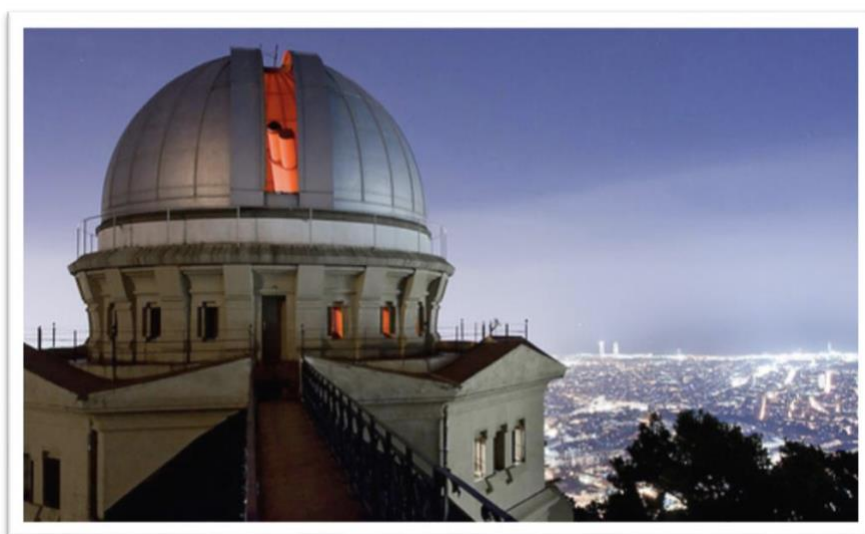


Pràctiques de Sistemes Operatius
Curs 2018-2019

Cosgrove System

Stairway to heaven



Alumnes	Login	Nom
	lluis.masdeu	Lluís Masdeu
	oriol.ramis	Oriol Ramis

Data	24/03/2019
------	------------

1. Índex

1. ÍNDEX	2
2. DESCRIPCIÓ DE REQUISITS.....	3
3. DISSENY.....	5
3.1. FASE1.....	5
3.2. FASE2.....	6
3.3. FASE3.....	7
3.4. FASE4.....	12
4. PROVES REALITZADES.....	13
4.1. FASE1.....	13
4.2. FASE2.....	13
4.3. FASE3.....	14
4.4. FASE4.....	14
5. PROBLEMES OBSERVATS I COM S'HAN SOLUCIONAT	14
6. ESTIMACIÓ TEMPORAL.....	15
7. CONCLUSIONS	17
8. BIBLIOGRAFIA	18
9. ANNEX.....	19
9.1. PROTOCOL DE COMUNICACIÓ GENERAL	19
9.2. PROTOCOL DE CONNEXIÓ DE McGRUDER AMB EL SERVIDOR LIONEL.....	19
9.3. DISTRIBUCIÓ DELS FITXERS DE LA NOSTRA APLICACIÓ	22

2. Descripció de requisits

En aquesta pràctica de l'assignatura de Sistemes Operatius se'ns demana la creació d'un software encarregat de realitzar observacions i mesures astronòmiques de l'Observatori Fabra.

Més concretament, els telescopis s'encarreguen de recollir les dades durant la nit tot captant imatges d'alta resolució. Així mateix, també s'encarreguen de recollir mesures periòdicament de les constel·lacions. Les mesures són dades numèriques i de poc volum d'emmagatzematge, a diferència de les imatges.

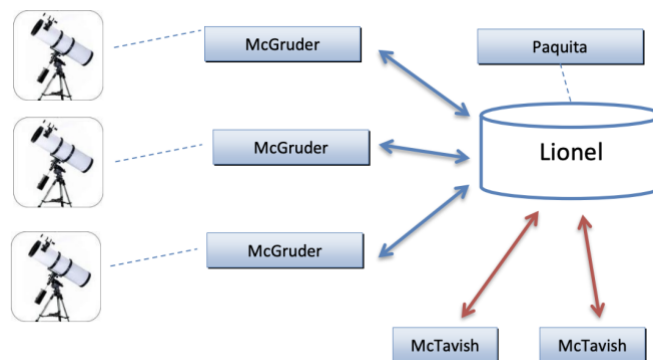


Figura 1 Esquema funcional del sistema Cosgrove

Així doncs, en aquest sistema cal destacar dues grans entitats encarregades de dur a terme dues tasques diferents. Primerament, tindríem Lionel, el qual és un procés central i serà un procés únic d'aquest tipus. D'altra banda tenim el procés McGruder, el qual hi serà tants cops com quantitat de telescopis connectats hi hagi a l'Observatori, tenint d'aquesta manera un sistema escalable.

Pel que fa a les dades, aquestes s'enviaran desde el McGruder cap a Lionel essent eliminades del primer un cop transmeses. Tal i com s'ha comentat anteriorment, aquestes podran ser de tipus .txt, en el cas de fitxers de mesures, i .jpg en el cas de les imatges d'alta resolució captades pels telescopis.

A part dels dos procediments, n'hi ha un tercer anomenat Paquita, situat dins de la mateixa màquina de Lionel i encarregat de realitzar els càlculs sobre les dades astronòmiques rebudes dels telescopis mitjançant el procediment McGruder.

Val a dir, però, que McGruder és un procediment autònom i que no depèn de l'usuari, ja que només mira si hi ha dades a enviar i les hi passa al Lionel tot seguit.

Cadascun dels dos grans blocs de la pràctica tindrà un fitxer de configuració, que se'ls hi passarà per paràmetre, per a facilitar la comunicació entre ells. Aquests fitxers contindran la següent informació:

- **Lionel**
<NomTelescopi>
<TempsRevisioNovaDada>
<IPLionel>
<PortLionel>
- **McGruder**
<IPServer>
<PortConnexioMcGruder>
<PortConnexioMcTavish>
<TempsEspera>

Per a facilitar la realització del que se'ns demana, s'ha decidit dividir la pràctica en 5 fases ben diferenciades:

- **Fase 1:** Es crea McGruder amb la seva connexió directa al telescopi. Tot seguit es processa el fitxer de configuració i es realitzen la detecció periòdica de dades rebudes amb el seu corresponent tractament.
- **Fase 2:** En aquesta segona fase, s'implementa el procés Lionel i es crea la connexió entre aquest i els diversos McGruder's.
- **Fase 3:** En aquesta fase toca implementar la transmissió de dades dels McGruder's cap a Lionel, així com la seva monitorització, el seu registre i el seu emmagatzematge.
- **Fase 4:** Aquesta és la última fase obligatòria i toca implementar tota l'operativa del procés Paquita, el qual serà l'encarregat de fer els càlculs de les dades astronòmiques que li demani Lionel.
- **Fase 5:** És una fase opcional i és l'encarregada de poder fer gestionar totes les funcionalitats dels processos McTavish a través de Lionel.

3. Disseny

En aquest apartat s'explicarà com hem afrontat les diferents fases.

3.1. Fase 1

En aquesta fase, el primer que s'ha de realitzar és la lectura de les configuracions passades per línia de comandes i els fitxers que ha capturat el telescopi. Per guardar-nos cada tipus de informació, hem creat un tipus propi per a cada una:

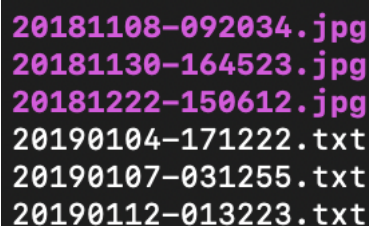
```
typedef struct _configparams {
    char * telescopeName;
    int waitTime;
    char * ipAddress;
    int port;
} ConfigParams;

typedef struct _packet {
    int type;
    char* header;
    short length;
    char* data;
    char* fileName;
} Packet;
```

Per tal de poder organitzar-ho tot millor. Hem creat dues classes noves, una encarregada de la gestió de configuració i una altre encarregada de la connexió.

Dins de la de configuració, tenim un procediment encarregat de splitejar el contingut del .dat, tot tallant pels '\n' i a posteriori, una classe encarregada de parsejar, posa el contingut en l'ordre correcte dins de l'estructura ConfigParams. Com és lògic, toca demanar memòria per a aquesta, ja que es tracta d'un apuntador a una regió de memòria

Per a la lectura del fitxer, hem creat, a part, una classe McGurder_List en forma de llista dinàmica, que s'encarrega d'emmagatzemar tots els fitxers trobats a la carpeta files, els quals posteriorment s'enviaran a Lionel. Hem utilitzat una llista per tal de fer-ho més òptim, donat que si en algun moment no es vol esborrar el contingut de la carpeta, però tampoc es vol enviar contingut duplicat, es podria controlar a partir d'aquesta.



```
20181108-092034.jpg
20181130-164523.jpg
20181222-150612.jpg
20190104-171222.txt
20190107-031255.txt
20190112-013223.txt
```

Figura 2 Exemple de fitxers trobats a la carpeta files

Dins del procediment `CONNECTION_programExecution`, el qual s'encarrega de l'execució del programa, hem controlat, mitjançant un bucle i un `sleep`, que aquest vagi fent pauses del temps indicat en el fitxer `.dat`, i que s'encarrega de quantificar la freqüència en que ha de detectar si hi ha noves dades a la carpeta `files`.

També hem creat el control de la interrupció del programa mitjançant `signals`, que posa a = la variable `isRunning` del bucle infinit d'execució del programa.

3.2. Fase 2

Començarem centrant-nos en `McGruder`, el qual hem decidit implementar, sense necessitat de cap eina com ara `threads` o `semàfors`, donat que la tasca de `McGruder` és lineal i sense interrupcions.

Pel que fa a aquesta fase, és una mica més complexa que l'anterior. Hem hagut de replantejar-la un cop creada, per tal de suportar correctament la convivència de diversos clients connectats a la vegada, tenint com a suport compartit a la pantalla. Per fer-ho, a Lionel hi hem creat el següent struct:

```
typedef struct {  
    int size;  
    int currentIndex;  
    int * clientFd;  
    pthread_t * clientServers;  
} Clients;
```

Aquest s'encarrega de controlar tot el relacionat amb els `threads` dels clients. creat procediment anomenat `CONNECTION_connectServer`, el qual li passem l'adreça `ip` i el port, i creem un socket. Si ens retorna un valor major a 0, sabem que ha estat creat satisfactòriament. Tot seguit, definim els paràmetres de la connexió:

```
sAddrIn.sin_family = AF_INET;  
sAddrIn.sin_port = htons(port);  
sAddrIn.sin_addr.s_addr = inet_addr(ipAddress);
```

I a continuació intentem connectar el socket, que també ens ha de retornar un valor superior a 0. És amb aquest socket que podrem fer els `writes` i els `reads` corresponents en cada cas com a `FileDescriptor`.

Pel que fa a Lionel, l'hem hagut de crear de 0. Al actuar com a servidor dedicat (se li poden connectar varis telescopis a la vegada i ha de poder estar per tots), hem decidit implementar-ho mitjançant `threads`. És per això que, després d'obrir el fitxer de configuració i guardar degudament les dades en el struct `ConfigParams`, comencem a crear els `threads` de seguida per tenir un fil d'execució per a cada telescopi tal i com s'exposa a continuació.

Anant per parts, però, caldria començar amb la comunicació amb el servidor. Primerament, segueix el mateix procediment que en el cas del client McGruder, fins que s'han definit els paràmetres de la connexió. A partir d'aquí, es bindeja i retorna el corresponent FileDescriptor.

És un cop obtingut aquest, que creem el fil d'execució del servidor Lionel, el qual crea un thread que crida a la funció `CONNECTION_runServer` tot passant-li el fileDescriptor que acabem d'obtenir. El motiu d'aquest primer thread es fa per tal de poder controlar les interrupcions al sortir del programa.

```
Starting Telescope1.  
Connecting to Lionel...  
Connection ready.
```

Figura 3 Connexió McGruder-Lionel.

```
Starting Lionel.  
Waiting...  
Connection Lionel-Telescope1 ready.
```

Figura 4 Connexió Lionel-McGruder

Dins de la funció cridada per aquest, hi hem creat un bucle infinit encarregat d'anar acceptant nous clients que passen a formar part d'un nou fil d'execució mitjançant un altre thread. És aquest procediment, anomenat `CONNECTION_clients`, el que s'encarregarà de llegir i enviar les trames concretes per a cada client, mitjançant el fileDescriptor que li hem passat.

Pel que fa a l'estructura de Lionel, hem seguit la mateixa que McGruder a l'hora de la divisió de classes, tenint-ne una principal, una encarregada de la configuració a nivell de paràmetres i una altre de la connexió amb els clients.

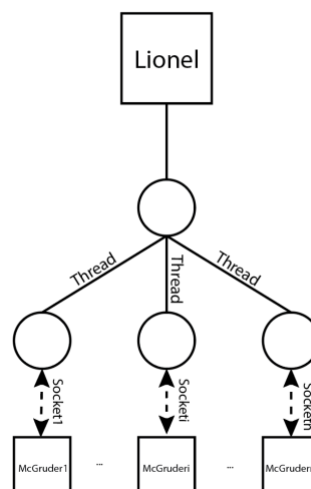


Figura 5 Diagrama de progrés de les comunicacions Lionel-McGruder

3.3. Fase 3

Quant a la fase 3, hem hagut d'implementar tot el protocol d'enviament de les dades. Per fer-ho, com que teníem problemes amb les peticions de memòria, utilitzem la funció `asprintf`, havent d'afegir també la següent macro:

```
#define _GNU_SOURCE
```

Aprofitant l'estructura ja creada a la fase 1, dins de la funció `CONNECTION_buildPacket`, la qual ja ens retorna l'struct amb el packet ple i li passem tots els paràmetres necessaris, utilitzem la funció `CONNECTION_sendPacket` per realitzar l'enviament d'aquest contingut.

Amb el file descriptor obtingut al connectar-se amb el servidor, va enviant write a write el contingut de packet cap al servidor:

```
write(clientFd, &packet->type, sizeof(char));  
write(clientFd, packet->header, strlen(packet->header)*sizeof(char));  
write(clientFd, &packet->length, sizeof(short));  
write(clientFd, packet->data, packet->length*sizeof(char));
```

Tot seguit aprofitem el mateix paquet pel qual tenim la memòria demanada per omplir-lo amb les dades de resposta del servidor. Això ho fem mitjançant la funció `CONNECTION_unbuildPacket` on, mitjançant reads anem guardant la informació rebuda dins del struct. És un cop rebut el contingut que l'analitzem per tal de revisar si el Lionel ens han enviat el "FILEOK" i, posteriorment el "CHECKOK".

És en aquest moment que per tal de no arrossegar possibles residus, netegem el contingut de l'estructura del packet amb la funció `CONNECTION_clearPacket`, donem la connexió per establerta i entrem dins del bucle de detecció i enviament de continguts dins de la carpeta "files".

Per fer-ho, analitzem el contingut de la carpeta i el posem a la llista dinàmica feta a la fase anterior i, en cas de no estar buida, anem bolcant fixter a fixter, n'obtenim l'extensió amb la funció `CONNECTION_getFileExtension` i obrim fitxer a fitxer, obtenint-ne el file descriptor. Un cop el tenim, mitjançant `lseek(fd, 0, SEEK_END)` obtenim el tamany total del fitxer, el qual dividim entre `MAX_FRAGMENT`, una constant que delimita el tamany màxim de data que podem passar per trama. Aquesta divisió la guardem tant en una constant entera com en un float, de manera que si veiem que la resta entre l'enter i el float és superior a 0, incrementem en 1 el resultat de l'enter, tenint així, el resultat de la divisió arrodonit a l'alça, on totes les porcions seran de `MAX_FRAGMENT` excepte la última.

Així doncs, enviem la metadata amb la informació obtinguda i entrem en un bucle on cada vegada que hi passem, anem actualitzant el tamany de memòria que anirem a utilitzar per omplir la data del packet (que com s'ha dit abans serà de `MAX_FRAGMENT` excepte la última) i també actualitzem la posició per començar a llegir del fitxer des d'on s'ha deixat en la darrera obertura. Tot seguit cridem a la funció

`CONNECTION_buildPacket` que omple l'estructura `packet` perquè mitjançant la funció `CONNECTION_sendPacket` se'ns envii el tros corresponent de trama.

Un cop sortits del bucle, s'envia la trama "ENDFILE" per avisar que s'ha acabat el fitxer. Prèviament, però, calculem el checksum d'aquest per poder-lo enviar juntament amb el header.

Al tornar al bucle principal, esperem a que passi el temps requerit pel fitxer .dat que li hem passat al iniciar el programa.

Pel que fa a Lionel, la cosa és una mica més complexa. Un cop generat el thread corresponent per al client que està connectant-se, ens preparam per rebre la seva trama de benvinguda. Això ho fem mitjançant la funció `CONNECTION_unbuildPacket` la qual funciona de la mateixa manera descrita anteriorment pel McGruder. Val a dir, però, que primerament cridem a una funció anomenada `CONNECTION_preparePacket` que demana memòria per als continguts del struct `packet`.

Com que volem que el contingut rebut en primera instància, ens perduri al llarg de totes les transaccions que vagi realitzant el client amb el servidor, hem creat un struct que s'encarrega de guardar aquestes dades les quals podrem anar consultant al llarg de l'execució.

```
typedef struct {
    char* fileOwner;
    char* fileName;
    long fileLength;
    long currentLength;
} FileProperties;
```

Com podem observar, l'estructura també conté el nom del fitxer, ja que només el rebem la primera vegada de l'enviament i durant les trames que rebem de contingut hem de poder-les associar a aquest. És per això que també ens guardem el tamany total i l'actual per tal de poder comprovar si l'hem rebut en la seva totalitat.

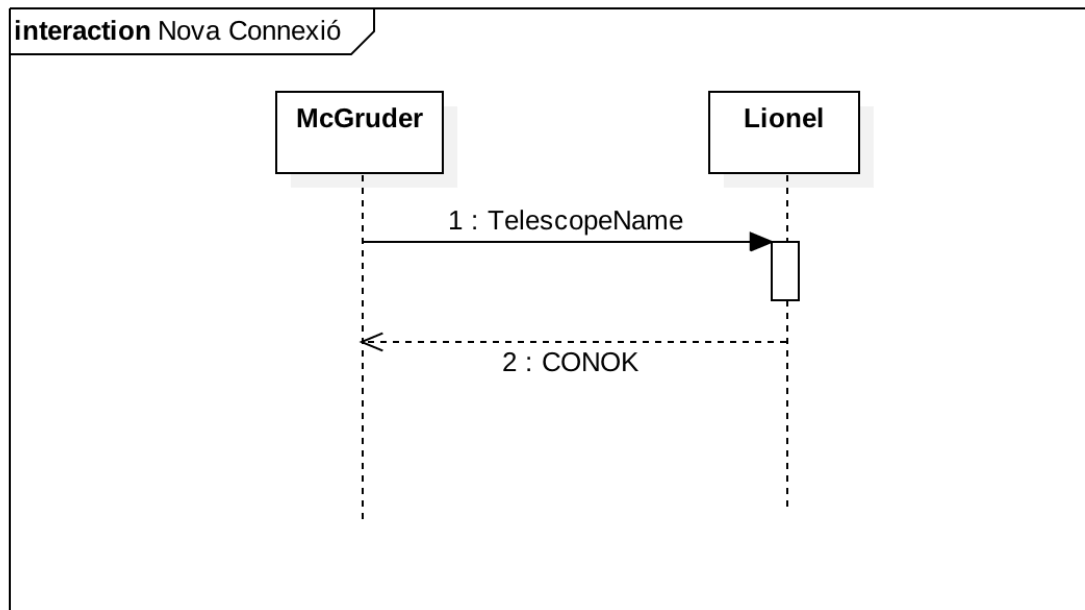


Figura 6 Comunicació de Nova Connexió entre McGruder i Lionel

Així doncs, un cop llegida la trama de nova connexió, d'on n'extraïem el nom del client, procedim a entrar en el bucle encarregat d'anar guardant el contingut rebut amb la funció `CONNECTION_unbuildPacket` comentada abans pel client, i tractant les dades mitjançant la funció `CONNECTION_managePacket`. Bàsicament consisteix en mirar quin tipus de trama és mitjançant un switch i segons el cas fer el que li correspon.

Pel que fa a una nova connexió, recopilem tota la informació que hem d'enviar com a resposta al client, la muntem dins l'estruct amb la funció `CONNECTION_buildPacket` i l'enviem mitjançant `CONNECTION_sendPacket`. Tal i com hem comentat també pel MCGruder, degut als problemes que hem tingut amb les peticions de memòria mitjançant `mallocs` i `reallocs`, hem utilitzat la funció `asprintf` per tal de solventar-ho.

El cas potser més complex a tractar és la gestió de la recepció dels fitxers. Primerament n'hem de comprovar el header, per tal de discernir entre si és metadata, dades o final de fitxer i a partir d'aquí emprendre les accions corresponents amb cada cas. Pel que fa a "METADATA", acabem d'omplir l'estruct `fProperties` anomenat anteriorment. En el cas de rebre el header buit, que voldrà dir que es tracta de la dada o d'una porció d'aquesta, utilitzem la funció `CONNECTION_storeFilePortion`. Aquesta funció, a partir del path, crea o afegeix (depenent de si existeix el fitxer o no), les dades noves al final del fitxer. Mentre estem en aquest cas, també ens encarreguem d'anar mostrant per pantalla la proporció d'informació rebuda. Això ho podem fer gràcies a l'estruct comentat anteriorment, el qual ens facilita el tamany actual i l'aconseguit fins ara, de manera que només ens cal fer una regla de 3, i com que va arrodonint, comprovar que no ens surti dos cops per pantalla el mateix percentatge.

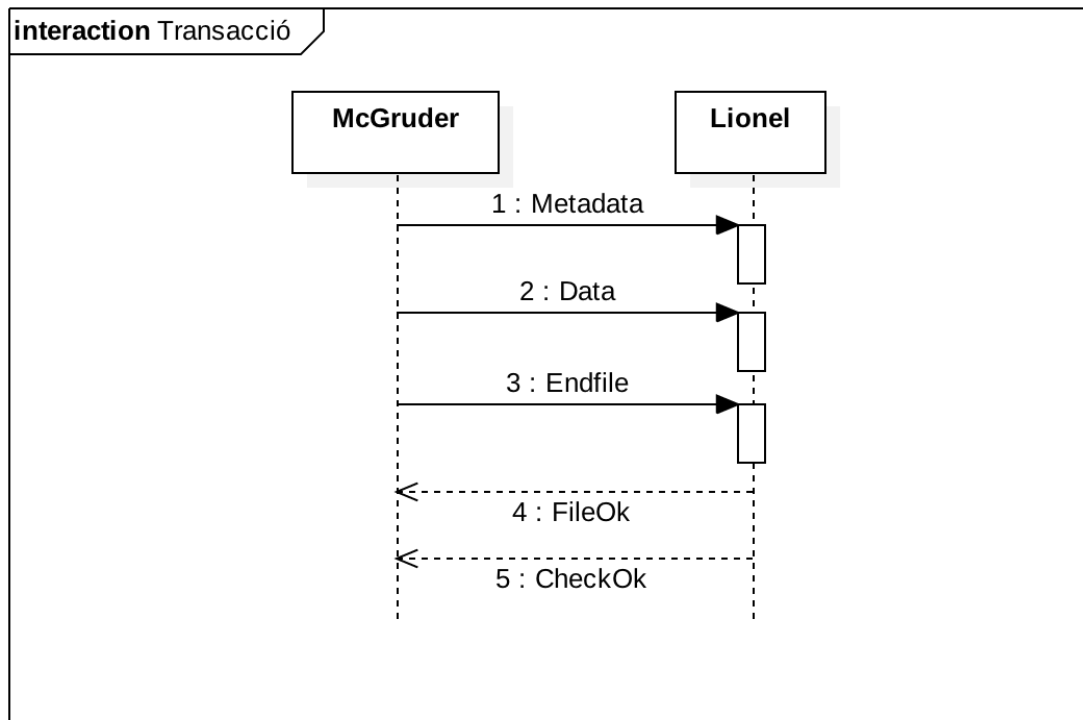


Figura 7 Comunicació de Transacció entre McGruder i Lionel

Finalment, si es tracta del header “ENDFILE”, comprovem, a partir del struct fProperties, si el tamany actual ha arribat a ser el mateix que el total i enviem el feedback en cada cas. En cas ko, reenviem el fitxer i en cas ok, comprovem que el contingut sigui correcte mitjançant el checksum. Per tal d’aplicar-lo, tenim la funció `CONNECTION_computeChecksum` la qual fa una crida al sistema a una comanda unix que retorna el checksum associat al fitxer que li hem passat per paràmetre a la funció. Amb aquest i el que rebem de McGruder comparem el resultat i així sabem si ho hem rebut bé. Un cop sabem que l’enviament ha estat satisfactori, mirem quin tipus de dada hem rebut i segons si és imatge o text ho guardem a l’struct corresponent, parsejant el contingut en cada cas.

Pel que fa a la desconnexió, globalment amb Lionel i McGruder, el que fem és que McGruder envii a Lionel la trama de petició de desconnexió. En el cas de que quan es demani s’hagi acabat d’enviar la dada o no s’estigui enviant res, Lionel retorna un “CONOK” però en cas contrari, el que fem és esperar que s’hagi acabat d’enviar per poder donar el feedback positiu. El control de gestió del client, el tenim mitjançant una funció que controla la senyal `isRunning`, a la `CONNECTION_controlDesconnexio`, la qual revisa en quin moment s’ha premut “ctrl + C” i gestiona si “autoritza” la desconnexió o no depenent del feedback del Lionel. És a dir, fins que Lionel no doni l’okay, per molt que es premi “ctrl+C”, McGruder no podrà finalitzar. Pel que fa a Lionel, un cop ha enviat l’autorització de desconnexió, aquest també tanca el socket amb el client que l’ha sol·licitada.

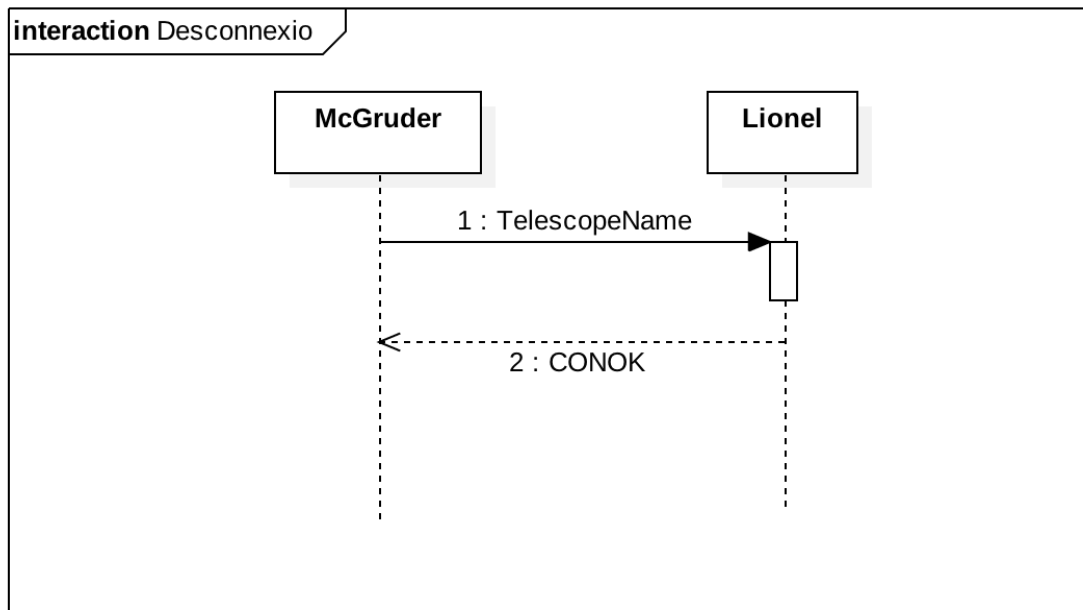


Figura 8 Comunicació de Desconnexió entre McGruder i Lionel

3.4. Fase 4

Aquesta és la última fase obligatòria. Ens ha fet replantejar l'estructura original que teníem funcionant en les altres fases. Primerament, hem plantejat un fork en Lionel, per tal de fer córrer Paquita a la vegada que Lionel seguia fent els processos anteriors. És per això que, un cop aconseguida la connexió amb McGruder, cridem a la funció `executePaquita(int * isRunning)`, la qual s'encarrega d'executar el fork. Pel que fa al pare, tenim a la crida de la funció creada anteriorment per executar Lionel. És aquesta crida la que, com s'ha vist en les fases anteriors, s'encarrega de gestionar el thread que crea el servidor dedicat, que a la seva vegada crea un thread per a cada client mitjançant la funció `CONNECTION_clients` tal i com s'ha vist.

La novetat en aquest cas, és de cara a la comunicació entre Paquita i Lionel. Al ser un fork, hem utilitzat cues de missatges per tal d'interactuar. Per començar, abans del fork, creem la cua de missatges així com dues llistes, una per a imatges i una altre per a fitxers. Seguidament, fem el fork i inicialitzem la cua de missatges. És llavors quan ja podem reinicialitzar les estadístiques i començar a escoltar si rebem alguna cosa mitjançant `paquitaExecution(idPaquita, &paquitaRunning, &statistics)`. Aquesta crea una llista i mentre “paquitaRunning” està actiu, va llegint i responent missatges de Lionel.

Pel que fa a Lionel, de cara a poder gestionar múltiples possibles clients McGruder que es connectin i enviïn informació a Paquita, utilitzem un mutex que implementem a la funció `CONNECTION_manageEndOfFilePacket`, encarregada de gestionar la part final del paquet.

De cara a les estructures utilitzades per enviar informació entre Paquita i Lionel hem utilitzat les següents:

```
typedef struct {
    char code[CODE_MAX];           // Codi de la constel·lació
    float density;                  // Densitat de la constel·lació.
    float magnitude;                // Magnitud de la constel·lació.
} ConstellationsMessagePaquita;

typedef struct {
    int imageFiles;                 // Número d'imatges rebudes.
    int textFiles;                  // Número de fitxers de text rebuts.
    double totalSize;               // Tamany total de tots els fitxers (en KB).
    float averageConstellations;    // Mitjana constel·lacions /num.fitxers.
    int constellationsNumber;        // Número total de constel·lacions.
    float averageDensity;           // Mitjana densitats constel·lacions.
    float maxMagnitude;              // Magnitud màxima de les constel·lacions.
    float minMagnitude;             // Magnitud mínima de les constel·lacions.
} StatisticsPaquita;

typedef struct {
    int queueType;                  // Identificador del tipus de cua.
    int killPaquita;               // Senyal per matar a Paquita.
    FilesMessagePaquita fMessage;  // Informació dels fitxers.
    ConstellationsMessagePaquita cMessage; // Informació de les constel·lacions.
    StatisticsPaquita statistics;   // Resultat
} MessagePaquita;
```

En quant al darrer struct, la variable killPaquita l'utilitzem per saber quan ha mort el pare (Lionel) i Paquita s'ha de "suïcidar". És aquesta variable la que, un cop activada, desactiva el "isRunning" encarregat de mantenir actiu el while de Paquita. Un cop fora del bucle, destruïm la llista, alliberem la memòria i destruïm la cua de missatges.

4. Proves realitzades

4.1. Fase 1

1. El primer que s'ha comprovat és que McGruder llegís correctament el seu fitxer de configuració.
2. A continuació, que pogués detectar tots els fitxers continguts dins del directori files.
3. Després, s'ha mirat com gestionar l'obtenció dels noms dels fitxers a l'hora d'emmagatzemar-los i administrar-los.
4. S'ha controlat si el temps especificat al fitxer de configuració s'aplicava correctament entre llegida i llegida del directori.
5. Finalment, s'ha comprovat que els signals interruptius funcionaven bé.

4.2. Fase 2

1. S'ha comprovat que Lionel llegís correctament el fitxer de configuració.
2. S'ha comprovat que els threads funcionaven bé.
3. S'ha comprovat que els signals interruptius funcionaven bé.

4. Tot seguit s'ha comprovat la connexió entre client-servidor funcionava correctament.

4.3. Fase 3

1. S'ha comprovat la correcta classificació segons tipus.
2. S'ha mirat si les respostes en base les premisses eren les correctes.
3. S'ha comprovat si el checksum era correcte.
4. S'ha mirat si el percentatge augmentava com era requerit al rebre un fitxer.
5. S'ha mirat que el kakun.txt guardés correctament l'històric de dades rebudes.

4.4. Fase 4

1. El primer que hem comprovat ha estat la correcta comunicació de les cues de missatges entre Lionel i Paquita.
2. S'ha mirat si a cada actualització el contingut rebut canviava.
3. S'ha comprovat que els indicadors fossin correctes.
4. S'ha mirat que el darrer fitxer de dades astronòmiques rebí el conjunt de paràmetres actualitzats demanats.
5. S'ha mirat el correcte funcionament del semàfor per tal de garantir l'exclusió mútua.
6. Finalment, s'ha revisat si la finalització de processos en moments clau alterava l'execució del procés principal. (matant Paquita, per exemple).

5. Problemes observats i com s'han solucionat

Al llarg de la realització d'aquesta pràctica i, principalment, a causa de la seva gran dimensió, ens hem trobat amb un nombre elevat de problemes. Molts d'ells eren senzills com ara un ';', un '*' o un '&', però d'altres eren més significatius i afectaven al funcionament global de la pràctica. A continuació, detallem alguns dels errors més significatius amb els que ens hem trobat:

1. Al demanar memòria dinàmica mitjançant mallocs, reallocs i frees, sense cap motiu aparent, a partir de la desena petició aproximadament, ens petava aquest amb errors tals com:

```
realloc(): invalid next size
```

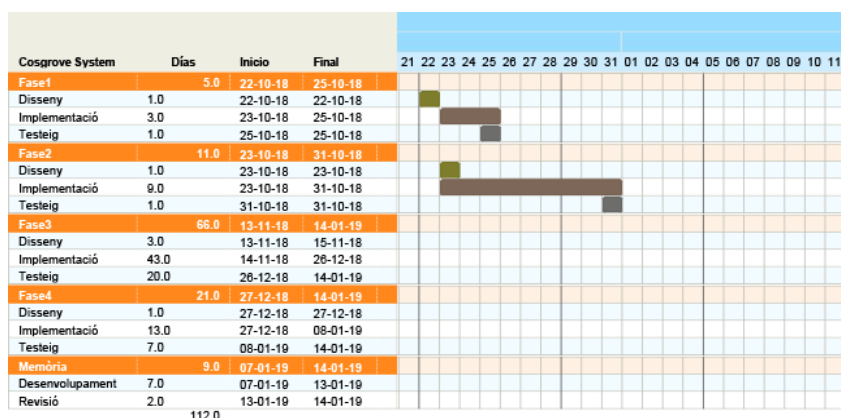
Val a dir que tot i haver anat a parlar amb profes i becaris no ho hem pogut aconseguir solventar i hem tirat pel alternatives com ara el asprintf o demanar un malloc general per un conjunt d'accions.

Ha estat principalment aquest problema el que no ens ha deixat avançar al ritme que voldríem. Finalment, però, hem descobert que executant el programa amb el debuggador valgrind s'ha pogut solucionar el problema.

2. Un altre problema que ens ha dut molts mal de caps i que ens ha fet enrederir bastant ha estat el de l'enviament de les trames. Malgrat haver fet mil comprovacions, de les quals totes eren satisfactòries, durant l'enviament de les trames, entre canvi i canvi de fitxer, en algun moment es descompensava el control del contingut i en el que creiem que era header llegíem data. La solució al final ha estat que en lloc d'enviar-ho tot de cop, ho hem anat fent paràmetre a paràmetre.
3. Un fet que també ens ha fet perdre temps, i bàsicament va estar per no entendre bé l'enunciat, va ser el fet de creure que a l'enviar la metadata del fixter, en lloc d'agafar i enviar el nom tal qual, havíem d'aconseguir la data de creació d'aquest i formatarla com a nou nom.
4. El problema que ens ha portat més incògnita recentment ha estat el misteri de les imatges. Aquestes s'enviaven seguint el mateix protocol que els textos, a diferència de que aquests últims tenien un checksum igual que en McGruder mentre que en les imatges no. Pel que feia al tamany en bytes, el de les imatges coincidia amb l'origen però malgrat això, no donava el checksum adequat. Al final hem descobert que l'error es tractava de que enviàvem el contingut de les dades amb char* en lloc de unsigned char*. Degut a caràcters que té la imatge ASCII especials, no els devíem agafat bé amb el format char. Tot i així, al descobrir el problema ens n'hem pogut ensortir.
5. Un altre problema que ens ha fet anar de bòlid, ha estat el de la gestió de memòria residual que quedava al matar al pare del fork sense matar al fill. La implementació final que vam fer va ser mitjançant signals. Al veure que no es podien utilitzar ho hem hagut de replantejar passant l'estat mitjançant l'estruct de la cua de missatges.

6. Estimació temporal

Per tal de poder veure de forma més ràpida i clara el temps invertit en la realització del projecte s'ha optat per realitzar el següent diagrama de Gantt.



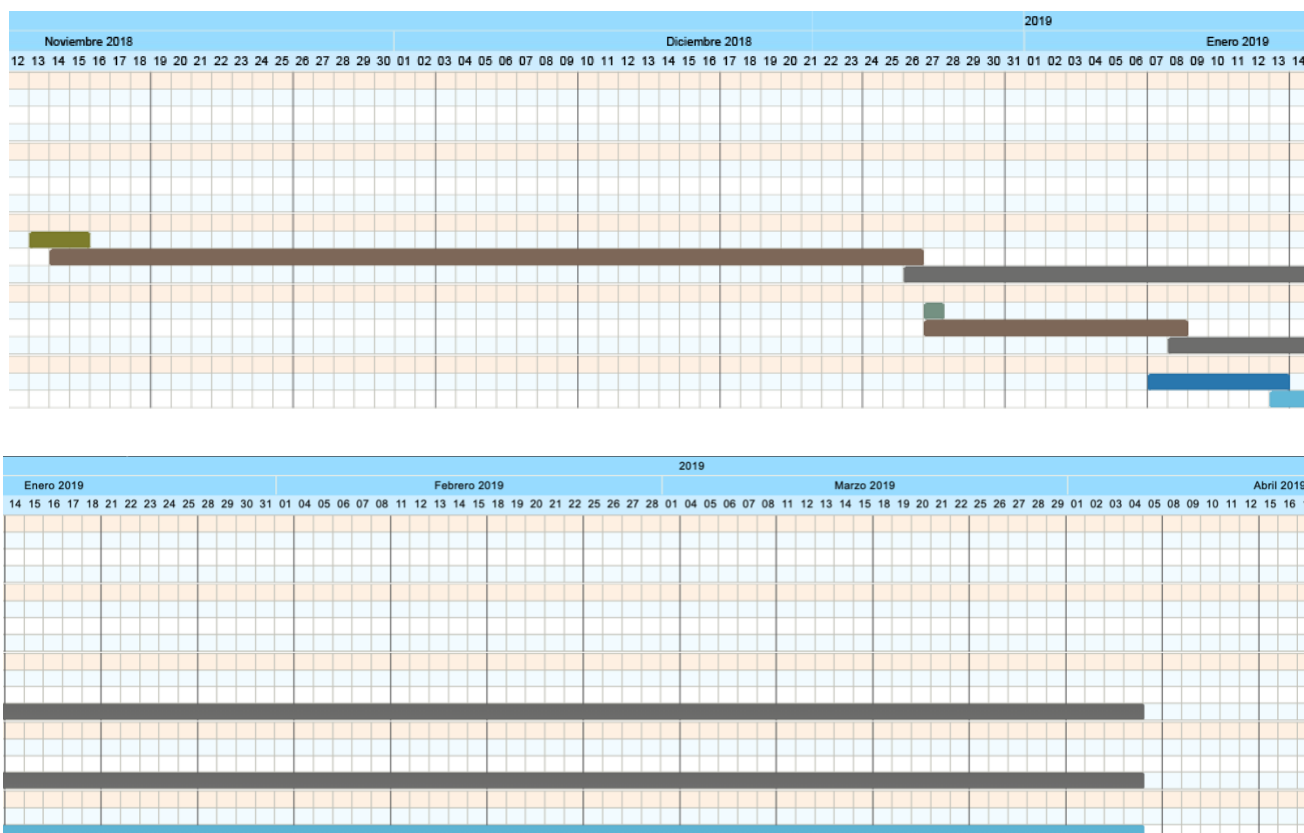


Figura 9. Diagrama de Gantt

Tal i com podem observar, la fase que ens ha dut més temps amb diferència ha estat la 4. Circumstàncies personals per part dels dos integrants del grup, ens han fet desquadrar-ho fins al punt de que ens ha destarotat tot el timing que portàvem.

A part d'això, però, tal i com hem comentat anteriorment, un problema que ens ha enrederit força ha estat l'enviament de fitxers, ja que sense motiu aparent, es desquadraven les trames durant el trajecte del client al servidor. Un altre problema, també ja esmentat, i possiblement el més problemàtic, ha estat que la pràctica tant sols funciona si utilitzem valgrind. Pel que fa a la resta de dates, les hem pogut anar respectant.

7. Conclusions

La realització d'aquesta pràctica de l'assignatura Sistemes Operatius ha servit per entendre i poder aplicar tot el que hem anat adquirint al llarg del curs, tant a classe com a les sessions de laboratori.

Ha estat a través de les hores invertides en aquesta pràctica que hem pogut veure com, a mesura que avançaven les fases, cada cop dominàvem més el que fèiem. Hem pogut presenciar el nostre aprenentatge i aquest camí ha resultat molt satisfactori malgrat els problemes que ens hem anat trobant durant el desenvolupament d'aquesta, els quals hem anat comentant.

D'altra banda, malgrat els patits, les sessions de laboratori han estat una eina que ens ha facilitat l'aprenentatge mitjançant la resolució de problemes curts en un temps limitat. Podem afirmar que, gràcies a les sessions, la pràctica semblava més resoluble i ens l'han fet molt més fàcil d'assolir.

Val a dir que per resoldre certs punts d'aquesta, ens ha anat bé poder recórrer a alguna de les sessions ja fetes per tal de recordar el funcionament d'algun dels mecanismes apresos. El fet de poder aplicar la teoria ens dona una idea molt més gran de la magnitud que tenen totes les eines adquirides al llarg de l'assignatura.

Finalment, esmentar que el fet d'haver fet la pràctica en grup, ens ha servit per veure diferents punts de vista a l'hora d'estructurar i arribar a fins comuns mitjançant camins diversos, que a vegades feien adonar-se de noves maneres d'entendre'l. El fet d'haver d'adaptar-se a codi que un per si mateix hauria fet seguint altres directrius, ens ha suposat un repte enriquidor de cara al que ens trobarem quan sortim al món laboral.

8. Bibliografia

- [1] "pthread_cancel(3) - Linux manual page", *Man7.org*, 2018. [Online]. Available: http://man7.org/linux/man-pages/man3/pthread_cancel.3.html. [Accessed: 08- Jan- 2019].
- [2] "socket(7) - Linux manual page", *Man7.org*, 2018. [Online]. Available: <http://man7.org/linux/man-pages/man7/socket.7.html>. [Accessed: 08- Jan- 2019].
- [3] "<netinet/in.h>", *Pubs.opengroup.org*, 2018. [Online]. Available: <http://pubs.opengroup.org/onlinepubs/000095399/basedefs/netinet/in.h.html>. [Accessed: 08- Jan- 2019].
- [4] "struct sockaddr_in, struct in_addr", *Gta.ufrj.br*, 2018. [Online]. Available: https://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html. [Accessed: 08- Jan- 2019].
- [5] "inet(3) - Linux manual page", *Man7.org*, 2018. [Online]. Available: http://man7.org/linux/man-pages/man3/inet_addr.3.html. [Accessed: 08- Jan- 2019].
- [6] "pthread_mutex_trylock(3): lock/unlock mutex - Linux man page", *Linux.die.net*, 2018. [Online]. Available: https://linux.die.net/man/3/pthread_mutex_trylock. [Accessed: 08- Jan- 2019].
- [7] "signal(7) - Linux manual page", *Man7.org*, 2018. [Online]. Available: <http://man7.org/linux/man-pages/man7/signal.7.html>. [Accessed: 08- Jan- 2019].
- [8] "utoa – IBM Knowledge Center", *IBM.com* [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zo.s.v2r3.bpxbd00/utoa.htm [Accessed: 08- Jan- 2019].

9. Annex

9.1. Protocol de comunicació general

L'objectiu principal d'un protocol de comunicació és que emissor i receptor es puguin entendre. Per a fer-ho, s'ha definit un únic tipus de trama, que pot tenir mida variable. Aquesta trama, està formada pels 4 camps següents:

- **TYPE:** Està format per 1 caràcter en format hexadecimal i que contindrà un identificador de la trama. Aquest camp ens descriu quin tipus de trama estem enviant.
- **HEADER:** Sempre estarà envoltat per claudàtors ([i]). La mida serà variable, però com a mínim el camp header serà: "[]".
- **LENGTH:** Està format per 2 bytes. Aquest camp ens informa de la llargada del camp DATA.
- **DATA:** Té una longitud variable (la longitud que ens indiqui el camp LENGHT). S'utilitza per a emmagatzemar valor o dades que ha d'enviar la trama.

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

Figura 10: Esquema format de la trama

9.2. Protocol de connexió de McGruder amb el servidor Lionel

En aquest apartat, s'explicarà el protocol que s'ha utilitzat per a la connexió entre Picard i Data.

9.2.1. Nova connexió

McGruder → Lionel

McGruder envia una trama per presentar-se i demanar una connexió.

- TYPE: 0x01
- HEADER.: []
- LENGTH: Llargària
- TELESCOPE name
- DATA: [TELESCOPE_name]

Lionel → McGruder

Trama OK connexió.

- TYPE: 0x01
- HEADER.: [CONOK]
- LENGHT: 0

-
- DATA: Empty

Trama KO connexió.

- TYPE: 0x01
- HEADER.: [CONKO]
- LENGHT: 0
- DATA: Empty

9.2.2. Desconnexió

McGruder → Lionel

Trama per notificar a Lionel d'una desconnexió.

- TYPE: 0x02
- HEADER.: []
- LENGTH: Llargària
- TELESCOPE name
- DATA: [TELESCOPE_name]

Lionel → McGruder

Trama OK desconnexió.

- TYPE: 0x02
- HEADER.: [CONOK]
- LENGHT: 0
- DATA: Empty

Trama KO desconnexió.

- TYPE: 0x02
- HEADER.: [CONKO]
- LENGHT: 0
- DATA: Empty

9.2.3. Enviar fitxer

McGruder → Lionel

Trama per indicar l'enviament de fitxers procedents dels McGruder i la metadata corresponent al fitxer.

- TYPE: 0x03
- HEADER.: [METADATA]
- LENGHT: Llargada camp DATA
- DATA: [tipus_de_fitxer&mida_del_fitxer&data_creacio_fitxer]

Trama per enviar les dades del fitxer corresponent.

- TYPE: 0x03
- HEADER.: []
- LENGHT: Llargada camp DATA
- DATA: [dades_del_fitxer]

Trama per indicar el final de la transmissió de dades i enviar el checksum corresponent.

- TYPE: 0x03
- HEADER.: [ENDFILE]
- LENGHT: Llargada camp DATA
- DATA: [checksum] Lionel->McGruder

Trama OK de transmissió d'un fitxer de text.



- TYPE: 0x03
- HEADER.: [FILEOK]
- LENGHT: 0 ▪ DATA: Empty

Trama KO de transmissió d'un fitxer de text.


- TYPE: 0x03
- HEADER.: [FILEKO]
- LENGHT: 0
- DATA: Empty

9.3. Distribució dels fitxers de la nostra aplicació















La nostra pràctica està estructurada en les següents 2 carpetes:

-  Lionel: carpeta on trobem tots els fitxer referents al procés Lionel amb Paquita.
-  McGruder: carpeta on trobem tots els fitxers referents al procés McGruder.

A més a més, a la carpeta principal disposem dels següents fitxers:

-  Makefile: fitxer makefile que permet compilar tots tres processos junts escrivint la comanda `make`, compilar-los per separat escrivint `make <nom_proces>` així com també esborrar tots els arxius compilables relacionats amb els diferents mòduls.

A continuació detallem els fitxers que es troben a cadascuna de les 2 carpetes esmentades:

-  Lionel
 -  ConfigLionel.dat: fitxer de configuració de Lionel.
 -  Lionel.c: és el fitxer principal i conté el `main` del procés Lionel. És el que també conté les rutines de servei d'interrupció del signal `SIGINT`.
 -  Lionel_Configuration.c/.h: és el fitxer que conté totes les funcions relacionades amb el fitxer de configuració.
 -  Lionel_Connection.c/.h: és el fitxer que conté totes les funcions relacionades amb la comunicació entre Lionel i McGruder.
 -  Lionel_Constellations_List.c/.h: és el fitxer que conté totes les funcions relacionades amb la llista de constel·lacions.
 -  Lionel_List.c/.h: és el fitxer que conté totes les funcions relacionades amb la llista encarregada de crear llistes de fitxers.
 -  Paquita.c: És el fitxer principal i conté el `main` del procés Paquita.
 -  Paquita_List.c/.h: és el fitxer que conté les estructures de les llistes on guardem les dades necessàries per realitzar la seva tasca.
-  McGruder
 -  ConfigT1.dat: fitxer de configuració de McGruder.
 -  McGruder.c: és el fitxer principal i conté el `main` del procés McGruder. És el que també conté les rutines de servei d'interrupció dels signal `SIGINT`.
 -  McGruder_Configuration.c/.h: és el fitxer que conté totes les funcions relacionades amb el fitxer de configuració.
 -  McGruder_Connection.c/.h: és el fitxer que conté totes les funcions relacionades amb la comunicació entre McGruder i Lionel.