CYPHER SYSTEM

Lluis Farré Castán - Joan Esteban Santaeularia

Gener 2019

Índex

1 Implementacio

La pràctica l'hem dividit en 6 mòduls clarament diferenciats. El mòdul commands tracta tot el relacionat amb les comandes, verifica que siguin correctes i llença el mètode del mòdul connections corresponent a l'ordre de l'usuari.
El mòdul connections conté tots els mètodes i funcions per el tractament de dades enviades i rebudes. També s'encarrega de fer de client i servidor, conté tots els sockets.
El módul llistaPDI implementa una llista PDI amb algunes funciones extra per buscar segons alguns camps de Connexió, a més implementa un semàfor per evitar que es trepitgin dos processos que intentant accedir a la mateixa llista.
El mòdul protocol conté els mètodes i funcions per aplicar el protocol d'enviament i rebuda especificat per aquesta pràctica. Conté dos mètodes molt importants, un per llegir un paquet, i l'altre per enviar un paquet.
El mòdul utilities.c conté totes aquelles funcions externes al motiu de existencia de cada mòdul. Conté funcions molt generalitzades que fan petites tasques i que no depenen d'altres llibreries privades. El main conté poques funcionalitats, tancament del programa, execució principal de inits i la lectura del fitxer de configuració. La lectura del fitxer la fem en el main perquè és una de les primeres tasques que fa el programa i també una de les últimes que hi ha de tancar quan surt.

1.1 Processos

1.1.1 Threads

A part del procés principal, tenim dos tipus de threads diferents. El thread *listener()* és l'encarregat de gestionar les connexions entrants i anar creant threads de tipus *acceptConnection()*, aquest nou thread és l'encarregat de mantenir la connexió com a servidor amb el client acceptat pel thread *listener()*. Aquest thread és tencara quan el client enviï una trama de exit, o quan nosaltres ens

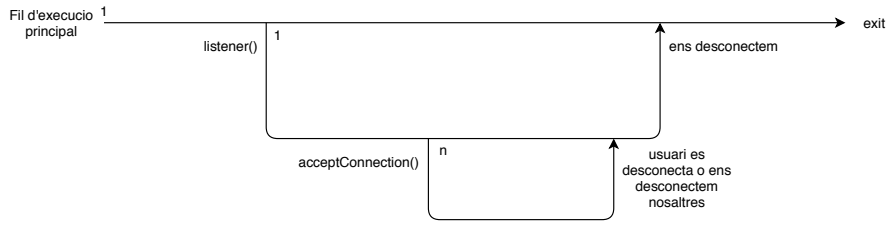

Figura 1:

desconnectem. Abans de tancar-se eliminarem la connexió de la llista de connexions servidor *lServer* i alliberarem tota la memòria i recursos associats aquesta connexió, finalment és tancarà.

1.1.2 Forks

El nostre programa remès fa ús de forks en dues ocasions.

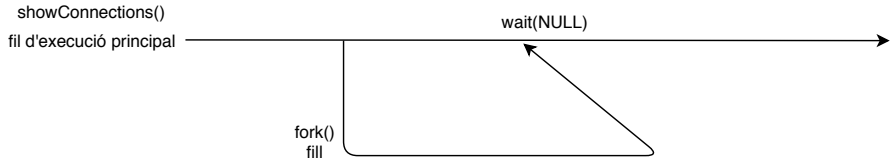

Figura 2:

Quan executem la comanda *SHOW CONNECTIONS* el programa esta executant el script extern *show_connections_v2.sh*, per poder executar alguna cosa extern al programa necessitem usar un fork. El procés que ha fet el fork, és quedarà a l'espera que el procés "fill" acabi l'execució, quan aquest acabi el procés "pare" continuarà.

El segon fork l'usem per poder executar la comanda md5sum al fitxer de so i així confirmar que s'ha descarregat correctament. Aquest fork es fa en dues ocasions, quan el servidor ha d'enviar el md5sum al client, i quan el client fa el md5sum del arxiu rebut.

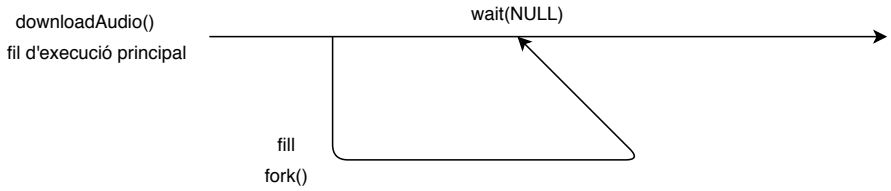

Figura 3:

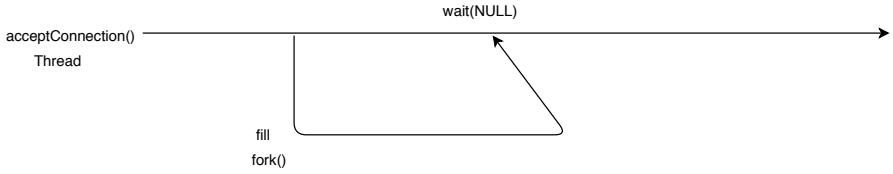

Figura 4:

Pel tancament correcte dels forks sense tenir problemes de processos zombi, usem un *wait(NULL)* que fa que el "pare" s'esperi a la finalització de l'execució del "fill".

1.2 Estructures de dades

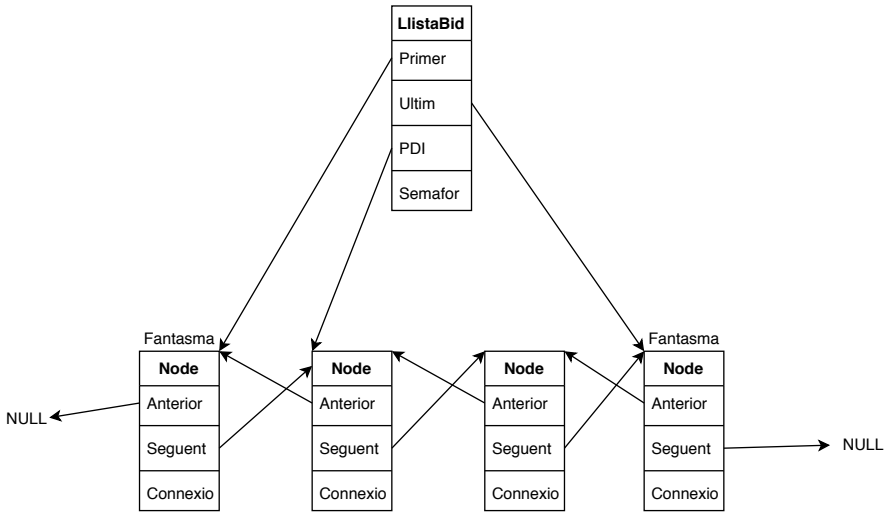

Figura 5:

Hem usat una llista PDI per guardar cada una de les connexions, tant del servidor com del client. Hem triat una llista PDI perquè ens permet moure'ns lliurement per la llista. Per al client la llista guarda tots els servidors als quals està connectat, així quan executem un *SHOW CONNECTIONS* ens mostrarà el port disponible i el nom de l'usuari en cas d'estar connectats a ell. Per al servidor usem llista PDI per guardar els PID dels threads servidor que mantenen la connexió amb cada un dels usuaris.

Aquesta llista PDI guarda l'estructura Connexió.

```
typedef struct{
  char *name
  char *folder
  char *ip
  int port
  char *server
  int port_begin
  int port_end
}Conexio
```

Hem modificat algunes coses de la llista per adaptar-la a les nostres necessitats. Li hem afegit un semàfor a la llista perquè dos processos diferents puguin accedir a ella sense trepitjar-se. El semàfor està implementat dins de la llista. Per usar-lo hem creat les funcions *LLISTABID_reservar(LlistaBid l)* i *LLISTABID_lliberar(LlistaBid l)*. La primera reserva la llista fins que la segona la alliberi.

1.3 Recursos del sistema

1.3.1 Semáfors

Tenim un semàfor per la llista PDI com hem comentat abans. Aquest semàfor és diferent per a cada llista creada. També tenim un semàfor per mostrar informació per pantalla. Aquest el tenim implementat a utilities.c. Amb aquest semàfor

evitem que diferents processos és trepitgin a l'hora de mostrar informació per pantalla, també evita que si per exemple rebem un missatge en mig d'un *SHOW CONNECTIONS* ens mostrin aquest missatge enredat amb la informació de la comanda, així que primer ens mostrarà el relacionat amb la comanda i després mostrarà el missatge entrant. Tenim un altre semàfor per evitar que dos usuaris no descarreguin el matexi arxiu de forma paral·la del mateix servidor.

1.3.2 Pipes

Les úniques pipes que tenim son les necessàries per comunicar el que retornen els execl() executats pels forks de ferMD5Sum() i showConnections(). Fem un dup2(), amb aquest redirigim el que surt per pantalla a la pipe que hem creat, i llavors llegim de la pipe la informació.

1.3.3 Signals

El nostre pograma resmés té un signal, SIGINT. Quan es rep aquest senyal s'inicia el protocol de tancament de tot el programa, el mateix que si l'usuari introduís la comanda *EXIT*

1.4 Opcionals

No hem implementat res fora de el demanat en l'enunciat de la pràctica.

2 Problemes Observats

Hem tingut varis problemes que hem anat solucionant a poc a poc. Vam tindre un problema a l'hora d'executar la comanda *SHOW CONNECTIONS*.

Quant teníem tres usuaris tots connectats entre ells i fèiem varis *SHOW CONNECTIONS* ens sortia el següent missatge d'error, fig.6, i el programa sencer començava a comportar-se de manera erràtica, vam intuir que aquest comportament era degut a recursos insuficients de la màquina. Utilitzant la comanda de Linux *top -H -u login*, fig.7, ens mostrava els processos actius del nostre usuari.

Figura 6:

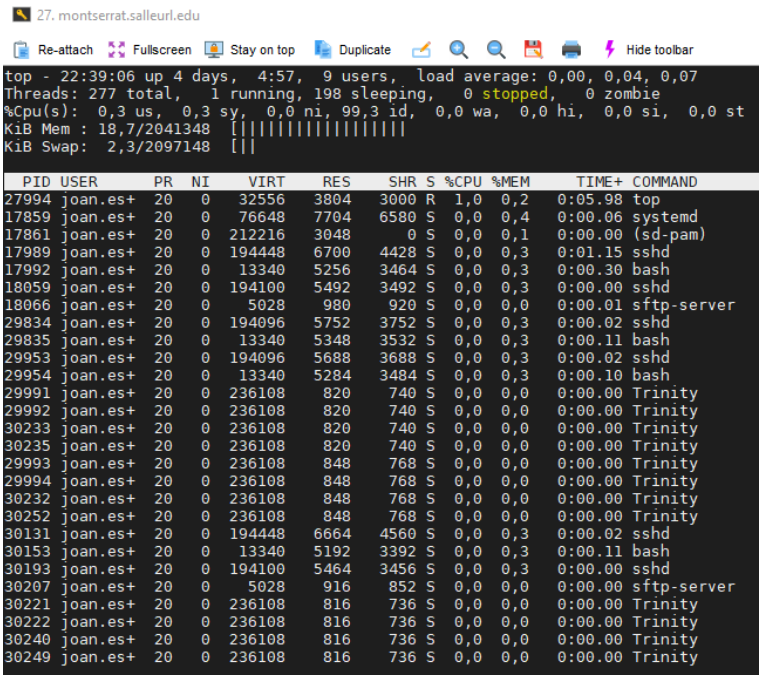

Figura 7:

Aquí vam descobrir dues coses. Que en fer els forks del *SHOW CONNECTIONS* els processos es quedaven en estat "zombi" i que no s'arribaven a tancar del tot. Després ens vam donar conte que aquests no és tancaven correctament perquè faltava un *wait(NULL)* al fork "pare". En ficar aquest *wait(NULL)*

vam evitar que els forks es quedessin en estat "zombi" però seguíem obtenint el missatge *Resource temporarily unavailable*. Intuïm que era degut a la quantitat de recursos que estàvem usant. Usant la comanda *ulimit -a* vam descobrir que estem limitats a un màxim de 30 processos i que estàvem arribant a aquest màxim perquè el programa que usàvem per connectar-nos al servidor Montserrat usava 3 processos per cada connexió: sftp, bash i sshd. Això vol dir que per cada usuari ocupàvem 3 processos. I amb varis usuaris tots connectats entre si arribàvem al màxim, podem veure tots els processos a la fig.8.

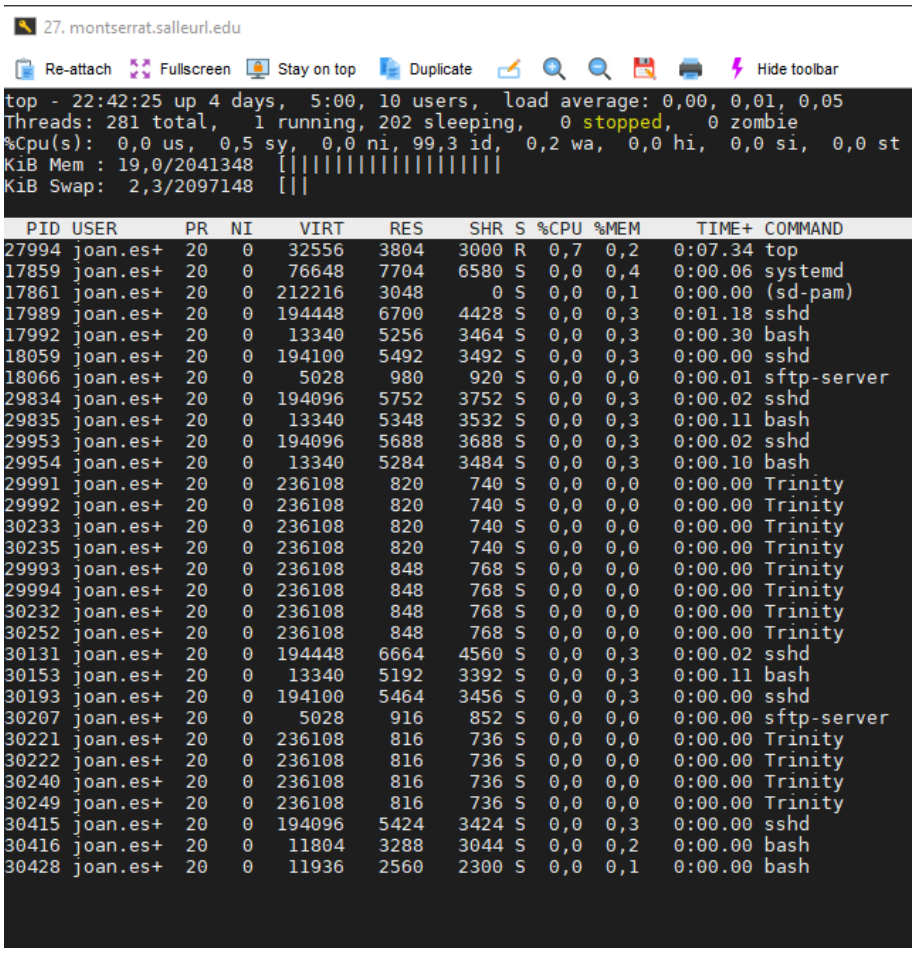

Figura 8:

També vam tindre un problema a la funció readLine(), aquí llegim caràcter a caràcter en el file descriptor indicat fins al delimitador que li passem. Dins la funció usem realloc() per anar demanant memòria per a cada nou caràcter que llegim. En un principi aquesta funció no retornava res, ja que el punter el passàvem per referència i aquest ja el teníem apuntat des de la super funció. Però el realloc quant en un bloc no té suficient memòria consecutiva, mou a un altra zona on si té tota la memòria demanada consecutiva, llavors el punter de la super funció perdia la referència, això ho vam solucionar retornant el punter.

Quan mostràvem el nom de l'usuari per línia d'ordres, ex: $GeneralKenobi:, el cursor se'ns ficava al principi de la línia, i tot el que l'usuari escrivia es ficava sobre aquest text. Això era degut al fet que l'editor de text Atom, quan edita un fitxer té dues maneres d'aplicar els salts de línia, la manera UNIX (LF) i la manera Windows (CLRF). La manera Windows, per fer un salt de línia, fa un \r \n , en canvi en Linux res més es fa un \n. El fitxer de configuració estava en mode Windows per tant afegia un \r \n, per això quan obríem i llegiem el fitxer el cursor se'ns movia al principi.
En descarregar audios ens vam trobar amb el problema que algunes vegades la descarrega fallava, ja que rebíem alguns paquets corruptes, la tassa d'èxit de descàrrega la vam millorar baixant la dimensio dels paquets i usant potències de dos. Curiosament les potències de dos feien millor la transmissió de paquets. No sabem perquè, però l'arxiu suministrat *semaphore_v2.h* no ens funcionava correctament, no ens deixava compilar el projecte si el volíem importar i fer servir en més d'un mòdul. Per això vam dividir aquest .h en un .c i un .h. En el fitxer c tenim la implementació de les funcions i en el fitxer h tenim la declaració. Això va solucionar el problema.

3 Estimació temporal

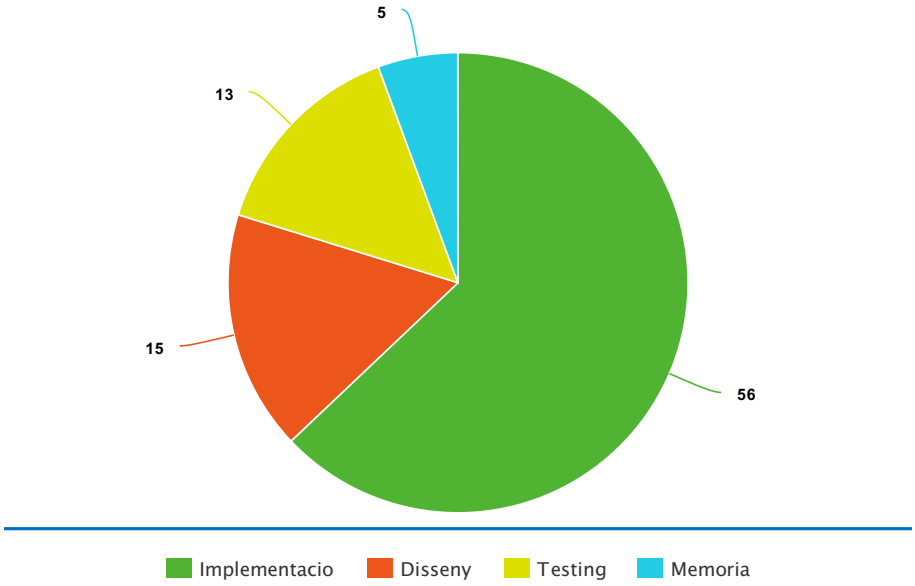

Figura 9:

4 Conclusions i millores

Aquesta pràctica ha sigut molt interessant i educativa. Ara som més amics de C. Implementant aquesta pràctica hem acabat d'entendre força el funcionament i la implementació de el estudiat a les sessions de laboratori. Després d'aquesta pràctica sentim que tenim un més profund coneixement de C i el funcionament bàsic d'un Sistema Operatiu i com usar els seus recursos.

Creiem que aquesta pràctica té marge per a moltes millores. Una que ens hagués agradat implementar de tindre més temps hauria sigut una millora en el protocol d'enviament d'audios. Quan ens vam trobar amb el problema que a vegades els arxius no s'envien correctament i que el checksum no concorda, vam pensar que podríem dividir la descarrega en "chunks" de varis paquets, una vegada enviats i rebuts tots els paquets d'un chunk, enviariem el checksum d'aquell "chunk", i en cas de no concordar el client enviaria una petició de reenviament d'aquell "chunk" al servidor. D'aquesta manera no es perdria tota la descarrega en cas que alguna cosa anés malament, res més es perdria un petit tros que automàticament es reenviaria si aquest és incorrecte. En el nostre cas que enviem arxius força petits potser no és necessari, però si l'arxiu és més gran hi ha més probabilitats que alguna cosa surti malament.

5 Bibliografía

Referències

[1] *c - Can a socket be closed from another thread when a send / recv on the same socket is going on?* URL: `https://stackoverflow.com/questions/3589723/can-a-socket-be-closed-from-another-thread-when-a-send-recv-on-the-same-socket` (cons. 02-02-2020).

[2] *c - Checking for EOF when using read() function.* URL: https://stackoverflow.com/questions/24941819/checking-for-eof-when-using-read-function (cons. 02-02-2020).

[3] *c - how can i combine and store 2 char variables in an integer variable?* URL: https://stackoverflow.com/questions/40562443/how-can-i-combine-and-store-2-char-variables-in-an-integer-variable (cons. 02-02-2020).

[4] *c - NUL undeclared- first use in this function.* URL: https://stackoverflow.com/questions/10546625/nul-undeclared-first-use-in-this-function (cons. 02-02-2020).

[5] *c - size of string returned by malloc is not correct.* URL: https://stackoverflow.com/questions/27760205/size-of-string-returned-by-malloc-is-not-correct (cons. 02-02-2020).

[6] *c - strlen() on non-null-terminated char string?* URL: https://stackoverflow.com/questions/631516/strlen-on-non-null-terminated-char-string (cons. 02-02-2020).

[7] *c - What does mode_t 0644 mean?* URL: https://stackoverflow.com/questions/18415904/what-does-mode-t-0644-mean/18415935 (cons. 02-02-2020).

[8] *C library function - strcat() - Tutorialspoint.* URL: https://www.tutorialspoint.com/c_standard_library/c_function_strcat.htm (cons. 02-02-2020).

[9] *c++ - Difference between include in .h file and .c file.* URL: https://stackoverflow.com/questions/9944707/difference-between-include-in-h-file-and-c-file (cons. 02-02-2020).

[10] Ciro S. Costa. *Implementing a TCP server in C.* en. Febr. de 2018. URL: https://ops.tips/blog/a-tcp-server-in-c (cons. 02-02-2020).

[11] *eclipse - C programming multiple definition error*. URL: https://stackoverflow.com/questions/9068058/c-programming-multiple-definition-error (cons. 02-02-2020).

[12] *How to prevent multiple definitions in C?* URL: https://stackoverflow.com/questions/672734/how-to-prevent-multiple-definitions-in-c (cons. 02-02-2020).

[13] *How to remove carriage returns in a text editor?* URL: https://askubuntu.com/questions/958543/how-to-remove-carriage-returns-in-a-text-editor (cons. 02-02-2020).

[14] Salvador Pont Jordi. *Programació en UNIX*. Enginyeria i Arquitectura La Salle, 2014.

[15] *linux - Manually closing a port from commandline*. URL: https://superuser.com/questions/127863/manually-closing-a-port-from-commandline (cons. 02-02-2020).

[16] Manish. *C Program to List Files in Directory*. Febr. de 2013. URL: https://www.sanfoundry.com/c-program-list-files-directory/ (cons. 02-02-2020).

[17] *open (C System Call) - Code Wiki*. URL: http://codewiki.wikidot.com/c:system-calls:open (cons. 02-02-2020).

[18] *Opening and Closing Files (The GNU C Library)*. URL: https://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html (cons. 02-02-2020).

[19] *printf - How to print an unsigned char in C?* URL: https://stackoverflow.com/questions/15736497/how-to-print-an-unsigned-char-in-c (cons. 02-02-2020).

[20] *pthread_detach(3) - Linux manual page*. URL: http://man7.org/linux/man-pages/man3/pthread_detach.3.html (cons. 02-02-2020).

[21] *Reading/Closing Directory (The GNU C Library)*. URL: https://www.gnu.org/software/libc/manual/html_node/Reading_002fClosing-Directory.html (cons. 02-02-2020).

[22] *Show All Running Processes in Linux using ps/htop commands*. en-US. Oct. de 2006. URL: https://www.cyberciti.biz/faq/show-all-running-processes-in-linux/ (cons. 02-02-2020).

[23] *Using open() to create a file in C*. URL: https://stackoverflow.com/questions/28466715/using-open-to-create-a-file-in-c (cons. 02-02-2020).