PROTOCOALE DE COMUNICAȚIE : LABORATOR 6

Socketi UDP

Responsabil: Catalin LEORDEANU

Cuprins

Obiective

In urma parcurgerii acestui laborator studentul va fi capabil sa:

- explica in ce consta protocolul de transport UDP
- scrie un program care foloseste socketi UDP

Notiuni teoretice

Nivelul transport

Nivelul transport oferă multiplexare la nivel de aplicații. Astfel 2 aplicații aflate pe două mașini diferite pot comunica prin intermediul acestui layer.

Porturi

Multiplexarea la nivelul transport se asigură prin porturi. Acestea sunt reprezentate pe 2 octeți, așadar numerele care sunt asignate porturilor sunt de la 0 la 65535, însă cele până la 1024 sunt rezervate pentru aplicaţii standard, precum:

- FTP: 20,21
- SSH: 22
- Telnet: 23
- SMTP: 25
- HTTP: 80
- POP3: 110
- IMAP: 143

O listă completa a porturilor rezervate găsiti la IANA: http://www.iana.org/assignments/port-numbers

UPD

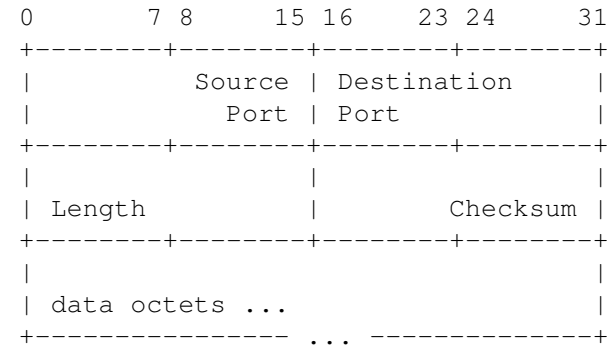
UDP (User Datagram Protocol) este un protocol ce trimite pachete independente de date, numite datagrame, de la un calculator către altul fără a garanta în vreun fel ajungerea acestora la destinaţie. Acest protocol nu stabileşte o conexiune permanentă între cele două calculatoare. Este descris in RFC 768 (http://tools.ietf.org/html/rfc768)

Protocolul UDP are următoarele proprietăţi:

- nu se stabileşte o conexiune între client-server. aşadar serverul nu va aştepta apeluri de conexiune, ci aşteaptă direct datagrame de la clienţi. Acest tip de comunicare este întâlnit in sistemele client-server în care se transmit puţine mesaje şi în general prea rar pentru a menţine activă o conexiune între cele două entităţi. (Un exemplu în acest caz este DNS-ul)
- nu se garantează ordinea primirii mesajelor şi nici prevenirea pierderilor pachetelor. UDP-ul se utilizează mai ales în reţelele în care există o pierdere foarte mică de pachete şi în cadrul aplicaţiile pentru care pierderea unui pachet nu este foarte gravă (Un exemplu: aplicaţiile peer-to-peer din cadrul unei reţele locale).
- are un overhead foarte mic, în comparaţie cu celelalte protocoale de transport (Are un header de 8 Bytes, în comparaţie cu TCP-ul care are minim 20Bytes).

Header UDP

Orice pachet UDP are următorul header:

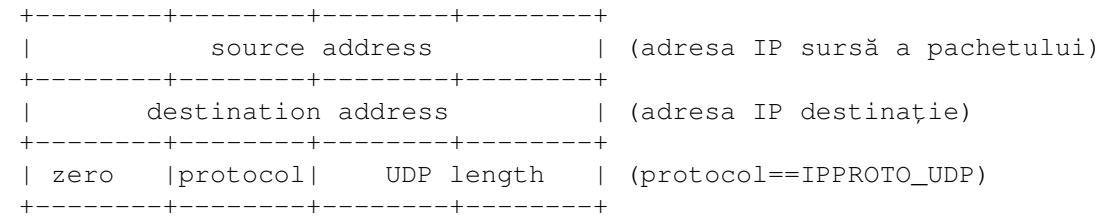


Explicaţii header:

- Portul sursă este ales random de către maşina sursă a pachetului dintre porturile libere existente pe acea maşină.
- Portul destinaţie este portul pe care maşina destinaţie poate recepţiona pachete.
- Lungime este lungimea in octeţi a datagramei (header+date).
- Checksum este valoarea sumei de verificare pentru datagramă. În sectiunea următoare , este prezentat modul de calcul a checksum-ului pentru un pachet UDP.

Calcul Checksum

Pentru a calcula acest număr header-ului UDP i se adaugă un *pseudo-header* format din :



Pe acest pseudo-header, împreună cu datagram UDP (header+date) se calculează suma în următorul mod:

- Se formează cuvinte de 16 biți. Dacă numărul de octeți al datelor este impar, atunci se mai adaugă la sfârșit un octet 0.
- Se calculează suma complementelor față de 1 al tuturor cuvintelor.
- Se calculează complementul față de 1 al sumei de la pasul anterior.

Checksum-ul datagramei este numărul returnat de ultimul pas.

API

Network byte-order

Un sistem de operare poate ordona octeţii într-un cuvânt în următoarele variantele:

- Big Endian - cel mai semnificativ octet primul
- Little Endian - cel mai puţin semnificativ octet primul

Datorită acestei împărțiri există o problemă dacă vrem să interconectăm două sisteme de operare diferite din punct de vedere al reprezentării datelor. În acest sens, Internet-ul a impus o secvenţă standard pentru sto-

carea datelor numerice, numită **network byte-order**. Spre deosebire, secvenţa octeţilor pentru calculatorul gazdă se numeşte **host byte-order**.

Indiferent de ordonarea octeţilor pe o maşină, în momentul în care se trimit date în reţea acestea trebuie sa fie în formatul Network Byte Order.

Pentru conversie există funcţii pentru două tipuri: short (2 octeţi) şi long (4 octeţi). Aceste funcţii sunt valabile şi pentru variantele unsigned.

```
#include <arpa/inet.h>

 uint32_t htonl(uint32_t hostlong);
 uint16_t htons(uint16_t hostshort);
 uint32_t ntohl(uint32_t netlong);
 uint16_t ntohs(uint16_t netshort);
```

Socket

Un "socket" reprezintă un canal generalizat de comunicare între procese, reprezentat în Unix printr-un descriptor de fişier. El oferă posibilitatea de comunicare între procese aflate pe maşini diferite într-o retea.

Structuri de date

struct sockaddr

În structura struct sockaddr definită în *¡sys/socket.h¿* se ţin informaţiile referitoare la adresa socketţilor:

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

struct sockaddr
{
  unsigned short sa_family;
  char sa_data[14];
}
```

Explicatii:

- *sa_family* - indica un format particular de adresa. Valori uzuale: *AF_INET* (protocol IPv4)
- *sa_data* - adresa utilizata

struct sockaddr_in

În cazul comunicaţiei prin Internet, toate adresele socketților sunt compuse dintr-un număr de port şi adresa IP a calculatorului respectiv. Pentru specificarea acestor adrese se utilizează în locul structurii *sockaddr*, structura*sockaddr_in* ("in" de la Internet):

```
#include <netinet/net.h>
struct sockaddr_in
{
  unsigned short sin_family;
  unsigned short int sin_port;
  struct in_addr sin_addr;
```

```
}
```

Explicatii:

- *sin_family* - aceeași semnificatie ca *sa_family*. Valoare constantă *AF_INET*
- *sin_port* - portul utilizat (Network Byte Order)
- *sin_addr* - adresa IP (Network Byte Order)

struct in_addr

Reține o adresă IPv4, care are 32 biți

```
struct in_addr
{
  uint32_t s_addr;
}
```

Pentru această structură există câteva constante speciale:

- *INADDR_LOOPBACK* - Reprezintă adresa maşinii, '127.0.0.1', care mai este numită şi 'localhost'. Această constantă poate fi folosită în loc de a obţine adresa propriului calculator.
- *INADDR_ANY* - Reprezintă orice adresă şi este folosită ca valoare pentru sin_addr. atunci când se doreşte să fie acceptate conexiuni.
- *INADDR_BROADCAST* - Adresa folosită pentru broadcast.
- *INADDR_NONE* - Adresa returnată de unele funcţii pentru a indica eroare.

Utilizarea adreselor IPv4

Funcţiile pentru manipularea adreselor Internet sunt definite în headerul *¡arpa/inet.h¿* și sunt explicate în secțiunea 3 a man-ului. (*man 3 inet_addr*)

inet_addr()

Convertește o adresă IPv4 din formatul standard (exemplu 10.2.5.160) în forma binară.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
[...]
struct sockaddr_in my_addr;
[...]
my_addr.sin_addr.s_addr = inet_addr("10.2.5.160");
[...]
```

Problema care apare la folosirea *inet_addr()* este ca în cazul în care apare o eroare se întoarce INADDR_NONE care este -1 şi reprezintă adresa "255.255.255.255", care este o adresă validă.

inet_aton()

De aceea, este recomandat folosirea funcţiei *inet_aton()* (ASCII to network):

```
int inet_aton (const char *NAME, struct in_addr *ADDR);
```

Rezultatul intors este diferit de zero daca adresa este valida si zero in caz de eroare

```
inet_aton("10.2.5.160", &(my_addr.sin_addr));
```

inet_ntoa()

Dacă se doreşte ca dintr-o structură in_addr să se afişeze adresa IPv4 în formatul standard se foloseşte funcţia inversă inet_ntoa() (network to ASCII):

```
char * inet_ntoa (struct in_addr ADDR);
```

Atenție! Rezultatul este întors într-un pointer alocat static, astfel că apeluri repetate ale acestei funcţii vor scrie în acelaşi buffer, de aceea este bine ca rezultatul să fie copiat dacă se doreşte a fi păstrat.

Functii socket-i UDP

În cele ce urmează se vor prezenta principalele funcţii pentru manipularea socketţilor.

Schema pentru o comunicare tipică client-server este:

```
Server: socket(...) -> bind(...) -> recvfrom(...) -> recvfrom(...) / sendto(...) ...
Client: socket(...) -> [bind(...)] ---> sendto(...) -> recvfrom(...) / sendto(...) ...
```

Atât serverul cât și clientul apelează *socket()*. Serverul este cel care așteaptă date, prin *recvfrom()*. Clientul trimite o cerere la server prin *sendto()*. Ulterior, schimburile între cele doua procese sunt în funcție de natura aplicației.

Clientul poate face opțional *bind*, dacă vrea să specifice o interfață pe care pleacă pachetele sau un port sursă, dar de cele mai multe ori acest aspect cade în sarcina sistemului de operare.

socket()

Pentru obţinerea descriptorului de fişier se foloseşte functia:

```
#include <sys/types.h>
#include <sys/socket.h>
[...]
int socket(int domain, int type, int protocol);
[...]
```

Explicatii:

- *domain* - reprezintă familia protocoalelor pe care urmează să le utilizăm în transferul informaţiei. Valori uzuale :
  - PF_INET (IPv4)
  - PF_INET6 (IPv6)
- *type* - tipul socketului. Valori uzuale:
  - SOCK_STREAM - Indică stabilirea unei comunicaţii bazată pe construirea unei conexiuni între sursă şi destinaţie. Comunicaţia este FIFO, fiabilă şi sigură.
  - SOCK_DGRAM - Oferă un flux de date bidirecţional dar care nu promite să fie sigur, în secvenţă sau neduplicat. Adică, un proces care recepţionează mesaje pe un socket datagramă, poate găsi mesaje duplicate şi posibil într-o ordine diferită faţă de cea în care au fost trimise.
  - SOCK_RAW - Permite accesul utilizatorului la protocoalele de comunicaţie inferioare care suportă abstractizarea socketurilor. Aceste socketuri sunt orientate datagramă, deşi caracteristicile lor sunt dependente de interfaţa oferită de protocol. Socketurile RAW nu sunt pentru utilizatorul general. Ele sunt oferite în mod deosebit celor interesaţi în dezvoltarea de noi protocoale de comunicaţie sau celor care doresc acces la nişte facilităţi mai rar folosite ale unui protocol existent.
- *protocol* - specifică protocolul de transport utilizat. Fiecare tip de socketi are un protocol specific și de aceea valoarea câmpului este bine să fie 0 pentru ca să se aleagă protocolul corect în funcţie de argumentul type specificat.
- link: *man 2 socket* http://www.kernel.org/doc/man-pages/online/pages/man2/socket.2.html

În cazul UDP se utilizează tipul SOCK_DGRAM.

Funcţia întoarce descriptorul socket-ului sau -1 în caz de eroare setând corespunzător variablia *errno*.

bind()

Odata ce am obţinut un socket, trebuie să îi asociem un port pe maşina locală (acest lucru este uzual în cazul în care se doreşte aşteptarea de conexiuni pe un anumit port).

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

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

Explicații:

- sockfd - descriptorul de fişier returnat de funcţia socket()
- my_addr - conține informaţii despre adresa IP şi port
- addrlen - lungimea celui de-al doilea parametru poate fi setat ca fiind *sizeof(struct sockaddr)*
- link - man 2 bind: http://www.kernel.org/doc/man-pages/online/pages/man2/bind.2.html

In urma apelului, socketului sockfd i se asignează adresa din my_addr. În caz de succes returnează zero, iar în caz de eroare -1 şi setează corespunzator errno.

Dacă în structura my_addr, portul este 0 (my_addr.sin_port = 0) atunci se alege orice port disponibil din cele nerezervate.

Observaţie: Câteodată, când se încearcă să se repornească un server, bind() nu reuşeşte, iar eroarea este "Address already in use". Asta înseamnă că un socket care a fost conectat pe acel port încă mai este agăţat şi utilizează portul.

În această situaţie fie se poate aştepta să se deconecteze câteva minute sau se poate specifica în program să se reutilizeze portul:

```
[...]
int yes = 1;
if (setsockopt(listener,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
```

```
perror("setsockopt");
exit(1);
}
[...]
```

close() / shutdown()

Pentru a închide un socket se folosește funcția de închidere a unui descriptor de fișier din Unix:

```
#include <unistd.h>

int close(int fd);
```

Acest lucru va împiedica alte citiri și scrieri din socket. Orice încercare de citire/scriere din acest socket va intoarce eroare. În cazul în care se dorește mai mult control asupra socketului care urmează să fie închis, se poate folosi funcția *shutdown()* care permite întreruperea comunicației într-un sens sau în ambele direcții (caz asemănător cu close()).

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

int shutdown(int sockfd, int how);
```

Explicații:

- sockfd - descriptorul de fişier returnat de funcţia socket()
- how - specifică modul de închidere. Valori luate:
  - 0 - Utilizatorul nu mai este interesat să citească datele.
  - 1 - Nu se mai pot face transmiteri de date.
  - 2 - ambele (caz asemănător cu close)
- link: man 2 shutdown

Observație: shutdown() nu inchide de fapt un descriptor de fișier, ci doar îi schimbă modul de utilizare. Pentru a elibera un socket trebuie folosit close().

recvfrom()/sendto()

Prin intermediul acestor apeluri specifice socketților UDP, se pot trimite/recepționa date.

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockfd, char* buff, int nbytes, int flags, struct sockaddr *to, int
    addrlen);
int recvfrom(int sockfd, char* buff, int nbytes, int flags, struct sockaddr *from, int
     addrlen);
```

Explicații:

- sockfd: socketul prin care se realizează comunicarea.
- buff: bufferul ce conține datele care se trimit/recepționează.

- flags: specifică condiții de efectuare a transmisiei/recepției.
- to/from: Structură ce indică adresa unde se trimit/de unde se primesc datele. (În cazul recvfrom() se populează de funcție).
- addrlen: Lungimea structurii to/from (in octeți).
- link: man 2 recvfrom - http://www.kernel.org/doc/man-pages/online/pages/man2/recvfrom.2.html

Daca nu apar erori, apelurile intorc lungimea mesajului in octeti. La eroare, se intoarce -1.

Apeluri extra

sendmsg()/recvmsg()

Apelurile de sistem folosite în comunicația fără conexiune sunt sendmsg() și recvmsg():

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

int sendmsg(int sockfd, struct msghdr* msg, int flags);
int recvmsg(int sockfd, struct msghdr* msg, int flags);
```

Explicații:

- sockfd - identificatorul socketului
- msg - structura care se va trimite/recepționa

```
#include <sys/types.h>
struct msghdr {
  void *msg_name; /* optional address */
  socklen_t msg_namelen; /* size of address */
  struct iovec *msg_iov; /* scatter/gather array */
  size_t msg_iovlen; /* # elements in msg_iov */
  void *msg_control; /* ancillary data, see below */
  size_t msg_controllen; /* ancillary data buffer len */
  int msg_flags; /* flags on received message */
};
```

- flags - aceeași semnificație ca la sendto()/recvfrom()
- link - man 2 sendmsg

gethostbyname()/ gethostbyaddr()

Funcțiile prezentate până acum se bazează pe faptul că știm adresa IP a entității la care ne conectăm. De cele mai multe ori, nu cunoaștem această adresă, ci doar numele sub care ea este recunoscută în rețea. (Este mult mai ușor de ținut minte un nume decât un număr, de aceea s-a și inventat și protocolul DNS).

Funcțiile prezentate în această secțiune rezolvă acest neajuns, returnând adresele și numele sub care este recunoscută o mașină în rețea.

```
#include <netdb.h>
#include <sys/socket.h> /* for AF_INET */

extern int h_errno;
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const void *addr, socklen_t len, int type);
```

Explicații:

- name - numele host-ului despre care se cer informații
- addr - un pointer la un buffer ce conține adresa, dată în formatul specific familiei de adrese din care face parte.
- len - lungimea bufferului anterior.
- type - familia de adrese (AF_INET pentru adrese de tip internet).
- link - man 3 gethostbyname: http://www.kernel.org/doc/man-pages/online/pages/man3/gethostbyname.3.html

Structura hostent este definită astfel:

```
#include <netdb.h>

struct hostent
{
  char *h_name;
  char **h_aliases;
  int h_addrtype;
  int h_length;
  char **h_addr_list;
}
```

Explicații:

- h_name: Reprezintă numele oficial al mașinii respective,
- h_aliases: Reprezintă alte nume sub care ea este cunoscută (alias-uri)
- h_addrtype: Reprezintă tipul adreselor acelei mașini (pentru adrese IP conține valoarea constantei AF_INET).
- h_length: Indică lungimea binară a tipului de adreăa utilizat (in cazul adreselor IP această lungime este de 4 octeți).
- h_addr_list: Reprezintă un vector de adrese sub care aceasta masină e recunoscută în rețea.(fiecare adresă are h_length octeți).

Exerciții

Pe baza scheletului de cod atasat laboratorului, veți implementa un mini server de backup fișiere și un client al acestui server.

Detalii implementare:

- Vă veți grupa în echipe de 2 persoane, una implementând serverul și cealaltă clientul; (la calculatoare diferite)
- Clientul va trimite un fișier binar prin UDP, iar serverul îl va recepționa.
- Nu trimiteți numele fișierului, ci primiți-l ca argument în cele 2 fișiere.
- Presupuneți că clientul va transmite următorul calup de date disponibil din fișier (setați dimensiunea calupului de date maxim la MTU-ul rețelei (în cazul practic 1500 octeți))
- Serverul va recepționa datele și le va scrie în fișierul lui.
- La final rulați md5sum pe cele 2 fișiere pentru a vedea dacă au fost recepționate corect.
- Pentru compilare & rulare & testare verificați fișierul Makefile.

Bonus: Implementați mecanismul de preluare a numelui fișierului la server și permiteți recepționarea mai multor fișiere consecutiv.