Diseño e Implementación de un SHELL

Proyecto #1 para la asignatura de sistemas operativos.

Jean Carlos Gomes Pinto
Escuela de Computación
Facultad de Ciencias, UCV.
Caracas, Venezuela
jeancarlosgomes@gmail.com

Miguel Angel Omaña Fernández
Escuela de Computación
Facultad de Ciencias, UCV.
Caracas, Venezuela
miguelomana@gmail.com

Resumen — Se presentan las motivaciones, consideraciones y criterios de implantación de un shell construido desde cero, utilizando el lenguaje C, apoyando en la llamada a sistemas y la creación de los procesos e hilos que posee el sistema operativo. Además se hará una breve explicación teórica de SHELL, con su reseña histórica y su mecanismo de ejecución de órdenes.

Términos de índice: sistema operativo, Shell, terminal de comandos, intérprete de comandos.

INTRODUCCIÓN

El usuario tendrá disponible un Shell (o terminal) de comandos donde podrá comunicarse con el núcleo del Sistema Operativo. Un Shell o Intérprete de Comandos es un programa de servicio que envía comandos de la forma correcta al núcleo (o kernel). El shell protege lógicamente al usuario, el núcleo y el resto de programas basados en él.

Los entornos gráficos para cualquier sistema operativo han evolucionado mucho en los últimos años. Se puede trabajar utilizando el sistema operativo X sin tener que abrir el intérprete de comandos conocido como Shell.

Sin embargo, es recomendable aprender cómo trabajar desde el intérprete de comandos de la Shell, porque usar los comandos mediante esté, puede ser muy rápido. En el tiempo que tarda en abrir el gestor de fichero, buscar el directorio, crear o modificar ficheros, utilizando la Shell, podría haber acabado con varios comandos.

En esta sección, le mostraremos cómo diseñar e implementar un Shell, como también como el Shell me permite manipular archivos y ejecutar tareas básicas.

Existe un componente de su sistema operativo vital para trabajar que es el intérprete de comandos, los usuarios escriben comandos en él y la Shell indica al sistema operativo cómo proceder. Muchos usuarios nuevos trabajan mayoritariamente en modo gráfico antes que con una Shell, pero existen algunas tareas que no podrá realizar en modo gráfico. Los usuarios experimentados pueden escribir scripts de la Shell para expandir sus habilidades aún más. Nos hemos referido con frecuencia a la Shell, también conocida por en el "intérprete de comandos de comandos" o "bash". Ha llegado el momento de aprender algo más sobre esta herramienta indispensable.

I. PLANTEAMIENTO DEL PROBLEMA

Muchas veces se utiliza el intérprete de comando y se sabe manejar con algunos pocos comandos, sin conocer como él se encarga de proveer y acceder a los servicios del sistema operativo, este desconocimiento se debe en parte por falta de conocimiento e interese de cómo funcionan las cosas. No obstante, poder implementar y diseñar un intérprete de comandos permitirá facilitar la forma en que se invocan o ejecutan los distintos servicios, llamadas del sistema y programas disponibles en el computador, que permitirá ver al Shell como algo más que un simple programa.

II. MARCO TEORICO

a. El sistema de ficheros

Todo sistema operativo necesita guardar multitud de archivos desde los de la configuración del sistema, los de log, los de los usuarios, etc. En general, cada sistema operativo utiliza su propio sistema de ficheros, caracterizándolo en muchos aspectos como pueden ser el rendimiento, seguridad, fiabilidad, etc. Lo primero que se debe tener claro antes de hablar de sistemas de fichero es el concepto de sistema operativo.[1]

b. Sistema operativo:

Un sistema operativo es una parte importante de casi todos los sistemas de computación. Este último se puede dividir a grandes rasgos en cuatro componentes:
1. Hardware
2. Sistema Operativo
3. Programas de aplicación
4. Usuarios.

En general, es un programa que actúa como intermediario entre el usuario de un computador y el hardware de éste. El propósito del sistema operativo es crear un entorno en el que un usuario pueda ejecutar programas, es hacer al sistema de computador cómodo de usar y utilizar el hardware del computador de forma eficiente. [2]

c. Los procesos

El hecho que el sistema operativo sea multitarea implica que puede lanzar más de un programa a la vez. Un proceso no es más que un programa o aplicación cargado en memoria y ejecutándose.

d. Hilos

Un hilo en un sistema operativo es la característica que permite a una aplicación realizar varias tareas a la vez concurrentemente, los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autentificación. Esta acción permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. [2]

e. Un intérprete de comandos (Shell)

Un intérprete de órdenes o de comandos, es un programa informático que tiene la capacidad de traducir las órdenes que introducen los usuarios, mediante un conjunto de instrucciones facilitadas por él mismo directamente al núcleo y al conjunto de herramientas que forman el sistema operativo.

III. HISTORIA

La independencia de la shell respecto al sistema operativo, ha permitido desde los primeros pasos de UNIX, la creación de múltiples de shells, aunque solo algunas han conseguido un uso "popular".

La primera shell más importante fue la Bourne shell, nombre que adquirió de su creador Steven Bourne. Ésta fue incluida en la primera versión popular de UNIX, la Versión 7, sobre el año 1987. El nombre que recibe la Bourne en el sistema operativo es sh. A pesar de los cambios que UNIX pueda haber sufrido, la shell Bourne sigue siendo una de las más populares, dependiendo hoy en día de ella gran cantidad de utilidades.

La primera gran alternativa a esta shell fue la C shell, o también csh. Su creador, Bill Joy en la Universidad de California en Berkeley, la escribió como una parte del Berkeley System Distribution (BSD), versión de UNIX que salió un par de años después de la versión 7. Esta shell está incluida en todas las versiones de UNIX más recientes.

La C shell recibe su nombre debido a la semejanza de sus comandos a las instrucciones del lenguaje C, lo cual hace a la shell más fácil de aprender para los programadores.

En los últimos años han continuado surgiendo diversas shell, cuya popularidad ha ido en aumento. La más notable es la Korn shell, la cual incorpora las mejores cualidades de la shell Bourne y de la C. El único inconveniente de esta shell es que se distribuye de forma comercial. Debido a esta característica otra shell ha tenido un mayor auge, la bash shell, de características semejantes a la korn, pero de distribución gratuita.

The Bourne Again Shell

La Bourne Again Shell, o también bash, fue creada para uso del proyecto GNU. La intención era que bash fuera la shell estándar de sistema GNU. Su "nacimiento" fue el Domingo 10 de Enero de 1988, y su creador fue Brian Fox, el cual continuó con el desarrollo de esta shell hasta 1993. En 1989 Chet Ramey fue el encargado de numerosas correcciones en la shell, e incluyó muchas nuevas características a ésta, convirtiéndose en el desarrollador oficial a partir de 1993.

La principal característica de esta shell, y seguramente, la que le ha dado un mayor auge, al menos en un primer golpe, es el modo de edición en linea de comandos. Con este modo es mucho más fácil volver atrás y corregir errores de escritura en los comandos en vez que tenerlos que volver a introducir completamente. La otra gran ventaja de esta shell frente a las otras es el control del trabajo, con lo que el usuario es capaz de parar, comenzar y pausar cualquier numero de comandos a la vez.

El resto de características de esta shell, son, en su mayoría, para programadores, como por ejemplo la inclusión de nuevas variables para la personalización del entorno, instrucciones avanzadas de I/O o mayor cantidad de estructuras de control, entre otras muchas. [3]

IV. OBJETIVOS DEL PROYECTO

- Entender el uso del intérprete de comando y como este trabaja.
- Proveer un sólido entendimiento teórico en la implementación de un Shell, con su reseña histórica y su mecanismo de ejecución de órdenes.
- Aprender comandos y sintaxis del intérprete de comando.
- Adquirir destrezas en el manejo de procesos, hilos, llamadas al sistemas y señales
- Diseñar, construir la implementación de un intérprete de comando.

V. DISEÑO DE LA SOLUCION Y DETALLE DEL DESARROLLO

Para el diseño e implemente del shell se creó un proceso padre el cual espera por una orden al usuario, una vez el usuario ingresa la orden, el proceso padre la recibe y crea un proceso hijo el cual la analiza y la ejecuta, simulando tal cual cómo funciona un Shell en un sistema operativo.

El shell es un programa que básicamente realiza las siguientes tareas:

```
for(;;)    {
    Imprime indicador de ordenes;
    Lee la línea de órdenes;
    Analiza la línea de ordenes (arg0,arg1,...,>,<,|,&,...);
    Prepara entorno según lo que aparece en línea de ordenes;
    Crea un proceso para ejecutar orden;
        if (estamos en el proceso hijo)    {
            Ejecuta la orden dada en arg0;
        else /* es el proceso padre */
            if (línea ordenes no aparece el símbolo &)
                espera hasta que finalice el hijo;
        }
    }
```

Cada shell, además de ejecutar las órdenes de Linux, tiene sus propias órdenes y variables, lo que lo convierte en un lenguaje de programación. La ventaja que presenta frente a otros lenguajes es su alta productividad, una tarea escrita en el lenguaje del shell suele tener menos código que si está escrita en un lenguaje como C.

Como ya se mencionó un Shell consta en implementar estas cinco funciones principales para poder diseñar e implementar el Shell, el cual consta en una función para mostrar el indicador, otra para leer la orden que el usuario ingresa, una para buscar la ruta, también una función para analizar el comando y por ultimo una para analizar la ruta donde se encuentra el comando. Si vemos más a detalle estas cinco funciones tengo que:

a. Mostar indicador:

Se construye el indicador que es una simple cadena de caracteres que identifica el nombre de la máquina o el directorio actual donde se encuentra ubicado. Este indicador posee el nombre del usuario seguido por dos caracteres especiales.

```c
void mostrarIndicador()
{
    char indicador[] = "JeanC-Miguel@JeanC-Miguel-Desktop$>";
    printf("%s ", indicador);
}
```

b. Leer orden:

Para obtener una línea de órdenes, el shell realiza una lectura bloqueante de forma que el proceso que ejecuta el shell será bloqueado hasta que el usuario teclea una línea de órdenes en respuesta del indicador. Cuando el usuario ha proporcionado una orden (y terminado con el carácter nueva línea), se devuelve la línea de órdenes al shell. Para esto se apoya en la biblioteca stdio para leer la línea de comando y almacenarla en el buffer usando la función fgets y luego borrando el carácter vacío de la cadena almacenada en el buffer.

```c
void leerOrden(char *buffer)
{
    fgets(buffer, TAMANO_ORDEN, stdin);
    buffer[strlen(buffer)-1] = '\0';
}
```

c. Analizar Orden:

Esta función permite guarda el nombre del comando para luego determinar si el nombre de comando será valida, e ir construyendo la lista de parámetros, este me permitirá separar los comando ya sea por espacios, tabulaciones, líneas, puntos.

Para la utilización de esta función se creor un tipo de dato Struct comando para guardar el comando con sus parámetros respectivos.

```c
int analizarOrden(char *lOrdenes, struct comando_t *orden)
{
    int argc;
    int i, j;
    char **lcPtr;
    lcPtr = &lOrdenes;
    argc = 0;
    while((orden->argv[argc++] = strsep(lcPtr, ESPACIOBLANCO)) != NULL) ;
    orden->argv[argc--] = '\0';
    orden->argc = argc;
    return  1;
}
```

d. Buscar ruta:

El shell buscara si el comando (el nombre de archivo) está en los directorios primero buscara en el directorio actual, después en /bin y finalmente en /usr/bin. Si no existiera el archivo con el nombre especificado en la orden en ninguno de los directorios especificados, el shell responderá al usuario que es imposible encontrar la orden.

```c
char *buscarPath(char **argv, char **dir)
{
    int i;
    char *resultado;
    char pName[LONG_MAX_PATH];
    if(*argv[0] == '/')
    {
        resultado = (char *) malloc(strlen(argv[0])+1);
        strcpy(resultado, argv[0]);
        return resultado;
    }
    for(i = 0; i < MAX_PATHS; i++)
    {
        if(dir[i] == NULL) break;
        strcpy(pName, dir[i]);
        strcat(pName, "/");
        strcat(pName, argv[0]);

        if(access(pName, X_OK | F_OK) != -1)
        {
            resultado = (char *) malloc(strlen(pName)+1);
            strcpy(resultado, pName);
            return resultado;
        }
    }
    fprintf(stderr, "%s: comando no encontrado\n", argv[0]);
    return NULL;
}
```

e. Analizar ruta:

Esta función lee la variable PATH de este entorno, entonces construye un array, dirs[], con los directories en PATH.

```c
int analizarPath(char *dirs[])
{   int i;      char *pathEnvVar;
    register char *elPath, *viejaRuta;
    for(i=0; i<MAX_ARGS; i++)
       dirs[i] = NULL;
    pathEnvVar = (char *) getenv("PATH");
    elPath = (char *) malloc(strlen(pathEnvVar) + 1);
    strcpy(elPath, pathEnvVar);
    i = 0;
    viejaRuta = elPath;
    for(;; elPath++)
    {
       if((*elPath == ':') || (*elPath == '\0'))
        {
            dirs[i] = viejaRuta;
            i++;
            if(*elPath == '\0') break;
            *elPath = '\0';
            viejaRuta = elPath + 1;
        }
    }
}
```

Otra funciones complementarias que se implementó en el Shell, son la función del time y del historial de comandos.

a. Historial de comandos:

Las órdenes que se escriben en el terminal se van guardando en un archivo histórico, es lo que se llama historia de órdenes.

Ésta va almacenando las órdenes con sus respectivos parámetros que se le van dando a la shell para que ejecute. El sentido de mantener la historia es facilitar y hacer más rápida la repetición de alguna orden ya ejecutados y poder reutilizar sus argumentos.

En este apartado, se estudiará cómo conseguir que el Shell mantenga una historia de las órdenes dadas, y cómo volver a llamar a órdenes dadas con anterioridad, modificarlas y reutilizarlas.

b. Time:

El comando de tiempo (Time) permite ver las estadísticas estándar de tiempo acerca del programa de ejecución. Estas estadísticas consisten en (i) el tiempo real transcurrido entre invocación y terminación, (ii) el tiempo de CPU de usuario (la suma de los valores tms_utime y tms_cutime en una estructura tms devuelto por tiempos, y (iii) la tiempo de CPU del sistema (la suma de los valores tms_stime y tms_cstime en un struct tms tal como lo devuelve.

Para poder realizar y medir el tiempo que tarda en interpretar un comando, se utiliza dos librerías provistas en c que son:

- #include <sys/resource.h>
- #include <sys/time.h>

Al leer una orden por el usuario, se ejecutará una función que inicia el tiempo, mientras esta orden para, el proceso que buscar y analizar la ruta, analizar la orden, una vez terminada de ejecutar esta orden, se vuelve a llamar a esta función para así poder calcular e imprimir los tiempos (real, usuario y de sistema) siempre y cuando se ejecute el comando time precedido de la orden.

Ejemplo: time ls [7]

VI. COMANDOS BASICOS:

A continuación se seleccionaron los comandos más usados en todos los Sistemas UNIX. Estos tambien pueden ser consultados en la página del manual (o man). [5]

- Manipulación de Directorios

mkdir. Crea un nuevo directorio cuyo nombre es el pasado por el argumento. Si ya existe un directorio con ese nombre retorna un error.

Modo de uso:

mkdir [opción] directorio...
Las banderas más utilizadas son: -p, crea directorios padres como se indique en el argumento. Las
rutas pueden ser absolutas y relativas. Por ejemplo:
[user@host ~]> mkdir -p dir1/dir2/dir3

rmdir. Elimina directorios, los directorios deben estar vacios.

Modo de Uso:

rmdir directorio

cd. Cambia el directorio actual al indicado por el argumento, si no se coloca argumento, el usuario será posicionado en el directorio por defecto de inicio de sesión, usualmente la carpeta de inicio del usuario..

Modo de uso:

cd [directorio]

- Manipulación de Archivos

ls. Lista el contenido del directorio pasado como argumento. Sin argumentos lista el contenido del directorio actual.

Modo de Uso:

ls [opción]... [archivo/dir]...
Las banderas más usadas son:
-l, lista el contenido usando un formato detallado, donde se pueden ver la permisología, el dueño, el tamaño, fecha de última modificación de los archivos, entre otras cosas.
-a, lista los archivos ocultos, es decir, cuyo prefijo es punto. Ejemplo: .archivo
-h, Solo es útil con la bandera -l, e imprime el tamano de los archivos en unidades conocidas como MB o KB en lugar de cantidad
bloques.
-R, Lista todos los directorios recursivamente.

cp. Copia archivos y directorios. Por defecto no copia directorios.

Modo de Uso:

cp [opción]... fuente destino
cp [opción]... fuente... directorio

Las banderas más comunes son:

-r -R, Permite la copia recursiva, especial para directorios
-s, En lugar de copiar realiza enlaces simbólicos.
-u, Copia solo cuando el archivo fuente es más nuevo que el destino o si no existe.

rm. Elimina archivos y directorios. Por defecto no borra directorios.

Modo de Uso:

rm [opción]... archivo...

Las banderas más usadas son:

-r -R, Permite la eliminación de directorios.
-f, Ignora archivos no existentes.

-i, Pregunta antes de hacer cualquier operación.
mv. Mueve (o renombra) archivos o directorios.

Modo de Uso:

mv [opción]... fuente destino
mv [opción]... fuente... directorio

Las banderas más comunes son:
-i, Confirma con el usuario antes de sobrescribir.
-u, Mueve solo cuando el archivo fuente es más nuevo que el destino o si no existe.

ln. Reliza un link entre archivos. Si un nombre de link no es especificado crea uno con el mismo nombre del Target. Por defecto crea un enlace duro.

Modo de Uso:
ln [opción]... target [nombre_link]
ln [opción]... target... Directorio

REFERENCIAS

[1] Alberto Luna y Pablo Sanz Mercado, Programacion de Shell Scripts. Editorial UA, 2012.

[2] Andrew S. Tanenbaum y Albert S. Woodhull, Operating Systems: Design and Implementation, 3ra. ed. New Jersey: Pearson Prentice Hall, 2006.

[3] Daniel Pecos Martínez. Blogger. Link http://danielpecos.com/documents/linux/

[4] Kay A. Robbins y Steven Robbins UNIX programación práctica : [guía para la concurrencia, la comunicación, los multihilos y manejo de señales], Prentice-Hall Hispanoamericana, cop. 1997 Edició 1ª editorial.

[5] Manual de comandos, man (Unix)

[6] Comandos Shell y Programacion en la Shell del Bash, 1995. Link www.uco.es/~in1lurom/materialDocente/apuntesSO.pdf

[7] Michael Beck, Harald Böhme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus y Dirk Verworner (1997). Linux Kernel Internals (Second Edition). Editorial: Addison-Wesley