Gestión de procesos

Ejercicio

Implementa un programa anomenat "examCAT" que rebi una llista de fitxers de text com a paràmetre i els mostri per pantalla utilitzant la comanda "cat". El format de la comanda és el següent:

```
> examCAT fitxer1 fitxer2 ... fitxern
```

Entre el bolcat d'un fitxer i el següent, el procés pare ha d'esperar a que es polsi una tecla. Per tant, la comanda "cat" s'ha de cridar tantes vegades com fitxers de text rebem com a paràmetre. Per executar la comanda "cat" utilitzarem la crida "execlp". Els processos necessaris s'han de crear i executar de forma seqüencial. És a dir, en un moment determinat, el procés pare només pot tenir un únic fill.

En el cas que hagin transcorregut 10 segons i encara no s'hagin mostrat tots els fitxers, el programa ha d'escriure el missatge "Temporitzador exhaurit" i avortar la seva execució i la del procés fill (en el cas que existeixi). Per matar al procés fill utilitza la crida al sistema "kill" (veure man 2 kill).

Si el programa es compila amb la constant DEBUG definida, el procés pare haurà de mostrar per pantalla el missatge "Nou procés PID" al crear un nou procés, i el missatge "Acaba el procés PID amb estat X" quan acabi un procés fill (on PID és el PID del procés fill i X és el 'exit code' retornat pel fill).

El programa també ha de comprovar que com a mínim es rebi un fitxer de text com a paràmetre i, en cas contrari, executar una funció Usage() que escriurà una descripció d'ús i acabarà el programa.

```
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
int pid; // Com que necessitem el pid a la funcio de l'alarma, fem global
la variable
void
Usage ()
{
  char buff[128];
  sprintf (buff, "usage:examCAT fich1 [fich2...fichN]\n");
  write (1, buff, strlen (buff));
  exit (0);
}

void
f_alarma (int s)
{
  char buff[128];
  int exit_code;
```

```
  sprintf (buff, "Temps finalitzat\n");
  write (1, buff, strlen (buff));
  kill (pid, SIGKILL);
  waitpid (pid, &exit_code, 0);
#ifdef DEBUG
  sprintf (buff, "Acaba proces %d amb estat %d\n", pid, exit_code);
  write (1, buff, strlen (buff));
#endif
  exit (0);
}

void
main (int argc, char *argv[])
{
  int fitxer;
  char buff[128];
  int exit_code;
  if (argc == 1)
    Usage ();
  signal (SIGALRM, f_alarma);
  alarm (10); // Programem el temporitzador
  for (fitxer = 1; fitxer < argc; fitxer++)
    {
      pid = fork ();
      if (pid == 0)
      {
        signal(SIGALRM,SIG_DFL);
        execlp ("cat", "cat", argv[fitxer], (char *) NULL);
        perror ("Fallo execlp\n");
        exit (1);
      }
      else if (pid < 0)
      {
        perror ("Fallo fork\n");
        exit (1);
      }
      else
      {
#ifdef DEBUG
        sprintf (buff, "Nou process %d\n", pid);
        write (1, buff, strlen (buff));
#endif
        getchar (); // esperem una tecla
        waitpid (pid, &exit_code, 0);
#ifdef DEBUG
        sprintf (buff, "Acaba proces %d amb estat %d\n", pid, exit_code);
        write (1, buff, strlen (buff));
#endif
      }

    }
}
```

Ejercicio

Implementa un programa anomenat "examPS" que executi la comanda "ps" cada "numSeg" segons, on "numSeg" és el primer i únic argument del programa. El format de la comanda és el següent:

```
> examPS numSeg
```

El "ps" ha d'escriure únicament el PID, el PID del pare (PPID), i el nom de l'executable dels processos de l'usuari "alumne". Per a cada "ps" heu de crear un nou procés fill i utilitzar la crida "execlp".

Si el procés pare rep un SIGUSR1 ha de mostrar el seu PID per pantalla. Si rep un SIGUSR2 ha d'escriure el número de vegades que s'ha cridat la comanda "ps" fins al moment.

El procés pare ha de controlar la finalització de cada un dels fills. El programa acaba després d'executar 5 vegades la comanda "ps".

El programa també ha de comprovar que el número d'arguments d'entrada sigui correcte i, en cas contrari, executar una funció Usage() que escriurà una descripció d'ús i acabarà el programa.

Una vegada realitzat i comprovat el codi, executa la comanda "examPS 5" i redirecciona la sortida del procés al fitxer "sortidaExam.txt". Des d'un altre terminal, envia al procés un signal SIGUSR1 i després un SIGUSR2.

```
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
int pss;
int alarma_recibida = 0;
int seg;
void
Usage ()
{
  char buff[128];
  sprintf (buff, "usage:examPS numSeg\n");
  write (1, buff, strlen (buff));
  exit (0);
}

void
f_alarma (int s)
{
  alarma_recibida = 1;
  alarm (seg);
}

void
f_sigusr (int s)
{
  char buff[128];
  if (s == SIGUSR1)
```

```
    {
      sprintf (buff, "Mi PID es %d\n", getpid ());
    }
  else
    {
      sprintf (buff, "Llevamos %d ps's\n", pss);
    }
  write (1, buff, strlen (buff));
}

void
main (int argc, char *argv[])
{
  char buff[128];
  int pid;
  int exit_code;
  if (argc == 1)
    Usage ();
  seg = atoi (argv[1]);
  signal (SIGALRM, f_alarma);
  signal (SIGUSR1, f_sigusr);
  signal (SIGUSR2, f_sigusr);
  alarm (seg);
  for (pss = 0; pss < 5; pss++)
    {
      while (alarma_recibida == 0)
      pause ();         //esperamos que llegue el signal
      alarma_recibida = 0;
      pid = fork ();
      if (pid == 0)
      {
        execlp ("ps", "ps", "-u", "alumne", "-o", "pid,ppid,cmd",
              (char *) NULL);
        perror ("Fallo execlp\n");
        exit (1);
      }
      else if (pid < 0)
      {
        perror ("Fallo fork\n");
        exit (1);
      }
      else
      {
        waitpid (pid, &exit_code, 0);
#ifdef DEBUG
        sprintf (buff, "Acaba proces %d amb estat %d\n", pid, exit_code);
        write (1, buff, strlen (buff));
#endif
      }

    }
}
```

Gestión de memoria

Ejercicio

En la siguiente URL docencia.ac.upc.edu/FIB/grau/SO/ControlT3.tar.gz tenéis el fichero ControlT3.tar.gz para realizar estos ejercicios. Cópialo, descomprímelo y compila los ficheros. Tenéis que realizar los siguientes ejercicios y en cada caso anotar las respuestas en un fichero de texto aparte (la respuesta a las preguntas y los comandos que has usado para obtenerlas)

1. Usando el programa ej1 (no necesita parámetros).
    - Ejecuta el programa y comprueba cuales las principales regiones de memoria antes y después del execlp tanto para el proceso PADRE como el HIJO. Calcula el tamaño del heap y de la región de código en cada caso. Contesta:
        - Comando que has usado para comprobar las regiones
        - Regiones obtenidas (puedes copiar&pegar)
        - Comenta a que son debidas las diferencias obtenidas
2. Usando el programa ej2 (parámetros= region_size y número de iteraciones).
    - Calcula el máximo de memoria que puede pedir un proceso (aproximado), usando como parámetros region_size=409600 y 10000 iteraciones. Comprueba si se está activando el mecanismo de swap. Para cada pregunta apunta el comando utilizado.
        - ¿Con qué comando compruebas si se están realizando operaciones de swap-in o swap-out?
        - ¿A partir de qué cantidad de memoria libre se dispara el mecanismo de swap con 1 proceso?
    - Si pones 2 procesos a la vez (lánzalos en background), ¿se pone en funcionamiento el mecanismo de swap?
    - El programa muestra el tiempo de acceso al bucle (access loop time)
        - ¿Qué diferencia en los tiempos de acceso a bucles que muestra el programa hay antes y después de que se active el mecanismo de swap?
        - ¿Cómo afecta a la cantidad de cambios de contexto por segundo? ¿Con qué comando lo has comprobado?
    - ¿Qué comando usarías para ver el % de CPU que consume cada proceso en el sistema?
        - Con el comando anterior también puedes ver el nombre de los comandos, ¿Qué nombre recibe el proceso que crees que se encarga de hacer swap?
    - Modifica el programa de forma que se libere la memoria de una región antes de pedir la siguiente.
        - Copia el trozo de código del bucle que has modificado en el fichero de texto a entregar
        - ¿Cuántos procesos podemos poner ahora simultáneamente sin que fallen?
        - ¿Cuántas iteraciones pueden ejecutar sin fallar? ¿Hay algún límite?
        - ¿Qué tiempos de bucles presentan ahora los procesos? ¿Depende de cuantos procesos hay?

Ejercicio

El fitxer "examen.c" conté el codi font d'un programa que reserva un bloc de memòria de 4KB i l'inicialitza (bucle 1). Una vegada inicialitzat el bloc, el programa crea un procés fill i, tant el pare com el fill, fan dos recorreguts sobre aquest bloc (bucles 2 i 3). El programa també mesura el temps que triguen en executar-se cada un dels 3 bucles.

Es demana que contesteu les següents preguntes en un fitxer de text (respostes.txt) i entregueu una versió modificada del fitxer "examen.c" amb els canvis que es demanen a continuació:

1) Compila el codi enllaçant-lo amb llibreries estàtiques. Quina comanda has utilitzat? Contesta els següents apartats compilant el codi amb aquesta comanda.

2) Modifica el codi perquè imprimeixi per pantalla, tant per al procés pare com per al fill, les adreces que ocupen en memòria les variables "i", "region_size", "tmp" i el bloc de 4KB. Quines adreces ocupen? Les adreces són les mateixes en el procés pare i en el fill? Raona la teva resposta.

3) Executa la comanda "nm" i busca les variables "i", "region_size" i "tmp". Hi apareixen les dues variables? Per què?

4) Pel procés pare, localitza a quina regió pertanyen les variables "i", "region_size", "tmp" i el bloc de 4KB. Per a cada una, indica el nom de la regió, i l'adreça incial i final de la regió.

5) Executa el programa "examen" perquè reservi un bloc de 4KB. Quina és la mida total del heap del procés pare? Com ho has calculat? És més gran, igual o més petit de 4KB. Per què?

6) Modifica el codi perquè després del getchar() s'alliberi el bloc de memòria reservat. Qui ha d'alliberar el bloc, el procés pare, el fill o els dos? Raona la teva resposta.

7) El bucle 1 l'executa només el procés pare, mentre que els bucles 2 i 3 s'executen tant en el pare com en el fill. Apunta quan triga en executar-se cada bucle en el procés pare i en el fill. Quin bucle triga més en executar-se, el bucle 1 o el 2? Per què?

8) Analitza el codi del bucle 3 en el pare i en el fill. Per què el bucle 3 triga a executar-se més en el fill que en el pare? A quina propietat és degut aquest comportament? Raona la teva resposta.

9) El programa "examen" accepta un argument per la línia de comandes que correspon a la inicialització de la variable "region_size". Si aquest argument no es proporciona, per defecte s'inicialitza a 4KB. Prova d'executar el programa de manera que creï un bloc de 1MB. Quin error es produeix? Per què?

10) Escriu una rutina d'atenció al SIGSEGV que escrigui per pantalla l'adreça de l'accés a p[i] que ha produït aquest signal i acabi el programa. Executa el programa amb "region_size" igual a 1MB i indica l'adreça que produeix el SIGSEGV.

11) Modifica la crida a malloc perquè en comptes de reservar 4KB reservi "region_size" bytes. Executa ara el programa perquè creï un bloc de F/2 bytes, on F és la quantitat de memòria física instal·lada a la teva màquina? Quanta memòria hi ha instal·lada (en bytes)? Quina comanda s'ha d'utilitzar per obtenir aquest valor en bytes?

12) Executa el programa amb els mateixos paràmetres que en l'apartat anterior i mesura el número de "swap-in" i "swap-out" que es produeixen durant la seva execució. Quina comanda has utilitzat? Amb quines opcions?

13) Comptabilitza el número de fallades de pàgina que ha experimentat el procés pare i el fill. Com has obtingut aquest valor? El resultat és diferent a l'observat a l'apartat anterior? Per què?

Gestión Entrada/Salida

Ejercicio

1) Fes un programa (exam_1.c) que creï un procés fill que executi la comanda /bin/cat (sense cap paràmetre d'entrada). Per defecte, el programa 'cat' copia el que rep per l'entrada estàndard a la sortida estàndard. El procés pare ha de rebre la sortida del programa 'cat' a través d'una pipe sense nom i comptar el número de caràcters rebuts. En acabar, el procés pare ha d'escriure per pantalla el missatge "S'han rebut X caracters del proces fill", on X és el número total de caràcters que el procés fill ha enviat al pare a través de la pipe.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
void
error (char *msg)
{
  perror (msg);
  exit (0);
}

void
main (int argc, char argv[])
{
  int fd[2];
  int pid, numc = 0;
  char c;
  char buff[128];
  if (pipe (fd) < 0)
    error ("pipe");
  pid = fork ();
  if (pid < 0)
    error ("fork");
  if (pid == 0)
    {
      close (fd[0]);
      dup2 (fd[1], 1);
      close (fd[1]);
```

```
      execlp ("cat", "cat", (char *) 0);
      error ("execlp");
    }
  close (fd[1]);
  while (read (fd[0], &c, sizeof (c)) > 0)
    numc++;
  sprintf (buff, "S'han rebut %d caracters\n", numc);
  write (1, buff, strlen (buff));
  waitpid (-1, NULL, 0);
}
2)
```

3) El fitxer enunciat_t4.tar.gz conté el codi del controlador de dispositiu myDriver1.c vist a la sessió 8. Compila el controlador executant la comanda make i executa el script d'instal·lació installDrivers.sh. Crea a continuació un nou dispositiu (myDevice) de tipus caràcter gestionat per MyDriver1. Escriu al fitxer "respostes.txt" la comanda que has utilitzat.

4) Modifica el programa exam_1 (anomena'l exam_1_disp.c) de manera que ara el programa 'cat' llegeixi la cadena de caràcters des del nou dispositiu MyDevice creat en l'apartat anterior en comptes de fer-ho pel teclat.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void
error (char *msg)
{
  perror (msg);
  exit (0);
}

void
main (int argc, char argv[])
{
  int fd[2], fich;
  int pid, numc = 0;
  char c;
  char buff[128];
  if (pipe (fd) < 0)
    error ("pipe");
  pid = fork ();
  if (pid < 0)
    error ("fork");
  if (pid == 0)
    {
      close (fd[0]);
      dup2 (fd[1], 1);
      close (fd[1]);
      close (0);
      fich = open ("MyDevice", O_RDONLY);
```

```
      if (fich != 0)
      error ("open");
      execlp ("cat", "cat", (char *) 0);
      error ("execlp");
    }
  close (fd[1]);
  while (read (fd[0], &c, sizeof (c)) > 0)
    numc++;
  sprintf (buff, "S'han rebut %d caracters\n", numc);
  write (1, buff, strlen (buff));
  waitpid (-1, NULL, 0);
}
```

Ejercicio

5) Escriu un programa (exam_2.c) que llegeixi per l'entrada estàndard una seqüència de números en format text separats per salts de línia (\n). Cada número s'ha de convertir primer a enter, utilitzant la funció atoi(), i a continuació s'ha d'escriure per la sortida estàndard en format binari (no s'ha d'escriure cap separador entre número i número). El programa acaba quan es rep l'EOF per l'entrada estàndard. Podeu suposar que per l'entrada estàndard només es rebran caràcters numèrics o el '\n'.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int
leer_numero ()
{
  int i = 0, ret;
  char num[16];
  char c;
  ret = read (0, &c, sizeof (c));
  while ((ret > 0) && (c != '\n'))
    {
      num[i] = c;
      i++;
      ret = read (0, &c, sizeof (c));
    }
  if (i == 0)
    return -1;
  num[i] = '\0';
  return atoi (num);
}



void
main (int argc, char *argv[])
{
  int num;
  while ((num = leer_numero ()) != -1)
    {
      write (1, &num, sizeof (num));
    }
}
```

6) Fes un programa (exam_3.c) que creï un procés fill que executi el programa exam_2. El pare i el fill s'han de comunicar a través d'una pipe sense nom. El procés pare rebrà la seqüència de números enters del procés fill a través de la pipe, i escriurà per pantalla el missatge "La suma es: X", on X és la suma de la seqüència de números.

```
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
void
error (char *msg)
{
  perror (msg);
  exit(0);
}

void
main (int agrc, char *argv[])
{
  int fd[2], pid, exit_code, ret;
  char buff[128];
  int num, total;
  if (pipe (fd) < 0)
    error ("pipe");
  pid = fork ();
  if (pid < 0)
    error ("fork");
  if (pid == 0)
    {
      close (fd[0]);
      dup2 (fd[1], 1);
      close (fd[1]);
      execlp ("./exam_2", "exam_2", (char *) NULL);
      error ("execlp");
    }
  total = 0;
  close (fd[1]);
  while (read (fd[0], &num, sizeof (int)) > 0)
    total = total + num;
  close (fd[1]);
  sprintf (buff, "La suma total es %d\n", total);
  write (1, buff, strlen (buff));
}
```

Sistemas de Ficheros

Ejercicios

Problema 1 (7 punts)

Fes un programa (exam_4.c) que donat un fitxer d'entrada en format text, generi un fitxer de sortida on, per a cada paraula del fitxer d'entrada, n'inverteixi els seus caràcters. La

posició de les paraules en el fitxer de sortida ha de ser el mateix que en el fitxer d'entrada. Per exemple, si el contingut del fitxer d'entrada és el següent:

```
"control de laboratori de SO"
```

el programa hauria de generar el següent fitxer de sortida:

```
"lortnoc ed irotarobal ed OS"
```

Definim una paraula com a una seqüència de caràcters consecutius acabada amb un espai o caràcter '\n'. Podem suposar que no hi haurà més d'un espai o caràcter '\n' seguits. El nom del fitxer d'entrada és el primer argument del programa. El fitxer de sortida ha de tenir el mateix nom que el fitxer d'entrada però afegint-hi l'extensió ".inv". Si el fitxer de sortida ja existia, el programa l'ha de sobreescriure. En cas contrari, el fitxer s'ha de crear amb permisos de lectura i escriptura pel propietari del fitxer i només de lectura per a la resta d'usuaris.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#define END 0
#define ESPACIO 1
#define CR 2

int fd_in, fd_out;

int
leer_palabra (char *p)
{
      int i=0,ret;
      char c;
      ret=read(fd_in,&c,sizeof(char));
      while((ret>0) && (c!=' ') && (c!='\n')){
            p[i]=c;
            i++;
            ret=read(fd_in,&c,sizeof(char));
      }
      p[i]='\0';
      if (i==0) return END;
      else if (c==' ') return ESPACIO;
      else return CR;

}

void
invertir_palabra (char *p1, char *p2)
{
      int s;
      int i=0;
      s=strlen(p1)-1;
      while(s>=0){
            p2[i]=p1[s];
            i++;s--;
```

```
        }
        p2[i]='\0';
}

void
escribir_palabra (char *p,int extra)
{
        write(fd_out,p,strlen(p));
        if (extra==CR) write(fd_out,"\n",1);
        else write(fd_out," ",1);
}

void
error (char *msg)
{
  perror (msg);
  exit (0);
}

void
usage ()
{
  char msg[128];
  sprintf (msg, "./exam_4 fitxer_input\n");
  write (1, msg, strlen (msg));
  exit (0);
}

void
main (int argc, char *argv[])
{
  char palabra[128];
  char palabra_inv[128];
  char file_out[128];
  int extra;
  if (argc == 1)
    usage ();
  if ((fd_in = open (argv[1], O_RDONLY)) < 0)
    error ("open input");
  sprintf (file_out, "%s.inv", argv[1]);
  if ((fd_out = open (file_out, O_WRONLY | O_CREAT | O_TRUNC,0640)) < 0)
    error ("open output");
  while ((extra=leer_palabra (palabra)) > 0)
    {
      invertir_palabra (palabra, palabra_inv);
      escribir_palabra (palabra_inv,extra);
    }
}
```

Problema 2 (3 punts)

Crea el directori "problema2" dins del directori $HOME. Situat dins del directori "problema2" i crea un fitxer de text anomenat "fitxer.txt" que contingui el valor "12345". A continuació, contesta les següents preguntes en el fitxer "respostes.txt":

1) Crea un soft link anomenat "link1" que apunti a "fitxer.txt". Quina comanda has utilitzat?

2) Crea un hard link anomenat "link2" que apunti a "fitxer.txt". Quina comanda has utilitzat?

3) Quin és el valor del comptador de referències de "fitxer.txt" i de "link2"? Quina comanda has utilitzat per obtenir-los? Raona els valors obtinguts?

4) Consulta el número d'inodes lliures en el sistema de fitxers que conté el directori "problema2". Quina comanda has utilitzat? Quants inodes creus que s'han creat com a conseqüència de les comandes executades en els apartats anteriors?

5) Esborra el fitxer "fitxer.txt". Observa ara els continguts de "link1" i "link2". Explica el motiu de les diferències observades. Després d'esborrar el fitxer canvia el número d'inodes lliures? Per què?

Ejercicios

Exercici 1 (7 punts)

Escriu un programa anomenat **lletra.c** que rebi com a argument un nom de fitxer i un valor enter. En ser executat, el procés escriurà per la sortida estàndard el caràcter del fitxer que ocupi la posició indicada pel valor enter, i acaba. Si el valor enter fa referència a una posició que no existeix, aleshores no escriu res.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>


void
error (char *msg)
{
  perror (msg);
  exit (0);
}

void
usage ()
{
  char msg[128];
  sprintf (msg, "./lletra fitxer posicio\n");
  write (1, msg, strlen (msg));
  exit (0);
}

void
main (int argc, char *argv[])
{
  char c;
  int fd_in,posicio,ret;
```

```
  posicio=atoi(argv[2]);
  if (argc !=3)
    usage ();
  if ((fd_in = open (argv[1], O_RDONLY)) < 0)
    error ("open input");
  ret=lseek(fd_in,posicio,SEEK_SET);
  if (ret!=posicio) exit(0);
  ret=read(fd_in,&c,sizeof(char));
  if (ret<0) error("read");
  write(1,&c,sizeof(char));
}
```

Escriu ara un programa anomenat **invert.c** que rebi com a argument el nom d'un fitxer. En ser executat, ha de generar un fitxer de sortida amb el contingut del fitxer original però invertit. El nom del fitxer creat ha de ser el mateix que l'original i acabat amb .inv.

Per realitzar aquesta tasca no podeu utilitzar cap buffer ni vector en el que guardar el contingut del fitxer original si no que s'ha d'usar repetidament el programa lletra.c de l'apartat anterior. Suggeriment: no cal usar pipes; n'hi ha prou amb redireccionar la sortida estàndard dels processos fills al fitxer nou.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>


void
error (char *msg)
{
  perror (msg);
  exit (0);
}

void
usage ()
{
  char msg[128];
  sprintf (msg, "./invert fitxer\n");
  write (1, msg, strlen (msg));
  exit (0);
}

void
main (int argc, char *argv[])
{
  char c;
  int fd_in, posicio, ret, fd_out;
  int pid;
  char nom[128], posc[128];
  if (argc != 2)
    usage ();
  if ((fd_in=open(argv[1],O_RDONLY))<0) error("input file");
  sprintf (nom, "%s.inv", argv[1]);
  close (1);
```

```
  if ((fd_out = open (nom, O_WRONLY | O_CREAT | O_TRUNC, 0640)) < 0)
    error ("out");
  posicio = lseek (fd_in, 0, SEEK_END);
  close(fd_in);
  while (posicio >= 0)
    {
      pid = fork ();
      if (pid < 0)
      error ("fork");
      if (pid == 0)
      {
        sprintf (posc, "%d", posicio);
        execlp ("./lletra", "lletra", argv[1], posc, (char *) NULL);
        error ("execlp");
      }
      posicio--;
      waitpid (pid, NULL, 0);
    }
}
```

Exercici 2 (3 punts)

Crea un directori anomenat **exercici2** dins del directori $HOME. Situat dins del directori exercici2 i crea un fitxer de text anomenat **fitxer.txt** amb el contingut "12345". A continuació, contesta les preguntes següents en el fitxer **respostes.txt**:

1. Crea un soft link anomenat "link1" que apunti a "fitxer.txt". Quina comanda has utilitzat? Crea un hard link anomenat "link2" que apunti a "fitxer.txt". Quina comanda has utilitzat?
2. Quin és el valor del comptador de referències de "fitxer.txt" i de "link2"? Quina comanda has utilitzat per obtenir-los? Raona els valors obtinguts.
3. Consulta el número d'inodes lliures en el sistema de fitxers que conté el directori "exercici2". Quina comanda has utilitzat?
4. Quants inodes creus que s'han creat com a conseqüència de les comandes executades en els apartats 1 i 2?
5. Esborra el fitxer "fitxer.txt". Observa ara els continguts de "link1" i "link2". Explica el motiu de les diferències observades.
6. Després d'esborrar el fitxer canvia el número d'inodes lliures? Per què?