Instituto de Informática - UFRGS

Sistemas Operacionais

Gerência de Memória

Introdução

Aula 11

Introdução

- Multiprogramação implica em manter vários processos em memória
- Necessidades:
  - Alocar memória eficiente para permitir o máximo possível de processos
  - Proteger o espaço de memória de um processo contra acessos indevidos de outros processos

Gerência de memória

Considerações gerais

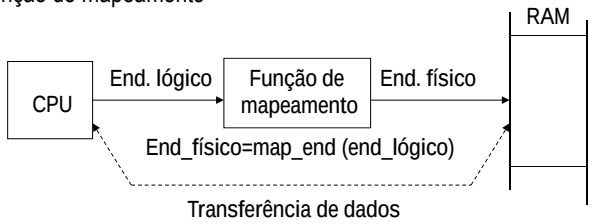
- Um sistema de memória possui pelo menos dois níveis:
  - Memória principal: RAM (acessada pela CPU)
  - Memória secundária: discos
- Programa para ser executado tem que estar em RAM
  - Transferência de memória secundária (disco) a memória primária (RAM)
- Qualquer sistema operacional tem gerência de memória
  - Complexidade depende do sistema (monotarefa ou multitarefa)
- Algoritmos de gerência de memória dependem de facilidades disponíveis pelo hardware da máquina

Memória lógica e memória física

- Memória lógica
  - É aquela que o processo “enxerga”
  - Endereços lógicos são aqueles manipulados por um processo
- Memória física
  - Implementada pelos circuitos integrados de memória
  - Endereços físicos são aqueles que correspondem a uma posição real de memória

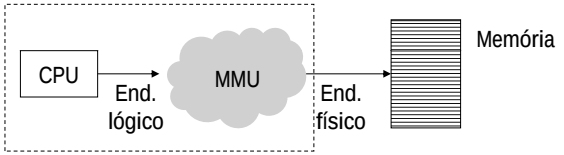
Endereço lógico versus endereço físico

- Espaço lógico de um processo é diferente do espaço físico
  - Endereço lógico: gerado pela CPU
  - Endereço físico: endereços enviados para a memória RAM
- Programas de usuário "enxergam" apenas endereços lógicos
- Necessidade de mapear endereços lógicos em endereços físicos
  - Função de mapeamento



Função de mapeamento

- Pode ser implementada em:
  - Software: ferramentas de desenvolvimento (compilador, ligador, etc)
  - Hardware:



  - Combinação de software e hardware
- Complexidade variável de acordo com a funcionalidade oferecida
  - Mecanismos de suporte para proteção, compilação, ligação, carga de programas, geração de códigos absolutos ou relocáveis, etc

Execução de programas

- Um programa deve ser transformado em um processo para poder ser executado
  - Alocação de um descritor de processo
  - Alocação de áreas de memória para código, dados e pilha
- Parte dessa transformação é feita através de uma série de passos com a ajuda do próprio programador
  - Compilação, diretivas de compilação e/ou montagem, ligação, etc...
- Amarração ou associação de endereços (*binding*)
  - Estático versus dinâmico

Amarração (*binding*)

- Associação de endereços de memória a dados e instruções
- Dois tipos:
  - Amarração estática: feita antes da execução (compilação+ligação+carga)
  - Amarração dinâmica: em tempo de execução (pilha + área de heap)

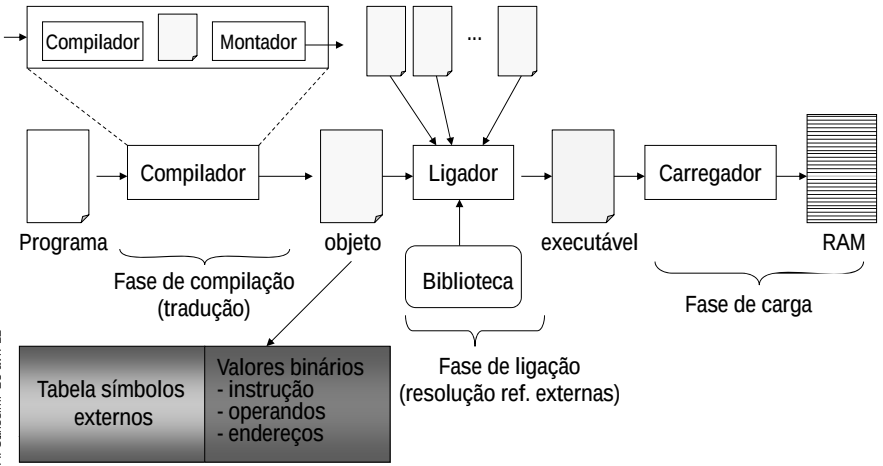
Ferramentas de desenvolvimento

- Compilador (montador)
  - Conversão de código para instruções dependentes de máquina
- Ligador
  - Existe devido a demanda de programação modular e aproveitamento de código comum (e.g., bibliotecas)
  - Geração de um espaço único de endereçamento a partir de *n* módulos
  - Resolução de referências externas
- Carregador
  - Transferência do executável para a memória
    - Envolve mapear endereços lógicos em endereços físicos

Preparação de um programa para execução

- Compilação
  - Programa objeto
  - Endereços de tradução
- Ligação
  - Programa executável (binário)
  - Ligação com módulos e bibliotecas
  - Endereços de ligação
    - Necessidade de ajustar os endereços (relocação)
- Carregador
  - Tradução do endereço de ligação para endereço de carga
    - Carregador absoluto: não faz o ajuste
    - Carregador dinâmico: ajusta endereço de ligação para carga

Preparação de programas para execução



Considerações com uso de memória:ligador

- Estática
  - Não há referências externas não resolvidas no programa binário
    - Vantagem: simplicidade
    - Desvantagem: desperdício de cópias
- Dinâmica
  - Há referências não resolvidas no programa binário
    - Resolução durante a execução (via stubs)
    - Vantagens: compartilhamento de código, atualização (ex. DLLs) e eficiência (módulos não usados não são carregados)
    - Desvantagens: complexidade e desempenho

Considerações com uso de memória: relocação

- Estática
  - Antes da execução iniciar
- Dinâmica
  - Em tempo de execução
- Problema:
  - Como se faz quando existe o procedimento de swapping?
    - Um processo sai da memória (swap-out) e, posteriormente, pode voltar (swap-in) para uma outra posição de memória.

Amarração de endereço lógico a físico (*binding*)

- Pode acontecer em diferentes etapas:
  - Desenvolvimento do programa
  - Compilação (montagem) — Amarração estática
  - Ligação
  - Carga
  - Execução — Amarração dinâmica
- Antes da amarração os endereços são manipulados através de referências simbólicas
  - Transformados em valores numéricos assumindo zero como endereço inicial
- Código absoluto e código realocável

Código absoluto

- Não identifica quais endereços devem ser corrigidos nas diferentes etapas de desenvolvimento de um programa
  - Não mantém nenhum tipo de tabelas de endereços a serem corrigidos
- Código objeto absoluto é ligado sem nenhuma correção com outros módulos
- Código executável absoluto é carregado para a memória sem nenhum tipo de tratamento
  - Basicamente é uma cópia

Relocação

- Sempre que no tratamento de um arquivo objeto ou executável for necessário movê-lo em relação a origem (zero) é necessário "corrigir" referências a endereços
- Habilidade de "correção" é a relocação
  - Código que permite a "correção": código relocável
  - Código que não permite a "correção": código absoluto
- Quem gera o tipo de código?
  - As ferramentas de desenvolvimento

Código relocável

- Necessidade de identificar quais endereços devem ser corrigidos
  - Mapeamento das posições a serem corrigidas é mantida através de tabelas
- Código objeto relocável informa posições a serem corrigidas na etapa de ligação
- Código executável relocável mantém as informações de quais posições devem ser corrigidas no momento da carga

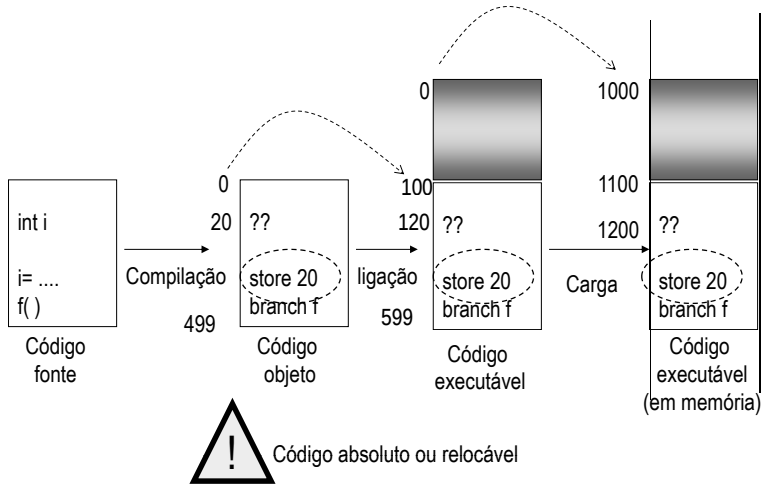
Amarração estática

- Definição de endereços físicos antes do programa ser executado
  - Pode ocorrer em diferentes estágios do ciclo de desenvolvimento
  - Uma vez definido o endereço ele não pode mais ser reajustado
    - Código absoluto
- Amarração em tempo de escrita/compilação do programa
  - Capacidade fornecida em alguns compiladores e montadores
  - Comum em sistemas embarcados e/ou tempo real
  - Restritivo: programa só executa nos endereços físicos para os quais foi gerado
  - Dificulta o desenvolvimento de um programa em vários módulos

Amarração estática (cont.)

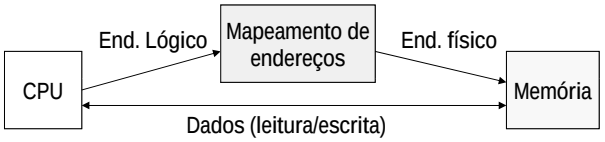
- Amarração em tempo de ligação
  - Facilita a programação de forma modular
  - Códigos objetos devem ser relocáveis
    - Indicam posições a serem ajustadas
  - Código executável é absoluto
    - Deve ser carregado na posição definida pelo ligador
      - Ligador carregador
- Amarração em tempo de carga
  - Ajustes de endereços no momento em que o processo é carregado na memória
  - Código executável é relocável
  - Necessidade de determinar quais posições devem ser ajustadas
    - Valores absolutos versus valores relocáveis

Exemplo de amarração estática

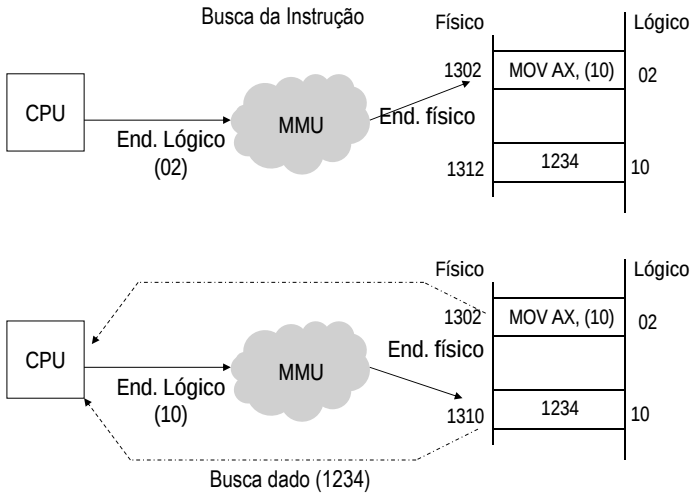


Amarração dinâmica

- relocação é feita no momento do acesso a referência de memória (tempo de execução) pela função de mapeamento
  - Feito por hardware (MMU) por questões de eficiência
- Usa carregador absoluto
  - Nenhum ajuste de endereços é feito no momento da carga

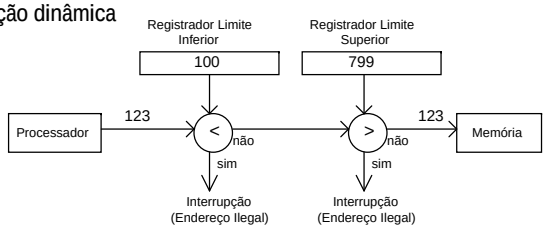


Exemplo de amarração dinâmica

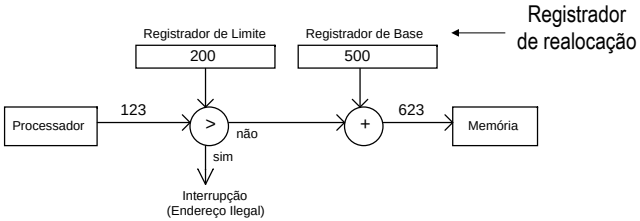


Exemplos de MMU

SEM amarração dinâmica



COM amarração dinâmica



Ligação dinâmica

- Ligação estática (bibliotecas estáticas)
  - Bibliotecas são tratadas como módulos objetos
  - Inclui cópia da biblioteca no executável
    - Desperdício de espaço
- Ligação dinâmicas (bibliotecas dinâmicas)
  - Bibliotecas/rotinas são carregadas e ligadas por demanda
    - Conceito de stub
  - Permite compartilhamento → se bibliteca e rotina já estão em memória basta ligar

Leituras complementares

- A. Tanenbaum. *Sistemas Operacionais Modernos* (3ª edição), Pearson Brasil, 2010.
  - Capítulo 3: seções 3.1 a 3.2.2
- A. Silberchatz, P. Galvin; *Sistemas Operacionais*. (7ª edição). Campus, 2008.
  - Capítulo 8 (seções 8.1 e 8.2)
- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Bookman 4ª edição, 2010
  - Capítulo 6 (seção 6.1 e anexo A)