Universitat de Girona

XARXES

PRÀCTICA 2: L'APLICACIÓ UEB AMB *SOCKETS* TCP/IP − PART I

MEMÒRIA

Jordi Badia Auladell, número UdG, u1978902@campu.udg.edu, GEINF
Aniol Juanola Vilalta, u1978893, u1978893@campus.udg.edu, GEINF
Dia i hora del grup de pràctiques i nom del professor
GP3 (Dimarts 15h-16h), Joaquim Puiggalí Tesouro

Girona, novembre de 2023

Continguts

En aquesta pràctica [1] s'ha dissenyat i construït les parts bàsiques de l'aplicació UEB, una aplicació en xarxa (o aplicació distribuïda) Client-Servidor (C-S) inspirada en l'aplicació *web* "real", però amb importants simplificacions. Per a construir-la s'ha fet servir la interfície de *sockets* TCP/IP [2].

1 Requisits mínims i millores

Els requisits (R) mínims fets són els següents:

- R1. Construcció d'una aplicació UEB, seguint un model C-S.
- R2. La interfície aplicació-usuari del C (execució en un terminal)
- R3. La interfície aplicació-usuari (administrador) del S (execució en un terminal)
- R4. Programació en llenguatge C i estructura del codi en diversos fitxers, per separar "interfície aplicació-usuari", "capa d'aplicació de UEB" i capa de transport TCP
- R5. Estudi dels protocols d'aplicació i transport i dels sockets TCP
- R6. La vostra aplicació UEB ha de funcionar amb les dels altres estudiants

Les millores senzilles (MS) fetes són les següents:

- MS1. Configuració de la carpeta "arrel del lloc UEB" en el S
- MS2. Fitxer de "log" en el S
- MS3. Afegir dues notificacions més del S al C en el protocol PUEB (una part)
- MS4. El temps de resposta, el temps d'enviament i la velocitat efectiva

2 L'arquitectura en capes

L'aplicació UEB segueix el model C-S i la seva arquitectura en forma de capes segueix el model de referència TCP/IP d'Internet, és a dir, el model de 3 capes d'Aplicació (A), Transport (T) i Xarxa (X), la darrera formada per les capes d'Interxarxa (I) i de xarxa. A la Fig. 1 es mostren les capes de l'aplicació UEB.

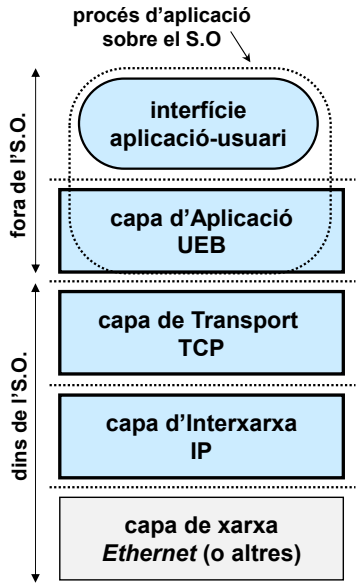

Figura 1: L'arquitectura en capes de l'aplicació UEB.

Suposarem que la capa de xarxa és *Ethernet* ja que les nostres estacions tenen una interfície *Ethernet*, però de fet podria ser qualsevol altra tecnologia de xarxa (p.e., Wi-Fi, 5G, etc.).

3 La interfície aplicació-usuari

La interfície aplicació-usuari tant del C com del S UEB són molt senzilles. De manera resumida (veieu els detalls a [1]), en el C, en un terminal, l'usuari entra per teclat l'identificador del fitxer que vol obtenir, un cop rep el fitxer "l'aboca" a la pantalla i el desa, i així contínuament fins que l'usuari indica que vol acabar l'execució; en el S, en un terminal, l'administrador l'engega, i cada cop que rep una petició en mostra informació a la pantalla. A la Fig. 1 es mostra la captura de pantalla d'una execució del C-S UEB.

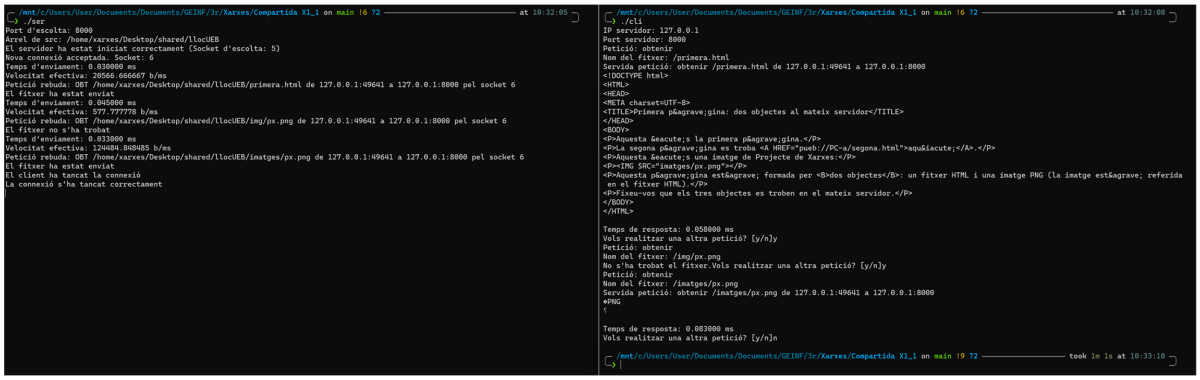

Figura 2: La interfície aplicació-usuari del C i S UEB.

A la funció main() del C i del S UEB s'han fet servir dos grups de funcions, i) les relatives a la interacció entre l'usuari i l'aplicació (via teclat, pantalla, etc.) i ii) les de la interfície de la capa d'aplicació de UEB. Les del segon grup es descriuen més avall, mentre que les del primer grup es descriuen a continuació.

Les funcions del main() relatives a la interacció entre l'usuari i l'aplicació són les següents:

- printf(), llibreria <stdio.h>: mostra a pantalla
- scanf(), llibreria <stdio.h>: llegeix via teclat
- open(), llibreria <fcntl.h>: crea el fitxer al client
- write(), llibreria <unistd.h>: escriu el fitxer rebut
- fopen(), llibreria <stdio.h>: obre el fitxer de configuració del servidor
- fgets(), llibreria <stdio.h>: llegeix línies del fitxer de configuració del servidor
- strerror(), llibreria <errno.h>: retorna el missatge d'error de les funcions del sistema com a string

4 Els serveis de les capes

Els serveis d'una capa són les tasques que la capa fa, la funcionalitat que proporciona.

El servei de la capa d'aplicació de UEB és:

- Inicialitza el servidor.
- Connecta el client de l'aplicació UEB al servidor UEB.
- Demanda peticions UEB al servidor per part del client.
- El servidor respon les peticions UEB del client.
- Tanca la connexió UEB.

El servei de la capa de transport TCP és:

- Inicialitzar un socket d'escolta (pel servidor).
- Inicialitzar una connexió a un servidor en mode escolta (pel client).
- Tancar la connexió actual.
- Rebre i enviar missatges TCP.

- Donat un socket, retornar la IP i port del propi socket i del peer connectat.

Els servei de les capes d'Interxarxa IP i de xarxa *Ethernet* són:

- El d'IP és portar paquets entres dues estacions d'Internet i el d'*Ethernet* portar paquets entre dues estacions d'una xarxa *Ethernet*.

5 Les interfícies de les capes

La interfície d'una capa són el conjunt de "mètodes" o "funcions" amb les quals es poden accedir als serveis de la capa.

En aquesta pràctica s'ha construït l'aplicació UEB a partir de la interfície de la capa de transport TCP. En la construcció s'ha definit una interfície de la capa d'aplicació UEB.

5.1 La interfície de la capa d'aplicació de UEB

Les funcions de la interfície de la capa UEB són les següents:

- Pel client:
    - **UEBc_DemanaConnexio**: Donada una ip i un port de servidor, retorna la ip i el port del client, un text de resultat i l'identificador del socket TCP connectat, o -1 si ha fallat.
    - **UEBc_ObteFitxer**: Donat un socket, un nom de fitxer i una longitud del nom, el demana al servidor i retorna el fitxer o l'error i un comentari sobre l'èxit o fracàs de la funció
    - **UEBc_TancaConnexio**: Donat un socket, tanca la connexió del client i retorna l'èxit o fracàs de la funció i un comentari sobre aquest.
- Pel servidor:
    - **UEBs_IniciaServ**: Donat un port inicialitza el socket d'ecolta i retorna l'èxit de la operació, amb un comentari sobre aquest.
    - **UEBs_AcceptaConnexio**: Donat un socket d'escolta accepta i retorna IP i port del client i del servidor i el socket TCP de connexió. En cas d'error retorna el codi d'error i un comentari.
    - **UEBs_ServeixPeticio**: Donat un socket de connexió, rep la petició, la deconstrueix, busca possibles errors (els retorna amb un comentari), construeix la resposta a la petició i l'envia.
        - Retorna el nom del fitxer demanat i el tipus de petició
        - Modificacions: rep el paràmetre "path" per canviar el directori de sources.
    - **UEBs_TencaConnexio**: Donat un socket, tanca la connexió del servidor i retorna l'èxit o fracàs de la funció i un comentari sobre aquest.

5.2 La interfície de la capa de transport TCP

La capa de transport TCP es troba dins el Sistema Operatiu (S.O.) i la seva interfície es coneix com la interfície de *sockets* TCP. Però en lloc de fer servir la interfície "original" de *sockets* TCP, a sobre seu s'ha construït una "nova" interfície, més senzilla de fer servir. Les funcions d'aquesta "nova" interfície de la capa TCP, tTCP, són les següents:

- **TCP_CreaSockClient**: Donada una adreça IP i un port, crea un socket i el retorna.
- **TCP_CreaSockServidor**: Donada una IP i un port, crea un socket d'escolta i el retorna.
- **TCP_DemanaConnexio**: Donat un socket, una IP i un port, es demana una connexió mitjançant el socket prèviament creat i es retorna el descriptor de la connexió establerta.
- **TCP_AcceptaConnexio**: Donat un socket d'escolta, el servidor espera a rebre una connexió i, un cop establerta i acceptada, retorna el descriptor d'aquesta nova connexió.
- **TCP_Envia**: Donat un descriptor, envia una seqüència de bytes pel descriptor.
- **TCP_Rep**: Donat un descriptor, rep una seqüència de bytes pel descriptor.
- **TCP_TancaSock**: Tanca el socket o descriptor passat per paràmetre.

- **TCP_TrobaAdrSockRem**: Donat un socket connectat, retorna l'adreça del socket remot amb qui està connectat.
- **TCP_TrobaAdrSockLoc**: Donat un socket connectat, retorna l'adreça local del propi socket.
- **T_ObteTextRes**: Donat un codi d'error, retorna el missatge d'error que descriu el codi.

5.3 Les interfícies de les capes d'Interxarxa IP i xarxa *Ethernet*

La interfície de la capa d'Interxarxa IP no la coneixem exactament, es troba dins del S.O., però podem suposar que seria de l'estil IP_Envia() i IP_Rep().

La interfície de la capa de xarxa *Ethernet* tampoc no la coneixem exactament, es troba dins del S.O., però podem suposar que seria de l'estil Eth_Envia() i Eth_Rep().

6 Els protocols de les capes i els *sockets* TCP (estudi amb *Wireshark* i "ss")

El protocol d'una capa és el conjunt de regles que governen la comunicació, regles que defineixen el diàleg de missatges entre les parts (o entitats) d'una capa d'una manera clara i precisa. La definició d'un protocol comprèn 3 aspectes: el nom i significat dels missatges, el seu format i la seva seqüència temporal.

6.1 El protocol de la capa d'aplicació de UEB (estudi amb *Wireshark*)

El nom i significat dels missatges UEB és el següent:

- **OBT**: És la petició per OBTenir un fitxer del servidor.
- **COR**: És la resposta del servidor: petició CORrecta. Retorna el fitxer demanat pel client.
- **ERR**: És la resposta del servidor: petició ERRònia. Retorna el missatge d'error adequat (no s'ha trobat el fitxer, falta la / a l'inici, etc.).

El format dels missatges UEB es mostra a la Fig. 3.

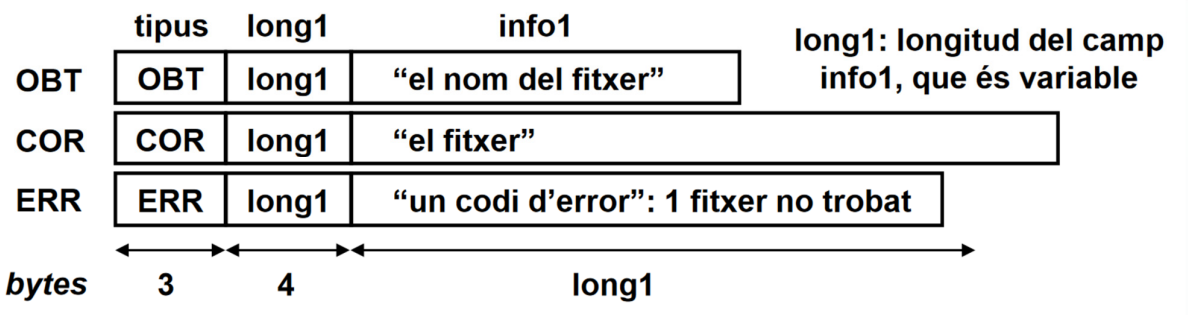

Figura 3: Format dels missatges del protocol UEB.

La seqüència temporal dels missatges UEB corresponent a l'estudi del cas 1 (el C demana /imatges/px.png, que existeix al S) fet amb *Wireshark* (veure les captures adjuntes) es mostra a la Fig. 4.

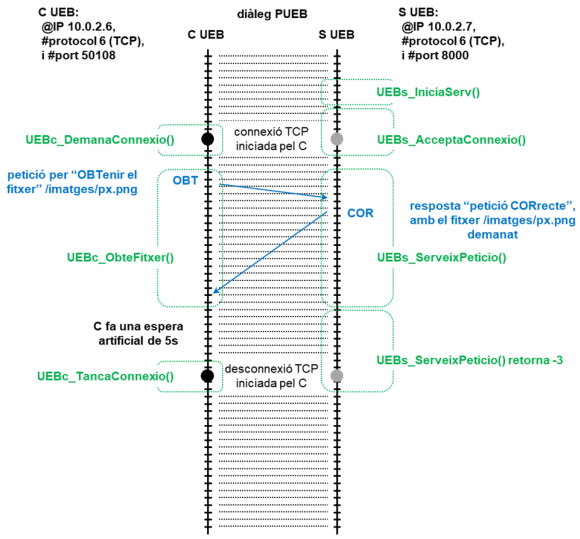

Figura 4: Seqüència temporal del protocol UEB en el cas 1.

La seqüència temporal dels missatges UEB corresponent a l'estudi del cas 2 (el C demana /noexisteix.html, que no existeix al S) fet amb *Wireshark* (veure les captures adjuntes) es mostra a la Fig. 5.

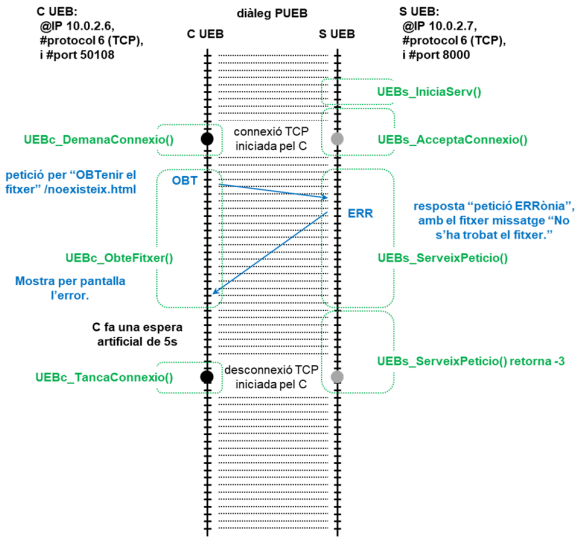

Figura 5: Seqüència temporal del protocol UEB en el cas 2.

6.2 El protocol de la capa de transport TCP (estudi amb *Wireshark*)

El nom i significat dels missatges TCP és el següent:

- Els paquets "SYN", "SYN+ACK" i "ACK" són els missatges de petició d'inici de connexió i resposta (≈PIC i RP)
- Els paquets "FIN+ACK", "FIN+ACK" i "ACK" són els missatges de petició de fi de connexió i resposta (≈PFC i RP)
- Els paquets amb informació són ACK amb informació, mentre que els paquets de confirmació (RP) són ACK **sense** informació.

El format dels missatges TCP [3] es mostra a la Fig. 6 (el tipus de paquet s'indica amb els bits o *flags*, principalment SYN, FIN i ACK).

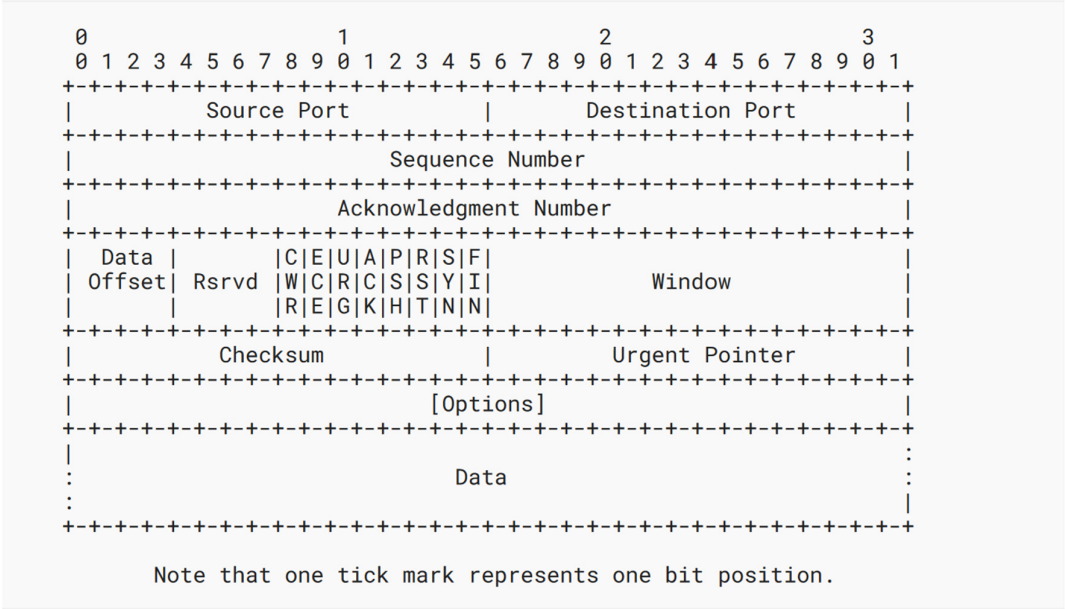

Figura 6: Format dels missatges del protocol TCP (imatge extreta de [3]).

La seqüència temporal dels missatges TCP corresponent a l'estudi del cas 1 (el C demana /imatges/px.png, que existeix al S) amb *Wireshark* (veure les captures adjuntes) es mostra a la Fig. 7.

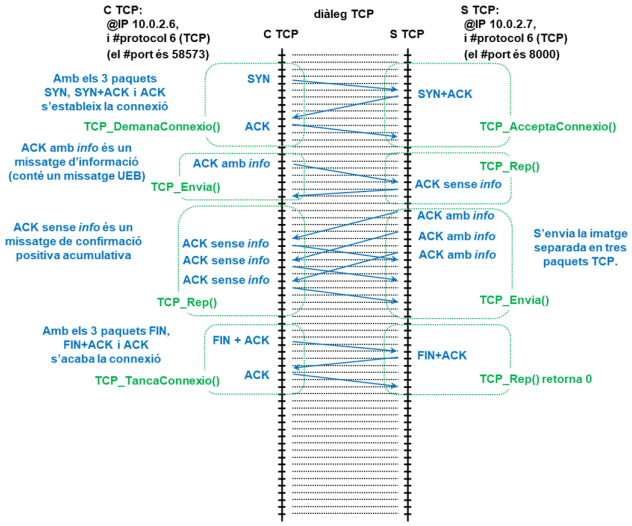

Figura 7: Seqüència temporal del protocol TCP en el cas 1.

La seqüència temporal dels missatges TCP corresponent a l'estudi del cas 2 (el C demana /noexisteix.html, que no existeix al S) amb *Wireshark* (veure les captures adjuntes) es mostra a la Fig. 8.

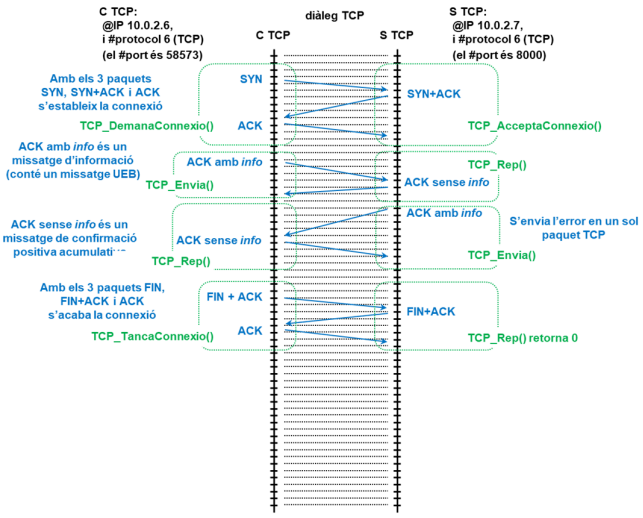

Figura 8: Seqüència temporal del protocol TCP en el cas 2.

6.3 Els protocols de les capes d'Interxarxa IP i xarxa *Ethernet*

En el protocol IP només hi ha un tipus de paquet, el paquet IP. El format del paquet IP versió 4 (IPv4) [4] es mostra a la Fig. 9.

Version $_{4\ bits}$	IHL $_{4}$	DSCP $_{6}$	ECN $_{2}$	Total Length $_{16}$
Identification $_{16}$			Flags $_{3}$	Fragment Offset $_{13}$
Time To Live $_{8}$	Protocol $_{8}$		Header Checksum $_{16}$	
Source Address $_{32}$				
Destination Address $_{32}$				
Options $_{m\ ((IHL - 5) \times 32)}$				
Data $_{n\ (total\ length \times 8)}$				

Figura 9: Format del paquet IPv4.

La seqüència temporal dels paquets IP no cal dibuixar-la: un paquet IP porta a dins un paquet TCP, és 1 a 1, i la seqüència, doncs, seria "igual" a la de TCP (però posant-hi IP).

En el protocol *Ethernet* [5] només hi ha un tipus de paquet, el paquet *Ethernet*. El format del paquet MAC d'*Ethernet* es mostra a la Fig. 10.

6 *bytes*	6	2	46-1500	4
Destination Address	Source Address	type / length	data	CRC

Figura 10: Format del paquet MAC d'*Ethernet*.

La seqüència temporal dels paquets *Ethernet* no cal dibuixar-la: un paquet *Ethernet* porta a dins un paquet IP, és 1 a 1, i la seqüència, doncs, seria "igual" a la d'IP (però posant-hi *Ethernet*).

6.4 L'encapsulació de protocols (estudi amb *Wireshark*)

De l'estudi fet amb *Wireshark* del cas 1 (el C demana /imatges/px.png, que existeix al S; veure la captura adjunta) hem escollit el missatge PUEB COR que porta la imatge px.png, i hem estudiat l'encapsulació dels seus protocols. El missatge COR és gran i TCP el fragmenta, de manera que hi ha diversos paquets TCP

d'informació. La llista de missatges de protocols i la seva longitud (escrita com "c+i", on "c" és la longitud de la capçalera i "i" la de la informació) és la següent:

- **PUEB COR:** 7+ 4108 (= 4115) *bytes*
- **TCP ACK amb *info*:** n'hi ha 3 (fragments 1448+1448+1219 = 4115 *bytes*), 32+1448 (= 1480), 32+1448 (= 1480), 32+1219(= 1251) *bytes*
- **IP:** n'hi ha 3, 20+1480 (= 1500), 20+1480 (= 1500), 20+1251 (= 1271) *bytes*
- **MAC d'*Ethernet* *:** n'hi ha 3, 14+4+1500 (=1518), 14+4+1500 (=1518), 1271+4+14 (=1289) *bytes*

(* Com que *Wireshark* no mostra el camp CRC de 4 *bytes* del paquet MAC d'*Ethernet*, cal afegir-lo)

6.5 Els *sockets* TCP de l'aplicació (estudi amb "ss")

En el cas d'estudi 1, els *sockets* TCP de l'aplicació en diferents instants de l'execució van ser els següents:

1. **Amb el C i S apagats:** Cap socket TCP creat a part dels del propi Sistema Operatiu.
2. **Un cop s'ha engegat el C i el S però encara no s'han connectat:** el C, res; el S, @IP *:TCP #port 8000 (listen).
3. **Durant una transferència:** el C, @IP 10.0.2.7 : TCP #port 60618 (established); el S, @IP 10.0.2.6 : TCP #port 8000 (established) i @IP * : TCP #port 8000 (listen).
4. **Un cop acabada la transferència i la connexió:** el C, @IP 10.0.2.7 : TCP #port 60618 (time-wait); el S @IP * : TCP #port 8000 (listen).
5. **Un cop s'acaba l'execució del C i del S:** Cap socket TCP creat a part dels del propi Sistema Operatiu.

A la Fig. 11 es mostra la captura de pantalla de l'execució de la comanda "ss" a l'instant 3 a les dues estacions.

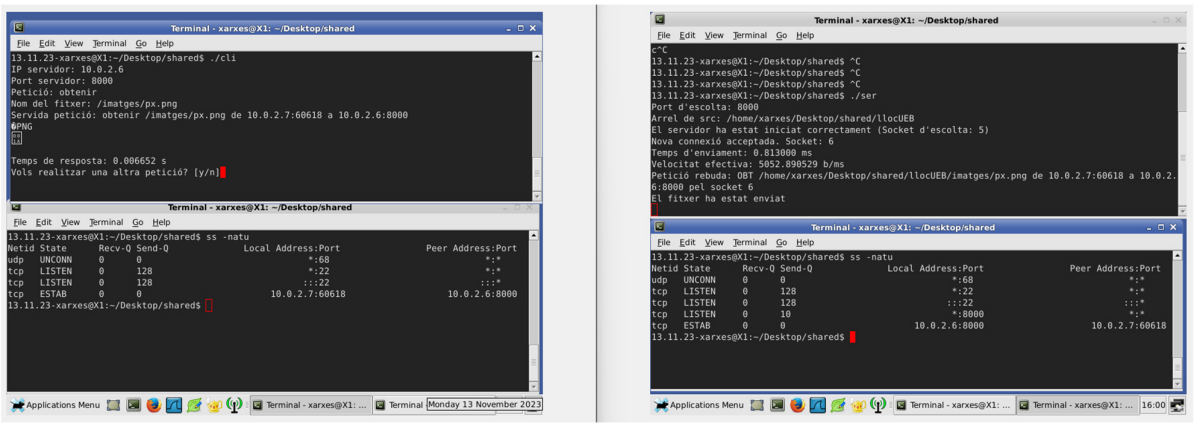

Figura 11: Execució de la comanda ss" a l'instant 3 a les dues estacions.

7 Les millores

En aquesta secció es descriuen les millores que s'han fet.

7.1 Configuració de la carpeta "arrel del lloc UEB" en el S

L'objectiu d'aquesta millora és afegir un paràmetre més al fitxer de configuració del servidor que determini el path de l'arrel on aquest anirà a buscar els fitxers que els clients li demanin.

Per solucionar-ho, hem creat la funció "int read_config(char* path, int* port)" que s'encarrega de llegir el fitxer "p2-serUEB.c" on la primera línia ha de contenir la paraula clau #portTCP seguida del port i la segona línia ha de contenir la paraula clau #arrel seguida del "path" on s'aniran a buscar els fitxers.

```c
int read_config(char* path, int* port) {

    FILE *fp;
    char linia[50];

    fp = fopen("p2-serUEB.cfg","r");
    if (fgets(linia,sizeof(linia),fp) == NULL) {
        printf("No s'ha pogut trobar el port a obrir");
        return -1;
    }
    *port = atoi(linia+9);

    if (fgets(linia,sizeof(linia),fp) == NULL) {
        printf("No s'ha pogut trobar l'arrel.");
        return -1;
    }
    strcpy(path, linia+7);
}
```

Figura 12: Codi de la implementació de la funció read_config()

A partir d'aquí, el "path" es passa per paràmetre a la funció "UEBs_ServeixPeticio()" que va a buscar, si cal, el fitxer demanat a la ruta rebuda.

Hem comprovat el seu funcionament afegint la clau #arrel seguida una nova ruta on es guarden els fitxers, i el servidor els ha trobat i enviat igualment al client.

7.2 Fitxer de "log" en el S

L'objectiu d'aquesta millora és que el servidor creï un arxiu anomenat "serUEB.log" on guardi un "log" del que ha anat fent. Com aquestes explicacions també s'imprimeixen per pantalla, s'ha decidit crear la funció "void escriure(const char* str)" que escriu str per pantalla i a "serUEB.log"

```c
void escriure(const char* str) {
    write(fd, str, strlen(str));
    printf("%s", str);
}
```

Figura 13: Codi de la implementació de la funció escriure()

Nota: el fitxer es sobreescriu cada cop que s'executa el servidor, doncs s'assumeix que aquest estarà sempre corrent.

La variable global "fd" és un descriptor de fitxer de "serUEB.log" que pren valor a l'inici del main amb la següent funció:

```c
fd = open("serUEB.log", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
```

Figura 14: Creació i assignació del descriptor de fitxer de "serUEB.log"

Per provar el funcionament de la millora s'ha realitzat una execució on s'han guardat tots els successos en el fitxer anomenat "serUEB.log" situat a la mateixa carpeta on hi ha l'executable.

Figura 15: "serUEB.log" després d'una execució

7.3 El temps de resposta, el temps d'enviament i la velocitat efectiva

L'objectiu d'aquesta millora és que el client vegi el temps de resposta que percep el client per pantalla i que el servidor mostri per pantalla (i pel log) el temps d'enviament del fitxer i la velocitat efectiva de la transmissió del fitxer que percep.

```
struct timeval start, end;
gettimeofday(&start, 0);
obtingutCorrectament = UEBc_ObteFitxer(socket_c,
if (obtingutCorrectament != 0) {
    printf("%s", text_res);
} else {
    gettimeofday(&end, 0);
```

Figura 16: Codi on es mesura el temps d'"UEBc_ObteFitxer"

Pel que fa al client, es mesura el temps que triga a rebre la resposta de "UEBc_ObteFitxer" i s'imprimeix per pantalla. Si hi ha hagut un error, no s'imprimeix res.

```
struct timeval start, end;
gettimeofday(&start, 0);
if (TCP_Envia(SckCon, send_buf, long1 + 7) < 0) {
    return -1;
}
gettimeofday(&end, 0);

double te = (end.tv_sec - start.tv_sec)*1000 + (end.tv_usec - start.tv_usec)*1e-3;
printf("Temps d'enviament: %f ms\n", te);
printf("Velocitat efectiva: %f b/ms\n", long1 / te);
```

Figura 17: Codi on es mesura el temps de "TCP_Envia" i s'imprimeix per pantalla

Pel que fa al servidor, es mesura el temps de la funció "TCT_Envia" i s'imprimeixen per pantalla les estadístiques prèviament esmentades.

Amb una simple execució del codi hem pogut veure que aquesta millora senzilla funciona perfectament.

Nota: S'ha afegit el codi d'aquesta millora dins la funció *ConstruieixIEnviaMissatge()*. Som conscients que el *printf()* seria recomanable **no** fer-lo dins aquesta funció, però per a simplificar el codi i no modificar els *headers* proporcionats, s'ha decidit optar per a aquesta implementació.

8 El servidor iteratiu

El S serveix peticions una darrera l'altra, no simultàniament, és a dir, només pot mantenir una connexió TCP alhora: és un S "iteratiu" (no és un S concurrent). Per estudiar-ho s'ha fet la següent prova de funcionament:

- En el C, les connexions TCP s'ha allargat "artificialment" 5 segons.
- En una estació E1 s'ha engegat un S; en una estació E2 s'han engegat 3 C, C1, C2 i C3, en diferents terminals; llavors els 3 C han fet peticions al S (p.e., obtenir fitxers que existeixin al S) més o menys alhora.

Nota: per no tocar el codi font, s'ha deixat el temps en 5 segons, però els resultats permeten treure les mateixes conclusions que si en fossin 20 (com a l'enunciat original).

Llavors:

- C1: Sí, amb un temps d'espera petit (0,004981 s)
- C2: Sí, amb un temps d'espera gran (4,300279 s)
- C3: Sí, amb un temps d'espera gran (9,357181 s)

A la Fig. 18 es mostra una captura de pantalla de la prova de funcionament del S iteratiu.

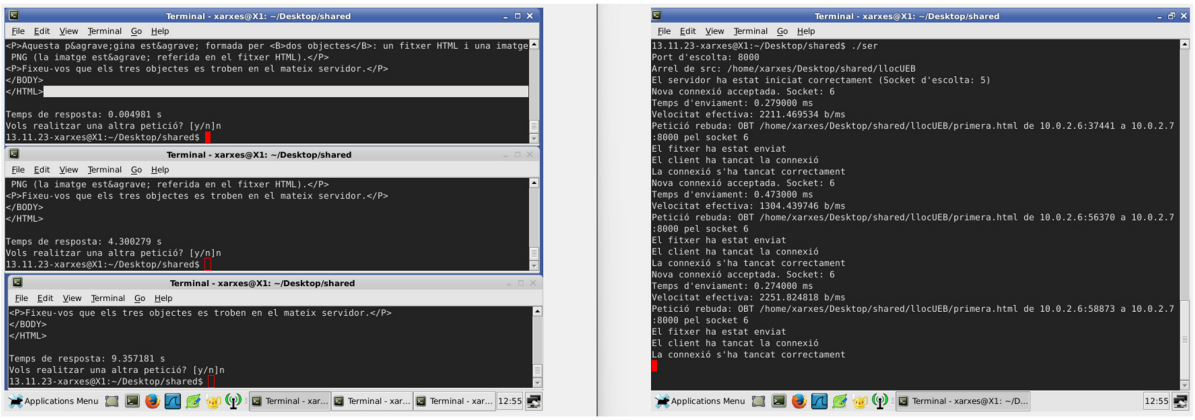

Figura 18: Prova de funcionament del S iteratiu.

9 Problemes i suggeriments

Ens hem trobat amb diversos petits problemes característics del desenvolupament de qualsevol codi, els quals hem aconseguit resoldre sense gaires dificultats, depurant codi.

En un primer moment vam escriure un codi una mica desordenat, i sense retornar o controlar certs errors, que un cop vam solucionar, vam descobrir certs errors que potencialment podien suposar un problema.

El contratemps més gran ha estat al final, on arrossegàvem un error de la capa TCP, present també a la pràctica anterior, on no vam detectar aquest error. Consisteix en que no s'inicialitzava la variable local "addrlon".

10 Treball en parella i dedicació

Hem fet el plantejament inicial durant les dues primeres sessions, abans de la tercera hem acabat els requisits mínims per poder preguntar dubtes, i a la tercera sessió hem implementat tres de les quatre millores senzilles.

Finalment, abans de la quarta sessió, hem solucionat l'error mencionat en l'apartat anterior i hem escrit aquest informe.

Hem destinat al desenvolupament d'aquesta pràctica les 4 hores de classe i unes 4 hores més cada un de nosaltres, treballant simultàniament per trucada.

Bibliografia

[1] Lluís Fàbrega, *Pràctica 2: L'aplicació UEB amb sockets TCP/IP − Part I*, curs 2023-24, UdG, 2023.

[2] Lluís Fàbrega, *La interfície de sockets de C a UNIX*, curs 2023-24, UdG, 2023.

[3] W. Eddy (Ed.), *RFC 9293 - Transmission Control Protocol (TCP)*, 2022. Disponible a: <https://www.rfc-editor.org/info/rfc9293>.

[4] J. Postel, *RFC 791 - Internet Protocol*, 1981. Disponible a: <https://www.rfc-editor.org/info/rfc791>.

[5] IEEE Standards Association, *IEEE Standard for Ethernet, IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, 2022. Disponible a: <https://doi.org/10.1109/IEEESTD.2022.9844436>.