Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Principios de Sistemas Operativos - IC6600

Proyecto II

Estudiantes:
Axel Alexander Chaves Reyes - 2021099588
Kevin Vinicio Núñez Cruz - 2021118002

Profesor:
Armando Arce Orozco

Semestre I

2024

Índice

Introducción

A continuación se presenta un informe técnico detallado del segundo proyecto asignado en el curso "Principios de Sistemas Operativos", a cargo del profesor Armando Arce Orozco. Este proyecto, en esencia, consiste en la creación de un programa diseñado para el empacamiento de archivo, imitando el comando *tar* utilizado en ambientes UNIX.. En términos más simples, su función principal radica en almacenar múltiples archivos en un solo archivo.

El propósito fundamental de este proyecto radica en la aplicación práctica de los conocimientos adquiridos durante las clases previas del curso. Esto implica distintos conceptos, tales como los relacionados con los sistemas de archivos y manejo de directorios, así como la habilidad para el manejo de bloques libres y desfragmentación de archivos.. Además, este ejercicio se enfoca en el empleo del lenguaje de programación C como el medio principal para el desarrollo de dichas habilidades y conocimientos. A través de la implementación de este proyecto, se consolida la comprensión de la teoría mediante la resolución de problemas prácticos, fomentando así un aprendizaje más profundo y significativo.

El presente informe contiene las secciones que se mencionan a continuación.

Primero, la descripción del problema, donde se menciona el enunciado que el profesor Armando brindó.

Segundo, la definición de las estructuras de datos, apartado donde se mencionan temas más técnicos y que detallan los elementos que fungen como las herramientas fundamentales de almacenamiento de datos para construir el proyecto.

Tercero, los componentes principales del programa, sección donde se describen herramientas brindadas por el lenguaje y también desarrolladas por el equipo que cumplen con los requisitos del proyecto.

Cuarto, el mecanismo de manejo de sistemas de directorios y fragmentación de archivos, donde se describe el lado creativo y técnico que llevó a la infraestructura de procesos desarrollada.

Quinto, las pruebas de rendimiento, las cuales demuestran la capacidad del programa de resolver el problema propuesto ante ciertos escenarios.

Finalmente, las conclusiones, donde se hace reflexión sobre el proyecto realizado y los resultados obtenidos a modo de análisis para posibles cambios o futuras implementaciones reales de este tipo de software.

Descripción del Problema

El enunciado propuesto es el siguiente:

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 múltiples archivos 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: empacar contenidos de archivo, si no está presente asume la entrada estándar.
- r, --append: agrega contenido a un archivo
- p, --pack: desfragmenta el contenido del archivo (no presente en tar)

Definición de Estructuras de Datos

El sistema desarrollado contiene varias estructuras de datos y definiciones, las cuales se detallan a continuación.

1. Estructura *FileMetadata*: esta estructura almacena metadatos sobre un archivo individual dentro del archivo contenedor. Se incluye el nombre del archivo, su tamaño total, las posiciones de los bloques de datos que lo componen dentro del archivo contenedor y el número total de bloques.
2. Estructura *Data*: esta estructura almacena las opciones y argumentos pasados al programa a través de la línea de comandos. Además, controla las operaciones que el programa debe realizar y los archivos involucrados. Contiene los campos booleanos *create, extract, list, delete, update, verbose, veryVerbose, file, append* y *defrag.* Además, contiene *outputFile* que contiene un puntero a un caracter que almacena el nombre del archivo contenedor sobre el cual operar. También se encuentra *inputFiles,* que contiene un puntero a un caracter que almacena los nombres de los archivos de entrada para ciertas operaciones. Finalmente, contiene *numInputFiles,* el cual es un entero que indica el número de archivos de entrada.
3. Estructura *AllocationTable:* Esta estructura actúa como una tabla de asignación de archivos, similar a las utilizadas en sistemas de archivos reales. Mantiene un registro de todos los archivos dentro del archivo contenedor, así como de los bloques de datos libres. Esta estructura contiene los campos *files, numFiles, freeBlocks y numFreeBlocks.*
4. Estructura *Block*: representa un bloque de datos dentro del archivo contenedor. Cada bloque tiene un tamaño fijo definido por *BLOCK_SIZE*. Esta estructura contiene un arreglo que almacena los datos del bloque. Su tamaño es justamente *BLOCK_SIZE*, definido como 256 KiloBytes en este código.

Además de las estructuras mencionadas, otras implementaciones que tiene el proyecto corresponden a funciones y procesos que realizan la copia de directorios, los cuales se detallan en la sección de componentes.

Descripción y Explicación de los Componentes Principales del Programa

A continuación, se describen los componentes principales del sistema desarrollado y su importancia en la implementación.

Primero, las bibliotecas. Este programa incluye distintos módulos externos que desempeñan varias funciones; entre ellas, podemos encontrar funcionalidades para trabajar con las entradas y salidas del sistema, manejo de archivos, mensajes, bibliotecas estándar y de propósito general, funciones de bajo nivel, manipulación de procesos, argumentos de la línea de comandos, entre otros.

Segundo, la función *findFreeBlock, la cual se encarga de* localizar un bloque libre en la tabla de asignación de archivos (FAT). Esta función recorre la lista de bloques libres y devuelve la posición del primer bloque disponible, marcándolo como ocupado. Si no hay bloques libres, devuelve un valor especial indicando que no se encontró ningún bloque libre. Esta función es crucial para la gestión eficiente del espacio dentro del archivo contenedor.

Tercero, la función *expandArchive,* la cual es responsable de expandir el tamaño del archivo contenedor cuando no hay bloques libres disponibles. Esta función mueve el puntero del archivo al final, determina el tamaño actual del archivo y lo incrementa en el tamaño de un bloque. Luego, actualiza la FAT para incluir el nuevo bloque libre. Esta expansión dinámica permite que el sistema maneje archivos de manera flexible, adaptándose a las necesidades de almacenamiento.

Cuarto, para listar el contenido del archivo contenedor, la función *listArchiveContents* abre el archivo en modo de lectura y carga la FAT. Luego, recorre la lista de archivos almacenados y muestra sus nombres y tamaños. Si se especifica el modo detallado, también muestra las posiciones de los bloques que componen cada archivo. Esta función proporciona una visión clara del contenido del archivo contenedor, facilitando la gestión de los archivos almacenados.

Quinto, lafunción *writeBlock,* la cual escribe un bloque de datos en una posición específica del archivo contenedor. Esta función mueve el puntero del archivo a la posición deseada y escribe el bloque de datos. Es fundamental para la operación de almacenamiento de datos dentro del archivo contenedor.

Sexto, la función *updateFAT* actualiza la FAT con la información de un nuevo bloque de datos. Si el archivo ya existe en la FAT, agrega la nueva posición del bloque a la lista de bloques del archivo y actualiza su tamaño. Si el archivo no existe, crea una nueva entrada en

la FAT. Esta función asegura que la FAT siempre refleje con precisión el estado actual del archivo contenedor.

Séptimo, para escribir la FAT en el archivo contenedor, se utiliza la función *writeFAT*. Esta función mueve el puntero del archivo al inicio y escribe la FAT. Es crucial para mantener la integridad de la FAT y asegurar que los cambios persistan en el archivo contenedor.

Octavo, la función *createArchive* crea un nuevo archivo contenedor y agrega archivos a él. Inicializa la FAT, escribe la FAT en el archivo contenedor y luego agrega los archivos especificados, dividiéndolos en bloques y actualizando la FAT en consecuencia. Esta función es fundamental para la creación y organización inicial del archivo contenedor.

Noveno, la función *extractArchive* extrae los archivos almacenados en el archivo contenedor. Abre el archivo contenedor, carga la FAT y recorre la lista de archivos, extrayendo cada uno de ellos bloque por bloque y escribiéndolos en archivos de salida. Esta función permite recuperar los archivos almacenados en el archivo contenedor.

Décimo, para eliminar archivos específicos del archivo contenedor, se utiliza la función *deleteFilesFromArchive*. Esta función abre el archivo contenedor en modo de lectura y escritura, carga la FAT y recorre la lista de archivos, eliminando los archivos especificados y marcando sus bloques como libres en la FAT. Luego, escribe la FAT actualizada en el archivo contenedor.

Undécimo, la función *updateFilesInArchive* actualiza archivos específicos dentro del archivo contenedor. Abre el archivo contenedor, carga la FAT, elimina los bloques antiguos del archivo a actualizar y agrega los nuevos bloques, actualizando la FAT en consecuencia. Esta función permite mantener los archivos almacenados actualizados.

Duodécimo, la función defragmentArchive desfragmenta el archivo contenedor para consolidar los bloques de datos y liberar espacio. Mueve los bloques de datos para que estén contiguos y actualiza la FAT con las nuevas posiciones de los bloques. Luego, trunca el archivo para eliminar el espacio no utilizado. Esta función mejora la eficiencia del almacenamiento dentro del archivo contenedor.

Finalmente, la función *appendFilesToArchive* agrega archivos adicionales al archivo contenedor existente. Abre el archivo contenedor, carga la FAT y agrega los nuevos archivos, dividiéndolos en bloques y actualizando la FAT. Esta función permite expandir el contenido del archivo contenedor de manera flexible.

Pruebas de Rendimiento

Al analizar el rendimiento de las distintas funciones que realiza el sistema basado en un par de archivos utilizados, es posible concluir lo siguiente:

1. El comando *-cvf*, encargado de empaquetar los archivos, termina su función en 0.002390 segundos y 3648 Bytes de memoria utilizados.
2. El comando *-x,* cuya función es crear un archivo *tar,* termina su ejecución en 0.000379 segundos consumiendo 3660 Bytes de memoria.
3. El comando *-tv* se encarga de listar los archivos que contenga un *tar*, utilizando 3676 Bytes de memoria a lo largo de 0.000556 segundos.
4. El comando *-d,* cuya función consiste en eliminar un archivo específico contenido en el *tar*, usa 3688 Bytes de memoria a lo largo de los 0.000956 segundos que toma su ejecución.
5. El comando *-p,* cuya función es desfragmentar el archivo, utiliza 3696 Bytes en 0.000237 segundos de ejecución.
6. El comando *-rf*, encargado de añadir nuevos archivos al *tar*, utiliza 3700 Bytes de memoria durante 0.039732 segundos que toma su ejecución.

Luego de realizar pruebas al código, es importante destacar que la solución implementada contiene bajos costos computacionales, tanto en memoria como en tiempo.

Conclusiones

Una vez finalizado el análisis y descripción del proyecto realizado, el equipo realizó las conclusiones que se mencionan a continuación.

Primero, el código desarrollado cumple adecuadamente con los requerimientos planteados en la especificación, por lo que el resultado es el indicado.

Segundo, el sistema creado muestra una solución limpia e incluso sencilla del problema a resolver, dado que las estructuras utilizadas no utilizan memoria en exceso y no generan bucles, errores o estados que comprometan el funcionamiento del programa.

Tercero, los resultados de las pruebas de rendimiento demuestran que la implementación del programa para manejar archivos contenedores es eficiente. El enfoque utilizado, que incluye la gestión de archivos mediante una tabla de asignación de archivos (FAT) y la expansión dinámica del archivo contenedor, permite aprovechar eficientemente los recursos del sistema y lograr tiempos de ejecución reducidos, incluso para cargas de trabajo significativas. Tanto la creación como la extracción de archivos arrojaron resultados muy positivos, lo que sugiere que la solución es escalable y capaz de adaptarse a diferentes requisitos. La inclusión de opciones detalladas y muy detalladas para la salida del programa también facilita el monitoreo y la optimización del rendimiento, asegurando que el sistema pueda manejar eficientemente una variedad de operaciones y tamaños de archivos.

En síntesis, el programa desarrollado cumple satisfactoriamente con los requisitos establecidos, ofreciendo una solución limpia y eficiente para el empaquetamiento de archivos. Los resultados de las pruebas de rendimiento demuestran que el enfoque utilizado, basado en la gestión de archivos mediante una tabla de asignación de archivos (FAT) y la expansión dinámica del archivo contenedor, permite aprovechar eficientemente los recursos del sistema y lograr tiempos de ejecución reducidos, incluso para cargas de trabajo significativas. Tanto la creación como la extracción de archivos arrojaron resultados muy positivos, lo que sugiere que la solución es escalable y capaz de adaptarse a diferentes requisitos. La inclusión de opciones detalladas y muy detalladas para la salida del programa también facilita el monitoreo y la optimización del rendimiento, asegurando que el sistema pueda manejar eficientemente una variedad de operaciones y tamaños de archivos. En conjunto, estas características convierten al programa en una alternativa sólida y confiable para la gestión de archivos empaquetados, ofreciendo una solución robusta y adaptable a diversas necesidades de almacenamiento y recuperación de datos.