5º Relatório de Laboratório de Montagem

Bruno Cardoso Maciel RA: 141046279

Rafael Stefanini Carreira RA: 141040726

16/11/2015

---

1. Introdução

Diversas linguagens de programação oferecem recursos de gerenciamento de arquivos externos. A linguagem Assembly também possui tal recurso que, apesar de ser mais complexo de se utilizar, oferece uma performance superior se comparada com as demais linguagens.

Diante isto, temos as *"system_calls"*, ou as chamadas de sistema, que permite realizar determinadas operações com os arquivos. Cada *syscall* possui seu próprio conjunto de parâmetros, que através dos registradores, usamos para identificar as operações e os arquivos utilizados. O registrador utilizado para definir a operação desejada é o eax. A seguir temos algumas *syscalls* úteis para se trabalhar com arquivos:

System_call	**eax**	**Função**
sys_read	3	Leitura do arquivo
sys_write	4	Escrever no arquivo
sys_open	5	Abrir o arquivo
sys_close	6	Fechar o arquivo
sys_create	8	Criar um novo arquivo

2. Objetivo

Desenvolver uma aplicação que copia arquivos de texto. O programa solicita ao usuário o nome do arquivo a ser copiado. Em seguida lê o conteúdo do arquivo e armazena em uma variável. Finalmente a aplicação cria um novo arquivo com o mesmo nome do inserido pelo usuário acrescido do apêndice "*(copia).txt*" e escreve o conteúdo do arquivo na tela.

3. Metodologia

Gerenciar arquivos em Assembly requer uma quantidade a mais de variáveis do que estamos acostumados a trabalhar, uma vez que precisamos armazenar o identificador de casa arquivo utilizado. Tal identificador será utilizado durante as demais operações em arquivos e deve ser passado como parâmetro durante as chamadas de sistema correspondente.

Foram criadas função para cada funcionalidade do programa, sendo assim, temos a seguinte execução:

- Por meio de uma chamada de função, imprime a mensagem introdutória da aplicação.

- Imprime uma mensagem solicitando o nome do arquivo de texto a ser copiado.
    - Através da *syscall write*, imprime a mensagem “Digite o nome do arquivo: “.

- É lido o nome do arquivo e armazenado na variável “nome_arq” anteriormente declarada.
    - Para armazenarmos o nome do arquivo, utilizamos a *system call read.*

- Função para inserir 0 ao fim da string.
    - Seguinte o enunciado proposto, a aplicação insere “\0” ao fim da *string*. Isso se faz necessário para criar e ler arquivos utilizando o nome recebido.
    - Para tal processo, usamos o registrador ebx como contador para um laço.
    - Realizamos uma comparação da *string* na posição ebx com o *byte* 10 (‘\n’), se for igual, então significa que estamos no fim da *string* e pulamos para a inserção do “\0” ao fim da *string*.
    - Caso contrário, o contador ebx é incrementado e o processo de comparação refeito.

- Função para ler o conteúdo do arquivo.
    - Para lermos o conteúdo de um arquivo, devemos necessariamente abri-lo por meio da *“sys_open”.* Informamos o valor de eax referente a *system_call* desejada e a variável que contém o nome do arquivo. Depois devemos escolher o modo de acesso que será imposto ao arquivo (leitura, escrita, ou os dois), e também informar em octal qual as permissões de uso do arquivo(apenas criador, grupo, ou outros podem acessar o arquivo).
    - Após termos aberto o arquivo, o valor existente em eax é o identificador do arquivo, guardamos esse valor na variável “*id_in*” (id de entrada). Feito isso,

podemos ler o conteúdo do arquivo por meio da *“sys_read”,* já explicado anteriormente, a única diferença é que agora usaremos o id armazenado para informar qual a variável queremos trabalhar e a variável “*info*” com seu respectivo tamanho para armazenarmos o conteúdo do arquivo.

    - Posteriormente, assim como na abertura do arquivo, agora na leitura, eax contém o tamanho do conteúdo de entrada, e armazenamos tal valor na variável “tamanho_info”
    - Encerrando o processo de leitura do contéudo, devemos fechar o arquivo por meio da “*sys_close*”, informamos o valor a eax referente ao fechamento do arquivo, e identificamos por meio do id o arquivo que queremos fechar.

- Função para copiar o nome do arquivo e inserir o apêndice.
    - Para tal processo, usamos os registradores ebx e ecx como contadores. Feito isso iniciamos uma label onde é realizada uma comparação entre o nome do arquivo na posição ebx com o byte ‘.’ (ponto). Se não for igual, ele guarda em dl o caractere do nome na posição ebx e incrementa o mesmo, depois disso o processo de comparação é refeito.
    - Caso a comparação seja verdadeira, o processo salta para outra *label* para comparar o contador ecx com o tamanho do apêndice (‘copia.txt’). Se for igual, ele guarda o valor de ebx na variável tamanho_nome, senão o apêndice é movido para edx e posteriormente colocado no fim do nome do arquivo de saída.
    - Após o processo, os contadores são incrementados e a comparação é refeita.

- Função para criar o arquivo de saída.
    - Identificamos qual “*system_call*” usaremos atribuindo o valor 8 (*sys_create*) à eax. E assim criamos o arquivo de saída com suas devidas permissões.
    - Após a criação, eax possui o valor referente ao identificador do arquivo de saída, portanto devemos armazenar tal valor em *id_out.*
    - Feito isso, utilizamos a *sys_write* informando o valor 4 a eax, para podermos escrever o conteúdo do arquivo inicial no de saída que acabamos de criar. Por meio do id anteriormente armazenado, identificamos qual arquivo queremos escrever, e pelas variáveis “*info”,* que contém o conteúdo do arquivo digitado pelo usuário, e *“tamanho_info”,* que contém o tamanho do conteúdo, escrevemos no arquivo de saída.
    - Encerrando o processo de escrita do conteúdo, devemos fechar o arquivo por meio da “*sys_close*”, já explicada anteriormente. Com o processo tendo sido

concluído corretamente, o programa exibe uma mensagem de sucesso ao usuário.

- Finaliza o programa.
    - Imprime uma linha indicando o fim da interação do usuário com o programa.
    - Finaliza a execução do programa através da *syscall exit.*

4. Resultados

Através do uso de diversas funções, obtemos um programa compreensível e legível. O programa, após executado, solicita o nome do arquivo a ser copiado e cria um arquivo idêntico com o apêndice “(copia).txt”. Dessa forma, o programa cumpre com o seu objetivo através de uma interface simples e sem ambiguidade.

5. Discussão

O uso da linguagem Assembly na manipulação de arquivos, apesar de mais complexa, cumpre sem problemas o objetivo do programa. Devemos apenas lembrar que para usar um arquivo preexistente devemos por meio das system_calls, abrir e fechar o arquivo para evitar erros futuros. Ao passo que quando se criar um arquivo pela aplicação só existe a necessidade de fechá-lo após o uso.

6. Conclusão

Como podemos observar, trabalhar com arquivos em Assembly não é uma tarefa trivial. Isso se deve à necessidade de diversas chamadas de sistema para as diferentes funções, além da necessidade de se informar o código de identificação em cada uma delas.

Essas operações com arquivos, apesar de mais complexas em relação as linguagens de mais alto nível, apresentam desempenho superior. Isso justifica o uso dessa linguagem em diversos dispositivos integrados e em algumas funções de mais baixo nível de sistemas operacionais.