Gestão de Memória
Parte II - algoritmos

Sistemas Operativos

2011 / 2012

Algoritmos de Gestão de Memória

- Os algoritmos de gestão de memória são utilizados para decidir:
  - Onde se deve colocar um bloco (segmento ou página) de programa dada a memória primária livre;
  - Quando transferir um bloco de memória secundária para memória primária e vice-versa;
  - Que bloco retirar de memória quando não existe mais memória primária livre ou quando a que existe disponível é inferior a um determinado valor considerado mínimo para o bom funcionamento do sistema.

Algoritmos de Gestão de Memória

- Tipos de decisões que o sistema operativo tem de tomar em relação à memória principal:
  - Alocação - Onde colocar um bloco na memória primária;
  - Transferência - Quando transferir um bloco de memória secundária para memória primária e vice-versa;
  - Substituição - Qual o bloco a retirar da memória primária.

Algoritmos de Alocação

Reserva de Memória

- Quando é necessária a reserva e a libertação de memória?
  - Na criação e terminação de processos
  - Na extensão do espaço de endereçamento (área de dados ou pilha)

Reserva de Memória

- Paginação
  - Muito simples basta encontrar uma página livre, normalmente existentes numa Lista de Páginas Livres do sistema operativo
- Segmentação
  - O tamanho variável dos segmentos torna mais complexa a reserva de espaço para um segmento
  - Na libertação de memória é necessário recompactar os segmentos

Reserva de Segmentos: Critérios de Escolha de Blocos Livres

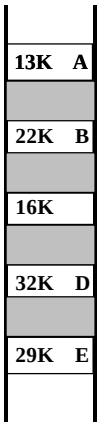
- Best-fit (o menor possível):
  - gera elevado número de pequenos fragmentos
  - em média percorre-se metade da lista de blocos livres na procura
  - a lista tem de ser percorrida outra vez para introduzir o fragmento
- Worst-fit (o maior possível):
  - pode facilmente impossibilitar a reserva de blocos de grandes dimensões
  - a lista de blocos livres tem de ser percorrida para introduzir o fragmento

Reserva de Segmentos: Critérios de Escolha de Blocos Livres

- First-fit (o primeiro possível):
  - minimiza a tempo gasto a percorrer a lista de blocos livres
  - gera muita fragmentação externa
  - acumula muitos blocos pequenos no início da lista, ficando para o fim os blocos maiores
- Next-fit (o primeiro possível a seguir ao anterior):
  - espalha os blocos pequenos por toda a memória

Critérios de Escolha de Blocos Livres (cont.)

- dimensão do pedido: 15k
  - best-fit – ?
  - worst-fit – ?
  - first-fit - ?



Critérios de Escolha de Blocos Livres: Algoritmo Buddy

- A memória livre é dividida em blocos de dimensão bn
- Se b = 2 então designa-se por buddy binário
- Para satisfazer um pedido de dimensão D percorre-se a lista à procura de um bloco de dimensão 2^k tal que $2^{k-1} < D \leq 2^k$
- Se não for encontrado procura-se um de dimensão 2^{k+i} , i >0, que será dividido em duas partes iguais (buddies)
- Um dos buddies será subdividido quantas vezes for necesssário até se obter um bloco de dimensão 2^k
- Se possível, na libertação um bloco é recombinado com o seu buddy, sendo a associação entre buddies repetida até se obter um bloco com a maior dimensão possível
- Consegue-se um bom equilíbrio entre o tempo de procura e a fragmentação interna e externa

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															

Processo A pede segmento de 34KB.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							

Processo B pede segmento de 66KB.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							

Processo C pede segmento de 35KB.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							

Processo D pede segmento de 67KB.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							

Processo C liberta o seu segmento.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							

Processo A liberta o seu segmento.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							

Processo B liberta o seu segmento.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							
$t = 7$	256K				D-128K		128K		512K							

Processo D liberta o seu segmento.

Fonte do exemplo: wikipedia

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							
$t = 7$	256K				D-128K		128K		512K							
$t = 8$	1024K															

Algoritmo de Buddy: conclusões

- Complexidade?
  - Reservar e libertar segmentos cresce logaritmicamente com o número de subdivisões de segmentos suportadas
  - e.g. 1MB até 64KB: 4 subdivisões
- Fragmentação externa?
  - Sim (como todos os algoritmos de reserva para segmentação)
- Fragmentação interna?
  - Sim! (ao contrário dos algoritmos anteriores)

Algoritmos de Transferência

Algoritmos de Transferência

- Existem três situações em que a transferência pode ser feita:
  - a pedido (**on request**): o programa ou o sistema operativo determinam quando se deve carregar o bloco em memória principal
    - normalmente usado na memória segmentada
  - por necessidade (**on demand**): o bloco é acedido e gera-se uma falta (de segmento ou de página), sendo necessário carregá-lo para a memória principal
    - normalmente usado na memória paginada
  - por antecipação (**prefetching**): o bloco é carregado na memória principal pelo sistema operativo porque este considera fortemente provável que ele venha a ser acedido nos próximos instantes

Transferência de Segmentos

- normalmente um processo para se executar precisa de ter pelo menos um segmento de código, de dados e de stack em memória
- caso haja escassez de memória os segmentos de outros processos que não estejam em execução são transferidos na íntegra para disco (swapping)
- os segmentos são guardados numa zona separada do disco chamada área de transferência (swap area)
- quando são transferidos todos os segmentos de um processo diz-se que o processo foi transferido para disco (swapped out)
- a transferência de segmentos faz-se usualmente a pedido:
  - em arquitecturas que suportem a falta de segmentos, certos segmentos de um programa podem ser transferidos para memória principal por necessidade

Transferência de Páginas

- o mecanismo normal de transferência de páginas é por necessidade:
  - páginas de um programa que não sejam acedidas durante a execução de um processo não chegam a ser carregadas em memória principal
- usam-se também políticas de transferência por antecipação para:
  - diminuir o número de faltas de página
  - optimizar os acessos a disco
- as páginas retiradas de memória principal são guardadas numa zona separada do disco chamada área de paginação:
  - apenas se ainda não existir uma cópia da página em disco
- as páginas modificadas são transferidas em grupos para memória secundária de modo a optimizar os acessos a disco

Swapping / Paging

- Quando é necessário libertar espaço na memória física o SO copia páginas para disco
  - escolhe aquelas que previsivelmente não irão ser usadas brevemente
  - zona do disco que as contém – "swap area"
- Terminologia: swapping vs. paging
  - granularidade: todas as páginas do processo (processo swapped out) vs. páginas individuais
- Minimizar latência: pre-fetching
  - traz páginas antes de serem pedidas

Algoritmos de Swapping de Processos ou de Segmentos

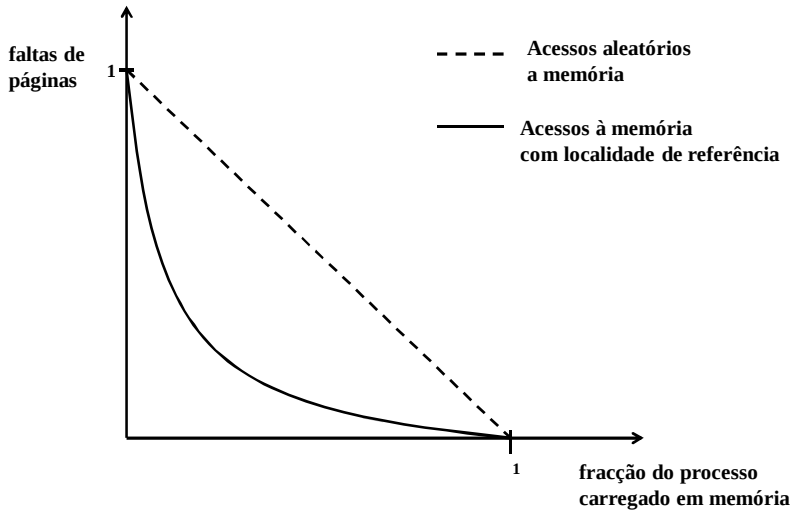
- Possíveis critérios para decidir qual o processo a transferir para disco:
  - estado e prioridade do processo: processos bloqueados e pouco prioritários são candidatos preferenciais
  - tempo de permanência na memória principal: um processo tem que permanecer um determinado tempo a executar-se antes de ser novamente enviado para disco
  - dimensão do processo

Espaços de Trabalho (working sets)

- Espaço de trabalho (working set) de um processo num determinado intervalo de tempo é:
  - o conjunto de páginas acedidas pelo processo nesse intervalo de tempo:
- Verifica-se que, para intervalos de tempo razoáveis:
  - o espaço de trabalho de um processo mantém-se constante e menor que o seu espaço de endereçamento
- Certos algoritmos adaptativos tentam manter o working set de cada processo em memória

- **O que acontece se a estimativa for muito baixa?**
- **E se for muito alta?**

Espaços de Trabalho (cont.)



Algoritmos de Substituição

Algoritmos de Substituição de Páginas

- Óptimo
  - Retira a página cujo próximo pedido seja mais distante no tempo
  - Requer conhecimento futuro
  - Usado como “benchmark”

Algoritmos de Substituição de Páginas

- Menos usada recentemente (Least Recently Used, LRU):
  - eficaz segundo o princípio de localidade de referência
  - latência associada à sua implementação rigorosa.
- Aproximação:
  - utilização de um contador por página que indique a que "grupo etário" ela pertence
    - actualizado regularmente pelo processo paginador
  - quando atingir um valor máximo, a página passa para a lista das livres ou das livres mas modificadas

Algoritmos de Substituição de Páginas

- Não usada recentemente (Not Recently Used, NRU):
  - agrupamento das páginas em 4 grupos:
    - 0: (R = 0, M = 0) Não referenciada, não modificada
    - 1: (R = 0, M = 1) Não referenciada, modificada
    - 2: (R = 1, M = 0) Referenciada, não modificada
    - 3: (R = 1, M = 1) Referenciada, modificada
  - o processo paginador percorre regularmente as tabelas de páginas e coloca o bit R a 0
  - libertam-se primeiro as páginas dos grupos de número mais baixo

Algoritmos de Substituição de Páginas

- FIFO:
  - é muito eficiente mas não atende ao grau de utilização das páginas (apenas ao seu tempo de permanência em memória primária)
  - existem variantes desta política que evitam a transferência de páginas antigas mas muito usadas para disco

Comparação: segmentação e paginação (1)

- Segmentação:
  - vantagens:
    - adapta-se à estrutura lógica dos programas
    - permite a realização de sistemas simples sobre hardware simples
    - permite realizar eficientemente as operações que agem sobre segmentos inteiros
  - desvantagens:
    - o programador tem de ter sempre algum conhecimento dos segmentos subjacentes
    - os algoritmos tornam-se bastantes complicados em sistema mais sofisticados
    - o tempo de transferência de segmentos entre memória principal e disco torna-se incomportável para segmentos muito grandes
    - a dimensão máxima dos segmentos é limitada

Comparação: segmentação e paginação (2)

- Paginação:
  - vantagens:
    - o programador não tem que se preocupar com a gestão de memória
    - os algoritmos de reserva, substituição e transferência são mais simples e eficientes
    - o tempo de leitura de uma página de disco é razoavelmente pequeno
    - a dimensão dos programas é virtualmente ilimitada
  - desvantagens:
    - o hardware é mais complexo que o de memória segmentada
    - operações sobre segmentos lógicos são mais complexos e menos elegantes, pois têm de ser realizadas sobre um conjunto de páginas
    - o tratamento das faltas de páginas representa uma sobrecarga adicional de processamento
    - Tamanho potencial das tabelas de páginas

UNIX
Gestão de Memória

Unix - Gestão de Memória

- Unix implementado sobre arquitecturas diferentes
- Dois grupos de implementações:
  - Segmentação com swapping
  - Paginação

Transferência (swapping)

- Arquitecturas segmentadas:
  - Regiões (texto, dados, stack) carregadas contiguamente em memória
  - Transfere para disco processos que estejam bloqueados ou com menor prioridade
- Existem 4 situações que potencialmente provocam a transferência:
  - Chamada fork - é preciso espaço para o novo processo
  - Chamada brk - expande o segmento de dados do processo
  - Crescimento natural da pilha
  - Sistema operativo precisa de espaço para carregar em memória um processo que estava swapped-out

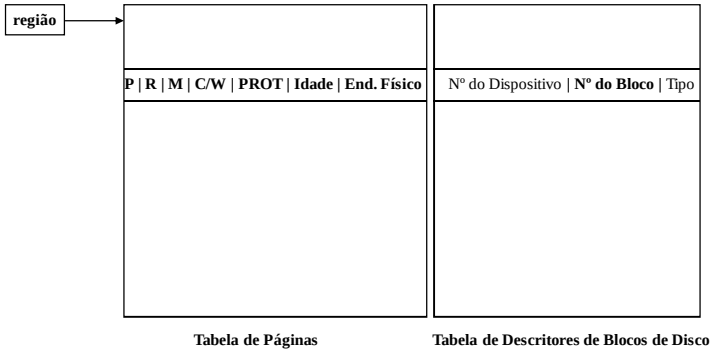
Swapper

- *Swapper*: processo que efectua as transferências de segmentos entre memória principal e secundária
- Área especial do disco reservada para os segmentos retirados de memória

Paginação

- Um processo tem inicialmente 3 regiões: código, dados e stack
- Cada região tem uma tabela de páginas própria

Tabela de Páginas e de Descritores de Blocos de Disco



Significado dos campos da tabela de páginas

- P – present – indica se a pagina está residente na memória primária
- R – referenced – foi acedida ou referenciada
-
- M – modified – modificada
- C/W – copy-on-write
- PROT – bits de protecção
- Idade – algoritmo do page stealer
- End. Físico da page frame
- Nº do Dispositivo | Nº do Bloco - disco e bloco onde se encontra
- Tipo – swap, demand fill, demand zero

Tabela pfdata

- Permite a gestão eficaz das páginas de memória física
- Indexada pelo número da página física
- Contém:
  - Estado da página (livre, existe cópia na área de swap ou num ficheiro executável, operação de leitura pendente)
  - Contador com número de processos que referenciam a página
  - Número de device e bloco onde existe cópia da página

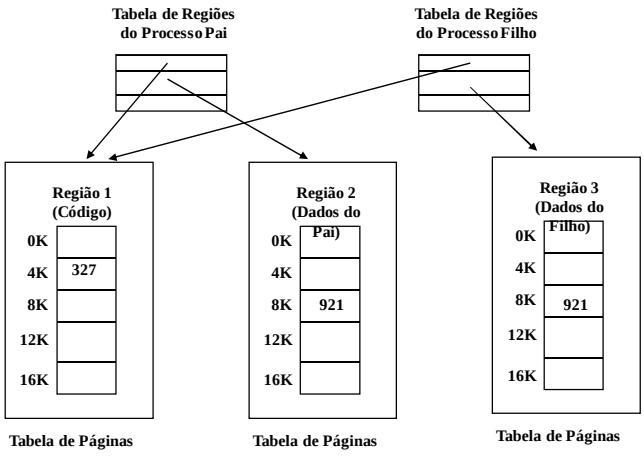
Substituição de Páginas

- Aproximação ao algoritmo Menos Usada Recentemente (LRU)
- Idade da página é mantida na PTE
- Page-stealer é acordado quando o número de páginas livres desce abaixo de um dado limite
- Percorre as PTE incrementando o contador de idade das páginas
- Se a página for referenciada a sua idade é anulada
- Se a página atingir uma certa idade marca-a para ser transferida

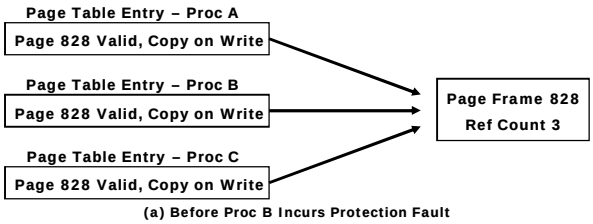
Criação de um Processo

- fork: duplica os segmentos de código, dados e pilha do pai
- não é feita nenhuma cópia física de memória:
  - cria uma nova tabela de regiões para o filho
  - dá-lhe a mesma região de código
  - copia as regiões de dados e pilha, i.e. aloca novas tabelas de páginas e copias as PTE e os descritores de blocos de disco
  - em seguida, percorre as PTE do pai e, para as páginas válidas, incrementa o contador de processos na pfdata e coloca a 1 o bit copy on write (no pai e no filho)
  - antes duma página ser escrita (pelo pai ou pelo filho):
    - o sistema copia-a para uma nova página que aloca,
    - preenchendo a PTE do processo onde ocorreu a falta com o endereço físico da nova página
    - só se copiam as páginas que forem de facto modificadas

fork



Tratamento do Copy on Write



(a) Before Proc B Incurs Protection Fault