Laboratorio

SHELL

Ciencias de la Computación VII

El objetivo de este laboratorio es aplicar los conceptos básicos de ejecución de procesos de los anteriores laboratorios, utilizando los syscall que nos provee el sistema operativo.

Introducción:

Un shell provee un **prompt** donde es ingresado el siguiente comando a ejecutar.

Por ejemplo, **cat** comando que muestra en pantalla el contenido del archivo prog.c.

```
sh>  cat prog.c
```

Una técnica para implementar un Shell es tener un **parent process** que lee comandos del teclado y luego crea un **child process** independiente quien se encarga de la ejecución del comando.

El parent process debe **esperar** hasta que el child process termine.

Sin embargo, los **Unix Shell** también permiten que el child process se ejecute en background (o concurrente con el shell) si se especifica un carácter **&** al final del comando.

Por ejemplo, el child process se crea utilizando el **syscall fork()** y los comandos de usuario se ejecutan utilizando uno de los syscalls de **la familia exec()**

```
sh> cat prog.c &
```

Descripción de Laboratorio:

Al final de la descripción de la práctica se le provee un programa en C que realiza las operaciones básicas de un command line shell. Este programa está compuesto de dos funciones principales: **main()** y **setup()**.

La función **setup()** lee los comandos del usuario (los cuales pueden contener hasta 80 caracteres), y luego aplica un parser para el comando separando los tokens que serán utilizados para llenar un vector con los argumentos a utilizar para la ejecución del comando. (Recuerde que si el comando debe correr en background, el mismo debe terminar con el carácter **'&**, por lo que la función **setup()** modificará un parámetro que le indicará al **main()** cómo debe ejecutar dicho comando. **Este programa finaliza** cuando el usuario se presiona **<Control><D>**, por lo que la función **setup()** invoca el syscall **exit()**.

La función **main()** presenta el command prompt *COMMAND>* y luego invoca a **setup()**, la cual **espera** por un comando por parte del usuario. El contenido del comando ingresado será cargado en el arreglo args.
Por ejemplo, si el usuario ingresa **ls –l**

args[0] sería igual á: **ls**

args[1] sería igual al string: **–l**

Por "string" entendemos que es una variable C-style string donde se encuentra null-terminated.

Creando el Child Process que ejecute el comando

Es implementar la función **setup()** que lee y aplica un parser para los comandos del command prompt. Luego debe modificar la función **main()** de tal forma que al retornar de la función **setup()** se levante un **child process** que se encargue de ejecutar el comando especificado por el usuario. *¡Intente jugar y definir sus propios comandos!*

En este caso, el child process debe ejecutar el syscall:

```
execvp(char *command, char *params[]);
```

Donde "command" representa el **comando** a ejecutar y "params" almacena los **parámetros** para dicho command. Para este proyecto **execvp** debe ser invocado de la forma **execvp(args[0],args)**. Asegúrese de revisar si el usuario ha seteado que la ejecución del comando debe ser en background

o no, de tal manera que su interfaz Shell espere o no a que la ejecución de un comando termine.

History Feature

Es agregar el history feature a su programa Shell. Este feature permite al usuario acceder a los últimos 10 comandos ingresados.

Si el usuario presiona **<Control><C>** debe poder listar los comandos que se encuentran en el historial del Shell. **<Control><C>** es el signal **SIGINT**.

Signals

Los sistemas Unix utilizan los **signals** para notificar a un proceso sobre la ocurrencia de un evento en particular. Los **signals** pueden ser sincrónicos como asincrónicos, dependiendo del origen y la razón del evento.

Cuando un **signal** ha sido generado

Por ejemplo:

División por cero

Acceso ilegal a memoria

Usuarios ingresando **<Control><C>**

El **signal** es entregado al proceso donde se debe manejar.

Un proceso puede manejar un signal por una de las siguientes técnicas:

Ignorar el signal

Utilizar el default signal handler

Proveer una función específica para el signal-handler

Los **signals** pueden ser manejados por una estructura C, struct sigaction y luego utilizando la función sigaction().
Los signals están definidos en /usr/include/sys/signal.h.
Por ejemplo, SIGINT representa el signal para terminar un programa, asociado a la secuencia <Control><C>. Por lo que el default signal handler terminaría el proceso.

Alternativamente, un programa puede elegir por su propio **signal handler**, estableciendo el nombre de la función específica en el campo sa_handler del struct sigaction. Luego al invocar sigaction() se pasa como parámetro el signal que manejar y un puntero a struct sigaction.

Al final del documento presentamos un programa en C que muestra la función handle_SIGINT como el handler para el signal SIGINT.

De momento esta función muestra el mensaje "Caught Control C" y luego invoca al syscall exit(). Debemos utilizar **write()** en vez de realizar el output con **printf()**, debido a que el primero es **signal-safe**, indicando que puede ser llamado desde una función que se encarga del **signal-handling**, además de otras garantías no presentes en el **printf()**. Este programa correrá dentro de un while(1) hasta que el usuario ingrese la secuencia **<Control><C>**. Cuando esto ocurre la función **handle_SIGINT()** será invocada.

La función que se encarga del signal-handling debe estar declarada arriba del main() como esta función puede ser invocada en cualquier momento, entonces no recibe ningún parámetro.

Pruebe la ejecución del programa Signal-Handling, ya que le dará la información necesaria para implementar el history feature de su práctica de laboratorio.

Si el usuario presiona **<Control><C>**, entonces su **signal** handler debe listar los últimos 10 comandos ingresados.
Con dicha lista, el usuario podrá re-ejecutar un comando anterior, ingresando **r x** donde **x** es la primera letra del comando.
Si hay más de un comando que comienza con **x** ejecute el comando más reciente.
Además, si el usuario solo ingresa **'r'** debe ejecutar el comando más reciente en la lista. Asuma que sólo un espacio separa la **'r'** de la primera letra del comando.

Cada comando que se ejecute de esta manera **debe ser impreso en pantalla** y luego debe invocar al syscall execvp() para ejecutar el comando.

Entrega:

- Este Laboratorio tiene que ser realizado en **C.**
- La calificación será realizada en Linux.
- El laboratorio debe ser entregado por medio del GES, con los archivos en un **ZIP.** No se calificarán laboratorios entregados tarde o por medio de URL externo y tendrá una nota de cero.
- El laboratorio debe de contener un archivo **Makefile** para su ejecución, ya sea que requiera o no cargar librerías para realizar la compilación, de lo contrario no se calificará, dentro de la carpeta de su laboratorio debe poder ejecutarse el comando **make** para compilar y ejecutar su laboratorio de lo contrario tendrá una nota de cero.
- El laboratorio puede tener una calificación de cero si no compila o se ejecuta.