

Laboratorul 08 - TCP și multiplexare I/O

Responsabili: Valeriu Stanciu, Silviu Pantelimon, Radu-Ioan Ciobanu

Obiective

În urma parcurgerii acestui laborator, studenții vor fi capabili să utilizeze multiplexarea pentru crearea unor aplicații server ce pot răspunde cererilor primite de la un număr variabil de clienți.

Multiplexarea I/O


Serverul din cadrul laboratorului trecut putea să lucreze cu un număr fixat de clienți, respectând o ordine strictă a operațiilor. La început, apela *accept()* pentru toți clienții (1 sau 2), apoi primea și trimitea date pe socketii activi (într-o anumită ordine). Altfel, ar apărea o problemă atunci când serverul se află blocat într-un apel *accept()* și totuși dorește să primească date cu *recv()* în același timp (sau invers). Situația devine și mai complicată dacă dorim ca serverul să funcționeze cu un număr variabil de clienți, care să se poată conecta/deconecta la/de la server oricând, chiar și după ce alți clienți au început să trimită/primească date.

În cazul clienților, am văzut data trecută că aveau, de asemenea, o ordine precisă a operațiilor: citire de la tastatură, trimitere pe socket, citire de pe socket, afișare. Din acest motiv, când primul client trimitea un mesaj, cel de-al doilea nu îl primea până nu trimitea și el un mesaj la rândul lui.

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

- *accept()* - citire de pe socketul inactiv pe care ascultă serverul
- *recv()* / *recvfrom()* - citire de pe socketi activi
- *scanf()* / *fgets()* / *read(0, ...)* - citire de la tastatură.

După cum am putut vedea, toate situațiile prezentate sunt generate, de fapt, de o problemă comună: un program se află blocat într-o citire pe un descriptor, dar primește date pe un alt descriptor. Avem nevoie, deci, de un mecanism care să ne permită să citim exact de pe descriptorul pe care au venit date.

Solutia este reprezentată de funcția *select()*, care ajută la controlarea mai multor descriptori (de fișiere sau socketi) în același timp. Pentru mai multe informații, putem accesa  [man 2 select](#).


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


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

Argumentele funcției *select()*:

- *numfds* - valoarea cea mai mare + 1 a unui descriptor din oricare cele 3 mulțimi
- *readfds* - mulțimea cu descriptori de citire
- *writefds* - mulțimea cu descriptori pentru scriere
- *exceptfds* - mulțimea cu descriptori pentru care sunt în așteptare excepții
- *timeout* - timpul maxim în care apelul *select()* trebuie să întoarcă (daca este *NULL*, se blochează până când apare un eveniment pe cel puțin un descriptor).

Apelul *select()* primește ca argumente pointeri spre trei mulțimi de descriptori (de citire, scriere sau excepții). Dacă utilizatorul nu este interesat de anumite condiții, argumentul corespunzător va fi setat la *NULL* (la server, pe noi ne interesează doar mulțimea de citire).

 **Atenție**, *select()* modifică mulțimile de descriptori: după apel, ele vor conține numai descriptorii pe care s-au primit date. Astfel, trebuie să ținem copii ale mulțimilor originale.

Fiecare mulțime de descriptori este, de fapt, o structură care conține un tablou de măști de biți. Dimensiunea tabloului este dată de constanta *FD_SETSIZE* (o valoare uzuală a acestei constante este 1024; pentru lucrul cu descriptori mai mari de aceasta valoare, recomandam  [poll\(\)](#)). Pentru lucrul cu mulțimile de descriptori preluate ca argumente de apelul *select()*, se pot folosi o serie de macro-uri:

- *void FD_ZERO(fd_set *set)* - șterge în întregime mulțimea de descriptori de fișiere *set*
- *void FD_SET(int fd, fd_set *set)* - adaugă descriptorul *fd* în mulțimea *set*
- *void FD_CLR(int fd, fd_set *set)* - șterge descriptorul *fd* din mulțimea *set*
- *int FD_ISSET(int fd, fd_set *set)* - testează dacă descriptorul *fd* aparține sau nu mulțimii *set*; întoarce o valoare diferită de 0 in caz afirmativ

Timeout-ul este de tipul *struct timeval*, care are definiția următoare:

```
#include <sys/time.h>
struct timeval {
    long    tv_sec;        /* seconds */
    long    tv_usec;       /* microseconds */
};
```

Un exemplu de server TCP ce folosește apelul *select()* pentru multiplexare se află în arhiva suport a acestui laborator. În mulțimea de citire a serverului se află inițial descriptorul pentru socketul inactiv. Apoi, pe măsură ce se conectează clienții, în mulțime vor fi adăugați și descriptorii pentru socketii activi (cei pe care se trimit/primesc date la/de la clienți).

Exerciții

Pornind de la codul disponibil  aici, aveți de implementat următoarele cerințe:

1. Modificați programul client astfel încât să se comporte ca în laboratorul trecut (să citească de la tastatură și să trimită serverului, apoi să primească de la server și să afișeze). Modificați și programul server astfel încât să funcționeze cu 2 clienți: să trimită clientului 1 ce a primit de la clientul 2 și invers.
2. Modificați programul client astfel încât să multiplexeze între citirea de la tastatură (vom adăuga descriptorul 0 în mulțimea de citire pentru *select()*) și citirea de pe socket. Din acest moment, eliminăm neajunsul ordonării acțiunilor clienților.
3. Modificați programul server ca să funcționeze cu mai multi clienți. Clienții vor trimite în mesaj și destinația mesajului (acest lucru se poate face și fără modificarea codului clienților, vedeți exemplul). În cadrul acestui laborator, putem folosi descriptorul socketului întors de *accept()* ca identificator pentru un client (în aplicații reale, clienții nu au acces la aceste valori). Exemplu: clientul cu socketul 5 poate trimite (mesaj citit de la tastatură) "4 ce mai faci", iar serverul parsează mesajul și îl trimite clientului conectat pe socketul 4 (puteți să lucrați și cu o structură de mesaj).
4. **(Bonus)** Modificați programul server ca să trimită (la conectarea) clienților lista cu clienții deja conectați, apoi să trimita clienților conectați update-uri despre ce client a mai intrat/ieșit din sistem (puteți să folosiți același sistem de identificatori pentru clienți ca la punctul 3).

Search

Cursuri

- [Cursul 01.](#)
- [Cursul 02.](#)
- [Cursul 03.](#)
- [Cursul 04.](#)
- [Cursul 05.](#)
- [Cursul 06.](#)
- [Cursul 07.](#)
- [Cursul 08.](#)
- [Cursul 09.](#)
- [Cursul 10.](#)
- [Cursul 11.](#)
- [Cursul 12.](#)

Laboratoare

- [Laboratorul 01 - Notiuni pregatitoare pentru laboratorul de PC](#)
- [Laboratorul 02 - Folosirea unei legaturi de date pentru transmiterea unui fisier](#)
- [Laboratorul 03 - Implementarea unui protocol cu fereastra glisanta. Suma de control](#)
- [Laboratorul 04 - Forwarding](#)
- [Laboratorul 05 - ICMP](#)
- [Laboratorul 06 - Socketi UDP](#)
- [Laboratorul 07 - Protocolul de transport TCP](#)
- [Laboratorul 08 - TCP și multiplexare I/O](#)
- [Laboratorul 09 - Protocolul DNS](#)
- [Laboratorul 10 - Protocolul HTTP](#)
- [Laboratorul 11 - E-mail](#)
- [Laboratorul 12 - Protocoale de securitate. OpenSSL CLI tools](#)
- [Laboratorul 13 - Protocoale de securitate. utilizarea programatica](#)

Resurse

- [Mașina virtuală](#)

Table of Contents

- [Laboratorul 08 - TCP și multiplexare I/O](#)
 - [Obiective](#)
 - [Multiplexarea I/O](#)
 - [Exerciții](#)