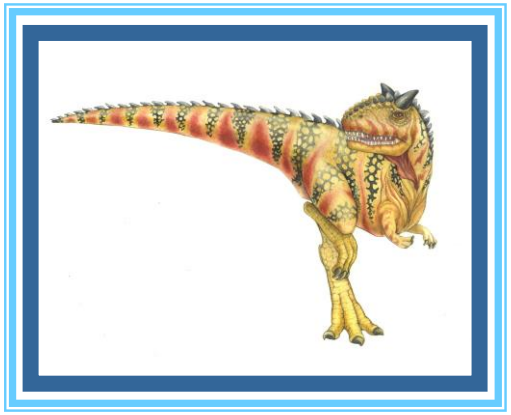
Capítulo 3: Processos



Capítulo 3: Processos

- Conceito de Processo
- Scheduling de Processos
- Operações sobre Processos
- Comunicação Interprocessos
- Exemplos de Sistemas IPC
- Comunicação em Sistemas Cliente-Servidor

Universidade de Brasília

Faculdade UnB Gama

Objetivos

- Introduzir a noção de processo — um programa em execução que forma a base de toda a computação.
- Descrever as diversas características dos processos, incluindo o scheduling, a criação e o encerramento.
- Explorar a comunicação entre processos com o uso da memória compartilhada e da transmissão de mensagens.
- Descrever a comunicação em sistemas cliente-servidor.

Conceito de Processo

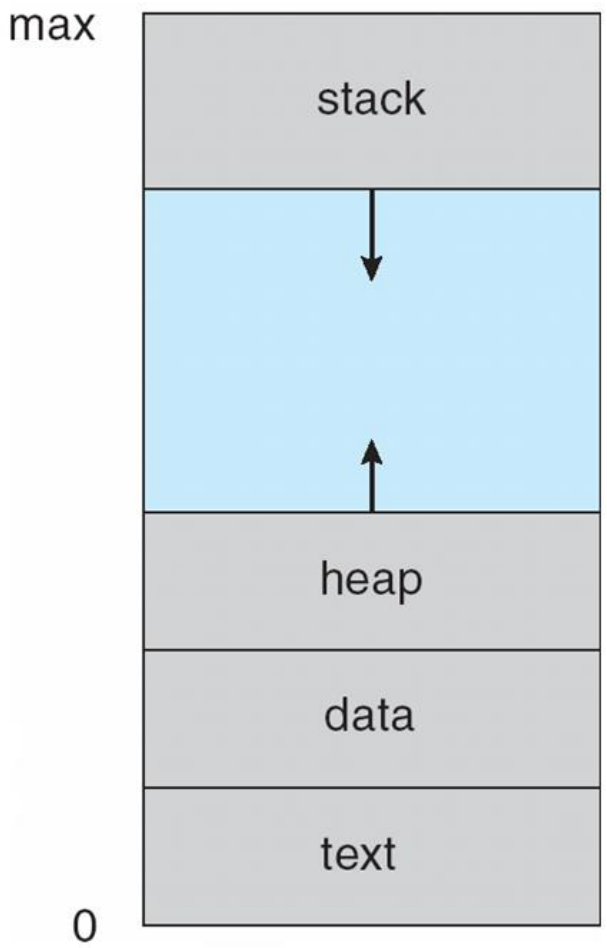
- Um sistema operacional executa uma variedade de programas :
  - Sistema batch – **jobs**
  - Sistema de tempo compartilhado – **user programs** ou **tasks**
- Os livros usam os termos ***job*** e ***processo*** de maneira quase intercambiável.
- **Processo** –um programa em execução ; a execução de processo deve progredir de forma sequencial.
- Múltiplas partes
  - O código do programa, também chamado de **seção de texto**
  - Atividade corrente incluindo **contador do programa**, registradores do processador
  - **Pilha** contém dados temporários
    - Parâmetros de funções, endereços de retorno e variáveis locais

Conceito de Processo (Cont.)

- **Seção de dados** contém variáveis globais
- **Heap** memória dinamicamente alocada durante o tempo de execução do processo

- Programa é uma entidade ***passiva*** armazendo em disco (**executable file**), processo é uma entidade ***ativa***
  - Um programa torna-se um processo quando um arquivo executável é carregado na memória
- Clicar duas vezes em um ícone representando o arquivo executável, dar entrada no nome do arquivo executável na linha de comando
- Um programa pode ser vários processos
  - Considere múltiplos usuários executando o mesmo programa

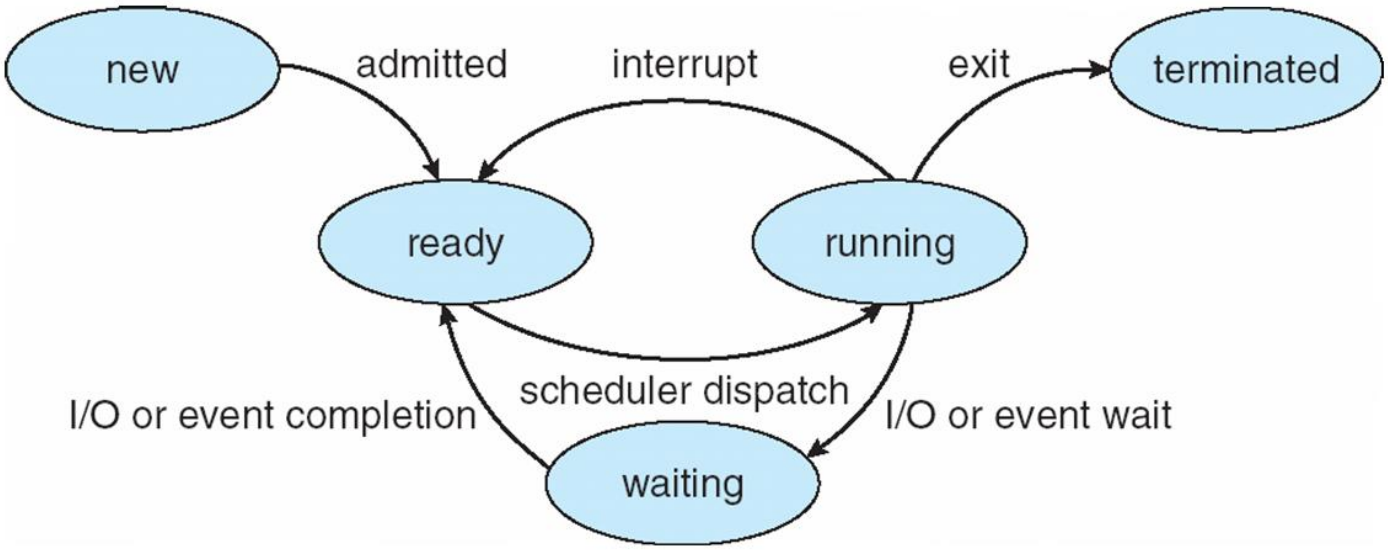
Processo na Memória



Process State

- Quando um processo é executado, ele muda de **estado**
  - **Novo:** O processo está sendo criado
  - **Em execução:** Instruções estão sendo executadas
  - **Em espera:** O processo está esperando que algum evento ocorra
  - **Pronto:** O processo está esperando que seja atribuído a um processador
  - **Concluído:** O processo terminou sua execução

Diagrama de Estado do Processo



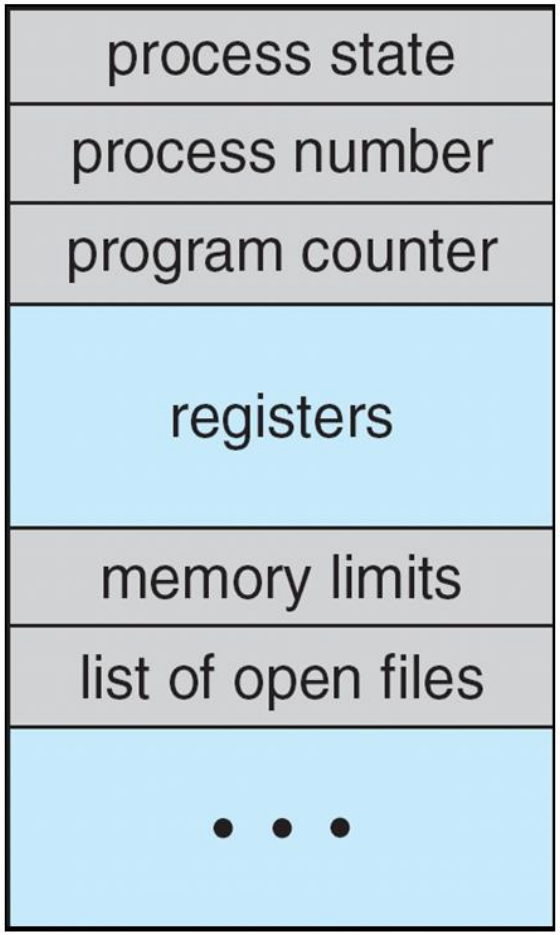
Bloco de Controle de Processo (BCP)

Informação associada com cada processo

(também chamado de **bloco de controle de tarefa)**

Estado de processo – em execução, em espera, etc

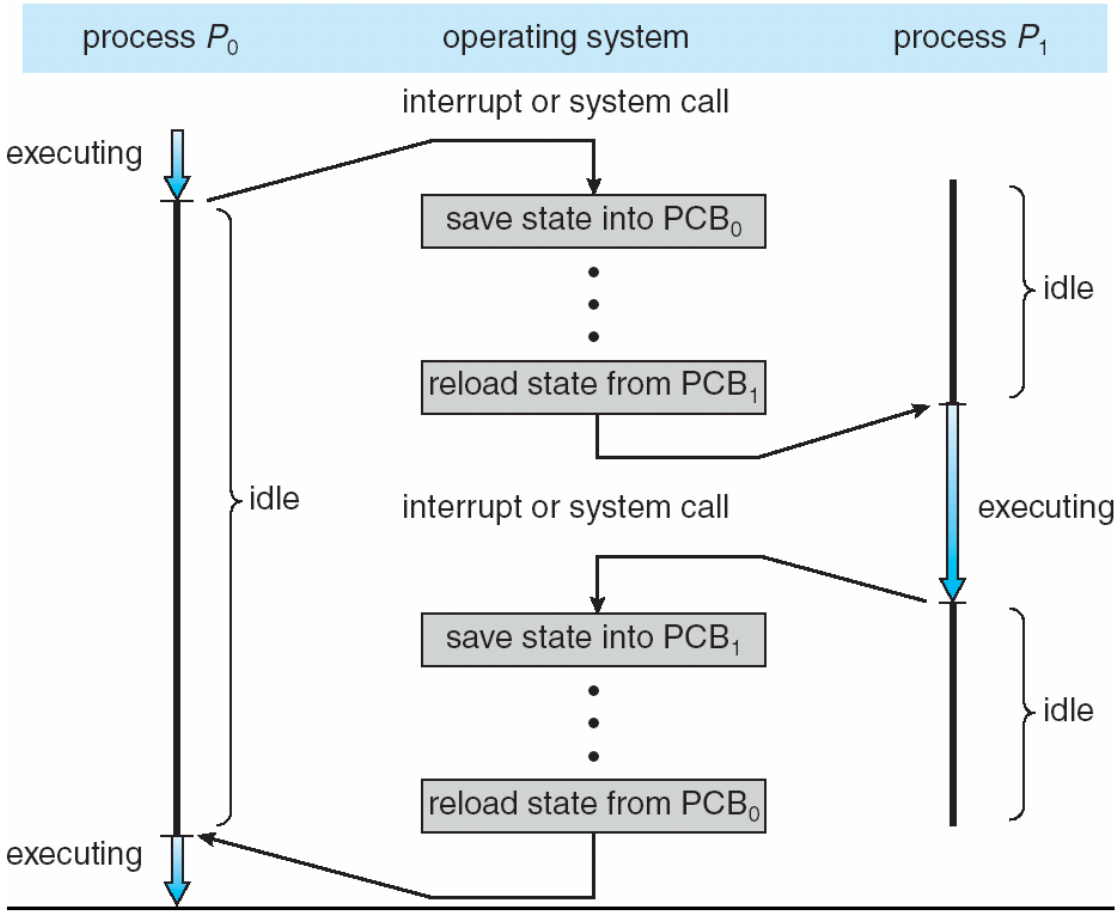
- Contador do programa – endereço da próxima instrução a ser executada
- Registradores da CPU – incluem todos os registradores relacionados a processos
- Informações de scheduling da CPU- prioridades, ponteiros para filas de scheduling
- Informações de gerenciamento da memória- memória alocada para o processo
- Informações de contabilização – CPU usado, montante de tempo real, limites de tempo
- Informações de status de I/O – lista de dispositivos de I/O alocados ao processo , lista de arquivos abertos



Universidade de Brasília

Faculdade UnB Gama

Alternância da CPU de um processo para outro



Threads

- O modelo de processo discutido até agora sugere que um processo é um programa que executa apenas um thread
- Considere ter múltiplos contadores de programas por processo
  - Múltiplos locais podem executar imediatamente
    - Múltiplos threads de controle-> **threads**
- Deve haver armazenamento para os detalhes de thread, contadores múltiplos de programas em PCB
- Veja o próximo capítulo

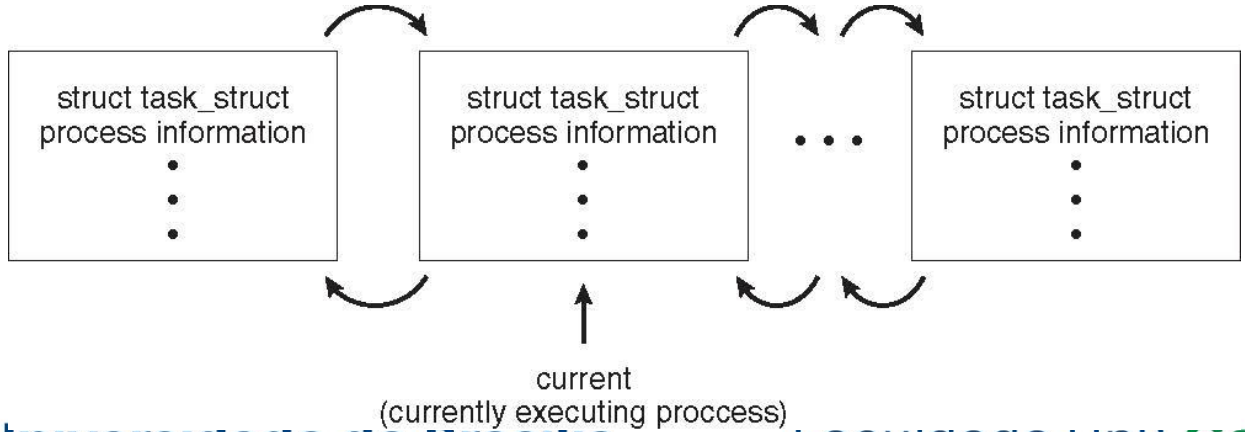
Universidade de Brasília

Faculdade UnB Gama

REPRESENTAÇÃO DE PROCESSOS NOLINUX

Represented by the C structure `task_struct`

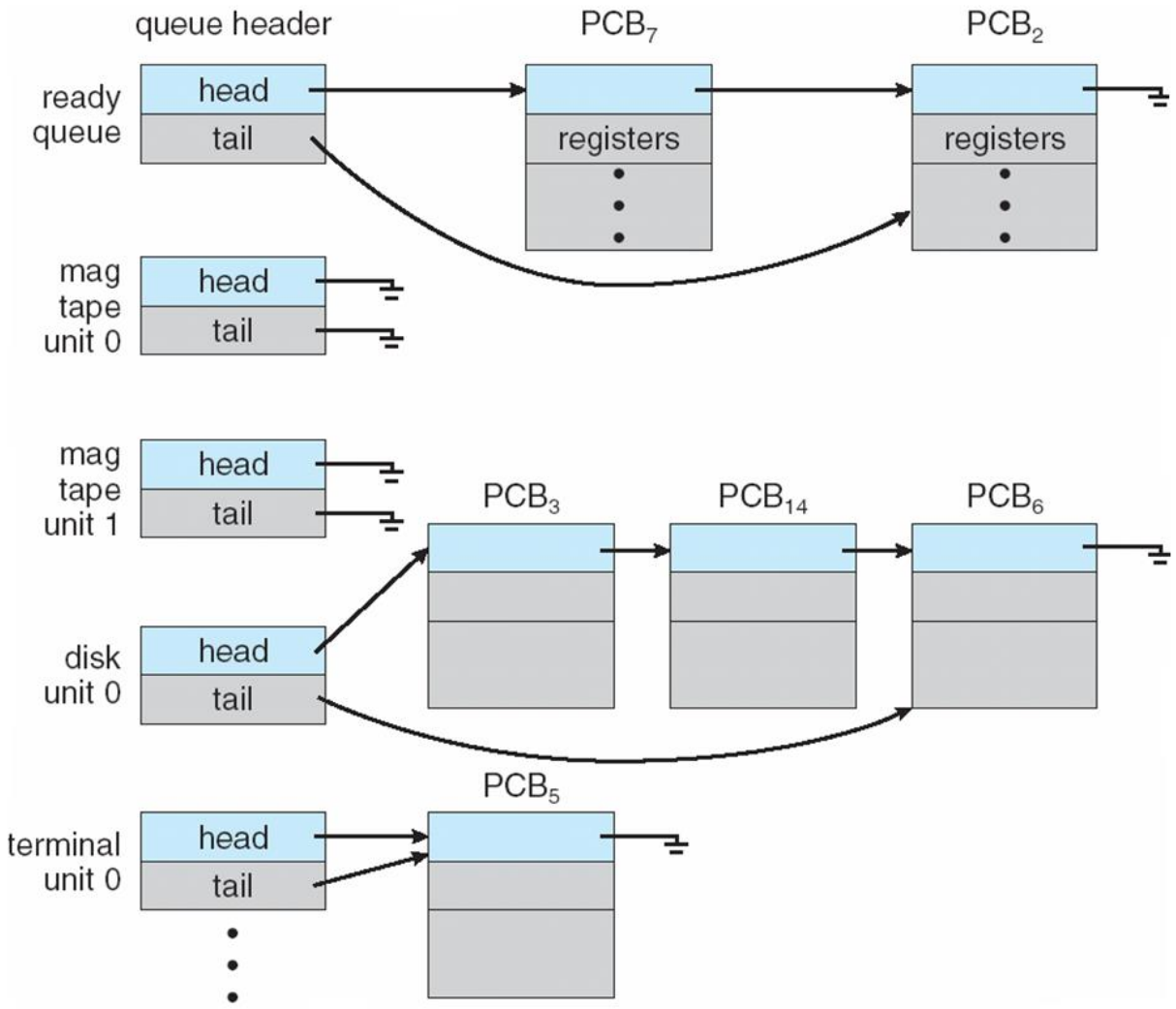
```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```



Scheduling de Processos

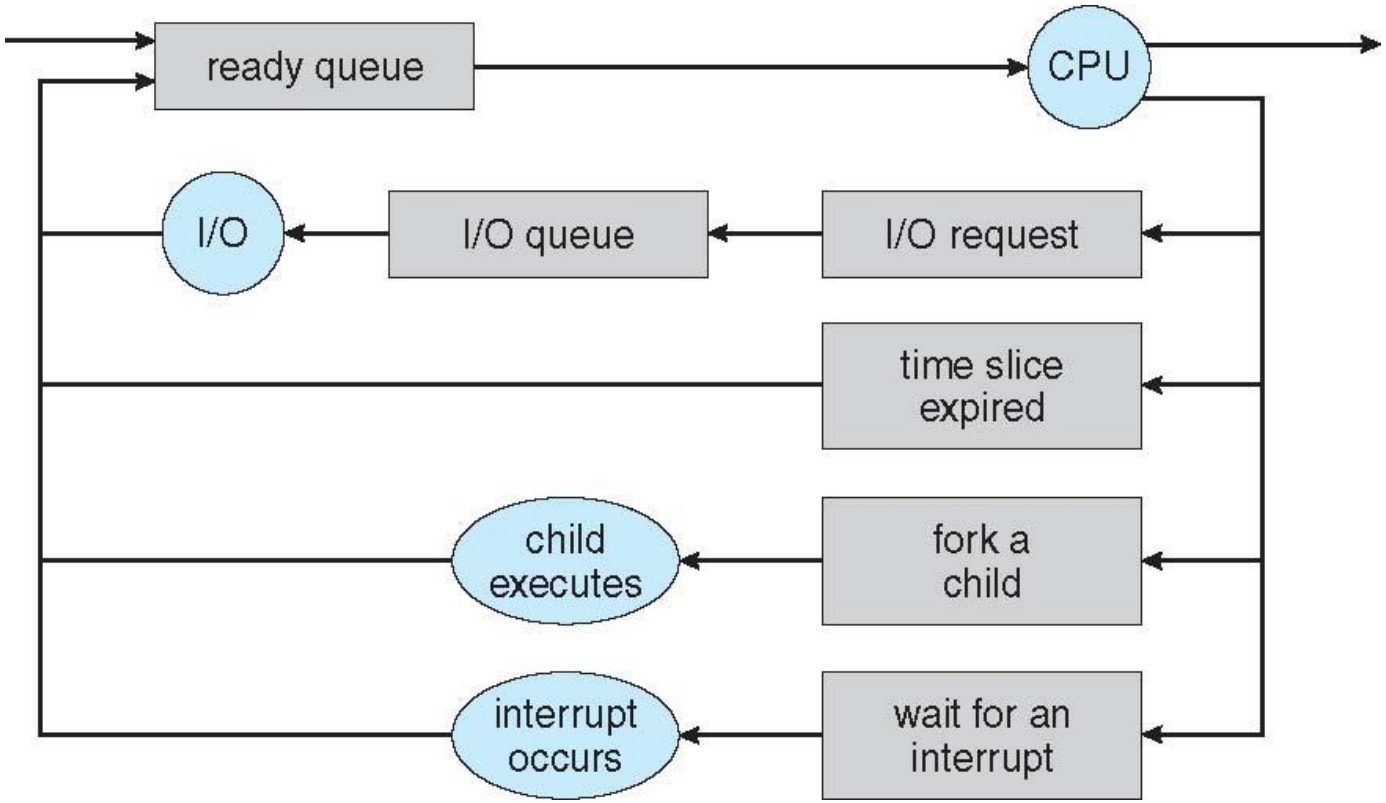
- Maximiza o uso do CPU, altenar a CPU rapidamente entre os processos para o compartilhamento de tempo
- **Scheduler de processos** seleciona um processo disponível para execução na CPU
- Mantém as **scheduling queues** de processos
  - **Job queue** – conjunto de todos os processos no sistema
  - **Ready queue** –processos que estão residindo na memória principal e estão prontos e esperando execução
  - **Device queues** – lista de processos em espera por um dispositivo de I/O específico
  - Os processos migram entre várias filas

Fila de prontos e várias filas de dispositivo de I/O.



Representação do scheduling de processos em diagrama de enfileira

- **Queueing diagram** representa filas, recursos, fluxos



Schedulers

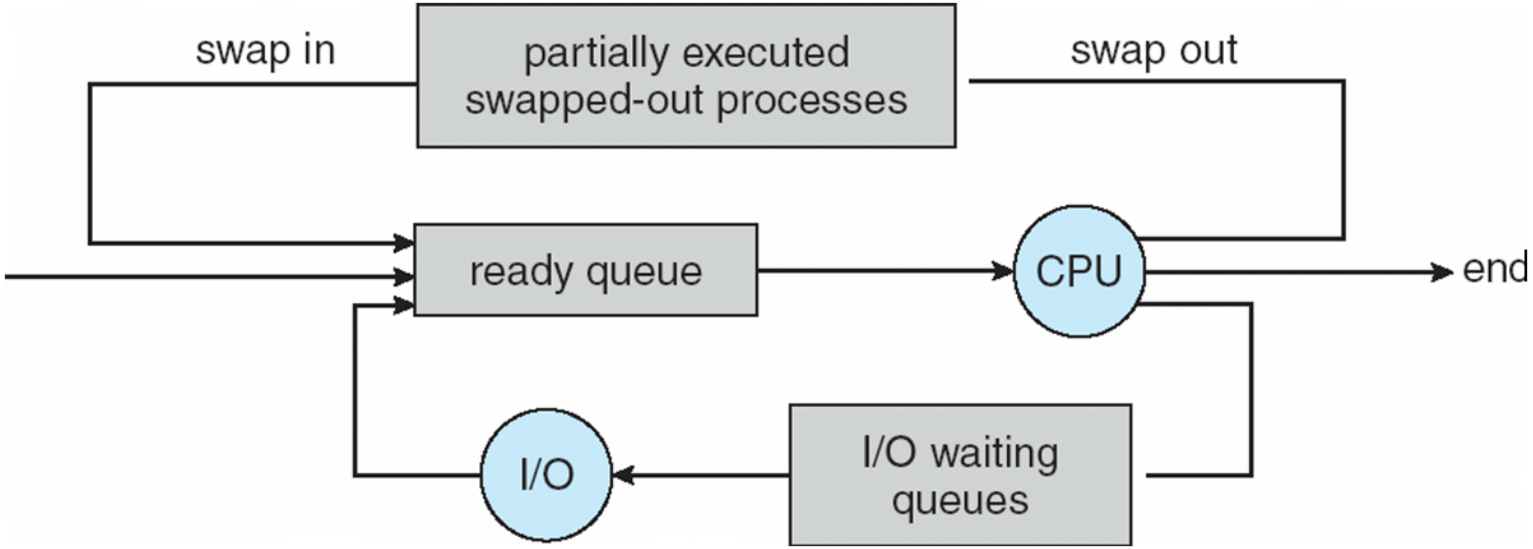
- **Short-term scheduler** (ou **CPU scheduler**) seleciona entre os processos que estão prontos para execução e aloca a CPU a um deles.
  - Às vezes é o único scheduler em um sistema
  - O scheduler de curto prazo é executado frequentemente (millisegundos) ⇒ (tem que ser rápido)
- **Long-term scheduler** (ou **job scheduler**) – seleciona processos nesse pool e os carrega na memória para execução
  - O scheduler de longo prazo é executado com muito menos frequência (segundos, minutos) ⇒ (pode ser lento)
  - O scheduler de longo prazo controla o **degree of multiprogramming**
- Os processos podem ser descritos de uma das duas formas:
  - **I/O-bound process** – gasta mais do seu tempo fazendo I/O do que executando computação , muitos surtos de CPU curtos
  - **CPU-bound process** – gasta mais tempo executando computação; poucos surtos de CPU longos
- O scheduler de longo prazo se esforça para selececionar um bom ***mix de processos***

Universidade de Brasília

Faculdade UnB Gama

Inclusão do scheduling de médio prazo no diagrama de enfileiramento

- **Medium-term scheduler** pode ser inlcluído se o grau de multiprogramação precisar ser reduzido
  - Remove o processo da memory, aramazena no disk, reintroduz o processo na memória para execução: **swapping**



Multitarefa em Sistemas Móveis

- Alguns sistemas móveis (por exemplo: primeiras versões do iOS) permitem apenas a execução de um processo, os outros são suspensos
- Devido a tempo de vida da bateria, e uso da memória a Apple fornece uma forma limitada de multitarefas ??????
  - Processo único (aplicação) de **foreground** controlado por meio da interface do usuário
  - Múltiplos processos de **background**– na memória, em execução, mas não ocupam a tela, e com limites
  - Os limites incluem tarefa única de tamanho finite, receiving notification of events, receber notificações sobre a ocorrência de um evento, tarefas de background de execução longa (como um reprodutor de áudio).
- Android executa a aplicação no foreground e background, com poucos limites
  - O processo de Background usa um **service** para executar tarefas
  - o serviço continuará a ser executado, mesmo se a aplicação de background for suspensa
  - Os serviços não têm uma interface de usuário e têm um footprint de memória pequeno

Mudança de Contexto

- Quando a CPU alterna para outro processo, o sistema deve **save the state** do processo anterior e ela restaura o **saved state** para o novo processo por meio de **context switch**
- **Context** de um processo representado no PCB
- O tempo gasto na mudança de contexto é overhead; o sistema não executa trabalho útil durante a permuta de processos
  - Quanto mais complexo for o Sitema operacional e o PCB ➔ maior será a mudança de contexto
- O tempo depende do suporte do hardware
  - Alguns hardwares fornecem vários conjuntos de registradores por CPU ➔ múltiplos contextos são executados (carregados) ao mesmo tempo

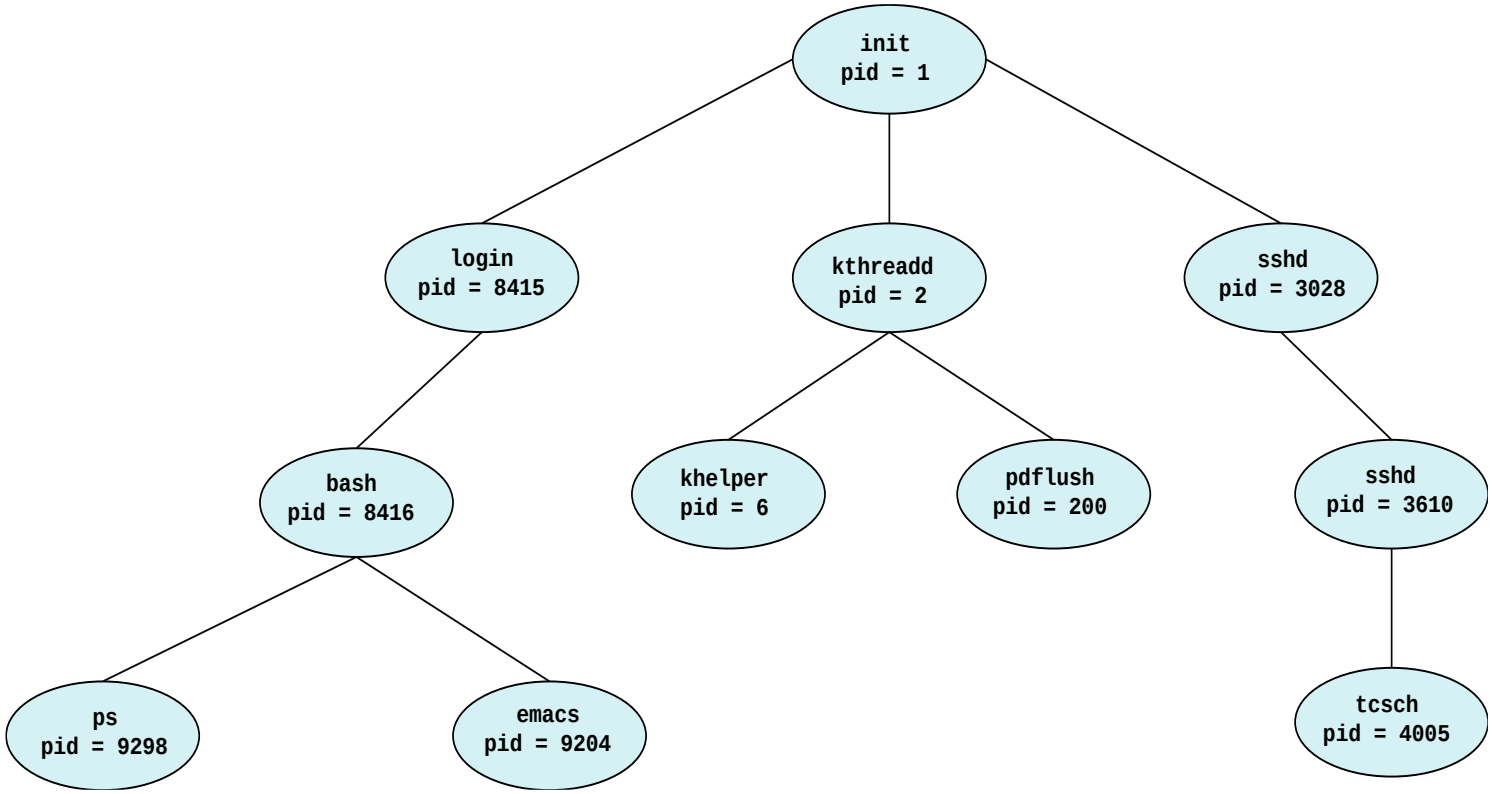
Operações sobre Processos

- O sistema tem que fornecer mecanismos para:
  - Criação de processos,
  - Encerramento de Processos ,
  - e outros que serão detalhados mais adiante

Criação de Processos

- Processo **Pai** cria processos **filhos**, os quais criam outros processos formando uma **árvore** de processos.
- Geralmente, os processso são identificados e gerenciados por meio de um identificador de processo - **process identifier** (**pid**)
- Opções de compartilhamento de recursos
  - Pai e filhos compartilham todos os recursos
  - Filhos compartiham um subgrupo dos recursos do pai
  - Pai e filhos não compartilham nenhum recurso
- Possibilidadades de execução
  - Pai e filhos são executados concorrentemente
  - Pai espera até que filhos sejam encerrados

Uma árvore de processos em um sistema Linux

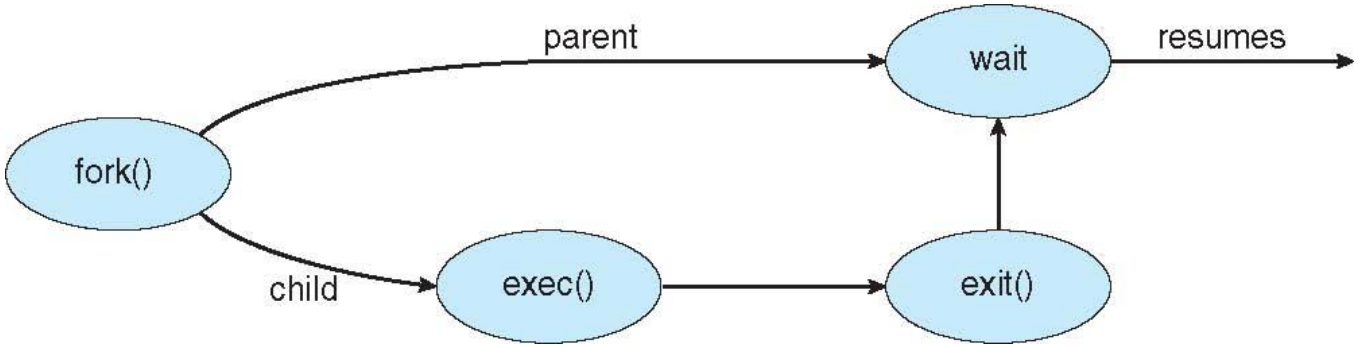


Universidade de Brasília

Faculdade UnB Gama

Criação de Processo (Cont.)

- Espaço de endereçamento
  - O processo-filho é uma duplicata do processo-pai
  - O processo-filho tem um novo programa carregado nele
- Exemplos de UNIX
  - **`fork()`** a chamda de sistema cria um novo processo
  - **`exec()`** chamada de system usada depois de uma **`fork()`** para realocar o espaço da memória de processo para um novo programa



Criando um processo separado usando a chamada de sistema fork( ) do UNIX.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

Criando um processo separado usando a API Windows.

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
      fprintf(stderr, "Create Process Failed");
      return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Encerramento de Processos

- O processo executa o último comando e depois solicita ao sistema operacional que o exclua, usando a chamada de sistema **`exit()`**.
  - Retorna o valor de status do filho para o pai [por meio da chamada de sistema **`wait()`**]
  - Todos os recursos do processo são desalocados pelo sistema operacional.
- Pai pode encerrar a execução de processo-filho usando a chamada de sistema **`abort()`**. Algumas razões para fazer isso:
  - O filho excedeu o uso de recursos alocados
  - A tarefa atribuída ao filho não é mais requerida
  - O pai está sendo encerrado e o sistema operacional não permite que um filho continue se seu pai for encerrado.

Encerramento de Processos

- Alguns sistemas não permitem que um filho exista se seu pai tiver sido encerrado. Se um processo for encerrado, todos os seus filhos também devem ser encerrados
  - **Encerramento em cascata:** Todos os filhos, netos, etc são encerrados.
  - O encerramento é iniciado pelo sistema operacional.
- Um processo-pai pode esperar o encerramento de um processo-filho usando a chamada de sistema **`wait()`.** A chamada retorna a informção de status e o pid do processo encerrado

```
pid = wait(&status);
```

- Se o processo-pai não chamou o **`wait()`** o processo é um (zumbi) **zombie**
- Se o processo-pai foi encerrado sem invocar **`wait()`** , o processo-filho será um (órfão) **orphan**

ARQUITETURA MULTIPROCESSOS — NAVEGADOR CHROME

- Muitos navegadores rodavam como processo único (alguns ainda são assim)
  - Se um site criar problemas, o navegador inteiro pode travar e até fechar.
- O navegador Google Chrome é um multiprocessos om 3 tipos diferentes de processos:
  - **Browser** O processo (Navegador) gerencia a interface de usuário, assim como o I/O de disco e de rede.
  - **Renderer** O processo (renderizador) renderiza páginas da web, contém a lógica para para manipulação de HTML, Javascript. Um novo renderizador é criado para cada website aberto em uma nova guia.
    - São executados no **sandbox** restringindo o acesso I/O de disco e de rede, minimizando os efeitos de qualquer invasão na segurança.
  - Um processo de **Plug-in** é criado para cada tipo de plug-in em uso.

Universidade de Brasília Faculdade UnB Gama

Comunicação Interprocessos

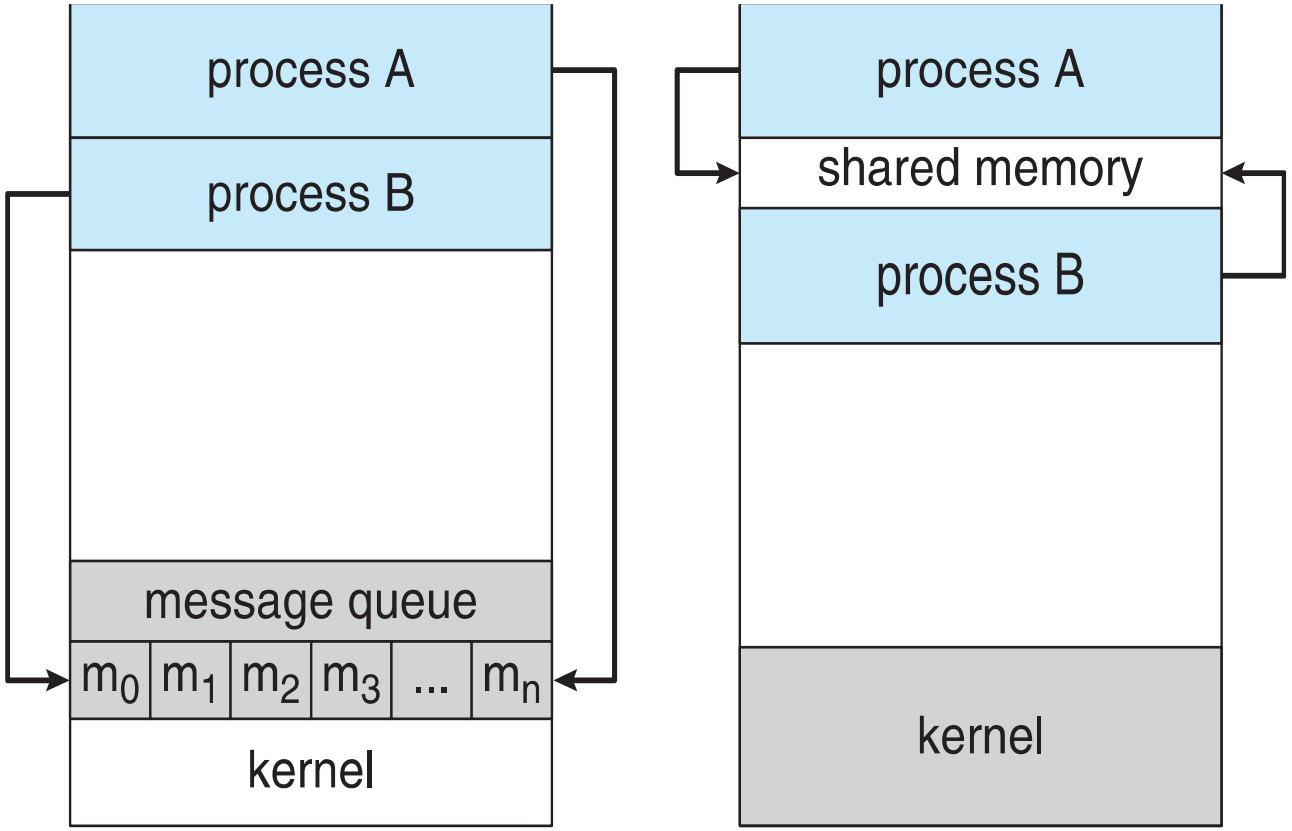
- Processos dentro de um sistema podem ser ***independentes*** ou ***cooperativos***
- Processos cooperativos podem afetar ou ser afetados por outros processos e incluem o compartilhamento de dados.
- Razões para a cooperação entre processos:
  - Compartilhamento de informações
  - Aumento da velocidade de computação
  - Modularidade
  - Conveniência
- Processos cooperativos demandam um mecanismo de (comunicação entre processos) **interprocess communication** (**IPC**)
- Dois modelos de IPC
  - **Shared memory (**Memória compartilhada)
  - **Message passing (**Transmissão de mensagens)

Universidade de Brasília

Faculdade UnB Gama

Modelos de comunicação

(a) transmissão de mensagens. (b) memória compartilhada.



Processos cooperativos

- ***Processos independentes*** não podem afetar ou ser afetados pela execução de outro processo.
- ***Processos Cooperativos*** podem afetar ou ser afetados pela execução de outro processo.
- Vantagens da cooperação de processos
  - Compartilhamento de informações
  - Aumento da velocidade de computação
  - Modularidade
  - Conveniência

Universidade de Brasília

Faculdade UnB Gama

Problema do produtor-consumidor

- Paradigma para os processos cooperativos, o processo produtor produz informações que são consumidas por um processo consumidor
  - **(buffer ilimitado) unbounded-buffer** não impõe um limite prático ao tamanho do buffer
  - **(buffer limitado) bounded-buffer** assume um tamanho de buffer fixo

Buffer limitado – Solução de compartilhamento de memória

- Shared data

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

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

- Solution is correct, but can only use BUFFER_SIZE-1 elements

Buffer limitado – Produtor

```
item next_produced;
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)  /* buffer is full */
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Buffer limitado – Consumidor

```
item next_consumed;

while (true) {
        while (in == out) /* empty buffer */
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

Comunicação Interprocessos – Memória Compartilhada

- Uma área da memória compartilhada entre os processos que desejam se comunicar
- A comunicação fica sob o controle dos processos do usuário não do sistema operacional.
- Um grande problema é fornecer um mecanismo que permita os processos do usuário sincronizarem as ações quando eles acessarem a memória compartilhada.
- A sincronização será discutida em mais detalhes no capítulo 5.

Comunicação Interprocessos – Transmissão de mensagens

- Mecanismo que permitir que os processos se comuniquem e sincronizem suas ações
- Sistema de mensagem – processos communicam entre si sem recorrer a variáveis compartilhadas
- Um recurso na transmissão de mensagens na Comunicação de interpocessos fornece duas operações:
  - **`send`**(*message*) – enviar mensagem
  - **`receive`**(*message*) receber mensagem
- O tamanho da mensagem pode ser fixo ou variável

Universidade de Brasília

Faculdade UnB Gama

Transmissão de mensagens (Cont.)

- Se os processos *P* e *Q* querem se comunicar eles devem: to :
  - Implementar um ***link de comunicação*** entre si
  - Trocar mensagens por meio de send/receive enviar/receber
- Problemas na implementação:
  - Como os links são implementados?
  - Um link pode ser associado a mais de dois processos?
  - Quantos links podem existir entre cada par de processos communicativos?
  - Qual é a capacidade de um link?
  - O tamanho da mensagem que um link comporta é fixo ou variável?
  - O link é unidirecional ou bidirecional?

Message Passing (Cont.)

- Implementação de um link de comunicação
  - Física:
    - Memórica compartilhada
    - Bus de Hardware
    - Rede
  - Lógica:
    - Direta ou indireta
    - Síncrona ou assíncrona
    - Armazenamento em buffer automático ou explícito

Comunicação Direta

- Os Processos devem se nomear explicitamente :
  - **`send`** (*P, message*) – Envia uma mensagem ao processo P
  - **`receive`**(*Q, message*) – Recebe uma mensagem do processo Q
- Propriedades de comunicação do link
  - Links são estabelecidos automaticamente
  - Um link é associado a exatamente dois processos que se comunicam.
  - Entre cada par de processos, existe exatamente um link.
  - O link pode ser unidirecional ou bidirecional

Comunicação Indireta

- As mensagens são enviadas para e recebidas de caixas postais, (também chamadas de portas)
  - Cada caixa postal tem uma identificação exclusiva
  - dois processos só podem se comunicar se tiverem uma caixa postal compartilhada
- Propriedades do link de comunicação
  - Um link é estabeleciodo somente se os processos compartilharem uma caixa postal
  - Um link pode estar associado a mais de dois processos
  - Entre cada par de processos em comunicação pode haver vários links de comunicação
  - O link pode ser unidirecional ou bidirecional

Comunicação Indireta

- Operações
  - Criar uma nova caixa postal. (porta)
  - Enviar e receber mensagens pela caixa postal.
  - Excluir uma caixa postal
- Primitivas são definidas como:

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

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

Universidade de Brasília

Faculdade UnB Gama

Comunicação Indireta

- Compartilhamento de caixa postal
  - P_1, P_2, and P_3 compartilham a caixa postal A
  - P_1, sends; P_2 and P_3 recebem
  - Quem recebe a mensagem?
- Soluções
  - Permitir que um link seja associado a, no máximo, dois processos
  - Permitir que, no máximo, um processo de cada vez execute uma operação *receive*
  - Permitir que o sistema selecione arbitrariamente o processo que receberá a mensagem. O emissor é notificado sobre quem é o receptor.

Sincronização

- A transmissão de mensagens pode ser com bloqueio ou sem bloqueio
- **Blocking** (Com bloqueio) é considerada **synchronous (síncrona)**
  - **Envio com bloqueio** -- O processo emissor é bloqueado até que a mensagem seja recebida
  - **Recebimento com bloqueio** -- O receptor é bloqueado até que a mensagem fique disponível
- **Non-blocking (Sem bloqueio)** é considerada **asynchronous (assíncrona)**
  - **Envio sem bloqueio** -- O processo emissor envia a mensagem e retoma a operação
  - **Recebimento sem bloqueio** -- o receptor recupera:
    - uma mensagem válida ou
    - uma mensagem nula
- Diferentes combinações são possíveis
  - Quando send ( ) e receive ( ) são com bloqueio, temos um ponto de encontro **rendezvous** entre o emissor e o receptor

Universidade de Brasília Faculdade UnB Gama

Sincronização (Cont.)

- Produtor – consumidor torna-se trivial

```
message next_produced;
while (true) {
    /* produce an item in next produced */
send(next_produced);
}

message next_consumed;
while (true) {
   receive(next_consumed);

   /* consume the item in next consumed */
}
```

Armazenamento em Buffer

- Filas de mensagens associadas a um link.
- Podem ser implementadas de três maneiras
  1. Capacidade zero – nenhuma mensagem em espera é associada ao link. Emissor deve esperar pelo receptor (rendezvous)
  2. Capacidade limitada – a fila tem tamanho finito n. Emissor tem que esperar se o link estiver cheio.
  3. Capacidade ilimitada – tamanho infinito. Emissor nunca espera.

Exemplos de Sistemas IPC - POSIX

- Memória Compartilhada no POSIX
  - o processo deve criar um objeto de memória compartilhada usando a chamada de sistema **`shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);`**
  - Também é usado para abrir um objeto e para compartilhá-lo.
  - Configurar o tamanho do objeto

    `ftruncate(shm_fd, 4096);`
  - Agora o processo poderia escrever na memória compartilhada

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

Universidade de Brasília

Faculdade UnB Gama

Produtor IPC POSIX

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* create the shared memory object */
   shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

   /* configure the size of the shared memory object */
   ftruncate(shm_fd, SIZE);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

   /* write to the shared memory object */
   sprintf(ptr,"%s",message_0);
   ptr += strlen(message_0);
   sprintf(ptr,"%s",message_1);
   ptr += strlen(message_1);

   return 0;
}
```

Consumidor IPC POSIX

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

    /* open the shared memory object */
    shm_fd = shm_open(name, O_RDONLY, 0666);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* read from the shared memory object */
    printf("%s",(char *)ptr);

    /* remove the shared memory object */
    shm_unlink(name);

    return 0;
}
```

Examplos de Sistemas IPC - Mach

- A maior parte da comunicação no Mach é executada por mensagens.
  - Até chamdas de Sistema são mensagens
  - Cada tarefa recebe duas caixas postais especiais ao ser criada- a Kernel e a Notify
  - Apenas três chamadas de sistema são necessárias para a transferência de mensagens

    `msg_send(), msg_receive(), msg_rpc()`
  - Caixas postais são necessárias para a comunicação e são criadas por meio de

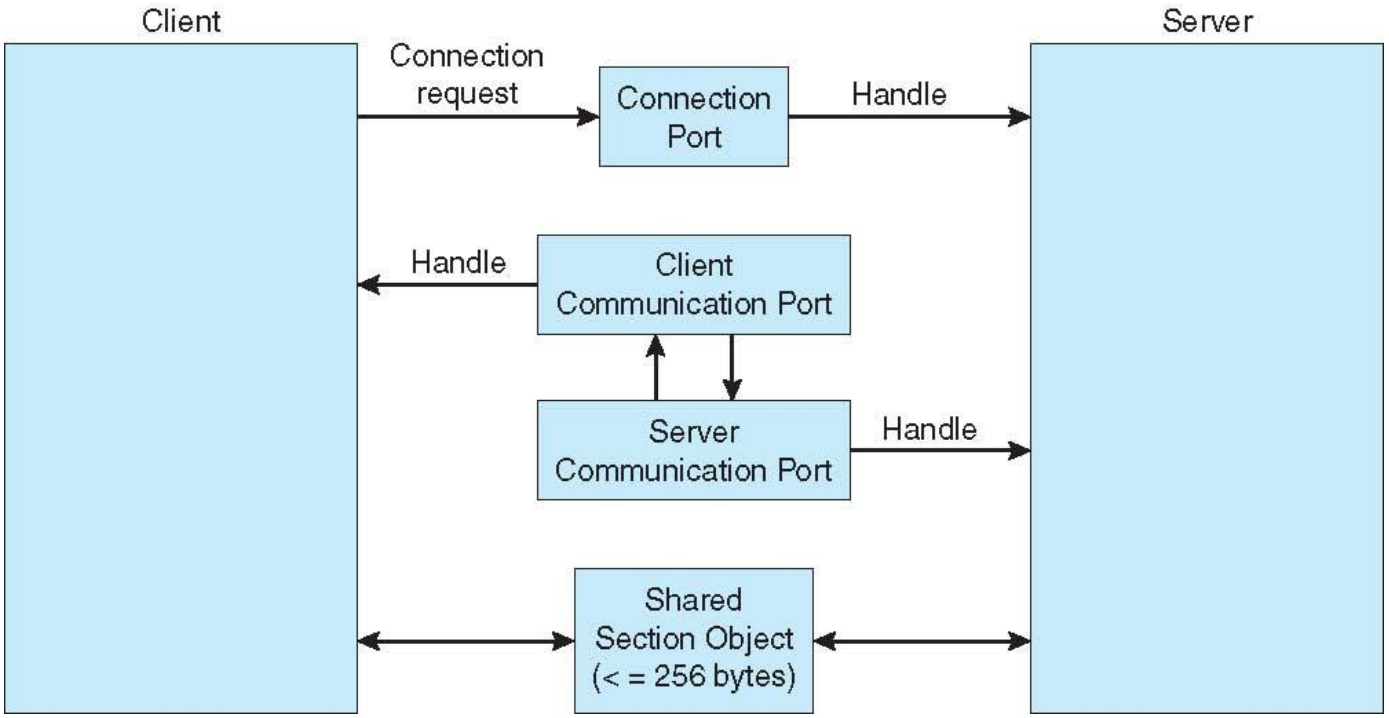
    `port_allocate()`
  - As operações de envio e recebimento são flexíveis . Quando a caixa postal está cheia, o thread emissor tem quatro opções:
    - Esperar indefinidamente
    - Esperar no máximo n milissegundos
    - Retornar imediatamente
    - Armazenar temporariamente uma mensagem em cache

Universidade de Brasília Faculdade UnB Gama

Exemplos de Sistemas IPC – Windows

- Esse sistema é centrado na transmissão de mensagens por meio do recurso (chamada de procedimento local avançada) **advanced local procedure call (LPC)**
  - Só funciona entre processos no mesmo sistema
  - Usa portas (como caixas postais) para estabelecer e manter uma conexão entre dois processos
  - A comunicação funciona da seguinte forma:
    - O cliente abre um manipulador para o objeto porta de conexão do servidor porta de conexão **connection port**.
    - Envia uma solicitação de conexão a essa porta
    - O servidor cria duas portas de comunicação **communication ports** e retorna um manipulador para o cliente .
    - O cliente e o servidor usam o manipulador de porta correspondente para enviar mensagens ou retorno de chamadas para aceitar solicitações quando estiverem esperando por uma resposta.

Chamadas de procedimento locais avançadas no Windows.



Comunicação em Sistemas Cliente-Servidor

- Sockets
- Chamadas de procedimento remotas
- Pipes
- Chamadas de métodos remotos (Java)

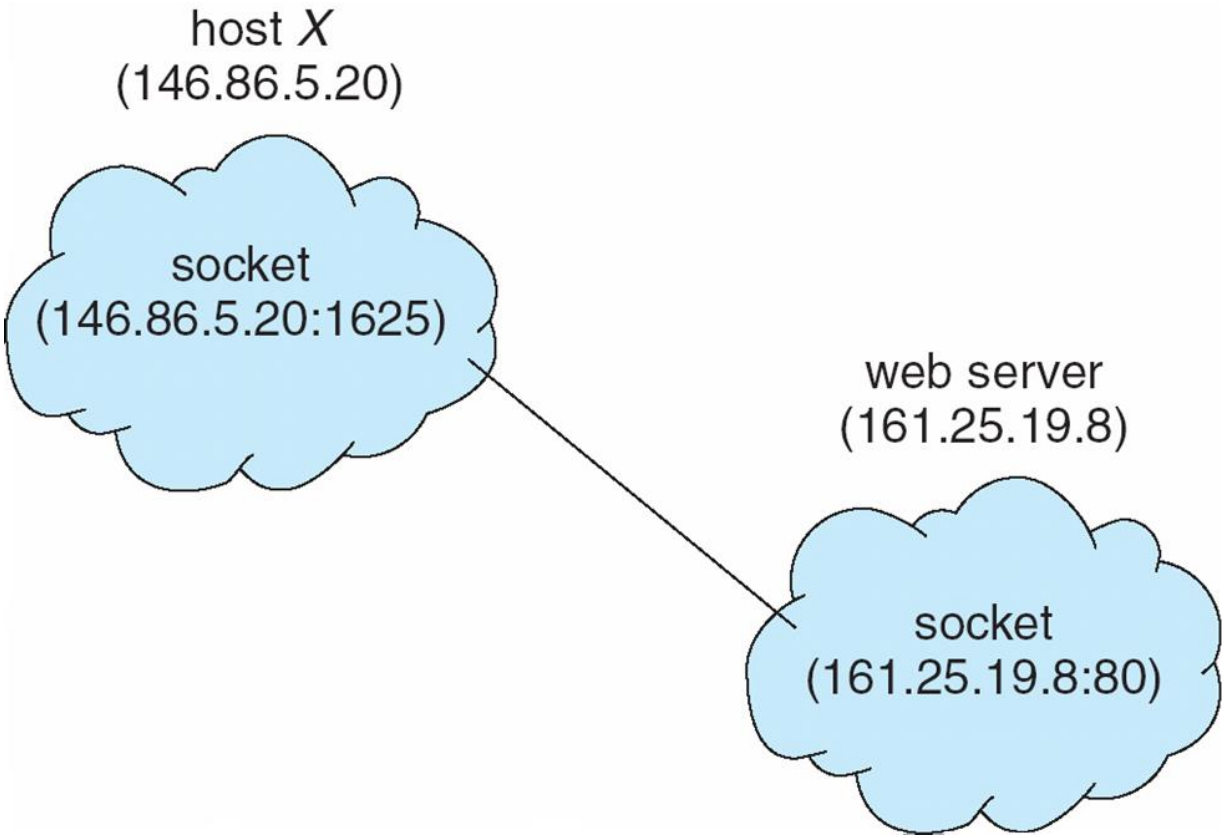
Sockets

- Um **socket** é definido como uma extremidade de comunicação
- Concatenação do endereço de IP e um número de porta **port** – um número incluído no começo de um pacote de mensagen para diferenciar serviços de rede em um hospedeiro.
- O socket **161.25.19.8:1625** significa porta **1625** no hospedeiro **161.25.19.8**
- A comunicação emprega um par de sockets
- Todas as portas abaixo de 1024 são ***bem conhecidas*** e são usadas para implementar serviços-padrão
- O endereço IP 127.0.0.1 é um endereço IP especial conhecido como autorretorno (**loopback**) e é para se referir ao sistema no qual o processo está sendo executado.

Universidade de Brasília

Faculdade UnB Gama

Comunicação com o uso de sockets



Universidade de Brasília

Faculdade UnB Gama

Sockets no Java

- Três tipos diferentes de sockets
  - **Connection-oriented** (**TCP**) sockets orientado à conexão
  - **Connectionless** (**UDP**) sockets sem conexão
  - **`MulticastSocket`** class– dados podem ser enviados a vários receptores
- Considere este servidor de data:

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Chamadas de Procedimento Remotas

- Chamadas de procedimento remotas (RPC) abstrai o mecanismo de chamada de procedimento para uso entre sistemas com conexões de rede
  - Novamente usa portas para diferenciação de serviços
- **Stubs** – é um proxy do lado do cliente para o procedimento real no servidor
- O stub do lado do cliente localiza a porta no servidor e organiza os parâmetros **marshalls**
- O stub do lado do servidor recebe essa mensagem, desempacota os parâmetros organizados, e faz o procedimento no servidor
- No Windows, o código do stub é compilado a partir de uma especificação escrita na **Microsoft Interface Definition Language** (**MIDL**)

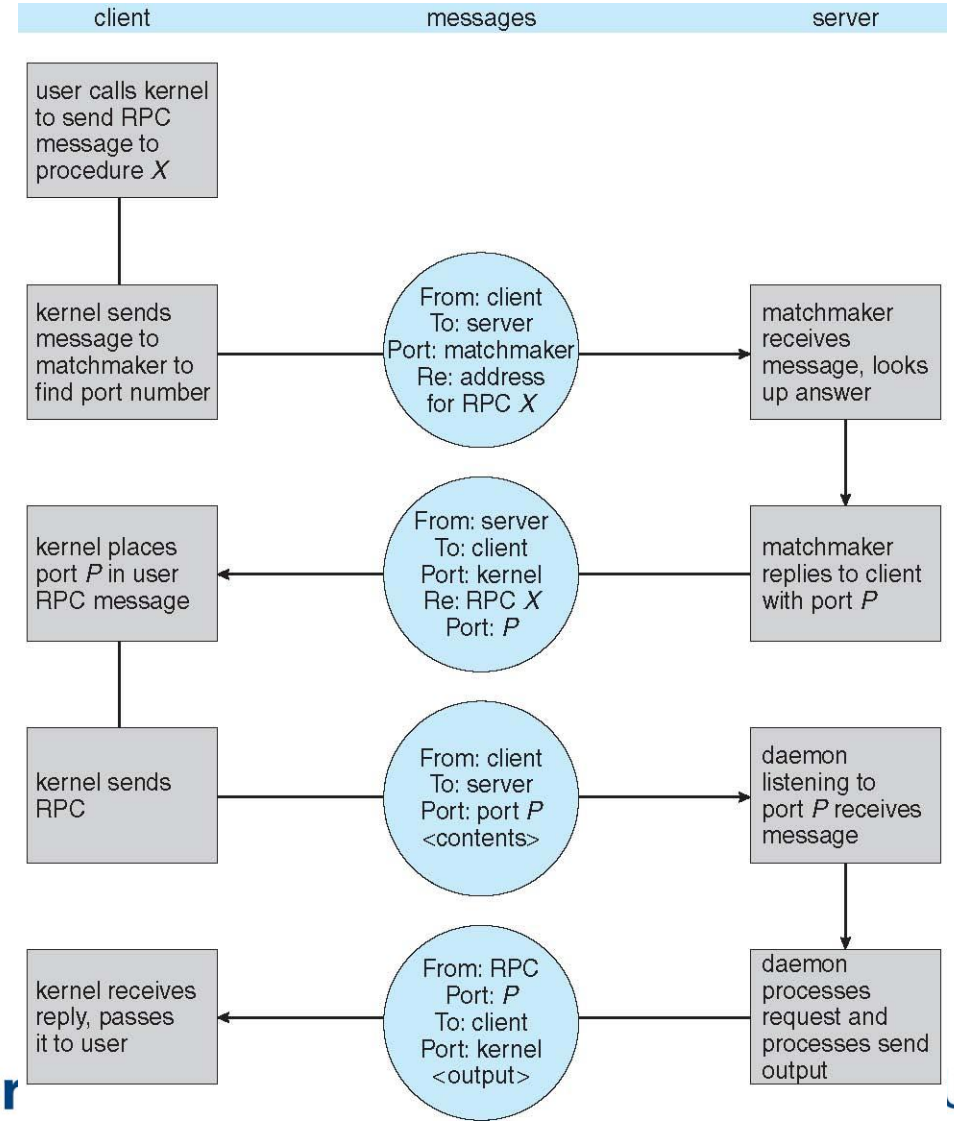
Chamadas de Procedimento Remotas (Cont.)

- Representação de dados é feita por meio de representação de dados externa **External Data Representation** (**XDR**) para achar uma solução considerando os diferentes tipos de arquitetura
  - **Big-endian** e **little-endian**
- Comunicação remota tem mais possibilidades de falhas do que as chamadas de procedimento locais
  - As mensagens sejam podem ser entregues ***exatamente uma vez***, em vez de ***no máximo uma vez***
- O sistema operacional fornece um serviço de rendezvous (or **matchmaker**) para conectar cliente e servidor

Universidade de Brasília

Faculdade UnB Gama

Execução de RPC

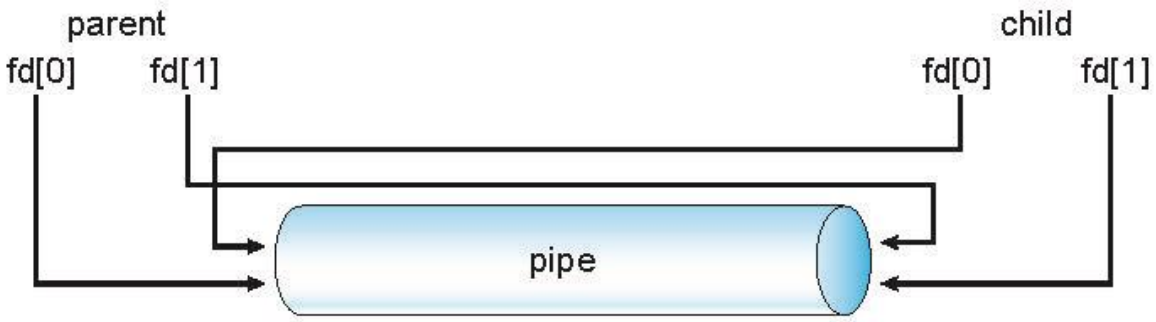


Pipes

- Atua como um canal que permite que dois processos se comuniquem
- Problemas:
  - A comunicação é unidirecional or bidirecional?
  - No caso de comunicação bidirecional, ela é half or full-duplex?
  - Deve existir um relacionamento (do tipo **pai-filho**) entre os processos em comunicação?
  - Os pipes podem se comunicar por uma rede?
- Pipes comuns – Não podem ser acessados de fora dos processos que os criaram. Normalmente, um processo-pai cria um pipe e o usa para se comunicar com um processo-filho que ele cria.
- Pipes nomeados – podem ser acessados sem um relação pai-filho.

Pipes Comuns

- Os pipes comuns permitem que dois processos se comuniquem na forma-padrão produtor-consumidor
- O produtor grava em uma extremidade do pipe (a extremidade de gravação) (**write-end**)
- O consumidor lê da outra extremidade (a extremidade de leitura) (**read-end**)
- Como resultado, os pipes comuns são unidirecionais
- Requerem um relacionamento pai-filho entre os processos em comunicação



- Windows chama eles de pipes anônimos **anonymous pipes**
- Veja uma amostra do códigos do Unix and Windows no livro texto

Pipes Nomeados

- Os pipes nomeados são mais poderosos do que os pipes comuns
- A comunicação é bidirecional
- Não há necessidade de relacionamento pai-filho entre os processos em comunicação
- Vários processos podem o usar o pipe nomeado para a comunicação
- São fornecidos em ambos os sitemas UNIX e Windows

Fim do Capítulo 3