255.

W celu zaprezentowania właściwości broadcast uruchamiamy kilka serwerów np. na itilab01-itilab03 oraz klienta np. na itilab10. Po uruchomieniu klienta oczekuje na podanie danych które są rozesłane po sieci natomiast serwery które nasłuchują po odebraniu danych wysyłają je do klienta. Poniżej umieszczono wydruk z konsoli klienta.

```
$ ./klientb_udp 192.168.202.255
czesc
odebrano z ip192.168.202.2,port 9000czesc
odebrano z ip192.168.202.1,port 9000czesc
odebrano z ip192.168.202.6,port 9000czesc
```

Widzimy że klient otrzymał dane od wszystkich trzech serwerów .Po adresach IP możemy wywnioskować, że serwery były uruchomione na itilab01, itilab02 oraz itilab06.

2.4. Zadanie 3. Multicast

Kolejny program, tak jak w poprzednim przypadku to komunikacja typu serwer-klient, ale wykorzystująca rozsyłanie typu multicast. Implementacja serwera i klienta znajduje się w plikach server_multicast.c i klient_multicast.c. Działanie serwera pozostaje prawie bez zmian. Natomiast kod programu klienta znajduje się poniżej, gdyż znajduje się tu kilka nowych funkcji jak np. dołączanie do grupy multicastowej.

```
1. #include <time.h>
2. #include <string.h>
3. #include <stdio.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <netinet/in.h>
7. #include <arpa/inet.h>
8.
9. #define PORT 12345
10.#define GROUP "225.0.0.37"
11.#define MSGBUFSIZE 256
12.
13.main(int argc, char *argv[])
14.{
15.     struct sockaddr_in addr;
16.     int fd, nbytes,addrlen;
17.     struct ip_mreq mreq;
18.     char msgbuf[MSGBUFSIZE];
19.     u_int yes=1;
20.     if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0)
21.     {
22.         perror("socket");
23.         exit(1);
24.     }
25.     setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes));
26.     memset(&addr,0,sizeof(addr));
27.     addr.sin_family=AF_INET;
28.     addr.sin_addr.s_addr=htonl(INADDR_ANY);
29.     addr.sin_port=htons(PORT);
30.     bind(fd,(struct sockaddr *) &addr,sizeof(addr));
31.     mreq.imr_multiaddr.s_addr=inet_addr(GROUP);
32.     mreq.imr_interface.s_addr=htonl(INADDR_ANY);
33.     setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,
        sizeof (mreq));
34.     while (1)
35.     {
36.         addrlen=sizeof(addr);
37.         nbytes=recvfrom(fd,msgbuf,MSGBUFSIZE,0,
            (struct sockaddr *) &addr,&addrlen);
```



```
38.          puts(msgbuf);
39.      }
40.}
```

Kod jest bardzo podobny do poprzednich programów, dlatego zostaną omówione jedynie nowe wywołania funkcji. W punkcie 25 ustawiania jest opcja umożliwiająca wielu gniazdom używanie tego samego numeru portu. W liniach 31-33 używana jest funkcja `setsockopt() która powoduje` że jądro dołączyła do grupy multicast określonej w strukturze *mreq*. Dalej mamy już tylko nieskończoną pętle w której znajduje się funkcja która odbiera pakiety.

Program ten należy skompilować i uruchomić zgodnie z wskazówkami z wcześniejszych zadań.

2.5. Zadanie 4 Zadanie do samodzielnej realizacji.

W tym ćwiczeniu należy napisać samodzielnie program działający podobnie jak wszystkie poprzednie w trybie (serwer – klient). Serwer ma rozgłaszać w sieci czas pobrany z systemu, ponadto w czasie wywoływania jako parametr powinna być opcja podania co jaki okres ma być rozsyłany nowy datagram z czasem. Klient natomiast ma odbierać wysyłane dane i wypisywać na konsolę, ponadto program będący klientem powinien razem z czasem wypisać na konsolę adres IP oraz port z jakiego odebrał pakiet. Wszystkie potrzebne funkcje w zasadzie były używane w powyższych przykładach. Jedynie do pobierania czasu z systemu służy funkcja `time_t time(time_t *t)` oraz funkcja `char* ctime(time_t *t)` która konwertuje podany czas do wartości typu string. W celu używania tych funkcji należy dołączyć plik nagłówkowy time.h .

2.6. Zadanie 5. Zadanie do samodzielnej realizacji.

Zadaniem dodatkowym do ćwiczenia będzie takie zmodyfikowanie kodu serwera , aby oprócz wysyłania wiadomości multicastowych umożliwiał równoczesne ich odbieranie. Kod programów do zmodyfikowania znajduje się w katalogu:

`~/pus/pus_09_unibroadmulti/zad5`

3. Opracowanie i sprawozdanie

Realizacja laboratorium polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.
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,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne podczas zajęć, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

