Sistemas Operativos

TRABAJO PRÁCTICO LABORATORIO Nº 2: Señales, Llamadas al sistema.

Observación: Los códigos fuentes de los ejercicios de interpretación, se encuentran en nuestro repositorio en GitHub.

Ejercicios entregables: los ejercicios 1, 2, 3 y 4 deben ser entregados en la zona de entrega destinada a tal fin.

Llamadas al sistema a usar en este práctico: fork, exec, open, write, read, pipe, dup, signal, exit y otras.

1. Dado el siguiente código fuente:

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5
6
7 int cant = 0;
8
9 void manejador(int num) {
10     printf("Recibí señal);
11     cant++;
12     if (cant == 3) {
13         printf("Finalizo mi ejecución");
14         exit(1);
15     }
16 }
17
18 int main(void) {
19     signal(2, manejador);
20     while (1);
21 }
```

a. Describir qué hace.
b. Modificar el script de forma tal que capture **SIGINT** e imprima el mensaje ***"Se recibió SIGINT"*** cada vez que recibe esa señal.
    i. ¿Es posible matar el programa con **<CTRL+C>**? ¿Por qué? ¿Cómo hacer para matarlo?
    ii. ¿Se pierde el manejador de la señal cuando ésta se recibe más de una vez? ¿Por qué?
    iii. ¿Es posible capturar SIGKILL? ¿Por qué?

2. Escriba un programa que comience esperando la señal **SIGINT**. Si al cabo de 5 segundos no recibió esa señal, debe imprimir ***"Fin"*** y terminar la ejecución. Si dentro de los 5 segundos recibe la señal **SIGINT** debe imprimir ***"SIGINT recibido"*** y esperar nuevamente la señal por otros 5 segundos y así sucesivamente. Es decir que mientras esta señal llegue a intervalos de tiempo menores a 5 segundos, el programa se mantiene

vivo. Cuando la señal demore 5 segundos o más en llegar, el programa terminará. Para el control de tiempos debe utilizar la señal **SIGALRM**.

3. Suponga que un programa configuró una alarma para ser interrumpido dentro de 30 segundos. Transcurridos algunos segundos ocurre un evento por el cual la lógica del programa indica que el mismo ya no debe ser interrumpido por la alarma. Una forma de evitar tal interrupción es invocando a **alarm(0)**.

¿Con qué otra sentencia podría evitarse la interrupción?

Escribir dos programas que hagan uso de los dos métodos, verificando si el comportamiento es equivalente en ambos casos.

4. Dado el siguiente código fuente:

```
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <sys/wait.h>
 4
 5 #define TRUE 1
 6
 7 int main() {
 8    int status;
 9
10    if (fork() != 0) /* Código del padre  */
11    {
12       printf("\nPADRE\n");
13       waitpid(-1, &status,0);
14    }
15    else
16       /* Código del hijo  */
17       execl("/usr/bin/firefox", "",NULL);
18
19    return 0;
20 }
```

a. Describir qué hace. (**Observación:** asegurarse de contar con el binario "/usr/bin/firefox". En caso de no tenerlo, reemplazar esa cadena por otra conocida y existente en el mismo directorio en el filesystem).
b. Modificar el script de forma tal que ejecute un ***ls -ltr*** usando ***execv*** para enviar comando y opciones.

5. Escriba un programa que muestre una pantalla de inicio de sesión con el mensaje **"login: "** y se quede esperando un string por parte del usuario. Cuando el usuario ingresa una cadena, el programa debe almacenar la línea ingresada en una variable en memoria y luego mostrar el mensaje **"password: "**, para quedarse nuevamente esperando que el usuario tipee algo y presione <ENTER>.

Una vez que el programa escribe **"password: "**, sólo esperará por 15 segundos. Si al cabo de este tiempo el usuario no ingresó ningún password, el programa debe imprimir

"Timeout" y finalizar su ejecución. Para el control de tiempos debe utilizar la señal **SIGALRM**. Si el usuario ingresa un password y presiona <ENTER> antes de los 15 segundos, debe cancelarse la alarma pendiente y mediante un **fork** y un **exec** ejecutar (en el hijo) **/bin/bash**. Por su parte el padre quedará esperando que termine la ejecución del hijo y cuando esto ocurra volverá a la pantalla de inicio de sesión, a pedir un nombre de usuario.

Observación: Por simplicidad, no se hace nada con el username y password ingresados.

6. Desarrolle su propia terminal (un shell o intérprete de comandos) que implemente las siguientes funcionalidades:

- **b.** Ejecutar procesos (comandos) con sus respectivos argumentos.
- **c.** Permitir conectar la salida estándar de un proceso con la entrada estándar de otro (ej: ps | grep "bash").
- **d.** Explique detalladamente (sin desarrollar el código fuente) cómo programaría un proceso para ejecutar comandos condicionalmente (usando && y ||).

Observación: Para resolver el ejercicio debe investigar las llamadas al sistema **fork, exec, dup, pipe,** entre otras.