Mr. J. System

Fase 1: Pràctica de Sistemes Operatius

Grau d'Enginyeria Electrònica de Telecomunicacions

Universitat Ramon Llull

Pau Olea Reyes
pau.olea@students.salle.url.edu

Alfred Chávez Fernández
alfred.chavez@students.salle.url.edu

20 d'octubre de 2024

Professors: Jordi Malé

Curs: 2024-2025

Índex

1 Introducció

Aquest informe detalla la primera fase de la pràctica de Sistemes Operatius, incloent el diagrama de blocs, les estructures de dades utilitzades i la justificació de les decisions de disseny preses.

2 Diagrama de Blocs

A continuació es presenta el diagrama de blocs que explica la Fase 1 de la pràctica. Aquest diagrama mostra les relacions entre els diferents fitxers que conformen el sistema i com aquests s'utilitzen en el procés de desenvolupament.

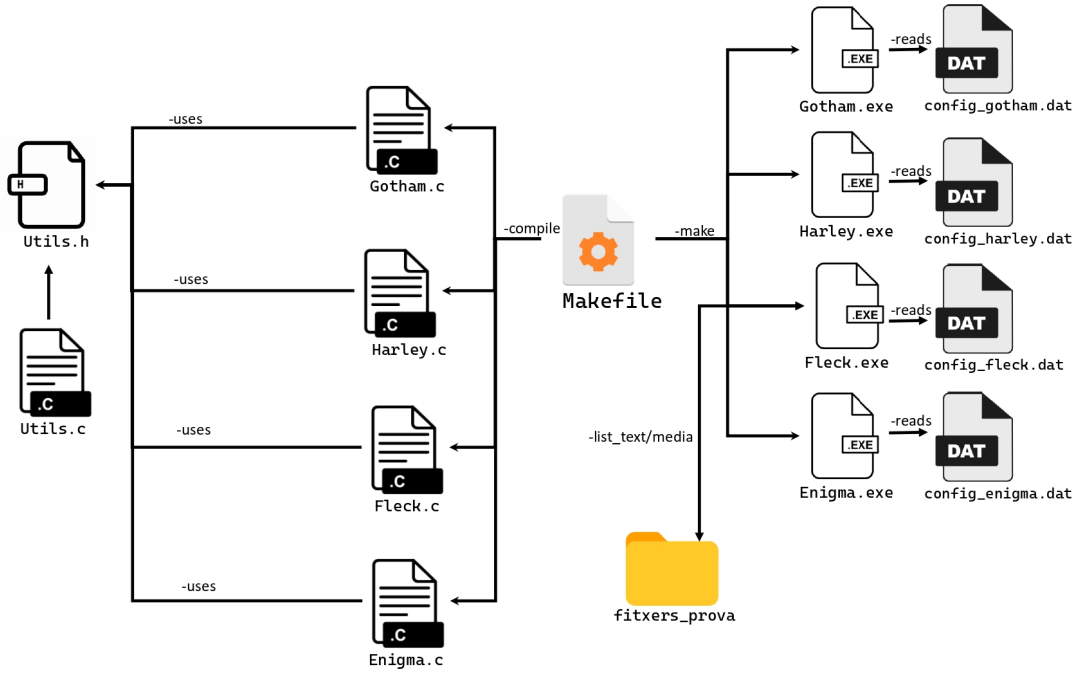

Figura 1: Diagrama de blocs de la Fase 1

El diagrama de blocs il·lustra com es creen i s'interconnecten els processos del sistema per a la Fase 1 del projecte "Mr. J. System". Aquesta fase se centra en la configuració i la preparació dels quatre processos principals: `Fleck`, `Harley`, `Enigma`, i `Gotham`, a partir de la informació dels fitxers de configuració. També s'han afegit les noves funcionalitats de LIST TEXT i LIST MEDIA que permeten gestionar fitxers de text i multimèdia.

- **Fitxers de codi font i capçaleres**: Els fitxers C (`.c`) són els fitxers de codi font que implementen la funcionalitat dels diferents components del sistema. El fitxer `Utils.c`, juntament amb el seu fitxer de capçalera `Utils.h`, conté funcions auxiliars comunes que són utilitzades per altres fitxers, com `Gotham.c`, `Harley.c`, `Fleck.c`, i `Enigma.c`. Aquesta organització modular facilita la reutilització de codi i permet la separació clara de les funcionalitats de cada component.
- **Processos de compilació i generació d'executables**: El fitxer `Makefile` s'encarrega de la compilació automàtica del projecte. Coordina la creació dels executables (`.exe`) a

partir dels fitxers de codi font i capçalera. Mitjançant l'ordre `make`, es generen els executables `Gotham.exe`, `Harley.exe`, `Fleck.exe`, i `Enigma.exe`. Aquesta automatització simplifica la construcció del projecte i garanteix que totes les dependències es gestionin correctament.

- **Configuració inicial dels processos**: Cada procés requereix un fitxer de configuració (`config_gotham.dat`, `config_harley.dat`, `config_fleck.dat`, `config_enigma.dat`) per definir els paràmetres necessaris per a la seva execució, com l'IP, el port i altres configuracions específiques del procés. Els fitxers de configuració són processats a l'inici per a cada procés per establir els paràmetres del sistema. Per exemple, `Fleck` utilitza el fitxer de configuració per obtenir el directori de treball i la informació de connexió amb `Gotham`.
- **Noves funcionalitats LIST TEXT i LIST MEDIA**: Les noves funcionalitats LIST TEXT i LIST MEDIA permeten llistar i gestionar fitxers de text i multimèdia, respectivament. Aquests fitxers es troben al directori `fitxers_prova`, on es troben els fitxers de mostra per executar les proves. La funció LIST TEXT cerca i llista tots els fitxers de text disponibles al directori, mentre que LIST MEDIA fa el mateix amb els fitxers multimèdia. Aquesta funcionalitat facilita la gestió automatitzada dels recursos del sistema.
- **Organització i interdependències**: La relació entre els diferents components està clarament definida. Els fitxers `.c` utilitzen `Utils.h` per accedir a funcions compartides, i el `Makefile` coordina la compilació de tot el projecte per generar els executables que, posteriorment, s'executaran amb les configuracions definides en els fitxers `.dat`. Les noves funcionalitats l·list_textï l·list_mediaïnteractuen amb els processos principals per llistar els fitxers necessaris per a l'execució del sistema.

Aquest diagrama mostra clarament com s'integren els diferents components per crear un sistema on els processos poden interactuar correctament segons la configuració proporcionada. Permet modificar paràmetres sense necessitat de recompilar, ja que els fitxers de configuració es poden actualitzar de manera independent.

3 Estructures de Dades Utilitzades

A continuació es presenten les principals estructures de dades utilitzades en la primera fase de la pràctica, juntament amb una explicació de la seva funció i justificació:

3.1 Estructura `FleckConfig`

Aquesta estructura s'utilitza per emmagatzemar la configuració necessària per la comunicació amb Gotham. Conté informació com l'usuari, directori i l'adreça IP i port de Gotham.

```
1        typedef struct {
2            char *user;
3            char *directory;
4            char *ipGotham;
5            int portGotham;
6        } FleckConfig;
```

Listing 1: Definició de `FleckConfig`

Justificació: L'ús de memòria dinàmica per a l'usuari, directori i IP permet gestionar diferents longituds d'entrada de forma flexible, millorant la gestió de memòria en relació a altres mètodes més rígids, com l'ús de `arrays` fixos.

- `user`: una cadena de caràcters que representa el nom de l'usuari que executa el sistema o està associat amb la configuració. Aquesta informació pot ser útil per a propòsits d'autenticació o identificació.
- `directory`: una cadena de caràcters que indica el directori en el qual es troben els fitxers de configuració o altres recursos necessaris per al sistema. El directori s'utilitza per accedir a aquests fitxers de manera dinàmica durant l'execució del programa.
- `ipGotham`: una cadena de caràcters que conté l'adreça IP del servidor Gotham. Aquesta IP s'utilitza per establir la connexió amb el servidor remot, la qual cosa és fonamental per permetre la comunicació en xarxa.
- `portGotham`: un enter que especifica el port del servidor Gotham. El port és un paràmetre crític per a la comunicació en xarxa, ja que determina el punt d'accés del servidor amb el qual es comunicarà el client.

3.2 Estructura `HarleyConfig`

Aquesta estructura conté les dades de configuració per a la comunicació amb Harley, de forma semblant a `FleckConfig`, però adaptada per Harley.

```
1         typedef struct {
2             char *user;
3             char *directory;
4             char *ipHarley;
5             int portHarley;
6         } HarleyConfig;
```

Listing 2: Definició de `HarleyConfig`

Justificació: L'ús de punters a `char` ens permet obtenir configuracions variables des de fitxers de configuració externs `.dat` i adaptar les estructures segons sigui necessari durant l'execució del programa.

- `ipGotham`: una cadena de caràcters que representa l'adreça IP del servidor Gotham. És utilitzada per connectar-se al servidor quan es necessita interacció amb ell.
- `portGotham`: un enter que especifica el port associat al servidor Gotham. Aquesta informació és necessària per configurar la comunicació de xarxa amb el servidor adequat.
- `ipFleck`: una cadena de caràcters que representa l'adreça IP del servidor Fleck. Es fa servir per establir connexions amb aquest servidor.
- `portFleck`: un enter que indica el port del servidor Fleck. El port permet identificar el servei amb el qual es comunicarà el sistema.
- `directory`: una cadena de caràcters que especifica el directori de treball on es poden trobar fitxers i altres recursos necessaris per a l'operació del sistema.
- `workerType`: una cadena de caràcters que representa el tipus de treballador o mòdul que utilitzarà el sistema. Això pot influir en el comportament o la configuració específica del sistema segons el tipus de tasca que es realitzi.

3.3 Estructura `GothamConfig`

L'estructura `GothamConfig` conté informació relacionada amb Gotham, incloent l'usuari, directori, l'adreça IP i port.

```
typedef struct {
    char *user;
    char *directory;
    char *ipGotham;
    int portGotham;
} GothamConfig;
```

Listing 3: Definició de `GothamConfig`

Justificació: Similar a `FleckConfig` i `HarleyConfig`, aquesta estructura emmagatzema els detalls de configuració del mòdul Gotham. S'utilitzen punters per permetre una gestió flexible de les dades.

- `ipFleck`: una cadena de caràcters que conté l'adreça IP del servidor Fleck. S'utilitza per connectar el sistema al servidor corresponent per a la comunicació de dades.
- `portFleck`: un enter que indica el port del servidor Fleck. El port és necessari per identificar el servei amb el qual es connectarà.
- `ipHarEni`: una cadena de caràcters que representa l'adreça IP del servidor Harley Enigma. Aquesta IP s'utilitza per connectar-se amb Harley Enigma per a operacions de xarxa.
- `portHarEni`: un enter que especifica el port del servidor Harley Enigma, necessari per accedir al servei correcte.

3.4 Estructura `EnigmaConfig`

L'estructura `EnigmaConfig` s'utilitza per emmagatzemar informació relativa a Enigma, incloent els mateixos camps que en les altres configuracions.

```
1         typedef struct {
2             char *user;
3             char *directory;
4             char *ipEnigma;
5             int portEnigma;
6         } EnigmaConfig;
```

Listing 4: Definició de `EnigmaConfig`

Justificació: L'estructura `EnigmaConfig` segueix el mateix patró que les altres configuracions, per tal de mantenir la coherència i facilitar la gestió del codi i les configuracions externes.

- `ipGotham`: una cadena de caràcters que conté l'adreça IP del servidor Gotham, per establir connexions.
- `portGotham`: un enter que representa el port associat amb el servidor Gotham.
- `ipFleck`: una cadena de caràcters que conté l'adreça IP del servidor Fleck, per a comunicacions.
- `portFleck`: un enter que representa el port associat amb el servidor Fleck.
- `directory`: una cadena de caràcters que especifica el directori on es troba la configuració.
- `workerType`: una cadena de caràcters que indica el tipus de treballador, per determinar el comportament de l'aplicació.

4 Funcions `readConfigFile`

Les funcions `readConfigFile` tenen un paper clau a l'hora de carregar les configuracions des dels fitxers `.dat`. A continuació, es presenta la versió de la funció per a cadascun dels quatre fitxers:

4.1 `readConfigFile` per a Fleck

```
1  // Funci per llegir el fitxer de configuraci i carregar la informaci a la estructura FleckConfig
2  /**********************************************
3  * @Finalitat: Llegeix el fitxer de configuraci especificat i emmagatzema la informaci a la estructura FleckConfig.
4  * @Par metres:
5  *   in: configFile = nom del fitxer de configuraci .
6  *   out: fleckConfig = estructura on s'emmagatzema la configuraci llegida.
7  * @Retorn: ----
8  **********************************************/
9  void readConfigFile(const char *configFile, FleckConfig *fleckConfig) {
10     int fd = open(configFile, O_RDONLY); // Obre el fitxer en mode nom s lectura
11     if (fd == -1) {
12         printF("Error obrint el fitxer de configuraci \n"); // Missatge d'error si no es pot obrir
13         exit(1); // Finalitza el programa en cas d'error
14     }
15
16     // Llegeix i assigna mem ria per a cada camp de la configuraci
17     fleckConfig->user = readUntil(fd, '\n'); // Llegeix el nom de l'usuari
18     removeChar(fleckConfig->user, '&'); // Elimina el car cter '&' del nom de l'usuari
19     fleckConfig->directory = trim(readUntil(fd, '\n')); // Elimina espais al voltant del directori
20     fleckConfig->ipGotham = readUntil(fd, '\n'); // Llegeix la IP del servidor Gotham
21
22     char *portStr = readUntil(fd, '\n'); // Llegeix el port com a cadena
23     fleckConfig->portGotham = atoi(portStr); // Converteix el port a enter
24     free(portStr); // Allibera la mem ria de la cadena temporal
25     close(fd); // Tanca el fitxer
26
27     fleckConfig->user = trim(fleckConfig->user); // Crear un buffer temporal per emmagatzemar el missatge
28     printF("\n");
29     printF(fleckConfig->user);
30     printF(" user initialized\n\n");
31     printF("File read correctly:");
32     printF("\nUser - ");
33     printF(fleckConfig->user);
34     printF("\nDirectory - ");
35     printF(fleckConfig->directory);
```

```
36              printF("\nIP - ");
37              printF(fleckConfig->ipGotham);
38
39              printF("\nPort - ");
40              char* portGothamStr = NULL;
41              asprintf(&portGothamStr, "%d\n", fleckConfig->portGotham); //
                    Converteix el port a cadena
42              printF(portGothamStr);
43              free(portGothamStr); // Allibera la mem ria de la cadena
                    temporal
44          }
```

Listing 5: Funció `readConfigFile` per a Fleck

La funció readConfigFile llegeix un fitxer de configuració per carregar la informació necessària dins de l'estructura FleckConfig. Comença obrint el fitxer especificat en mode només lectura utilitzant la funció open, i si no pot obrir el fitxer, mostra un missatge d'error i finalitza el programa. A continuació, la funció llegeix cada línia del fitxer fins al salt de línia mitjançant readUntil, que llegeix caràcter per caràcter fins trobar el delimitador especificat. La primera línia correspon al nom de l'usuari i s'emmagatzema al camp user de l'estructura, eliminant qualsevol caràcter '' que pugui estar present. La segona línia correspon al directori, s'aplica la funció trim per eliminar espais en blanc, i s'assigna al camp directory. La tercera línia conté la IP del servidor Gotham i s'emmagatzema al camp ipGotham. La quarta línia llegeix el port del servidor en format de text, que es converteix a un enter amb atoi i s'assigna al camp portGotham, alliberant la memòria utilitzada per la cadena de text temporal. Després de llegir i emmagatzemar les dades, es tanca el fitxer amb la funció close. La funció mostra la informació llegida per verificar que s'ha carregat correctament, incloent el nom d'usuari, el directori, la IP i el port de Gotham, assegurant-se també que el nom d'usuari no tingui espais en blanc no desitjats amb una nova aplicació de trim. Aquesta funció és clau per carregar la configuració inicial del sistema Fleck, ja que utilitza memòria dinàmica per gestionar cadenes de longitud variable i garanteix una correcta neteja de memòria per evitar fuites.

4.2 `readConfigFile` per a Harley

Funció semblant a la de `Fleck`, però assignada a `HarleyConfig`:

```
1         // Funci  per llegir el fitxer de configuraci  de Harley
2         /*********************************************
3         * @Finalitat: Llegeix el fitxer de configuraci  especificat i
              emmagatzema la informaci  a la estructura HarleyConfig.
4         * @Par metres:
5         *   in: configFile = nom del fitxer de configuraci .
6         *   out: harleyConfig = estructura on s'emmagatzema la configuraci
              llegida.
7         * @Retorn: ----
8         *********************************************/
9         void readConfigFile(const char *configFile, HarleyConfig *
              harleyConfig) {
10            int fd = open(configFile, O_RDONLY); // Obre el fitxer en mode
                  nom s lectura
11
12            if (fd == -1) {
13                printF("Error obrint el fitxer de configuraci \n"); //
                      Missatge d'error si no es pot obrir
14                exit(1); // Finalitza el programa en cas d'error
15            }
16
17            // Llegeix i assigna mem ria per a cada camp de la
                  configuraci
18            harleyConfig->ipGotham = readUntil(fd, '\n'); // Llegeix la IP
                  del servidor Gotham
19            char *portGotham = readUntil(fd, '\n'); // Llegeix el port com a
                   cadena de text
20            harleyConfig->portGotham = atoi(portGotham); // Converteix el
                  port a enter
21            free(portGotham); // Allibera la mem ria de la cadena temporal
22
23            harleyConfig->ipFleck = readUntil(fd, '\n'); // Llegeix la IP
                  del servidor Fleck
24            char* portFleck = readUntil(fd, '\n'); // Llegeix el port com a
                  cadena de text
25            harleyConfig->portFleck = atoi(portFleck); // Converteix el port
                   a enter
26            free(portFleck); // Allibera la mem ria de la cadena temporal
27
28            harleyConfig->directory = readUntil(fd, '\n'); // Llegeix el
                  directori de treball
29            harleyConfig->workerType = readUntil(fd, '\n'); // Llegeix el
                  tipus de treballador
30
31            close(fd); // Tanca el fitxer
32
33            // Mostra la configuraci  llegida per a verificaci
34            printF("Fitxer llegit correctament:\n");
35            printF("Gotham IP - ");
36            printF(harleyConfig->ipGotham);
37            printF("\nGotham Port - ");
38            char* portGothamStr = NULL;
39            asprintf(&portGothamStr, "%d", harleyConfig->portGotham); //
                  Converteix el port a cadena de text
40            printF(portGothamStr);
```

```
41              free(portGothamStr); // Allibera la mem ria de la cadena
                    temporal
42
43              printF("\nIP Fleck - ");
44              printF(harleyConfig->ipFleck);
45              printF("\nPort Fleck - ");
46              char* portFleckStr = NULL;
47              asprintf(&portFleckStr, "%d", harleyConfig->portFleck); //
                    Converteix el port a cadena de text
48              printF(portFleckStr);
49              free(portFleckStr); // Allibera la mem ria de la cadena
                    temporal
50
51              printF("\nDirectory - ");
52              printF(harleyConfig->directory);
53              printF("\nWorker Type - ");
54              printF(harleyConfig->workerType);
55              printF("\n");
56          }
```

Listing 6: Funció `readConfigFile` per a Harley

La funció readConfigFile per a Harley llegeix un fitxer de configuració i emmagatzema la informació llegida a l'estructura HarleyConfig. Primer, obre el fitxer especificat en mode només lectura utilitzant la funció open. Si no és possible obrir el fitxer, mostra un missatge d'error i finalitza l'execució del programa. La funció continua llegint les línies del fitxer una a una, utilitzant la funció readUntil, que llegeix fins al salt de línia. La primera línia conté la IP del servidor Gotham, que s'emmagatzema al camp ipGotham de l'estructura. La següent línia és el port del servidor Gotham, que es llegeix com una cadena de text, es converteix a un enter amb atoi, i es guarda a portGotham, alliberant després la memòria de la cadena temporal. Les següents dues línies segueixen el mateix procés per a la IP i el port del servidor Fleck, assignant els valors als camps ipFleck i portFleck, respectivament. Després, llegeix el directori de treball i el tipus de treballador, i els guarda als camps directory i workerType. Un cop s'ha llegit tota la informació, el fitxer es tanca amb la funció close.

4.3 `readConfigFile` per a Gotham

```
// Funci per llegir el fitxer de configuraci de Gotham
/*********************************************
* @Finalitat: Llegeix el fitxer de configuraci especificat i
    emmagatzema la informaci a la estructura GothamConfig.
* @Par metres:
*   in: configFile = nom del fitxer de configuraci .
*   out: gothamConfig = estructura on s'emmagatzema la configuraci
    llegida.
* @Retorn: ----
*********************************************/
void readConfigFile(const char *configFile, GothamConfig *
   gothamConfig) {
    int fd = open(configFile, O_RDONLY); // Obre el fitxer en mode
       nom s lectura

    if (fd == -1) {
        printF("Error obrint el fitxer de configuraci \n"); //
           Missatge d'error si no es pot obrir
        exit(1); // Finalitza el programa en cas d'error
    }

    // Llegeix i assigna mem ria per a cada camp de la
       configuraci
    gothamConfig->ipFleck = readUntil(fd, '\n'); // Llegeix la IP
       del servidor Fleck
    char* portFleck = readUntil(fd, '\n'); // Llegeix el port com a
       cadena
    gothamConfig->portFleck = atoi(portFleck); // Converteix el port
        a enter
    free(portFleck); // Allibera la mem ria de la cadena temporal

    gothamConfig->ipHarEni = readUntil(fd, '\n'); // Llegeix la IP
       del servidor Harley Enigma
    char *portHarEni = readUntil(fd, '\n'); // Llegeix el port com a
        cadena
    gothamConfig->portHarEni = atoi(portHarEni); // Converteix el
       port a enter
    free(portHarEni); // Allibera la mem ria de la cadena temporal

    close(fd); // Tanca el fitxer

    // Mostra la configuraci llegida per a verificaci
    printF("IP Fleck - ");
    printF(gothamConfig->ipFleck);
    printF("\nPort Fleck - ");
    char* portFleckStr = NULL;
    asprintf(&portFleckStr, "%d", gothamConfig->portFleck); //
       Converteix el port a cadena de text
    printF(portFleckStr);
    free(portFleckStr); // Allibera la mem ria de la cadena
       temporal

    printF("\nIP Harley Enigma - ");
    printF(gothamConfig->ipHarEni);
    printF("\nPort Harley Enigma - ");
    char* portHarEniStr = NULL;
```

```
43              asprintf(&portHarEniStr, "%d\n", gothamConfig->portHarEni); //
                    Converteix el port a cadena de text
44              printF(portHarEniStr);
45              free(portHarEniStr); // Allibera la mem ria de la cadena
                    temporal
46          }
```

Listing 7: Funció `readConfigFile` per a Gotham

La funció readConfigFile per a Gotham té com a objectiu llegir el fitxer de configuració especificat i emmagatzemar la informació en una estructura GothamConfig. Comença obrint el fitxer en mode de lectura utilitzant la funció open, i si el fitxer no es pot obrir, es mostra un missatge d'error i el programa es finalitza amb exit per indicar un error crític.

La funció llegeix la configuració camp per camp. Primer, llegeix la IP del servidor Fleck amb la funció readUntil, i després llegeix el port associat com una cadena de text. Aquesta cadena es converteix a un enter mitjançant la funció atoi i s'assigna al camp portFleck de l'estructura. A continuació, allibera la memòria assignada per la cadena temporal que contenia el port.

Després, la funció continua llegint la IP i el port del servidor Harley Enigma, seguint el mateix procés. Llegeix la IP, després el port com una cadena de text, la converteix a un enter, l'assigna al camp portHarEni, i allibera la memòria de la cadena temporal.

Un cop s'ha llegit tota la informació necessària, la funció tanca el fitxer amb close. Finalment, es mostren per pantalla els valors llegits per verificar que s'han carregat correctament. Això inclou la impressió de les IPs i els ports de Fleck i Harley Enigma. Els ports es converteixen a cadena de text utilitzant asprintf per assegurar-se que es mostren correctament, i la memòria temporal utilitzada per aquestes conversions s'allibera després de cada ús.

4.4 `readConfigFile` per a Enigma

```
// Funci per llegir el fitxer de configuraci d'Enigma
/*********************************************
* @Finalitat: Llegeix el fitxer de configuraci especificat i
    carrega la informaci en una estructura EnigmaConfig.
* @Par metres:
*   in: configFile = nom del fitxer de configuraci .
*   out: enigmaConfig = estructura on s'emmagatzema la configuraci
    llegida.
* @Retorn: ----
*********************************************/
void readConfigFile(const char *configFile, EnigmaConfig *
   enigmaConfig) {
    int fd = open(configFile, O_RDONLY); // Obre el fitxer en mode
       nom s lectura

    if (fd == -1) {
        printF("Error obrint el fitxer de configuraci \n"); //
           Missatge d'error si no es pot obrir
        exit(1); // Finalitza el programa en cas d'error
    }

    // Llegeix i assigna la mem ria per a cada camp de la
       configuraci
    enigmaConfig->ipGotham = readUntil(fd, '\n'); // Llegeix la IP
       del servidor Gotham
    char* portGotham = readUntil(fd, '\n'); // Llegeix el port com a
        cadena de text
    enigmaConfig->portGotham = atoi(portGotham); // Converteix el
       port a enter
    free(portGotham); // Allibera la mem ria de la cadena temporal

    enigmaConfig->ipFleck = readUntil(fd, '\n'); // Llegeix la IP
       del servidor Fleck
    char *portFleck = readUntil(fd, '\n'); // Llegeix el port com a
       cadena de text
    enigmaConfig->portFleck = atoi(portFleck); // Converteix el port
        a enter
    free(portFleck); // Allibera la mem ria de la cadena temporal

    enigmaConfig->directory = readUntil(fd, '\n'); // Llegeix el
       directori de configuraci
    enigmaConfig->workerType = readUntil(fd, '\n'); // Llegeix el
       tipus de treballador

    close(fd); // Tanca el fitxer

    // Mostra la configuraci llegida per a verificaci
    printF("Ip Gotham - ");
    printF(enigmaConfig->ipGotham);
    printF("\nPort Gotham - ");
    char* portGothamStr = NULL;
    asprintf(&portGothamStr, "%d", enigmaConfig->portGotham); //
       Converteix el port a cadena de text
    printF(portGothamStr);
    free(portGothamStr); // Allibera la mem ria de la cadena
       temporal
```

```
41
42              printF("\nIp Fleck - ");
43              printF(enigmaConfig->ipFleck);
44              printF("\nPort Fleck - ");
45              char* portFleckStr = NULL;
46              asprintf(&portFleckStr, "%d", enigmaConfig->portFleck); //
                    Converteix el port a cadena de text
47              printF(portFleckStr);
48              free(portFleckStr); // Allibera la mem ria de la cadena
                    temporal
49
50              printF("\nDirectory - ");
51              printF(enigmaConfig->directory);
52
53              printF("\nWorker Type - ");
54              printF(enigmaConfig->workerType);
55              printF("\n");
56          }
```

Listing 8: Funció `readConfigFile` per a Enigma

La funció readConfigFile per a Enigma s'encarrega de llegir un fitxer de configuració i emmagatzemar la informació obtinguda en una estructura EnigmaConfig. Comença obrint el fitxer especificat en mode de lectura amb la funció open. Si el fitxer no es pot obrir, la funció mostra un missatge d'error i finalitza el programa mitjançant la funció exit.

Després, la funció llegeix cada línia del fitxer per obtenir els diferents paràmetres de configuració. Utilitza la funció readUntil per llegir fins al caràcter de salt de línia. La primera línia llegeix la IP del servidor Gotham i l'emmagatzema al camp ipGotham de l'estructura. La següent línia llegeix el port del servidor Gotham, que es converteix de cadena de text a enter amb atoi i s'assigna al camp portGotham, alliberant la memòria de la cadena temporal. A continuació, llegeix la IP i el port del servidor Fleck de manera similar, guardant aquests valors als camps ipFleck i portFleck, respectivament, i també alliberant la memòria de les cadenes temporals.

Un cop llegides aquestes dades, la funció continua amb la lectura del directori de configuració i el tipus de treballador, emmagatzemant els valors obtinguts als camps directory i workerType de l'estructura EnigmaConfig. Un cop tota la informació ha estat llegida i assignada, el fitxer de configuració es tanca amb la funció close.

Finalment, la funció mostra per pantalla els valors llegits per verificar que s'han carregat correctament. Per a això, imprimeix els valors de les IPs i ports de Gotham i Fleck, així com el directori i el tipus de treballador, utilitzant la funció printF. Aquesta verificació inclou convertir els ports a cadena de text utilitzant asprintf per assegurar-se que s'imprimeixen correctament.

5 Conclusió

En la Fase 1 del projecte "Mr. J. System", hem establert una arquitectura modular que permet la gestió i configuració fàcil dels diferents components del sistema. Les estructures de dades han estat dissenyades per ser coherents i flexibles, permetent l'ús de memòria dinàmica per gestionar diferents longituds d'entrada.

El treball en equip ha estat clau per a l'èxit del projecte. Hem utilitzat GitHub per gestionar el codi, amb revisions de codi i *pull requests* per assegurar la qualitat. Les reunions a Discord ens han permès resoldre problemes en temps real i coordinar les tasques de forma eficient.

En aquesta fase, hem assegurat que cada procés pugui evolucionar independentment i ser configurat fàcilment a partir de fitxers `.dat`. La modularitat del codi facilita la futura extensió i manteniment del sistema, assegurant que es poden afegir nous components o modificar els existents sense afectar negativament la resta del projecte.