Lab 8: Sincronització de Processos

Objectius:

- Aprendre les comandes bàsiques de creació i d'utilització de semàfors (IPC).
- Aprendre les comandes bàsiques de creació i d'utilització de bústies (IPC).

Fitxers d'exemple:

```
/usr/local/assignatures/fso/exemples_C/mp_sincro.c
/usr/local/assignatures/fso/exemples_C/mp_car_s.c
/usr/local/assignatures/fso/exemples_C/memoria.h
/usr/local/assignatures/fso/exemples_C/semafor.c
/usr/local/assignatures/fso/exemples_C/missatge.h
/usr/local/assignatures/fso/exemples_C/missatge.c
/usr/local/assignatures/fso/exemples_C/Makefile
```

Funcions de gestió de semàfors ('semafor.h'):

```
int ini_sem(int valor);
```

La funció `ini_sem` crea un semàfor que servirà per sincronitzar diversos processos. El paràmetre indica el valor inicial que tindrà associat el semàfor. La funció retorna un identificador (IPC) del semàfor creat.

```
void elim_sem(int id_sem);
```

La funció `elim_sem` elimina el semàfor especificat per l'identificador `id_sem`.

```
void waitS(int id_sem);
```

La funció `waitS` decrementa el valor associat al semàfor especificat per l'identificador `id_sem`, en cas que aquest valor sigui superior a zero. Altrament, el procés que invoca la funció es queda bloquejat en una cua de processos pròpia del semàfor.

```
void signalS(int id_sem);
```

La funció `signalS` incrementa el valor associat al semàfor especificat per l'identificador `id_sem`, en cas que aquest valor sigui igual o superior a zero. Altrament, la crida a aquesta funció desbloqueja el primer procés que estigui bloquejat a la cua d'aquest semàfor.

Funcions de gestió de bústies ('missatge.h'):

`int ini_mis();`

La funció `ini_mis` crea una bústia que servirà per a que els processos s'intercanviïn missatges (1 missatge = vectors de fins a 128 bytes). La funció retorna un identificador (IPC) de la bústia creada.

`void elim_mis(int id_mis);`

La funció `elim_mis` elimina la bústia especificada per l'identificador `id_mis`, juntament amb tots els missatges que contingui.

`void sendM(int id_mis, void * missatge, int nbytes);`

La funció `sendM` encua un missatge a la bústia especificada per l'identificador `id_mis`. El missatge es passa per referència en el segon paràmetre de la funció, mentre que en el tercer paràmetre cal especificar el número total de bytes del missatge.

`int receiveM(int id_mis, void * missatge);`

La funció `receiveM` extreu el primer missatge de la bústia especificada per l'identificador `id_mis`, en cas que n'hi hagi algun. Altrament, el procés que crida a aquesta funció es queda bloquejat fins que algún altre procés enviï algun missatge a través de la funció `sendM`. La funció copia el missatge rebut dins de la zona de memòria indicada pel paràmetre `missatge` (pas per referència) i retorna el número total de bytes copiats.

Programes d'exemple:

```
/*****************************************************************************/
/*                            mp_sincro.c                                    */
/*                                                                           */
/*    Compilar i executar:                                                   */
/*    $ gcc -Wall mp_sincro.c memoria.o semafor.o missatge.o -o mp_sincro */
/*    $ ./mp_sincro  num_procs  n_vegades  n_lletres                        */
/*                                                                           */
/*    Crea num_procs processos, on cada proces executara el fitxer           */
/*    'mp_car_s' que ha d'estar accessible al mateix directori que           */
/*    'mp_sincro'.                                                           */
/*    Cada proces fill visualitzara un caracter identificatiu                */
/*    (proc 1 -> 'a', proc 2 -> 'b', ...) tantes vegades com                 */
/*    indiqui l'argument n_vegades, esperant un temps aleatori               */
/*    entre dues visualitzacions.                                            */
/*    El programa mp_sincro acabara la seva execucio quan s'hagin            */
/*    escrit un total de n_lletres entre tots els processos fill.            */
/*                                                                           */
/*    Aquest programa esta basat en 'multiproc.c', pero afegeix              */
/*    mecanismes de sincronitzacio entre processos (semafor, bustia).        */
/*****************************************************************************/

#include <stdio.h>
#include "memoria.h"
#include "semafor.h"
#include "missatge.h"

#define MAX_PROCS       10
#define MAX_VEGADES     50
#define MAX_LLETRES     100

            /* Variables Globals */
pid_t tpid[MAX_PROCS];       /* taula d'identificadors dels processos fill */


int main(int n_args, char * ll_args[])
{
    int i,n,t,t_total,n_proc,n_veg,n_lletres;
    int *p_lletres,id_lletres;
    int id_sem, id_bustia;
    char a1[20],a2[20],a3[20],a4[20];

    if (n_args != 4)
    {   fprintf(stderr,"comanda: multiproc num_procs n_vegades n_lletres\n");
        exit(1);
    }
    n_proc = atoi(ll_args[1]);           /* convertir arguments a num. enter */
    n_veg = atoi(ll_args[2]);
    if (n_proc < 1) n_proc = 1;          /* i filtrar el seu valor */
    if (n_proc > MAX_PROCS) n_proc = MAX_PROCS;
    if (n_veg < 1) n_veg = 1;
    if (n_veg > MAX_VEGADES) n_veg = MAX_VEGADES;
    if (n_lletres < 1) n_lletres = 1;
    if (n_lletres > MAX_LLETRES) n_lletres = MAX_LLETRES;

    id_lletres = ini_mem(sizeof(int));   /* crear zona mem. compartida */
    p_lletres = map_mem(id_lletres);     /* obtenir adres. de mem. compartida*/
    *p_lletres = n_lletres;              /* inicialitza variable compartida */
    sprintf(a2,"%i",id_lletres);    /* convertir identificador mem. en string*/
```

```
    id_sem = ini_sem(1);          /* crear semafor IPC inicialment obert */
    sprintf(a3,"%i",id_sem);      /* convertir identificador sem. en string */

    id_bustia = ini_mis();        /* crear bustia IPC */
    sprintf(a4,"%i",id_bustia);  /* convertir identif. bustia en string */

    printf("PID del proces pare = %d\n", getpid());
    n = 0;
    for ( i = 0; i < n_proc; i++)
    {
      tpid[n] = fork();       /* crea un nou proces */
      if (tpid[n] == (pid_t) 0)          /* branca del fill */
      {
          sprintf(a1,"%i",(i+1));
      execlp("./mp_car_s", "mp_car_s", a1, ll_args[2], a2, a3, a4, (char *)0);
      fprintf(stderr,"error: no puc executar el process fill \'mp_car_s\'\n");
          exit(0);
      }
      else if (tpid[n] > 0) n++;         /* branca del pare */
    }
    printf("He creat %d processos fill, espero que acabin!\n\n",n);

    t_total = 0;
    for (i = 0; i < n; i++)
    {
      waitpid(tpid[i],&t,NULL);     /* espera finalitzacio d'un fill */
      t = t >> 8;             /* normalitza resultat  */
      printf("el proces (%d) ha escrit %d lletres\n",tpid[i],t);
      t_total += t;
    }
    printf("\nJa han acabat tots els processos creats!\n");
    printf("Entre tots els processos han escrit %d lletres.\n",t_total);

  printf("\nLa sequencia de lletres enviades pels processos es la seguent:\n");
    for (i = 0; i < t_total; i++)
    {                   /* per a totes les lletres enviades */
      receiveM(id_bustia,a1);       /* treu un missatge (string) de la bustia */
      printf("%c",a1[0]);           /* escriu el primer caracter */
    }
    printf("\n\n");

    elim_mis(id_bustia);      /* elimina bustia */
    elim_sem(id_sem);         /* elimina semafor */
    elim_mem(id_lletres);     /* elimina zona de memoria compartida */

    return(0);
}
```

```
/*****************************************************************/
/*                          mp_car_s.c                           */
/*                                                               */
/*    Compilar i executar:                                       */
/*    $ gcc -Wall mp_car_s.c memoria.o semafor.o missatge.o -o mp_car_s */
/*    $ ./mp_car_s  n_car  n_vegades  id_lletres  id_sem  id_bustia     */
/*                                                               */
/*    Escriu per la sortida estandard un caracter identificatiu  */
/*    segons el primer argument 'n_car' (1 -> 'a', 2 -> 'b', ...),  */
/*    tantes vegades com indica el segon argument 'n_vegades',   */
/*    esperant un temps aleatori entre dues visualitzacions; el  */
/*    programa acaba quan s'han escrit tantes lletres com indiqui el */
/*    contingut de la zona de memoria compartida identificada pel */
/*    tercer argument 'id_lletres'.                              */
/*    A més, utilitza el semafor indicat en 'id_sem' per realitzar */
/*    exclusio mutua a l'hora d'accedir a la variable compartida. */
/*    Finalment, totes les lletres que genera les envia a la bustia */
/*    especifica en el cinqué argument.                          */
/*                                                               */
/*    Aquest programa s'executara com un proces fill del programa */
/*    'mp_sincro' (veure fitxer 'mp_sincro.c').                  */
/*****************************************************************/

#include <stdio.h>
#include "memoria.h"
#include "semafor.h"
#include "missatge.h"

int main(int n_args, char *ll_args[])
{
  int i,nc,nv;
  int *p_lletres, id_lletres;
  int id_sem, id_bustia;
  char mis[2];
:
:
  id_sem = atoi(ll_args[4]);        /* obtenir identificador de semafor */
  id_bustia = atoi(ll_args[5]);     /* obtenir identificador de bustia */

  srand(getpid());
  for (i=0; i < nv; i++)      /* per a totes les vegades (nv) */
  {
    sleep(1 + rand() % 3);    /* dormir entre 1 i 3 segons */
    waitS(id_sem);            /* tanca semafor */
    if (*p_lletres > 0)       /* si falten lletres */
    { sprintf(mis,"%c",('a'+nc-1)); /* genera marca del proces */
      sendM(id_bustia,mis,2);       /* envia marca al pare */
      (*p_lletres)--;               /* una lletra menys */
      signalS(id_sem);        /* obre semafor */
     }
     else
     {signalS(id_sem);        /* obre semafor */
      exit(i);    /* provoca la sortida anticipada del proces */
     }
  }
  return(i);      /* retorna el numero de lletres que ha impres el proces */
}
```

Comandes de control d'IPCs:

Si compilem i executem els programes anteriors:

```
$ make
$ ./mp_sincro 5 10 30
PID del proces pare = 11807
He creat 5 processos fill, espero que acabin!
```

Llavors aturem l'execució amb Control-Z:

```
^Z
[1]+  Stopped                 ./mp_sincro 5 10 30
$
```

Podem veure els processos que s'han creat:

```
$ ps
   PID TTY      TIME CMD
 11833 pts/2    0:00 mp_car_s
 11835 pts/2    0:00 mp_car_s
 11834 pts/2    0:00 mp_car_s
 10641 pts/2    0:00 bash
 11832 pts/2    0:00 mp_sincr
 11838 pts/2    0:00 ps
 11836 pts/2    0:00 mp_car_s
 11837 pts/2    0:00 mp_car_s
```

Podem veure els *jobs* que s'han creat:

```
$ jobs
[1]+  Stopped                 ./mp_sincro 5 10 30
```

Podem veure els *IPCs* que s'han creat:

```
$ ipcs
IPC status from <running system> as of Thu Nov 11 20:29:48 CET
2004
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
q        700   0          --rw-------  sromani   profes
Shared Memory:
m       1600   0          --rw-------  sromani   profes
Semaphores:
s     983040   0          --ra-------  sromani   profes
```

Si el programa falla, els *ipcs* NO s'eliminaran. Podem provar-ho "matant" el *job*:

```
$ kill -9 %1
```

Tornant a executar la comanda `ipcs` es pot veure que no s'han eliminat. Podem eliminar-los individualment amb la comanda `ipcrm`:

```
$ ipcrm -q 700
$ ipcrm -m 1600
$ ipcrm -s 1600
```

En el servidor *sol*, podem eliminar-los tots de cop amb la comanda composta següent:

```
$ ipcrm $(ipcs | grep $LOGNAME | awk '{print "-" $1,$2}')
```

Aquesta última comanda es disposa de forma empaquetada com un *script* anomenat '`mataipc`', al directori d'exemples. És molt convenient tenir una còpia d'aquest *script* dins del directori on es desenvolupin els programes que manipulin IPCs, i utilitzar el *script* de forma periòdica.

Exercici:

Completar la versió definitiva de la pràctica 2.2, és a dir, introduir els mecanismes de sincronització de processos per tal de regular l'accés a les zones de memòria compartida (zona de dibuix, etc.) i per permetre als processos fill intercanviar-se informació a través de missatges.