1. OBJECTIU

S'ha de paral·lelitzar la verificació CRC de diversos fitxers, els camins de dades dels quals es passaran com a arguments de línia de comandes. El codi te curses de dades. Has d'afegir les eines de sincronització pertinents al fitxer "fileManager.c" i "main.c" per evitar-les.

La paral·lelització es fa utilitzant fils, cadascun triarà contínuament un fitxer del qual cap altre fil estigui llegint actualment, llegirà un bloc de dades de 256 bytes del fitxer de dades i el seu CRC, i el verificarà.

En cas d'inconsistència (el CRC llegit i el calculat no quadra), el fil ha de registrar-ho a la sortida estàndard, especificant el fitxer on es va trobar la inconsistència.

Cal tenir en compte que diferents blocs del mateix fitxer poden ser llegits per diferents fils, però no simultàniament. Per ajudar-te amb això, el codi ja utilitza una estructura de FileManager (pots consultar el seu codi o).

Hauràs de completar les seves funcions (markFileAsFinished i getAndReserveFile). A més, els fils hauran de sortir només quan no hi hagi més fitxers per processar.

2. LLIURAMENT

- Imagine that you have a data source which allows N simultaneous reads. What would you change, in order to implement it?

- Why there is no need for synchronisation in the function unreserveFile? Would it be in the case of the previous question?

- Compare the execution time between a version of 1 thread, and 4 threads, using the timer seen in P1. Create a table when processing a number of files ranging from 1 to 10 (you can make copies of large file.txt).

3. HINTS DEL CODI

To compile with pthreads, remember to use the adequate options:

```
gcc main.c myutils.c fileManager.c crc.c -lpthread -o main    # Linux
```

Remember the usage of the pthread functions to create and wait for a thread, mutex locks:

```
pthread_create(pthread_t *tid, NULL, fthread, void* p);
int pthread_join(pthread_t thread, NULL);

pthread_mutex_t lock;
pthread_mutex_lock(pthread_mutex_t* lock);
pthread_mutex_unlock(pthread_mutex_t* lock);

pthread_cond_t cond; //Condition associated to a monitor
pthread_cond_init(pthread_cond_t * cond, NULL);
pthread_cond_wait(pthread_cond_t * cond, pthread_mutex_t * lock);
pthread_cond_signal(pthread_cond_t * cond);
```

And our implementation of semaphores using monitors, which can be found in myutils.c:

```
typedef struct semaphore_struct {
        int i;
        pthread_mutex_t lock;
        pthread_cond_t cond;
} my_semaphore;

void my_sem_init(my_semaphore* sem, int i);
void my_sem_wait(my_semaphore* sem);
void my_sem_signal(my_semaphore* sem);
```

4. EXPLICACIÓ DEL FILE MANAGER

Per ajudar-te a assignar fitxers als fils, tens una estructura de FileManager ("fileManager.c") que conté funcions que busquen fitxers lliures i els marquen com a ocupats. Aquesta estructura de dades conté el descriptor de fitxer dels fitxers d'entrada, el nombre de fitxers que ja han acabat de verificar, així com la informació sobre si actualment estan sent llegits per un fil. Els camps són els següents:

nFilesTotal	El nombre total de fitxers que s'han proporcionat per processar.
nFilesRemaining	El nombre de fitxers que encara no han acabat el processament.
fdData i fdCRC	Aquestes matrius emmagatzemaran els descriptors de fitxer dels fitxers de dades i CRC.
fileFinished	Aquesta matriu serà 1 si el fitxer i-èsim s'ha llegit completament, i 0 altrament.
fileAvailable	Aquesta matriu indicarà si el fitxer i-èsim està sent llegit actualment per un fil, i 0 altrament.

El FileManager té les següents funcionalitats. Quan es consideren les race conditions, recorda que cada fil pot cridar a aquesta estructura en qualsevol moment, i que el seu contingut és compartit per tots els fils.

· void initialiseFdProvider(FileManager *fm, int argc, char **argv): Has d'acabar la seva implementació. Aquesta funció rebrà una estructura de FileManager no inicialitzada i la inicialitzarà (incloent qualsevol eina de sincronització que puguis necessitar), obrirà tots els fitxers de dades i CRC i emmagatzemarà els seus descriptors de fitxer. També haurà d'assignar la memòria necessària per al seu array utilitzant malloc.

· void destroyFdProvider(FileManager *fm): Allibera tota la memòria assignada i tanca els fitxers oberts. S'ha de cridar al final del programa. Ja implementada.

· int getAndReserveFile(FileManager *fm, DataEntry *de): Has d'acabar la seva implementació. Aquesta funció buscarà al FileManager un fitxer que no estigui ni acabat ni sigui llegit actualment per un altre fil i retornarà la seva informació (fds, índex) a la DataEntry (consulta a continuació), de manera que el fil pugui llegir un bloc de dades i el seu CRC. Per evitar que dos fils llegeixin del mateix fitxer, marcarà el fitxer com a no disponible. Si no hi ha cap fitxer disponible, retornarà 1; sinó, retornarà 0. Avís: què pot passar si dos fils criden a aquesta funció al mateix temps?

· void unreserveFile(FileManager *fm, DataEntry *d): Aquesta funció s'ha de cridar quan el fil hagi acabat de llegir el bloc del fitxer especificat a l'estructura DataEntry i actualitzarà el FileManager per marcar el fitxer com a lliest per ser llegit.

· void markFileAsFinished(FileManager *fm, DataEntry *d): Aquesta funció s'ha de cridar quan un fitxer hagi estat completament llegit, per marcar-lo com a acabat a l'estructura FileManager.

Per emmagatzemar el resultat de les trucades, s'ofereix una estructura anomenada DataEntry, que emmagatzemarà els descriptors de fitxer del CRC i dels fitxers de dades retornats, i l'índex del fitxer dins de l'estructura FileManager, de manera que pugui ser identificat més endavant pel FileManager (per marcar el fitxer com a disponible en acabar). Has de completar les funcions getAndReserve i initialiseFdProvider. AVÍS: la implementació donada conté curses de dades. Afegeix semàfors i/o bloquejos per evitar-les.

A la funció principal, s'inicialitzarà el FileManager amb la llista de fitxers a obrir i es crearan exactament 4 fils de treball. Després, esperarà fins que tots els fils hagin acabat. En aquest exercici, se t'ha demanat que facis un seguiment dels fitxers disponibles al FileManager utilitzant semàfors, de manera que quan cridis a getAndReserveFile, sempre obtinguis un fitxer vàlid. Utilitza semàfors per aconseguir-ho sense esperes actives.