Sistemes Operatius

Curs 2022-2023

Eä System

Login	Nom
alex.cano	Alex Cano Gallego
marc.geremias	Marc Geremias Serra

Data 09/01/23

Index

1. Disseny

En aquesta pràctica de sistemes operatius, se'ns ha plantejat l'objectiu de crear un sistema de comunicació entre diferents Iluvatars per a que es puguin enviar informació i dades independentment de si estan en la mateixa terra o no.

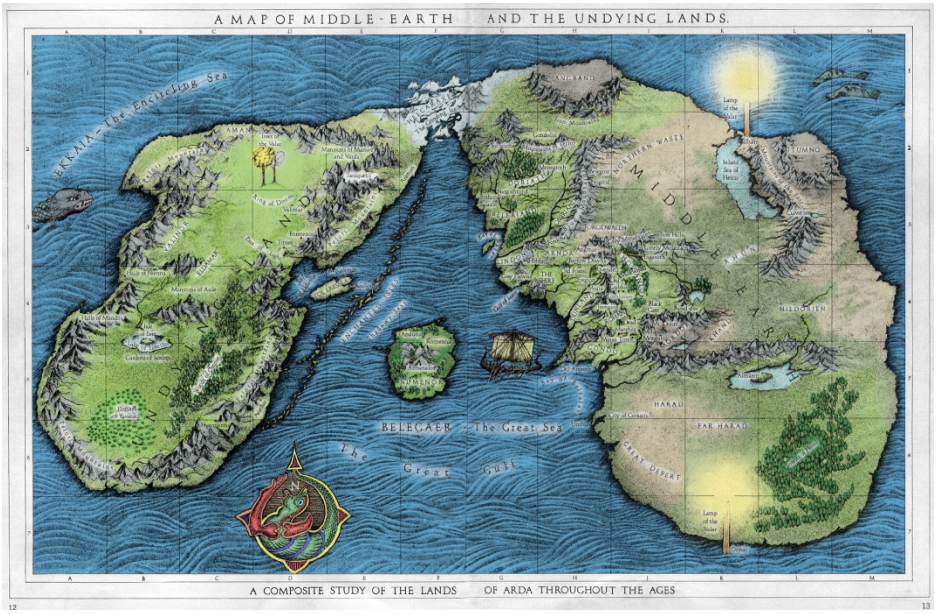
També, se'ns ha demanat la implementació d'un servidor central el qual es Arda, aquest s'encarregarà de connectar els diferents Iluvatars, enviar les llistes d'Iluvatars connectats en temps real, controlar les desconnexions dels Iluvatars i finalment un registre dels missatges que s'envien entre ells amb un comptador.

Per tal implementar aquesta pràctica s'ha dividit el gran problema en diversos petits dividint el projecte en diferents fases, a cada fase s'ha anat implementat noves funcionalitats necessàries per complir amb els requeriments de l'enunciat.

A més, per tal d'interconnectar els Iluvatars i Arda s'han utilitzat els servidor de lasalle, concretament Matagalls, Puigpedrós i Montserrat. Cada servidor és una maquina diferent i representa una terra diferent.

Seguidament, s'ha tingut en compte la bona modulació i arquitectura del projecte per tal de simplificar l'escalabilitat en cas de voler augmentar el nombre d'Iluvatars, servidors o en cas de voler afegir noves funcionalitats. Per poder fer-ho el projecte s'ha desenvolupat en el llenguatge de programació C i s'ha dividit en diferents arxius els quals representen diferents mòduls del projecte i tots junts fan possible la compilació i correcte execució d'ell.

A continuació s'expliquen detalladament els punts claus del projecte desglossat en cada fase corresponent.



Il·lustració 1. Diagrama de seqüència d'enviament de fitxers entre diferents màquines.

1.1. Fase 1

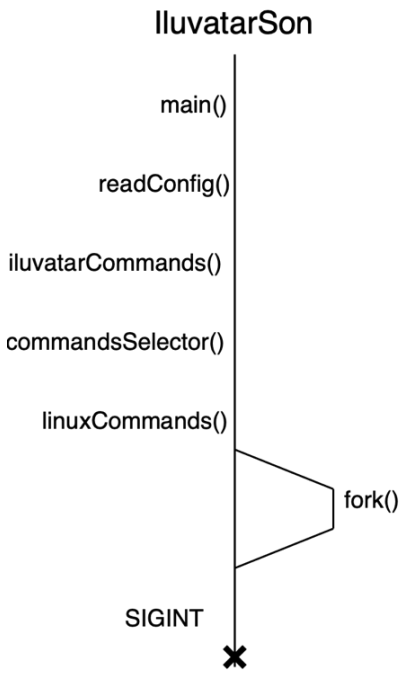
En aquesta primera fase del projecte, se'ns planteja començar la implementació del disseny dels IluvatarSon. Per això, primer de tot haurem d'implementar la lectura dels fitxers de configuració amb la utilització de les funcions "open(), close(), read() i write()" per tal de poder treballar amb els file descriptors. Per poder guaradar tota aquesta informació de configuració que anirem llegint dels fitxers, s'ha creat una estructura iluvatarConf amb els diferents camps com el nom d'usuari, la IP, el port...

Un dels conceptes principals de la pràctica apareix ja en la primera fase del projecte, la execució de comandes Linux a més d'unes quantes comandes personalitzades que haurem de crear més endavant durant les següents fases. Per poder executar les comandes Linux, s'han utilitzat forks i pipes.

Els forks els utilitzem principalment per poder clonar o duplicar un procés, el qual és l'encarregat d'executar la comanda introduïda al sistema amb la funció "execv()" i mostrar el resultat de la comanda per pantalla. Si durant el procés de la execució la comanda és errònia, mostrem un missatge d'error, ja que aquella comanda no existeix i no s'ha pogut executar. És per això que utilitzem els pipes, ja que calculem el valor de l'estat de la espera de la funció "wait()", que és la funció per a que el fill del fork acabi. Un cop el fill ha acabat, si aquesta espera és de 127, significarà que no s'ha executat la comanda i per tant notificarem per pantalla la comanda errònia, cancel·lant amb el pipe la sortida errònia de la pròpia comanda.

Per aquesta fase, també se'ns ha demanat crear un makefile per poder garantir una compilació adequada per al projecte. Aquest makefile serveix per a compilar cada mòdul per separat i un cop compilats s'uneixin per acabar amb dos compilats l'Iluvatar i l'Arda. A més és molt útil perquè simplifica la tasca de compilació, ja que amb la única comanda "make" tot el projecte es compila.

Finalment, per aquesta primera fase hem implementat la utilització dels signals, concretament per poder finalitzar el programa utilitzant el Ctrl + C. Els signals, son un mecanisme del nucli per comunicar esdeveniments als processos. Per tant utilitzem la funció "signal()" per detectar durant l'execució del programa si ens han introduït el SIGINT, es a dir un Ctrl + C. Un cop detectat, el procés és direcciona directament cap a la funció del signal per alliberar tota la memòria demanada, tancar file descriptors, threads, semàfors... que es podrien haver utilitzat en el programa.



Il·lustració 1. Diagrama de seqüència d'un Iluvatar.

1.2. Fase 2

En aquesta segona fase del projecte, cal fer un primer pas per a la interconnectivitat dels IluvatarSon amb Arda. Concretament una connexió clients – servidor amb la utilització dels sockets. Aquesta connexió te en compte un protocol de comunicació molt important que hem de tenir en compte en tot moment, que son les trames.

En aquest protocol de comunicacions per trames, s'utilitza únicament un sol tipus de trama. Aquestes tenen una dimensió variable depenent de la seva utilització i sempre estaran formades per quatre camps:

- **Type:** Descriu el tipus de trama que és en hexadecimal, aquest serà l'identificador de la trama.
- **Header:** És l'identificador de tipus de la trama en forma de missatge sempre amb "[]" i per a cada tipus de trama hi ha de diferents.
- **Length:** És la llargària del camp data, esta en format numèric de 2 bytes.
- **Data:** És un camp que serveix per enviar la informació per les trames. Depenent del cas, aquest por contenir diferents valors.

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

Aquestes trames s'aniran enviant client – servidor a traves dels sockets. Tant Arda com els IluvatarSon, al inicialitzar el programa hauran de configurar-se per poder establir aquesta comunicació. Per això, hem utilitzat per als IluvatarSon la estructura del tipus sockaddr_in per anar guardant la informació en els diferents camps per poder establir una connexió correcte. Per al client, hem utilitzat les funcions "socket()" per obtenir el file descriptor del servidor, "inet_pton()" per configurar la IP i "connect()" per establir la connexió amb el servidor. Per l'altra banda, al Arda també hem utilitzat la estructura sockaddr_in i la funció "socket()" per poder obtenir el file descriptor, però en el cas del servidor, utilitzem la funció "bind()" per indicar al SO que l'aplicació espera informació a un port determinat i la funció "listen() que obre el port assignat al socket i determina el nombre màxim de connexions a acceptar". D'aquesta manera, ja tindríem configurada la connexió clients – servidor.

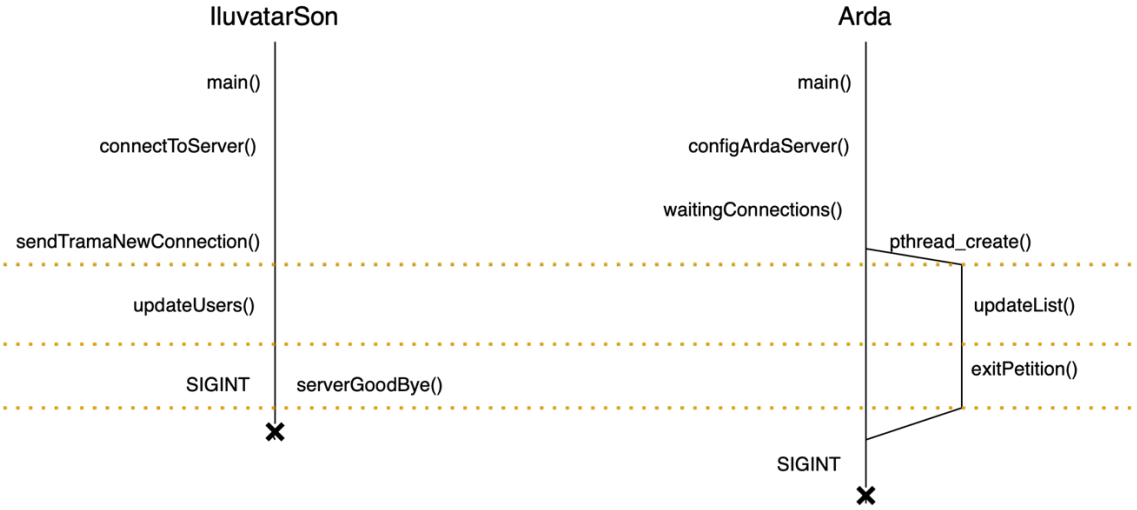
Primer de tot, hem de especificar el disseny d'Arda. Aquest servidor l'hem dissenyat com a un servidor dedicat. Per a cada connexió d'un client IluvatarSon crearem un thread per a que s'ocupi de les seves peticions en un procés diferent del principal. D'aquesta manera tindrem tants fils d'execució com clients connectats a Arda.

A continuació, connectarem Arda llegint el seu fitxer de configuració al igual que fèiem a la fase 1 amb els IluvatarSon. Tot seguit, ja haurem d'enviar la primera trama de comunicació notificant a Arda la connexió del primer IluvatarSon amb els sockets i es

crearà el primer fil d'execució. Es podran connectar d'aquesta manera molts IluvatarSon diferents en tot moment des de servidors diferents. Cada IluvatarSon podrà anar demanant informació a Arda per trames i aquest respondrà amb una altra trama. Per exemple, quan un IluvatarSon vulgui executar la comanda "Update users", aquesta comanda personalitzada enviarà una trama demanant la llista d'usuaris connectats, i Arda respondrà amb una altre trama amb la llista actualitzada.

Arda en tot moment anirà informant per pantalla les diferents peticions dels IluvatarSon. I com que Arda té diferents fils d'execució depenent del numero de clients que estan connectats, aquets poden tenir exclusió mútua i mostrar per pantalla dues o més peticions que es podrien haver demanat en el mateix moment. Per això hem utilitzat semàfors, una eina molt útil que ens ajuda a bloquejar un procés fins que el primer que ha entrat no finalitzi. El semàfor estarà inicialitzat amb la funció "Init()" amb un valor de 1 i farà la espera dels processos amb "SEM_wait() i SEM_signal()" de la llibreria proporcionada al estudy "semaphore_v2.h".

Finalment, Arda també utilitza signals, concretament SIGINT per a la funcionalitat del Ctrl + C per tancar el programa. En aquesta funcionalitat, se'ns demana el control de tancament de Arda amb la funcionalitat de que aquest també tanqui tots els IluvatarSon que en aquell moment estiguin connectats al servidor. Per poder complir això, hem utilitzat el select, una eina molt útil que anirà escoltant si Arda s'ha tancat o no. Si finalment Arda s'ha tancat, ja que haurem fet un Ctrl + C, el select que escolta el file descriptor de Arda activarà la seva funcionalitat i tancarà els respectius IluvatarSon com si aquets també haguessin rebut una comanda de exit o un Ctrl + C.



Il·lustració 2. Diagrama de seqüència de connexió d'un Iluvatar a Arda.

1.3. Fase 3

En aquesta tercera fase s'ha implementat l'enviament de missatges i fitxers entre diferents processos IluvatarSon que es trobin en diferents màquines. Per poder realitzar aquesta comunicació s'han tornat a utilitzar sockets com a la fase 2, però en aquest cas s'ha fet que cada Iluvatar a més de ser un client d'Arda també pugui ser un servidor perquè altres Iluvatars puguin parlar amb ell i col·lateralment s'ha hagut de tenir en compte que cada Iluvatar pugui ser client d'un altre Iluvatar.

Primerament s'han afegit en el Iluvatar dues noves estructures de tipus "sockaddr_in", una per fer de de servidor i l'altre per poder establir la connexió amb algun altre Iluvatar, al iniciar l'execució del procés Iluvatar es crea el servidor amb la informació del fitxer de configuració afegit com a argument. Junt amb el servidor es crea un altre fil d'execució el qual s'encarrega d'acceptar totes les connexions que li arribin i seguidament crea un thread exclusiu amb el file descriptor del nou usuari connectat per tal de rebre el missatge o fitxer que l'altre Iluvatar vol compartir, un cop ja s'ha rebut el missatge o fitxer de l'usuari es tanca el fil d'execució exclusiu per aquest client creat anteriorment.

Com es pot comprovar, es realitza el mateix mecanisme que amb la comunicació d'Arda, es a dir una arquitectura de servidors dedicats on cada client Iluvatar té el seu propi fil d'execució per tal d'agilitzar el procés de transferència de missatges o fitxers.

D'aquesta manera s'ha hagut d'implementar tres funcions clares, reciveFromIluvatar() la qual es crida amb un fil d'execució nou per rebre la informació i les funcions sendFile() i sendMsg() perquè un Iluvatar pugui enviar la informació.

La funció d'enviar fitxers primer enviar la trama on s'identifica el fitxer a enviar i seguidament es van enviant trames d'un màxim de 512Bytes on s'afegeix la informació del fitxer llegida, un cop s'acaba d'enviar tota la informació del fitxer s'espera a que l'Iluvatar receptor contesti amb un OK o KO en funció de si els hash del fitxer enviat i del rebut coincideixen. La mida total de la trama era un requisit de l'enunciat alhora d'enviar fitxers i per tal de complir es van fer els següents càlculs:

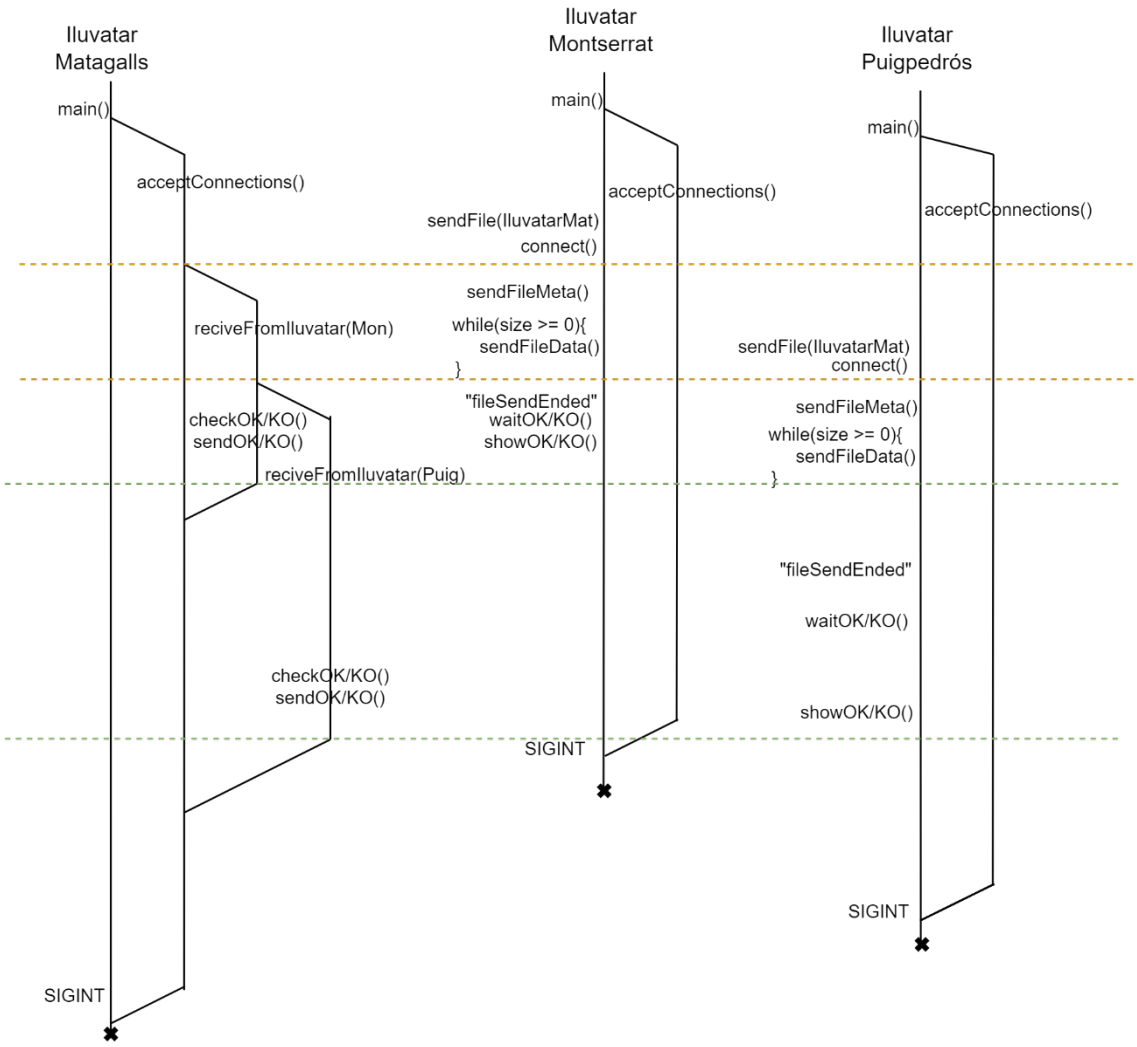
```
Trama = Max data size 512Bytes
Type + Header = 1Byte (TT_SEND_FILE) + 12Bytes (TH_FILE_DATA)
Length = 2Bytes
Data = (512 – 1 – 12 – 2 – 1("\0") ) = 496Bytes to use
```

Així doncs, realment cada paquet contenia un total de 496Bytes d'informació del fitxer.

Per poder calcular el hash i saber la mida del fitxer s'ha hagut de crear una funció la qual s'encarrega d'executar comandes de Linux i retorna l'output, aquesta funció funciona amb un fork el qual crea un procés fill i fa que executi la comanda Linux amb la funció "execvp" i l'output d'aquesta comanda es redirigeix amb la funció "dup2" a

un extrem d'una pipe que esta connectada amb el procés pare. Finalment el pare espera a que el fill acabi i llegeix per l'altre extrem de la pipe el resultat de la comanda. A més de la transmissió, s'ha hagut de contemplar el cas on qui envia el fitxer vol cancel·lar la transmissió d'aquest, per fer-ho s'ha afegit una variable global la qual marca val 1 en cas que s'estigui enviant i 0 en cas contrari, d'aquesta manera en cas de detectar un intent de tancar Iluvatar s'informarà a l'usuari que s'ha d'esperar a que s'acabi l'enviament del fitxer, un cop acabat d'enviar l'usuari pot sortir sense problema.

Finalment es pot veure el funcionament de la comunicació en el següent diagrama:



Il·lustració 3. Diagrama de seqüència d'enviament de fitxers en diferents màquines.

1.4. Fase 4

En aquesta quarta fase s'ha implementat l'enviament de missatges i fitxers entre diferents processos IluvatarSon que es trobin a la mateixa màquina. Per poder realitzar aquesta comunicació s'han utilitzat les cues de missatges "mailboxes" de tipus enllaç indirecte amb una mida fixa, més concretament s'han utilitzat dues cues de missatges, una anomenada "msg_mailbox" utilitzada per enviaments d'únicament missatges i com a conseqüència únicament d'estructures de tipus "MsgTrama", i una segona cua anomenada "file_mailbox" que com diu el propi nom només s'envien fitxers i com a conseqüència únicament s'envien estructures de tipus "FileTrama".

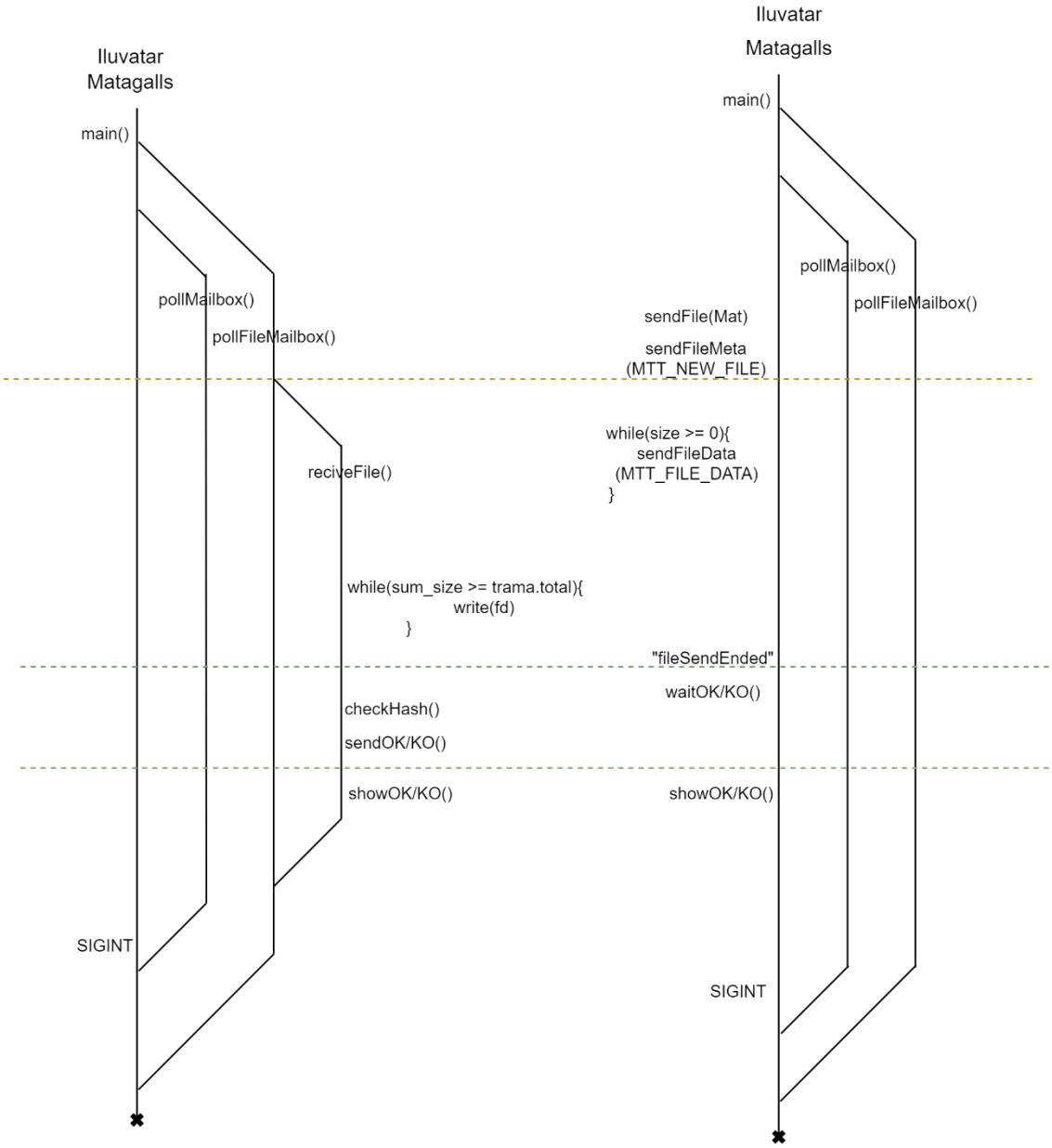
Les dos estructures tenen un camp anomenat "idtrama", el qual serveix per identificar cada trama amb un identificador únic i a més també tenen un altre camp que es diu "destPort" el qual conté el port de a qui va dirigit, ja que dins uns mateixa màquina els ports són únics i així es poden identificar fàcilment els Iluvatars entre ells.

El funcionament de la recepció de missatges o fitxers és molt semblant a la de la fase anterior, primerament quan l'Iluvatar s'inicia es creen les dos bústies i seguidament es crea un fil d'execució per cada bústia creada per poder anar fent "polling" i saber en tot moment si hi ha algun nou missatge dins d'alguna de les bústies. Un cop arriba algun nou missatge a la bústia exclusivament de missatges i es comprova que el destinatari es l'Iluvatar en qüestió es crida la funció msgRecived() la qual s'encarrega de mostrar d'informació del missatge rebut en cas de ser una trama de tipus "MTT_MSG" o en cas de ser de tipus "MTT_ACK_MSG" es mostra per pantalla que s'ha enviat correctament el missatge. Per l'altre banda en cas de voler enviar un missatge utilitzant les bústies simplement s'afegeix la informació del missatge a l'estructura junt amb el port destinatari i s'envia a la bústia corresponent.

En cas que la bústia de fitxers rebi informació d'un nou fitxer i que el tipus de trama sigui "MTT_NEW_FILE" es crea un nou fil d'execució exclusiu per a la recepció d'un fitxer en concret, aquest nou fil d'execució s'encarregarà d'anar esperant nous trossos del fitxer a rebre i els va afegint en un nou fitxer creat. Un cop s'acaba la recepció de tots els trossos del fitxer es contesta amb un "MTT_OK_FILE" en cas que el hash del fitxer esperat a rebre i el fitxer finalment rebut coincideixin, "MTT_KO_FILE" en cas contrari. Per l'altre banda, en cas de voler enviar un fitxer utilitzant bústies de missatges s'utilitza un procediment molt semblant al de la fase anterior, primer s'envia una primera trama amb el tipus "MTT_NEW_FILE" perquè el receptor sàpiga que ha de crear un nou fil d'execució per rebre el fitxer, i seguidament es van enviant trames de tipus "MTT_FILE_DATA" amb la informació del fitxer a enviar, un cop acabat l'enviament del fitxer s'espera a que el receptor contesti amb un OK o KO.

Així doncs es podria considerar l'enviament de fitxer dins la mateixa màquina de tipus servidor dedicat ja que es crea un fil d'execució secundari per cada nou fitxer a rebre.

També, com a la fase anterior s'ha de gestionar la possible cancel·lació de l'enviament de fitxer per part del emissor, per fer-ho s'ha seguit la mateixa estratègia, es a dir, que fins que el fitxer no s'hagi acabat d'enviar no es deixi finalitzar l'execució del Iluvatar. Finalment es pot veure un diagrama de seqüencia de l'enviament de fitxers dins una mateixa màquina.



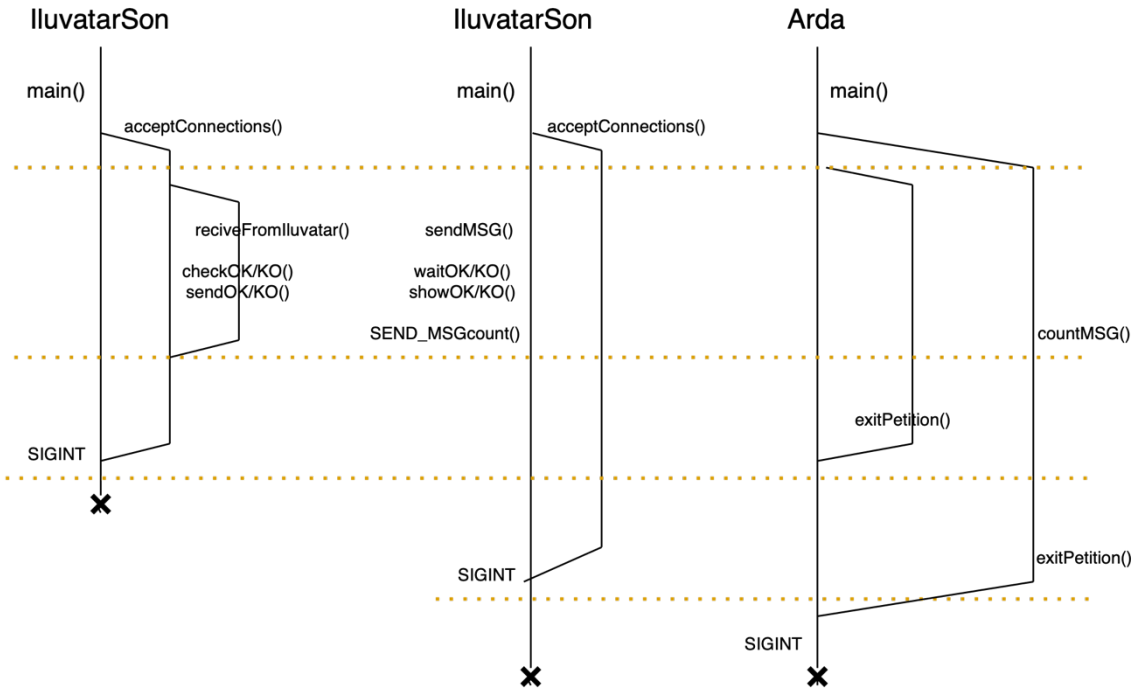
Il·lustració 4. Diagrama de seqüència d'enviament de fitxers a la mateixa màquina.

1.5. Fase 5

Finalment, en aquesta última fase ens demanen tenir un registre dels missatges que s'envien entre els diferents IluvatarSon. Aquest registre s'ha de guardar quan tanquem Arda i tornar a llegir el valor per seguir el registre dels missatges.

Primer de tot, hem creat una funció que llegirà el valor dels missatges enviats anteriorment en el fitxer que hem nombrat "countMSG.txt", i en el cas de que aquest no existeixi, el crearà. Un cop obtingut aquet valor, el guardarem en una variable global del tipus enter, el qual anirà incrementant-se cada cop que Arda rep una trama dels IluvatarSon notificant de que s'ha enviat un missatge. Aquesta trama només serà enviada si el missatge entre aquest IluvatarSon s'ha enviat correctament. Si finalment Arda rep un CONTROL + C per sortir del programa, guardarem el valor del comptador de missatges en el mateix fitxer que haurem llegit o creat si aquets no existia abans.

Finalment, hem hagut d'utilitzar semàfors per a la exclusió mútua, ja que de la manera que tenim implementat el nostre servidor explicat anteriorment a la fase 2, podem tenir exclusió mútua, ja que diferents IluvatarSon podrien enviar-se un missatge en el mateix instant i que dos o més fils d'execució voldrien modificar aquesta variable. Per això, utilitzem un semàfor per bloquejar un fil d'execució fins que el que estigui modificant la variable haguí acabat.



Il·lustració 5. Diagrama de seqüència d'enviament trama MSGCount a Arda.

2. Problemes observats i com s'han solucionat

Al llarg del desenvolupament de la pràctica es van trobar diversos problemes ja que al llarg del desenvolupament i implementació del projecte també es van anar consolidant noves eines vistes a classe. Més concretament, ens vam trobar amb els següents problemes els quals es van aconseguir solucionar per tal de que l'execució de la pràctica funciones en la seva totalitat.

- El primer problema amb el qual ens vam trobar a l'inici de la pràctica i que es va anar arrastrant al llarg de la pràctica va ser la modulació del projecte. Fins ara mai ens havíem trobat amb una pràctica tant gran i feta per implementar-se en C, i com a conseqüència no tenim cap experiència en la modulació i escalabilitat de projectes d'aquesta magnitud, llavors ens vam trobar amb la necessitat d'entendre el projecte en la seva totalitat per poder preveure futurs mòduls que en un futur es necessitarien. A més, com que a l'inici de la pràctica no sabíem encara amb quines eines es podrien resoldre les diferents fases era complicat preveure l'estructuració de cada futur mòdul.

  Malgrat aquestes dificultats, vam fer un esforç per comprendre el projecte i vam aconseguir dividir-lo en diferents mòduls que poguessin ser implementats de forma independent. Això ens va permetre avançar més ràpidament i ens va facilitar la depuració i el manteniment del codi.

- Un altre problema amb el qual ens vam trobar va ser alhora d'enviar fitxers, ja que "debuggar" fitxers rebuts que no son de text és una feina impossible. Per solucionar-ho es va anar provant de diferents maneres l'enviament fins que executant la comanda md5sum els hash coincidien, i es va donar per bo l'enviament i recepció.

- Finalment, amb l'últim problema amb el que ens va, trobar va ser alhora de serialitzar la trama, ja que s'havia de fer amb un únic "write" de manera que primer havia d'estar tot en format de cadena de caràcters i seguidament s'havia d'enviar de cop, i la funció "asprintf" no va acabar de ajudar ja que en alguns casos canviava la mida d'algun text i feia que la trama serialitzada resultant no tingues la mida correcte.

3. Estimació temporal

L'estimació temporal és una part important de qualsevol projecte, ja que permet planificar els recursos i els esforços necessaris per completar les tasques de manera eficient. En aquest apartat hem dedicat temps a analitzar les diferents tasques que s'han de realitzat i a estimar el temps que es necessitarà per completar-les.

A continuació, es detalla l'estimació aproximada de temps dedicat a cada fase del projecte:

	Investigació	**Disseny**	**Implementació**	**Testing**	**Documentació**	**Total**
Alex	5h	2h	26h	8h	4h	45h
Marc	5h	2h	30h	10h	4h	51h
Total	10h	4h	56h	18h	8h	96h

4. Conclusions i propostes de millora

Gràcies a la realització d'aquest projecte s'han pogut consolidar i posar a prova les habilitats pràctiques en l'ús de crides al sistema (signals, sockets, pipes, semàfors, etc.), modulació en C i també s'han millorat les habilitats en la planificació i estimació temporal dels projectes. També hem après a treballar de manera més eficient en equip i a gestionar els problemes que es van presentar al llarg del desenvolupament del projecte.

En general, la realització d'aquest projecte ha estat una gran oportunitat per aprofundir en els coneixements adquirits a classe i desenvolupar habilitats pràctiques valuoses. Esperem que aquesta experiència ens serveixi com a base per a futurs projectes i ens ajudi a seguir millorant.

Les nostres propostes per a millorar, son concretament en la part de la explicació de la pràctica. Durant el projecte, hem dubtat en molts moments de la manera en la que es podia entendre el que ens demanaven. Per exemple, en el document que ens han donat, no diu que estigues prohibit avisar els IluvatarSon de que Arda s'havia tancat, i si no haguéssim comentat posteriorment als becaris, no ens haguéssim donat compte.

Finalment, l'altre punt de vista nostre per a millorar és en el feedback dels checkpoints. Estan molt bé i ens ajuden molt alhora de seguir amb la pràctica, però en alguns casos el feedback ha sigut molt poc concret deixant-nos dubtes per al progres nostre. Tot i així, si que han hagut feedbacks molt bons i que ens han ajudat molt.

5. Bibliografia utilitzada

Developers, V. (25 / 12 / 2022). *Valgrind*. Recollit de https://valgrind.org/docs/manual/manual.html

Man7. (05 / 11 / 2022). Recollit de https://man7.org/linux/man-pages/man2/connect.2.html

Mutschlechner, R. (14 / 12 / 2022). *Stackoverflow*. Recollit de https://stackoverflow.com/questions/27541910/how-to-use-execvp

Ram, V. (14 / 12 / 2022). *DigitalOcean*. Recollit de How to use the execvp() function in C/C++: https://www.digitalocean.com/community/tutorials/execvp-function-c-plus-plus

Stackoverflow. (02 / 01 / 2023). Recollit de https://stackoverflow.com/questions/19364942/points-to-uninitialised-bytes-valgrind-errors