Sistemes Operatius

Curs 2020-2021

Miquel Zamora Hidalgo (miquel.zamora@students.salle.url.edu)

Guillem Serra Cazorla (guillem.serra@students.salle.url.edu)

Index

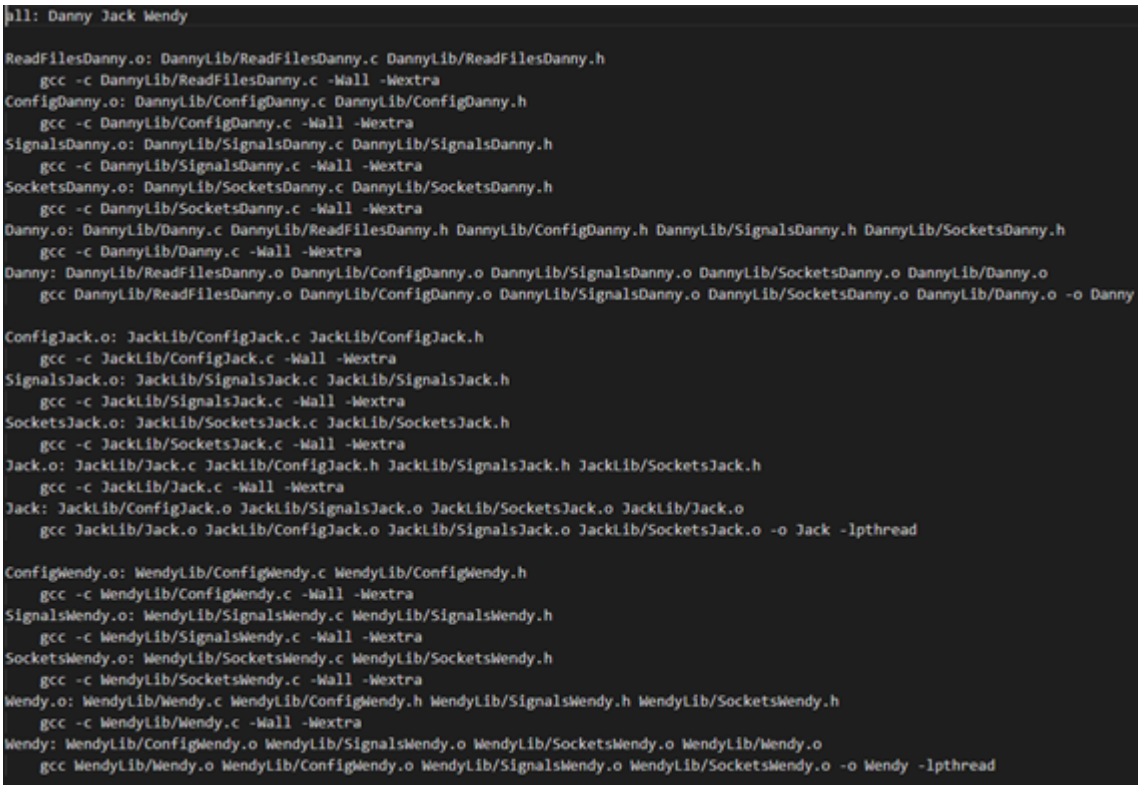
1. Disseny

Estructuració de l'arquitectura del codi

Al principi, teníem una estructuració de codi molt simple, ja que a les primeres fases el sistema era més petit i ens havíem centrat molt més en el plantejament de l'arquitectura del sistema i les tecnologies que utilitzaríem, en comptes de l'arquitectura del codi.

Però a mesura que avançava la pràctica, tenir només un o dos fitxers .c per cada procés, feia que resultés molt complicat trobar les línies de codi de cada fase, i per tant seguir avançant correctament amb la pràctica. Així que vam decidir fer un replantejament, i dedicar una bona estona a organitzar i dividir correctament el codi.

Al final hem decidit modular molt més el codi, en molts més fitxers .c i .h, ja que al compilar s'utilitzarà un Makefile, així que al final és un procés automàtic un cop creat aquest fitxer per molts mòduls que hi hagi. A continuació es pot veure una fotografia del Makefile final:



```
all: Danny Jack Wendy

ReadFilesDanny.o: DannyLib/ReadFilesDanny.c DannyLib/ReadFilesDanny.h
	gcc -c DannyLib/ReadFilesDanny.c -Wall -Wextra
ConfigDanny.o: DannyLib/ConfigDanny.c DannyLib/ConfigDanny.h
	gcc -c DannyLib/ConfigDanny.c -Wall -Wextra
SignalsDanny.o: DannyLib/SignalsDanny.c DannyLib/SignalsDanny.h
	gcc -c DannyLib/SignalsDanny.c -Wall -Wextra
SocketsDanny.o: DannyLib/SocketsDanny.c DannyLib/SocketsDanny.h
	gcc -c DannyLib/SocketsDanny.c -Wall -Wextra
Danny.o: DannyLib/Danny.c DannyLib/ReadFilesDanny.h DannyLib/ConfigDanny.h DannyLib/SignalsDanny.h DannyLib/SocketsDanny.h
	gcc -c DannyLib/Danny.c -Wall -Wextra
Danny: DannyLib/ReadFilesDanny.o DannyLib/ConfigDanny.o DannyLib/SignalsDanny.o DannyLib/SocketsDanny.o DannyLib/Danny.o
	gcc DannyLib/ReadFilesDanny.o DannyLib/ConfigDanny.o DannyLib/SignalsDanny.o DannyLib/SocketsDanny.o DannyLib/Danny.o -o Danny

ConfigJack.o: JackLib/ConfigJack.c JackLib/ConfigJack.h
	gcc -c JackLib/ConfigJack.c -Wall -Wextra
SignalsJack.o: JackLib/SignalsJack.c JackLib/SignalsJack.h
	gcc -c JackLib/SignalsJack.c -Wall -Wextra
SocketsJack.o: JackLib/SocketsJack.c JackLib/SocketsJack.h
	gcc -c JackLib/SocketsJack.c -Wall -Wextra
Jack.o: JackLib/Jack.c JackLib/ConfigJack.h JackLib/SignalsJack.h JackLib/SocketsJack.h
	gcc -c JackLib/Jack.c -Wall -Wextra
Jack: JackLib/ConfigJack.o JackLib/SignalsJack.o JackLib/SocketsJack.o JackLib/Jack.o
	gcc JackLib/Jack.o JackLib/ConfigJack.o JackLib/SignalsJack.o JackLib/SocketsJack.o -o Jack -lpthread

ConfigWendy.o: WendyLib/ConfigWendy.c WendyLib/ConfigWendy.h
	gcc -c WendyLib/ConfigWendy.c -Wall -Wextra
SignalsWendy.o: WendyLib/SignalsWendy.c WendyLib/SignalsWendy.h
	gcc -c WendyLib/SignalsWendy.c -Wall -Wextra
SocketsWendy.o: WendyLib/SocketsWendy.c WendyLib/SocketsWendy.h
	gcc -c WendyLib/SocketsWendy.c -Wall -Wextra
Wendy.o: WendyLib/Wendy.c WendyLib/ConfigWendy.h WendyLib/SignalsWendy.h WendyLib/SocketsWendy.h
	gcc -c WendyLib/Wendy.c -Wall -Wextra
Wendy: WendyLib/ConfigWendy.o WendyLib/SignalsWendy.o WendyLib/SocketsWendy.o WendyLib/Wendy.o
	gcc WendyLib/Wendy.o WendyLib/ConfigWendy.o WendyLib/SignalsWendy.o WendyLib/SocketsWendy.o -o Wendy -lpthread
```

Com es pot observar a la imatge anterior, s'ha dividit el sistema en 3 executables principals (Danny, Jack i Wendy), els quals tindran els seus mòduls específics per tal de que facin les seves funcions correctament. En general, aquests mòduls s'han dividit en tres blocs principals: Configuració del procés, Treball amb Signals, i Treball amb Sockets. A més d'alguns mòduls extres com el ReadFiles de Danny, en el que s'hi guardaran tots els processos relacionats amb la lectura de carpetes i fitxers.

Per tal de que no quedessin tots els fitxers a la mateixa carpeta principal desordenats, hem dividit els fitxers en carpetes. D'aquesta manera es guardaran tots els mòduls corresponents a DannyLib, JackLib i WendyLib, aconseguint així que a la carpeta principal només hi quedin els fitxers de configuració, executables i arxius compartits (com el netejador de memòria o el Valgrind).

D'aquesta manera també, es separen els processos per simular la portabilitat que tindrien si per exemple estiguessin en estacions reals on cada executable s'executés des d'una màquina diferent, podent així treballar cada un per separat.

Recursos utilitzats

FASE 1

En aquesta primera fase calia generar un procés Danny, el qual després d'una configuració inicial a partir del fitxer que se li passi per paràmetre, accediria a un directori del sistema on hauria de llegir tots els fitxers de text i imatges que s'hi trobés.

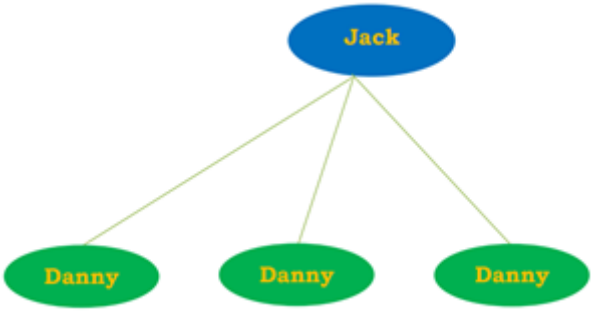
Com bé requereix l'enunciat, per tal d'obrir i llegir els fitxers s'ha utilitzat File Descriptors i les comandes open(), read(), write(), close(), en comptes de les típiques fopen(), fgets(), fclose().

També calia controlar que, en qualsevol moment, Danny es podia desconnectar si l'usuari utilitzava un CTRL+C, així que mitjançant Signals s'ha reescrit el comportament del procés quan se li envia una senyal SIGINT per deixar el sistema el més estable possible.

FASE 2

A la segona fase s'havia de millorar i afegir funcionalitats al procés Danny, a part de crear un procés anomenat Jack.

Jack tindrà la funció d'un servidor, al qual s'hi podrà connectar Danny (el client). Com que Jack i Danny poden estar en màquines diferents, caldrà utilitzar Sockets per la connexió i comunicació. La configuració d'aquests Sockets (IP i Port), s'obtindrà a través dels fitxers de configuració corresponents que es passaran per paràmetre.



Com es pot observar a l'esquema anterior, múltiples Danny's es poden connectar i enviar dades alhora a Jack, per tant Jack haurà d'actuar com a servidor dedicat. Per implementar-ho, Jack estarà constantment acceptant nous clients, i per tal de poder tractar-los cada un per separat, crearà un nou thread per cada connexió. Per tal d'enviar i llegir paquets entre servidor i clients, s'utilitzaran les comandes write() i read() juntament amb el File Descriptor del servidor i client determinat.

Com s'ha comentat a la fase anterior, Danny es pot desconnectar en qualsevol moment. Per tal d'optimitzar el sistema i anar eliminant els threads quan s'hi desconnecta el client determinat, des de Danny s'enviarà un paquet amb la següent estructura:

Origen: "DANNY"

Tipus: 'Q'

Dades: <nom de l'estació meteorològica>

Quan Jack rebi aquest paquet, el thread dedicat es tancarà automàticament. Per altra banda, si Jack es desconnecta, caldrà desconnectar tots els processos Danny, ja que no és útil llegir dades si després no es poden enviar al servidor. Per fer-ho, Jack es guardarà el PID de tots els processos Danny's que s'hi connectin, i en el moment de desconnexió, enviarà una senyal de SIGINT a cada un d'aquests per desconnectar-los.

FASE 3

Resumidament, la fase 3 consistia en generar Lloyd, el qual serà l'encarregat d'escriure les mitjanes de les estacions al fitxer Hallorann.txt. Lloyd serà creat a partir de Jack, per tant compartiran moltes de les característiques de la fase 2.

Lloyd no serà un procés independent, sinó que serà creat a partir de Jack, concretament utilitzant un Fork, on el pare serà el propi servidor dedicat Jack, i el fill actuarà com a Lloyd. Aquests, hauran de compartir les dades de les estacions, i per tant utilitzaran Memòria Compartida amb una variable pròpia on es guardaran les dades de les estacions llegides i les dades de les seves mitjanes aritmètiques.

Al treballar amb una mateixa variable (memòria compartida), podria arribar a generar-se conflictes si s'hi intenta llegir/escriure a la vegada, és per això que caldrà utilitzar semàfors per complir la sincronització.

Més concretament s'utilitzaran dos semàfors, un controlarà quan Lloyd llegirà les dades, i l'altre controlarà quan Jack escriurà les dades. El semàfor de Lloyd començarà aturat (Wait()), el qual només s'activarà quan Jack (semàfor del qual comença actiu) l'activi amb la comanda Signal() un cop acabada l'escriptura, i desactivant així el seu propi semàfor. Un cop llegides les dades, es desactivarà el propi semàfor de Lloyd i s'activarà el de Jack per escriure, creant així un bucle de sincronització fins que s'hagin escrit i llegit totes les dades.

De nou, per escriure al fitxer Halloran, s'utilitzaran els File Descriptors del fitxer.

FASE 4

Primer de tot, semblant a la fase 2, en aquesta fase també s'haurà de millorar i afegir funcionalitats al procés Danny, a part de crear un procés addicional anomenat Wendy. Wendy també actuarà com a servidor dedicat, per tant tindrà el mateix funcionament que Jack en quant a la connexió (sockets, threads, etc.).

Una característica d'aquesta fase és que s'haurà de calcular el MD5SUM de les imatges per controlar que s'hagin enviat correctament. Per poder executar la comanda de terminal md5sum, s'haurà d'utilitzar l'eina excev, però la qual atura el procés actual per fer-ho, aturant així els processos de Wendy i Danny. Per aquesta raó, caldrà fer un Fork específic per executar aquesta comanda, evitant així la desconnexió dels processos principals. La manera més senzilla de comunicar els Forks (pare i fill), és mitjançant Pipes, com sempre un d'escriptura i un de lectura.

Estructures de dades

Packet

La unitat de informació bàsica que s'utilitza en tot el sistema es la de "Packet". Aquest tipus propi està format per 3 atributs. Destacar que aquest paquet te un cost de espai de 115 bytes.
El origen es la cadena de caràcters de longitud màxima de 14 caràcters on es guarda d'on prové aquest paquet que s'ha rebut.
El tipus es un caràcter únic que depenent de l'aplicació pot indicar que s'ha rebut correctament, que no s'ha rebut correctament, quines dades s'estan enviant…
Per últim trobem dades la cadena de caràcters de longitud 100 caràcters en la que es basa el paquet. En aquest atribut es on es disposa la informació principal a enviar.

```
typedef struct {
    char origen[14];
    char tipus;
    char dades[100];
} Packet;
```

Station

Aquesta estructura de dades anomenada “Station” es la encarregada d’emmagatzemar tota la informació que defineix una estació meteorològica. Trobem 7 atributs.
El atribut “fileName” es on es guarda el nom de la estació, en “date” i “hour” trobem la data de la mesura juntament amb la seva hora en que s’ha fet.
Els següents 4 paràmetres són les unitats de mesura que s’han llegit en la estació, trobem temperatura, humitat, pressió atmosfèrica i precipitació.

```
typedef struct {
    char *fileName;
    char *date;
    char *hour;
    float temperature;
    int humidity;
    float atmosphericPressure;
    float precipitation;
} Station;
```

Config

Per tal de guardar els diferents paràmetres de configuració hem creat un tipus propi encarregat d'emmagatzemar-ho per cada sistema. El més extens es el de Danny mentre que el de Wendy i Jack idèntics.
A continuació explicarem el de Danny ja que conté inclou els atributs tant de Wendy com de Jack.
Trobem 7 atributs, en primer lloc el nom de la estació seguit de el path a la carpeta d’on s’extreuràn els fitxers, en tercer lloc trobem el temps de refres que tindrà aquella estació. Finalment trobem tant el IP com el PORT tant de Jack com de Wendy.

(Danny)

```
typedef struct {
    char *stationName;
    char *pathFolder;
    int revisionFilesTime;
    char *ipJack;
    int portJack;
    char *ipWendy;
    int portWendy;
} Config;
```

```
typedef struct {
    char *ip;
    int port;
} ConfigJack;
```

```
typedef struct {
    char *ip;
    int port;
} ConfigWendy;
```

Image (Danny)

Amb l'objectiu de guardar les imatges hem creat dos tipus propis (un que inclou l'altre). En aquesta guardem tota la informació associada a una imatge.
Està format per 4 atributs, nom del fitxer, mida de les dades en bytes, el string del hash del MD5SUM i finalment les dades de la imatge.

```c
typedef struct {
    char *fileName;
    int size;
    char *md5sum;
    char *data;
} Image;
```

```c
typedef struct {
    char *fileName;
} Image;
```

Halloran

Aquesta estructura de dades es l'encarregada de anar realitzant la mitjana aritmètica de totes les estacions que va llegint. Es utilitzada en el Fork anomenat Lloyd. Conté 5 atributs, en primer lloc el numero de lectures i les següents les sumes dels diferents paràmetres com poden ser humitat, temperatura, pressió atmosfèrica i precipitació.

```c
typedef struct {
    int countLectures;
    int totalHumidity;
    float totalTemperature;
    float totalAtmospheric;
    float totalPrecipitation;
} Hallorann;
```

Diagrama general

Com podem veure tenim un número indeterminat de clients (en el diagrama es representa amb 3 Dannies), que es comuniquen amb Jack i Wendy utilitzant sockets i Servidors Dedicats. Alhora Jack i Wendy utilitzen fork per generar processos i aquests gasten memòria compartida, pipes i semàfors.

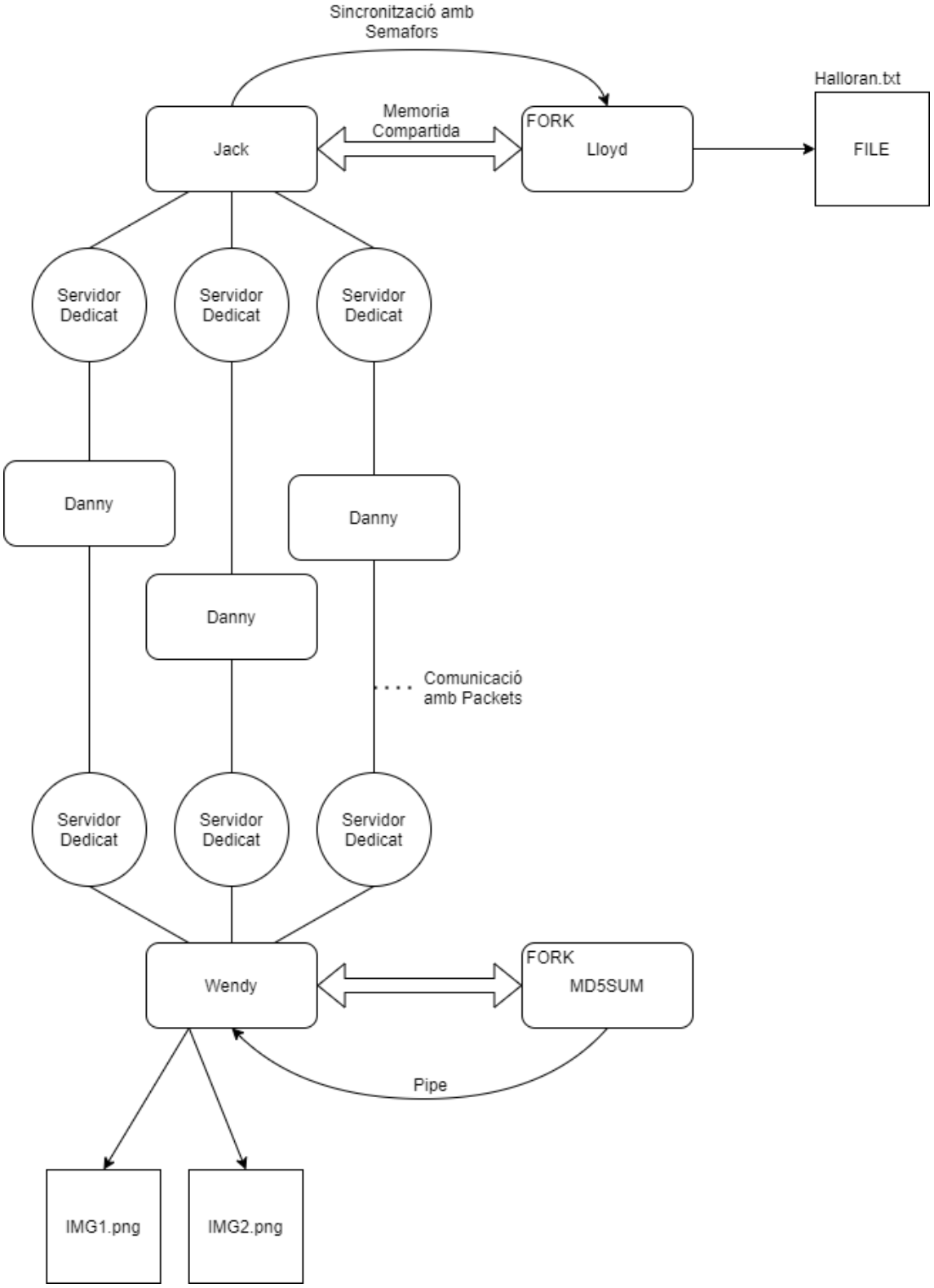


Diagrama comunicació (Danny - Jack)

Podem observar que trobem diferents comunicacions en forma de “Packets” entre Jack i Danny. Amb l’objectiu de transferir-se informació es fa ús de maneres de comprovar que les dades s’han enviat OK.

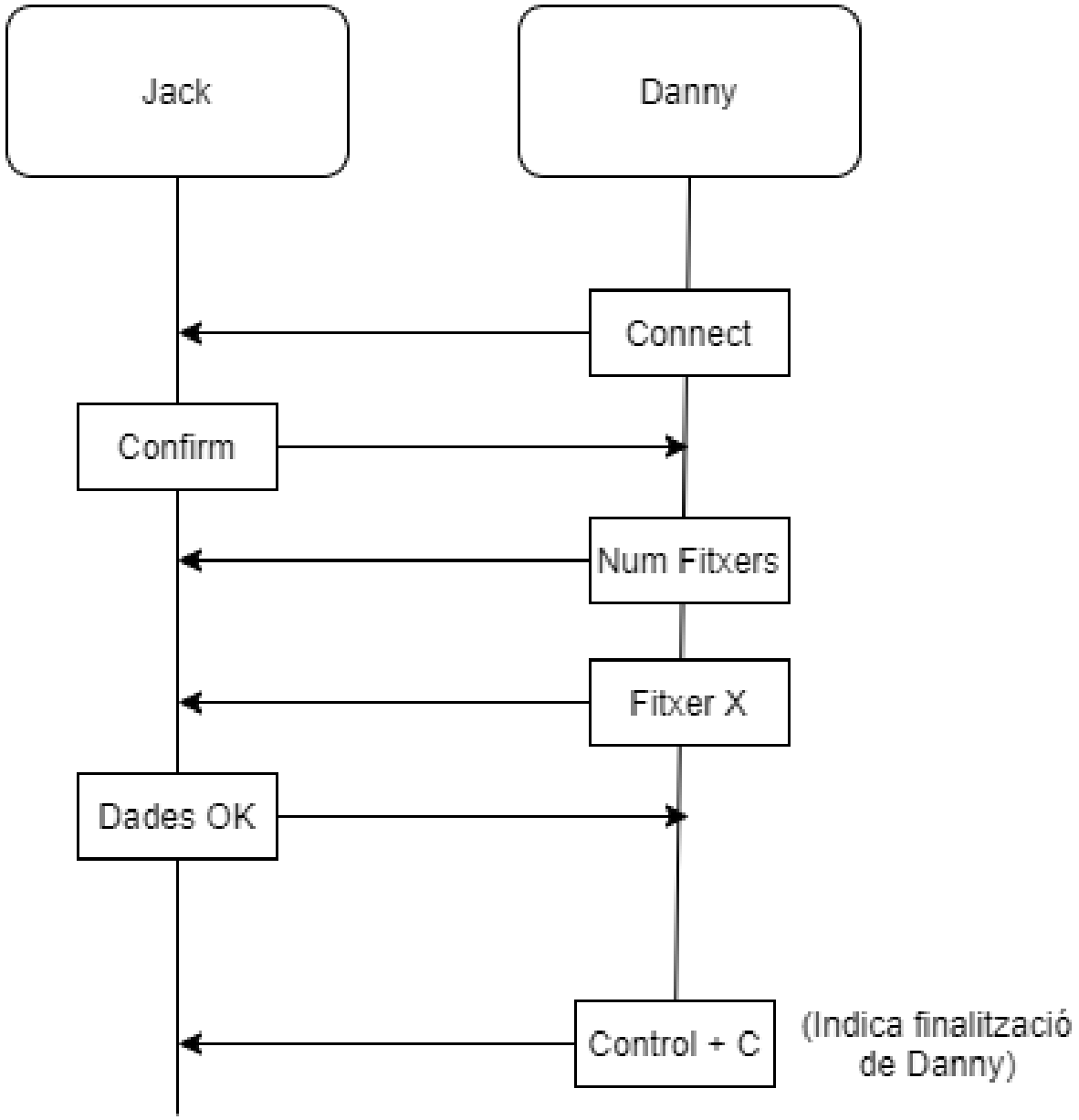
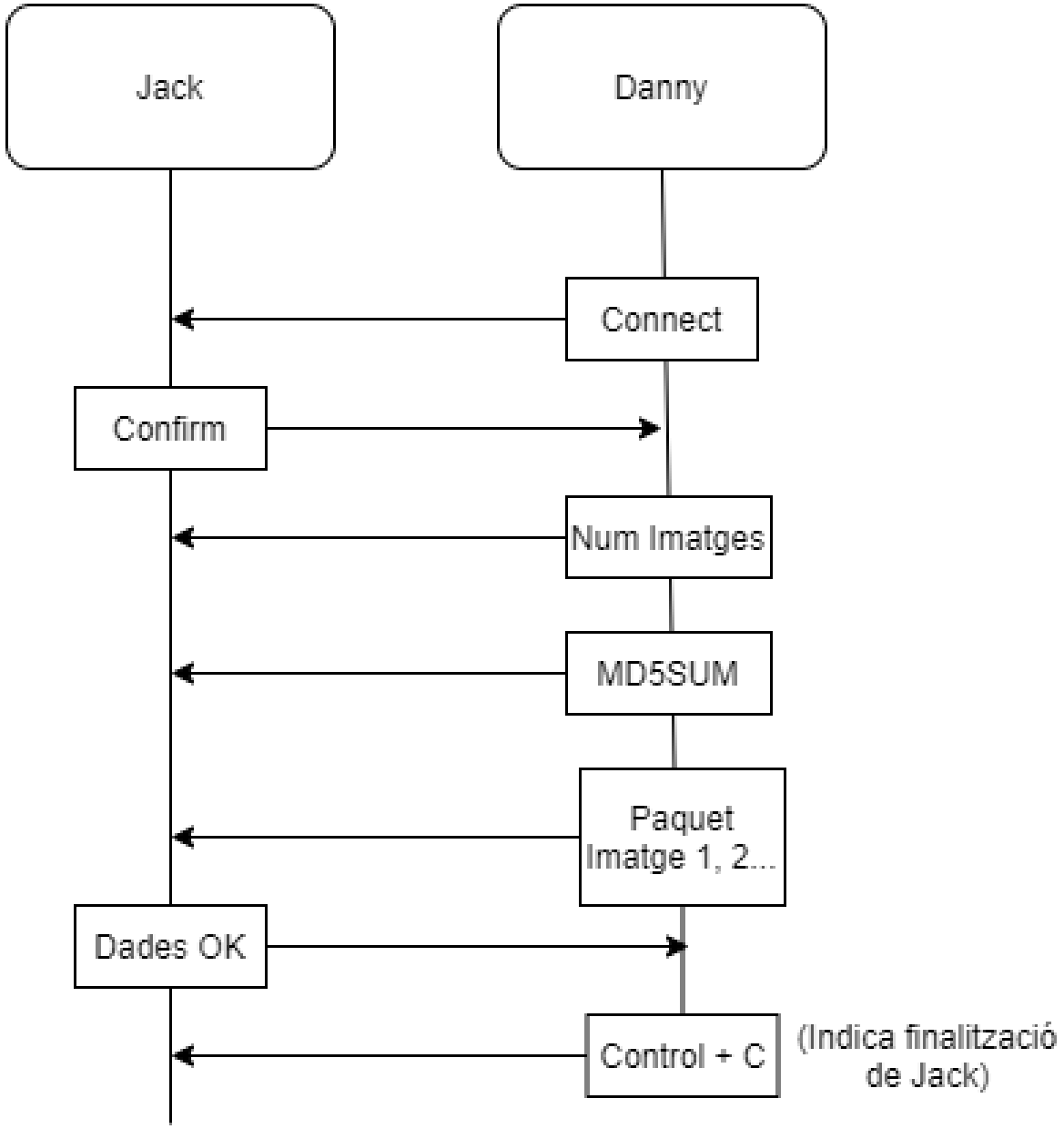


Diagrama comunicació (Danny - Wendy)

En el següent diagrama veurem característiques comunes amb el diagrama anterior en la seva estructura general. Destacar que en aquest diagrama per la robustesa de les dades s'aplica el MD5SUM (hash de un fitxer) per comprovar que el fitxer és exactament igual a el seu origen que al destí.



2. Problemes Observats

Durant la realització d'aquesta pràctica ens hem trobat amb bastants errors, els quals han relantitzat molt el desenvolupament de la pràctica. A continuació es detallen alguns d'aquests problemes:

En general, molts dels problemes que hem tingut en les fases inicials han estat de no estructurar correctament algunes arquitectures. Aquesta causa produïa que un cop estiguéssim arribant al final del desenvolupament d'aquell apartat, ens donéssim compte que aquella solució que havíem plantejat no era vàlida. Per millorar aquest problema, hem hagut de dedicar més temps a la planificació i disseny de la pràctica, per tal de evitar arribar a carrers sense sortida.

Alguns altres dels problemes que hem trobat estaven relacionats amb el servidor Matagalls/Montserrat, ja que a l'assignatura es treballa amb eines de molt baix nivell, era bastant freqüent col·lapsar-lo o produir errors. Per exemple, ens hem trobat amb problemes de masses sessions concurrents (ja que calia tenir bastantes sessions SSH per poder comprovar el funcionament), algun problema alhora de eliminar fitxers, es creaven alguns fitxers estranys (com es pot observar a la fotografia següent), etc.

```
guillem.serra@montserrat:~/Projecte1_SistemesOperatius/Projecte1_SistemesOperatius>ls -a
.  ..  .nfs000000000001f72100000052  .nfs00000000000644da00000051
guillem.serra@montserrat:~/Projecte1_SistemesOperatius/Projecte1_SistemesOperatius>rm .nfs00000000000
rm: cannot remove '.nfs00000000000': No such file or directory
guillem.serra@montserrat:~/Projecte1_SistemesOperatius/Projecte1_SistemesOperatius>rm .nfs000000000001f72100000052
rm: cannot remove '.nfs000000000001f72100000052': Device or resource busy
guillem.serra@montserrat:~/Projecte1_SistemesOperatius/Projecte1_SistemesOperatius>
```

Un altre error bastant comú era al fer bind del port pels servidors Jack i Wendy, ja que sempre que donava Segmentation Fault Core Dumped o el servidor no es tancava correctament (bastant comú a l'hora de fer testing), solia implicar que no es desconnectés correctament el servidor, i per tant quedés l'espai ocupat. En general es solucionava ràpid, però en alguns casos hem hagut de canviar temporalment la IP utilitzada pels sockets.

```
miquel.zamora@montserrat:~/so/Projectel_SistemesOperatius>./Jack ConfigJack.dat

Starting Jack...

Error durant el bind del port (Servidor Jack)!
```

La resta d'errors, tot i no ser pocs, eren de caràcter més general i solució ràpida, sobretot per no implementar correctament el codi (no coincidir els write's amb els read's, no utilitzar el socket correcte, utilitzar incorrectament algunes comandes, no posar on pertocava el '\0' en els arrays de caràcters, etc.).

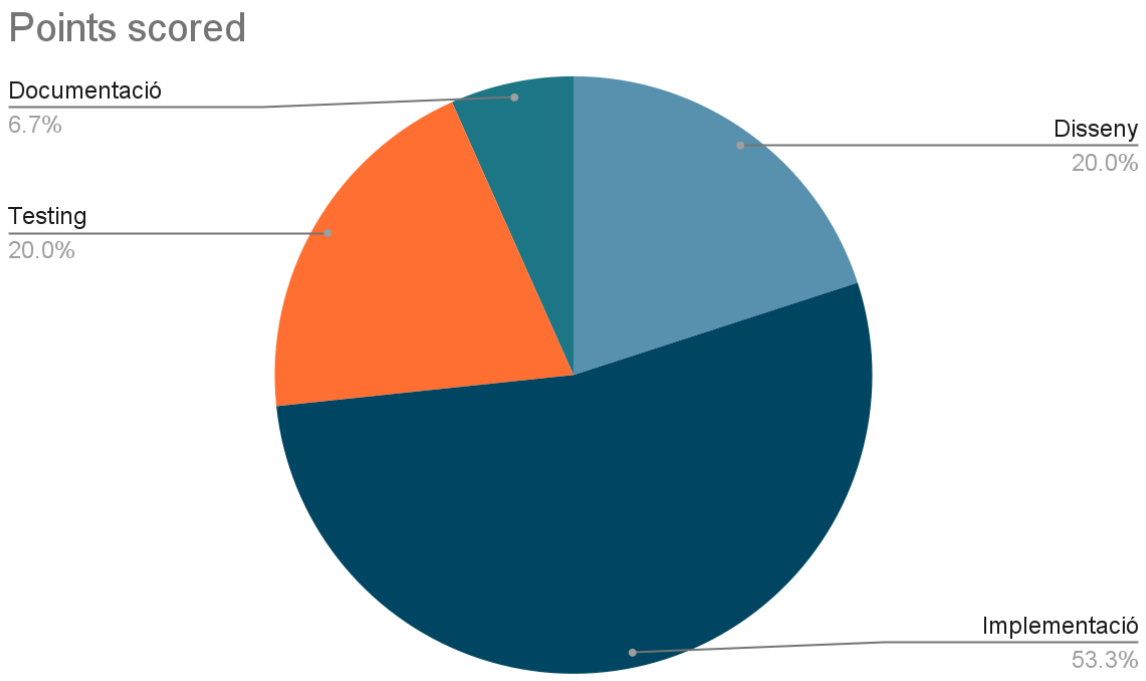
3. Estimació temporal

Durant la realització d'aquesta pràctica ens hem trobat amb bastants errors, els quals han relantitzat molt el desenvolupament de la pràctica. A continuació es detallen alguns d'aquests problemes i les divisions del temps que hem considerat.

En primer lloc trobem la fase de Disseny, durant aquesta fase inicial plantejàvem els dissenys que consideràvem oportuns per la seva posterior implementació. En aquesta fase els dissenys els parlàvem i com a millora per un futur seria interessant prendre el temps de documentar-los durant les reunions.

A continuació trobem tant la implementació com el testing. Aquests dos apartats van molt lligats ja que durant la implementació es realitzava el testing per anar veient si funcionava el que s'havia implementat. Destacar que una part important del temps de Testing es deguda a memòries no alliberades que produïen errors, ports bindejats…

Finalment trobem l'apartat de Documentació, entenent la creació dels Diagrames, manteniment dels mateixos amb els canvis que anàvem realitzant, i la redacció de la memòria.



4. Conclusions i propostes millora

Aquesta pràctica ens ha permès aprendre a implementar i utilitzar les eines donades a classe. Juntament amb les sessions de laboratori hem après no només a simplement utilitzar-les, sinó també a fer un disseny d'un sistema per el nostre compte amb bastanta llibertat. També ens ha funcionat molt bé per detectar errors de concepte ja que alhora de implementar per el nostre compte ens hem adonat que hi havien conceptes que realment no havíem entès.

Una de les avantatges de treballar en una pràctica com aquesta en grup ha estat que el disseny havia d'estar consensuat. Aquestes discussions sobre les diferents arquitectures que es farien ens ha permès trobar millor solucions que si s'hagués fet de forma individual.

Alhora ens ha plantejat la dificultat de fer un bon disseny a la primera, ja que a causa de no haver treballat amb aquestes eines amb anterioritat ens hem anat trobant amb dificultats en la implementació que produïen iteracions. Aquestes iteracions són una de les coses que ens ha permès aprendre i interioritzar millor els conceptes.

Com a treballs futurs interessants trobem que ens agradaria muntar aquest sistema en un model més físic com podria ser Arduinos o Raspberrys, realitzant totes les adaptacions necessàries. D'aquesta manera podriem veure d'una manera una mica més "real" el sistema que hem muntat i veure'l interactuar amb la realitat.

Una millora interessant ja no només de cara a l'assignatura sinó fins i tot en un futur seria fer alguna sessió de testing on aprendre a fer-los i programar-los. Cada cop que implementàvem o tocàvem alguna cosa s'havia de testejar tota l'aplicació. Donat que en C el testing Unitari i Funcional es "bastant complex" i realment ens ho hem plantejat cap al final del desenvolupament, no l'hem implementat pero en fases futures del projecte creiem que no es una facilitat sinó que es una necessitat. Destacar que vam començar uns scripts de bash perquè fessin el testing, però es complicava ja que redirigíem la sortida de la terminal a fitxers i després s'havia de fer la lectura d'ells.

Una proposta de millora que crec que tot i que es va mig plantejar durant les sessions seria interessant de fer de cara a la pràctica té a veure amb els servidors Montserrat i Matagalls. La proposta es basa en donar suport oficial per a realitzar la pràctica no només en una "iso" de una màquina virtual, sinó penjar el mateix compilador que te putty per poder fer-ho en el entorn que preferim. Entenem les raons que per la que no es proporciona, però de totes maneres creiem que desde el punt de vista de usabilitat seria una millora substancial.

5. Bibliografia

NOTA: Els continguts i materials de la classe estaven bastant bé, juntament amb les solucions de les sessions, els apunts… Molta part tant dels coneixements teòrics com els més pràctics els hem extret d'allí.

J. Salvador "Programació en UNIX per a pràctiques de Sistemes Operatius" [Online] https://estudy.salle.url.edu/pluginfile.php/982014/mod_folder/content/0/LlibreUNIX_14-15.pdf?forcedownload=1

DZone. (2020, Jun. 16). Parallel TCP/IP Socket Server With Multithreading and Multiprocessing in C [Online]. Available: https://dzone.com/articles/parallel-tcpip-socket-server-with-multi-threading

Geek for Geeks. (2019, Dec. 9). fork() in C [Online]. Available: https://www.geeksforgeeks.org/fork-system-call/

Man7. (2021, Jun. 20). fork(2) — Linux manual page [Online]. Available: https://man7.org/linux/man-pages/man2/fork.2.html

Linux Hint (). How to use signal handlers in C language? [Online]. Available: https://linuxhint.com/signal_handlers_c_programming_language/