

# 50 - exame teórico

## Unidade 5

### 5.1. Conceitos Básicos

Registros e RAM são as unidades de armazenamento acessíveis à CPU. Enquanto que o acesso a registos é mais rápido (consumo apenas um ciclo de relógio) o acesso à RAM é mais lento (consumo vários ciclos de relógio).

Os programas são armazenados em disco como ficheiros binários executáveis e precisam de ser carregados na memória para execução. Para a co-existência de vários programas em memória, não necessários mecanismos de proteção.

As instruções dos programas em execução, e os dados associados, são acedidos através dos respetivos endereços em RAM (ciclo fetch-decode-execute).

### 5.2. Associação de Endereços

As associações de endereços são essenciais para a execução de programas, permitindo a tradução de endereços lógicos em endereços físicos.

Os programas passam por várias fases antes de execução. Em cada fase, os endereços podem ser representados de diversas formas. No código fonte, os endereços são representados de forma simbólica e convertidos em endereços relativos (ex: 14 bytes a partir do início do módulo) ou absolutos.

A associação entre instruções e os respetivos endereços absolutos ocorre em três etapas:

- **tempo de compilação:** se o espaço de memória for conhecido, o código pode ser gerado com referências absolutas, mas precisa ser recompilado se o programa for executado numa área de memória diferente.

- **tempo de carregamento:** se o espaço de memória não for conhecido, deve-se gerar código relocável, adiando as associa-

ções até ao carregamento

- **tempo de execução:** se a localização do programa puder mudar, a associação é adiada para esse tempo e requer hardware auxiliar, como registradores de base e limite e uma Unidade de Gerenciamento de Memória (UMU).

→ **Endereço lógico/virtual:** gerado pela CPU, durante a execução do programa.

→ **Endereço físico/real:** endereço de uma posição real da RAM.

### 5.3. Alociação Contígua

A alocação contígua organiza a memória em partições para gerenciar processos de forma eficiente.

A memória é dividida em duas partições:

- uma para o vetor de interrupções e sistema operativo.
- outra para processos de utilizadores.

Existem duas abordagens:

- múltiplas partições de tamanho fixo.
- múltiplas partições de tamanho variável.

Os algoritmos de armazenamento dinâmico satisfazem um pedido de um bloco de tamanho  $m$  de entre uma lista de blocos livres de várias dimensões:

- **First-fit:** aloca o primeiro bloco livre suficiente que encontra.
- **Best-fit:** aloca o bloco mais pequeno de entre os que têm tamanho suficiente.
- **Worst-fit:** escolhe o maior bloco livre.

O First-fit e o Best-fit são equiparáveis em termos de velocidade e utilização de memória, sendo o First-fit um pouco mais rápido. Ambos têm rendimento superior ao Worst-fit.

Os algoritmos de armazenamento dinâmico geram **fragmentação externa**, isto é, existe memória suficiente para satisfazer um pedido, mas essa memória não é contígua, encontra-se dispersa em pequenos blocos livres.

Se ocorrer fragmentação externa, há a necessidade de compactação para otimizar o uso da memória. Esta consiste em juntar todos os blocos livres num único bloco contíguo. Uma vez que a compactação pelo algoritmo mais simples pode ser muito pesada, há esquemas de compactação alternativos, com menor impacto no desempenho.

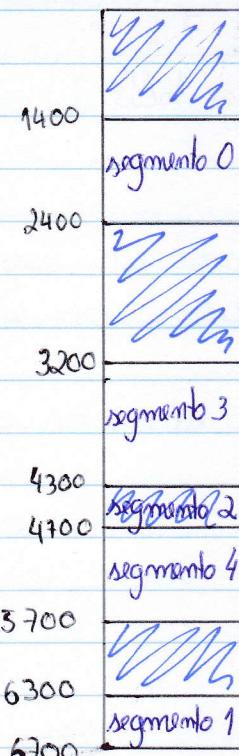
#### 5.4. Segmentação

A segmentação permite uma visão funcional da memória, dividindo-a em segmentos de tamanhos e funções variáveis.

A segmentação é um mecanismo de realocação dinâmica baseado numa tabela de segmentos que contém pares de base (índice físico) e limite (tamanho do segmento), havendo apenas um par por segmento.

	ladrão	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table



memória física

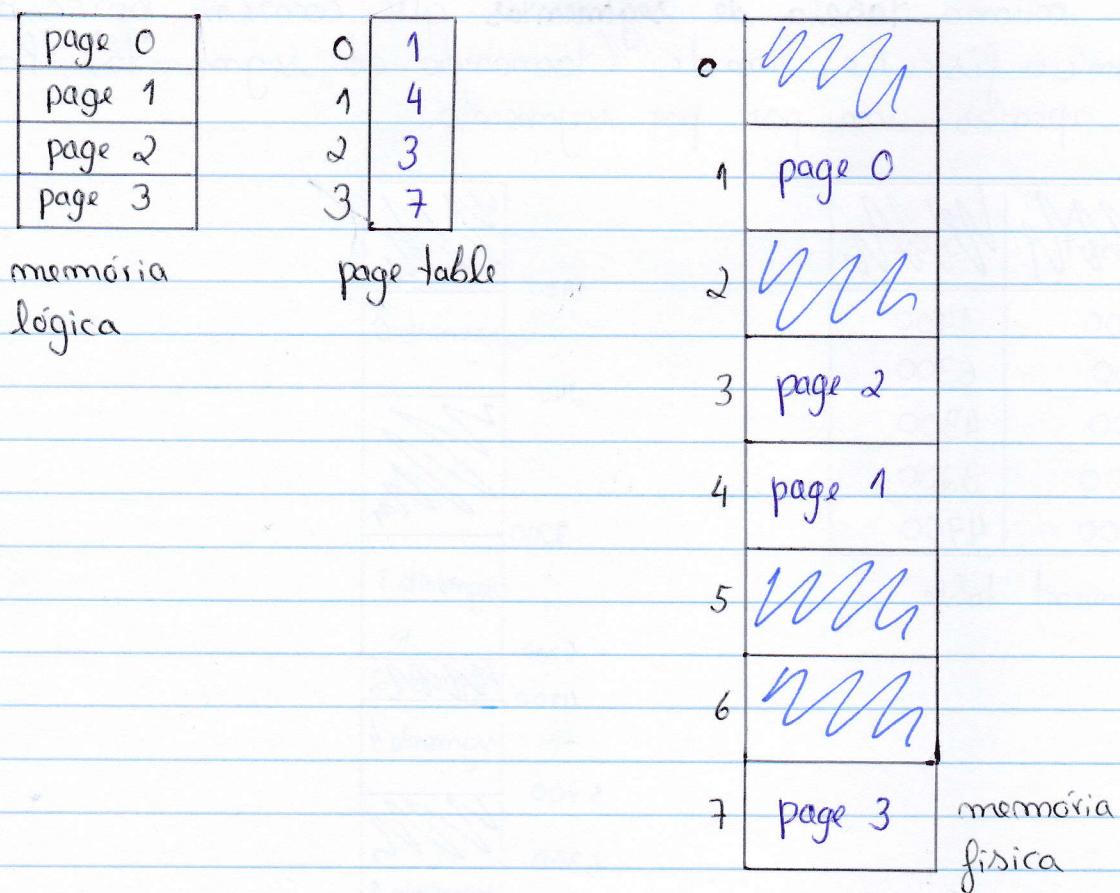
Em segmentação, a divisão em segmentos é explícita e ocorre antes da execução do programa.

Tal como a alocação contínua, também a segmentação sofre de fragmentação externa.

## 5.5. Paginação

A paginação oferece uma visão linear da memória, dividindo-a em páginas de tamanho fixo.

A ideia central é dividir a memória física (RAM) em pequenos blocos de tamanho fixo chamados **frames** e dividir a memória lógica (a visão do processo) em blocos do mesmo tamanho chamados **páginas**. Desta forma, o SO pode carregar qualquer página de um processo em qualquer frame livre da memória física, acabando completamente com a fragmentação externa.



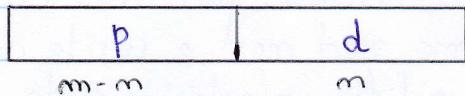
Quando um programa é executado, são gerados endereços lógicos (virtuais). A unidade de gestão de memória (MMU) tem de traduzir estes para endereços físicos, em tempo real. O endereço lógico divide-se em duas partes:

- número da página (p): funciona como o índice. A MMU usa este número para consultar a Tabela de Páginas do processo e descobrir em que Frame (f) está a página guardada.
- deslocamento na página (d): indica a quantidade de bytes que é necessário avançar desde o início da página para encontrar o dado que queremos.

Assim, o endereço físico é formado pela junção do número do Frame (f) com o Deslocamento (d).

A dimensão das páginas/frames é normalmente uma potência de 2. Um Espaço de Endereçamento Virtual (EEV) de  $m$  bits tem  $2^m$  endereços virtuais, distribuídos por  $2^{m-m}$  páginas, com  $2^m$  endereços por página.

→ Estrutura de um endereço virtual



Exemplo:

- EEV de  $m=5$  bits  $\rightarrow 2^5 = 32$  endereços virtuais  $\langle p, d \rangle$
- EEV de 4 bits  $\rightarrow 2^4 = 16$  endereços reais  $\langle f, d \rangle$

• Com páginas de 4 bytes, d usa  $\rightarrow m = \log_2(4) = 2$  bits

• Sobram  $m-m = 3$  bits para identificar um p ( $2^3 = 8$  páginas)

• Sobram  $4-2 = 2$  bits para identificar uma f ( $2^2 = 4$  frames)

A Tabela de Páginas de cada processo reside na RAM, o que cria um problema de desempenho, uma vez que para aceder a um dado, a CPU teria de fazer dois acessos à memória (um para saber onde está o frame e o outro para ler o dado em si). Para resolver isto, utiliza-se a TLB (Translation Look-aside Buffer), uma memória cache muito rápida que guarda os pares  $\langle$  página, frame  $\rangle$  acedidos mais recentemente.

## Umidade 6

### ■ 6.1. Conceitos sobre Ficheiros

O SO cria uma abstracção chamada ficheiro para não termos de lidar diretamente com os setores físicos do disco.

Um ficheiro é uma coleção de informações inter-relacionadas, com nome próprio, que pode conter programas ou dados.

Cada ficheiro possui atributos como, o nome, id, tipo, localização, tamanho, proteção, marcas temporais...

Qualquer SO tem de suportar um conjunto de operações básicas sobre os ficheiros. O acesso a estas, faz-se através de operações básicas como criar, abrir, escrever, ler e remover.

Existem dois métodos de acesso aos dados:

- Acesso sequencial: os dados são acedidos por ordem, registo após registo, com operações como read next e write next.
- Acesso direto: os dados são acedidos aleatoriamente, com base no número do registo, com operações como read n e write n.

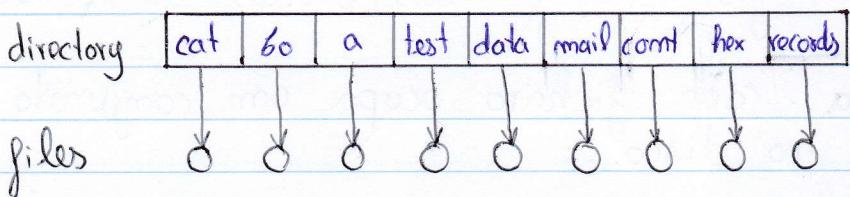
### ■ 6.2. Conceitos sobre Diretórios

Uma diretoria é uma tabela que associa nomes de ficheiros aos seus atributos, mais precisamente, ao bloco de disco que os contém (File Control Block (FCB)).

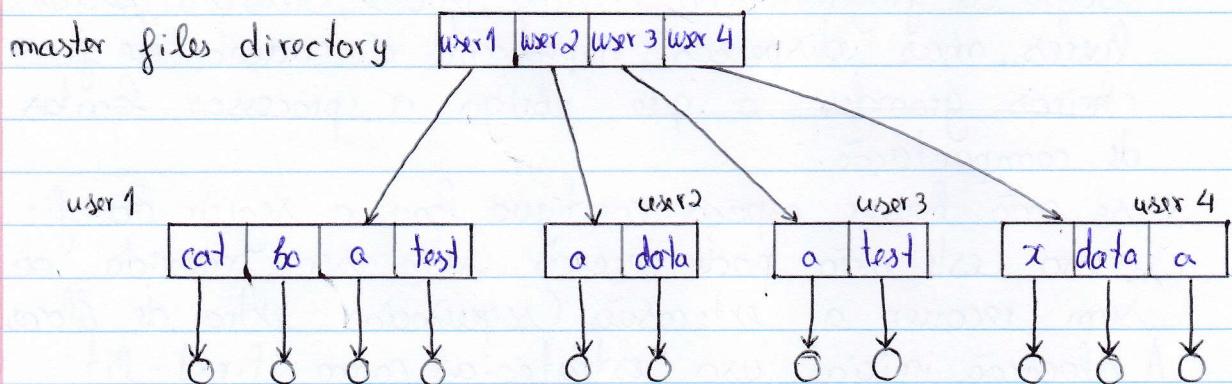
Esta tabela deve permitir a realização de várias operações, entre as quais, criar, remover, listar, renomear ficheiros.

As diretórios devem ter funcionalidades que tornem a sua utilização mais prática, como por exemplo, permitir o uso do mesmo nome em locais diferentes, omitir a extensão no nome, permitir que o mesmo ficheiro tenha vários nomes, permitir caracteres especiais e nomes longos. Estas funcionalidades variam conforme a estrutura adotada, que pode ser de 1 nível, 2 níveis, em árvore ou grafo.

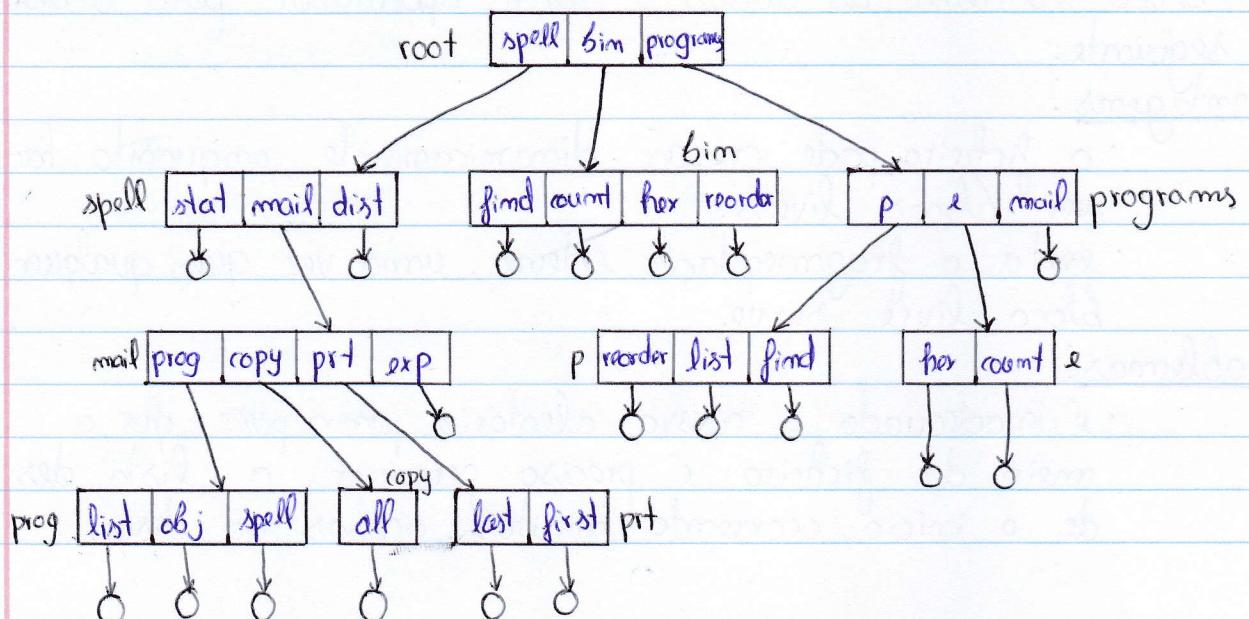
- 1 Nível: uma única diretoria com todos os ficheiros, o que causa conflitos de nomes entre utilizadores.



- 2 Níveis: uma diretoria para cada utilizador, permitindo nomes iguais para ficheiros de utilizadores diferentes. No entanto, o agrupamento de ficheiros em subpastas ainda não é permitido, bem como a partilha de ficheiros.



- Árvore: suporta a múltiplas níveis. Introduz os conceitos de caminho absoluto (desde a raiz) e caminho relativo (desde a diretoria corrente). Ainda não suporta a partilha de ficheiros entre utilizadores.



## 6.5. Métodos de Alocação

Existem três métodos principais para que os ficheiros sejam distribuídos fisicamente pelos blocos do disco.

- **Alocação Contígua**: cada ficheiro ocupa um conjunto de blocos contíguos no disco.

→ Vantagens:

- desempenho excelente, pois o acesso sequencial exige apenas um movimento da cabeça de leitura (disk seek) para ler o ficheiro todo.
- o endereço físico calcula-se somando o índice desejado ao endereço base.

→ Problemas:

- sofre de fragmentação externa, isto é, existem blocos livres, mas dispersos, impedindo a criação de ficheiros grandes, o que obriga a processos lentos de compactação.
- se não houver espaço contíguo logo a seguir ao ficheiro, este não pode crescer sem ser movido ou sem recorrer a extensões (sequências extra de blocos).

A alocação inicial usa estratégias como First-fit, Best-fit ou Worst-fit para tentar mitigar a fragmentação.

- **Alocação Ligada**: trata cada ficheiro como uma lista encadeada de blocos, que podem estar espalhados pelo disco. Cada bloco contém os dados e um apontador para o bloco seguinte.

→ Vantagens:

- o ficheiro pode crescer dinamicamente, enquanto houver blocos livres.
- evita a fragmentação externa, uma vez que, qualquer bloco livre serve.

→ Problemas:

- é inadequado a acesso aleatório, pois para ler o meio do ficheiro é preciso percorrer a lista desde o início, ocorrendo muitos acessos ao disco.

- perde-se espaço dentro de cada bloco para guardar o apontador.
- a fiabilidade é menor, pois se um apontador se corromper, perde-se o resto do ficheiro.

### - FAT (file allocation table)

A FAT é uma evolução da Alocação Ligada. Assim, em vez de guardar os apontadores dentro dos blocos de dados, estes são centralizados numa tabela única (a FAT) localizada no início do volume.

→ Vantagens:

- Como a tabela FAT é relativamente pequena, pode ser carregada para a memória cache. Isto forma o acesso aleatório muito mais rápido, uma vez que não é necessário ir ao disco para seguir os apontadores.

### ■ 6.6. Gestão do Espaço Livre

O SO necessita de saber quais são os blocos do disco que estão disponíveis para gravação. Existem duas abordagens principais:

- **Vetor de Bits:** utiliza um vetor de  $m$  bits, onde cada bit representa um bloco do disco. Se o bit for 1, o bloco está livre e se for 0, está ocupado.

$$\text{Vetor}[i] = \begin{cases} 1 \Rightarrow \text{bloco } i \text{ está livre} \\ 0 \Rightarrow \text{bloco } i \text{ está ocupado} \end{cases}$$

0	1	2	$\dots$	$m-1$

→ Índice do 1º bloco livre:

$$(\text{m.º bits/palavras}) \times (\text{m.º palavras a } 0) + (\text{deslocamento do 1º bit} + 1) - 1$$

Exemplo: vetor = 100000000|00000000|00101111...

$$\text{índice do 1º bloco livre} = (8 \times 2) + 3 - 1 = 18$$

Esta abordagem é eficiente para encontrar espaço contíguo (basta procurar uma sequência de 1s), beneficiando muitas vezes de instruções de hardware específicas para encontrar o primeiro bit a 1 rapidamente.

Para ser rápido, o vetor deve residir na RAM, o que exige sincronização periódica com o disco para evitar inconsistências em caso de falha.

• **Lista Ligada**: todos os blocos livres são encadeados numa lista. O cabeçalho da lista aponta para o primeiro bloco livre, que por sua vez aponta para o próximo, e assim sucessivamente.

Com esta abordagem é fácil encontrar um bloco livre, basta tirar o primeiro da lista. No entanto, é pouco eficiente para alocar espaço contíguo, pois obriga a percorrer a lista à procura de endereços vizinhos.

Em sistemas com FAT, a gestão de livres é feita de forma similar através de uma lista ligada embbebida nas entradas livres da própria tabela.