40090 - 45491

27 de Junho 2021

Universidade
de Évora

Arquitetura de Computadores I

Trabalho Prático - Editor de texto em linha

40090 Duarte Anastácio
45491 Beatriz Girão

1 Introdução

Este trabalho teve como objetivo o desenvolvimento de um editor de linha que implementasse os principais comandos de abertura, edição e gravação de ficheiros. A primeira fase foi realizada na linguagem C, este relatório é relativo apenas à segunda parte que foi implementada em Assembly RISC-V.

2 Estrutura de Dados

Para o desenvolvimento do programa pedido, decidimos que seria viável a utilização de uma Lista Duplamente Ligada como estrutura de dados, isto porque durante a execução do programa iremos inserir linhas de texto em qualquer posição, se usássemos um array as operações de inserção poderiam ter uma complexidade quadrática, isto porque ao inserir no inicio por exemplo, teriamos que deslocar todos elementos do array consecutivamente, o que seria um problema cada vez maior à medida que o numero de elementos aumentava. Com a Lista Duplamente Ligada é possível economizar tempo em cada operação à excessão do acesso, que seria mais rápido com a utilização de um array. - Esta estratégia facilitou bastante a resolução do problema em C, no entanto ao traduzir para Assembly RISC-V encontrámos vários desafios e o resultado não foi o idealizado.

3 Funções

Para a implementação da Lista Duplamente Ligada implementámos as seguintes funções:

Node* newnode(char* data): Esta função aloca 12 bytes (3 words) na Heap, a primeira word contem um apontador para o próximo nó, a segunda word um apontador para o nó anterior e a terceira word um apontador para uma string que corresponde a uma linha de texto. Devolve o endereço do espaço alocado.

List* newlist(Node* N) Esta função aloca 12 bytes na Heap (3 words), a primeira word vai conter o tamanho da lista, a segunda word vai conter um apontador para a Head da lista e a terceira word um apontador para a Tail da lista. O tamanho é inicialiado a 0 ou 1 caso exista argumento ou não, o Nó passado como argumento irá definir a Head e a Tail da lista. Devolve o endereço do espaço alocado.

List* newcmd(char* data) Esta função aloca 12bytes na Heap (3 words) e recebe uma string que irá corresponder ao comando do utilizador, de seguida na primeira word é colocado o primeiro argumento numérico, na segunda o segundo arguemnto e na terceira word o char correspondete ao comando a ser executado. Devolve o endereço do espaço alocado.

Txt* newEdTxt() Esta função aloca 8bytes na Heap (2 words), e vai ser a representação em memória do ficheiro de texto. A primeira word contém um apontador para uma lista duplamente ligada que contém as linhas do texto em cada nó, a segunda word contém um apontador para a linha atual que é apenas o ultimo nó da lista presente numa operação. Devolve o endereço do espaço alocado.

void listAdd(List L, char* data) Esta função vai criar um novo nó com o argumento passado no **data** e vai adicionar no fim da lista.

void listPush(List L, Node N) Esta função vai colocar um nó no inicio da lista.

void deletelist(List L) Esta função função faz uma *Lazy deletion* na lista, isto é, define a head e a tail como sendo NULL.

void listinsert(List L, char* data, int pos) Esta função cria um novo nó e insere-o na lista no indíce passado na **pos**.

void insertTxt(Txt txt, char* data, int pos Esta função pede input ao utilizador, cria um nó com esse input e insere na posição anterior à passada no **pos**.

void appendTxt(Txt txt, char* data, int pos Esta função pede input ao utilizador, cria um nó com esse input e insere na posição seguinte à passado no **pos**.

void changeTxt(Txt txt, char* data) Esta função pede input ao utilizador e modifica o nó na posição passada no **pos**, este nó agora contém o input.

void deleteTxt(Txt txt, int pos) Esta função destrói o nó (linha) na posição definida em **pos**.

void printList(List L) Esta função imprime todos os nós (linhas) da lista (texto).

void printinRange(List L, int a, int b) Imprime a string contida em todos os nós (linhas) entre o indíce **a** e **b**.

void deleteinrange(List L, int a, int b) Destrói todos os nós entre o índice **a** e **b**.

void printNode(List L, int x) Imprime a string contida no nó (linha) do indíce **x**.

Node* listGet(List L, int x) Devolve o endereço do nó no índice **x**.

bool isop(char c) Devolve o 1 caso o char **c** seja uma das operações possíveis do editor de texto, caso contrário devolve 0.

bool isNum(char c) Devolve 1 caso o char **c** represente um número entre 0 e 9, caso contrário devolve 0.

bool isToken(char c) Devolve 1 caso o char **c** represente um caracter especial (% ou $), caso contrário devolve 0.

void strcpy(char* dest, char* src, int n) Copia **n** chars da **src** para a **dest**.

int wtoi() Traduz uma word com 4bytes correspondentes a chars numéricos (0-9) com ordenação Little Endian para um inteiro. Não tem equivalente em C.

int atoi(char* s) Traduz uma string de chars numéricos (0-9) para um inteiro.

void puts(char *s, int n) Imprime **n** chars da string **s**.

void gets(char *s, int n) Pede **n** chars de input ao utilizador e guarda-os no endereço **s**.

4 Conclusão e Reflexão

O desenvolvimento deste trabalho obrigou-nos a compreender como se comporta a memória durante a execução de um programa e como se gere a mesma para resolver um problema computacional. Ao tentarmos usar uma lista duplamente ligada explorámos acima de tupo a região da Heap pois tivemos que alocar memória dinâmica em todos os nossos constructores, o maior desafio foi de longe a organização do código Assembly em si, a implementação de uma estrutura de dados mais complexa rapidamente tornou o nosso código mais confuso e fácil de perder controlo sobre o fluxo de execução do mesmo ainda que comentado extensivamente, isto criou mais espaço para erros "não percétiveis" durante o desenvolvimento.

Este documento foi produzido em LaTeX.