Capítulo 2

Processos e Threads

2.1 Processos

2.2 Threads

2.3 Comunicação entre Processos

2.4 Escalonamento

2.5 Problemas Clássicos de IPC

Processos

O Modelo de Processo

Definição:

Uma instância de um *PROGRAMA EM EXECUÇÃO* +

- PC
- REGISTRADORES
- VARIÁVEIS DE ESTADO

Características principais:

✓ cada processo tem sua própria "CPU virtual"

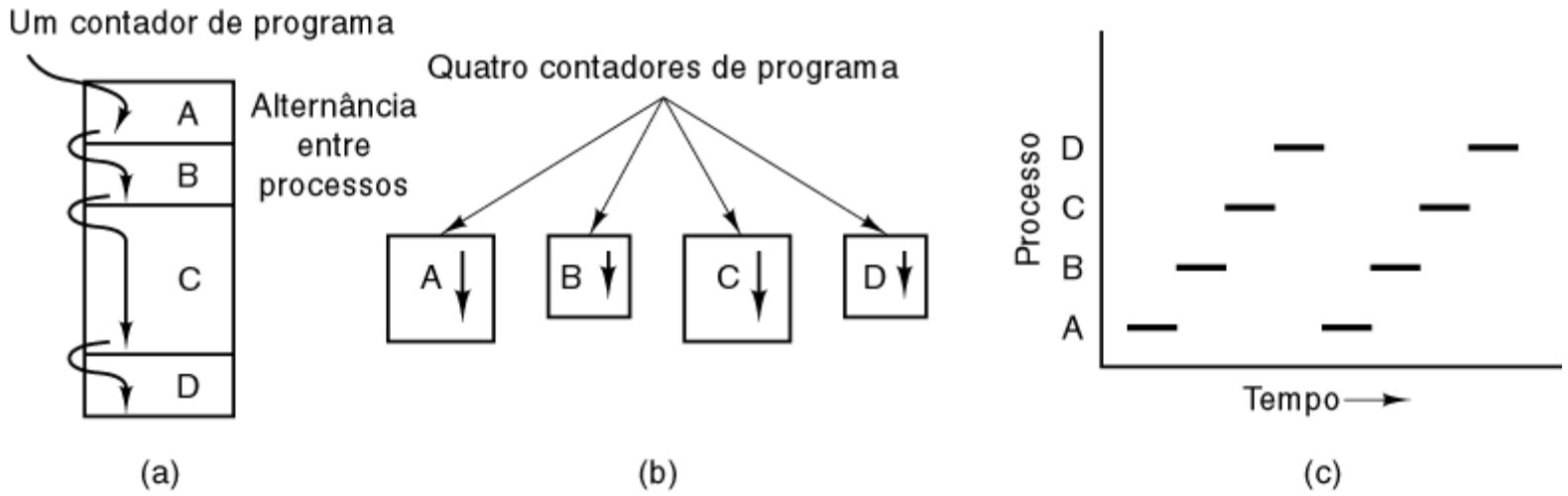
→ CPU alterna entre os diversos processos

⇒ ***Multiprogramação***

✓ cada processo tem seu próprio espaço de endereçamento

✓ podem ser criados e terminados dinamicamente

O Modelo de Processo



a) Multiprogramação de quatro programas na memória. Modelo conceitual de 4 processos sequenciais, independentes

b) Cada processo com seu próprio fluxo de controle. Somente um programa está ativo a cada momento (somente um contador de programa físico e 4 contadores lógicos, salvos quando acaba o tempo de CPU do processo)

c) A cada instante, somente um processo é executado

O Modelo de Processo

Ideia principal:

- ✓ um processo constitui uma atividade
- ✓ possui um programa, entrada, saída e um estado
- ✓ um único processador compartilhado entre vários processos
- ✓ algum ***algoritmo de escalonamento:***

  ⇒ usado para determinar quando parar o trabalho sobre um processo e "servir" outro.

Obs: um programa executando duas vezes = 2 processos distintos

Ex: ***iniciar um processador de texto duas vezes***

Criação de Processos

- Sistemas operacionais precisam criar processos.

- Quatro eventos principais fazem com que os processos sejam criados:

1. ***Inicialização do sistema.***

2. ***Execução de uma chamada de sistema de criação de processo por um processo que está em execução.***

3. ***Solicitação de um usuário para criar um novo processo.***

4. ***Início de uma tarefa em lote.***

Criação de Processos

1º caso: ***DURANTE A CARGA DO SO CRIAM-SE VÁRIOS PROCESSOS***

- **Processos em primeiro plano:**
  interagem com os usuários e realizam tarefas para eles.

- **Processos em segundo plano:**
  não estão associados a um usuário particular.

  ⟶ Apresentam funções específicas.

  Ex: aceitar emails ou **solicitações de páginas web**

  denominados ***daemons***

Pode-se vê-los: **UNIX** – ***ps:*** lista os processos em execução

Windows – gerenciador de tarefas

Criação de Processos

2º caso

- um processo em execução pode emitir chamadas ao sistema

  ⇒ criar um ou mais novos processos para ajudá-lo em seu trabalho;

- utilidade de criar novos processos?

  R: a tarefa a ser executada pode ser resolvida através da execução de vários processos relacionados, mas interagindo de maneira independente;

Ex: uma grande quantidade de dados trazida via rede. Um processo traz os dados enquanto um segundo processo remove os dados e os processa

Criação de Processos

- Outra utilidade (2º caso):
  → sistema multiprocessador

⇒ **trabalho mais rápido (cada processo executa numa CPU)**

3º caso

- Sistemas interativos:
  → dois cliques em um ícone ou a digitação de um comando inicia um novo processo e executa nele o programa selecionado

4º caso (aplicado somente a computadores de grande porte)

- Usuários submetem ***tarefas em lote*** para o sistema
- Processos são criados ***(e entram na fila***). De acordo com a disponibilidade de recursos, SO decide se pode executar a próxima tarefa da fila.

Criação de Processos

Em todos os casos:

→ **processo filho é criado por um processo existente (processo pai) através de uma *chamada de sistema***

Tanto no UNIX quanto no Windows, depois que um processo é criado:

→ **pai e filho têm seus *próprios e distintos* espaços de endereçamento (apesar do conteúdo ser igual, a princípio)**

⇒ ***a mudança de uma palavra em um espaço de endereçamento não será visível ao outro processo***

Criação de Processos

- Algumas implementações compartilham o ***código do programa***, pois este não pode ser alterado;

- **Nenhuma memória para escrita é compartilhada.** Algumas vezes, alguma memória pode ser compartilhada, desde que não se escreva nela **(*copy-on-write*).**

- É possível o compartilhamento de ***arquivos abertos*** (com o processo que o criou).

Término de Processos

Condições que levam ao término de processos:

1. **Saída normal (voluntária)**
2. **Erro fatal (involuntário)**
3. **Saída por erro (voluntária)**
4. **Cancelamento por um outro processo (involuntário)**

Término de Processos

Maioria das vezes:

→ **término pelo fim do trabalho;**

Ex: compilador acaba sua tarefa e envia uma chamada ao SO para dizer que terminou;

(UNIX: exit; Windows ExitProcess)

Segunda razão:

→ **processo descobre um erro fatal;**

Ex: usuário manda compilar um arquivo que não existe.

⇒ compilador emite uma chamada de saída do Sistema!

cc foo.c (foo.c não existe!)

Término de Processos

Terceira razão:

→ Erro de programa;

Ex: divisão por zero, execução de instrução ilegal,

referência a memória inexistente

Quarta razão:

→ chamada ao sistema operacional p/ cancelar um processo.

Ex: comando ***kill*** (UNIX) ou

TerminateProcess (Windows)

Obs: 1. Em ambos os casos o usuário deve ter autorização para isso!!

2. Em alguns sistemas, o término do processo pai finaliza todos os seus filhos (não é o caso do Unix nem do Windows!)

Hierarquias de Processos

- **pai cria um processo filho**
- **processo filho pode criar outros processos**
- **um processo tem apenas um pai**

⇒ ***formam uma hierarquia de processos***

No UNIX: ***pai, filho e demais descendentes formam um grupo de processos***

No Windows: não existe o conceito de hierarquia de processos

⇒ ***Todos os processos são criados iguais!!***

Hierarquias de Processos

Quando um processo cria outro:

→ **processos pai e filho estão *associados* após a criação;**

Exemplo:

- usuário envia um sinal do teclado, o sinal é entregue a todos os membros do **grupo de processos associado** com o teclado;

- cada processo pode *capturar* o sinal, *ignorá-lo* ou *tomar uma ação* pré-definida (ex: ser cancelado pelo sinal)

Hierarquias de Processos

UNIX:

- processo ***init*** na ***carga do sistema:***

  → Lê arquivo com o número de terminais existentes

  (se bifurca em um novo processo para cada terminal)

- Processos esperam a conexão de algum usuário:

  → processso de conexão cria ***Shell (interpret. de comandos)***

- comandos do usuário podem iniciar mais processos...

  ...e assim sucessivamente.

Hierarquias de Processos

UNIX (cont.):

Init: **raiz da árvore**

⇒ ***Todos*** os processos pertencem à mesma árvore com a raiz ***init***

Windows:

Não existe herarquia de processos!

Todos os processos são iguais;

→ um processo pode controlar o outro através de uma identificação, mas "o pai" pode passar esta identificação para qualquer outro processo

⇒ **invalida o conceito de hierarquia!!!**

Estados de Processos

Processos são independentes (próprio contador e próprio estado interno) mas, muitas vezes, precisam interagir uns com os outros

Um processo pode bloquear em duas situações:

1. **Não pode prosseguir,** esperando por uma entrada que ainda não está disponível

   Ex: cat chap1 chap2 | grep tree

   grep está pronto para executar, mas ainda não existe ***entrada*** para ele devido ao tempo de CPU que cada processo deteve

   ⇒ bloqueia até que exista entrada disponível!!!

Estados de Processos

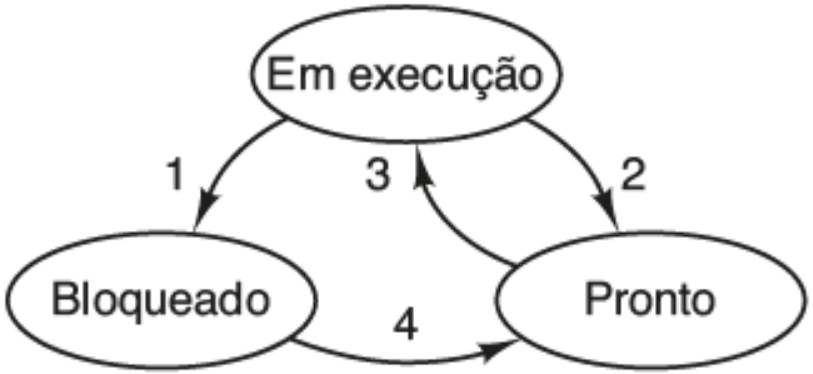
2. **Está pronto para executar, mas CPU é alocada para outro processo**

primeiro caso é inerente ao problema.

Ex.: programa não pode prosseguir porque está esperando algum dado.

segundo caso depende de onde esteja executando (falta de CPU)

Estados de Processos



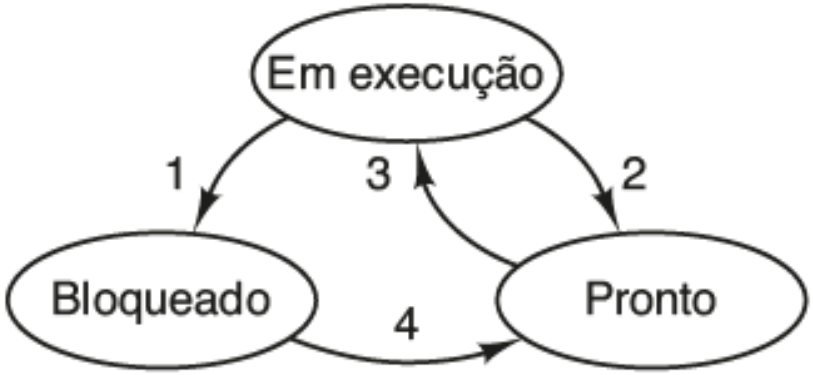
Três estados nos quais um processo pode se encontrar:

a) **Em execução** (realmente usando a CPU naquele instante).

b) **Pronto** (executável, temporariamente parado para deixar outro processo ser executado).

c) **Bloqueado** (incapaz de ser executado até que algum evento externo aconteça).

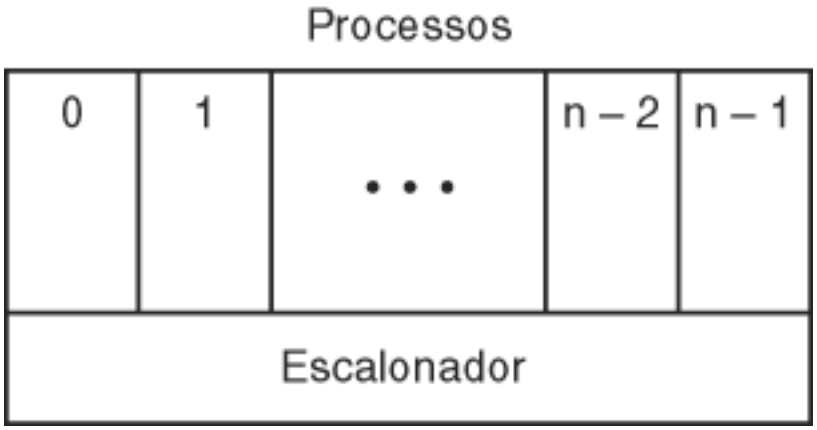
Estados de Processos



Possíveis transições de estados:

1. **SO** descobre que o processo não pode continuar agora.
   - ✓ Processo executa uma chamada de sistema do tipo ***pause***
   - ✓ Falta alguma **entrada *(ex: grep)*** ou outro **evento externo**
2. **Escalonador** decide trocar o processo em execução.
3. **Escalonador** decide que chegou a hora do processo executar (os outros já ocuparam tempo suficiente de CPU).
4. **Entrada** ou **evento externo,** pelo qual o processo estava esperando, acontece

Estados de Processos



Escalonador:

nível mais baixo de um SO estruturado em processos

Funções:

- escolher o próximo processo a ser executado ***(escalonamento)***;
- tratar acessos aos dispositivos de E/S e consequentes interrupções. Ex: processos de disco e de terminais

Obs: acima do escalonador estão os processos sequenciais

Implementação de Processos

Para implementar o modelo de processos:

- SO mantém uma tabela (***tabela de processos***)
  - ✓ uma entrada para cada processo;
  - ✓ cada entrada contém informações do ***estado do processo:***
    - **PC**
    - **SP (ponteiro para a pilha)**
    - **Alocação de memória**
    - **Estados de seus arquivos abertos**
    - **Infos de contabilidade e parâmetros de escalonamento**
    - **Todas as informações salvas quando da mudança de estado do processo** (***execução*** p/ ***pronto*** ou p/ ***bloqueado***)

<u>Objetivo:</u> reiniciar o processo como se nunca tivesse sido bloqueado

Implementação 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		

Campos da entrada de uma tabela de processos

Implementação de Processos

Tratamento de interrupções

Resumo do que o nível mais baixo do SO faz quando ocorre uma interrupção de algum dispositivo de E/S, por exemplo

1. O hardware empilha o contador de programa etc.
2. O hardware carrega o novo contador de programa a partir do vetor de interrupção.
3. O procedimento em linguagem de montagem salva os registradores.
4. O procedimento em linguagem de montagem configura uma nova pilha.
5. O serviço de interrupção em C executa (em geral lê e armazena temporariamente a entrada).
6. O escalonador decide qual processo é o próximo a executar.
7. O procedimento em C retorna para o código em linguagem de montagem.
8. O procedimento em linguagem de montagem inicia o novo processo atual.

Modelando a multiprogramação

A princípio, se um processo de tamanho médio ocupa a CPU 20% tempo no qual está na memória (80% do tempo fica esperando por uma E/S)

⇒ colocar 5 processos na memória para utilização máxima

Mas… processos podem estar esperando por uma E/S ao mesmo tempo!!!

Modelo melhor (probabilístico):

n* – *número de processos na memória

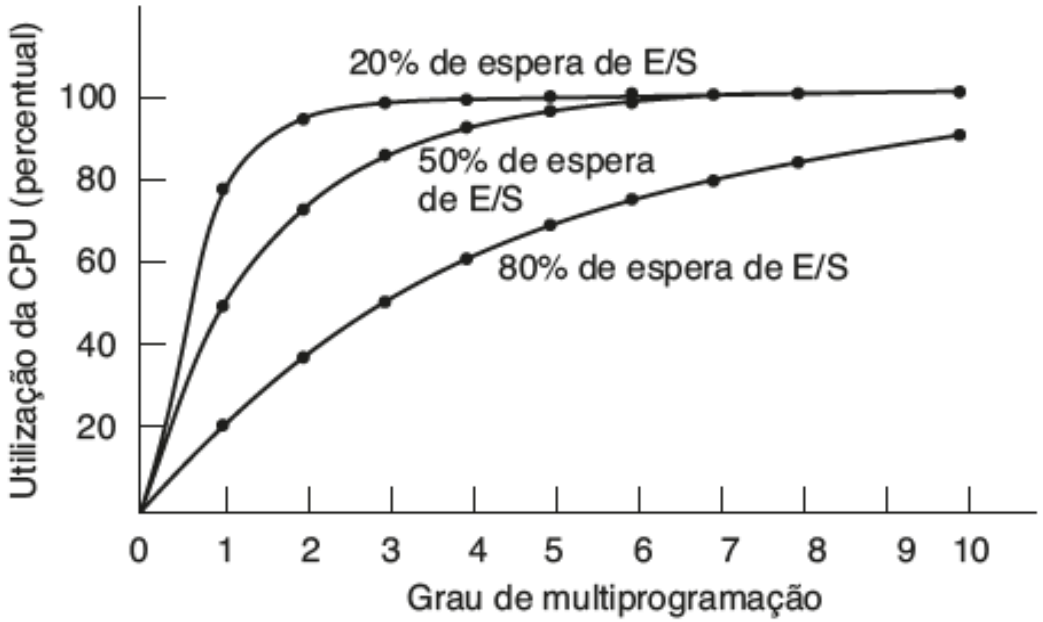
P* – *probabilidade do processo estar esperando por uma E/S

P^n* – *probabilidade dos n processos estarem esperando por uma E/S ao mesmo tempo

U* – *utilização da CPU

$$U = 1 - P^n$$

Modelando a multiprogramação



- Quando a multiprogramação é usada, a utilização da CPU pode ser aperfeiçoada.

- A figura mostra a utilização da CPU como uma função de *n*, que é chamada de **grau de multiprogramação.**