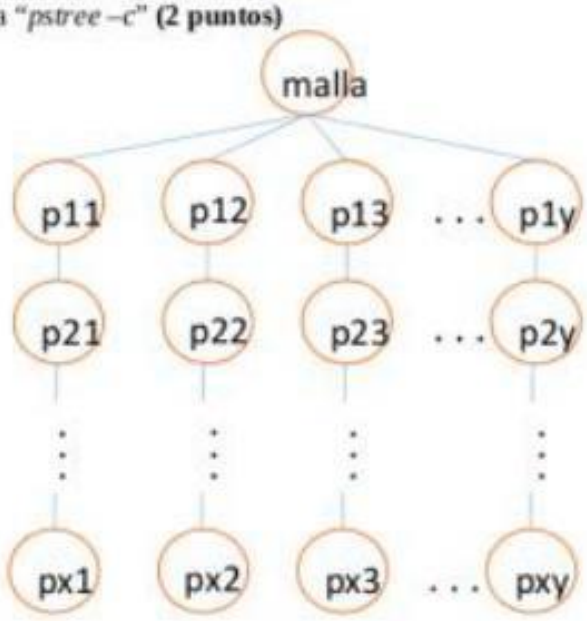
Sistemas Operativos
2020/2021

Práctica 1

Gestión de procesos y archivos. Comunicación entre procesos: tuberías y memoria compartida

Ejercicio 1. Gestión básica de procesos. (**4 puntos**)

a) Realiza un programa llamado **malla.c** que produzca el siguiente árbol de procesos. El programa recibirá dos argumentos 'x' e 'y' que representan el número de filas y columnas. Para comprobar la estructura de procesos que realmente estamos creando debemos emplear la llamada al sistema "*pstree –c*" **(2 puntos)**



El programa recibe los argumentos x como filas e y como columnas, primeramente creo un for que se encargará de crear tantos procesos hijo (llamadas a fork()) tantas veces como indique el parámetro x

```c
int main(int argc, char *argv[]){
    pid_t pid;
    pid_t pid2;

    for(int i=0;i<atoi(argv[2]);i++){
        pid=fork();
        if(pid){
            printf("Soy el proceso %d\n",getpid());
            sleep(2);
        }
        else{
            printf("soy el hijo %d, mi padre es %d\n",getpid(),getppid());
            sleep(2);
```

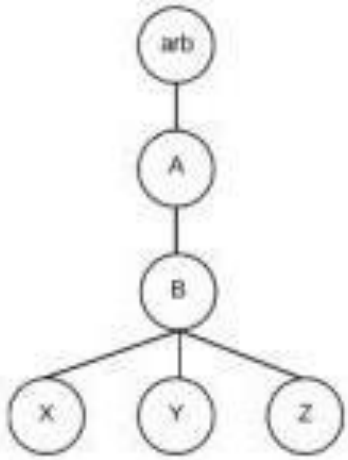
2 Argumento de la consola (string-> int)

dentro de este for incluyo un segundo que se encarga de hacer lo mismo pero con el parámetro y. Es decir para cada hijo 'x' creo 'y' Hijos, con el fin de hacer la malla que propone el enunciado, mientras se crean dichos procesos voy almacenando los PID de los hijos que se van generando con el fin de imprimirlos después.

```c
        else{
            printf("soy el hijo %d, mi padre es %d\n",getpid(),getppid());
            sleep(2);
            for(int j=0;j<atoi(argv[1]);j++)
            {
                pid2=fork();
                if(pid2)
                {
                    printf("Soy el proceso %d\n",getpid());
                    sleep(2);
                }
                else
                {
                    printf("Soy el proceso %d\n",getppid());
                    sleep(2);
                    exit(0);
                }
            }
            exit(0);
        }
        printf("\n");
    }
    exit(0);
```

```
dani@DaniD:~$ ^C
dani@DaniD:~$ pstree -c | grep hacha
dani@DaniD:~$ pstree -c | grep hacha
dani@DaniD:~$ pstree -c | grep malla
           |-bash---malla-+-malla-+-malla
           |               |       `-malla
           |               |-malla---malla
           |               `-malla
dani@DaniD:~$ pstree -c | grep malla
           |-malla-+-malla
           |       `-malla
           |-malla---malla
dani@DaniD:~$ pstree -c | grep malla
           |-malla-+-malla
           |       `-malla
dani@DaniD:~$ pstree -c | grep malla
```

b) Realiza un programa llamado **ejec.c** que reciba un argumento. El programa tendrá que generar el árbol de procesos que se indica y llevar a cabo la funcionalidad que se describe a continuación. El proceso Z, transcurridos los segundos indicados por el argumento, ejecutará el comando "pstree". No se puede utilizar el comando "sleep", por lo que el proceso Z debe planificarse una alarma con los segundos indicados por el argumento. Se deberá controlar la correcta destrucción del árbol (los padres no pueden morir antes que los hijos). **(2 puntos)**



El programa Recibe un argumento que es almacenado en una variable tiempo para poder usarlo posteriormente.
Al comienzo del programa se crean los distintos PIDs de los procesos que se van a crear junto al del padre que se irán asignando para poder mostrarlos por consola.

```
int main (int argc,char* argv[])
{
    pid_t pidPadre,pidA,pidB,pidX,pidY,pidZ;    //Creo los PID de todos los procesos para posteriormente almacenarlos

    int tiempo = atoi(argv[1]);

    pidPadre = getpid();
    printf("\nSoy el procesor ejec mi pid es %d \n",pidPadre);
    signal(SIGALRM,alarma);                     //comando signal para usar alarm
```

El programa comienza con una llamada a fork() para crear el proceso A, después se guarda el PID de este en la variable previamente mencionada con la instrucción getpid() y se muestra por pantalla tanto su PID como el PID de sus procesos antecesores (padre,abuelo… etc), este proceso se repite para el proceso B.

```
if(pid == 0)    // Proceso A
{
    pidA = getpid();
    wait(0);
    printf("Soy el proceso A: mi pid es %d,mi padre es %d\n",pidA,pidPadre);
```

*Luego Se repite el mismo desarrollo para el Proceso B

Después dentro del proceso B se hacen 3 llamadas a fork() con el fin de crear los procesos X Y Z y que estos sean todos hijos de B.

```
pid = fork();
if(pid == 0)    //Proceso X
{
    pidX = getpid();
    alarm(tiempo);
    printf("Soy el proceso X: mi pid es %d,mi padre es %d , mi abuelo es %d y mi bisabuelo es %d\n",pidX,pidB,pidA,pidPadre);
    pause();

}

pid = fork();
if(pid == 0)    //Proceso Y
{
    pidY = getpid();
    alarm(tiempo);
    printf("Soy el proceso X: mi pid es %d,mi padre es %d , mi abuelo es %d y mi bisabuelo es %d\n",pidY,pidB,pidA,pidPadre);
    pause();
}

pid = fork();
if(pid == 0)    //Proceso Z
{
    pidZ = getpid();
    alarm(tiempo);
    printf("Soy el proceso X: mi pid es %d,mi padre es %d , mi abuelo es %d y mi bisabuelo es %d\n",pidZ,pidB,pidA,pidPadre);

}
```

Entre procesos trato de hacer una llamada a alarm (con la instrucción signal previamente) pasándole como parámetro el recibido en consola.

```
void alarma()
{
    system("pstree -p | grep arbol");
}
```

Como se puede observar los PID de Padre de A coinciden con los de abuelo de B…

```
dani@DaniD:~/Escritorio/SO/SOtrue$ ./ejec 4

Soy el procesor ejec mi pid es 17324
Soy el proceso A: mi pid es 17325,mi padre es 17324
Soy el proceso B: mi pid es 17326,mi padre es 17325 y mi abuelo es 17324
dani@DaniD:~/Escritorio/SO/SOtrue$ Soy el proceso X: mi pid es 17333,mi padre es 17293 , mi abuelo es 17292 y mi bisabuelo es 17291
Soy el proceso X: mi pid es 17334,mi padre es 17293 , mi abuelo es 17292 y mi bisabuelo es 17291
```

Ejercicio 2. Comunicación entre procesos: tuberías. (**4 puntos**)

Realizar un programa llamado **hacha.c** que divida un archivo en varios trozos con el mismo nombre y extensión h00, h01, … teniendo en cuenta el formato y las siguientes consideraciones:

$ hacha <archivo> <tamaño>
archivo a dividir
tamaño en bytes de los archivos divididos

- Para dividir, el proceso hacha (proceso padre) generará tantos procesos hijos como archivos se tengan que crear. Mediante tuberías el proceso padre enviará la información a los hijos y serán estos últimos los que creen los archivos de destino y escriban en ellos.

```
$ hacha at_madrid.mp3 50000
$ ls
at_madrid.mp3.h00
at_madrid.mp3.h01
at_madrid.mp3.h02
```

Consideraciones

- Las entradas y salidas a los archivos se realizarán con llamadas al sistema estudiadas en esta práctica (no se puede utilizar printf, scanf, etc.)
- El padre creará tantos hijos como fragmentos del archivo a realizar, siendo decisión del alumno si los hijos se lanzan secuencial o concurrentemente.
- El padre lee el archivo origen y mediante tuberías le pasa la información del archivo a los hijos.
- Cada hijo crea el archivo de destino y escribe la información que le ha pasado el padre.

El programa comienza acotando el tamaño del fichero que se recibe en la llamada del programa, después calcula el número total de hijos a crear como nFichero/nHijos siendo nFichero el tamaño del archivo.

```
struct stat sb;
stat(argv[1], &sb);
nFichero=sb.st_size;  //Tamaño del fichero origen
totalArchivos=ceil(nFichero/nHijos);// archivosTotal será el numero de hijos
```

Después se crea una pipe (tubería), y entra en un for donde se hacen tantas llamadas a fork() como el total de archivos a crear (previamente calculado), dentro del for se van abriendo, creando nuevos archivos y leyendo y escribiendo la información del fichero entrante a estos nuevos.

```
for(int i=0;i<totalArchivos;i++){
    if(fork() > 0){
    if(i==0){
        fd = open(argv[1], O_RDONLY);
    }
    read(fd, buff, nHijos);                 //read (archivo, buffer nbytes)
    write(tuberia[1], buff, nHijos);        //write (archivo, buffer, nbytes)
    if(i==totalArchivos-1){
        close(fd);
    }
    }
    else{
    sprintf(archivo,"%s.h0%d",argv[1],i);
    fd = creat(archivo, 0666); //0666 tipo de permisos

    read(tuberia[0], buff, nHijos); //leo de la tubería y escribo en el archivo creado por el hijo
    write(fd, buff, nHijos);
```

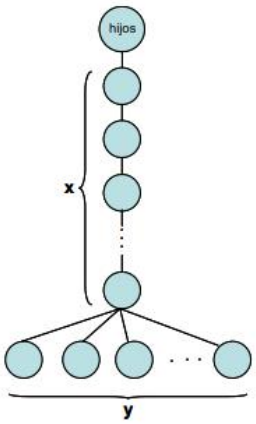
La llamada ha creado 5 archivos de tamaño 2b dado que el archivo inicial era de 10b, como se puede ver en la siguiente captura.

```
dani@DaniD:~/Escritorio/SO/SOtrue$ ./hacha doc1.txt 2
dani@DaniD:~/Escritorio/SO/SOtrue$ ls -la
total 136
drwxr-xr-x 2 dani dani  4096 oct 26 19:00 .
drwxrwxr-x 3 dani dani  4096 oct 26 17:57 ..
-rw-rw-r-- 1 dani dani   686 oct 20 20:15 arbol.c
-rw-r--r-- 1 dani dani    10 oct 26 18:07 doc1.txt
-rw-r--r-- 1 dani dani     2 oct 26 19:00 doc1.txt.h00
-rw-r--r-- 1 dani dani     2 oct 26 19:00 doc1.txt.h01
-rw-r--r-- 1 dani dani     2 oct 26 19:00 doc1.txt.h02
-rw-r--r-- 1 dani dani     2 oct 26 19:00 doc1.txt.h03
-rw-r--r-- 1 dani dani     2 oct 26 19:00 doc1.txt.h04
```

Ejercicio 3. Comunicación entre procesos: memoria compartida. (**2 puntos**)

Diseña un programa llamado **hijos.c** que cree un árbol de procesos según la siguiente estructura a partir de dos parámetros

```
$hijos x y
```



El proceso **hijos** debe mostrar el siguiente mensaje:

"Soy el superpadre (pid) : mis hijos finales son: `pidx1, pidx2, pidx3, …, pidxy`"

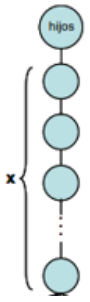
Los procesos de la parte baja deben mostrar el siguiente mensaje**:**

"Soy el subhijo pidx1, mi padres son: `pid1, pid2, …, pidx`"

El programa comienza acotando los parámetros y punteros (para usar shmat).

```c
int main(int argc, char * argv[]) {
  int i, x, y, pid;

  // Punteros para el uso de 'shmat'.
  int * puntero1, * puntero2, shm1, shm2;

  int hijos = 0;

  if (argc != 3)
  {
    printf("ERROR! Parámetros Incorrectos.\n");
  }
  else
  {
    printf("\n");

    x = atoi(argv[1]);
    y = atoi(argv[2]);

    if (x <= 0 || y <= 0)
    {
      printf("ERROR! los valores de x o y son incorrectos\n");
    }
```

Si nada de esto falla, el programa principal comienza con un for, donde se hacen tantas llamadas a fork(), como nos han pasado en el primer parámetro, con el fin de conseguir la estructura del enunciado:



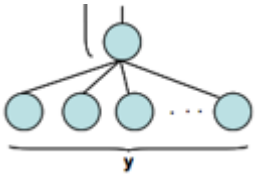
Dentro del for también vamos almacenando los PID de los procesos a crear con el fin de mostrarlos más tarde

```
for (i = 1; i <= x; i++)
{
  pid = fork();
  if (pid != 0)
    break; // El padre deja de hacer fork()
  else
  {
    printf("Soy el subhijo %d mis padres son: ", getpid());

    // Imprimo los procesos padre
    for (hijos = 0; hijos < i; hijos++)
    {
      printf("%d", puntero1[hijos]);

      if (hijos != i - 1)
        printf(", ");
    }
    printf("\n");

    // Guardo los PID de los hijos
    puntero1[i] = getpid();
  }
}
```

Mientras no haya terminado el bucle, los procesos esperan (a que el último hijo sea destruido) y se imprimen los PID almacenados por filas y columnas.

```
if (i <= x)
{

  wait(NULL);

  if (i == 1)
  {
    printf("Soy el superpadre %d mis hijos son ", getpid());

    // Imprimo los IDs de la primera fila
    for (i = 1; i <= x; i++)
    {
      printf("%d", puntero1[i]);
      printf(", ");
    }

    // Imprimo los IDs de la primera columna
    for (i = 1; i <= y; i++)
    {
      printf("%d", puntero2[i - 1]);
      if (i < y)
        printf(", ");
    }

    printf("\n");
  }
```

Una vez terminado el bucle para la fila (x) se procede a crear los E hijos (columna) con el fin de conseguir el esquema del enunciado:




```
else
{

  for (i = 1; i <= y; i++) {

    pid = fork();

    if (pid == 0) // El Hijo no tiene más hij
    {
      puntero2[i - 1] = getpid();
      sleep(20);
      break;
    }
  }

  if (i == y + 1) //El padre espera
  {
    for (i = 1; i <= y; i++)
      wait(NULL);
  }
}
```

↓Y esta es una captura de la creación de los procesos con pstree↓

```
dani@DaniD:~$ pstree -c | grep hijos
        |       |               |-bash---hijos---hijos---hijos---hijos-+-hijos
        |       |               |                                     |-hijos
        |       |               |                                     |-hijos
        |       |               |                                     |-hijos
        |       |               |                                     |-hijos
        |       |               |                                     `-hijos
dani@DaniD:~$ pstree -c | grep hijos
dani@DaniD:~$
```