ANDRÉ LUCHESI
CESAR HENRIQUE KÁLLAS

SISTEMAS OPERACIONAIS II
ESPECIFICAÇÃO CAMADA INDEPENDENTE

Pontifícia Universidade Católica de Campinas
Faculdade de Engenharia de Computação
Turma III – Grupo 9
Agosto de 2004

Especificação Camada Independente

Introdução

A visão e definição de arquivo é diferente para o Sistema Operacional e para humanos. Para o Sistema Operacional um arquivo é nada mais que vários bytes seguidos, definidos por um começo e um tamanho total. Já para os humanos, um arquivo possui uma certa abstração, porém muito facilmente definido e indentifcado, todas essas facilidades que humanos tem em lidar com arquivos ocorrem pelo tratamento dado aos arquivos pelo Sistema de Arquivos.

A camada independente tem como função tratar, dar sequência e retorno a chamadas vindas de uma camada superior no sistema operacional, camada essa muita das vezes comandada por um operador humano, um exemplo de requisição seria abrir um arquivo.

As requisições referentes ao sistema de arquivos que vamos tratar nesse trabalho são: open, read, write, tendo como base o sistema de arquivo FAT (File Allocation Table), desenvolvido e mantido pela empresa Microsoft.

O sistema de arquivos FAT trabalha com um índice para servir informações sobre os clusters existentes no disco (áreas de armazenamento), uma tabela. Essa tabela serve como referência para o Sistema Operacional, para saber se o cluster está livre ou ocupado, e toda vez que o cluster é alterado (ocupado ou desocupado), o sistema operacional deve atualizá-la.

Descrição do Sistema de Arquivos

Os arquivos são seqüência de bytes delimitados por um começo e um tamanho total gravados em um disco de dados (normalmente um disco rígido) e controlado por um sistema de arquivos.

Cada arquivo possui seu próprio descritor e um registro que contém informações sobre ele, tudo armazenado no próprio disco de dados, informações tais como:

- nome
- tamanho em bytes
- data de criação
- data da ultima modicação
- atributos de oculto/somente leitura
- localização no disco rígido

Quando é feito uma chamada de sistema para abrir um arquivo, o Sistema Operacional apenas carrega o descritor do arquivo para a memória e não o arquivo todo. Para poupar tempo e ganhar agilidade, o Sistema Operacional pode manter na memória uma certa quantidade de blocos, sendo assim o acesso fica mais rápido, tal técnica tem o nome de cache, cache de disco.

Para acessar o sistema de arquivos, o Sistema Operacional juntamente com o driver do disco, precisa saber qual tamanho do disco a ser lido, sua configuração completa, para isso, ele possui uma tabela de configuração com todos esses dados, provinientes do disco de dados.

O sistema de arquivos chamado FAT, possui três regiões:

Super Bloco: situado no bloco 0, tem como principal finalidade guardar informações gerais sobre o sistema de arquivos, veja abaixo:

- **check_number:** número que permite identificar se o sistema de arquivos é válido
- **block_size:** tamanho de cada bloco do sistema (256, 512(padrão) ou 1024 bytes)
- **fat_type:** tipo de FAT (8, 10, 12,16,32)
- **root_block:** número do primeiro bloco a que corresponde o diretório principal
- **free_block:** número do primeiro bloco da lista de blocos não utilizados

FAT: traduzido por "tabela de alocação de arquivos", situada a partir do bloco 1, é uma tabela contínua de blocos de tamanho 2^fat_type-1, que permite referenciar se o bloco está ocupado ou livre. Através do valor de free_block é possível ir direto ao primeiro bloco de dados não utilizado. Um valor de -1 numa entrada FAT marca o fim da tabela.

Blocos de dados(BFD): são utilizados para guardar os arquivos e diretórios, o número de blocos de dados é igual ao número de entradas na FAT, o primeiro bloco de dados é o diretório root (raiz). A implementação de diretórios é de certa forma idêntica à dos arquivos, a diferença reside no fato dos blocos de dados para diretórios seguirem um formato predefinido.

Em todos discos que utilizam o sistema de arquivos FAT, possuem duas cópias da FAT: uma principal e uma secundária, para backup da principal, devido à extrema importância da FAT. Caso o diretório seja perdido por qualquer motivo, os dados ainda podem ser recuperados, pois o diretório aponta apenas o primeiro cluster de cada arquivo, porém perdendo-se a FAT, não há mais como acessar os arquivos existentes, e os dados do disco serão perdidos.

Um subdiretório aponta a um cluster inicial na FAT. Assim como qualquer arquivo pode crescer, um subdiretório também. É possível criar um subdiretório com quantos arquivos você queira, pois a medida que o espaço para novas entradas de arquivo dentro do subdiretório acaba, um novo cluster é adicionado ao "arquivo" subdiretório, estando limitado somente ao espaço disponível no disco.

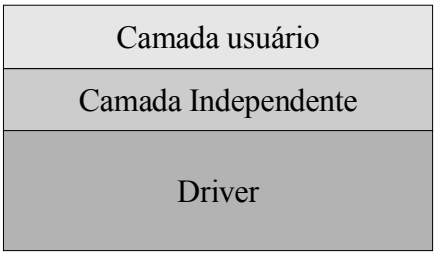
Cada subdiretório ao ser listado com o comando DIR apresenta dois arquivos: "." e "..". Esses arquivos contêm, respectivamente, a posição do subdiretório atual dentro do disco e a posição do diretório no qual esse subdiretório está incluído (diretório "pai"), de modo que o sistema operacional não se perca ao manipular subdiretórios. O diretório raiz não precisa dessas marcações, afinal não há outro diretório superior à ele e ele está em um local bem definido do disco.

Chamadas

Para o uso das chamadas usadas nesse trabalho (open, read, write) são envolvidas três camadas do Sistema Operacional: Camada referente ao operador ou usuário, Camada Independente de Dispositivo (tratada nesse trabalho) e a Camada de Driver.

Camada Referente ao Operador: é a camada em que o operador tem contato direto, enviando comando e recebendo resposta. Essa camada por sua vez se comunica com a Camada Independente de Dispositivo.

Camada Independente de dispositivos: É a camada encarregada de receber requisições da camada do operador e processá-las afim de retornar valores de acordo com as requisições recebidas. Essa camada se necessitar, pode recorrer a uma camada abaixo dela, a camada de driver.



Através da camada do operador são chamadas operações como:

Open: open <nome_arquivo> <modo>

Função: int ffopen(char *filename, int mode)

Abre ou cria o arquivo descrito pelo nome <filename>, dependendo do modo passado como parâmetro. Veja os modos:

Modos de abertura de um arquivo		
Modo	**Descrição**	**Operação**
0	Acesso apenas de leitura	Abre o arquivo para leitura, se não existir o arquivo, a operação falha.
1	Acesso apenas de escrita	Abre o arquivo sobrescrevendo-o se já existir, criando um novo e sem conteúdo.
2	Acesso de leitura e escrita	Abre o arquivo ou cria um novo se não existir.

Após a chamada open, a camada independente de dispostivo interpreta a mesma e retorna um valor numérico, descrito logo abaixo, esses retornos identificam se houve sucesso, falha ou algum imprevisto. Sendo assim o usuário ou a camada do usuário pode reconhecer o retorno e tomar providências se assim necessário.

Retornos possíveis da função ffopen	
Código retorno	**Descrição**
inteiro positivo	Sucesso, retorna um número inteiro positivo que será usado como indice de manipulação do arquivo.
-1	Impossível abrir arquivo, arquivo não existe
-2	Erro ao iniciar estruturas de manipulação do arquivo
-3	Não foi possível comunicar com o driver
-4	Não foi possível acessar o BFD
-5	Lista de descritores cheia

Pode-se notar que a função ffopen é de simples operação por parte da camada do usuário ou operador, visto que todo o trabalho fica com a camda indepentende de dispositivo, que trata a função e se comunica com a camada de driver, se preciso.

Note também que há um retorno de um número que será usado como indice para outras funções de manipulação do arquivo, como ffread e ffwrite.

Read: read <indice> <buffer> <tamanho>

Função : int ffread(int indice, char *buffer, int tamanho)

A função ffread recebe três parâmetros para a leitura de um arquivo, o primeiro parâmetro seria a indicação do arquivo a ser lido, porém, não é o nome do arquivo, é um índice que foi concedido com a utlização da função ffopen anteriormente, sendo assim, quando for ler

um arquivo, necessariamente deve-se abrí-lo antes, isso é uma regra. O segundo parâmetro chamado de buffer, seria o local ou melhor, uma variável para armazenar os dados lidos pela função descrita e o terceiro parâmetro é a quantidade de dados que serão lidos do arquivo indicado. Sendo assim, o ponteiro posicionará até *tamanho* lendo todo o conteúdo por onde passar.

O retorno da função será de acordo com o quadro abaixo:

Retornos possíveis da função ffread	
Código retorno	**Descrição**
inteiro positivo	Sucesso, retorna um número inteiro positivo que é a quantidade de bytes lidos.
0	Se o ponteiro já estiver no final do arquivo, esse é um caso típico de arquivo vazio, o retono nesse caso é visto como sucessivo.
-1	Impossível ler arquivo, corrompido.
-2	Posição inválida do ponteiro (posição > tamanho do arquivo).
-3	Dispositivo inválido.
-4	Arquivo não encontrado.

Todo o tratamento da função ffread será feito na camada independente do usuário, que trata essa chamada e retorna esses possíveis valores acima.

Write: write <indice> <buffer> <tamanho>

Função: int ffwrite(int indice, char *buffer, int tamanho)

A função ffwrite recebe três parâmetros e sua finalidade é gravar dados em um arquivo designado. O primeiro parâmetro é o indice, podemos entender esse índice como sendo o indicador retornado pela chamada da função ffopen, esse indicador está apontando para um arquivo presente no disco, porém já mapeado na memória pela função ffopen.

O segundo parâmetro, buffer, é o local ou melhor, a variável aonde estpa armazenados os dados para serem gravados no arquivo (indice) indicado, o tamanho dos dados a serem gravados é indicado pelo terceiro parâmetro da função.

Note, a escrita é sequencial, sendo assim, se o ponteiro do arquivo apontar para o começo do arquivo e esse possuir dados, esses serão sobrescrevidos, a função grava e não apenas inseri, para haver inserção, basta que o ponteiro aponte para área não armazenada de dados.

O retorno como sempre é seguido de acordo com a tabela:

Possíveis retornos da função ffwrite	
Código retorno	**Descrição**

Possíveis retornos da função ffwrite	
inteiro positivo	Sucesso, retorna um número inteiro positivo que é a quantidade de bytes gravados.
-1	Impossível ler arquivo, arquivo inexistente, corrompido, etc.
-2	Posição inválida do ponteiro (posição > tamanho do arquivo).
-3	Dispositivo inválido.
-4	Disco cheio.
-5	Tamanho máximo do arquivo atingido.
-6	Arquivo não encontrado.

Driver

É a camada que manipula os dispositivos físicos presentes na máquina, sendo assim, quando um sistema operacional possuir o driver específico e desenvolvido para o dispositivo físico, um disco de dados por exemplo, ele poderá controlá-lo de maneira correta, fazendo com que seu funcionamento seja positivo com a requisição recebida.

Os drivers são vários e possuem versões de desenvolvimento, um dos principais fatores presentes nos drivers é sua compatibilidade com o Sistema Operacional, visto que, um driver feito para um Sistema Operacional nem sempre funcionará em outro diferente.

Quando um fabricante desenvolve um dispositivo, ele fica encarregado de desenvolver o driver compatível para que ele funcione no Sistema Operacional, contudo, o fabricante também necessita saber o funcionamento do Sistema Operacional, bem como as *Chamadas de Sistema* existentes e seus parâmetros. Um problema que pode ocorrer, é quando o dispositivo não possui driver, a solução viável para esse caso é utilizar drivers genéricos para o disposito ou em casos mais exóticos, um programador qualificado desenvolve um driver para o mesmo, sem ao menos saber como foi construído o dispositivo.

A camada independente, tratada nesse trabalho, faz referências diretas quando preciso a camada de driver, que é outro processo que estará rodando concorrentemente.

O driver citado assima, aguarda requisições da camada independente, quando recebe, lança uma nova linha de execução requerendo junto a controladora a ação deferida ficando no aguardo até o retorno da controladora, como não teremos uma controladora no nosso caso de teste, o driver fará a simulação de uma.

Para uma requisição da camada independente, são necessários alguns parâmetros a serem passados para o driver:

- **Índice do cluster:** refere-se a qual cluster a camada independente de dispositivos está se referindo para realizar uma leitura ou escrita.
- **Operação:** Indica a natureza da operação que pode ser 0 (read), 1 (write), 2 (seek), 3 (recalibrate) e 4 (reset).

- **Endereço de memória:** No caso da operação ser de escrita, indica o endereço da memória principal referente ao dado a ser gravado no disco. No caso da operação ser de leitura, refere-se em que lugar da memória principal será gravado o dado lido do disco.
- **Número do processo da requisição (PID):** Este número serve como identificação da requisição para que se possa, posteriormente à realização da operação requerida, retornar o status final da operação para o processo que a requisitou.

Após a requisição, o driver analisa se é possível comprí-la antes de ocorrer o deadline, sendo possível, a requisição é tratada e o driver retorna um valor sobre o tratamento da requisição:

Possíveis retornos da chamada ao driver	
Código retorno	**Descrição**
Inteiro positivo	Sucesso, retorna o tamanho do bloco lido
-1	Não foi possivel consultar o arquivo em disco.

- Erro -1: Não foi possível localizar o arquivo em disco.

Processos

Todas as requisições citadas nesse trabalho, bem como as camadas que os tratam funcionam quando carregados na memória. De fato, Sistemas Operacionais Modernos existentes manipulam e gerenciam vários processos na memória, não ficando para o operador essa tarefa.

Porém, o ponto mais importante é que, podemos utilizar mais de uma chamada (open, read, write) ao mesmo tempo, perante o operador, mas a Camada Independente tem de estar pronta a tratar mais de uma requisição recebida, utlizando-se de threads e controles de execução, como semáfaros.

Todos processos possuem um PCB que os representam perante ao Sistema Operacional, nesse PCB são armazenadas informações sobre o processo, tais como:

- **Estado do processo:** processo novo, em execução, pronto para executar, em espera, suspenso
- **Contador do programa:** indica endereço da próxima instrução a ser realizada
- **Registradores de CPU:** função principal, armazenar os dados dos registradores usados pelo processo, quando o processo é interrompido na sua execução.
- Informações de escalonamento da CPU, gerência de memória, contabilização, I/O entre outros.