Overlook System

Maria Aguilà (maria.aguilà)
Xavier Cavaller Allés (xavier.cavaller)

1. Introducció:

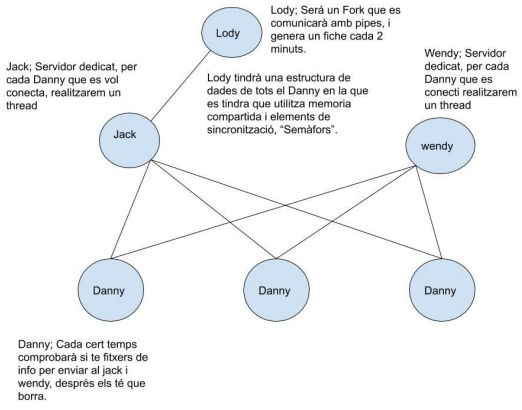
En aquesta pràctica de l'assignatura de sistemes operatius de la universitat de la Salle, s'ha demanat implementar un control d'estacions meteorològiques remotes centralitzades amb uns servidors, els quals recopilen tota la informació transmesa per les estacions remotes.

Aquestes estacions seran les encarregades de llegir la informació que els instruments meteorològics vagin introduint a un fitxer, aquestes dades mostraran l'hora, data, temperatura, humitat, precipitació i humitat, també hi haurà imatges en format jpg captades per una càmera, una vegada l'estació faci la lectura d'una de les informacions esborrarà el fitxer i enviarà via socket la informació al servidor corresponent, si la informació llegida són dades se li enviarà al servidor Jack i si es una imatge l'enviament es produirà al servidor wendy.

Els servidors recaptaran la informació i la gestionaran de la forma que explicita l'enunciat, tindrem dos servidors un que s'encarrega de les dades i un altre que serà l'encarregat de guardar les imatges.

2. Predisseny:

En la següent imatge es pot veure un esbós de que serà la pràctica en la seva totalitat.



Imatge 1: Predisseny

La interacció d'un usuari amb tot el sistema serà mínima, per aquest motiu s'ha desdit fer una bateria de missatges específics de cada procés i de cada bloc de codi per informar del que està succeint en tot moment.

3. Disseny:

1. Danny.

- Introducció a Danny.

En aquesta part s'ha començat a programar el codi que controla les funcionalitats de l'estació meteorològica Danny, després de fer un predisseny de la pràctica en la seva totalitat destaquen un seguit de funcionalitats les quals enumerarem seguidament.

Les següents funcionalitats són les necessàries per implementar l'estació Danny.

1. Llegir fitxer: Aquest codi dotarà de tot el necessari per tractar amb els diferents fitxers que Danny ha de llegir, aquest hauria de ser capaç tant de llegir fitxers de txt com de jpg fitxers binaris.

   Els fitxers que llegirà Danny seran un de txt per realitzar la configuració adequada, un altre de txt on estan les dades a processar i també serà capaç de llegir un fitxer tipus de dades binarias.

2. Comunicació amb els servidors Jack i Wendy: Aquest apartat s'ha solucionat proporcionat dos blocs de codis, un d'aquest és el de connexió, que realitza tot el referit a obertura de sockets i protocols de connexió i enviament de dades. L'altre bloc ha de donar format al missatge, i realitza tot el referit al tractament d'aquest.

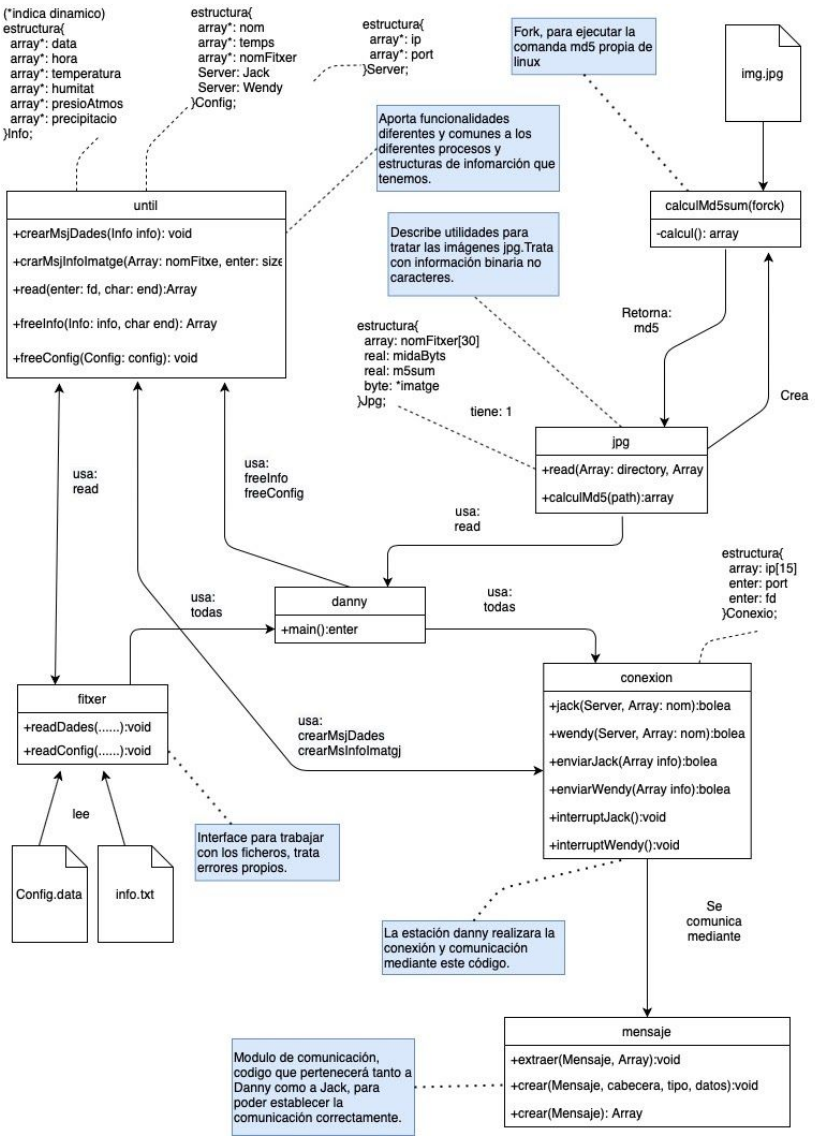
3. Tractament jpg: Proporciona funcionalitat particular per poder tractar correctament amb les dades binàries, també dotarà d'una funcionalitat particular que és la de calcular el checksum md5, que es realitza fent una crida a un executable extern al nostre programa.

4. Funcion generals i comuns: Aquest tipus de funcions se agrupen en un bloc de codi anomenat until, el qual proporciona diferents funcionalitats als diferents blocs de la pràctica.

5. Per acabar: Tenim, danny procés que s'encarrega de gestionar tots aquests blocs de funcions. Més endavant es tracta en més deteniment.

1. Diagrama general de Danny.

En la següent imatge es pot veure com es relacionen els diferents blocs del codi com també les estructures de dades en la qual es basen, les fletxes indiquen el flux de la informació i els quadres ens donen una idea de les funcions que poden ser cridades des d'altres procediments.



Imatge 2: Danny

2. Control Errors.

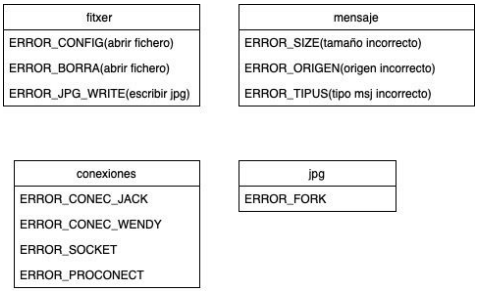


Imagen3: Control Errores Danny

- Estructures de dades usades.

Seguidament és veure quines estructures de dades s'han utilitzat per implementar Danny.

```
(*indica dinamico)
estructura{
  array*: data
  array*: hora
  array*: temperatura
  array*: humitat
  array*: presioAtmos
  array*: precipitacio
}Info;

estructura{
  array*: nom
  array*: temps
  array*: nomFitxer
  Server: Jack
  Server: Wendy
}Config;

estructura{
  array*: ip
  array*: port
}Server;

estructura{
  array: nomFitxer[30]
  real: midaByts
  real: m5sum
  byte: *imatge
}Jpg;

estructura{
  array: ip[15]
  enter: port
  enter: fd
}Conexio;

estructura{
  array de char: origen
  array de char: dades
  char: tipo
}Mensaje
```

Imagen 4: estructuras de dades Danny

"Info": Estructura en la qual es guardarà el llegit del fitxer de dades meteorològiques, aquestes dades es tractaran per poder ser enviades en el format requerit per l'enunciat de la pràctica.

"Config": Tot el necessari per establir les connexions com per poder llegir els fitxers de dades serà llegit d'un fitxer de text i introduït a aquesta estructura.

"Server": Informació necessària dels servidors per poder establir la connexió.

"Jpg": Tota informació necessària per poder tractar un fitxer jpg, aquí es pot destacar tant checksum md5 com el tamany en Bytes de la imatge, amb aquests dues variables es pot llegir el fitxer i comprovar que ha arribat correctament.

"Connexió": Informació referent a la connexió ja establerta amb un dels servidors.

"Mensaje": Estructura de dades en la qual es crea tot el necessari per tractar el missatge, aquest està format per una capçalera, d'un tipus de missatge i el contingut o dades.

- Recursos del sistema utilitzats, (signals, sockets, semàfors, pipes etc…).

Signals: s'ha utilitzat dos signals per implementar l'estació Danny, per un costat "sigAlarm", aquest signal s'ha configurat perquè cada cert temps determinat per la configuració del Dany faci una crida advertint al sistema de què pot comprovar si existeixen fitxers a la carpeta indicada.

L'altre signal reconfigurat és de "sigInt" o com tothom el coneix "Control + c", d'aquesta manera si l'usuari vol tancar el programa o podrà fer amb aquesta comanda que farà una criada a una subrutina per allibera i tancarà tots els recursos utilitzats a l'estació.

Socket, s'utilitza dos sockets per comunicar-se amb els dos servidors, l'encarregat de controlar tot el referent a aquest recurs del sistema és "connexió", aquí admès de tractar amb els protocols de comunicació és capaç d'obrir connexions amb els servidors.

També s'utilitza un forck per calcular el md5, el motiu pel qual s'utilitza aquesta eina és que necessiten executà un programa extern al nostre codi. Per comunicar el procés fork amb el procés principal s'utilitzen pipes.

2. Wendy:

- Introducció a Wendy.

Wendy és un programa que implementa un servidor dedicat, això implica que per cada estació connectada, Wendy crearà un fil d'execució el qual controla les comunicacions entre Danny i Wendy, aquest processos s'encarreguen de llegir i escriure en els "file descriptors" creades pels socket.

Aquest s'encarregaran de tractar amb imatges, aquest fitxer seran enviades per les estacions i el servidor verificarà i emmagatzemarà les imatges, el procés de verificació es realitzarà amb el md5 checksum i si són correctes es guardaran a una carpeta anomenada Barry. Les següents funcionalitats són les necessàries per implementar el servidor Wendy.

1. Llegir i escriure, fitxer: Aquest codi dotarà de tot el necessari per tractar amb els diferents fitxers que Wendy ha de llegir i escriure, fitxer de txt, per la configuració del servidor, i guardar imatge en format binari.

2. La Comunicació: Ha d'existir una comunicació constant amb les estacions connectades, aquest apartat s'ha solucionat implementat un servidor dedicat.

3. Tractament jpg: Proporciona funcionalitat particular per poder tractar correctament amb les dades binàries, també podrà calcular el checksum md5.

4. Función generals i comuns: Aquest tipus de funcions s'agrupan en un bloc de codi anomenat until, el qual proporciona de recursos als diferents blocs de la pràctica.

1. Diagrama general de Wendy.

En la següent imatge es pot veure com es relacionen els diferents blocs del codi com també les estructures de dades en la qual es basen, les fletxes indiquen el flux de la informació i els quadres ens donen una idea de les funcions que poden ser cridades des d'altres procediments.

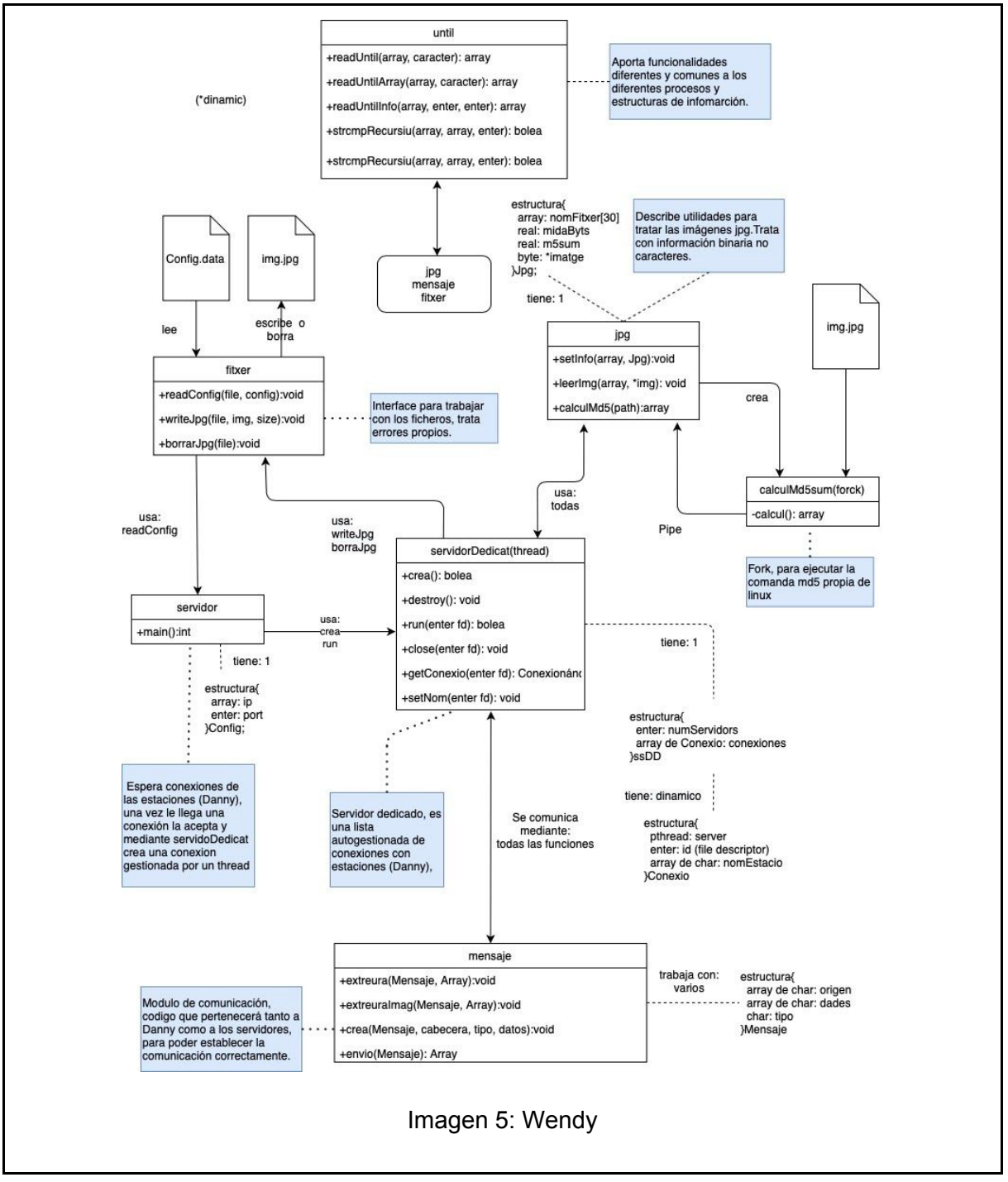


Imagen 5: Wendy

2. Control Errors.

En la següent imatge es pot veure el tractament dels errors que es fa a cada bloc de codi, cada un dels codis es fa el seu propi control d'errors, mostrant per pantalla quin és, i indicat al procés principal que ha succeït un error.

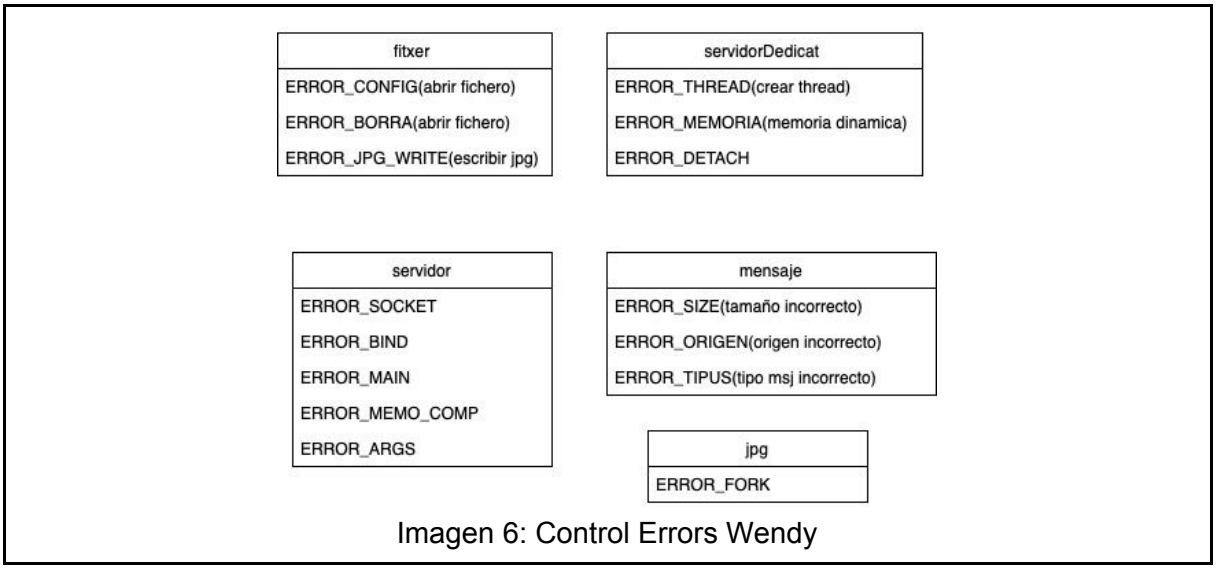


Imagen 6: Control Errors Wendy

- Estructures de dades usades.

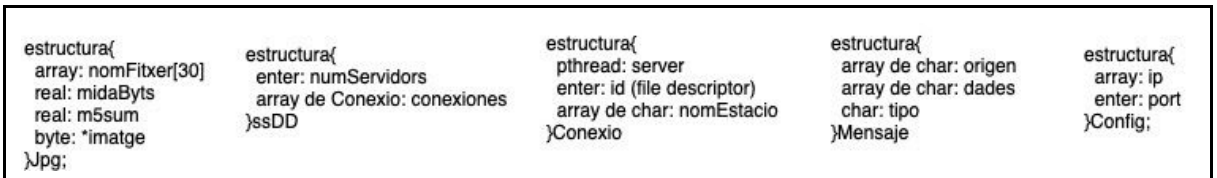


Imagen 7: estructuras de dades Wendy

"Jpg": Tota informació necessària per poder tractar un fitxer jpg, aquí es pot destacar tant checksum md5 com el tamany en Bytes de la imatge, amb aquestes dues variables es pot llegir el fitxer i comprovar que ha arribat correctament.

"ssDD": Aquesta estructura de dades descriu una llista de servidors dedicats, els servidors es guardaran de forma consecutiva i dinàmicament en memòria, si en algun moment és tancar algun s'esborrarà de memòria i es configura la llista per tornar a ser concatenat.

"Connexió": Element que conforma la llista de ssDD, aquest guarda tota la informació per mantenir la comunicació correctament.

"Mensaje": Estructura de dades en la qual es crea tot el necessari per tractar el missatge, aquest està format per una capçalera, d'un tipus de missatge i el contingut o dades.

- Recursos del sistema utilitzats, (signals, sockets, semàfors, pipes etc…).

Signals: s'ha utilitzat un signal, "sigInt" o "Ctr + c", aquest s'ha reconfigurat perquè quan l'usuari el generi el sistema tanqui el procediment alliberant tota la memòria i tancant tots els recursos oberts.

Sockets: Les connexions s'han establert mitjançant sockets, per tant es crearà un socket per cada estació que es vulgui connectar.

Semàfors: Es fan servir mutex, ja que cada un dels servidors dedicats és un thread, per tant a l'hora d'accedir a les regions de memòria compartida es fa servir aquesta eina, aquesta regió de memòria és la llista de servidors dedicats on es guarda tota la informació referent a la connexió.

Fork i pipe: S'utilitza aquests dos recursos igual que en el programa danny per poder calcular el checksum. El fork genera un fill del programa el qual executarà la comanda md5, aquesta retorna el checksum d'un fitxer, aquest array de caràcters es transmeti al procés principal mitjançant un pipe.

2. Jack:

- Introducció a Jack.

Jack és un servidor dedicat que tracta les dades meteorològiques enviades per les estacions Danny, això implica que per cada estació connectada i aura un fil d'execució creat amb un "thread" que controli les comunicacions.

1. Llegir i escriure fitxers: Aquest codi dotarà de tot el necessari per tractar amb els diferents fitxers que Jack té per llegir i escriure, fitxer de txt per la configuració del servidor, i escriure fitxer, per anar guardant cada ser temps les dades guardades.

2. La Comunicació: Ha d'existir una comunicació constant amb les estacions connectades. Aquest com en el servidor Wendy se solucionarà amb un servidor dedicat.

3. Funcions generals i comuns: Aquest tipus de funcions s'agrupen en un bloc de codi anomenat until, el qual proporciona de recursos als diferents blocs de la pràctica.

4. Mitjançant un fork i memòria compartida s'administra la info enviada per les estacions.

1. Diagrama general de Jack.

En la següent imatge es pot veure com es relacionen els diferents blocs del codi com també les estructures de dades en la qual es basen, les fletxes indiquen el flux de la informació i els quadres ens donen una idea de les funcions que poden ser cridades des d'altres procediments.

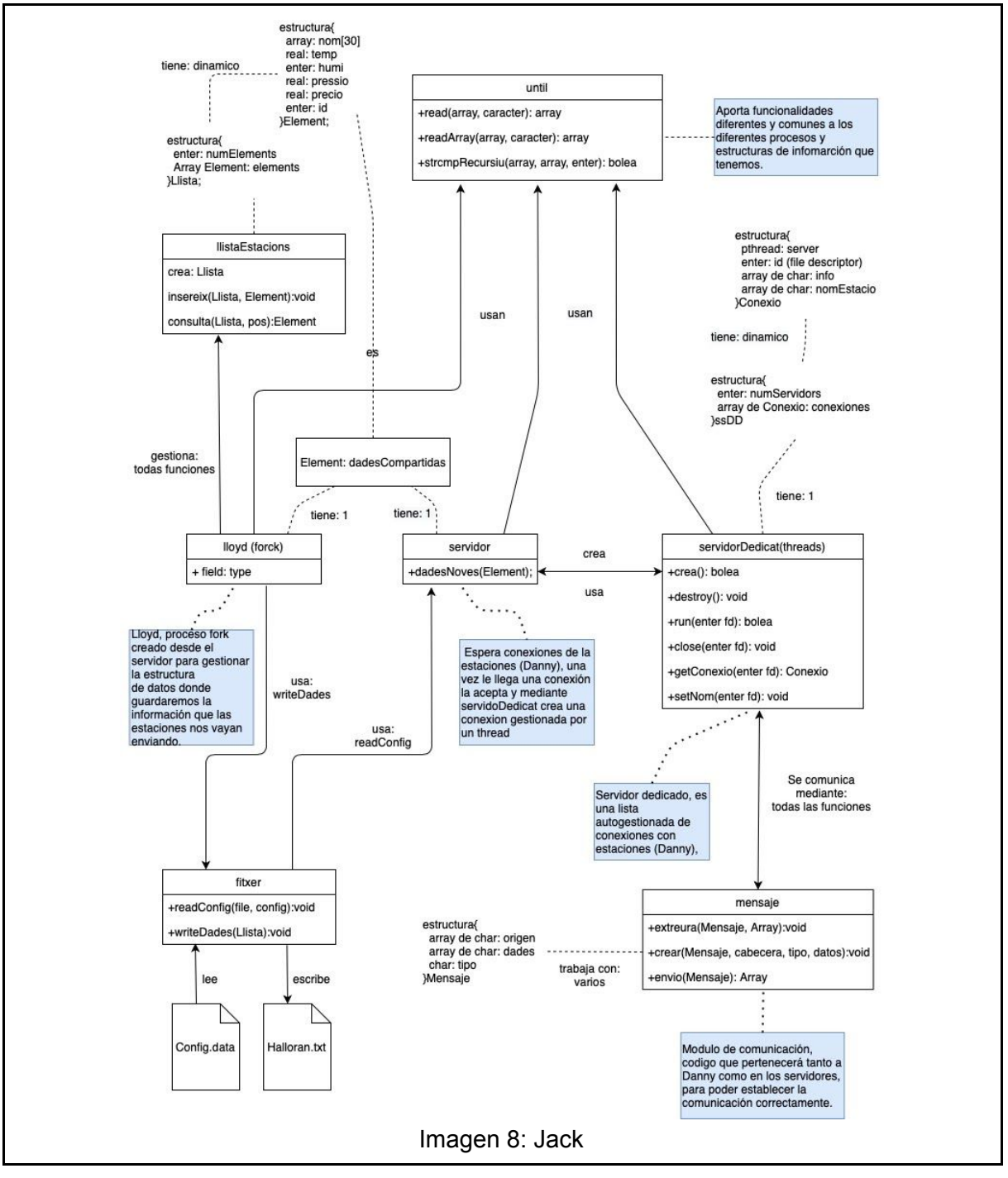


Imagen 8: Jack

2. Control Errors.

En la següent imatge es pot veure el tractament dels errors que es fa a cada bloc de codi, cada un del codis es fa el seu propi control d'errors, mostrant per pantalla quin és, i indicat al procés principal que ha succeït un error.

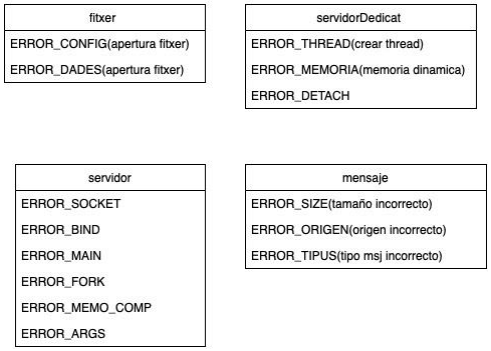


Imagen 9: Control Errors

- Estructures de dades usades.

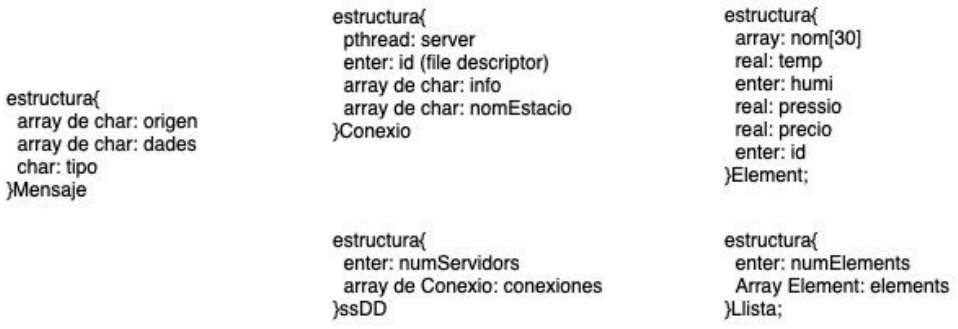


Imagen 10: Estructuras dades Jack

"Mensaje": Estructura de dades en la qual es crea tot el necessari per tractar el missatge, aquest està format per una capçalera, d'un tipus de missatge i el contingut o dades.

"ssDD" i "Connexió": Llista dinàmica de connexions establertes amb les estacions, aquest tindran tota la informació per gestionar tant els protocols de comunicació com els tractaments dels sockets.

"Llista" i "Element": Llista dinàmica d'informació transmesa per l'estació, guardarem en forma de mitjana les dades de cada estació per separat, per tant tindrem un element per cada una de les estacions que se connecten al llarg del temps al servidor, aquesta informació es va emmagatzemant a un fitxer cada cert temps. Lloyd és l'encarregat de gestionar aquest espai de memòria. (Lloyd, fil d'execució mitjançant un fork creat al programa principal de Jack.

- Recursos del sistema utilitzats, (signals, sockets, semàfors, pipes etc…).

Signals: se ha utilitzat un signal, "sigInt" o "Ctr + c", aquest se ha reconfigurat perquè quan l'usuari el generi el sistema tanqui el procediment alliberant tota la memòria i tancant tots el recursos oberts.

Sockets: Les connexions se han establert mitjançant sockets, per tant es creara un socket per cada estació que es vulgui conectar.

Semàfors: Es fan servir mutex, ja que cada un dels servidors dedicats és un thread, per tant a l'hora d'accedir a les regions de memòria compartida es fa servir aquesta eina, aquesta regió de memòria és la llista de servidors dedicats on es guarda tota la informació referent a la connexió, també es protegeix una regió de memòria que és la compartida entre el Lloyd procediment fill i el programa principal pare, Lloyd s'encarrega de gestionar les dades enviades per les estacions, guardar, processar i calcular les mitjanes. Aquesta regió de memòria també s'ha protegit mitjançant un semàfor de sincronització, per tant realitza una exclusió amb mutex i una sincronització amb semàfors.

2. Problemes observats i com s'han solucionat.

Un dels problemes en el que ens hem trobat és amb els protocols de comunicació, es va començar la pràctica sense crear un tipus abstracte de dades com "Missatge" per controlar les comunicacions, aquest fet va complicar molt els protocols de comunicació fent els codis llargs i poc llegibles. Una vegada introduït aquest TAD vàrem ser capaços de tractar els missatges d'una manera més eficient.

El tractament de les imatges ha estat tot un repta, ja que al principi es va intenta reutilitzar codi el qual tracta les dades del missatge com si fossin caràcters aquest fet implicava que ni Danny ni Jack tractessin les dades correctament, una vegada localitzat l'error es va recodificar les funcions per poder tractar dades binàries.

En la creació de forks per executar el programa md5sum per comprovar les imatges, ens vam trobar que el procés era correcte, però es quedava el procés obert després de tancar-lo. Vam veure que ens faltava un wait en el pare per tal que esperes el fill i poder seguir.

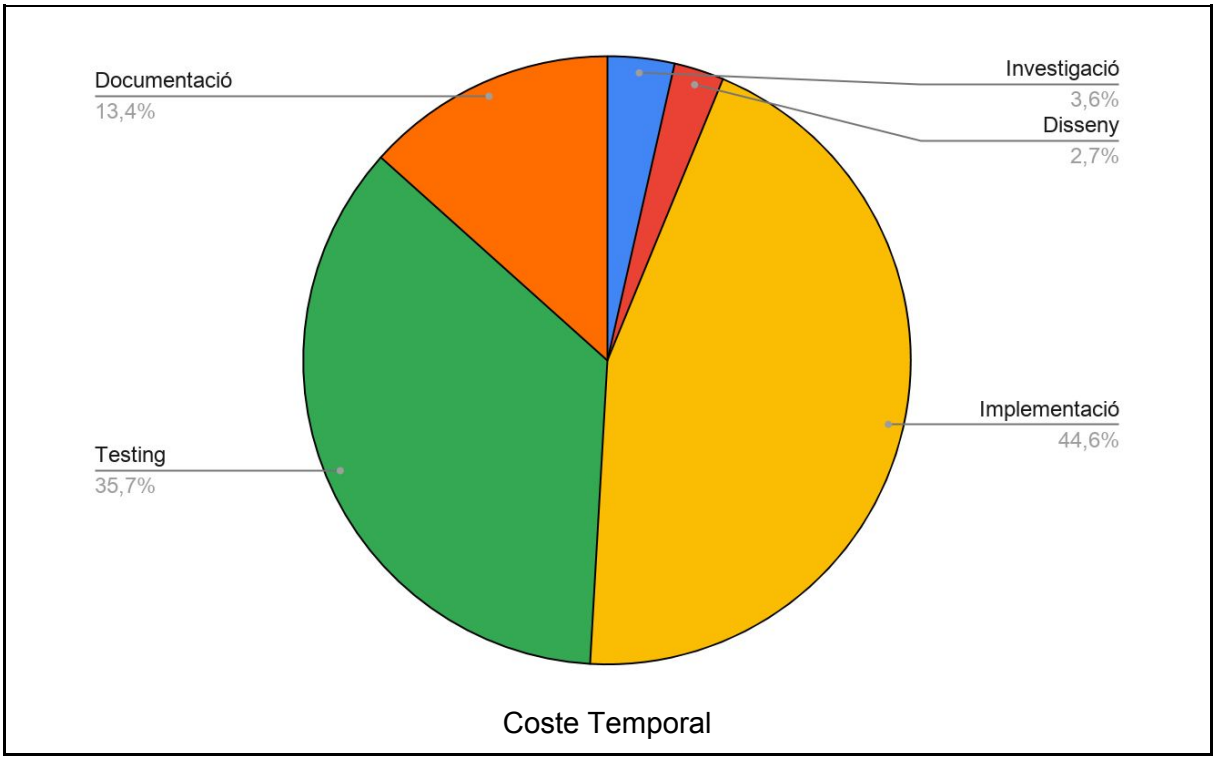
Per poder enviar la mida de la imatge, ens vam trobar que necessitàvem passar d'integrer a ascii, vam provar d'utilitzar la funció itoa(), però Montserrat no l'acceptava, també vam provar de fer-la nosaltres, però vam decantar per fer un sprintf que fa el mateix però és més senzill.

Per tal de saber si alliberàvem tota la memòria, ens va ser molt útil l'eina valgrind. El principi no enteníem com utilitzar-la o com trobar on es estava l'error. Una vegada après com funcionava ens va ser molt útil per trobar on exactament faltava algun free o on no alliberàvem tota l'estructura de dades.

3. Estimació temporal:

Per tal de realitzar la pràctica de manera eficient, de manera conjunta ens posàvem un dia, hi anàvem fent els dos a la vegada aportant idees i propostes de com encarar cada part.
Per tal de resoldre la pràctica vam fer un disseny general i un de cada part abans de començar a picar codi.

- Investigació: 4 hores
- Disseny: 3 hores
- Implementació: 50 hores
- Testing: 40 hores
- Documentació: 15 hores



Coste Temporal

4. Conclusions i propostes de millora.

Aquesta pràctica ha estat realitzada durat tot el semestre, ens ha ajudat a consolidar els conceptes vists de manera teòrica a classe i de manera pràctica durant les sessions de cada dimecres.

Hem pogut experimentar de primera mà com és treballar en un projecte de grans dimensions en llenguatge c i com desenvolupar aquest.

A nivell de codi, per tal que el projecte estigues ben estructurat, ho hem dividit en tres carpetes; Danny, Jack i Wendy, i en cadascuna hi ha els diferents fitxers organitzats segons la seva utilitat.

A nivell de coneixements, hem aplicat tot allò après a classe i en les sessions, també hem pogut descobrir funcionalitats com la lectura de directoris o executar programes a través d'un fork.
També hem pogut experimentar i treballar amb l'eina valgrind.
Aquesta eina ens ha ajudat a saber com alliberarem la memòria i l'ús que en fèiem d'aquesta.

Finalment, hem pogut realitzar la pràctica correctament tot i tenir petits problemes, els hem pogut resoldre i seguir endavant.
Propostes de millora, una memòria amb més detall del que s'està demanant, temps per realitzar la pràctica a clase.

5. Bibliografia utilitzada.

scandir(3) - Linux manual page
https://man7.org/linux/man-pages/man3/scandir.3.html

gcc error : undefined reference to `itoa'
https://stackoverflow.com/questions/12970439/gcc-error-undefined-reference-to-itoa

Process terminating with default action of signal 13 (SIGPIPE)
https://libevent-users.monkey.narkive.com/1EBKF7UJ/process-terminating-with-default-action-of-signal-13-sigpipe

md5sum of file in Linux C
https://stackoverflow.com/questions/3395690/md5sum-of-file-in-linux-c

cómo usar correctamente fork, exec, wait
https://stackoverrun.com/es/q/5210918

Procesos con exec, fork y wait en c
https://es.stackoverflow.com/questions/402894/procesos-con-exec-fork-y-wait-en-c

execvp(3): execute file - Linux man page
https://linux.die.net/man/3/execvp

Return value of execl
https://stackoverflow.com/questions/27676893/return-value-of-execl

C library function - sprintf() - Tutorialspoint
https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm

Warning: X may be used uninitialized in this function
https://stackoverflow.com/questions/12958931/warning-x-may-be-used-uninitialized-in-this-function

Invalid read of size 1 in strstr() using valgrind
https://stackoverflow.com/questions/10435736/invalid-read-of-size-1-in-strstr-using-valgrind

Kill Example (The GNU C Library)
https://www.gnu.org/software/libc/manual/html_node/Kill-Example.

C Library - - Tutorialspoint
https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

itoa - C++ Reference
http://www.cplusplus.com/reference/cstdlib/itoa/

How to use execvp()
https://stackoverflow.com/questions/27541910/how-to-use-execvp

execvp(3): execute file - Linux man page
https://linux.die.net/man/3/execvp

open (C System Call) - Code Wiki
http://codewiki.wikidot.com/c:system-calls:open

Comando cp de Linux. Un par de trucos útiles
https://victorhckinthefreeworld.com/2017/07/06/comando-cp-de-linux-un-par-de-trucos-utiles/

Invalid read of size 1 in strstr() using valgrind
https://stackoverflow.com/questions/10435736/invalid-read-of-size-1-in-strstr-using-valgrind

C library function - strstr() - Tutorialspoint
https://www.tutorialspoint.com/c_standard_library/c_function_strstr.htm