

# Implementação de Sistemas de Ficheiros

Paulo Sérgio Almeida

Grupo de Sistemas Distribuídos  
Departamento de Informática  
Universidade do Minho



# Estrutura de sistemas de ficheiros

- Ficheiros:
  - unidade lógica de armazenamento
  - colecção de informação relacionada
- Sistemas de ficheiros:
  - residem em armazenamento secundário (disco)
  - podem ser organizados em camadas
  - contém diversas estruturas de dados
  - gerem a alocação de espaço em disco
  - necessitam de gerir espaço livre



# Organização em camadas

- Sistemas de ficheiros podem ser organizados em camadas
  - **controlo de I/O**: contém **device drivers** e **interrupt handlers**
    - comunica com os controladores dos dispositivos
  - **sistema de ficheiros básico**: transmite comandos aos device drivers para ler e escrever blocos físicos em disco
  - **módulo de organização de ficheiros**:
    - conhece a disposição dos ficheiros em disco
    - traduz blocos lógicos dos ficheiros em blocos físicos
    - faz pedidos de transferência ao sistema de ficheiros básico
    - inclui o gestor de espaço livre em disco
  - **sistema de ficheiros lógico**:
    - manipula metadados do sistema de ficheiros:
    - manipula estrutura de directórios
    - manipula file control blocks
    - responsável por protecção no acesso aos ficheiros
- As duas camadas mais baixas podem ser partilhadas por diversos sistemas de ficheiros



# Estruturas de um sistema de ficheiros

- **boot control block**: contém informação necessária ao arranque de um sistema operativo nesse volume
  - pode ser chamado de **boot block** ou **partition boot sector**
- **volume control block**: informação sobre o volume ou partição
  - número e tamanho dos blocos
  - número e apontadores para blocos livres
  - número e apontadores para file control blocks livres
- **Estrutura de directório**: relaciona nomes de ficheiros com FCBs
- **FCB –File control block**: estrutura com informação sobre ficheiro; **inode** em Unix
  - criador, grupo, ACL
  - datas (criação, acesso, escrita)
  - permissões
  - tamanho
  - localização dos blocos do ficheiro



# Estruturas de dados em memória

- São importantes para obter eficiência
- Servem de chache de informação em disco
  - com hit rate médio de 85% em BSD UNIX
- Podem ser trazidas para memória ao montar sistema de ficheiros ou abrir ficheiros
- Exemplos:
  - tabela de sistemas de ficheiros montados
  - cache de directórios acedidos recentemente
  - tabela global de ficheiros abertos – contém FCBs
  - tabelas de ficheiros abertos, por processo



# Sistemas de ficheiros virtuais

- **Virtual File Systems (VFS):** implementação de sistema ficheiros orientada por objectos
- Permitem que a mesma interface de chamadas ao sistema (API) seja usada para aceder a diferentes sistemas de ficheiros
- Programas usam interface genérica de sistema de ficheiros
- Interface genérica invoca API do VFS e não de um sistema de ficheiros específico
- Exemplo: permite encapsular um sistema de ficheiros local e um remoto sob a mesma interface



# Implementação de directórios

- **Lista** com nomes de ficheiros e apontadores para blocos
  - simples
  - procura de nome potencialmente demorada
  - lista ordenada facilita procura mas dificulta alterações
- **Hash table**
  - reduz tempo de procura no directório
  - possibilidade de colisões
  - problema: tamanho fixo da hash table e dependência da função de hash desse tamanho



# Alocação de espaço em disco

- Espaço em disco dividido em blocos
- Vários blocos por ficheiro
- Vários métodos para atribuir blocos a ficheiros:
  - contígua
  - lista ligada
  - indexada
- normalmente um sistema de ficheiros usa apenas um método





# Contígua

- Cada ficheiro ocupa conjunto contíguo de blocos
- Simples: necessário guardar bloco inicial e tamanho
- Acesso eficiente: pouco deslocamento cabeça do disco
- Permite acesso aleatório a ficheiros
- Desperdício de espaço: fragmentação externa
- Ficheiros não podem crescer ou grande fragmentação interna
- Compactação periódica para eliminar fragmentação; custosa
- Variante: alocação por **extents**
  - um extent é uma sequência contígua de blocos
  - um ficheiro é alocado a um conjunto de extents



# Lista ligada

- Ficheiro constituído por lista ligada de blocos
- Blocos espalhados pelo disco
- Acesso pouco eficiente: deslocamento cabeça do disco
- Viável para acesso sequencial; acesso aleatório muito ineficiente
- Bom uso do espaço: apenas pequena fragmentação interna
- Ficheiros podem crescer sem problema
- Variante: agrupar blocos em **clusters** e alocar clusters



# File allocation table (FAT)

- Variante de lista ligada
- Tabela separada com uma entrada por bloco do disco
- Cada ficheiro é representado por uma lista ligada na FAT
- Pode ser feita cache da FAT para aumentar eficiência no acesso
- FAT usada pelo MS-DOS e OS2



# Alocação indexada

- Cada ficheiro tem um **index block**
- Contém apontadores para blocos; análogo a tabela de páginas
- Acesso aleatório a ficheiros
- Ausência de fragmentação externa
- Problema: tamanho do index block
  - grande – desperdício espaço para ficheiros pequenos
  - pequeno – problema: como guardar ficheiros grandes



# Alocação indexada

Como suportar ficheiros grandes:

- lista ligada de index blocks
- indexação multi-nível: hierarquia de index blocks
- esquema misto (e.g. inodes em UNIX):
  - primeiras entradas apontam para blocos;
  - uma entrada para index block;
  - uma entrada para hierarquia de index blocks com 2 níveis;
  - uma entrada para hierarquia de index blocks com 3 níveis;



# Gestão do espaço livre – bitmap

- **bitmap** – vector de bits
- Bloco ocupado – bit = 1
- Número de bits = número de blocos em disco
- Fácil localizar espaço livre contíguo:
  - sequências de longs a 0 – 64 blocos livres por palavra
  - sequências de bytes a 0 – 8 blocos livres por byte
- Bitmap pode ocupar muito espaço; exemplo:
  - tamanho do bloco:  $4\text{KB} = 2^{12}$  bytes
  - tamanho do disco:  $64\text{GB} = 2^{36}$  bytes
  - número de bits:  $2^{36}/2^{12} = 2^{24} = 4\text{M}$  bits
- Pode demorar tempo a percorrer bitmap e encontrar espaço livre



# Gestão do espaço livre – lista ligada

- Lista ligada de blocos livres
- Encontra-se nos próprios blocos – não necessita espaço extra
- Inapropriada para encontrar blocos contíguos
- Imediato encontrar bloco livre – não necessita procura
- FAT usa este mesmo método para blocos livres e ficheiros
- Variante – com agrupamento:
  - bloco livre mantém apontadores para  $n$  blocos livres
  - o último destes contém outros  $n$  apontadores
- Variante – contar blocos livres contíguos:
  - cada bloco contém número de blocos livres contíguos à sua frente



# Eficiência e desempenho

- Eficiência depende de:
  - algoritmos de alocação de disco e directórios
  - tipos de dados na entrada do ficheiro no directório
- Performance:
  - cache em memória de blocos de disco
  - cache unificada:
    - para páginas de ficheiros mapeados em memória e
    - blocos de ficheiros acedidos via sistema de ficheiros
    - cache unificada evita **double caching**; melhora eficiência
  - **read-ahead** e **free-behind** para otimizar acesso sequencial
  - I/O assíncrono – nomeadamente escritas
  - dedicar parte da memória para disco virtual – **RAM disk**





# Eficiência e desempenho: estratégias

- Pre-alocar inodes e espalha-los pelo disco:
  - algum espaço pode ser desperdiçado
  - ideia: tentar manter blocos de um ficheiro perto do seu inode
  - reduz **seek time**
- Clusters de tamanho variável:
  - agrupamento de blocos em clusters melhora tempo de acesso
  - mas causa fragmentação
  - solução: usar clusters de tamanho variável
  - clusters pequenos para ficheiros pequenos e último cluster de ficheiro



# Consistência do sistema de ficheiros

- Dados em memória podem não ser logo gravados em disco
- Se o computador falha, informação pode ser perdida
- Possibilidades:
  - conteúdo de ficheiro perde actualizações
  - sistema de ficheiros fica inconsistente
- Inconsistência de sistema de ficheiros mais grave
- Resulta de invariantes das estruturas de controlo serem violados
- Solução parcial: estruturas de controlo devem ser gravadas sincronamente quando modificadas
- Programas de **verificação de consistência** comparam estrutura de directórios com blocos no disco e tentam corrigir problemas



# Sistemas de ficheiros estruturados por log

- Pretendem evitar corrupções num sistema de ficheiros
- Sistemas de ficheiros **log structured** ou **journaling** guardam cada actualização ao sistema de ficheiros como uma **transacção**
- As transacções são escritas num log
  - uma transacção é **committed** depois de escrita no log
  - o sistema de ficheiros pode ser actualizado mais tarde
- As transacções no log são aplicadas ao sistema de ficheiros assincronamente
  - quando este é modificado, a transacção é removida do log
- Se há um crash, as transacções restantes do log não se perdem e serão aplicadas mais tarde
- Acesso sequencial ao log melhora desempenho

