Universidad de Antioquia
Facultad de Ingeniería
Sistemas Operativos y Laboratorio

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Ejercicio de Programación
Implementación Intérprete Básico (shell)

1. Introducción: ¿Qué es un intérprete de línea de comandos?

Un intérprete de línea de comandos (shell) es un programa que permite a un usuario ingresar comandos o instrucciones para que sean ejecutados. Luego de ingresar el comando y la tecla "enter", el programa analiza la secuencia de caracteres ingresada (parser) y la ejecuta utilizando una serie de programas previamente compilados. Un intérprete de comandos sencillo muestra en la salida estándar (pantalla) un **prompt** y ejecuta las órdenes introducidas por teclado (ver Figura 1).

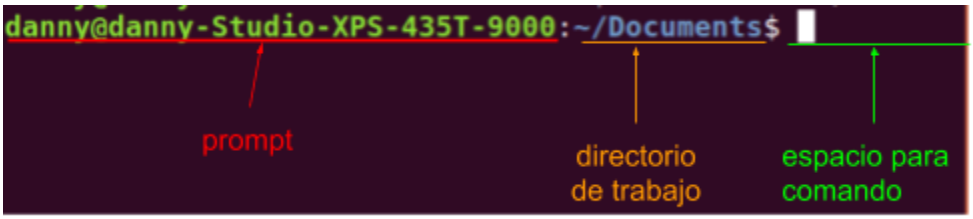

Figura 1. Ejemplo de un intérprete de línea de comandos

Uno de los más populares intérpretes de el sistema operativo Linux es el bash (bourne-again shell). En linux, la mayor parte de programas que se pueden ejecutar en el bash se encuentran en la ruta "/bin". Por ejemplo, cuando en se tecla en la consola el comando "ls -l /home", el intérprete llama el programa "/bin/ls" y le pasa los argumentos "-l" y "/home". A continuación se presenta un ejemplo del uso del programa ls en la consola de linux:

Figura 2. Ejemplo del comando ls

2. Tarea de Programación

El desarrollo de un intérprete de línea de comandos es un buen ejemplo para entender el uso

de las llamadas al sistemas y de los procesos. En particular, este ejercicio nos permitirá familiarizarnos con un caso típico en el uso de los llamados al sistema para el manejo de procesos: fork, wait y exec.

La idea es que en parejas, desarrollen un intérprete de línea de comandos propio llamado “udea-shell”. Nuestro udea-shell podrá ejecutar todos los comandos externos que se encuentran en la ruta “/bin”. No es necesario realizar programas para los comando externos, solo se debe crear la interfaz para poder ejecutarlos.

Adicionalmente, el udea-shell debe implementar los siguientes comandos internos:

Comando	**Descripción**
`pwd`	Imprime en pantalla el directorio de trabajo actual (tal y como lo hace el comando pwd). **Tip**: Para implementar este comando use la llamada getcwd.
`cd`	Cambia el directorio de trabajo del sheel, Tip: Para implementar este comando use la llamada al sistema chdir.
`echo <msg>`	Imprime el mensaje **`msg`** en pantalla**.** **Tip**: Basta con usar printf para implementar este comando.
`clr`	Limpia pantalla.
`time`	Imprime el tiempo actual del sistema **Tip**: Para la implementar este comando use la llamada time.
`exit`	Finaliza la ejecución del intérprete

Las órdenes (comandos) introducidas por el usuario podrán recibir cualquier número de parámetros. La función para obtener el número de parámetros (parser) le será entregada de modo que lo único que deberá hacer es usarla no implementarla. El intérprete debe permitir la ejecución de órdenes de segundo plano (**&**).

2.1. Ciclo de ejecución de una orden del intérprete

El ciclo de ejecución de una orden del intérprete, consta de las siguientes etapas:

I. **Desplegar el prompt**: El prompt es la cadena de caracteres (normalmente terminada con un carácter como %, # o >) que se despliega cuando el intérprete es ejecutado.
II. **Obtener la línea de comandos**: Para obtener una línea de comandos, el intérprete realiza una operación de lectura bloqueante (scanf), de manera que el proceso que ejecuta el intérprete estará dormido hasta que el usuario digite el comando que está esperando. El comando introducido por teclado queda almacenado en una cadena de caracteres para ser posteriormente procesado.

III. **Procesamiento del comando (parser)**: El objetivo del parser es procesar el comando para generar la orden asociada. Básicamente lo que hace el parser es escanear el comando digitado de izquierda a derecha, separa todas las palabras que se encuentran entre los espacios en blanco (espacio, tab o NEWLINE). La primera palabra es el nombre del comando, las demás son sus parámetros.

IV. **Preparación de los parámetros**: En esta etapa el intérprete pasa los parámetros al comando como un arreglo de apuntadores a cadenas de caracteres (similar a la variable **argv**).

V. **Ejecución del comando**: En esta fase existen dos posibles situaciones dependiendo del tipo de orden:

A. **Órdenes internas**: Son aquellas órdenes implementadas como funciones propias del intérprete tal y como se muestra a continuación en la parte resaltada:

```
/** Comandos como funciones */
tipo_retorno orden1 (tipo args, ...);
...
tipo_retorno ordenN(tipo args, ...);


/** Dentro del main del intérprete*/
  ...
  /* Hacer el parsing de la entrada */
  num = separaItems (expresion, &items, &background);
  ...
  /* Obtener comando */
  if(ordenIngresada == orden1) {
     /* Lanzar el ejecutable asociado a la orden 1 */
     // Código...
  }
  ...
  else if(ordenIngresada == ordenN) {
     /* Lanzar el ejecutable asociado a la orden 1 */
     // Codigo: suponiendo que es interna…
     ordenN(parametros); // Como se llame dependerá si tiene o no &
  }
  ...
```

B. **Órdenes externas**: Son programas ejecutables almacenados en el disco. En este caso el intérprete realiza el llamado al sistema fork (crea un proceso nuevo intérprete hijo) y luego el proceso hijo es sobrescrito con un proceso asociado al programa solicitado. Los ejecutables de las órdenes externas se encuentran compilados en la carpeta “/bin” de un sistema operativo Linux.

```
/** Dentro del main del interprete*/
  ...
  /* Hacer el parsing de la entrada */
  num = separaItems (expresion, &items, &background);
  ...
  /* Obtener comando */
  if(ordenIngresada == orden1) {
     /* Lanzar el ejecutable asociado a la orden 1 */
```

```
    // Codigo...
  }
  ...
  else if(ordenIngresada == ordenN) {
     /* Lanzar el ejecutable asociado a la orden 1 */
     // Codigo: suponiendo que es externa…
     if (fork () == 0) {
        ...
        execv (...); // Acá va la invocación de la orden externa
        ...
     }
     ...
  }
  ...
```

VI. **Esperar a la finalización de la orden**: Esto depende de si la orden a ejecutar se solicitó, o no, en segundo plano. Si la orden **no** se solicitó para ejecución en segundo plano (es decir, sin &), entonces el intérprete debe esperar a que la orden finalice. En caso opuesto el intérprete puede seguir su ejecución normal en espera de una nueva orden.

El intérprete de comandos a implementar debe seguir estos pasos. Para leer lo introducido por el usuario desde el teclado se hará uso de la función separar ítems la cual se proporciona y cuya sintaxis es la siguiente:

```
int separarItems (char *expresion, char ***items, int background)
```

Dónde:

- **expresión**: Expresión que introduce el usuario en el intérprete de órdenes.
- **items**: Apuntador a un array de apuntadores a las cadenas de caracteres que componen cada uno de los elementos de la expresión.
- **background**: Entero que indica si se ha lanzado una orden para ejecución en segundo plano.

A continuación, se muestra un posible esqueleto de un intérprete[1]:

```
const char *mypath[] = {
  "./",
  "/usr/bin/",
  "/bin/",
  NULL
};

while (...) {
  /* Wait for input */
  printf ("prompt> ");
  fgets (...);
  /* Parse input */
  while (( ... = strsep (...)) != NULL) {
```

[1] Esqueleto tomado de: https://liacs.leidenuniv.nl/~rietveldkfd/courses/os2016/os2016-assignment1.pdf

```
    ...
  }

  /* If necessary locate executable using mypath array */
  /* Launch executable */
  if (fork () == 0) {
    ...
    execv (...);
    ...
  }
  else
  {
    wait (...);
  }
}
```

2. Recomendaciones

Es recomendable abordar el problema realizando pequeños pasos verificables. Se sugieren las siguientes etapas de desarrollo:

I. Crear un programa que muestre de forma cíclica el **prompt** y lea una instrucción pedida al usuario.
II. Añadir la función **separarItems** obteniendo la orden a ejecutar.
III. Añadir la lógica para órdenes internas, es decir el conjunto de instrucciones que se implementaran en el cuerpo de la función asociado a cada una de las órdenes.
IV. Implementar el soporte para órdenes externas, esto es, el archivo de código fuente que implementará la función externa.
V. Añadir la posibilidad de ejecución en segundo plano.

3. Enlaces de interés

- https://www.cs.rutgers.edu/~pxk/416/notes/04-processes.html
- http://minnie.tuhs.org/CompArch/Lectures/week08.html
- https://syscalls.kernelgrok.com/
- http://people.cs.aau.dk/~adavid/teaching/Uppsala-OS/
- https://liacs.leidenuniv.nl/~rietveldkfd/courses/os2016/os2016-assignment1.pdf
- https://www2.cs.duke.edu/courses/spring14/cps110/projects/lab2/lab2.pdf
- https://liacs.leidenuniv.nl/~rietveldkfd/courses/os2016/os2016-assignment1.pdf
- http://www.it.uu.se/edu/course/homepage/datsyst1/vt06/lab1.pdf
- http://www.users.miamioh.edu/stephamd/elec377/pdf/E377A1.pdf
- http://www.rozmichelle.com/pipes-forks-dups/
- http://web.stanford.edu/class/cs140e/
- http://web.stanford.edu/class/cs140e/