Universidad Nacional
Facultad de Ciencias Exactas y Naturales
Escuela de Informática

EIF212 – Sistemas Operativos
Máster Jose Pablo Calvo Suárez

Tarea Programada #1 – Valor 10%

Objetivos

- Describir el uso de APIs para creación y gestión de procesos
- Implementar técnicas de programación asociadas al uso de APIs del sistema
- Implementar una aplicación utilizando el lenguaje C estándar en Linux
- Utilizar Pipes para la comunicación entre procesos
- Desarrollar el código bajo el estándar POSIX

Descripción

El proyecto consiste en diseñar un programa en C que sirva como una interfaz de línea de comandos (Shell), el cual acepta comandos de usuario y luego ejecuta cada comando en un proceso separado. Éste proyecto debe ser desarrollado en un ambiente Linux.

La interfaz de Shell proporciona un "**prompt**" al usuario, después del cual un comando es ingresado. El ejemplo muestra el prompt **tiger>** y el comando proporcionado por el usuario *cat ejemplo.c*

tiger> cat ejemplo.c

Una técnica para implementar la interfaz Shell es permitirle al proceso padre leer lo que el usuario digita en la línea de comandos (*cat ejemplo.c*) y luego crear un proceso hijo separado que se encarga de la ejecución del comando. A menos de que se especifique de otra manera, el proceso padre espera a que el proceso hijo termine la ejecución del comando antes de continuar. Sin embargo, shells de UNIX también permiten que el proceso hijo se ejecute en segundo plano, o concurrentemente. Para lograr ésto, se agrega un '**&**' al final del comando. Por ejemplo:

tiger> cat ejemplo.c &

El proceso hijo es creado utilizando la llamada al sistema ***fork***, y el comando ingresado por el usuario se ejecuta con la llamada al sistema ***exec***. Tome en cuenta que si el usuario digita el carácter '**&**', el proceso hijo debe correr en el background, en cuyo caso el proceso padre (el shell) recupera el control inmediatamente (es decir, el usuario podría digitar más comandos mientras el proceso hijo se está ejecutando). Note que el carácter '**&**' solamente se coloca al final del comando.

Como mecanismo de comunicación entre procesos, utilice **tuberías (pipes)**. El shell debe implementar la opción de correr un único **pipe**. Por ejemplo:

tiger> cat ejemplo.c | less

La salida de **cat** sirve como entrada para **less**. En este caso, debe redirigir la salida de **exec** de **stdout** hacia el proceso padre o un archivo. Lo más sencillo es utilizar un **pipe**. De esta manera la salida del comando pasa al padre nuevamente, el cuál usará la salida para invocar al siguiente proceso hijo, pasándole la información de salida del proceso hijo anterior.

Además de la funcionalidad descrita, el proyecto debe implementar un historial de comandos, que le permita al usuario acceder a los comandos que se introdujeron más recientemente. Los comandos deben ser numerados de manera secuencial iniciando en 1, y la secuencia de números debe seguir incrementándose. Por ejemplo, si el usuario digitó 35 comandos, los comandos estarán numerados del 1 al 36.

El usuario deberá poder ver el historial digitando el comando ***historial*** desde el prompt. A manera de ejemplo, suponga que el usuario digitó los siguientes comandos (de más a menos reciente):

ps, ls -l, top, cal, who, date

El historial mostrará lo siguiente:

6 *ps*
5 *ls -l*
4 *top*
3 *cal*
2 *who*
1 *date*

Además, el programa debe técnicas para obtener comandos de manera individual desde el historial. Cuando el usuario digita el signo de admiración ! seguido de un número N, el enésimo comando del historial es devuelto. Por ejemplo, !3 devolvería el tercer comando existente en el historial.

Note que debe manejar situaciones de error (como por ejemplo, si no existen comandos en el historial, o si no se proporciona un número con el signo de admiración !)

Notas adicionales

- Utilice la función '**gets**' o '**gets_s**' para capturar el comando del usuario. Este comando captura toda la secuencia de entrada y la almacena en la cadena especificada (incluido los espacios) hasta que se presiona ENTER.

- Una vez que captura los datos, debe "parsearlos" – un comando de usuario puede estar conformado por: el nombre del programa a ejecutar, parámetros (opcionales) y el carácter '**&**' (opcional), además del **pipe '|'** (opcional). No es necesario que valide el nombre del programa, pero sí es importante que verifique la posición de los caracteres '**&**' y '**|**'. La idea de "parsear" el comando es poder separar cada una de las entradas, de manera que pueda hacer la llamada a **exec** correctamente y además determinar si debe utilizar **pipes** o correr el comando en el background. Se recomienda almacenar el nombre del programa en una variable tipo apuntador a cadena de caracteres y los parámetros en un arreglo de apuntadores a cadena (cada posición del arreglo contiene un parámetro).

- El nombre del programa junto con los parámetros se ejecutarán mediante una llamada a 'exec' (para más información, vea **man exec**), mientras que los caracteres '**&**' y '**|**' debe manejarlos en la lógica de su programa.

- La implementación del shell va a tener dos comandos tipo "**built-in**": **exit** el cual terminará la ejecución del shell y el comando **historial**. Investigue la diferencia entre un "**built-in**" del shell a un comando regular.

- Recuerde que el shell se ejecutará de manera indefinida y permitirá la ejecución de comandos hasta que el usuario digite **exit**.

Observaciones

- Utilice el lenguaje de programación C bajo ambiente Linux.
- La tarea debe ser realizada en grupos de tres personas como máximo.
- Se debe entregar el proyecto con el código fuente (un único archivo .c).
- La fecha límite para la entrega es el día domingo 14 de mayo a las 11:59pm.
- Es posible que se soliciten adelantos del proyecto en cualquier momento.
- En caso de copia (dos proyectos o más con mucha similitud que lo demuestren) o plagio (códigos descargados de Internet, libros o cualquier otro material), la nota de la tarea es 0.
- Debe indicar todos los recursos consultados para le elaboración del proyecto.