Gestão de Memória
Algoritmos

Sistemas Operativos

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 Física

- Paginação
  - Muito simples:
    - basta encontrar uma página livre
    - normalmente existentes numa Lista de Páginas Livres do SO
- 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 (com lista ordenada por tamanho)
  - 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

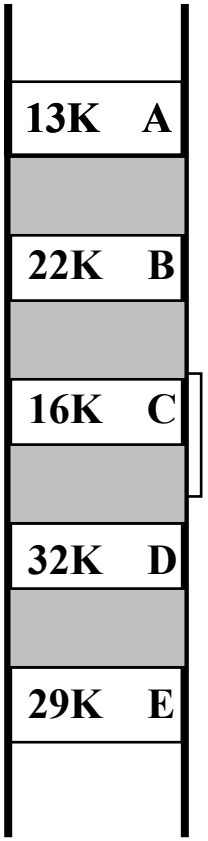
Reserva de Segmentos: Critérios de Escolha de Blocos Livres

- 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 à pesquisa 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 - ?



Algoritmos de Transferência

Três abordagens para a transferência

- 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 atualizada 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

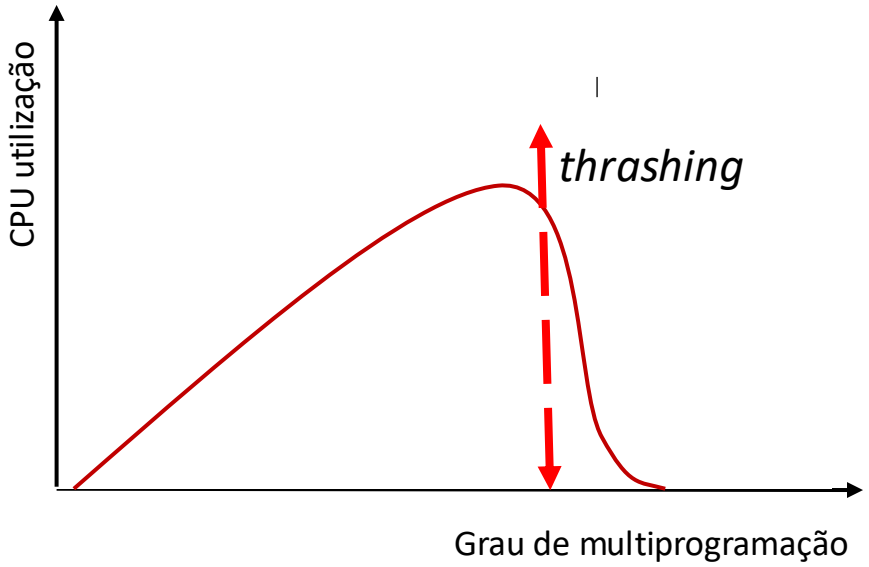
Algoritmos de Swapping de Processos ou de Segmentos

- 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**

Quanto espaço deve estar reservado/ocupado em memória física por um processo?

Thrashing

- O problema conhecido como *thrashing* ocorre quando o grau de multiprogramação é muito elevado
- Nesta situação os processo têm poucas páginas carregadas e portanto estão sempre a incorrer em faltas de página
- O *thrashing* ocorre quando os processos consomem pouco CPU porque gastam mais tempo na paginação (bloqueando a transferir páginas) do que na execução



Espaços de Trabalho (working sets)

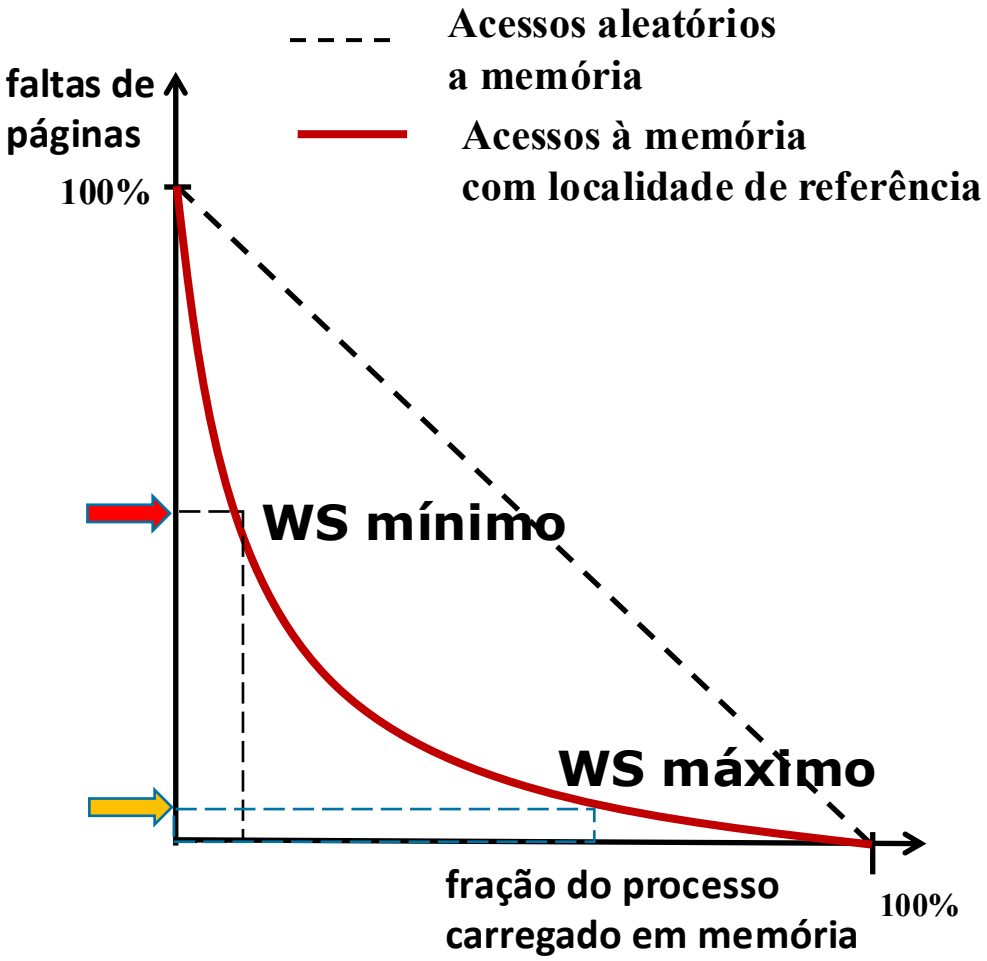
Espaço de trabalho de um processo num dado intervalo de tempo

=

conjunto de páginas acedidas pelo processo nesse intervalo de tempo

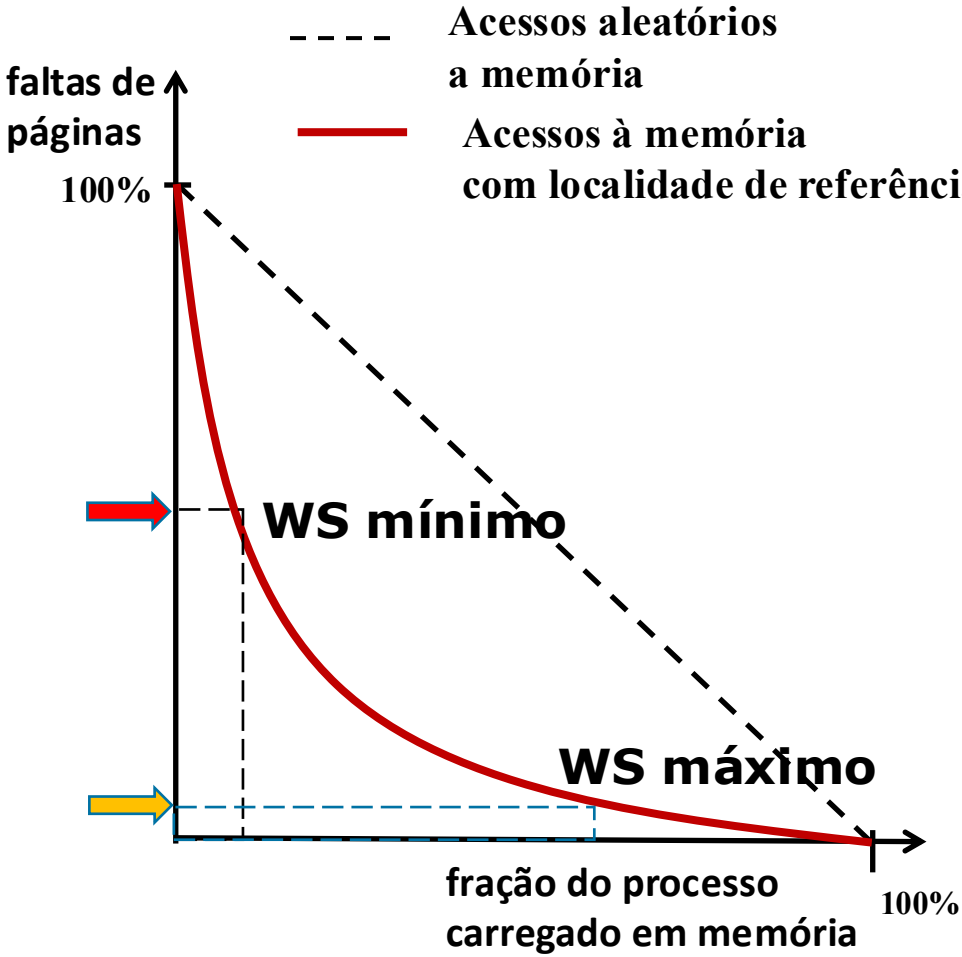
- O espaço de trabalho de um processo tende a ter dimensão **constante** e **muito menor** que o seu espaço de endereçamento
  - SO pode tentar estimar essa dimensão!

Espaços de Trabalho evitam o *Thrashing*



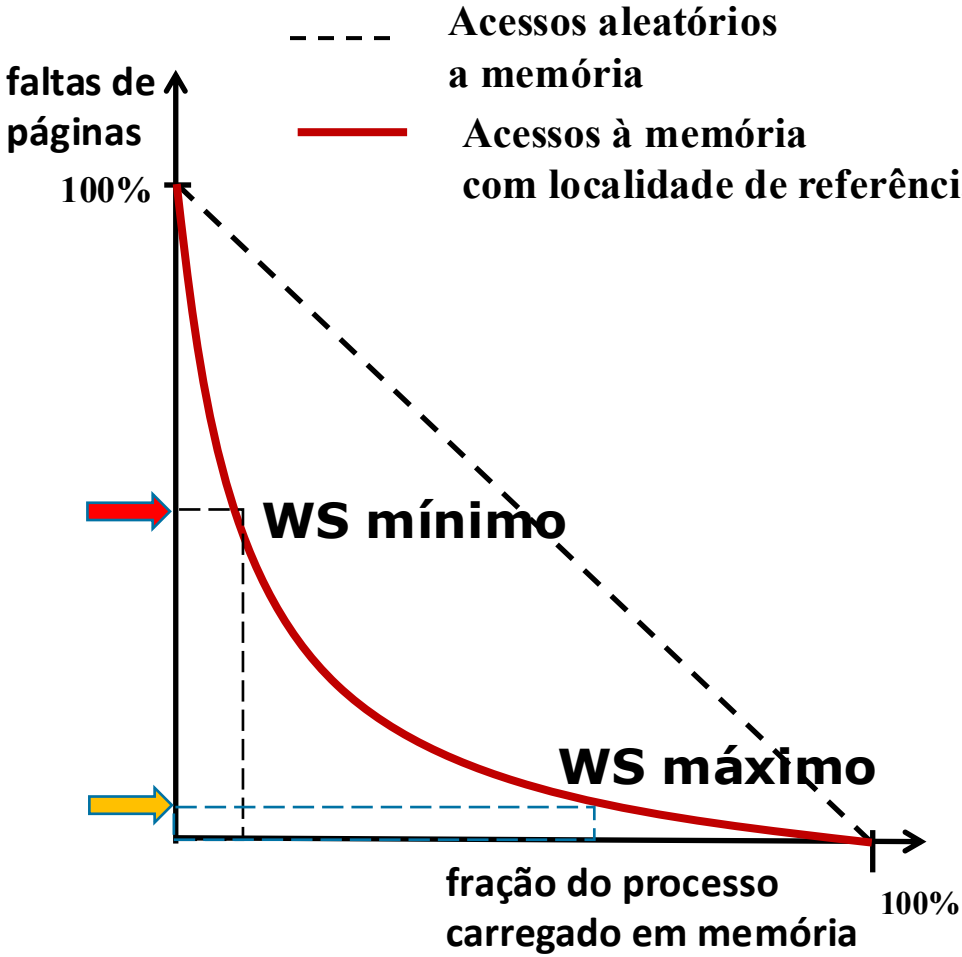
Espaços de Trabalho evitam o *Thrashing*

- Um processo só é colocado em memória primária se existir um número mínimo de páginas livres (**espaço de trabalho mínimo**)
- Quando o processo arranca são carregadas algumas páginas em antecipação, até se atingir o espaço de trabalho mínimo.



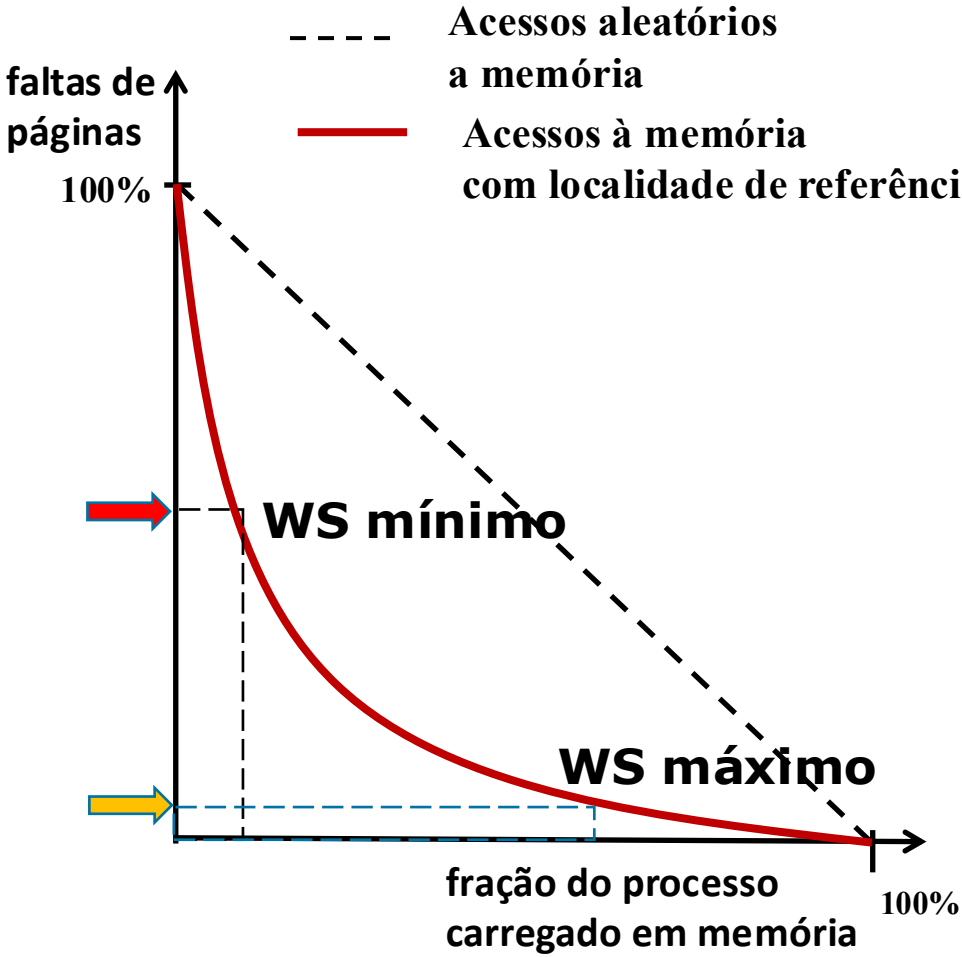
Espaços de Trabalho evitam o *Thrashing*

- A partir daí o número de páginas em memória (**espaço de trabalho residente**) pode crescer até um valor máximo, retirando páginas da lista de páginas livres.



Espaços de Trabalho evitam o *Thrashing*

- Quando atinge o **espaço de trabalho máximo** o processo começa a paginar contra si mesmo:
  - para carregar uma nova página tem que libertar uma das suas ("passa a retirar páginas a si próprio em vez da lista de páginas livres global")



Espaços de Trabalho (cont. II)

- Se estimarmos que o espaço de trabalho de um processo é W, podemos
  - Evitar colocar o processo em execução enquanto não houver pelo menos suficientes páginas livres em RAM
  - Limitar o número de páginas do processo em RAM

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

Algoritmos de Substituição

(analisaremos apenas soluções usadas com paginação)

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

- FIFO:
  - Associar a cada PTE um timestamp de quando esta foi colocada em RAM
  - Muito eficiente mas não atende ao grau de utilização das páginas
    - Apenas ao seu tempo de permanência em memória primária

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:
  - Bit R na tabela de páginas
    - colocado a 1 pela UGM quando página é acedida
  - Gestor de memória do núcleo mantém um contador por página que indica a que "grupo etário" ela pertence
    - Actualizado regularmente pelo paginador
    - Quando R=0, grupo etário incrementa
    - Quando R=1, volta ao grupo etário inicial
    - Bit R recolocado a 0
  - Quando atingir um grupo etário 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):
  - Bits R e M mantidos na tabela de páginas
  - UGM automaticamente coloca R=1 quando há leitura, M=1 quando há escrita
  - O paginador percorre regularmente as tabelas de páginas e coloca o bit R a 0
  - Páginas ordenadas 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
  - libertam-se primeiro as páginas dos grupos de número mais baixo

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, p.e., alocação de segmentos na memória fisica
    - 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, p.e., instruções precisam de ser recomeçãveis
    - 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

Anexos

Fora da matéria

Critérios de Escolha de Blocos Livres: Algoritmo Buddy

- Procura um bom equilíbrio entre o tempo de procura e a fragmentação interna e externa
- Pedidos de alocação satisfeitos usando blocos de dimensão **fixa** b^i, i=[min, max]
  - Permite fragmentação interna
- Subdivide recursivamente os blocos livres até:
  - Obter um bloco de tamanho mínimo para satisfazer o pedido de alocação
  - Ou atingir o tamanho mínimo possível para os blocos alocados (b^min)
  - Alocação e libertação de blocos têm custo logarítmico

Critérios de Escolha de Blocos Livres: Algoritmo Buddy

- A memória livre é dividida em blocos de dimensão b^n
  - 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 necessário 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

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)

UNIX
Gestão de Memória

Unix - Gestão de Memória

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

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

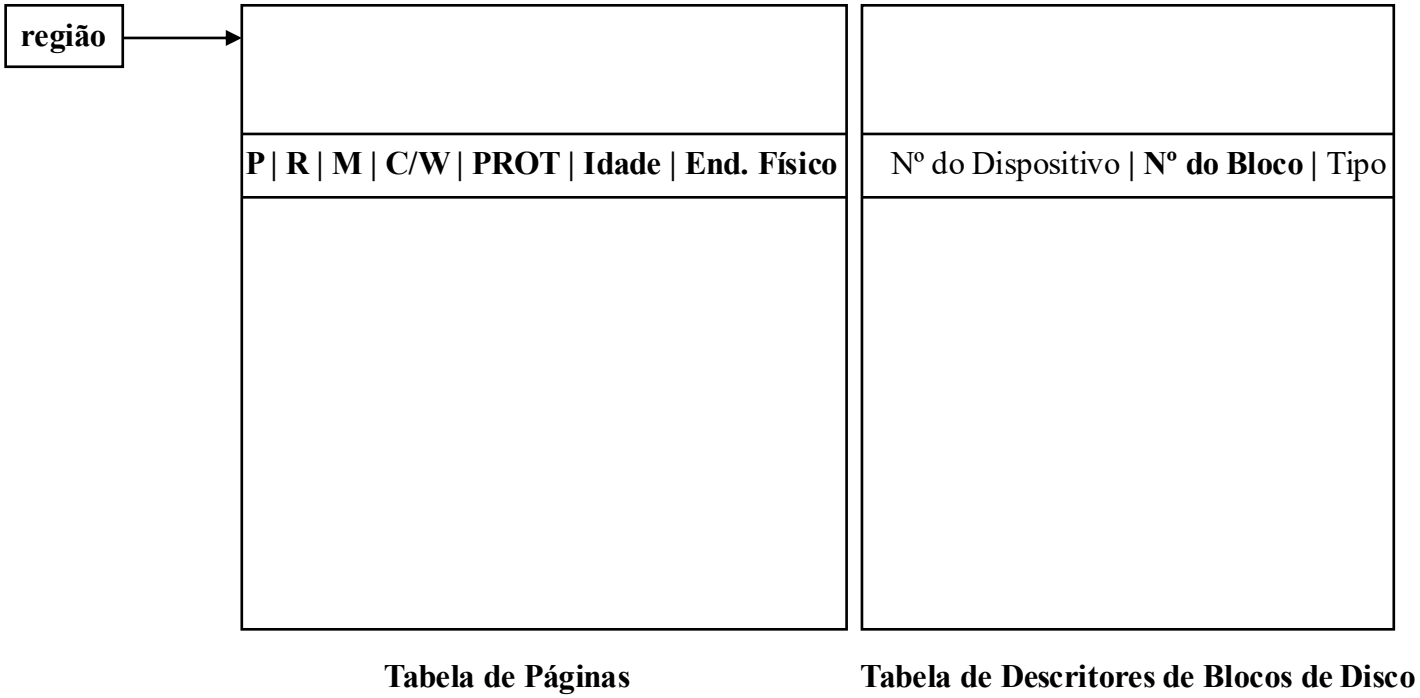


Tabela pfdata

- 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

Significado dos campos das tabelas

- 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

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