

WebS

Tipa Adrian-Ionut

17.01.2020

Grupa A6

1 Introducere

In acest document voi prezenta aspectele de baza ale proiectului "WebS", ceea ce face, tehnologii folosite, o diagrama care sa cuprinda arhitectura aplicatiei, voi specifica ce protocol am folosit si voi preciza anumite detalii de implementare.

Aplicatia consta intr-un program server si unul client. In care clientul poate manipula sau extrage anumite informatii/documente apartinand serverului.

1.1 Descrierea proiectului.

Cele doua programe interactioneaza prin intermediul unor comenzi de care clientul dispune. De asemenea serverul accepta mai multi clienti conectati si ii serveste pe toti in acelasi timp.

Cand un client se conecteaza la server, acesta se poate autentifica, sau se poate inregistra in cazul in care nu dispune de un cont. De asemenea, optiunea de a parasii sesiunea este mereu disponibila.

Pana cand clientul nu se logheaza, nicio comanda in afara de cea pentru autentificare/inregistrare/exit nu vor fi luate in considerare.

Dupa conectarea acestuia la server, acesta dispune de o serie de comenzi, care ii permit sa vada continutul serverului(acesta contine doar fisiere html), sa stearga un fisier, sa extraga un fisier, sau sa adauge unul(doar fisiere de tip html).

Toate exceptiile sunt tratate, iar continutul unei pagini urcate sau extrase de pe server va fi transmis in totalitate.

Clientul se poate deloga, putandu-se loga ulterior cu un alt username, sau poate parasii oricand aplicatia fara repercusiuni. Iar clientului ii vor fi disponibile mereu comenzile cat si o scurta descriere a acestora, fiind accesibile prin comanda 'help'.

2 Tehnologii utilizate

2.1 TCP – Conectivitate

Comunicarea dintre client si server se realizeaza utilizand protocolul TCP din diverse motive:

- 1.)Asigura retransmiterea pachetelor, in cazul in care acestea se pierd pe drum.
 - 2.)Ordoneaza pachetele ajunse la destinatie.
 - 3.)Stabileste o conexiune intre server-client.
- Cu alte cuvinte TCP asigura o conexiune mai sigura a datelor.
- 4.)In cerinta imi este impus sa folosesc acest protocol.

2.2 Thread si Mutex – Concurenta serverului

Concurenta serverului se realizeaza folosind thread-uri.

In server, cand acesta va fi pornit se va specifica un numr maxim de clienti. Se vor crea mai apoi toate aceste threaduri, fiind blocate folosind mutex pana cand accepta un client de care se vor ocupa ulterior.

Am ales sa utilizez threadurile deoarece, daca sunt mai mult de 1 client conectat la server, pot exista cazuri cand acestia vor utiliza aceleasi variabile globale, sau vor extrage extrage acelasi fisier, sau unul va suprascrie un fisier, in timp ce altul va incerca sa-l extraga din server, etc. sunt multe alte cazuri care pot aparea, si este mult mai usor de eliminat aceste cazuri utilizand threaduri si mutex-uri decat procese (in cazul proceselor, variabilele globale modificate in timpul procesului, vor fi vizibile doar in cadrul procesului respectiv).

2.3 sqlite3 – Login si Autentificare

Pentru partea de login si inregistrare a unor clienti noi, am folosit sqlite3, pentru a tine o evidenta a username-urilor. Acest lucru impiedica vizualizarea/extragerea/alterarea continutului serverului de catre clienti care nu sunt inregistrati. Ajuta la mentinerea unei evidente a cator clienti sunt, si impiedica crearea mai multor clienti sub acelasi username.

2.4 Particularitati ale aplicatiei

De asemenea exista si comanda help care prezinta, la fiecare apelare o lista a comenzilor si o scurta descriere a acestora.

Tot in client avem pentru fiecare comanda in parte cate o functie, deoarece pentru fiecare comanda apar o serie de mesaje si cazuri specifice comenzii, iar pentru a le rezolva mai usor, am decis sa le tratez separat.

Cat despre server, totul se intampla intr-o singura functie, care se ocupa de fiecare mesaj primit de la client, chiar si de cele straine serverului.

Bineinteles ca mai folosesc si cateva functii, insa acele functii reprezinta deseori verificari sau sunt functii care se ocupa de baza de date.

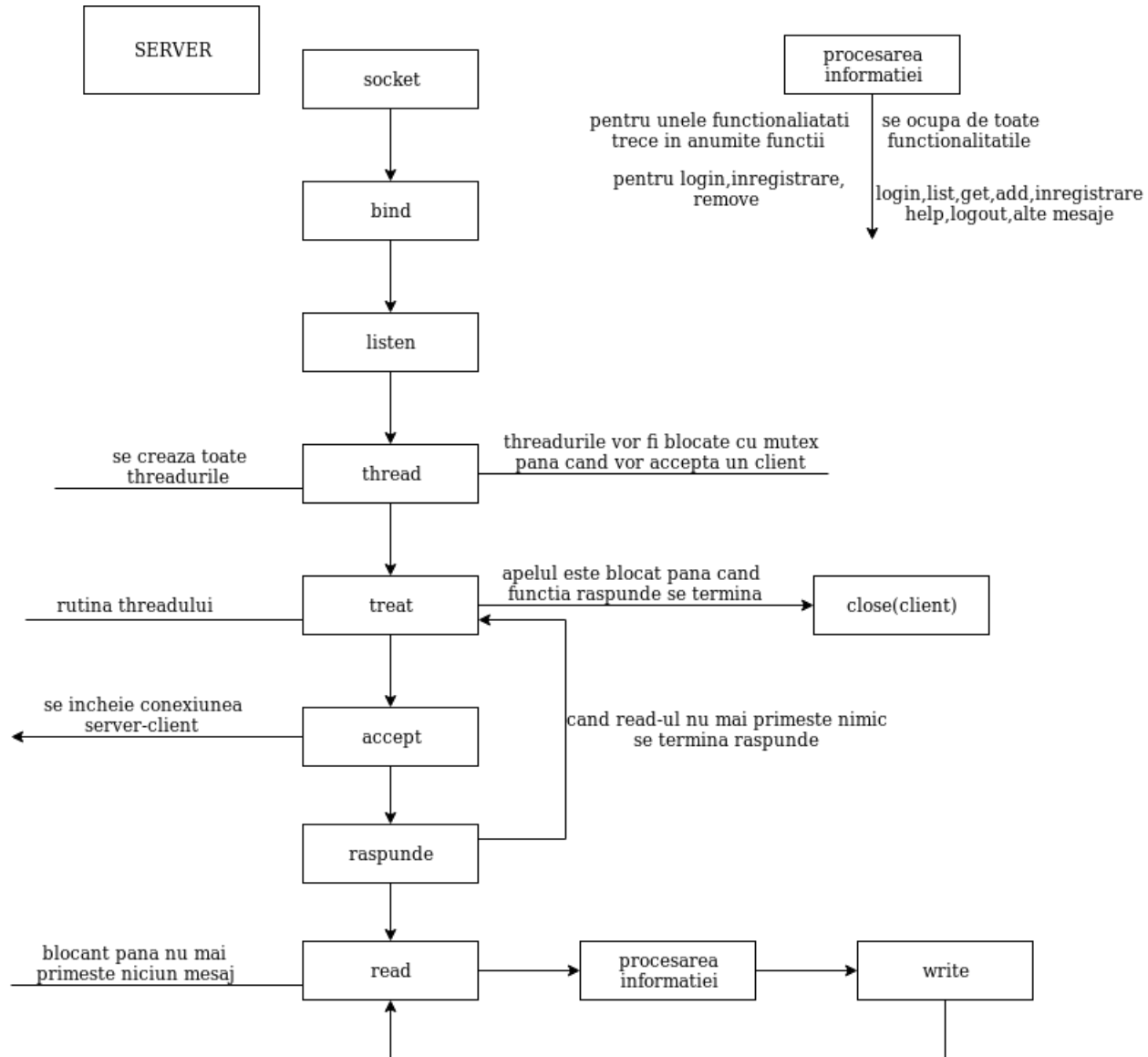
Un detaliu legat de transmiterea paginilor web, server \Rightarrow client si viceversa.

Inainte sa se execute transmiterea efectiva a datelor, se transmit dimensiunile fisierelor, astfel incat se va putea determina numarul de write-uri, respectiv read-uri, acest lucru se intampla pentru comenzile add/get. Deoarece doar in aceste cazuri mesajele ar putea depasi dimensiunea maxima fixata.

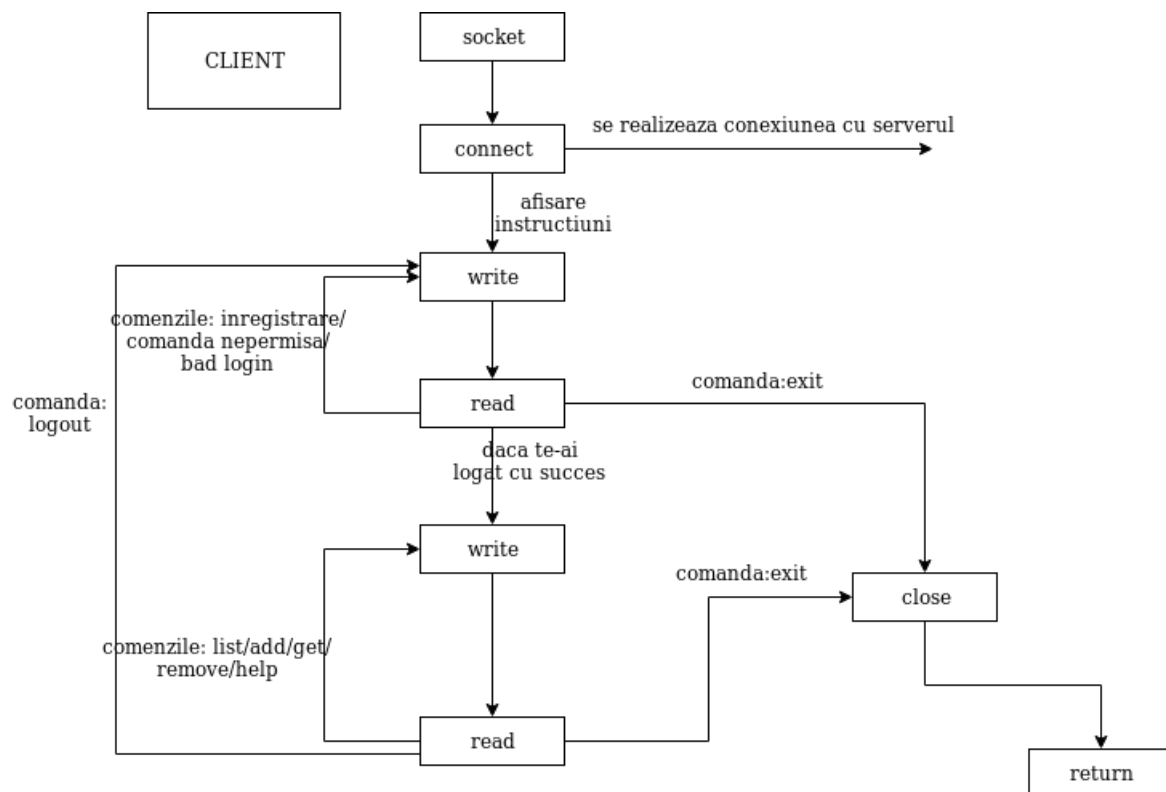
Unul dintre cele mai importante aspecte ale acestei aplicatii, este faptul ca in programul server, la pornirea acestuia se va specifica si un numar de threaduri care vor fi create, si care vor astepta sa se conecteze cu un client.

3 Arhitectura

3.1 Server



3.2 Client



4 Detalii de implementare

4.1 Server

Voi prezenta in urmatoarele poze cateva din partile importante ale codului din server.

Crearea threadurilor.

```
C servThreadM.c > treat(void *)
104 void threadCreate(int i)
105 {
106     void *treat(void *);
107
108     pthread_create(&threadsPool[i].idThread, NULL, &treat, (void *)i);
109     return; /* threadul principal returneaza */
110 }
111 void *treat(void *arg)
112 {
113     int client;
114
115     struct sockaddr_in from;
116     bzero(&from, sizeof(from));
117     printf("[thread]- %d - pornit...\n", (int)arg);
118     fflush(stdout);
119
120     for (;;)
121     {
122         int length = sizeof(from);
123         pthread_mutex_lock(&mlock);
124         //printf("Thread %d trezit\n", (int)arg);
125         if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
126         {
127             perror("[thread]Eroare la accept()...\n");
128         }
129         pthread_mutex_unlock(&mlock);
130         threadsPool[(int)arg].thCount++;
131
132         raspunde(client, (int)arg); //procesarea cererii
133         /* am terminat cu acest client, inchidem conexiunea */
134         close(client);
135     }
```

Funcțiile get si add din server. Get face write continuu, pana cand trimite tot continutul fisierului deschis, iar add primeste continuu pana cand copiaza tot continutul fisierului in fisierul nou creat in server.

```
C servThreadM.c x C cliThread.c uno.html / uno.html ~/../proiectR
C servThreadM.c > my_callback1(void *,int, char **, char **)
277     if (strcmp(s, "ok") == 0)
278     {
279         int size1;
280         fp2 = fopen(name, "r");
281         struct stat st;
282         stat(name, &st);
283         size1 = st.st_size;
284         //write number of bytes to client
285         printf("Dimensiune: %d biti.\n", size1);
286         if (write(cl, &size1, sizeof(int)) <= 0)
287         {
288             printf("[Thread %d] ", idThread);
289             perror("[Thread] Eroare la write() catre client.\n");
290         }
291         else
292             printf("[Thread %d] Mesajul a fost transmis cu succes.\n", idThread);
293         while ((character = fgetc(fp2)) != EOF) //citeste caracter cu caracter si trimite catre client
294         {
295             sprintf(content + (strlen(content)), "%c", character);
296             if (strlen(content) == 1023)
297             {
298                 if (write(cl, content, sizeof(content)) <= 0)
299                 {
300                     printf("[Thread %d] ", idThread);
301                     perror("[Thread] Eroare la write() catre client.\n");
302                 }
303                 else
304                     printf("[Thread %d] Mesajul a fost transmis cu succes.\n", idThread);
305                 strcpy(content, "");
306             }
307         }
308         if (write(cl, content, sizeof(content)) <= 0)
```

```
C servThreadM.c x C cliThread.c uno.html / uno.html ~/../proiectR
C servThreadM.c > my_callback1(void *,int, char **, char **)
429     if (strcmp("ok", s) == 0)
430     {
431         removefile(name);
432         FILE *fp3;
433         int size1;
434         if (fp3 = fopen(name, "w"))
435             fclose(fp3);
436         if (read(cl, &size1, sizeof(int)) <= 0)
437         {
438             printf("[Thread %d]\n", idThread);
439             perror("Eroare la read() de la client.\n");
440         }
441         printf("Dimensiune: %d biti.\n", size1);
442         while ((size1 - 1023) > 0)
443         {
444             if (read(cl, &s, sizeof(s)) <= 0)
445             {
446                 printf("[Thread %d] ", idThread);
447                 perror("[Thread] Eroare la read() catre client.\n");
448             }
449             else
450                 printf("[Thread %d] Mesajul a fost primit cu succes.\n", idThread);
451             writefile(name, s);
452             size1 = size1 - 1023;
453         }
454         if (read(cl, &s, sizeof(s)) <= 0)
455         {
456             printf("[Thread %d] ", idThread);
457             perror("[Thread] Eroare la read() catre client.\n");
458         }
459         else
460             printf("[Thread %d] Mesajul a fost primit cu succes.\n", idThread);
```

Functia de login si de inregistrare, in care se face si conexiunea la baza de date.

```
servThreadM.c > my_callback1(void *, int, char **, char **)
598 }
599 int login(char name[1024])
600 {
601     sqlite3 *db;
602     char *ErrMsg = 0;
603     int errcatch;
604     char *sql;
605     char *data;
606     strcpy(name_global, name);
607     printf("%s", name_global);
608     /* Open database */
609     errcatch = sqlite3_open("/home/adi/Desktop/proiectRMutex/database.db", &db);
610
611     if (errcatch)
612     {
613         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
614         return (0);
615     }
616     sql = "SELECT * from USERS";
617
618     errcatch = sqlite3_exec(db, sql, my_callback1, (void *)data, &ErrMsg);
619     if (errcatch != SQLITE_OK)
620     {
621         fprintf(stderr, "SQL error: %s\n", ErrMsg);
622         sqlite3_free(ErrMsg);
623     }
624     sqlite3_close(db);
625     if (strcmp(name_global, "ok") == 0)
626         return 1;
627
628     return 0;
629 }
```



```

585 }
586 char name_global[100];
587 static int my_callback1(void *data, int argc, char **argv, char **azColName)
588 {
589     int i;
590     for (i = 0; i < argc; i++)
591     {
592         if (strcmp(name_global, argv[i]) == 0)
593         {
594             strcpy(name_global, "ok");
595         }
596     }
597     return 0;
598 }
599 int login(char name[1024])

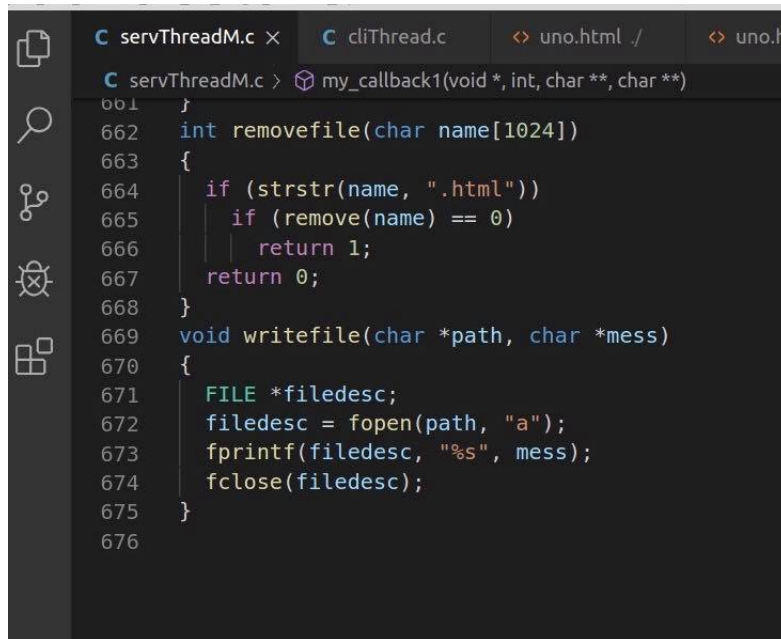
```

```

C servThreadM.c > my_callback1(void *,int, char **, char **)
633
634 int record(char name[1024]) {
635     sqlite3 *db;
636     char *ErrMsg = 0;
637     int rc;
638     char sql[1024];
639     /* Open database */
640     rc = sqlite3_open("/home/adi/Desktop/proiectRMutex/database.db", &db);
641     sql[0]=0;
642     if( rc ) {
643
644         return 0;
645     }
646
647     strcpy(sql,"");
648     if(login(name))
649     {
650         return 0;
651     }
652     else
653     sprintf(sql,"INSERT INTO USERS (USERNAME) VALUES ('%s'); ",name);
654     rc = sqlite3_exec(db, sql, my_callback2, 0, &ErrMsg);
655
656     if( rc != SQLITE_OK ){
657         return 0;
658     }
659     sqlite3_close(db);
660     return 1;
661 }
662 int removefile(char name[1024])
663 {
664     if (strstr(name, ".html"))

```

De asemenea si functia care se ocupa de stergerea paginilor html din server. Si o functie de scriere in fisier, aceasta functie apare si in client, si este folosita atat in get(client) cat si in add(server).

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'servThreadM.c' (active), 'cliThread.c', 'uno.html ./', and 'uno.h'. The active tab 'servThreadM.c' shows C code. The cursor is at line 661, column 1. The code includes a function 'removefile' and a function 'writefile'.

```
661 }
662 int removefile(char name[1024])
663 {
664     if (strstr(name, ".html"))
665         if (remove(name) == 0)
666             return 1;
667     return 0;
668 }
669 void writefile(char *path, char *mess)
670 {
671     FILE *filedesc;
672     filedesc = fopen(path, "a");
673     fprintf(filedesc, "%s", mess);
674     fclose(filedesc);
675 }
676
```

4.2 Client

In client printre cele mai importante bucati de cod, sunt cele care se ocupa de comezile get si add. Mai exact de trimiterea continutului paginilor.

```

C servThreadM.c  C cliThread.c  uno.html ./  uno.html ~/.../proiectR
C cliThread.c > writefile(char *, char *)
129 FILE *fp1;
130 if (fp1 = fopen(path, "w+"))
131 {
132     if (write(sd, "ok", 3) <= 0)
133     {
134         perror("[client]Eroare la write() spre server.\n");
135         return errno;
136     }
137     //read number of bytes to read from server
138     if (read(sd, &size1, sizeof(int)) < 0)
139     {
140         perror("[client]Eroare la read() de la server.\n");
141         return errno;
142     }
143     close(fp1);
144     printf("\nDimesiune: %d biti.", size1);
145     while ((size1 - 1023) > 0)
146     {
147         if (read(sd, &message, sizeof(message)) < 0)
148         {
149             perror("[client]Eroare la read() de la server.\n");
150             return errno;
151         }
152         writefile(path, message);
153         size1 = size1 - 1023;
154     }
155     if (read(sd, &message, sizeof(message)) < 0)
156     {
157         perror("[client]Eroare la read() de la server.\n");
158         return errno;
159     }
160     writefile(path, message);

```

```

C cliThread.c > writefile(char *, char *)
240     }
241     char character;
242     int size1;
243     struct stat st;
244     stat(path, &st);
245     size1 = st.st_size;
246     printf("\nDimensiune: %d biti.", size1);
247     if (write(sd, &size1, sizeof(int)) <= 0)
248     {
249         perror("[client]Eroare la write() spre server.\n");
250         return errno;
251     }
252     while ((character = fgetc(fp4)) != EOF)
253     {
254         sprintf(content + (strlen(content)), "%c", character);
255         if (strlen(content) == 1023)
256         {
257             if (write(sd, &content, sizeof(content)) <= 0)
258             {
259                 perror("[client]Eroare la write() spre server.\n");
260                 return errno;
261             }
262             strcpy(content, "");
263         }
264     }
265     if (write(sd, &content, sizeof(content)) <= 0)
266     {
267         perror("[client]Eroare la write() spre server.\n");
268         return errno;
269     }
270     fclose(fp4);
271     return 1;

```

5 Idei si Concluzie

Codul ar mai putea fi imbunatatit, unele instructiuni read-write ar putea fi scoase, poate serverul ar mai putea fi descarcat putin, unele conditii sau afisari putand fi puse direct in programul client.

La login ar putea fi puse parole si o interdictie de a fi conectati doi clienti in acelasi timp.

De asemenea ar fi fost o idee buna ca pentru fiecare user sa avem o baza de date care retine activitatea lor pe server.

In concluzie, acest proiect este unul destul de captivant care lasa foarte mult loc de imbunatatiri.

6 Bibliografie

1. Computer Networks - FII Course, <https://profs.info.uaic.ro/computer-networks/cursullaboratorul.php>
2. sqlite3 tutorial, <https://www.tutorialspoint.com/sqlite/sqliteccpp.htm>
3. printing files from a dir, <https://www.geeksforgeeks.org/c-program-list-files-sub-directories-directory/>