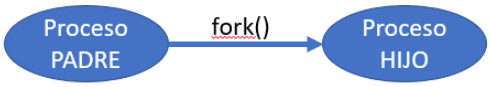
UT 1. Programación Multiproceso

Ejercicios Procesos – fork()

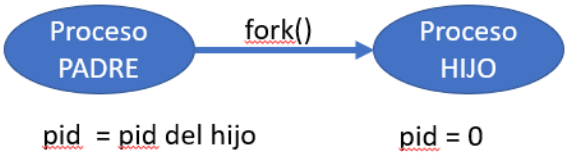
Como se ha explicado en clase, en Linux un proceso es creado mediante la llamada al sistema ***fork()***. El proceso que realiza la llamada se denomina proceso padre (parent process) y el proceso creado a partir de la llamada se denomina proceso hijo (child process).

La sintaxis de la llamada, efectuada desde el proceso padre, es: ***pid = fork()***



La llamada ***fork()*** **devuelve un valor distinto para los procesos padre e hijo**:

- Al proceso padre se le devuelve el PID del proceso hijo,
- Al proceso hijo se le devuelve "0".



Para obtener el identificador de un proceso, se utilizan 2 funciones que devuelven un tipo ***pid_t***. Las funciones son:

- ***pid_t getpid()*** devuelve el identificador del proceso que realiza la llamada, es decir, del proceso actual
- ***pid_t getppid()*** devuelve el identificador del proceso padre del proceso actual

La llamada wait() permite que el proceso padre espera la finalización del proceso hijo, el proceso padre quedará bloqueado hasta que termine el hijo. La sintaxis es la siguiente:

- ***pid_t wait (int *status)*** devuelve el identificador del proceso hijo cuya ejecución ha finalizado. La sentencia wait(NULL) es la forma más simple de esperar a que un hijo termine.

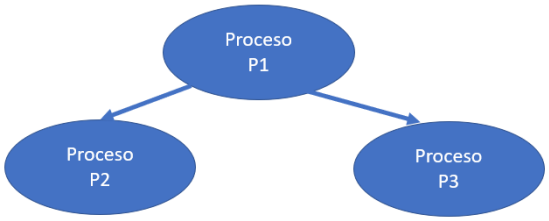
1. Crear un programa en C **mp1.c** que realice las siguientes funciones:

i. El proceso padre deberá crear un proceso hijo que mostrará el nombre del alumno.
ii. El proceso padre deberá esperar a que su hijo termine, y mostrará por pantalla el pid del hijo y el suyo propio

2. Crear un programa en C **mp2.c** que simule un árbol de procesos como el de la figura:

i. El proceso P1 es padre de P2 y abuelo de P3
ii. El proceso P2 es padre de P3 e hijo de P1
iii. El proceso P3 es hijo de P2 y nieto de P1
iv. El proceso P3 deberá mostrar por pantalla su pid y el de su padre, indicando que es el proceso P3
v. El proceso P2 deberá mostrar por pantalla su pid y el de su padre, indicando que es el proceso P2
vi. El proceso P1 deberá mostrar por pantalla su pid y el de su hijo, indicando que es el proceso P1
vii. Los procesos padres deberán esperar a que sus hijos terminen

3. Crear un programa en C **mp3.c** que simule un árbol de procesos como el de la figura:



i. El proceso P1 es padre de P2 y de P3
ii. El proceso P2 deberá dormir 10 segundos y mostrar el mensaje “Despierto” al finalizar
iii. El proceso P3 deberá mostrar por pantalla su pid y el de su padre, indicando que es el proceso P3
iv. El/los proceso/s padre/s deberá/n esperar a que sus hijo/s termine/n

4. Dado el siguiente código de programa **mp4.c**

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

void main()
{
   printf("CCC \n");
   if (fork()!=0)
   {
      printf("AAA \n");

   } else printf("BBB \n");

  exit(0);
}
```

a) Dibuja un gráfico de la jerarquía de procesos que genera la ejecución de este código, suponiendo que el pid del programa **mp4** es el 1000 y los pids se generan de uno en uno en orden creciente.

b) ¿Qué salida genera este código? ¿Podría producirse otra salida? Justifica la respuesta

c) Modificar el código para que la salida por pantalla sea:

CCC
BBB
AAA

5. Dado el siguiente código de programa **mp5.c**

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

void main()
{
   pid_t pid1, pid2;

   printf("AAA \n");

   pid1 = fork();

   if (pid1==0)
   {

      printf("BBB \n");

   }
   else
   {

      pid2 = fork();
      printf("CCC \n");

   }

  exit(0);
}
```

a) Dibuja un gráfico de la jerarquía de procesos que genera la ejecución de este código, suponiendo que el pid del programa **mp5** es el 1000 y los pids se generan de uno en uno en orden creciente.

b) ¿Qué salida genera este código? ¿Podría producirse otra salida? Justifica la respuesta

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten.