INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

SISTEMAS OPERATIVOS

# Práctica 2

# Administración de Procesos / hilos


- Alumnos:
  1) Arcos Hermida Aldo Alejandro (N° lista 5)
  2) Chávez Becerra Bianca Cristina (N° lista 9)
  3) Islas Osorio Enrique (N° lista 20)
  4) Juárez Cabrera Jessica (N° lista 21)
  5) Palmerin García Diego (N° lista 28)
- Grupo: 4CM1
- Materia: Sistemas operativos
- Profesor: Araujo Díaz David

Práctica 2 : Administración de Procesos / hilos

**Objetivo**

Entender y hacer uso de las llamadas al sistema en LINUX que permiten crear procesos.

**Descripción**

La administración de procesos en Linux es una de las tareas fundamentales del kernel. Para empezar, consideremos el caso de un sistema que posea un solo procesador. En la práctica, solamente un proceso puede utilizar el procesador en un momento dado. Luego de haberlo hecho, será el turno de otro proceso, y así sucesivamente. Por lo general, el tiempo que se le otorga a cada proceso es suficiente para que se lleve a cabo la tarea que deba realizar. Esta franja de tiempo es asignada por el kernel antes de darle acceso al proceso para que utilice el procesador. Debido a su corta duración, nosotros como usuarios percibimos como si en realidad varios procesos estuvieran utilizando el procesador de manera simultánea. El administrar procesos de esta manera es lo que hace de Linux un sistema operativo multitarea.

La creación de procesos se lleva a cabo mediante llamadas al sistema, en este caso en particular, debido a que usamos la distribución de Linux Ubuntu, haremos uso de la función **fork()**. Esta se encarga de crear una copia del proceso actual, misma que comparte los valores de las variables, ficheros y estructuras de datos (esto no quiere decir que tengan memoria compartida), pero con un identificador de proceso (PID) diferente. **fork()** no requiere argumentos, pero si devuelve un número que puede significar lo siguiente:

1. Si devuelve –1, significa que se produjo algún error al momento de ejecutar el fork.
2. Si devuelve 0, es porque no se produjo ningún error y nos encontramos en el proceso hijo.
3. Si devuelve un valor mayor que 0, es porque no se produjo ningún error, pero nos encontramos en el proceso padre. El numero devuelto será el PID asignado al proceso hijo.

Si en algún momento precisamos el PID de algún proceso, este podemos obtenerlo con la función **getpid()**, la cual nos devuelve precisamente el identificador del proceso en el que nos encontramos.

Así como podemos crear un solo proceso hijo, también podemos varios procesos hijos sin problemas, e inclusive crear procesos hijos para ellos.

Sin embargo, no siempre precisamos la creación de procesos independientes en nuestros programas, por lo que también tenemos la opción de crear hilos que realicen tareas específicas de manera pseudo-lineal. Para esto, podemos utilizar hilos, los cuales son definidos como flujos del programa que pueden realizar partes de una tarea en particular. Para su creación, utilizamos la función **pthread_create(),** la cual retorna un 0 cuando la llamada se completa de manera satisfactoria, cualquier otro valor de retorno indica que ocurrió un error durante la llamada.

Para lograr que un programa espere a que termine la ejecución de la rutina de un hilo, utilizamos la función **pthread_join()**, la cual recibe el id del hilo y un puntero a void* donde se puede almacenar el estado de salida del hilo especificado.

**Prueba de escritorio**

En este caso no es viable la realización de una prueba de escritorio, pues los valores mostrados pueden variar, según el sistema operativo.

**Código fuente**

Programa 1: este programa crea un proceso hijo, al ejecutarse el proceso hijo, el proceso padre se detiene hasta que este termine su ejecución. Hecho esto, el proceso padre procede a completar y terminar su ejecución.

*Nota: el valor de pidSon representa el valor regresado por la función **fork()**.

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

int main()
{
   //variable para guardar el PID del proceso hijo
   pid_t pidSon;

   //imprimimos el id del proceso padre, este lo obtenemos con getpid()
   printf("Proceso padre. PID: %d \n", getpid());

   //creamos una difurcación del programa principal, guardamos el PID
del hijo en pidSon
   pidSon = fork();

   //si pidSon es 0, significa que el hijo se está está ejecutando
   if(pidSon == 0)
   {
       //imprimimos el pid del proceso hijo y el valor de retorno de
fork (0)
```

```
        printf("Proceso hijo. PID: %d, pidSon: %d \n", getpid(),
pidSon);
        //dormimos el proceso por 3 segundos
        sleep(3);
    }
    //en el caso contrario, se está ejecutando el padre
    else
    {
        //esperamos a que el proceso hijo termine de ejecutarse e
imprimimos el pid del proceso padre
        wait(NULL);
        printf("Proceso hijo terminado, el padre se sigue ejecutando.
PID: %d \n", getpid());
        printf("Proceso padre ha terminado su ejecución.");

    }


    return 0;
}
```

Programa 2: en este programa se muestra la creación de 5 procesos hijos, mostrando sus respectivos identificadores de proceso (PID):

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

int main()
{

    pid_t pidSon;

    //imprimimos el id del proceso padre, este lo obtenemos con getpid()
    printf("Proceso padre. PID: %d . Creando procesos hijo...\n",
getpid());


    for(int i = 0 ; i < 5 ; i++)
    {
        //creamos un nuevo proceso hijo, almacenamos su pid
        pidSon = fork();

        //si pidSon es 0, significa que el proceso padre está trabajando
        if(pidSon > 0)
            printf("Creando procesos hijo... \n");

        //si pidSon es 0, significa que alguno de los hijos está
trabajando
        else if(pidSon == 0)
```

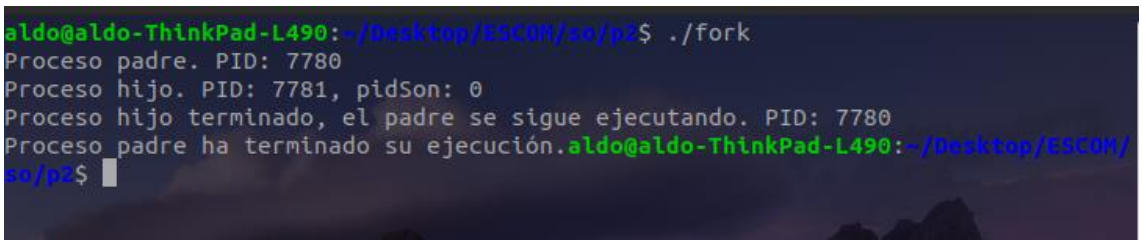
```
        {
            printf("Proceso hijo creado. PID: %d, pidSon: %d \n",
getpid(), pidSon);
            //terminamos el proceso
            exit(1);
        }
    }

    return 0;
}
```

Programa 3: este programa crea 2 hilos, los cuales imprimen mensajes en la pantalla 5 veces cada uno de manera pseudo-paralela:

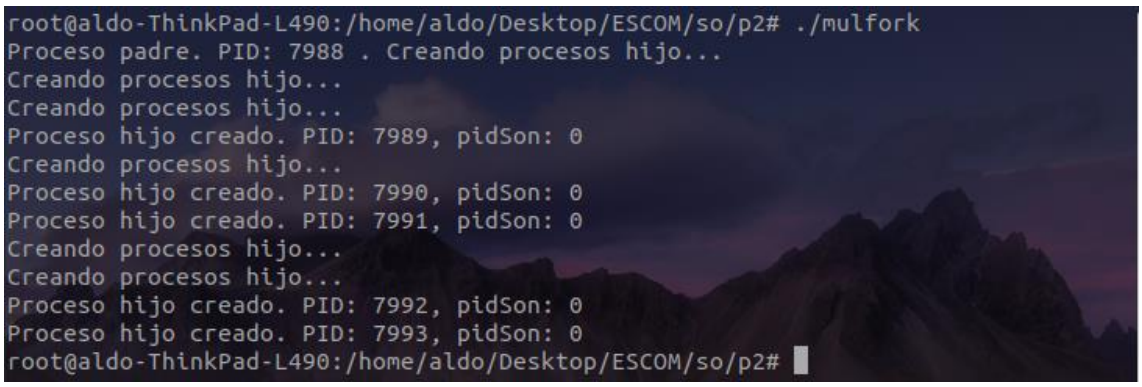
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>


//Funcion que establece la rutina de los hilos
void *troutine(void *arg)
{
    char *texto = (char *) arg;
    for(int i = 0 ; i < 5 ; i++)
    {
        printf("%s", texto);
        sleep(1);
    }
    return 0;
}


int main()
{

    pthread_t thread1, thread2;
    int value;

    //creamos los hilos
    pthread_create(&thread1, NULL, &troutine, "soy el hilo 1\n");
    pthread_create(&thread2, NULL, &troutine, "soy el hilo 2\n");

    //sumamos los hilos a la ejecucion
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

}
```

**Prueba de Pantalla**

Programa 1:


Programa 2:




Programa 3:

**Conclusiones**

Después de llevar a cabo la realización de esta práctica, se pudo apreciar cómo es que los sistemas operativos pueden crear y gestionar los procesos y los hilos, con el objetivo de hacer optimizar los tiempos de realización de ciertas tareas, pues ambos nos ayudan a desglosar tareas de mayor volumen en pequeñas secciones. Esto sin mencionar que el uso de uno u otro debe analizarse detenidamente, tomando siempre en cuenta las características particulares de cada uno. Se comprendió que todo proceso parte de un hilo principal y gracias al sistema operativo se realizan llamadas al sistema con lo cual se le permite al programador crear y destruir hilos, los hilos creados dentro de un proceso tienen en común el código y segmentos de datos.

También, se pudo ver y crear un programa multihilo pseudolineal, comprobando así la manera en que el sistema operativo simula llevar a cabo dos tareas al mismo tiempo, mismo que se observó con los procesos, pues estos pueden llevarse de manera paralela al proceso principal, o bien, configurarlos para que el proceso padre siga su ejecución después de que uno o todos sus procesos hijos hayan concluido.

**Referencias**


- Castillo, J. A. (2019, 1 abril). *Qué son los hilos de un procesador ? Diferencias con los núcleos*. Profesional Review. Recuperado 28 de febrero de 2022, de https://www.profesionalreview.com/2019/04/03/que-son-los-hilos-de-un-procesador/
- Silberschatz, A., Galvin, P. and Gagne, G. (2006). *Fundamentos de sistemas operativos*. 7th ed. Madrid: McGraw Hill.
- Khintibidze, L. (2021, 11 marzo). *Usa la función fork en C*. Delft Stack. Recuperado 28 de febrero de 2022, de https://www.delftstack.com/es/howto/c/fork-in-c/#:%7E:text=La%20funci%C3%B3n%20fork%20se%20utiliza,uno%20reci%C3%A9n%20creado%2C%20proceso%20hijo.
- Haylem, M. (2019, 20 febrero). *Cómo crear hilos en C de forma fácil*. GUTL. Recuperado 28 de febrero de 2022, de https://gutl.jovenclub.cu/como-crear-hilos-en-c-de-forma-facil/