2018-2019

Cosgrove System

Stairway to heaven

Meritxell Ciuraneta Solé – meritxell.ciuraneta

Christian García Mascarell – christian.garcia

Índex

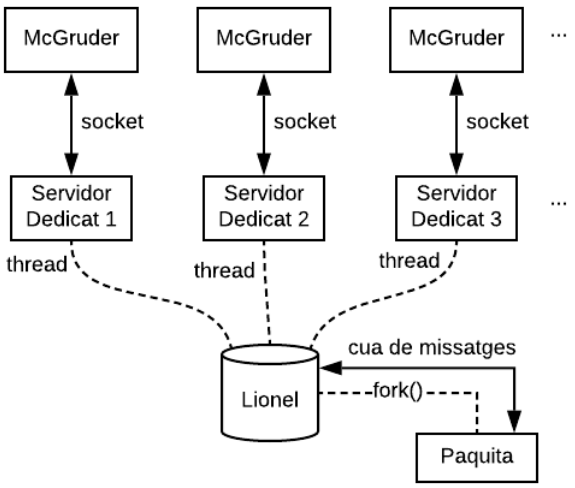
1. Descripció de requisits

L'escenari tracta sobre un conjunt de telescopis que recullen dades periòdicament (imatges d'alta resolució i mesures de les constel·lacions), els quals són anomenats McGruder.

Aquestes dades recopilades es pretenen enviar a un servidor centralitzat anomenat Lionel, el qual estarà disponible per a la comunitat científica per poder consultar les dades (estructura client-servidor). Per accedir-hi faran servir un procés anomenat McTavish.

A més a més, per tal de no saturar el servidor, es disposarà d'un procés addicional al servidor per tal de realitzar els càlculs pertinents respecte les dades rebudes dels telescopis McGruder. Aquest procés s'anomenarà Paquita.

Cal mencionar que, en aquest projecte, per tal de poder realitzar l'entrega a temps, s'ha optat per prescindir del procés McTavish, de manera que no s'explicarà en aquest document.



a. Fase 1

En aquesta fase es dissenyarà el procés McGruder (a nivell local, és a dir, sense connexió amb el servidor), els quals correspondran als clients del sistema.

Com a tal, el procés McGruder haurà de comprovar si hi ha dades noves per enviar, de manera que necessitarà:

- Un nom de telescopi associat per tal d'identificar-lo quan es realitzi la connexió amb el servidor Lionel.
- Un temps d'espera per consultar si hi ha noves dades per enviar per tal de no saturar la CPU (livelock).
- Una direcció IP associada al servidor Lionel on enviar les dades, ja que els telescopis no es troben en la mateixa màquina que el servidor Lionel.
- Un port per on enviar les dades al servidor McGruder.
- Una carpeta on es trobaran els fitxers de dades recollides pel telescopi.

Totes aquestes dades seran emmagatzemades en un fitxer de configuració per tal de poder carregar-les al programa un cop s'executi passant-lo com a paràmetre.

Pel que respecta als fitxers a enviar, aquests seguiran el format "aaaammdd-hhmmss.end" on "end" pot ser "jpg" o "txt" (case insensitive), distingint entre imatge i fitxers de dades.

b. Fase 2

En aquesta fase es resoldrà l'establiment de connexió entre el servidor Lionel i qualsevol quantitat de McGruders (és necessari l'ús de servidors dedicats).

i. Lionel

Pel que respecta al servidor, necessitarem d'una configuració de connexió basada en[1]: una direcció IP per on escoltar les peticions de connexió dels clients, un port per rebre les peticions dels McGruder i un port per rebre les peticions dels McTavish.

Aquestes dades també seran emmagatzemades en un fitxer de configuració que es llegirà quan s'executi el servidor Lionel i serà passat per paràmetres.

A més a més, per tal de poder estar escoltant les peticions de connexió per part dels clients i de poder estar atent de les peticions dels clients ja connectats, es farà servir un servidor dedicat per cada client connectat (necessitat de més d'un fil d'execució).

ii. Protocol de comunicació

Tant els clients com el servidor requeriran d'un protocol de comunicació per tal de poder notificar i respondre les peticions de connexió i de desconnexió.

A més, com que aquest no només consistirà en aquests dos processos, el protocol haurà de ser escalable[2].

[1] A l'enunciat de la pràctica es proposava un darrer paràmetre basat en segons per parametritzar l'enviament de dades, però no s'ha usat pas.
[2] El protocol de comunicació es troba detallat a l'annex del document.

c. Fase 3

En aquesta fase s'implementarà la transmissió de dades entre els McGruder i el servidor Lionel.

Per tal de poder realitzar l'enviament de dades, cal destacar que, per poder enviar imatges (degut a la seva mida), caldrà descomposar-les per tal de no saturar la connexió, mentre que els fitxers de text no ho necessitaran.

Per tal de mostrar aquest fraccionament en el servidor, caldrà mostrar el percentatge de compleció de la transmissió de les dades. Tot i així, caldrà comprovar la rebuda del fitxer original (ús de md5sum).

A més, caldrà registrar els fitxers rebuts al servidor en un fitxer de text (Kalkun.txt per les imatges i Kalkun_v2.txt per els fitxers de dades) un cop es tanqui el servidor, on s'enregistrarà: la data i hora de rebuda del fitxer i el nom i mida (en bytes) de l'arxiu rebut.

d. Fase 4

En aquesta fase es dissenyarà el procés Paquita, el qual serà un fil d'execució independent del procés Lionel.

Paquita com a tal, degut a la seva funció de càlcul d'estadístiques quan hi hagi nova informació, caldrà que esperi a obtenir informació proporcionada pel procés Lionel per tal de poder calcular:

- Actualitzar el nombre de imatges o fitxers de dades rebuts.
- La mida total de les imatges rebudes (en kB).
- La mitjana de constel·lacions per arxiu.

La informació de la qual partiran aquests càlculs consta, a part de la mida del fitxer i el seu tipus, de:

- El número de constel·lacions del fitxer.
- La mitjana de densitats.
- El valor màxim i mínim de magnituds.

2. Disseny

a. Com s'han assolit els requeriments

i. Fase 1

1) Configuració del client

Res més començar, la primera cosa que ha calgut ha estat comprovar que el fitxer de configuració introduït com a paràmetre existís i bolcar les dades del fitxer en una variable global del programa[3].

Si succeís algun error, es procediria a tancar el client directament.

2) Notificació cada x segons

Un cop obtinguda la configuració i la connexió amb el servidor[4], necessitem quedar en *blocked* fins que passin els segons especificats en la configuració per tal de no gastar temps de CPU.

Per tal d'aconseguir això, just després d'obtenir la configuració del fitxer, es procedirà a sobreescriure la senyal d'alarma (SIGALRM) i li assignem una funció que s'encarregui de comprovar l'existència de fitxers per enviar.

A més, crearem un bucle que generi el senyal SIGALRM i es quedi a *blocked* (fent servir la funció *pause*), de manera que, quan es rebi el senyal de SIGALRM, procedirem a usar la funció establerta i així infinitament[5].

3) Escaneig de fitxers

La funció prèviament especificada, farà ús de la funció *scandir*, *output* el qual serà tractat per tal de separar els tipus de fitxers segons la terminació que aquests tenen (jpg o txt).

Cal remarcar que, si succeís algun error, es procediria a tancar el client alliberant la configuració guardada en el programa (motiu pel qual és una variable global) i notificar el servidor de la desconnexió[6].

[3] A l'apartat 3 d'aquesta fase s'explica el motiu pel qual la variable és global.
[4] Implementat a la Fase 2.
[5] En el punt 4 d'aquesta mateixa fase veurem que no és exactament així.
[6] Implementat a la Fase 2.

4) Tancament inesperat del programa

Per tal d'evitar que el programa deixi *memory leaks* quan es tanqui inesperadament, sobreescriurem el senyal de SIGINT.

Juntament a aquesta sobreescriptura, farem servir una variable global a mode de *flag* que ens indiqui si volem tancar el programa o no.

D'aquesta manera, es comprovarà el *flag* just abans de fer operacions importants per tal d'alliberar les variables que estiguessin sent utilitzades i notificar al servidor de la desconnexió del client[7].

Aquestes operacions importants són:

- La condició de continuació del bucle d'espera del senyal SIGALRM.
- L'escaneig de fitxers.

Cal mencionar que l'ús d'aquest *flag* pot ser usat per les posteriors fases.

[7] Implementat a la Fase 2.

ii. Fase 2

1) Configuració del servidor

A l'igual que el client, es necessita obtenir la configuració del servidor un cop iniciat el programa a partir del fitxer introduït com a paràmetre, bolcant la informació en una variable, en aquest cas local, ja que només serà usat pel procés Lionel.

Si succeís algun error, es disposaria a tancar el servidor directament.

2) Configuració de la connexió

Al client, per tal de configurar la connexió, requerirem de la funció *socket* (obtenció del *socket* de la connexió), de la configuració del *socket* (establiment de l'adreça IP i del port) i de la connexió amb el servidor usant la funció *connect*.

Cal tenir present que s'hauran d'acceptar adreces IPv4 de tot tipus, de manera que caldrà usar la funció *gethostbyname* i emprar-la correctament.

Pel que respecta al servidor, parteix de la mateixa premissa que el client, però, en comptes d'usar *connect*, es farà servir la funció *bind* (el *socket* serà usat per escoltar les peticions de connexió del client).

Cal mencionar que en ambdós processos els *sockets* de connexió seran variables globals, ja que pel que respecta al client, es necessitarà per al procés de comunicació, mentre que el servidor haurà de poder tancar-se en cas de tancament inesperat[8].

En el cas que sorgís algun error, tant el client com el servidor es desconnectarien adequadament.

3) Escoltar sol·licituds de connexió

Pel que respecta al servidor, aquest necessitarà d'un bucle infinit[9] que estarà escoltant constantment les peticions de connexió dels clients.

Un cop rebuda la sol·licitud per part d'un client, es disposarà a crear un nou *thread* per tal d'escoltar les peticions en particular del McGruder recent connectat.

[8] Això serà comentat a l'apartat 6 d'aquesta mateixa fase.
[9] Es veurà que això no és del tot cert a l'apartat 6 d'aquesta mateixa fase.

Cal mencionar que, per tal de reutilitzar la variable usada per obtenir el *socket* del client connectat que necessita el *thread* que farà de servidor dedicat, farem ús d'un semàfor usant la sincronització per notificar al servidor que escolta noves peticions de connexió que el *socket* ja està copiat.

A més, s'afegiria el *thread* generat en una llista (variable local, ja que només la necessita el procés Lionel) per tal de tenir consciència de quins *threads* s'han creat i alliberar-los de memòria al tancar el programa.

En el cas que sorgís algun error, el servidor seguiria a l'espera de noves sol·licituds, ja que el programa segueix sent operatiu per als altres clients.

4) Procés de connexió

Per tal de poder connectar-nos a un servidor amb McGruder, s'enviarà una sol·licitud de connexió fent servir el protocol de connexió pertinent[10].

En el cas que es rebés una resposta del servidor indicant que tot ha anat bé, es procediria amb l'escaneig periòdic dels possibles fitxers del telescopi, sinó es procediria a tancar el client (si sorgís un error, faria el mateix).

Pel que respecta al servidor dedicat, aquest estaria pendent d'una notificació de connexió del client amb el nom del mateix.

En el cas que el missatge rebut no fos l'adequat (o sorgís un error), es procediria a tancar el servidor dedicat.

En canvi, si tot anés bé, es procediria a estar a l'espera de nous missatges per part del McGruder associat en un bucle infinit[11].

5) Procés de desconnexió

Per tal de poder desconnectar-nos d'un servidor amb McGruder, s'enviarà una sol·licitud de desconnexió al servidor.

Cal mencionar que s'esperarà una resposta del servidor, però, tot i que la resposta del servidor no sigui la correcta o sorgeixi un error, es tancarà el client, ja que el client no

[10] El protocol de comunicació es troba detallat a l'annex del document.
[11] Es veurà que això no és del tot cert a l'apartat 6 d'aquesta mateixa fase.

haurà de controlar res més i el servidor, un cop tancat el *socket*, si tracta d'enviar quelcom, s'adonarà del client desconnectat.

Pel que respecta al servidor, en el bucle d'espera de missatges del client, podrà rebre una sol·licitud de desconnexió (també en el cas que es llegeixi una trama no identificada o sorgeixi error al llegir del *socket*), moment en què es procedirà a enviar una resposta al client.

A l'igual que passa amb el client, si donés algun error en la resposta, es procediria a tancar el servidor dedicat, ja que el client procedirà a tancar-se igualment si detecta que el *socket* s'ha tancat.

6) Tancament inesperat del programa

Per tal d'evitar que el programa del servidor deixi *memory leaks* quan es tanqui inesperadament, sobreescriurem el senyal de SIGINT.

Juntament a aquesta sobreescriptura, farem servir una variable global a mode de *flag* que ens indiqui si volem tancar el programa o no, com ara amb els clients i també ens disposarem a tancar el *socket* usat per escoltar noves peticions de connexió, ja que aquest podria estar en una espera bloquejant.

D'aquesta manera, es comprovarà el *flag* just abans de fer operacions importants per tal d'alliberar les variables que estiguessin sent utilitzades, tancar els *sockets* i *threads* corresponents i notificar els clients de la desconnexió del servidor.

Aquestes operacions importants són:

- El bucle d'espera de missatges per part del client.
- El processament de la resposta del servidor a una petició d'un client, enviant una resposta de desconnexió.

Com a resultat de l'activació del *flag*, també es procediria a esperar tots els *threads* creats per tal d'alliberar tota la memòria emprada pel programa (en comptes de fer un *pthread_detach*, que no alliberaria la memòria usada in situ, però sí quan acabés el procés en teoria).

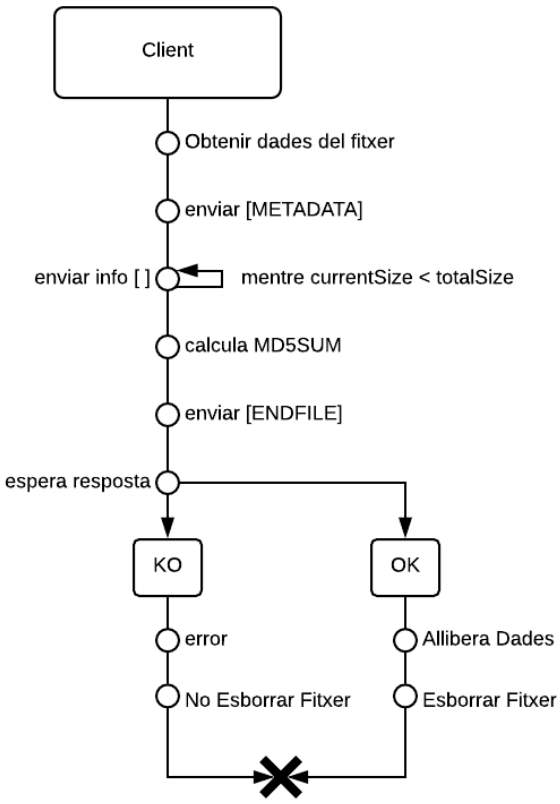
iii. Fase 3

1) Enviament de fitxers

Partint del client que ja feia un escaneig dels fitxers que té disponibles per enviar, un cop fet això, es requerirà llegir el seu contingut abans d'enviar la informació del mateix (llegir byte a byte fins que la funció *read* dona error).

Un cop llegida la informació, caldrà enviar-la al servidor seguint el procés següent:

- Enviar la *metadata* del fitxer, ja sigui "jpg" o "txt".
- Enviar la informació del fitxer fragmentada (si es tracta d'un fitxer de dades menor a 4kB, s'enviarà sencer) fins acabar l'enviament.
- Enviar el *md5sum* del fitxer[12].



Cal mencionar que sempre que s'enviï informació del servidor, aquest ens respondrà per tal de saber si ha anat bé o si ha anat malament i tancar el client davant de la fallida (o davant d'una trama no esperada).

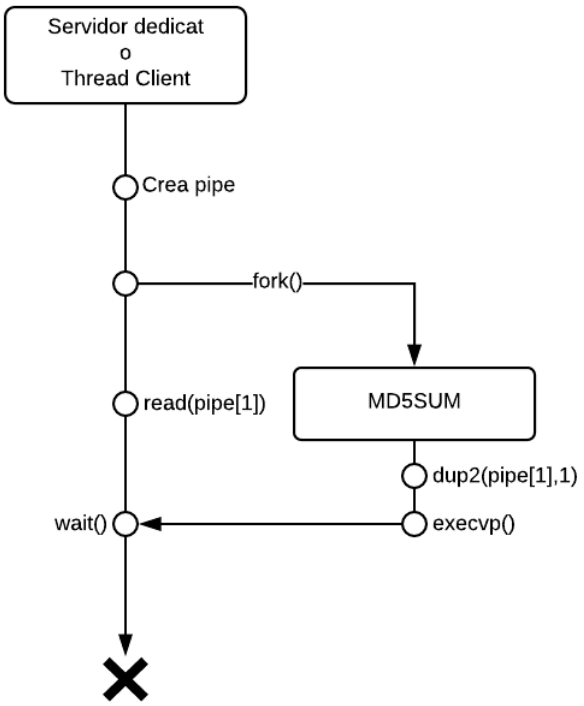
[12] Serà explicat a l'apartat 2 d'aquesta mateixa fase.

A més a més, es revisarà el *flag* de tancament del servidor:

- Abans i durant la lectura del contingut del fitxer.
- A cada resposta del servidor que sigui vàlida (per defecte les invàlides acaba resultant en el tancament del client).

2) Càlcul md5sum

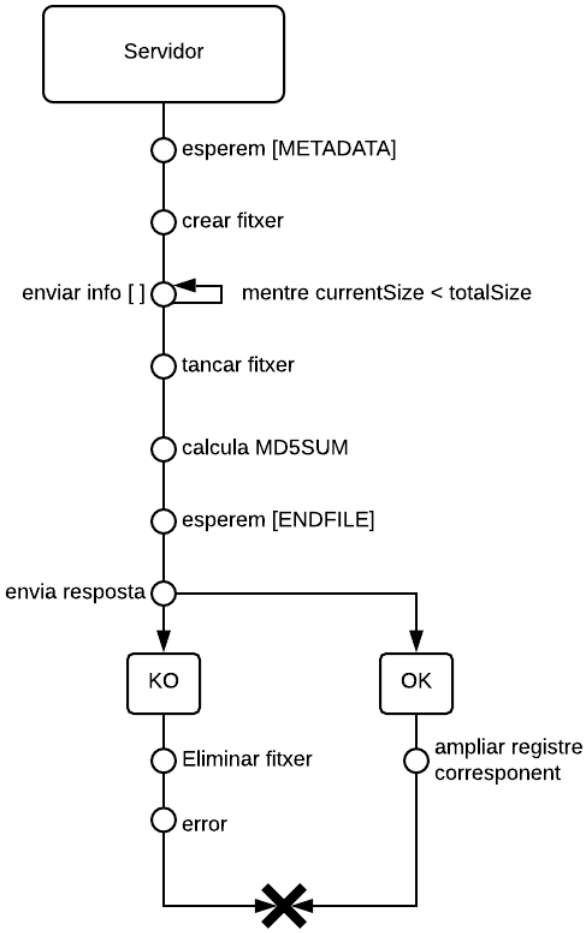
Per tal de calcular el *md5sum* (al client i servidor), caldrà fer ús de la funció *execvp* per poder executar la comanda *md5sum* de Linux.



Per tal d'aconseguir això, caldrà fer ús d'un *fork* i d'una *pipe* a la qual serà redirigit el *file descriptor* d'escriptura mitjançant la funció *dup2*, de manera que podrem llegir la resposta *md5sum* del fill des de la *pipe* en el procés pare.

3) *Rebuda de fitxers*

El servidor, per tal rebre fitxers, haurà de rebre un missatge per part del client que especifiqui el començament de l'enviament d'un fitxer, especificant el tipus de fitxer, la data de creació i la seva mida, dades que es guardaran en una variable al servidor durant l'enviament de dades (addicionalment tindrà el registre de quantes dades s'han rebut fins al moment).



Un cop analitzada la *metadata*, es contestarà al client respecte si ha anat bé o no aquest anàlisi i estarem a l'espera de paquets que ens donin nova informació o, en cas que s'hagi rebut tota la informació, el *md5sum* origen, responent a tots els missatges del client.

Cal remarcar que només es bolcarà la informació rebuda un cop s'hagi rebut tota per tal d'evitar un excés d'accés a la memòria de disc.

Un cop rebuda tota la informació, es calcularà el *md5sum* local i es compararà amb el rebut, responent al client en base al resultat obtingut.

Si tot ha anat bé, es guardarà la informació necessària en una variable que farà de registre dels enviaments (una per fitxers de dades i una per les imatges), variable que necessitarà un semàfor per garantir exclusió mútua (poden modificar les dades qualsevol servidor dedicat).

Un cop s'acabi el programa, es bolcarà la informació dels registres en uns fitxers que faran d'històric dels enviaments (Kalkun.txt per les imatges i Kalkun_v2.txt pels fitxers de dades).

Cal mencionar que, davant de qualsevol error de processament o d'enviament, es tancarà el servidor dedicat i, consegüentment, també el client.

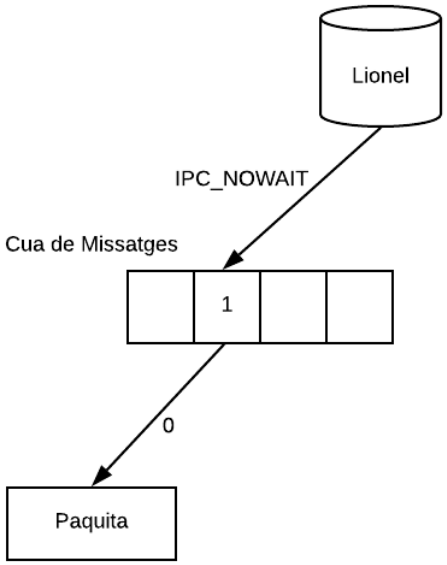
A més a més, per tal de tancar el programa correctament davant del tancament amb Ctrl+C, revisarem el *flag* abans de processar el contingut del paquet rebut pel client en l'enviament de dades.

iv. Fase 4

1) Plantejament del procés Paquita, comunicació i dades

Per tal de poder crear un nou procés que no faci ús de *threads*, es farà un *fork* just abans d'executar Lionel, de manera que serà independent del procés Lionel.

Cal remarcar que, abans de fer el *fork*, es crearà una cua de missatges per tal de poder enviar les dades necessàries per actualitzar les estadístiques del procés Paquita. A més, gràcies a això, si Paquita cau per algun motiu, el funcionament de Lionel no es veurà afectat.



Pel que respecta a les dades estadístiques, degut que no es realitzarà la Fase 5, no hi ha necessitat d'usar memòria compartida per a les estadístiques de Paquita (ni els semàfors corresponents).

2) *Enviament de dades*

Aprofitant l'actualització dels registres un cop rebut un fitxer, podem usar les dades per crear el missatge a enviar al procés Paquita a la bústia 1 en una escriptura no bloquejant (IPC_NOWAIT)[13].

A més, com que Paquita no pot operar amb fitxers, es llegirà la informació del fitxer rebut (si és de dades) per enviar-la en el paquet.

3) *Rebuda de dades*

El procés Paquita disposarà d'un bucle infinit (fins que el *flag* ho indiqui) en què estarà a l'espera d'un missatge a la bústia 1 (només es farà servir una) en una lectura bloquejant.

Un cop rebuda informació vàlida, segons el tipus de fitxer rebut, actualitzarà la informació relativa als fitxers de dades o de les imatges.

A més, per tal de mostrar l'actualització d'aquests valors, es mostraran les dades per pantalla a cada recepció d'informació i quan el programa es tanqui.

[13] El contingut del missatge serà explicat a l'apartat d'Estructures de dades usades, més concretament a NewDataMsg.

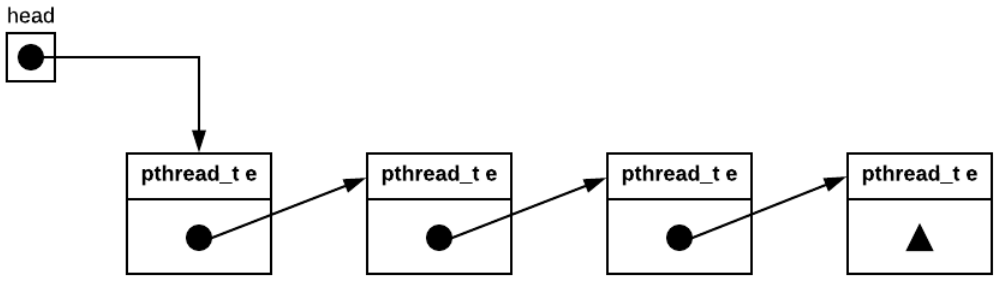
b. Estructures de dades usades

i. Lionel

Flist

Segueix l'estructura d'una FIFO[14]. S'utilitza per emmagatzemar els identificadors dels threads que es llencen per atendre als McGruder per tal de poder gestionar-los en cas de tancament inesperat o error.

- Node* head: Punter al primer Node de la FIFO.
- Node:
    - pthread_t e: Id del thread.
    - Node* next: Punter al següent Node.



Config

Estructura que conté els paràmetres de configuració de Lionel per tal d'iniciar el socket que escolta les peticions dels possibles clients.

- char* ip: IP del socket.
- int mcgruder_port: Port on es connectaran els McGruder.
- int mctravish_port: Port on es connectaran els McTravish.
- int param: Paràmetre sense ús.

SendConfig

Conté la informació per enviar/rebre un fitxer del tipus que sigui.

- char type[4]: Tipus del fitxer (jpg o txt).
- long size: Mida total de les dades del fitxer.

[14] Originàriament era una llista sense punt d'interès, però amb els canvis constants per debugging es podria substituir per un array dinàmic.

- long current_size: Mida de les dades rebudes.
- char* data: Dades.
- char* file: Nom del fitxer.

AstroData

Estructura que s'encarrega d'emmagatzemar les dades astronòmiques llegides del darrer fitxer de text rebut a Lionel. Aquestes dades Lionel les ha d'enviar a Paquita.

- long total_const: Número total de constel·lacions.
- double density_avg: Mitjana de les densitats de les constel·lacions.
- double min_mag: Valor mínim de les magnituds.
- double max_mag: Valor màxim de les magnituds.

NewDataMsg

Estructura que conté totes les dades que Lionel ha d'enviar a Paquita per mitjà d'una cua de missatges.

- long id: Id de la bústia de la cua de missatges a la què es vol enviar el missatge (només s'usarà la bústia 1).
- int type: Indica de quin tipus de fitxer (jpg o txt) s'estan enviant les dades.
- double kbytes: Total de kBytes que ocupen les imatges rebudes a Lionel.
- struct AstroData: Les dades astronòmiques extretes d'un fitxer txt rebut a Lionel.

ii. McGruder

Config

Estructura que conté els paràmetres de configuració del McGruder.

- char* telescope: Nom del Telescope associat a aquest McGruder.
- int revision_time: Cada quants segons ha de revisar si hi ha fitxers la carpeta de fitxers.
- char* address: IP on s'ha de connectar el McGruder.
- int port: Port on s'ha de connectar el McGruder.
- char* files_dir: Nom de la carpeta on ha de buscar els fitxers.

iii. Lionel i McGruder

NetPack

Estructura de dades que conté els camps necessaris per realitzar el protocol de comunicació[15].

- char type: Tipus de la trama en format hexadecimal.
- char* header: Header de la trama.
- long length: Indica la llargària del camp data.
- char* data: Aquest camp emmagatzema les dades que s'han d'enviar en la trama.

[15] El protocol de comunicació es troba detallat a l'annex del document.

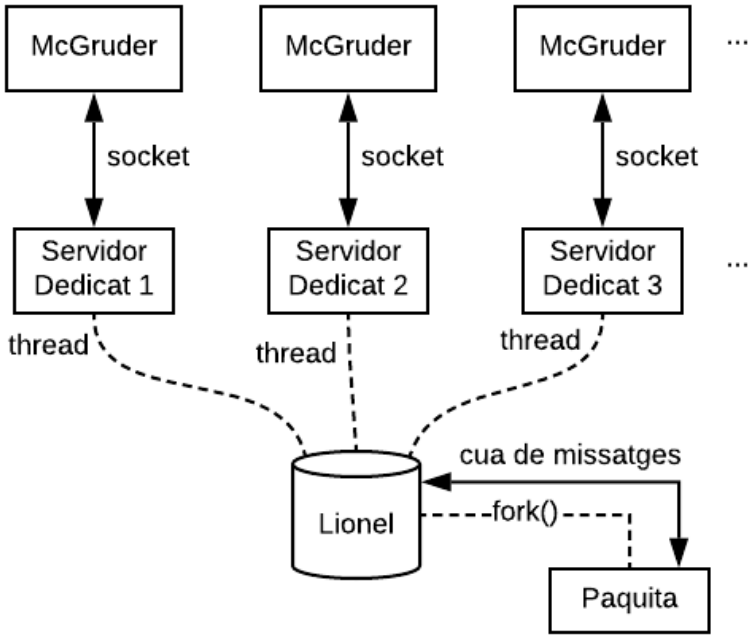
c. Explicació dels recursos emprats

i. Sockets

Com que Lionel i els McGruders s'han de poder comunicar entre sí i pot ser que no estiguin a la mateixa màquina, s'han d'utilitzar sockets, els quals permeten l'enviament d'informació per mitjà de trames a partir d'una adreça IP i un port.

Per tal d'assegurar que els dos processos es puguin comunicar correctament s'ha implementat un protocol de comunicació[16] basat en unes trames. Aquestes trames tenen diferents camps que permeten distingir que és el que es vol comunicar a l'altre procés.

Cada cop que un procés envia un missatge a un altre, aquest últim (si està "viu") sempre respon amb la informació que se li ha demanat o amb una notificació, la qual normalment indica si una acció s'ha pogut dur a terme amb èxit o no (a excepció de quan es notifica que no hi ha fitxers a enviar).



[16] El protocol de comunicació es troba detallat a l'annex del document.

ii. Threads

Lionel és un servidor que necessita mantenir la connexió estable mentre un McGruder s'executa i, com que un sol Lionel ha de poder gestionar més d'un McGruder alhora, s'ha optat per crear servidors dedicats mitjançant threads.

El fil d'execució principal de Lionel s'encarrega d'anar mirant si hi ha noves connexions de nous McGruder. Quan hi ha una nova connexió es llença un fil d'execució separat (servidor dedicat -> thread) per gestionar totes les demandes del McGruder.

iii. Forks

Quan es crida a la funció *fork* es genera un duplicat del procés actual. Aquest duplicat comparteix els valors actuals de totes les dades, però quan s'actualitzen en un dels dos llocs (procés pare o fill) no queden actualitzats en l'altre.

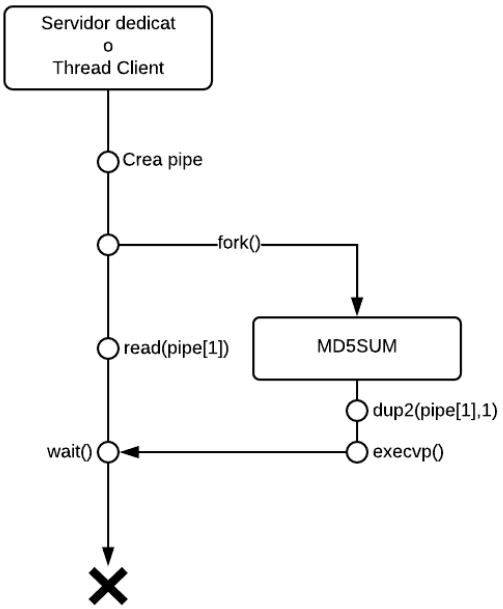
- MD5SUM

  Per tal de realitzar l'md5sum del fitxer sense tenir que implementar-lo, el que s'ha fet ha estat llençar un fork en el qual s'ha cridat la funció execvp passant-li per arguments el nom del procés que volem executar, en aquest cas l'md5sum del sistema Linux:

  ```
  char * args[] = {"md5sum" , nomFitxer, NULL};
  execvp(args[0], args);
  ```

  A més a més, abans de crear el fork s'ha tingut de crear una pipe per tal de poder passar-li al procés pare el resultat del md5sum des del fill. Alhora, s'ha fet un dup2 per tal de redirigir el file descriptor de la pantalla (on s'escriu originalment l'md5sum del sistema) al file descriptor de la pipe que s'ha creat per tal d'obtenir el checksum en el procés pare.

A continuació es mostra el procés que es realitza per tal de calcular l'md5sum d'un fitxer.



- Paquita

  El procés Paquita ha d'estar en la mateixa màquina que Lionel i per tant s'hauria de poder executar-la tant utilitzant un thread com utilitzant un fork, però com què un dels requeriments de la Fase 4 era no utilitzar un thread per Paquita, s'ha utilitzat un fork.

iv. Cua de Missatges

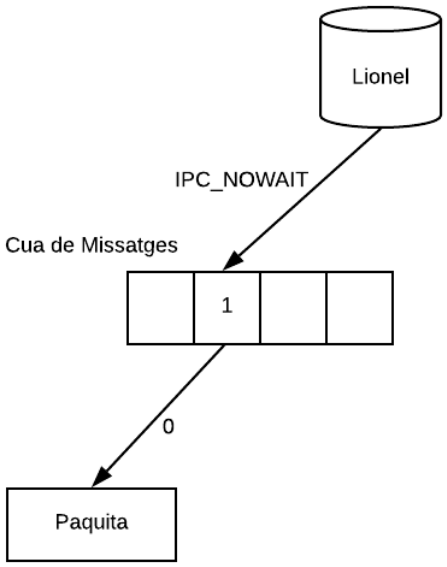
Per tal de comunicar Paquita amb Lionel s'ha utilitzat una cua de missatges, ja que Paquita és un fork de Lionel i encara que tingui una còpia de les variables de Lionel, aquestes no s'actualitzen en Paquita quan es canvien en Lionel i viceversa.

Lionel va enviant a la bústia de la cua sense esperar-se a que Paquita hagi llegit el missatge anterior, és a dir, treballa amb escriptura no bloquejant.

En canvi, Paquita s'espera a que hi hagi algun missatge a la bústia per llegir-hi. Per tant, treballa amb una lectura bloquejant.

Lionel només envia i Paquita només rep.

El tipus de missatge que utilitzen per comunicar-se és del tipus NewDataMsg[17].



v. Semàfors

Els semàfors són una eina per garantir l'exclusió mútua sobre una mateixa direcció de memòria o per sincronitzar diferents processos o fils d'execució.

D'una banda s'utilitza un semàfor per sincronitzar el procés principal Lionel amb els seus servidors dedicats, per tal que Lionel s'esperi a que el thread s'hagi guardat la id del client abans de que Lionel accepti un altre client i aquesta id canviï.

D'altra banda s'utilitzen dos semàfors per assegurar l'exclusió mútua de dues variables, una relacionada amb la informació dels fitxers txt i l'altra amb les imatges. Aquest mutex impedeix que més d'un servidor dedicat alhora les pugui modificar.

vi. Signals

Tant Lionel com McGruder han de poder suportar un CTRL-C (SIGINT). De fet, Lionel i McGruder s'aturen normalment d'aquesta manera.

Per tractar aquesta situació el que s'ha fet ha estat sobreescriure el handler de la interrupció SIGINT per tal que, abans de tancar el programa, el procés alliberés tota la memòria dinàmica i notifiqués a qui fos necessari del seu tancament.

[17] Se n'ha explicat l'estructura en l'apartat d'Estructures de dades usades.

McGruder ha de mirar si hi ha fitxers en la seva carpeta cada un cert temps, per fer-ho s'ha sobreescrit el handler de la interrupció SIGALARM per tal de fer-ho cada vegada que salta la interrupció.

Finalment, a McGruder també s'ha sobreescrit la interrupció SIGPIPE per tal que detecti el tancament d'un socket per part del servidor quan no s'està enviant informació de cap mena (motiu pel qual s'ha afegit una nova trama al protocol de connexió)[18].

[18] Per a més informació, consultar l'annex del document.

3. Proves realitzades

Cal remarcar que cada fase té les seves proves, però hi ha una prova comuna a totes: comprovar l'alliberament de tota la memòria davant d'un tancament inesperat.

a. Fase 1

En aquesta fase s'han realitzat les següents proves:

- Mostrar per pantalla els valors obtinguts de la configuració per comprovar que la lectura ha funcionat com és degut.
- Mostrar per pantalla els fitxers obtinguts amb l'escaneig de la carpeta assignada per comprovar el seu correcte funcionament.
- Modificar el període d'escaneig dels fitxers juntament amb un rellotge per comprovar la validesa de l'espera.
- Modificar la carpeta d'escaneig per comprovar si identifica correctament el destí.

b. Fase 2

En aquesta fase s'han realitzat les següents proves:

- Mostrar per pantalla els valors obtinguts de la configuració per comprovar que la lectura ha funcionat com és degut (Lionel).
- Comprovar la connexió de diferents clients al servidor amb noms de clients diferents.
- Comprovar la desconnexió dels diferents clients provant la desconnexió en ordres diferents i intercalades amb noves connexions.

c. Fase 3

En aquesta fase s'han realitzat les següents proves:

- Usar imatges de diferents mides (petites i grans) per a l'enviament.
- Enviar fitxers mesclats i d'un sol tipus.
- Mostrar la data de recepció dels fitxers.
- Comprovar la mida resultant i el càlcul md5sum dels fitxers rebuts comparat amb l'original.

- Tancar un client o el servidor enmig d'un enviament per comprovar si notificava correctament a l'altre extrem.
- Comprovar els fitxers de registre durant una transmissió de dades, un cop acabada i al tancar el programa.

d. Fase 4

En aquesta fase s'han realitzat les següents proves:

- Enviar diversos fitxers (mesclats) i comprovar les dades actualitzades del procés Paquita, incloent els càlculs fets a mà.
- Mostrar les dades obtingudes del fitxer de dades astronòmiques rebuts per comprovar la seva validesa.
- El tancament de Paquita per comprovar si Lionel seguia operatiu.

4. Problemes observats i com s'han solucionat

a. Lectura de la configuració

Aquest problema consistia que el fitxer de configuració a vegades es generava al servidor de Montserrat en un entorn Linux i d'altres vegades es generava en un entorn Windows.

Això provocava que al mostrar les dades, per algun motiu es sobreescrivien les sortides per consola depenent de si el fitxer era creat en un entorn o un altre.

El problema consistia que Windows, quan fa un salt de línia, usa el caràcter '\r', el qual consisteix en moure el cursor al principi de la línia, provocant que es sobreescrivís la sortida.

Això s'ha solucionat comprovant si aquest caràcter es trobava al contingut extret del fitxer i eliminant-lo de les dades.

b. Enviament fraccionat

Un problema bastant recurrent al llarg de la Fase 3 era que a vegades s'enviaven bé les dades i altres vegades no.

Això causava que només es poguessin enviar correctament els fitxers de dades astronòmiques, mentre que les imatges no arribaven intactes al servidor, sinó que arribaven distorsionades.

Això era provocat pel mal tractament de les dades a enviar, ja que sempre se li afegia un '\0' al final de les dades fraccionades i es considerava part de la informació quan s'enviava al servidor.

A més a més, com que les imatges podien contenir un '\0' i s'usava *strlen* per saber la mida de les dades a enviar, es distorsionaven els fitxers rebuts sempre que s'haguessin fraccionat per enviar.

Això s'ha solucionat especificant la mida de les dades sense fer servir la funció *strlen*, de manera que, encara que pel manegament de dades del client o servidor s'usés el caràcter '\0', no afectaria a l'enviament del fitxer.

c. Data de recepció

Quan ens disposàvem a generar els registres dels enviaments de fitxers, vam adonar-nos que la data obtinguda no era la correcta.

Llegint la documentació de les funcions *time* i *localtime*, vam trobar l'origen d'aquest error.

Sembla ser que, usant aquestes funcions, aconseguim l'hora local (configurada al servidor de Montrserrat, que va amb una hora de retràs), però l'any corresponia als anys que han passat des del 1900, de manera que s'havia d'afegir aquest offset a l'any.

A més, cal tenir present que el mes estava entre els valors 0 i 11, de manera que s'ha sumat un 1 per equivaler al mes real.

d. Tancament del programa

Un dels principals problemes que ha sorgit amb aquesta pràctica han estat els *memory leaks* un cop es tancava el client o servidor.

El principal problema sorgia perquè, en un principi, al tractar la interrupció SIGINT, es procedia a tancar immediatament el client i el servidor, de manera que si s'estava usant alguna variable auxiliar pel motiu que fos, aquesta no s'alliberava.

És per això que finalment s'ha optat per usar el *flag* comentat en la fase de disseny que permetia alliberar la memòria que estava sent usada.

e. Tancament dels servidors dedicats

Quan es tancava el servidor, quan s'usava la funció *pthread_detach*, deixava *memory leaks* deguts, segurament, al *thread* que seguia en execució a l'espera del tancament del client.

Això s'ha pogut solucionar, però amb una petita pega: el servidor no es tancarà per complet fins que tots els clients s'hagin desconnectat.

La solució a consistit en fer un *pthread_join* de tots els *threads* que s'haguessin creat abans de tancar el servidor per complet.

f. Client infinit

Un dels problemes que s'havia mesclat amb altres ha estat que, en el cas que el client no tingues fitxers a enviar, no hi havia manera que el servidor notifiqués al client del seu tancament i que aquest se n'adonés d'això, ja que no hi havia motiu per parlar amb el servidor.

Per solucionar això, s'ha creat un nou tipus de trama que servís per notificar a Lionel que el client seguia viu i, de manera indirecta, poder adonar-nos que el servidor s'havia tancat gràcies al senyal SIGPIPE.

g. Bytes flotants

Un dels problemes més desapercebuts fins a l'ús de *Valgrind* era que hi havia cadenes de caràcters que no disposaven del valor '\0' les quals feien servir la funció *strcpy* i les seves variants, causant errors interns a partir del seu ús.

Això, però, ha tingut fàcil solució, ja que es solucionava afegint un '\0' com a delimitador.

h. Missatge no inicialitzat

Un problema bastant peculiar ha estat que, quan s'enviava un missatge usant la cua de missatges entre Lionel i Paquita, sempre sorgia un error usant *Valgrind*.

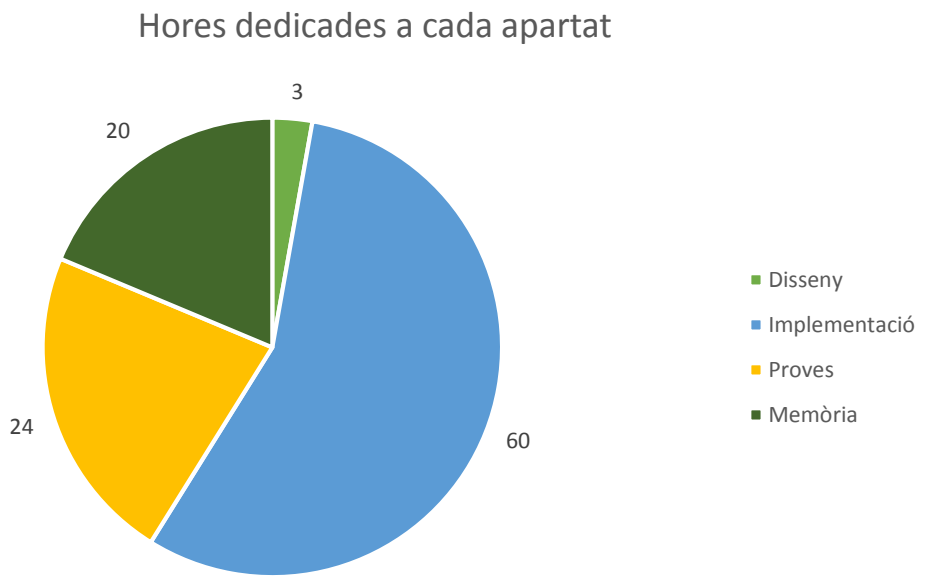
Aquest error notificava que hi havia dades no inicialitzades, cosa que no arribàvem a solucionar usant la funció *memset*.

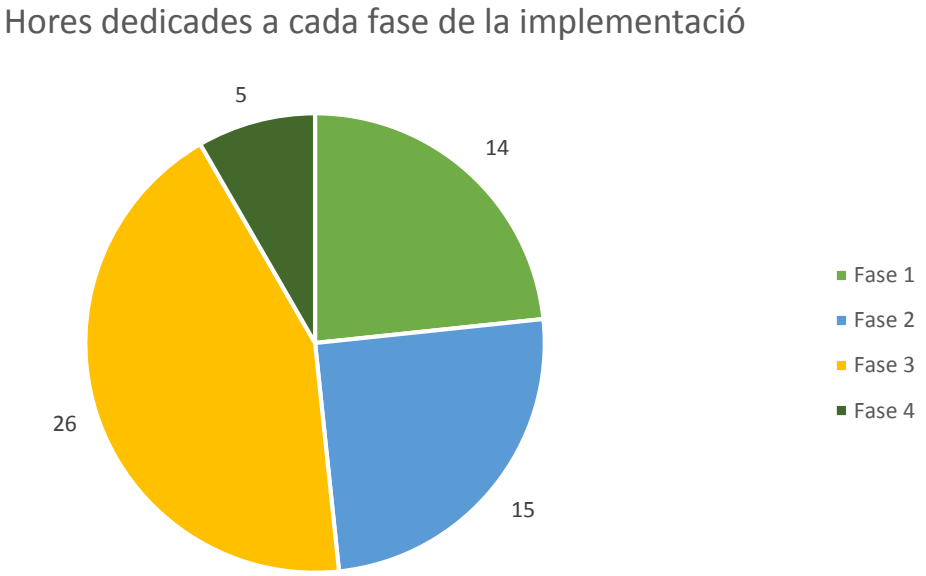
Aquest problema és degut que, depenent de l'arquitectura sistema, la cua de missatges pot aplicar un farciment de bytes sense inicialitzar per aconseguir la mida que necessita per manegar les dades internament.

És per això que, sense poder controlar aquest aspecte degut al funcionament de la cua de missatges, s'ha prescindit d'aquest error donat per *Valgrind*.

5. Estimació temporal

Finalment, s'han requerit aproximadament unes 107 hores per tal de poder implementar les 4 fases obligatòries de la pràctica.





Tal i com es pot observar en el gràfic anterior, la part de la implementació que va dur més temps va ser la de la fase 3. Això es degut que era la fase amb més complicació de les 4 i en la que van sorgir més complicacions.

6. Conclusions

La implementació d'aquesta pràctica ens ha servit per consolidar tots els coneixements explicats a classe i practicats en les diverses sessions realitzades en el laboratori, així com aprendre'n de nous.

Inicialment sentíem les sessions de laboratori com un moment de molta tensió, ja que només pensàvem amb els "patitos", però després d'unes poques sessions ens vam adonar de què els "patitos" eren el que menys importava. Vam observar com, practicant els conceptes explicats a classe en aquestes sessions, ens ajudava molt a l'hora d'implementar la pràctica. A més a més, hem pogut reutilitzar algun codi de les sessions per la pràctica.

Un cop acabada l'assignatura, si mirem enrere al primer dia de Sistemes Operatius on només sabíem amb C el que ens van explicar a Programació de 1r, ens n'adonem del poc que realment sabíem en aquell moment i el munt de coses útils que sabem ara (i de les que ens queden per conèixer).

Per tal de poder tirar la pràctica endavant, hem vist que és molt important validar un disseny lògic que es vol implementar, ja que, encara que fallin algunes coses, saps que vas ben encaminat i que l'estructura amb la que treballes és correcta. D'altra banda, si no es valida el disseny la pràctica, pot fallar per moltes coses i no saps si és culpa del disseny que has implementat o del disseny del qual es parteix.

En la implementació de les dues primeres fases es van poder seguir els *checkpoints*, ja que vam tenir més temps per dedicar a la pràctica i perquè eren senzilles. Però, quan va arribar el *checkpoint* de la Fase 3, aquesta encara no estava acabada, degut a falta de temps per dedicar-li, ja que teníem moltes altres pràctiques a entregar abans de Nadal. Hem vist que ha faltat una mica d'organització i que per tant hem de millorar en aquest aspecte.

Finalment, un cop acabada la pràctica hem pogut observar com la unió dels conceptes apresos a Programació de 1r i els apresos aquest any a Sistemes Operatius ens han donat una visió més amplia del rang de possibilitats que el llenguatge de programació C ens ofereix i com poder aprofitar al màxim aquestes eines.

7. Bibliografia

Linux man pages online. (2018). Recollit de man7.org/linux/man-pages/index.html

Pont, J. S. (2014). *Programació en UNIX per a pràctiques de Sistemes Operatius.*

Stack Overflow - Where Developers Learn, Share, & Build Careers. (2008). Recollit de https://stackoverflow.com/

8. Annex

a. Protocol de comunicació Lionel – McGruder

Camps de la trama del protocol:

TYPE (1 Byte)	HEADER (X Bytes)	LENGTH (2 Bytes)	DATA (LENGTH Bytes)

- Type: Indica el tipus de la trama en format hexadecimal.
- Header: És de mida variable. Sempre estarà envoltat per claudàtors, per tant com a mínim el cams serà: "[]".
- Length: Camp de 2 bytes que indica la llargària del camp DATA.
- Data: La seva mida en Bytes s'especifica en el camp LENGTH. Aquest camp emmagatzema les dades que s'han d'enviar en la trama.

i. Nova connexió

TYPE: 0x01

✉ McGruder -> Lionel

HEADER: []

LENGTH: Mida del nom del Telescope

DATA: Nom del Telescope

✉ Lionel -> McGruder

HEADER: [CONOK] (Si la connexió s'ha realitzat correctament)

[CONKO] (Si no s'ha pogut establir la connexió)

LENGTH: 0

DATA: Buit

ii. Desconnexió

TYPE: 0x02

✉ McGruder -> Lionel

HEADER: []

LENGTH: Mida del nom del Telescope

DATA: Nom del Telescope

✉ Lionel -> McGruder

HEADER: [CONOK] (Si s'ha pogut desconnectar correctament)

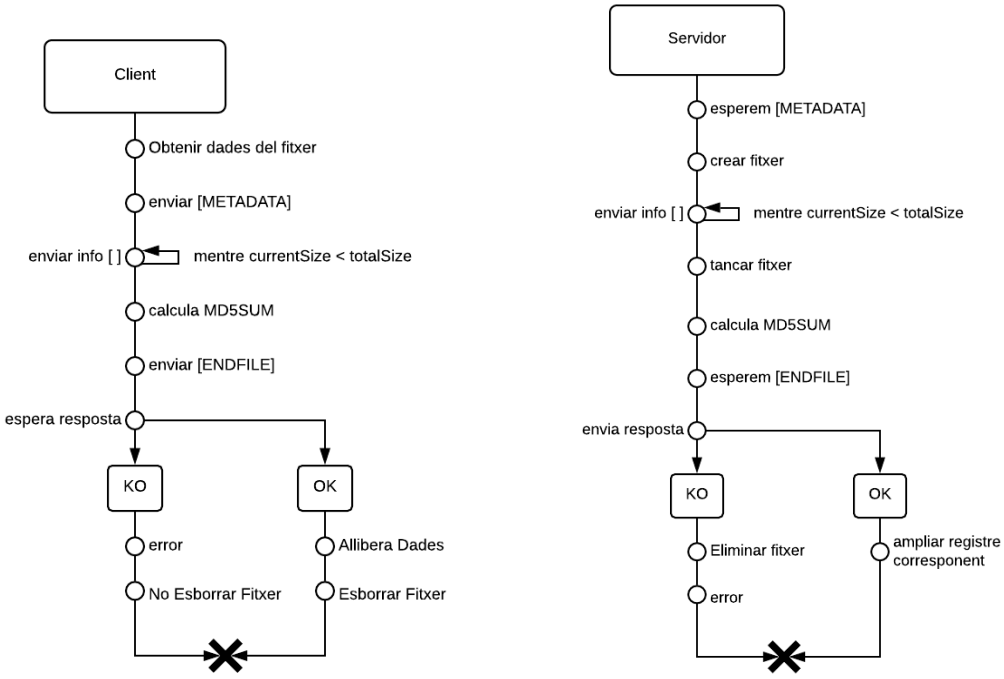
[CONKO] (Si no s'ha pogut desconnectar correctament)

LENGTH: 0

DATA: Buit

iii. Enviar Fitxer

Esquemes de la comunicació per enviar/rebre un fitxer:



TYPE: 0x03

✉ McGruder -> Lionel

- Trama per indicar l'enviament

HEADER: [METADATA]

LENGTH: Mida de Data

DATA: Tipus del fitxer + Mida del fitxer + Data de creació del fitxer

- Trama per enviar les dades (depenent de la mida del fitxer aquesta trama s'haurà d'enviar vàries vegades)

HEADER: []

LENGTH: Mida de Data

DATA: Dades del fitxer

- Trama per indicar el final de la transmissió de dades

HEADER: [ENDFILE]

LENGTH: Mida de Data

DATA: *Checksum* realitzat amb l'*md5sum*

✉ Lionel -> McGruder

- Trama per indicar si la transmissió del fitxer ha estat correcta

HEADER: [FILEOK] (Si s'ha transmès el fitxer correctament)

[FILEKO] (Si no s'ha transmès bé el fitxer)

LENGTH: 0

DATA: Buit

- Trama per indicar si el *checksum* és correcte o no

HEADER: [CHECKOK] (Checksum correcte)

[CHECKKO] (Checksum incorrecte)

LENGTH: 0

DATA: Buit

iv. McGruder no troba més fitxers. Lionel estàs viu?

TYPE: 0x06

✉ McGruder -> Lionel

HEADER: []

LENGTH: 0

DATA: Buit