Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 09 - Segmentação

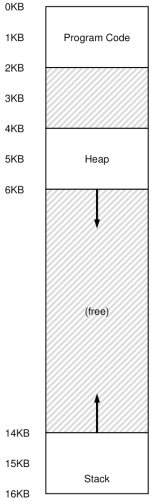
Aula de Hoje

1. Segmentação
   - Mecanismo
   - Politicas de Segmentação
2. Gerenciamento de espaço livre
   - Mecanismos de alocação de espaço livre
   - Políticas de alocação de espaço livre

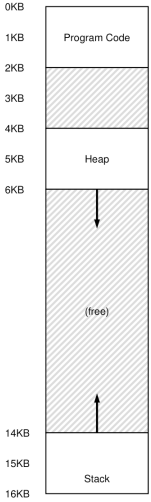
Anteriormente

- Vimos como alocar o espaço de endereçamento de vários processos na memória.
  - Utilizando registradores de base e bound.
- Mas talvez você tenha percebido que, no espaço de endereços de um processo, tem bastante espaço livre. Mas **onde**?

Anteriormente



Anteriormente



Entre o Stack e o Heap.

Com exemplos pequenos, parece besteira, mas num espaço de 32 bits cada processo ocuparia 4GB em endereços.

Aula de Hoje

1. Segmentação
   - Mecanismo
   - Politicas de Segmentação
2. Gerenciamento de espaço livre
   - Mecanismos de alocação de espaço livre
   - Políticas de alocação de espaço livre

Segmentação

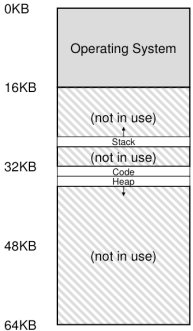
Segmentação é a primeira solução para o problema da memória (vem dos anos 60).

- Porém, muitos dos conceitos e hardware criados para ele existem em sistemas modernos até hoje.

Ao invés de um par base e bound por processo, ter um par para cada segmento lógico do processo.

- Segmento é qualquer porção contígua de memória.
- Em um exemplo simples, código, pilha e heap poderiam ser considerados segmentos e alocados separadamente.
- Evitando o buraco no meio do processo.

Segmentação



Só o espaço usado é alocado.

Boa parte do espaço de endereçamento do processo não está alocada de verdade.

- Endereços esparsos

Mecanismo

Para implementar segmentação, deve haver uma modificação na MMU, com cada segmento do processo tendo seu par de base e bound.

Segment	Base	Tamanho
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Como seria o processo de tradução do endereço 100 (que faz parte do Code)?

E do 4200 (que faz parte do Heap)?

E se o endereço passar do limite do bound?

Mecanismo

Para implementar segmentação, deve haver uma modificação na MMU, com cada segmento do processo tendo seu par de base e bound.

Segment	Base	Tamanho
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Como seria o processo de tradução do endereço 100 (que faz parte do Code)?

E do 4200 (que faz parte do Heap)?

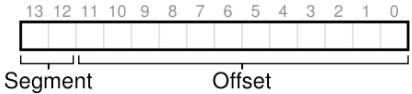
E se o endereço passar do limite do bound?**Segmentation Fault**!

Aula de Hoje

Mecanismo de Segmentação

Primeira duvida é: Como fazer para saber a qual segmento um endereço pertence?

Abordagem 1: **Explícita**. Endereço é quebrado em duas partes, com a primeira, de poucos bits, contendo o identificador do segmento, e o resto para o endereço em si.



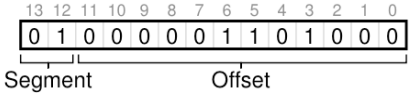
Nesse caso, se os dois bits são:

- 00 - Code
- 01 - Heap
- 10 - Stack

Alguns sistemas colocam o code e o heap no mesmo segmento, para usar apenas um bit identificador.

Exemplo de segmentação com endereço explícito

Endereço 4200:



Segmentação com endereço implícito

O Segmento também pode ser determinado implicitamente.

Hardware determina a qual segmento o endereço pertence através da fonte do endereço.

- Se veio do PC (via fetch), é de código.
- Se veio de registradores Base ou de pilha, é stack.
- Senão tem que ser do Heap.

Pilha na segmentação

Um detalhe importante deixado de lado até agora é a pilha: Ela cresce para trás!!

Comofas/

Pilha na segmentação

Um detalhe importante deixado de lado até agora é a pilha: Ela cresce para trás!!

Comofas/Mais Hardware.

Além do Base e bound, é adicionado um bit de controle para a direção que o segmento cresce.

Segmentação fina vs grossa

Até aqui, estamos vendo processos com poucos segmentos relativamente grandes.

- Segmentação grossa.

Contudo, alguns sistemas suportam uma segmentação mais fina, contendo diversos segmentos para cada processo.

Estes sistemas usam tabelas para guardar todos os dados segmentos, o que aumenta a flexibilidade.

Os segmentos são definidos durante a compilação.

Aula de Hoje

Suporte do S.O.

1. O que fazer durante uma troca de contexto?

Suporte do S.O.

1. O que fazer durante uma troca de contexto?
2. Como gerenciar o espaço livre na memória. (logo mais).

Sistemas Operacionais modernos geralmente não usam segmentação, e sim paginação. Contudo, boa parte das bibliotecas de alocação de memória usam segmentação.

Aula de Hoje

1. Segmentação
   - Mecanismo
   - Politicas de Segmentação
2. Gerenciamento de espaço livre
   - Mecanismos de alocação de espaço livre
   - Políticas de alocação de espaço livre

Gerenciamento de espaço Livre

Uma parte importante do gerenciamento de memória é como lidar com o espaço livre de memória.

- E como escolher quais espaços são dados para cada processo.

Com segmentação, o gerenciamento de espaço livre é hard.

Fragmentação

Gerência de memória tem como um de seus principais problemas a fragmentação.

- Pedaços de memória inutilizados por serem pequenos demais.

Pode ser fragmentação interna ou externa.

Segmentação causa externa.

Introdução a memória livre

Segmentação não é usada em S.O.s modernos, mas sim paginação.

Contudo, isto é transparente pros programas.

Se cada alocação de memória for uma chamada de sistemas, isto pode ser ruim para a performance.

Solução: Bibliotecas de alocação de memória nível usuário fazem a sua própria segmentação.

Exemplo: *glibc* e o *malloc*.

Alocação nível usuário

No nível usuário, é comum alocar blocos de memória maiores do que o pedido, para que futuras alocações sejam feitas totalmente em modo usuário, sem chamadas de sistema.

Para isto, os processos devem tanto organizar estes segmentos de memória quanto controlar o seu espaço livre.

- Com uma espécie de lista de espaço livre.

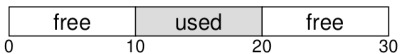
Focaremos na fragmentação externa (espaço livre entre blocos) e não na interna (espaço livre dentro dos blocos).

Aula de Hoje

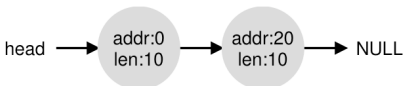
Split e Coalesce

Duas técnicas básicas para entender a gerencia de espaço livre é o split (dividir) e coalesce (aglutinar).

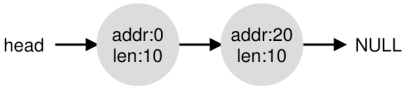
Dada a memória abaixo:



A memória livre poderia ser representada por esta lista:



Exemplo

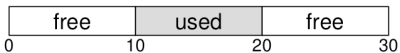


Caso se peça qualquer pedaço de memória acima de 10 bytes, a alocação falhará pois não há espaço deste tamanho.

Caso seja pedido, por exemplo, 1 byte, ocorre o splitting: Um dos blocos é quebrado, com uma parte sendo devolvida na alocação e outra sendo atualizada:

Coalesce

Voltando ao inicial:



Imagine agora que o espaço do meio sofre um *free*, sendo liberado. O que acontece? Isso?

Qual o problema aqui?

Coalesce

Sempre que um *free* ocorre, o gerente verifica se os endereços contíguos a esta memória estão livres, para neste caso aglutiná-los. Então isto:

Vira isto:

Mantendo o tamanho de regiões alocadas

Já repararam que, enquanto no `malloc` você passa o tamanho da área como parâmetro, no `free` você só passa o ponteiro?

Como funciona isso? Magia?

Mantendo o tamanho de regiões alocadas

Já repararam que, enquanto no `malloc` você passa o tamanho da área como parâmetro, no `free` você só passa o ponteiro?

Como funciona isso? Magia?Não, **tecnologia**.

Boa parte dos alocadores guarda uma informação de tamanho no seu início, seguido de um número mágico arbitrário, que garante que aquela região foi de fato alocada e não está rolando um `free` louco.

Exemplo

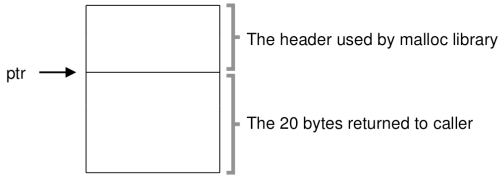


Figure 17.1: **An Allocated Region Plus Header**

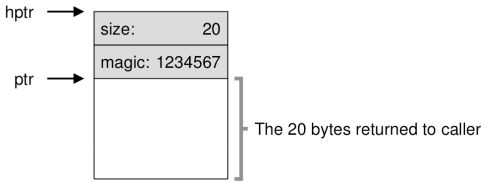


Figure 17.2: **Specific Contents Of The Header**

Crescimento e diminuição do Heap

Normalmente, as bibliotecas de alocação começam com blocos relativamente pequenos.

Conforme os blocos vão enchendo/ alocações muito grande são feitas, novos blocos são pedidos do S.O. via chamada de sistema.

De forma análoga, conforme vão ocorrendo `free`, a memória não é liberada imediatamente, o alocador mantém posse dela para futuras alocações. Qual o problema disso?

Crescimento e diminuição do Heap

Normalmente, as bibliotecas de alocação começam com blocos relativamente pequenos.

Conforme os blocos vão enchendo/ alocações muito grande são feitas, novos blocos são pedidos do S.O. via chamada de sistema.

De forma análoga, conforme vão ocorrendo `free`, a memória não é liberada imediatamente, o alocador mantém posse dela para futuras alocações. Qual o problema disso?

Desperdício de memória.

Aula de Hoje

Políticas de alocação de espaço livre

Existem centenas de abordagem para a escolha de qual pedaço de espaço livre distribuir para cada alocação.

Cada uma com vantagens e desvantagens, especialmente dependendo do workload.

Fiquem calmos, nesta disciplina focaremos apenas em alguns mais simples para dar uma noção.

Mas saibam que o glibc usa uma mistura deles.

Best Fit

Percorre a lista de espaço livre e, dado todos os blocos livres em que a memória a ser alocada cabe, escolha o menor dele.

Intuição: Reduzir o espaço desperdiçado por fragmentação causada por blocos grandes serem pulverizados em diversos pequenos.

Problemas: Percorrer a lista de livres inteira pode ser caro em termos de desempenho.

Worst Fit

O inverso do Best Fit, sempre aloca no maior espaço disponível.

Intuição: Deixar o máximo de pedaços grandes disponíveis (pois as alocações vão sempre se focar no maior de todos os blocos).

Problemas: Caro como o Best Fit, e ainda por cima continua levando a bastante fragmentação.

First Fit

Aloca no primeiro bloco livre em que a alocação caiba.

Intuição: Solução mais simples, pouco overhead de procura.

Problemas: Pode gerar uma lista com diversos blocos livres pequenos no começo da lista que sempre gerarão overhead na procura. Isto pode ser resolvido com coalescing, aglutinando estes pequenos blocos em blocos maiores.

Next Fit

Similar ao First Fit, o Next Fit mantém um ponteiro para a posição da última alocação e continua a procura a partir dali.

Intuição: Diminuir a fragmentação no início do First Fit, espalhando mais as alocações.

Glibc

O GLibc, como tudo nessa vida de S.O., usa várias estruturas para cuidar do alocamento:

Cada processo tem arenas para a main e suas threads, e cada arena destas possui diversos blocos.

Os blocos são divididos em diversas filas de livres:

- uma fila de blocos de tamanho pequeno e fixo (com first fit)
- uma fila de recém liberados (para aglutinação)
- uma de blocos pequenos de tamanhos pequenos sortidos (first fit também)
- uma de blocos grandes, onde usam best fit.

Trabalhinho

Até amanhã, já estará nosso primeiro trabalho deste módulo no colabweb.

Será o nosso próprio alocador de memória!