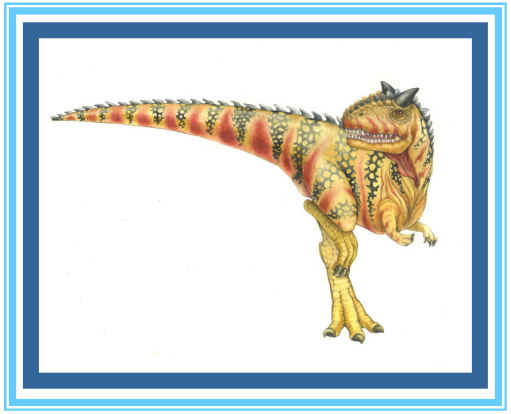
Capítulo 3: Processos



Sobre a apresentação (About the slides)

Os slides e figuras dessa apresentação foram criados por Silberschatz, Galvin e Gagne em 2009. Esse apresentação foi modificada por Cristiano Costa (cac@unisinos.br). Basicamente, os slides originais foram traduzidos para o Português do Brasil.

É possível acessar os slides originais em http://www.os-book.com

Essa versão pode ser obtida em http://www.inf.unisinos.br/~cac

The slides and figures in this presentation are copyright Silberschatz, Galvin and Gagne, 2009. This presentation has been modified by Cristiano Costa (cac@unisinos.br). Basically it was translated to Brazilian Portuguese.

You can access the original slides at http://www.os-book.com

This version could be downloaded at http://www.inf.unisinos.br/~cac

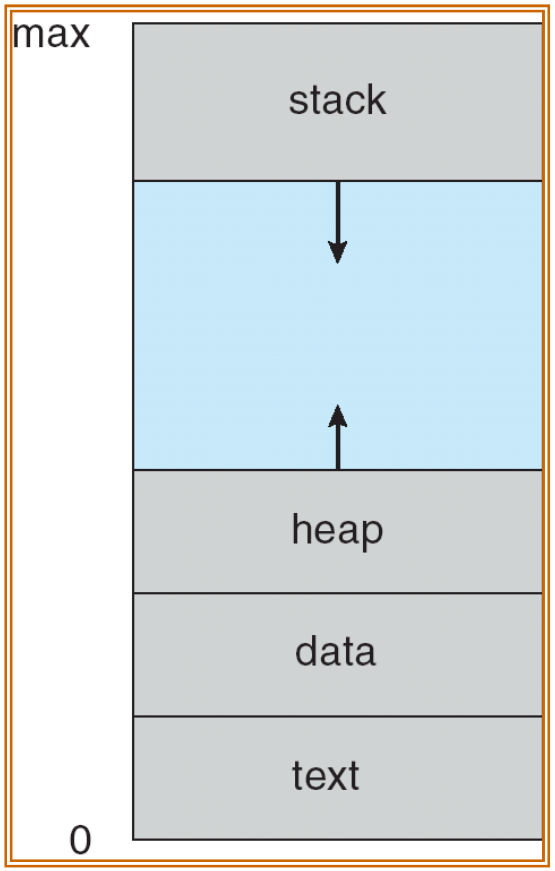
Capítulo 3: Processes

- Conceito de Processo
- Escalonamento de Processos
- Operações com Processos
- Processos Cooperativos
- Comunicação entre Processos
- Comunicação em sistemas Cliente-Servidor

Conceito de Processo

- Um sistema operacional executa uma variedade de programas:
  - Sistema Batch – *jobs*
  - Sistema Tempo Compartilhado (*Time-shared)* – programas do usuário ou tarefas
- Livros usam os termos *job* e *processo* quase que indeterminadamente.
- Processo – um programa em execução; execução do processo deve progredir de maneira seqüencial.
- Um processo inclui:
  - Contador de programa
  - Pilha
  - Seções de dados

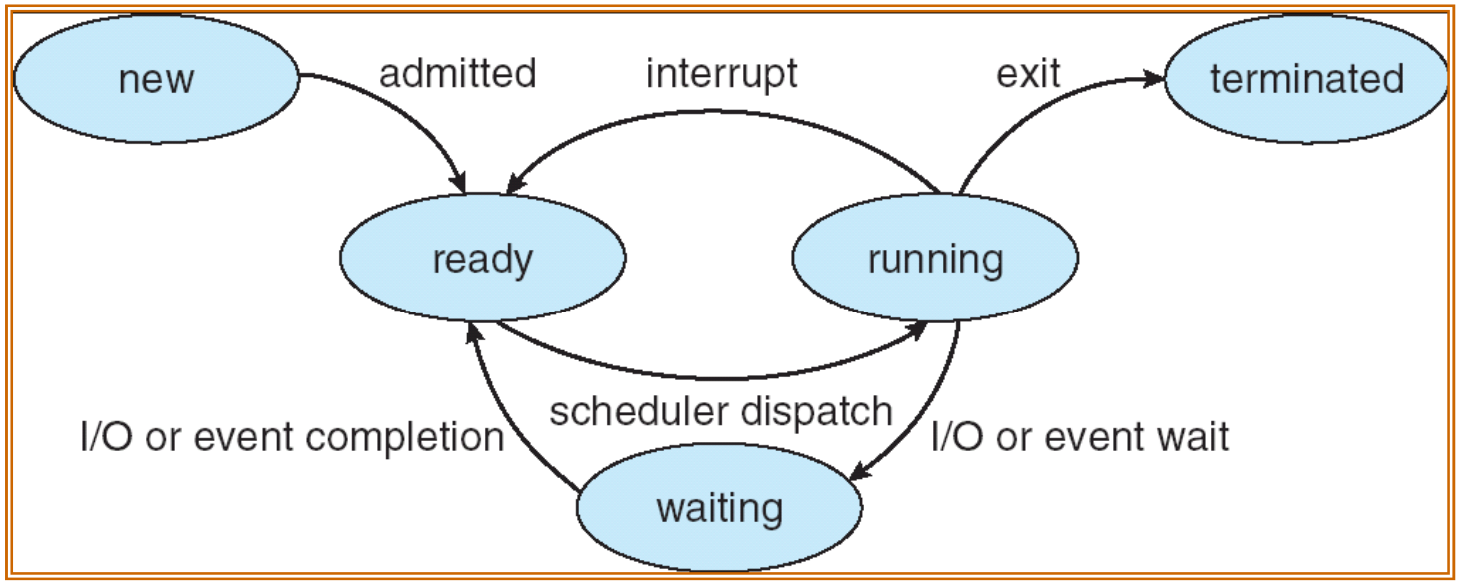
Processo na Memória



Estados de Processo

- Durante a execução de um processo, ele altera seu *estado*
  - **Novo** (*new*): O processo está sendo criado.
  - **Executando** (*running*): instruções estão sendo executadas.
  - **Esperando** (*waiting*): O processo está esperando algum evento acontecer.
  - **Pronto** (*ready*): O processo está esperando ser associado a um procesador.
  - **Terminado** (*terminated*): O processo terminou sua execução.

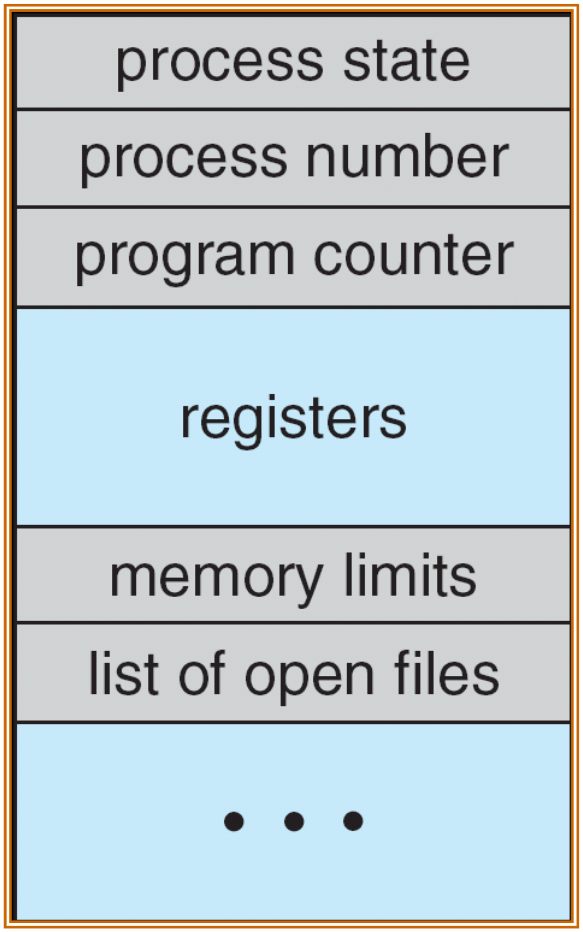
Diagrama de Estados de Processos



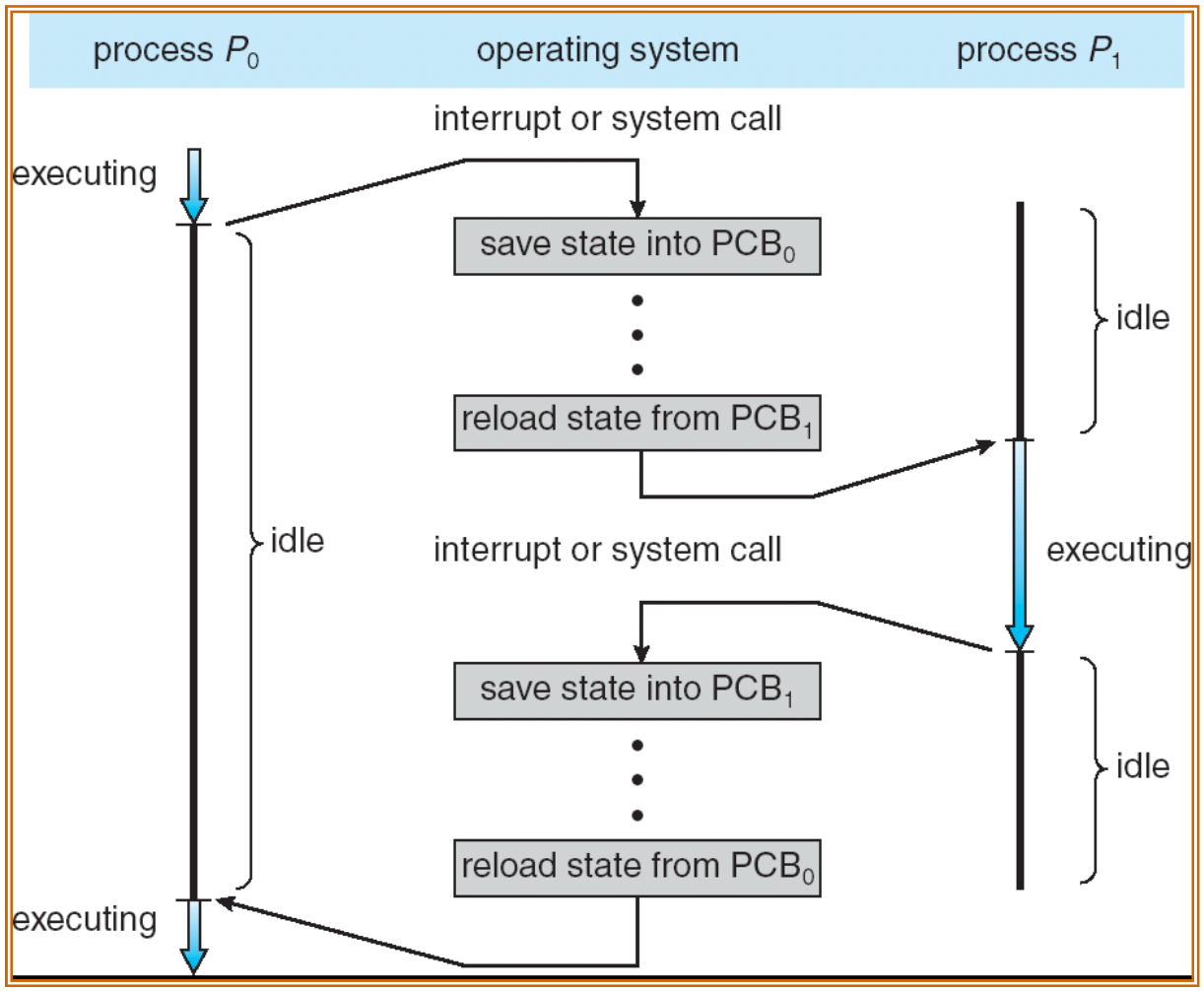
Process Control Block (PCB)

- A PCB ou Bloco de Controle de Processos armazena informações associada com cada processo.
  - Estado do Processo
  - Contador de Programas
  - Registradores da CPU
  - Informações de escalonamento da CPU
  - Informação de Gerenciamento de memória
  - Informação para Contabilidade
  - Informações do status de E/S

Process Control Block (PCB)



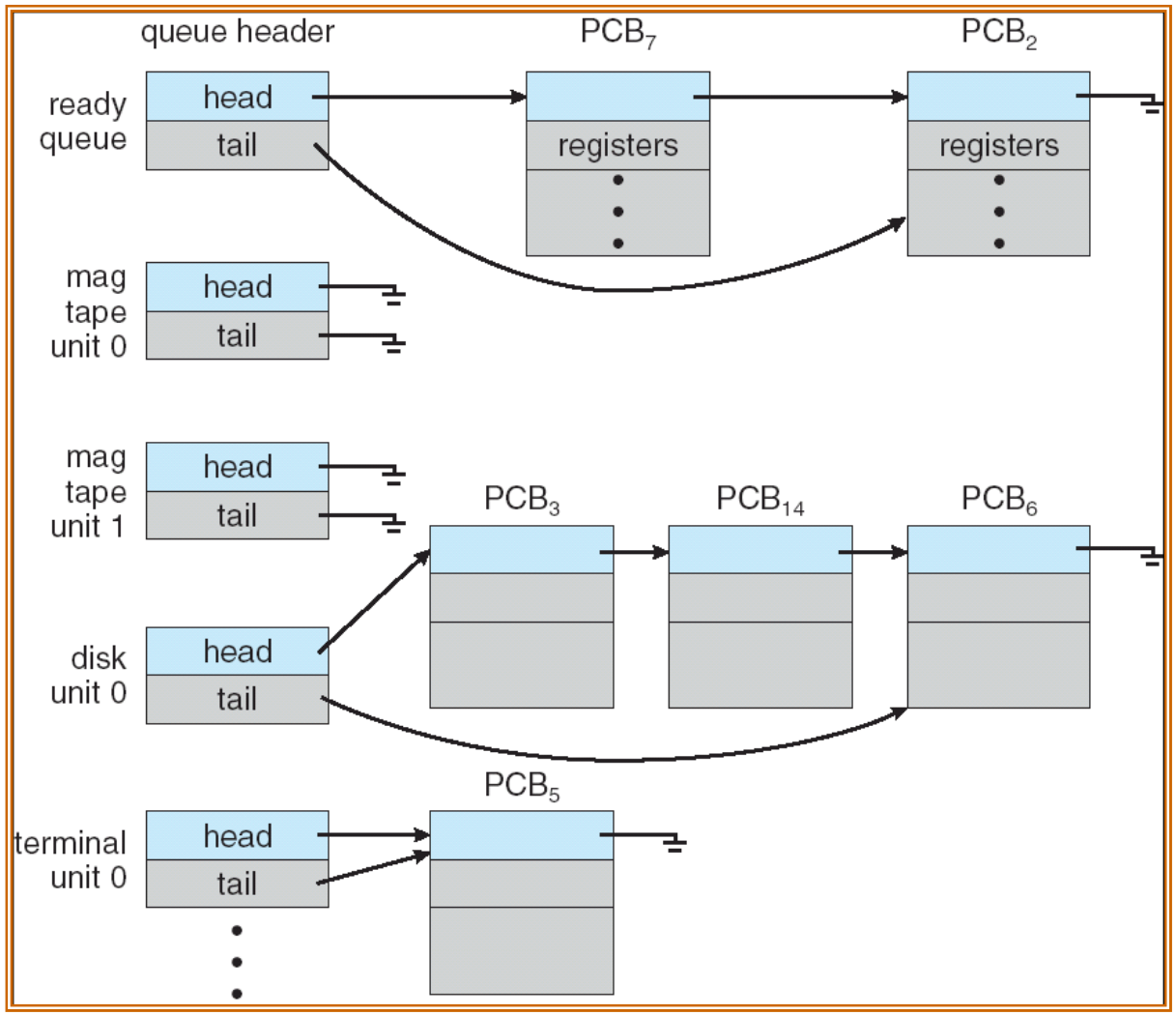
Troca de CPU entre Processos



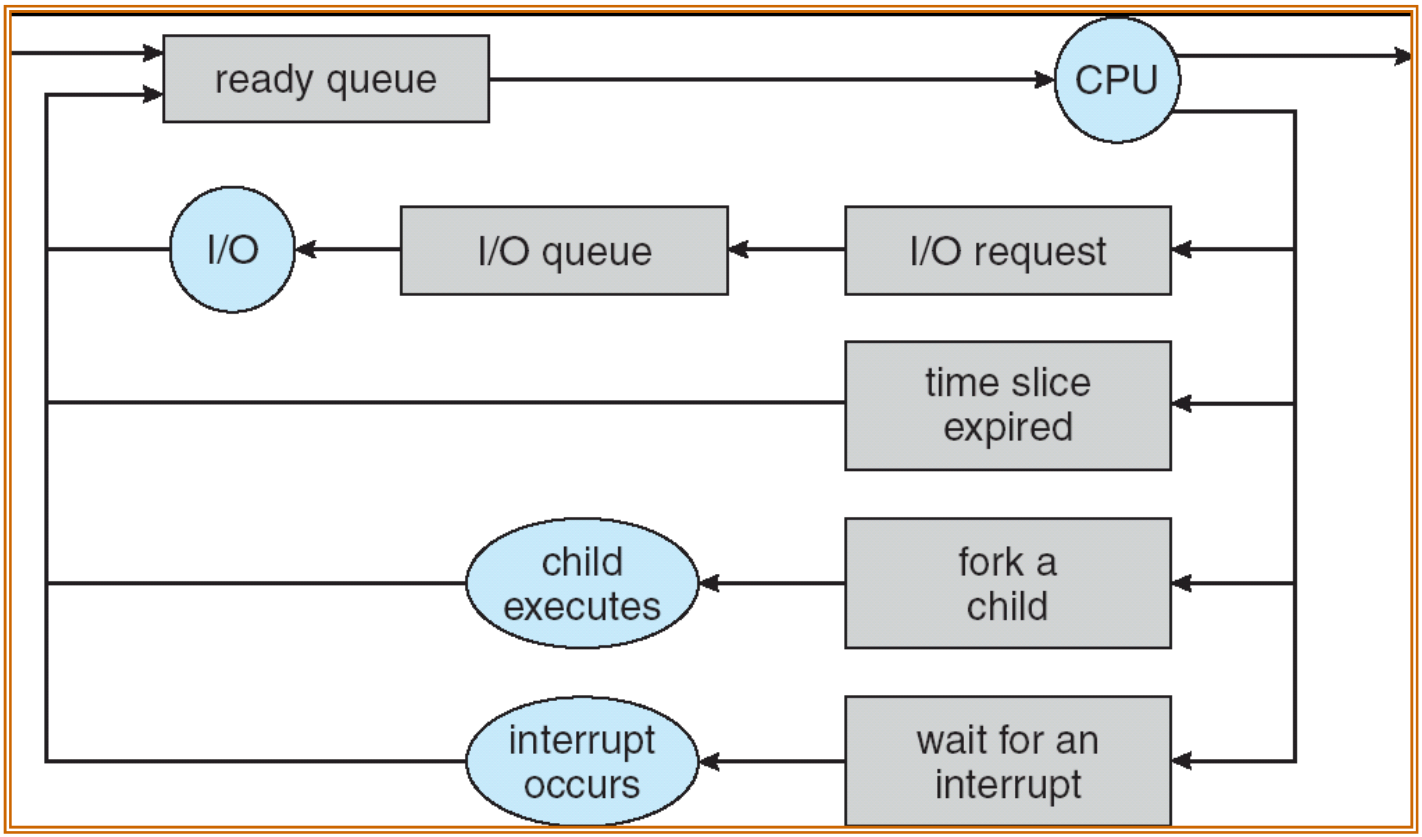
Filas de Escalonamento de Processos

- **Fila de *Job*** – conjunto de todos os processos no sistema.
- **Fila de Processos prontos** (*Ready queue*) – conjunto de todos os processos residentes na memória principal, prontos e esperando para executar.
- **Fila de dispositivos** – conjunto dos processos esperando por um dispositivo de E/S.
- Migração de processos entre as várias filas.

Fila de Processos Pronto e Várias Filas de E/S



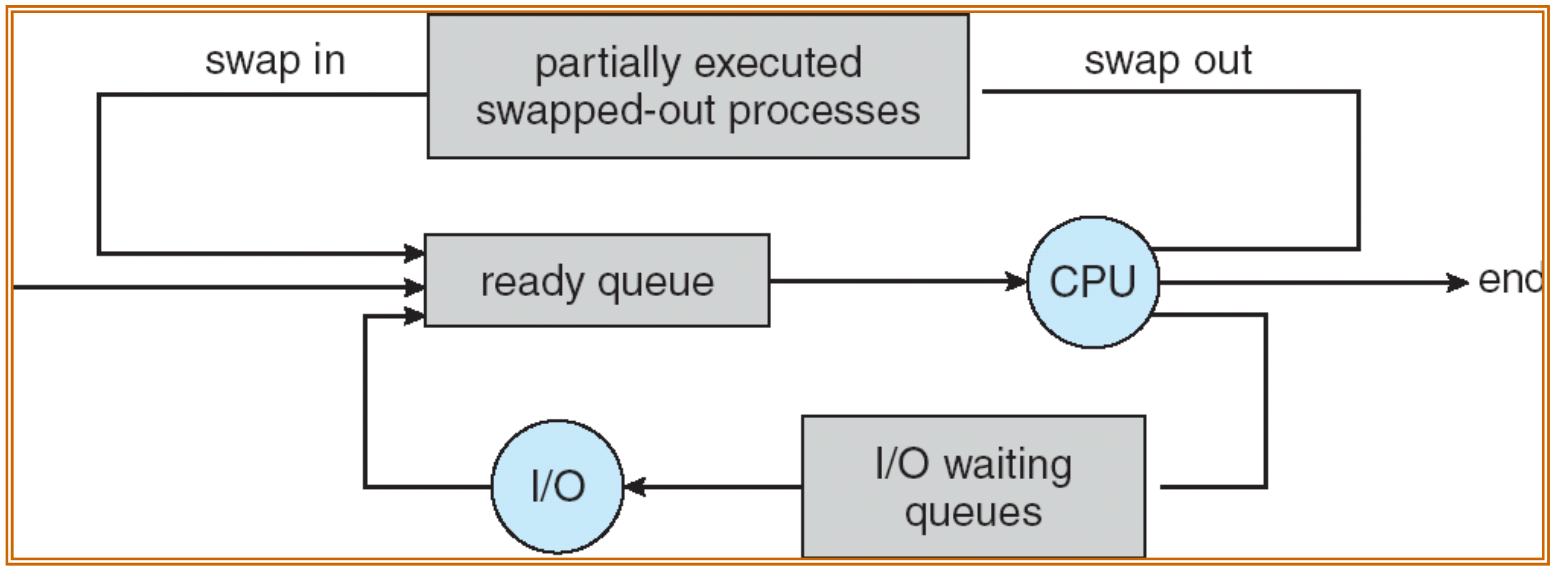
Representação de Escalonamento de Processos



Escalonadores

- **Escalonador de Longo Prazo** (ou escalonador de *Jobs*) – seleciona quais processos devem ser trazidos para a fila de processos prontos.
- **Escalonador de Curto Prazo** (ou escalonador da CPU) – seleciona qual processo deve ser executados a seguir e aloca CPU para ele.

Inclusão do Escalonador Intermediário



Escalonadores (Cont.)

- Escalonador de curto prazo é invocado muito freqüentemente (milisegundos) ⇒ (deve ser rápido).
- Escalonador de longo prazo é invocada muito infreqüentemente (segundos, minutos) ⇒ (pode ser lento).
- O escalonador de longo prazo controla o *grau de multiprogramação.*
- Processos podem ser descritos como:
  - **Processos com E/S predominante** (I/O-*bound process)* – gasta mais tempo realizando E/S do que computando, muitos ciclos curtos de CPU.
  - **Processos com uso de CPU predominante** (CPU-*bound process)* – gasta mais tempo realizando computações; poucos ciclos longos de CPU.

Troca de Contexto

- Quando CPU alterna para outro processo, o sistema deve salvar o estado do processo deixando o processador e carregar o estado anteriormente salvo do processo novo via **troca de contexto**.
- Contexto de um processo é representado na PCB
- Tempo de troca de contexto é sobrecarga no sistema; o sistema não realiza trabalho útil durante a troca de contexto.
- Tempo de Troca de Contexto é dependente de suporte em hardware.

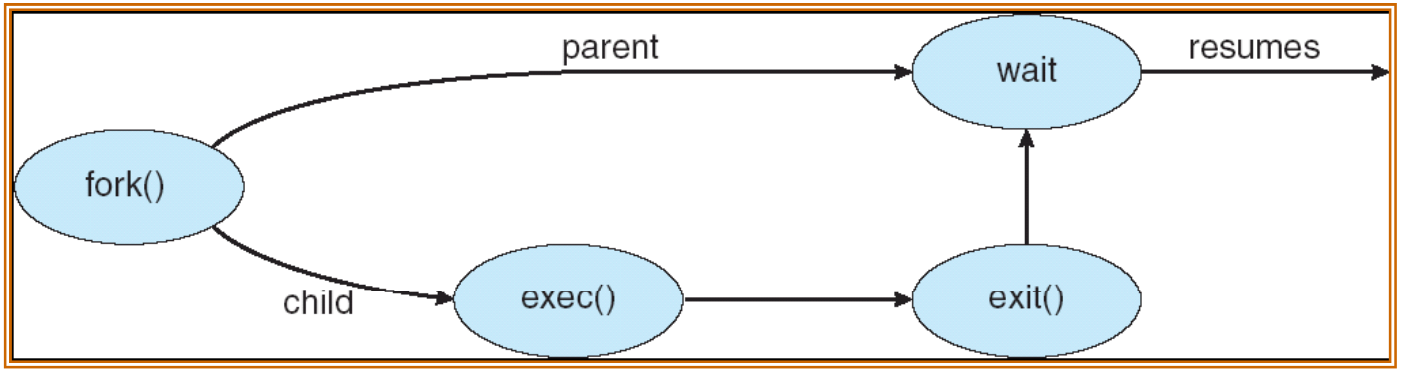
Criação de Processos

- Processo pai cria processo filho, o qual, por sua vez, pode criar outros processos, formando uma árvore de processos.
- Geralmente, processos são identificados e gerenciados via um **Identificador de Processos** (*Process IDentifier* - ***PID***)
- Compartilhamento de Recursos
  - Pai e filho compartilham todos os recursos.
  - Filho compartilha um subconjunto dos recursos do pai.
  - Pai e filho não compartilham recursos.
- Execução
  - Pai e filho executam concorrentemente.
  - Pai espera até filho terminar.

Criação de Processos (Cont.)

- Espaço de endereçamento
  - Filho duplica espaço do pai.
  - Filho tem um programa carregado no seu espaço.
- Exemplos no UNIX
  - Chamada de sistemas ***fork*** cria um novo processo.
  - Chamada de sistemas ***exec*** é usada após o ***fork*** para sobrescrever o espaço de memória do processo com um novo programa.

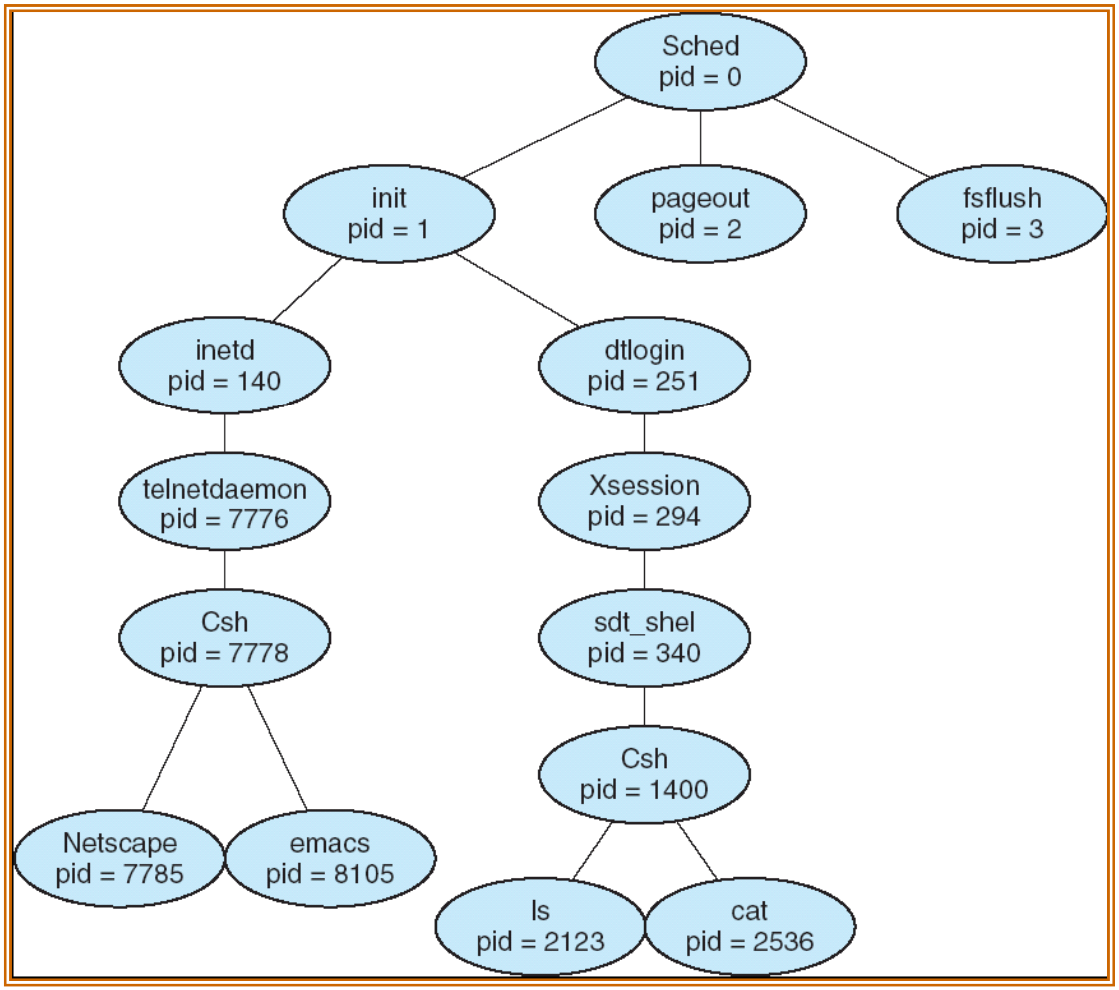
Criação de Processos (Cont.)



Programa em C Criando Processos Separados

```
int main()
{
Pid_t  pid;
      /* cria outro processo */
      pid = fork();
      if (pid < 0) { /* ocorrência de erro*/
                fprintf(stderr, “Criação Falhou");
                exit(-1);
      }
      else if (pid == 0) { /* processo filho*/
                execlp("/bin/ls", "ls", NULL);
      }
      else { /* processo pai */
                /* pai irá esperar o filho completar execução */
                wait (NULL);
                printf (“Filho Completou Execução");
                exit(0);
      }
}
```

Uma Árvore de Processos em um Sistema Solaris



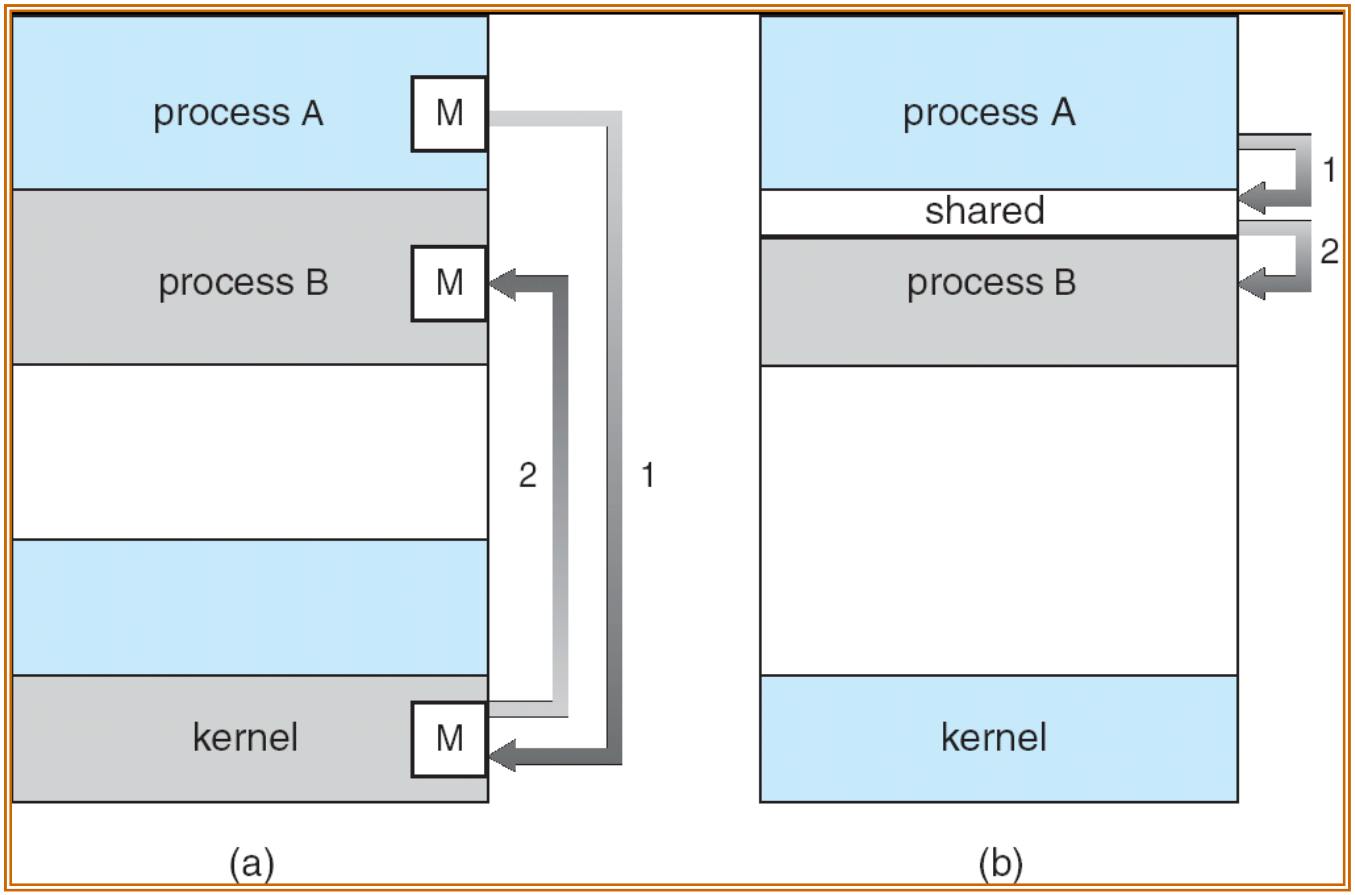
Terminação de Processos

- Processo executa última declaração e pede ao sistema operacional para decidir (***exit***).
  - Dados de saída passam do filho para o pai (via ***wait***).
  - Recursos do processo são desalocados pelo sistema operacional.
- Pai pode terminar a execução do processo filho (***abort***).
  - Filho se excedeu alocando recursos.
  - Tarefa delegada ao filho não é mais necessária.
  - Pai está terminando.
    - Sistema operacional não permite que um filho continue sua execução se seu pai terminou.
    - Todos os filhos terminam - Terminação em cascata.

Comunicação entre Processos (IPC)

- Processos em um sistema podem ser **Independentes** ou **Cooperantes**
- Processos **Independentes** não podem afetar ou ser afetados pela execução de outro processo.
- Processos **Cooperantes** podem afetar ou ser afetados pela execução de outro processo
- Razões para cooperação entre processos:
  - Compartilhamento de Informações
  - Aumento na velocidade da computação
  - Modularidade
  - Conveniência
- Processos cooperantes precisam de **Comunicação entre Processos** (**IPC** – *interprocess communication*)
- Dois modelos de IPC: memória compartilhada e troca de mensagens

Modelos de Comunicações



Problema do Produtor-Consumidor

- Paradigma para processos cooperantes, processo *produtor* produz informação que é consumida por um processo *consumidor*.
  - Buffer de tamanho ilimitado *(unbounded-buffer)* não coloca limite prático no tamanho do buffer.
  - Buffer de tamanho fixo (*bounded-buffer*) assume que existe um tamanho fixo do buffer.

Solução Buffer Tamanho Fixo - Memória Compartilhada

- Dados Compartilhados

```
#define BUFFER_SIZE 10
Typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solução está correta, mas somente pode usar BUFFER_SIZE-1 elementos

Buffer Tamanho Fixo – Produtor

```
while (true) {
   /* Produz um item */

   while (((in = (in + 1) % BUFFER SIZE count)  == out)
         ;   /* não faz nada – sem buffers livres*/
    /* Insere um item no buffer */
    buffer[in] = item;
    in = (in + 1) % BUFFER SIZE;
}
```

Buffer Tamanho Fixo – Consumidor

```
while (true) {
    while (in == out)
        ; /* não faz nada -- nada para consumir */
    /* Remove um item do buffer */
    item = buffer[out];
    out = (out + 1) % BUFFER SIZE;
    return item;

    /* Consome um item */
}
```

Comunicação entre Processos – Troca de mensagens

- Mecanismo para processos se comunicarem e sincronizarem suas ações.
- Sistema de mensagens – processos se comunicam uns com os outros sem utilização de variáveis compartilhadas.
- Suporte a IPC (*InterProcess Communication*) provê duas operações uma para envio outra para recebimento:
  - **send**(*mensagem*) – tamanho da mensagem fixo ou variável
  - **receive**(*mensagem*)
- Se *P* e *Q* querem se comunicar, eles necessitam:
  - Estabelecer um *link de comunicação* entre eles
  - Trocar mensagens via send/receive
- Implementação de links de comunicação
  - Físico (ex. Memória compartilha, barramento de hardware)
  - Lógico (ex. Propriedades lógicas)

Questões de Implementação

- Como são estabelecidas as ligações?
- Pode um link estar associado com mais de dois processos?
- Quantos links podem existir entre cada par de processos comunicantes?
- Qual a capacidade de um link?
- O tamanho da mensagem utilizado pelo link é fixo ou variável?
- O link é unidirecional ou bidirecional?

Comunicação Direta

- Processos devem nomear o outro explicitamente:
  - **send** (*P, mensagem*) – envia uma mensagem ao processo P
  - **receive**(*Q, mensagem*) – recebe uma mensagem do processo Q

- Propriedades dos links de comunicação
  - Links são estabelecidos automaticamente.
  - Um link é associado com exatamente um par de processos comunicantes.
  - Entre cada par de processos existe exatamente um link.
  - O link pode ser unidirecional, mas é usualmente bidirecional.

Comunicação Indireta

- Mensagens são dirigidas e recebidas de caixas postais – *mailboxes* (também chamadas de portas).
  - Cada *mailbox* possui uma única identificação.
  - Processos podem se comunicar somente se eles compartilham a *mailbox*.
- Propriedades do link de comunicação:
  - O link é estabelecido somente se os processos compartilham uma *mailbox* comum
  - Um link pode estar associado com muitos processos.
  - Cada par de processos pode compartilhar vários links de comunicação.
  - Link pode ser unidirecional ou bidirecional.

Comunicação Indireta (Cont.)

- Operações
  - Criar uma nova caixa postal
  - Enviar e receber mensagens através da caixa postal
  - Destruir uma caixa postal

- Primitivas são definidas como:

  send(*A, mensagem*) – envia uma mensagem para a caixa postal A

  receive(*A, mensagem*) – recebe uma mensagem da caixa postal A

Comunicação Indireta (Cont.)

- Compartilhamento de Caixa Postal
  - P_1, P_2, e P_3 compartilham caixa postal A.
  - P_1, envia; P_2 e P_3 recebem.
  - Quem recebe a mensagem?

- Soluções:
  - Permitir que um link esteja associado com no máximo dois processos.
  - Permitir somente a um processo de cada vez executar uma operação de recebimento.
  - Permitir ao sistema selecionar arbitrariamente por um receptor. Remetente é notificado de quem foi o receptor.

Sincronização

- Troca de Mensagens pode ser bloqueante ou não-bloqueante
- **Bloqueante** é considerado **síncrono**
  - **send Bloqueante** inibe o remetente até que a mensagem seja recebida
  - **receive Bloqueante** inibe o receptor até uma mensagem estar disponível
- **Não-Bloqueante** é considerado **assíncrono**
  - **send Não-bloqueante** o remetente envia a mensagem e continua executando
  - **receive Não-bloqueante** o receptor obtém uma mensagem válida ou null

Bufferização

- Fila de mensagens associada ao link; implementada em uma dentre três formas.
  1. Capacidade Zero – 0 mensagens
     Remetente deve esperar pelo receptor (*rendezvous*).
  2. Capacidade Limitada – tamanho finito de n mensagens
     Remetente deve aguardar se link está cheio.
  3. Capacidade Ilimitada – tamanho infinito
     Remetente nunca espera.

Exemplos de Sistemas IPC - POSIX

- Memória Compartilhada no POSIX
  - Processo cria primeiro um segmento de memória compartilhado

```
segment id = shmget(IPC PRIVATE, size, S IRUSR | S
  IWUSR);
```

  - Processo que deseja acesso a essa memória compartilhada deve se anexar a ela

```
shared memory = (char *) shmat(id, NULL, 0);
```

  - Agora o processo pode escrever na memória compartilhada

```
sprintf(shared memory, "Writing to shared
  memory");
```

  - Quando terminar, um processo pode desanexar a memória compartilhada do seu espaço de armazenamento

```
shmdt(shared memory);
```

Exemplos de Sistemas IPC - Mach

- Comunicação no Mach é baseado em mensagens
  - Até mesmo chamada de sistemas são mensagens
  - Cada tarefa obtém duas *mailboxes* na criação - *Kernel* e *Notify*
  - Somente três chamadas de sistemas são necessárias para transferência de mensagens

```
msg_send(), msg_receive(), msg_rpc()
```

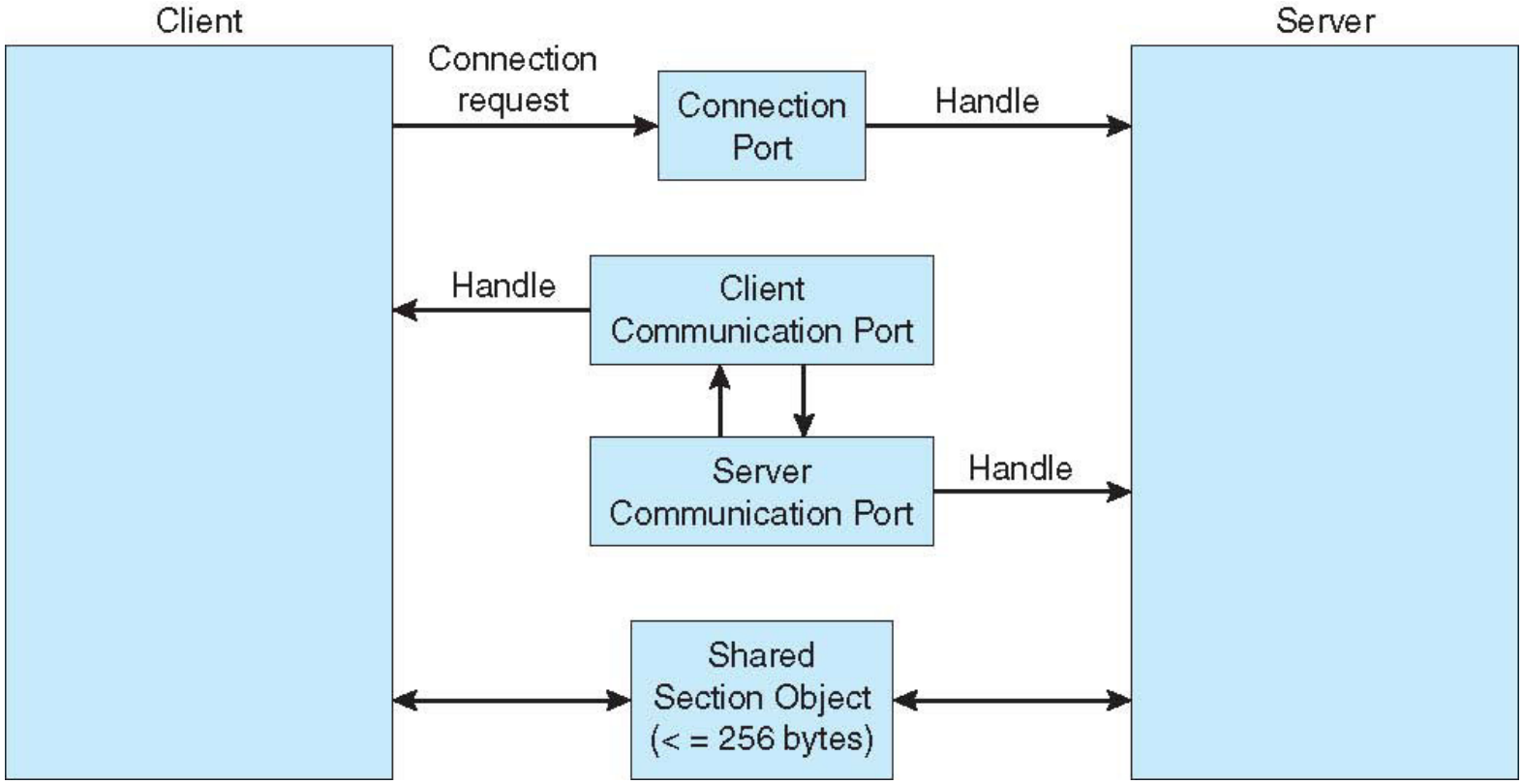
  - *Mailboxes* necessárias para comunicação, criadas via

```
port_allocate()
```

Exemplos de Sistemas IPC – Windows XP

- Recurso de troca de mensagens é chamado de ***local procedure call (LPC)***
  - Só funciona entre processos no memso sistema
  - Usa portas (como *mailboxes*) para estabelecer e manter canais de comunicação
  - Comunicação funciona da seguinte forma:
    - O cliente abre um manipulador para o objeto porta de conexão do subsistema.
    - O cliente envia uma solicitação de conexão.
    - O servidor cria duas portas de comunicação privadas e retorna o manipulador de uma delas para o cliente.
    - O cliente e o servidor usam o manipulador da porta correspondente para enviar mensagens ou retornos de chamadas e ouvir respostas.

Local Procedure Calls no Windows XP



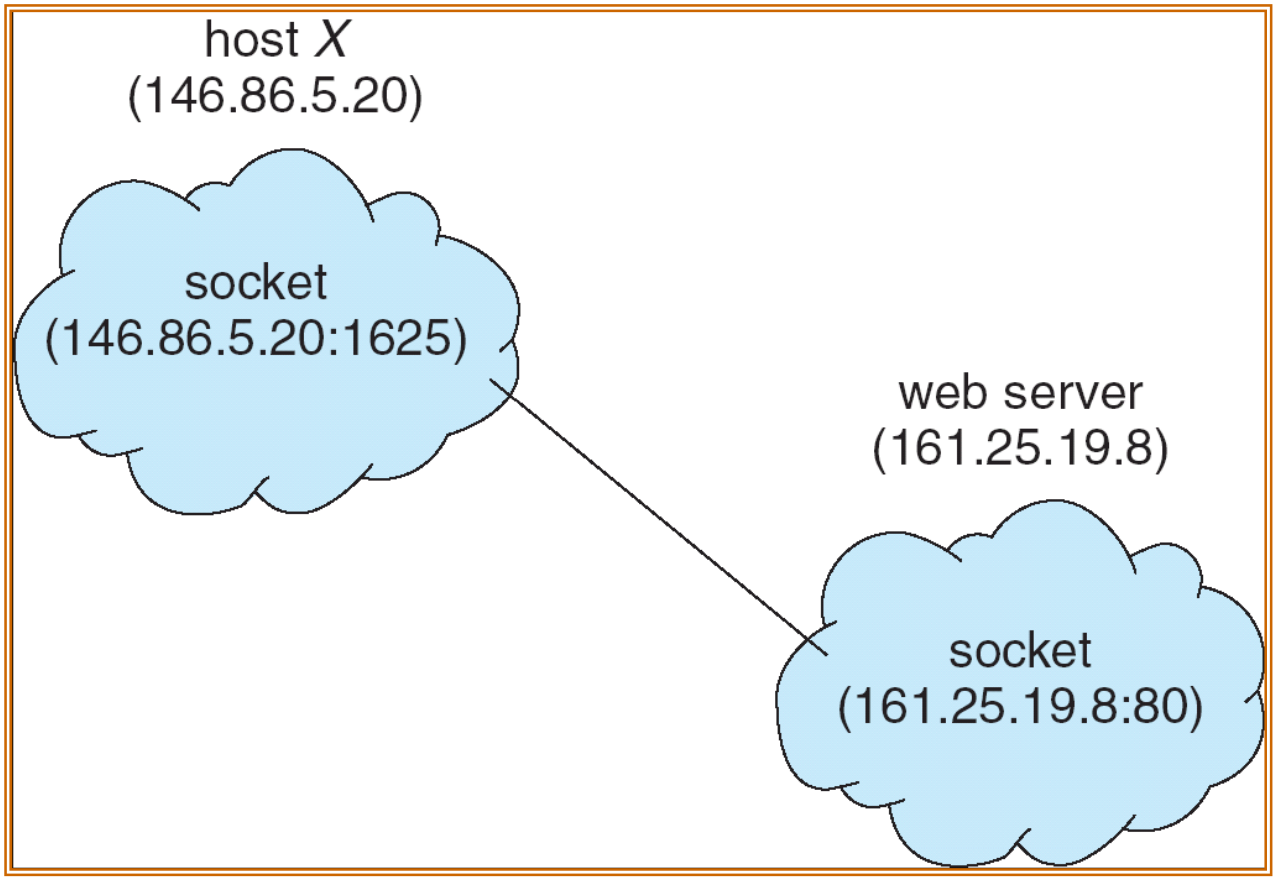
Comunicação Cliente-Servidor

- Sockets
- Pipes
- Chamada a Procedimento Remoto (RPC)
- Invocação Remota de Método (RMI em Java)

Sockets

- Um socket é definido como um ponto final de comunicação
- Concatenação de um endereço IP e porta
- O socket **161.25.19.8:1625** refere a porta **1625** na máquina **161.25.19.8**
- Comunicação ocorre entre um par de sockets

Comunicação com Socket



Pipes

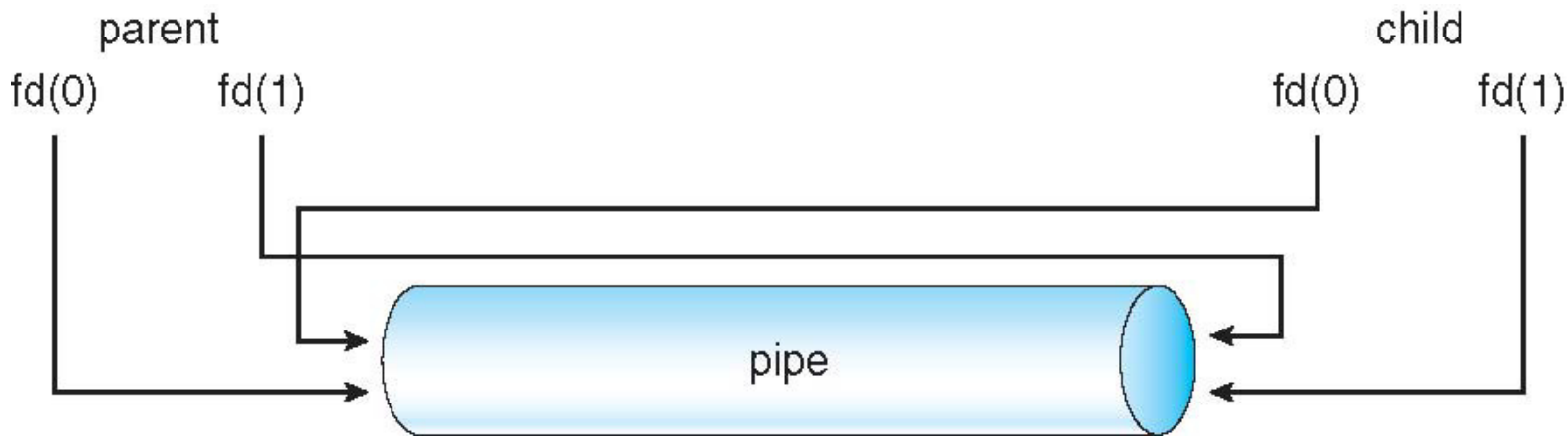
- Agem como canalizações permitindo a comunicação entre dois processos

- Questões
  - A comunicação é unidirecional ou bi-direcional?
  - No caso da comunicação de duas vias, ela é half ou full-duplex?
  - Existe uma relação (ex. Pai-filho) entre os processos comunicantes?
  - É possível usar pipes em uma rede?

Pipes Comuns

- **Pipes comuns** permitem a comunicação no estilo produtor-consumidor
- Produtor escreve em um extremo (o extremo de escrita do pipe)
- Consumidor lê do outro extremo (o extremo de leitura do pipe)
- Pipes comuns são unidrecionais
- Necessitam de relação pai-filho entre os processos comunicantes

Pipes Comuns



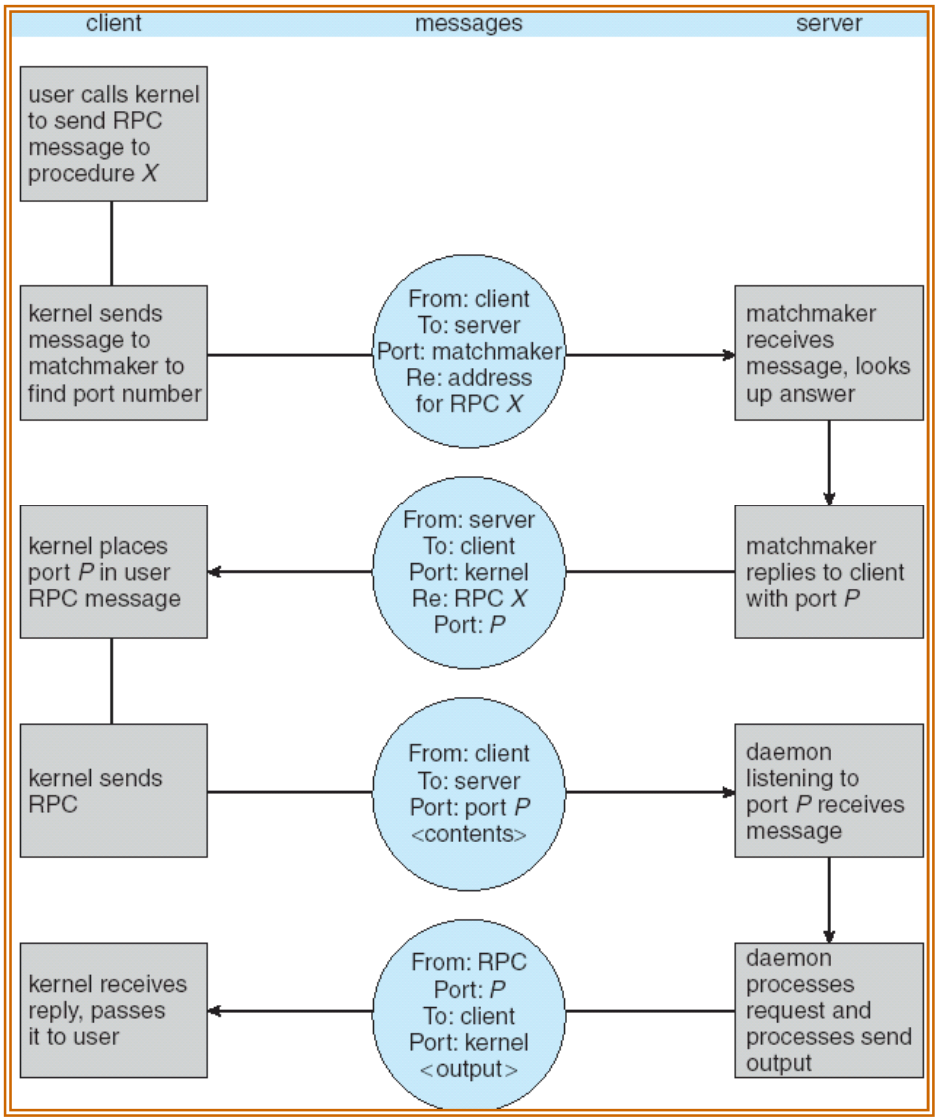
Pipes Nomeados

- Pipes Nomeados são mais poderosos que pipes comuns
- Comunicação é bi-direcional
- Não é necessária relação pai-filho entre processos comunicantes
- Vários processos podem usar os pipes noemados para se comunicarem
- Fornecidos nos sistemas UNIX e Windows

Chamada a Procedimento Remoto

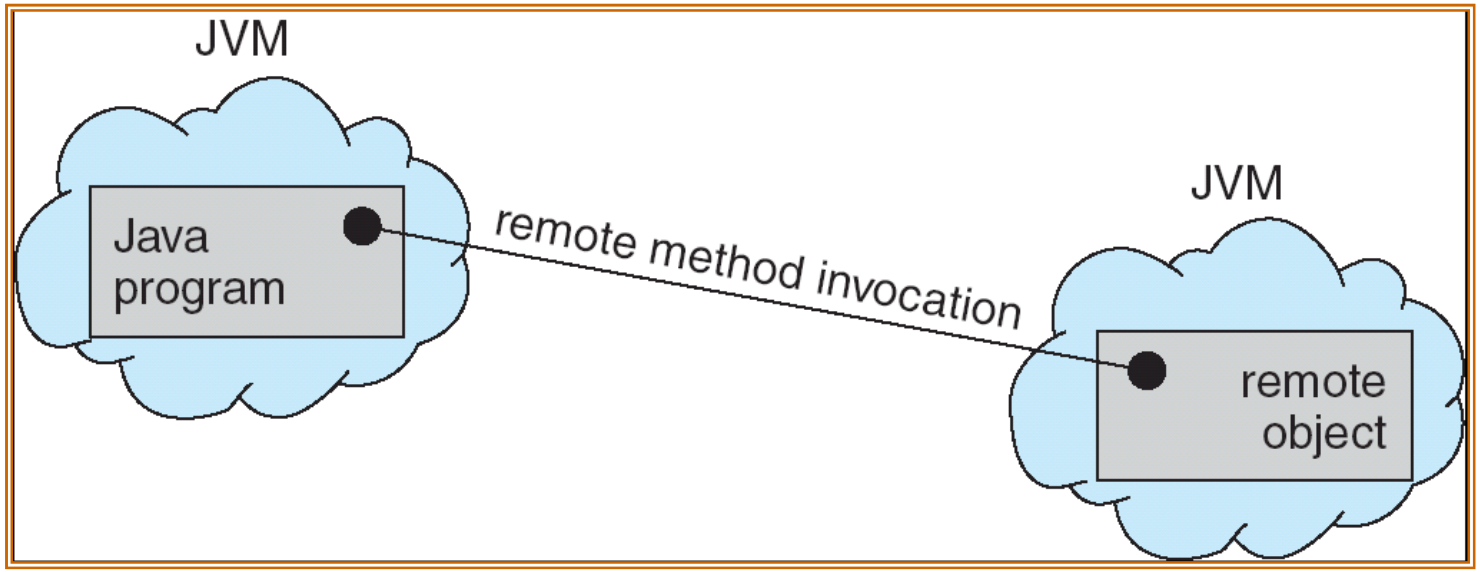
- Chamada a Procedimento Remoto ou *Remote procedure call* (RPC) abstrai chamadas de procedimentos entre processos executando nos sistemas em rede.
- **Stubs** – proxy no lado do cliente para o procedimento real no servidor.
- O stub no lado do cliente localiza o servidor e empacota (*marshall*) os parâmetros.
- O stub no lado do servidor recebe esta mensagem, desempacota os parâmetros e dispara a execução do procedimento no servidor.

Execução de RPC



Invocação Remota de Método

- Invocação Remota de Método ou *Remote Method Invocation* (RMI) é um mecanismo Java similar a RPC.
- RMI permite a um programa Java executando em uma máquina invocar um método em um objeto remoto.



Fim do Capítulo 3