Presentació

Aquesta pràctica planteja un seguit d'activitats amb l'objectiu que l'estudiant pugui aplicar sobre un sistema Unix alguns dels conceptes introduïts als darrers mòduls de l'assignatura.

L'estudiant haurà de realitzar un seguit d'experiments i respondre les preguntes plantejades. També haurà d'escriure un programa en llenguatge C.

Competències

Transversals:

- Capacitat per a adaptar-se a les tecnologies i als futurs entorns actualitzant les competències professionals

Específiques:

- Capacitat per a analitzar un problema en el nivell d'abstracció adequat a cada situació i aplicar les habilitats i coneixements adquirits per a abordar-lo i resoldre'l
- Capacitat per a dissenyar i construir aplicacions informàtiques mitjançant tècniques de desenvolupament, integració i reutilització

Enunciat

Per a fer aquesta pràctica us facilitem el fitxer `pr2so.zip` amb fitxers font. Descompacteu-lo amb la comanda `unzip pr2so.zip`. Per compilar un fitxer, per exemple `prog.c`, cal executar la comanda `gcc -o prog prog.c`

1. (6 punts, 1 punt per apartat) Donats els següents codis, indiqueu justificadament quin és el resultat d'executar-los. Indiqueu quins paràmetres esperen, la jerarquia de processos creada, quins missatges (i en quin ordre) es mostren pels dispositius d'entrada/sortida, bloquejos,...

   S'aconsella que en primer terme intenteu determinar el seu comportament sense executar-los al terminal. Executeu-los per a verificar la vostra resposta. Podeu assumir que cap crida al sistema tornarà error.

   No heu de *corregir* els codis. Només es demana que expliqueu el seu comportament.

   (a) Codi 1:

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main (int argc, char *argv[])
{
  int i;
```

```
  if (argc < 3)
    exit (0);

  for (i = 0; i < argc; i++)
    {
      write (1, argv[i], strlen (argv[i]));
      write (1, "␣", 1);
    }
  write (1, "\n", 1);

  sleep (atoi (argv[1]));

  execvp (argv[2], &argv[2]);
  execvp (argv[2], &argv[2]);
  exit (0);
}
```

El codi necessita com a mínim 2 paràmetres. Per l'ús que en fa el codi, s'espera que el primer sigui un nombre i el segon un nom de fitxer executable; si el codi rep més de 2 paràmetres, aquests s'utilitzaran com a paràmetres de l'executable indicat com a segon paràmetre.

El codi imprimeix tots els paràmetres que ha rebut (incloent argv[0]) a la mateixa línia (separats per un espai en blanc).

A continuació, fa una espera de tants segons com indiqui el primer paràmetre. Finalment, fa un canvi d'imatge (execv) passant a executar l'executable especificat al segon paràmetre amb els paràmetres que s'hagin especificat com a tercer, quart,... paràmetres del codi.

El segon canvi d'imatge no té efecte perquè el primer canvi d'imatge (assumint que no retorni error) substitueix el programa actual per un altre.

Per tant, el codi no crea cap nou procés, només fa un canvi d'imatge.

Per la sortida estàndar apareixerà el resultat d'executar l'executable amb els paràmetres indicats.

En global, aquest codi executa un programa amb els paràmetres especificats després d'esperar una determinat nombre de segons.

Exemple: `./code1 5 ls -l -a /tmp` executaria `ls -l -a /tmp` després d'esperar 5 segons.

(b) Codi 2:

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main (int argc, char *argv[])
{
  int i;

  for (i = 1; i < argc; i++)
    {
      if (fork () == 0)
```

```
        execlp (argv[1], argv[1], argv[i], NULL);
      wait (NULL);
    }
  exit (0);
}
```

Si el codi rep algún paràmetre, s'espera que el primer paràmetre sigui un nom de fitxer executable i la resta siguin paràmetres d'aquest executable.

El codi entra en un bucle on a cada iteració es crearà un procés fill. El codi passarà a la següent iteració quan es detecti la mort del fill. En total crearà tants fills com paràmetres rebuts (argc-1, sense contar argv[0]).

Cada fill executa l'executable indicat per argv[1], passant-li com a paràmetre argv[i], des de i=1 fins a argc-1 (per tant, el primer fill passa com a paràmetre a l'executable el propi nom de l'executable).

Exemple: `./code2 ls -l -a` crearia tres fills, que executarien seqüencialment `ls ls`, `ls -l` i `ls -a`. Per la sortida estàndar apareixerà el resultat d'aquestes tres execucions en aquest ordre.

(c) Codi 3:

```
#include <unistd.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  if (argc < 3)
    exit (1);

  sleep (atoi (argv[argc - 1]));

  argv[argc - 1] = NULL;
  execvp (argv[1], &argv[1]);
  exit (0);
}
```

Aquest codi és similar al code1 però canvia l'ordre com espera els paràmetres.

El nombre de segons és el darrer paràmetre rebut (el argc-1). El nom del fitxer executable a executar és argv[1] i els seus paràmetres estaran entre argv[2] i argv[argc-2].

Per evitar que el nou executable rebi com a paràmetre el nombre de segons, abans de fer el canvi d'imatge amb execv cal "eliminar"el paràmetre que representa el nombre de segons. Per fer-ho, sobreescrivim la posició argc-1 del vector argv amb un NULL (la marca que indica que no hi ha més paràmetres).

Per tant, un cop esperat el temps indicat, es sobreescriu la posició addient d'argv i es fa un canvi d'imatge a un nou executable amb els

paràmetres indicats.

Exemple: `./code3 ls -l -a /tmp 5` executaria `ls -l -a /tmp` després d'esperar 5 segons.

(d) Codi 4:

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main (int argc, char *argv[])
{
  int fd[2];
  char c;

  pipe (fd);

  if (fork () == 0)
    {
      while (read (fd[0], &c, 1) > 0)
        write (1, &c, 1);
    }
  else
    {
      write (fd[1], argv[0], strlen (argv[0]));
      wait (NULL);
    }
  exit (0);
}
```

El codi inicial crea una pipe i un procés fill que hereda l'accés a la pipe creada per procés inicial.

El procés fill entra en un bucle de lectura de la pipe i escriptura per la sortida estàndar fins a detectar final de fitxer de la pipe. El procés inicial escriu a la pipe el seu nom (argv[0]) i espera la mort del fill.

Per tant, el fill anirà llegint de la pipe els caràcters que composen el nom del procés inicial i els escriurà per la sortida estàndar. Ara bé, mai detectarà final de fitxer perquè es mantenen oberts dos canals d'escriptura sobre la pipe (tant el del pare com el del fill); per tant, quan el fill intenti llegir de la pipe i aquesta sigui buida, el fill es quedarà bloquejat. Per un altre costat, el pare també es quedarà bloquejat esperant la mort del fill (que no es produirà perquè està bloquejat llegint de la pipe).

En global, el codi escriu per la sortida estàndar el seu nom però no terminarà mai.

(e) Codi 5:

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
```

```c
#include <sys/wait.h>

int main (int argc, char *argv[])
{
  int fd[2];
  char c;

  if (fork () == 0)
    {
      pipe (fd);
      close (fd[1]);
      while (read (fd[0], &c, 1) > 0)
        write (1, &c, 1);
    }
  else
    {
      pipe (fd);
      write (fd[1], argv[0], strlen (argv[0]));
      close (fd[1]);
      wait (NULL);
    }
  exit (0);
}
```

El codi crea un fill.

El procés inicial crea una pipe, hi escriu el seu nom, tanca el canal d'escriptura a la pipe i espera la mort del fill creat.

El procés fill crea una altra pipe (independent de la creada pel procés inicial) tanca el canal d'escriptura a la seva pipe i entra en un bucle de lectura de la pipe. A la primera lectura ens trobem que la pipe és buida i que no hi ha cap procés en condicions d'escriure sobre la pipe (el fill acaba de tancar l'únic canal d'escriptura que hi existia). Per tant, el fill no completarà cap iteració del bucle, sortirà de l'if, executarà exit(0) i morirà.

Quan el procés inicial detecti la mort del fill, també sortirà de l'if, executarà exit(0) i morirà.

Per tant, el codi finalitza sense haver escrit res per la sortida estàndar.

(f) Codi 6:

```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

int main (int argc, char *argv[])
{
  int fd[2];

  if (argc != 4)
    exit (1);

  pipe (fd);
  if (fork () == 0)
    {
```

```
      dup2 (fd[1], 1);
      execlp (argv[1], argv[1], NULL);
    }
  else
    {
      close (fd[1]);
      dup2 (fd[0], 0);

      close (1);
      open (argv[3], O_WRONLY | O_TRUNC | O_CREAT, 0600);

      execlp (argv[2], argv[2], NULL);
      wait (NULL);
    }
  exit (0);
}
```

El codi espera rebre tres paràmetres; altrament, finalitza immediatamente. El primer i el segon paràmetre s'interpretaran com a noms de fitxers executables; el tercer com el nom del fitxer on deixar el resultat de l'execució.

El programa crea una pipe i un procés fill. El procés fill redirecciona la sortida estàndar a la pipe i executarà el programa indicat com a primer paràmetre. El procés original tanca el canal d'escriptura sobre la pipe, redirecciona la seva entrada estàndar al canal de lectura de la pipe i la sortida estàndar al fitxer indicat com a tercer paràmetre; finalment, executa el programa indicat con a segon paràmetre. Si l'execlp no retorna error, no s'executarà cap sentència més del codi 6.

Per tant, el codi utiliza una pipe per a comunicar dos processos. El resultat de l'execució s'escriurà sobre un fitxer. El procés original podrà detectar final de fitxer llegint de la pipe quan el proces fill hagi mort i la pipe sigui buida perquè l'únic canal d'escriptura sobre la pipe el tenia obert el procés fill.

Exemple: `./code6 ls wc out.txt` seria equivalent a executar `ls | wc > out.txt`

2. (4 punts)

   Escriviu un programa en llenguatge C que creï `N` processos fills que executaran concurrentment el programa `child.c` (us el donem implementat i no heu de modificar-lo). Si analitzeu el codi de `child.c`, veureu que cada fill es bloquejarà per un temps aleatori i finalitzarà amb un codi d'acabament (*exit code*) igual al nombre de segons que ha estat bloquejat.

   Un cop creats els `N` fills, el pare esperarà la mort de tots els fills i n'examinara el codi d'acabament a mesura que finalitzin. Sempre que el pare trobi un fill amb un codi d'acabament més gran que 5, el pare crearà un nou fill que també executarà `child.c` i del que també n'examinarà el codi d'acabament.

   El pare finalitzarà quan tots els fills (els `N` inicials i els que s'hagin creat

addicionalment) hagin finalitzat.

Us adjuntem un possible resultat de l'execució del programa.

```
[enricm@willy dev]$  ./father 3
Father creates child 0 with pid=11471
Father creates child 1 with pid=11472
Father creates child 2 with pid=11473
    Child 1 ends with exit code 5
Father detects death of child 1 with exit code 5
    Child 2 ends with exit code 10
Father detects death of child 2 with exit code 10
Father recreates child process
Father creates child 3 with pid=11478
    Child 0 ends with exit code 10
Father detects death of child 0 with exit code 10
Father recreates child process
Father creates child 4 with pid=11479
    Child 4 ends with exit code 4
Father detects death of child 4 with exit code 4
    Child 3 ends with exit code 10
Father detects death of child 3 with exit code 10
Father recreates child process
Father creates child 5 with pid=11481
    Child 5 ends with exit code 1
Father detects death of child 5 with exit code 1
Father ends
[enricm@willy dev]$
```

Observacions:

- El nombre `N` es rebrà per la línia d'ordres. No existeix un límit per al valor de `N`.
- Cal que el vostre procés pare mostri la mateixa informació que mostra l'exemple.
- S'aconsella que dividiu el programa en parts i que no comenceu una nova part fins que l'anterior funcioni.
- No podeu utilitzar la rutina `system`.
- És precís fer el tractament d'errors a les crides al sistema.

```c
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>

struct child_info {
  int valid;
  int pid;
  int order;
} *children;

void panic(char *s)
{
  fprintf(stderr, "ERROR: %s (%d %s)\n",
        s, errno, strerror(errno));
  exit(1);
}

void create_child(int table_pos, int order)
{
    char str[5];
```

```
    children[table_pos].valid = 1;
    children[table_pos].order = order;
    sprintf(str, "%d", order);
    switch(children[table_pos].pid = fork()) {
      case -1: panic ("fork");
      case  0: execlp("./child", "child", str, NULL);
               panic("exec child");
    }
    printf("Father creates child %d with pid=%d\n",
        order, children[table_pos].pid);
}

int main(int argc, char *argv[])
{
  int table_entries, pending_children, child_id;

  if (argc!=2) panic("Missing argument");
  table_entries = pending_children = atoi(argv[1]);

  children = malloc(table_entries*sizeof(struct child_info));
  if (children == NULL) panic("Out of memory");

  /* Create first N child processes */
  for (child_id=0; child_id<table_entries; child_id++)
    create_child(child_id, child_id); // At first, child_id and table_pos are the same

  while (pending_children>0) {
    int pidchild, st, j, exitcode;

    pidchild = wait(&st);
    if (pidchild==-1) panic("wait");

    for (j=0; j<table_entries; j++)
      if ((children[j].valid == 1) &&
        (children[j].pid == pidchild)) break;

    if (j==table_entries) panic("Unexpected child");
    if (WIFEXITED(st)==0) panic("Unexpected child end");

    exitcode = WEXITSTATUS(st);

    printf("Father detects death of child %d with exit code %d\n",
        children[j].order, exitcode);

    if (exitcode > 5) {
      printf("Father recreates child process\n");
      create_child(j, child_id); // New child re-uses previous table-entry
      child_id++;
    }
    else {
      children[j].valid = 0; // Invalidate table entry
      pending_children--;    // Decrement number of pending children
    }
  }
  printf("Father ends\n");
  free(children);
}
```

Recursos

- Mòduls 1, 2, 3, 4, 5 i 6 de l'assignatura.
- Document "Introducció a la programació de UNIX" (disponible a l'aula) o qualsevol altre manual similar.
- Document "Intèrpret de comandes UNIX" (disponible a l'aula) o qualsevol altre manual similar.
- L'aula "Laboratori de Sistemes Operatius" (podeu plantejar els vostres dubtes relatius a l'entorn Unix, programació,...).

Criteris d'avaluació

Es valorarà la justificació de les respostes presentades. S'agrairan les respostes breus i concises.

El pes de cada pregunta està indicat a l'enunciat.

Format i data de lliurament

Es lliurarà un fitxer `zip` que contingui un fitxer tots els fitxers del lliurament (un `pdf` amb la resposta a les preguntes i fitxers font).

Data límit de lliurament: 24:00 del 9 de desembre de 2019.