	UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA **FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)** **DOCUMENTO DEL SISTEMA DE CALIDAD**

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
4	NOMBRE DE LA PRÁCTICA	Procesos	2

1. INTRODUCCIÓN

En C bajo Linux en proceso es creado mediante la llamada al sistema fork( ). El proceso que realiza la llamada se denomina proceso padre y el proceso creado a partir de la llamada se denomina proceso hijo. En esta práctica veremos algunos ejemplos que ilustran el funcionamiento de la función fork().

2. OBJETIVO (COMPETENCIA)

En esta práctica el alumno se familiarizará con la llamada al sistema fork( ) y será capaz de comprender la diferencia en ejecución entre el los procesos padres e hijos.

3. FUNDAMENTO

En UNIX, cada vez que se ejecuta un programa se crea un proceso, es decir, un proceso es un programa en ejecución. Cada vez que el programa sea invocado se creará un proceso diferente. Los procesos a su vez pueden crear subprocesos, llamados procesos hijo (child process), un proceso puede tener varios procesos hijo, pero un proceso hijo sólo tendrá un padre. Cada proceso tiene un identificador único llamado PID. El identificador de un proceso padre se conoce como PPID.

El mando en el sistema operativo Linux que se utiliza para desplegar la información de los procesos existentes en el sistema **ps.** La forma de utilizarlo es:

ps <opciones>

Entre las opciones más comunes están (para la lista completa consultar man ps):

-A Selecciona todos los procesos.
-G Selecciona los procesos por grupo.
-U Selecciona los procesos por usuario.
-f Muestra un listado completo de información orientada al administrador.

Formuló	Revisó	Aprobó	Autorizó
MC Alicia del R. López Aguirre	MC Gloria E. Chávez Valenzuela		Dr. Maximiliano de Las Fuentes Lara
Maestro	Coordinador de Programa Educativo	Gestión de Calidad	Director de la Facultad

Procesos Foreground (en primer plano)

Cuando se teclea un mando, el shell crea un proceso hijo en el cual se ejecuta el mando. El proceso padre (el shell en este caso) se queda esperando a que se termine de ejecutar este proceso hijo. Mientras se esté ejecutando, el usuario no puede teclear otro mando, se dice entonces que el proceso hijo está ejecutándose en el foreground.

Una vez que el mando termina de ejecutarse, el proceso hijo muere, el proceso padre se reactiva, el prompt del shell aparece y entonces se puede teclear otro mando.

En resumen, un proceso que está ejecutándose en el foreground es aquél que tiene la atención total de la terminal.

Background Jobs (tareas en segundo plano)

Un proceso que se inicie desde el shell se puede ejecutar en el background como un **job** y así evitar que se bloquee la terminal. Para hacer esto solo se requiere agregar un **&** (ampersand) al final de la línea del mando.

Ejemplo:

```
[tu1316@tiburon tu1316]$ ./programa &
```

En el ejemplo anterior, *programa* fue invocado desde la línea de mando con la instrucción de que se ejecute en el background, entonces Linux inicia el proceso y retorna el PID que le asignó así como el número de job.

Para saber cuáles procesos están ejecutándose en el background utilizamos el mando *jobs*.

Ejemplo:

```
[tu1316@tiburon tu1316]$ jobs

[1]+ Running ./programa &
[tu1316@tiburon tu1316]$ fg 1
./programa
```

En el ejemplo anterior el proceso ahora es dueño de la terminal.
Para poner un proceso foreground en background es necesario realizar los siguientes pasos:

1. Suspender el proceso presionando la tecla Ctrl y la tecla z (Ctrl/z)
2. Teclear el mando bg para enviar el proceso al background.

Ejemplo:

```
[tu1316@tiburon tu1316]$ ./programa
Aquí se presiona CTRL-Z
[1]+ Stopped ./programa


[tu1316@tiburon tu1316]$ bg
[1]+ ./programa &
```

```
[tu1316@tiburon tu1316]$ jobs
[1]+ Running ./programa &
[tu1316@tiburon tu1316]$
```

kill

Este mando sirve para terminar procesos, la forma más común de utilizarlo es:

kill -9 [PID].

Ejemplo:

```
[tu1316@tiburon tu1316]$ ps
PID TTY TIME CMD
14327 pts/1 00:00:00 bash
30550 pts/1 00:21:18 programa
1469 pts/1 00:00:00 ps
[tu1316@tiburon tu1316]$ kill -9 30550
[1]+ Killed ./programa
```

La función fork()

La llamada fork() crea procesos nuevos haciendo una copia de la imagen en la memoria del padre. El nuevo proceso que se crea recibe una copia del espacio de direcciones del padre. Los dos procesos continua su ejecución en la instrucción siguiente al fork:

Sintaxis de fork()

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

La creación de dos procesos totalmente idénticos no suele ser útil. Así el valor devuelto por fork() permite diferenciar el proceso padre del hijo, ya que *fork devuelve 0 al hijo y el ID del hijo al padre*. En caso de error devuelve -1.

En el ejemplo 1 se muestra las acciones internas de la llamada fork( )

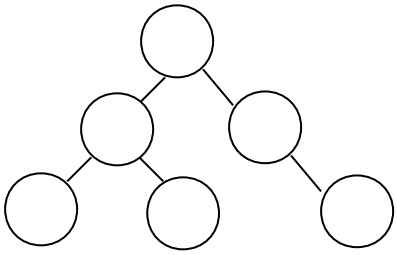
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    pid_t rf;

    rf = fork();
    switch (rf)
    {
        case -1:
            printf ("No he podido crear el proceso hijo \n");
            break;
        case 0:
            printf ("Soy el hijo, mi PID es %d y mi PPID es
                     %d \n", getpid(), getppid());
            sleep (20);
            break;
        default:
            printf ("Soy el padre, mi PID es %d y el PID de
                    mi hijo es %d \n", getpid(), rf);
            sleep (30);
    }
    printf ("Final de ejecución de %d \n", getpid());
    exit (0);
}
```

Ejemplo 1

4. PROCEDIMIENTO (DESCRIPCIÓN)	
A) EQUIPO NECESARIO	**MATERIAL DE APOYO**

Computadora con sistema operativo Linux.

1.- Copie y compile el programa del Ejemplo 1.
2.- Ejecute el programa del Ejemplo 1 en segundo plano
3.- Verifique su ejecución con la orden ps -l
4.-Observe los valores de PID y PPID e identifique que atributos son heredados entre padre e hijo y cuales no.
5.-Anote el valor mostrado por el shell inmediatamente después de lanzar el proceso en el segundo plano e investigue que significa dicho valor.
6.-¿Cuáles son los PID del proceso padre e hijo?
7.-¿Qué tamaño de memoria ocupan los proceso padre e hijo?
8.-¿Qué proceso concluye antes su ejecución?
9.-¿Cuál es la primer instrucción que ejecuta el proceso hijo?
10.- Modifique el código del programa para que el proceso padre imprima en su mensaje de presentación (“Soy el porceso….”) antes que el proceso hijo imprima el suyo.
11.-Modifique el código del Ejemplo 1 declarando una variable entera llamada **varfork .** Coloque como valor inicial a esta variable el valor de 10. Dicha variable deberá incrementarse 10 veces en el padre y de 10 en 10, mientras que el hijo la incrementará 10 veces de uno en uno.
Anote cada una de las impresiones que realiza el programa, así como el valor final de la variable **varfork** para el padre y para el hijo y explique por que se tiene como salida dicha infomación.
12-. Escriba un programa cuyo nombre será arbolfork que permita generar el siguiente árbol de procesos



Cada proceso deberá imprimir un mensaje diferente

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para bifurcar procesos.

6. ANEXOS

7. REFERENCIAS

http://148.231.82.20/~so