Lista de Exercícios para Programação de Software Básico

Fernando Antonio M. Schettini[1,2]

[1]Faculdade e Centro Universitário SENAI CIMATEC
41650-010 – Salvador – BA – Brasil

[2]Universidade Federal da Bahia
40170-115 – Salvador – BA – Brasil

fernandoschettini@Outlook.com

Abstract. *This exercise list was designed to provide essential practice for developing skills in basic software programming. Each exercise covers different aspects of the software development process, such as programming logic, control structures, and algorithms. It was created as part of the assessment for the Basic Software Development course taught at UFBA in the first semester of 2023.*

Resumo. Esta lista de exercícios foi elaborada com o objetivo de fornecer práticas essenciais para o desenvolvimento de habilidades em programação de software básico. Cada exercício aborda diferentes aspectos do processo de desenvolvimento de software, como lógica de programação, estruturas de controle e algoritmos. Ela foi feita como parte da avaliação da disciplina de Desenvolvimento de Software Básico lecionada na UFBA no primeiro semestre de 2023.

1. Registradores[1] - 05/06/2023

Exercícios e suas resoluções da atividade disponibilizada no dia 05/06/2023.

1.1. Liste os registradores de uso geral da arquitetura Intel 64 bits e a descrição de cada.

Tabela 1. Registradores de Uso Geral Endereçáveis na arquitetura Intel 64 Bits[2]

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8B - R15B
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

De acordo com o tamanho de bits e a presença de REX, existem os mesmos registradores, apenas em classes diferentes. Os registradores utilizados no modo de 64 bits contém quatro palavras, sendo eles[2]:

• RAX - Acumulador para dados de operandos e resultados.

• RBX - Ponteiro para dados no segmento DS.

• RCX - Contador para operações de string e loop.

• RDX - Ponteiro de E/S.

• RSI - Ponteiro para dados no segmento apontado pelo registro DS; ponteiro de origem para operações de string.

• RDI - Ponteiro para dados (ou destino) no segmento apontado pelo registro ES; ponteiro de destino para operações de string.

• RSP - Ponteiro de pilha (no segmento SS).

• RBP - Ponteiro para dados na pilha (no segmento SS).

1.2. Quais são os registradores especiais que têm 128 bits de tamanho? descreva a tecnologia SSE.

São eles[2]:

- XMM0 - XMM15: Provenientes da tecnologia SSE.
- MXCSR: Registrador de controle e estado do SSE. Ele controla e monitora o comportamento das instruções SSE, como modos de arredondamento, tratamento de exceções e flags de status.

"A tecnologia Intel® Streaming SIMD Extensions (Intel® SSE), ela melhora o desempenho dos processadores IA-32 em gráficos avançados 2D e 3D, vídeo em movimento, processamento de imagens, reconhecimento de fala, síntese de áudio, telefonia e videoconferência. Ela também expandiu o modelo de execução SIMD adicionando recursos para lidar com valores de ponto flutuante de precisão simples empacotados e escalares contidos em registradores de 128 bits."[2] Desta forma, SSE adiciona as seguintes funcionalidades[2]:

- Oito registradores de dados de 128 bits (chamados de registradores XMM) em modos não 64 bits; 16 registradores XMM estão disponíveis em modo 64 bits.
- O registrador MXCSR de 32 bits, que fornece bits de controle e status para operações realizadas nos registradores XMM.
- O tipo de dados de ponto flutuante de precisão simples empacotado de 128 bits (quatro valores de ponto flutuante de precisão simples IEEE empacotados em uma dupla quadword).
- Instruções que realizam operações SIMD em valores de ponto flutuante de precisão simples e que estendem operações SIMD que podem ser realizadas em números inteiros:
- Instruções de ponto flutuante de precisão simples empacotadas e escalares de 128 bits que operam em dados localizados em registradores MMX.
- Instruções inteiras SIMD de 64 bits que suportam operações adicionais em operandos inteiros empacotados localizados em registradores MMX.
- Instruções que salvam e restauram o estado do registrador MXCSR.

- Instruções que suportam pré-busca explícita de dados, controle da cacheabilidade de dados e controle da ordem de operações de armazenamento.
- Extensões para a instrução CPUID.

1.3. Qual uso do registrador *rip* e qual sua função?

"Este registrador armazena o deslocamento de 64 bits da próxima instrução a ser executada. No modo de 64 bits, também é suportada uma técnica chamada de endereçamento relativo ao RIP. Usando essa técnica, o endereço efetivo é determinado adicionando um deslocamento ao RIP da próxima instrução."[2]

É usado para guiar o processador na ordem de execução de instrução no ciclo de execução de uma instrução.

1.4. *rflags*, qual função desse registrador? O que significam as flags CF, AF, ZF, OF e SF?

"O registrador EFLAGS de 32 bits contém um grupo de flags de status, uma flag de controle e um grupo de flags do sistema."[2]

CF, AF, ZF, OF e SFs são flags de status possíveis dentro do registrador, elas indicam resultados de instruções aritméticas como ADD, SUB, MUL e DIV.

- "CF (bit 0) Flag de Carry — Define-se se uma operação aritmética gera um carry (vai-um) ou empréstimo a partir do bit mais significativo do resultado; caso contrário, é zerado. Essa flag indica uma condição de estouro para aritmética de números não assinados. Também é usada em aritmética de precisão múltipla.
- PF (bit 2) Flag de Paridade — Define-se se o byte menos significativo do resultado contém um número par de bits iguais a 1; caso contrário, é zerado.
- AF (bit 4) Flag de Carry Auxiliar — Define-se se uma operação aritmética gera um carry (vai-um) ou empréstimo do bit 3 do resultado; caso contrário, é zerado. Essa flag é usada em aritmética de código decimal binário (BCD).
- ZF (bit 6) Flag de Zero — Define-se se o resultado é zero; caso contrário, é zerado.
- SF (bit 7) Flag de Sinal — Define-se igual ao bit mais significativo do resultado, que é o bit de sinal de um inteiro assinado. (0 indica um valor positivo e 1 indica um valor negativo.)
- OF (bit 11) Flag de Estouro — Define-se se o resultado inteiro é um número positivo muito grande ou um número negativo muito pequeno (excluindo o bit de sinal) para caber no operando de destino; caso contrário, é zerado. Essa flag indica uma condição de estouro para aritmética de números assinados (complemento de dois)."[2]

1.5. Qual registrador armazena o endereço que está no topo da pilha?

É o RSP[2], ele aponta para o endereço de memória do topo da pilha, desta forma, é usado para controlar a alocação e desalocação de memória na pilha durante a execução do programa, sendo atualizado à medida que valores são empilhados ou desempilhados.

1.6. Segundo código abaixo qual valor estará em *rsp*? por que isso ocorre?

```
1  section .data
2      msg db 5,6,7
3
4  section .text
5      global _start
6
7  _start:
8
9      push 5
10     push 6
11     push 7
12
13
14     mov rdi, [rsp]
15     mov rax, 60
16     syscall
17
```

Figura 1. Código da questão 1.6 [1]

O valor que estará no registrador *rps* no final do código será o último valor inserido na pilha. Como o último valor inserido foi 7, o valor de *rps* também será 7. Isso acontece porque a pilha é uma estrutura LIFO, onde o último a ser inserido será o primeiro a sair, desta forma, o rps vai apontar para o último valor inserido na pilha.

1.7. Reelabore o código acima inserindo o valor 5 no registrador *rax*.

```
section .data
     msg db 5,6,7
section .text
     global _start
_start:
     push 5
     push 6
     plush 7
     mov rdi, [rsp]
     mov rax, 5
     syscall
```

2. Mapa Mental[3][4] - 23/06/2023

Disponível neste link, conforme representado pela figura 2.

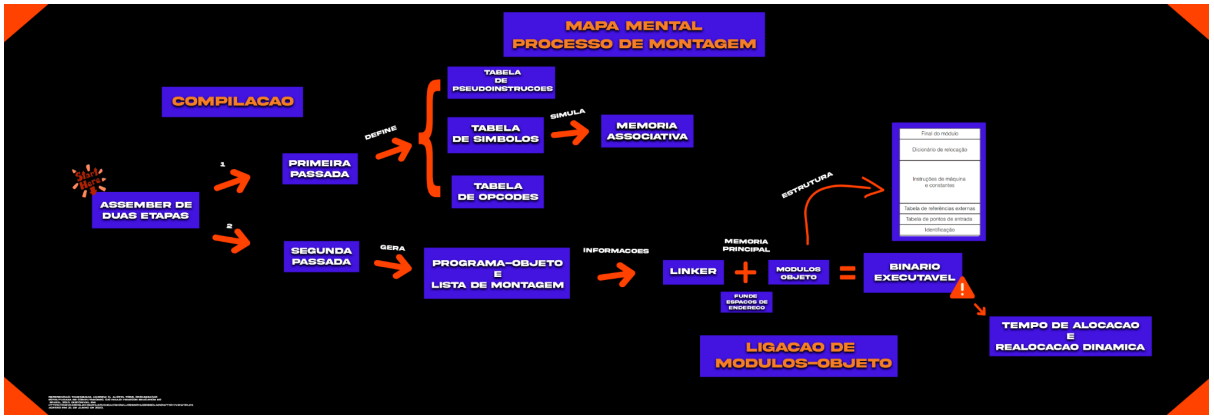

Figura 2. Mapa Mental - Processo de Montagem.[4][5] Fonte própria.

3. Processos de ligação dinâmica[4][6][7][8] - 23/06/2023

Ligação dinâmica é o processo no qual as bibliotecas de código (ou funções) utilizadas por um programa não são incorporadas diretamente nele, mas são carregadas e veiculadas em tempo de execução, como representadas pela figura 3. O objetivo é reutilizar bibliotecas de código compartilhado entre vários programas, aumentando a economia de espaço em disco e manutenção de códigos que usam essas bibliotecas.

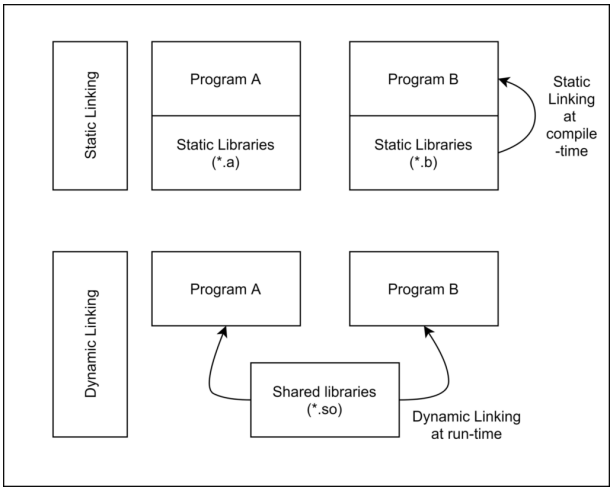

Figura 3. Representação de ligação dinâmica. Disponível em: https://www.baeldung.com/cs/dynamic-linking-vs-dynamic-loading Acesso em 23 de Junho de 2023.

Tanto os sistemas operacionais Windows quanto UNIX a ligação dinâmica possui um princípio similar em ambos os sistemas, mas sua implementação e uso variam significativamente. Para construção desse texto, foram utilizadas as **referências marcadas por [8], [6], [5] e [7] majoritariamente.**

3.1. Windows[6]:

No Windows[6], é utilizada a tecnologia "Dynamic Link Libraries" (DLLs) para realizar a ligação dinâmica. As DLLs são arquivos externos .dll, .drv (para biblioteca de drivers) e .fon (para bibliotecas de fontes tipográficas)[5], que contêm dados e código compartilhados, acessíveis por vários programas ao mesmo tempo. Elas são carregadas conforme necessário durante a execução do programa, economizando espaço em disco ao permitir que diversos aplicativos utilizem uma única biblioteca, em vez de terem cópias separadas. Além disso, o Windows realiza a ligação dinâmica de forma implícita, o que significa que, quando um programa é executado, o sistema operacional localiza e carrega as DLLs necessárias.

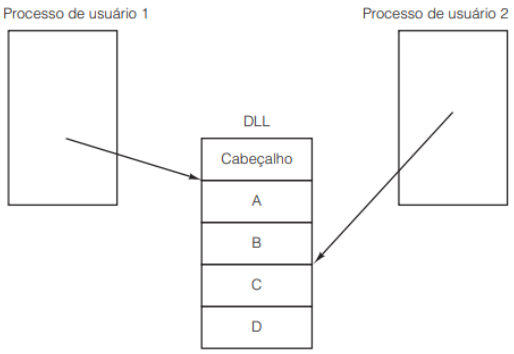

Figura 4. Dois Processos usando a mesma biblioteca .dll no Windows.[5]

Uma DLL[6] pode definir dois tipos de funções: exportadas e internas. As funções exportadas devem ser chamadas por outros módulos, bem como de dentro da DLL em que são definidas. Normalmente, as funções internas são destinadas a serem chamadas somente de dentro da DLL em que são definidas. Embora uma DLL possa exportar dados, seus dados geralmente são usados apenas por suas funções. No entanto, não há nada que impeça que outro módulo leia ou grave esse endereço.

As DLLs[6] fornecem uma maneira de modularizar aplicativos para que sua funcionalidade possa ser utilizada e reutilizada com mais facilidade. As DLLs também ajudam a reduzir a sobrecarga de memória quando vários aplicativos usam a mesma funcionalidade ao mesmo tempo, pois embora cada aplicativo receba sua própria cópia dos dados de DLL, os aplicativos compartilham o código DLL.

A API[6] (interface de programação de aplicativos) do Windows é implementada como um conjunto de DLLs, portanto, qualquer processo que usa a API do Windows usa vinculação dinâmica.

3.2. UNIX[8][5]:

No UNIX[8], a ligação dinâmica é chamada de "Shared Objects" ou "Shared Libraries" (SOs). Assim como as DLLs do Windows, os SOs são arquivos que contêm código e dados compartilhados. No entanto, no UNIX, a ligação dinâmica é geralmente feita de forma

implícita. Os programas devem especificar quais SOs desejam utilizar, e o sistema operacional carrega essas bibliotecas durante a execução.

Desta forma, uma biblioteca compartilhada consiste em duas partes: uma biblioteca hospedeira, ligada estaticamente com o arquivo executável, e uma biblioteca-alvo[5]. Com esse método, os desenvolvedores têm mais controle, pois podem escolher quais versões das bibliotecas desejam utilizar.

Para ser mais específico, nos sistemas baseados no GNU glibc, incluindo todos os sistemas Linux, ao iniciar um executável binário ELF, o carregador de programas é automaticamente carregado e executado. Nos sistemas Linux, esse carregador é chamado /lib/ld-linux.so.X (onde X é um número de versão). Esse carregador, por sua vez, encontra e carrega todas as outras bibliotecas compartilhadas usadas pelo programa.

A lista de diretórios a serem pesquisados é armazenada no arquivo /etc/ld.so.conf. Muitas distribuições derivadas do Red Hat normalmente não incluem /usr/local/lib no arquivo /etc/ld.so.conf. Considero isso um bug, e adicionar /usr/local/lib a /etc/ld.so.conf é uma correção comum necessária para executar muitos programas em sistemas derivados do Red Hat.

Se você deseja apenas substituir algumas funções em uma biblioteca, mas manter o restante da biblioteca, pode inserir os nomes das bibliotecas de substituição (.o filé) em /etc/ld.so.preload; essas bibliotecas "pré-carregadas" terão prioridade sobre o conjunto padrão. Esse arquivo de pré-carregamento é geralmente usado para patches de emergência; normalmente, uma distribuição não incluirá esse arquivo quando entregue.

Pesquisar todos esses diretórios durante a inicialização do programa seria extremamente ineficiente, então é usado um sistema de armazenamento em cache. O programa ldconfig(8), por padrão, lê o arquivo /etc/ld.so.conf, configura os links simbólicos apropriados nos diretórios de link dinâmico (para seguir as convenções padrão) e, em seguida, escreve um cache em /etc/ld.so.cache que é usado por outros programas. Isso acelera bastante o acesso às bibliotecas. A implicação é que ldconfig deve ser executado sempre que uma DLL é adicionada, removida ou quando o conjunto de diretórios de DLL é alterado; executar ldconfig é frequentemente um dos passos executados pelos gerenciadores de pacotes ao instalar uma biblioteca. Na inicialização, portanto, o carregador dinâmico realmente usa o arquivo /etc/ld.so.cache e, em seguida, carrega as bibliotecas de que precisa.

A propósito, o FreeBSD usa nomes de arquivos ligeiramente diferentes para esse cache. No FreeBSD, o cache ELF é /var/run/ld-elf.so.hints e o cache a.out é /var/run/ld.so.hints. Eles ainda são atualizados pelo ldconfig(8), portanto, essa diferença de localização deve ser relevante apenas em algumas situações exóticas.

3.3. Comparação:

A ligação dinâmica possui vantagens e desvantagens em ambos os sistemas operacionais. O método implícito simplifica o processo de desenvolvimento no Windows, pois elimina a necessidade de os programadores definerem explicitamente as bibliotecas que desejam utilizar. No entanto, podem ocorrer problemas de compatibilidade se várias versões de DLLs estiverem instaladas no sistema[6]. Por outro lado, o método explícito utilizado pelo UNIX oferece mais liberdade e controle, permitindo que os desenvolvedores utilizem versões específicas das bibliotecas. No entanto, isso também pode tornar o desenvolvimento e a implantação mais complexos.

Em conclusão, a ligação dinâmica é utilizada por aplicativos no Windows e no UNIX para compartilhar bibliotecas de código. Enquanto o UNIX utiliza um método implícito em que os aplicativos declaram quais SOs desejam utilizar, o Windows adota uma abordagem implícita em que o sistema operacional carrega automaticamente as DLLs. A escolha entre cada técnica depende dos requisitos e preferências dos desenvolvedores, pois cada uma possui suas próprias vantagens e desvantagens.

References

[1] MOURA, Mirlei. (2023). MATA49 Atividade 3 - REGISTRADORES. Disponível em: https://docs.google.com/document/d/1zJ4unI8PraZyh48VHyqt67udV8z2v9gK/edit. Acesso em: 5 de junho de 2023.

[2] Intel Corporation. (2023). Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 1: Basic Architecture. Santa Clara, CA: Intel Corporation. Disponível em: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html. Acesso em: 5 de junho de 2023.

[3]LUCIDCHART. O que é mapa mental e como fazer. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-mapa-mental-e-como-fazer. Acesso em: 20 jun. 2023.

[4] MOURA, Mirlei. (2023). MATA49 Atividade 3 - Mapa Mental.

[5] TANENBAUM, Andrew S.; AUSTIN, Todd. Organização Estruturada de Computadores. São Paulo: Pearson Education do Brasil, 2013. Disponível em: https://drive.google.com/file/d/1HgAxSHGwLu3ZBNOYAg9o8bclmnOwT78y/view?pli=1 Acesso em: 21 de junho de 2023.

[6] Microsoft Developer Network (MSDN). Dynamic-Link Libraries (DLLs). Disponível em: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx. Acesso em: 23 de junho de 2023.

[7] IBM Knowledge Center. Introduction to dynamic linking. Disponível em: https://www.ibm.com/docs/en/aix/7.2?topic=programming-introduction-dynamic-linking. Acesso em: 23 de junho de 2023.

[8] The Linux Documentation Project. Shared Libraries. Disponível em: https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html. Acesso em: 23 de junho de 2023.

[9]anthonywritescode. what is a .so / .dll / shared object? (intermediate - advanced) anthony explains #399. YouTube, 23 de junho de 2023. Disponível em: https://www.youtube.com/watch?v=PDkKz3zQVls. Acesso em: 23 de junho de 2023.