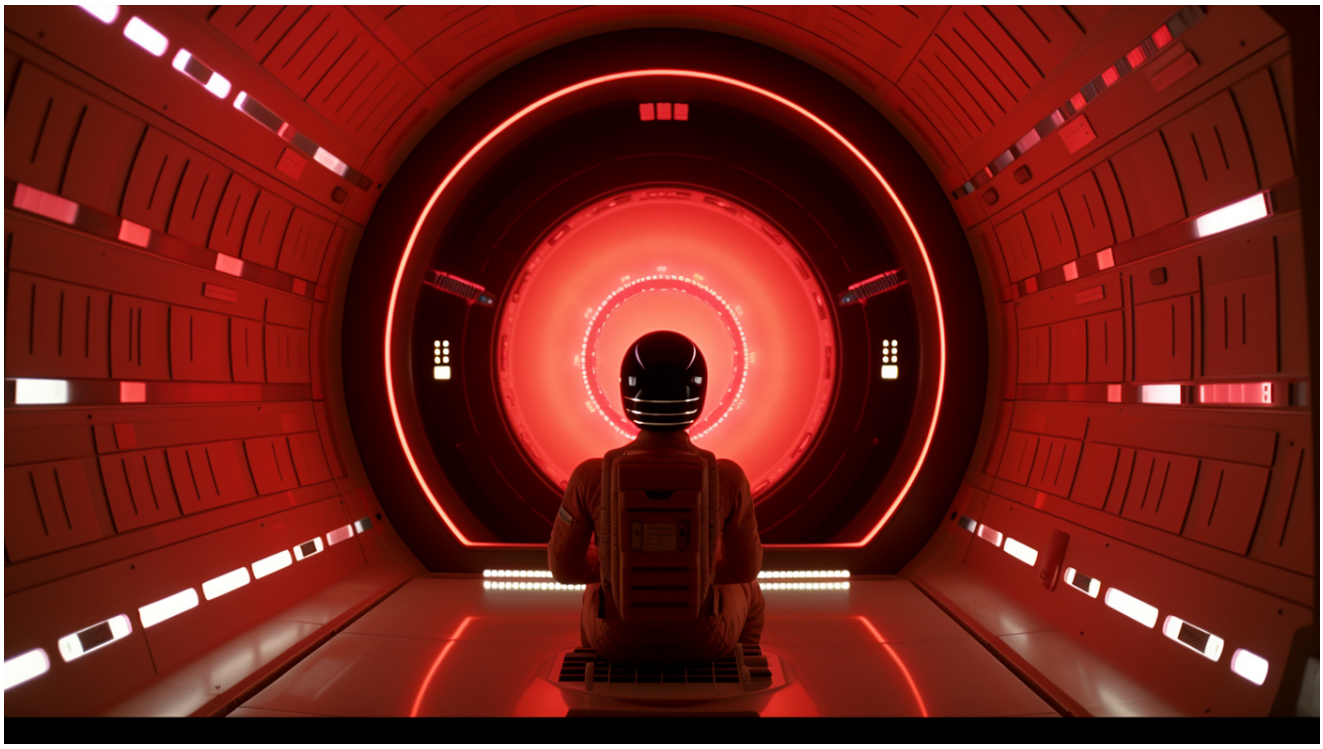
# HAL-9000-System



## Sistemes Operatius – Curs 23/24

Carlos Romero Rodríguez

Aniol Vergés Herrera

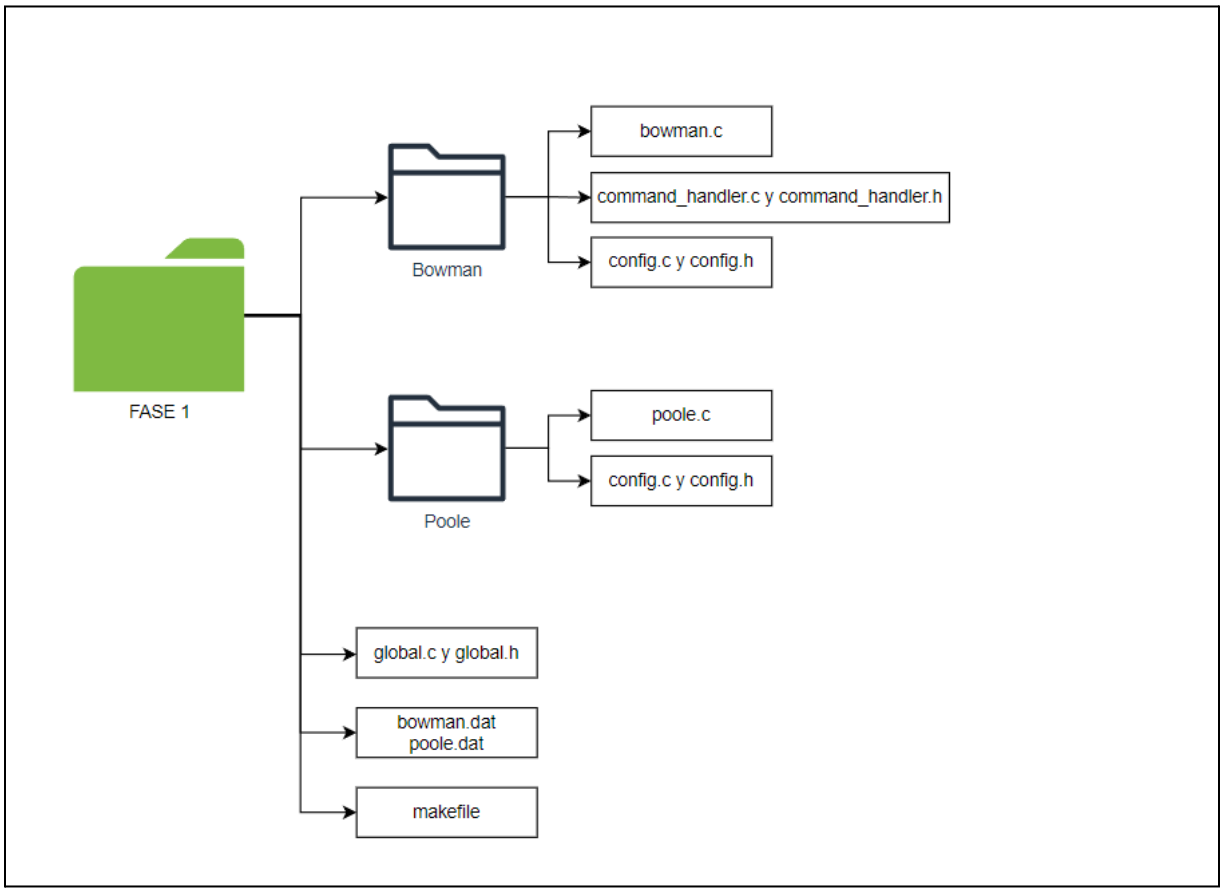
# Índex

# Disseny

## Fase 1

A la Fase 1 del nostre projecte, vam establir les bases per a un sistema altament modular i escalable. Aquest enfocament ens permetrà facilitar l'ampliació futura del sistema amb noves funcionalitats i components. Vam organitzar el sistema en subdirectoris, assignant a cada entitat (Bowman per als usuaris i Poole pels servidors, amb la previsió d'afegir posteriorment un Balancejador de Càrrega anomenat Discovery i un Subprocés d'Estadístiques conegut com a Monòlit) el seu propi espai dins de l'estructura de carpetes. Aquesta organització no només promou la encapsulació i la separació de responsabilitats, sinó que també simplifica la integració de components addicionals en les fases posteriors del desenvolupament.

L'estructura de directoris és com segueix:




*Imatge de la organització dels arxius*

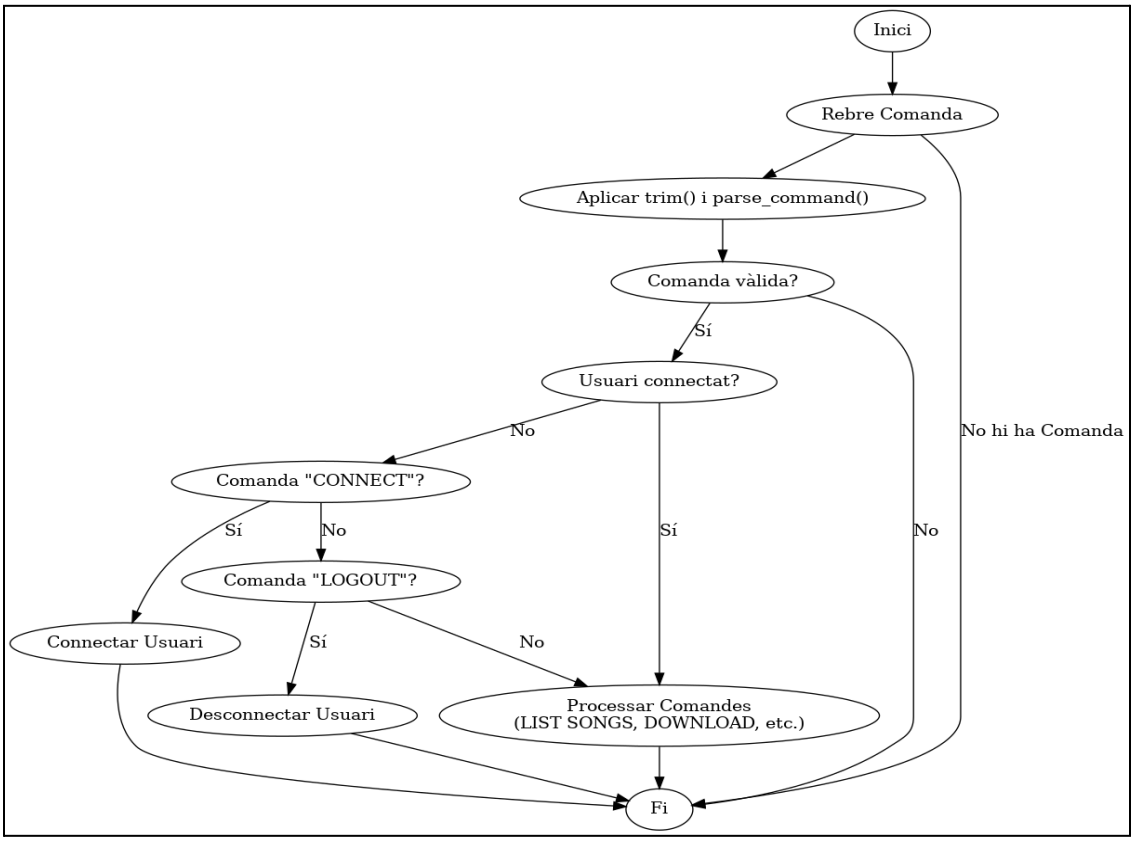
**Carpeta Bowman**: Aquesta carpeta conté tots els elements requerits per a la gestió i execució de les funcionalitats dels usuaris del sistema. Inclou:

- **bowman.c**: El punt d'entrada del programa, el qual manté una lògica mínima, delegant la majoria de responsabilitats a submòduls especialitzats.
- **command_handler.c i command_handler.h**: Mòduls encarregats d'interpretar i validar les comandes introduïdes per l'usuari, garantint que aquestes siguin processades de manera eficient i correcta.

Donat que les possibles comandes que l'usuari pot introduir son les següents:

| | |
|---|---|
| **CONNECT** | Connecta l'usuari al sistema. Primer es connectarà al balancejador de càrrega (Discovery) i, automàticament i de manera transparent per l'usuari, a un servidor Poole. |
| **LOGOUT** | Desconnecta l'usuari del sistema. |
| **LIST SONGS** | Llista totes les cançons disponibles del servidor Poole on està connectat l'usuari. |
| **LIST PLAYLISTS** | Llista totes les llistes de reproducció disponibles del servidor Poole on està connectat l'usuari. |
| **DOWNLOAD <SONG/PLAYLIST>** | Descarrega una cançó o una llista d'aquestes. |
| **CHECK DOWNLOADS** | Consulta l'estat de les descàrregues en curs, en forma de barres de progrés. |
| **CLEAR DOWNLOADS** | Neteja les descàrregues ja acabades de la llista de descàrregues. |

Per aquest motiu hem seguit el següent flux de comandes per tal de poder garantir el correcte funcionament del programa:




Donat que per especificacions de l'enunciat les comandos introduides per l'usuari poden tenir més d'un espai entre paraules, o espais a l'inicial i/o al final, hem hagut d'afegir una mica més de lógica al rebre la comanda. Aquest problema es soluciona amb aquestes dues funcions d'aquest mateix TAD:

La funció **parse_command()** és responsable de parsejar la comanda eliminant els espais addicionals entre les paraules. Això es fa utilitzant strtok per separar la cadena basant-se en els espais i després combinar les paraules parsejades en una nova cadena sense espais sobrants. Per exemple, converteix "LIST SONGS" en "LIST SONGS".

Per altra banda, la funció **trim_whitespace()** s'encarrega de eliminar qualsevol espai innecessari que pugui haver al principi o al final de la comanda. Això s'aconsegueix iterant a través de la cadena des de l'inici i el final, eliminant els espais fins a trobar un caràcter no espai. Aquesta funció garanteix que la cadena que es passa a parse_command estigui lliure d'espais al principi i al final, preparant-la per a la seva posterior anàlisi i processament.

- **config.c i config.h**: Responsables de llegir i emmagatzemar la configuració inicial que és essencial per a l'operativitat de Bowman.

**Carpeta Poole**: Aquesta carpeta alberga els components necessaris per al funcionament dels servidors de música. Conté:

- **poole.c**: El fitxer principal que executa la lògica del servidor, gestionant les peticions dels usuaris i la distribució de contingut.
- **config.c i config.h**: Similar a Bowman, aquests fitxers són fonamentals per configurar el servidor al seu inici.

**Arxius Globals**: Amb global.c i global.h, definim les variables i les constants que seran utilitzades de manera transversal en tot el projecte.

**Fitxers de Configuració**: bowman.dat i poole.dat són els fitxers de dades que proporcionen la informació persistent necessària per a cada entitat (tals com nom, ip, port,...)

**Makefile**: Aquest fitxer és essencial per a la compilació del projecte, garantint que l'ensamblatge del codi sigui consistent i fiable.

## Fase 2

En aquesta fase, el projecte introdueix el balancejador de càrrega **Discovery**. Aquest nou component esdevé un eix central en la gestió de les connexions dels usuaris Bowman als servidors Poole. Quan un usuari inicia una sessió de connexió, el sistema Bowman crea i envia una trama al balancejador Discovery mitjançant sockets. Discovery, al rebre la informació de l'usuari en un frame, procedeix a determinar quin servidor Poole té menys càrrega per assignar l'usuari, garantint així una distribució equitativa de recursos i una optimització de l'ús del sistema.

La creació i enviament de trames es maneja a través de les funcions **frame_creator**, **send_frame** i **receive_frame**, les quals han estat meticulosament dissenyades per a gestionar la comunicació entre els diferents components del sistema. Aquestes funcions permeten un intercanvi de dades estructurat i eficient, essencial per al correcte funcionament del balancejador de càrrega.

A més a més, en aquesta fase s'implementen les comandes LIST SONGS, LIST PLAYLISTS i LOGOUT. La comanda **LOGOUT** s'encarrega de notificar a Discovery que l'usuari s'ha desconnectat, permetent a Discovery actualitzar el recompte d'usuaris del servidor Poole adient.

Hem obtat per la opció 1 que facilitaba l'enunciat:

- Bowman avisa a Poole, el qual avisa a Discovery. Discovery, llavors, esborra a Bowman de la llista i envia un ACK a Poole. Poole envia un ACK a Bowman, el qual es desconnecta.

D'altre banda, les funcions **LIST SONGS** i **LIST PLAYLISTS** són peticions que Bowman realitza una vegada establerta la connexió amb Poole, i requereixen que el servidor llisti el contingut del seu directori a través de l'estructura DIR *, enviament la informació en una o més trames segons la mida total en bytes del contingut.

És important destacar que, tenint en compte la possibilitat que l'usuari finalitzi la sessió amb un CTRL+C en qualsevol moment, el sistema està dissenyat per manejar aquesta eventualitat de forma gràcil, alliberant tota la memòria dinàmica utilitzada de manera adequada i segura.

Per preparar-nos per a les fases futures, com la Fase 3, on es permetrà la descàrrega de cançons i llistes de reproducció completes, hem implementat un switch que determina el tipus de capçalera que porta la trama rebuda del servidor. Això assegura que el socket únic de l'usuari (és a dir, el socket que manté mitjançant una connexió TCP el servidor y l'usuari connectats) pugui manejar diversos tipus de trames, establint així una base sòlida per a la gestió avançada de les peticions en les Fases 3 i 4.

```
597  void *receive_frames(void *args) {
598      thread_receive_frames trf = *(thread_receive_frames *)args;
599
600      while (!bowman_sigint_received) {
601          Frame response_frame;
602          if (receive_frame(*trf.poole_socket, &response_frame) <= 0) {
603              continue;
604          }
605          //RECIBIR EL TAMAÑO DE LIST_SONGS A MOSTRAR
606          if (!strncasecmp(response_frame.header_plus_data , "LIST_SONGS_SIZE", response_frame.header_length)) {
607              handleListSongsSize(response_frame.header_plus_data + response_frame.header_length);
608          }
609          else if (!strncasecmp(response_frame.header_plus_data , "SONGS_RESPONSE", response_frame.header_length)) {
610              handleSongsResponse(response_frame.header_plus_data + response_frame.header_length);
611          }
612          else if (!strncasecmp(response_frame.header_plus_data , "LIST_PLAYLISTS_SIZE", response_frame.header_length)) {
613              handleListPlaylistsSize(response_frame.header_plus_data + response_frame.header_length);
614          }
615          else if (!strncasecmp(response_frame.header_plus_data , "PLAYLISTS_RESPONSE", response_frame.header_length)) {
616              handlePlaylistsResponse(response_frame.header_plus_data + response_frame.header_length);
617          }
618          else if (!strncasecmp(response_frame.header_plus_data, "NEW_FILE", response_frame.header_length)) {
619              handleNewFile(response_frame.header_plus_data + response_frame.header_length);
620          }
621          else if (!strncasecmp(response_frame.header_plus_data, "FILE_DATA", response_frame.header_length)) {
622              handleFileData(response_frame.header_plus_data + response_frame.header_length);
623          }
624          else if (!strncasecmp(response_frame.header_plus_data , "LOGOUT_OK", response_frame.header_length)) {
625              printF(GREEN, "Logout successful\n");
626              disconnect_notification_to_discovery(trf.username, trf.discovery_ip, trf.discovery_port);
627              pthread_exit(NULL);
628          }
629          else if (!strncasecmp(response_frame.header_plus_data , "LOGOUT_KO", response_frame.header_length)) {
630              printF(RED, "Logout failed\n");
631              pthread_exit(NULL);
632          }
633          else if (!strncasecmp(response_frame.header_plus_data , "POOLE_SHUTDOWN", response_frame.header_length)) {
634
635              printF(RED, "\nPoole server has shutdown. Reconnecting to Discovery...\n");
636
637              Frame logout_frame = frame_creator(0x06, "SHUTDOWN_OK", "");
```

## Fase 3

Un cop acabades les diferents connexions entre els processos, havíem de començar a gestionar les diferents comandes de Bowman, per tal de que el sistema funciones correctament. El nostre disseny ha estat el següent:

Com s'ha explicat en els apartats anteriors, amb la funció "handle_bowman_comand" es gestionen les diferents comandes que pot entrar l'usuari. Les comandes a implementar en aquesta fase varen ser:

- DOWNLOAD "SONG_NAME"
- CHECK DOWNLOADS
- CLEAR DOWNLOADS

**COMANDA DOWNLOAD**

Per la implementació d'aquesta comanda a Bowman es segueixen els següents passos:

1. Verificació del tipus de descàrrega

   Primer hem de saber de si es tracta d'un fitxer ".mp3" o una llista de cançons, ja que segons del que es tracti realitzarem uns passos o uns altres. Aquest procés es fa senzillament mirant el nom de la descarrega, en cas de que aquest tingui la extensió ".mp3" al final del seu nom, es tractarà d'un fitxer de musica, en cas contrari, ho tractarem com a llista de cançons.

2. Creació i enviament de trames

   Un cop hem verificat el tipus de descàrrega, creem les trames necessàries mitjançant el sistema de creació i enviament de trames ja creat i explicat en aquest document anteriorment. En el cas de que es tracti d'una cançó, el header de la trama serà "DOWNLOAD_SONG" i serà de tipus "0x03", i en el cas de que es tracti d'una llista, el header de la trama serà "DOWNLOAD_LIST" amb el mateix tipus. En ambdós casos el camp de les dades contindrà el nom del que el client desitja descarregar.

3. Inici del procés de descarrega

   En aquest moment la comanda de descarrega ja ha estat enviada a Poole i es notifica al usuari que la descarrega ja ha començat. Rebem una trama amb header "NEW_FILE" al thread que tenim escoltant de Poole, i un cop la rebem, creem un thread per gestionar la descarrega de la cançó. La trama que rebem de Poole conté la seguent informació: Nom del arxiu, mida, MD5SUM i un ID generat automàticament per Poole.

4. Creació del fitxer

   En el thread creat especialment pel fitxer, guardem la informació necessària enviada per Poole en la trama, un cop la tenim guardada procedim a la creació del fitxer mitjançant el nom enviat per Poole, ja guardat. El fitxer s'obre amb mode escriptura, en cas de que no existeixi es crea i en cas de que existeixi en borra el seu contingut. Els permisos del fitxer son els necessaris per a que l'usuari el pugui llegir i escriure.

5. Recepció i escriptura de dades

   Mentres el thread anterior s'encarrega de la creació del fitxer, el thread que tenim escoltant de Poole, va rebent trames amb header "FILE_DATA", aquestes trames tenen com a dades els bytes del fitxer que estem descarregant. Un cop rebem aquests bytes, els "encuem" a la cua global que tenim a Bowman, per tal de que les funcions puguin accedir-hi. Per "encuar" els missatges a la cua, utilitzem l'ID de la cançó que estem descarregant, així el thread que s'encarrega de rebre la informació, sap a quina bústia ha d'accedir per tal de rebre les dades del fitxer. Per tant, quan el thread que s'encarrega de l'escriptura del fitxer acabi de crear el fitxer, es posarà a llegir de la cua de missatges amb l'ID de la seva respectiva cançó, i els va escribint dins el fitxer creat enteriorment, fins a arribar a la mida total, rebuda anteriorment (fins que s'acabi de descarregar completament).

6. Finalització del procés

   Un cop s'ha escrit tot el fitxer, el thread  allibera tota la memoria que ja no s'usarà i tanca el file descriptor emparat per l'escriptura del fitxer. Un cop fet això el thread acaba, completant així la descarrega del fitxer.

Per la implementació d'aquesta comanda a Poole es segueixen els següents passos:

1. Rebre trama

   Per l'inici de la descarrega, primer ha d'arribar a Poole una trama des de Bowman on el header s'especifiqui "DOWNLOAD_SONG". Un cop rebem la trama a Poole, llencem un thread, aquest serà el thread que s'encarregarà de enviar els bytes necessaris cap a Bowman. La trama provinent de Bowman la rebem al thread encarregat del Bowman corresponent (client_handler)

2. Enviament de confirmació

   Un cop dins el thread que s'encarregarà de la cançó, és necessari buscar la cançó al directori, per comprovar si realment existeix, això ho fem amb la funció "searchSong". Si la cançó no existeix dins el directori, es cancel·la la descarrega. Un cop hem localitzat el fitxer que haurem d'enviar a Bowman, aconseguim diferents variables que haurem d'enviar, com poden ser el MD5SUM o la mida del fitxer. Per aconseguir l'MD5SUM ho fem mitjançant la funció "getFileMD5", aquesta funció crearà un "pipe" i seguidament realitzarà un "fork", el procés fill s'encarregarà d'executar la comanda "md5sum" mitjançant "execlp" i la enviarà al procés pare a traves de la "pipe", finalment el pare tanca el "pipe" de lectura del fill i el fill para la seva execució. El "fork" és

necessari en aquest cas ja que en el cas de que la comanda md5sum fallés, podria portar problemes al procés principal, d'aquesta manera ens assegurem que el procés principal es mantingui exacte tot i que el md5sum falli. Finalment enviem la trama amb header "NEW_FILE" a Bowman, amb totes les dades necessàries ja esmentades anteriorment.

3. Preparació per enviament de bytes

   Abans de començar l'enviament dels bytes cap a Bowman, fem els següent. Primer hem d'obrir el fitxer que volem enviar, per tal d'obtenir el seu file descriptor i a partir d'aquest anar llegint el bytes necessaris. Un cop tenim el file descriptor hem de realitzar un càlcul per tal de saber quants bytes podem enviar a cada trama, ja que el límit esta en 256. El càlcul realitzat ha estat el següent:

   max_bytes_to_send = HEADER_MAX_SIZE (253) - strlen("FILE_DATA") - id_length - 1;

   Un cop tenim el file descriptor i els bytes que podem enviar a cada trama, comencem l'enviament.

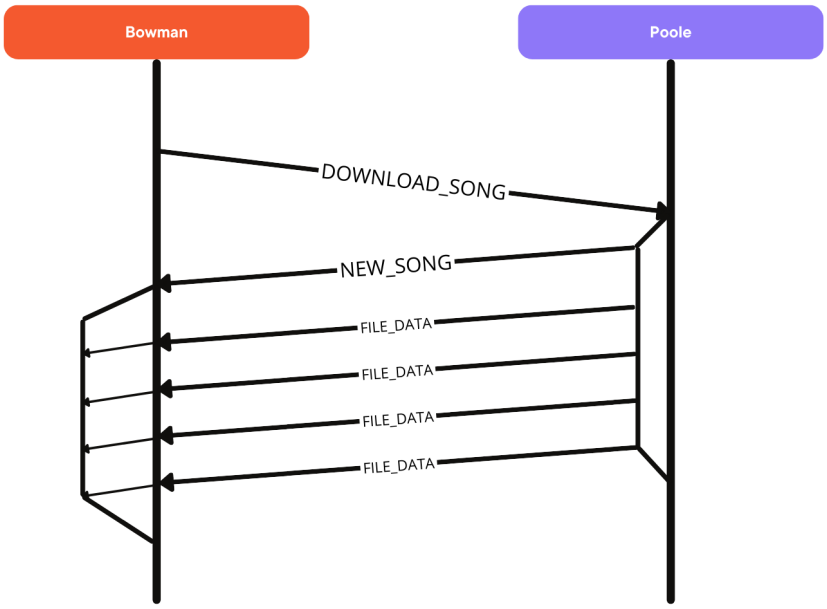
4. Enviament de fitxer

   Creem un bucle amb la condició de que només pari quan els bytes enviats siguin iguals als bytes totals del fitxer, així sabrem que s'ha enviat tot el fitxer. Al inici del bucle hem de mirar quants bytes podem enviar, ja que no sempre seran els que hem calculat anteriorment, és molt probable que la ultima volta s'enviïn menys bytes dels que hem calculat, per tant hem de realitzar el següent càlcul: Si el numero de bytes que enviem per volta més els totals que hem enviat, supera el numero de bytes totals del fitxer, en aquesta volta haurme d'enviar el numero total de bytes menys el numero de bytes que hem enviat (és a dir que enviarem els que falten). Llavors l'únic que queda fer és llegir del fitxer els bytes que pertoquin a la volta, crear la trama amb el header "FILE_DATA" i enviar-la. Finalment sumarem els bytes enviats al total de bytes ja enviats i seguirem amb la següent volta.

5. Finalització de la descàrrega

   Quan la descàrrega hagi acabat, tancarem el thread i alliberarem la memòria utilitzada.

Exemple gràfic de com seria el timeline de la descàrrega d'una cançó:




*Timline del procés de descàrrega*

**COMANDA CHECK DOWNLOADS**

Per aquesta comanda senzillament tenim una llista de structs de cançons que s'estan descarregant. Aquest struct té la següent forma:

typedef struct {

Song song; -> Variable que guarda la cançó que s'està descarregant

long downloaded_bytes; -> bytes que ja s'han rebut

long song_size; -> bytes totals de la cançó

pthread_t thread_id; -> id del thread on s'està descarregant la cançó

} Song_Downloading;

Per tant, un cop tenim l'struct només ens fa falta una funció que ens serveixi per afegir les cançons a la llista i anar actualitzant la variable "downloaded_bytes" cada cop que rebem bytes de Poole. Per mostrar la barra de càrrega simplement calculem el percentatge actual de la cançó i el mostrem amb un array de 100 caselles.

```c
void printSongDownloading(Song_Downloading songDownloading) {

    float percentage = (float)songDownloading.downloaded_bytes / (float)songDownloading.song_size * 100;
    char bar[101];
    memset(bar, 0, sizeof(bar));
    int i;
    for (i = 0; i < percentage; i++) {
        bar[i] = '=';
    }
    bar[i++] = '%';
    for (; i < 100; i++) {
        bar[i] = ' ';
    }
    bar[100] = '\0';

    printF(GREEN, "%s [%s] %.2f%%\n", songDownloading.song.fileName, bar, percentage);

}
```

*Funció que s'encarrega de mostrar per pantalla el progrés de la descàrrega*

**COMANDA CLEAR DOWNLOADS**

Per la realització d'aquesta comanda l'únic que vàrem haver d'implementar va ser una funció que iterés sobre la llista de cançons que s'estaven descarregant, les que tinguin numero de bytes descarregats igual al numero de bytes totals, s'eliminen i es "shifta" la llista.

## Fase 4

Per a la implementació d'aquesta era necessaria la creació d'un nou procés sense utilitzar sockets ni cues de missatges, així que vàrem optar per utilitzar un fork del procés principal a Poole. Aquest fork es fa just a l'iniciar Poole. A part de realitzar el fork, era necessària una manera de comunicar els dos processos, per això utilitzem una "pipe" com a variable global per tal de poder enviar les cançons al procés de Monòlit. Quan enviem una cançó de Poole cap a Bowman escrivim el nom de la cançó a la pipe per tal de que el Monòlit gestioni la cançó.

Finalment quan rebem una nova cançó al procés de Monòlit, actualitzem les dades del fitxer stats.txt. Això ho fem amb la funció "updateStatistics" que s'encarrega d'obrir el fitxer, buscar si la cançó ja ha estat guardada anteriorment, si ho ha estat, sumem 1 al numero al cantó del nom, sinó simplement afegim el titol de la cançó seguit d'un "1".

```
void startMonolitServer(void) {
    int fd[2];
    if (pipe(fd) == -1) {
        printF(RED, "ERROR: Error creating pipe\n");
        exit(1);
    }

    pid_t pid = fork();
    if (pid == -1) {
        printF(RED, "ERROR: Error creating fork\n");
        close(fd[0]);
        close(fd[1]);
        exit(1);
    } else if (pid == 0) {
        printF(GREEN, "Starting Monolit Server...\n");
        close(fd[1]); // Cerrar el extremo de escritura del pipe
        throwMonolitServer(fd[0]); // Llamar a la función del Monòlit
        exit(0);
    } else {
        // Proceso padre (Poole)
        close(fd[0]); // Cerrar el extremo de lectura del pipe
        global_write_pipe = fd[1]; // Guardar fd[1] en una variable global para pasarlo a las funciones que lo necesiten
    }
}
```

*Funció que inicialitza el procés Monòlit mitjançant un fork*

## Problemes observats i solucions

1. Problema Valgrind al fer Ctrl + C

   Vam experimentar errors al pressionar Ctrl + C quan s'estaven descarregaven cançons. Tot i implementar un maneig adequat de les senyals i assegurar-nos de alliberar tota la memòria abans de que acabes la execució del programa, l'error persistia. No hem estat capaços de solucionar-ho

2. Directori de cançons hardcoded

   Inicialment el directori on es buscaven les cançons estava codificat de forma fixa en el programa la qual cosa limitava la flexibilitat d'ús. Vam modificar el programa per permetre que l'usuari especifiqui el directori de cerca. Si l'usuari no proporciona un directori vàlid, el programa informa a l'usuari de l'error, deixant en les seves mans la correcció del problema.

3. Problemes en escriure a 'stats.txt':

   Malgrat utilitzar mutex per controlar l'accés concurrent, vam trobar que en aproximadament 1 de cada 3 intents, les entrades de diverses cançons es fusionaven o s'escribien incorrectament a l'arxiu 'stats.txt'. Vam realitzar una revisió detallada de la implementació dels mutex i vam corregir els problemes de sincronització. Tot i així l'error persisteix.

4. Modulació del projecte

   El programa no estava adequadament modularitzat a causa de restriccions de temps, la qual cosa afectava la claredat del codi. Vam reestructurar el codi, delegant responsabilitats específiques a diferents Tipus Abstractes de Dades (TADs). Això va millorar l'organització del codi i va facilitar la seva comprensió i manteniment.

5. Descàrrega de Cançons Binàries:

   En descarregar cançons en format binari, ens vam trobar amb algunes falles inicials. Després d'investigar, vam descobrir que el problema es devia a l'addició d'un caràcter '\0' al final de cada trama. Vam implementar una depuració utilitzant arxius de text per rastrejar el problema. Un cop identificat, vam eliminar l'addició del '\0' al final de les trames, la qual cosa va resoldre el problema amb els arxius de música MP3.

6. Problemes amb Ctrl + c i threads amb trames:
   Vam trobar dificultats en finalitzar els threads correctament, especialment en els bucles d'enviament de trames. Vam introduir una senyal global que, en activar-se (posada a 1) mitjançant Ctrl+C, provoca que tots els bucles finalitzin la seva execució, independentment del seu estat actual. Això va permetre un tancament més controlat i segur dels threads.

## Estimació temporal

En aquest apartat, detallem la distribució del temps dedicat a les diferents etapes del nostre projecte, ajustant el total a 100 hores. La distribució s'ha realitzat de la següent manera:

### Investigació: 12 hores

Aquesta fase inicial va incloure l'estudi dels requisits i l'exploració de les funcions que utilitzaríem. Tot i ser breu, va ser una etapa clau per a la planificació efectiva del projecte al llarg de les diferents fases.

Tota la informació que hem obtingut per realitzar la pràctica ha sigut del llibre de UNIX, el qual creiem que es el més complet possible per tal de realitzar pràctiques com aquesta o encara més avançades.

### Disseny: 18 hores

En aquesta etapa, vam concentrar-nos en el disseny de l'arquitectura del sistema. La col·laboració dels becaris va ser crucial per a optimitzar aquesta fase, proporcionant-nos consells que ens van ajudar a evitar complicacions futures i agilitzar el temps total.

### Implementació: 48 hores

La major part del temps es va dedicar a la implementació, on vam desenvolupar i codificar les funcionalitats clau del sistema. Aquesta va ser la fase més intensiva i va requerir de entendre i comprovar que el que realment estàvem implementant funcionava correctament.

### Testing: 15 hores

Aquesta fase va incloure proves exhaustives per assegurar la funcionalitat, la fiabilitat i la seguretat del sistema.  Tot i que a l'estudy es proporcionaven una seria de tests, hem hagut de fer tests més en profunditat i més precisos per tal d'assegurar que el sistema funciona correctament en tots els casos possibles.

### Documentació: 15 hores

Finalment, vam dedicar temps a la redacció de la documentació del projecte, incluint la preparació d'aquest informe i materials de suport com el README.md.

# Conclusió i propostes de millora

## Reflexions sobre el Projecte

El projecte HAL 9000 System ha estat una oportunitat per aplicar i aprofundir en els coneixements avançats de sistemes operatius posant en pràctica els coneixements teòrics assolits a classe.

## Assoliments Clau

- **Arquitectura Modular i Escalable:** Hem demostrat una comprensió avançada en el disseny de sistemes operatius a través de la creació d'un sistema altament modular i escalable. Aquest disseny facilita l'expansió futura del sistema, permetent la integració de nous mòduls i funcionalitats de manera eficient.

- **Comunicació Interprocessos:** La implementació de mecanismes de comunicació com els sockets, cues de missatges i la gestió de trames destaca com a exemple de com els sistemes operatius moderns faciliten la interacció entre processos distribuïts.

- **Gestió de Concurrència i Memòria:** El maneig efectiu de múltiples processos concurrents com la descàrrega de cançons/playlists i la gestió/alliberació òptima de la memòria reflecteixen un enteniment profund de dos dels aspectes més desafiants en el disseny de sistemes operatius.

## Propostes de Millora

- **Implementació de Seguretat Millorada**: La seguretat és un àrea crítica en qualsevol sistema operatiu modern. Suggerim la integració de capes de seguretat addicionals, per protegir contra vulnerabilitats futures. Per exemple, mitjançant més cifrats de dades a part del MD5SUM, ja que algú amb un sniffer, podria saber, en aquest cas, les cançons que cada usuari descarrega. Però si en un futur en comptes de cançons són comptes bancàries, caldria protegir més les dades.

- **Optimització del Balancejador de Càrrega:** Encara que el balancejador de càrrega “Discovery” ja és eficaç, sempre hi ha marge per a la millora. La implementació d'algoritmes d'aprenentatge automàtic per predir la càrrega de treball i distribuir la càrrega de manera més eficient podria ser un avanç significatiu. Simplement és una suggeriment, ja que amb llenguatge C podria ser bastant complex dissenyar-ho, seria relativament més simple amb Python gràcies a que implementa llibreries preparades per l’aprenentatge automàtic com PyTorch.

- **Ampliació de la Temàtica Cinematogràfica:** Inspirats per la tradició de les pràctiques anteriors, suggerim que la pròxima pràctica sigui anomenada "*Nostromo System*", en honor a la nau espacial de la pel·lícula "Alien". Aquest nou nom podria simbolitzar l'exploració de desafiaments desconeguts dins del món dels sistemes operatius, així com la superació d'obstacles imprevistos.

- **Usuaris (Ripley):** Inspirats en Ellen Ripley, el protagonista.
- **Servidors (Xenomorph):** Anomenats per l'alienígena versàtil i imprevisible, reflecteixen la gestió eficient de múltiples processos.
- **Balancejador de Càrrega (Mother):** Basat en el sistema informàtic del Nostromo, representa control i distribució intel·ligent de càrrega.

## (Opcional) D'on Venen els Noms dels Processos

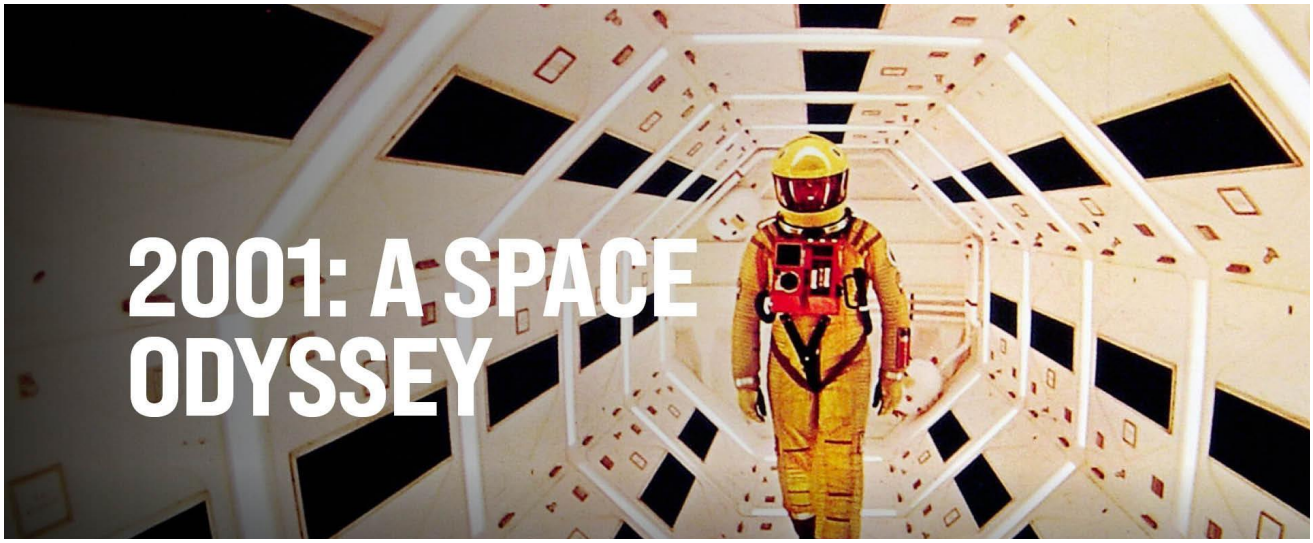
Els noms dels processos utilitzats en el nostre projecte HAL-9000 System, com Monolit, Poole, Bowman, Discovery, i el mateix HAL-9000, tenen un origen inspirat en la icònica pel·lícula de ciència-ficció "2001: A Space Odyssey", dirigida per Stanley Kubrick i basada en la novel·la homònima d'Arthur C. Clarke.

**HAL-9000:** És el nom del superordinador a bord de la nau espacial Discovery One en la pel·lícula. HAL (*Heuristically programmed ALgorithmic computer*), es presenta com un exemple avançat d'intel·ligència artificial, capaç de prendre decisions complexes i interactuar amb la tripulació. En el nostre projecte, aquest nom simbolitza la complexitat i la capacitat del sistema central.

**Bowman i Poole:** Aquests noms provenen dels protagonistes humans de la pel·lícula, els astronautes David Bowman i Frank Poole. Aquests personatges representen la interacció humana amb la tecnologia avançada, un tema que ressona profundament en el nostre projecte, on "Bowman" és utilitzat per a representar els usuaris (astronautes) i "Poole" per als servidors (la tripulació).

**Discovery:** És el nom de la nau espacial on transcorre gran part de l'acció en "2001: A Space Odyssey". En el nostre sistema, "Discovery" actua com a balancejador de càrrega, una metàfora de com la nau dirigia la missió i coordinava les accions de la tripulació i HAL-9000.

**Monolit:** En "2001: A Space Odyssey", els Monòlits són entitats enigmàtiques que impulsen l'evolució de la intel·ligència. En el nostre projecte, "Monolit" fa referència a un subprocés dins del servidor Poole, especialitzat en l'anàlisi i el registre de les interaccions dels usuaris amb el sistema. Aquest subprocés s'encarrega de monitoritzar i emmagatzemar informació crucial, com el nom de les cançons i el nombre de vegades que aquestes han estat descarregades, guardant aquestes dades en el fitxer stats.txt. Aquesta funcionalitat reflecteix el paper dels Monòlits en la pel·lícula com a catalitzadors de coneixement i descobriment, simbolitzant així el paper crític de Monolit en la recopilació i anàlisi de dades dins del nostre sistema, i contribuint a la comprensió més profunda del comportament dels usuaris.



# Bibliografia

Salvador Pont, J. (2014). Programació en UNIX per a pràctiques de Sistemes Operatius. Enginyeria i Arquitectura La Salle.

Wikipedia contributors. (2024, 6 enero). *2001: A Space Odyssey*. Wikipedia. https://en.wikipedia.org/wiki/2001:_A_Space_Odyssey