Universidad Nacional Experimental Del Táchira
Departamento De Ingeniería En Informática
Sistemas Operativos Cod. 0425603t
Laboratorio 2 – Gestión de procesos

Gestión de procesos utilizando la shell de GNU/Linux

El principal objetivo del laboratorio es comprender el funcionamiento básico de los procesos en linux con la ayuda de la consola.

El directorio /proc

En el Linux existe un directorio llamado **/proc**. En este directorio se almacena la información del sistema y de cada uno de los procesos activos del mismo. Existe un sub-directorio por cada proceso y dentro del mismo se indica cuánta memoria ha consumido el proceso, los archivos abiertos, sus conexiones, entre otros. Generalmente los comandos relacionados con la monitorización local o remota (**ps, top**) obtienen su información de este directorio.

Para cumplir con este objetivo, se usará el comando **ps** con distintas opciones:

- **ax**: muestra todos los procesos activos en el sistema.
- **u**: muestra la identidad del usuario que creó los procesos.
- **f:** muestra las relaciones padre-hijo en la jerarquía de procesos.

Ejemplo:

```
$ ps -auxf
```

Para probar el efecto de la ejecución del comando **ps**, repita después de ejecutar varias veces un navegador de Internet, abrir varias aplicaciones. A continuación observe la salida de **ps –axuf**, estudie la jerarquía de procesos y responda a las siguientes preguntas:

- ¿Cuántos procesos hay en ejecución en el Sistema?
- ¿Cuántos procesos son del usuario root?

- ¿Cuál es el proceso que más tiempo de CPU ha consumido?
- ¿Qué procesos llevan más tiempo arrancados?
- ¿En qué fecha y hora arrancó el Sistema?

Ejecutar el comando **pstree** y describir la salida del mismo.

Como ya debemos saber, un proceso es un programa en ejecución con recursos asignados. También sabemos que un proceso puede tener distintos estados que se pueden clasificar según distintos criterios, algunos de los cuales no son incompatibles entre sí. Como ahora lo que nos interesa es el aspecto puramente práctico, vamos a distinguir tres estados:

1. **Primer plano**: Un proceso que se ejecuta bloqueando para él la terminal desde la que se lanzó. Un proceso se lanza en primer planos simplemente introduciendo su nombre (y la ruta de acceso si fuera necesario) en el indicador de la línea de órdenes y pulsando Enter.
2. **Segundo plano**: Un proceso que se ejecuta sin bloquear la terminal, aunque sí puede escribir en ella los resultados de su ejecución. Un proceso se lanza en **segundo plano** poniendo al final de la línea de órdenes el símbolo **&** separado por al menos un espacio del nombre del programa.
3. **Detenido**: Podemos detener un proceso y que se quede en espera en el sistema hasta que demos la orden para que continúe su ejecución. En el presente texto nos vamos a referir a procesos detenidos por orden directa del usuario. No vamos a hacer referencia a procesos suspendidos por causas internas del sistema operativo.

Además tenemos que tener en cuenta otra característica: cada proceso tiene un propietario que generalmente es el usuario que lo ejecuta. Además al proceso se le aplican los mismos permisos que tenga el usuario propietario, es decir el proceso sólo podrá acceder a la información a la que pueda acceder el propio usuario con sus permisos. Con un proceso en primer plano podemos realizar dos acciones desde la terminal que tiene asociada:

1. Matar el proceso: **(Ctrl-c)** Sólo podremos matar procesos sobre los que tengamos permiso. Si intentamos matar el proceso de otro usuario el sistema no nos lo permitirá, salvo a root.
2. Parar el proceso: **(Ctrl-z)** En el primer caso se cancela el proceso y se liberan todos los recursos que tuviera asignados. En el segundo caso sólo

se detiene la ejecución del proceso, conservando su estado y sus recursos para poder continuar en el momento que se dé la orden adecuada.

Experimentar con los siguientes comandos:

- Con la orden **jobs** podemos obtener una lista de los trabajos lanzado en el sistema.
- La orden **fg** se utiliza para traer a primer plano un trabajo en segundo plano, bien esté activo o bien esté detenido.
- La orden **bg** se utiliza para poner en ejecución en segundo plano un trabajo que está en segundo plano detenido.
- La orden nohup lanza un proceso y lo independiza del terminal que estamos usando.
- Con la orden **kill** podemos mandar señales a los procesos en Linux. (CONT, STOP, INT)
- Con la orden **killall** podemos matar procesos que tengan el mismo nombre.
- Investigar como matar un proceso en Windows desde la línea de comandos.

GESTION DE PROCESOS USANDO LENGUAJE C

En esta práctica se estudiarán los principales servicios que ofrece POSIX para la gestión de procesos: Identificación, Entorno y Creación de un proceso.

Identificación de procesos: POSIX, identifica cada proceso por medio de un entero único denominado identificador de proceso de tipo **pid_t**. Los servicios relativos a la identificación de los procesos son los siguientes:

- **pid_t getpid(void)**; // Devuelve el identificador del proceso que realiza la llamada
- **pid_t getppid(void);** //Devuelve el identificador del proceso padre.

Cada proceso lleva asociado un usuario que se denomina propietario, cada usuario en el sistema tiene un identificador único denominado identificador de usuario de tipo **uid_t**. El proceso también tiene un identificador de usuario efectivo, que determina los privilegios que un proceso tiene cuando se encuentra ejecutando. El sistema también incluye grupos de usuarios del tipo **gid_t**, cada usuario debe ser miembro de al menos un grupo. Los servicios respectivos son los siguientes:

- **uid_t getuid(void);** //Devuelve el identificador de usuario real
- **uid_t geteuid(void);** //Devuelve el identificador de usuario efectivo
- **gid_t getgid(void);** //Devuelve el identificador del grupo real
- **gid_t getegid(void);** //Devuelve el identificador del grupo efectivo

Entorno de un proceso:

El entorno de un proceso viene definido por una lista de variables que se pasan al mismo en el momento de comenzar su ejecución. Estas variables se denominan variables de entorno y son accesibles a un proceso a través de la variable externa environ, declarada de la siguiente forma:

```
extern char **environ;
```

Esta variable apunta a una lista de variables de entorno. un vector de punteros a cadenas de caracteres de la forma nombre=valor, donde nombre

hace referencia al nombre de una variable de entorno y valor al contenido de la misma. Cada aplicación interpreta la lista de variables de entorno de forma específica. POSIX establece el significado de determinadas variables de entorno, las más comunes son: **HOME, LOGNAME, PATH, TERM, TZ**. Para obtener el valor de una variable de entorno se emplea el servicio **getenv** que retorna el valor de la variable o NULL si no está definida, con la siguiente sintaxis:

```
char *getenv(const char *name);
```

Creación de procesos:

- **Crear un proceso:** El servicio para la creación de procesos es el comando fork(); Éste clona el proceso que lo solicite, convirtiendo al solicitante en el proceso padre, y al nuevo en el proceso hijo. Devuelve un valor entero de tipo **pid_t**. El valor que retorna el sistema operativo como resultado del fork() es distinto en el hijo y en el padre:
  - El hijo recibe un 0.
  - EL padre recibe el identificador del proceso hijo.
  - Retorna -1 si el fork() no se ejecutó con éxito.
- **Ejecutar un programa:** el servicio para la ejecución de programas es el comando exec. A continuación se mencionan los prototipos del comando exec:

```
int exc1(char *path, char *arg, . . . );
int execv(char *path, char *argv[]);
int execle(char *path, char *arg, . . . );
int execve(char *path, char *argv[], char  *envp[]);
int execlp(char *file, const char *arg, . . . ):
int execpv(char *file, char *argv[]);
```

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;
        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;
                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 a las siguientes preguntas:

- ¿Por qué aparecen mensajes repetidos?
- Respecto al orden de terminación de los procesos ¿Qué observas? y ¿Por qué?

ASIGNACIÓN

Elaborar un programa en lenguaje C utilizando el estándar POSIX para gestionar procesos en GNU/Linux, el programa debe mostrar un menú al usuario para realizar las siguientes operaciones:

1. Ejecutar un nuevo proceso, donde permitira al usuario escribir una línea de comando a ejecutar, este proceso se debe lanzar en segundo plano o de lo contrario detendrá la ejecución de programa de la asignación.
2. Listado de procesos, donde se listan todos los procesos del sistema.
3. Matar un proceso, permite seleccionar de la lista el proceso que desea eliminar.

Nota: No puede utilizar la función system para ejecutar comandos de consola. Para el segundo punto, puede navegar por la carpeta **/proc** para obtener la información requerida.