Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 08 - Virtualização de Memória

Aula de Hoje

Aula de Hoje

Na aula de hoje, veremos conceitos de virtualização de memória.

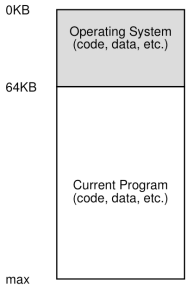
Iremos abordar as primeiras implementações de memória em sistemas operacionais, e veremos pouco a pouco como foram evoluindo.

Também veremos uma versão simplificada (por enquanto) de como funciona a tradução de endereços virtuais pelo sistema.

Aula de Hoje

Histórico

Nos primeiros S.O., implementação da memória era simples: Bibliotecas do S.O. ocupavam uma região pequena da memória, e o programa do usuário utilizava o resto.



Multiprogramação e Time share

Depois de um tempo, a preocupação dos projetistas era utilizar os recursos do sistema ao máximo.

Multiprogramação surgiu para que o S.O. pudesse chavear um processo que ficasse ocupado (fazendo E/S, por exemplo) para outro que estivesse pronto.

Em seguida surgiu o conceito de time-share e sistemas mais interativos.

Como isso afetou a implementação da memória?

Multiprogramação e Time share

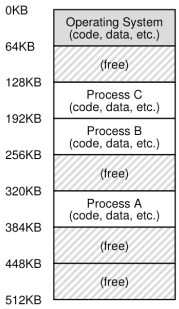
Opção 1: O programa que está na memória tem acesso a toda memória e, quando for trocado, tem seus dados salvos no disco para a troca de contexto.

Qual o problema desta abordagem?

Multiprogramação e Time share

Opção 1: O programa que está na memória tem acesso a toda memória e, quando for trocado, tem seus dados salvos no disco para a troca de contexto.

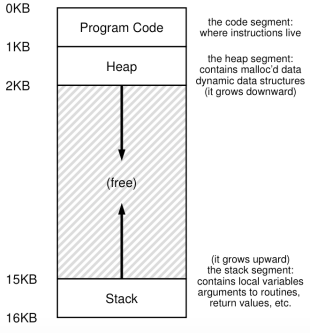
Qual o problema desta abordagem?
Opção 2: Dividir a memória entre os vários processos:



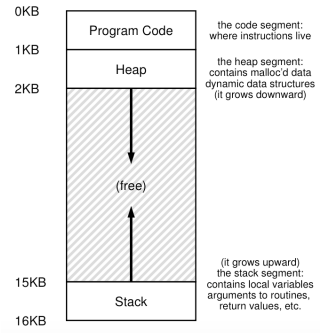
Espaço de endereçamento

Para facilitar a vida dos usuários, o S.O. cria uma abstração para a memória, chamada de **espaço de endereçamento**.

O espaço de endereçamento contém todos os estados da memória utilizada pelo programa.



Espaço de Endereçamento



A área de código contém as instruções do programa sendo executado.

A pilha é utilizada para guardar as variáveis locais e as chamadas de função (escopo).

O heap guarda as variáveis dinamicamente alocadas.

S.O.s modernos podem ter outras regiões, mas estas três são as principais.

Espaço de Endereçamento

Os endereços de um programa não são os endereços reais, mas sim uma abstração da memória criada pelo sistema operacional.

Isso simplifica a compilação, pois a mesma não precisa se preocupara com endereços de memória.

Mas como o S.O. faz para gerenciar e mapear a memória real para a memória do processo?

Virtualização de memória.

Objetivos de virtualizar memória

Transparência: para o programa do usuário, só existe ele na memória inteira.

Eficiência: Esta virtualização deve ser simples e eficiente em termos de tempo e de espaço. A virtualização de memória usa bastante hardware dedicado para isto.

Proteção: O S.O. deve proteger os processos e ele próprio da influência de outros processos. Virtualização gera o isolamento dos processos, ou seja, um processo não consegue acessar o espaço de endereçamento de outro.

Aula de Hoje

Mecânica de tradução de endereços

Em sistemas modernos, a tradução de endereços virtuais para endereços físicos é uma tarefa essencial.

Ela é executada **cada vez** que se acessa uma posição de memória.

Por isto, esta tradução é feita via hardware.

A tarefa do S.O. é gerenciar este hardware, para garantir que a tradução seja feita corretamente sempre.

- **Gerenciar memória**.

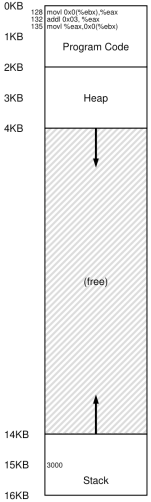
Simplificando...

Por enquanto, vamos assumir algumas simplificações para o entendimento ser melhor:

- O espaço de memória do usuário será alocado contiguamente na memória.
- Espaço de endereçamento virtual é menor que a memória física.
- Espaço de endereçamento de todos os processos terá o mesmo tamanho.

Mais adiante nas aulas eliminaremos estas simplificações.

Exemplo




```
void func() {
int x = 3000;
x = x + 3; // Preste atencao nesta linha
```

A terceira linha gera o seguinte código em assembler:

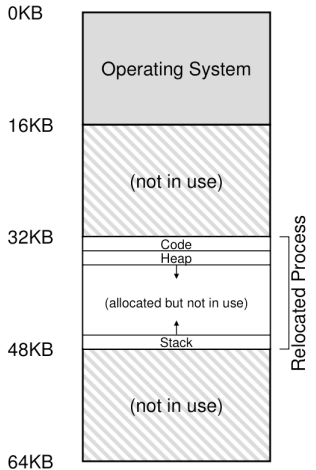
```
128: movl 0x0(%ebx), %eax ;carrega 0+ebx em eax
132: addl $0x03, %eax ;soma 3 no registrador eax
135: movl %eax, 0x0(%ebx) ;guarda eax de volta na memoria
```

Na realidade...

Na hora de compilar os programas, o compilador deve colocar todas as referências à memória nas instruções

Nem todo programa pode começar de fato na posição de memória 0 da memória física (normalmente as posições mais baixas são reservadas ao S.O., por exemplo.

O S.O. precisa de alguma forma de realocar o processo para outra posição de memória de forma transparente.



Relocação Dinâmica

No início: Relocação por software. Um programa, chamado **loader**, na hora de carregar um programa para a memória, atualiza todos os endereços das instruções para iniciarem da posição inicial do processo ao invés de zero.
Problema: Custos extras, segurança.

Na sequência: Base and Bounds. Dois registradores, o **base** e o **bounds** (ou limit) auxiliam na tradução de endereços.

Base and bounds

Programas são compilados normalmente, achando que iniciam do endereço zero.

No início da execução, o S.O. escolhe uma posição para este processo e define o registrador base para o valor do início desta área.

Todas as instruções de acesso à RAM são traduzidas **via hardware**.

O endereço virtual (do programa) é traduzido a um endereço físico. Este processo é chamado de relocação dinâmica.

Bounds

E o bounds?

- Serve para proteção.
- Cada acesso à memória é acompanhado de uma checagem caso o endereço virtual acessado esteja na área de um processo: 0 como limite inferior, bounds (tamanho do espaço de endereçamento) do processo como limite superior.
- Checagem toda via Hardware na própria CPU.

Registradores Base e Bound são hardwares que ficam na CPU. Normalmente hardwares usados para controlar memória são coletivamente chamados de Memory Management Unit (MMU).

Exemplo de tradução

Imaginem um processo com espaço de endereçamento de 4KB.

Ao rodar, o S.O. o coloca no endereço 16KB.

Todos os acessos à memória então serão traduzidos somando o valor de base e verificando se o endereço resultante passou do bounds.

Virtual Address		Physical Address
0	→	16 KB
1 KB	→	17 KB
3000	→	19384
4400	→	*Fault (out of bounds)*

Suporte de Hardware

- registradores Base e bound: Só podem ser modificados via instruções em modo privilegiado (modo kernel).
- Gerar excessões quando há um acesso de memória ilegal (*out of bounds*). Normalmente um processo é finalizado nestes casos.

Dificuldades para o S.O.

- Na criação de processos: O S.O. deve encontrar uma região de memória real livre para disponibilizar para o processo. Normalmente usa uma estrutura para guardar os espaços livres.
- No fim de um processo: O S.O. tem que recuperar toda a memória usada pelo processo e atualizar a lista de espaço livre.
- A cada troca de contexto: Quando se troca um processo, se troca também o seu base-and-bounds, então estes devem ser trocados juntos.
- S.O. pode mover o espaço de endereçamento de um processo na memória real, desde que ele não esteja executando no momento.

Finalizando

Nesta aula, aprendemos um pouco sobre o que é virtualização de memória, vimos algumas abordagens históricas, e abordamos os mecanismos por trás da tradução de endereços (em uma forma simplificada).

Nas próxima aulas, iremos ver pouco a pouco as tecnologias, tanto em termos de mecânica quanto de política que foram sendo utilizadas na virtualização de memória.

Iremos também pouco a pouco vendo a complexidade aumentar conforme relaxamos as simplificações feitas na aula de hoje.

Até o fim do dia, haverá no colabweb uma pequena atividade simples para vocês ;-)