Instituto Politécnico Nacional

Escuela Superior de Cómputo

Asignatura	**Sistemas operativos**
Prof.	**Ana Belem Juárez Méndez**
Práctica 2	**Procesos**
Entrega	**5 de septiembre del 2019**

Objetivo

Aplicar los conocimientos sobre la programación de procesos en situaciones que se pueden presentar en el Sistema Operativo UNIX- LINUX.

Introducción

Un programa es una colección de instrucciones y de datos que se encuentran almacenados en un archivo. Un proceso es un programa cargado en memoria listo para ejecutarse o en ejecución. Un proceso se compone de tres bloques fundamentales, llamados segmentos:

- segmento de texto
- segmento de datos
- segmento de pila

Cuando un proceso es creado, el sistema operativo le asigna un identificador único llamado PID. En el sistema operativo UNIX, todos los procesos, a excepción del proceso 0 (creado por el núcleo cuando arranca el sistema), son creados por la llamada al sistema `fork`. El proceso que invoca a fork, se llama proceso padre y el proceso creado es el proceso hijo. La sintaxis de `fork` es:

```
pid_t fork();
```

La llamada a fork hace que el proceso actual se duplique. En la programación concurrente, una situación común es que el proceso padre espere a que termine su hijo antes de continuar su ejecución. Para sincronizar los procesos padre e hijo se ocupan las llamadas `exit` y `wait`.

La sintaxis de `wait` es:

```
pid_t wait(int *stat_loc);
```

Wait suspende la ejecución del proceso que la invoca hasta que alguno de sus procesos hijo termina. Wait regresa el pid del proceso hijo que halla terminado su ejecución y en `stat_loc` se almacena el valor que el proceso hijo le envía al proceso padre.

La sintaxis de `exit` es:

```
void exit(int status);
```

Exit termina la ejecución de un proceso y le devuelve el valor de `status` al sistema.

La manera de invocar a un programa desde otro, es mediante la llamada `exec`. Existe un conjunto de funciones `exec` que obedecen al mismo funcionamiento. La sintaxis de este conjunto de funciones es:

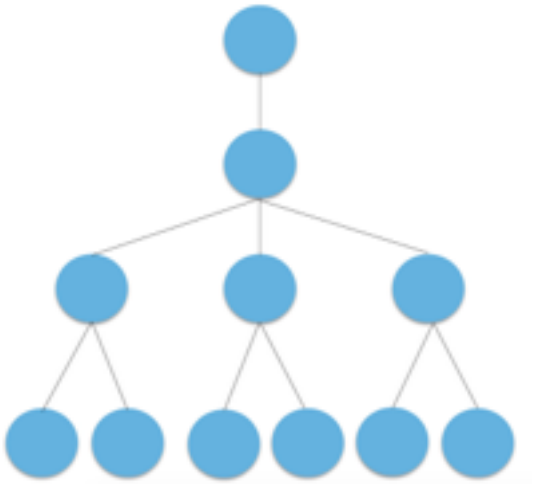
```
int execl (char *path, char *arg0, ... char *argn,(char *)0);

int execv(char *path, char *argv[]);

int execle(char *path, char *arg0, ... char *argn,(char *)0,char
*envp[]);

int execve(char *path, char *argv[], char *envp[]);

int execlp(char *file, char *arg0, ... char *argn,(char *)0);

int execvp(char *file, char *argv[]);
```

Donde `path` apunta a la ruta de un archivo ejecutable y `file` apunta al nombre de un archivo ejecutable. Los parámetros `arg0, … argn` son la lista de argumentos que se le pasan al nuevo programa, y son punteros a cadenas de caracteres. Después de `argn` se pasa un puntero `NULL` `((char *)0)` para indicar el final de los parámetros. El parámetro `argv` es un arreglo de cadenas de caracteres que forman la lista de parámetros recibida por el nuevo programa, el ultimo elemento de `argv` es un puntero `NULL` que indica el final de los parámetros. El parámetro `envp` es un arreglo de punteros a cadenas que definen el entorno en el que se ejecutará el nuevo programa, este arreglo termina también con un puntero `NULL`.

Desarrollo

Ejercicio 1. • Realizar un programa que cree diez procesos hijos del mismo padre y que cada uno muestre el mensaje “Hola soy el proceso hijo N y mi pid es XXXX. El pid de mi padre es XXXX”. N es el conteo del uno al diez. El padre deberá esperar a sus hijos antes de terminar.

Ejercicio 2. Realizar un programa que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará tres procesos hijos más. A su vez cada uno de los tres procesos creará dos procesos más. Cada uno de los procesos creados imprimirá en pantalla el PID de su padre y su propio PID. El árbol de procesos de este programa se verá así:



Ejercicio 3. Realizar un programa que invoque a la orden `ls -al /usr/bin` con las funciones `exec` tal como se indica:

```
$./programa opción
```

Donde `opción`, puede ser:

- `-l`, invocar a la orden `ls` con la función `execl`.
- `-v`, invocar a la orden `ls` con la función `execv`.
- `-lp`, invocar a la orden `ls` con la función `execlp`.