Princípios de Entrada e Saída

Neste capítulo serão apresentados conceitos básicos dos dispositivos de entrada e saída, .os mecanismos de comunicação entre a CPU e os periféricos, a organização dos subsistemas de entrada e saída e os princípios de entrada e saída no sistema operacional Linux.

2.1 Dispositivos de entrada e saída

Um sistema de computação é formado por Unidade Central de Processamento (UCP), memória e periféricos. Periféricos são dispositivos que permitem aos usuários a comunicação com o sistema. Os periféricos podem ser de entrada, de saída ou de entrada e saída. Exemplos de periféricos de entrada são mouse, teclado, scanner, leitora ótica, etc. Periféricos de saída de dados são impressora, monitor de vídeo, etc. e periféricos de entrada e saída são discos magnéticos, fitas magnéticas, pendriver, CD´s disquetes, etc.

Os periféricos podem ser orientados a bloco e orientados a caractere. Os periféricos orientados a blocos armazenam as informações em blocos de tamanho fixo e cada bloco possui o seu próprio endereço. Os blocos podem ser de 512, 1024, 2048, ... bytes. A principal característica destes dispositivos é a capacidade de se endereçar blocos aleatoriamente, o que permite que se acesse um dado pertencente a um arquivo sem ter que acessar todos os anteriores. Fazem parte deste tipo de periférico os discos magnéticos.

Os periféricos orientados a caractere executam operações de transferência de cadeia de caracteres. Não possuem estrutura de blocos,

portanto não possuem a capacidade de endereçamento aleatório. Exemplos desses periféricos são os teclados e as impressoras.

2.2 Comunicação entre a CPU e os Periféricos

Em um sistema de computação a Unidade Central de Processamento (UCP) acessa a memória para fazer busca de instruções e dados, executa as instruções e interage com os periféricos nas operações de entrada e saída. Em relação a UCP, os periféricos são extremamente lentos, e operam de forma assíncrona. A comunicação da UCP com o periférico é feita através de registradores. Diferentes periféricos possuem número distinto de registradores. Três registradores são importantes:

- Registrador de dados: Neste registrador o processador coloca os dados a serem gravados no periférico e o periférico coloca os dados resultantes de uma operação de leitura.
- Registrador de status: Contém informações sobre o estado do periférico (livre/ocupado).
- Registrador de controle: Neste registrador o processador escreve os comandos a serem executados pelo periférico.

Os dispositivos de entrada e saída podem ser mapeados em um espaço de endereçamento separado, distinto do espaço de endereçamento de memória e a comunicação e a troca de informações entre o processador e o controlador do periférico é feita através deste espaço, ou o espaço de endereçamento pode ser mapeado diretamente na memória, ocupando endereços no espaço de memória.

Com o espaço de endereçamento distinto do espaço de endereçamento de memória são necessárias instruções específicas de leitura e escrita de dados (Ex. IN e OUT). IN é usada para leitura de dados dos registradores usados na comunicação com o controlador do periférico. OUT é usada para escrita de dados nestes registradores.

Com o I/O mapeado na memória, para a comunicação entre o processador e o controlador do periférico, podem ser usadas todas as instruções que fazem movimentação de dados na memória.

Tipos de Entrada e Saída

I/O Programado (pooling).

As operações de entrada e saída são realizadas sob o controle do Sistema Operacional, que fica testando, na interface com o periférico, o estado do I/O (pooling). O controlador do periférico não interrompe a CPU. Ex. de funcionamento:

1. SO envia ao periférico um comando
2. Ler *status* do periférico
3. Se *Status NOT* ready goto 2
4. Se comando executado é de leitura então
   Ler informação do periférico
   Escrever a informação lida na memória
6. Se existe novo pedido de I/O então go to 1

O grande inconveniente do I/O programado é que a maior parte do tempo do processador é desperdiçado testando o status do periférico (pooling: linhas 2 e 3).

I/O com interrupção

O periférico, ao término de uma operação de transferência de dados (entrada ou saída), sinaliza o processador gerando um pedido de interrupção. O processador passa então a executar o procedimento que implementa a função correspondente a interrupção ocorrida. Existe um tratamento específico à cada fonte de interrupção. Como a ação do periférico é independente da ação do processador, durante a operação de transferência de dados o processador fica livre para executar outros processos.

Acesso Direto à Memória (DMA)

A informação é transferida diretamente do periférico para a memória sem intervenção do processador. Ao término da operação de transferência de dados é gerada uma interrupção, que será tratada pelo Sistema Operacional. Com DMA o processador, durante as operações de I/O, o processador fica executando outros processos.

Origem das Interrupções

As interrupções podem ser:

a) Geradas pelo programa: implementadas com o uso de instruções especiais (ex. INT, TRAP).
b) Geradas por um erro: divisão por zero, referência a memória fora do espaço permitido, etc.
c) Geradas pelo relógio;
d) Geradas pelos periféricos: sinalização de final de operação de E/S ou condição de erro.

Tratamento de Interrupções

O sistema operacional reage à ocorrência de interrupções, que provocam a execução da rotina de tratamento correspondente à fonte de interrupção.

Tratamento de interrupções de software

Estas interrupções são geradas por uma instrução especial (int, trap, ...). Quando ocorre uma interrupção de software:

- O programa para de executar;
- O sistema operacional executa o serviço solicitado pelo processo do usuário ou o encaminha ao periférico (no caso de E/S);
- Ao final da execução do serviço, o controle retorna ao processo solicitante ou um novo processo é selecionado (no caso do processo solicitante necessitar esperar pela realização do serviço).

Tratamento de interrupções de hardware
Quando ocorre uma interrupção de hardware, as ações executadas pelo processador são as seguintes:

- O processador acaba execução da instrução atual;
- O processador testa existência de interrupção;
- O processador salva estado atual;
- O processador carrega contador de programa com o endereço da rotina de tratamento da interrupção;
- A execução começa na rotina de tratamento da interrupção;
- A rotina de tratamento da interrupção executa;
- O contexto de execução anterior é restaurado;
- A execução retorna para a rotina interrompida.

2.3 Organização do subsistema de Entrada e Saída

O objetivo de um subsistema de entrada e saída é permitir a um programa de aplicação acessar os periféricos para realização de operações de transferência de dados. Para tal, deve possuir uma camada de mais alto nível lógico para servir de interface com o programa de aplicação e uma camada para interagir diretamente com o hardware dos periféricos, formada pelos drivers de periféricos. Também são relacionados diretamente também com as operações de entrada e saída os tratadores de interrupção. Possuem, portanto, três níveis lógicos: tratadores de interrupção, drivers dos periféricos e Interface com os programas de aplicação,.

Tratadores de Interrupção

Quando uma operação de entrada e saída é solicitada por um processo, o sistema operacional recebe o pedido e aciona o driver que o encaminhará ao periférico específico. O processo solicitante fica bloqueado, sem disputar o processador, até que o pedido se complete. Os tratadores de interrupção são

componentes do núcleo do sistema operacional, acionados pelas interrupções. Quando a interrupção ocorre, a rotina de tratamento executará as ações correspondentes à interrupção. O resultado de uma interrupção é que o processo bloqueado na operação será desbloqueado pelo tratador da interrupção.

Além dos periféricos, outra fonte de interrupção é timer. Quando uma interrupção do timer ocorre, em um sistema de tempo compartilhado, muitas ações relacionadas ao tempo devem ser executadas.

No sistema operacional Linux, a interrupção do timer está associada ao IRQ0 e ocorre a cada 10 ms. A entrada IRQ0 contém o endereço da rotina *timer_interrupt(),* que executa as seguintes ações:

- Calcula o tempo de retardo entre a ocorrência da interrupção e o disparo da rotina de tratamento, que será utilizado para corrigir o tempo atribuído ao processo do usuário;
- Chama a rotina *do_timer_interrupt ().*

A rotina *do_timer_interrupt ()* roda com interrupções desabilitadas. Os passos que executa são:

- Atualiza a variável jiffies, que armazena o número de ticks ocorridos desde que o sistema foi disparado (um jiffie = 1 tick e um tick ocorre a cada 10 ms);
- Atualiza o tempo do processo do usuário:
    - Atualiza o campo do descritor do processo corrente que mantém o tempo de execução em modo usuário;
    - Atualiza o campo do descritor do processo corrente que mantém o tempo de execução em modo kernel;
    - Conta o número de processos prontos para rodar e atualiza estatísticas de uso da CPU;
    - Verifica se o tempo do processo corrente se esgotou.
- Dispara a rotina *timer_bh ()* .

A rotina *timer_bh ()* dispara os procedimentos responsáveis pela execução das seguintes funções:

- Atualizar tempo corrente do sistema e a data do computador e calcula a carga corrente do sistema:
- Tratamento dos relógios lógicos: a implementação de relógios lógicos é feita com com cada relógio possuindo um campo que indica quando o tempo expira. Este campo é obtido somando o número de ticks ao valor corrente da variável jiffies. Cada vez que jiffies é incrementado, é comparado com este campo. Sendo maior ou igual, o tempo expirou e a ação correspondente deve ser executada. Relógios lógicos são usados:
    - Por programadores para disparar uma função em um tempo futuro ou para esperar por passagem de tempo;
    - Por drivers para detectar situações de erros (ex. periféricos que não respondem em um certo período de tempo)
    - Etc.

Driver dos dispositivos

Drivers de dispositivos são componentes do sistema operacional que interagem com um periférico, ou com uma classe de periféricos. Para cada tipo de periférico deve existir um driver específico, com pequenas variações para periféricos diferentes dentro da mesma classe. A função de um driver é receber um pedido do sistema operacional, que corresponde a uma solicitação de um usuário (ex. ler dados de um arquivo em disco), preparar a requisição, e encaminhar para a controladora do periférico. Se o periférico está ocioso, a operação deverá ser realizada imediatamente. Caso contrário, o pedido será colocado na fila de requisições do periférico.

No caso de uma operação de leitura de dados de um disco, o sistema operacional recebe do usuário os parâmetros (identificador do arquivo, tipo da

operação (leitura ou escrita) , endereço de memória onde devem ser armazenados os dados, quantidade de bytes a serem lidos), calcula o número do bloco e encaminha para o driver o número do bloco a ser lido, juntamente com os demais parâmetros. O driver constrói a requisição (estrutura de dados na qual coloca os parâmetros recebidos e na qual acrescenta o identificador do processo solicitante, o posicionamento do braço no cilindro próprio e a operação requerida (leitura ou escrita) e coloca na fila de requisições do disco, se o periférico estiver ocupado.

A interrupção da controladora sinaliza final de operação de entrada e saída. A rotina de tratamento da interrupção salva o contexto e acorda o driver relacionado à fonte específica de interrupção. Tratando-se de uma interrupção do disco, sendo final de operação de leitura, o driver identifica o processo para o qual a transferência de dados se completou e o incorpora na fila de processos aptos a executar. A seguir, pega a primeira requisição da fila de requisições e a entrega para a controladora, escrevendo os parâmetros nos registradores da controladora. Após essas ações, a execução retorna ao processo que estava sendo executado quando ocorreu a interrupção.

Interface com os programas de aplicação

Os componentes do subsistema de entrada e saída são quase que completamente implementados no núcleo do sistema operacional. O software que não faz parte do sistema operacional está implementado a nível usuário, pelas bibliotecas de funções. Assim, quando um usuário necessita ler de um arquivo no Linux, utiliza a função

```
n = read (f, &b, nbytes);
```

que lê do arquivo ***f*** e armazena no **&b**, quantidade de **nbytes.**

Esta função faz parte de uma biblioteca de funções e é incorporada ao código do programa do usuário. Assim, a comunicação dos programas de aplicação com o subsistema de entrada e saída é implementada por um conjunto de procedimentos de biblioteca, que possui a interface de cada primitiva. Fazem

parte desta biblioteca, além da função read, as funções write, open, close, create, etc.

2.4 Entrada e saída no Linux

A nível usuário, a comunicação do programa de aplicação e o subsistema de entrada e saída é feita pelas chamadas de sistema relacionadas às operações de entrada e saída. Cada chamada de sistema possui a identificação da operação e os parâmetros necessários à sua execução. Exemplos de chamadas de sistema são as primitivas open, close, read, etc. As chamadas de sistema referentes as operações de entrada e saída são quase que exclusivamente tratadas pelo VFS (Virtual File System).

O Linux implementa o conceito de independência do dispositivo que, permite que, por exemplo, o comando

write(f, &b, nbytes);

que escreve no arquivo ***f*** , ***nbytes,*** armazenados no endereço ***&b***, seja executado para o arquivo ***f*** armazenado em um disco ou em um disquete. Todos os procedimentos referentes a denominação do arquivo, proteção de acesso, alocação e liberação de buffers, tratamento de erros são realizados pelo software independente de dispositivo. O driver é responsável pelas operações de mais baixo nível, que dependem do tipo de periférico específico.

O sistema operacional Linux suporta um grande número de periféricos. Para cada periférico deve existir um driver específico. Para o LInux existem dois tipos de dispositivos: orientados a caractere e orientados a bloco. Cada dispositivo é identificado por um maior número e um menor número. O maior número identifica o tipo de periférico e o menor número é usado para acessar o periférico específico, no caso de um disco, um inteiro entre 0 e 255.

Para um driver ser utilizado, necessita ser registrado e inicializado. Registrar um driver significa fazer a ligação com os arquivos necessários a sua

execução. Na inicialização, são alocados os recursos necessários a execução do driver. Um driver pode ser compilado no kernel, neste caso é registrado quando o kernel executa as rotinas de inicialização, ou pode ser compilado como um módulo do kernel. Neste caso, sua inicialização é realizada quando o módulo é carregado.

O Linux identifica o final de uma operação de entrada e saída através de uma interrupção ou fazendo pooling. O funcionamento geral de um driver de impressora, que opera com pooling é apresentado a seguir.

```
void  imp (char c, int lp) {
   while (!READY_STATUS(status) && count > 0) {
        count - - ;
        }
    if (count == 0) return (TIMEOUT) ;
    out (char, lp) ;
   }
```

A variável count é utilizada para implementar um tempo máximo em que o driver testa o estado da impressora para enviar um novo caractere.