

Programarea in retea (III)

Lenuta Alboaie
adria@info.uaic.ro

Cuprins

- Primitive I/O - discutii
- Server concurent UDP
- TCP sau UDP – aspecte
- Instrumente
- Alternative de proiectare si implementare al modelului client/server TCP

Primitive I/O

- Citire de date
 - read() / recv() / **readv()** / recvfrom() / **recvmsg()**
- Trimitere de date
 - write() / send() / **writv()** / sendto() / **sendmsg()**

Alte primitive | I/O

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

```
ssize_t readv (int filedes, const struct iovec *iov, int iovcnt);
```

```
ssize_t writev (int filedes, const struct iovec *iov, int iovcnt);
```

```
    struct iovec
```

```
    {
```

```
        void *iov_base; /* adresa de start a bufferului */
```

```
        size_t iov_len; /* dimensiunea bufferului */
```

```
    };
```

Mai generale decit *read()/write()*, ofera posibilitatea de a transmite date aflate in zone necontigue de memorie

Cele 2 apeluri returneaza la executie normala lungimea transferului in octeti

Alte primitive | I/O

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

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

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

Ambele functii au majoritatea optiunilor incorporate in structura *msghdr*

Cele mai generale functii I/O; apelurile
read/readv/recv/recvfrom pot fi inlocuite de recvmsg

Cele 2 apeluri returnează la execuție normală lungimea transferului în octeți; -1 in caz de eroare

Alte primitive | I/O

Comparatie intre primitivele I/O:

Function	Orice descriptor	Doar descriptor de socket	Un singur read/write buffer	Scatter/gather read/write	Flag-uri optionale	Adresa nodului <i>peer</i>
read, write	○		○			
readv, writev	○			○		
recv, send		○	○		○	
recvfrom, sendto		○	○		○	○
recvmsg, sendmsg		○		○	○	○

Server UDP | situatii

Majoritatea serverelor UDP sunt iterative

- Server UDP care citește cererea clientului, procesează cererea, trimite răspunsul și termină cu acel client
- Dacă este nevoie de schimb de datagrame multiple cu clientul?

Server UDP concurrent

- dacă elaborarea răspunsului ia mult timp serverul poate crea (*fork()*) un proces copil care va rezolva cererea

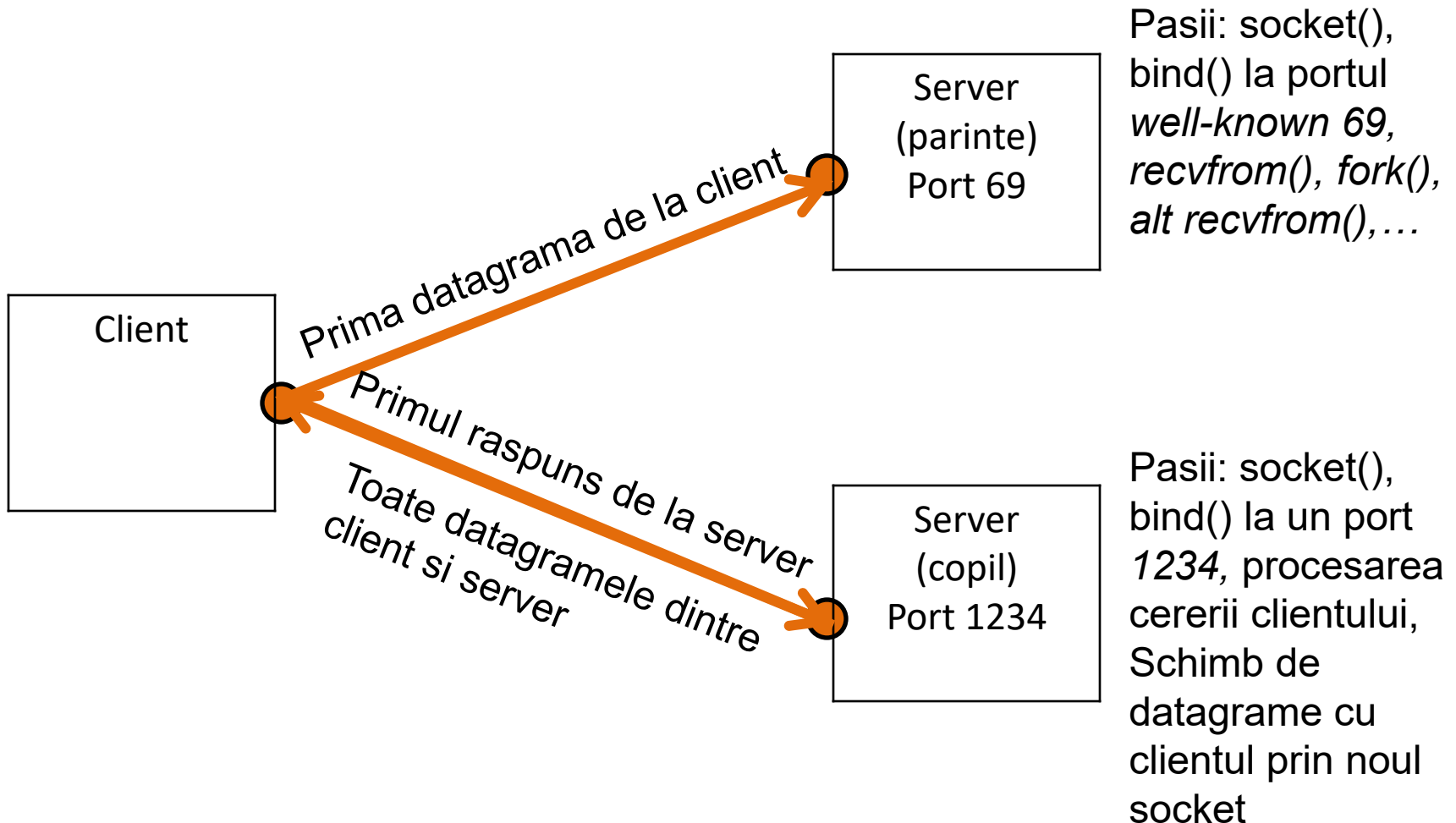
Server UDP | situatii

Server UDP concurent

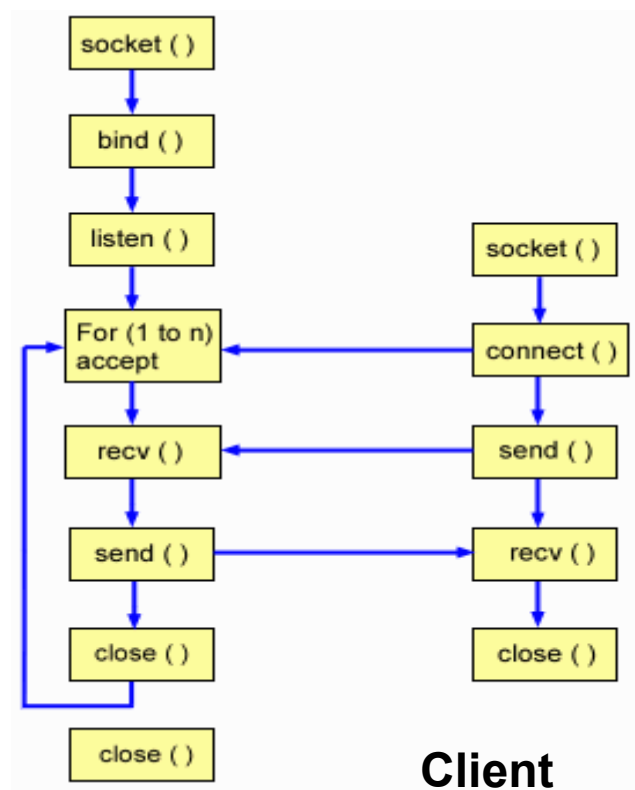
- Server UDP care schimba datagrame multiple cu un client
 - Problema: Doar un numar de port este cunoscut de client ca fiind un port “*wellknown*”
 - Solutia: serverul creaza un socket nou pentru fiecare client, si il ataseaza la un port “*efemer*”, si utilizeaza acest socket pentru toate raspunsurile.
 - Obligativu clientul trebuie sa preia din primul raspuns al serverului noul numar de port si sa faca urmatoarele cereri la acel port
 - Exemplu: TFTP - Trivial File Transfer Protocol

Server concurrent UDP

– TFTP utilizeaza UDP si portul 69

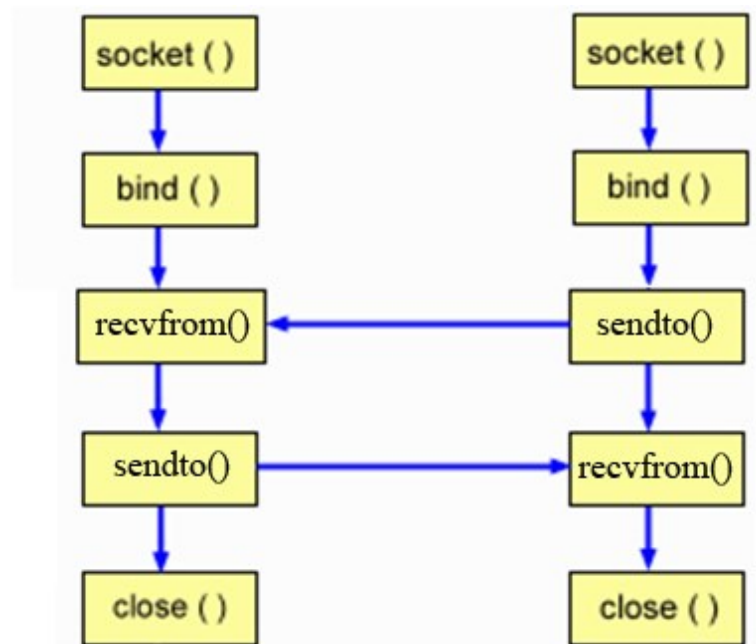


TCP sau UDP - discutii



Server

Client



Server UDP

Client UDP

Model server/client TCP

Model client/server UDP

TCP sau UDP – discutii

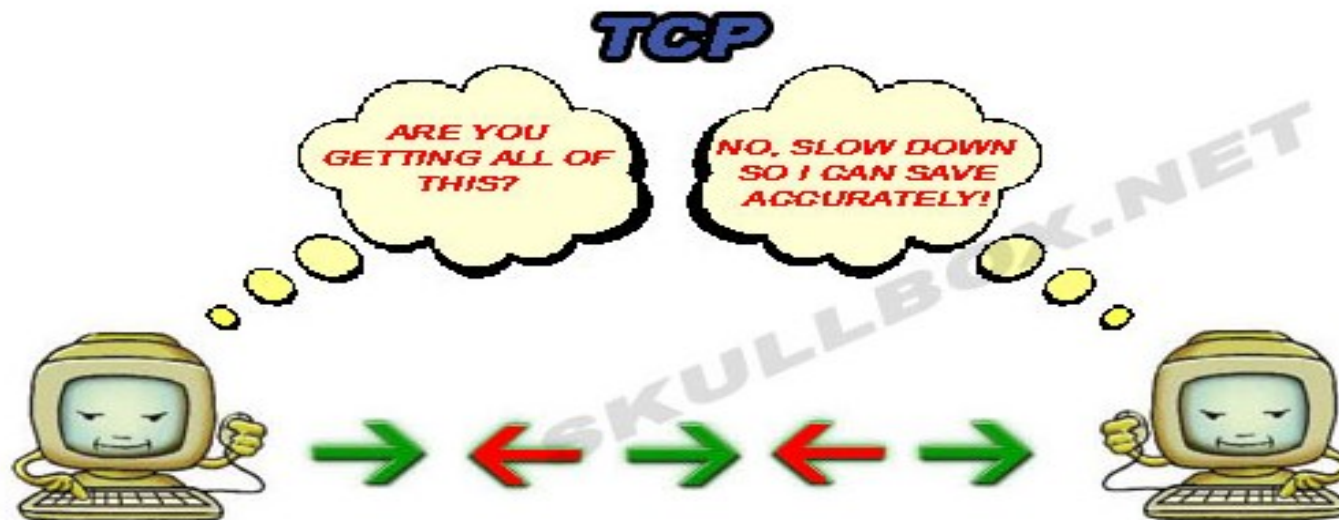
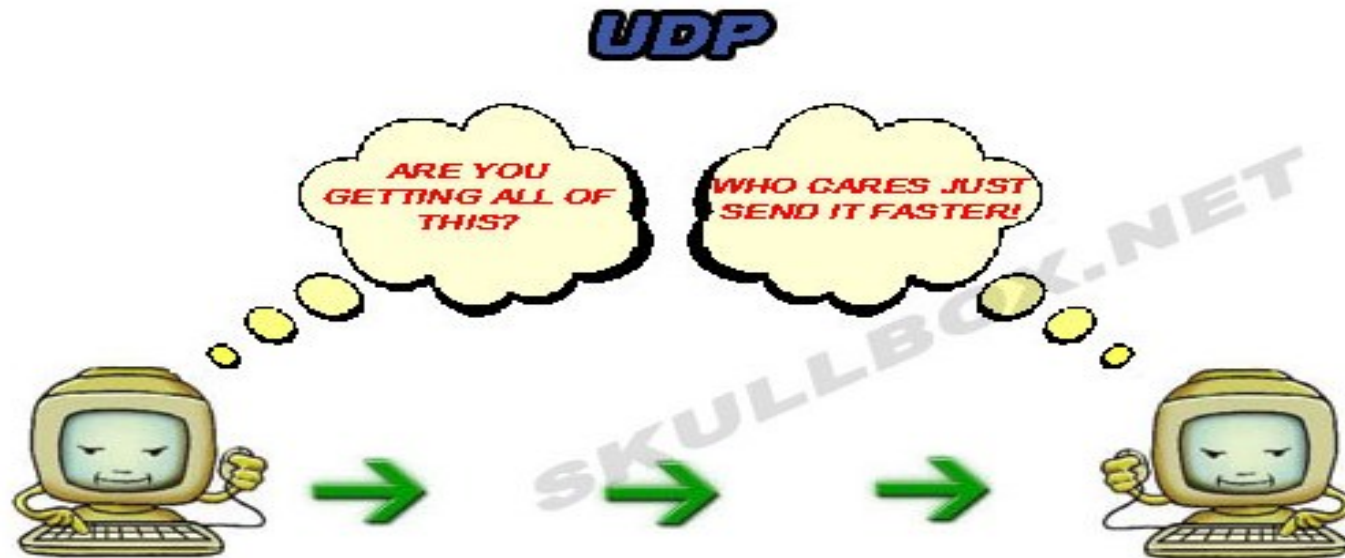
Aspecte privind utilizarea UDP:

- UDP suporta broadcasting si multicasting
- UDP nu are nevoie de un mecanism de stabilire a conexiunii
- Minimul de timp necesar unei tranzactii UDP cerere-raspuns este: $RRT(Round Trip Time) + SPT(server processing time)$

Aspecte privind utilizarea TCP:

- TCP suporta point-to-point
- TCP este orientat conexiune
- Oferă siguranță și asigură transmiterea în ordine a datelor;
- Oferă mecanisme de control al fluxului și control al congestiei
- Minimul de timp necesar unei tranzactii TCP cerere-raspuns dacă se creează o nouă conexiune este: $2 * RRT + SPT$

TCP sau UDP – discutii



[<http://www.skullbox.net>]

TCP sau UDP – discutii

Folosirea UDP , respectiv TCP – recomandari

- UDP *trebuie* folosit pentru aplicatii multicast sau broadcast

Controlul erorilor trebuie (eventual) adaugat la nivelul serverului sau clientului

- UDP *poate* fi folosit pentru operatii de cerere-raspuns simple; erorile trebuie tratate la nivelul aplicatiei

Exemple: streaming media, teleconferinte, DNS

TCP sau UDP – discutii

Folosirea UDP , respectiv TCP – recomandari

- TCP *trebuie* folosit pentru *bulk data transfer* (e.g. transfer de fisiere)
 - S-ar putea folosi UDP? → Reinventam TCP la nivelul aplicatiei!

Exemple: HTTP (Web), FTP (File Transfer Protocol),
Telnet, SMTP

Instrumente

- Multe sisteme UNIX ofera facilitatea de “*system call tracing*”

```
adria@ubu: ~/S6
I A test.c (Modifi Row 8 Col 28 8:15)
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    char *sir=NULL;
    printf("program de debugs: ");
    //sir = (char *) malloc(100*sizeof(char));
    fgets(sir, 1024, stdin);
    printf(sir);
    return 1;
}
```



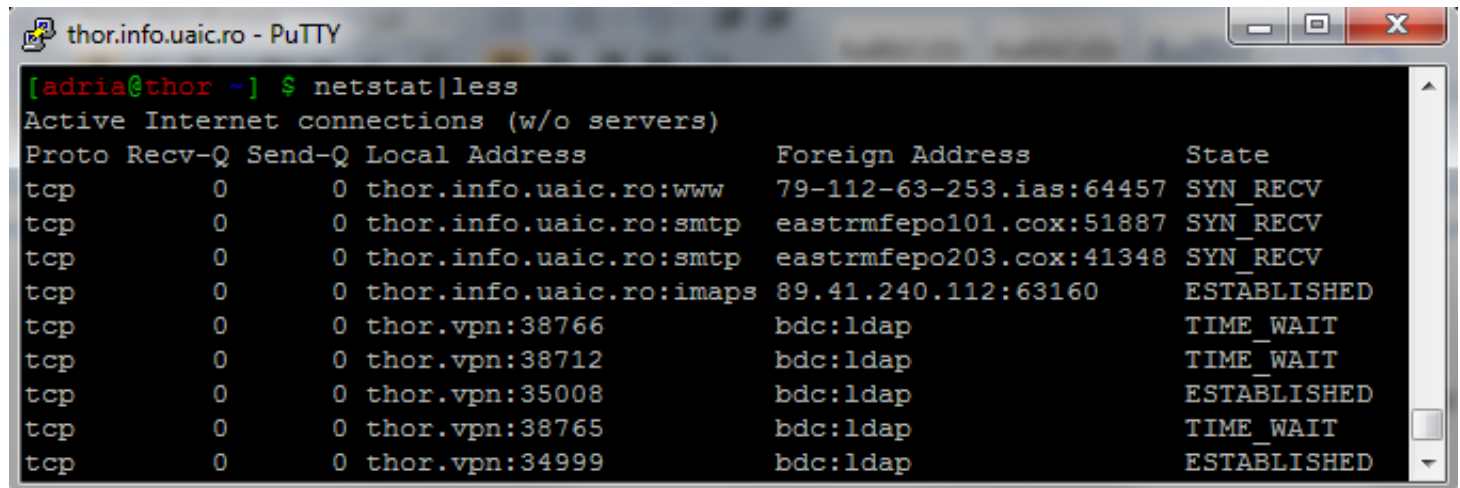
strace



```
write(1, "program de debug\n"... , 17program de debug
) = 17
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fdb000
read(0, 0xb7fdb000, 1024) = ? ERESTARTSYS (To be restarted)
--- SIGWINCH (Window changed) @ 0 (0) ---
read(0, Test in saptamina 6
"Test in saptamina 6\n"... , 1024) = 20
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
+++ killed by SIGSEGV +++
```

Instrumente

- Programe de test de dimensiuni reduse
- Instrumente:
 - **tcpdump** – majoritatea versiunilor de Unix
 - Oferă informații asupra pachetelor din rețea
 - <http://www.tcpdump.org/>
 - **snoop** – Solaris 2.x
 - **lsof**
 - Identifică ce procese au un socket deschis la o adresă IP sau port specificat
 - **netstat**



```
thor.info.uaic.ro - PuTTY
[adria@thor ~] $ netstat | less
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 thor.info.uaic.ro:www   79-112-63-253.ias:64457 SYN_RECV
tcp        0      0 thor.info.uaic.ro:smtp  eastrmfepo101.cox:51887 SYN_RECV
tcp        0      0 thor.info.uaic.ro:smtp  eastrmfepo203.cox:41348 SYN_RECV
tcp        0      0 thor.info.uaic.ro:imaps 89.41.240.112:63160    ESTABLISHED
tcp        0      0 thor.vpn:38766          bdc:ldap                TIME_WAIT
tcp        0      0 thor.vpn:38712          bdc:ldap                TIME_WAIT
tcp        0      0 thor.vpn:35008          bdc:ldap                ESTABLISHED
tcp        0      0 thor.vpn:38765          bdc:ldap                TIME_WAIT
tcp        0      0 thor.vpn:34999          bdc:ldap                ESTABLISHED
```


Instrumente

- Instrumente:
 - **tcptrack**



Client	Server	State	Idle	A	Speed
172.23.195.11:48328	67.39.222.44:22	ESTABLISHED	0s		38 KB/s
172.23.195.11:48646	196.30.80.10:80	ESTABLISHED	1s		30 KB/s
172.23.195.11:48661	64.37.246.17:80	ESTABLISHED	0s		387 B/s
172.23.195.11:48620	216.239.39.99:80	RESET	2s		0 B/s
128.230.225.95:3531	172.23.195.10:1220	ESTABLISHED	5s		0 B/s
172.23.195.11:48621	216.239.39.99:80	ESTABLISHED	7s		0 B/s
172.23.195.11:48606	64.233.167.99:80	ESTABLISHED	10s		0 B/s
172.23.195.11:48014	67.39.222.44:22	ESTABLISHED	16s		0 B/s
172.23.195.11:47988	67.39.222.44:22	ESTABLISHED	18s		0 B/s
TOTAL					69 KB/s
Connections 1-9 of 9					Unpaused Sorted

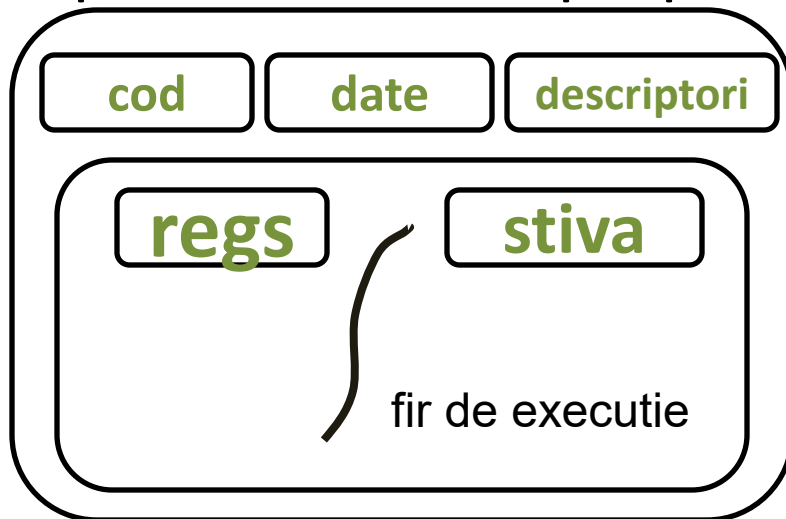
Alternative de proiectare si implementare al modelului client/server TCP

Fire de executie | Necesitate

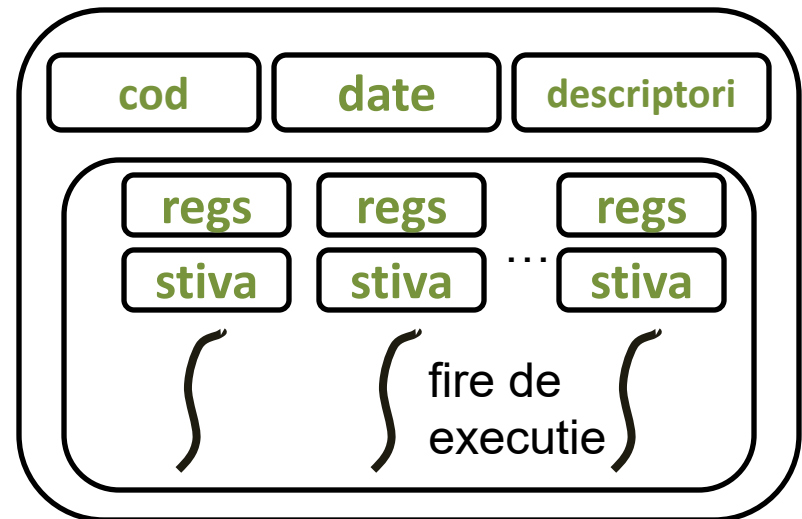
- `fork()` poate fi un mecanism costisitor
 - implementările curente folosesc mecanismul *copy-on-write*
- IPC (Inter-Process Communication) necesita trimiterea informatiei intre parinte si copil *dupa* `fork()`

Fire de executie | Caracteristici

- Firele de executie (*threads*) sunt numite si *lightweight processes (LWP)*
- Pot fi vazute ca un program aflat in executie fara spatiu de adresa proprie



Procese cu un fir de executie



Procese cu mai multe fire de executie

Procese, Fire de executie | Comparatii

- Exemplu: Costurile asociate crearii si managementului proceselor (50.000) este mai mare decat in cazul firelor de executie(50.000)

Platform	fork()		pthread_create()	
	user	sys	user	sys
AMD 2.4 GHz Opteron (8cpus/node)	2.2	15.7	0.3	1.3
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	30.7	27.6	0.6	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	48.6	47.2	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	1.5	20.8	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	1.1	22.2	1.2	0.6

[<https://computing.llnl.gov/tutorials/pthreads/>]

Fire de executie | Implementare

- **Pthreads** (POSIX Threads) standard ce definește un API pentru crearea și manipularea firelor de executie
 - FreeBSD
 - NetBSD
 - GNU/Linux
 - Mac OS X
 - Solaris
- Pthread API pentru Windows – pthreads-w32

Fire de executie | Primitive de baza

```
#include <pthread.h>
```

```
int pthread_create(
```

```
pthread_t *tid,
```

```
const pthread_attr_t *attr,
```

```
void *(*func)(void *),
```

```
void *arg);
```

pthread_t (-> adeseori un unsigned int)
(Identificatorul *thread*-ului)

Structura ce specifica attributele noului fir creat (e.g. dimensiunea stivei, prioritatea, NULL = comportamentul implicit)

Referinta la functia ce va fi executata de *thread*

Argumentul catre *thread* ce este transmis functiei

Returneaza: 0 in caz de succes

o valoare Exxx pozitiva in caz de eroare

Fire de executie | Primitive de baza

```
#include <pthread.h>
int pthread_join(
    pthread_t *tid,
    void **status );
```

Identificatorul *thread*-ului

... va stoca valoarea de *return* a *thread*-ului (un pointer la un obiect)

- Realizeaza asteptarea terminarii unui anumit *thread*

Returneaza: 0 in caz de succes

o valoare Exxx pozitiva in caz de eroare

Fire de executie | Primitive de baza

```
#include <pthread.h>
```

```
pthread_t pthread_self();
```



Identificatorul *thread*-ului

Returneaza: ID-ul *thread*-ului care a apelat primitiva

Fire de executie | Primitive de baza

```
#include <pthread.h>
```

```
int pthread_detach(pthread_t tid);
```



Identificatorul *thread*-ului

Thread-urile pot fi:

- *joinable*: cind thread-ul se termina, ID-ul si codul de iesire sunt pastrate pina cand se apeleaza `pthread_join()` ← comportament implicit
- *detached*: cand thread-ul se termina toate resursele sunt eliberate

Returneaza: 0 in caz de succes

o valoare Exxx pozitiva in caz de eroare

Exemplu: `pthread_detach(pthread_self());`

Fire de executie | Primitive de baza

```
#include <pthread.h>
```

```
void pthread_exit(void* status);
```

- Terminarea unui *thread*

Thread-urile se pot termina:

- Functia executata de *thread* returneaza (Obs. Valoarea de return este void * si va reprezenta codul de iesire a *thread*-ului)
- Daca functia *main* a procesului returneaza sau oricare din *thread*-uri a apelat *exit()*, procesul se termina

Fire de executie | Exemplu

Exemplu de server TCP concurent care nu foloseste *fork()* pentru a deservi clientii, ci foloseste *thread*-uri

Obs. Compilarea: **gcc -lpthread server.c** sau
gcc server.c -lpthread



DEMO

Alternative de proiectare al modelului client/server TCP

- **Client TCP - modelul uzual**

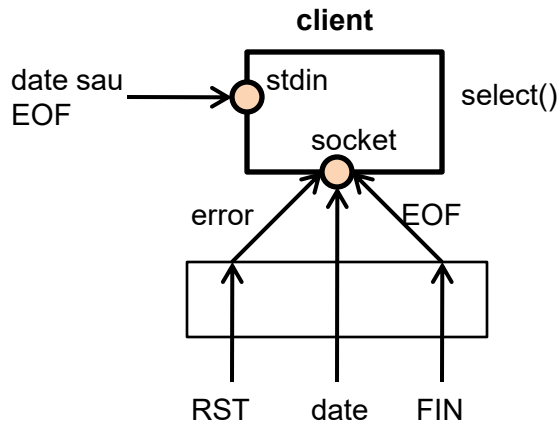
- Aspecte:

- Atat timp cat este blocat asteptind date de la utilizator, nu sesizeaza evenimentele de retea (e.g. *peer close()*)
 - Functioneaza in modul “*stop and wait*”
 - “*batch processing*”

Alternative de proiectare al modelului client/server TCP

- **Client TCP – utilizind `select()`**

- Clientul este notificat de evenimentele din retea in timp ce asteapta date de intrare de la utilizator



Daca *peer*-ul trimite date, *read()* returneaza o valoare >0 ;

Daca *peer*-ul TCP trimite FIN, *socket*-ul devine “citibil” si *read()* intoarce 0;

Daca *peer*-ul trimite RST (*peer*-ul a cazut sau a *rebootat*), *socket*-ul devine “citibil” si *read()* intoarce -1;

Aspecte:

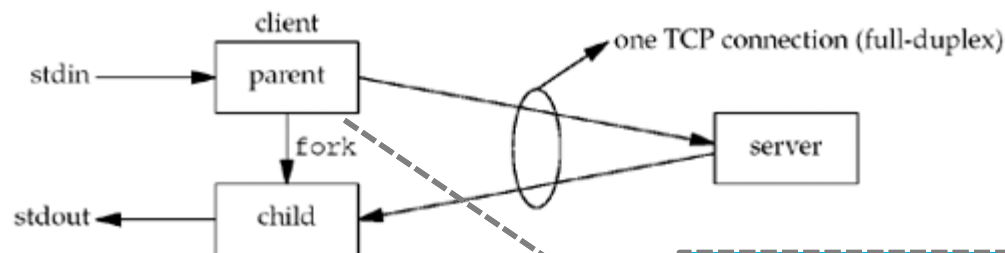
- Apelul *write()* poate fi blocant daca *buffer*-ul de la *socket*-ul emitator este plin

Alternative de proiectare al modelului client/server TCP

- **Client TCP** – utilizind `select()` si operatii I/O neblocante
 - Aspecte:
 - Implementare complexa => cand sunt necesare operatii I/O neblocante se recomanda utilizare de procese (*fork()*) sau de *thread*-uri (vezi *slide*-urile urmatoare)

Alternative de proiectare al modelului client/server TCP

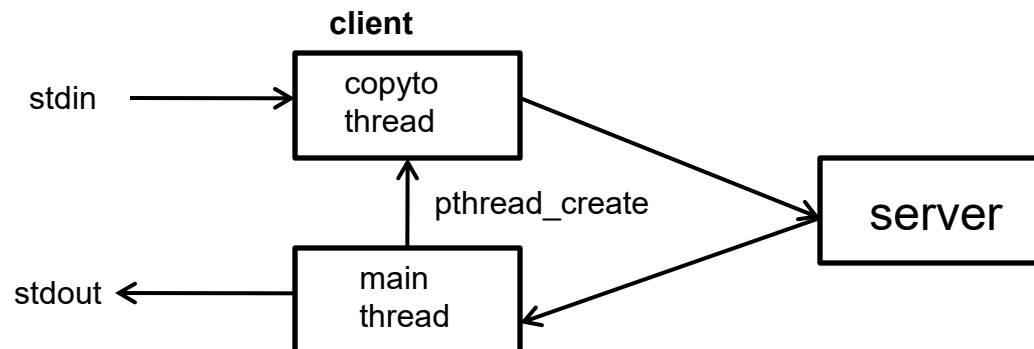
- **Client TCP – utilizind fork()**
 - Mecanismul de functionare:
 - exista doua procese
 - Un proces face managementul datelor client-server
 - Un proces face managementul datelor server-client



Parintele si copilul
partajeaza acelasi socket

Alternative de proiectare al modelului client/server TCP

- **Client TCP – utilizind pthread()**
 - Mecanismul de functionare:
 - exista doua fire de executie
 - Un fir de executie face managementul datelor client-server
 - Un fir de executie face managementul datelor server-client



Alternative de proiectare al modelului client/server TCP

- Comparatie a timpilor de executie a clientilor TCP cu arhitecturile client discutate

Tip client TCP	Timp executie (secunde)
Modelul uzual (stop-and-wait)	...
Modelul folosind select si I/O blocante	12.3
Modelul folosind select si I/O nebloccante	6.9
Modelul folosind fork()	8.7
Modelul folosind thread-uri	8.5

- Obs. Masuratoarea s-a realizat folosindu-se comanda *time* pentru implementari **client**/server *echo*

[Unix Network Programming, R. Stevens B.
Fenner, A. Rudoff - 2003

Alternative de proiectare al modelului client/server TCP

- **Server TCP – iterativ**

- Se realizeaza procesarea completa a cererii clientului inainte de a deservi urmatorul client

Aspecte:

- Sunt destul de rar intilnite in implementarile reale
- Un astfel de server serveste foarte rapid un client

Alternative de proiectare al modelului client/server TCP

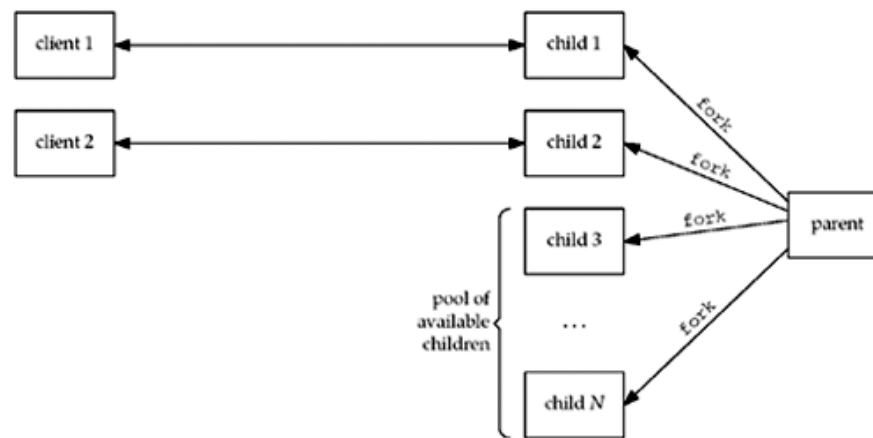
- **Server TCP** – cate un proces copil pentru fiecare client
 - Serverul deserveste clientii in mod simultan
 - Este des intilnit in practica
- Exemplu de mecanism folosit pentru distribuirea cererilor: *DNS round robin*

Aspecte:

- Crearea fiecarui copil (fork()) pentru fiecare client consuma mult timp de CPU

Alternative de proiectare al modelului client/server TCP

- **Server TCP – *preforking*; fara protectie pe `accept()`**
 - Serverul creaza un numar de procese copil cand este pornit, si apoi acestia sunt gata sa serveasca clientii



Aspecte

- Daca numarul de clienti este mai mare decat numarul de procese copil disponibile, clientul va resimti o “degradare” a raspunsului in raport cu factorul timp
- Acest timp de implementare merge pe sisteme ce au `accept()` primitiva de sistem

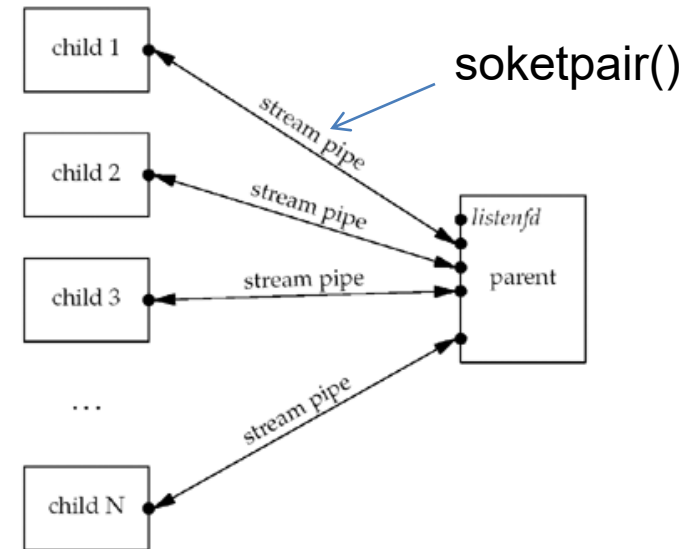
Alternative de proiectare al modelului client/server TCP

- **Server TCP** – *preforking*; cu blocare pentru protectia *accept()*
Implementare:
 - Serverul creaza un numar de procese copil cand este pornit, si apoi acestia sunt gata sa serveasca clientii
 - Se foloseste un mecanism de blocare (e.g. *fcntl()*) a apelului primitivei *accept()*, si doar un singur proces la un moment dat va putea apela *accept()*; procesele ramase vor fi blocate pina vor putea obtine accesul
- Exemplu: Apache (<http://www.apache.org>) foloseste tehnica de *preforking*

Alternative de proiectare al modelului client/server TCP

- **Server TCP – *preforking***; cu “transmiterea” socket-ului conectat
Implementare:

- Serverul creaza un numar de procese copil cand este pornit, si apoi acestia sunt gata sa serveasca clientii
- Procesul parinte este cel care apeleaza *accept()* si “transmite” *socket*-ul conectat la un copil



Aspecte:

Procesul parinte trebuie sa aiba evidenta actiunilor proceselor fii => o complexitate mai mare a implementarii

Alternative de proiectare al modelului client/server TCP

- **Server TCP** – cate un *thread* pentru fiecare client

Implementare:

Thread-ul principal este blocat la apelul lui `accept()` si de fiecare data cind este acceptat un client se creaza (`pthread_create()`) un *thread* care il va servi

DEMO (Slide 28)

Aspecte:

Aceasta implementare este in general mai rapida decat cea mai rapida versiune de server TCP *preforked*

Alternative de proiectare al modelului client/server TCP

- **Server TCP** – *prethreaded*; cu blocare pentru protectia *accept()*

Implementare:

- Serverul creaza un numar de *thread*-uri cand este pornit, si apoi acestea sunt gata sa serveasca clientii
- Se foloseste un mecanism de blocare (e.g. *mutex lock*) a apelului primitivei *accept()*, si doar un singur *thread* la un moment dat va apela *accept()*;

Obs. *Thread*-urile nu vor fi blocate in apelul *accept()*

DEMO

Alternative de proiectare al modelului client/server TCP

- **Server TCP – *prethreaded***; cu “transmiterea” socket-ului conectat

Implementare:

Serverul creaza un numar de *thread*-uri cand este pornit, si apoi acestia sunt gata sa serveasca clientii

Procesul parinte este cel care apeleaza *accept()* si “transmite” *socket*-ul conectat la un *thread* disponibil

Obs. Deoarece *thread*-urile si descriptorii sunt in cadrul aceluiasi proces, “transmiterea” *socket*-ului conectat inseamna de fapt ca *thread*-ul vizat sa stie numarul descriptorului

Alternative de proiectare al modelului client/server TCP | Discutii

- Daca serverul nu este foarte solicitat, varianta traditionala de server concurent (un *fork()* *per* client) este utilizabila
- Crearea unei multimi de procese copil (eng. *pool of children*) sau multimi de *thread*-uri (eng. *pool of threads*) este mai eficienta din punct de vedere al factorului timp; trebuie avut grija la monitorizarea numarului de procese libere, la cresterea sau descresterea acestui numar a.i. clientii sa fie serviti in mod dinamic
- Mecanismul prin care procesele copil sau *thread*-urile pot apela *accept()* este mai simplu si mai rapid decit cel in care *thread-ul* principal apeleaza *accept()* si apoi “transmite” descriptorul proceselor copil sau *thread*-urilor.
- Aplicatiile ce folosesc *thread*-uri sunt in general mai rapide decat daca utilizeaza procese, dar alegerea depinde de ce ofera SO sau de specificul problemei

Rezumat

- Primitive I/O - discutii
- Server concurrent UDP
- TCP sau UDP – aspecte
- Instrumente
- Alternative de proiectare si implementare al modelului client/server TCP

Bibliografie

- UNIX Network Programming: The sockets networking API, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff
- The Illustrated Network: How TCP/IP Works in a Modern Network (The Morgan Kaufmann Series in Networking), Walter Goralski



Intrebari?