Sistemas Operativos 1

Desarrollo de Shell – “DISH”

Introducción

Esta es la documentación del shell “DISH”, que se divide por los archivos que componen el código fuente. En cada sección estarán descritas las funciones que contiene cada archivo, junto con las estructuras de datos usadas cuando aplique. El lenguaje utilizado es el lenguaje C, y se utilizan múltiples funciones de librería estándar para el SO Linux.

El nombre del shell proviene de “Giovanni [D]ueck’s [I]nteractive [Sh]ell”, y sirve para darle al proyecto algo de identidad, y para no llamarlo por el termino general “Shell” constantemente.

Librerías utilizadas

Diversas librerías estándar se utilizan en el programa. Estas son:

ctype.h

dirent.h

errno.h

grp.h

pwd.h

shadow.h

stdio.h

stdlib.h

string.h

sys/stat.h

sys/types.h

sys/wait.h

termios.h

time.h

unistd.h

Una librería no estándar también se utilizó: libgcrypt, o lcrypt. El header es crypt.h

Instalación

Para instalar DISH existen instrucciones en el archivo README.md. Consiste en la creación de directorios de instalación, que son /programs, /programs/bin, y /programs/dish-help, la compilación, mover archivos a los directorios de instalación, y agregar /programs/bin al PATH.

Listado de archivos

Los archivos que componen el proyecto son los siguientes:

[Headers](#_Headers):

* [defines.h](#_defines.h)
* [dish.h](#_dish.h)
* [builtins.h](#_builtins.h)
* [err.h](#_err.h)

[C](#_C):

* [main.c](#_main.c)
* [dish.c](#_dish.c)
* [builtins.c](#_builtins.c)
* [err.c](#_err.c)

[Shell script](#_Shell_scripts):

* [compile.sh](#_compile.sh)
* [gdb.sh](#_gdb.sh)
* [check.sh](#_check.sh)
* [install.sh](#_install.sh)

[Archivos de texto](#_Archivos_de_texto):

* [TODO.txt](#_TODO.txt)
* [README.md](#_README.md)
* [help/\*.txt](#_help/*.txt)

# Headers

## defines.h

En este header se definen varias constantes de preprocesador. Estas son:

SUCCESS: 0, retorno al finalizar el programa correctamente.

FAILURE: 1, retorno al finalizar el programa con algún error.

TRUE: 1

FALSE: 0

DIR\_BUFSIZE: 1024, tamaño de buffer para directorios.

LINE\_BUFSIZE: 1024, tamaño de buffer para la línea de comando.

ARGS\_BUFSIZE: 64, tamaño de buffer para cantidad de argumentos en una línea de comando.

MSG\_LENGTH: 128, tamaño de buffer para mensajes de error.

HOST\_NAME\_MAX: 64, tamaño de buffer para el hostname.

TOK\_DELIM: " \t\n\r\a", caracteres que delimitan argumentos de un comando en una línea.

## dish.h

Contiene declaraciones de las funciones presentes en dish.c, funciones que conforman el esqueleto del programa (ver sección [dish.c](#_dish.c)).

Define además el tipo de dato *struct uinfo*, un registro que se usa para almacenar los datos adicionales de usuarios:

* int hh\_i: hora de inicio de sesión
* int mm\_i: minuto de inicio de sesión
* int hh\_f: hora de cierre de sesión
* int mm\_f: minuto de cierre de sesión
* int num\_lugares: cantidad de lugares posibles de inicio de sesión

Esta estructura se almacena en un archivo binario, seguida de las cadenas con el nombre de los lugares de conexión, para ser usadas al registrar inicios y cierres de sesión y comparar hora y lugar de los mismos con la información guardada.

## builtins.h

Contiene declaraciones de funciones que ejecutan los comandos integrados de DISH (ver sección builtins.c).

Declara también dos vectores, uno de cadenas y uno de funciones, que facilita el llamado a las mismas y simplifica la expansión del número de comandos.

## err.h

Contiene declaraciones de funciones que registran errores encontrados en la ejecución de comandos (ver sección err.c).

# C

## main.c

Archivo principal, contiene la función main y funciones de registro en logs, además del loop central de DISH.

Funciones:

void check\_user(): busca y almacena el nombre de usuario y el nombre del host en variables globales. Par el nombre de usuario usa getenv, que no es la opción más segura, pero getlogin no funciona en LFS. Para el hostname usa gethostname.

void check\_logs(): revisa si los archivos de log existen, estos son el historial de comandos, log de errores, y el registro de inicios y cierres de sesión. Los crea si es necesario.

void log\_add(char \*line): agrega un comando al historial con la hora actual.

Parámetros:

line: una cadena.

void dish\_log(char mode): agrega una entrada al registro de sesión, según el parámetro mode. Escribe la hora y lugar de inicio de sesión, y la hora de cierre, comparándolas con la información adicional disponible del usuario (ver sección [dish.h](#_dish.h)).

Parámetros:

mode: un carácter, puede ser ‘i’, que denota inicio de sesión, u ‘o’, que denota cierre de sesión.

void dish\_loop(): loop central del programa. Se encarga de imprimir el nombre de usuario y el directorio actual, y de leer los comandos del usuario. Luego divide la línea en argumentos y los interpreta. Según el resultado continúa leyendo comandos o se interrumpe para terminar el programa: cualquier retorno de comandos diferente de 0 continúa la ejecución, el retorno igual a 0 termina la ejecución. Solo el comando “salir” retorna 0.

int main(): el programa se inicializa con llamadas a las funciones check\_user(), check\_logs(), y dish\_log(‘i’). Luego corre dish\_loop(), y para terminar llama dish\_log(‘o’).

Retorno:

SUCCESS: programa terminado con éxito.

## dish.c

Contiene funciones esenciales para el funcionamiento de DISH, además de algunas funciones auxiliares.

Estructuras de datos:

static char \*builtin\_str[]: array de cadenas, que son los nombres de los comandos. Cada comando tiene un índice y un nombre único. El uso de un array facilita la comparación de los comandos dados por el usuario con los comandos existentes de forma iterativa.

static int (\*builtin\_func[]) (char\*\*): array de referencias de funciones. Cada comando tiene un índice y una función única, el índice el mismo que en builtin\_str. El uso de un array para realizar la llamada a las funciones correspondientes a los comandos permite hacer la llamada de forma iterativa, y demostró ser muy práctico a la hora de implementar un comando nuevo ya que se evita usar numerosos if..else o un switch enorme (para ver la descripción de las funciones ver sección builtins.c).

Parámetros:

char\*\*: para que todas las funciones puedan ser llamadas desde el mismo array, deben ser casi iguales, por lo que todas toman un parámetro que es un array de cadenas.

Retorno:

int: para que todas las funciones puedan ser llamadas desde el mismo array, deben ser casi iguales, por lo que todas retornan un entero.

Funciones:

char \*dish\_read\_line(): lee una línea de stdin hasta encontrar EOF o ‘\n’, y la retorna.

Retorno:

char\*: línea leída de stdin.

char \*\*dish\_split\_line(char \*line): divide la cadena dada en tokens (que son funcionalmente los argumentos de un comando), y los retorna en un array de cadenas. Usa la constante TOK\_DELIM.

Parámetros:

char \*line: una cadena. Típicamente leída con dish\_read\_line, pero no necesariamente.

Retorno:

char\*\*: un array de cadenas. La última cadena es un puntero NULL, esto hace recorrer el array sin saber su longitud posible.

int dish\_execute(char \*\*args): ejecuta el comando ingresado por el usuario. Si el comando es una línea vacía, simplemente retorna 1, si el comando es válido retorna el resultado del comando, y si no lo es informa al usuario.

Parámetros:

char \*\*args: argumentos del comando, creados con dish\_split\_line.

Retorno:

int: 1 cuando args es un array vacío, 1 cuando el comando no es válido, y el resultado de la ejecución de un comando si el comando es válido.

int dish\_command\_not\_found(char \*\*args): informa al usuario que el comando ingresado no es válido, junto con un mensaje de ayuda.

Parámetros:

char \*\*args: argumentos del comando.

Retorno:

int: 1, siempre.

void dish\_sort(char \*\*array, int size): función auxiliar. Implementación de Selection sort para ordenar un array de cadenas. Usada por el comando “listar.”

Parámetros:

char \*\*array: array a ordenar.

int size: tamaño de array.

double dish\_pow(double x, double y): funcion auxiliar. Implementación recursiva de la operación matemática potencia. Retorna x elevado a la y potencia, aunque y debe ser un entero positivo para que funcione. Usada por el comando “propietario.”

Parámetros:

double x, y: dos números.

Retorno:

double: x elevado a la potencia y.

## builtins.c

Contiene las funciones que ejecutan los comandos de DISH, además de algunos pocos comandos de ayuda.

Por lo general todas las funciones que ejecutan comandos interpretan primero las opciones dadas como argumentos, y luego ejecutan el comando según están opciones. La opciones pueden ser diversas, como imprimir el manual del comando o mostrar más detalles en el resultado, y siempre tienen el formato – [o] o – – [opción], es decir un guion seguido de una letra o dos guiones seguidos de una palabra. Todos los comandos toman las opciones – h y – – ayuda, que imprime el manual de uso del comando.

Todos los comandos (o las funciones que los representan, este es el sentido en el que uso la palabra “comando” en esta sección) toman char \*\*args como único parámetro, por lo que no se especifica en la descripción a continuación. Por lo general retornan 1 en caso de éxito, y 2 en caso de error, por lo que solo se detallarán retornos diferentes a estos. Estas excepciones solo se aplican a funciones que ejecutan comandos. En el caso de que solo tomen las opciones de ayuda que toman todos los comandos, la sección de opciones se omitirá. De todas formas, la sintaxis para obtener el manual de un comando es “comando –h” o “comando --ayuda”.

Obs.: algunos comandos pedidos en los requerimientos del trabajo no se incluyeron por razones de tiempo o frustración (“contraseña” en particular). Pensé que sería bueno decirlo antes de listar los comandos que si están desarrollados.

Estructuras de datos:

static char \*builtin\_str[]: (ver sección [dish.c](#_dish.c))

static int (\*builtin\_func[]) (char\*\*): (ver sección [dish.c](#_dish.c))

Obs.: se repite la declaración porque de otra forma el compilador daba un error. Esta fue la solución más simple.

Funciones:

int dish\_print\_help(char \*command\_name): imprime el manual del comando command\_name, que se encuentra en el archivo help/command\_name.txt. No es un comando, sino una función adicional usada por los comandos.

Parámetros:

char \*command\_name: nombre del comando, típicamente el índice correspondiente al comando en builtin\_str.

Retorno:

int: 1 en caso de éxito, 2 en caso de error. Errores son registrados en el registro de errores e impresos en pantalla.

Comandos:

int dish\_cd(char \*\*args): comando “ir”, cambia de directorio.

Uso: ir [DEST]

DEST es el directorio de destino.

int dish\_help(char \*\*args): comando “ayuda”, imprime la lista de comandos disponibles, no toma argumentos.

Uso: ayuda

int dish\_exit(char \*\*args): comando “salir”, termina el programa.

Uso: salir

Retorno: 0, excepto si se da la opción de ayuda.

int dish\_sys(char \*\*args): comando “sys”, permite ejecutar comandos no propios de DISH. Los ejecuta mediante un subproceso.

Uso: sys [COMANDO]

COMANDO es el comando de bash junto con sus argumentos y opciones.

int dish\_history(char \*\*args): comando “historial”, imprime el historial de comandos.

Uso: historial

historial [OPCION]

Opciones:

-v: imprime el historial junto con la hora de ejecución de cada comando.

-r: borra el historial.

Retorno: char ‘r’ cuando se da la opción –r. La función dish\_loop se encarga de reestablecer el historial.

int dish\_useradd(char \*\*args): comando “usuario”, permite agregar un usuario.

Uso: usuario [NOMBRE]

Usuario [OPCION] [NOMBRE] [NOMBRE\_GRUPO]

NOMBRE es el nombre de usuario, NOMBRE\_GRUPO es el nombre de grupo

Opciones:

-g: permite especificar el nombre del grupo. Si se omite el usuario es parte de su propio grupo.

int dish\_passwd(char \*\*args): comando “contrasena”, permite cambiar la contraseña de un usuario. En realidad esta función llama al comando “sys passwd”.

Uso: contraseña <USUARIO>

USUARIO es el usuario a cambiar de contraseña, si se omite se cambia la contraseña propia.

int dish\_cp(char \*\*args): comando “copiar”, copia un archivo.

Uso: copiar [ORIGEN] <DESTINO>

ORIGEN es el nombre del archivo de origen, DESTINO es el nombre de la copia. Si se omite, se autogenera un nombre.

int dish\_mv(char \*\*args): comando “mover”, mueve un archive a otro directorio. Usa los comandos “copiar”, “renombrar” y “remover” por debajo, por lo que su comportamiento afecta a este comando.

Uso: mover [ARCHIVO] [DIRECTORIO]

mover -r [ORIGEN] [DESTINO] [DIRECTORIO]

ARCHIVO es el archivo a mover, DIRECTORIO es el directorio de destino. En caso de dar la opción –r, ORIGEN es el archivo a mover, DESTINO es el nombre nuevo, y DIRECTORIO es el directorio de destino.

Opciones:

-r: renombra el archivo movido.

int dish\_rn(char \*\*args): comando “renombrar”, renombra un archivo.

Uso: renombrar [ARCHIVO] [NOMBRE]

ARCHIVO es el nombre actual del archivo, NOMBRE es el nombre nuevo.

int dish\_mkdir(char \*\*args): comando “creardir”, crea un directorio.

Uso: creardir [NOMBRE]

NOMBRE es el nombre del directorio.

int dish\_rm(char \*\*args): comando “remover”, elimina un archivo. Pide confirmación antes de eliminar.

Uso: remover <OPCION> [NOMBRE]

NOMBRE es el nombre del archivo a eliminar.

Opciones:

-f: elimina sin pedir confirmación.

int dish\_rmdir(char \*\*args): comando “removerdir”, elimina un directorio si esta vacio.

Uso: removerdir [NOMBRE]

NOMBRE es el nombre del directorio a eliminar.

int dish\_ls(char \*\*args): comando “listar”, lista los archivos en el directorio actual.

Uso: listar <OPCION>

Opciones:

-a: muestra archivos ocultos

int dish\_chmod(char \*\*args): comando “permisos”, cambia los permisos de un archivo.

Uso: permisos [ARCHIVO] [777]

ARCHIVO es el archivo a modificar los permisos, 777 son los permisos en formato octal.

int dish\_chown(char \*\*args): comando “propietario”, cambia el propietario de un archivo.

Uso: propietario [ARCHIVO] [USUARIO] <GRUPO>

ARCHIVO es el nombre del archivo, USUARIO es el nombre de usuario del nuevo propietario, GRUPO es el nuevo grupo propietario, si se omite, USUARIO se toma como nombre del grupo.

int dish\_userinfo(char \*\*args): comando “uinfo”, toma información adicional sobre un usuario. Usa el struct uinfo de [dish.h](#_dish.h).

Uso: uinfo <OPCION> <USUARIO>

USUARIO es el nombre del usuario cuya información se da. Si se omite se toma al usuario actual.

Opciones:

-s: muestra la información actual de USUARIO sin modificarla.

## err.c

Contiene funciones que se encargan de imprimir mensajes de error en pantalla y registrarlos en el registro de errores.

Funciones:

char \*get\_err\_msg(char \*str): revisa errno y retorna una cadena con un mensaje apropiado segun el valor de errno.

Parámetros:

char \*str: cadena que almacena el mensaje.

Retorno:

char\*: cadena que almacena el mensaje, NULL en caso de fallo.

void err\_put\_date(FILE \*file): escribe la hora actual en el registro de errores.

Parámetros:

FILE \*file: descriptor del archivo de registro de errores.

char \*compose\_line(char \*dest): procedimiento inverso a dish\_split\_line, toma char \*\*args, que está disponible de forma global en el archivo err.c, y compone la linea de comando de forma contigua.

Parámetros:

char \*dest: cadena que almacena la línea.

Retorno:

char\*: cadena que almacena la línea.

void err\_log\_add\_msg(char \*msg): escribe el mensaje de error dado en el registro de errores.

Parámetros:

char \*msg: cadena del mensaje a escribir.

void err\_print(char \*msg): imprime un mensaje de error de acuerdo a errno, msg típicamente contiene el nombre del comando que generó el error.

Parámetros:

char \*msg: nombre del comando que generó el error.

# Shell scripts

## compile.sh

Contiene el comando de que compila el código fuente. El ejecutable se nombra “dish”.

## gdb.sh

Contiene el comando contenido en compile.sh, pero con la opción –g, que permite debuggear o depurar el código, en mi caso, con el depurador GDB.

## check.sh

Revisa que los directorios necesarios para la ejecución de DISH sea correcta, además de revisar si libgcrypt está instalada.

## install.sh

Compila y mueve al programa a su directorio de instalación. No es posible modificar el lugar en el que se instala porque implicaría modificar código fuente también.

# Archivos de texto

## TODO.txt

De uso en el desarrollo, no tiene implicaciones objetivas sobre el proyecto. Contiene anotaciones que sirvieron de guía durante el desarrollo del shell.

## README.md

Archivo de GitHub, de significancia similar a TODO.txt, aunque detalla los últimos cambios realizados e instrucciones de instalación. Contiene el número de versión, que no tiene mucha importancia de por sí, pero me sirvió para detallar el progreso del trabajo. Varios “commit”s de git llevan el mensaje “Version x.y”, en donde los cambios recientes son considerados completos y detallados en este archivo. Números ‘x’ revelan si es una versión no terminada (0) o terminada (1), y números ‘y’ incrementan con versiones x consecutivas.

## help/\*.txt

Archivos de texto que contienen los manuales de uso de cada comando. El archivo “help/template.txt” describe el formato de los manuales en mayor detalle. Los contenidos de estos archivos es lo que imprime la función de [builtins.c](#_builtins.c) dish\_print\_help.