SISTEMAS OPERACIONAIS

Prof. Me. Napoleão Póvoa Ribeiro Filho

UNITINS

ESTRATÉGIAS DE ALOCAÇÃO

Para minimizar a ocorrência de fragmentação externa, cada pedido de alocação pode ser analisado para encontrar a área de memória livre que melhor o atenda.
Essa análise pode ser feita usando um dos seguintes critérios:

First-fit (primeiro encaixe): consiste em escolher a primeira área livre que satisfaça o pedido de alocação; tem como vantagem a rapidez, sobretudo se a lista de áreas livres for muito longa

ESTRATÉGIAS DE ALOCAÇÃO

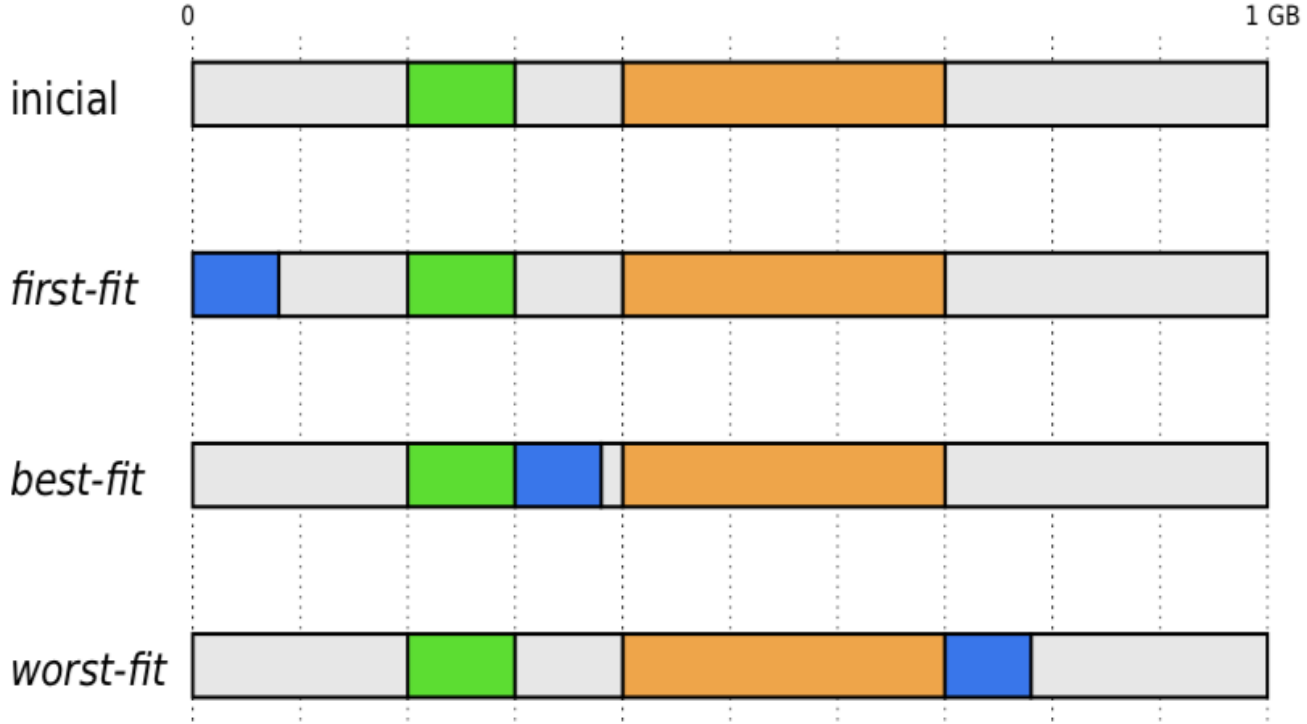
- **Best-fit** (melhor encaixe): consiste em escolher a menor área possível que possa receber a alocação, minimizando o desperdício de memória. Contudo, algumas áreas livres podem ficar pequenas demais e com isso se tornarem inúteis.

- **Worst-fit** (pior encaixe): consiste em escolher sempre a maior área livre possível, de forma que a “sobra” seja grande o suficiente para ser usada em outras alocações.

ESTRATÉGIAS DE ALOCAÇÃO

- **Next-fit** (próximo encaixe): variante da estratégia first-fit que consiste em percorrer a lista de áreas a partir da última área alocada ou liberada, para que o uso das áreas livres seja distribuído de forma mais homogênea no espaço de memória.

ESTRATÉGIAS DE ALOCAÇÃO



UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALOCADOR BUDDY

- A estratégia Buddy sempre aloca blocos de memória de tamanho 2^n, com n inteiro e ajustável.

- Por exemplo, para uma requisição de 85 KBytes será alocado um bloco de memória com 128 KBytes (2^7KBytes), e assim por diante.

- O uso de blocos de tamanho 2^n reduz a fragmentação externa, mas pode gerar muita fragmentação interna.

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALOCADOR BUDDY

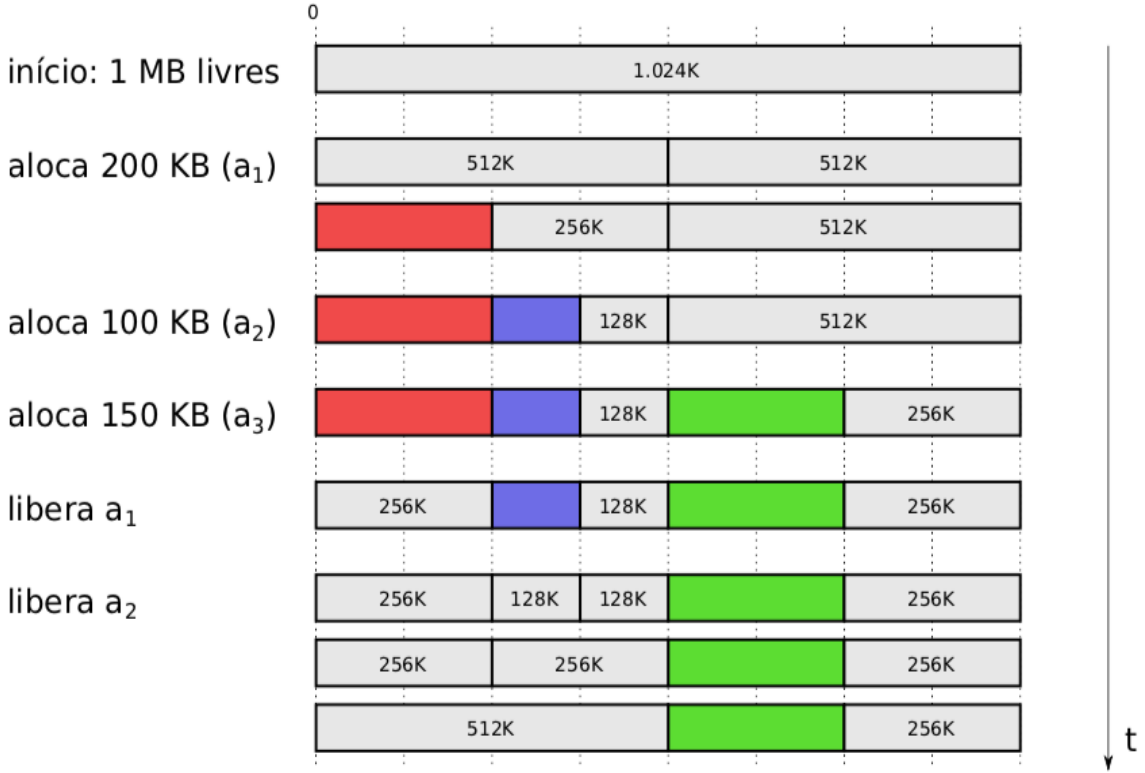
- Ao receber uma requisição de alocação de memória de tamanho 40 KBytes (por exemplo), o alocador procura um bloco livre com 64 KBytes (pois 64 KBytes é o menor bloco com tamanho 2n que pode conter 40 KBytes).

- Caso não encontre um bloco com 64 KBytes, procura um bloco livre com 128 KBytes, o divide em dois blocos de 64 KBytes (os buddies) e usa um deles para a alocação.

ALOCADOR BUDDY

- Caso não encontre um bloco livre com 128 KBytes, procura um bloco com 256 KBytes para dividir em dois, e assim sucessivamente.

ALOCADOR BUDDY



ALOCADOR BUDDY

- O alocador Buddy é usado em vários sistemas. Por exemplo, no núcleo Linux ele é usado para a alocação de memória física (page frames), entregando áreas de memória RAM para a criação de processos, para o alocador de objetos do núcleo e para outros subsistemas.

- O arquivo /proc/buddyinfo permite consultar informações das alocações existentes

ALOCADOR SLAB

- É especializado na alocação de “**objetos de núcleo**”, ou seja, as pequenas estruturas de dados que são usadas para representar descritores de processos, de arquivos abertos, *sockets* de rede, *pipes*, etc.

- Esses objetos de núcleo são continuamente criados e destruídos durante a operação do sistema, são pequenos (dezenas ou centenas de bytes) e têm tamanhos relativamente padronizados.

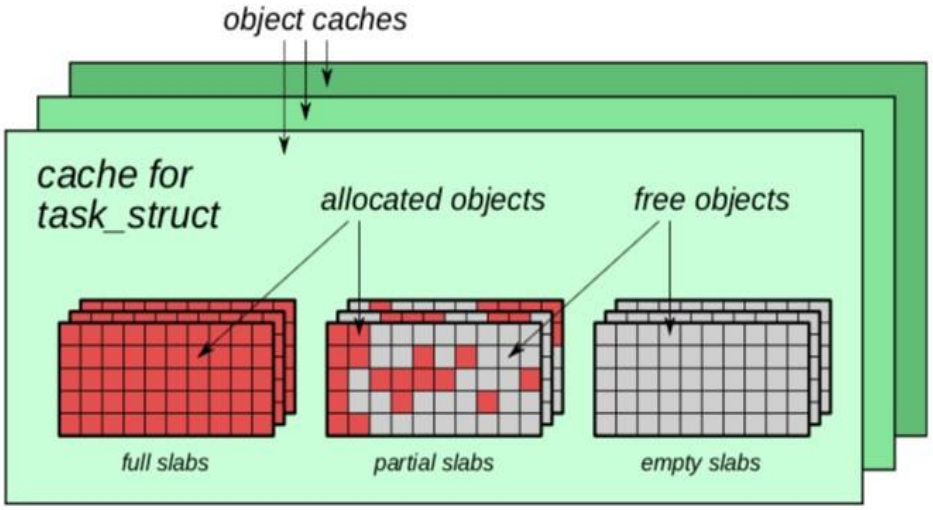
ALOCADOR SLAB

- O alocador Slab usa uma estratégia baseada no *caching* de objetos. É definido um *cache* para cada tipo de objeto usado pelo núcleo: descritor de processo, de arquivo, de *socket*, etc.

- Cada cache é então dividido em *slabs* (placas) que contêm objetos daquele tipo, portanto todos com o mesmo tamanho.

ALOCADOR SLAB

- Um slab pode estar **cheio**, quando todos os seus objetos estão em uso, **vazio**, quando todos os seus objetos estão livres, ou **parcial**.



UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALOCADOR SLAB

- O alocador Slab é usado para a gestão de objetos de núcleo em muitos sistemas operacionais, como Linux, Solaris, FreeBSD e Horizon (usado no console Nintendo Switch).

- No Linux, contadores dos *slabs* em uso podem ser consultados no arquivo **/proc/slabinfo.**

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

EXTENDENDO A MEMÓRIA RAM

Existem diversas técnicas para usar um espaço de armazenamento secundário como extensão da memória RAM, com ou sem o auxílio do hardware. As mais conhecidas são:

Overlays: o programador organiza seu programa em módulos que serão carregados em uma mesma região de memória em momentos distintos. Esses módulos, chamados de *overlays*, são gerenciados através de uma biblioteca específica.

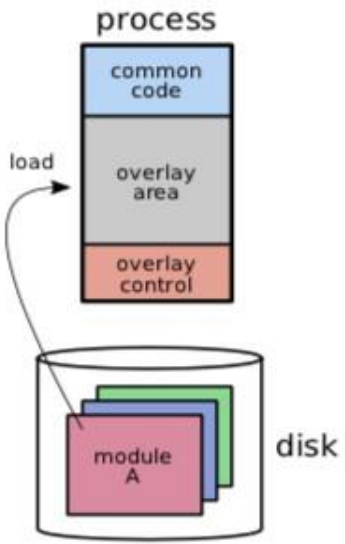
EXTENDENDO A MEMÓRIA RAM

Swapping: consiste em mover um processo ocioso da memória RAM para um disco (*swap-out*), liberando a memória para outros processos. Mais tarde, quando esse processo for acordado, ele é carregado de volta na memória (*swap-in*). A técnica de *swapping* foi muito usada até os anos 1990, mas hoje é pouco empregada em sistemas operacionais de uso geral.

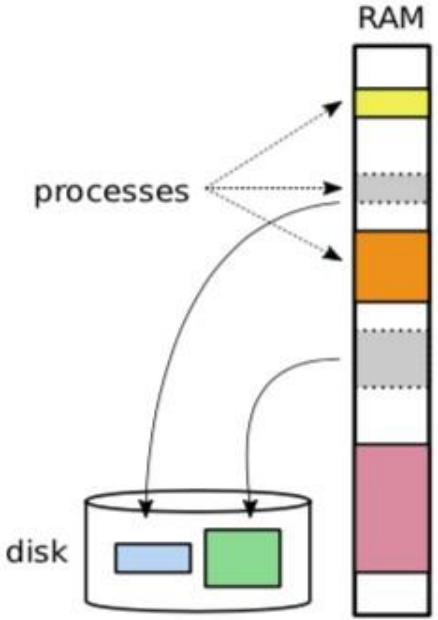
EXTENDENDO A MEMÓRIA RAM

Paging(paginação): consiste em mover páginas individuais, conjuntos de páginas ou mesmo segmentos da memória para o disco (*page-out*). Se o processo tentar acessar uma dessas páginas mais tarde e ela não estiver na memória RAM, a MMU gera uma interrupção de falta de página e o núcleo do SO recarrega a página faltante na memória (*page-in*). Esta é a técnica mais usada nos sistemas operacionais atuais, por sua flexibilidade, rapidez e eficiência.

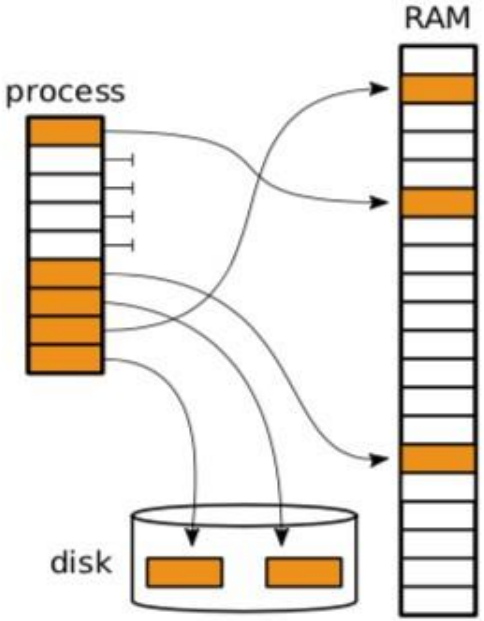
GERÊNCIA DE MEMÓRIA



overlay



swapping



paging

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

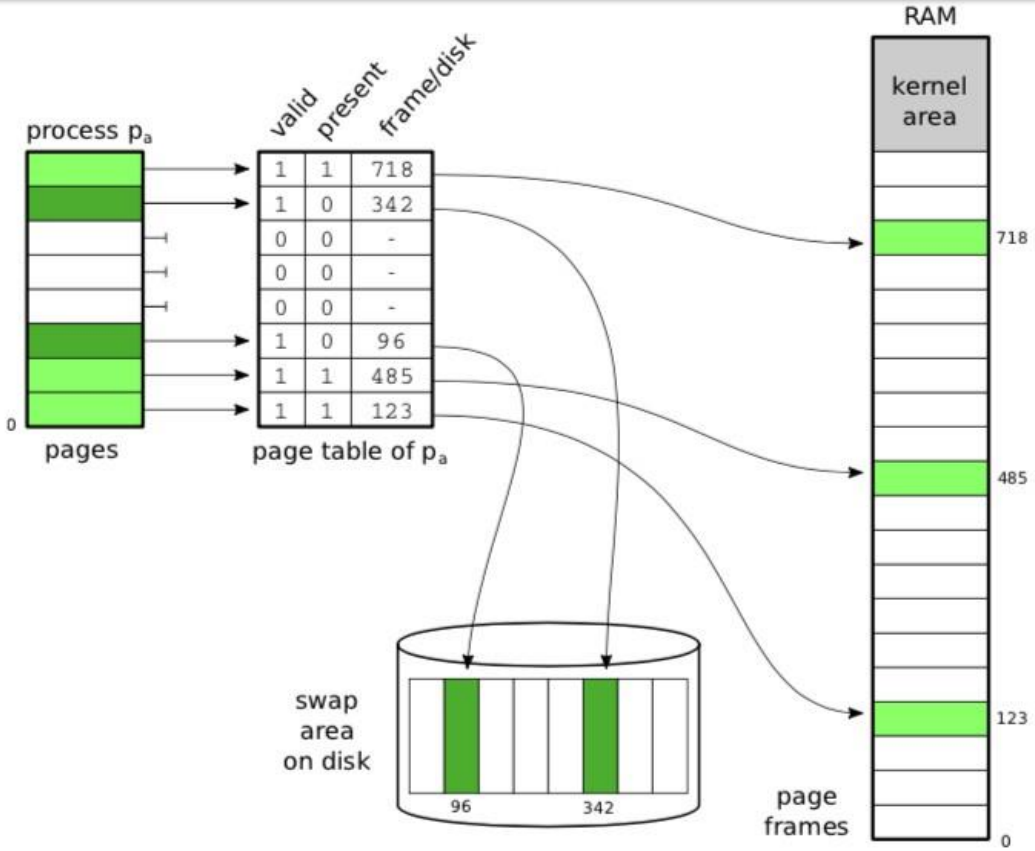
PAGINAÇÃO

A transferência de páginas entre a memória e o disco é realizada pelo núcleo do sistema operacional. As páginas a serem retiradas da memória são escolhidas por ele, de acordo com algoritmos de substituição de páginas.

Quando um processo tentar acessar uma página que está em disco, o núcleo é alertado pela MMU e traz a página de volta à memória para poder ser acessada.

Para cada página transferida para o disco, a tabela de páginas do processo é ajustada: o *flag* de presença da página em RAM é desligado e a posição da página no disco é registrada, ao invés do quadro.

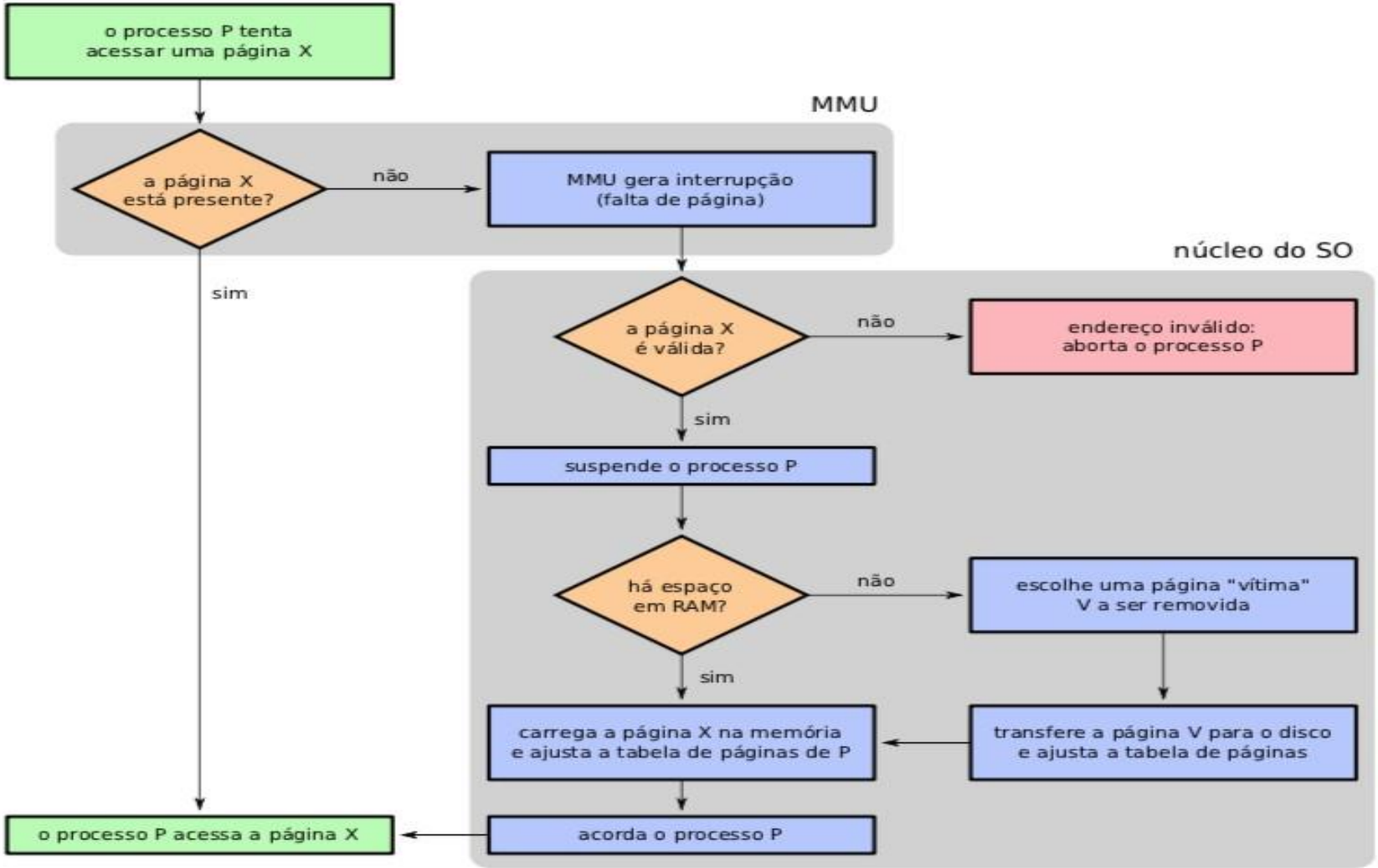
PAGINAÇÃO


RAM
kernel area
process p_a
valid
present
frame/disk
1 1 718
1 0 342
0 0 -
0 0 -
0 0 -
1 0 96
1 1 485
1 1 123
0
pages
page table of p_a
718
485
123
0
swap area on disk
96
342
page frames

PAGINAÇÃO

O armazenamento externo das páginas pode ser feito em um **disco exclusivo** (usual em servidores de maior porte), em **uma partição do disco principal** (usual no Linux e outros UNIX) ou em **um arquivo reservado** dentro do sistema de arquivos (como no Windows NT e sucessores).

Em alguns sistemas, é possível **usar uma área de troca remota**, em um servidor de rede; todavia, essa solução apresenta baixo desempenho.


o processo P tenta acessar uma página X
MMU
a página X está presente?
não
MMU gera interrupção (falta de página)
sim
núcleo do SO
a página X é válida?
não
endereço inválido: aborta o processo P
sim
suspende o processo P
há espaço em RAM?
não
escolhe uma página "vítima" V a ser removida
sim
carrega a página X na memória e ajusta a tabela de páginas de P
transfere a página V para o disco e ajusta a tabela de páginas
acorda o processo P
o processo P acessa a página X

PAGINAÇÃO

Caso a memória principal já esteja cheia, uma página deverá ser movida para o disco antes de trazer de volta a página faltante. Isso implica em mais operações de leitura e escrita no disco e portanto em mais demora para atender o pedido do processo.

Muitos sistemas, como o Linux e o Solaris, evitam essa situação mantendo um *daemon* responsável por escolher e transferir páginas para o disco, sempre que a quantidade de memória livre estiver abaixo de um certo limiar.

ALGORITMOS DE SUBSTITUIÇÃO DE PÁGINA

- Existem vários algoritmos para a escolha de páginas a substituir na memória, visando reduzir a frequência de falta de páginas, que levam em conta alguns dos fatores acima enumerados.

- A escolha correta das páginas a retirar da memória física é um fator essencial para a eficiência do mecanismo de paginação. Más escolhas poderão remover da memória páginas muito usadas, aumentando a taxa de faltas de página e diminuindo o desempenho do sistema.

ALGORTIMO FIFO

- Nessa estratégia, as páginas mais antigas podem ser removidas para dar lugar a novas páginas. Os números das páginas recém carregadas na memória são registrados no final da lista, enquanto os números das próximas páginas a substituir na memória são obtidos no início da lista.

- Seu principal defeito é considerar somente a idade da página, sem levar em conta sua importância**.** Páginas carregadas na memória há muito tempo podem estar sendo frequentemente acessadas, como é o caso de áreas de memória contendo bibliotecas dinâmicas compartilhadas por muitos processos, ou páginas de processos servidores lançados durante a inicialização (*boot*) da máquina.

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

t	página acessada	quadros q_o	q_1	q_2	falta de página?	ação realizada
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (carregada em $t = 1$)
5	0	2	0	1		p_0 já está na memória
6	3	2	3	1	✓	p_3 substitui p_0
7	0	2	3	0	✓	p_0 substitui p_1
8	4	4	3	0	✓	p_4 substitui p_2
9	2	4	2	0	✓	p_2 substitui p_3
10	3	4	2	3	✓	p_3 substitui p_0
11	0	0	2	3	✓	p_0 substitui p_4
12	3	0	2	3		p_3 já está na memória
13	2	0	2	3		p_2 já está na memória
14	1	0	1	3	✓	p_1 substitui p_2
15	2	0	1	2	✓	p_2 substitui p_3
16	0	0	1	2		p_0 já está na memória
17	1	0	1	2		p_1 já está na memória
18	7	7	1	2	✓	p_7 substitui p_0
19	0	7	0	2	✓	p_0 substitui p_1
20	1	7	0	1	✓	p_1 substitui p_2

ALGORITMO LRU (Least Recently Used)

Neste algoritmo, a escolha recai sobre as páginas que estão na memória há mais tempo sem ser acessadas. Assim, páginas antigas e menos usadas são as escolhas preferenciais. Páginas antigas mas de uso frequente não são penalizadas por este algoritmo, ao contrário do que ocorre no algoritmo FIFO.

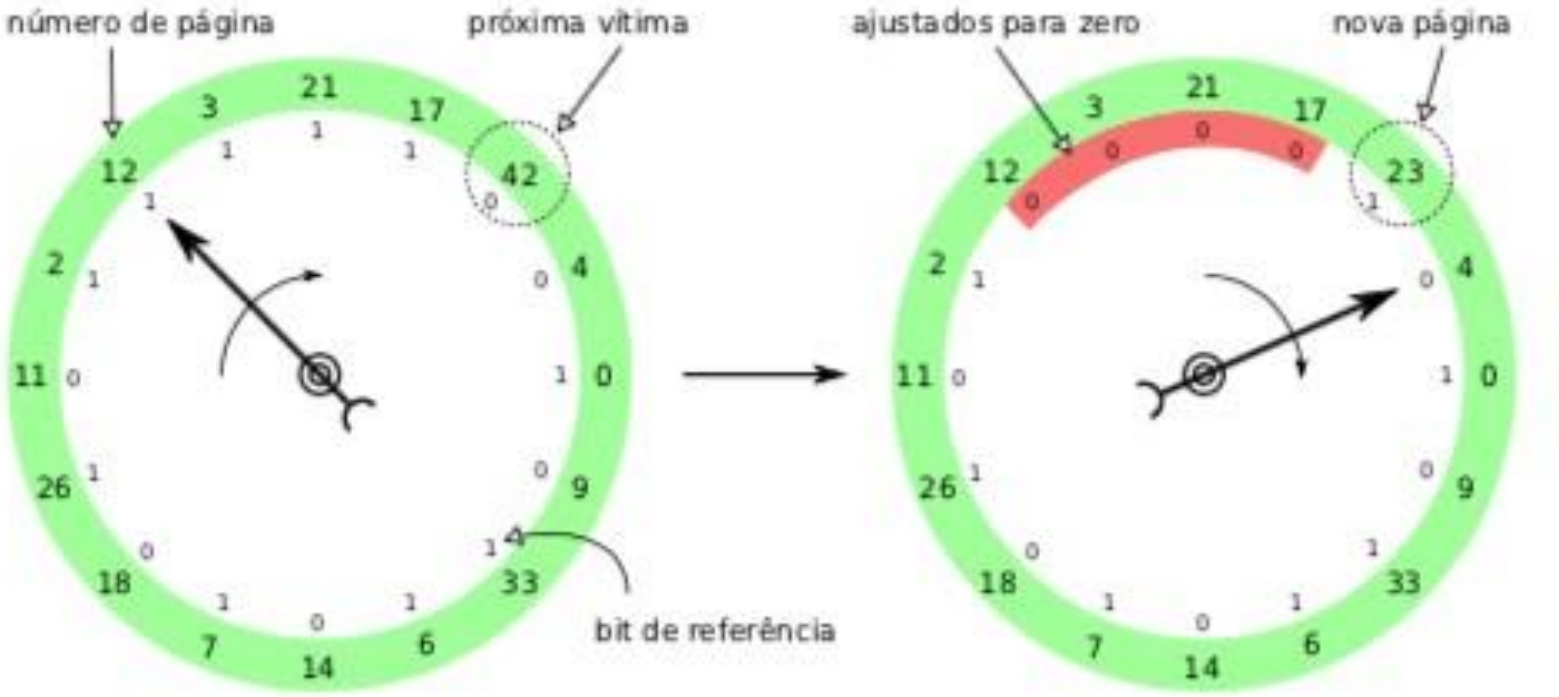
t	página acessada	quadros q_0	q_1	q_2	falta de página?	ação realizada
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (há mais tempo sem acesso)
5	0	2	0	1		p_0 já está na memória
6	3	2	0	3	✓	p_3 substitui p_1
7	0	2	0	3		p_0 já está na memória
8	4	4	0	3	✓	p_4 substitui p_2
9	2	4	0	2	✓	p_2 substitui p_3
10	3	4	3	2	✓	p_3 substitui p_0
11	0	0	3	2	✓	p_0 substitui p_4
12	3	0	3	2		p_3 já está na memória
13	2	0	3	2		p_2 já está na memória
14	1	1	3	2	✓	p_1 substitui p_0
15	2	1	3	2		p_2 já está na memória
16	0	1	0	2	✓	p_0 substitui p_3
17	1	1	0	2		p_1 já está na memória
18	7	1	0	7	✓	p_7 substitui p_2
19	0	1	0	7		p_0 já está na memória
20	1	1	0	7		p_1 já está na memória

ALGORITMO SEGUNDA CHANCE

É um melhoramento do FIFO, onde o bit de referência de cada página candidata é analisado para verificar se houve um acesso recente. Caso tenha sido, essa página recebe uma “segunda chance”, voltando para o fim da fila com seu bit de referência ajustado para zero. Dessa forma, evita-se substituir páginas antigas mas muito acessadas.

Caso todas as páginas sejam muito acessadas, o algoritmo vai varrer todas as páginas, ajustar todos os bits de referência para zero e acabará por escolher a primeira página da fila, como faria o algoritmo FIFO.

ALGORITMO SEGUNDA CHANCE



ALGORITMO NRU (Not Recentry Used)

- O algoritmo da segunda chance leva em conta somente o bit de referência de cada página ao escolher as vítimas para substituição.

- O algoritmo NRU (*Not Recently Used*) melhora essa escolha, ao considerar também o bit de modificação, que indica se o conteúdo de uma página foi modificado após ela ter sido carregada na memória.

ALGORITMO NRU

- O Usando os bits *R* (referência) e *M* (modificação), é possível classificar as páginas em memória em quatro níveis de importância:

- 00 (*R* = 0, *M* = 0): páginas que não foram referenciadas recentemente e cujo conteúdo não foi modificado. São as melhores candidatas à substituição, pois podem ser simplesmente retiradas da memória.

- 01 (*R* = 0, *M* = 1): páginas que não foram referenciadas recentemente, mas cujo conteúdo já foi modificado. Não são escolhas tão boas, porque terão de ser gravadas na área de troca antes de serem substituídas.

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALGORITMO NRU

- 10(*R*=1,*M*=0):páginas referenciadas recentemente, cujo conteúdo permanece inalterado. São provavelmente páginas de código que estão sendo usadas ativamente e serão referenciadas novamente em breve.

- 11 (*R* = 1, *M* = 1): páginas referenciadas recentemente e cujo conteúdo foi modificado. São a pior escolha, porque terão de ser gravadas na área de troca e provavelmente serão necessárias em breve.

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALGORITMO DO ENVELHECIMENTO

- A cada página é associado um contador inteiro com *N* bits (geralmente 8 bits são suficientes). Periodicamente, o algoritmo varre as tabelas de páginas, lê os bits de referência e agrega seus valores aos contadores de acessos das respectivas páginas.

- Uma vez lidos, os bits de referência são ajustados para zero, para registrar as referências de páginas que ocorrerão durante próximo período.

ALGORITMO DO ENVELHECIMENTO

- O valor lido de cada bit de referência não deve ser simplesmente somado ao contador, por duas razões:
- o contador chegaria rapidamente ao seu valor máximo (*overflow*)

- a simples soma não permitiria diferenciar acessos recentes dos mais antigos.

- Por isso, cada contador é deslocado para a direita 1 bit, descartando o bit menos significativo (LSB - *Least Significant Bit*).

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS

ALGORITMO DO ENVELHECIMENTO

- Em seguida, o valor do bit de referência é colocado na primeira posição à esquerda do contador, ou seja, em seu bit mais significativo (MSB - *Most Significant Bit*).

- Dessa forma, acessos mais recentes têm um peso maior que acessos mais antigos, e o contador nunca ultrapassa seu valor máximo.

ALGORITMO DO ENVELHECIMENTO

	R		*contadores*	
p_1	0		0000 0011	(3)
p_2	1	com	0011 1101	(61)
p_3	0		1010 1000	(168)
p_4	1		1110 0011	(227)

$\Longrightarrow$

	R		*contadores*	
p_1	0		**0**000 0001	(1)
p_2	0	e	**1**001 1110	(158)
p_3	0		**0**101 0100	(84)
p_4	0		**1**111 0001	(241)

UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS


UNITINS
UNIVERSIDADE ESTADUAL DO TOCANTINS