Instituto de Informática - UFRGS

Sistemas Operacionais

Sistema de arquivos

Gerenciamento do espaço livre em disco e questões de desempenho

Aula 26

Introdução

- Dados são lidos de arquivos em unidades lógicas
  - e.g.: caracter, inteiro, ponto flutuante, strings, registros…
- Bloco lógico é a unidade de armazenamento de dados no meio físico
  - Conjunto de um ou mais setores (blocos físicos)
- Dados (registros) são lidos e armazenados em unidades de blocos
  - Conversão dados/registros ↔ bloco
  - Menor unidade de alocação no disco é o bloco lógico
- Questão: qual o tamanho ideal para um bloco lógico?

Instituto de Informática - UFRGS
A. Carissimi -21-jun-12

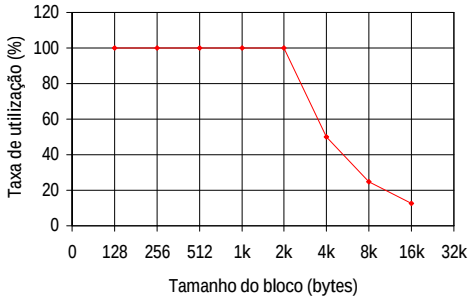
Tamanho do bloco

- Relação custo x benefício (mais uma vez)
  - Bloco grande implica que mais dados podem ser manipulados em uma única operação de E/S, porém gera desperdício (fragmentação interna)
  - Bloco pequeno reduz o desperdício, porém gera queda de desempenho por gerar mais *seek* e latência rotacional
- Análise depende também do tipo de acesso
  - Sequencial ou randômico
- Custo se traduz:
  - Gerência de buffers na memória principal
  - Gerência de espaços livres e ocupados no disco
  - Desempenho (*seek* + latência rotacional + tempo de transferência)

Instituto de Informática - UFRGS
A. Carissimi -21-jun-12

Análise: taxa de utilização do disco

- Análise do tamanho de arquivos *versus* o tamanho de bloco
  - Unix (1 KB médio, 1984); Unix (2475 bytes [mediana], 2005);
  - Windows (4.2 KB média, 1999)
- Supondo arquivos de tamanho 2 Kbytes



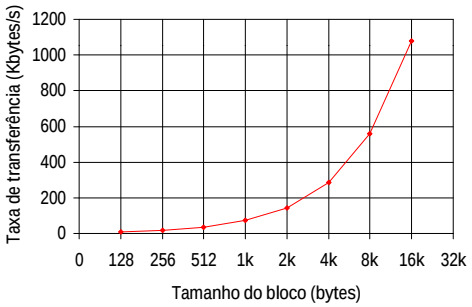
Instituto de Informática - UFRGS
A. Carissimi -21-jun-12

Análise: taxa de transferência

- Supondo:
  - t_{seek} = 10 msec; $t_{latência}$ = 8.3 msec (7200 rpm), trilha = 131072 bytes (128 KB)
  - Cálculo da taxa de transferência para blocos de 128, 256, 512... bytes

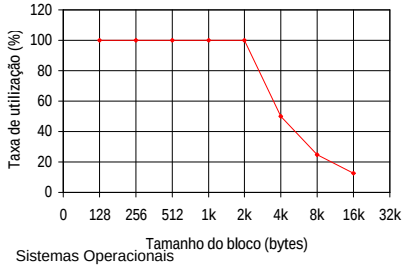
$$T_{acesso} = t_{seek\_médio} + \frac{1}{2r} + \frac{b}{rN}$$

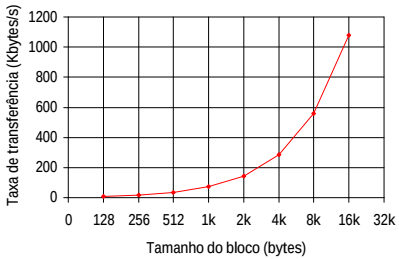
Onde:
b = número de bytes a serem transferidos
N = número de bytes em uma trilha
r = rotação do disco em rpm (passar segundo!!)



Tamanho do bloco: conclusão

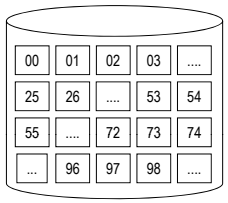
- Objetivos conflitantes:
  - Boa utilização do espaço em disco leva a baixo desempenho
  - Bom desempenho leva a desperdício de espaço em disco
- Compromisso é escolher um tamanho de bloco adequado as necessidades e uso do sistema
  - Solução paliativa:Tamanho do bloco é por partição





Gerenciamento do espaço livre

- Necessário manter a informação de blocos livres e ocupados
  - Criar ou expandir um arquivo necessita alocar blocos
  - Remover ou diminuir um arquivo implica em liberar blocos (reaproveitá-los!!!)
- Métodos de base:
  - Lista encadeada
  - Mapa de bits (*bitmap*)
- Consideram que blocos no disco são numerados sequencialmente



Lista encadeada

- Os blocos livres formam um arquivo “espaço livre”
  - Usa técnicas de organização estudadas (encadeado, indexado, combinado)
- Organização encadeada é a mais opção simples
  - Um bloco possui a localização dos blocos livres
  - Última entrada aponta para um novo bloco de blocos livres

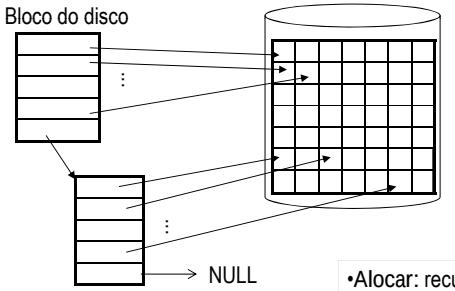
•Alocar: recuperar um endereço de bloco livre
•Liberar: inserir endereço na lista de livres

Problemas com lista encadeada

- Acesso aos blocos do disco para manutenção da lista encadeada
- Com o uso do disco a tendência é que entradas consecutivas apontem para blocos não contíguos no disco
  - Fragmentação do arquivo
  - Possibilidade: manter a lista de blocos livres ordenada ou na alocação de *n* blocos procurar blocos contíguos → implica varrer a lista
    - Problema é desempenho

Agrupamento

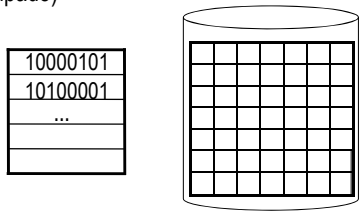
- Armazenar os endereços dos blocos livres no primeiro bloco livre
  - Na veradade, armazenar n-1 blocos livres, pois a n-enésima entrada server para apontar para outro bloco que contém blocos livres



•Alocar: recuperar um endereço de bloco livre
•Liberar: inserir endereço na lista de livres

Mapa de bits (*Bit maps*)

- Cada bloco lógico ou está livre ou está ocupado
  - Um bit por bloco (ex.: 1:livre 0:ocupado)



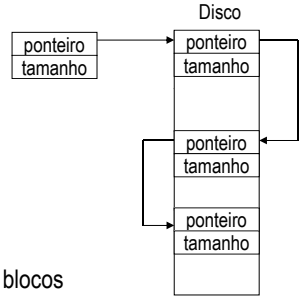
- Prós e contras:
  - Lista de tamanho reduzido
    - (*n* blocos = *n* bits)
  - Localização espacial pode ser identificada por uma sequência de bits em um mesmo estado
  - Desempenho no varrer o *bitmap*
    - Operações de manipulação de bits

Alternativas para lista e bit *maps* de blocos livres

- Métodos para reduzir tamanho e/ou tempo de acesso a lista de blocos livres
- Alternativa I
  - Lista de porções livres
- Alternativa III
  - Emprego de estruturas de dados auxiliares

Lista de porções livres

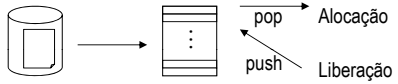
- Variação de lista encadeada
- Definir grupos de blocos adjacentes (porções) e encadeá-los



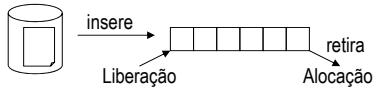
- Prós e contras:
  - Reduz o tamanho da lista
  - Alocação e liberação é por porções
    - Facilita a localização espacial dos blocos
  - A medida que as porções vão sendo alocadas e liberadas a lista começa a crescer e há tendência de formar várias "porções pequenas"

Estrutura de dados auxiliares

- Idéia é manter parcialmente em memória a lista ou *bit map* de blocos livres → desempenho
- Opção I: Utilizar uma pilha (*push-down*)



- Opção II: Utilizar uma fila



- Transferência envolvendo uma lista parcial ocorre quando:
  - Disco → memória: lista vazia ou Memória → disco: lista cheia

Cache de disco

- Objetivo é manter na memória principal uma certa quantidade de blocos lógicos do disco
- Não adiciona nem elimina funcionalidades ao sistema de arquivos
  - Função é exclusivamente melhorar o desempenho do sistema de arquivos
- Normalmente a cache de disco é mantida em uma área da memória principal e é controlada pelo sistema operacional
  - Pode ser global ou exclusiva (uma para cada sistema de arquivo suportado pelo sistema)

Princípio de funcionamento da cache de disco

- Em uma requisição de E/S verifica se o bloco está na cache
  - Sim: realiza o acesso a partir dessa cópia em memória
  - Não: realiza o acesso a partir do disco e carrega o bloco para a cache
    - Mecanismo de *read-ahead*
- A modificação de valores é feito diretamente nos blocos na cache
  - Problema de quando atualizar o disco após um bloco ter sido alterado
- Problema da perda de informações e da consistência do sistema de arquivos em caso de pane do sistema (falta de energia)
  - Política de atualização

Políticas de atualização

- Posterga ao máximo
  - Grava apenas quando o bloco for removido do cache (se modificado)
  - Cache está plena
- Gravar blocos da cache no disco de forma periódica
  - Operação SYNC (sistemas UNIX)
- Gravar blocos da cache a medida que são modificados
  - Denominado de *write-through caches*
- Atualiza imediatamente apenas informações sensíveis a consistência do sistema do arquivo

Política de substituição

- A cache de disco é um recurso limitado
- O que fazer quando um novo bloco deve ser inserido na cache e não há espaço livre ?
  - Problema similar a gerência de memória virtual (substituição de páginas)
  - Bloco "vítima" é transferido da cache para o disco (se modificado)
- Tipicamente é empregado a política *Least-Recently-Used* (LRU) modificada por considerar que:
  - O bloco pode vir a ser utilizado em um espaço breve de tempo?
    - e.g: bloco de diretório, bloco de i-node indiretos, etc...
  - Bloco possui informação crítica para a consistência do sistema de arquivos?
  - Blocos nessa situação "desobedecem" a regra LRU

Consistência do sistema de arquivos

- Problema de uma pane ocorrer antes que blocos modificados sejam completamente atualizados no disco persiste
- Necessidade de após uma pane garantir a consistência do sistema de arquivos
  - Consistência é vínculada a cada partição
  - Garante-se a coerência da gerência de blocos livres e ocupados
- Tarefa de utilitários do sistema operacional
  - e.g.: chkdisk, fsck

Exemplo: funcionamento do *fsck* (arquivos)

- Ferramenta UNIX para verificação de consistência de sistemas de arquivos
- Baseado na construção de dois vetores, cada um com uma entrada (zerada) para cada bloco lógico do disco:
  - Um vetor monitora quantas vezes um bloco está presente em um arquivo
  - Outro vetor contabiliza quantas vezes um bloco aparece na lista de livres
- Funcionamento:
  - Para cada arquivo verifica a lista de blocos que utiliza
  - Localiza na lista de livres todos os blocos livres
  - Sistema de arquivos está consistente SE cada bloco do disco possuir uma entrada de um dos vetores com o valor 1 e outro zero

fsck: situações inconsistentes (arquivos)

- Bloco não aparece em nenhuma das listas
  - Ação: adicionar bloco na lista de livres
- Bloco ocorre duas ou mais vezes na lista de livres
  - Ação: reconstruir a lista de livres
- Bloco ocorre duas ou mais vezes no interior de um arquivo
  - Ação: alocar um bloco livre, copiar o conteúdo do bloco para o bloco recém-alocado, e atualizar as estruturas livres/ocupados
    - *Importante*: pode resultar em arquivos "embaralhados" porém o sistema de arquivos estará consistente

Funcionamento do *fsck* (diretórios)

- Procedimento similar a verificação de arquivos
- Um vetor com uma entrada (zerada) para cada i-node da partição
  - Corresponde uma entrada para cada arquivo
- Funcionamento:
  - Varre recursivamente todos os diretórios contabilizando a ocorrência de cada *i-node* na sua posição correspondente no vetor
  - No final compara o número de ocorrências de cada *i-node* com o contador de *links* mantido em cada *i-node*
  - Sistema de arquivos está consistente SE o número de ocorrências de cada *i-node* for igual ao contador de links mantido pelo próprio *i-node*

fsck: situações inconsistentes (diretórios)

- Detalhe:
  - i-node é considerado livre quando o contador de links é igual a zero
- Contador de links é maior que o número de ocorrências
  - Problema causado é o desperdício de espaço em disco
    - i-node (e bloco) não são liberados
  - Ação: igualar o contador de links com o número de ocorrências
- Contador de links é menor que o número de ocorrências
  - Problema causado é uma entrada de diretório apontar para um *i-node* "*livre*"
  - Ação: igualar o contador de links com o número de ocorrências

Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani; Sistemas Operacionais. Editora Sagra-Luzzato, 2001.
  - Capítulo 5, seção 5.3.1.4
  - Capítulo 8, seções 8.5 e 8.6
- A. Silberchatz, P. Galvin; *Operating System Concepts*. Addison-Wesley.
  - Capítulo 11 seções 11.2e 11.3

Problema de confiabilidade

- Problema de coerência da lista de blocos livres na presença de falhas quando mantida em memória
- Situação:
  - Usuário A aloca uma série de blocos
  - Alocação é realizada e a tabela na memória é atualizada
  - Sistema entra em pane e é reinicializado
  - Usuário B solicita a alocação de uma série de blocos e os blocos alocados coincidem com a prévia alocação do usuário A

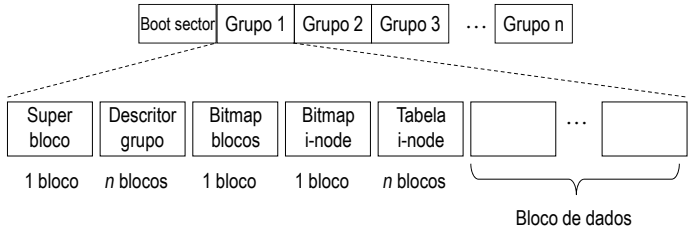
Solução

- Pré-alocar um conjunto de blocos e mantê-los em memória com a indicação gravada em disco de "em uso"
- Proteger mecanismo de alocação com *lock*
  - Realizar um *lock* na tabela de alocação no disco
  - Pesquisar na tabela de alocação em memória a quantidade de blocos "em uso" necessários para satisfazer a alocação
  - Alocar o espaço, atualizar a tabela na memória, atualizar a tabela no disco
  - Liberar o *lock* na tabela de alocação do disco
- Em presença de falhas, os blocos marcados como "em uso" são considerados "livres"
  - Mantém a coerência do sistema de arquivos porém não garante a integridade de arquivos

Estudo de caso: estrutura física do disco em ext2fs

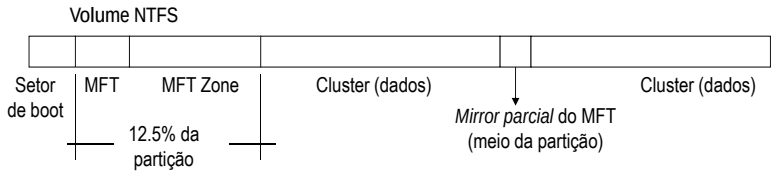
- Fortemente influenciada pelo sistema de arquivos UNIX BSD
- Sistema de arquivos é organizado como um grupo de blocos
  - Cilindros na terminologia do UNIX BSD
- Objetivo é favorecer uma localização espacial dos blocos e das estruturas de gerência do sistema de arquivos
  - Reduzir tempo de *seek*
  - Tentativa de alocar blocos para dados é sempre a partir do grupo de blocos mais próximo da posição atual do cabeçote
    - Se não tem disponível, tenta nos vizinhos mais próximos
- Bloco tem tamanho configurável (por partição)
  - Valores típicos são 1024 bytes, 2048 bytes, 4096 bytes (*default*)

Estudo de caso: layout do sistema de arquivos ext2fs



- Super bloco: descritor do tamanho e formato do sistema de arquivos
- Descritor grupo: organização do grupo (tamanho e formato)
- Bitmap blocos: indicação se um bloco do grupo está livre/ocupado
- Bitmap *i-nodes*: indicação se um *i-node* do grupo está livre/ocupado
- Tabela *i-nodes*: associação de blocos de dados aos *i-nodes* do grupo

Estudo de caso: estrutura de uma partição NTFS



- Espaço do disco é dividido em clusters
  - Utiliza um arquivo de bitmap para manter clustes livres/ocupados
  - Tamanho do cluster é definido em função da capacidade do disco
    - Compromisso é desempenho versus desperdício de disco
  - Um volume NTFS tem até 2^{64}-1 clusters (teórico)
- Uma partição NTFS é dividida em duas partes:
  - Área MFT (*Master File Table*)
  - Área destinada ao armazenamento de dados