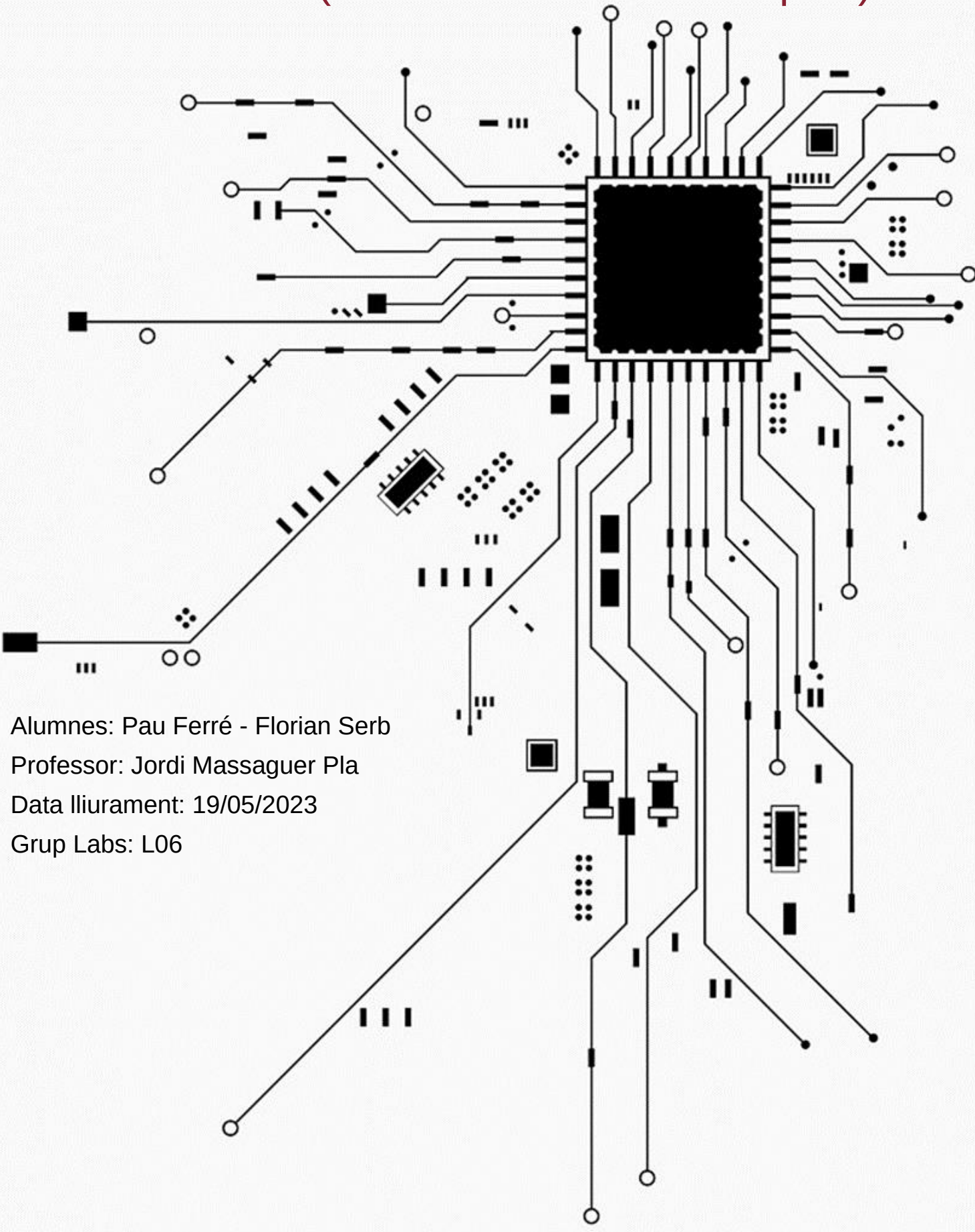
PRÀCTICA 2.1 (Threads+SeccionsCrítiques)



Alumnes: Pau Ferré - Florian Serb

Professor: Jordi Massaguer Pla

Data lliurament: 19/05/2023

Grup Labs: L06

Índex

Introducció

En aquesta primera part del treball, en centrem en com funcionen els threads i com podem crear diferents multithreads dins d'un mateix.

La primera part del treball consisteix a agafar un codi sobre el famós joc menjacocos i aconseguir que el menjacocos i el fantasma es puguin executar amb funcions de fils diferents sense tenir cap mena de sincronisme (el qual causarà errors).

La segona part del treball consisteix a obtenir que els múltiples threads, puguin tenir un sincronisme correcte tenint fils independents i que el joc es pugui executar de manera correcta i sense cap error.

Explicarem les especificacions de la finalitat de la fase, com hem dissenyat les diverses fases del treball, el codi final i un petit joc de proves per a veure quin seria el resultat obtingut.

1. Fase 1: Creació de threads

1.1 Especificació

En aquesta fase, hem d'aconseguir que les funcions de moviment del menjacocos i els fantasmes puguin funcionar com a fils d'execució independents. Per a fer-ho hem de modificar les funcions que ja estan creades dins de cocos0.

Cada funció implementa el seu propi bucle per generar el moviment dels objectes. Per a finalitzar l'execució dels fils de moviment, es fan variables globals que indiquen alguna condició de final de joc.

Amb el fitxer de configuració obtindrem un nou format amb el qual podem controlar els múltiples fantasmes. Obtindrem la posició i direcció de moviment inicial dels fantasmes. Amb el nombre de files del fitxer determinarem quants fantasmes es volen crear.

Finalment, també mostrem al final del joc quant de temps ha tardat l'usuari a guanyar o perdre la partida i també afegim que si l'usuari prem la tecla "Return", llavors s'acaba el joc.

1.2 Disseny

Les capçaleres de les funcions canvien per a poder-les fer independents i queden així:

- void * mou_fantasma(void * index)
- void * mou_menjacocos(void * null)

Per a que cada fil de moviment sigui independent, primer de tot posarem les variables "fi1" i "fi2" com a globals perquè si en cas que qualsevol dels motius es compleixi la condició de final de joc, acabi i no continuï executant els fils.

També hem posat la variable "p" com a global, ja que amb aquesta variable afegim tant als fantasmes com al menjacocos en la seva posició inicial.

Un cop ja tenim les variables locals, agafem el bucle principal del programa i el dividim en dos per a afegir-los dins de les funcions i que cada funció tingui el seu fil independent. Per a poder-los cridar de forma independent, dins del mai afegim la creació i crida de dos threads, un per al fantasma i l'altre per al menjacocos.

Com que no afegim cap mena de sincronisme entre els fils, apareixen errors ocasionals a causa de l'execució concurrent i descontrolada d'aquests.

Finalment, afegim un condicional que si l'usuari prem la tecla "Return" la variable "fi1" tingui el valor -1 i també acabi el joc. També hem afegit una variable que comenci a contar des de que comença el codi fins que acaba per a controlar quant de temps està l'usuari executant el programa i finalment el mostra per pantalla.

1.3 Implementació

```
#define _REENTRANT
#include <stdio.h>  /* incloure definicions de funcions estandard */
#include <stdlib.h> /* per exit() */
#include <stdint.h>
#include <string.h>
#include <unistd.h>    /* per getpid() */
#include "winsuport.h" /* incloure definicions de funcions propies */
#include <pthread.h>
#include <time.h>
#define MIN_FIL 7 /* definir limits de variables globals */
#define MAX_FIL 25
#define MIN_COL 10
#define MAX_COL 80
#define MAX_THREADS 10 /*Declarem la constant maxima de threads*/
#define MAX_FAN (MAX_THREADS - 1)
/* 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;
pthread_t tid[MAX_THREADS]; /*taula d'identificadors dels threads*/
/* variables globals */
```

```c
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[MAX_FAN]; /* 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 */
int fi1, fi2, p;
int num_fan;
/* 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)
{
    int i = 1;
    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);
}
if (!feof(fit))
    fscanf(fit, "%d %d %d %f\n", &mc.f, &mc.c, &mc.d, &mc.r);
else
{
    fprintf(stderr, "Falten parametres al fitxer \'%s\'\n", nom_fit);
    fclose(fit);
    exit(2);
}
if ((mc.f < 1) || (mc.f > n_fil1 - 3) ||
    (mc.c < 1) || (mc.c > n_col - 2) ||
    (mc.d < 0) || (mc.d > 3))
{
    fprintf(stderr, "Error: parametres menjacocos incorrectes:\n");
```

```
    fprintf(stderr, "\t1 =< mc.f (%d) =< n_fil1-3 (%d)\n", mc.f, (n_fil1 - 3));
    fprintf(stderr, "\t1 =< mc.c (%d) =< n_col-2 (%d)\n", mc.c, (n_col - 2));
    fprintf(stderr, "\t0 =< mc.d (%d) =< 3\n", mc.d);
    fclose(fit);
    exit(4);
}
fscanf(fit, "%d %d %d %f\n", &f1[0].f, &f1[0].c, &f1[0].d, &f1[0].r);
while (!feof(fit))
{
    fscanf(fit, "%d %d %d %f\n", &f1[i].f, &f1[i].c, &f1[i].d, &f1[i].r);
    i++;
}
num_fan = i;
i = 1;
if ((f1[i].f < 1) || (f1[i].f > n_fil1 - 3) ||
    (f1[i].c < 1) || (f1[i].c > n_col - 2) ||
    (f1[i].d < 0) || (f1[i].d > 3))
{
    fprintf(stderr, "Error: parametres fantasma 1 incorrectes:\n");
    fprintf(stderr, "\t1 =< f1.f (%d) =< n_fil1-3 (%d)\n", f1[i].f, (n_fil1 - 3));
    fprintf(stderr, "\t1 =< f1.c (%d) =< n_col-2 (%d)\n", f1[i].c, (n_col - 2));
    fprintf(stderr, "\t0 =< f1.d (%d) =< 3\n", f1[i].d);
    fclose(fit);
    exit(5);
}
fclose(fit); /* fitxer carregat: tot OK! */
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, w;
    char strin[12];
    r = win_carregatauler(tauler, n_fil1 - 1, n_col, c_req);
    if (r == 0)
    {
        mc.a = win_quincar(mc.f, mc.c);
        if (mc.a == c_req)
            r = -6; /* error: menjacocos sobre pared */
        else
        {
            for (w = 0; w < num_fan; w++)
            {
                f1[w].a = win_quincar(f1[w].f, f1[w].c);
                if (f1[w].a == c_req)
                    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(mc.f, mc.c, '0', NO_INV);
                win_escricar(f1[w].f, f1[w].c, '1', NO_INV);

                if (mc.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 ret;
    int k, vk, nd, vd[3];
    ret = 0;
    do
    {
        win_retard(retard);
        if ((p % 2) == 0)
```

```
{
    for (int w = 0; w < num_fan; w++)
    {
        nd = 0;
        for (k = -1; k <= 1; k++) /* provar direccio actual i dir. veines */
        {
            vk = (f1[w].d + k) % 4; /* direccio veina */
            if (vk < 0)
            {
                vk += 4; /* corregeix negatius */
            }
            seg.f = f1[w].f + df[vk]; /* calcular posicio en la nova dir.*/
            seg.c = f1[w].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) /* si no pot continuar, */
        {
            f1[w].d = (f1[w].d + 2) % 4; /* canvia totalment de sentit */
        }
        else
        {
            if (nd == 1) /* si nomes pot en una direccio */
            {
```

```
                    f1[w].d = vd[0]; /* li assigna aquesta */
                }
                else /* altrament */
                {
                    f1[w].d = vd[rand() % nd]; /* segueix una dir. aleatoria */
                }
                seg.f = f1[w].f + df[f1[w].d]; /* calcular seguent posicio final */
                seg.c = f1[w].c + dc[f1[w].d];
                seg.a = win_quincar(seg.f, seg.c);              /* calcular caracter seguent posicio
*/
                win_escricar(f1[w].f, f1[w].c, f1[w].a, NO_INV); /* esborra posicio anterior */
                f1[w].f = seg.f;
                f1[w].c = seg.c;
                f1[w].a = seg.a;                        /* actualitza posicio */
                win_escricar(f1[w].f, f1[w].c, '1', NO_INV); /* redibuixa fantasma */
                if (f1[w].a == '0')
                {
                    fi1 = 1; /* ha capturat menjacocos */
                }
            }
        }
    } while (!fi1 && !fi2);
}
/* 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 *null)
```

```
{
   char strin[12];
   objecte seg;
   int tec, ret;
   ret = 0;
   do
   {
      win_retard(retard);
      tec = win_gettec();
      if (tec != 0)
      {
         switch (tec) /* modificar direccio menjacocos segons tecla */
         {
         case TEC_AMUNT:
            mc.d = 0;
            break;
         case TEC_ESQUER:
            mc.d = 1;
            break;
         case TEC_AVALL:
            mc.d = 2;
            break;
         case TEC_DRETA:
            mc.d = 3;
            break;
         case TEC_RETURN:
            ret = -1;
            break;
```

```
            }
        }
        fi1 = ret;
        seg.f = mc.f + df[mc.d]; /* calcular seguent posicio */
        seg.c = mc.c + dc[mc.d];
        seg.a = win_quincar(seg.f, seg.c); /* calcular caracter seguent posicio */
        if ((seg.a == ' ') || (seg.a == '.'))
        {
            win_escricar(mc.f, mc.c, ' ', NO_INV); /* esborra posicio anterior */
            mc.f = seg.f;
            mc.c = seg.c;                          /* actualitza posicio */
            win_escricar(mc.f, mc.c, '0', NO_INV); /* redibuixa menjacocos */
            if (seg.a == '.')
            {
                cocos--;
                sprintf(strin, "Cocos: %d", cocos);
                win_escristr(strin);
                if (cocos == 0)
                {
                    fi2 = 1;
                }
            }
        }
        p++;
    } while (!fi1 && !fi2);
}
/* programa principal                  */
int main(int n_args, const char *ll_args[])
```

```
{
    int temps_total = 0;
    time_t begin = time(NULL);
    int t, t_total, rc, n; /* variables locals */
    srand(getpid()); /* inicialitza numeros aleatoris */
    if ((n_args != 2) && (n_args != 3))
    {
        fprintf(stderr, "Comanda: cocos0 fit_param [retard]\n");
        exit(1);
    }
    carrega_parametres(ll_args[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();
        p = 0;
        fi1 = 0;
        fi2 = 0;
        win_retard(retard);
        // Creem els threads necessaris, un per al menjacocos i l'altre per al fantasma
        pthread_create(&tid[0], NULL, mou_menjacocos, (void *)(intptr_t)0);
        pthread_create(&tid[0], NULL, mou_fantasma, (void *)(intptr_t)0);
        for (int i = 0; i <= num_fan; i++)
        {
```

```
        pthread_join(tid[i], (void **)&t);
        t_total += t;
    }
    win_fi();
    if (fi1 == -1)
        printf("S'ha aturat el joc amb tecla RETURN!\n");
    else
    {
        if (fi1)
        {
            printf("Ha guanyat l'ordinador!\n");
            printf("Han faltat %d cocos ha menjar\n", cocos);
        }
        else
            printf("Ha guanyat l'usuari!\n");
    }
    win_retard(retard);
}
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);
    }
    // Calculem el temps que tarda en finalitzar el joc
    time_t end = time(NULL);
    temps_total += (int)(end - begin);
    printf("El joc ha tardat %d segons\n", temps_total);
    return (0);
}
```

1.4 Joc de proves

Descripció	Característiques	Resultat
L'usuari executa cocos1.c sense valors ni fitxers	El joc hauria de mostrar un missatge d'error a mesura que falten valors	Es mostra un missatge advertint a l'usuari que falta informació per a poder executar el programa
L'usuari executa cocos1.c amb els valors i fitxers correctes.	El joc comença a executar-se correctament, però al no tenir sincronisme entre fils, el joc no funciona correctament.	L'usuari executa el programa, es mostra la taula, el menjacocos i el fantasma comencen a la seva posició inicial, però comencen a sortir errors.
L'usuari executa el programa correctament apreta la tecla "Return"	En prémer la tecla return, la variable fi1 es posa amb el valor -1 i finalitza el joc.	El joc finalitza.
L'usuari executa el programa amb un fitxer de configuració amb paràmetres incorrectes.	El programa veurà quins paràmetres estan erronis i li mostrarà a l'usuari quin tipus d'error ha tingut	L'usuari executa el programa i se li mostra quin tipus d'error té dins del fitxer de configuració
L'usuari executa correctament el programa i finalitza el joc.	El programa al final del joc mostrarà quants cocos quedaven, quant de temps ha tardat i qui és el guanyador	El programa al final del joc mostrarà quants cocos quedaven, quant de temps ha tardat i qui és el guanyador

2. Fase 2: Sincronització de threads

2.1 Especificació

En aquesta fase cal establir seccions crítiques que evitin els problemes de concurrència de la fase anterior.

Les seccions crítiques han de servir per impedir l'accés concurrent a determinats recursos per part dels diferents fils d'execució. En el nostre cas, cal establir seccions crítiques per accedir a l'entorn de joc, per usar les rutines de curses, i per accedir alguna variable global que no permeti els accessos concurrents.

2.2 Disseny

Per a cocos2.c hem implementat en les funcions mou_menjacocos i mou_fantasma els pthread_mutex_lock i pthread_mutex_unlock de tal manera el menjacocos i el fantasma se sincronitzen correctament de tal manera no ens surt per pantalla brossa.

Els lock i unlock els hem implementat a totes les variables globals que estan accedint a una posició per escriure un caràcter, a tots els que tenen win_quincar i win_escricar d'aquesta manera si el menjacocos accedeix a una posició per a fer un moviment el fantasma s'espera fins que el menjacocos fa el moviment i d'aquesta manera evitem que estiguin apareixent a la mateixa posició i també gestionem que no mostri brossa.

En mou_menjacocos també hem afegit un lock/unlock al win_gettec() ja que aquest espera l'entrada de la tecla de l'usuari i així podem veure els moviments de l'usuari (cap a dalt/baix/dreta/esquerra o si vol sortir del joc return).

Com ja hem explicat en cocos1, en cocos2 també hem implementat que ens mostri el temps que ha durat el jugador jugant (hem utilitzat aquesta pàgina de referència per a calcular el temps que ha estat jugant) i quants cocos li ha faltat per menjar, també hem implementat la funció de si l'usuari ja no vol continuar jugant, aquest pot acabar el joc en qualsevol moment clicant la tecla return → que en aquest cas és l'enter.

2.3 Implementació

```
#define _REENTRANT
#include <stdio.h>  /* incloure definicions de funcions estandard */
#include <stdlib.h> /* per exit() */
#include <stdint.h>
#include <string.h>
#include <unistd.h>    /* per getpid() */
#include "winsuport.h" /* incloure definicions de funcions propies */
```

```
#include <pthread.h>   //per a la implementacio de threads
#include <time.h>      //hem utilitzat aquesta per a saber els segons que passen mentres funciona el joc
#define MIN_FIL 7 /* definir limits de variables globals */
#define MAX_FIL 25
#define MIN_COL 10
#define MAX_COL 80
#define MAX_THREADS 10 /*Declarem la constant maxima de threads*/
#define MAX_FAN (MAX_THREADS - 1)
/* 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;
pthread_t tid[MAX_THREADS];                    /*taula d'identificadors dels threads*/
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // creem el mutex, per a poder utilitzar-lo despres per a la sincronitzacio de threads
/* 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[MAX_FAN]; /* 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 */
int fi1, fi2, p;
int num_fan;
/* 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)
{
  int i = 1;
  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);
}
if (!feof(fit))
    fscanf(fit, "%d %d %d %f\n", &mc.f, &mc.c, &mc.d, &mc.r);
else
{
    fprintf(stderr, "Falten parametres al fitxer \'%s\'\n", nom_fit);
    fclose(fit);
    exit(2);
}
if ((mc.f < 1) || (mc.f > n_fil1 - 3) ||
    (mc.c < 1) || (mc.c > n_col - 2) ||
    (mc.d < 0) || (mc.d > 3))
{
    fprintf(stderr, "Error: parametres menjacocos incorrectes:\n");
    fprintf(stderr, "\t1 =< mc.f (%d) =< n_fil1-3 (%d)\n", mc.f, (n_fil1 - 3));
    fprintf(stderr, "\t1 =< mc.c (%d) =< n_col-2 (%d)\n", mc.c, (n_col - 2));
    fprintf(stderr, "\t0 =< mc.d (%d) =< 3\n", mc.d);
    fclose(fit);
    exit(4);
}
```

```
  fscanf(fit, "%d %d %d %f\n", &f1[0].f, &f1[0].c, &f1[0].d, &f1[0].r);
  while (!feof(fit))
  {
     fscanf(fit, "%d %d %d %f\n", &f1[i].f, &f1[i].c, &f1[i].d, &f1[i].r);
     i++;
  }
  num_fan = i;
  i = 1;
  if ((f1[i].f < 1) || (f1[i].f > n_fil1 - 3) ||
     (f1[i].c < 1) || (f1[i].c > n_col - 2) ||
     (f1[i].d < 0) || (f1[i].d > 3))
  {
     fprintf(stderr, "Error: parametres fantasma 1 incorrectes:\n");
     fprintf(stderr, "\t1 =< f1.f (%d) =< n_fil1-3 (%d)\n", f1[i].f, (n_fil1 - 3));
     fprintf(stderr, "\t1 =< f1.c (%d) =< n_col-2 (%d)\n", f1[i].c, (n_col - 2));
     fprintf(stderr, "\t0 =< f1.d (%d) =< 3\n", f1[i].d);
     fclose(fit);
     exit(5);
  }
  fclose(fit); /* fitxer carregat: tot OK! */
  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, w;
char strin[12];
r = win_carregatauler(tauler, n_fil1 - 1, n_col, c_req);
if (r == 0)
{
    mc.a = win_quincar(mc.f, mc.c);
    if (mc.a == c_req)
        r = -6; /* error: menjacocos sobre pared */
    else
    {
        for (w = 0; w < num_fan; w++)
        {
            f1[w].a = win_quincar(f1[w].f, f1[w].c);
            if (f1[w].a == c_req)
                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(mc.f, mc.c, '0', NO_INV);
                win_escricar(f1[w].f, f1[w].c, '1', NO_INV);
                if (mc.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 ret;
    int k, vk, nd, vd[3];
    ret = 0;
    do
    {
        win_retard(retard);
        if ((p % 2) == 0)
        {
            for (int w = 0; w < num_fan; w++)
            {
                nd = 0;
                for (k = -1; k <= 1; k++) /* provar direccio actual i dir. veines */
                {
                    vk = (f1[w].d + k) % 4; /* direccio veina */
                    if (vk < 0)
```

```
            {
                vk += 4; /* corregeix negatius */
            }
            seg.f = f1[w].f + df[vk]; /* calcular posicio en la nova dir.*/
            seg.c = f1[w].c + dc[vk];
            pthread_mutex_lock(&mutex);//bloquejem per a que el fantasma mire el seguent caracter en la posicio que es vol moure
            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++;
            }
            pthread_mutex_unlock(&mutex);//desbloquejem despres del moviment correcte del fantasma per a donar pas al menjacocos
        }
        if (nd == 0) /* si no pot continuar, */
        {
            f1[w].d = (f1[w].d + 2) % 4; /* canvia totalment de sentit */
        }
        else
        {
            if (nd == 1) /* si nomes pot en una direccio */
            {
                f1[w].d = vd[0]; /* li assigna aquesta */
            }
            else /* altrament */
            {
```

```
                f1[w].d = vd[rand() % nd]; /* segueix una dir. aleatoria */
            }
            seg.f = f1[w].f + df[f1[w].d]; /* calcular seguent posicio final */
            seg.c = f1[w].c + dc[f1[w].d];
            pthread_mutex_lock(&mutex);//bloquejem per a que puegue fer el seguent moviment a una nova posicio
            seg.a = win_quincar(seg.f, seg.c);          /* calcular caracter seguent posicio */
            win_escricar(f1[w].f, f1[w].c, f1[w].a, NO_INV); /* esborra posicio anterior */
            pthread_mutex_unlock(&mutex);//despres de moures el fantasma es moura el menjacocos
            f1[w].f = seg.f;
            f1[w].c = seg.c;
            f1[w].a = seg.a;
            pthread_mutex_lock(&mutex);                 /* actualitza posicio, i despres deixa actualitzar el fantasma */
            win_escricar(f1[w].f, f1[w].c, '1', NO_INV); /* redibuixa fantasma */
            pthread_mutex_unlock(&mutex);               // despres d'actualitzar la posicio del fantasma s'actualitza la posicio del menjacocos
            if (f1[w].a == '0')
            {
                fi1 = 1; /* ha capturat menjacocos, si el fantasma es troba en lo menjacocos acaba el joc */
            }
        }
    }
  } while (!fi1 && !fi2);
}
/* 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 *null)
{
  char strin[12];
  objecte seg;
  int tec, ret;
  ret = 0;
  do
  {
    win_retard(retard);
    pthread_mutex_lock(&mutex);//bloquejem per a que l'usuari trie una moviment per al menjacocos
    tec = win_gettec();
    pthread_mutex_unlock(&mutex);//desbloquejem despres de cada moviment si decidieix sortir acaba el joc clicant al enter
    if (tec != 0)
    {
      switch (tec) /* modificar direccio menjacocos segons tecla */
      {
      case TEC_AMUNT:
        mc.d = 0;
        break;
      case TEC_ESQUER:
        mc.d = 1;
        break;
      case TEC_AVALL:
        mc.d = 2;
```

```
            break;
        case TEC_DRETA:
            mc.d = 3;
            break;
        case TEC_RETURN:
            ret = -1;
            break;
        }
    }
    fi1 = ret;
    seg.f = mc.f + df[mc.d]; /* calcular seguent posicio */
    seg.c = mc.c + dc[mc.d];
    seg.a = win_quincar(seg.f, seg.c); /* calcular caracter seguent posicio,
    en aquest no fa falta sincronitzar ja que sols accedeix menjacocos a esta variable*/
    if ((seg.a == ' ') || (seg.a == '.'))
    {
        pthread_mutex_lock(&mutex);//aqui bloqueijem perque esborra una posicio anterior i d'aquesta manera evitem problemes en lo fantasma
        win_escricar(mc.f, mc.c, ' ', NO_INV); /* esborra posicio anterior */
        pthread_mutex_unlock(&mutex);//desbloquejem per a que el fantasva pugue accedir a la nova posicio
        mc.f = seg.f;
        mc.c = seg.c;
        pthread_mutex_lock(&mutex);            /* actualitza posicio, el menjacocos accedeix a una nova posicio
        i d'aquesta manera evitem que els dos accedeixin a la mateixa posicio */
        win_escricar(mc.f, mc.c, '0', NO_INV); /* redibuixa menjacocos */
        pthread_mutex_unlock(&mutex);//desbloquejem per a que pugue el fantasma accedir a una posicio
```

```
        if (seg.a == '.')
        {
            cocos--;
            sprintf(strin, "Cocos: %d", cocos);
            pthread_mutex_lock(&mutex);//bloquejem per a que mentres estigue menjan cocos segueixi el joc i cuan se troba en lo fantasma aquest acaba
            win_escristr(strin);//decrementem els cocos si el menjacocos es menja un .
            pthread_mutex_unlock(&mutex);//desbloquejem per a que el fantasma puegues moures
            if (cocos == 0)
            {
                fi2 = 1;
            }
        }
    }
    p++;
  } while (!fi1 && !fi2);
}
/* programa principal                    */
int main(int n_args, const char *ll_args[])
{
  int temps_total = 0;
  // time_t la utilitzem per a controlar el temps que passa mentres funciona el joc
  time_t begin = time(NULL);
  int t, t_total, rc, n; /* variables locals */
  // Cridem al mutex per a que s'execute cada vegada que inicialitzem el programa
  pthread_mutex_init(&mutex, NULL);
  srand(getpid()); /* inicialitza numeros aleatoris */
```

```
if ((n_args != 2) && (n_args != 3))
{
    fprintf(stderr, "Comanda: cocos0 fit_param [retard]\n");
    exit(1);
}
carrega_parametres(ll_args[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();
    // p = 0;
    fi1 = 0;
    fi2 = 0;
    win_retard(retard);
    // creem el threads i tambe inicialitzem els mutex cada vegada que es compleix la condicio
    pthread_create(&tid[0], NULL, mou_menjacocos, (void *)(intptr_t)0);
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&tid[0], NULL, mou_fantasma, (void *)(intptr_t)0);
    for (int i = 0; i <= num_fan; i++)
    {
        pthread_join(tid[i], (void **)&t);
        t_total += t;
    }
```

```
    win_fi();
    if (fi1 == -1)
        printf("S'ha aturat el joc amb tecla RETURN!\n");
    else
    {
        if (fi1)
        {
            printf("Ha guanyat l'ordinador!\n");
            printf("Han faltat %d cocos ha menjar\n", cocos);
        }
        else
            printf("Ha guanyat l'usuari!\n");
    }
    win_retard(retard);
}
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);
    }
    // Matem els mutex al finalitzar el programa
    pthread_mutex_destroy(&mutex);
    // controlem el tems que passa desde que sinicia el joc i fins que acaba i la mostrem per pantalla
    time_t end = time(NULL);
    temps_total += (int)(end - begin);
    printf("El joc ha tardat %d segons\n", temps_total);
    return (0);
}
```

2.4 Joc de proves

Descripció	Característiques	Resultat
L'usuari executa el joc: ./cocos2 joc34.txt 250	El joc s'hauria d'executar correctament, mostrant primerament les instruccions de joc, i deixar jugar al jugador, fins que el menjacocos fa finalitzar el joc o l'usuari decideix acabar amb la tecla return.	El jugador comença el joc, se li mostra les instruccions i també l'opció d'acabar el joc amb la tecla return. Un cop començat el joc i el menjacocos se'l menja aquest acaba correctament. Si decideix acabar abans amb la tecla return, aquest també acaba correctament.
L'usuari: vull saber quants menjacocos m'han quedat per menjar	El joc hauria de mostrar per pantalla en finalitzar el joc, els menjacocos restants per a haver guanyat el joc.	El joc en finalitzar aquest mostra els cocos restants per haver pogut guanyar el joc.
L'usuari: vull saber quant de temps en segons he estat jugant.	El joc hauria de mostrar per pantalla en finalitzar el joc, quants segons ha estat jugant el jugador.	El joc mostra per pantalla en finalitzar el joc quants segons el jugador ha estat jugant.
L'usuari: vull saber qui ha guanyat el joc.	El joc hauria de mostrar en finalitzar el joc qui és el guanyador, l'usuari o l'ordinador?	El joc mostra per pantalla qui és el jugador, si el menjacocos se'l menja en aquest cas guanya l'ordinador, si en cas que el jugador menja tots els cocos en aquest cas guanya el jugador.

Conclusió

Finalment, aquest apartat ha estat molt interessant per a saber un poc més com sincronitzen diversos processos i ens ha donat un poc més la idea de com funcionen els videojocs, ja que estan funcionant amb moltíssims processos a l'hora movent el jugador, Carregant els gràfics, i a més a més si un jugador mata a un altre aquest creï un altre procés que fa reapareix el mateix jugador, tal com els videojocs funcionen amb diversos processos i se sincronitzen diversos a la vegada, també tenim la funcionalitat dels bancs en anar a treure diners, tal com ha explicat el professor de laboratori, si dues persones volen treure diners a la vegada del mateix compte, doncs primerament bloquejant a la segona persona que entra uns segons més tard, fins que l'usuari que a entrar primer al copte acabi de retirar els diners, un cop aquest acaba d'ona pas a la segona persona per a poder retirar diners amb la informació bancària actualitzada, ha estat molt entretinguda i interessant.