



Resumo 3^a Prova - SO



Revisão - Processos

Um processo muda de estado conforme interage com recursos (CPU, I/O).

Estado	Descrição
Novo	Processo está sendo criado.
Pronto	Aguardando para usar a CPU.
Execução	Instruções estão sendo processadas pela CPU.
Espera	Bloqueado por I/O ou evento externo (ex: esperando dados do disco).
Encerrado	Finalizado (voluntariamente ou por erro).

* Escalonamentos de Processos: decide qual processo deve usar a CPU e por quanto tempo.

> FCFS = First Come, First Served

> RR = Round Robin - todos processos têm um limite de tempo

> Prioridade = De acordo com a Prioridade

(o quantum)

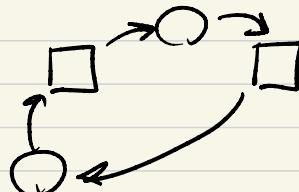
> SJF = Shortest Job First

Turnaround = Completion Time - Arrival Time

Turnaround Médio = $\frac{\text{Soma dos Turnarounds}}{\text{n. de processos}}$

* Mais detalhes no resumo da 1ª prova!

► Deadlock = Exclusão mútua, espera circular, não preemptivo, põe e espera.



Revisão - Gerência de Memória

(um processo enxerga endereços virtuais (lógicos), gerados pelo CPU)

* **MMU** → traduz endereçamentos lógicos em físicos usando um registrador de realocação
ex: registrador = 14000 } endereço físico = 14396
end. lógico = 346 }

- Swapping → Quando a memória principal (^memória) está cheia, processos são temporariamente movidos p/ o disco (out = p/a disco / in = p/a memória)

• Alocacão Contígua de Memória → memória é dividida em partições seguradas p/ processos

- First-fit (1º bloco que couber)
- Best-Fit (menor bloco)
- Worst-Fit (maior bloco)

↳ fragmentação:



- * - externa → bloco cai em espacos livres não segurais (buracos)
- interna → espaço desperdiçado dentro de uma partição alocada

* **Paginação** → divide a memória em páginas (lógicas) e quadros (físicos) de tam. fixo.

- elimina fragmentação externa
- permite alocação não contígua.
- a **TLB** (Tabela de páginas) mapeia os páginas p/ quadros → Translation Lookaside Buffer = cache de traduções recentes de endereços p/ acelerar o acesso

- * Se a pg estiver na TLB (hit) → tradução instantânea
- * Se miss → consultar a tabela de páginas da memória

* A compactação pode resolver o problema da fragmentação, reorganizando arquivos no disco p/ consolidar espaços livres em blocos contíguos.

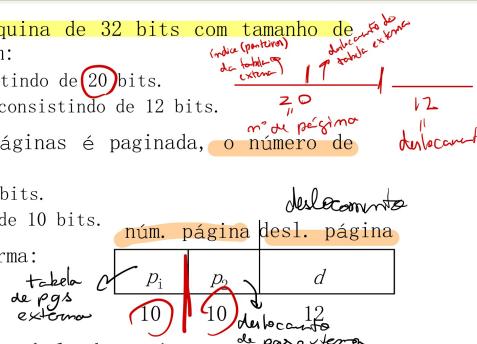
$$\text{Tempo de acesso efectivo} \rightarrow (\text{taxa de acerto} \cdot \text{tempo com acerto}) + (\text{taxa de miss} \cdot \text{tempo com miss})$$

↳ acesso à TLB
+
qntd de níveis · acesso à memória
+
último acesso à memória

* Paginação Multinível

2 níveis:

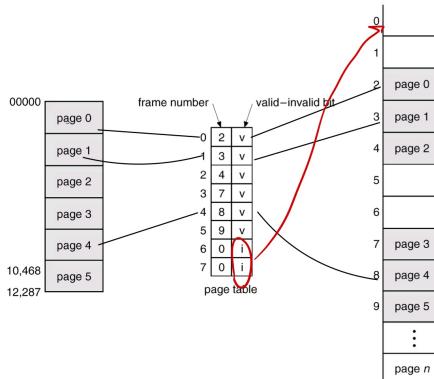
- Um endereço lógico (em máquina de 32 bits com tamanho de página de 4K) é dividido em:
 - um número de página consistindo de 20 bits.
 - Um deslocamento de página consistindo de 12 bits.
- Uma vez que a tabela de páginas é paginada, o número de página é dividido em:
 - um número de página de 10 bits.
 - Um deslocamento de página de 10 bits.
- Um endereço lógico é da forma:



* Bits de Validação:

1 = A página está no local associado

2 = A página não está no espaço de endereçamento lógico do processo



8. Páginas Compartilhadas

Intuição:

Permitir que processos compartilhem código (ex: bibliotecas).

- Código compartilhado: Só leitura (evita inconsistências).
- Dados privados: Cada processo tem sua própria cópia.

Exemplo:

Vários processos usando a mesma biblioteca `libc.so` não precisam de cópias separadas na memória.

Memória Virtual

→ É uma separação da memória lógica da memória física que permite executar programas que necessitam de + memória do que a física disponível. É gerenciada pelo SO.

da memoria principal!

→ ela pode ser implementada por:

- paginacão sob demanda
- segmentação sob demanda

→ erros de "simulação" da memória real.

* paginacão sob demanda

- traz uma página à memória só quando necessário
- página necessária → referência a ela; referência inválida → abortar; fora de memória → trazer à memória.

→ indicado pelo bit válido (1) / inválido (0)

- se o endereço não está na memória (bit inválido) = Page Fault

Depois de um Page Fault:

- ♦ S.O. olha em outra tabela para decidir:
 - Referência inválida ⇒ abortar.
 - Apenas não está na memória.
- ♦ Selecionar quadro vazio.
- ♦ Trazer página para o quadro.
- ♦ Alterar tabelas, bit de validação = 1.
- ♦ Instrução de recomeço

→ se não tiver quadros livres p/ trazer o dado ⇒ algoritmo p/ trocar a página por ela (uma que não esteja sendo usada, etc)

Performance de Paginação sob Demanda

- ♦ Taxa de falta de página $0 \leq p \leq 1.0$

- se $p = 0$ não há falta de páginas
- se $p = 1$, toda referência é uma falta

- ♦ Tempo de Acesso Efetivo (EAT)

$$EAT = (1 - p) \times \text{acesso à memória}$$

+ p (overhead de falta de página)

+ [tirar a página]

+ trazer nova página

+ overhead de reinício)

$$EAT = (1 - p) \cdot \text{acesso à memória} + p \cdot \text{overhead de falta de página})$$

* Usando um bit de modificação → apenas páginas modificadas são escritas em disco.

* Algoritmos de substituição de página
- objetivo: reduzir o Page Fault

↳ FIFO

- Seqüência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 quadros (3 páginas na memória por processo em determinado instante)

1	1	4	5
2	2	1	3
3	3	2	4

9 faltas



Substituição + erros iniciais

Anomalia de Belady:

+quadros, +faltas

↳ LRU

Least Recently Used

- Seqüência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	5
2	
3	4
4	3

Sempre que a página for referenciada, coloca contador de clock, bit de referência ou pilha.

↳ Second Chance

• É uma modificação do FIFO que considera o bit de referência (R)

* antes de substituir, verifica o bit R da pg mais antiga

- se $R=0 \rightarrow$ substitui a pg.

- se $R=1 \rightarrow$ zero o bit R, move a pg p/ o fim da fila e continua buscando até encontrar uma pg c/ $R=0$.

♦ Segunda chance

- Precisa do bit de referência – fila circular.
- Se a página a ser substituída (sentido horário) tem bit=1, então:
 - atribuir 0 ao bit.
 - Deixar a página na memória.
 - Substituir a próxima página (sentido horário), sujeito às mesmas regras.

* bit de referência = indica se a pg foi acessada recentemente

↳ NRU (Not Recently Used)

* Usa dois bits de controle:

- Bit de referência (R)
- Bit de modificação (M) = Dirty Bit (indica se a pg foi modificada após ser carregada na memória)

Classificação por Classes (R, M)

O algoritmo NRU (Not Recently Used) combina os bits R e M criando 4 classes de prioridade :

Classe	R	M	Descrição	Prioridade
Classe 0	0	0	Não usada recentemente, nem modificada	MAIS ALTA
Classe 1	0	1	Não usada recentemente, mas modificada	ALTA
Classe 2	1	0	Usada recentemente, porém não modificada	BAIXA
Classe 3	1	1	Usada e modificada recentemente	MAIS BAIXA

Algoritmos de Contagem

- ♦ Manter um contador do número de referências feitas a cada página.
- ♦ Algoritmo LFU: substituir página com menor contador.
- ♦ Algoritmo MFU: baseia-se no argumento de que a página com menor contador provavelmente acabou de ser trazida à memória e ainda será usada.

Alocacão de Quadros

* Cada processo tem um n° máx fixo de frames (quadros), definido na criação do processo.

fixa
alocacão igualitária (ex: n 100 quadros
e 5 processos → 20 pgs p/ cada).

prioritária
alocacão proporcional c/
prioridades no lugar do
tamanho

proporcional
aloca proporcionalmente as tamanhos
 $A_i = \left(\frac{S_i}{\sum S_i}\right) \cdot q$
tamanho do processo i
soma dos tamanhos
nº total de quadros

se o Processo Pi gera Page Fault:
substitua 1 de seus quadros
substitua 1 quadro de um
processo com menor prioridade

→ Pode remover pg.
de qualquer processo.

- ♦ Substituição Global – processo seleciona um quadro de substituição do conjunto de quadros; um processo pode tomar um quadro de outro.
- ♦ Substituição Local – cada processo seleciona um quadro dos seus próprios.

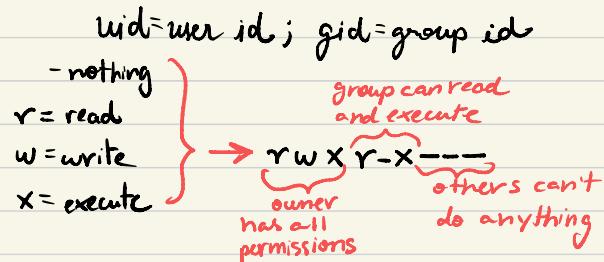
remove página de
próprio processo

- ♦ Se um processo não tem páginas suficientes, a taxa de falta é muito alta. Isto leva a:
 - baixa utilização de CPU.
 - S.O. pensa que precisa aumentar o grau de multiprogramação.
 - Outro processo adicionado ao sistema.
- ♦ Thrashing ≡ um processo mais ocupado movendo páginas que executando

Arquivos

* modo de permissão:

---|---|---
owner group others



* dono \Rightarrow uid = 12 e gid = 1

* usuário \Rightarrow uid = 6 e gid = 1 \Rightarrow ele não é o dono, mas está no grupo, então pode ler e executar o arquivo.



* Alocacão Contígua \rightarrow blocos sequenciais / fragmentacão externa \uparrow / arquivos não podem crescer / posição inicial e tamanho do bloco são necessários.

* Encodada \rightarrow cada arquivo é uma lista encadeada de blocos no disco, espalhados em qualquer posição / Alocado conforme necessidade / Ele precisa de endereço inicial / Sem acesso randômico / FAT

* Indexada \rightarrow reúne todos os ponteiros em um bloco / tabela de índice / Acesso randômico e direcional sem fragmentacão externa, mas com overhead p/ indexação

funciona como um "mapa do disco"

FAT = File Allocation Table \rightarrow tabela usada pelo SO para gerenciar o espaço de armazenamento em discos, registrando quais blocos estão ocupados, livres ou pertencem a qual arquivo

Na alocação encodada com FAT, armazenamos os ponteiros na tabela FAT

b-Node = estrutura de dados fundamental em sistemas de arquivos baseados em UNIX, armazenando metadados de cada arquivo (uid, gid, modificacão e acesso, etc).

A alocação indexada utiliza uma estrutura de dados chamada i-node que ocupa normalmente um espaço menor do que a FAT (File Allocation Table) na memória principal

* Tipos de Arquivos

- ↳ de dados (numéricos, caracteres, binário)
- ↳ de programas

* Estrutura de Arquivos

- ↳ menúme (sequência de bytes → como .txt)
- ↳ simples (linhas, tamanho fixo, tamanho variável)
- ↳ complexa (documentos formatados, arquivo de carga relocalizável)

* o SO e o programa decidem qual diretório, qual tipo de arquivo pode manipular e qual informação pode colocar no disco

↳ **atributos**: mantidos na estrutura de diretórios, residente em

- nome = única info mantida legível p/ o usuário
- tipo = qual extensão de aplicativo p/ abrir
- localização = apontador p/ a posição do arquivo no dispositivo
- tamanho = quanto de espaço converte o arquivo vai ocupar
- proteção = controle de quem pode ler, escrever ou exec.
- data e id de usuário = dados p/ proteção e segurança

menor
metade
entre

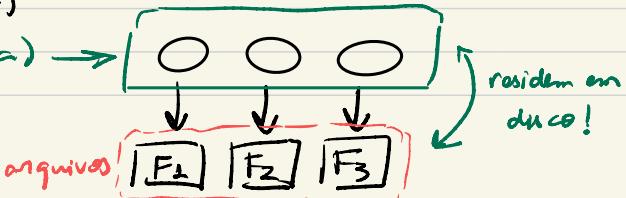
* Operações com Arquivos

- ↳ criar, gravar, ler, reposicionar o ponteiro (seek), apagar, truncar
- ↳ obter (Fi) - percorrer a estrutura de diretório no disco pela entrada de Fi, e mover o conteúdo da entrada p/ a memória
- ↳ fechar (Fi) - mover o conteúdo da entrada Fi da memória p/ o disco. ↗ arquivo Fi

* Estrutura de Diretório → coleção de más informações sobre todos os arquivos (Fi)

diretório (parte) →

mantém informações
sobre arquivos (arquivo
especial).



ou um diretório!

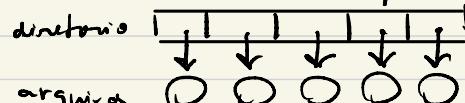
→ operações no diretório = pesquisar por um arquivo, criar, remover, listar, renomear.

► Um mesmo arquivo pode ter vários nomes

► Diferentes usuários podem dar o mesmo nome a arquivos diferentes

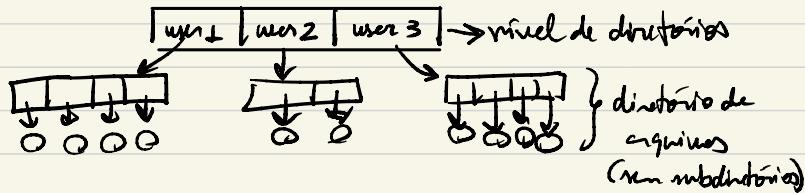
* diretório de 1 nível → um único diretório p/ todos os usuários (não vale a pena)

↓
problemas de identificação



* diretório de 2 níveis → 1 diretório separado p/ cada usuário

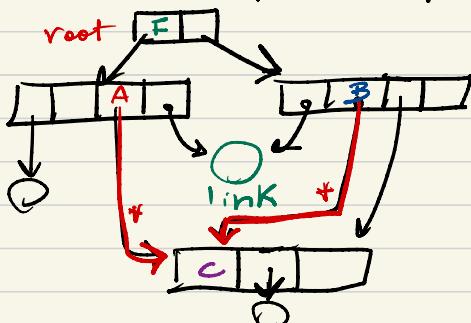
(busca eficiente)



* diretório c/ estrutura de gráfes

↳ Mais caminhos p/ acessar um mesmo arquivo

↳ Diretórios e arquivos compartilhados



- ↳ Dois nomes diferentes (Aliasing)
- ↳ se F remove C → ponteiro pendente
- ↳ solução = contadores de referências
- ↳ problema = ciclos

* Como garantir que não haverá ciclos?

- permitir links apenas p/ arquivos, e não diretórios
- coleta de lixo
- cada vez que um link for adicionado, usar um algoritmo detector de ciclos.

Diretórios com Estrutura em Árvore

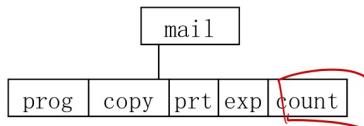
→ formado entre o diretório raiz e o arquivo

- ♦ Caminho absolutos ou relativos
- ♦ Criação de novos arquivos é feita no diretório corrente.
- ♦ Remover arquivo
(remove)
`rm <nome>`
- ♦ Criação de novos subdiretórios é feita no diretório corrente.



Exemplo: se dir. corrente /spell/mail

`mkdir count`



★ Mais Infos. sobre o FAT:

- permite fragmentação, pois os blocos (clusters) de um arquivo podem ser aloçados em posições não contíguas no disco
- porém, arquivos menores que o tamanho do cluster não são fragmentados, pois ocupam apenas um cluster inteto!

I/O

- ↳ São dispositivos externos que requerem diferentes lógicas de operações
 - ↳ Instruções de I/O controlam dispositivos
 - ↳ Possuem:
 - porta (ponto físico da conexão entre o dispositivo e o PC)
 - barramento (canal de comunicação que liga CPU, memória e dispositivos)
 - controladora (circuito que traduz comandos da CPU para operações)
 - driver (software que permite o SO conversar com a controladora)

* Comunicação CPU - Dispositivo:

a) Interrupções

- dispositivo anisa a CPU quando uma operação termina
 - CPU salva o contexto, trata as interrupções e retoma a execução
 - libera CPU p/ outras tarefas durante operações lentas

b) DMA (Acesso Direto à Memória)

- usado p/ evitar I/O programada (PIO) p/ grandes movimentações de dados
 - não usa CPU p/ transferir dados entre os dispositivos I/O e a memória

• 1º - driver recebe instrução p/ transferir dados do disco p/ o buffer na memória

• 2º - driver fala p/ o controlador DMA e informa quantos bytes e o endereço dos dados e do buffer de destino

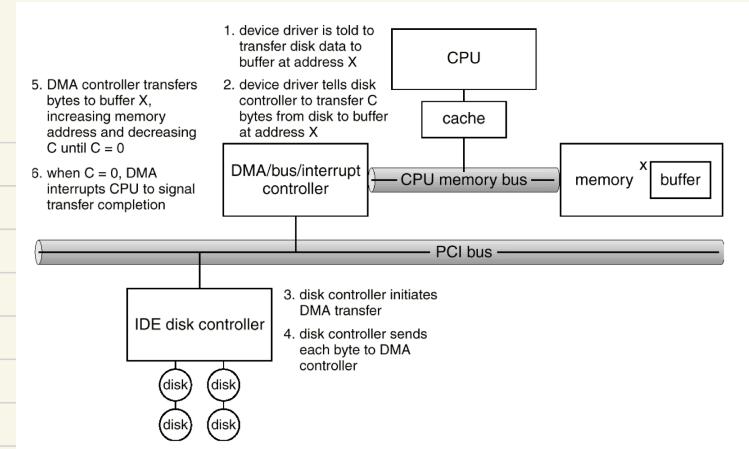
p/ a CPU não intervir

• 3º - controlador de disco inicia a transferência e assume o controle de barramento

• 4º - controlador disco acaba fisicamente o disco e envia cada byte p/ o controlador

• 5º - o controlador DMA recebe os bytes do controlador de disco e transfere p/ o endereço X.

• 6º - o controlador DMA gera uma interrupção p/ notificar a CPU e agora o driver pode processar os dados.



★ Interface I/O

↳ SO esconde a complexidade dos dispositivos, oferecendo uma interface uniforme.

★ Estrutura do subsistema de I/O:

- chamadas de sistema (read(), write(), etc) p/ I/O encapsulam o comportamento dos dispositivos em classes genéricas
- drivers de dispositivo traduzem essas chamadas genéricas p/ comando específicos do hardware
- gerenciamento de filos organiza requisitos de I/O p/ minimizar movimentos desnecessários

★ Tipos de Dispositivos

bloco

vs

Caractere

comandos de leitura, escrita e posicionamento

disco rígido, SSDs

rotativos de buffering com fluxo contínuo de dados

teclado, mouse, serial

★ Estratégias de Gerência de I/O

→ objetivo = melhorar eficiência e evitar gargalos.

• Buffering → armazenamos dados na memória enquanto são transferidos entre dispositivos

↳ (temporariamente, na memória RAM)

↳ o buffering é usado p/ lidar c/ diferenças de velocidade (ex: modem lento vs CPU rápida) ou c/ diferenças de tamanho de blocos de dados.

• Caching → manterem cópias de dados frequentemente acessados em memória rápida p/ aumento de performance.

• Spooling → armazena jobs de I/O em filas (retém a saída de um dispositivo, quando ele não pode atender uma requisição por ex.).
↳ ex: dispositivos dedicados como Impressora.

• Escalonamento → requisições de I/O ordenadas pela fila do dispositivo (P3 - início do disco; P2 - meio; P1 - fim).

↳ decidem se o processo deve esperar ou não pela conclusão do I/O

I/O Bloqueante vs Não-Bloqueante

- processo dorme até o I/O concluir
- uso em aplicações simples
- insuficiente p/ algumas aplicações

- processo continua executando (usa multi-threading)
- uso em servidores, interf. gráficas
- chamados de I/O retornam o quanto antes

Modos de Acesso:

- usuário = processos só podem usar I/O por chamadas de sistema
- Kernel = drivers executam c/ privilégios totais.

Clocks e Timers:

- Provêem hora atual, tempo decorrido e cronômetros
- temporizador de intervalo programável usado para interrupções periódicas

Tratamento de Pedidos de I/O

- ♦ Considere a leitura de um arquivo em disco para um processo
 - Determinar o dispositivo que guarda o arquivo
 - Traduzir nome para representação do dispositivo
 - Ler dados do disco para buffer
 - Disponibilizar dados para o processo solicitante
 - Retornar controle para o processo