Vina Plus Plus

Mardoqueu Freire Nunes
GRR20211773
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
mardoqueu@ufpr.br

***Resumo*—Neste trabalho, tive como objetivo implementar um arquivador chamado vina++.**

I. Introdução

O trabalho consiste em implementar um arquivador chamado de Vina++. O arquivador tem sete opções de entrada que consiste em: inserir/acrescentar, inserir com opção -a, extrair, mover, remover, listar e uma help. O programa rodará por linha de comando.

II. Insere

O insere ele insere/acrescenta os arquivos ao arquivador. Se caso o arquivo já existir ele o substitui. Se caso ele ainda não existir no arquivador ele é adicionado no fim.

O processo de insere quando os arquivos são passados de uma só vez é bem tranquila. Porém quando e preciso acrescentar ou susbstituir, a coisa complica.

Para o processo de substituição, o programa trabalha com o que chamei de borda. Ela pega todo o contéudo quem vem logo depois do membro que quero substituir e jogo no final do arquivo, a borda. Depois é feito a substituição do membro e volto com todo o conteúdo que estava na borda logo após onde o novo membro termina. Nesse processo, para não perder os metadados, guardei em uma lista encadeada com estrutura mais abaixo.

Quando é preciso apenas acrescentar no fim, guardo os meta dados na estrutura e escrevo o novo membro no final, depois reescrevo os metadados.

A. Estrutura:

```
struct nodo
char *nome;
char *modo;
char *id;
char *size;
char *data;
char *ordem;
struct nodo *prox;
;

struct Lista
struct nodo *cabeca;
struct nodo *cauda;
;
```

B. Insere com -a:

O insere com -a, se diferencia do insere normal, pois ele só substitui se o arquivo é mais recente que o que está no arquivador. Para a vereficação foi criada um fução de protótipo: int compareRecently(FILE *arq, char *membro, char *sub).

Ela compara as duas datas, uma retirada do meta dados e a outra na passada do parâmetro e com a ajuda da biblioteca time.h, da função strptime() e da struct tm, foi feita comparação das duas datas.

III. Extrai

A opção de extrai, primeiramente, verefica se deve extrai todos os membros ou um específico.(O meu extrai está funcionando com arquivos textos apenas)

A. Extrai tudo:

Quando é pedido para extrair tudo, é guardado os meta dados na estrutura e dentro de um loop é pego o seu nome e o seu tamanho. E, então é criado um novo arquivo com o nome do arquivo a ser extraido, seu conteúdo e seu modo. É feito isso até a extração de todo os arquivos.

B. Extrair específico:

O extrair de forma específica, trabalha de maneira semelhante. Porém é procurado em qual bloco do arquivo ele sem encontra primeiro, para depois realizar e extração.

IV. Move:

O parametro -m, aciona a função move do arquivador onde faz com que um membro seja movido para trás de um outro membro dentro do arquivador. Sem dúvida umas das funções mais complicadas.

Toda a função foi feita com o mesmo pensamento da borda. O membro target, era o membro referencia, indicava que deveria ser movido um membro para trás do target. Então, para isso foi pego todo o conteúdo que se tinha atrás do target e colocado na borda, no final do arquivador. O membro a ser movido também era jogado para lá, mas em um outro bloco que eu chamei de beyondEdge, alé da borda. Dessa forma eu sabia onde estava os membros bordas e o membro além da borda.

Depois de mover para borda e além da borda, eu trago o membro além da borda para trás do target, reescrevo todos os

membros atrás do membro que foi movido e trunco o arquivo com a função ftruncate(). Trunco o arquivo nessa posição porque o que vem depois não importa mais, ao não ser os meta dados, porém estão todos em uma estrutura.

Por fim, escrevo as estrutura com os metadados no fim do arquivo. Toda essa brincadeira de mover arquivo para borda, além da borda, e depois volta com o arquivdor como era, fui muito trabalhoso e custou muito tempo (isso para todas as opções).

V. Remove

Remover um membro do arquivador também é muito trabalhoso. Para a remoção eu criei a função int foundFile(FILE *arq, char *membro, int *fim, char **nome, int *comeco), que me diz o nome do membro a remover e o começo e o fim de seu bloco.

Definido isso, pego todo o conteúo que vem logo depois dele, guardo em um buffer e apenas escrevo por cima do membro que queria remover. Trunco o arquivo no fim dessa escrita e reescrevo os meta dados, atualizado sem o membro que que foi removido, no final do arquivador.

No código é tratado dois casos, quando o membro é o primeiro ou não. A diferença se dá nos cálculos para os pulos com o fseek(). Quando o memebro não é o primeiro, esses cálculos se complicam ainda mais.

Função sujeita a erros:(

VI. Listar

O listar é uma opção que lista os arquivos que estão no arquivador. O list foi um dois primeiros a ser implementado no arquivador, por isso não é usado nenhuma estrutura de dados neles, todo o conteúdo a ser mostrado é pego direto dos meta dados no arquivador.

Foi implentado funções de forma que pudesse fazer essa manipulação no arquivador, de pegar o conteúdo diretamene de lá e listar. A ideia no começo era que todo o arquivador fosse implementado dessa forma, porém ficou nítido depois que uma estrutura deveria ser criada ou que deveria ter sido criada.

Junto com essas funções de manipulações, foi precisso com que o arquivador fosse construido pelo Insert de uma forma que elas não quebrassem, então foi criado ali uma base de arquivador para ser usado durante todo o trabalho.

VII. Organização do Arquivador

No começo foi criado um arquivador na mão, sem ser pelo insert, para saber como deveria ser criado um arquivador que facilitaria o meu trabalho. Então foi feitos os seguintes ajustes:

A. Bloco e Nome:

O inicio de cada membro guarda o tamanho de seu bloco e o nome de seu bloco, que é o nome do membro. O tamanho de cada bloco facilita os pulos para cada membro dentro do arquivador. Foi implementado a função takeJump(), que pega esse bloco, converte para inteiro e realiza o pulo com o fseek() para qualquer membro.

exemplo: 586@poema.txt@, os @ são separados para facilitar na identificação.

B. Meta Dados:

Os meta dados são todos guardados no fim do arquivador, pois facilitado quando devo acrescentar novos membros. Os meta dados são escritos de uma forma que eles fiquem juntos, porém separados com separadores, que facilitam na leitura de funções antigas, antes implementadas sem estruturas.

exemplo:poema.txt-ID:1000-MODE:33204-SIZE:572-TIME:Sun Jun 25 15:37:21 2023-ORDEM:1, no final do meta dados tem o caracter de cerquilha no final é para indicar o fim do diretorio de metadados .

VIII. Conclusão

Esse foi sem dúvida o maior, mais longo e mais dificil trabalho no qual eu fiz. Foram muitas semanas e noites de programação para a realizão de algumas das opções propostas. Apesar de implementar todas as opções pedidas, não são todas que executam com exitos. Durante todo o trabalho, foi realizado testes com arquivos textos, para facilitar, dessa forma é possível que algo fora dos conformes possa acontecer.