Sistemas Operacionais
Aula 3 – Chamada de Sistema e Interrupção; Processo

Chamada de Sistemas

Se uma aplicação precisa realizar alguma instrução privilegiada (imprimir um arquivo) ela realiza uma chamada de sistema, que altera do modo usuário para o modo kernel;

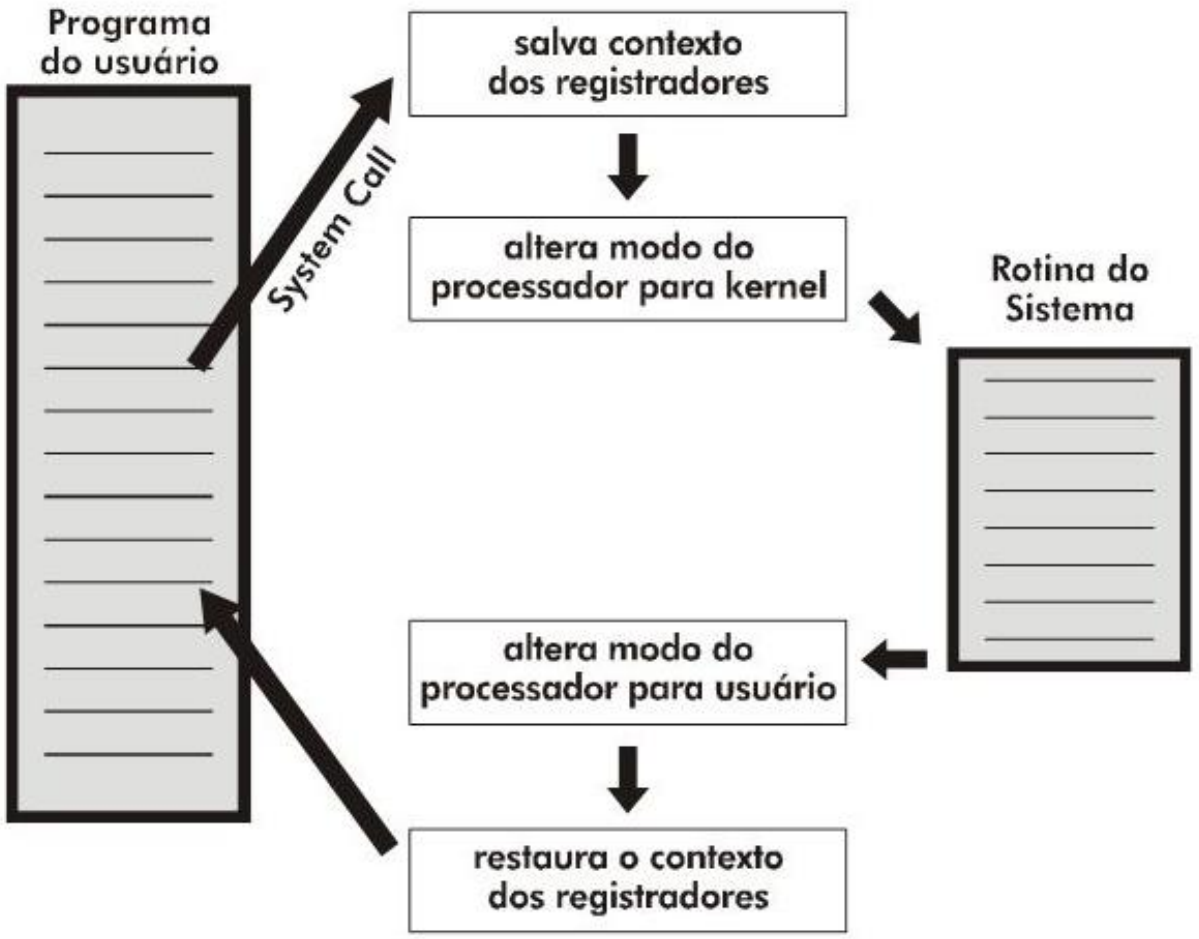
outros ex. Ler um arquivo

Chamada de sistema são a porta de entrada para o modo kernel.

Como elas são realizadas?

- As chamadas de sistemas são realizadas através de instruções Traps
- Traps são conhecidas como interrupções de software
- Após o término da chamada (ex. ler um arquivo) a execução continua após a chamada de um sistema

Chamada a um rotina do sistema

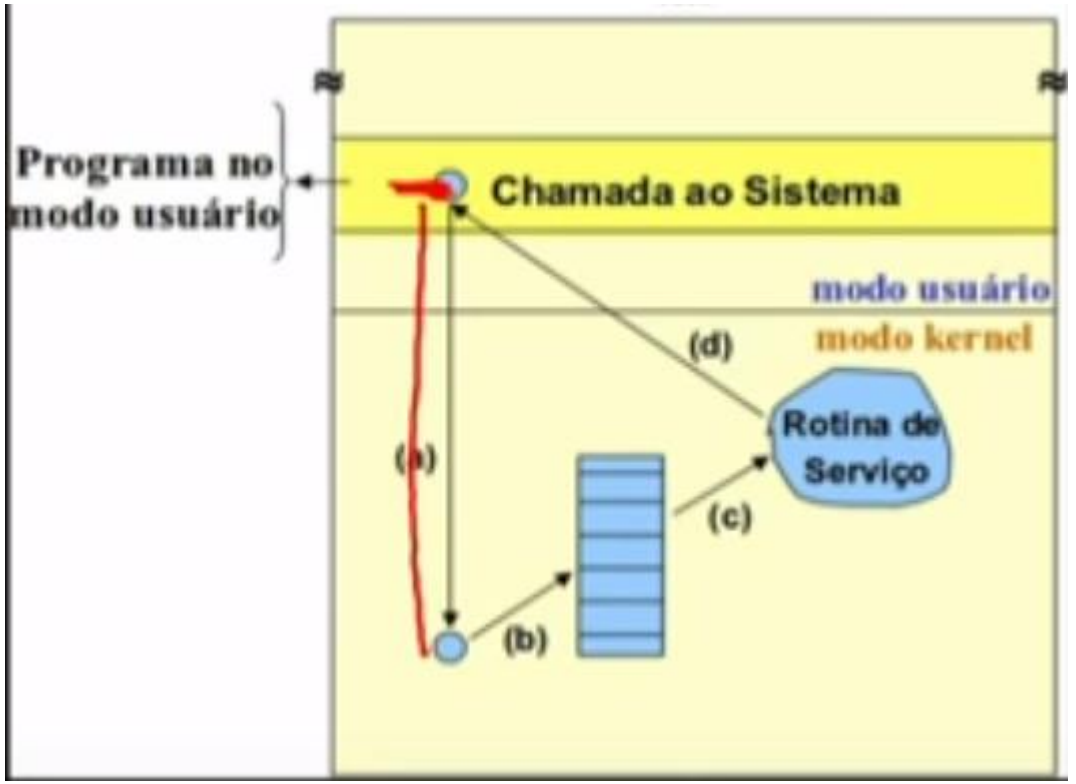


(a) Aplicativo faz a chamada ao sistema (Trap)

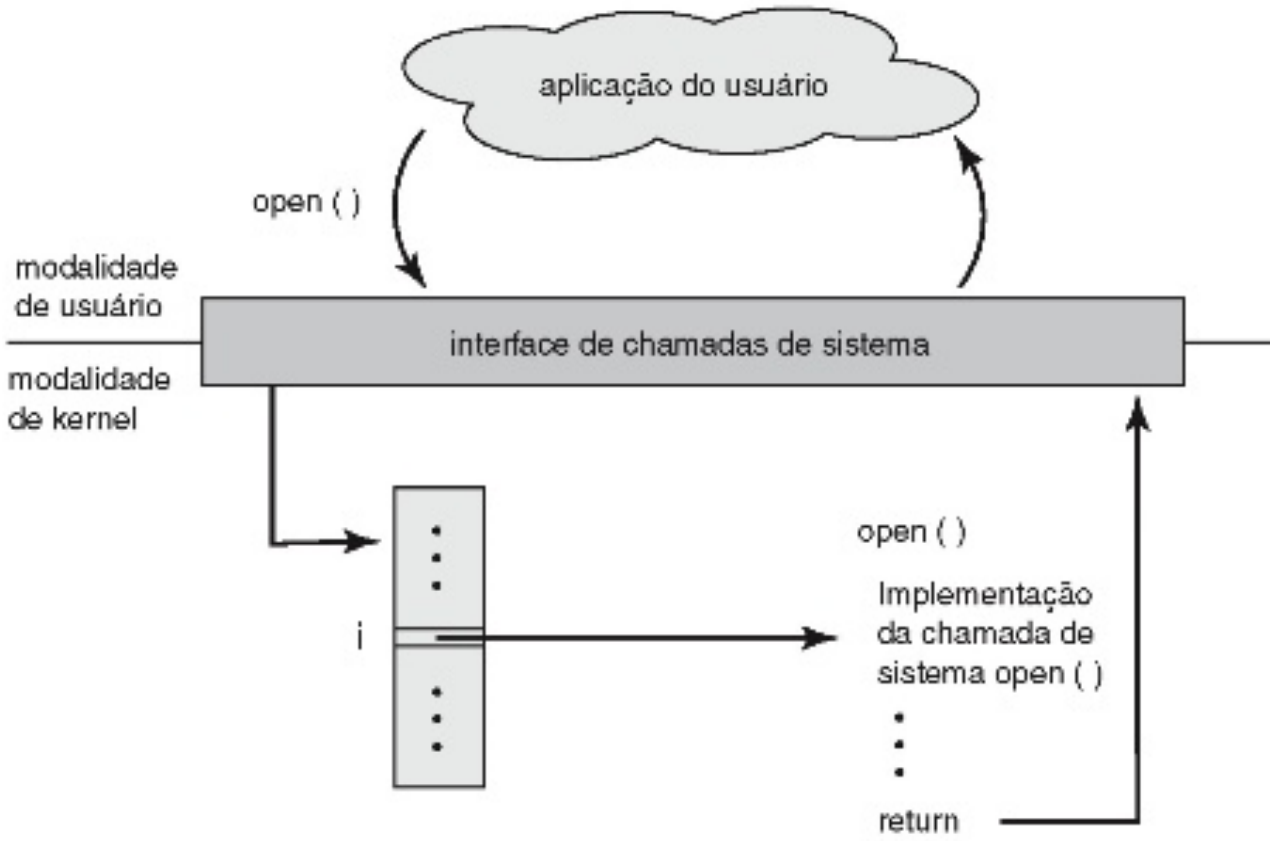
(b) Através de uma tabela, o SO determina o endereço da rotina

(c) Rotina de Serviço é acionada (rotina compartilhada)

(d) Serviço solicitado é executado e o controle retorna ao aplicativo



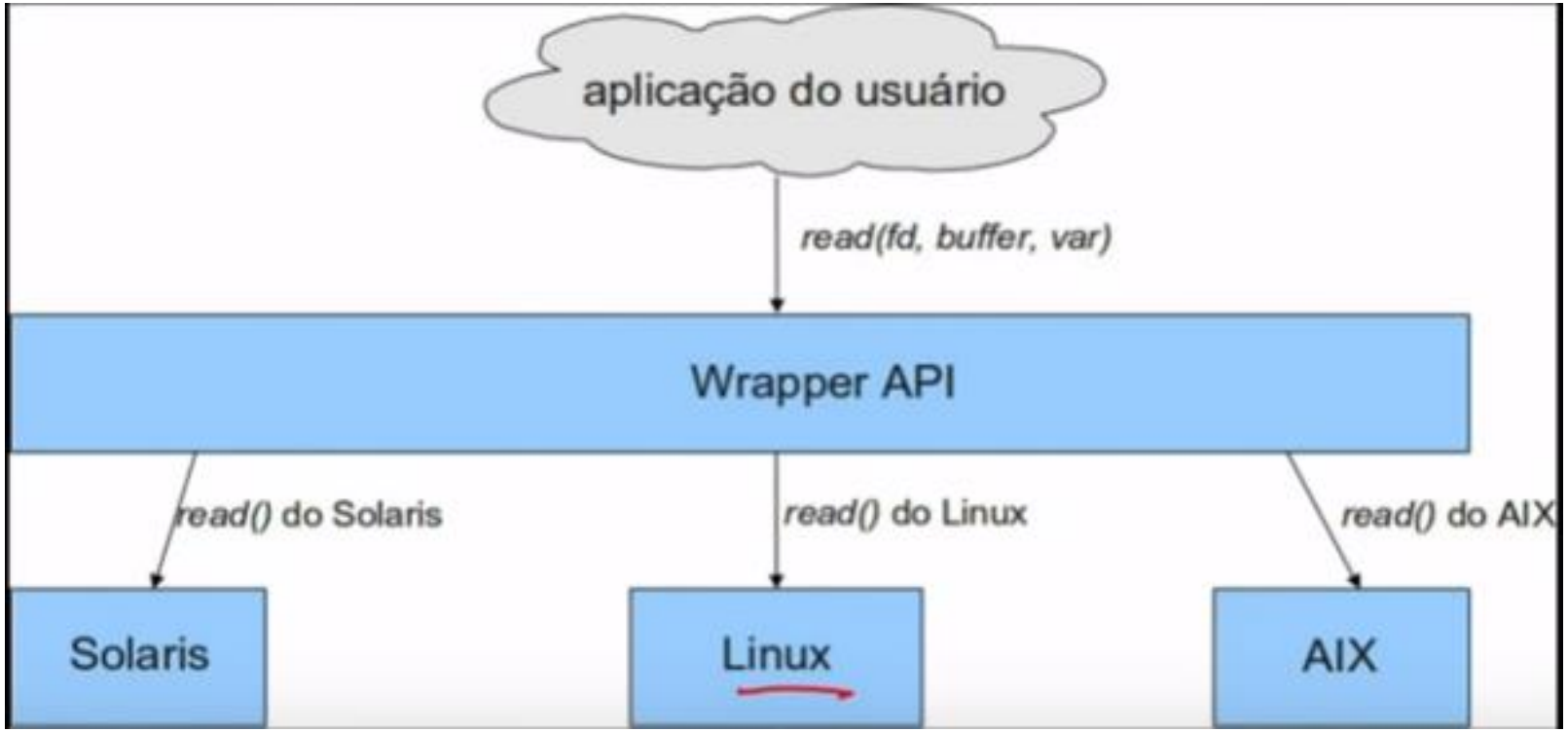
Exemplo com chamada open()



Interface das SysCalls

- Interface para esconder a complexidade das Syscalls.
- Interface de programação fornecida pelo SO
- Geralmente escrita em linguagem de alto nível (C, C++ ou Java)
- Normalmente as aplicações utilizam uma Application Program Interface (API)
- Interface que encapsula o acesso direto às chamadas ao sistema

Portabilidade usando Wrappers



Interface das SysCalls

- Interface das Chamadas de Sistemas (Wrappers) mais utilizadas:
  - Win32 API para Windows
  - POSIX API para praticamente todas as versões de UNIX
  - Java API para a Java Virtual Machine (JVM)

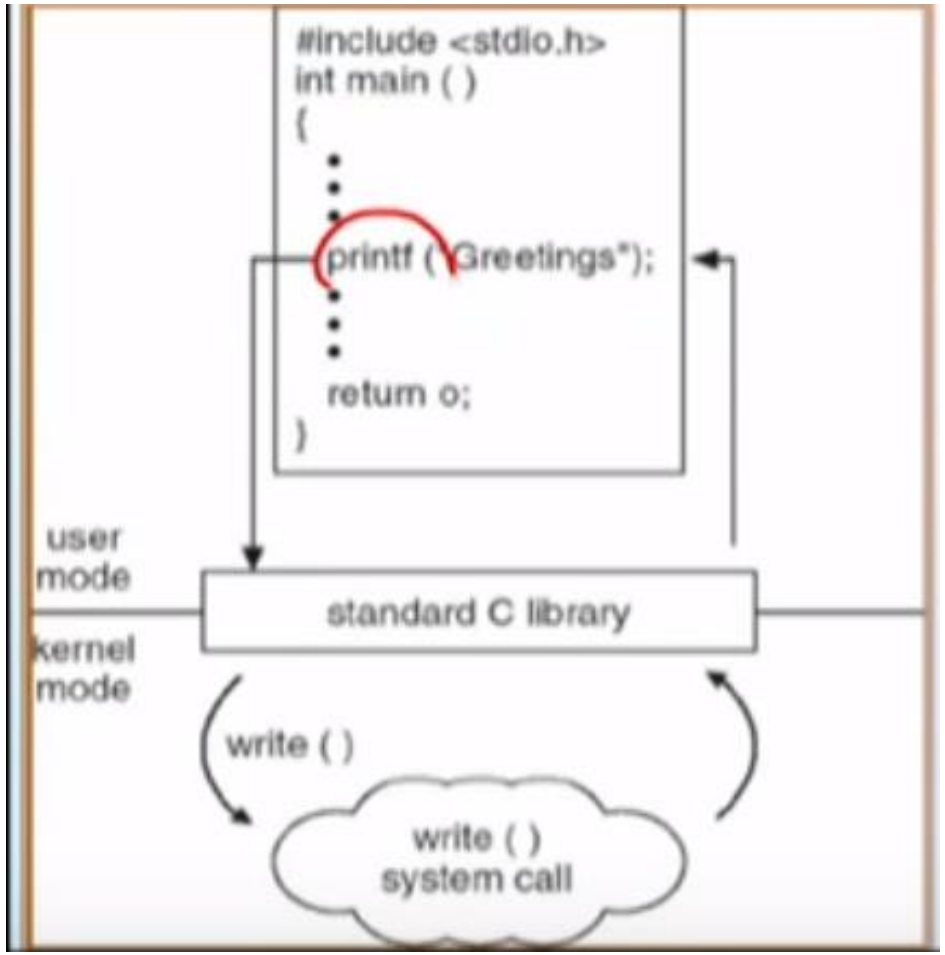
Interface das SysCalls

- Motivos para utilizar APIs em vez de chamada ao sistema diretamente:
  - Portabilidade – independência da plataforma
  - Esconder complexidade inerentes às chamadas ao sistema
  - Acréscimo de funcionalidades que otimizam o desempenho

Exemplo de Uso com printf()

- Programa em C que invoca a função de biblioteca printf(), que por sua vez chama o system call write()

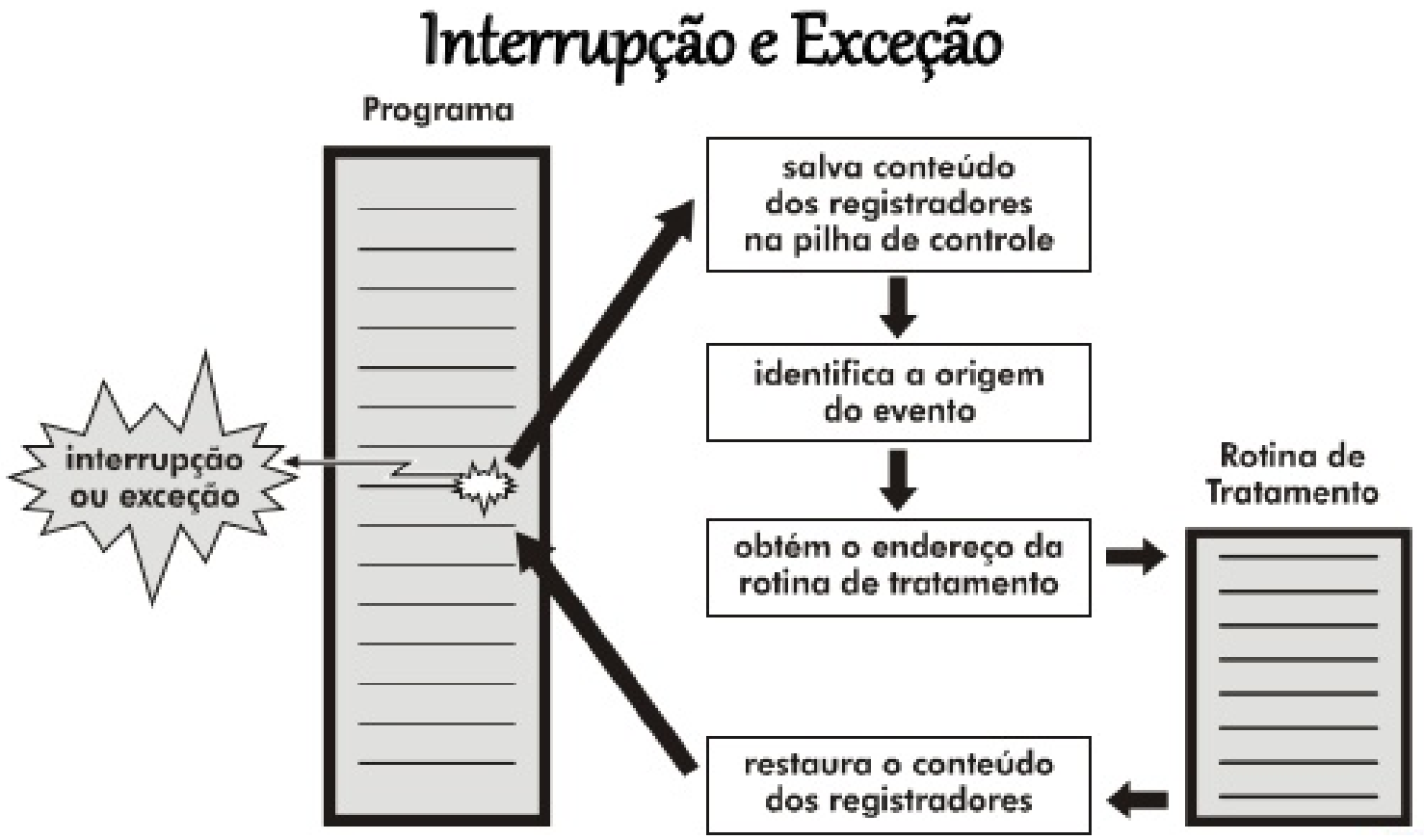
- A chamada printf() ocasiona a chamada write() e exit()



Interrupções

- Vimos que um software pode interromper seu próprio processo (ao fazer uma chamada ao sistema):
  - Usando traps (Interrupções de software ou Exceções)
  - Para isso, a aplicação tem que estar rodando.
- Mas ocorrem interrupções que não são causadas por aplicações em execução:
  - Interrupção de hardware (eventos externos)
  - Um sinal elétrico no hardware
  - Causa: dispositivos de E/S ou o clock

Passos para Tratar a Interrupção



Interrupção vs. Traps

- Interrupção
  - Evento externo ao processador
  - Gerada por dispositivos que precisam da atenção do SO
  - Pode não estar relacionada ao processo que está rodando

- Traps
  - Eventos inesperados vindo de dentro do processador
  - Causados pelo processo corrente no processador (seja por chamada ao SO, seja por interrupção ilegal)

Processos

Definição de Processo

- Um processo é caracterizado por um programa em execução

- Diferença entre processo e programa?
  - Um processo é uma instância de um programa e possui dados de entrada, dados de saída e um estado (executando, bloqueado, pronto)

Programa vs. Processo

- Um programa pode ter várias instâncias em execução (em diferentes processo)
- Forma como o programador vê a tarefa a ser executada

- Um processo é único
- Código acompanhando de dados e estado
- Forma pela qual o SO vê um programa e possibilita sua execução

Processo em Primeiro Plano

- Interage com o usuário
  - Ex. Ler um arquivo; Iniciar um programa (linha de comando ou um duplo clique no mouse).

Processo em Segundo Plano

- Processo com funções específicas que independem de usuários – daemons:
  - Ex. Recepção e envio de e-mails; Serviço de impressão

Cada Processo possui

- Conjunto de instruções
- Espaço de endereçamento (espaço reservado para que o processo possa ler e escrever – 0 até max)
- Contexto de hardware (valores nos registradores, como PC, ponteiro de pilha, e reg. De prop. Gerais)
- Contexto de software (atributos em geral, como lista de arquivos abertos, vaiáveis, etc.

Espaço de endereçamento

- Texto: código executável do(s) programas(s)
- Dados: as variáveis
- Pilha de Execução:
  - Controla a execução do processo
  - Empilhando chamadas a procedimento, seus parâmetros e variáveis locais, etc.

Tabela de Processos

- Também chamada de BCP (Bloco de Controle de Processo)
- Contém informações de contexto de cada processo (ex. ponteiro de arquivo abertos, posição do próximo byte a ser lido em cada arquivo, etc.
- Contém informações necessárias para trazer o processo de volta, caso o SO tenha que tirá-lo de execução
- Contém estados de um processo em um determinado tempo.

Tabela de Processos

Gerenciamento de processos	Gerenciamento de memória	Gerenciamento de arquivos
Registradores	Ponteiro para o segmento de código	Diretório-raiz
Contador de programa	Ponteiro para o segmento de dados	Diretório de trabalho
Palavra de estado do programa	Ponteiro para o segmento de pilha	Descritores de arquivos
Ponteiro de pilha		Identificador (ID) do usuário
Estado do processo		Identificador (ID) do grupo
Prioridade		
Parâmetros de escalonamento		
Identificador (ID) do processo		
Processo pai		
Grupo do processo		
Sinais		
Momento em que o processo iniciou		
Tempo usado da CPU		
Tempo de CPU do filho		
Momento do próximo alarme		

Tabela de Processos

- O BCP só não guarda o conteúdo do espaço de endereçamento do processo

- Assim, um processo é constituído de seu espaço de endereçamento BCP (com seus registradores, etc). Representando uma entrada na tabela de processos.

Características de Processo

- Processos CPU-bound (orientados à CPU): processo que utilizam muito o processador
  - Tempo de execução é definido pelos ciclos de processador;
- Processo I/O-bound (orientados à E/S)
  - Tempo de execução é definido pela duração das operações de E/S)
- Ideal: existir um balanceamento entre processos CPU-bound e I/O-bound

Criação de Processos

- Inicialização do sistema
- Execução de uma chamada de sistema para a criação de processo, realizado por algum processo em execução
- Requisição de usuário para criar um novo processo (duplo clique do mouse)
- Inicialização de um processo em batch (em sistemas mainfrmaes com proc. em batch

Processos criando outros processos

- No Unix com a função fork()
  - Cria clone do processo Pai: cópias extas na memória, mas com identificadores diferentes

- No Windows com o CreateProcess
  - Cria processo Filho, já carregando novo programa nele

Criando processo com fork()

```
int main(int argc, char*argv[]){
        int pid;
        pid=fork();
        printf(“%d”,num);
        if(pid==0){ /*filho*/
                num=1;}
        else if(pid>0){ /*pai*/
                num=2;}
        printf(“%d”,num);}
```

Finalizando Processos

- Término normal (voluntário)
  - A tarefa a ser executada é finalizada
  - Ao terminar, o processo executa uma chamada (comunicando o SO que terminou): exit(UNIX) e ExitProcesso(Windows)

- Término por erro (voluntário):
  - O processo sendo executado não pode ser finalizado. Ex. gcc filename.c; o arquivo filename.c não existe

Finalizando Processos

- Término com erro fatal (involuntário):
  - Erro causado por algum erro no programa (bug)
  - Ex. Divisão por 0(zero); Referência à memória inexistente; Execução de uma instrução ilegal.

- Término (involuntário) causado por algum outro processo via chamada a:
  - Kill(UNIX)
  - TerminateProcess(Windows)

Estados de Processos

- Executando: realmente usando a CPU naquele momento
- Bloqueado: incapaz de executar enquanto um evento esterno não ocorrer
- Pronto: em memória, pronto para executar (ou para continuar sua execução), apenas aguardado a possibilidade do processador

Transições entre Estados

1- O processo bloqueia aguardando uma entrada

2- O escalonador seleciona outro processo

3- O escalonador seleciona esse processo

4- A entrada torna-se disponível