Exercícios de Sistemas Operacionais II
Gerência de Memória

Universidade Estadual do Centro-Oeste (UNICENTRO)
Departamento de Ciência da Computação (DECOMP)
Sistemas Operacionais II
Professor Diego Marczal
Paulo R. Urio
Eduardo T. Feliczaki

17 de março de 2012

1. Quais as funções básicas da gerência de memória?

A função básica do gerenciador de memória é alocar dinamicamente porções de memória para programas, quando requisitadas, e liberar para reuso quando não for mais necessária (BARTLETT, 2004). Também deve gerenciar toda a memória disponível (memória principal e secundária) de forma transparente aos programas do *userspace* (RUSLING, 1999).

2. Considere um sistema computacional com 40KB de memória principal e que utilize um sistema operacional de 10KB que implemente alocação contígua de memória. Qual a taxa de subutilização da memória principal para um programa que ocupe 20KB de memória?

Assumindo que o sistema operacional ocupará os 10 KB ao início da memória e os outros 30 KB restantes formem um único bloco contíguo. Quando o processo que necessita de 20KB de memória for iniciado, será alocado o único bloco contíguo de memória, com tamanho 30KB, por ser grande o suficiente para acomodá-lo na memória. Haverá **10 KB** não utilizados neste bloco. Esse espaço perdido é chamado de fragmentação interna (MURTAGH, 2011).

3. Suponha um sistema computacional com 64KB de memória principal e que utilize um sistema operacional de 14KB que implemente alocação contígua de memória. Considere também um programa de 80KB, formado por um módulo principal de 20KB e três módulos independentes, cada um com 10KB, 20KB e 30KB. Como o programa poderia ser executado utilizando-se apenas a técnica de overlay?

O sistema operacional será alocado no início da memória principal, deixando 50 KB para programas em *userspace* utilizarem. *Overlay* é o processo de transferir um bloco de código de programa ou dados para a memória interna, substituindo pelo bloco atual (OXFORD, 2010). Os blocos independentes serão colocados na memória, no espaço dedicado ao *overlay*, um por vez para serem utilizados. A memória ficará organizada de acordo com a Figura 1. O módulo principal é encarregado de executar os *overlays*.

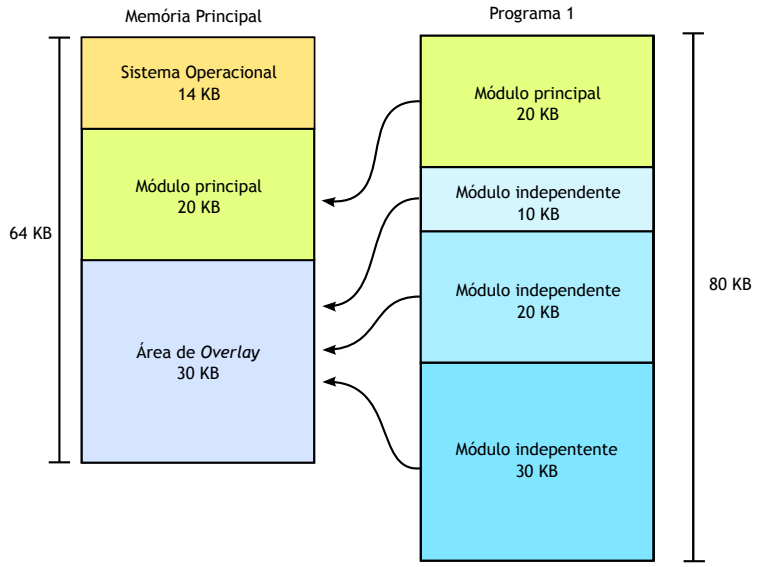

Figura 1: Estado da memória principal

4. Considerando o exercício anterior, se o módulo de 30KB tivesse seu tamanho aumentado para 40KB, seria possível executar o programa? Caso não possa, como o problema poderia ser contornado?

Não seria possível executar o programa. Para se resolver o problema, o módulo de 40KB deveria ser quebrado em dois módulos indepententes para assim poder ser executado. Ao tentar executar *overlay* no módulo independente de 40 KB em uma área de *overlay* de 30KB, ocorre a fragmentação externa (MUR-

TAGH, 2011).

5. Qual a diferença entre fragmentação interna e externa da memória principal?

A fragmentação interna ocorre quando os programas não preenchem as partições onde são carregados, ocorre com técnicas de alocação absoluta, relocável e contígua. A fragmentação externa ocorre em técnicas de alocação dinâmica quando programas são terminados sem serem completamente liberados da memória, deixando cada vez espaços menores na memória, assim novos programas não podem ser executados.

6. Suponha um sistema computacional com 128KB de memória principal e que utilize um sistema operacional de 64KB que implementa alocação particionada estática relocável. Considere também que o sistema foi inicializado com três partições: P1 (8KB), P2 (24KB) e P3 (32KB). Calcule a fragmentação interna da memória principal após a carga de três programas: PA, PB e PC.

$frag(P)$ é uma função que calcula a fragmentação interna da partição P.

a) P1 ← PA (6KB); P2 ← PB (20KB); P3 ← PC (28KB)

$$frag(P1) + frag(P2) + frag(P3) = 2KB + 4KB + 4KB = 10KB$$

b) P1 ← PA (4KB); P2 ← PB (16KB); P3 ← PC (26KB)

$$frag(P1) + frag(P2) + frag(P3) = 4KB + 8KB + 6KB = 18KB$$

c) P1 ← PA (8KB); P2 ← PB (24KB); P3 ← PC (32KB)

$$frag(P1) + frag(P2) + frag(P3) = 0KB + 0KB + 0KB = 0KB$$

7. Considerando o exercício anterior, seria possível executar quatro programas concorrentemente utilizando apenas a técnica de alocação particionada estática relocável? Se for possível, como? Considerando ainda o mesmo exercício, seria possível executar um programa de 36KB? Se for possível como?

Não, não há quatro partições para os quatro processos. Seria apenas possível

com *swapping*. Não, geraria fragmentação externa.

8. Qual a limitação da alocação particionada estática absoluta em relação a alocação estática relocável?

Na absoluta programas exclusivos são feitos para partições específicas. Na relocável os programas podem rodar em qualquer partição.

9. Considere que os processos da tabela a seguir estão aguardando para serem executados e que cada um permanecerá na memória durante o tempo especificado. O sistema operacional ocupa uma área de 20KB no início da memória e gerencia a memória utilizando um algoritmo de particionamento dinâmico modificado. A memória total disponível no sistema é de 64KB e é alocada em blocos múltiplos de 4KB. Os processos são alocados de acordo com sua identificação (em ordem crescente) e irão aguardar até obter a memória que necessitam. Calcule a perda de memória por fragmentação interna e externa sempre que um processo é colocado ou retirado da memória. O sistema operacional compacta a memória apenas quando existem duas ou mais partições livres adjacentes.

Processo	Memória	Tempo
1	30 KB	5
2	6 KB	10
3	36 KB	5

O processo 1 é iniciado, sendo alocados 30 KB para ele na memória principal. Juntamente com o processo 1, o processo 2 é alocado logo no primeiro endereço de memória não alocado, após a memória destinada ao processo 1.

O processo 3 deve esperar a finalização do processo 2, para que haja um bloco de memória não alocado suficiente para ele. Há memória não alocada suficiente para o processo, mas não um bloco inteiro suficiente. Este problema é chamado de fragmentação externa (MURTAGH, 2011).

10. Considerando as estratégias para escolha da partição dinamicamente, conceitue as estratégias best-fit e worst-fit especificando prós e contras de cada uma.

O alocador com estratégia *best-fit* procura o menor espaço de memória não alocada, onde caiba o processo (BARNETTE, 1999). Nesta estratégia, pode haver

uma lista ordenada por tamanho de blocos livres para aumentar a eficiência da busca.

Na estratégia *worst-fit*, o gerenciador de memória coloca o processo no maior bloco de memória não alocado. A ideia nesta estratégia é que após a alocação deste processo, irá sobrar a maior quantidade memória após o processo, aumentando a possibilidade de, comparado ao *best-fit*, outro processo poder usar o espaço restante (BARNETTE, 1999). Assim, o *worst-fit* tende a causar menos fragmentações.

11. Considere um sistema que possua as seguintes área livres na memória principal, ordenadas crescentemente: 10KB, 4KB, 20KB, 18KB, 7KB, 9KB, 12KB e 15KB. Para cada programa abaixo, qualseria a partição alocada utilizando-se as estratégias first-fit, best-fit e worst-fit (Tanenbaum, 1992)?

a) 12KB

First-Fit: 20 KB

Best-Fit: 12 KB

Worst-Fit: 20 KB

b) 10KB

First-Fit: 10 KB

Best-Fit: 10 KB

Worst-Fit: 20 KB

c) 9KB

First-Fit: 10 KB

Best-Fit: 9 KB

Worst-Fit: 20 KB

Referências

BARNETTE, O. B. William S. Gilley Robin J. Adams Emre Tunar N. D. Memory allocation. In: BOWIE STATE UNIVERSITY. *Operating Systems*. 1999. Disponível em: <http://courses.cs.vt.edu/csonline/OS/Lessons/>. Acesso em: 16 mar. 2012.

BARTLETT, J. Inside memory management. *developerWorks*, nov. 2004.

MURTAGH, T. P. Contiguous memory allocation techniques. In: WILLIAMS COLLEGE. *Operating Systems*. Williamstown, MA 01267, 2011. v. 1. Disponível em: <http://www.cs.williams.edu/ tom/courses/432/outlines/lect18_2.html>. Acesso em: 16 mar. 2012.

OXFORD. overlay. In: OXFORD DICTIONARIES. *English Dictionary*. 2010. Disponível em: <http://oxforddictionaries.com/definition/overlay>. Acesso em: 16 mar. 2012.

RUSLING, D. A. Memory management. In: DAVID A RUSLING. *The Linux Kernel*. 3 Foxglove Close, Wokingham, Berkshire RG41 3NF, UK, 1999. v. 1. Disponível em: <http://tldp.org/LDP/tlk/mm/memory.html>. Acesso em: 15 mar. 2012.