CENTRO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE COMPUTAÇÃO
SISTEMAS OPERACIONAIS

Sistemas Linux:

Gerenciamento de Memória
Sistema de Arquivos
Entrada e Saída
Estrutura de Redes

Alunos:

André Ricardo Gonçalves
Daniel César Romano Luvizotto
Heber A. A. Nascimento
Luiz Gustavo Andrade dos Santos
Luiz Gustavo Castilho Martins

Prof. Fabio Sakuray

LONDRINA - PR
2007

André Ricardo Gonçalves
Daniel César Romano Luvizotto
Heber A. A. Nascimento
Luiz Gustavo Andrade dos Santos
Luiz Gustavo Castilho Martins

Sistemas Linux:

Gerenciamento de Memória
Sistema de Arquivos
Entrada e Saída
Estrutura de Redes

Trabalho apresentado à Universidade Estadual de Londrina, como parte de requisito de avaliação do 2^o Bimestre da disciplina de Sistemas Operacionais, do curso de Ciência da Computação sobre a orientação da Profa. Fabio Sakuray.

LONDRINA - PR
2007

Sumário

Lista de Figuras

Capítulo 1

Introdução

O Linux, é um sistema operacional livre e de código aberto, ou seja, não necessita pagar licença para utilizá-lo, e seu código pode ser facilmente encontrado na internet, para possíveis alterações, caso necessário. O Linux é fortemente baseado no Sistema Operacional UNIX.

Ele foi criado por Linus Torvalds, o qual desenvolveu um kernel e solicitou a ajuda de internautas para ajudá-lo a terminar o projeto. Isso deu muito certo e a partir disso milhares de colaboradores de todo mundo contribuíram e contribuem para o desenvolvimento do Linux.

O Linux é basicamente o *kernel*, que consiste no núcleo do sistema operacional, o qual tem a função de comunicação com o hardware. A partir de um kernel pode-se utilizar vários aplicativos livres, este pacote consistindo de kernel e aplicativos é conhecido como distribuição. Existem várias distribuições Linux, dentre as mais conhecidas estão o Slackware, Fedora, Ubuntu, Suse, Debian, Conectiva, dentre outras.

Este trabalho visa abordar de forma clara e consisa, algumas das principais funções exercidas pelo Sistema Operacional Linux, como a gerência de memória, sistema de arquivos, entrada e saída e estrutura de redes.

Capítulo 2

Gerência de Memória

Segundo [SILBERSCHATZ] sistema Linux possui dois componentes de gerência de memória. Um deles trata da alocação e liberação de memória físíca, por exemplo: páginas, grupo de páginas e pequenos blocos de memória. E o outrotrata da memória virtual.

2.1 Gerência de Memória Física

Dentro do gerenciamento de memória física do sistema Linux encontramos dois tipos de gerentes, um é o que faz a gerencia de memória física no kernel que é conhecido como alocador de páginas, e o outro é o que realiza alocação de comprimento variável que é conhecido como **kmalloc**.

Iniciamos com o estudo do alocador de páginas. Este é responsável pela alocação e liberação de todas as paginas físicas e capaz de alocar intervalos de paginas fisicamente contíguas sob demanda. Tal alocador utiliza o algoritmo **Buddy-heap** para localizar as paginas físicas disponíveis. Cada região da memória alocável tem uma parceira adjacente, ou buddy, e sempre que duas regiões parceiras são alocadas para constituir uma maior região para um determinado uso, após o fim de seu uso são liberadas. Cada região pode se combinar com sua parceira para constituir regiões maiores.

Mas não é todos os casos que se necessita de uma combinação de regiões para alocação, há casos em que se necessita de um pequeno espaço, então é realizado o oposto da combinação, ou seja, uma região livre será subdividida em duas parceiras, a fim de fornecer o espaço adequado. Usa-se listas encadeadas independentes para registrar as regiões de memória livre de cada tamanho permitido, no sistema Linux o menor tamanho alocável segundo esse mecanismo é uma página física. A figura 2.1 mostra um exemplo de alocação **buddy-heap**: uma região de 4Kb está sendo alocada, mas a menor região disponível é 16 Kb, então a região é subdividida recursivamente até que seja disponibilizada uma parte do tamanho desejado.

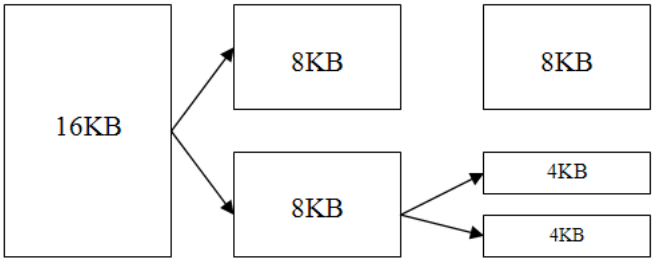

Figura 2.1: Divisão da memória em um buddy-heap

De maneira geral todas as alocações de memória no kernel Linux ocorre de maneira estática, por drivers que reservam uma área de memória contígua, ou seja, espaços seqüentes na inicialização do sistema, ou dinamicamente, pelo alocador de páginas.

Como mencionando anteriormente o sistema Linux possui dois tipos de gerentes de memória, uma já foi citada, agora concentremo-nos no segundo tipo. As funções do kernel não necessariamente têm de utilizar o mesmo alocador básico para reservar memória. Diversos subsistemas de gerencia de memória especializados lançam mão do alocador de páginas subjacentes para gerenciar seu próprio pool de memória. Os mais importantes são o sistema de memória virtual, o alocador de comprimento variável **kmalloc** e os dois caches de dados persistentes do kernel, o cache de buffers e o cache da páginas.

Vários componentes do sistema operacional Linux necessitam alocar paginas à vontade, mas freqüentemente são solicitados blocos de memória menores. O kernel proporciona um alocador adicional para solicitações de tamanho arbitrário, em que o tamanho do pedido não necessariamente é conhecido antecipadamente podem ser de apenas alguns bytes, em vez de uma pagina inteira. Analogamente à função malloc da linguagem C, esse serviço **kmalloc** aloca páginas inteiras sob demanda, mas, em seguida, divide-as em pedaços menores. O kernel mantém o conjunto de listas de páginas em uso pelo serviço **kmalloc**, em que todas as páginas de uma dada lista foram divididas em pedaços de um tamanho especifico. A alocação de memória envolve determinar lista apropriada e a tomada do primeiro pedaço disponível na lista ou na alocação de uma nova pagina e sua divisão.

Segundo [SILBERSCHATZ] os dois alocadores, **kmalloc** e de páginas, são à prova de interrupções. Uma função que deseje alocar memória passa uma prioridade de pedido para a função de alocação. As rotinas de interrupção empregam uma propriedade atômica, que garante a solicitação seja satisfeita ou, se não houver mais memória, falhe de imediato. Em compensação, os processos de usuário normais que requerem uma alocação de memória começarão a tentar encontrar memória existente para liberar e permanecerão bloqueados até que a memória seja disponibilizada. A prioridade de alocação pode também ser utilizada para especificar que a memória é necessária para DMA, apara uso em arquitetura tais como a de PC, em que certos pedidos de DMA não são suportadas em todas as paginas

de memória física.

As regiões de memória reivindicadas pelo sistema **kmalloc** são alocadas permanentemente até receberem uma liberação explicita. O sistema **kmalloc** não pode relocar ou reivindicar essas regiões em resposta a uma eventual escassez de memória.

Os outros três subsistemas principais que tratam de sua própria gerencia de páginas físicas são intimamente relacionados, são o cache de buffers, o cache de paginas e o sistema de memória virtual. O cache de buffers é o cache principal do kernel para dispositivos orientados a bloco tais como unidade de disco, que é o principal mecanismo por meio do qual é efetuado o I/O desse dispositivo. O cache de páginas mantém em cache páginas inteiras de conteúdo de arquivos, e não se limita a dispositivos de bloco, podem também fazer cache de dados de rede, e é utilizado tanto pelos sistemas de arquivos baseados em disco nativos do Linux quanto pelos sistemas de arquivos em rede NFS. O sistema de memória virtual gerencia o conteúdo do espaço de endereçamento virtual de cada processo.

Esses três sistemas interagem intimamente entre si. A leitura de uma página de dados para o cache de páginas também podem ser mapeadas no sistema de memória virtual, caso um processo tenha mapeado um arquivo em seu espaço de endereçamento. O kernel mantém um contador de referências em cada página de memória física, de modo que páginas compartilhadas por dois ou mais desses subsistemas possam ser liberadas quando não se encontrarem mais em uso em lugar algum.

2.2 Memória Virtual

O sistema de memória do Linux é responsável pela manutenção do espaço de endereçamento visível para cada processo. Ele cria páginas de memória virtual sob demanda e gerencia a carga dessas páginas do disco ou sua descarga de volta para o disco conforme necessário. No Linux, o gerente de memória virtual mantém duas perspectivas do espaço de endereçamento de um processo: como um conjunto de regiões separadas e com um conjunto de páginas.

Segundo [SILBERSCHATZ] a primeira visão de um espaço de endereçamento é a visão lógica, descrevendo as instruções recebidas pelo sistema de memória virtual referente ao layout do espaço de endereçamento. Nessa visão, o espaço de endereçamento consiste em um conjunto de regiões não-superpostas, cada uma das quais representando um subconjunto continuo e alinhado por páginas do espaço de endereçamento. Cada região é descrita internamente por uma única estrutura **vm_area_struct**, que define as propriedades da região, inclusive as permissões de leitura, escrita e execução do processo na região, assim como informações sobre todos os arquivos associados com tal região. As regiões para cada espaço de endereçamento são encadeadas em uma árvore binária balanceada, de modo a

possibilitar uma pesquisa rápida da região correspondente a qualquer endereço virtual.

O kernel mantém também uma segunda visão, física, de cada espaço de endereçamento. Essa visão fica armazenada nas tabelas de páginas do hardware para o processo. As entradas da tabela de páginas determinam à localização atual exata de cada página de memória virtual, quer esteja em disco, quer em memória física. A perspectiva física é gerenciada por um conjunto de rotinas, invocado a partir dos tratadores de interrupção de software do kernel sempre que um processo tenta acessar uma página que não se encontra pressente no momento nas tabelas de páginas. Cada **vm_area_struct** na descrição do espaço de endereçamento contém um campo que aponta para uma tabela de funções que implementam as funções básicas de gerencia de páginas para qualquer região dada da memória virtual. Todos os pedidos de leitura ou escrita em uma pagina não disponível acabam sendo enviados para o tratador apropriado na tabela de funções da **vm_area_struct**, de modo que as rotinas de gerencia de memória centrais não precisam conhecer os detalhes da gerencia de cada tipo possível de região de memória.

2.2.1 Regiões de memória virtual

O Linux implementa uma série de diferentes tipos de regiões de memória virtual. A primeira propriedade que caracteriza um tipo de memória virtual é o tipo de armazenamento secundário associado a essa região, esse armazenamento descreve a origem das páginas de uma região. A maior parte das regiões de memória tem por armazenamento de apoio a um arquivo, ou nada. Uma região sem armazenamento de apoio é a modalidade mais simples de memória virtual e representa a memória de demanda zero, quando um processo procura ler uma página em tal região, tem como retorno, simplesmente, uma página de memória cheia de zeros. [SILBERSCHATZ]

Uma região que tem um arquivo como armazenamento de apoio funciona como visor para uma seção do arquivo, sempre que o processo tenta acessar uma página dentro daquela região, a tabela de páginas é preenchida com o endereço de uma página dentro do cache de páginas do kernel que corresponda ao deslocamento apropriado no arquivo. A mesma página de memória física é utilizada tanto pelo cache de página quanto pelas tabelas de páginas do processo, de forma que qualquer mudança efetuada no arquivo pelo sistema de arquivos fica visível de imediato para todos os processos que tiverem mapeado aquele arquivo em seu espaço de endereçamento. Qualquer quantidade de processos diferentes pode mapear a mesma região do mesmo arquivo, e todas acabarão utilizando a mesma página de memória física para esse fim.

Uma região de memória virtual também é definida por sua reação a escrita. O mapeamento de uma região para o espaço de endereçamento do processo pode ser privado, o paginador detectará a necessidade de uma operação copy-on-write para manter as alterações

locais com relação ao processo. Por outro lado, as escritas em uma região compartilhada resultam na atualização do objeto mapeado para tal região, de modo de a mudança ficará visível de imediato para qualquer outro processo que estiver mapeado tal objeto.

2.2.2 Tempo de vida de um espaço de endereçamento virtual

Há exatamente duas situações em que o kernel cria um novo espaço de endereçamento virtual, na execução, por um processo, de um novo programa através da chamada ao sistema exec e na criação de um novo processo, por meio da chamada ao sistema fork. O primeiro caso é fácil, quando um novo programa é executado, o processo recebe um espaço de endereçamento novo, completamente vazio. Cabe as rotinas carregar o programa para ocupar o espaço de endereçamento com regiões de memória virtual [SILBERSCHATZ].

No segundo caso, a criação de um novo processo via fork implica na criação de uma cópia integral do espaço de endereçamento virtual do processo existente. O kernel copia os descritores da **vm_area_struct** do processo pai e, em seguida, cria um novo conjunto de tabelas de páginas para o filho. As tabelas de páginas do pai são copiadas diretamente para as do filho, com o contador de referência de cada página afetada sendo incrementada, assim sendo, após o fork, pai e filho compartilham as mesmas páginas físicas da memória em seus espaços de endereçamento.

Ocorre um caso especial quando a operação de cópia atinge uma região de memória virtual com mapeamento privado. Todas as paginas dentro dessa região em que i processo pai tiver realizado alguma escrita são provadas, e alterações subseqüentes a essas páginas realizadas pelo pai ou filho não poderão atualizar a página dessas regiões forem copiadas, serão configuradas como sendo de leitura e assinaladas para cópia em caso de escrita (copy-on-write). Entretanto se essas páginas não forem modificadas por nenhum dos dois processos, ambos compartilharão a mesma pagina de memória física. Se caso algum deles tentar modificar uma página copy-on-write, o contador de referencia na página será verificado. Se a página ainda estiver compartilhada, o processo copiará o conteúdo da pagina para uma nova página de memória física e fará uso dessa sua cópia. Esse mecanismo garante que as páginas privadas de dados serão compartilhadas sempre que possível, as cópias só são realizadas quando absolutamente necessário.

2.2.3 Swapping e paginação

Uma importante tarefa para um sistema de memória virtual é relocar as páginas da memória física para o disco quando há necessidade de memória. Os primeiros sistemas UNIX realizavam essa relocação fazendo o swapping do conteúdo de processos inteiros e uma só vez, mas os UNIXes de hoje baseiam-se ais na paginação, a movimentação de

paginas individuais de memória virtual entre a memória física e o disco. O Linux não implementa o swapping de processos integral ele utiliza exclusivamente o mecanismo mais recente de paginação [SILBERSCHATZ].

O sistema de paginação pode ser dividido em duas seções. Em primeiro lugar, o algoritmo de políticas, que decide que página gravar no disco e quando fazê-lo. Depois, o mecanismo de paginação, que realiza a transferência e pagina os dados de volta para a memória física quando se tornam necessários novamente.

A política de descarga de páginas do Linux emprega uma versão modificada do algoritmo do relógio (segunda chance). No Linux é utilizado um relógio de passagens múltiplas, e cada página tem uma idade que é ajustada a cada passagem do relógio. A idade é, mais precisamente, uma medida da juventude da página, ou de quanta atividade ela tem visto recentemente. Páginas acessadas com freqüéncia atingirão um valor de idade mais elevado, mas a idade de páginas que não são acessadas com freqüéncia decresce a cada passagem. Essa atribuição de idade permite ao paginador selecionar páginas para descarregar com base em uma política de utilização com menor freqüéncia (LFU - Least Frequently Used).

O mecanismo de paginação suporta a paginação tanto em partições e dispositivos de troca dedicados quanto em arquivos normais, muito embora o swapping para um arquivo seja significativamente mais lento devido ao custo adicional provocado pelo sistema de arquivos. Os blocos são alocados a partir dos dispositivos de troca conforme um mapa de bits de blocos usados, mantido todo o tempo na memória física. O alocador emprega um algoritmo next-fit para tentar gravar páginas em carreiras contínuas de bloco de disco, visando um melhor desempenho. O alocador registra o fato de que uma página foi descarregada para o disco por meio de um recurso das tabelas de páginas dos processadores atuais, ajusta-se o bit de página ausente (page-not-present-bit) da entrada da tabela de páginas, permitindo que o restante da entrada da tabela de páginas seja preenchido com um índice, identificando o local da escrita da página.

2.2.4 Memória virtual do kernel

O Linux reserva para seu próprio uso interno uma região do espaço de endereçamento virtual de cada processo, constante e dependente da arquitetura. As entradas da tabela de páginas que mapeiam nessas páginas do kernel são marcadas como protegidas, de modo que as páginas não são visíveis nem modificáveis quando o processador está funcionando em modo de usuário. Essa área de memória virtual do kernel contém duas regiões. A primeira seção é uma área estática, que contém referências da tabela de páginas a cada página física da memória disponível no sistema, de forma que ocorre uma translação simples dos endereços físicos para os virtuais quando o código do kernel é executado. O

núcleo do kernel, bem como todas as páginas alocadas pelo alocador de páginas normal, reside nessa região.

O restante da seção de espaço de endereçamento reservada do kernel não se destina a nenhum fim especifico. As entradas de tabelas de páginas nesse intervalo de endereçamento podem ser modificadas pelo kernel de maneira a apontar qualquer outra área da memória, conforme desejado. O kernel proporciona uma par de maneiras a apontar qualquer outra área da memória, conforme desejado. O kernel proporciona um par de recursos que permitem que os processos façam uso dessa memória virtual. A função **vmalloc** aloca um número arbitrário de páginas físicas de memória, e mapeia-os em uma única região de memória virtual do kernel, possibilitando a alocação de grandes pedaços contíguos de memória, mesmo que não haja um número suficiente de páginas físicas adjacentes livres para satisfazer o pedido. A função vremap mapeia uma seqüência de endereços virtuais para que apontem para uma área de memória utilizada por um driver de dispositivo, para I/O mapeado em memória. [SILBERSCHATZ]

2.3 Execução e carga de programas do usuário

A execução de programas do usuário pelo kernel Linux é desencadeada por uma chamada ao sistema exec. Essa chamada ordena que o kernel execute um novo programa dentro do processo atual, sobrepondo totalmente o contexto de execução atual com o contexto inicial do novo programa. A primeira tarefa desse serviço do sistema é verificar se o processo que efetua a chamada tem permissão adequada sobre o arquivo a executar. Uma vez verificada essa questão, o kernel invoca uma rotina de carga para começar a executar o programa. O carregador não necessariamente carrega o conteúdo do arquivo de programa para a memória física, mas ao menos configura o mapeamento do programa na memória virtual.

Não há uma rotina exclusiva no Linux para carregar um novo programa. Em vez disso, o Linux mantém uma tabela de possíveis funções carregadoras e atribui a cada uma dessas funções a oportunidade de tentar carregar o arquivo quando é feita uma chamada ao sistema exec. A razão inicial dessa tabela de carregadores foi que, entre o lançamento dos kernels 1.0 e 1.2, o formato padrão dos arquivos binários do Linux foi alterado. Os primeiros kernels Linux compreendiam o formato a.out para arquivos binários um formato relativamente simples comum em sistemas UNIX mais antigos. Os sistemas Linux mais recentes empregam o formato ELF, mais moderno, agora suportado pelas implementações mais atuais do UNIX. O formato ELF apresenta algumas vantagens sobre o formato a.out, incluindo flexibilidade e extensibilidade, podem ser acrescentadas novas seções a um binário ELF (por exemplo, para adicionar informações extras de depuração), sem que

as rotinas de carga fiquem confusas. Possibilitando o registro de várias rotinas de carga, o Linux pode suportar facilmente os formatos binários ELF e a.out em um mesmo sistema em execução.

2.3.1 Mapeamento de programas na memória

O carregamento de um arquivo binário para a memória física não é realizado pelo carregador binário no Linux. Em vez disso, as páginas do arquivo binário são mapeadas em regiões da memória virtual. Só quando o programa tenta acessar uma determinada página é que uma falta de página resulta em seu carregamento na memória física.

É responsabilidade do carregador binário do kernel configurar o mapeamento inicial de memória. Um arquivo binário em formato ELF é constituído por um cabeçalho, seguido de várias seções alinhadas por página. O carregador ELF funciona lendo o cabeçalho e mapeando as seções do arquivo em regiões separadas da memória virtual.

A figura logo abaixo mostra o layout típico das regiões de memória configurada pelo carregador ELF. O kernel situa-se em uma região reservada, em uma das extremidades do espaço de endereçamento, em sua própria região privilegiada de memória virtual, inacessível aos programas normais de modo usuário. O restante de memória virtual encontra-se disponível para as aplicações, as quais podem utilizar as funções de mapeamento do kernel para criar regiões que mapeiam uma parte de um arquivo ou que permanecem disponíveis para dados de aplicações. A figura 2.2 apresenta um layout de memória para programa ELF.

A tarefa do carregador consiste em configurar o mapeamento inicial de memória a fim de permitir que a execução do programa tenha início. As regiões que precisam ser inicializadas incluem a pilha e as regiões de dados e texto do programa.

A pilha é criada no alto da memória virtual de modo usuário e cresce para baixo, em direção aos endereços de numeração menor. Ela inclui cópias dos argumentos e das variáveis de ambiente atribuídos ao programa na chamada ao sistema exec. As demais regiões são criadas perto da extremidade inferior da memória virtual. As seções do arquivo binário que contêm texto de programa (instruções executáveis) ou dados de leitura são mapeadas na memória como região protegida contra escrita. Os dados inicializados alteráveis são mapeados em seguida, depois, todos os dados não-inicializados são mapeados como uma região privada de demanda zero.

Imediatamente além dessas regiões de tamanho fixo encontra-se uma região de tamanho variável que os programas podem expandir conforme a necessidade, a fim de manter dados alocados em tempo de execução. Cada processo possui um ponteiro, brk, que aponta a extensão da região de dados no momento, assim, os processos podem estender ou contrair a região do seu brk mediante a uma única chamada ao sistema.

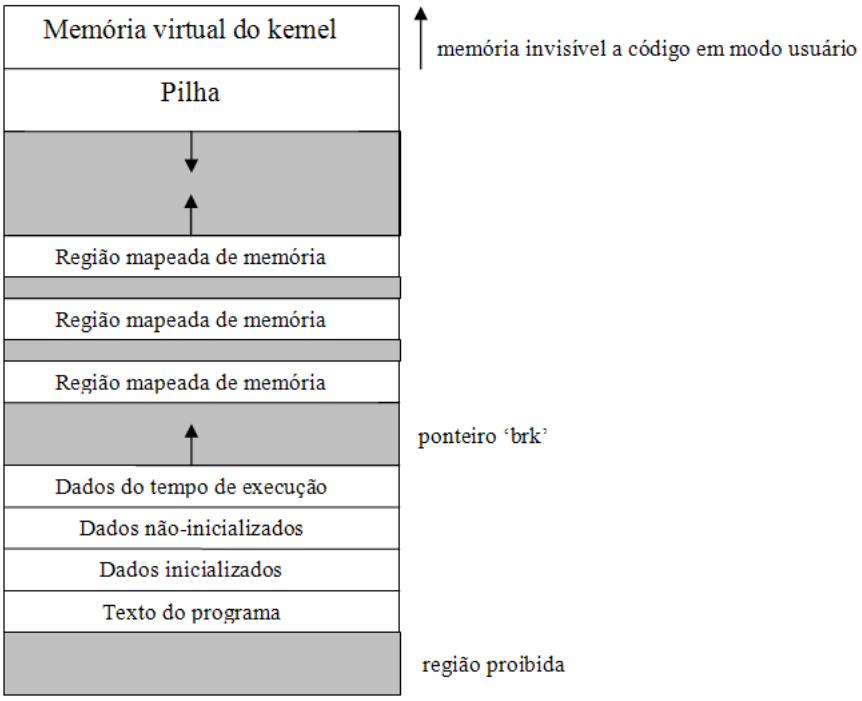

Figura 2.2: Layout de memória para programa ELF

Uma vez configurados esses mapeamentos, o carregador inicializa o registrador do contador do programa do processo, com o ponto de partida registrado no cabeçalho ELF, e o processo pode ser escalonado [SILBERSCHATZ].

2.3.2 Linking estático e dinâmico

Uma vez carregado o programa e iniciada sua execução, todo o conteúdo necessário do arquivo binário terá sido carregado no espaço de endereçamento virtual do processo. Entretanto, a maioria dos programas também precisa executar funções a partir das bibliotecas do sistema, essas funções de biblioteca também tem de ser carregadas. No caso mais simples, quando um programador constrói uma aplicação, as funções de biblioteca necessárias são incorporadas diretamente no arquivo binário executável do programa. Tal programa é "linkado"estaticamente ás suas bibliotecas, e a execução dos executáveis linkados estaticamente pode ter início imediatamente após sua carga [SILBERSCHATZ].

A principal desvantagem do método de linking estático é que todo programa gerado tem de conter cópias de exatamente as mesmas funções de biblioteca do sistema comuns. É muito mais eficiente, em termos tanto de memória física como de uso de espaço em disco, carregar as bibliotecas do sistema uma única vez. O linking dinâmico possibilita a ocorrência dessa carga única.

O Linux implementa o linking dinâmico em modo de usuário por meio de uma biblioteca de linking especial. Todos os programas linkados dinamicamente contêm uma

pequena função linkada estaticamente, chamada quando o programa é iniciado. Essa função estática apenas mapeia a biblioteca de linkagem na memória e executa o código que tal função contém. A biblioteca de linkagem determina a lista de bibliotecas dinâmicas requeridas pelo programa e os nomes das variáveis e funções necessárias dessas bibliotecas, lendo as informações contidas em seções do binário ELF. Em seguida, mapeia as bibliotecas no meio da memória virtual e resolve as referências aos símbolos nelas contidos. Não importa exatamente onde na memória essas bibliotecas compartilhadas vêm a ser mapeadas: são compiladas em código independente de posição (PIC - Position-Independent Code), que pode ser executado em qualquer endereço na memória.

Capítulo 3

Sistema de Arquivos

Segundo [SILBERSCHATZ] um sistema de arquivos é quem realiza o mecanismo de armazenamento e acesso aos dados e programas que pertencem tanto ao sistema operacional quanto aos usuários. É ele também que controla a proteção dos arquivos, no qual é importante para sistemas multiusuários.

Como o Linux é totalmente baseado no Unix, muitos de suas implementações são herdadas das implementações do Unix. Os sistemas de arquivos do Linux é fortemente baseado nos sistemas de arquivos do Unix, em alguns os modelos são os mesmos.

Os arquivos são manipulados diferentemente em cada sistema operacional e em cada sistema de arquivos, o Linux faz distinção entre maiúscula e minúscula, normalmente o nome de um arquivo possui o nome e a extensão, separados por '.' (ponto), no Linux um arquivo pode ter mais de uma extensão, como: 'exemplo.tar.gz', um arquivo que resultou a união de outros (.tar) e posteriormente ocorreu uma compactação (.gz). Outro ponto relevante do Linux, assim como no Unix, ele enxerga tudo como arquivos, os drivers dos dispositivos, as pastas, no qual cada arquivo é um *inode*. Apresentaremos formalmente o *inode* no decorrer deste trabalho.

O Linux diferencia de muitos outros sistemas operacionais pelo seu suporte para uma gama de sistema de arquivos, alguns exemplos são: *Minix*, *proc*, *iso9660*, *msdos*, *ext2*, *ext3*, *reiserfs* além de muitos outros. Neste trabalho apresentaremos os sistemas de arquivos ext3 e reiserfs, por serem os mais utilizados atualmente.

Em um mesmo sistema Linux, pode ser utilizado vários sistemas de arquivos diferentes. Cada um destes sistemas de arquivos é montado sobre um diretório, criando neste diretório um ponto de montagem deste sistema de arquivos. Pode-se montar vários sistemas de arquivos em vários pontos de montagens, estes por sua vez pode ser até um diretório de uma máquina remota.

Para que o Linux possa manipular esta gama de sistemas de arquivos, ele utiliza um Virtual File System (VFS), que gerencia estes diversos sistemas de arquivos e seus res-

pectivos pontos de montagem, dando ao usuário final a impressão de um único sistema de arquivos. Apresentaremos os VFS ao decorrer deste trabalho, para isso deve-se introduzir alguns conceitos, os quais são apresentados abaixo.

3.1 Inode

Um *inode* é uma estrutura que armazena todos as informações e os dados de um arquivo. Segundo [SILBERSCHATZ] o *inode* contém os identificadores dos usuários e do grupo ao qual este arquivo pertence, além das datas de última modificação e último acesso, o tamanho do arquivo em bytes, o número de hard links ao arquivo e o tipo de arquivo, se é um diretório, um link simbólico, driver, um arquivo simples, etc.

Um exemplo de um *inode* é apresentado na figura 3.1, na qual podemos observar as características descritas acima.

Os *inodes* possuem ponteiros de blocos de discos, para armazenar os dados do arquivo. Estes ponteiros apontam para blocos, os quais são divididos em quatros tipos:

1. blocos diretos;
2. blocos indiretos;
3. blocos indiretos duplos;
4. blocos indiretos triplos.

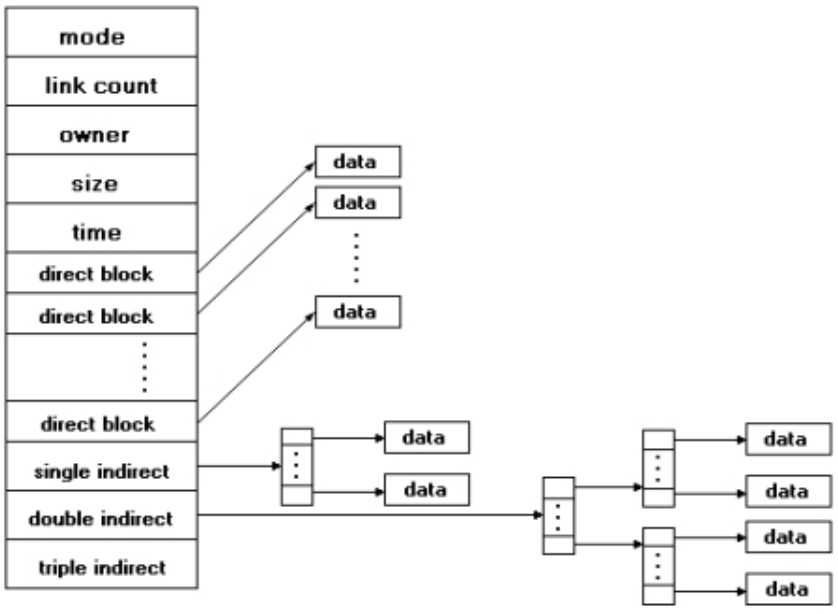

Figura 3.1: Exemplo de um Inode utilizado pelo Linux

Por exemplo, um *inode* com 15 ponteiros para blocos, os primeiros 12 ponteiros apontariam para blocos de dados, caso o arquivo seja pequeno, estes blocos seriam suficientes para armazenar tais dados. Os 3 ponteiros seguintes apontariam para blocos indiretos, ou seja, blocos que não contém dados, mas sim endereços de blocos de dados, os quais armazenam endereços de blocos de dados reais.

Além disso possui dois ponteiros finais, um para blocos indiretos duplos e outro para blocos indiretos triplos. Os blocos indiretos duplos apontam para blocos de ponteiros, onde cada ponteiro aponta para outro bloco de ponteiros, os quais estes sim apontam para blocos de dados reais. Os blocos indiretos triplos trabalham com a mesma idéia dos blocos indiretos duplos, apenas diferenciam por sua tripla indireção [SILBERSCHATZ]. Se neste exemplo fosse utilizado um bloco de 4K teríamos um arquivo de no máximo 4GB.

Para que os sistema operacional saiba qual *inode* ele deve buscar, os *inodes* são identificados através de um número, o qual cada *inode* possui um número distinto. O VFS utiliza dois parâmetros para acessar um *inode* (partição , número do inode), por meio disto podemos observar que a cada partição a identificação dos *inodes* é inicializada novamente, pois podem haver sistemas de arquivos diferentes em diferentes partições.

3.2 Virtual File System

O Virtual File System (VFS) tem a função de gerenciar os diversos sistemas de arquivos montados em um sistema Linux e também de fornecer uma camada de software para que os aplicativos trabalhem na mesma maneira em qualquer um destes sistemas de arquivos, ficando a cargo do VFS trabalhar com estes diferentes sistemas de arquivos[RUBINI].

A figura 3.2 mostra como o VFS trabalha com os diferentes sistemas de arquivos. Quando uma abertura de arquivo é solicitada pelo usuário é o VFS quem se encarrega de localizar em qual *inode* está armazenado este arquivo e realizar uma chamada para o sistema de arquivos no qual está contido este *inode*, que então é realizada a rotina de abertura do arquivo.

Assim como no Unix, o VFS do Linux trabalha com os conceitos de **superblock**, **inode**, **diretório** e **arquivos**. Segundo [RUBINI] os diretórios são muitas vezes tratados como arquivos comum, mas o kernel possui um estrutura própria para armazenar diretórios, os sistemas de arquivos possuem funções distintas para trabalhar com diretórios, então cabe ao VFS controlar estas estruturas.

O VFS trabalha com o sistema de arquivos por meio de um superblock, o qual é definido por [RUBINI], como sendo um bloco de metadados, que armazena informações sobre a partição e sobre o sistema de arquivos nela utilizado.

Os metadados são informações sobre o sistema de arquivo como os *inodes*, o mapa de

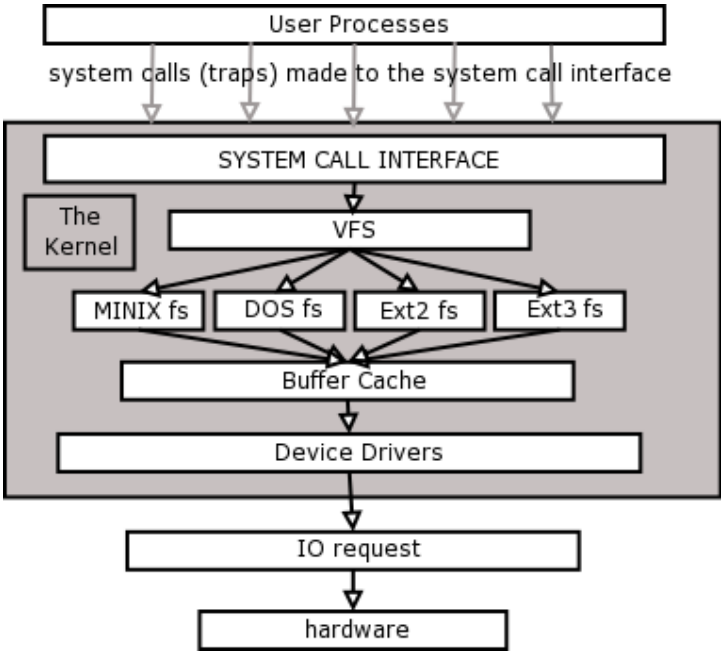

Figura 3.2: Estrutura do do Virtual File System

blocos livres, diretórios, lista de *inodes* livres, tamanho do disco, etc.

O VFS trabalha com os diferentes sistemas de arquivos por meio dos superblocks, que é criado para cada sistema de arquivos montado.

3.2.1 VFS inode

Para armazenar as informações dos arquivos, o VFS possui seu próprio *inode*, o *VFS inode*, que cada *VFS inode* é associado à um *inode* do sistema de arquivos real. O *VFS inode* além de armazenar todos os dados do *inode* do sistema de arquivo, ele também armazena outros atributos, os quais são descritos abaixo:

Dispositivo Identificação de qual dispositivo ou partição está o arquivo, o qual está controlando;

Número do Inode É a combinação entre o *inode* e o dispositivo no qual o *inode* está armazenado, este número é único no VFS;

Operações no Inode É um vetor de funções que desempenham operações para este *inode*, cada sistema de arquivo possui rotinas distintas;

Lock É utilizado para bloquear o *VFS inode*, quando por exemplo o sistema de arquivo esta lendo o *inode*;

3.2.2 Superblocks

Como foi abordado anteriormente o VFS trabalha com cada sistema de arquivo através de um superblock, que guarda todas informações necessárias para o VFS manipular tal sistema de arquivo. Cada sistema de arquivo montado, o VFS cria um superblock para este.

Os atributos guardados pelo superblock são descritos abaixo:

Dispositivo Armazena o dispositivo o qual o sistema de arquivo está contido, por exemplo: `/dev/sda1`;

Ponteiro do Inode Armazena alguns ponteiros para alguns *inodes* específicos, para controle interno, como: ponteiro para o primeiro *inode* deste sistema de arquivos, para o *inode* que representa o ponto de montagem do sistema de arquivos;

Blocksize O tamanho do bloco que o sistema de arquivo trabalha, a unidade é o byte.

Operações do Superblock Um ponteiro para um vetor de rotinas para o superblock executar sobre o sistema de arquivo;

Tipo de Sistema de Arquivos Contém um ponteiro para a estrutura do sistema de arquivo montado;

Sistema de Arquivo específico Algumas informações sobre o sistema de arquivo que o superblock representa.

3.2.3 VFS Inode Cache

Quando os arquivos são acessados o VSF necessita buscar os *VFS inodes*, por meio do par (partição, número do inode), caso alguns arquivos sejam muito utilizados, o VFS teria que realizar toda vez a combinação e obter o *VFS inode*.

Para melhorar isto, é utilizado uma cache de *VFS inode*, na qual os *VFS inodes* mais acessados são armazenados em uma cache. Segundo [RUBINI] esta cache é uma tabela hash, onde o índice da tabela que armazenará o *VFS inode* é determinado pela combinação (partição, número inode), o qual é único em um VFS.

Caso solicitado um arquivo, primeiramente é verificado na cache, caso o *VFS inode* estiver presente obtém-o, senão realiza o processo de busca no dispositivo de armazenamento.

Há também uma outra cache chamada de **cache de diretório**, que é utilizada para guardar os *inodes* dos diretório mais freqüentemente acessados, ou seja, armazena o mapeamento entre os diretórios específicos e seus números de *inodes*.

3.3 Ext3

O ext3 (Third Extended File System) foi desenvolvido por Stephen Tweedie como uma evolução do sistema de arquivos ext2. O ext3 possui total compatibilidade com o sistema de arquivos ext2, pois o ext3 é praticamente um ext2 com a propriedade de journal presente.

Segundo [NEMETH] no sistema de arquivos ext3 os dados do arquivo são armazenados em unidades chamadas blocos. Estes blocos podem ser numerados seqüencialmente. Um arquivo também tem um inode. Como os blocos, os inodes são numerados seqüencialmente, embora tenham uma seqüência diferente. Uma entrada de diretório consiste do nome do arquivo e um número de inode.

O sistema de arquivos ext3 consiste de cinco componentes estruturais:

1. Células de armazenamento inode;
2. Superblocos distribuídos;
3. Mapa de blocos no sistema de arquivos;
4. Resumo de emprego de blocos;
5. Conjunto de blocos de dados.

O Linux mantém para cada sistema de arquivos montado uma cópia do superbloco na memória RAM. A chamada de sistema "sync"despeja os dados sobre os superblocos que estão armazenados em memória cache para seus locais em disco, sincronizando as informações sobre o sistema de arquivos. Essa sincronização torna o sistema de arquivos consistente em um tempo extremamente pequeno. Esse salvamento ou sincronização ocorre em intervalos constantes de trinta segundos para sistemas ext2 e a cada 5 segundos para ext3, reduzindo, ainda mais, as falhas em servidores com intensas atividades de gravação de arquivos.

São descarregados tanto inodes modificados quanto blocos de dados armazenados em cache. O comando "update"executado no boot aciona o daemon bdflush, que executa uma sincronização nesse intervalo de tempo.

Um mapa de blocos do disco é uma tabela de blocos livres que o disco contém. No momento da gravação de arquivos novos esse mapa é verificado de modo que uma disposição eficiente seja utilizada. Os resumos de empregos de blocos registram informações básicas sobre os blocos que já se encontram em uso.

Corrigir um sistema de arquivos manualmente é uma tarefa complexa que exige do administrador conhecimentos sobre a estrutura e o funcionamento interno do sistema de

arquivos. Alterações introduzidas nessa tentativa de recuperação que sejam indevidas podem danificar permanentemente o acesso aos dados.

3.3.1 Journal

A introdução do Journal em sistemas ext3 modifica essa abordagem de recuperação de sistemas de arquivos e reduz o tempo de parada do sistema para valores muito baixos, introduzindo uma confiabilidade muito superior ao servidor. Na instalação do sistema de arquivos, uma área é reservada para a alocação do journal ou log.

Segundo [NEMETH], as operações efetuadas nos arquivos são registradas nessa área de log do mesmo modo que o controle de transações em bancos de dados. As operações são primeiramente gravadas no journal. Quando a atualização do registro de ações (log) é completada, um registro de complemento (commit record) é gravado sinalizando o final da entrada. Então, as mudanças são efetivamente gravadas em disco no sistema de arquivos normal.

Uma falha nesse ponto permite, através da consulta ao journal, a reconstrução das operações ainda não concluídas e a rápida recuperação do sistema. Após a gravação em disco no sistema de arquivos, um marcador confirma a operação e descarta o log, já que a operação foi corretamente confirmada.

O ext3 não precisa lidar com a complexidade dos Journalings que trabalham gravando bytes. Ele suporta três diferentes modos de trabalho do Journaling, de acordo com [NEMETH]:

- **Journaling (Registro de ações)**: grava todas as mudanças em sistema de arquivos e usa um arquivo de registros de ações maior. Isso pode retardar a recuperação durante a reinicialização. É o mais lento dos três modos, sendo o que possui maior capacidade de evitar perdas de dados. Uma forma de aperfeiçoar a velocidade dessa opção é gravar os registros em bancos de dados externos.

- **Ordered (Ordenado)**: grava somente mudanças em arquivos metadados (arquivos que possuem informações sobre outros arquivos), mas registra as atualizações no arquivo de dados antes de fazer as mudanças associadas ao sistema de arquivos. Este Journaling é o padrão nos sistemas de arquivos ext3 sendo a melhor opção para a maioria dos sistemas.

- **Writeback**: também só grava mudanças para o sistema de arquivo em metadados, mas utiliza o processo de escrita do sistema de arquivos em uso para gravação. É o mais rápido Journaling ext3, porém é o menos confiável e mais suscetível à corrupção de arquivos após uma queda do sistema. Esse modo é equivalente à instalação de um sistema com ext2 nativo.

3.4 ReiserFS

O ReiserFS foi criado por Hans Reiser, atualmente **namesys** é a equipe que está reponsável pelo desenvolvimento e manutenção do ReiserFS, o objetivo inicial do ReiserFS era fornecer um sistema de arquivo seguro, rápido, robusto e de fácil recuperação de arquivos.

O ReiserFS utiliza como base os componentes apresentados nas seções anteriores, sua eficiência é dada pelas novas abordagens sobre estes componentes, as quais serão apresentadas nesta seção.

Algumas das principais características do reiserFS são descritas por [ALECRIM], as quais são apresentadas abaixo:

1. Utilização de árvores balanceadas;
2. *Journalling*;
3. Alocação dinâmica de *inodes*;
4. Suporte a arquivos com mais de 2GB;

3.4.1 Árvores balanceadas

Segundo [NAMESYS] o ReiserFS utiliza como estrutura de armazenamento, as árvores balanceadas(Balanced Tree - B+), que melhoram em muito o tempo de leitura e escrita em arquivos. As árvores são utilizadas para armazenar os inodes dos arquivos, pelo motivo da árvore ser uma estrutura de dados rápida, o tempo de acesso aos dados é também muito rápido.

A figura 3.3 apresenta de um exemplo de uma árvore balanceada, nesta árvore os dados são armazenados apenas em suas folhas.

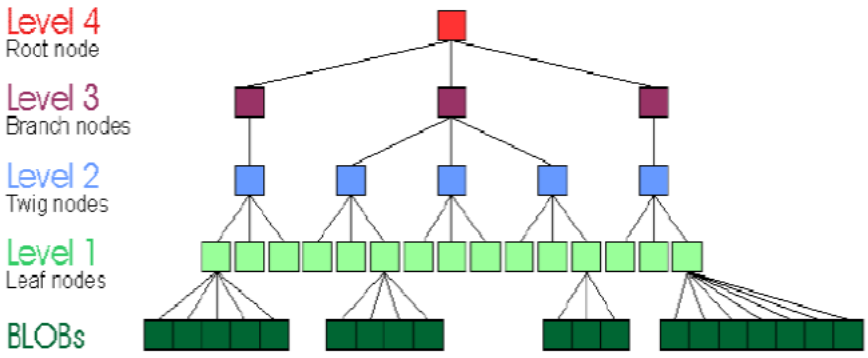

Figura 3.3: Exemplo de uma árvore balanceada utilizada no ReiserFS

Apesar de utilizar uma complexa estrutura de dados, as árvores balanceadas, os ReiserFS tem um excelente tempo de execução de processos de leitura e escrita [NAMESYS].

3.4.2 *Journalling*

O *journalling* é baseado na abordagem de inconsistência de dados, na teoria de banco de dados, no qual antes de realizar o processo de gravação dos dados é gerado um **log**, que conterá as instruções a serem realizadas, para efetuar o processo de gravação dos dados, após gravado no log o sistema executa a instrução por meio do log, caso ocorra alguma falha do sistema, é realizado novamente as instruções pois o log está salvo.

Segundo [FILGO] no ReiserFS, o *journalling* guarda apenas os metadados, o que preserva a integridade do sistema de arquivo não a dos dados. Isso garante que não ocorrerá problemas com as partições. Isso é um dos principais fatores do ganho de desempenho do ReiserFS, caso haja necessidade de checagem do sistema de arquivos, por meio do *reiserfsck*, apenas os metadados são checados, da mesma maneira na gravação apenas dos metadados no log, diferentemente de outros sistema de arquivos que gravam os metadados e os dados no log.

O tamanho do *journalling* do ReiserFS é de 33MB [FILGO], sendo assim inviável para dispositivos com menos de 50 MB.

3.4.3 Alocação Dinâmica de Inodes

Diferentemente da maioria dos outros sistemas de arquivos, o reiserFS tem seu tamanho de bloco variável, contrário de outros sistemas de arquivos, no qual o tamanho do bloco é fixo. Em arquivos pequenos, mesmo sendo de tamanho variável, os dados são armazenados em pequenos blocos fixos, os quais ficam próximos aos metadados, no inode, permanecendo assim próximos uns dos outros, necessitando de uma pequena movimentação da cabeça de leitura. Por isso o seu excelente desempenho em manipular arquivos pequenos.

ReiserFS possui um recurso muito relevante, para melhor alocação de dados no dispositivo de armazenamento o *tail packing*, que consiste em armazenar na árvore B+, os bytes finais de arquivos que não conseguiram ocupar por completo os blocos finais, conseguindo assim um ganho no processo de alocação de espaço em disco. Uma desvantagem deste recurso é a fragmentação do arquivo e a perda de performance. Este recurso pode ser desabilitado do sistema de arquivo, utilizando a opção *notail* no `/etc/fstab` [FILGO].

Capítulo 4

Entrada e Saída

Um dos principais encargos de um sistema operacional é fazer o controle dos dispositivos de E/S (entrada/saída) [TANENBAUM]. Ou seja, enviar comandos aos dispositivos, analisar interrupções, tratar erros possíveis e também prover uma interface que permita fazer chamadas ao sistema (*system calls*) para leitura e gravação nos dispositivos.

O sistema de E/S do Linux fornece uma visão semelhante a de qualquer sistema UNIX, onde os drivers de dispositivos (*device drivers*) são apresentados para o usuário como arquivos normais, a fim de abstrair o resto do sistema ou o usuário das particularidades do hardware. Assim, o acesso a um dispositivo ocorre da mesma maneira da abertura de um arquivo. Porém, estes arquivos que fornecem acesso a um dispositivo são denominados de arquivos especiais e estão associados a um diretório que se encontra dentro de `/dev` (lembrando que no Linux, diretórios são tratados como arquivos).

Estes arquivos especiais são arquivos com referências a um driver de dispositivo. Assim, permissões de leitura e escrita nos dispositivos são concedidas aos usuários do sistema operacional da mesma forma com que é feita em arquivos. Porém, em um driver de dispositivo de um alto-falante é possível apenas escrever, assim como no de um mouse só se pode fazer leitura.

Os drivers de dispositivos são conectados ao núcleo (*kernel*) através de módulos carregáveis, que podem ser ligados ao núcleo, quando este estiver em atividade. Logo, ao se acrescentar ou remover dispositivos, o correspondente driver deve ser ativado ou desativado, respectivamente, junto ao kernel. Isto evita uma ação manual no núcleo para inclusão ou remoção de drivers.

No Linux, não há dois ou mais drivers para cada dispositivo e sim um para cada. Mas pode-se utilizar o mesmo driver para dispositivos diferentes, como no caso de pendrives que utilizam o mesmo driver feito para tratar discos SCSI. Assim, há para cada driver um número de dispositivo principal que o identifica e também um número de dispositivo secundário, que identifica o dispositivo. No nosso exemplo, para distinguir o pendrive do

SCSI.

Nos sistemas UNIX, os arquivos especiais são separados em duas classes: blocos e caracteres. No Linux existe mais uma: rede (que será abordada mais adiante). Cada classe é representada por uma tabela.

Através do número principal, distingui-se `bdevsw`, tabela dos arquivos especiais de blocos, da `cdevsw`, tabela dos arquivos especiais de caracteres. A cada linha de uma dessas tabelas se associa um dispositivo e a cada coluna uma função. As funções mais comuns são: `open`, `close`, `read`, `write`, `ioctl`. Cada elemento dessa tabela tem um ponteiro para a função correspondente a determinado dispositivo. Ou então será nulo, caso não esta função não exista, como no caso da memória, que não apresenta `open` nem `close`, apenas `read` e `write` [TANENBAUM].

4.1 Arquivos especiais de blocos

Os blocos são unidades enumeradas com tamanho fixo em uma seqüência. Assim, cada bloco tem seu endereço próprio e possui acesso individual, ou seja, apresenta leitura e escrita independente dos outros blocos.

Os discos são o principal exemplo de dispositivos de blocos. Estes dispositivos normalmente são usados para sistemas de arquivos, mas também podem ser acessados diretamente, como no caso de um banco de dados que possui organização própria para seus dados.

Os arquivos especiais de blocos normalmente se associam a dispositivos com taxa de transferência alta. Para o sistema operacional é de fundamental importância o desempenho dos discos. Então, os dispositivos de blocos devem ser preparados para fornecer funcionalidades que mantenham uma qualidade do acesso a este meio, isto é, que seja feito da melhor forma possível[SILBERSCHATZ].

4.1.1 Cache de buffers de bloco

Utilizado para minimizar a quantidade de transferências entre o dispositivo e a memória. É uma tabela situada no kernel que abriga os blocos mais recentemente utilizados. Então, quando um é necessário acessar um bloco, deve verificar antes se ele está nessa cache, evitando assim, o acesso ao disco.

No Linux, o cache de buffers de bloco engloba duas características principais, ser um pool de buffers para E/S ativas e um cache para E/S concluídas. Pois o cache manipula tanto escritas quanto leituras. A figura 4.1 apresenta a presença da cache de buffer no mecanismo de busca de dados no disco.

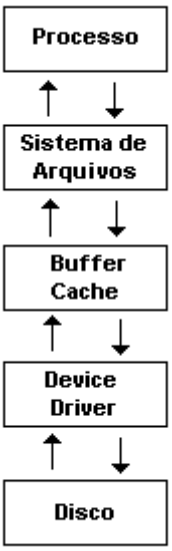

Figura 4.1: Utilização de uma cache de buffer

Segundo [SILBERSCHATZ] um buffer é uma parte de uma página alocada do pool de memória principal do kernel.

Além dos buffers, o cache de buffers é formado também pelos descritores destes buffers, chamados de `buffer_heads`, na proporção de um para um.

Nos `buffer_heads` se encontram todas as informações dos buffers que o kernel necessita para controlá-los. Como qual dispositivo de bloco o buffer está servindo, a parte (o deslocamento mais o tamanho) dos dados que este buffer refere no dispositivo, que juntos formam o identificador do buffer.

Para o controle, o kernel utiliza quatro listas de buffers: intacta ou limpa, modificada ou suja, bloqueada e livre. Nessas listas os últimos blocos acessados são colocados no início da lista. Assim, quando é necessário retirar blocos, remove os menos recentemente usados, ou seja, os que estão no final da lista.

A lista modificada é descarregada à força no disco quando está cheia. Mas para evitar uma espera grande, os blocos desta lista são escritos a cada 30 segundos no disco.

Já a lista livre, pode-se ter blocos inseridos por eventos de sistema de arquivos, como no caso de um arquivo ser deletado. Mas dela é removida pelo kernel, quando este tem necessidade de mais buffers. O kernel também pode ampliar a lista caso haja memória livre disponível[TANENBAUM].

4.1.2 Gerente de pedidos

Responsável pelo controle de leituras e escritas dos buffers nos drivers de dispositivos [TANENBAUM]. Ele utiliza o `buffer_head` mais a determinada ação (leitura ou escrita)

e encaminha um pedido, mas não fica aguardando por sua conclusão. Assim, ele guarda o ponteiro da lista de `buffer_head` emitida. O pedido é removido da lista apenas quando concluído o evento de E/S.

Ele reserva uma lista de pedidos para cada driver de dispositivo de bloco. Mas, ao surgir novos pedidos, ele procura agrupar pedidos antes de encaminhar e bloqueia o grupo. Assim, sua política é mandar um pedido maior ao invés de vários menores, que aumenta a eficiência. Porém, quando os `buffer_heads` vão sendo atendidos, eles já vão sendo desbloqueados individualmente, a fim de não causar espera até terminar de atender todo o grupo.

4.2 Arquivos especiais de caracteres

Empregados em dispositivos que não apresentam acesso aleatório para blocos de dados. Geralmente utilizados para dispositivos que realizam E/S por meio de fluxo de caracteres com baixa taxa de transferência, tais como, impressoras, mouses, teclados.

Atualmente, dispositivos de rede não são mais considerados exemplos de dispositivos de caracteres. Pois, além de não serem mais dispositivos demorados, assim como as primeiras redes, para se acessar os dados nela é necessário abrir uma conexão junto ao kernel[TANENBAUM].

Os dispositivos de caracteres devem prover ao kernel funções para suas operações de E/S, pois o kernel não faz nenhum pré-processamento, apenas encaminha pedidos de leitura ou escrita para estes dispositivos, que ficam responsáveis pelo tratamento dos pedidos. Porém, há exceção: os dispositivos de terminais.

O kernel mantém um buffer para o fluxo de dados e oferece uma interface padrão em um interpretador denominado de tratador de linhas (*line discipline*). Este tratador substitui tabulação por espaços, retorno de carro por avanços de linha, operando como uma espécie de filtro. A `tty` é o tratador mais comum e faz a comunicação do fluxo básico de E/S entre processos e terminais.

Capítulo 5

Transmissão em Redes

O processo de transmissão de dados em redes, e similar à um exemplo de E/S, este conceito de transmissão foi introduzida pelo UNIX de Berkeley. O principal conceito é o *socket*. Os *sockets* são análogos aos soquetes telefônicos e aos soquetes telefônicos fixados nas paredes, e as caixas postais, uma vez que permitem a comunicação dos usuários com a rede, como o soquete telefônico permite a conexão do telefone com o sistema telefônico, e a caixa postal faz a conexão com o sistema telefônico. Depois da criação do socket, é retornado um descritor de arquivos, que é necessário para a leitura e escrita de dados, estabelecer e liberar a conexão [SILBERSCHATZ].

Existem vários tipos de sockets, cada um suporta um tipo diferente de transmissão, este tipo de transmissão é especificado na criação do socket, os tipos são:

- Fluxo confiável de bytes orientado a conexão;
- Fluxo confiável de pacotes orientado a conexão;
- Transmissão não confiável de pacotes.

5.1 Fluxo confiável de bytes orientado a conexão

Permite que dois processos em máquinas diferentes estabeleçam entre si um pipe (permite um fluxo de bytes unidirecional confiável entre dois processos, de tamanho pequeno raramente são escritos em discos, ficam mantidos na memória pelo cache de buffers do bloco normal) entre si. Os bytes são emitidos numa extremidade e recebidos na outra, na mesma ordem. Com isso o sistema garante que todos os bytes enviados cheguem do outro lado, e na mesma ordem que foram enviados.

5.2 Fluxo confiável de pacotes orientado a conexão

Muito parecido com o primeiro tipo, mas preserva a fronteira de pacotes, se é chamada a operação `write` de dois pacotes de 128 bytes, com o socket do tipo 1 é enviado os 256 bytes de uma vez só, enquanto neste tipo os dados são enviados 128 bytes em cada chamada, mas será necessário duas chamadas para satisfazer a transmissão total dos dados[TANENBAUM].

5.3 Transmissão não confiável de pacotes

Este tipo o socket dá ao usuário acesso a recursos de rede de baixo nível. É utilizado para aplicações de tempo real e quando o usuário pretende implementar um esquema para tratamento de erros. Os pacotes podem ser perdidos e reorganizados pela rede, não é garantida a segurança como nos dois primeiros tipos. Neste tipo o desempenho prevalece sobre a confiabilidade.

Depois de ser criado o socket, um parâmetro é utilizado para especificar qual protocolo pode ser usado. O TCP(*Transmition Control Protocol*, Protocolo de controle de transmissão) é o protocolo mais usado para Fluxo confiável de rede. O UDP (*User datagram protocol*, protocolo de datagrama do usuário) é protocolo mais usado para a transmissão não confiável de pacotes, não existe nenhum protocolo comum para Fluxo não confiável de pacotes. Ambos os protocolos executam no topo, o IP (*Internet Protocol*, Protocolo da Internet). Todos estes protocolos que formam a base da Internet foram criados na ARPANET do Departamento de Defesa dos EUA [SILBERSCHATZ].

Um socket precisa ter endereço ligado a ele, antes de ser usado para transmitir dados, o endereço pode estar em apenas um dos muitos domínios de nomes. O mais comum é o domínio da Internet (AF_INET), que na versão 4 utiliza endereços de 32 bits para nomear os pontos da rede, e 128 na versão 6 (a versão 5 foi experimental não deu certo), o protocolo utilizado na Internet é TCP/IP. Outro domínio é o domínio XEROX Network Services - NS (AF_NS), e o domínio UNIX (AF_UNIX) o formato de endereço são nomes de caminhos normais do sistema de arquivos, como `alpha/beta/gamma`.

Depois de já criado nos computadores de destino e origem, a conexão está pronta para ser estabelecida entre eles, para uma comunicação orientada a conexão. Um lado faz a chamada `listen` no socket local, que cria um buffer e fica bloqueada até a chamada chegar ao outro lado. O outro faz a chamada `connect`, e passa como parâmetro o endereço do socket remoto e o descritor de arquivos do socket local. Se o lado remoto aceitar a chamada, a conexão do sistema entre os sockets estará estabelecida. A figura 5.1 apresenta a estrutura do processo de comunicação via sockets.

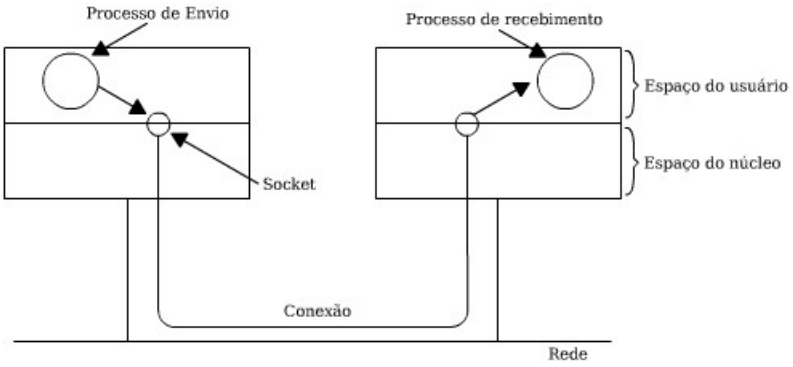

Figura 5.1: Exemplo de comunicação via sockets

Depois de estabelecida a conexão um processo pode ler e escrever nele usando o descritor de arquivo de seu socket local. Quando se deseja fechar a conexão basta usar a chamada ao sistema, `close`.

5.4 Suporte de rede

Praticamente todos os sistemas Linux, assim como o Unix possuem suporte aos recursos de rede UUCP usado em linhas telefônicas discadas para suportar rede de correio e rede de notícias ESNET. Esses recursos são de pequena utilização, pois não permitem login remoto, e muito menos chamada de procedimento remoto.

O UNIX 4.3BSD, suporta protocolos DARPA Internet, UDP, TCP, IP e ICMP em variedade de interfaces Token ring, Ethernet e ARPANET. O framework no kernel facilita implementação de outros protocolos e são acessíveis através da interface do soquete.

A rede implementada no 4.3BSD usa o soquete orientado, parecidamente com o modelo ARPANET (ARM). ARPANET versão anterior a isso, que é muito parecidas com a versão anterior ARM. O modelo ARM possui as seguintes camadas:

Processos/Aplicações corresponde às camadas de aplicação, apresentação e sessão do modelo ISO. Usa os protocolos de nível usuário como o FTP (*File Transfer Protocol*) e o Telnet (login remoto).

HOST-HOST corresponde a parte superior da camada de rede do modelo ISO e usa o protocolo TCP/IP.

Interface de Rede equivalente a parte inferior da camada de rede da ISO e a camada de enlace de dados. Os protocolos usados aqui dependem de do tipo de rede física os ARPANET usa protocolos IMP-HOST e Ethernet os da Ethernet.

Hardware de rede como o ARM se preocupa principalmente com software, não existe uma camada explicita para hardware de rede, mas qualquer rede possuirá hardware que corresponda á camada física do modelo ISO.

A rede no 4.3BSD é mais generalizada do que no modelo ISO ou ARM, possuindo menos camadas. Os processos se comunicam com os protocolos de rede, portanto com outros processos em outras máquinas via soquete, que corresponde à camada de Sessão da ISO, pois é responsável por configurar e controlar as comunicações.

5.5 Modelo de referência e camada

Os soquetes são suportados por um ou vários tipos de protocolos, dispostos em camadas, e ele pode fornecer diferentes serviços (entrega confiável, controle de fluxo, etc.) dependendo do tipo de soquete sendo suportado e dos serviços exigidos por protocolos mais altos.

Um protocolo comunica-se com outro ou pode comunicar-se com a interface de rede apropriada para o hardware de rede em questão. Há poucas restrições em relação a que protocolos podem se comunicar entre si. Cada interface de rede possui a tendência de um driver de hardware de rede. O driver é responsável pelas características específicas da rede local sendo acessada, para que os protocolos que a utilizem não se preocupem com essas características. As funções da interface de rede dependem do hardware de rede utilizado.

A estrutura de rede e os soquetes utilizam o mesmo conjunto de buffers de memória, ou *mbufs*. Um *mbuf* possui tamanho intermediário, 128 bytes de comprimento, 112 para dados, o resto para ponteiros para encadear *mbfus* em filas. Os dados são passados entre camadas (protocolo-protocolo, soquete-protocolo, protocolo-rede) via mbufs [SILBERSCHATZ].

Capítulo 6

Conclusão

Face a esta visão sobre o sistema operacional Linux, podemos concluir que o linux trata-se de um poderoso sistema operacional, fortemente baseado UNIX, podemos perceber a sua grande influência sobre vários processos realizados pelo Linux, como os sistemas de arquivos, o processo de gerência de memória dentre outros.

Por motivo de sua grande flexibilidade e confiabilidade o Linux é largamente utilizado em servidores, sua utilização em desktop apesar de ainda pequena, vem crescendo violentamente nos últimos anos.

Referências Bibliográficas

[SILBERSCHATZ] SILBERSCHATZ, A., Galvin, P., Gagne, G. *Sistemas Operacionais: Conceitos e Aplicações* ;tradução de *Adriana Richie*. 1ª Edição. Elsevier. Rio de Janeiro. 2000.

[RUBINI] RUBINI, Alessandro. *The "Virtual File System"in Linux. Disponível em*: http://www.linux.it/ rubini/docs/vfs/vfs.html. Acessado em: 20/06/2007.

[NAMESYS] NAMESYS-project, Reiser, H., Saveljev, V. *Three reasons why ReiserFS is great for you*. Disponível em: http://www.namesys.com/X0reiserfs.html. Acessado em: 20/06/2007.

[ALECRIM] ALECRIM, Emerson. *Introdução ao sistema de arquivos ReiserFS*. Disponível em: http://www.infowester.com/reiserfs.php. Acessado em: 20/06/2007.

[FILGO] FILGO, João E. M. *Descobrindo o Linux*. Novatec. 2006.

[TANENBAUM] TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 2 Edição. Prentice Hall. 2003.

[NEMETH] NEMETH, Evi, *et al.*, *Manual Completo do Linux: Guia do Administrador*. 1 Ed. Makron Books. 2004.