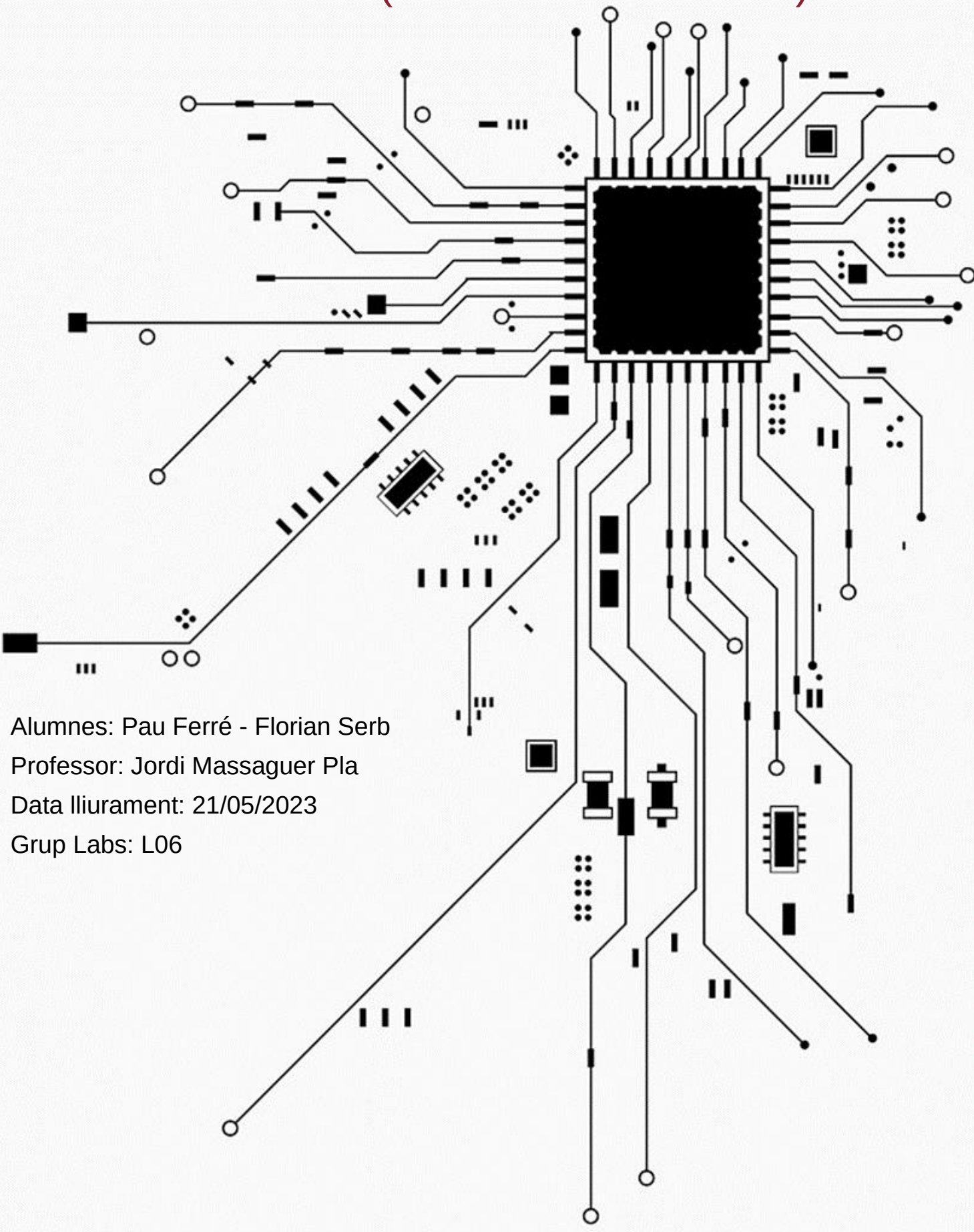
PRÀCTICA 2.2 (Threads+Proccessos)



Alumnes: Pau Ferré - Florian Serb

Professor: Jordi Massaguer Pla

Data lliurament: 21/05/2023

Grup Labs: L06

Índex

Introducció

En aquest treball, continuarem amb les anteriors fases. Canviarem els threads per processos per a veure si el famós joc menjacocos pot funcionar correctament.

Hem dividit el treball en 2 fases diferents:

1. En la primera fase canviarem la funció mou_fantasmes i la guardarem un nou fitxer anomenat "fantasmes3". Dins d'aquest, canviarem els threads per processos. La funció mou_menjacocos no realitzarem cap canvi.

2. En la segona fase, utilitzarem bústies i missatges per a poder comunicar els processos entre si.

1. Fase 1: Creació de processos

En aquesta fase adaptem el codi de la pràctica 2.1 i treballem amb diversos processos per controlar els fantasmes i un procés principal per a controlar el menjacocos.

Per a poder accedir a pantalla utilitzem el "winsuport2" el qual està format per unes rutines adaptades a l'ús d'una mateixa finestra des de diferents processos.

1.1 Procés pare

1.1.1 Especificació

En el procés pare creem els processos fills passant-los com argument la referència de la memòria compartida del cap de joc, a més de les dimensions i altres informacions necessàries. També executa un bucle principal on actualitza per pantalla el contingut del camp de joc i un cop acabada l'execució del programa destrueix la zona de memòria del camp de joc.

Dins del procés pare, encara mantenim la funció menjacocos el qual també el continuem utilitzant amb threads.

1.1.2 Disseny

Primer de tot, hem modificat la funció menjacocos. Opcionalment, podem eliminar els "mutex", ja que en aquesta fase no hi ha sincronització entre processos i threads, llavors el "mutex" no funciona.

Un cop hem modificat la funció menjacocos, passem a modificar el procés pare. Dins del procés pare també eliminem tots els mutex, passem les variables globals les quals modifiquen tant el procés pare com el procés fill i les convertim a memòria compartida. Seguidament, deixem creat el thread de la funció menjacocos i realitzarem un bucle en el qual realitzarem forks per a crear els processos i cridar-los del fitxer de configuració "fantasma3". Finalment eliminem la zona de memòria compartida

1.1.3 Implementació

```
#include "estructura.h"

pid_t tid[MAX_PROCS];

/* 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 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 i, t, t_total, rc; /* variables locals */
  int num;
  void *p_win;
  n_fi1 = atoi(ll_args[2]);
  n_fi2 = atoi(ll_args[3]);
  if (n_fi1 < 1) n_fi1 = 0;
  if (n_fi1 > 1) n_fi1 = 1;
  if (n_fi2 < 1) n_fi2 = 0;
  if (n_fi2 < 1) n_fi2 = 1;

  id_retard = ini_mem(sizeof(int));
  retard = map_mem(id_retard);
  *retard = n_retard;
  sprintf(a2, "%i", id_retard);

  id_fi1 = ini_mem(sizeof(int));
  fi1 = map_mem(id_fi1);
  *fi1 = n_fi1;
  sprintf(a2, "%i", id_fi1);

  id_fi2 = ini_mem(sizeof(int));
```

```
fi2 = map_mem(id_fi2);
*fi2 = n_fi2;
sprintf(a2, "%i", id_fi2);

srand(getpid()); /* inicialitza numeros aleatoris */
t = 0;
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[4]);
else
    *retard = 100;

rc = win_ini(&n_fil1, &n_col, '+', INVERS); /* intenta crear taulell */
id_win = ini_mem(sizeof(70));
p_win = map_mem(id_win);
sprintf(a3, "%i", id_win);
sprintf(a4, "%i", n_fil1);
sprintf(a5, "%i", n_col);
win_set(p_win, n_fil1, n_col);
if (rc > 0)                          /* si aconsegueix accedir a l'entorn CURSES */
{
    inicialitza_joc();
```

```
    //p = 0;
    *fi1 = 0;
    *fi2 = 0;
  win_retard(*retard);
  num = 0;
  pthread_create(&tid[0], NULL, mou_menjacocos, (void *)(intptr_t)0);
  for (i = 0; i < num_fan; i++)
  {
    tid[num] = fork();      /* crea un nou proces */
    if (tid[num] == (pid_t) 0)
       {
            sprintf(a1, "%i", (i+1));
       execlp("./fantasma3", "fantasma3", a1, ll_args, a2, a3, a4, a5, num_fan, f1, dc, df,
(char *)0);
       win_update();
       exit(0);
       }
    else if (tid[num] > 0) num++;
  }
    num = 0;

    for (i = 0; i <= num; i++)
    {
    waitpid(tid[i], &t, NULL);
    t = t >> 8;
    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);
    }

    elim_mem(id_retard);        /* Eliminem zona memoria compartida */
    elim_mem(id_fi1);
    elim_mem(id_fi2);
    elim_mem(id_win);
    time_t end = time(NULL);
    temps_total += (int)(end - begin);
    printf("El joc ha tardat %d segons\n", temps_total);
    return (0);
}
```

1.2 Procés fill

1.2.1 Especificació

En els processos fills mapejem la referència de memòria compartida que conté el camp de joc, utilitzant la referència que s'ha passat per argument des del procés pare. Seguidament, invoquem l'adreça de la zona de memòria mapejada anteriorment i les dimensions que han passat per arguments.

1.2.2 Disseny

Dins del document on es configuren els processos fills, el primer que realitzem és la crida del valor de les variables de la memòria compartida. Un cop hem dut a terme la crida, canviem les variables per punters.

1.2.3 Implementació

```
#include "estructura.h"


/* 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).           */


/* 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 seg;
   int k, vk, nd, vd[3];
   int p;


   id_retard = atoi(ll_args[4]);
```

```
retard = map_mem(id_retard);

id_fi1 = atoi(ll_args[2]);
fi1 = map_mem(id_fi1);

id_fi2 = atoi(ll_args[3]);
fi2 = map_mem(id_fi2);

n_fil1 = atoi(ll_args[2]);
n_col = atoi(ll_args[2]);
win_set(p_win, n_fil1, n_col);
fprintf(stderr,"\n%d\n", id_win);
p = 0;

setbuf(stdout, NULL);
srand(getpid());

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);
}
```

1.3 Joc de proves

En aquest cas no hem pogut realitzar el joc de proves ja que no ens ha acabat de funcionar el codi, però en el cas de que hagués funcionat correctament aquest hagués tingut que ser el joc de proves:

Descripció	Característiques	Resultat
L'usuari executa cocos3.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 cocos3.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. Sincronització de processos i modificació del joc

Completar la fase anterior de manera que l'execució dels processos es sincronitzi a l'hora d'accedir als recursos compartits (per exemple, l'entorn de dibuix, rutines de curses i variables globals necessàries). D'aquesta manera s'han de solucionar els problemes de visualització que presenta la fase anterior, els quals es fan patents quan s'intenta executar el programa sobre el servidor remot bsd.deim.urv.cat.

2.1 Procés pare

2.1.1 Especificació

En aquesta fase cal establir seccions crítiques que evitin els problemes de concurrència de la fase anterior. També cal implementar la funcionalitat de comunicació entre els diversos processos fill (i potser també amb el procés pare), per practicar amb el mecanisme de comunicació entre processos. Per establir seccions crítiques cal utilitzar semàfors. La comunicació entre processos es realitzarà mitjançant bústies de missatges.

2.1.2 Disseny

En cocos4 el que hem implementat ha sigut el mateix codi que teníem en el cocos3, però en aquest cas, el que volem és que els processos se sincronitzen entre ells, de tal manera no ens sorgeix cap mena de brossa per la pantalla, en aquest cas hem implementat waitS i signalS a totes les variables que estan compartint informació entre el pare i el fill, de tal manera el joc funciona sincronitzadament i no surt brossa per pantalla.

Malauradament com el cocos3 no acaba de funcionar correctament, ja que no podem comprovar que funcioni correctament, encara així els waitS i signalS, estan en la mateixa posició que els lock/unlock del cocos2, això vol dir que si poguéssim resoldre el problema en cocos3, aquest apartat funcionaria correctament.

2.1.3 Implementació

```
#include "estructura.h"

pid_t tid[MAX_PROCS];

/* 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);
   }
}
void *mou_menjacocos(void *null)
{
```

```
char strin[12];
objecte seg;
int tec, ret;
int id_sem, id_bustia;
char mis[2];
id_sem = atoi(ll_args[4]);    // OBTENIR IDENTIFICADO SEMAFORO
id_bustia = atoi(ll_args[5]); // OBTERNIR IDENTIFICADOR BUSTIA
ret = 0;
do
{
    win_retard(*retard);
    waitS(id_sem);
    tec = win_gettec();
    singalS(id_sem);
    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;
    waitS(id_sem);
    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 */
    signalS(id_sem);
    if ((seg.a == ' ') || (seg.a == '.'))
    {
        waitS(id_sem);
        win_escricar(mc.f, mc.c, ' ', NO_INV); /* esborra posicio anterior */
        signalS(id_sem);
        waitS(id_sem);
        mc.f = seg.f;
        mc.c = seg.c;                    /* actualitza posicio */
        win_escricar(mc.f, mc.c, '0', NO_INV); /* redibuixa menjacocos */
        signalS(id_sem);
        if (seg.a == '.')
        {
            cocos--;
            sprintf(strin, "Cocos: %d", cocos);
            win_escristr(strin);
```

```
            if (cocos == 0)
            {
                *fi2 = 1;
            }
        }
    }
  } while (!*fi1 && !*fi2);
  elim_mis(id_bustia);
  elim_sem(id_bustia);
}
/* programa principal                    */
int main(int n_args, const char *ll_args[])
{
  int temps_total = 0;
  time_t begin = time(NULL);
  int i, t, t_total, rc; /* variables locals */
  int num;
  void *p_win;
  n_fi1 = atoi(ll_args[2]);
  n_fi2 = atoi(ll_args[3]);
  if (n_fi1 < 1)
    n_fi1 = 0;
  if (n_fi1 > 1)
    n_fi1 = 1;
  if (n_fi2 < 1)
    n_fi2 = 0;
  if (n_fi2 < 1)
    n_fi2 = 1;
```

```
id_retard = ini_mem(sizeof(int));
retard = map_mem(id_retard);
*retard = n_retard;
sprintf(a2, "%i", id_retard);
id_fi1 = ini_mem(sizeof(int));
fi1 = map_mem(id_fi1);
*fi1 = n_fi1;
sprintf(a2, "%i", id_fi1);
id_fi2 = ini_mem(sizeof(int));
fi2 = map_mem(id_fi2);
*fi2 = n_fi2;
sprintf(a2, "%i", id_fi2);
srand(getpid()); /* inicialitza numeros aleatoris */
t = 0;
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[4]);
else
    *retard = 100;
rc = win_ini(&n_fil1, &n_col, '+', INVERS); /* intenta crear taulell */
id_win = ini_mem(sizeof(70));
p_win = map_mem(id_win);
sprintf(a3, "%i", id_win);
```

```
  sprintf(a4, "%i", n_fil1);
  sprintf(a5, "%i", n_col);
  win_set(p_win, n_fil1, n_col);
  if (rc > 0) /* si aconsegueix accedir a l'entorn CURSES */
  {
     inicialitza_joc();
     *fi1 = 0;
     *fi2 = 0;
     win_retard(*retard);
     num = 0;
     pthread_create(&tid[0], NULL, mou_menjacocos, (void *)(intptr_t)0);
     pthread_join(tid[i], (void **)&t);
     for (i = 0; i < num_fan; i++)
     {
        tid[num] = fork(); /* crea un nou proces */
        if (tid[num] == (pid_t)0)
        {
           sprintf(a1, "%i", (i + 1));
           execlp("./fantasma3", "fantasma3", a1, ll_args, a2, a3, a4, a5, num_fan, f1, dc, df,
(char *)0);
           win_update();
           exit(0);
        }
        else if (tid[num] > 0)
           num++;
     }
     num = 0;
     for (i = 0; i <= num; i++)
```

```
    {
        waitpid(tid[i], &t, NULL);
        t = t >> 8;
        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);
    }
    elim_mem(id_retard); /* Eliminem zona memoria compartida */
    elim_mem(id_fi1);
    elim_mem(id_fi2);
    elim_mem(id_win);
    time_t end = time(NULL);
    temps_total += (int)(end - begin);
    printf("El joc ha tardat %d segons\n", temps_total);
    return (0);
}
```

2.2 Procés fill

2.2.1 Especificació

En aquesta fase cal establir seccions crítiques que evitin els problemes de concurrència de la fase anterior. També cal implementar la funcionalitat de comunicació entre els diversos processos fill (i potser també amb el procés pare), per practicar amb el mecanisme de comunicació entre processos. Per establir seccions crítiques cal utilitzar semàfors. La comunicació entre processos es realitzarà mitjançant bústies de missatges.

2.2.2 Disseny

En el procés fill de cocos4, el que hem implementat ha sigut els semàfors de tal manera se sincronitzen els processos i d'aquesta manera evitem mostrar per pantalla brossa. Tal com he dit en el procés pare, si el cocos3 funciones correctament, aquest també funcionaria, ja que el que hem fet és canviar els unlock/lock del cocos2 per waitS i signalS, això vol dir que en principi hauria de funcionar correctament.

2.2.3 Implementació

```
#include "estructura.h"
int main(int n_args, char *ll_args[])
{
  objecte seg;
  int k, vk, nd, vd[3];
  int p;
  int id_sem, id_bustia;
  char mis[2];
  id_sem = atoi(ll_args[4]);    // OBTENIR IDENTIFICADOR SEMAFORO
  id_bustia = atoi(ll_args[5]); // OBTENIR IDENTIFICADOR BUSTIA
  id_retard = atoi(ll_args[4]);
  retard = map_mem(id_retard);
  id_fi1 = atoi(ll_args[2]);
  fi1 = map_mem(id_fi1);
  id_fi2 = atoi(ll_args[3]);
  fi2 = map_mem(id_fi2);
  n_fil1 = atoi(ll_args[2]);
```

```
n_col = atoi(ll_args[2]);
win_set(p_win, n_fil1, n_col);
fprintf(stderr, "\n%d\n", id_win);
p = 0;
setbuf(stdout, NULL);
srand(getpid());
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 */
                }
                waitS(id_sem);
                seg.f = f1[w].f + df[vk]; /* calcular posicio en la nova dir.*/
                seg.c = f1[w].c + dc[vk];
                signalS(id_sem);
                waitS(id_sem);
                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) /* si no pot continuar, */
    {
        waitS(id_sem);
        f1[w].d = (f1[w].d + 2) % 4; /* canvia totalment de sentit */
        signalS(id_sem);
    }
    else
    {
        if (nd == 1) /* si nomes pot en una direccio*/
        {
            waitS(id_sem);
            f1[w].d = vd[0]; /* li assigna aquesta */
            signalS(id_sem);
        }
        else /* altrament */
        {
            waitS(id_sem);
            f1[w].d = vd[rand() % nd]; /* segueix una dir. aleatoria */
            signalS(id_sem);
        }
        waitS(id_sem);
        seg.f = f1[w].f + df[f1[w].d]; /* calcular seguent posicio final */
```

```
                seg.c = f1[w].c + dc[f1[w].d];
                signalS(id_sem);
                waitS(id_sem);
                seg.a = win_quincar(seg.f, seg.c);
                win_escricar(f1[w].f, f1[w].c, f1[w].a, NO_INV); /* esborra posicio anterior */
                signalS(id_sem);
                waitS(id_sem);
                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 */
                signalS(id_sem);
                if (f1[w].a == '0')
                {
                    waitS(id_sem);
                   *fi1 = 1; /* ha capturat menjacocos */
                   signalS(id_sem);
                }
            }
        }
    }
  } while (!*fi1 && !*fi2);
  elim_mis(id_bustia);
  elim_sem(id_bustia);
}
```

2.3 Joc de proves

En aquest cas no hem pogut realitzar el joc de proves ja que no ens ha acabat de funcionar el codi, però en el cas de que hagués funcionat correctament aquest hagués tingut que ser el joc de proves:

Descripció	Característiques	Resultat
L'usuari executa cocos4.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 cocos4.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

Conclusió

Finalment, aquest apartat ens ha costat bastant entendre que és el que demanaven, ja que primerament hem tingut que el mou_menjacocos i mou_fantasma, que aquests dos han de ser fills, però realment sols el mou_fantasma havia de ser fill, i just quan ens han dit que això és així, ja teníem tot el codi modificat, ara ja l'hem reestructurat, però per la falta de coneixements i de temps, no hem aconseguit fer que funcioni correctament el cocos3 i el cocos4, hem provat d'adjuntar el cocos3.c amb el mou_menjacocos.c i d'aquesta manera sols tindrem com a fill al mou_fantasma, un cop ja ho teníem ajuntat, hem fet les proves i les investigacions necessàries, però no hi ha hagut manera de solucionar els problemes, encara així adjuntem el codi i l'explicació de com ho hem fet i els errors que ens sorgeixen.