FUNDAMENTOS DE SISTEMAS OPERATIVOS
PRÁCTICA 2

UNIVERSITAT ROVIRA i VIRGILI

Alumnos: Adrián Izquierdo Niño y Jordi Grau
Fecha: 21-05-2023
Profesor: Stéphane Salaet
Laboratorio: L3

INDEX

1. Decisiones de diseño

Esta práctica trataba de aprender cómo crear, sincronizar, y utilizar threads y procesos, para recrear el famoso juego del comecocos.

La práctica en cuestión se dividió en varias partes, una continuación de otra, a las que se les irían implementando mejoras.

A la hora de dividir las tareas en el grupo, lo intentamos hacer de la manera más equitativa posible.

A medida que se realizaban cambios, se aplicaban juegos de pruebas para verificar su funcionamiento.

Todo el programa se realizó en lenguaje C, utilizando funciones dadas por los docentes.

A continuación se realizará una breve explicación de cómo se realizó cada uno de los apartados:

· ***Fase 1:***

Durante esta fase se tenía que modificar el programa inicial para poder implementarlo con threads.

Primero se tuvieron que crear las variables globales necesarias para guardar los datos de los fantasmas y el comecocos, y que más adelante pudieran entrar sin problemas. Era necesario que fueran globales debido a que si fueran locales, más adelante, cuando se modificarán las funciones para operar con los elementos, no podrían entrar a sus respectivos datos.

También junto a estas se crearon las estructuras de datos para almacenar los threads y la variable de control del programa.

Segundo, se modificaron las funciones de mover fantasmas y mover comecocos, debido a que estas serían las funciones manejadas por los threads, por lo que había que prepararlas para ello, para recibir parámetros, y devolver o no datos.

En estas funciones modificarían sus propios datos (las cuales como se comentó anteriormente pasaron a ser datos globales), que se trataría de una estructura donde se indicaba la posición actual del elemento (fantasma o comecocos), elemento en la posición actual, y siguiente posición. Se ha de tener en cuenta que cada función es un thread.

En esta fase no hay sincronización por lo que el funcionamiento no es el adecuado y aparecen problemas.

· ***Fase 2:***

En esta segunda fase se continuaría con la primera.

En este caso se tendría que implementar sincronización mediante ***mutex_locks.***

Se continúa con las mismas estructuras de datos e implementación de la fase anterior, la única diferencia es que ahora los threads, a la hora de cambiar datos en estructuras globales o cambiar datos en la pantalla, habrían de cerrar el lock, cambiar los valores o modificar la pantalla, y finalmente abrir el lock de nuevo.

Algo a tener en cuenta en esta fase era la de asegurarnos de que después de cerrar un lock, a continuación se tendría que abrir, y evitar cerrar uno, y volver a cerrar, ya que podría generar un ***deadlock.***

Ya en este caso el funcionamiento debería ser correcto y sin que aparezca basura por la pantalla y con un funcionamiento fluido y sin errores.

· Fase 3:

En esta fase se debían realizar grandes cambios.

Primeramente eliminar del programa principal la función de mover fantasma, ya que esta pasaría de ser un thread a un proceso, por lo que tendría su propia función ***main***.
El comecocos seguiría siendo un thread.

En este caso se tendría que generar una zona de memoria compartida para, entre los varios procesos, poder manejar la condición de final de juego.

También a la hora de crear los procesos, se les tendría que pasar por argumentos, sus propios datos a modificar junto, con el retardo, IPC de la memoria compartida, e IPC del tamaño del tablero junto con las filas y columnas.

En esta fase se tendría que realizar un set y actualización de la pantalla cada cierto tiempo que sería el retardo.

Como en la primera fase, está solo trataba de crear los procesos sin necesidad de sincronizarlos, por lo que se esperaban errores.

· ***Fase 4:***

En esta última fase, introduciría la sincronización en la fase 3.

En este caso a la hora de crear los procesos de los fantasmas, también se les tendría que pasar un IPC del semáforo, el cual trabajaría como un ***mutex_lock***.

Al igual que en la fase 2, se tendría que tener cuidado de no cerrar el semáforo varias veces seguidas ya que el proceso se quedaría atascado en su sección crítica lo que llevaría al proceso a un ***deadlock***.

2. Código

A continuación se muestra el código de cada apartado de la práctica

· *Cocos0.c (Cocos1 + Cocos2)*

```
#include <stdio.h>    /* incloure definicions de funcions estandard */
#include <stdlib.h>   /* per exit() */
#include <unistd.h>   /* per getpid() */
#include "winsuport.h"    /* incloure definicions de funcions propies
*/
#include <stdbool.h>
#include <pthread.h>
#include <stdint.h>   /* intptr_t per màquines de 64 bits */



#define MIN_FIL 7   /* definir limits de variables globals */
#define MAX_FIL 25
#define MIN_COL 10
#define MAX_COL 80
#define MAX_ELEMENTOS 10

       /* definir estructures d'informacio */
typedef struct {    /* per un objecte (menjacocos o fantasma) */
 int f;         /* posicio actual: fila */
 int c;         /* posicio actual: columna */
 int d;         /* direccio actual: [0..3] */
   float r;             /* per indicar un retard relati */
 char a;        /* caracter anterior en pos. actual */
} objecte;


/* variables globals */
int n_fil1, n_col;    /* dimensions del camp de joc */
char tauler[70];    /* nom del fitxer amb el laberint de joc */
char c_req;         /* caracter de pared del laberint */

objecte mc;         /* informacio del menjacocos */
```

```
objecte f1;          /* informacio del fantasma 1 */

int df[] = {-1, 0, 1, 0}; /* moviments de les 4 direccions possibles */
int dc[] = {0, -1, 0, 1}; /* dalt, esquerra, baix, dreta */

int cocos;       /* numero restant de cocos per menjar */
int retard;        /* valor del retard de moviment, en mil.lisegons */

/*
VAMOS A DEFINIR LA LISTA DE OBJETOS POSIBLES
LA LISTA RESERVARA MEMORIA PARA COMO MAXIMO 10 ELEMENTOS DE LOS CUALES
SIMEPRE EL ULTIMO SERA EL COMECOCOS Y LOS DEMAS LOS FANTASMAS
*/
objecte elementos[MAX_ELEMENTOS];

/*
AHORA VAMOS A DEFINIR LOS THREADS AL IGUAL QUE LOS ELEMENTOS
*/
pthread_t threads[MAX_ELEMENTOS];

/*
ESTO SERA EL SEMAFORO QUE CONTROLARÁ LA EJECUCION Y LAS SECCIONES
CRITICAS
*/
pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER; /* crea un sem.
Global*/

/*
A CONTINUACION VAMOS A DEFINIR UNA VARIABLE GLOBAL QUE NOS SERVIRA PARA
LO SIGUIENTE:
1) SERVIRA PARA SABER EL INDICE DEL COMECOCOS EN LAS ANTERIORES
ESTRUCTURAS
2) PODER RECORRER LAS ESTRUCTURAS PARA INICIARLAS O USARLAS
ESTA SE INICIA A 0 POR DARLE UN VALOR CUALQUIERA YA QUE NADA MÁS
EMPEZAR EL PROGRAMA SE CAMBIARA
*/
int totalElem = 0;

/*
LA SIGUIENTE VARIABLE NOS SERVIRA PARA DEFINIR SI SE ACABO EL JUEGO O
NO Y PUEDE TENER 3 ESTADOS:
0 --> COMECOCOS GANAS
1 --> FANTASMAS GANAN
```

```
2 --> JUGADOR APRIETA RETURN
*/
/*Prueba*/
int condicion = -1;


/* funcio per realitzar la carrega dels parametres de joc emmagatzemats
*/
/* dins d'un fitxer de text, el nom del qual es passa per referencia a
*/
/* 'nom_fit'; si es detecta algun problema, la funcio avorta l'execucio
*/
/* enviant un missatge per la sortida d'error i retornant el codi per-
*/
/* tinent al SO (segons comentaris al principi del programa).       */
void carrega_parametres(const char *nom_fit)
{
 FILE *fit;

 fit = fopen(nom_fit,"rt");    /* intenta obrir fitxer */
 if (fit == NULL)
 {
   fprintf(stderr,"No s'ha pogut obrir el fitxer \'%s\'\n",nom_fit);
   exit(2);
 }

 if (!feof(fit)) fscanf(fit,"%d %d %s
%c\n",&n_fil1,&n_col,tauler,&c_req);
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 if ((n_fil1 < MIN_FIL) || (n_fil1 > MAX_FIL) || (n_col < MIN_COL) ||
(n_col > MAX_COL))
 {
   fprintf(stderr,"Error: dimensions del camp de joc incorrectes:\n");
   fprintf(stderr,"\t%d =< n_fil1 (%d) =<
%d\n",MIN_FIL,n_fil1,MAX_FIL);
   fprintf(stderr,"\t%d =< n_col (%d) =< %d\n",MIN_COL,n_col,MAX_COL);
   fclose(fit);
   exit(3);
```

```
 }

 int i = 0;
  if (!feof(fit))  fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }

 printf("Datos del comecocos");
 printf(" %d | %d | %d | %f \n", elementos[i].f, elementos[i].c,
elementos[i].d, elementos[i].r);

 if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
 {
   fprintf(stderr,"Error: parametres menjacocos incorrectes:\n");
   fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
   fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
   fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
   fclose(fit);
   exit(4);
 }
 i++;
 if (!feof(fit))
 {
     while(!feof(fit))
     {
       fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);

       if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
       {
         fprintf(stderr,"Error: parametres fantasma 1 incorrectes:\n");
```

```
         fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
         fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
         fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
         fclose(fit);
         exit(5);
       }
        printf("\nDatos del fantasma numero %d ", i);
         printf(" %d | %d | %d | %f \n", elementos[i].f,
elementos[i].c, elementos[i].d, elementos[i].r);

       i++;
     }
 }else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 fclose(fit);       /* fitxer carregat: tot OK! */
  totalElem = i;
 printf("Elementos totales en juego %d | %d\n\n", totalElem, i);
 printf("Joc del MenjaCocos\n\tTecles: \'%c\', \'%c\', \'%c\', \'%c\',
RETURN-> sortir\n",
   TEC_AMUNT, TEC_AVALL, TEC_DRETA, TEC_ESQUER);
 printf("prem una tecla per continuar:\n");
 getchar();
}



/* funcio per inicialitar les variables i visualitzar l'estat inicial
del joc */
void inicialitza_joc(void)
{
 int r,i,j;
 char strin[12];
 int aux=0;

 r = win_carregatauler(tauler,n_fil1-1,n_col,c_req);
 if (r == 0)
```

```
{
   elementos[0].a = win_quincar(elementos[0].f,elementos[0].c);

  if (elementos[0].a == c_req) r = -6;     /* error: menjacocos sobre
pared */
  else
  {
    for (int z=1; z<totalElem;z++)
    {

      elementos[z].a = win_quincar(elementos[z].f,elementos[z].c);

      if (elementos[z].a == c_req) aux = 1;
    }

    if (aux == 1) r = -7; /* error: fantasma sobre pared */
    else
    {
      cocos = 0;       /* compta el numero total de cocos */
      for (i=0; i<n_fil1-1; i++)
        for (j=0; j<n_col; j++)
          if (win_quincar(i,j)=='.') cocos++;
            /*
            win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);
            */

            for(int z=1; z<totalElem;z++)
            {
              printf("Numero de fantasma a pintar %d", z);
              win_escricar(elementos[z].f,elementos[z].c,'1',NO_INV);
            }




          if (elementos[0].a == '.') cocos--; /* menja primer coco */

        sprintf(strin,"Cocos: %d", cocos); win_escristr(strin);
     }
  }
}

if (r != 0)
```

```
 { win_fi();
 fprintf(stderr,"Error: no s'ha pogut inicialitzar el joc:\n");
 switch (r)
 { case -1: fprintf(stderr,"  nom de fitxer erroni\n"); break;
   case -2: fprintf(stderr,"  numero de columnes d'alguna fila no
coincideix amb l'amplada del tauler de joc\n"); break;
   case -3: fprintf(stderr,"  numero de columnes del laberint
incorrecte\n"); break;
   case -4: fprintf(stderr,"  numero de files del laberint
incorrecte\n"); break;
   case -5: fprintf(stderr,"  finestra de camp de joc no oberta\n");
break;
   case -6: fprintf(stderr,"  posicio inicial del menjacocos damunt la
pared del laberint\n"); break;
   case -7: fprintf(stderr,"  posicio inicial del fantasma damunt la
pared del laberint\n"); break;
 }
 exit(7);
 }
}




/* funcio per moure un fantasma una posicio; retorna 1 si el fantasma
*/
/* captura al menjacocos, 0 altrament         */
void *mou_fantasma(void *index)
{
 objecte seg;
 int k, vk, nd, vd[3];
 int indice = (intptr_t) index;
  fflush(stdout);
 while (condicion == -1)
 {
   nd = 0;
   for (k=-1; k<=1; k++)   /* provar direccio actual i dir. veines */
   {
     pthread_mutex_lock(&mutex);
     vk = (elementos[indice].d + k) % 4;    /* direccio veina */
     if (vk < 0) vk += 4;     /* corregeix negatius */
     seg.f = elementos[indice].f + df[vk]; /* calcular posicio en la
nova dir.*/
```

```
     seg.c = elementos[indice].c + dc[vk];
     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
     pthread_mutex_unlock(&mutex); /* obre semafor */
     if ((seg.a==' ') || (seg.a=='.') || (seg.a=='0'))
     {
       vd[nd] = vk;       /* memoritza com a direccio possible */
       nd++;
     }
   }
   if (nd == 0)
   {
     pthread_mutex_lock(&mutex); /* tanca semafor */
     elementos[indice].d = (elementos[indice].d + 2) % 4;     /* canvia
totalment de sentit */
     pthread_mutex_unlock(&mutex); /* obre semafor */
   }
   else
   {
     pthread_mutex_lock(&mutex); /* tanca semafor */
     if (nd == 1)       /* si nomes pot en una direccio */
     {
       elementos[indice].d = vd[0];       /* li assigna aquesta */
     }
     else         /* altrament */
       elementos[indice].d = vd[rand() % nd];     /* segueix una dir.
aleatoria */
     pthread_mutex_unlock(&mutex); /* obre semafor */
     seg.f = elementos[indice].f + df[elementos[indice].d];  /*
calcular seguent posicio final */
     seg.c = elementos[indice].c + dc[elementos[indice].d];
     pthread_mutex_lock(&mutex);
     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */

win_escricar(elementos[indice].f,elementos[indice].c,elementos[indice].
a,NO_INV); /* esborra posicio anterior */
     elementos[indice].f = seg.f; elementos[indice].c = seg.c;
elementos[indice].a = seg.a;  /* actualitza posicio */
     win_escricar(elementos[indice].f,elementos[indice].c,
(char)('0'+indice),NO_INV);    /* redibuixa fantasma */
     if (elementos[indice].a == '0') condicion = 1;     /* ha capturat
menjacocos */
```

```
     pthread_mutex_unlock(&mutex); /* obre semafor */
   }
   win_retard(retard);
 //printf("Valor de a -->%c\n", elementos[indice].a);
 }
 return ((void *) NULL);
}




/* funcio per moure el menjacocos una posicio, en funcio de la direccio
de   */
/* moviment actual; retorna -1 si s'ha premut RETURN, 1 si s'ha menjat
tots  */
/* els cocos, i 0 altrament */
void *mou_menjacocos(void *n)
{
 char strin[99];
 objecte seg;
 int tec;
 while (condicion == -1)
 {
   //fprintf(stderr,"Antes de escoger tecla\n");
   pthread_mutex_lock(&mutex);
   tec = win_gettec();
   pthread_mutex_unlock(&mutex);
   if (tec != 0)
   pthread_mutex_lock(&mutex); /* tanca semafor */
   switch (tec)    /* modificar direccio menjacocos segons tecla */
   {
     case TEC_AMUNT:   elementos[0].d = 0; break;
     case TEC_ESQUER:  elementos[0].d = 1; break;
     case TEC_AVALL:   elementos[0].d = 2; break;
     case TEC_DRETA:   elementos[0].d = 3; break;
     case TEC_RETURN:  condicion = 2; break;
   }
   pthread_mutex_unlock(&mutex); /* obre semafor */
   seg.f = elementos[0].f + df[elementos[0].d];  /* calcular seguent
posicio */
   seg.c = elementos[0].c + dc[elementos[0].d];
   pthread_mutex_lock(&mutex);
```

```
   seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
   pthread_mutex_unlock(&mutex);
   if ((seg.a == ' ') || (seg.a == '.'))
   {
     pthread_mutex_lock(&mutex); /* tanca semafor */
     win_escricar(elementos[0].f,elementos[0].c,' ',NO_INV);   /*
esborra posicio anterior */
     elementos[0].f = seg.f; elementos[0].c = seg.c;     /* actualitza
posicio */
     elementos[0].a = win_quincar(seg.f,seg.c);
     win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);   /*
redibuixa menjacocos */
     pthread_mutex_unlock(&mutex); /* obre semafor */
     if (seg.a == '.')
     {
       cocos--;
       pthread_mutex_lock(&mutex);
       sprintf(strin,"Cocos: %d", cocos); win_escristr(strin);
       if (cocos == 0) condicion = 0;
       pthread_mutex_unlock(&mutex);
     }
   }
   win_retard(retard);
 }

 return ((void *) NULL);
}



/* programa principal           */
int main(int n_args, const char *ll_args[])
{
 int rc;   /* variables locals */
 int i = 0;
 srand(getpid());    /* inicialitza numeros aleatoris */

 if ((n_args != 2) && (n_args !=3))
 {
   fprintf(stderr,"Comanda: cocos0 fit_param [retard] [numero
fantasmas]\n");
   exit(1);
```

```
 }

 /*
 ANTES DE INICIAR LOS PARAMETROS VAMOS A RECOGER EL NUMERO DE ELEMENTOS
 TENER EN CUENTA QUE DE POR PARAMETRO SE ENTRA EL NUMERO DE FANTASMAS
POR ENDE SE LE HA DE SUMAR 1 TENIENDO EN CUENTA EL COMECOCOS
 */
 pthread_mutex_init(&mutex, NULL); /* inicialitza el semafor */
 carrega_parametres(ll_args[1]);
 printf("El numero total de elementos sera de %d\n -Fantasmas: %d\n
-Comecocos: 1\n", totalElem, totalElem-1);

 if (n_args == 3) retard = atoi(ll_args[2]);
 else retard = 100;

 rc = win_ini(&n_fil1,&n_col,'+',INVERS);  /* intenta crear taulell */
 if (rc == 0)    /* si aconsegueix accedir a l'entorn CURSES */
 {
   inicialitza_joc();

   /*
   JUSTO CUANDO SE INICIA EL JUEGO Y LOS ELEMENTOS ESTAN SOBRE EL
TABLERO, SE INICIAN LOS THREADS PARA QUE COMIENCEN A EJECUTARSE
   */
  printf("Hay %d elementos\n", totalElem);
  while(i<totalElem)
  {
   if (i == 0)
   {
     if (pthread_create(&threads[i],NULL,mou_menjacocos,(void *) NULL)
== 0);
   }else
   {
       if(pthread_create(&threads[i],NULL,mou_fantasma,(void *)
(intptr_t) i) == 0);
   }
   i++;
  }
 /*
  UNA VEZ CREADOS SE REALIZA EL JOIN PARA QUE ESPERAR A QUE ACABEN
  */

 i=0;
```

```
 while(i<totalElem)
 {
   pthread_join(threads[i], (void *) NULL);
   i++;
 }

 win_fi();

 if (condicion == 0)
 {
   printf("EL JUGADOR GANO");
 }else if (condicion == 1)
 {
   printf("VAYA LOS FANTASMAS GANARON");
 }else if (condicion == 2)
 {
   printf("EL JUGADOR DETUVO EL JUEGO");
 }
}
else
{
 fprintf(stderr,"Error: no s'ha pogut crear el taulell:\n");
 switch (rc)
 {
   case -1: fprintf(stderr,"camp de joc ja creat!\n");
     break;
   case -2: fprintf(stderr,"no s'ha pogut inicialitzar l'entorn de
curses!\n");
     break;
   case -3: fprintf(stderr,"les mides del camp demanades son massa
grans!\n");
     break;
   case -4: fprintf(stderr,"no s'ha pogut crear la finestra!\n");
     break;
 }
 exit(6);
 }
 pthread_mutex_destroy(&mutex);
 return(0);
}
```

· ***Cocos3.c***

```c
#include <stdio.h>    /* incloure definicions de funcions estandard */
#include <stdlib.h>   /* per exit() */
#include <unistd.h>   /* per getpid() */
#include "winsuport2.h"   /* incloure definicions de funcions propies
*/
#include <stdbool.h>
#include <stdint.h>   /* intptr_t per màquines de 64 bits */
#include <sys/types.h>
#include <sys/wait.h>
#include "memoria.h"
#include <pthread.h>


#define MIN_FIL 7   /* definir limits de variables globals */
#define MAX_FIL 25
#define MIN_COL 10
#define MAX_COL 80
#define MAX_ELEMENTOS 10

       /* definir estructures d'informacio */
typedef struct {    /* per un objecte (menjacocos o fantasma) */
 int f;         /* posicio actual: fila */
 int c;         /* posicio actual: columna */
 int d;         /* direccio actual: [0..3] */
   float r;             /* per indicar un retard relati */
 char a;        /* caracter anterior en pos. actual */
} objecte;


/* variables globals */
int n_fil1, n_col;    /* dimensions del camp de joc */
char tauler[70];    /* nom del fitxer amb el laberint de joc */
char c_req;         /* caracter de pared del laberint */

objecte mc;         /* informacio del menjacocos */
objecte f1;         /* informacio del fantasma 1 */

int df[] = {-1, 0, 1, 0}; /* moviments de les 4 direccions possibles */
int dc[] = {0, -1, 0, 1}; /* dalt, esquerra, baix, dreta */

int cocos;      /* numero restant de cocos per menjar */
int retard;       /* valor del retard de moviment, en mil.lisegons */
```

```
/*
VAMOS A DEFINIR LA LISTA DE OBJETOS POSIBLES
LA LISTA RESERVARA MEMORIA PARA COMO MAXIMO 10 ELEMENTOS DE LOS CUALES
SIMEPRE EL ULTIMO SERA EL COMECOCOS Y LOS DEMAS LOS FANTASMAS
*/
objecte elementos[MAX_ELEMENTOS];

/*
AHORA VAMOS A DEFINIR LOS THREADS AL IGUAL QUE LOS ELEMENTOS
*/
pid_t tpid[MAX_ELEMENTOS];    /* taula d'identificadors dels processos
fill */

pthread_t threads[1];

/*
A CONTINUACION VAMOS A DEFINIR UNA VARIABLE GLOBAL QUE NOS SERVIRA PARA
LO SIGUIENTE:
1) SERVIRA PARA SABER EL INDICE DEL COMECOCOS EN LAS ANTERIORES
ESTRUCTURAS
2) PODER RECORRER LAS ESTRUCTURAS PARA INICIARLAS O USARLAS
ESTA SE INICIA A 0 POR DARLE UN VALOR CUALQUIERA YA QUE NADA MÁS
EMPEZAR EL PROGRAMA SE CAMBIARA
*/
int totalElem = 0;

/*
LA SIGUIENTE VARIABLE NOS SERVIRA PARA DEFINIR SI SE ACABO EL JUEGO O
NO Y PUEDE TENER 3 ESTADOS:
0 --> COMECOCOS GANAS
1 --> FANTASMAS GANAN
2 --> JUGADOR APRIETA RETURN
*/
/*Prueba*/
int condicion = -1;

int *p_sharedMemory, id_sharedMemory;
/* funcio per realitzar la carrega dels parametres de joc emmagatzemats
*/
/* dins d'un fitxer de text, el nom del qual es passa per referencia a
*/
```

```
/* 'nom_fit'; si es detecta algun problema, la funcio avorta l'execucio
*/
/* enviant un missatge per la sortida d'error i retornant el codi per-
*/
/* tinent al SO (segons comentaris al principi del programa).       */
void carrega_parametres(const char *nom_fit)
{
 FILE *fit;

 fit = fopen(nom_fit,"rt");    /* intenta obrir fitxer */
 if (fit == NULL)
 {
   fprintf(stderr,"No s'ha pogut obrir el fitxer \'%s\'\n",nom_fit);
   exit(2);
 }

 if (!feof(fit)) fscanf(fit,"%d %d %s
%c\n",&n_fil1,&n_col,tauler,&c_req);
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 if ((n_fil1 < MIN_FIL) || (n_fil1 > MAX_FIL) || (n_col < MIN_COL) ||
(n_col > MAX_COL))
 {
   fprintf(stderr,"Error: dimensions del camp de joc incorrectes:\n");
   fprintf(stderr,"\t%d =< n_fil1 (%d) =<
%d\n",MIN_FIL,n_fil1,MAX_FIL);
   fprintf(stderr,"\t%d =< n_col (%d) =< %d\n",MIN_COL,n_col,MAX_COL);
   fclose(fit);
   exit(3);
 }

 int i = 0;
  if (!feof(fit))  fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
```

```
 }

 printf("Datos del comecocos");
 printf(" %d | %d | %d | %f \n", elementos[i].f, elementos[i].c,
elementos[i].d, elementos[i].r);

 if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
 {
   fprintf(stderr,"Error: parametres menjacocos incorrectes:\n");
   fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
   fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
   fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
   fclose(fit);
   exit(4);
 }
 i++;
 if (!feof(fit))
 {
     while(!feof(fit))
     {
       fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);

       if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
       {
         fprintf(stderr,"Error: parametres fantasma 1 incorrectes:\n");
         fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
         fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
         fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
         fclose(fit);
         exit(5);
       }
        printf("\nDatos del fantasma numero %d ", i);
         printf(" %d | %d | %d | %f \n", elementos[i].f,
elementos[i].c, elementos[i].d, elementos[i].r);
```

```
      i++;
    }
 }else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 fclose(fit);      /* fitxer carregat: tot OK! */
  totalElem = i;
 printf("Elementos totales en juego %d | %d\n\n", totalElem, i);
 printf("Valores de filas y columnas %d | %d\n\n", n_fil1, n_col);
 printf("Joc del MenjaCocos\n\tTecles: \'%c\', \'%c\', \'%c\', \'%c\',
RETURN-> sortir\n",
   TEC_AMUNT, TEC_AVALL, TEC_DRETA, TEC_ESQUER);
 printf("prem una tecla per continuar:\n");
 getchar();
}



/* funcio per inicialitar les variables i visualitzar l'estat inicial
del joc */
void inicialitza_joc(void)
{
 int r,i,j;
 char strin[12];
 int aux=0;

 r = win_carregatauler(tauler,n_fil1-1,n_col,c_req);

 if (r == 0)
 {
    elementos[0].a = win_quincar(elementos[0].f,elementos[0].c);

   if (elementos[0].a == c_req) r = -6;    /* error: menjacocos sobre
pared */
   else
   {
     for (int z=1; z<totalElem;z++)
     {
```

```
      elementos[z].a = win_quincar(elementos[z].f,elementos[z].c);

      if (elementos[z].a == c_req) aux = 1;
    }

    if (aux == 1) r = -7; /* error: fantasma sobre pared */
    else
    {
      cocos = 0;       /* compta el numero total de cocos */
      for (i=0; i<n_fil1-1; i++)
        for (j=0; j<n_col; j++)
          if (win_quincar(i,j)=='.') cocos++;

            win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);


            for(int z=1; z<totalElem;z++)
            {
              printf("Numero de fantasma a pintar %d", z);
              win_escricar(elementos[z].f,elementos[z].c,'1',NO_INV);
            }




          if (elementos[0].a == '.') cocos--; /* menja primer coco */

        sprintf(strin,"Cocos: %d", cocos); win_escristr(strin);
     }
   }
 }

 if (r != 0)
 { win_fi();
 fprintf(stderr,"Error: no s'ha pogut inicialitzar el joc:\n");
 switch (r)
 { case -1: fprintf(stderr,"  nom de fitxer erroni\n"); break;
   case -2: fprintf(stderr,"  numero de columnes d'alguna fila no
coincideix amb l'amplada del tauler de joc\n"); break;
   case -3: fprintf(stderr,"  numero de columnes del laberint
incorrecte\n"); break;
```

```
   case -4: fprintf(stderr,"  numero de files del laberint
incorrecte\n"); break;
   case -5: fprintf(stderr,"  finestra de camp de joc no oberta\n");
break;
   case -6: fprintf(stderr,"  posicio inicial del menjacocos damunt la
pared del laberint\n"); break;
   case -7: fprintf(stderr,"  posicio inicial del fantasma damunt la
pared del laberint\n"); break;
 }
 exit(7);
 }
}

/* funcio per moure el menjacocos una posicio, en funcio de la direccio
de   */
/* moviment actual; retorna -1 si s'ha premut RETURN, 1 si s'ha menjat
tots  */
/* els cocos, i 0 altrament */
void *mou_menjacocos(void *n)
{
 char strin[99];
 objecte seg;
 char cocos_left[50];
 int tec;
 //fprintf(stderr, "Comecocos empezo");
 while (*p_sharedMemory == -1)
 {
   //fprintf(stderr,"Antes de escoger tecla\n");
   tec = win_gettec();
   if (tec != 0)
   switch (tec)    /* modificar direccio menjacocos segons tecla */
   {
     case TEC_AMUNT:   elementos[0].d = 0; break;
     case TEC_ESQUER:  elementos[0].d = 1; break;
     case TEC_AVALL:   elementos[0].d = 2; break;
     case TEC_DRETA:   elementos[0].d = 3; break;
     case TEC_RETURN:  *p_sharedMemory = 2; break;
   }

   seg.f = elementos[0].f + df[elementos[0].d];  /* calcular seguent
posicio */
   seg.c = elementos[0].c + dc[elementos[0].d];
```

```
   seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
   if ((seg.a == ' ') || (seg.a == '.'))
   {
     win_escricar(elementos[0].f,elementos[0].c,' ',NO_INV);   /*
esborra posicio anterior */
     elementos[0].f = seg.f; elementos[0].c = seg.c;     /* actualitza
posicio */
     elementos[0].a = win_quincar(seg.f,seg.c);
     win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);   /*
redibuixa menjacocos */
     if (seg.a == '.')
     {
       cocos--;

       sprintf(strin,"Cocosno s: %d", cocos); win_escristr(strin);
       if (cocos == 0) *p_sharedMemory = 0;

     }
   }
   win_retard(retard);
   sprintf(cocos_left,"Cocos: %d",cocos);
   win_escristr(cocos_left);
   win_update();
 }

 return ((void *) NULL);
}



/* programa principal           */
int main(int n_args, const char *ll_args[])
{
 int rc;   /* variables locals */
 int i = 0;
 srand(getpid());    /* inicialitza numeros aleatoris */
 char object_str[100];
 char idSM_str[10];
 char a1[20], a2[20], a3[20], a4[20];
 void *p_win;
 int id_win;
```

```
 if ((n_args != 2) && (n_args !=3))
 {
   fprintf(stderr,"Comanda: cocos0 fit_param [retard] [numero
fantasmas]\n");
   exit(1);
 }

 /*
 INICIAR MEMORIA COMPARTIDA
 */
 id_sharedMemory = ini_mem(sizeof(int));         /* crear zona mem.
compartida */
 p_sharedMemory = map_mem(id_sharedMemory);  /* obtenir adres. de mem.
compartida */
 *p_sharedMemory = condicion;

 /*
 CARGA DE PARAMETROS
 */
 carrega_parametres(ll_args[1]);
 printf("El numero total de elementos[indice] sera de %d\n -Fantasmas:
%d\n -Comecocos: 1\n", totalElem, totalElem-1);

 if (n_args == 3) retard = atoi(ll_args[2]);
 else retard = 100;
 rc = win_ini(&n_fil1,&n_col,'+',INVERS);  /* intenta crear taulell */
 if (rc >= 0)    /* si aconsegueix accedir a l'entorn CURSES */
 {

   /*
   INICIAR MEMORIA COMPARTIDA DE CAMPO
   */
   id_win = ini_mem(rc); /* crear zona mem. compartida */
   p_win = map_mem(id_win);  /* obtenir adres. de mem. compartida */

   win_set(p_win,n_fil1,n_col);    /* crea acces a finestra oberta */

   inicialitza_joc();
   win_update();
   /*
   JUSTO CUANDO SE INICIA EL JUEGO Y LOS elementos[indice] ESTAN SOBRE
EL TABLERO, SE INICIAN LOS THREADS PARA QUE COMIENCEN A EJECUTARSE
   */
```

```
  sprintf(idSM_str, "%i", id_sharedMemory);
  i=0;
  while(i<totalElem)
  {
   if (i==0)
   {
     pthread_create(&threads[0],NULL,mou_menjacocos,(void *) NULL);
   }else{
     tpid[i] = fork();    /* crea un nou proces */
     if (tpid[i] == (pid_t) 0)    /* branca del fill */
     {
       sprintf(object_str, "%d,%d,%d,%.2f,%c", elementos[i].f,
elementos[i].c, elementos[i].d, elementos[i].r, elementos[i].a);
       sprintf(a1,"%i",id_win);
       sprintf(a2,"%i",n_fil1);
       sprintf(a3,"%i",n_col);
       sprintf(a4,"%i",i);
       /*
       PARAMETROS A ENVIAR
       PARAM0 --> Nombre del programa
       PARAM1 --> Objecto fantasma
       PARAM2 --> Retardo
       PARAM3 --> Id de la memoria comaprtida
       */
       execlp("./Fantasmas3", "Fantasmas3", object_str, ll_args[2],
idSM_str, a1, a2, a3, a4, (char *)0);
       fprintf(stderr,"error: no puc executar el process fill
\'mp_car\'\n");
       elim_mem(id_sharedMemory);
       elim_mem(id_win);
       exit(0);
     }
   }
   i++;
  }
 /*
  UNA VEZ CREADOS SE REALIZA EL JOIN PARA QUE ESPERAR A QUE ACABEN
  */
 i=0;
 while(i<totalElem)
 {
   if (i==0)
   {
```

```
     pthread_join(threads[i], (void *) NULL);
   }else{
      waitpid(tpid[i],NULL,0); /* espera finalitzacio d'un fill */
   }
   i++;
 }


 win_fi();

 if (*p_sharedMemory == 0)
 {
   printf("EL JUGADOR GANO\n");
 }else if (*p_sharedMemory == 1)
 {
   printf("VAYA LOS FANTASMAS GANARON\n");
 }else if (*p_sharedMemory == 2)
 {
   printf("EL JUGADOR DETUVO EL JUEGO\n");
 }
}
else
{
 elim_mem(id_sharedMemory);
 elim_mem(id_win);
 switch (rc)
 {
   case -1: fprintf(stderr,"camp de joc ja creat!\n");
     break;
   case -2: fprintf(stderr,"no s'ha pogut inicialitzar l'entorn de
curses!\n");
     break;
   case -3: fprintf(stderr,"les mides del camp demanades son massa
grans!\n");
     break;
   case -4: fprintf(stderr,"no s'ha pogut crear la finestra!\n");
     break;
 }
 exit(6);
 }

 elim_mem(id_sharedMemory);
 return(0);
```

```
}
```

· Fantasmas3.c

```
#include <stdio.h>    /* incloure definicions de funcions estandard */
#include <stdlib.h>   /* per exit() */
#include <unistd.h>   /* per getpid() */
#include "winsuport2.h"   /* incloure definicions de funcions propies
*/
#include <stdbool.h>
#include <stdint.h>   /* intptr_t per màquines de 64 bits */
#include "memoria.h"
/*
Estructura
*/

       /* definir estructures d'informacio */
typedef struct {    /* per un objecte (menjacocos o fantasma) */
 int f;         /* posicio actual: fila */
 int c;         /* posicio actual: columna */
 int d;         /* direccio actual: [0..3] */
   float r;             /* per indicar un retard relati */
 char a;        /* caracter anterior en pos. actual */
} objecte;

int df[] = {-1, 0, 1, 0}; /* moviments de les 4 direccions possibles */
int dc[] = {0, -1, 0, 1}; /* dalt, esquerra, baix, dreta */



/* funcio per moure un fantasma una posicio; retorna 1 si el fantasma
*/
/* captura al menjacocos, 0 altrament         */
int main(int n_args, char *ll_args[])
{
 objecte elementos;
 objecte seg;
 int k, vk, nd, vd[3], id_win;
 int *p_sharedMemory, id_sharedMemory;
 void *p_win;
 int retard = atoi(ll_args[2]);

 /*
```

```
 SE CONECTA A LA MEMORIA COMPARTIDA Y COMPRUEBA SI ES CORRECTA
 */
 id_sharedMemory = atoi(ll_args[3]);
 p_sharedMemory = map_mem(id_sharedMemory);  /* obtenir adres. de mem.
compartida */
 if (p_sharedMemory == (int*) -1)
 {   fprintf(stderr,"proces (%d): error en identificador de
memoria\n",(int)getpid());
 exit(0);
 }
 //mapa
 int n_fill = atoi(ll_args[5]);
 int n_col = atoi(ll_args[6]);
 //
 id_win = atoi(ll_args[4]);
 p_win = map_mem(id_win);  /* obtenir adres. de mem. compartida */

 if (p_win  == (int*) -1)
 {   fprintf(stderr,"proces (%d): error en identificador de
memoria\n",(int)getpid());
 exit(0);
 }

 win_set(p_win,n_fill,n_col);    /* crea acces a finestra oberta */
  /*
 RECUPERAMOS EL OBJETO
 */
 sscanf(ll_args[1], "%d,%d,%d,%f,%c", &elementos.f, &elementos.c,
&elementos.d, &elementos.r, &elementos.a);
 fflush(stdout);
 int numero = atoi(ll_args[7]);
 while (*p_sharedMemory == -1)
 {
   nd = 0;
   for (k=-1; k<=1; k++)   /* provar direccio actual i dir. veines */
   {

     vk = (elementos.d + k) % 4;   /* direccio veina */
     if (vk < 0) vk += 4;    /* corregeix negatius */
     seg.f = elementos.f + df[vk]; /* calcular posicio en la nova
dir.*/
     seg.c = elementos.c + dc[vk];
```

```
     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */

     if ((seg.a==' ') || (seg.a=='.') || (seg.a=='0'))
     {
       vd[nd] = vk;       /* memoritza com a direccio possible */
       nd++;
     }
   }
   if (nd == 0)
   {

     elementos.d = (elementos.d + 2) % 4;    /* canvia totalment de
sentit */

   }
   else
   {

     if (nd == 1)       /* si nomes pot en una direccio */
     {
       elementos.d = vd[0];       /* li assigna aquesta */
     }
     else         /* altrament */
       elementos.d = vd[rand() % nd];    /* segueix una dir. aleatoria
*/

     seg.f = elementos.f + df[elementos.d];  /* calcular seguent
posicio final */
     seg.c = elementos.c + dc[elementos.d];

     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
     win_escricar(elementos.f,elementos.c,elementos.a,NO_INV); /*
esborra posicio anterior */
     elementos.f = seg.f; elementos.c = seg.c; elementos.a = seg.a;  /*
actualitza posicio */
     win_escricar(elementos.f,elementos.c, (char)('0'+numero),NO_INV);
/* redibuixa fantasma */
     if (elementos.a == '0') *p_sharedMemory = 1;    /* ha capturat
menjacocos */

   }
```

```
   win_retard(retard);
 }
 elim_mem(id_sharedMemory);
 elim_mem(id_win);
 exit(0);
}
```

· Cocos4.c

```
#include <stdio.h>    /* incloure definicions de funcions estandard */
#include <stdlib.h>   /* per exit() */
#include <unistd.h>   /* per getpid() */
#include "winsuport2.h"   /* incloure definicions de funcions propies
*/
#include <stdbool.h>
#include <stdint.h>   /* intptr_t per màquines de 64 bits */
#include <sys/types.h>
#include <sys/wait.h>
#include "memoria.h"
#include <pthread.h>
#include "semafor.h"


#define MIN_FIL 7   /* definir limits de variables globals */
#define MAX_FIL 25
#define MIN_COL 10
#define MAX_COL 80
#define MAX_ELEMENTOS 10

       /* definir estructures d'informacio */
typedef struct {    /* per un objecte (menjacocos o fantasma) */
 int f;         /* posicio actual: fila */
 int c;         /* posicio actual: columna */
 int d;         /* direccio actual: [0..3] */
   float r;             /* per indicar un retard relati */
 char a;        /* caracter anterior en pos. actual */
} objecte;


/* variables globals */
int n_fil1, n_col;    /* dimensions del camp de joc */
char tauler[70];    /* nom del fitxer amb el laberint de joc */
char c_req;         /* caracter de pared del laberint */

objecte mc;         /* informacio del menjacocos */
objecte f1;         /* informacio del fantasma 1 */

int df[] = {-1, 0, 1, 0}; /* moviments de les 4 direccions possibles */
int dc[] = {0, -1, 0, 1}; /* dalt, esquerra, baix, dreta */
```

```
int cocos;      /* numero restant de cocos per menjar */
int retard;       /* valor del retard de moviment, en mil.lisegons */

/*
VAMOS A DEFINIR LA LISTA DE OBJETOS POSIBLES
LA LISTA RESERVARA MEMORIA PARA COMO MAXIMO 10 ELEMENTOS DE LOS CUALES
SIMEPRE EL ULTIMO SERA EL COMECOCOS Y LOS DEMAS LOS FANTASMAS
*/
objecte elementos[MAX_ELEMENTOS];

/*
AHORA VAMOS A DEFINIR LOS THREADS AL IGUAL QUE LOS ELEMENTOS
*/
pid_t tpid[MAX_ELEMENTOS];    /* taula d'identificadors dels processos
fill */

pthread_t thread;

/*
A CONTINUACION VAMOS A DEFINIR UNA VARIABLE GLOBAL QUE NOS SERVIRA PARA
LO SIGUIENTE:
1) SERVIRA PARA SABER EL INDICE DEL COMECOCOS EN LAS ANTERIORES
ESTRUCTURAS
2) PODER RECORRER LAS ESTRUCTURAS PARA INICIARLAS O USARLAS
ESTA SE INICIA A 0 POR DARLE UN VALOR CUALQUIERA YA QUE NADA MÁS
EMPEZAR EL PROGRAMA SE CAMBIARA
*/
int totalElem = 0;

/*
LA SIGUIENTE VARIABLE NOS SERVIRA PARA DEFINIR SI SE ACABO EL JUEGO O
NO Y PUEDE TENER 3 ESTADOS:
0 --> COMECOCOS GANAS
1 --> FANTASMAS GANAN
2 --> JUGADOR APRIETA RETURN
*/
/*Prueba*/
int condicion = -1;

int *p_sharedMemory, id_sharedMemory;
```

```
int id_sem = 0;
/* funcio per realitzar la carrega dels parametres de joc emmagatzemats
*/
/* dins d'un fitxer de text, el nom del qual es passa per referencia a
*/
/* 'nom_fit'; si es detecta algun problema, la funcio avorta l'execucio
*/
/* enviant un missatge per la sortida d'error i retornant el codi per-
*/
/* tinent al SO (segons comentaris al principi del programa).       */
void carrega_parametres(const char *nom_fit)
{
 FILE *fit;

 fit = fopen(nom_fit,"rt");    /* intenta obrir fitxer */
 if (fit == NULL)
 {
   fprintf(stderr,"No s'ha pogut obrir el fitxer \'%s\'\n",nom_fit);
   exit(2);
 }

 if (!feof(fit)) fscanf(fit,"%d %d %s
%c\n",&n_fil1,&n_col,tauler,&c_req);
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 if ((n_fil1 < MIN_FIL) || (n_fil1 > MAX_FIL) || (n_col < MIN_COL) ||
(n_col > MAX_COL))
 {
   fprintf(stderr,"Error: dimensions del camp de joc incorrectes:\n");
   fprintf(stderr,"\t%d =< n_fil1 (%d) =<
%d\n",MIN_FIL,n_fil1,MAX_FIL);
   fprintf(stderr,"\t%d =< n_col (%d) =< %d\n",MIN_COL,n_col,MAX_COL);
   fclose(fit);
   exit(3);
 }

 int i = 0;
  if (!feof(fit))  fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);
```

```
 else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }

 printf("Datos del comecocos");
 printf(" %d | %d | %d | %f \n", elementos[i].f, elementos[i].c,
elementos[i].d, elementos[i].r);

 if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
 {
   fprintf(stderr,"Error: parametres menjacocos incorrectes:\n");
   fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
   fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
   fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
   fclose(fit);
   exit(4);
 }
 i++;
 if (!feof(fit))
 {
     while(!feof(fit))
     {
       fscanf(fit,"%d %d %d
%f\n",&elementos[i].f,&elementos[i].c,&elementos[i].d,&elementos[i].r);

       if ((elementos[i].f < 1) || (elementos[i].f > n_fil1-3) ||
(elementos[i].c < 1) || (elementos[i].c > n_col-2) || (elementos[i].d <
0) || (elementos[i].d > 3))
       {
         fprintf(stderr,"Error: parametres fantasma 1 incorrectes:\n");
         fprintf(stderr,"\t1 =< f1.f (%d) =< n_fil1-3
(%d)\n",elementos[i].f,(n_fil1-3));
         fprintf(stderr,"\t1 =< f1.c (%d) =< n_col-2
(%d)\n",elementos[i].c,(n_col-2));
         fprintf(stderr,"\t0 =< f1.d (%d) =< 3\n",elementos[i].d);
         fclose(fit);
```

```
          exit(5);
        }
         printf("\nDatos del fantasma numero %d ", i);
          printf(" %d | %d | %d | %f \n", elementos[i].f,
elementos[i].c, elementos[i].d, elementos[i].r);

        i++;
      }
 }else
 {
   fprintf(stderr,"Falten parametres al fitxer \'%s\'\n",nom_fit);
   fclose(fit);
   exit(2);
 }
 fclose(fit);       /* fitxer carregat: tot OK! */
  totalElem = i;
 printf("Elementos totales en juego %d | %d\n\n", totalElem, i);
 printf("Valores de filas y columnas %d | %d\n\n", n_fil1, n_col);
 printf("Joc del MenjaCocos\n\tTecles: \'%c\', \'%c\', \'%c\', \'%c\',
RETURN-> sortir\n",
   TEC_AMUNT, TEC_AVALL, TEC_DRETA, TEC_ESQUER);
 printf("prem una tecla per continuar:\n");
 getchar();
}



/* funcio per inicialitar les variables i visualitzar l'estat inicial
del joc */
void inicialitza_joc(void)
{
 int r,i,j;
 char strin[12];
 int aux=0;

 r = win_carregatauler(tauler,n_fil1-1,n_col,c_req);

 if (r == 0)
 {
    elementos[0].a = win_quincar(elementos[0].f,elementos[0].c);
```

```
   if (elementos[0].a == c_req) r = -6;     /* error: menjacocos sobre
pared */
   else
   {
     for (int z=1; z<totalElem;z++)
     {

       elementos[z].a = win_quincar(elementos[z].f,elementos[z].c);

       if (elementos[z].a == c_req) aux = 1;
     }

     if (aux == 1) r = -7; /* error: fantasma sobre pared */
     else
     {
       cocos = 0;       /* compta el numero total de cocos */
       for (i=0; i<n_fil1-1; i++)
         for (j=0; j<n_col; j++)
           if (win_quincar(i,j)=='.') cocos++;

             win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);


             for(int z=1; z<totalElem;z++)
             {
               win_escricar(elementos[z].f,elementos[z].c,'1',NO_INV);
             }




           if (elementos[0].a == '.') cocos--; /* menja primer coco */

         sprintf(strin,"Cocos: %d", cocos); win_escristr(strin);
      }
   }
 }

 if (r != 0)
 { win_fi();
 fprintf(stderr,"Error: no s'ha pogut inicialitzar el joc:\n");
 switch (r)
 { case -1: fprintf(stderr,"  nom de fitxer erroni\n"); break;
```

```
   case -2: fprintf(stderr,"  numero de columnes d'alguna fila no
coincideix amb l'amplada del tauler de joc\n"); break;
   case -3: fprintf(stderr,"  numero de columnes del laberint
incorrecte\n"); break;
   case -4: fprintf(stderr,"  numero de files del laberint
incorrecte\n"); break;
   case -5: fprintf(stderr,"  finestra de camp de joc no oberta\n");
break;
   case -6: fprintf(stderr,"  posicio inicial del menjacocos damunt la
pared del laberint\n"); break;
   case -7: fprintf(stderr,"  posicio inicial del fantasma damunt la
pared del laberint\n"); break;
 }
 exit(7);
 }
}

/* funcio per moure el menjacocos una posicio, en funcio de la direccio
de   */
/* moviment actual; retorna -1 si s'ha premut RETURN, 1 si s'ha menjat
tots  */
/* els cocos, i 0 altrament */
void *mou_menjacocos(void *n)
{
 char strin[99];
 objecte seg;
 char cocos_left[50];
 int tec;
 //fprintf(stderr, "Comecocos empezo");
 while (*p_sharedMemory == -1)
 {
   //fprintf(stderr,"Antes de escoger tecla\n");
   waitS(id_sem);
   tec = win_gettec();
   signalS(id_sem);
   if (tec != 0)
   waitS(id_sem);
   switch (tec)     /* modificar direccio menjacocos segons tecla */
   {
     case TEC_AMUNT:   elementos[0].d = 0; break;
     case TEC_ESQUER:  elementos[0].d = 1; break;
     case TEC_AVALL:   elementos[0].d = 2; break;
     case TEC_DRETA:   elementos[0].d = 3; break;
```

```
     case TEC_RETURN:  *p_sharedMemory = 2; break;
   }
   signalS(id_sem);
   waitS(id_sem);
   seg.f = elementos[0].f + df[elementos[0].d];  /* calcular seguent
posicio */
   seg.c = elementos[0].c + dc[elementos[0].d];
   seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
   signalS(id_sem);
   if ((seg.a == ' ') || (seg.a == '.'))
   {
     waitS(id_sem);
     win_escricar(elementos[0].f,elementos[0].c,' ',NO_INV);   /*
esborra posicio anterior */
     elementos[0].f = seg.f; elementos[0].c = seg.c;     /* actualitza
posicio */
     elementos[0].a = win_quincar(seg.f,seg.c);
     win_escricar(elementos[0].f,elementos[0].c,'0',NO_INV);   /*
redibuixa menjacocos */
     signalS(id_sem);
     if (seg.a == '.')
     {
       cocos--;
       sprintf(strin,"Cocosno s: %d", cocos); win_escristr(strin);
       waitS(id_sem);
       if (cocos == 0) *p_sharedMemory = 0;
       signalS(id_sem);

     }
   }
   win_retard(retard);
   sprintf(cocos_left,"Cocos: %d",cocos);
   win_escristr(cocos_left);
   win_update();
 }

 return ((void *) NULL);
}



/* programa principal           */
```

```
int main(int n_args, const char *ll_args[])
{
 int rc;   /* variables locals */
 int i = 0;
 srand(getpid());    /* inicialitza numeros aleatoris */
 char object_str[100];
 char idSM_str[10];
 char a1[20], a2[20], a3[20], a4[20], a5[20];
 void *p_win;
 int id_win = 0;

 if ((n_args != 2) && (n_args !=3)){
   fprintf(stderr,"Comanda: cocos0 fit_param [retard] [numero
fantasmas]\n");
   exit(1);
 }

 /*
 INICIAR MEMORIA COMPARTIDA
 */
 id_sharedMemory = ini_mem(sizeof(int));         /* crear zona mem.
compartida */
 p_sharedMemory = map_mem(id_sharedMemory);  /* obtenir adres. de mem.
compartida */
 *p_sharedMemory = condicion;

 /*
 CARGA DE PARAMETROS
 */
 carrega_parametres(ll_args[1]);
 printf("El numero total de elementos[indice] sera de %d\n -Fantasmas:
%d\n -Comecocos: 1\n", totalElem, totalElem-1);

 if (n_args == 3) retard = atoi(ll_args[2]);
 else retard = 100;
 rc = win_ini(&n_fil1,&n_col,'+',INVERS);  /* intenta crear taulell */
 if (rc >= 0)    /* si aconsegueix accedir a l'entorn CURSES */
 {

   /*
   INICIAR MEMORIA COMPARTIDA DE CAMPO
   */
   id_win = ini_mem(rc); /* crear zona mem. compartida */
```

```
   p_win = map_mem(id_win);  /* obtenir adres. de mem. compartida */

   id_sem = ini_sem(1); /*Crear semaforo*/

   win_set(p_win,n_fil1,n_col);    /* crea acces a finestra oberta */

   inicialitza_joc();
   win_update();
   /*
   JUSTO CUANDO SE INICIA EL JUEGO Y LOS elementos[indice] ESTAN SOBRE
EL TABLERO, SE INICIAN LOS THREADS PARA QUE COMIENCEN A EJECUTARSE
   */

   sprintf(idSM_str, "%i", id_sharedMemory);
   i=1;

   sprintf(a1,"%i",id_win);
   sprintf(a2,"%i",n_fil1);
   sprintf(a3,"%i",n_col);
   sprintf(a4,"%i",id_sem);
   //printf("%d, %d, %d, %f %d", elementos[i].f, elementos[i].c,
elementos[i].d, elementos[i].r, elementos[i].a);
   pthread_create(&thread ,NULL,mou_menjacocos,(void *) NULL);
   while(i<totalElem){
     tpid[i] = fork();   /* crea un nou proces */
     if (tpid[i] == (pid_t) 0)   /* branca del fill */
     {
       sprintf(object_str, "%d,%d,%d,%.2f,%c", elementos[i].f,
elementos[i].c, elementos[i].d, elementos[i].r, elementos[i].a);
       sprintf(a5,"%i",i);
       /*
       PARAMETROS A ENVIAR
       PARAM0 --> Nombre del programa
       PARAM1 --> Objecto fantasma
       PARAM2 --> Retardo
       PARAM3 --> Id de la memoria comaprtida
       PARAM4 --> Id de campo
       PARAM5 --> Filas
       PARAM6 --> Columnas
       PARAM7 --> Semaforo
       */
       execlp("./Fantasmas4", "Fantasmas4", object_str, ll_args[2],
idSM_str, a1, a2, a3, a4, a5, (char *)0);
```

```
       fprintf(stderr,"error: no puc executar el process fill
\'mp_car\'\n");
       exit(0);
     }
   i++;
  }
 /*
  UNA VEZ CREADOS SE REALIZA EL JOIN PARA QUE ESPERAR A QUE ACABEN
  */
 pthread_join(thread, (void *) NULL);
 i=1;
 while(i<totalElem)
 {
   waitpid(tpid[i],NULL,0);  /* espera finalitzacio d'un fill */
   printf("Espero al fill\n");
   i++;
 }
  win_fi();

 if (*p_sharedMemory == 0){
   printf("EL JUGADOR GANO\n");
 }else if (*p_sharedMemory == 1){
   printf("VAYA LOS FANTASMAS GANARON\n");
 }else if (*p_sharedMemory == 2){
   printf("EL JUGADOR DETUVO EL JUEGO\n");
 }

}else{
 switch (rc)
 {
   case -1: fprintf(stderr,"camp de joc ja creat!\n");
     break;
   case -2: fprintf(stderr,"no s'ha pogut inicialitzar l'entorn de
curses!\n");
     break;
   case -3: fprintf(stderr,"les mides del camp demanades son massa
grans!\n");
     break;
   case -4: fprintf(stderr,"no s'ha pogut crear la finestra!\n");
     break;
 }
 exit(6);
}
```

```
 elim_mem(id_sharedMemory);
 elim_mem(id_win);
 elim_sem(id_sem);
 return(0);
}
```

· Fantasmas4.c

```
#include <stdio.h>    /* incloure definicions de funcions estandard */
#include <stdlib.h>   /* per exit() */
#include <unistd.h>   /* per getpid() */
#include "winsuport2.h"   /* incloure definicions de funcions propies
*/
#include <stdbool.h>
#include <stdint.h>   /* intptr_t per màquines de 64 bits */
#include "memoria.h"
#include "semafor.h"
/*
Estructura
*/

       /* definir estructures d'informacio */
typedef struct {    /* per un objecte (menjacocos o fantasma) */
 int f;         /* posicio actual: fila */
 int c;         /* posicio actual: columna */
 int d;         /* direccio actual: [0..3] */
   float r;             /* per indicar un retard relati */
 char a;        /* caracter anterior en pos. actual */
} objecte;

int df[] = {-1, 0, 1, 0}; /* moviments de les 4 direccions possibles */
int dc[] = {0, -1, 0, 1}; /* dalt, esquerra, baix, dreta */



/* funcio per moure un fantasma una posicio; retorna 1 si el fantasma
*/
/* captura al menjacocos, 0 altrament         */
int main(int n_args, char *ll_args[])
{
 objecte elementos;
 objecte seg;
 int k, vk, nd, vd[3], id_win;
 int *p_sharedMemory, id_sharedMemory;
 void *p_win;
 int retard = atoi(ll_args[2]);
 int id_sem = 0;
 /*
```

```
 SE CONECTA A LA MEMORIA COMPARTIDA Y COMPRUEBA SI ES CORRECTA
 */
 id_sharedMemory = atoi(ll_args[3]);

 p_sharedMemory = map_mem(id_sharedMemory);  /* obtenir adres. de mem.
compartida */
 if (p_sharedMemory == (int*) -1)
 {   fprintf(stderr,"proces (%d): error en identificador de
memoria\n",(int)getpid());
 exit(0);
 }
  //mapa
 int n_fill = atoi(ll_args[5]);
 int n_col = atoi(ll_args[6]);
 //
 id_win = atoi(ll_args[4]);
 p_win = map_mem(id_win);  /* obtenir adres. de mem. compartida */

 id_sem = atoi(ll_args[7]);    /* obtenir identificador de semafor */

 if (p_win  == (int*) -1)
 {   fprintf(stderr,"proces (%d): error en identificador de
memoria\n",(int)getpid());
 exit(0);
 }

 win_set(p_win,n_fill,n_col);    /* crea acces a finestra oberta */
  /*
 RECUPERAMOS EL OBJETO
 */
 sscanf(ll_args[1], "%d,%d,%d,%f,%c", &elementos.f, &elementos.c,
&elementos.d, &elementos.r, &elementos.a);
 fflush(stdout);
 int numero = atoi(ll_args[8]);
 while (*p_sharedMemory == -1)
 {
   nd = 0;
   for (k=-1; k<=1; k++)   /* provar direccio actual i dir. veines */
   {
     waitS(id_sem);
     vk = (elementos.d + k) % 4;   /* direccio veina */
     if (vk < 0) vk += 4;    /* corregeix negatius */
```

```
     seg.f = elementos.f + df[vk]; /* calcular posicio en la nova
dir.*/
     seg.c = elementos.c + dc[vk];
     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
     signalS(id_sem);
     if ((seg.a==' ') || (seg.a=='.') || (seg.a=='0'))
     {
       vd[nd] = vk;       /* memoritza com a direccio possible */
       nd++;
     }
   }
   if (nd == 0)
   {
     waitS(id_sem);
     elementos.d = (elementos.d + 2) % 4;    /* canvia totalment de
sentit */
     signalS(id_sem);
   }
   else
   {
     waitS(id_sem);
     if (nd == 1)       /* si nomes pot en una direccio */
     {
       elementos.d = vd[0];       /* li assigna aquesta */
     }
     else         /* altrament */
       elementos.d = vd[rand() % nd];    /* segueix una dir. aleatoria
*/
     signalS(id_sem);
     seg.f = elementos.f + df[elementos.d];  /* calcular seguent
posicio final */
     seg.c = elementos.c + dc[elementos.d];
     waitS(id_sem);
     seg.a = win_quincar(seg.f,seg.c); /* calcular caracter seguent
posicio */
     win_escricar(elementos.f,elementos.c,elementos.a,NO_INV); /*
esborra posicio anterior */
     elementos.f = seg.f; elementos.c = seg.c; elementos.a = seg.a;  /*
actualitza posicio */
     win_escricar(elementos.f,elementos.c, (char)('0'+numero),NO_INV);
/* redibuixa fantasma */
```

```
     if (elementos.a == '0') *p_sharedMemory = 1;     /* ha capturat
menjacocos */
     signalS(id_sem);
   }
   win_retard(retard);
 }
 elim_mem(id_sharedMemory);
 elim_mem(id_win);
 exit(0);
}
```

3. Juego de pruebas

A continuación se realizarán los distintos juegos de pruebas.
Debido a que en este caso el juego de pruebas se realizaría mediante los distintos juegos predefinidos y preparados por los docentes, el juego de pruebas a realizar sería ir probando estos juegos preparados y ver el funcionamiento.

A continuación se irán especificando en una tabla:
· ***Cocos0.c*** **(Cocos1 + Cocos2) (genera basura o bien)**
Cocos1:

ENTRADAS	SALIDA ESPERADA	SALIDA
joc11.txt	Genera basura	Genera basura
joc21.txt	Genera basura	Genera basura
joc22.txt	Genera basura	Genera basura
joc31.txt	Genera basura	Genera basura
joc34.txt	Genera basura	Genera basura

Cocos2:

ENTRADAS	SALIDA ESPERADA	SALIDA
joc11.txt	Funciona bien	Funciona bien
joc21.txt	Funciona bien	Funciona bien
joc22.txt	Funciona bien	Funciona bien
joc31.txt	Funciona bien	Funciona bien
joc34.txt	Funciona bien	Funciona bien

· ***Cocos3.c***

ENTRADAS nombre + retardo	SALIDA ESPERADA	SALIDA
joc11.txt 100	Funciona bien	Funciona bien
joc21.txt 100	Funciona bien	Funciona bien
joc22.txt 100	Funciona bien	Funciona bien
joc31.txt 100	Funciona bien	Funciona bien
joc34.txt 100	Funciona bien	Funciona bien

· ***Cocos4.c***

ENTRADAS nombre + retardo	SALIDA ESPERADA	SALIDA
joc11.txt 100	Funciona bien	Funciona bien
joc21.txt 100	Funciona bien	Funciona bien
joc22.txt 100	Funciona bien	Funciona bien
joc31.txt 100	Funciona bien	Funciona bien
joc34.txt 100	Funciona bien	Funciona bien

Funciona bien significa que se cumplen los casos de:

- Jugador detiene el juego
- Gana el jugador
- Ganan los fantasmas

Sino es cuando genera basura o bien se bloquea.

4. Conclusiones

Se trata de una práctica muy interesante y útil, ya que se aprende el uso de los threads y procesos que consiguen no solo poder ejecutar programas o partes de éste de manera concurrente, sino optimizar el tiempo de ejecución.

A la hora de crear, tantos los threads como los procesos, nos hemos encontrado con problemas bastantes difíciles de solucionar como el segment fault, el cuál apareció debido a entrar en zonas de memoria restringidas.

También se ha podido apreciar las ventajas del uso de la memoria compartida para trabajar entre procesos, la cual es muy útil no solo para compartir recursos sino para tener un control de la ejecución de los procesos.

Finalmente en la sincronización se ha podido aprender cómo funciona y los efectos positivos y negativos que conlleva, tanto no tener sincronización como tener demasiada sincronización.