ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

PRA1. Processos, *pipes* i senyals: EPSlotto

Publicació: Divendres 07/OCT./2022
Entrega: Divendres, 28/OCT./2022

1. Objectius

Els objectius de la pràctica són:

- Comprensió crides a Sistema UNIX.
- Conèixer les funcions de creació de processos *fork* i *exec*. Entendre com funcionen, com es criden, què retornen.
- Conèixer els mètodes de sincronització de processos *exit* i *wait* i la seva relació.
- Veure els *pipes* com a mecanismes de comunicació i sincronització entre processos.
- Treball amb els senyals entre processos.

2. Presentació

MOLT IMPORTANT: L'enviament de codi que no compili correctament, suposarà suspendre TOTA la pràctica.

No s'acceptaran pràctiques entregades fora del termini.

La presentació d'aquests exercicis és **obligatòria** i pot realitzar-se per parelles.

Cal presentar els fitxers amb el codi font realitzat. Tota la codificació es farà exclusivament en llenguatge C.

La data límit serà les **14:50 h** del **divendres 28/OCTUBRE/2022**, dia en el qual **es farà una prova de validació INDIVIDUAL I PRESENCIAL AL LABORATORI**, que podrà ser oral o un test a través del Campus Virtual.

Per presentar la pràctica adreceu-vos a l'apartat Activitats del Campus Virtual a l'assignatura de Sistemes Operatius, aneu a l'activitat **PRA1. Processos, *pipes* i senyals: EPSlotto – Pralab 2** i seguiu les instruccions.

TOTA LA PARELLA ha de pujar al Campus Virtual un fitxer agrupat i comprimit amb tar, que contingui el codi font dels vostres programes DEGUDAMENT COMENTAT i una memòria amb els aspectes que considereu importants. Per generar aquest fitxer podeu usar l'ordre `tar -zcvf COGNOM1_1_COGNOM1_2_PRA1_SO.tgz <llista_fitxers>`.

El nom del fitxer comprimit ha de ser: COGNOM1_1_COGNOM1_2_PRA1_SO.tgz on:

- COGNOM1_1: el 1r cognom d'un membre del grup.
- COGNOM1_2: el 1r cognom de l'altre membre del grup.

Comenceu els vostres programes amb els comentaris:

```
/* ------------------------------------------------------------------
PRA1: EPSlotto
Codi font: <nom>.c

Nom complet autor1.
Nom complet autor2.
------------------------------------------------------------------ */
```

3 Restriccions en l'ús de funcions C

Per realitzar la pràctica heu d'utilitzar les crides a sistema d'Unix. **No es podran utilitzar funcions de C d'entrada/sortida** (`getc`, `scanf`, `printf`...), en el seu lloc s'utilitzaran les crides a sistema `read` i `write`. Sí que podeu utilitzar funcions de C per al formatatge de cadenes (`sprintf`, `sscanf`...).

4 Enunciat

La primera pràctica consta de dos exercicis: el primer, de caràcter més teòric, perquè treballeu les bases de les funcions *fork* i *exec*, i al segon programareu una aplicació concurrent i usant mecanismes de comunicació.

4.1 Programació multiprocés: *fork* i *exec*

Responeu les següents preguntes de **forma justificada**:

a) Explica de forma breu i aclaridora que fa la funció *fork()*.

b) Quin tipus de dades retorna una crida a la funció de creació de processos *fork()*?

□ pid_t □ int □ char

c) A quin tipus de dades fonamental equival?

d) Pot la funció *fork()* retornar el valor -125? I el valor -1?

e) Quines diferències hi ha amb el grup de funcions *execl()*.

f) Col·loca els tres blocs de codi següents al forat corresponent del programa que hi ha a continuació (al quadre en blanc que correspongui). Després compila i executa el programa i digues quins missatges s'imprimeixen per la sortida estàndard. **Justifica la resposta**:

- Bloc de codi 1:

```
sprintf(cadena, "%s[%d] PROCES PARE suspen execucio en espera de la finalitzacio del fill %d ... %s\n", color_blue, getpid(), pid, color_end);
write(1, cadena, strlen(cadena));

if((fpid=wait(&estat)) == -1)
{
	sprintf(cadena, "%s[%d] PROCES PARE no te fills ... %s\n", color_blue, getpid(), color_end);
	write(1, cadena, strlen(cadena));
	exit(0);
}

sprintf(cadena, "%s[%d] PROCES PARE ha rebut acabament proces fill [%d] amb codi [%d] i acaba. %s\n\n", color_blue, getpid(), fpid, WEXITSTATUS(estat), color_end);
write(1, cadena, strlen(cadena));

exit(0);
```

- Bloc de codi 2:

```
sprintf(cadena, "\t%s[%d] ***** Iniciant proces fill de %d ... ***** %s\n", color_red, getpid(), getppid(), color_end);
write(1, cadena, strlen(cadena));
llavor = (unsigned int) time(NULL);
sprintf(cadena, "\t%s[%d] Estableix llavor [%d]%s\n", color_red, getpid(), llavor, color_end);
write(1, cadena, strlen(cadena));
srand(llavor);      //Establiment de la llavor, diferent en cada execució per assegurar seqüències diferents.

for(i=0;i<SEGONSFILL;i++)
{
	sleep(1);
	sprintf(cadena, "\t%s[%d] El proces fill acabara en %d segons%s\n", color_red, getpid(), SEGONSFILL-i, color_end);
	write(1, cadena, strlen(cadena));
}

exitcode = rand()%10;      // Genera un nombre aleatori entre 0 i 9.
sprintf(cadena, "\t%s[%d] El proces fill ha finalitzat retornant amb codi de finalitzacio [%d]!%s\n", color_red, getpid(), exitcode, color_end);
write(1, cadena, strlen(cadena));
exit(exitcode)
```

- Bloc de codi 3.

```
sprintf(cadena, "%s[%d] Error en l'execució del fork.%s\n", color_blue, getpid(), color_end);
write(2, s, strlen(s));
exit(0);
```

Programa al qual heu d'omplir els blancs amb els blocs de codi anteriors:

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

#define       SEGONSFILL    (int)5

char *color_blue    = "\033[01;34m";
char *color_red     = "\033[01;31m";
char *color_end     = "\033[00m";

int    main()
{
       int     estat,i, exitcode;
       unsigned int llavor;
       char    *s;
       pid_t   pid,fpid;
       char   cadena[100];

       sprintf(cadena, "\n%s[%d] ********** Iniciant PROCES des de %d **********
%s\n", color_blue, getpid(), getppid(), color_end);
       if((write(1, cadena, strlen(cadena)))==-1)
              perror("ERROR: write 1");

       pid = fork();

       if(pid==-1)
       {
              

       }
       else if(pid==0)
       {
              

       }
       else
       {
              

       }
}
```

g) Torna a programar el codi anterior perquè faci el mateix i canviant l'estructura condicional *if-else* per una estructura *switch-case*.

h) **Treball amb nombres aleatoris**: en qualsevol de les dues versions (amb *if-else* o *switch-case*) feu 3 execucions anotant la llavor establerta (i que es mostra per pantalla) i el codi d'acabament del procés fill. Després comenteu les línies del procés fill que van de `llavor = (unsigned int) time(NULL);` fins a `srand(llavor);`. Torneu a fer 3 execucions i anoteu els 3 codis d'acabament. **Comenta raonadament els codis d'acabament de les 3 primeres execucions comparats amb els codis d'acabament de les 3 darreres execucions**.

4.2 Processos, senyals i canonades (*pipes*) : EPSlotto

MOLT IMPORTANT: Heu de fer la pràctica en dos executables (per tant dos programes .c), i emprant recobriment (funcions execl).

Es tracta que feu la simulació d'un programa de generació de nombres aleatoris d'un sorteig de loteria, emprant canonades (*pipes*) i senyals.

En aquest simulador hi haurà un **procés pare** i **cinc processos fill** que s'encarregaran de **generar cadascun dels dígits** d'un número de loteria (unitats, desenes, centenes, unitats de miler i desenes de miler).

Cada fill es comunicarà amb el pare a través de dues pipes:

- Una canonada sentit pare-fill, per on el pare li passarà al fill la llavor d'un nombre aleatori.
- Una canonada sentit fill-pare, per on el fill li passarà al pare el dígit obtingut.

Proces pare:

El procés pare, **anomenat loteria**, i que serà el que inici tot el procés (s'executarà des de l'intèrpret de comandes), rebrà per paràmetre un enter que serà la llavor de la seqüència de nombres aleatoris:

1. Crearà les canonades, els fills, gestionarà les canonades i els senyals i es quedarà a l'espera de rebre el senyal SIGQUIT (que es genera amb el teclat prement Ctrl+4).
2. Quan rebi el senyal SIGQUIT, generà cinc nombres aleatoris, i els enviarà per les canonades corresponents a cadascun dels fills.
3. Restarà a l'espera de rebre per les canonades els dígits que hagin calculat els fills.
4. Un cop hagi rebut els cinc dígits, imprimirà el resultat per pantalla i restarà de nou a l'espera de rebre un altre senyal (SIGQUIT o SIGINT).

Processos fill:

Per la seva part, els processos fill:

1. Gestionaran les canonades i els senyals que corresponguin.
2. Restaran a l'espera de rebre per la canonada corresponent la llavor del nombre aleatori a generar.
3. Quan rebin la llavor, generaran el nombre i li enviaran al pare per la canonada corresponent.
4. Restaran a l'espera de rebre una altra llavor de nombre aleatori (punt 2).

Acabament del programa:

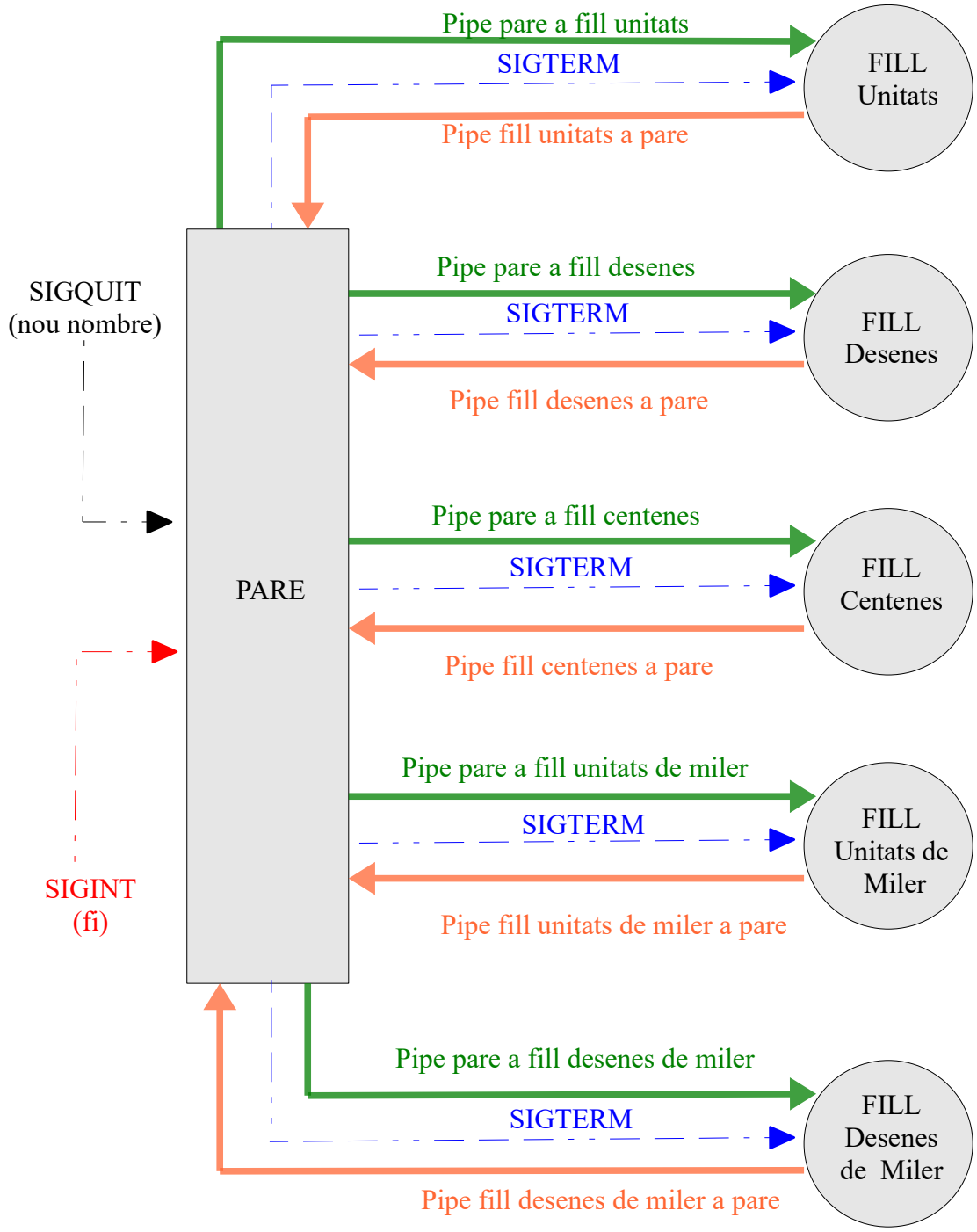
L'acabament del programa es farà controlant el senyal SIGINT que només haurà de gestionar el pare (els fills l'han d'ignorar):

1. El pare rebrà el senyal SIGINT (Ctrl+C), i enviarà el senyal SIGTERM als cinc fills i esperarà que acabin els cinc fills.
2. Quan els fills rebin el senyal SIGTERM, acabaran.
3. Quan el pare detecti la fi dels cinc fills, imprimirà un missatge per pantalla.

Cal que tingueu en compte quins senyals ha de gestionar cada procés (pare i fills) i quins ha d'ignorar.

Arquitectura de la solució:

L'arquitectura de processos de la solució es pot esquematitzar com:



Recordeu que heu de fer la pràctica en dos executables (per tant dos programes .c), i emprant recobriment (funcions execl).

Exemple d'execució:

```
[root@localhost src]# ./loteria 5

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 98292

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 5297

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 55055

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 72775

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 32693

(s'ha pitjat Ctrl+4)
Pare> Número premiat: 24850

(s'ha pitjat Ctrl+C)
Pare> Adeu
```

5 Avaluació

Els exercicis es ponderaran:

1. Exercici 1: **15%**
2. Exercici 2: **85%**. Es valorarà l'ús correcte de les crides a sistema, el control d'errors en la seva utilització i la correcta programació, estructura i funcionament de l'aplicació. **Cal que els programes estiguin correctament tabulats diferenciant els diferents blocs de codi.**

   Concretament:

   2.1. Prova validació: **10%**

   2.2. Senyals: **20%**

   2.3. *Pipes:* **20%**

   2.4. Disseny: **20%**

   2.5. Funcionament: **20%**

   2.6. Documentació (codi comentat, memòria amb les consideracions que cregueu oportunes, etc.): **10%**

6 Annexos

Annex A: Crides al Sistema involucrades

A la pràctica podeu utilitzar les següents crides a Sistema:

- `int open(const char *pathname, int flags);`
- `int close(int fd);`
- `off_t lseek(int fildes, off_t offset, int whence);`
- `ssize_t read(int fd, void *buf, size_t nbytes);`
- `ssize_t write(int fd, const void *buf, size_t num);`
- `sighandler_t signal(int signum, sighandler_t handler);`
- `int kill(pid_t pid, int sig);`
- `int pause(void);`
- `unsigned int alarm(unsigned int seconds);`
- `pid_t fork(void);`
- `int execlp(const char *file, const char *arg, ...);`
- `pid_t wait(int *status);`
- `void exit(int);`

Annex B: Algunes funcions que us poden ajudar

Altres funcions que us poden ser d'utilitat:

Generació nombres aleatoris:

- **srand()**[1] : estableix la llavor per a la generació de nombres aleatoris. A diferent llavor, diferent seqüència de nombre generats per rand().
- **rand()** : genera un nombre aleatori.

Podeu veure un exemple d'ús d'aquestes funcions a l'exercici 4.1 d'aquesta pràctica.

Mida informació a llegir/escriure:

- sizeof(). Retorna el numero de bytes de cada tipus de variable (int, char, long…)

Annex C: Fòrum per a dubtes al Campus Virtual

Fora de l'aula, l'única via per a plantejar qualsevol dubte és l'apartat **Fòrums** a l'espai de l'assignatura al Campus Virtual, on hi ha disponible el fòrum **PraLab → PRA1. Processos, *pipes* i senyals: EPSlotto**, on podeu participar exposant els vostres dubtes i suggerint respostes. No adjunteu solucions completes.

1 http://linux.die.net/man/3/rand