MAC5753 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 10 de Novembro de 2020

Roteiro

Introdução à gerência de memória

Abstração de memória e Swap

Introdução à gerência de memória

Por que gerenciar a memória?

- Na arquitetura de Von Neumman a memória tem papel importante (instruções, e dados utilizados por essas instruções, passam pela memória antes de ir para a CPU)
- Assim como outros recursos, é um recurso finito que precisa ser bem utilizado (permitir que todos os processos usem de forma justa)

A memória ideal para um processo / programador

- Exclusiva do processo
- De tamanho infinito
- De velocidade de acesso infinita (atraso $= 0$)
- Não volátil
- De custo financeiro \$ 0,00
- Tal memória não existe :(

A memória real para um processo / programador

- A memória é gerenciada pelo SO e precisa ser disponibilizada para todos os processos (não é exclusiva de um processo)
- Armazenar dados na memória ocupa espaço físico. Mais espaço, mais elementos físicos, o que leva a limitar a sua capacidade em bytes (não tem tamanho infinito)
- Acessar a memória exige transferência de dados entre várias unidades do computador (CPU, disco e dispositivos de E/S). Esses dados são transferidos via barramento, que por mais rápidos que sejam possuem um limite de largura de banda (não tem velocidade de acesso infinita)

A memória real para um processo / programador

- A memória principal (RAM) precisa ser usada por todos os processos. Os dados mantidos nela estão sempre mudando. Ideal é que perca os dados quando o computador é desligado (A memória é volátil)
- Quanto mais rápida, mais cara e menor (hierarquia de memória)

Hierarquia de memória

- As memórias podem ser separadas em vários níveis
- Memórias mais rápidas tem menos capacidade e ficam mais próximas do processador. Por outro lado são mais caras
- Memórias mais lentas tem mais capacidade, ficam mais longe do processador e podem ser não voláteis. O custo por byte é bem menor que o custo por byte das memórias mais próximas do processador
- Memória cache, Memória RAM, Discos
- O ideal é que isso tudo seja transparente para o processo/programador. **O gerenciador de memória do SO tem a tarefa de fazer isso.** Ele abstrai essa hierarquia.

Objetivos do gerenciador de memória

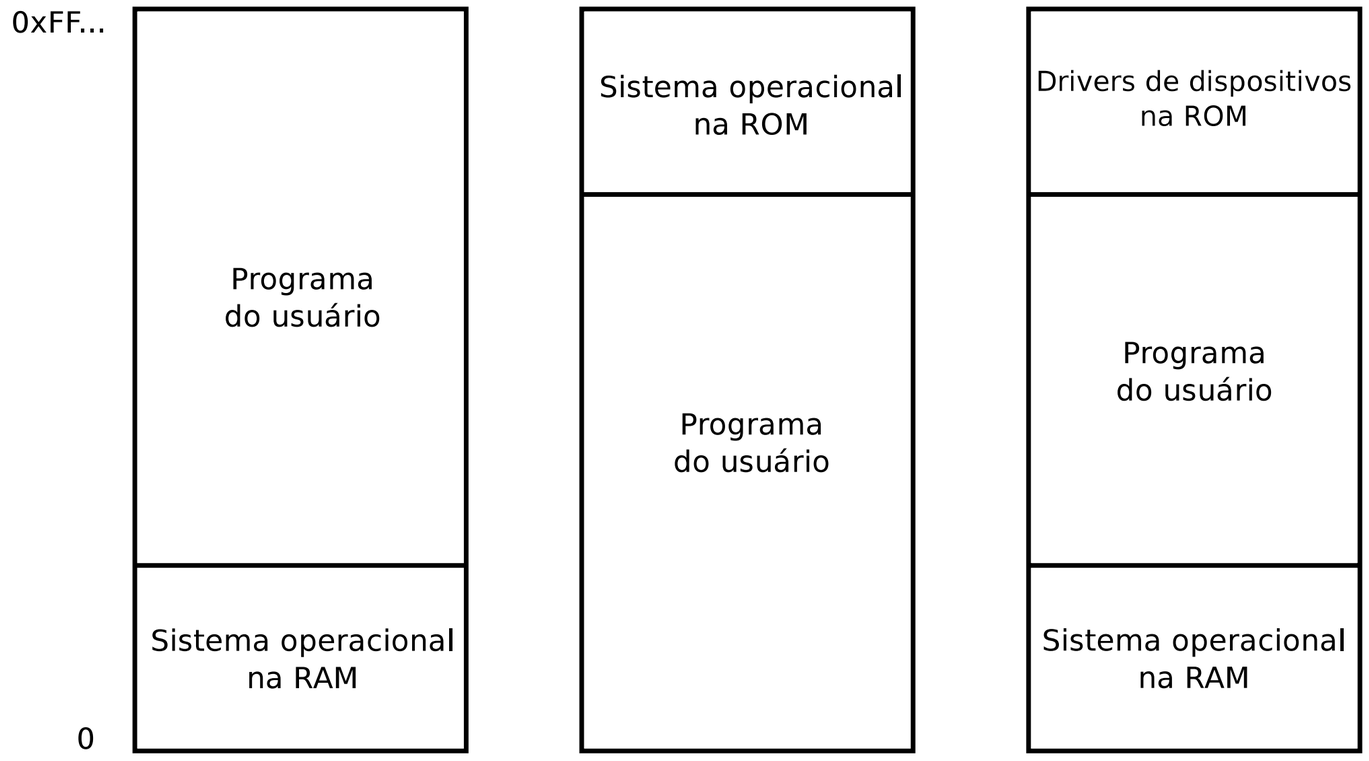
- Gerenciar a memória de forma eficiente (usar abstração para os processos)
- Manter um controle de quais partes da memória estão sendo usadas
- Alocar memória para processos quando eles precisarem
- Desalocar memória quando os processos terminarem de executar

Abstração de memória e Swap

Primeiro modo – Sem abstração

- Faz sentido se há um único processo ou mesmo quando não havia sistema operacional
- Para obter paralelismo num cenário desses é necessário usar múltiplas threads, o que limita tal paralelismo já que múltiplas threads fazem sentido quando há um único problema a ser resolvido. Dois programas com propósitos completamente diferentes deveriam ser dois processos e não duas threads.

Organizando a memória quando não há abstração



- No primeiro e no terceiro modo o programa pode “quebrar” o SO se acessar uma posição de memória indevida

Múltiplos programas quando não há abstração

- Embora o caso sem abstração faça sentido quando há apenas 1 processo em execução, é possível rodar múltiplos processos num SO que siga esse modo
- A ideia é que na mudança de contexto (ou no bloqueio do processo em execução por conta de uma operação de E/S), o conteúdo inteiro da memória seja salvo no disco e recuperado depois.
- Problema: salvar a memória inteira é custoso! Mas como não há abstração e um processo pode ter escrito em qualquer posição, o jeito é salvar tudo.

Segundo modo – Espaços de endereço

- Problema 1 de não haver abstração: um programa de usuário pode facilmente quebrar o SO
- Problema 2 de não haver abstração: é custoso ter mais de um processo em execução (não é um problema sério em um cenário com mainframes antigos dedicados a resolver um único problema mas é um problema que deve ser resolvido em computadores pessoais de propósito geral)

Noção de espaços de endereço

- Para possibilitar multiprocessamento a memória precisa de proteção e realocação para evitar que processos escrevam em áreas que não são deles e para garantir uma melhor utilização do espaço de memória disponível
- A ideia de espaços de endereço é limitar a memória que é vista por um processo e mostrar essa área limitada para ele como se fosse toda a memória

O que é um espaço de endereço?

- Um conjunto de endereços que um processo pode usar para endereçar posições de memória
- Cada processo tem seu próprio espaço de endereço (de um modo geral sem intersecção com os espaços de endereço de outros processos)

Base e limite

- Como cada processo pode acessar as mesmas áreas de memória de outro (por exemplo, os dois podem ter um JMP 28), é importante que esse valor (28) seja relativo para cada processo e não absoluto na memória inteira
- Para permitir a realocação dos processos, cada um deles passa a ter dois valores (endereços de memória) associados: base e limite. Toda a área de memória localizada entre base e entre limite pertence ao processo. Esses valores precisarão ser salvos sempre que um processo for criado. É responsabilidade do gerente de memória informar para o processo quais são esses valores, quando o processo é criado.

Base e limite

- Base = uma posição de memória livre
- Limite = o maior endereço que o programa pode acessar (desconsiderando alocação dinâmica por enquanto) [1]
- O gerente de memória tem que encontrar alguma área da memória que tenha espaço livre contínuo com tamanho igual ao tamanho do programa

[1] na prática um registrador armazena na verdade o tamanho do programa

Base e limite

- O gerente de memória precisa somar o valor de Base em todas as instruções que fazem referências a posições de memória no programa
- Sempre que houver algum acesso à memória, o gerente de memória tem que verificar se o acesso estará dentro do endereço Limite. Se não estiver, uma falha deve ocorrer informando o problema
- Ex.: processo foi alocado para começar no endereço 1023. O valor limite é 2056. Se há um `JMP, 100`, o gerente de memória tem que transformar em `JMP, 1123`. Como 1123 é menor que 2056, essa instrução será executada sem falha
- Problema de usar base e limite: cada acesso à memória exige uma soma e uma comparação