Ejercicios de Gestión de Procesos

Ejercicio 1

Observa el siguiente código y escribe la jerarquía de procesos resultante.

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

int main (int argc, char *argv[]) {
int num; pidmore t pid;

for (num= 0; num< 3; num++) {
 pid= fork();

//Hacer la Segunda prueba sacando estos comentarios
// printf ("Soy el proceso de PID%d y mi padre tiene%d de PID.\n", getpid(), getppid());

if (pid!= 0) break;

//Hacer la primera prueba sacando estos comentarios
//printf ("Soy el proceso de PID%d y mi padre tiene%d de PID.\n", getpid(), getppid());

srandom(getpid());
 sleep (random()%3);
 }

 if (pid!= 0) printf ("Fin del proceso de PID%d.\n", wait (NULL)); return 0;
 }
```

Ahora compila y ejecuta el código para comprobarlo.

Contesta las siguientes preguntas:

¿Por qué aparecen mensajes repetidos? Presta atención al orden de terminación de los procesos, ¿qué observas? ¿por qué?

Ejercicio 2

Observa el siguiente código y escribe la jerarquía de procesos resultante.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
int num;
pid t pid;
srandom(getpid());

for (num= 0; num< 3; num++) {
pid= fork();
printf ("Soy el proceso de PID %d y mi padre tiene %d de PID.\n",getpid(), getppid());
if (pid== 0)
break;
}

if (pid== 0)
sleep(random() %5);
else
for (num= 0; num< 3; num++)
printf ("Fin del proceso de PID %d.\n", wait (NULL));
return 0;
}
```

Ahora compila y ejecuta el código para comprobarlo. Presta atención al orden de terminación de los procesos, ¿que observas? ¿por que?´

Ejercicio 3

Dibuja la estructura del árbol de procesos que obtendríamos al ejecutar el siguiente fragmento de código:

```
for (num= 0; num< 2; num++) {
nuevo= fork(); /* 1 */
if (nuevo== 0)
break;
}
nuevo= fork(); /* 2 */
nuevo= fork(); /* 3 */
printf("Soy el proceso %d y mi padre es %d\n", getpid(), getppid());
```

Ejercicio 4
Considerando el siguiente fragmento de código:

```
for (num= 1; num<= n; num++){
nuevo= fork();
if ((num== n) && (nuevo== 0))
execlp ("ls", "ls", "-l", NULL);
}
```

a) Dibuja la jerarquía de procesos generada cuando se ejecuta y n es 3.
b) Indica en que procesos se ha cambiado la imagen del proceso usando la función execlp.

Ejercicio 5
Dibuja la jerarquía de procesos que resulta de la ejecución del siguiente código. Indica para cada nuevo proceso el valor de las variables i y j en el momento de su creación.

```
for (i= 0; i< 2; i++) {
pid= getpid();
for (j= 0; j< i+2; j++) {
nuevo= fork(); /* 1 */
if (nuevo!= 0) {
nuevo= fork(); /* 2 */
break;
}
}
if (pid!= getpid())
break;
}
```

Ejercicio 6

Estudia el siguiente código y escribe la jerarquía de procesos resultante. Después, compila y ejecuta el código para comprobarlo (debes añadir llamadas al sistema getpid, getppid y wait para conseguirlo).

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#define L1 2
#define L2 3
int main (int argc, char *argv[]) {
int cont1, cont2;
pid t pid;
for (cont2= 0; cont2< L2; cont2++)
        {
        for (cont1= 0; cont1< L1; cont1++)
                {
                pid= fork();
                if (pid== 0)
                break;
                }
}
if (pid!= 0) break;
}

//*Para Simplificar agregar al final
printf ("Soy el proceso de PID %d y mi padre tiene %d de PID.\n",
getpid(), getppid());
if (pid!= 0)
for (cont1= 0; cont1< L1; cont1++)
printf ("Fin del proceso de PID %d.\n", wait (NULL));
return 0;
```

- }

Ejercicio 7

Dibuja la jerarquía de procesos que resulta de la ejecución del siguiente código. Introduce las llamadas al sistema wait para que una vez generado el árbol de procesos los hijos sean esperados por sus respectivos padres. Además, haz que se informe de los tiempos de ejecución de las aplicaciones xload y kcalc que se generen así como del tiempo total de ejecución. Para calcular el tiempo transcurrido, puedes utilizar la función time() de la librería estandar time.h. La llamada time(NULL) devuelve los segundos transcurridos desde las 00:00:00 del 1/1/1970 hasta el instante de la llamada.

```
int main (int argc, char ∗argv[]) {
int i, j;
pid t pid, nuevo, nuevo1;
time t ini, fin;
for (i= 0; i< 2; i++){
pid= getpid();
for (j= 0; j< i+2; j++){
nuevo= fork();
if(nuevo== 0){
break;
nuevo1= fork();
if(nuevo1== 0)
execlp ("xload", "xload", NULL);
}
}
if (pid!= getpid())
execlp ("kcalc", "kcalc", NULL);
}
return 0;
}
```

Ejercicio 8

Dado el siguiente código. Grafique el diagrama de procesos y mencione que realiza cada uno

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

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

pid=fork();

if(pid!=0)
{
printf("SOY EL PROCESO PADRE:%d (PROCESOS) Y MI PADRE ES:%d\n",getpid(),getppid());
wait(NULL);
//execlp ("ps", "ps","-eaf", NULL);
}
else
{
pid=fork();
      if(pid!=0)
      {
      printf("SOY EL PROCESO HIJO: %d (ESPACIO EN DISCO) Y MI PADRE
ES:%d\n",getpid(),getppid());
      execlp ("df","df","-h", NULL);
      wait(NULL);
      }

      else
      {
      printf("SOY EL PROCESO HIJO:%d (LS) Y MI PADRE ES:%d\n",getpid(),getppid());
      execlp("ls","ls","-l",NULL);
      exit(0);
      }
}

return 0;
}
```

Ejercicio 9

Dado el siguiente código. Grafique el diagrama de procesos.
Que diferencia encuentra con el diagrama del ejercicio 8.

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

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

pid=fork();

if(pid==0)
{
printf("SOY EL PROCESO HIJO:%d (PROCESOS) Y MI PADRE ES:%d\n",getpid(),getppid());
execlp ("ps", "ps", NULL);
exit(0);
}
else
{

pid=fork();
      if(pid==0)
      {
      printf("SOY EL PROCESO HIJO: %d (ESPACIO EN DISCO) Y MI PADRE
ES:%d\n",getpid(),getppid());
      execlp ("df","df","-h", NULL);
      exit(0);
      }

      else
      {
      printf("SOY EL PROCESO PADRE:%d (LS) Y MI PADRE ES:%d\n",getpid(),getppid());
      execlp("ls","ls","-l",NULL);
      wait(NULL);
      }
}
wait(NULL);
return 0;
```