Editor de Texto "ed"

André Baião 48092
Gonçalo Barradas 48402

junho, 2021

U.C. Arquitectura de Computadores
Licenciatura em Engenharia Informática

Docentes
Miguel Barão
Pedro Salgueiro

1 Introdução

Para avaliar o conhecimento e o domínio em *RISC-V* adquirido ao longo do semestre par, foi-nos proposto elaborar um código em *RISC-V* para simular o editor de texto "ed", editor de texto padrão do Unix. Este editor de texto pode ser testado num terminal de um computador (S.O Linux por exemplo). O projecto deve elaborar as funções do editor de texto como a função *insert* (comando i) que insere texto antes da linha actual, a função *append* (comando a) que insere texto na linha actual, a função *change* (comando c) que substitui a linha actual por novo texto entre outras, e foi dividido em duas fases.
A primeira fase deste projecto foi a elaboração de um código em linguagem c, para auxiliar a realização do código em *Assembly*. A segunda fase do projecto consiste na elaboração do código em *Assembly* utilizado *RISC-V*.

2 Metodologia

Como mencionado acima o projecto foi dividido em duas fases, a primeira consistia em elaborar um código em linguagem c e a segunda um código em *Assembly RISC-V*. Para realizar o código em linguagem c, foi necessário avaliar e definir a estrutura de dados que mais se adequava ao problema em questão. Desta maneira foi decidido elaborar o código utilizando "*Doubly Linked Lists*". A segunda opção seria utilizar *Arrays*, mas apesar desta forma ser menos propensa a erros optámos por utilizar as *Doubly Linked Lists* com o objectivo de poder dinamizar o projecto de forma eficiente. Foi assim elaborado o código em "c", com todas as funções necessárias para simular este editor de texto, para que pudéssemos proceder para a realização do código em *Assembly* com o auxílio de uma linguagem de alto nível.

3 Desenvolvimento

Nesta fase do projecto começámos por tentar perceber como simular apontadores, visto que não é possível utilizá-los como na linguagem "c", e com o desenvolver das funções, reparou-se que não seguíamos o código em c realizado e que seguiríamos por desenvolver o código em *assembly* de forma autónoma.
Foram estabelecidos no início deste projecto alguns registos de função para guardar argumentos:
a1 == Guarda a posição final do texto.
a2 == *"SAFE"* Este registo guarda 1 ou 0 caso o ficheiro esteja guardado ou não, é necessário para a função "q" que só sai do editor se o ficheiro estiver guardado.
a3 == Este registo guarda o número de linhas, isto é, conta as linhas já inseridas e por si conta também quando necessário a posição a inserir ou a apagar do ficheiro.
a4 == Neste registo vai ficar guardada a posição a ser inserido ou eliminado o texto (dependendo da instrução), assim este registo pode ser comparado com "a3".
Serão utilizados outros registos durante a elaboração do código mas serão registos de memória temporária.
Contudo, com alguma pesquisa, e auxílio das ferramentas fornecidas pelos professores percebemos que teríamos de criar um espaço na *heap* pois esta é uma memória dinâmica que cria espaço conforme o necessário, tal como os nós das nossas listas em "c".

A função ***"inicializelist"*** é a função responsável pela criação de espaço e que torna possível colocar informação nesse espaço de memória, como a criação de um novo nó de uma lista. Para isso damos a instrução ao simulador para reservar um espaço na memória dinâmica *heap*

A função ***"insert"*** está dividida em duas funções distintas. Isto deve-se à existência de um comando que insere texto no meio de linhas anteriormente inseridas.
Estas funções são a ***"insert_end"*** e a função ***"insertmidle"***.
A função ***"insert_end"*** é responsável por inserir um novo texto no fim do texto. Para isto instruímos o processador a ler uma string, e, com a

ajuda de uma função auxiliar *"count"* conta o tamanho da string inserida, introduzindo-a na *heap* utilizando o espaço necessário.

A função ***"insertmidle"*** é um pouco mais complexa, pois tem de procurar a posição a inserir o novo texto e criar o espaço necessário para a sua insersão.

Tal como a função ***"insert"*** a função ***"delete"*** também está dividida em duas funções distintas pelo mesmo motivo, a existência de um comando para apagar o último texto inserido ou texto já inserido anteriormente.

Uma destas funções é a função ***"delete_last"*** que é a responsável por eliminar a última linha de texto inserida, para isso começa no final do texto e vai percorrendo para trás até encontrar o caractere de final de linha "\n" e subtrai ao número de linhas o valor 1.

A função que corresponde à eliminação de uma linha do texto chama-se ***"delete_midle"***, e tal como a função ***"insertmidle"***, esta também tem de percorrer o texto até chegar à posição onde começa a linha a eliminar e depois calcula o tamanho da string e coloca o restante texto na posição da string eliminada e subtrai ao numero de linhas 1.

De seguida foram desenvolvidas as funções ***"print"*** e ***"printall"***.
A ***"print"*** carrega o endereço da *heap* para "a0" (inicio do texto) e depois percorre ate ao inicio da linha pretendida, depois dá ao processador a instrução para imprimir os caracteres em *"ASCII"*.

A função ***"printall"*** funciona de uma forma semelhante, mas começa no inicio do texto e vai percorrendo todo o texto e imprimindo linha a linha até chegar ao final

As funções desenvolvidas seguidamente foram as funções *quit*. Estas funções têm como objectivo sair do editor de texto, de duas maneiras diferentes pois uma da funções sai do editor mesmo que o ficheiro editado não esteja guardado e a outra apenas sai do editor se o ficheiro estiver guardado.

A primeira ***"quitQ"*** apenas contém a instrução para fechar o programa.

A segunda ***"quit"*** faz a comparação entre o nosso valor *"SAFE"* (a2) para

saber se o texto esta guardado caso o texto esteja guardado num ficheiro saltamos para a função ***"quitQ"***, caso contrário mostra uma mensagem com um ponto de interrogação e não fecha o programa.

Após todas estas funções concretizarem o desejado procedemos para o desenvolvimento do código para a leitura e analise dos comandos.

A função **comando** dá a instrução ao processador para ler uma *string*, após essa leitura vai analisar a string e determinar qual a instrução a realizar. Caso não reconheça a instrução mostra uma mensagem com um ponto de interrogação .

Instrução **"i"**: esta instrução pode ser chamada de duas maneiras. Quando o comando é apenas "i" insere texto no fim do documento. Quando é dado o comando i, mas com um número agregado **(5i)** é necessário utilizar a função *"atoi"* que detecta qual o número agregado à instrução e assim insere na linha desejada.

A instrução **"a"** tal como a instrução *"i"* também pode ser chamada de duas maneiras. No caso de ser apenas "a" vai inserir no fim do documento. Quando com um número agregado **(5a)** vai inserir na linha a seguir à linha indicada na chamada da instrução.

Instrução **"c"** é outra instrução semelhante às instruções anteriores, pois também se pode chamar de duas maneiras. Apenas **"c"**, eliminando a última linha e insere numa nova última linha, ou agregado a um número **"5c"** e assim a função vai eliminar a linha em questão e começar a inserir nessa mesma linha.

A instrução **"d"** é um pouco diferente das outra funções, pois pode ser chamada de três maneiras diferentes. Tal como as funções acima mencionadas quando é apenas um "d" a última linha é eliminada, agregado a um número (5d) a linha pretendida é eliminada, mas acrescenta-se uma nova maneira de chamar esta função para que opere num intervalo (2,4d ou 4,$d (da linha 2 á linha 4)) neste caso o intervalo em questão vai ser eliminado. Caso o segundo valor do intervalo seja um **"$"** vão ser eliminadas todas as linhas a partir da linha lida até ao fim do texto.

A instrução **"p"** é igual à função **"d"** só que em vez de eliminar, imprime a linha ou conjunto de linhas desejadas. No entanto, esta função tem ainda mais uma forma de ser chamada pois se a instrução for **"%p"** todo o texto deverá ser impresso.

Instrução **"q"** salta para a função **"quit"**

Instrução **"Q"** salta para a função **"quitQ"**.

A instrução **"e"** é chamada seguida do nome do ficheiro (e file_name) e deverá abrir e ler o respectivo ficheiro para a **heap**.

Instrução **"f"** deve ser chamada seguida do nome do ficheiro a criar (f file_name) e após a sua chamada é criado um ficheiro com o nome pretendido e deve ser impresso o nome do ficheiro na consola.

Por fim a instrução **"w"** é responsável por escrever texto num ficheiro seja este obtido pela instrução **"e"** ou criado pela instrução **"f"**. No caso de não existir um ficheiro aberto vai ser mostrada uma mensagem com um ponto de interrogação na consola.

4 Discussão de Resultados

Os resultados obtidos deste projecto foram os esperados contudo existiram alguns obstáculos difíceis de ultrapassar na realização deste código. Facilmente conseguimos obter todas as funções a concretizar o seu propósito individualmente, e foi quando testámos várias funções em simultâneo que começaram a aparecer erros, e por isso, foi necessário uma análise
detalhada da execução das nossas funções e severas remodelações ao nosso código inicial de forma a que todas as funções trabalhassem em harmonia sem se prejudicarem umas às outras.

O primeiro erro apareceu quando testámos a função **"delete_last"** e de seguida a função **"insert_end"**, que após apagar a última linha inserida, e ao inserir uma nova linha, a mesma era inserida na linha a seguir á linha eliminada. Isto ocorria porque, após a linha ser eliminada o endereço que continha a posição final permanecia inalterado.

O segundo erro aconteceu quando testámos a função **"printall"** após a função **"delete_last"**, quando todo o texto era impresso, também era impressa a linha que supostamente foi eliminada. Este erro foi fácil de resolver uma vez que após a execução da função **"delete_last"**, reparámos que o número de linhas não se alterava.

As funções que nos deram mais problemas foram funções relacionadas com ficheiros, **"open_file" (e) "write_file"(w) "define_name"(f)**, pois foram funções mais complicadas de perceber a sua implementação.

5 Conclusão

Com a realização deste trabalho compreendemos que a utilização de uma linguagem de programação de baixo nível, é muitas vezes, limitada nos seus recursos e ferramentas para que possamos desenvolver um código eficaz. No entanto com bastante prática é possível dominar a utilização destas linguagens.
A posição das instruções e das label's afectam bastante o código e é um dos pontos a ter bastante atenção.
É necessário também ter bastante atenção com os registos utilizados, pois uma má utilização dos registos, principalmente com registos de função, pode-se perder informação e danificar o código.
Para uma correcta utilização de ficheiros neste programa seria necessário uma abordagem mais detalhada sobre o próprio tema, pois foi onde sentimos as maiores dificuldades na implementação do código.
O projecto em sim foi interessante e ajudou bastante a aperfeiçoar os conhecimentos adquiridos ao longo deste semestre e ainda adquirir novos conhecimentos na utilização deste tipo de linguagem.

6 Bibliografia

Barão, Miguel and Salgueiro, Pedro. 2021. "Aulas de Arquitectura de Computadores I" In. Universidade de Évora.