GSI018 – Sistemas Operacionais

Software de E/S

Universidade Federal de Uberlândia
Faculdade de Computação
Prof. Dr. rer. nat. Daniel D. Abdala

Na Aula Anterior...

Nesta Aula

- Camadas de Software de E/S;
- Princípios do Software de E/S;

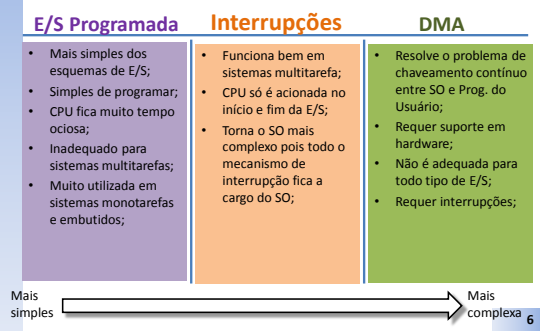
E/S Programada

- Três maneiras fundamentais de executar E/S/:
  1. E/S Programada;
  2. E/S orientada a Interrupções;
  3. E/S via DMA;

E/S Programada

- A CPU faz todo o trabalho;
- Um processo que deseja executar E/S monopoliza a CPU e o recurso de E/S até que a ação de E/S seja finalizada;
- Vantagem:
  - muito mais simples de programar;
- Desvantagem:

Formas de E/S



Interrupções

- Nível mais fundamental do suporte em software a E/S de dados;
- Fundamental em SOs modernos;
- Fundamental para o mecanismo de DMA;
- Passos para o tratamento de uma Interrupção:
  1. Salva registradores relevantes na pilha;
  2. Estabelece um contexto para a rotina de tratamento de interrupção. Isso pode envolver a configuração da TLB, MMU e uma tabela de páginas;
  3. Estabelece uma pilha para a rotina de tratamento da interrupção;

7

Passos para o tratamento de uma Interrupção

4. Sinaliza o controlador de interrupção. Se não existe um controlador de interrupção centralizado reabilita as interrupções;
5. Copia os registradores de onde eles foram salvos (possivelmente de alguma pilha) para a tabela de processos;
6. Executa a rotina de tratamento de interrupção. Ela extrairá informações dos registradores do controlador do dispositivo que está interrompendo;
7. Escolhe o próximo processo a executar. Se a interrupção deixou pronto algum processo de alta prioridade anteriormente bloqueado, este pode ser escolhido para executar agora;
8. Estabelece o contexto da MMU para o próximo processo a executar. Algum ajuste na TLB também pode ser necessário;
9. Carrega os registradores do novo processo, incluindo a PSW;
10. Inicializa a execução do novo processo;

8

Camadas de Software de E/S

- Dispositivos de entrada e saída podem ser muito simples ou inacreditavelmente complicados;
- Fazê-los funcionar em um sistema computacional é uma tarefa complexa que requer a cooperação de diversas partes do sistema;
- Parte da responsabilidade do funcionamento recai sobre o dispositivo propriamente dito e sobre o adaptador (hardware);
- No entanto, uma parcela considerável da responsabilidade pelo funcionamento recai sobre o software, em especial o Sistema Operacional;

9

Camadas de Software de E/S

- A arquitetura von Neumann é um conjunto de princípios de como organizar um sistema computacional;
- No entanto ela dita muito pouco sobre como operacionalizar o funcionamento da E/S com o restante do sistema;
- Fica a cargo das arquiteturas de sistemas computacionais específicos definir tais mecanismos;

10

Camadas de Software de E/S

- O Software de E/S é geralmente organizado em quatro camadas:
  - **Tratamento de Interrupções**;
  - **Drivers de Dispositivos**;
  - **Software de E/S Independente de Dispositivo**;
  - **Software de E/S do Espaço do Usuário**;
- A divisão em camadas visa:
  - Promover o desacoplamento entre partes do SO que são potencialmente danosas em caso de falha;
  - Facilitar o desenvolvimento do software necessário para controlar dispositivos de E/S;
- Função bem definida para cada camada e uma interface entre camadas igualmente bem definida;

11

Princípios do Software de E/S

- Alguns princípios são geralmente observados no desenvolvimento de software de E/S/:
- **INDEPENDÊNCIA DE DISPOSITIVO** – deve ser possível escrever programas de E/S sem a necessidade de especificar antecipadamente o dispositivo;

Ex: Não importa se o arquivo a ser lido está no HD, em um CD ou em um Pen Drive, o software de leitura deve ser exatamente o mesmo;

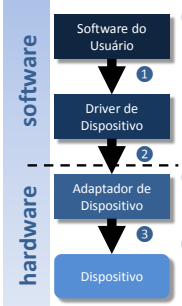
- **NOMEAÇÃO UNIFORME** – o nome de um arquivo ou dispositivo deve ser simlesmente uma cadeia de caracteres ou um inteiro totalmente independente do tipo do dipositivo;
- **TRATAMENTO DE ERROS** – possíveis erros no funcionamento ou acesso a informação produzida ou consumida por um dispositivo deveriam ser tratados o mais próximo possível do hardware

12

Princípios do Software de E/S

- **FORMA DE TRANSFERÊNCIA** – define claramente como os dados são transferidos do dispositivo para o adaptador. Há duas formas gerais utilizadas: a) comunicação síncrona (bloqueante) e b) assíncrona (orientada à interrupção);
- **FORMA DE ARMAZENAMENTO TEMPORÁRIO** – muitas vezes os dados provenientes de um dispositivo não podem ser armazenados diretamente em seu destino. Em geral são realizados como buffers de dados;
- **DISPOSITIVOS DEDICADOS VS COMPARTILHADOS** – define claramente se o dispositivo pode ser compartilhado entre um ou mais processos (e.g. um disco rígido) ou são de uso dedicado, ou seja, apenas um processo pode ser utilizado por vez (e.g. o teclado ou a impressora);

Estrutura Geral



1. chamadas abstratas de leitura e escrita;
   - inicializar o dispositivo;
   - gerenciar energia (alimentação);
   - registrar eventos (logging);
2. leitura e escrita de registradores de configuração;
3. Protocolo de comunicação de baixo nível;

Driver de Dispositivo

- Requer código específico pois controla intimamente o funcionamento do Adaptador que por sua vez controla o Dispositivo;
- Em geral é escrito pelo fabricante e fornecido juntamente com o dispositivo;
  - Drivers podem ser escritos igualmente por terceiros, esse geralmente foi o caso no Linux durante muito tempo;
- Cada driver de dispositivo trata de um dispositivo ou no máximo uma classe de dispositivos;
- Normalmente os drivers dos dispositivos devem ser parte do núcleo do sistema operacional;
- É possível no entanto escrever drivers que rodam no modo usuário;
- Além das chamadas abstratas de leitura e escrita, o driver de dispositivo também deve:
  - Inicializar o dispositivo;
  - Gerenciar energia (alimentação);
  - Registrar eventos (loging);

Driver em Modo Usuário

- Acessam o adaptador via chamadas do sistema;
- As syscalls permitem que os programas em modo usuário leiam e escrevam os registradores de controle dos adaptadores via Portas de E/S ou E/S mapeada em memória;
- Eles têm como grande vantagem o desacoplamento do Sistema Operacional;
  - Desta forma um driver em mal funcionamento não compromete todo o sistema operacional, apenas o dispositivo que ele controla;
- A desvantagem desta forma de desenvolvimento de drivers é que o processo de controlar um dispositivo tem que pular de modo usuário para núcleo várias vezes tornando o processo mais lento;
  - Naturalmente dispositivos críticos que empregam alta banda de comunicação têm problemas com esta estratégia;

Implementação de Drivers de Dispositivo

- Driver em modo Kernel são implementados como **MODULOS**;
- No LINUX pelo menos duas funções são necessárias:
  - `init_module;`
  - `cleanup_module;`
- Não há função principal (`main`);
- Drivers devem ser **reentrantes**;

Ex:

```
#define MODULE
#include <linux/module.h>

int init_module(void) { printk("<1>Hello, world\n"); return 0; }
void cleanup_module(void) { printk("<1>Goodbye cruel world\n"); }
```


Driver de Dispositivo

- Controlar o dispositivo significa emitir uma sequência de comandos para ele;
- O driver é a parcela de código responsável por determinar a sequência de comandos;
- Os comandos são repassados ao adaptador por meio da escrita dos registradores de controle.
- Eventualmente pode ser necessário ler o estado do registrador de estado do dispositivo para determinar se o adaptador aceitou a configuração;
- Uma vez configurado uma de duas situações podem ocorrer:

① Na maioria dos casos, o driver do dispositivo espera até que o controlador faça algum trabalho para ele, de modo que ele se autobloqueia até que uma interrupção venha a desbloqueá-lo;

② Em outros casos porém, a operação finaliza sem atraso, de maneira que o driver não precise bloquear;

Interfaces de Drivers de Dispositivos

- Conjunto de funções que o driver invoca do SO e que o SO invoca do driver;
- A interface deve ser uniforme (mesmas funções sempre) para que o SO possa tratar os drivers de maneira regular;
- Na prática esta regularidade nem sempre pode ser alcançada;

19

Estrutura Geral dos Drivers de Dispositivo

1. Verifica se os parâmetros de entrada são válidos;
2. Caso não sejam válidos um erro é gerado;
3. Em alguns casos pode ser necessário que os parâmetros sejam interpretados/processados (e.g. um disco precisa calclular o cilindo, trilha, setor, a partir do nº do bloco);
4. Verifica se o dispositivo está atualmente em uso;
5. Se o dispositivo estiver ocioso o status do hardware será examinado para ver se a requisição pode ser tratada imediatamente;
6. Se o dispositivo estiver ocupado, a requisição será enfileirada;

20

Software de E/S Independente de Dispositivo

a. Uniformizar interfaces para os drivers de dispositivos;
b. Armazenar no buffer dados e controle;
c. Reportar erros;
d. Alocar e liberar dispositivos dedicados;
e. Providenciar um tamanho de bloco independente de dispositivo;

- Fazer todos os dispositivos de E/S e drivers parecerem mais ou menos os mesmos para o restante do sistema operacional;
- As funções definidas no SO passíveis de serem chamadas pelos drivers devem sempre ser as mesmas. O mesmo é válido para as funções definidas nos drivers passíveis de serem chamadas pelo SO;

21

Bibliografia

22