2017

Proyecto 2: Simple Tar(Star)

PRINCIPIO DE SISTEMAS OPERATIVOS

Estudiantes:
Karen Lépiz Chacón – 2013005341
André Solís Barrantes - 2013389035

Contenido

Introducción.

El proyecto consiste en implementar las funciones de empaquetar y desempaquetar con el comando tar, que en este caso se llamará star(simple tar) y va a ser desarrollado en lenguaje C. El objetivo principal es dominar el manejo de archivos y bloques de espacio que es en donde se almacenarán los archivos dentro del paquete.

El programa tar se usa para almacenar archivos o directorios dentro de un solo archivo o paquete y se ejecuta por medio de la línea de comandos de una consola de texto o desde la terminal de UNIX. Y tiene el siguiente formato:

```
tar <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>
```

- ArchivoSalida: el nombre del archivo “paquete” con la extensión .tar
- ArchivoN : nombre de archivo que serán empaquetados.

Descripción del problema.

El programa va a implementar las funciones más comunes de tar:

- -c, –create : crea un nuevo archivo
- -x, –extract : extraer de un archivo
- -t, –list: listar los contenidos de un archivo
- –delete: borrar desde un archivo
- -u, –update: actualiza el contenido del archivo
- -v, –verbose: ver un reporte de las acciones a medida que se van realizando
- -f, –file: empaquetar contenidos de archivo, si no está presente asume la entrada estándar.
- -r, –append: agrega contenido a un archivo

Para desarrollar el programa se tomará en cuenta los siguientes aspectos:

• El archivo “tar” deberá estar estructurado mediante bloques de espacio continuo que serán asignados en forma individual a los archivos que se deben almacenar.

• Al crear un archivo empacado éste se crea del tamaño necesario para almacenar los archivos agregados. Cuando se borra algún contenido, el archivo empacado no cambia de tamaño, sino que se lleva registro de los espacios (huecos) liberados. Si posteriormente se agrega nuevo contenido entonces se reutiliza el espacio libre. Si aun así el nuevo contenido no cabe, se hace crecer el archivo empacado.

• En el archivo tar se almacenará una tabla que indique la dirección en donde empieza cada archivo y su tamaño. Internamente también se debe llevar registro del contenido de los directorios. Para ello se deberá utilizar espacio adicional bien el archivo empacado. El directorio se almacenará como cualquier archivo, pero internamente contará con secuencia de nombres de archivo y sus posiciones en el archivo “tar”. Puede asumir que los nombres de archivo no son mayores a los 256 caracteres.

• Para asignar el espacio a cada archivo se debe buscar un hueco en el archivo tar en el cual se pueda introducir. En este caso se utilizará la estrategia de “el primer ajuste” para realizar esta asignación. Para ello se debe llevar un control interno de los huecos sin utilizar en el archivo y se debe realizar la fusión de dichos huecos cuando sea necesario.

• Tome en cuenta que un archivo que se agrega puede ya existir en el archivo empacado. Es decir, lo que se desea hacer es actualizar su contenido. Para 2 esto existe la opción update (-u) que sobrescribirá el contenido de un archivo siempre y cuando su fecha de modificación sea posterior a la del archivo empacado.

• Debe programar una solución eficiente, es decir, debe minimizar la cantidad de espacio utilizado tanto por los archivos como por las estructuras internas.

• Este programa no utilizará los derechos de acceso, que normalmente almacenaría un archivo empacado tar en ambiente UNIX.

• No utilice ningún archivo adicional (ni siquiera archivos temporales o intermedios) para implantar este programa, con excepción del archivo .tar

Para realizar las pruebas de correctitud se incluirá un nueva opción -d, –dump que imprimirá en pantalla la tabla de contenido del archivo, así como la lista de bloques libres. La idea es poder seguir el rastro de las diferentes operaciones conforme se van ejecutando en el archivo. Luego se deberán ejecutar diversos conjuntos de pruebas a su programa con diferentes combinaciones de operaciones de: crear, eliminar, actualizar y desfragmentar contenido. Estas pruebas deben ser realizadas tanto con archivos individuales como con directorios completos. Utilizado las salidas de la opción dump se deberán documentar los resultados de dichas pruebas y se deberá mostrar el correcto funcionamiento de las diferentes operaciones.

Definición de estructuras de datos.

Se definieron dos estructuras de datos para poder manejar información importante de la tabla de bloques usadas.

- Tabla_bloque: información importante o básica de los bloques.
- Bloque: se usa para los segmentos de la tabla.

```
typedef struct {
   int id; //id del bloque
   int estado; //libre=0 y ocupado=1
   int dirInicial; //dir de inicio del bloque
   int tam; //tam del bloque
   char ruta[MAXNOMBRE]; //ruta del archivo
} tabla_bloque;


typedef struct {
  char datos[500*1024];
} Bloque;
```

Además, se usará una estructura de datos que implementa el sistema de llamada: stat. En stat se guarda información de los archivos que se encuentran en una dirección determinada. Para este proyecto solo se usará el dato de st_mode para

poder identificar el tipo de archivo.

```c
struct stat sdata; // llamar el struct de stat

/*
  para este caso solo se usara el st_mode para saber el tipo de archivo
*/
  //stat(path,buf)
if (stat(fname, &sdata) == -1)
  {
    return DT_UNKNOWN;
  }

switch (sdata.st_mode & S_IFMT) // S_IFMT : considerar los bits que indican el tipo del archivo
  {
  case S_IFBLK:  return DT_BLK; // block device
  case S_IFCHR:  return DT_CHR; // character device
  case S_IFDIR:  return DT_DIR; // directorio
  case S_IFIFO:  return DT_FIFO; // FIFO
  case S_IFLNK:  return DT_LNK; // symbolic link
  case S_IFREG:  return DT_REG; // archivo regular
  case S_IFSOCK: return DT_SOCK; // socket
  default:       return DT_UNKNOWN; // en caso que el tipo del archivo sea desconocido
  }
```

Por último, se usa la estructura de datos dirent que se usa para obtener información del archivo que se está extrayendo.

```c
struct dirent *ent;
```

Descripción detallada y explicación de los componentes principales del programa.

Mecanismo de acceso a archivos y directorios.

Para el manejo de archivos y directorios se usan funciones para el manejo de ficheros/directorios que están definidas en la librería de C stdio.h.

- fread: leer un bloque.
- fwrite: escribir hacia un fichero uno o más registros de la misma longitud almacenada a partir de una dirección de memoria dada.
- fseek: da la posición del archivo del stream en el desplazamiento dado.
- fopen: función para abrir archivos.
- fclose: función para cerrar archivos.
- opendir: función para abrir directorios.
- readdir: función para leer el contenido del directorio.
- closedir: función para cerrar directorios.

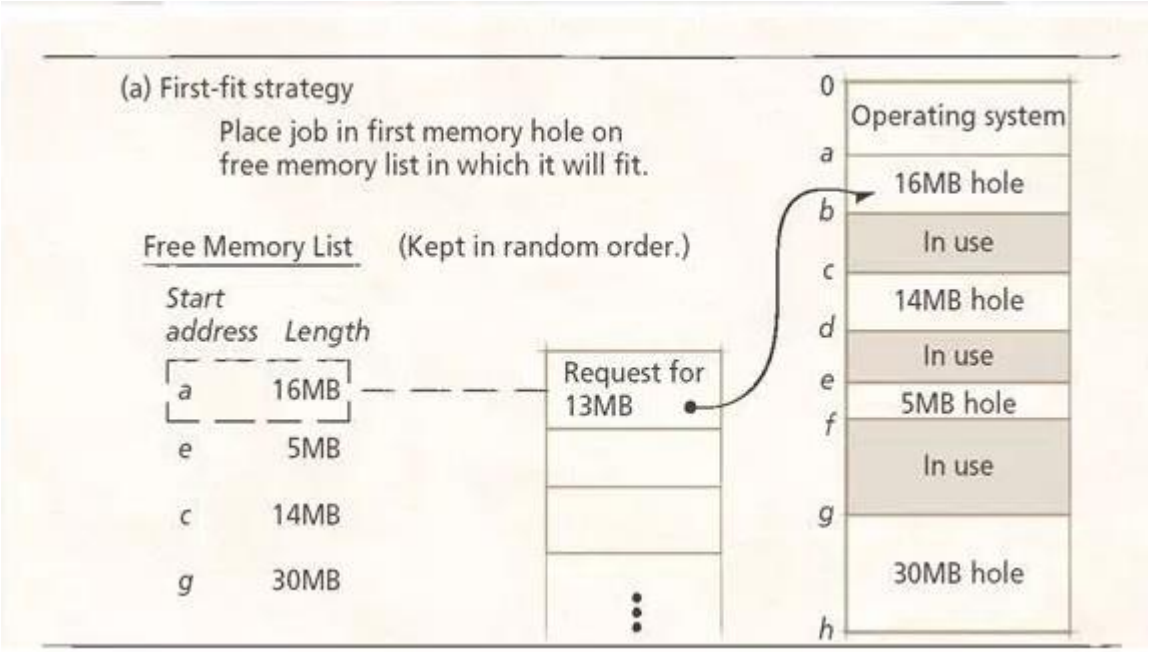
Estructura de la tabla de asignación de espacio.

La tabla de asignación tendrá los siguientes datos:

- Identificador del bloque o segmento.
- Un estado en donde se indicará cuando está el bloque libre o está ocupado.
- Dirección de inicio del bloque.
- Tamaño el bloque
- Ruta del archivo.

Estrategia de administración de bloques libres.

Para la administración de bloques libre se utilizará el algoritmo de primer ajuste, que dice que se asignará el archivo a un bloque si este se encuentra libre. A continuación, se mostrará un ejemplo de cómo funciona este algoritmo:



Análisis de resultados de pruebas.

- Crear.
  - Archivos:
    - `./star -cvf  prueba.star A`

prueba.star

```
Lista de Contenido
Nombre: A
```

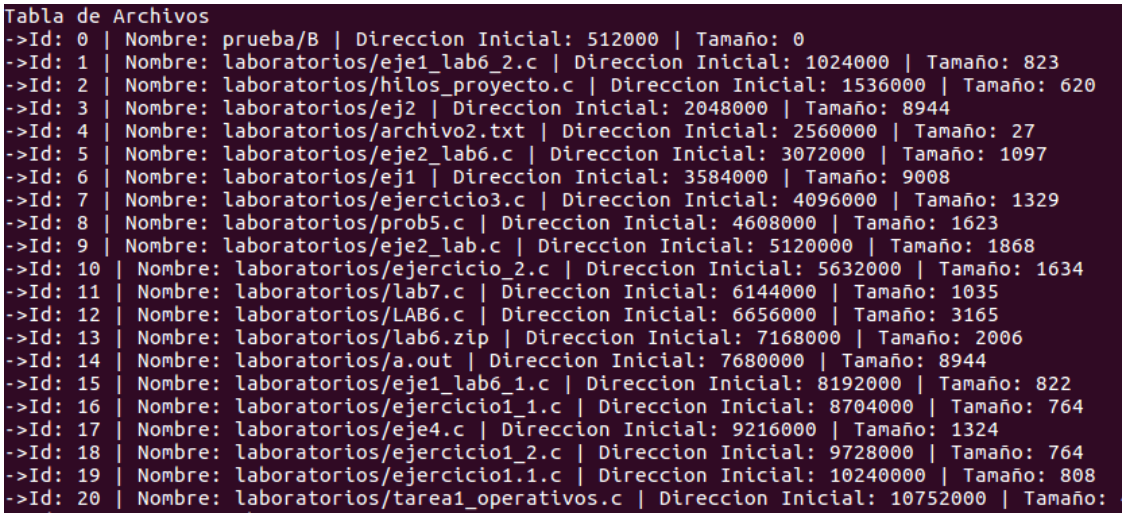
  - Directorios:
    - `./star -cvf  prueba_.star laboratorios`

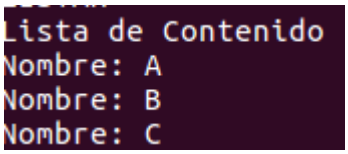
prueba_.star

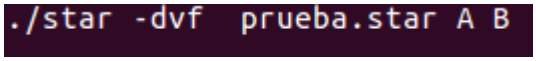
```
Lista de Contenido
Nombre: laboratorios/eje1_lab6_2.c
Nombre: laboratorios/hilos_proyecto.c
Nombre: laboratorios/ej2
Nombre: laboratorios/archivo2.txt
Nombre: laboratorios/eje2_lab6.c
Nombre: laboratorios/ej1
Nombre: laboratorios/ejercicio3.c
Nombre: laboratorios/prob5.c
Nombre: laboratorios/eje2_lab.c
Nombre: laboratorios/ejercicio_2.c
Nombre: laboratorios/lab7.c
Nombre: laboratorios/LAB6.c
Nombre: laboratorios/lab6.zip
Nombre: laboratorios/a.out
Nombre: laboratorios/eje1_lab6_1.c
Nombre: laboratorios/ejercicio1_1.c
Nombre: laboratorios/eje4.c
Nombre: laboratorios/ejercicio1_2.c
Nombre: laboratorios/ejercicio1.1.c
Nombre: laboratorios/tarea1_operativos.c
```

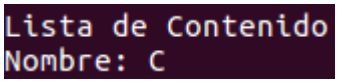
- Actualizar.
  - Archivos:
    - ```
      ./star -cvf  prueba.star A B
      ```
    - ```
      Tabla de Archivos
      ->Id: 0 | Nombre: A | Direccion Inicial: 512000 | Tamaño: 0
      ->Id: 1 | Nombre: B | Direccion Inicial: 1024000 | Tamaño: 0
      ```
    - ```
      ./star -uvf  prueba.star C
      ```
    - ```
      Lista de Contenido
      Nombre: C
      ```
  - Directorios:
    - ```
      Lista de Contenido
      Nombre: laboratorios/eje1_lab6_2.c
      Nombre: laboratorios/hilos_proyecto.c
      Nombre: laboratorios/ej2
      Nombre: laboratorios/archivo2.txt
      Nombre: laboratorios/eje2_lab6.c
      Nombre: laboratorios/ej1
      Nombre: laboratorios/ejercicio3.c
      Nombre: laboratorios/prob5.c
      Nombre: laboratorios/eje2_lab.c
      Nombre: laboratorios/ejercicio_2.c
      Nombre: laboratorios/lab7.c
      Nombre: laboratorios/LAB6.c
      Nombre: laboratorios/lab6.zip
      Nombre: laboratorios/a.out
      Nombre: laboratorios/eje1_lab6_1.c
      Nombre: laboratorios/ejercicio1_1.c
      Nombre: laboratorios/eje4.c
      Nombre: laboratorios/ejercicio1_2.c
      Nombre: laboratorios/ejercicio1.1.c
      Nombre: laboratorios/tarea1_operativos.c
      ```
    - ```
      ./star -uvf  prueba_.star prueba
      ```
    - ```
      Lista de Contenido
      Nombre: prueba/B
      ```
      ```

- Agregar.
  - Archivos:
    - ```
      ./star -avf  prueba.star star_.c
      ```
    - ```
      Lista de Contenido
      Nombre: C
      Nombre: star_.c
      ```
  - Directorios:
    - ```
      ./star -avf  prueba_.star laboratorios
      ```
    - ```
      Lista de Contenido
      Nombre: prueba/B
      Nombre: laboratorios/eje1_lab6_2.c
      Nombre: laboratorios/hilos_proyecto.c
      Nombre: laboratorios/ej2
      Nombre: laboratorios/archivo2.txt
      Nombre: laboratorios/eje2_lab6.c
      Nombre: laboratorios/ej1
      Nombre: laboratorios/ejercicio3.c
      Nombre: laboratorios/prob5.c
      Nombre: laboratorios/eje2_lab.c
      Nombre: laboratorios/ejercicio_2.c
      Nombre: laboratorios/lab7.c
      Nombre: laboratorios/LAB6.c
      Nombre: laboratorios/lab6.zip
      Nombre: laboratorios/a.out
      Nombre: laboratorios/eje1_lab6_1.c
      Nombre: laboratorios/ejercicio1_1.c
      Nombre: laboratorios/eje4.c
      Nombre: laboratorios/ejercicio1_2.c
      Nombre: laboratorios/ejercicio1.1.c
      Nombre: laboratorios/tarea1_operativos.c
      ```

- Dump.
  - Para directorios y archivos:
    - 

- Eliminación.
  - Archivos y directorios.
    - 

    - 

    - 

- Validaciones.
    - Validar que el formato de entrada sea correcto.

```
karen@ubuntu:~/Documents$ ./star -d0
opcion: -d0
comando: 0
cantidad de tipos de archivos 0 y cantidad de directorios 0
El numero de parametros son incorrectos
star <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>
```

    - En caso de ingresar una opción de comando incorrecta.

```
karen@ubuntu:~/Documents$ ./star -a prueba.star A
opcion: -a
comando: 0
Es un archivo
cantidad de tipos de archivos 1 y cantidad de directorios 0
archivo
es un archivo
El comando que ingreso es incorrecto
Los comandos disponibles son:
               Crear un archivo: -cvf
               Eliminar un archivo: -dvf
               Extraer el contenido un archivo: -xvf
               Actualizar el contenido un archivo: -uvf
               Agregrar contenido de un archivo: -avf
               Listar los contenidos de  un archivo: -l
               Pruebas de correctitud: -d
```

Conclusiones sobre rendimiento.

El uso de compresores puede traer beneficios al espacio del disco que tiene la computadora, ya que se tendría una cantidad n de archivos en uno solo, ahorrando espacio. Y en caso de que se quieran usar, basta con extraerlos del archivo.tar. Además, el uso de estas funcionalidades por medio de línea de comandos facilita su ejecución, pues se ahorra el tiempo de descargar e instalación de un compresor, ya que tar viene incluido en Unix.

El tamaño que se le asigne a cada bloque también es importante, pues se debe dar el adecuado para que no se desperdicie espacio, por lo que se calcula el tamaño del archivo que se quiere comprimir para dar el espacio beneficioso, además del uso de reutilización de espacios que quedan libres entre bloques que están en uso.