Escuela de ingeniería en computación

Principios de Sistemas Operativos

Proyecto 2

Realizado por: Ricardo Andrey Sánchez Delgado

Carné: 2014010852

22 de mayo del 2017

Contenido

Introducción 3
Descripción del problema 5
Definición de estructuras de datos 7
Principales del programa 7
Mecanismo de acceso 8
Estructura de la tabla de asignación de espacio 9
Estrategia de administración de bloques libres 10
Procedimiento de desfragmentación del archivo 11
Análisis de resultados de pruebas 12
Creación de un archivo 13
Append 15
Actualizar 18
Listar 20
Eliminar 21
Extraer 23
Desfragmentar 25
Conclusiones sobre rendimiento 26

Introducción

Hoy en día debido a que existe mucha información hace falta poder dividir los archivos por categorías, un claro ejemplo con lo que se puede hacer esto es agruparlo en una carpeta, pero y ¿si se requiere pasar a través de la red el contenido de esta?

Puede complicarse un poco debido a que se tendrían que hacer múltiples conexiones, o simplemente establecer métodos para pasar la información y que del otro lado donde se reciben los archivos estos estén contenidos en una estructura de datos igual desde la que se mandó, aquí es donde surgió la idea de los famosos archivos tar que son usados principalmente en Linux, con este modelo de datos (archivo tar) entonces se pueden simplificar los errores que se indicaron anteriormente. Para propósitos de este proyecto se realizará una implementación simple del star, la cual hará uso de memoria contigua, utilizando la técnica de primer ajuste.

De esta manera entonces se provee una interfaz con la cual se pueden comprimir y descomprimir archivos, otorgando modularidad

Descripción del problema

El objetivo de este proyecto consiste en programar un empacador de archivos. Este es el tipo de funcionalidad que provee el comando tar en ambientes UNIX. El programa tar, es usado para almacenar archivos y directorios en un solo archivo. Dentro de los entornos Unix tar aparece como un comando que puede ser ejecutada desde la línea de comandos de una consola de texto o desde un simple terminal. El formato del comando tar es, comúnmente
tar <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>
donde archivoSalida es el archivo resultado y archivo1, archivo2, etc., son los diferentes archivos que serán "empaquetados" en archivoSalida.
Las opciones más comunes son las siguientes:

-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

Ejemplos
1. Si queremos empacar un directorio llamado "html" y guardar los datos en "html-paq.tar", lo haríamos con la instrucción
tar -cvf html-paq.tar html
2. Si queremos desempaquetar todo el contenido de un archivo llamado xxx.tar podemos utilizar un comando como este
tar -xvf xxx.tar
3. Para archivar el contenido de tres archivos doc1.txt, doc2.txt y data.dat
tar -cvf foo.tar doc1.txt doc2.txt data.dat
4. Si ahora se desea eliminar el contenido del archivo data.dat se ejecutaría
1tar --delete -vf foo.tar data.dat
5. Para agregar ahora un nuevo archivo test.doc a foo.tar se ejecutaría
tar -rvf foo.tar test.doc
Los contenidos se desempaquetarán en el mismo directorio donde estamos situa-dos.

Implementación

Se deberá programar el comando star (“simple tar”, NO “estrella”), de tal forma que acepte los comandos básicos mostrados anteriormente. Note que debe permitir tanto el empacar archivos individuales como directorios completos en forma recursiva. Para desarrollar su programa usted debe tomar 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 en 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.

• Si no hay un hueco del tamaño adecuado en el archivo se debe utilizar alguno de los siguientes mecanismos: (a) expandir el tamaño del archivo ó (b) realizar la compactación del archivo para eliminar huecos pequeños (opcional)

• 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 2esto 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 alma-cenaría 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.

Definición de estructuras de datos

Para este proyecto se utilizó únicamente una estructura de datos:

```
struct table{
    int libres[blocks][2];
    int ocupados[blocks][2];
    char archivos[blocks][256];
    int s_ocupados;
    int s_libres;
    int base;
    int limite;
};
```

De la cual los campos son los siguientes:

libres: Arreglo de enteros el cual guarda en su posición [0] la base de los bloques que se encuentran libres y en la [1] el límite de los mismos

ocupados: Arreglo de enteros el cual guarda en su posición [0] la base de los bloques que se encuentran ocupados y en la [1] el límite de los mismos.

s_ocupados: Mantiene el índice del siguiente bloque en el que se puede insertar un archivo.

s_libres: Mantiene el índice del siguiente bloque que se podría liberar.
En general los dos campos anteriores mantienen el puntero de a donde se tiene que insertar la próxima vez.

Base: Indica el límite del archivo después de haber realizado la inserción de la tabla de direcciones.

Límite: Indica el final del archivo una vez se han insertado todos los archivos.

Principales del programa

Mecanismo de acceso

Archivos

La forma en la que se accedió a los archivos fue con las funciones de fread, fwrite, fopen y fclose el cual se puede considerar el equivalente de las típicas llamadas read, write, open, close. La diferencia entre ambos es que fread cuenta con buffers optimizados para la carga y descarga de contenido.
En este caso las llamadas se hicieron de la siguiente manera

```
FILE *fp;
fp = fopen(filename, "r+");
```

r+ indica que el archivo se abrirá en modo append, mientras que si se utiliza “w+” esto indica que el archivo se truncara y lo creará como si fuera uno nuevo.

```
char temp = 0;
fread(&temp, sizeof(char), 1, fq);  //Se lee los datos del archivo que se quiere empaquetar
fwrite( &temp, sizeof(char), 1, fp); //Se escribe en el nuevo archivo
```

Los procesos de lectura y escritura son los siguientes:

Funciones que para funcionar necesitan de un buffer que en este caso se declara de tipo char ya que este lee únicamente un byte, no es la opción más conveniente debido a que si son archivos muy grandes los que se van a leer y escribir entonces no tiene como que mucho sentido el realizar constantemente llamadas, no obstante, es la forma que representa menor complejidad por lo que se implementó así.

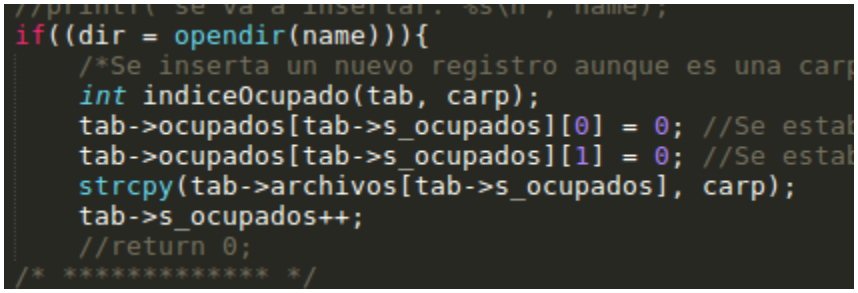
Y por último fclose, el cual es una función a la cual se le pasa el file descriptor para que esta lo cierre.

Directorios

```
DIR *dir;
struct dirent *entry;
```

En la cual se utilizaron variables del tipo DIR y un struct dirent para almacenar la ruta del directorio que se está abriendo y recorrer cada una de las entradas que este directorio posee.
Esta función es recursiva por lo simplemente se realiza la llamada de esta en caso de que la variable de tipo dirent tome un directorio.

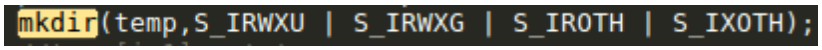
En este caso hay formas para identificar si es un directorio o no:



En ese caso si lo puede abrir exitosamente es un directorio, sino es un archivo.

Parte de los requerimientos de esta tarea es la extracción de los archivos por lo que si se extrae una carpeta que se empaqueto en el archivo tar entonces lo que se debe de hacer es crearla para después extraer cada uno de los archivos en su correspondiente ruta.

La función que se utilizó para la creación de las carpetas es la siguiente:



La cual recibe como parámetros el nombre de la carpeta y los permisos que se tendrán sobre esa carpeta.

Estructura de la tabla de asignación de espacio

La estructura de asignación de espacio se puede evidenciar en el apartado de “Definición de estructuras”, básicamente se puede evidenciar que cuenta con un arreglo de direcciones, tanto la base como la limite, y un arreglo con los nombres de los archivos, en este caso se da una correspondencia entre los índices, por lo que el nombre del archivo que se encuentra en la posición [n] entonces las direcciones asociadas a este archivo se encuentran en el arreglo de ocupados en la posición [n].

Por lo que se puede decir que esta tabla simplemente maneja las direcciones a un nivel lógico, con base en esta tabla se lee o se escribe un archivo. La ubicación de esta tabla siempre se encuentra al inicio del archivo.

Estrategia de administración de bloques libres

A como se especificó en el anunciado del proyecto se utiliza la estrategia del primer ajuste, ya anteriormente se explicó la estructura del arreglo donde se almacenan los bloques libres, entonces básicamente a la hora de asignación espacio se tienen 2 situaciones:

El caso en el que cabe el nuevo archivo en uno de los bloques libres sea lo suficientemente grande como para que el nuevo archivo sea almacenado entonces simplemente se almacena y con el espacio restante se crea un bloque libre
Unir huecos:
Si un archivo no cabe dentro de un hueco existente entonces antes de hacer el archivo más grande, se procede a unir los huecos que se puedan en un hueco de mayor tamaño, en caso de que aún no quepa entonces se hace el archivo más grande.

```
int unirHuecos(struct table *tab){
    int i;
    for(i=0;i<tab->s_libres;i++){
        if(tab->libres[i][1] == tab->libres[i+1][0]){
            tab->libres[i][1] = tab->libres[i+1][1];
            reacomodarLista(tab, 1, i+1);
        }
    }

    return 0;
}
```

Procedimiento de desfragmentación del archivo

La función que se encarga de realizar la desfragmentación del archivo es la siguiente:

```
for(i = 0;i<tab->s_libres;i++){
    if(tab->libres[i][0]<tab->base){
        continue;
    }
    diferencia = tab->libres[i][1] - tab->libres[i][0];
    for(e = 0;e<tab->s_ocupados;e++){
        if(tab->ocupados[e][0]>=tab->libres[i][1]){
            if(v){
                printf("Direccion anterior de: %s, base: %d, limite: %d\n", tab->archivos[e], tab->ocupados[e][0]-tab->base, tab->ocupados[e][1
            }
            tab->ocupados[e][0] = tab->ocupados[e][0] - diferencia;
            tab->ocupados[e][1] = tab->ocupados[e][1] - diferencia;
            if(v){
                printf("Direccion actualizada de: %s, base: %d, limite: %d\n", tab->archivos[e], tab->ocupados[e][0] - tab->base, tab->ocupados
            }
            for(int j = 0;j<tab->ocupados[e][1] - tab->ocupados[e][0];j++){
                fseek( fp, tab->ocupados[e][0] + diferencia +j, SEEK_SET );
                fread(&temp, sizeof(char), 1, fp);
                fseek( fp, tab->ocupados[e][0] +j, SEEK_SET );
                fwrite( &temp, sizeof(char), 1, fp); //Se escribe en el nuevo archivo
            }
        }
    }
}
tab->s_libres = 0;
fseek( fp, 0, SEEK_SET ); // Se reescribe la tabla de direcciones
fwrite( tab, sizeof(struct table), 1, fp);
fclose(fp);

if(d){
    printf("---Tabla de valores despues de la desfragmentacion--\n");
    imprimirValores(tab);
}
```

La cual funciona de la siguiente manera:
Por cada uno de los bloques libres entonces se toman los que tienen direcciones superiores a estos, luego desde el bloque ocupado que tiene una dirección más baja entonces simplemente este se desplaza de una forma negativa el tamaño del bloque libre. En otras palabras, se puede decir que los bloques ocupados se mueven hacia la izquierda de esta manera como compactando, luego finalmente lo que se hace es eliminar todos los bloques libres y de esta manera disminuyendo el tamaño del archivo resultante. Esto se podrá evidenciar en la sección de resultados de pruebas.

Análisis de resultados de pruebas

En este apartado se probará cada una de las funciones que se solicitan en la especificación para que de esta manera quede evidenciado que el correcto funcionamiento de las mismas, no obstante, antes de todo se debe de tomar en cuenta algunas consideraciones:
- Cuando se evidencia la función “dump” entonces la tabla imprimirá las direcciones base como la resta de la dirección real menos la dirección límite de la tabla, esto con el objetivo de que se pueda apreciar mejor el resultado, si se desea la dirección real basta con realizar la suma de la dirección que se imprime en la tabla con la dirección base de la tabla que también se puede observar en la tabla.

-En la misma función “dump“ se verán algunas entrada con números de dirección negativos, estos hacen referencia a carpetas por lo que estas no requieren de espacio.

-En el enunciado se utiliza una vez “--delete” para indicar la operación de eliminar algo, no obstante, en la lista de funciones aparece como “-delete” por lo que se implementó de esta segunda manera.

Para las pruebas se utilizarán carpetas numeradas del 1 al 3 así como archivos numerados del uno al 5 los cuales contienen su mismo nombre.

Creación de un archivo

Archivo

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -cfdv resultado.tar archiv
o3 archivo5
Verbose habilitado, se mostraran las acciones
Dump habilitado, se mostrara el contenido de la tabla de direcciones
Se creara un nuevo archivo
Insertando: archivo3
No hay espacio disponible para asignar, redimensionando archivo
Insertando: archivo5
No hay espacio disponible para asignar, redimensionando archivo
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivo5 base: 9 limite: 18
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 0 base: 557072 limite: 557090
----Fin del resumen----
Se guardo la tabla de direcciones en el archivo, tamaño: 557072
Se copia el contenido de: ./archivo5 al archivo principal
Se establece la base: 557081 y el limite: 557090
Se copia el contenido de: ./archivo3 al archivo principal
Se establece la base: 557072 y el limite: 557081
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivo5 base: 9 limite: 18
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 0 base: 557072 limite: 557090
```

En este caso se puede observar que se realizó la llamada con las opciones de crear, verbose y dump, dump en este caso simplemente despliega el contenido de la tabla de direcciones, verbose indica paso por paso que es lo que se está realizando en el programa.
El resultado persistente del programa es el siguiente:

Siempre que se inserta un archivo que se encuentra en la raíz donde está el ejecutable entonces se añade la entrada [./] para hacer referencia a la misma y tener la integridad de los archivos.

Carpetas

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -cfd resultadoC.tar archiv
o3 carpeta2/
Dump habilitado, se mostrara el contenido de la tabla de direcciones
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 6
nom = [./] base: -557072 limite: -557072
nom = archivo3 base: 0 limite: 9
nom = [carpeta2//] base: -557072 limite: -557072
nom = archivo2 base: 9 limite: 18
nom = [carpeta2//carpeta3/] base: -557072 limite: -557072
nom = archivo4 base: 18 limite: 27
------------
s_ocupados: 6 s_libres: 0 base: 557072 limite: 557099
----Fin del resumen----
```

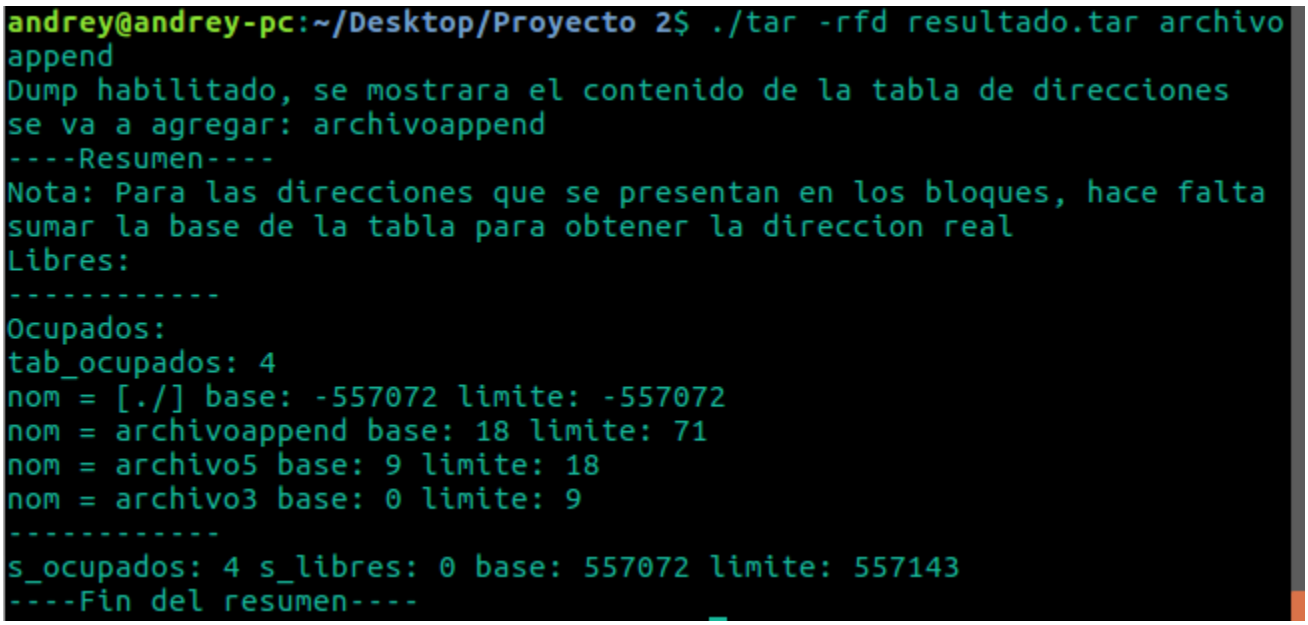
Resultado:

Para hacer referencia a una carpeta es necesario poner el símbolo “/” al final del nombre del parámetro

Append

Para este caso se creará un nuevo archivo llamado: archivoappend que será utilizado para demostrar esta funcionalidad:

Archivos



Siempre que se inserte un archivo este se mostrara seguido de la carpeta a la que pertenece, en este caso se puede evidenciar que el ultimo archivo que se creó se le asignó 18 como límite, el nuevo archivo append cuenta con este número como base.

Carpetas

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -rfd resultadoC.tar carpet
a1/
Dump habilitado, se mostrara el contenido de la tabla de direcciones
se va a agregar: carpeta1/
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 8
nom = [./] base: -557072 limite: -557072
nom = archivo3 base: 0 limite: 9
nom = [carpeta2//] base: -557072 limite: -557072
nom = archivo2 base: 9 limite: 18
nom = [carpeta2//carpeta3/] base: -557072 limite: -557072
nom = archivo4 base: 18 limite: 27
nom = [carpeta1//] base: -557072 limite: -557072
nom = archivoappend base: 27 limite: 80
------------
s_ocupados: 8 s_libres: 0 base: 557072 limite: 557152
----Fin del resumen----
```

Aquí se añadió la carpeta1 el cual contenía el archivoappend, se puede ver que la base de este es el límite de archivo4

```
----Fin del resumen----
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -rfd resultadoC.tar carpet
a1/archivoappend2
Dump habilitado, se mostrara el contenido de la tabla de direcciones
se va a agregar: carpeta1/archivoappend2
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 9
nom = [./] base: -557072 limite: -557072
nom = carpeta1/archivoappend2 base: 80 limite: 95
nom = archivo3 base: 0 limite: 9
nom = [carpeta2//] base: -557072 limite: -557072
nom = archivo2 base: 9 limite: 18
nom = [carpeta2//carpeta3/] base: -557072 limite: -557072
nom = archivo4 base: 18 limite: 27
nom = [carpeta1//] base: -557072 limite: -557072
nom = archivoappend base: 27 limite: 80
------------
s_ocupados: 9 s_libres: 0 base: 557072 limite: 557167
```

En este otro caso se añadió un archivo especifico que se encontraba en la carpeta1 llamado archivoappend 2, de igual manera que el caso anterior se puede evidenciar que este se funciona correctamente.

Actualizar

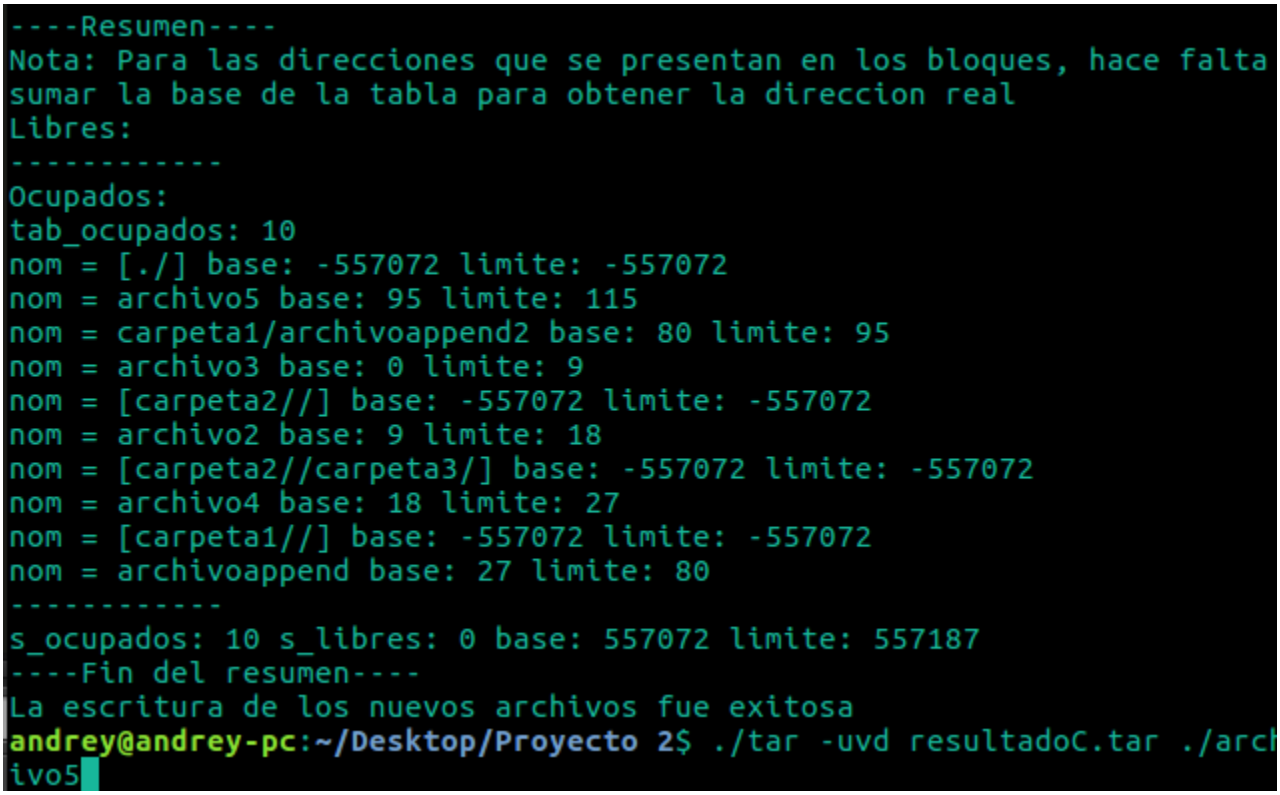
Archivos

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -uvd resultado.tar ./archi
vo5
Verbose habilitado, se mostraran las acciones
Dump habilitado, se mostrara el contenido de la tabla de direcciones
Se procedera a actualizar los archivos seleccionados
Apertura de archivo: resultado.tar
```

```
Libres:
i = 0 base: 9 limite: 18
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivoappend base: 18 limite: 71
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 1 base: 557072 limite: 557143
----Fin del resumen----
Se eliminaron los archivos
Apertura de archivo: resultado.tar
Cargando tabla de direcciones
se va a agregar: archivo5
No hay espacio disponible para asignar, redimensionando archivo
Se copia el contenido de: ./archivo5 al archivo principal
Se establece la base: 557143 y el limite: 557163
Se copia el contenido de: ./archivoappend al archivo principal
Se establece la base: 557090 y el limite: 557143
Se copia el contenido de: ./archivo3 al archivo principal
Se establece la base: 557072 y el limite: 557081
Guardando cambios
Tabla de direcciones
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
i = 0 base: 9 limite: 18
------------
Ocupados:
tab_ocupados: 4
nom = [./] base: -557072 limite: -557072
nom = archivo5 base: 71 limite: 91
nom = archivoappend base: 18 limite: 71
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 4 s_libres: 1 base: 557072 limite: 557163
----Fin del resumen----
La escritura de los nuevos archivos fue exitosa
```

En este caso lo que se hizo fue modificar archivo5 añadiéndole mas texto, por lo que aumento de tamaño, esta es la razón por la que se haya asignado más espacio y se haya dejado libre el hueco anterior.

Carpetas



En este caso se puede ver que el funcionamiento es correcto, no obstante, el valor del contador de libres se disminuye a cero por lo que no se presenta, pero realmente hay un hueco desde la posición 80 que es el límite del archivo append y 95 que es la base del archivo 5.

Listar

Archivos

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -t resultado.tar
Lista de archivos:
-[./]
-archivo5
-archivoappend
-archivo3
```

Siguiendo la línea anterior del dump se puede verificar que la solución es correcta.

Carpetas

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -t resultadoC.tar
Lista de archivos:
-[./]
-archivo5
-carpeta1/archivoappend2
-archivo3
-[carpeta2//]
-archivo2
-[carpeta2//carpeta3/]
-archivo4
-[carpeta1//]
-archivoappend
```

Siguiendo la línea anterior del dump se puede verificar que la solución es correcta

Eliminar

Archivos

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -delete -d resultado.tar .
/archivo5
Apertura de archivo: resultado.tar
Eliminando en cascada
el archivo es: ./archivo5
Eliminando archivo: archivo5
Actualizando tabla de direcciones
```

Se puede ver que al eliminar el archivo 5 entonces en la tabla de direcciones se puede comprobar el nuevo bloque libre.

```
Libres:
i = 0 base: 9 limite: 18
i = 1 base: 71 limite: 91
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivoappend base: 18 limite: 71
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 2 base: 557072 limite: 557163
----Fin del resumen----
Lista de archivos:
```

Carpetas

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -delete -vd resultadoC.tar
 carpeta2/
Apertura de archivo: resultadoC.tar
Eliminando en cascada
se agrego: [carpeta2//]
se agrego: archivo2
se agrego: [carpeta2//carpeta3/]
se agrego: archivo4
Eliminando archivo: carpeta2/
no se puede eliminar: carpeta2/
Eliminando archivo: [carpeta2//]
Eliminando archivo: archivo2
Eliminando archivo: [carpeta2//carpeta3/]
Eliminando archivo: archivo4
Actualizando tabla de direcciones
Se eliminaron los archivos
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -td resultadoC.tar
Dump habilitado, se mostrara el contenido de la tabla de direcciones
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
i = 0 base: -557072 limite: -557072
i = 1 base: 9 limite: 18
i = 2 base: -557072 limite: -557072
i = 3 base: 18 limite: 27
------------
Ocupados:
tab_ocupados: 6
nom = [./] base: -557072 limite: -557072
nom = archivo5 base: 95 limite: 115
nom = carpeta1/archivoappend2 base: 80 limite: 95
nom = archivo3 base: 0 limite: 9
nom = [carpeta1//] base: -557072 limite: -557072
nom = archivoappend base: 27 limite: 80
------------
s_ocupados: 6 s_libres: 4 base: 557072 limite: 557187
```

En este caso se eliminó carpeta2, esta carpeta a su vez tenía otras carpetas y archivos, por lo que se procede a realizar el borrado en cascada. Una vez eliminados los archivos los bloques se dejan libres, esto se puede evidenciar en el screenshot anterior, basado en la tabla de direcciones.

Extraer

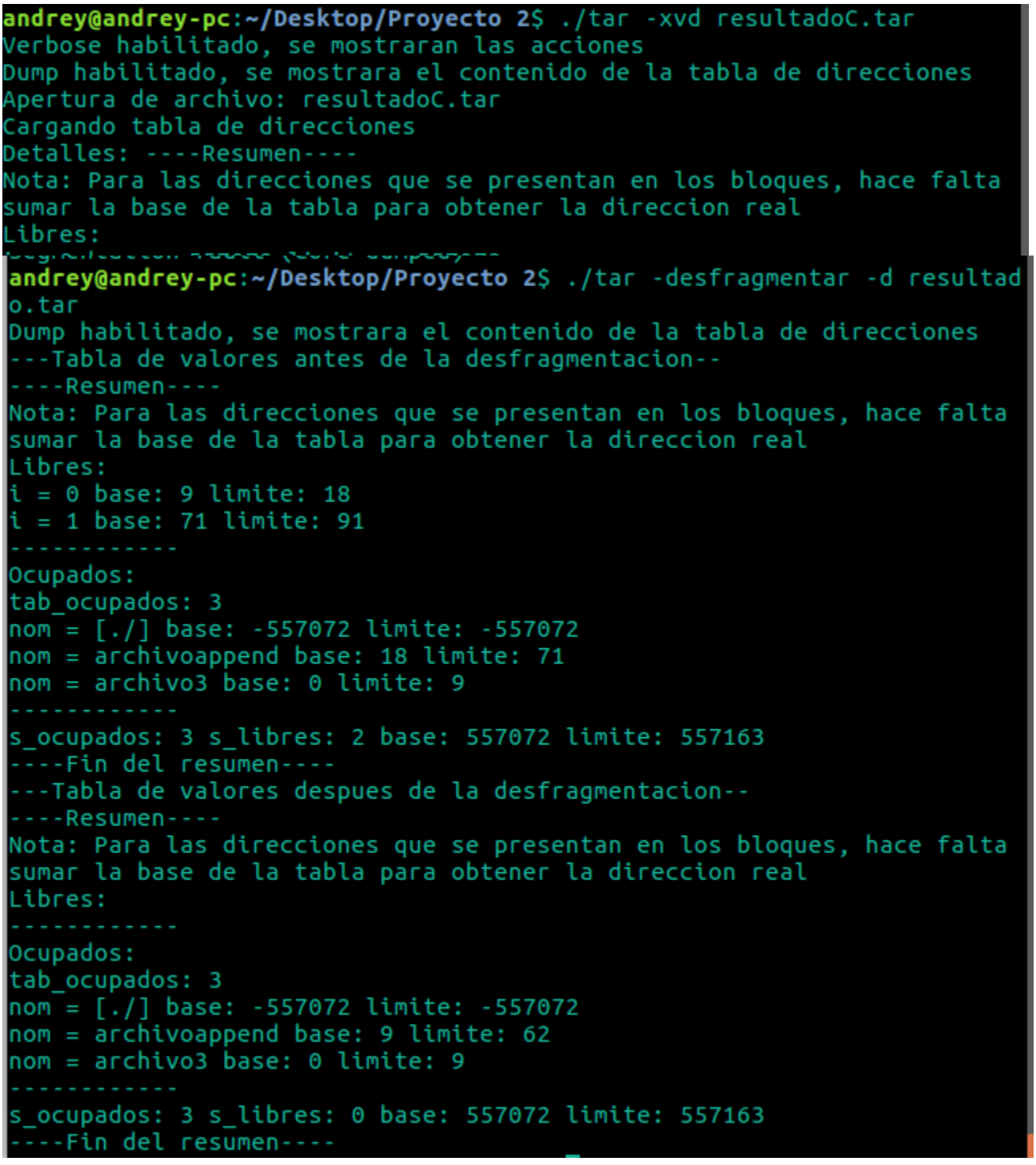
Archivos

```
Definiendo direccion base: 557072 y limite: 557081
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -xvd resultado.tar
Verbose habilitado, se mostraran las acciones
Dump habilitado, se mostrara el contenido de la tabla de direcciones
Apertura de archivo: resultado.tar
Cargando tabla de direcciones
Detalles: ----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivoappend base: 9 limite: 62
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 0 base: 557072 limite: 557163
----Fin del resumen----
file: [./]
file: archivoappend
ruta: ./
Definiendo direccion base: 557081 y limite: 557134
file: archivo3
ruta: ./
```

Si se ejecuta este comando después de realizar la secuencia de operaciones anteriores y se extrae el resultado será el que se muestra anteriormente. Si se ejecuta lo anterior se puede comprobar la veracidad de esta afirmación

Carpetas

Igual que en el caso anterior, lo presentado en el screenshot es el log de las acciones que suceden a la hora de realizar la extracción.



Desfragmentar

```
andrey@andrey-pc:~/Desktop/Proyecto 2$ ./tar -desfragmentar -d resultad
o.tar
Dump habilitado, se mostrara el contenido de la tabla de direcciones
---Tabla de valores antes de la desfragmentacion--
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
i = 0 base: 9 limite: 18
i = 1 base: 71 limite: 91
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivoappend base: 18 limite: 71
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 2 base: 557072 limite: 557163
----Fin del resumen----
---Tabla de valores despues de la desfragmentacion--
----Resumen----
Nota: Para las direcciones que se presentan en los bloques, hace falta
sumar la base de la tabla para obtener la direccion real
Libres:
------------
Ocupados:
tab_ocupados: 3
nom = [./] base: -557072 limite: -557072
nom = archivoappend base: 9 limite: 62
nom = archivo3 base: 0 limite: 9
------------
s_ocupados: 3 s_libres: 0 base: 557072 limite: 557163
----Fin del resumen----
```

Como se puede observar la desfragmentación fue exitosa debido a que inicialmente existían bloques libres, tras realizar la desfragmentación se puede ver que estos se eliminaron y se redimensionaron las direcciones de las entradas que estaban ocupadas.

Conclusiones sobre rendimiento

- El uso de la memoria contigua no es una tan buena opción debido a que si se requiere realizar muchas operaciones de una forma ocasional se tienen que reacomodar el archivo internamente.
- Se puede hacer uso de hilos para optimizar estas rutinas debido a que no hay funcionalidades que interactúan directamente con algunas zonas criticas
- Debido a que la programación es realizada en C y se hace uso de estructuras lineales, no multidimensionales entonces el rendimiento es mayor.
- Si el archivo estuviera comprimido de cierta manera entonces sería más rápido tanto la lectura como escritura además de posiblemente sería más fácil el direccionamiento de los archivos.
- Ocasionalmente un archivo temporal puede bajar el nivel de complejidad, pero no obstante puede bajar el rendimiento