Socketi TCP. Multiplexarea I/O

Multiplexarea I/O

Serverul din cadrul laboratorului trecut putea sa lucreze cu un numar fixat de clienti. La inceput apela accept() pentru toti clientii (1 sau 2), apoi primea si trimitea date pe socketii activi (intr-o anumita ordine). Altfel, ar aparea o problema atunci cand serverul se afla blocat intr-un apel accept() si totusi doreste sa primeasca cu recv() date in acelasi timp(sau invers). Situatia se inrautateste daca dorim ca serverul sa functioneze cu un numar variabil de clienti, care sa se poata conecta/deconecta la/de la server oricand, chiar si dupa ce alti clienti au inceput sa trimita/primeasca date.

In cazul clientilor, ati vazut data trecuta ca aveau o ordine precisa a operatiilor: citire de la tastatura(si trimitere pe socket), apoi citire de pe socket (si afisare). Din acest motiv, cand primul client trimitea un mesaj, cel de-al doilea nu-l primea pana nu trimitea si el un mesaj la randul lui.

Am intalnit trei (tipuri de) apeluri blocante, care sunt de fapt citiri din descriptori (de socketi sau fisiere):

```
- accept()  (citire de pe socketul inactiv - pe care asculta serverul)
- recv()/recvfrom() (citire de pe socketi activi)
- scanf()/fgets()/read(0,..)  (citire de la tastatura = citire din descriptorul 0)
```

Am vazut ca pot aparea probleme atunci cand un program se afla blocat intr-o citire pe un descriptor, dar primeste date pe alt descriptor. Avem nevoie de un mecanism care sa ne permita sa citim exact de pe descriptorul pe care au venit date.

Aceasta problema este rezolvata de functia select(), care ajuta la controlarea mai multor descriptori(de fisiere sau socketi) in acelasi timp.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int  select(int  numfds,  fd_set  *readfds,  fd_set  *writefds,  fd_set  *exceptfds,
struct timeval *timeout);
```

Apelul select() primeste ca argumente pointeri spre trei multimi de descriptori (de citire, scriere sau exceptii). Daca utilizatorul nu este interesat de anumite conditii, argumentul corespunzator va fi setat la null (pe noi ne intereseaza doar multimea de citire). Atentie, select() modifica multimile de descriptori: la iesire, ele vor contine numai descriptorii pe care s-au primit date. Astfel, trebuie sa tinem copii ale multimilor originale.

Argumente:

```
 - numfds: cea mai mare valoare+1 a unui descriptor din cele 3 multimi
 - readfds: multimea de descriptori de citire
 - writefds: multimea cu descriptori pentru scriere
 - exceptfds: multimea cu descriptori pentru care sunt in asteptare exceptii
 - timeout: timpul in care apelul select() trebuie sa intoarca (daca e NULL, se
blocheaza pana cand vin date pe
            cel putin un descriptor)
```

Fiecare multime de descriptori este de fapt o structura ca contine un tablou de masti de biti; dimensiunea tabloului este data de constanta FD_SETSIZE (o valoare uzuala a acestei constante este

1024). Pentru lucrul cu multimile de descriptori preluate ca argumente de apelul select se pot folosi o serie de macrouri:

1) Sterge multimea de descriptori de fisiere *set*:

```
FD_ZERO(fd_set *set);
```

2) Adauga descriptorul *fd* in multimea *set*:

```
FD_SET(int fd, fd_set *set);
```

3) Sterge descriptorul *fd* din multimea *set*:

```
FD_CLR(int fd, fd_set *set);
```

4) Testeaza daca descriptorul *fd* apartine sau nu multimii *set*:

```
FD_ISSET(int fd, fd_set *set);
```

Un exemplu de server TCP ce foloseste apelul select() pentru multiplexare se afla in resurse. In multimea de citire a serverului se afla initial descriptorul pentru socketul inactiv. Apoi pe masura ce se conecteaza clientii, in multime vor fi adaugati si descriptorii pentru socketii activi (cei pe care se primesc/trimit date la clienti).

Exercitii

0. Modificati programul client din resurse, ca sa se comporte ca in laboratorul trecut (sa citeasca de la tastatura si sa trimita serverului, sa primeasca de la server si sa afiseze). Modificati si serverul, astfel incat sa functioneze cu 2 clienti: sa trimita clientului 1 ce a primit de la clientul 2 si invers.

1. Modificati programul client astfel incat sa multiplexeze intre citirea de la tastatura (vom adauga descriptorul 0 in multimea de citire pentru select) si citirea de pe socket. Din acest moment eliminam neajunsul ordonarii actiunilor clientilor.

2. Modificati programul server ca sa functioneze cu mai multi clienti. Clientii vor trimite in mesaj si destinatia mesajului (acest lucru se poate face si fara modificarea codului clientilor, vedeti exemplul). In cadrul acestui laborator putem folosi descriptorul socketului intors de accept() ca identificator pentru un client (in aplicatii reale clientii nu au acces la aceste valori). Exemplu,: clientul cu socketul 5 poate trimite (mesaj citit de la tastatura): “4 ce mai faci” iar serverul parseaza mesajul si-l trimite clientului conectat pe socketul 4. (Puteti sa lucrati si cu structura de mesaj.)

3. (Bonus) Modificati programul server ca sa trimita (la conectarea) clientilor lista cu clientii deja conectati, apoi sa trimita clientilor conectati update-uri despre ce client a mai intrat/a iesit din sistem. (puteti sa folositi acelasi sistem de identificatori pentru clienti, ca mai sus)