Resumo Prova 3 SO

Guilherme Christopher Michaelsen Cardoso

21 de junho de 2017

1 Sistemas de Arquivos

- Programas de computador precisam armazenar e recuperar informação.
  - Quando um processo está rodando, ele pode armazenar uma quantidade limitada de informação dentro do seu próprio espaço de endereçamento.
  - Porém, a capacidade desse armazenamento está limitada ao tamanho do espaço de endereçamento virtual.
  - Para alguns programas, esse tamanho é adequado; para outros, é muito pequeno.
  - Outro problema de se utilizar o espaço de enderaçmento virtual é que ao encerrar o processo, a informação é perdida. Para muitas aplicações, a informação deve ser retida por semanas, meses, ou mesmo para sempre. Não se aceita que essa informação desapareça quando um processo termina ou sofre um *crash*.
  - O terceiro problema é que frequentimente é necessário que múltiplos processos acessem partes da informação ao mesmo tempo. Por isso, a informação deve ser independente dos processos.
- Assim, surgem 3 requisitos essenciais para armazenamento de informação à longo prazo:
  1. Deve ser possivel armazenar uma quantidade muito grande de informação
  2. A informação deve sobreviver ao término do processo que a está utilizando
  3. Múltiplos processos devem ser capazes de acessar a informação de uma vez.
- Dispositivos comuns de armazenamento não-volátil incluem:
  1. Discos magnéticos
  2. SSDs (não possuem partes móveis que podem quebrar, oferecem acesso rápido)
  3. Fitas e discos ópticos (usados tipicamente para backup devido a sua baixa performance).
- Pode-se pensar no disco como sendo uma sequência linear de blocos de tamanho fixo, capazes de suportar duas operações:
  1. Ler o bloco k
  2. Escrever no bloco k
  - Na verdade existem mais operações, mas essas duas poderiam, em princípio, resolver o problema do armazenamento à longo prazo.
  - Na realidade, essas são operações muito inconvenientes, especialmente em sistemas grandes usado por muitas aplicações e, possivelmente, por muitos usuários (por exemplo, em um servidor). Alguns problemas que surgem são:

1. Como encontrar informação?
2. Como garantir que um usuário não leia os dados de outro?
3. Como saber quais blocos estão livres?

e existem muito mais.

- Da mesma forma que o SO abstraiu o conceito do processador para criar o conceito do processo, e abstraiu o conceito da memória física para oferecer a processos espaços de endereçamento virtuais, os problemas relacionados a disco podem ser resolvidos com uma nova abstração: **o arquivo**.
  - Esses três conceitos (processos, espaços de endereçamento e arquivos) são os mais importantes em Sistemas Operacionais.
- **Arquivos** são unidades lógicas de informação criadas pelos processos. Um disco normalmente contém milhares ou até mesmo milhões deles, cada um independente dos outros. Assim como os espaços de endereçamento modelam a RAM, arquivos modelam o disco.
- Processos podem ler arquivos existentes e criar novos arquivos de acordo com a necessidade. Informação armazenada em arquivos deve ser persistente, isto é, não pode ser afetatda pela criação e término dos processos. Um arquivo deve desaparecer apenas quando seu usuário explicitamente o remove. Apesar de operações de leitura e escrita de arquivos serem as mais comuns, existem muitas outras.
- Arquivos são gerenciados pelo sistema operacional. A forma em que eles são estruturados, nomeados, acessados, usados, protegidos, implementados e gerenciados são tópicos importantes em design de sistemas operacionais. A parte do SO que lida com arquivos é chamada de **sistema de arquivos**.

1.1 Arquivos

Resumo

Nessa subseção, observa-se os arquivos do ponto de vista do usuário, isto é, como eles serão usados e quais propriedades eles terão.

1.1.1 Nomeação de Arquivos

- Um arquivo é um mecanismo de abstração, que provê uma forma de armazenar informação no disco e recuperá-la depois.
  - Isso deve ocorrer de forma a esconder do usuário os detalhes sobre como e onde essa informação é armazenada, e como os discos funcionam.
- Quando um processo cria um arquivo, ele dá a esse arquivo um nome.
  - Quando o processo termina, esse arquivo continua a existir e pode ser acessado por outros processos usando seu nome.
- As regras para nomear arquivos variam de sistema para sistema.
  - Todos os SOs atuais permitem strings de 1 a 8 letras como nome de arquivo.
  - Frequentemente digitos numéricos e caracteres especiais também são permitidos.
  - Muitos sistemas de arquivos suportam nomes longos de até 255 caracteres.
  - Alguns sistemas de arquivos realizam distinção entre letras maiúsculas e minúsculas (UNIX, por exemplo), e outras não (MS-DOS, por exemplo).
    - Um adendo sobre sistemas de arquivos: Tanto o Windows 95 quanto o Windows 98 usam o sistema de arquivos do DOS, chamado de **FAT-16**, herdando muitas de suas propriedades (como a construção de nomes de arquivos). Windows 98 introduziu algumas extensões ao FAT-16, criando o **FAT-32**, mas esses dois são similares. As versões posteriores do Windows suportam os sistemas de arquivos FAT, que já são obsoletos, porém, esses sistemas já utilizam um

sistema mais avançado (**NTFS**), que possui propriedades diferentes (como nomes de arquivo em Unicode). No Windows 8 ainda existe um segundo sistema de arquivos, chamado **ReFS**, mas esse é usado apenas na versão para servidores. Nesse capítulo, ao mencionar aos sistemas de arquivos do MS-DOS ou FAT, está se referenciando o uso dos sistemas FAT-16 e FAT-32 no Windows. Ainda existe um outro sistema baseado em FAT, conhecido como **exFAT**, que é uma extensão do FAT-32 otimizada para pendrives e sistemas de arquivos grandes.

- Muitos sistemas operacionais suportam nomes de arquivos divididos em duas partes, separadas por um ponto (ex.: "prog.c"). A parte depois do ponto é chamada de **extensão de arquivo** e geralmente indica algo sobre o arquivo. No MS-DOS, por exemplo, nomes de arquivos contém de 1 a 8 caracters, mais uma extensão adicional de 1 a 3 caracteres. No UNIX, o tamanho da extensão é definido pelo usuário, e um arquivo pode ter até mesmo mais de uma extensão (exemplo homepage.html.zip).
  - No caso do UNIX e alguns outros sistemas, as extensões de arquivo são apenas convenções e não são obrigatórias pelo sistema operacional. Um arquivo chamado file.txt pode ser algum tipo de arquivo de texto, mas o nome serve mais para informar o usuário do que para prover qualquer informação para o computador. Por outro lado, um compilador C pode exigir que os arquivos de entrada estejam com extensão .c, mas o SO não se importa.
  - Essas convenções são especialmente úteis quando o mesmo programa pode lidar com diversos tipos de arquivos.
  - Por outro lado, o Windows utiliza as extensões para identificar quais programas são "donos"de cada extensão. Quando o usuário chama um arquivo, o SO invoca o programa associado à sua extensão.

1.1.2 Estrutura de Arquivos

Arquivos podem ser estruturados de várias formas. Três possibilidades são comuns:

1. Uma sequência não-estruturada de bytes. O SO não se preocupa com o que está no arquivo. Tudo que ele vê são os bytes. Qualquer significado deve ser imposto pelos programas à nível de usuário. Tanto o UNIX quanto o Windows usam essa aproximação.

   Fazer com que o SO considere os arquivos como nada mais do que sequências de bytes provê a maior flexibiliade. Programas de usuário podem colocar qualquer coisa em seus arquivos e nomeá-los de qualquer forma que achar conveniente. O SO não ajuda, mas também não atrapalha. Para usuários que querem fazer coisas incomuns, isso é muito importante. Todas as versões do UNIX (incluindo Linux e OSX) e o Windows usam esse modelo de arquivos.

2. Uma sequência de records de tamanho fixo, cada um com uma estrutura interna. I idéia central de um arquivo ser uma sequencia de records é a idéia de que a operação de leitura retorna um record e a operação de escrita substitui ou acrescenta um record. Essa técnica era usada quando cartões perfurados de 80 colunas eram comuns, e os SOs usavam arquivos que consistiam de records de 80 caracteres.

3. Uma árvore de records, não necessariamente todos do mesmo tamanho, cada um contendo um campo **chave** em uma posição fixa do record. Essa árvore é ordenada no campo chave, para permitir a busca rápida de uma determinada chave. Esse tipo de arquivo é usado em computadores mainframe para processamento de dados comerciais.

1.1.3 Tipos de Arquivos

- Vários SOs suportam vários tipos de arquivos.
- UNIX e Windows, por exemplo, possuem arquivos regulares e diretórios. O UNIX também possui arquivos especiais de caracteres e blocos.
  - **Arquivos regulares** contém informação de usuário.
  - **Diretórios** são arquivos de sistema que mantém a estrutura do sistema de arquivos.
  - **Arquivos especiais de caracteres** são relacionados à entrada/saída e são usados para modelar dispositivos seriais de I/O como terminais, impressoras e redes.
  - **Arquivos especiais de bloco** são usados para modelar discos.

- Arquivos regulares geralmente são ASCII ou binários. Arquivos ASCII contém linhas de texto. Em alguns sistemas, cada linha é terminada por um caractere de carriage return. Em outros, o caractere de line feed é usado. Em alguns sistemas (e.g., Windows), ambos são usados. Linhas não precisam ser todas do mesmo comprimento.
- A grande vantagem de arquivos ASCII é que eles podem ser mostrados e impressos como são, e podem ser editados com qualquer editor de texto. Além disso, se grandes números de programas usam arquvios ASCII para entrada e saída, é facil conectar a saída de um programa na entrada de outro (como nos pipelines de shell).
- Arquivos binários são aqueles que não são ASCII. Tentar imprimir seu conteúdo na tela geraria um monte de lixo aleatório. Geralmente eles tem alguma estrutura interna conhecida apenas pelos programas que os utilizam.
  - Um exemplo de arquivo binário seria um executável do UNIX. Apesar de tecnicamente ser apenas uma sequência de bytes, o SO vai executar um arquivo apenas se el tiver um formato apropriado, com cinco sessões: header, text, data, relocation bits e symbol tabe. O header começa com um **magic number** que identifica o arquivo como um executável (para impedir a execução acidental de um arquivo que não esteja nesse formato), seguido pelos tamanhos das várias partes do arquivo, o endereço no qual a execução começa, e alguns bits de flag. Em seguida estão os segmentos de texto e dados do programa em si. Eles são carregados em memória e relocados usando os bits de relocação. A tabela de símbolos é usada apenas para debug.
  - Outro exemplo seria um arquivo compactado, também do UNIX. Ele consiste de uma coleção de rotinas de bibliotecas compiladas mas não linkadas. Cada uma é prefaciada por um header contendo seu nome, data de criação, dono, código de proteção, e tamanho.
- Cada SO deve reconhecer pelo menos um tipo de arquivo: seu próprio arquivo executável. Alguns reconhecem mais.

1.1.4 Acesso à Arquivos

- Sistemas operacionais antigos proviam apenas um tipo de acesso a arquivos: **acesso sequencial**. Nesse tipo de sistema, um processo poderia ler todos os bytes ou records de um arquivo em orde, começando do começo, mas não poderia pular bytes ou lê-los fora de ordem. Eles podiam ser rebobinados, para poder ser lidos quantas vezes necessários. Esses arquivos eram convenientes pois o tipo de armazenamento usado era fita magnética, ao invés de disco.
- Com a popularização dos discos para armazenamento de arquivos, tornou-se possível ler os bytes ou records de um arquivo fora de ordem, ou acessá-los por chave ao invés de posição. Esse tipo de acesso se chama **acesso aleatório**, e é necessário para muitas aplicações.

1.1.5 Atributos de Arquivos

- Cada arquivo tem um nome e seus dados. AÍém disso, todso os sistemas operacionais associam outras informações a cada arquivo, como por exemplo a data e hora em que o arquivo foi modificado pela última vez e o tamanho do arquivo. Esses itens a mais são chamados de **atributos** ou **metadados**. A lista de atributos varia consideravelmente de sistema para sistema. Algumas possibilidades restão na tabela abaixo, mas existem outras.

Tabela 1: Alguns possíveis atributos de arquivos

Atributo	**Significado**
Proteção	Quem pode acessar o arquivo e de que forma
Criador	ID da pessoa que criou o arquivo
Dono	Dono atual
Read-only flag	0 para read/write; 1 para read only
Hidden flag	0 para arquivos normais; 1 p/ arquivos ocultos
System flag	0 p/ arquivos normais; 1 p/ arquivos do sistema
Archive flag	0 p/ possui backup; 1 p/ precisa de backup
ASCII/Binary flag	0 p/ arquivo ASCII; 1 p/ arquivo binário
Random Access flag	0 p/ acesso sequencial; 1 p/ acesso aleatório
Temporary flag	0 p/ normal; 1 p/ deletar arquivo ao encerrar processo.
Lock flags	0 p/ unlocked; nonzero p/ locked
Record length	Número de bytes num record
Key position	Offset da chave dentro de cada record
Key length	Número de bytes no campo chave
Creation time	Data/hora da criação do arquivo
Time of last access	Data/hora em que o arquivo foi acessado por ultimo
Time of last change	Data/hora da última modificação do arquivo
Current size	Número de bytes em um arquivo
Maximum size	Tamanho máximo que o arquivo pode atingir