Lab 7: Creació de Processos

Objectius:

- Entendre el concepte de programació multiprocés.
- Aprendre les comandes bàsiques de creació i de control de processos fill.
- Aprendre les comandes bàsiques de creació i d'utilització de memòria compartida.

Fitxers d'exemple:

```
/usr/local/assignatures/fso/exemples_C/multiproc.c
/usr/local/assignatures/fso/exemples_C/mp_car.c
/usr/local/assignatures/fso/exemples_C/memoria.h
/usr/local/assignatures/fso/exemples_C/memoria.c
```

Funcions bàsiques de sistema:

```
pid_t fork(void);
```

La funció `fork` crea un duplicat (codi, dades, pila) del procés que l'executa. El nou procés s'anomena *procés fill*, mentre que el procés original s'anomena *procés pare*. Els dos processos continuen la seva execució a partir de la instrucció següent a la crida `fork`. Habitualment, en aquest punt del programa caldrà determinar si s'està executant el codi del procés pare o del procés fill, avaluant l'identificador que retorna la funció `fork`: si retorna 0, es tracta del procés fill; si retorna més gran que 0, es tracta del procés pare (el valor retornat és el PID del procés fill); si retorna menor que 0, es tracta d'un error en l'execució del `fork`.

```
int execlp(char *file, char *arg0, char *arg1, char *arg2, ...);
```

La funció `execlp` substitueix el codi, les dades i la pila del procés que la invoca per un nou entorn que es carregarà a partir del fitxer executable `file`. L'única cosa que es conservarà de l'antic entorn és la taula de canals oberts (`stdin`, `stdout`, `stderr`, més altres fitxers). Els paràmetres `arg0`, `arg1`, `arg2`, etc. seran els arguments que es passaran al nou programa, com si s'hagués executat des de l'intérpret de comandes. Per tant, l'argument `arg0` hauria de ser el nom del fitxer executable. Per indicar el final de la llista d'arguments cal afegir el *paràmetre sentinella* `(char *)0`. Si tot va bé, les instruccions que segueixen a la crida `execlp` no s'executaran.

```
pid_t getpid(void);
```

La funció `getpid` retorna l'identificador del procés que la invoca.

```
pid_t waitpid(pid_t wpid, int *status, int options);
```

La funció `waitpid` bloqueja l'execució del procés que la invoca, a l'espera de que el procés especificat en el paràmetre `wpid` acabi la seva execució. Quan el procés indicat acaba, el valor que retorna es copia dins de la variable `status` (pas per referència). Habitualment, el paràmetre `options` es posa a `NULL`.

Funcions de manipulació de memòria compartida ('memoria.h'):

```
int ini_mem(int mida);
```

La funció `ini_mem` crea una zona de memòria (reserva dinàmica) que podrà ser compartida per diversos processos. El paràmetre `mida` indica la mida en bytes que ha de tenir la zona de memòria. La funció retorna un identificador (IPC) de la zona de memòria creada.

```
void elim_mem(int id_shm);
```

La funció `elim_mem` elimina la zona de memòria compartida especificada per l'identificador `id_shm`.

```
void * map_mem(int id_shm);
```

La funció `map_mem` permet l'accés a la zona de memòria compartida especificada per l'identificador `id_shm`. La funció retorna l'adreça (`void *`) de l'inici de la zona de memòria.

Programes d'exemple:

```
/*****************************************************************/
/*                      multiproc.c                              */
/*                                                               */
/*    Compilar i executar:                                       */
/*          $ gcc -Wall multiproc.c memoria.o -o multiproc       */
/*          $ ./multiproc  num_procs  n_vegades  n_lletres       */
/*                                                               */
/*    Crea num_procs processos, on cada proces executara el fitxer */
/*    'mp_car' que ha d'estar accessible al mateix directori que */
/*    'multiproc'.                                               */
/*    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 multiproc acabara la seva execucio quan s'hagin */
/*    escrit un total de n_lletres entre tots els processos fill. */
/*****************************************************************/

#include <stdio.h>
#include "memoria.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;
    char a1[20],a2[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]);
    n_lletres = atoi(ll_args[3]);
    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 adreça mem. compartida */
    *p_lletres = n_lletres;         /* inicialitza variable compartida */
    sprintf(a2,"%i",id_lletres);    /* convertir id. memoria 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", "mp_car", a1, ll_args[2], a2, (char *)0);
          fprintf(stderr,"error: no puc executar el process fill \'mp_car\'\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);

    elim_mem(id_lletres);     /* elimino zona de memoria compartida */

    return(0);
}
```

```
/*****************************************************************/
/*                      mp_car.c                                 */
/*                                                               */
/*    Compilar:                                                  */
/*          $ gcc  mp_car.c memoria.o -o mp_car                  */
/*                                                               */
/*    Escriu per la sortida estandard un caracter identificatiu  */
/*    segons l'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'.                              */
/*                                                               */
/*    Aquest programa s'executara com un proces fill del programa*/
/*    'multiproc' (veure fitxer 'multiproc.c').                  */
/*****************************************************************/

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

int main(int n_args, char *ll_args[])
{
  int i,nc,nv;
  int *p_lletres, id_lletres;

  if (n_args != 4)
  {   fprintf(stderr,"proces: mp_car  n_car  n_vegades  id_lletres\n");
      exit(0);
  }
  nc = atoi(ll_args[1]);
  nv = atoi(ll_args[2]);

  id_lletres = atoi(ll_args[3]);
  p_lletres = map_mem(id_lletres); /*obtenir adres. de mem. compartida */
  if (p_lletres == (int*) -1)
  {   fprintf(stderr,"proces (%d): error en identificador de memoria\n",
                                getpid());
      exit(0);
  }
  setbuf(stdout,NULL);  /* anular buffer de sortida estandard */
  srand(getpid());

  for (i=0; i < nv; i++)      /* per a totes les vegades (nv) */
  {
    if (*p_lletres > 0)       /* si falten lletres */
    {
      printf("%c",('a'+nc-1));      /* escriure marca del proces */
      sleep(1 + rand() % 3);        /* dormir entre 1 i 3 segons */
      (*p_lletres)--;               /* una lletra menys */
     }
     else exit(i);      /* sino, provoca la sortida anticipada del proces */
  }
  return(i);      /* retorna el numero de lletres que ha impres el proces */
}
```

Exercici:

Adaptar la pràctica 2.1 a una primera versió de la pràctica 2.2. Per fer això cal convertir alguns *threads* en processos fill. Cal tenir en compte que els processos fill no podran accedir a l'entorn global de CURSES creat pel procés pare. Així, s'haurà de crear una zona de memòria compartida on estiguin representats tots els objectes de la pantalla. El procés pare s'haurà d'encarregar d'actualitzar periòdicament la visualització del contingut d'aquesta zona de memòria compartida o "pantalla virtual".

IMPORTANT: per poder treballar amb l'entorn de finestres curses des de diferents processos, es disposa d'un conjunt de noves rutines a winsuport2.h/c. Aquestes rutines són equivalents a les utilitzades amb threads, però cal utilitzar-les de forma diferent. Es disposa dels fitxers multiproc2.c i mp_car2.c que adapten els exemples d'aquesta sessió utilitzant les noves rutines. Tots aquests fitxers es troben al directori:
`/usr/local/assignatures/fso/exemples_C/`.