Programarea in retea (IV)

Lenuta Alboaie
adria@infoiasi.ro

Cuprins

- Fire de executie (*thread*-uri)
- Alternative de proiectare si implementare al modelului client/server TCP

Fire de executie| Necesitate

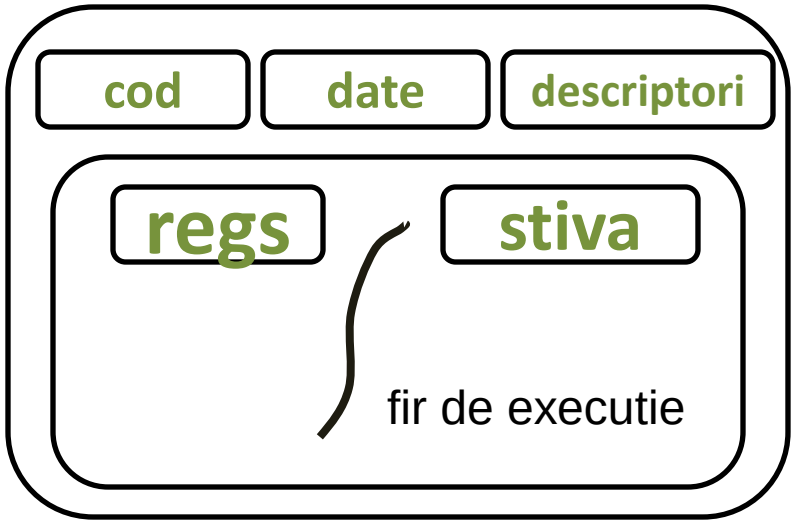
- fork() poate fi un mecanism costisitor
  - implementarile curente folosesc mecanismul *copy-on-write*
- IPC (Inter-Process Comunication) necesita trimiterea informatiei intre parinte si copil *dupa* fork()

Fire de executie| Caracteristici

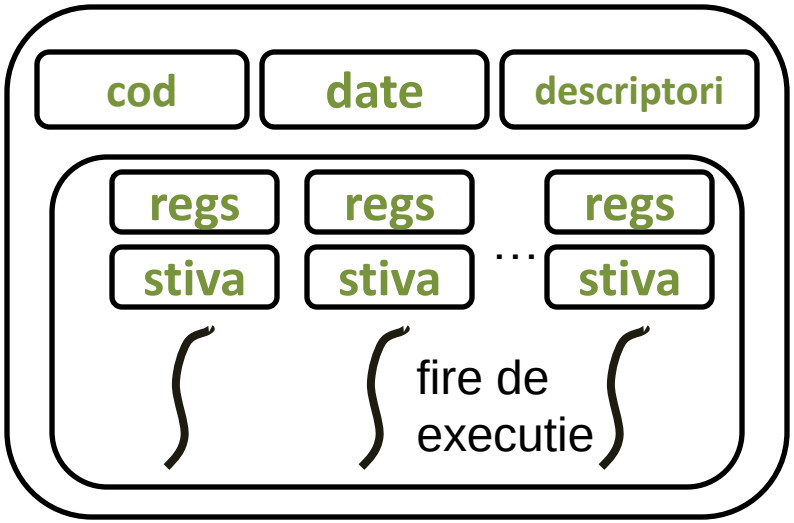
- Firele de executie (*threads*) sunt numite si

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 defineste un API pentru crearea si 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 atributele 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)

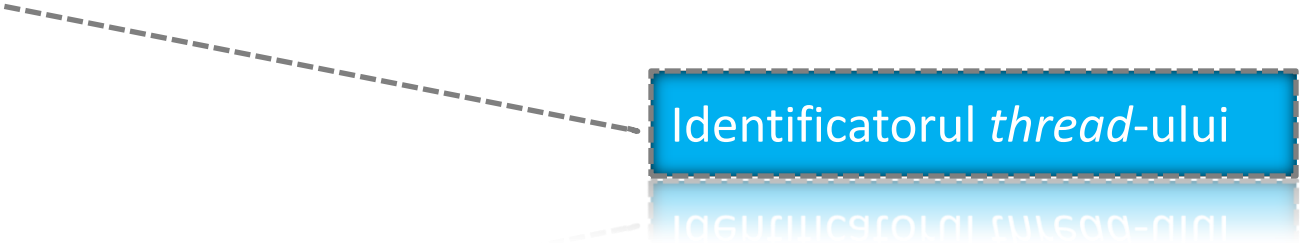
- Realizeaza asteptarea terminarii unui anumit *thread*

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();
```



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

Fire de executie| Primitive de baza

```
#include <pthread.h>
int pthread_detach(pthread_t tid);
```

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

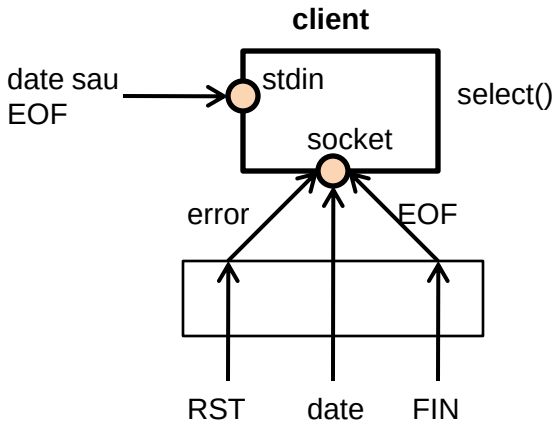
DEMO

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:

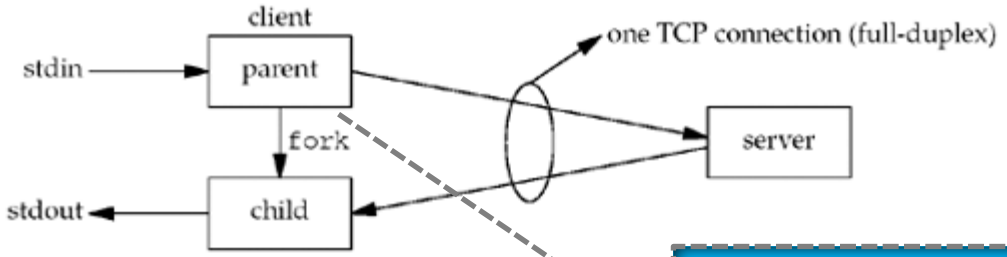
- Apelul *write*() poate fi blocant daca *buffer*-ul de la *socket*-ul emitator este plin

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

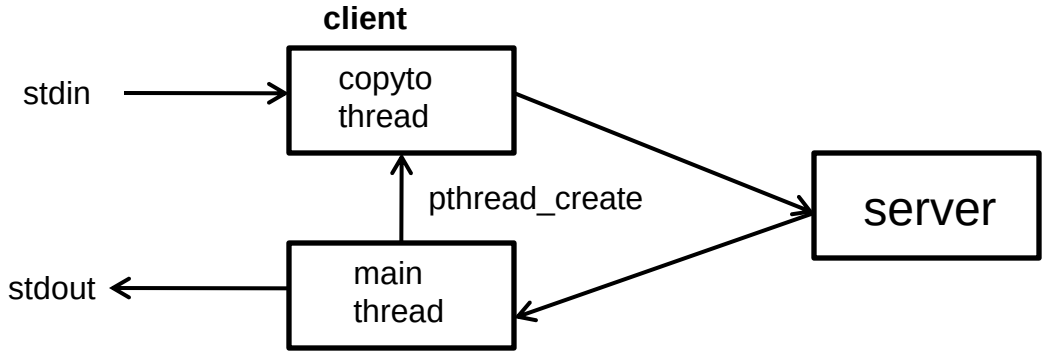
- **Client TCP – utilizind fork()**
  - Mecanismul de functionare:
    - exista doua procese
      - Un proces face managementul datelor client-server
      - Un proces face managementul datelor server-client



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 neblocante	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

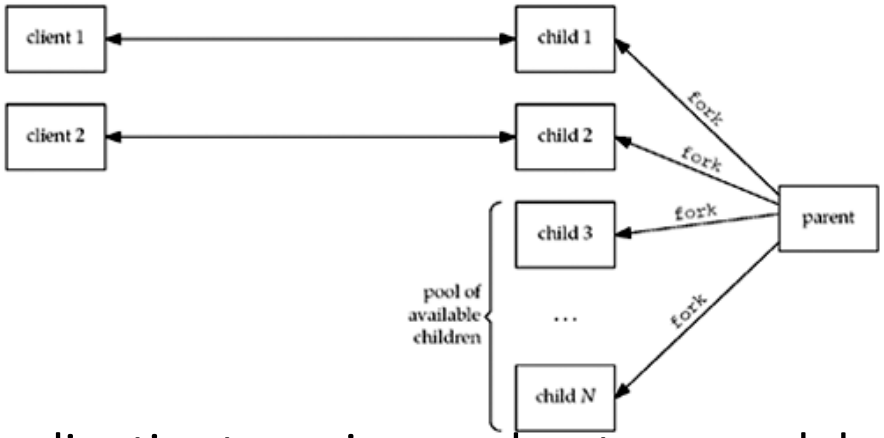
- **Server TCP** – cate un proces copil pentru fiecare client
  - Serverul deserveste clintii in mod simultan
  - Este des intilnit in practica
  - Exemplu de mecanism folosit pentru distribuirea cererilor: *DNS round robin*

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:

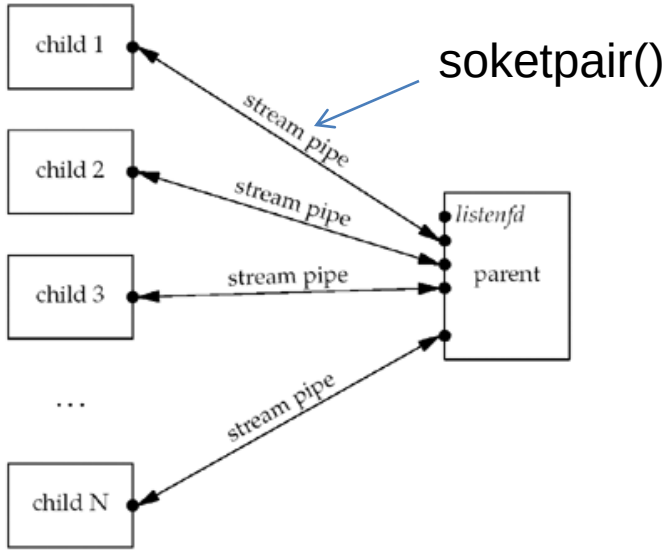
  – Serverul creaza un numar de procese copil cand este pornit, si apoi acestia sunt gata sa serveasca clientii

  – Se foloseste un mecansim 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 12)

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 mecansim 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

- Fire de executie (*thread*-uri)
- Alternative de proiectare si implementare al modelului client/server TCP

Intrebari?