UNC Universidad Nacional de Córdoba

FCEFyN Facultad de Ciencias Exactas, Fisicas y Naturales

TRABAJO PRÁCTICO Nº 2:

“myshell”

Cátedra: Sistemas Operativos 1

Año: 2019

Integrantes:

- Ferrero Alejandro (Matrícula n° 40054394)
- Pichetti Augusto (Matrícula n° 41018392)

Carrera: Ingeniería en Computación.

Profesores:

- Martínez Pablo
- Alonso Martín

Introducción

En este trabajo se desarrolló un programa en lenguaje c para la implementación de un shell de linux, independiente del propio shell del sistema operativo.

Desarrollo del proyecto

Este proyecto está dividido en 6 archivos .c acompañados de sus correspondientes headers que se fueron desarrollando incrementalmente. Estos son main.c, prompt.c, commands.c, program.c, batchFile.c y redirection.c. Cada uno contiene las funciones que se utilizaron para la implementación de los servicios particulares que ofrece el shell.

Los servicios que debe proveer myshell son:

- Proporcionar un prompt que contenga el camino al directorio actual.
- Soportar alguno comando internos como cd, clr, echo y quit.
- Invocación de programas.
- Debe ser capaz de tomar sus comandos a ejecutar desde un archivo.
- Debe soportar redirección de entrada/salida en stdin y/o stdout.
- Debe permitir ejecución en segundo plano.

Command line prompt

En esta instancia se genera un prompt clásico, similar al de la distribución Ubuntu, de la forma *nombreUsuario@nombreMaquina:~$.* Se genera a partir de la concatenación de las siguientes cadenas de caracteres:

- nombreUsuario se obtiene mediante el uso del comando whoami (de forma alternativa se podría escribir la variable de entorno mediante echo $USER). Dicha salida se imprime en un archivo temporal, el cual posteriormente es leído mediante la función leerArchivo().
- nombreMaquina se obtiene accediendo al directorio `/proc/sys/kernel/hostname`.
- Para obtener el camino al directorio actual se utiliza la función de las bibliotecas de c getcwd(), la cual escribe el nombre del directorio actual en una memoria dada pasada como parámetro.

Todo esto es realizado en la función prompt() del archivo prompt.c

Internal commands

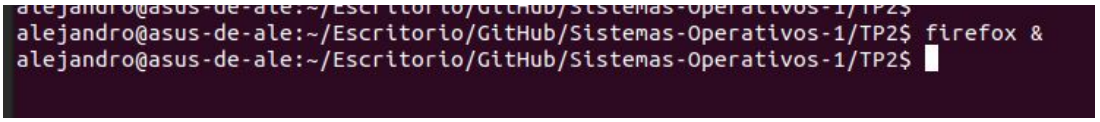
En este apartado, se realizó la implementación de los comandos internos del shell:

- cd <directorio>. La utilización de este comando cambia el directorio actual al especificado. Fue implementado utilizando la función chdir(const char *path), de la librería uinstd, está el directorio de trabajo actual, así como también la variable de entorno PWD.
- clr. Este comando limpia la pantalla. Esta implementado mediante el macro clrscr() el cual está definido con la impresión del código de escape ANSI “\033[1;1H\033[2J”, de manera que siendo \e el caracter escape:
    - "\033[1;1H" mueve el el cursor a la posicion (1,1).
    - "\033[2J" limpia la pantalla.
- echo <comentario>. Muestra el comentario especificado en una línea nueva, siendo múltiples espacios y tabs reducidos a un solo espacio. Esto se implementó utilizando la función eliminarEspacios(char linea*), que tal como su nombre indica, se encarga de eliminar los espacios o tabs que pudiesen encontrarse en el comandoingresado.
- quit. Cierra el shell. Este comando se implementa simplemente haciendo uso de la función exit(int status) de la librería stdlib, la cual finaliza el proceso que la invoca de forma inmediata.

Program invocation y Background execution

Todas aquellas entradas del usuario que no son comandos internos son interpretados como la invocación de un programa. Dicha funcionalidad está implementada en program.c. La función ejecutarPrograma() recibe el nombre del programa a ejecutar con sus argumentos y un flag que indica si el programa se debe ejecutar en segundo plano, es decir, se debe volver inmediatamente al prompt una vez lanzada la ejecución de dicho programa. Para el programa se ejecute en segundo plano se debe colocar un & (ampersand) al final de la línea de comandos.

Ejemplo de ejecución de un programa en segundo plano:



La función ejecutarPrograma() llama a crearProceso(), la cual duplica el proceso mediante fork(). Al proceso hijo se le encomienda la ejecución del programa deseado mediante la función execvp(). El proceso padre debe esperar por su proceso hijo mediante la función waitpid() en caso de que la ejecución del programa fuese en primer plano (sin &), sino retorna inmediatamente al prompt.

Batch File

En este apartado se implementa la posibilidad de la toma de comandos del shell mediante la ejecución de un archivo externo de la forma **myshell batchfile**.

Esto fue implementado, primero verificando si la ejecución del shell fue acompañado de algún argumento extra, esto puede comprobarse observando el estado de argv[1], el cual será NULL en el caso que la shell haya sido ejecutada sin argumentos. Caso contrario, se utilizará la función leerBatchFile(char *archivo), la cual abrirá el archivo especificado por el argumento y procederá a leer los comandos a ejecutarse del mismo. En caso que el archivo no exista se imprimirá un mensaje de error.

I/O redirection

El programa soporta la redirección de entrada/salida en stdin y/o stdout. Por ejemplo: program < inputfile > outputfile. Ejecuta la el programa ‘program’ , stdin es reemplazado por inputfile y stdout por outputfile

Esta implementación puede dividirse en dos partes, para la primera, que incluye toda redirección excepto para el comando interno echo si el comando ingresado contiene alguno de los caracteres ‘<’ o ‘>’, caso en el cual se guardará el programa a ejecutar, el input/output dependiendo de qué tipo de redirección se haya solicitado, y los posibles argumentos de la misma. Se creará un proceso hijo y este se encargara de ver de qué tipo de redirección se trata, y de la ejecución del programa solicitado por esta.

En el caso de ser una redirección de entrada, se abrirá archivo especificado por el input y se obtendrá su file descriptor utilizando la función open(const char *path, int oflags, mode_t mode) de la librería fcntl, el cual se duplicará en stdin utilizando la función int dup2(int fildes, int fildes2) de la librería unistd. En el caso que se trate de una redirección de salida, se abrirá/creará un archivo especificado por output utilizando la función creat(const char *pathname, mode_t mode) de la libreria fcntl, y se obtendra su file descriptor, el cual se duplicará en stdout utilizando la función dup2. En ambos casos posteriormente se ejecutará el programa correspondiente utilizando la función execvp(const char *file, char *const argv[]) de la librería unistd.

Para la redirección utilizando el comando interno echo, se hace uso de las funciones de las funciones chekEspacios() y separarPalabras() para dar el formato deseado al comando, el cual luego será utilizado luego en la función redireccionar(char **) para encontrar los argumentos y el output deseado. Se realizará un procedimiento análogo a la redirección previamente descrita y mediante la utilización de un proceso hijo, se ejecutará la redirección requerida.