Problema 5.1 Esperar fills

Programeu dues funcions:

- `int EsperarFill(int pid)`: rep el pid d'un procés fill concret, esperi que aquest procés acabi i retorna l'estat en que ha acabat.
- `int EsperarTotsElsFills()`: espera que acabin tots els fills d'un procés, retornant el nombre de fills que tenia aquest procés.

Objectiu: Veure el funcionament d'esperar la finalització d'un procés fill. Mireu al man el funcionament de la crida waitpid.

Problema 5.2 Exec recursiu

Feu un programa que escriu el seu identificador de procés i un número rebut com a paràmetre per la línia de comandes (per exemple, 10). Si el número és més gran que zero, el programa s'executa a si mateix (`execlp`/`execvp`/...) passant-se com a argument el número decrementat en 1 (en el nostre exemple, 9) i en cas contrari acaba. Què passa amb l'identificador del procés? Es modifica? Raona el resultat.

- Objectiu: Veure que carregar un executable es una operació que un procés es fa a ell mateix. Utilització de la crida `execvp`/`execlp`/...

Problema 5.3 Comunicació pare/fill mitjançant pipes

Escriviu un programa en el qual es creï una pipe i un procés fill. El procés pare ha d'escriure una línia de text a la pipe. El procés fill l'ha de llegir i escriure-la per la sortida standard.

- Objectiu: Veure con el procés pare i el fill es poden enviar dades per la pipe
- Crides a sistema: De processos: `fork, exit, wait`. De pipes: `pipe`. De E/S, `read,write`.
- Nota: Fes que el procés pare esperi que acabi el fill fent un wait. Comproba que passa si pare i fill no han tancat be els canals de la pipe.

Problema 5.4 Crear hijos

Escribid un programa que cree tantos hijos como se le indique en la línea de comandos. Cada uno de estos hijos le enviará al padre su identificador de proceso (PID) mediante una pipe (todos los hijos por la misma pipe) y a continuación morirá. El padre escribirá en un fichero (segundo parámetro) todos los pids que le lleguen en formato ascii (separados por espacios). Los hijos escriben su pìd en formato interno de la máquina. Por ejemplo:

```
> crea 15 hijos.pid
```

creará 15 procesos hijos cuyos identificadores aparecerán en el fichero "hijos.pid".

- Objetivo: Ver como el mecanismo de pipes sirve también para hacer comunicacion de datos de N procesos a 1, no solo 1 a 1 como en el caso anterior
- Llamadas a sistema: Procesos(`fork, exit, wait`). Pipes (`pipe`), E/S(`read,write`).

Problema 5.5 Crear hijos_redireccionando

Suponed que tenemos hecho un programa que simplemente escribe su pid por su salida estándar en formato interno de la máquina. El nombre del programa es `escribe_pid` y no recibe ningun parámetro. Del programa `escribe_pid` sólo tenemos acceso al ejecutable, no es posible modificarlo.

Escribid un programa que cree tantos procesos hijos como se le indique por la línea de comandos. Queremos que cada uno de los procesos hijos ejecute el programa `escribe_pid`. El padre debe recoger los pids que escribiran los hijos. Para ello, el padre creará una pipe que utilizará para comunicarse con los procesos hijos.

El padre tendrá el mismo formato que en el ejercicio anterior, es decir, recibirá:

> `crea num_hijos fichero_salida`

Donde el primer parámetro es el número de hijos a crear y el segundo es el fichero donde queremos escribir los pids en formato ASCII (separados por espacios). Por ejemplo:

```
> crea 15 hijos.pid
```

creará 15 procesos hijos cuyos identificadores aparecerán en el fichero “hijos.pid”.

- Objetivo: Utilizar el mecanismo de redirección de canales visto en clase.
- Llamadas a sistema: Procesos(`fork, exec`(decide cuál), `exit, wait`). Pipes (`pipe`), Entrada/Salida(`read,write,dup,dup2`).

Problema 5.6 Timeout llegint de teclat

Feu un programa que llegeixi una línia del terminal. Si durant un període de 10 segons no s’ha teclejat un return, caràcter ‘\n’, el programa ha d’acabar donant un error tal com

```
Error reading command input Timeout period expired
```

Si s'ha llegit una línia en menys de 10 segons, el programa cridarà a la rutina `processar_linia(char *)`.

- Objectiu: Controlar el temps transcorregut en un programa.
- Crides a sistema: Gestio events: `alarm, signal`. Entrada/sortida: `read, write`

Problema 5.7 Jugant amb la taula de canals

Donats els següents programes (`pare.c` i `fill.c`), executem `pare`, corresponent a `pare.c`:

```
/* Pare.c */
#define NULL (char *) 0
main()
{ int    fd,fp[2],err,status,id;
  char   c;

fd = open ("pp.pp",O_RDONLY);
err = pipe(fp);
if (fd == -1)||(err==-1) exit(1);
                                   A
id = fork();
switch (id) {
       case 0:  close(0);
                dup(fp[0]);
                close(fp[0];
                close(fp[1]);
                 execl("fill",fill",NULL);
                exit(1);
        case -1:exit(1);
        default:close(1);
                dup(fp[1]);
                close(fp[0]);
                close(fp[1]);
                                   B
                while(read(fd,&c,1)!=0)
                       write(1,&c,1);
                close(1);
                wait(&status);
                exit(0);}
}
```

```
/* Fill.c */
main()
{ char    c;
                                   B
while (read(0,&c,1)>0)
        write(1,&c,1);
exit(0);
}
```

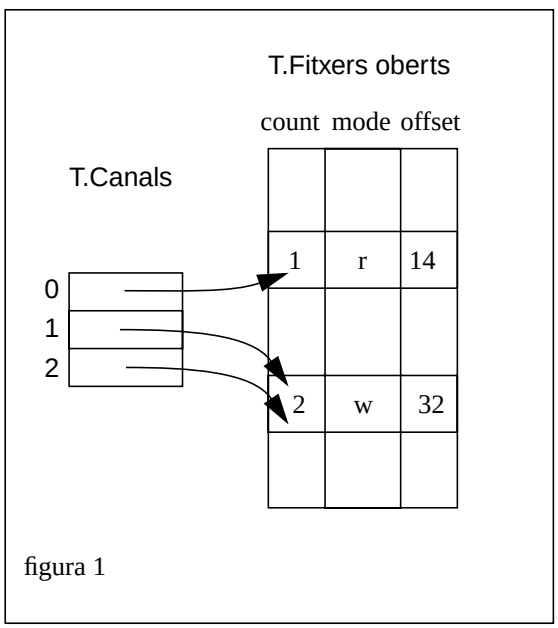

figura 1

1.- Dibuixeu la Taula de Canals i la Taula de Fitxers Oberts al punt A. Cal que ompliu els camps explicats a la figura 1. Suposeu que podem tenir fins a 10 canals oberts.

2.- Feu el mateix que a la pregunta anterior però en el punt d'execució B, suposant que tant el pare com el fill estan, simultàniament, al punt B.

3.- Què fa aquest programa si l'executem des de la shell de la següent forma:
`> pare <in.dat >& out.dat` on < indica redireccionament del canal d'entrada i >& el redireccionament dels canals de sortida i error?

- Objectiu: Comprobar com es modifiquen les estructures de dades en funció de les crides a sistema.

Problema 5.8 Dos mètodes

Disposem de dos programes diferents (`mètode1` i `mètode2`) que realitzen un mateix càlcul. Llegeixen 3 nombres de la seva entrada estàndar (en el format intern de la màquina), i per la sortida estàndar escriuen un enter que és el resultat del càlcul (en el format intern de la màquina) . La diferència entre els dos programes és el temps d'execució; per algunes entrades `mètode1` ens donarà abans el resultat que `mètode2`, però per les altres entrades serà `mètode2` el primer en donar-nos el resultat.

Quan un d'aquests mètodes finalitza el seu càlcul, envia un signal al seu procés pare (un signal del tipus `SIG_USR1` el `mètode1`, i de tipus `SIG_USR2` el `mètode2`), i escriu per la seva sortida estàndar (en el format intern de la màquina) el resultat. Si en qualsevol moment a un dels dos mètodes li arriba un signal del tipus `SIGTERM`, abortarà inmediatament el càlcul sense escriure res per la seva sortida estàndar.

Volem aplicar aquest càlcul a 3 enters i obtenir el resultat el més ràpid possible. Com a priori no sabem quin dels mètodes ens donarà abans el resultat, executarem tots dos mètodes concurrentment, i quan amb un obtinguem el resultat aturarem l'altre.

Heu d'escriure un programa que tindrà com a únic paràmetre un nom de fitxer (on estaran els 3 nombres enters amb el format esperat per `metode1` i `metode2`). Aquest programa escriurà per la seva sortida estandar el resultat del càlcul (en el format intern de la màquina), indicant, a més a més, quin mètode ha obtingut el resultat.

Si passats 10 segons cap dels mètodes ens dóna un resultat, aturarem inmediatament tots dos mètodes i el programa finalitzarà notificant-ho, escribint un missatge per la seva sortida estàndar.

Problema 5.9 Signals entre germans

Escriu un programa que tingui el següent comportament:

- Ha de crear dos processos fills (`A` i `B`)
- El procés `A` ha d'enviar un signal `SIGUSR1` al procés `B` cada cop que el procés `A` llegeixi el caràcter 'a' per la seva entrada estàndar.
- El procés `B` envia signals `SIGUSR1` al procés `A`. Inicialment n'enviarà un cada segon; però cada cop que `B` rebi un `SIGUSR1`, incrementarà en un segon el temps que hi ha entre l'enviament de dos signals.
- `A` acabarà quan detecti el final de stdin. Quan el pare detecti la mort de `A`, el pare haurà de provocar la finalització de `B` i la seva pròpia.
- Objectiu:Veure con es poden enviar events dos processos.
- Crides a sistema: Procesos:fork,exit,wait. Gestió d'events: kill, signal, alarm. Entrada/sortida:read,write.

Problema 5.10 Finalització de processos

Si executes la línia de comandes: `% ls -l / | cat | sort > fitxer`

- Indica quants processos crearà el shell i com estaran comunicats.
- En condicions normals, en quin ordre creus que acabaran els processos? Justifica-ho.
- Imagina que s'envia un event `SIGKILL` al procés `cat` quan encara no ha mort cap dels processos creats. Què passarà amb la resta de processos?

Problema 5.11 ls i sort

Volem fer un programa que faci, **el mateix** que faria l'interpret de comandes a:

>˜ ls -l | sot >fitxer

El nostre programa es dira ls_sort i rebra com a parametre el nom del fitxer de sortida. El format serà:

>˜ ls_sort fitxer

- Objectiu: Veure la gestió que fa l'interpret de comandess quan ha de comunicar dos procesos i redireccionar canals.
- Crides a sistema: Gestió de procesos:fork,exec(el que calgui),wait. Comunicació:pipe. Entrada/Sortida: read,write,dup,dup2

Problema 5.12 Timeout

Haced un programa llamado `timeout` que ejecute otro comando. Si dicho comando no acaba en el tiempo especificado, será abortado cuando se acabe el plazo. El formato del comando será: `timeout [-sec] comando [lista_de_parámetros]`

Si no se especifica el parámetro `sec,` se considerará que el tiempo de ejecución es un segundo. A priori no sabemos la cantidad de parámetros que tendrá el comando.