Mr. J. System

Introducció a les pràctiques de Sistemes Operatius

Un sistema operatiu està format, bàsicament, per 4 mòduls o subsistemes:

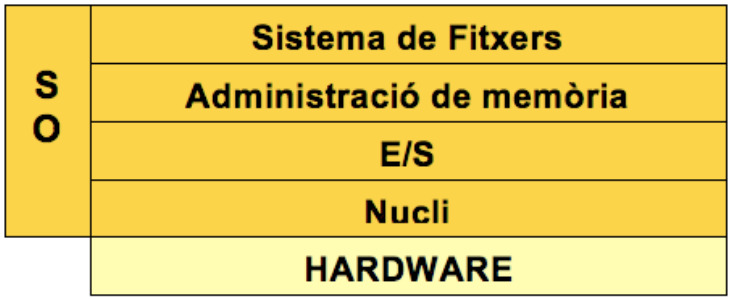

Figura 1. Arquitectura per capes d'un Sistema Operatiu.

NUCLI. És l'única capa que pot inhibir interrupcions. Les seves funcions bàsiques són la representació dels processos en execució, el control de la concurrència de processos, la gestió i control de les interrupcions, i dotar al sistema de mecanismes de comunicació entre processos, de sincronització i exclusió mútua.

SISTEMA E/S. És l'única capa que pot executar instruccions del tipus entrada i sortida. La seva funció és la comunicació amb els perifèrics. Per tant, les capes superiors, per a dialogar amb el *hardware*, ho han fer a través d'aquesta capa.

GESTIÓ DE MEMÒRIA. Conjuntament amb el *hardware*, crea la possibilitat de disposar de memòria virtual. A més a més, aquesta capa és la responsable de garantir (també conjuntament amb el *hardware*) la protecció de les dades a memòria i també la seva compartició.

SISTEMA DE FITXERS. Dota al sistema d'una visió estructurada de les dades emmagatzemades al disc.

Les aplicacions d'usuari són programes creats per l'usuari. Quan aquests programes necessiten alguna funcionalitat del sistema operatiu, realitzen el que s'anomena una "crida al sistema", que consisteix en fer crides a funcions que ofereixen les diferents capes. És a dir, si l'aplicació vol mostrar un caràcter per pantalla, haurà de cridar a una funció de la capa d'E/S per a aconseguir-ho.

El contingut de la pràctica de l'assignatura de Sistemes Operatius està orientat a l'aprenentatge d'una arquitectura distribuïda mitjançant la diferents mecanismes de comunicació (*sockets* principalment), posant de relleu la problemàtica de la concurrència de processos i la multitasca. L'objectiu principal és establir connectivitat entre els diferents equips i permetre la transmissió d'informació entre els diversos nodes d'un sistema distribuït. A més, cal la utilització de mecanismes del nucli del sistema operatiu (memòria compartida, mètodes d'exclusió mútua, sincronització i creació de processos) per a poder resoldre la pràctica.

NOTA 0. Les quatre pràctiques del curs 2004-2005 rebien els noms de *Arda*, *Arda++*, *Harkonnen* i *IluvatarSon*. Tot era en honor a Frank Herbert i la seva famosa novel·la *Dune*. Força alumnes van recordar l'origen i van descobrir d'on provenia el nom.

NOTA 1. Les quatre pràctiques del curs 2005-2006 rebien els noms de *Oedipus Rex*, *Sphinx*, *Antigona* i *Teiresias*. Com totes les pràctiques de SO, no hi ha res a l'atzar. Els noms provenien de la tragèdia d'Èdip de Sòfocles i dotaven de sentit a les pràctiques. S'iniciava el curs amb una pràctica 1 senzilla, Èdip Rei. Èdip viu feliç com a rei mentre ignora que ha matat el seu pare i s'ha casat amb la seva mare. La ignorància el fa feliç. Els alumnes de SO viuen feliços perquè la pràctica 1 és fàcil i no saben que els espera. La pràctica 2 és l'Sphinx. Èdip ha de superar la prova de l'Sphinx ja que, sinó, aquesta el matarà. Òbviament, la pràctica 2 del curs 2005-2006 era la més difícil de totes i si no la superaves no aprovaves l'assignatura. La pràctica 3 es deia Antígona, qui ajuda a Èdip un cop aquest cau en desgràcia i en la desesperació. Antígona era la salvació dels alumnes que volien anar a l'examen de juny i necessitaven una pràctica llarga lliurada. Us estalvio els detalls de Teiresias. Tot i que molts alumnes van seguir els noms i les referències, cap d'ells va acabar de copsar el sentit de la tragèdia...

NOTA 2. Les pràctiques del curs 2006-2007 es van centrar en l'obra mestra de Dante Alighieri, "La Divina Commedia". Les pràctiques es deien "Inferno", "Purgatorio" i "Paradiso". Suposo que no calen gaire comentaris per entendre què passa a la pràctica 1. La segona pràctica es suavitza però la necessites si vols presentar-te a examen. Finalment, la tercera és relativament curta i senzilla, un paradís comptant amb tot el que has fet anteriorment.

NOTA 3. Les pràctiques del curs 2007-2008 es van centrar en l'obra mestra dels germans Wachowski, "Matrix", on els humans (alumnes) han de lluitar contra les màquines per a sobreviure (aprovar l'assignatura). Les pràctiques es deien "Matrix", "El Ferroviario" i "Keymaker". Matrix era el servidor central on s'havien de connectar els clients, com *Nebuchadnezzar*, per a poder intercanviar arxius i comunicar-se mitjançant un xat distribuït. També podien connectar-se amb altres dimonis, com *Oraculo* o *Link*, que els donaven consells. *El Ferroviario* era una pràctica de simulació de càrrega de processos i administració de memòria. Si coneixeu el paper de *El Ferroviario* a la pel·lícula *Matrix* veureu que la relació és directa. Finalment, la codificació de fitxers en EXT2. Vam considerar que un bon nom per descodificar era el personatge de *Matrix* anomenat *Keymaker*.

NOTA 4. Les pràctiques del curs 2008-2009 es van centrar en l'obra mestra de Isaac Asimov, "La fundació". Les pràctiques es deien: "Trantor", el planeta central de la galàxia, tenint, per tant, la mateixa magnitud que la pràctica 1 de Sistemes Operatius; "Trevize, the loader", nom del conseller de la primera fundació, amb una intuïció insòlita, però amb perilloses intencions; i "Pelorat, the file reader", nom del professor d'història que ajudà a Trevize a trobar el planeta Terra, la llar on descansar després d'aprovar les tres pràctiques de Sistemes Operatius.

NOTA 5. Les pràctiques del curs 2009-2010 es van centrar en el grup australià de *hard rock* AC/DC, com a homenatge a la marxa del que va ser professor de pràctiques de l'assignatura els tres darrers anys, Hugo Meza. La primera pràctica, sota el nom de *Highway to shell* (petita modificació del nom de la cançó més popular del grup), conduïa als alumnes per tres fases. La

primera, *Welcome to the Shell*, donava la benvinguda a l'infern als alumnes amb la implementació d'un intèrpret de comandes *shell*, sota el nom de Malcolm (cantant del grup). A la segona fase, per tal de fugir de les tenebres en què es trobaven, els alumnes es van veure pactant amb el diable (*Dealing with the Devil)*, on havien de seguir tot un protocol per a comunicar-se amb el dimoni Angus (guitarrista del grup). Finalment, quan creien que tot havia passat, es topaven amb *Ballbreaker* (disc que van gravar al 1995), nom que no requereix gaires explicacions. La segona pràctica del curs 2009-2010 rebia el nom de *Back in Black*, nom del disc llençat al 1980, on els alumnes havien d'identificar el format d'un volum donat i extreure'n informació.

NOTA 6. Les pràctiques del curs 2010-11 es van centrar en les obres del polèmic *Donatien Alphonse François de Sade*, més conegut com "el Marqués de Sade". Així, la pràctica es titulava "*Les 120 journées de Sodome*". Aquesta estava dividida en tres fases: "*Le Château de Silling"* era la primera d'elles, on els alumnes gestionaven el comportament del client de l'aplicatiu, el qual rebia el nom del personatge de la novel·la "Blangis". Posteriorment els alumnes s'endinsaven al castell en una segona fase anomenada "*Les quatre madames*", on Blangis es connectava a un dimoni anomenat "Thérèse". Finalment, la cosa es desmadrava a la tercera fase, on els alumnes havien de crear el servidor "*Libertinage*", que, com el seu nom indica, és on començava el "festival".

NOTA 7. Les pràctiques del curs 2011-12 es van centrar en la mundialment coneguda sèrie televisiva del Simp(so)ns. Aquesta pràctica estava dividida en cinc fases: "La shell de Homer", "el bar de Mou", "Clancy Wiggum", "Living in Springfield" i "Living in Springfield++", en el seu conjunt aquestes fases formaven un sistema que permetia executar comandes de forma local, algunes pròpies en un servidor remot i l'activació de diversos serveis que mostraven frases mítiques de la sèrie, tot això seguint una arquitectura client - servidor.

NOTA 8. La pràctica del curs 2012-13 es va anomenar LsBox. El motiu va ser força simple: es tractava de dissenyar i implementar un sistema molt similar (de fet simplificat) del conegut Dropbox. L'únic detall curiós era el logotip de la pràctica que portava un mapa d'Austràlia encobert doncs era un petit homenatge a un monitor de SO que deixava aquestes tasques.

NOTA 9. La pràctica del curs 2013-14 es va anomenar LsHangIn. El motiu era que era una versió simplificada del conegut Google Hangouts: es tractava de dissenyar i implementar un sistema de xats amb múltiples sales per a usuaris. A més, cadascuna de les Fases tenien relació amb la pel·lícula Resacón en las Vegas.

NOTA 10. La pràctica del curs 2014-15 es va anomenar Gekko. Gekko és un empresari i principal protagonista de la pel·lícula Wall Street, relació directa amb la Borsa i l'objectiu de la pràctica. Però, a més, hi havia diversos homenatges més: TumblingDice, el generador de fluctuacions, és també una cançó dels Rolling Stones i Dozer, l'operador de Borsa, és un dels personatges de Matrix.

NOTA 11. La pràctica del curs 2015-16 es va anomenar LsTransfer, the force awakens. Era un clar homenatge al retorn de la saga d'Starwars. A més el servidor es deia Naboo (planeta que surt en diferents episodis d'Starwars) i els clients Gungan (habitants d'aquest planeta).

NOTA 12. La pràctica del curs 2016-17 es va anomenar LsTinder, may the Love be with you. Era per la clara similitud a la xarxa social a la qual la pràctica feia referència i el lema un altre homenatge a la saga Starwars. A més, els diferents processos es deien Rick i Morty en referència a una sèrie americana de televisió d'animació per adults.

NOTA 13. La pràctica del curs 2017-18 es va anomenar LsEat, may the Food be with you. Primerament, el nom en referència a la moda del Just Eat. El lema un altre homenatge a la saga Starwars, en la seva imminent estrena de l'Episodi VIII, The Last Jedi. A més, els diferents processos tenien noms d'homenatge a StarTrek. Picard i Data personatges i Enterprise, la nau.

NOTA 14. La pràctica del curs 2018-19 es va anomenar Cosgrove System, Stairway to heaven. Primerament, Cosgrove és el cognom de la família protagonista d'una pel·lícula mítica de Peter Jackson: Braindead, considerada un clàssic de les pel·lícules gore de l'època. El lema, Stairway to Heaven, no era res relacionat amb els observatoris de la pràctica sinó un homenatge a la famosa cançó de Led Zeppelin. A més, els diferents processos tenien noms dels protagonistes de la pel·lícula Braindead: McGruder, McTavish, Paquita i Lionel.

NOTA 15. La pràctica del curs 2019-20 es va anomenar Cypher System era un homenatge de nou a Matrix, doncs s'està gravant pel 2022 Matrix IV, un mite en el món de la ciència ficció. L'script de la portada si l'executàveu feia les lletres verdes caient per la pantalla de Matrix. Trinity és la protagonista principal de Matrix i els exemples sempre són noms de personatges de la saga. No hi ha res a l'atzar.

NOTA 16. La pràctica del curs 2020-21 es va anomenar Overlook System. Això era un homenatge a la mítica pel·lícula de El Resplandor (the Shining) d'Stanley Kubrick on el 2020 feia 40 anys de la seva estrena. L'Hotel s'anomenava Overlook i la imatge de la portada de la pràctica era la d'un dels seus mític passadissos. Els diferents processos del sistema a dissenyar eren Jack, Wendy, Danny i Lloyd que són noms de personatges de la saga. No hi ha res a l'atzar.

NOTA 17. La pràctica del curs 2021-22 es va anomenar Arrakis System. Això era un homenatge a la mítica pel·lícula de Dune de Denis Villeneuve del 2021, però pensant amb l'original de David Lynch de 1984. Arrakis és el planeta on es desenvolupa la gran part de la pel·lícula. Hi havia els Atreides (els bons), els Harkonnen (els dolents) i els Fremen (la raça nativa del planeta Arrakis) que són elements importants i claus a la mítica pel·lícula.

NOTA 18. La pràctica del curs 2022-23 es va anomenar Eä System. Això era un homenatge a la sèrie El Senyor dels Anells: Els anells de poder (originalment en anglès, The Lord of the Rings: The Rings of Power). És una sèrie de televisió basada en la novel·la El Senyor dels Anells i els seus apèndixs de J. R. R. Tolkien. Transcorre milers d'anys abans d'El hòbbit i El Senyor dels Anells de Tolkien a la Segona Era de la Terra Mitjana. Està produïda per Amazon Studios.

NOTA 19. La pràctica del curs 2024-25 es va anomenar HAL 9000 System. Això era un homenatge HAL 9000, personatge mític de la coneguda sèrie de pel·lícules Odissea espacial; en concret 2001: una odissea de l'espai. 2001: una odissea de l'espai (títol original en anglès: 2001: A Space Odyssey) és una pel·lícula de ciència-ficció del 1968, dirigida per Stanley Kubrick, escrita per ell mateix i per Fleck C. Clarke. La pel·lícula tracta temes com l'evolució humana, la intel·ligència

artificial, el futur i la vida extraterrestre. David Bowman i Frank Poole eren dos dels protagonistes i per això els noms dels processos. La nau on viatjaven es deia Discovery (un altre procés) i el Monolit era una altra element important a la pel·lícula.

Ara us toca a vosaltres identificar la relació d el apràctica d'aquest any 2024-25 i posar-ho a la memòria.

Mr. J. System

Arthur Fleck està cansat de ser l'únic que percep el món de manera diferent. Convençut que la seva visió és la correcta i hauria de ser l'única, ha decidit que tots els ciutadans de Gotham comparteixin la seva percepció distorsionada de la realitat. Amb l'ajuda d'en Enigma i la Harley, vol crear un sistema capaç de modificar la realitat mateixa, fent que tothom vegi el món com ell. Però aquesta vegada, després de tants intents frustrats pel protector de Gotham, volen assegurar-se que el seu sistema sigui el més robust possible, i que no pugui ser desmantellat per ningú.

És per això que us han contractat a vosaltres, l'equip de Sistemes Operatius, per desenvolupar el projecte més ambiciós fins ara: Mr. J. System, en honor a com la Harley es refereix a l'Arthur. Aquest sistema ha de ser capaç de distorsionar la realitat de manera infal·lible, resistint qualsevol tipus de fallada i recuperant-se automàticament davant de qualsevol contratemps, amb l'excepció del servidor central, Gotham, que serà el nucli de tot, el únic punt dèbil del sistema. Tot i que la percepció dels usuaris serà alterada, el sistema que heu de dissenyar haurà de ser d'una fiabilitat a prova de falles.

L'arquitectura que desenvolupareu inclourà un servidor central, Gotham, i diversos clients, anomenats Fleck, que faran peticions de distorsió. El servidor central gestionarà aquestes sol·licituds mitjançant processos (workers) especialitzats: Enigma, que es dedicarà a la distorsió de text, i Harley, encarregada de manipular mitjans audiovisuals (imatge i àudio). El sistema funcionarà sobre una infraestructura basada en Linux, utilitzant una xarxa TCP/IP, assegurant una connexió ràpida i estable entre els diferents components.

La vostra tasca serà dissenyar i implementar tot el programari necessari per gestionar aquest ecosistema, tenint en compte les especificacions que se us proporcionaran. Aquest projecte exigeix especial èmfasi en la robustesa, l'eficiència i l'escalabilitat, perquè malgrat les possibles interrupcions o intents de sabotatge, el sistema pugui continuar funcionant sense problemes.

Per facilitar el procés, el desenvolupament es dividirà en fases incrementals. A cada fase es detallaran les funcionalitats a implementar, les limitacions i es proporcionaran proves per validar el progrés del vostre treball.

Descripció General

Una primera aproximació funcional de l'arquitectura de processos del sistema seria la que es presenta la Figura 2 tot i que, evidentment, l'arquitectura de disseny interna serà molt més complexa.

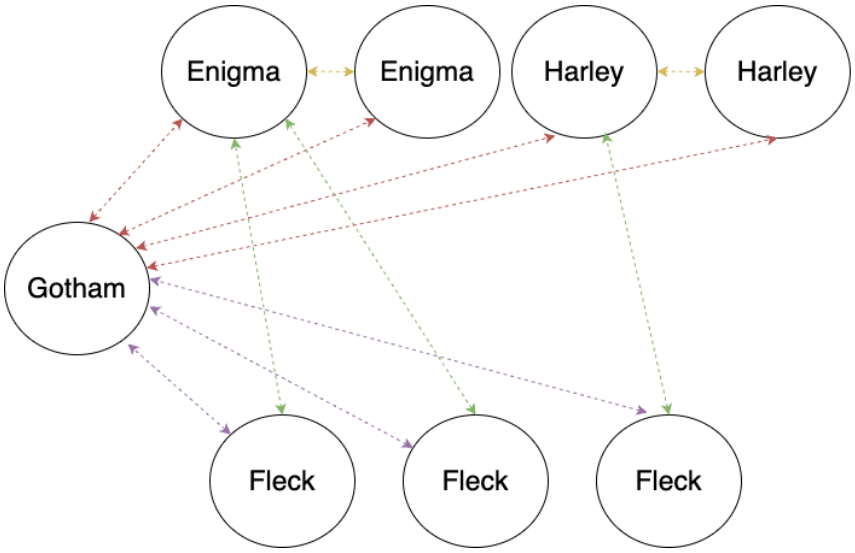

Figura 2. Mr. J System: esquema funcional.

Els processos **Fleck** seran els diferents usuaris de l'aplicació, els quals es voldran connectar al procés **Gotham** per a poder fer peticions de distorsió de text i de media. Per evitar un coll d'ampolla al procés **Gotham**, tindrem uns altres processos en màquines diferents a **Gotham** que seran els encarregats de distorsionar el que demani el client. Primer, tindrem els processos **Enigma**, en una màquina, els quals distorsionaran text. De la mateixa manera, tindrem els processos **Harley**, en una altre màquina, encarregats de distorsionar media (foto i àudio).

D'aquesta manera, quan un usuari vulgui connectar-se al sistema **MR. J**, aquest es connectarà al servidor **Gotham** i, posteriorment i de manera transparent per a l'usuari, efectuarà connexió amb el servidor **Enigma / Harley**, tot depenent de la petició que hagi realitzat.

Per tal d'assegurar la robustesa del sistema i evitar interrupcions en el servei, s'ha d'implementar un mecanisme de recuperació automàtica per als processos workers. Quan un procés **Enigma** o **Harley** cau, un altre worker disponible pren immediatament el relleu, assegurant que les peticions de distorsió en curs es continuïn processant sense

pèrdua de dades. Aquest nou worker resumeix totes les tasques que estaven en progrés, garantint que els usuaris no notin cap interrupció en el servei.

En resum, l'objectiu principal del sistema és que els diferents processos **Fleck** puguin comunicar-se amb un servidor de distorsió (**Enigma** / **Harley**), on puguin fer peticions de distorsió sense saturar el procés **Gotham**.

Per poder dissenyar i implementar la pràctica es recomana que es faci una **lectura complerta de l'enunciat**. Per facilitar la seva realització s'han planificat 4 fases:

1. A la **fase 1** es crearan els processos Fleck, Harley, Enigma i Gotham. Del procés Fleck, es processarà el fitxer de configuració i es realitzarà tota la implementació del *shell* amb les comandes esteses corresponents. Dels processos Harley, Enigma i Gotham, només caldrà processar els seus fitxers de configuració.
2. A la **fase 2** s'implementaran totes les comunicacions del sistema, tot seguint l'esquema funcional de la Figura 2.
3. A la **fase 3** s'implementarà el funcionament dels processos Harley (els distorsionadors de media), juntament amb un procés de recuperació davant de caigudes per a millorar la robustesa del sistema i tenir redundància.
4. A la **fase 4** s'implementarà el funcionament dels processos Enigma (els distorsionadors de text), amb el seu sistema de recuperació, juntament amb un sistema de logs del servidor central per a monitoritzar el correcte funcionament del sistema.

Fase 1: Calienta que sales

En la Fase 1 es començarà el disseny i programació dels quatre processos del sistema. En tots, primer de tot caldrà processar el fitxer de configuració, el nom del qual es rep per paràmetre.

A més a més, el procés Fleck funcionarà en modus línia de comandes i tindrà programades un conjunt reduït de funcions (Veure Taula 1).

Fitxer de configuració - Fleck

Aquest fitxer de text tindrà diversos camps:

- Nom de l'usuari que executa.
- Nom de la carpeta on es troben els arxius de l'usuari.
- La IP i port (cadascun en una línia que acaba amb un '\n') d'on es troba el servidor Gotham.

Exemple de fitxer de text de configuració a la Figura 3:

```
Arthur
/arthur
192.168.50.51
8011
```

Figura 3. Fitxer de configuració d'un procés Fleck.

Fitxer de configuració - Gotham

Aquest fitxer de text tindrà diversos camps:

- La IP i port (cadascun en una línia que acaba amb un '\n') d'on s'obrirà el servidor per comunicar-se amb Fleck.
- La IP i port (cadascun en una línia que acaba amb un '\n') d'on s'obrirà el servidor per comunicar-se amb Harley/Enigma.

Exemple de fitxer de text de configuració a la Figura 4:

```
192.168.50.51
8011
192.168.50.51
8015
```

Figura 4. Fitxer de configuració d'un procés Gotham.

Fitxer de configuració – Enigma / Harley

Aquest fitxer de text tindrà diversos camps:

- La IP i port (cadascun en una línia que acaba amb un '\n') on es troba el servidor Gotham.
- La IP i port (cadascun en una línia que acaba amb un '\n') d'on s'obrirà el servidor per comunicar-se amb Fleck.
- Nom de la carpeta on s'han de guardar els arxius que rebi i en la que ha de operar aquell worker en tot moment. Ha de ser diferent per cada worker.
- Tipus de worker. Això podrà ser o *Media* o *Text*. Qualsevol altre tipus es considerarà un error.

Exemple de fitxer de text de configuració a la Figura 5:

```
192.168.50.51
8015
192.168.50.51
8016
/riddler
Media
```

Figura 5. Fitxer de configuració d'un procés Enigma / Harley.

Seguidament caldrà dissenyar i implementar tota la lògica de programa que interpreta totes les comandes, les quals permeten el funcionament de Fleck. Aquestes comandes són *case insensitive*.

COMANDA	DESCRIPCIÓ
CONNECT	Connecta l'usuari al sistema. Primer es connectarà al sistema principal, Gotham. No es connectarà al respectiu worker (Enigma/Harley) fins que no faci una petició de distorsió
LOGOUT	Desconnecta l'usuari del sistema
LIST MEDIA	Llista tota la media disponible dins del directori de l'usuari
LIST TEXT	Llista tots els fitxers de text disponibles dins del directori de l'usuari
DISTORT <file.xxx> <f>	Petició de distorsió d'un fitxer. El primer paràmetre serà el nom del fitxer. Depenent de l'extensió del fitxer, serà un fitxer de text o de media. El segon paràmetre serà el factor de distorsió. Depenent de l'extensió del fitxer, tindrà diferents efectes.
CHECK STATUS	Consulta l'estat de les distorsions en curs, en forma de barres de progrés.
CLEAR ALL	Neteja les distorsions ja acabades de la llista de progrés.

Taula 1. Llistat de comandes esteses.

```
$montserrat:> Fleck config.dat

Arthur user initialized
$ DISTORT laugh.wav 2
Cannot distort, you are not connected to Mr. J System
$ CONNECT PLEASE
Unknown command
$ CONNECT
Arthur connected to Mr. J System. Let the chaos begin!:)
$ CHECK sTaTuS
You have no ongoing or finished distorsions
$ LIST MEDIA
There are 6 media files available:
1. laugh.wav
2. card.png
3. song.wav
4. gotham.jpg
$ DISTORT song.wav 23
Distorsion started!
$ something else
ERROR: Please input a valid command.
$ LIST TEXT
There are 2 text files available:
1. the_family.txt
2. psyc.txt
$ DISTORT death_of_the_family.txt 5
Distorsion started!
$ CHECK STATUS
song.wav                          96% |====================% |
the_family.txt                    61% |============          |
$ LOGOUT
Thanks for using Mr. J System, see you soon, chaos lover :)

$montserrat:>
```

Figura 6. Execució d'un procés Fleck.

Es demana:

- Que dissenyeu i programeu Fleck.c (nom del programa que farà servir els clients).
- Que programeu Gotham.c, Enigma.c, Harley.c.
- Fleck primer processa el fitxer de configuració i guarda la seva informació en una estructura adequada.
- Fleck haurà de reconèixer les comandes pròpies descrites anteriorment. Només caldrà implementar les comandes list media / text. Per les demés mostreu un missatge de “Command OK” quan es detecti correctament, i “Unknown command” quan hi hagi algun error.
- Gotham, Enigma, Harley hauran de processar el fitxer de configuració i guarda la seva informació en una estructura adequada.

Consideracions Fase 1:

- Un cop finalitzada l'execució de TOTS els processos s'ha d'**alliberar** tot tipus de memòria dinàmica significativa si n'hi hagués.
- El nom dels usuaris no pot contenir '&'. En cas que en continguin, s'hauran d'eliminar. Això és degut al protocol de comunicació del sistema, el qual està a l'Annex. Aquest serà *case sensitive*.
- Les comandes pròpies són *case insensitive*.
- Podem considerar que el **format del fitxer de configuració és correcte**.
- **No** es poden utilitzar les funcions *printf*, *scanf*, *gets*, *puts*, etc. Només es podrà interactuar amb la pantalla i amb fitxers amb les funcions *read* (lectura) i *write* (escriptura). Sí que es permet fer ús de la funció *sprintf* i similars.
- **No** es poden utilitzar les funcions *system*, *popen* ni cap variant.
- Cal garantir l'estabilitat de l'aplicació i el seu correcte funcionament. En cap cas es poden produir bucles infinits, *core dumpeds*, esperes actives, *warnings* al compilar, etc. També cal controlar tots aquells aspectes susceptibles de donar algun error i, en cas que es produeixin, **informar degudament a l'usuari** i, si és possible, seguir amb el funcionament normal de l'aplicació.
- Des d'ara i fins al final de la pràctica cal considerar que el procés Fleck pot finalitzar la seva execució utilitzant la comanda adequada o prement CTRL+C. Cal tenir-ho en compte i procurar perquè tot el sistema quedi el més **estable** possible en cas que es produeixi.
- És obligatori la utilització d'un *makefile* per generar l'executable.
- No és suficient que l'arxiu que pugeu al pou tingui extensió .tar, sinó que s'ha de poder desempaquetar amb la comanda tar. Qualsevol pràctica o *checkpoint* que no es pugui "descomprimir" d'aquesta manera **no serà corregida**.
- Recordeu que el codi ha d'estar degudament modulat, és a dir: no es pot ubicar tot en un sol fitxer .c, i és obligatori que hi hagi un fitxer *makefile*. Tingueu en compte que si en intentar corregir una pràctica aquesta no compila amb la comanda *make* (pel motiu que sigui), l'entrega serà qualificada com a no apta (2).
- Juntament amb el codi, cal entregar una explicació de com s'ha dissenyat i estructurat la fase. És molt recomanable que, abans de començar a programar, feu aquest anàlisi i ho valideu amb els monitors de l'assignatura. **Reviseu l'Annex IV per a més detalls**. L'informe ha d'incloure de manera clara i entenedora:
    - Diagrames que expliquin processos s'han creat.
    - Estructures de dades usades i la seva justificació.

Fase 2: Knock, knock… who's there?

En aquesta segona fase caldrà implementar les connexions dels Fleck amb el servidor Gotham i els workers Enigma/Harley. Els usuaris, Fleck, es connectaran a Gotham. A mesura que el client faci peticions de distorsió, s'anirà connectant als workers.

Cal esmentar que un mateix client només podrà tenir en paral·lel una distorsió de text i de media. No podrà fer dues peticions de distorsió de text a la vegada, o de media a la vegada.

Com que els diversos processos Fleck, Gotham, els workers Enigma i els workers Harley és evident que poden estar en servidors físics diferents, aquesta connectivitat caldrà dissenyar-la mitjançant ***sockets***. Reviseu el **protocol de comunicació** als Annexos.

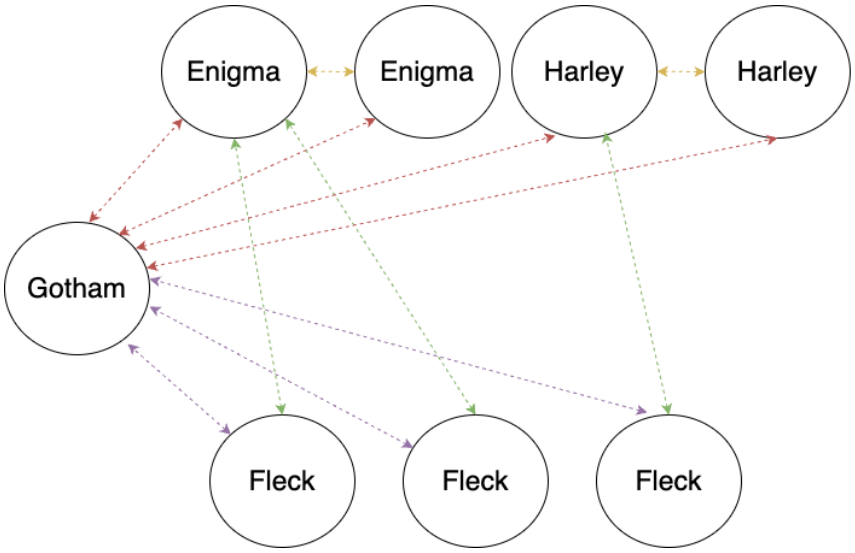

Figura 7. Esquema funcional a dissenyar i implementar.

Els servidors Enigma / Harley es connectaran a Gotham per a indicar que estan llestos per a rebre peticions de clients.

De la mateixa manera, els processos Fleck SEMPRE es connectaran primer a Gotham. Aquesta connexió serà estable, i es mantindrà fins que Fleck es vulgui desconnectar del sistema.

Quan Fleck faci una petició de distorsió, Gotham respondrà amb la direcció a la qual cal que Fleck es connecti, un servidor Enigma o servidor Harley, depenent del tipus de fitxer a distorsionar. Aquesta nova connexió també serà estable, i es mantindrà fins que s'acabi el procés de distorsió del fitxer, el qual s'implementarà a la fase 3 i 4. Recordeu que tots aquests processos tenen el seu protocol de trames específic, el qual trobareu a l'Annex.

Durant l'execució del sistema, caldrà gestionar les caigudes dels diferents processos:

1. Caigudes de Gotham
2. Caigudes de Fleck
3. Caigudes de Harley / Enigma

Caldrà gestionar-les i fer que el sistema quedi el més estable possible, tot enviant les trames de desconnexió pertinents o detectant caigudes inesperades.

Si un client fa una petició de distorsió de Media, i no hi ha cap worker Media actiu, el sistema funcionarà igualment, però respondrà amb la trama corresponent notificant que aquesta funcionalitat no està disponible en aquest moment.

Per tancar, caldrà implementar el següent:

1. La connectivitat de Fleck amb Gotham (estable).
2. La connectivitat de Fleck amb Enigma/Harley (estable).
3. La connectivitat de Enigma/Harley amb Gotham.

Connectivitat del sistema

A continuació s'exposa un exemple de connexions amb un entorn amb 1 procés Gotham (sempre hi haurà només 1, i sempre estarà encès), 1 procés Enigma, 1 procés Harley, i 1 procés Fleck.

1. Gotham comença la seva execució i escolta peticions de processos Harley, Enigma i Fleck
2. Els dos processos Harley i Enigma es connecten al sistema, fent peticions de connexió a Gotham, i notificant que estan actius.
3. Es connecta un procés Fleck a Gotham.
4. El procés Fleck connectat fa una petició de distorsió de media. Gotham respon amb respon amb la direcció IP i port del servidor Harley.
5. Fleck es connecta a Harley i fan el procés de distorsió (de moment no caldrà implementar-lo, podeu simular-lo enviant un parell de trames simples) i, una vegada acabat, Fleck es desconnecta de Harley.
6. Fleck fa una segona petició a Gotham, de distorsió de text. Gotham respon amb respon amb la direcció IP i port del servidor Enigma.
7. Fleck es connecta a Enigma i fan el procés de distorsió (de moment no caldrà implementar-lo, podeu simular-lo enviant un parell de trames simples) i, una vegada acabat, Fleck es desconnecta d'Enigma.
8. Fleck es desconnecta del sistema, tot enviant petició de desconnexió a Gotham.

Una vegada Fleck es desconnecti, torneu a tenir dues opcions d'implementació:

1. Acabar el procés
2. Tornar a posar a disposició la Shell, per si es vol tornar a connectar

Tanmateix, si un Fleck fa logout sense estar connectat, aquest acabarà la seva execució.

Les decisions de disseny del logout caldrà explicar-les de manera adient a la memòria de la pràctica, tot fent un diagrama de la comunicació implementada.

`$matagalls:> **Gotham config.dat**`
 	
`Reading configuration file`
 	
`Gotham server initialized`
 	
`Waiting for connections…`
 	
`New Enigma worker connected – ready to distort!`
 	
`New user connected: Arthur.`
 	
`Arthur has sent a text distortion petition – redirecting Arthur to Enigma worker.`
 	
`…`
	`$matagalls:> **Enigma config.dat**`
	`Reading configuration file`
	`Connecting Enigma worker to the system..`
	`Connected to Mr. J System, ready to listen to Fleck petitions`
	
	`Waiting for connections...`
	
	`New user connected: Arthur.`
	
	`New request – Arthur wants to distort some text, with factor 6.`
	`Receiving original text…`
	`Distorting…`
	`Sending distorted text to Arthur…`
	`…`

Figura 8. Exemple execució: Enigma (anàleg a Harley)

Consideracions Fase 2:

- Les de la Fase 1 segueixen essent vàlides.
- En aquesta Fase NO s'ha d'implementar la transferència de fitxers.
- Caldrà gestionar les caigudes de tots els processos, i mantenir l'estabilitat del sistema. Mireu de contemplar els següents casos:
    - Si cau Fleck, estarà connectat a Gotham però potser està enmig d'una distorsió, el worker també s'ha d'enterar. El worker haurà de cancel·lar la distorsió, i seguir a la espera de noves peticions. Gotham desconnectarà a aquest Fleck però seguirà pendent dels demés.
    - Si cau Gotham, ha de tancar-se correctament tot el sistema. En cas que hi hagués algun Worker treballant per algun Fleck fent alguna distorsió, aquests acabaran el procés de distorsió abans de finalitzar la seva execució. Aquell worker no acceptarà més peticions de distorsió que les que té en procés. Els demés tots han de finalitzar la seva execució.

  - Si cau un worker, Gotham haurà d'assignar el nou worker principal de manera aleatòria entre els que hi ha connectats. Si no queden workers d'aquell tipus de media ho tindrà present per futures peticions de distorsió denegant-les. En cas que aquell worker estigués enmig de fer una distorsió per ara l'aturarà. En futures fases implementareu que aquesta distorsió la segueixi un altre worker.

- Juntament amb el codi, cal entregar una explicació de com s'ha dissenyat i estructurat la fase. És molt recomanable que, abans de començar a programar, feu aquest anàlisi i ho valideu amb els monitors de l'assignatura. **Reviseu l'Annex IV per a més detalls**. L'informe ha d'incloure de manera clara i entenedora:
  - Diagrames que expliquin processos s'han creat, les diferents comunicacions entre processos, etc.
  - Estructures de dades usades i la seva justificació.
  - Recursos del sistema utilitzats (*threads*, *forks*, *signals*, *sockets*, semàfors, *pipes*, etc.) amb la seva justificació.
  - Opcionals implementats.

Fase 3: The Queen of Distortion

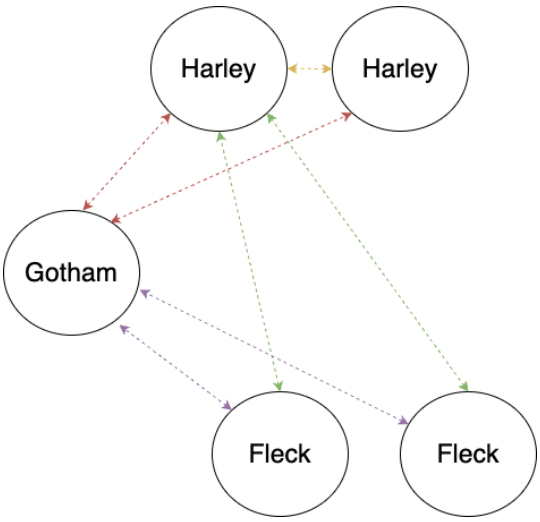

Figura 8. Esquema funcional distorsió de Media

En aquesta fase es començarà el procés de distorsió de fitxers media del sistema Mr. J. Implementant els workers Harley. En el sistema Mr. J. Podrà haver-hi múltiples workers Harley oberts alhora a la mateixa màquina, però Gotham triarà 1 per ser el principal per totes les codificacions de media. Mai podrem trobar Harleys a màquines diferents. Tots els Harleys han d'estar oberts a la mateixa. Que serà diferent a la de Gotham, i diferent màquina a on s'obriràn els diferents Enigmas, que s'explicaràn en detall a la propera fase.

El que també caldrà implementar per aquesta fase és el procés de recuperació davant de la caiguda del worker principal. En aquesta recuperació, un altre worker assignat per Gotham d'entre els connectats serà qui prengui el relleu de worker principal, i resumirà totes les peticions de distorsió que estiguessin en procés. En cas que no hi hagi cap altre worker de media, es cancel·laran les peticions de distorsió, i notificarà de que la funcionalitat de media ja no està disponible. En quant es torni a connectar un altre worker de media, tornarà a estar disponible, però no caldrà notificar-ho fins que Fleck torni a intentar fer una petició de distorsió.

El procés de distorsió consistirà en modificar el fitxer enviat pel client, tot depenent de la seva extensió. Per fer-ho disposareu de una llibreria ja implementada que trobareu explicada al Annex III que implementa el següent:

1. Si és una imatge (.png, .jpg, .jpeg, .bmp o .tga) el procés de distorsió consistirà en reescalar la dimensió d'aquesta imatge per un factor. Aquest factor es passarà com a argument. Per exemple, una imatge que tingui una mida de 1000x500

píxels amb un factor d'escalatge de 4 passarà a mesurar 250x125 píxels. Això farà que la mida en B que ocupa aquesta imatge, també es vegi reduït.
El factor d'escalatge pot anar de 1 a la mida del costat més petit de la imatge. En el exemple de la imatge de 1000x500 el factor d'escalatge pot anar de 1 a 500. En cas que fos 500 la imatge resultant seria de 2x1 píxels.

2. Si és un document d'àudio (.wav) el procés de distorsió consistirà en escurçar la durada d'aquest àudio. Per fer-ho es demanarà al client un interval, i cada Xms del interval desapareixerà aquella porció d'audio. Si el interval és de 50ms, els primers 50ms del audio estaran, els següents 50ms desapareixeran, els 50ms queden, els següents desapareixen, etc.

Aquest procés pot trigar una estona, i per això farem que cada Fleck pugui fer una única sol·licitud de distorsió a la vegada. Podrà fer distorsions de diferents tipus (media o text) alhora, però per aquesta fase no s'han d'implementar les de text. Dit això, podria donar-se el cas que varis processos Fleck estiguessin distorsionant media amb el mateix procés Harley a la vegada.

Aquesta fase, com s'ha mencionat, també serà l'encarregada d'implementar un procés de recuperació davant de caigudes. Caldrà que els diferents processos Harley connectats al sistema puguin detectar quan un d'ells cau, i reprendre sense perdre informació els processos de distorsió que el Harley desconnectat estava duent a terme.

Per a fer això, caldrà comunicar els diferents processos Harley del sistema. Aquests processos estaran a la mateixa màquina. El Harley que es tanqui, notificarà a Gotham que es tanca, i Gotham triarà el nou worker principal. A més, el Fleck que estigués enviant alguna cosa ha de detectar-ho, i fer una petició de resum a Gotham. El que farà falta però, és dissenyar un protocol de recuperació davant de caigudes que permeti a la nova Harley principal resumir totes les distorsions en procés de la que s'ha tancat, per el qual teniu lliure elecció (tot justificant les decisions a la documentació del projecte).

A més, tal i com hem dit, **per la implementació de la compressió tant d'imatges com de audio, se us proporciona una llibreria anomenada SO_compression.h** juntament amb el seu pre-compilat SO_compression.o per que no hagueu d'implementar manualment aquestes funcionalitats. Les explicacions sobre la llibreria es troben al Annex III.

Caldrà enviar tot el fitxer de media al worker, el worker el distorsionarà, i contestarà amb la versió distorsionada. Caldrà validar abans de que el worker el distorsioni que l'ha rebut bé, i en contestar amb la versió distorsionada, que aquesta s'ha rebut bé també, amb md5sum.

$matagalls:> **Harley config.dat** Reading configuration file Connecting Harley worker to the system.. Connected to Mr. J System, ready to listen to Fleck petitions Waiting for connections... New user connected: Arthur. New request – Floyd wants to distort some text, with factor 6. Sending distorted text to Floyd… ^C Disconnecting, sending current file to another Harley worker.	$montserrat:> **Harley config1.dat** Reading configuration file Connecting Harley worker to the system.. Connected to Mr. J System, ready to listen to Fleck petitions Waiting for connections... Another Harley worker has disconnecting while processing a file, recovering… New user connected: Arthur. Current distortion status: 62% Sending distorted text to Arthur… ...

$matagalls:> **Gotham config.dat**

```
Reading configuration file

Gotham server initialized

Waiting for connections…

New Harley worker connected – ready to distort!

New Harley worker connected – ready to distort!

New user connected: Arthur.

Arthur has sent a media distortion petition – redirecting Arthur to Harley worker.

Harley worker disconnected – Recovering…

Media distortion recovered – Arthur is now connected to another Harley worker

…
```

Figura 9. Exemple d'execució: 2 worker Harley i recuperació del sistema davant una caiguda

Consideracions Fase 3:

- Les de les fases anteriors segueixen essent vàlides.
- La comunicació entre processos Harley no es pot implementar amb sockets.
- El ús de qualsevol altre llibreria que no sigui SO_compression.h per la compressió no està permès. Reviseu el seu funcionament al Annex III abans d'implementar aquesta fase.
- No es pot realitzar la recuperació fent ús de memòria secundària.
- **El correcte enviament de fitxers de media s'haurà de verificar amb l'eina MD5SUM.**
- No es permès usar codi programat per fer l'MD5SUM. Cal executar la comanda md5sum que proporciona el mateix *bash* del sistema operatiu (md5sum).
- Juntament amb el codi, cal entregar una explicació de com s'ha dissenyat i estructurat la fase. És molt recomanable que, abans de començar a programar, feu aquest anàlisi i ho valideu amb els monitors de l'assignatura. **Reviseu l'Annex IV per a més detalls**. L'informe ha d'incloure de manera clara i entenedora:
    - Diagrames que expliquin processos s'han creat, les diferents comunicacions entre processos, etc.
    - Estructures de dades usades i la seva justificació.
    - Recursos del sistema utilitzats (*threads*, *forks*, *signals*, *sockets*, semàfors, *pipes*, etc.) amb la seva justificació.
    - Opcionals implementats.

Fase 4: Logging some riddles

En la fase final de la pràctica s'implementarà el procés de distorsió dels fitxers de text del sistema Mr. J. De la mateixa manera que el la fase anterior, en aquesta fase també caldrà implementar un procés de recuperació davant de caigudes per a millorar la robustesa del sistema i tenir redundància.

El procés de distorsió de text no vindrà implementat a la llibreria SO_compression.h. El client enviarà, juntament amb la petició de distorsió de text, un llindar de llargària. El worker haurà de generar un fitxer on totes les paraules de llargària inferior al llindar proporcionat no hi siguin, reduint així la seva mida.

Aquest procés pot trigar una estona, i per això farem que cada Fleck pugui fer una única sol·licitud de distorsió a la vegada de text. Seguirà podent fer-ne un altre de media. En un moment només podrà tenir 1 de text, i 1 de media alhora. Dit això, podria donar-se el cas que varis processos Fleck estiguessin distorsionant text amb el mateix procés Enigma a la vegada.

De la mateixa manera que la fase anterior, s'implementarà un procés de recuperació davant de caigudes. Caldrà que els diferents processos Enigma connectats al sistema puguin detectar que un d'ells cau, i reprendre sense perdre informació els processos de distorsió que el Enigma desconnectat estava duent a terme. Les especificacions del sistema de recuperació de la fase anterior són anàlogues per aquesta fase.

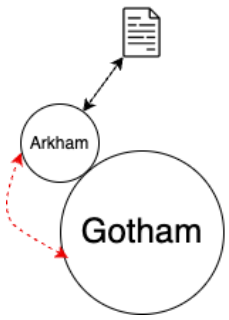

Figura 9. Esquema funcional procés Arkham

Finalment, implementarem un sistema de registres (logs) per monitoritzar el funcionament del sistema. Per això s'ha d'implementar un quart procés anomenat **Arkham**. Aquest procés ha de ser creat de l'execució de Gotham, és a dir, amb el programa Gotham.c s'haurà de crear un subprocés (Arkham) i s'hauràn de comunicar amb ell per a poder enviar-li les dades pertinents. A l'Annex es detalla el protocol de trames, on es defineix que cada trama inclou un camp "timestamp". Cada vegada que es produeixi un esdeveniment al sistema, Gotham li dirà a Arkham quina informació ha de registrar sobre el succés: què ha passat i quan, en un fitxer de registres.

Consideracions Fase 4:

- Les de les Fases anteriors segueixen essent vàlides.
- No es poden emprar sockets per a comunicar processos Enigma entre ells.
- El fitxer a actualitzar serà únic, i s'anomenarà "logs.txt".
- Cal garantir l'exclusió mútua en l'escriptura del fitxer.
- No es pot realitzar la recuperació fent ús de memòria secundària.

Requeriments de lliurament i planificació

Aquesta pràctica disposarà de diferents **punts de control o *checkpoints*** per poder fer-ne un seguiment acurat i garantir la consolidació de cadascuna de les fases de manera incremental. Concretament se seguirà el següent calendari de dates límit:

CONCEPTE	**DEADLINE**
Fase 1	20/10/2024
Fase 2	17/11/2024
Fase 3	15/12/2024
Lliurament Final 1 (sobre 10)	06/01/2025
Lliurament Final 2 (sobre 10)	02/02/2025
Lliurament Final 3 (sobre 8)	18/05/2025
Lliurament Final 4 (sobre 6)	24/06/2025

Els *checkpoints* no són obligatoris, tot i ser altament recomanables per poder garantir la robustesa de la pràctica i per saber que aneu en els terminis correctes per poder lliurar al final del semestre. Aquests *checkpoints* ponderen un 10% de la qualificació de la pràctica.

També és recomanable **validar el disseny global de la pràctica** amb els monitors de pràctiques abans d'iniciar les fases 2, 3 i 4. Així podreu garantir que la implementació no patirà d'inconsistències insalvables per fases posteriors. Per això cal aprofitar els horaris de dubtes i **portar els dissenys** impresos a validar. Cal recordar que una pràctica funcioni no és cap garantia que sigui vàlida, doncs pot ser que tot el disseny no compleixi els requisits de l'enunciat. Per això és molt important tenir garanties sobre el disseny a implementar. Podeu revisar el Annex IV per normes de disseny.

Els lliuraments es realitzaran a l'eStudy en un pou amb un fitxer que inclogui el codi font (fitxers .c, .h i el makefile) de la pràctica, funcionant completament sobre Montserrat, així com els fitxers de dades i configuració utilitzats. El fitxer haurà de ser obligatòriament en format .tar. El podeu generar amb la comanda següent:

```
tar cf Gx_Fn.tar *.c *.h makefile *.ext_fitxers
```

on Fn és el número de Fase a lliurar. Per exemple, el grup 12 enviaria G12_F1.tar per al lliurament del *checkpoint* de la Fase 1.

Qualsevol entrega que no segueixi el format d'entrega no serà corregida.

En els lliuraments parcials també cal incloure la memòria que es va realitzant per a que es pugui valorar la seva evolució. A partir de la Fase 2 cal que aquesta contingui obligatòriament el disseny de la pràctica.

En els lliuraments finals també caldrà dipositar-la **memòria final en format PDF** en el tar. La memòria ha de constar, obligatòriament, dels punts que s'indiquen a continuació en l'Annex I.

Annex I: Contingut de la memòria

La memòria ha de ser una documentació externa correctament maquetada. Ha de contenir els següents apartats:

1. **Portada**
2. **Índex**
3. **Disseny:** explicació de com s'ha dissenyat i estructurat la pràctica. Es pot explicar per fases o de la pràctica en global. Ha d'incloure de manera clara i entenedora:
   a. Diagrames que expliquin processos s'han creat, les diferents comunicacions entre processos, etc.
   b. Estructures de dades usades i la seva justificació.
   c. Recursos del sistema utilitzats (*signals*, *sockets*, semàfors, *pipes*, etc.) amb la seva justificació.
   d. Opcionals implementats.
4. **Problemes observats i com s'han solucionat**.
5. **Estimació temporal**: temps dedicat en el desenvolupament total de la pràctica per a cadascun dels estudiants. Les categories a tenir en compte són:
   a. Investigació: temps dedicat a buscar eines i/o a aprendre components a implementar (fora del temps dedica a les sessions).
   b. Disseny: temps dedicat a dissenyar la pràctica
   c. Implementació: temps de codificació
   d. *Testing*: temps dedicat a fer proves
   e. Documentació: temps dedicat a escriure la memòria i a la documentació interna del codi (comentaris, etc.).
6. **Conclusions i propostes de millora**.
7. **(Opcional) Explicació de TOTS els noms dels processos de la pràctica. En quin tema ens hem basat aquest any per a fer l'enunciat? ☺**
8. **Bibliografia utilitzada** (en un format bibliogràfic correcte IEEE, APA, etc.): tant recursos bibliogràfics com links.

Es valorarà la redacció i la correctesa gramatical i ortogràfica. La memòria ha d'estar amb **numeració de pàgines.**

La memòria s'ha d'entregar en **format PDF** juntament amb el lliurament final però es recomana anar-la elaborant durant el transcurs de la pràctica.

Annex II: protocol de comunicació general

Per dur a terme la comunicació entre processos en diferents màquines, s'utilitzarà un protocol específic d'enviament de trames que s'explicarà a continuació. S'ha de tenir en compte que els missatges s'enviaran mitjançant *sockets* utilitzant una comunicació orientada a connexió.

En aquest protocol s'utilitzarà un **únic tipus** de trama. Sent aquestes de dimensió **fixa (256 Bytes)** i sempre formades pels següents 4 camps:

Type: Descriu el tipus de la trama en format hexadecimal 1 caràcter que contindrà un identificador de trama.

Data Length: Camp de 2 bytes en format numèric que serveix per indicar la llargària del camp data.

Data: Aquest camp serveix per emmagatzemar-hi valors o dades que ha d'enviar la trama. La llargària d'aquest camp depèn directament de la llargada total de la trama i el valor de data_length.

Checksum: Camp de 2 bytes que ajudarà a validar que la trama no ha tingut problemes. Aquest camp es calcularà sumant tots els bytes de la trama (sense contar el propi checksum) i fent mòdul 2^16. Serà important omplir cada trama enviada amb aquest checksum, i validar-lo en arribar als respectius servidors.

Timestamp: Camp de 4 bytes on desar el valor de time(NULL) en enviar una trama. Donat que cal seguir uns logs al sistema, el timestamp ens ajudarà a portar un recompte d'aquests, així com altres controls de debugging.

TYPE (1 Byte)	DATA_LENGTH (2 Bytes)	DATA (256 – 9 - X Bytes)	CHECKSUM (2 Bytes)	TIMESTAMP (4 Bytes)

És molt important de cara a garantir la bona comunicació amb el procés que la definició de l'estructura de la trama de comunicació sigui estrictament, en ordre i format, la descrita amb anterioritat. Això vol dir que qualsevol canvi de tipus dels camps o ordre dels mateixos dins de l'estructura a dissenyar farà que les trames rebudes o enviades a procés no compleixin el protocol de comunicació i, per tant, no funcioni aquesta comunicació correctament. Si la mida de les dades a enviar són inferiors a 256, caldrà omplir la trama amb el *padding* corresponent.

És del tot normatiu **l'enviament de trames serialitzades senceres** en una única escriptura al *socket*. És a dir, llegir amb un sol read, i escriure amb un sol write de 256B.

Per **totes les trames** els camps CHECKSUM i TIMESTAMP es calcularan de la mateixa manera:

CHECKSUM: Suma de tots els bytes de la trama % (2^16). Podeu fer servir el valor 65536 directament. (Suma bytes % 65536)

TIMESTAMP: Fent ús de la funció time(NULL) que retornarà el temps actual com el número de segons que han passat des de l'epoch de Unix. És un int amb signe, un total de 4B. Caldrà sempre indicar el timestamp d'enviament d'una trama.

Per simplicitat, no s'han inclòs a les explicacions individuals de cada trama donat que es faría repetitiu, però han de ser-hi a totes. Tot el que està entre <> ha de substituir-se pel valor corresponent. **Els <> NO van a la trama.**

NOVES CONNEXIONS A FLECK - GOTHAM

Fleck → Gotham

Trama per demanar una connexió al servidor Gotham.

- TYPE: 0x01
- DATA_LENGTH: Llargària de les dades.
- DATA: <*userName*>&<*IP*>&<*Port*>

Gotham → Fleck

Trama OK connexió.

- TYPE: 0x01
- DATA_LENGTH: 0
- DATA: Buit

Trama KO connexió. S'envia en el cas que no s'hagi pogut establir la connexió.

- TYPE: 0x01
- DATA_LENGTH: Llargària de les dades.
- DATA: *CON_KO*

NOVES CONNEXIONS A WORKER (Harley/Enigma) – GOTHAM

Worker ➔ Gotham

Trama per demanar una connexió al servidor Gotham.

- TYPE: 0x02
- DATA_LENGTH: Llargària de les dades.
- DATA: <*workerType*>&<*IP*>&<*Port*>

Gotham ➔ Worker

Trama OK connexió.

- TYPE: 0x02
- DATA_LENGTH: 0
- DATA: Buit

Trama KO connexió. S'envia en el cas que no s'hagi pogut establir la connexió.

- TYPE: 0x02
- DATA_LENGTH: Llargària de les dades.
- DATA: *CON_KO*

PETICIÓ DISTORT FILE A GOTHAM – FLECK

Fleck ➔ Gotham

Quan Fleck vulgui iniciar una nova petició de distorsió, consultarà amb Gotham primer a qui ha de fer-la. Per fer-ho farà una primera trama amb:

- TYPE: 0x10
- DATA_LENGTH: Llargària de les dades.
- DATA: *<mediaType>&<FileName>*

Gotham ➔ Fleck

En cas que el nom del fitxer coincideixi amb el mediaType i hi hagi algún worker disponible d'aquell tipus de media (>=1) Gotham contestarà amb:

- TYPE: 0x10
- DATA_LENGTH: Llargària de les dades.
- DATA: *<IP>&<port>*

En cas contrari, si no hi ha workers d'aquell tipus, o la petició de distort no és del tipus de media correcte, contestarà amb:

- TYPE: 0x10
- DATA_LENGTH: Llargària de les dades.
- DATA: *DISTORT_KO*

Per saber si és del tipus de media correcte, comproveu l'extensió del fitxer.

PETICIÓ PER RESUMIR DISTORT FILE A GOTHAM – FLECK

Fleck ➔ Gotham

Quan Fleck hagi detectat la caiguda del worker a qui estava enviant el fitxer, o de qui l'estava rebent, farà una petició de Gotham per que li torni a assignar un nou worker per seguir amb la tasca.

- TYPE: 0x11
- DATA_LENGTH: Llargària de les dades.
- DATA: *<mediaType>&<FileName>*

Gotham ➔ Fleck

En cas que el nom del fitxer coincideixi amb el *mediaType* (la extensió del fitxer es correspon amb el tipus de media i el *mediaType* es vàlid) i hi hagi com a mínim un worker disponible d'aquell tipus de media Gotham contestarà amb la IP i port del Worker principal d'aquell tipus de media.

- TYPE: 0x11
- DATA_LENGTH: Llargària de les dades.
- DATA: *<IP>&<port>*

En cas contrari, si no hi ha workers d'aquell tipus contestarà amb:

- TYPE: 0x11
- DATA_LENGTH: Llargària de les dades.
- DATA: *DISTORT_KO*

O si la petició de distort no és del tipus de media correcte (extensió != mediaType o mediaType no existeix) amb:

- TYPE: 0x11
- DATA_LENGTH: Llargària de les dades.
- DATA: *MEDIA_KO*

DISTORT FILE - NOVES CONNEXIONS A WORKER (Harley/Enigma) – FLECK

Fleck → Worker

Trama per demanar una connexió al Worker de media. Aquesta ja inclou información del fitxer a distorsionar.

- TYPE: 0x03
- DATA_LENGTH: Llargària de les dades.
- DATA: *<userName>&<FileName>&<FileSize>&<MD5SUM>&<factor>*

On:

- FileSize: en bytes expressat en format ASCII
- MD5SUM: 32 caràcters del MD5SUM
- Factor: El factor de compressió en funció del tipus d'arixu en ASCII.
  - L'escala en el cas d'imatges.
  - El interval de temps en ms en cas d'audio.
  - El llindar en cas de text.

Worker → Fleck

Trama OK connexió. Indicant que pot començar a enviar l'arxiu.

- TYPE: 0x03
- DATA_LENGTH: 0
- DATA: Buit

Trama KO connexió. S'envia en el cas que no s'hagi pogut establir la connexió. El Fleck desistirà d'enviar el arxiu per distorsionar.

- TYPE: 0x03
- DATA_LENGTH: Llargària de les dades.
- DATA: *CON_KO*

DISTORT FILE

Fleck → Worker | Worker → Fleck

Trama per a enviar un fitxer de Fleck a Worker o quan el Worker contesta amb el fitxer ja distorsionat, de Worker a Fleck.

Al connectar-se, ja s'envia la informació necessària sobre la mida del fitxer, factor de compressió o md5sum al origen.

Quan el Worker contesti amb la imatge distorsionada a Fleck, sí caldrà enviar una primera trama amb la informació del fitxer:

- TYPE: 0x04
- DATA_LENGTH: Llargària de les dades.
- DATA: *<FileSize>&<MD5SUM>*

No envía el nom del fitxer donat que aquest es queda igual.

Posteriorment, s'enviarà les dades del fitxer.

- TYPE: 0x05
- DATA_LENGTH: Llargària de les dades.
- DATA: *dades_del_fitxer*

Cal tenir en compte que caldrà enviar N trames de 256 bytes per a cada fitxer.

Worker ➔ Fleck | Fleck ➔ Worker

En acabar l'enviament d'un fitxer, caldrà comprovar que el md5sum del fitxer rebut és el mateix que el d'orígen. Caldrà contestar al emissor del fitxer amb la trama següent:

Comprovació del MD5SUM correcta.

- TYPE: 0x06
- DATA_LENGTH: Llargària de les dades.
- DATA: *CHECK_OK*

Comprovació de MD5SUM incorrecta.

- TYPE: 0x06
- DATA_LENGTH: Llargària de les dades.
- DATA: *CHECK_KO*

DESCONNEXIÓ

Fleck → Gotham

Trama per notificar al Servidor d'una desconnexió.

- TYPE: 0x07
- DATA_LENGTH: Llargària de les dades.
- DATA: <*userName*>

Fleck → Worker

Trama per notificar al Servidor d'una desconnexió.

- TYPE: 0x07
- DATA_LENGTH: Llargària de les dades.
- DATA: <*userName*>

Worker → Gotham

Trama per notificar al Servidor d'una desconnexió.

- TYPE: 0x07
- DATA_LENGTH: Llargària de les dades.
- DATA: <*mediaType*>

ASSIGNAR NOU WORKER PRINCIPAL – Gotham → Worker

Quan Gotham hagi de triar un worker principal nou, enviarà al worker triat la trama següent:

- TYPE: 0x08
- DATA_LENGTH: 0.
- DATA: Buit

Procediment de detecció de trames errònies

Si qualsevol dels processos del sistema MR. J. rep una trama que no es correspon a cap dels formats definits o amb un Checksum que no és vàlid caldria que s'enviés una trama de retorn per poder detectar l'error amb la següent trama:

- TYPE: 0x09
- DATA_LENGTH: 0.
- DATA: Buit

HEARTBEAT (Opcional)

Opcionalment per detectar la desconnexió d'un servidor o client, podeu fer ús de una trama com a *heartbeat* una "batejada de cor" que anirà enviant periòdicament. No és necessària per resoldre la pràctica, però està permesa.

- TYPE: 0x12
- DATA_LENGTH: 0
- DATA: *Buit*

Annex III: Llibrería SO_compression.h

Se us proporcionarà un fitxer .h i un fitxer .o per que pugueu fer servir la llibreria. Aquesta es compilarà amb els demés mòduls fent per exemple, fent:

gcc Harley.c -c -Wall -Wextra

gcc Harley.o SO_compression.o ..<els demés .o>.. -o Harley -Wall -Wextra -lpthread

Aquesta llibreria té 3 funcions principals:

Compressió d'Imatge

Per la compressió d'imatge, disposarem de la funció:

int SO_compressImage(*char* **input_image*, *int scale_factor*);

Aquesta rep com a arguments, el nom de la imatge a comprimir amb la seva extensió, i un factor d'escala.

La imatge pot ser .png, .jpg, .jpeg, .bpm, .tga i la funció reduirà la seva mida en píxels, reduint així la seva mida en B total. Per fer-ho, dividirà les dimensions de la imatge per el factor d'escala. Si la imatge mesura 780x240 i el factor d'escala es 3 dividirà les dues entre 3, resultant en una imatge de 260x80.

Per poder fer això, necessita que existeixi el fitxer sencer, i el que farà serà **substituir el fitxer original per el comprimit**. Per tant, un cop acabada la crida, el fitxer input_image serà substituït per el comprimit.

A més, retorna un enter, que és el status de la compressió. Teniu al .h els diferents status possibles, però qualsevol número negatiu significarà un error. Teniu detallats quins errors pot ser al .h, per poder debuggar.

Compressió d'Audio

Per la compressió d'audio es farà servir la funció:

int SO_compressAudio(*char* **input_file*, *int interval_ms*);

Com veiem, també rep aquest input_file, i també el substituirà el fitxer d'entrada per el comprimit. També retorna aquest status de la compressió.

A diferència de les imatges, que es comprimeixen reduint-ne la mida, en el cas de l'àudio el procés consisteix a escurçar la seva durada. Això s'aconsegueix eliminant petits fragments d'àudio de manera intermitent, amb l'objectiu que el contingut sigui encara comprensible, sempre que no s'eliminin massa parts.

Per a dur a terme aquesta eliminació, el sistema rep un interval en mil·lisegons que determinarà la periodicitat de les parts eliminades. Per exemple, si l'interval és de 100ms, els primers 100ms de l'àudio es mantindran intactes, però els següents 100ms s'eliminaran. A continuació, els propers 100ms es conserven, i així successivament de manera intermitent fins al final de l'àudio. Aquest procés redueix la longitud total del fitxer d'àudio, però conserva prou fragments perquè sigui comprensible.

Esborrar Arxiu

Per últim, la funció d'esborrar arxiu, és molt senzilla i no requereix de gaire explicació addicional:

void SO_deleteFile(*char *file_name*);

Donat el path d'un arxiu, l'esborra.

A més, veureu detallats els codis d'error que poden retornar aquestes funcions, així com tot els comentaris en detall de com funcionen.

Per debugar, també veureu una constant anomenada ADDITIONAL_DELAY que podeu incrementar o decrementar per fer que es trigui més o menys en fer la compressió. Les compressió es per "trossos", podeu imaginar-vos un bucle. El additional delay farà que cada iteració del bucle trigui més.

Annex IV: Guia per a un bon disseny de una pràctica

Quan treballes en un sistema complex com el Mr. J System, és essencial aplicar un bon disseny per gestionar la complexitat de manera eficient i escalable. El disseny ha de contar amb conceptes clau com la modularització, la concurrència i la robustesa. Com a enginyers, és fonamental tenir clar el disseny abans de començar la implementació, ja que un plantejament ben estructurat ens permetrà detectar possibles problemes, optimitzar recursos i garantir que el projecte sigui mantenible a llarg termini. A continuació, es presenten punts generals a tenir en compte en qualsevol projecte en C, centrats en els aspectes importants d'un sistema operatiu i com es poden aplicar al context del projecte.

Modularització: Disseny de mòduls compartits

En projectes complexos com el **"Mr. J System"**, modularitzar correctament el codi és crucial per garantir la claredat, la reutilització i la mantenibilitat. No es tracta només de dividir el sistema en processos independents com **Fleck**, **Gotham**, **Enigma** i **Harley**, sinó també d'identificar funcionalitats comunes i encapsular-les en **mòduls compartits**. Aquests mòduls, utilitzats per diferents processos, són essencials per evitar la duplicació de codi i assegurar una major eficiència.

En qualsevol projecte de certa complexitat, és fonamental determinar quines parts del codi poden ser comunes entre els diferents processos i encapsular-les de manera independent. Si diverses funcionalitats es repeteixen en diferents punts del sistema, com la manipulació de dades o l'accés a recursos compartits, és molt més eficient organitzar-les en mòduls reutilitzables. Això facilita la mantenibilitat del sistema, ja que redueix la duplicació de codi i centralitza els canvis: qualsevol modificació en un mòdul es propagarà automàticament als processos que l'utilitzin, estalviant temps i errors potencials.

Cada **mòdul** hauria de tenir una **responsabilitat clara** i ser fàcilment reutilitzable sense modificacions per part dels processos que l'utilitzen. Això no només millora l'estructura del codi, sinó que permet **centralitzar actualitzacions** i mantenir un sistema més net, escalable i fàcil de gestionar.

Un bon disseny modular en C implica la separació del codi en **fitxers .h** i **fitxers .c**. Els **fitxers .h** contenen declaracions de funcions, constants i tipus de dades que poden ser compartits entre mòduls o processos, mentre que els **fitxers .c** implementen la lògica funcional. Aquesta separació no només facilita l'organització del projecte, sinó que també permet que el compilador gestioni millor les dependències, cosa que millora la integració i compilació del codi.

A l'hora de modularitzar, és important seguir aquests **criteris**:

1. **Reutilització:** Identifica les funcionalitats necessàries en diferents processos i encapsula-les en mòduls reutilitzables. Això evita la duplicació de codi i fa que el sistema sigui més fàcil de mantenir.

2. **Clarificació de responsabilitats:** Cada mòdul ha de tenir una responsabilitat única i clara. No barregis funcionalitats diferents dins d'un mateix mòdul per evitar confusions i complicar la seva reutilització.

3. **Independència dels mòduls:** Els mòduls no han de dependre fortament entre ells. Si hi ha dependències, han d'estar ben definides i documentades per evitar problemes futurs d'integració o manteniment.

Finalment, és important evitar l'ús de **variables globals** dins dels mòduls compartits. Les variables globals haurien de residir únicament en els processos principals com **Fleck**, **Gotham**, **Enigma** i **Harley**, mentre que els mòduls han de funcionar únicament amb funcions que rebin tots els arguments necessaris com a paràmetres. Això no només millora la modularitat del codi, sinó que també ajuda a evitar conflictes quan aquests mòduls són utilitzats per diversos processos en diferents contextos del sistema.

Concurrència: Apropar-se al món real

En el disseny d'un sistema operatiu o d'un sistema complex com el **"Mr. J System"**, és fonamental entendre que, en el món real, moltes tasques han de succeir simultàniament per garantir el funcionament correcte del sistema. La **concurrència** permet que múltiples usuaris, processos o recursos treballin alhora, cosa que fa que el sistema pugui respondre a múltiples peticions de forma eficient.

Per exemple, si diversos usuaris estan fent peticions al mateix temps (com els processos **Fleck**), la concurrència assegura que aquestes es puguin atendre simultàniament, evitant colls d'ampolla i millorant l'eficiència general del sistema. A més, és possible que un mateix procés hagi de rebre, processar i enviar respostes al mateix temps. Això exigeix un disseny que permeti gestionar diverses operacions paral·leles sense interferir entre elles.

A l'hora de dissenyar un sistema concurrent, pots fer-te preguntes per garantir que estàs prenent decisions ben fonamentades com:

1. **Com gestionaràs múltiples connexions simultànies?**
   Quina és la millor estratègia per gestionar múltiples usuaris alhora? Com assegураràs que el sistema no es bloqueja si augmenta el nombre de peticions?
2. **Com garantiràs la robustesa del sistema?**
   Què passarà si un fil o procés falla mentre està atenent una petició? Com assegураràs que el sistema pugui continuar funcionant sense

interrupcions? Quines tècniques de recuperació automàtica pots implementar per mantenir la continuïtat del servei?

3. **Com controlaràs l'accés a recursos compartits**
   Si diversos fils o processos han d'accedir al mateix recurs alhora, com gestionaràs aquest accés de manera segura? On creus que podrien sorgir conflictes d'accés simultani?
4. **Com optimitzaràs l'ús dels recursos del sistema?**
   Estàs fent un ús òptim de la CPU i la memòria? Com pots assegurar-te que no estàs generant més processos o fils dels necessaris? Quin impacte pot tenir sobre el sistema la creació massiva de fils o processos? Com pots evitar la saturació de recursos
5. **Quin és el millor enfocament per cada part del sistema?**
   Cada part del sistema necessita el mateix tipus de solució concurrent? O és possible que diferents parts del sistema requereixin enfocaments diferents? Què seria més òptim per les diferents tasques que s'han de realitzar?

El disseny concurrent d'un sistema no es tracta només de garantir que funcioni correctament, sinó que funcioni **bé sota càrrega** i en condicions **reals**. La clau està en trobar solucions que equilibren **robustesa** i **eficiència**, permetent que el sistema sigui flexible i resisteixi diferents situacions. Això implica reflexionar contínuament sobre les preguntes adequades durant el disseny de cada part del sistema, per trobar l'enfocament més adequat en funció dels requisits de **rendiment** i **estabilitat**.

Com a enginyer, has de ser capaç d'analitzar **críticament** cada component del sistema i adaptar-te als requisits particulars de cada procés. Això significa buscar el millor compromís entre **rendiment**, **robustesa** i **eficiència**, tenint en compte que el que és òptim per una part del sistema pot no ser-ho per una altra. La **concurrència** és una eina poderosa, però la seva correcta aplicació depèn de comprendre a fons les necessitats del sistema. La solució perfecta en un context pot no ser aplicable a altres escenaris, i part de la teva tasca és ajustar i adaptar aquestes solucions per assegurar que cada part funcioni de manera òptima en el seu context particular.