FSO
PRÀCTICA 2.2

Eduard Bel
Antonio Torres Cabero
Cristina Izquierdo Lozano

Índex

Especificacions

Joc del “Tron”: motos que deixen un rastre, l’objectiu de les quals es fer que els oponents xoquin contra aquests rastres.

- Tron usuari: ‘0’
- Trons oponents: ‘[1-9]’
- Tecles de direcció de moviment de l’usuari: 'w' (adalt), 's' (abaix), 'd' (dreta) i 'a' (esquerra).
- Variabilitat del moviment dels oponents: 0-3, indicat per paràmetre.

Quan un tron, usuari o oponent, xoca s’elimina la seva traça mentres que la resta de trons continuen jugant. És a dir, si mentre un tron retira la seva traça, un altre tron xoca també morirà i, per tant, serà un empat.

Execució: `./tron[3,4] num_oponents[0-9] fitxer variabilitat[0-3] [retard]`

En aquesta segona part de la pràctica es demana canviar els threads per processos, el quals controlaran els trons oponents des d’un fitxer executable diferent al del programa principal.
A més a més, els trons oponents es comunicaran entre ells, en cas de xocar, mitjançant bústies de missatges.

Fases

1. Creació de processos (‘tron3.c’/ ‘oponent3.c’)

Convertir els threads que controlen els trons oponents en processos independents. El control de l’usuari continuarà com un thread dins del procés pare.
Programa principal crea:

- Un thread per l’usuari
- Varis processos per als oponents.

2. Sincronització de processos (‘tron4.c’/ ‘oponent4.c’)

Sincronitzar l’execució dels processos a l’hora d’accedir als recursos compartits i implementar la interacció entre els trons oponents mitjançant les bústies de missatges.

Disseny

Primerament haurem de separar les funcions pròpies dels oponents a un nou fitxer anomenar `oponent3.c i oponent4.c,` per la segona fase. Aquestes són:

- `mou_oponent`:
    - El funcionament és el mateix que a la primera fase. La única cosa que canvia es que la matriu de posicions `p_opo[index][n_opo]` ara passa a ser un vector `p_opo[n_opo]`, ja que cada oponent tindrà el seu procés.
- `esborrar_posicions`:
    - La mateixa funció que es troba pels usuaris a `tron3.c` es copia a `oponent3.c`.

A més a més, el programa principal de `oponent3.c` i `oponent4.c` s'encarrega de controlar les variables passades per paràmetre des del procés pare i d'obtenir les adreces de memòria compartida per les variables de final del joc i de pantalla. Aquest també s'encarrega de inicialitzar els trons oponents.

Finalment, mantenim les seccions crítiques que es van crear a la fase anterior i les controlem amb semàfors ipc.

Implementació

Funcions

Respecte a les funcions i el codi principal del programa, tan sols mostrarem els canvis que hem realitzat nosaltres al codi proporcionat per a la pràctica.

Main oponents (processos fills)

```
int main(int n_args, char *ll_args[])
{

 ind = atoi(ll_args[1]) + 1;
 pid = atoi(ll_args[2]);
 fila  = atoi(ll_args[3]);
 col = atoi(ll_args[4]);
 id_win  = atoi(ll_args[5]);
 sem_globals= atoi(ll_args[6]);
 sem_pantalla= atoi(ll_args[7]);
 final1= atoi(ll_args[8]);
 oponents_ab= atoi(ll_args[9]);
 final2= atoi(ll_args[10]);
 num_oponents= atoi(ll_args[11]);
 retard= atoi(ll_args[12]);
 varia = atoi(ll_args[13]);
 id_bustia = atoi(ll_args[14]);


 p_win = map_mem(id_win);  /* obtenir adres. de mem. compartida */
 fi1 = map_mem(final1);  /* obtenir adres. de mem. compartida */
 oponents_abatuts = map_mem(oponents_ab);  /* obtenir adres. de mem.
compartida */
 fi2 = map_mem(final2);  /* obtenir adres. de mem. compartida */

 win_set(p_win, fila, col);
 p_opo = calloc(fila*col/2, sizeof(pos));  /* per a les posicions ant.
*/

 if (!p_opo) /* si no hi ha prou memoria per als vectors de pos. */
```

```
 {
   win_fi();       /* tanca les curses */
   if (p_opo) free(p_opo);    /* allibera el que hagi pogut obtenir */
   fprintf(stderr,"Error en alocatacion de memoria dinamica.\n");
   exit(3);
 }

 int n_fil =fila+ind + num_oponents;
 opo.f = ind*2 + num_oponents; /* Els trons oponents mantenen la
columna pero es reparteixen de forma equidistant per les files */
 opo.c = (col*3)/4;    /* fixa posicio i direccio inicial oponent */
 opo.d = (rand() % 4); /* direccio inicial aleatoria (0-3) --> rand() %
N+1 = aleatori [0..N]*/
 win_escricar(opo.f,opo.c,'0'+ind,INVERS); /* escriu la primer posicio
oponent */
 p_opo[0].f = opo.f;   /* memoritza posicio inicial */
 p_opo[0].c = opo.c;
  mou_oponent((void *)(intptr_t) ind);
 free(p_opo);       /* allibera la memoria dinamica obtinguda */
}
```

Main tron (procés pare)

```
[...]

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

 p_oponents_abatuts = map_mem(id_oponents_abatuts);  /* obtenir adres. de mem.
compartida */

 p_fi1=map_mem(fi1); /* obtenir adres. de mem. compartida */

 p_fi2=map_mem(fi2); /* obtenir adres. de mem. compartida */

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

 p_usu = calloc(n_fil*n_col/2, sizeof(pos)); /* demana memoria dinamica */

 if (!p_usu) /* si no hi ha prou memoria per als vectors de pos. */
```

```
{ win_fi();       /* tanca les curses */

  if (p_usu) free(p_usu);

  fprintf(stderr,"Error en alocatacion de memoria dinamica.\n");

  exit(3);

}

    /* Fins aqui tot ha anat be! */

inicialitza_joc();

 n = 0; /*num del proces*/

/*Creem un thread per a l'usuari*/

sem_globals=ini_sem(1);

sem_pantalla=ini_sem(1);

id_bustia = ini_mis();    /* crear bustia IPC */

 char a0[10], a1[10], a2[10], a3[10], a4[10], a5[10], a6[10], a7[10], a8[10], a9[10], a10[10],
a11[10], a12[10], a13[10];

 pthread_create(&tid[0], NULL, mou_usuari, (void *)(intptr_t) 0);

 sprintf(a1, "%i", n_fil);

 sprintf(a2, "%i", n_col);

 sprintf(a3, "%i", id_win);

 sprintf(a5, "%i", sem_globals);
```

```
sprintf(a6, "%i", sem_pantalla);

sprintf(a7, "%i", fi1);

sprintf(a8, "%i",id_oponents_abatuts);

sprintf(a9, "%i",fi2);

sprintf(a10, "%i",num_oponents);

sprintf(a11, "%i",retard);

sprintf(a12, "%i", varia);

sprintf(a13, "%i", id_bustia);

/*Creem tants processos com a oponents s'hagin indicat per parametre*/

for (i = 0; i < num_oponents; i++)

{

  tid[n] = fork();    /* crea un nou proces */

  if (!tid[n])    /* branca del fill */

  {

    sprintf(a0, "%i", getpid());

    sprintf(a4, "%i", i);

    execlp("./oponent3", "oponent3", a4, a0, a1, a2, a3, a5, a6, a7, a8, a9, a10, a11, a12,
a13, (char *)0);

    fprintf(stderr,"error: no puc executar el process fill \'oponent3\'\n");
```

```
    exit(0);
  }
  else if (tid[n] > 0) n++;   /* branca del pare */
}
double seconds=0;
int minutes=0;
char strin[45];
do      /********** bucle principal del joc **********/
{
  win_retard(retard);
  seconds = seconds + (double) retard / 1000;
  if (seconds>=60) {
      minutes++;
      seconds = seconds - 60;
  }
  sprintf(strin, "Time %d : %.0f", minutes, seconds);
  win_escristr(strin);
  win_update();     /* actualitza visualitzacio CURSES */
} while (!*p_fi1 && !*p_fi2);
```

```
/*esperar a que els processos fills acabin la seva execucio*/

int status = 0;

for ( i = 0; i <= n; i++)

{

  waitpid(tid[i],&status,0);  /* espera finalitzacio d'un fill */

}

win_fi();       /* tanca les curses */

if (*p_fi1 == -1) printf("S'ha aturat el joc amb tecla RETURN!\n\n");

else {

  if (*p_fi1)

  {

    fprintf(fitxer, "%s %s tron guanyat %c: %d\n",dia,hora,'1',n_opo);

     printf("Ha guanyat l'ordinador!                              \n\n");

  }

  else {

   fprintf(fitxer, "%s %s tron guanyat %c: %d\n",dia,hora,'0',n_usu);

     printf("Ha guanyat l'usuari!                                 \n\n");

  }

}
```

```
fclose(fitxer);

free(p_usu);

elim_mis(id_bustia);/* elimina bustia */

elim_sem(sem_globals);/* elimina semafor */

elim_sem(sem_pantalla);

elim_mem(id_win);/* elimina zona de memoria compartida */

elim_mem(id_oponents_abatuts);/* elimina zona de memoria compartida */

elim_mem(fi1);/* elimina zona de memoria compartida */

elim_mem(fi2);/* elimina zona de memoria compartida */

 return(0);

}
```

Joc de proves

Casos possibles:

1. **Introduir dades errònies:**

    a. Falta de paràmetres: missatge d'error on indica els paràmetres necessaris.
    b. Paràmetres erronis:
        i. si *(n_oponents < 1)* n_oponents = 1
        ii. si *(n_oponents > 9)* n_oponents = 9
        iii. si *(variabilitat < 0)* n_oponents = 0
        iv. si *(variabilitat > 3)* n_oponents = 3
        v. si hi ha retard (per defecte, retard = 100):
            1. si *(retard < 10)* retard = 10
            2. si *(retard > 1000)* retard = 1000

2. **Tron usuari xoca però encara queden trons oponents al joc:**
    a. Els trons oponents segueixen amb la seva execució mentre s'esborra l'usuari.

3. **Es reserva memoria per a un oponent i falla:**
    a. Aquest oponent no jugarà la partida pero la resta sí, a més a més, es notificarà

4. **El tron oponent es queda sense espai on dirigir-se:**
    a. Xocarà i s'esborraràn les seves posicions.