Proiect Retele de Calculatoare : Lunch01

Silviu Rusu

Facultatea de Informatica, Universitatea Alexandru Ioan Cuza Iasi, Romania

[silviu.rusu@info.uaic.ro](mailto:silviu.rusu@info.uaic.ro)

1. Introducere

Prin intermediul acestui proiect mi-am propus sa aplic concepte ce tin de programarea in retea. In lucrare este implementata o situatie cunoscuta, concreta si anume dinamica servirii clientilor unei cantine.

1. Tehnologiile utilizate

Comunicarea in retea intre componentele aplicatiei se bazeaza la nivelul transport pe folosirea protocolului TCP. Orientarea catre conexiune a acestui protocol este principala caracteristica ce a determinat alegerea.

1. Arhitectura aplicatiei
   1. Concepte implicate

Aplicatia functioneaza dupa modelul client-server, cu tratarea clientilor in maniera concurenta. Server-ul foloseste tehnica de pre-threading, cu blocare la acceptarea clientilor prin utilizarea unui mutex; numarul de thread-uri dedicate clientilor ramane constant pe toata durata functionarii server-ului. Codul sursa al aplicatiei este scris in intregime in limbajul C; se foloseste Linux socket API, iar mecanismele prin care se realizeaza concurenta(thread-uri, mutex-uri) sunt de tip POSIX.

* 1. Diagrama aplicatiei



1. Detalii de implementare
   1. Server

Numarul maxim de conexiuni concurente N este definit ca o constanta.Se folosesc doua mutex-uri declarate global, accept\_lock,pentru protectia pe accept() si

ord\_by\_dish\_lock, pentru contorizarea comenzilor.

Comenzile sunt contorizate intr-un array:

int ordersByDish[TOTAL\_DISHES],iar felul preferat este pastrat in variabila favDish.Prin valoarea NOT\_AVAILABLE li se transmite thread-urilor din server faptul ca felul preferat nu a fost calculat inca.

Server-ul lanseaza cele N thread-uri care se vor ocupa de clienti, apoi in thread-ul principal asteapta T secunde dupa care calculeaza felul preferat:

while(1)

{

sleep(T);

//felul preferat se calculeaza din datele pana la momentul curent

pthread\_mutex\_lock(&ord\_by\_dish\_lock);

//celelalte thread-uri nu pot modifica datele culese

favDish = getFavDish();

//felul preferat este disponibil thread-urilor

pthread\_mutex\_unlock(&ord\_by\_dish\_lock);

favDish = NOT\_AVAILABLE;//se mai asteapta date

}

Timpul T dupa care server-ul primeste datele ii este transmis o singura data acestuia de un intermediar, programul clientLauncher, printr-o conexiune TCP separata.

Thread-urile folosesc procedura manageDish pentru a administra comenzile clientilor.

void manageDish(int cliSock)

{

int client\_dish, current\_favDish;

int nbytes;

//comunicam cat timp clientul nu a inchis conexiunea sau nu apar erori

while((nbytes = read(cliSock, &client\_dish, sizeof(int))) > 0){

//contorizeaza cererea

pthread\_mutex\_lock(&ord\_by\_dish\_lock);

++ordersByDish[client\_dish - 1];

pthread\_mutex\_unlock(&ord\_by\_dish\_lock);//permite altor thread-uri sa contorizeze

//asteapta stabilirea felului preferat

while(favDish == NOT\_AVAILABLE) ;

current\_favDish = favDish; //favDish se poate schimba intre timp

if(client\_dish == current\_favDish){

//terminam cu acest client

send\_dish\_state(cliSock, READY);

break;//nu mai asteptam comenzi de la client

}

else{

//continuam cu acest client

send\_dish\_state(cliSock, NOT\_READY);

}

}

if(nbytes < 0)

{

perror("[server] Eroare la read()\n");

//return errno;

}

}

* 1. Client

Caracterul aleator al alegerilor este implementat cu generatorul srand(time(NULL)) si functia rand().

* 1. Client Launcher

Scopul acestui intermediar este de a lansa un numar variabil de clienti care sa comunice cu server-ul.

Se foloseste acelasi mecanism ca in client pentru alegerile aleatoare.

Se transmite la server timpul T de primire a datelor de la clienti si se primeste, tot de la server, numarul maxim N de clienti ce pot fi conectati simultan.

Mecanismul de lansare:

for(;;)

{

//lansam clienti numai daca pot fi serviti

if(active\_clients < N)

{

current\_load\_size = rand() % (N - active\_clients) + 1;//numarul de clienti lansati in acest pas

active\_clients += current\_load\_size;

printf("\n[LAUNCHER] active\_clients : %d\n", active\_clients); fflush(stdout);

for(i = 0; i < current\_load\_size; ++i)

{

pid = fork();

if(pid == -1)

{

perror("fork");

exit(1);

}

else if(pid == 0)

{

//lansarea efectiva

execl("cLunch01","cLunch01",(char\*)NULL);

printf ("\nEroare exec() !\n"); fflush(stdout);

exit (1);

}

}

}

}

Eliberarea resurselor la terminarea clientilor se face intr-un thread separat, cu rutina urmatoare:

void \* track\_active\_clients(void \* arg)

{

while(1)

{

if(wait(NULL) != -1) --active\_clients;

}

}

* 1. Use case

Din T in T secunde (T < 120) serverul va primi un mesaj de la o parte din clienti ce va identifica printr-un numar un fel de mancare (de la 1 la 5) ales random de respectivul client.

Serverul va contoriza cererile iar pentru felul cel mai preferat (avand cele mai multe cereri) va trimite la clientii solicitanti ca raspuns: "Masa e servita".

Celorlalti clienti (care au ales alt fel), le va raspunde cu "Indisponibil".

Un client ce a fost servit cu pranzul va afisa un mesaj "Satul!" si isi va incheia executia.

Un client refuzat va mai incerca o cerere aleasa tot random dupa un interval ales de asemeni aleator (pana in 60 de secunde).

La trei cereri refuzate, un client va afisa "Schimb cantina! Aici mor de foame!" si isi va incheia executia.

1. Concluzii

Solutia curenta are dezavantajul ca nu adapteaza numarul de thread-uri la numarul de clienti.

Implementarea respecta specificatiile din enunt si aduce un plus prin utilizarea unui intermediar intre comunicarea dintre clientii efectivi si server.

1. Bibliografie
2. Linux man pages
3. Exemplele de pe

http://profs.info.uaic.ro/~adria/teach/courses/net/cursullaboratorul.php