Relatório Final - Laboratório de Montagem

Bruno Cardoso Maciel RA: 141046279

Rafael Stefanini Carreira RA: 141040726

24/11/2015

---

1. Introdução

Através de diversos programas desenvolvidos na linguagem Assembly, podemos reconhecer o poder desta linguagem no desenvolvimento de aplicações. Podemos observar também, com o decorrer das aulas, a desmistificação da linguagem, normalmente reconhecida por ser extremamente complicada.

Sendo assim, através dos trabalhos anteriores, podemos usar as técnicas e recursos aprendidos até o momento para desenvolver um editor de textos. Tais recursos nos auxiliam durante o desenvolvimento da aplicação, tornando o código mais organizado e legível, além de facilitar a programação do mesmo.

2. Objetivo

Desenvolver um editor de textos, onde o arquivo a ser editado é passado como parâmetro na execução do programa. O programa deverá abrir o arquivo informado e caso o arquivo não exista, o programa se encarregará de criá-lo.

Todo o conteúdo presente no arquivo será exibido na tela e em seguida, tudo que for digitado será anexado ao arquivo. O programa deve responder aos comandos inseridos pelo usuário, executando as operações de salvar, sair e contar palavras.

3. Metodologia

Primeiramente, foram criados macros paras as funções mais básicas do programa, tal como ler e escrever (syscall read e syscall write), porém com um diferencial, tais macros recebem também o *file descriptor* de onde devem realizar tais funções. Dessa maneira, podemos generalizar essas funções, podendo aplicá-las em arquivos quando necessário, sem a necessidade de alteração ou reescrita.

O primeiro passo na execução do programa é verificar a quantidade de elementos passados por parâmetro em sua execução, para isso, verificamos o primeiro valor na pilha, verificando o valor de eax após a execução da instrução “pop eax”. Se o valor for maior que 1, significa que foi passado pelo menos um parâmetro, e assim podemos continuar com a execução do programa. Caso contrário, o programa exibe uma mensagem de erro juntamente com uma instrução de uso do programa.

Após verificar a quantidade de parâmetros, o programa chama uma função para armazenar o nome passado em uma variável. Para isso, primeiramente salvamos o endereço de retorno em um registrador, visto que estamos em uma função. Em um registrador diferente do anterior, obtemos o nome passado por parâmetro através de duas instruções “pop”, visto que o primeiro valor é o nome do programa em si. Tendo o nome do arquivo em um registrador, basta copiarmos este valor *byte* por *byte,* até que seja encontrado o valor “\0”, sinalizando o fim do nome. Uma vez finalizado esse processo, empilhamos novamente o endereço de retorno, de modo com que a instrução “ret” retorne ao seu devido lugar.

Uma vez com o nome do arquivo a ser editado, utilizamos a *syscall open* para abri-lo. Após a interrupção do sistema, constará em “eax” o *file descriptor* necessário para manipular o arquivo. Caso esse valor seja menor que 0, significa que o arquivo não existe, e assim devemos criá-lo através da *syscall creat.* Finalmente, armazenamos o *file descriptor* resultante em “eax”, após uma das interrupções citadas, na variável “id”.

Antes de podermos editar o arquivo, devemos mostrar todo o seu conteúdo na tela, para isso, utilizamos a *syscall read* passando a identificação de arquivo obtida anteriormente. Dessa forma, tentamos ler 1000 bytes do arquivo para o buffer (a quantidade real de bytes lidos consta em “eax”), em seguida imprimimos na tela os bytes lidos do arquivo e verificamos se a

quantidade lida corresponde a 1000. Caso corresponda, então o processo de leitura e impressão é repetido, caso contrário, foram lidos menos bytes que o solicitado, representando assim um “EOF” (End of File), assim todo o conteúdo já foi lido e o processo termina.

A partir desse momento, o programa entra em um modo de espera, onde o usuário poderá entrar com o conteúdo que passará a fazer parte do arquivo. Para isso, todo caractere digitado pelo usuário será armazenado primeiramente em um buffer de tamanho 1000, onde o mesmo só será armazenado permanentemente no arquivo em duas situações: caso o buffer atinja sua capacidade máxima ou seja inserido explicitamente o comando de salvar.

Durante a digitação normal do conteúdo, sem utilizar comandos, cada caractere é lido individualmente e inserido no buffer. Utilizamos o registrador “ebx” para marcar a posição de inserção no buffer e para verificar quando o mesmo atinge sua capacidade máxima. Uma vez atingida a capacidade máxima, todo o conteúdo temporário é gravado no arquivo através da *syscall write* e o contador “ebx” é zerado, sinalizando que o buffer está novamente vazio.

O programa conta com a interpretação de três comados:

- **/slvr:** Salva o conteúdo digitado até o momento pelo usuário no arquivo.
- **/sair:** Exibe uma mensagem ao usuário, perguntando se deseja salvar todo o conteúdo digitado até o momento. O usuário deverá responder com “s” para salvar ou “n” para não salvar. Em seguida o programa encerra a sua execução.
- **/cpal:** Este comando serve para contar a quantidade de palavras salvas no arquivo até o momento, informando este valor ao usuário.

O processo de interpretação dos comandos foi desenvolvido da seguinte forma:

1. Compara-se cada caractere lido com “ / “, caso seja igual, siga para o passo 2.
2. São lidos 5 bytes do terminal e armazenados na variável “comando”
3. Compara-se o conteúdo da variável “comando” com os 3 comandos preestabelecidos, caso seja igual a um deles, ocorre um *jump* para o local adequado.
4. Caso não corresponda a nenhum comando, o conteúdo digitado é inserido no buffer e a leitura é retomada. Lembrando que ocorre uma verificação prévia do contador do buffer para verificar se o mesmo suporta a inserção desses caracteres. Caso não suporte, todo o conteúdo temporário é armazenado no arquivo durante o processo.

O comando salvar, como já explicado anteriormente, escreve o conteúdo do buffer no arquivo e zera seu contador.

O comando sair serve para finalizar a edição do arquivo, e após executado, o usuário é questionado se o conteúdo temporário deve ser salvo no arquivo. Somente serão aceitos duas respostas para a pergunta, “s” (sim) ou “n” (não), e caso o usuário digite algo diferente, a pergunta é refeita mostrando as opções disponíveis. Se o usuário optar por salvar o conteúdo, o procedimento realizado é o mesmo do comando salvar, e após uma das opções, o arquivo é fechado pelas *syscall close* e o programa é finalizado.

O comando contar palavras é relativamente mais extenso do que os anteriores, isso se deve à utilização de diversas estruturas, tais como as necessárias para converter números para caracteres e imprimi-los na tela em ordem inversa. Entretanto, o processo em si não é complicado, mas exige atenção adicional devido a utilização de diversas variáveis e contadores ao mesmo tempo. Abaixo segue um esquema de como foi implementada esta função:

1. Primeiramente salvamos o contador do buffer, empilhando-o;
2. Através da *syscall lseek,* colocamos o cursor no inicio do arquivo. Tal ação é necessária para lermos novamente o conteúdo do arquivo, desde o começo;
3. Lemos 100 bytes do arquivo e armazenamos em um buffer próprio para o contador;
4. Contamos a quantidade de palavras presentes neste intervalo;
5. Continuamos lendo e contando as palavras enquanto a quantidade de bytes lidos corresponder a quantidade solicitada na *syscall read;*
6. Após lido e contabilizado todo o arquivo, a quantidade de palavras é convertida para um valor legível e impressa adequadamente na tela.

Após realizados os comandos de salvar e contar palavras, o programa continua em execução, sendo possível continuar a edição do arquivo, lembrando que o texto referente aos comandos não são armazenados no arquivo.

4. Resultados

Através do uso de diversas técnicas aprendidas ao longo do curso, obtemos um programa compreensível e legível, sendo de fácil operação, até mesmo para um usuário leigo, visto as mensagem e instruções presentes durante sua utilização.

Além de podermos editar um arquivo de texto, o programa reconhece os comandos definidos e os executa corretamente. Dessa forma, a aplicação cumpre com o seu objetivo.

5. Discussão

Uma característica deste programa é que o mesmo permanece em execução, respondendo a comandos inseridos pelo usuário. Para revolver essa questão, foi necessário o uso de técnicas ainda não utilizadas, constituindo assim, um desafio a mais. Outra técnica ainda não utilizada previamente, mas muito observada em outros programas, é a passagem de argumentos. Tal recurso é amplamente utilizado em diversos programas e torna mais prático a execução do programa desenvolvido.

6. Conclusão

Diante do objetivo proposto e de todo conteúdo assimilado ao longo do curso, o projeto final se mostra mais complexo do que os anteriores, sem levar em consideração seu código demasiadamente extenso. Dessa forma, justifica-se o aprendizado prévio e uso de diversos recursos úteis no desenvolvimento desta última aplicação, tais como os macros, chamadas de funções, gerenciamento de arquivos, além de outros.

Os recursos aprendidos ofereceram uma gama de possibilidades para a realização do programa, dessa forma obtemos um código-fonte estruturado e organizado, sem comprometer o desempenho da aplicação. Por fim os resultados obtidos foram satisfatórios e concisos.