SISTEMAS OPERACIONAIS

Aula 3
Fundamentos do SO
Arquitetura de Computadores
Estruturas de dados

Prof. Dr. Aldo Díaz
Instituto de Informática
Universidade Federal de Goiás
aldo.diaz@ufg.br

Tradução: Rogério R. Rocha

Review: Aula 2

Review: Aula 2

- Os sistemas operacionais (SO) estão presentes em todos os tipos de sistemas de computação modernos
- Foram introduzidos conceitos fundamentais do SO:
  - Características: Multiprogramação, multitarefa,
  - Gerenciamento/Management: tempo de CPU, E/S, processos, memória, sistema de arquivos, armazenamento em massa
  - Estrutura/Structure: serviços de SO, chamadas de sistema
  - Estruturas de dados do kernel: listas, pilha, fila
  - Tipos: Monolítico, microkernel
- O sistema operacional é fundamental para o gerenciamento eficiente de recursos de hardware de computação

Aula 3: Processos

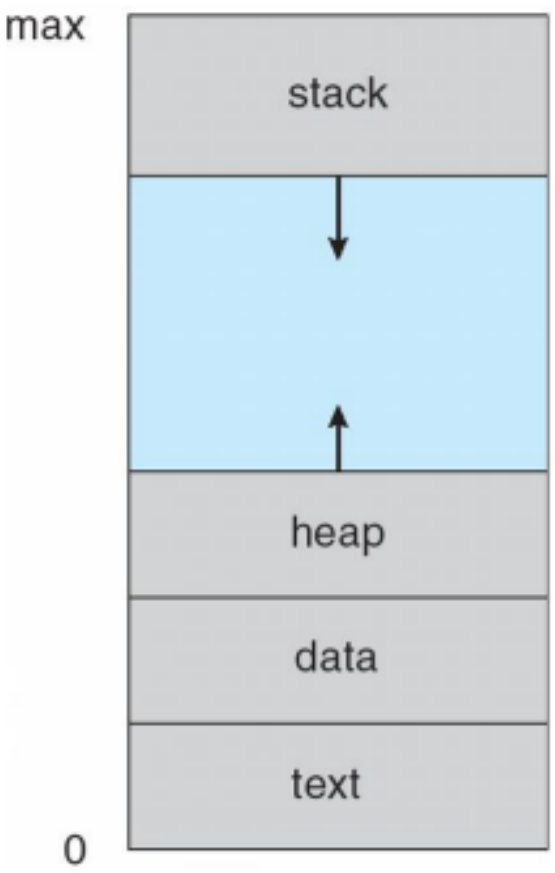
Objetivos

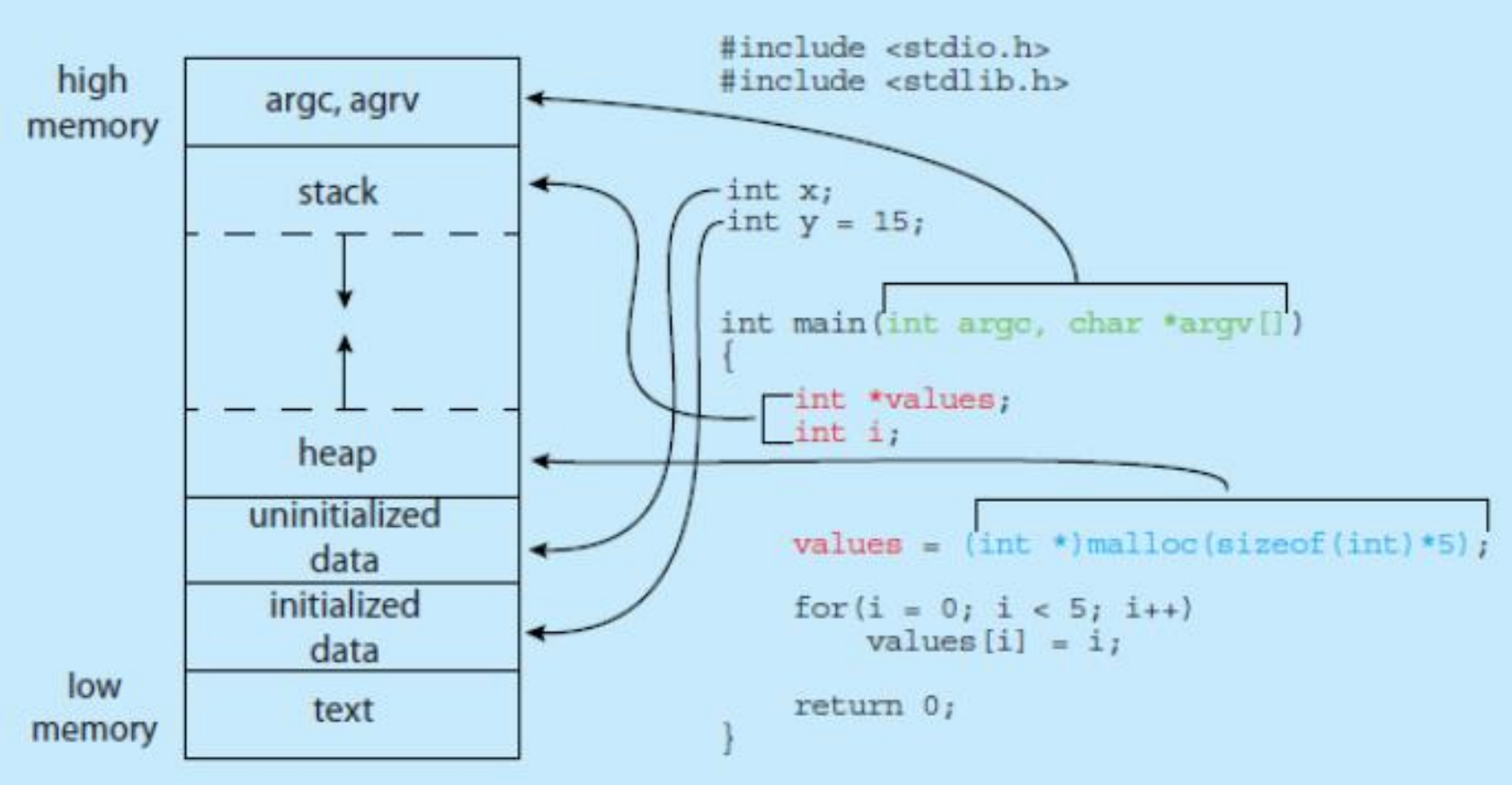
- Introduzir a noção de processo – um programa em execução, que forma a base de toda computação
- Descrever como os processos são criados e finalizados em um sistema operacional, incluindo o desenvolvimento de programas usando as chamadas de sistema apropriadas que executam essas operações

Conceito de Processo

- Os computadores modernos permitem que vários programas sejam carregados na memória e executados simultaneamente
- Processo ou job – um programa em execução; a execução do processo deve progredir de forma sequencial
- Múltiplas peças
  - O código do programa, também chamado de seção de texto (“*text section*”)
  - Atividade atual, incluindo contador de programa (“*program counter*”), registros do processador
  - Pilha (“*Stack*”) contendo dados temporários
    - Parâmetros de função, endereços de retorno, variáveis locais
  - Seção de dados contendo variáveis globais
  - Heap contendo memória alocada dinamicamente durante o tempo de execução

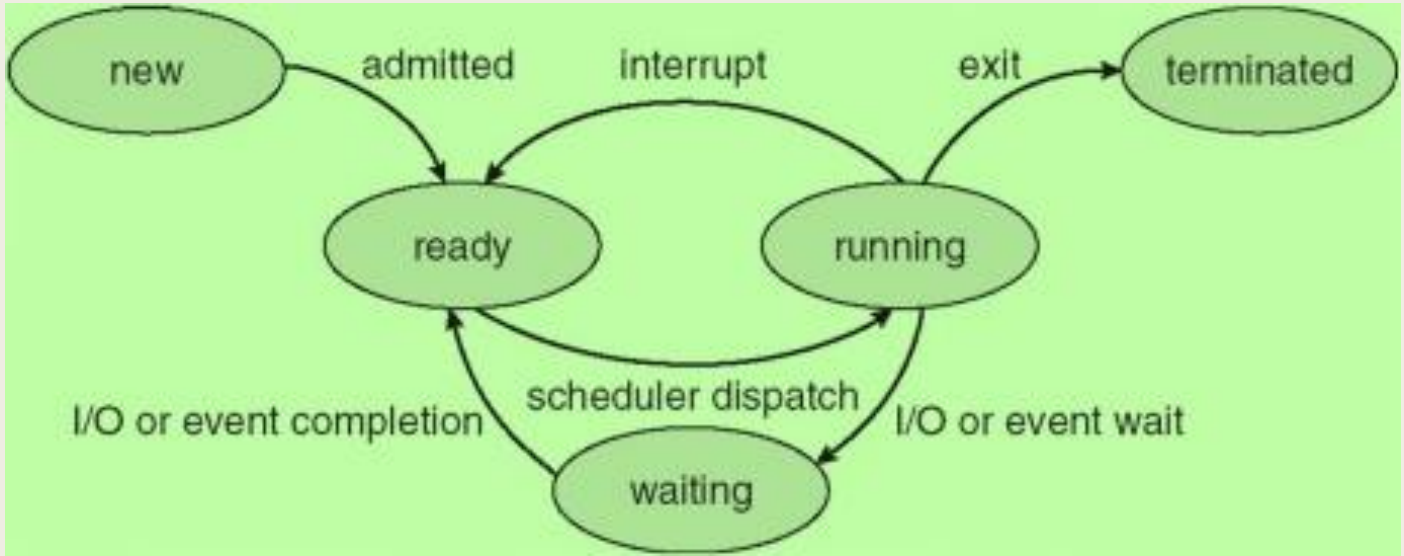
Layout de um processo na memória





Estado do processo

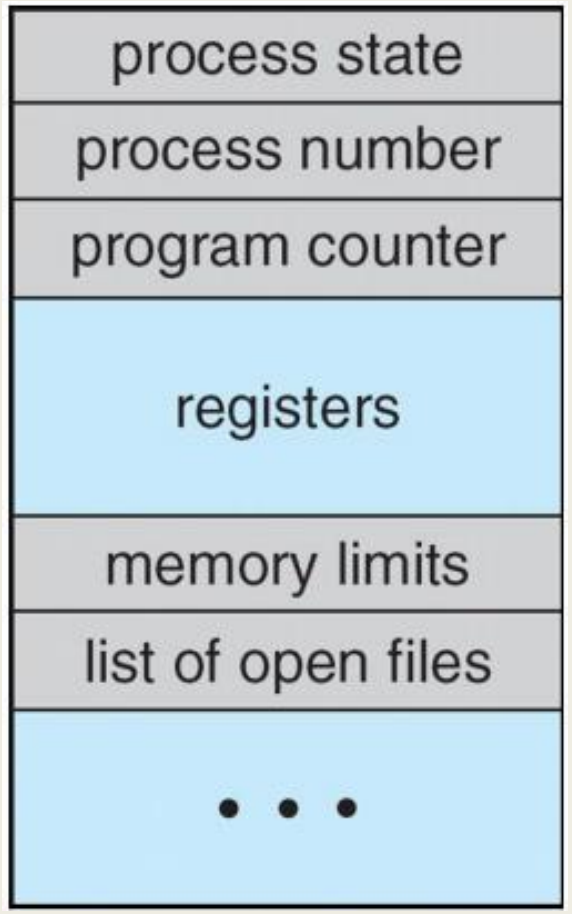
- À medida que um processo é executado, ele muda de estado
  - **novo/new**: o processo está sendo criado
  - **running**: As instruções estão sendo executadas
  - **esperando/waiting** : O processo está esperando que algum evento ocorra, por exemplo, conclusão de E/S ou recepção de um sinal
  - **pronto/ready** : o processo está aguardando para ser atribuído a um processador
  - **encerrado/terminated** : O processo terminou a execução



Bloco de controle de processo (PCB)

Informações associadas a cada processo (também chamado de bloco de controle de tarefas)

- **Estado do processo** (“Process state”) – em execução, em espera, etc.
- **Contador de programa** (“Program counter”) – localização da instrução a ser executada em seguida
- **Registros de CPU** (“CPU registers”) – conteúdo de todos os processos – registros centrados
- **Informações de agendamento de CPU** (“CPU scheduling information”) – prioridades, ponteiros de fila de agendamento
- **Informações de gerenciamento de memória** (“Memory -management information”) – memória alocada para o processo
- **Informações contábeis** (“Accounting information”) – CPU usada, tempo decorrido desde o início, limites de tempo
- **Informações de status de E/S** (“I/O status information”) – dispositivos de E/S alocados para processo, lista de arquivos abertos



Agendamento de Processos

Agendamento de Processos

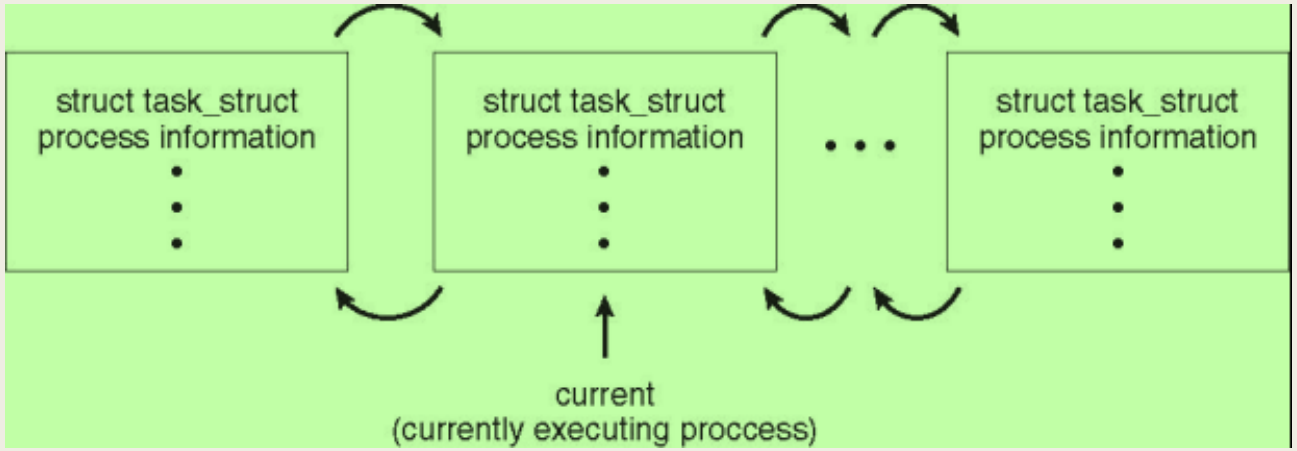
- Objetivo: Maximizar o uso da CPU, alternar rapidamente os processos para a CPU para compartilhamento de tempo (lembre-se: a interatividade é importante para o usuário!)
- O agendador de processos (“process scheduler”) seleciona entre os processos disponíveis para a próxima execução no núcleo da CPU
- Mantém filas de agendamento de processos
  - Fila de trabalhos (“*Job*”) – conjunto de todos os processos do sistema
  - Fila pronta (“*Ready*”) – conjunto de todos os processos residentes na memória principal, prontos e aguardando para serem executados
  - Fila de espera (“*Wait*”) – conjunto de processos aguardando a ocorrência de um determinado evento, por exemplo, a conclusão de uma solicitação de E/S
  - Os processos migram entre as diversas filas

Representação do Bloco de Controle de Processo no Linux

O PCB no Linux é representado pela estrutura C task_struct

- Linux prefere o termo tarefa/task em vez de processo

```
long state;                     /* state of the process */
struct sched_entity se;         /* 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 */
```

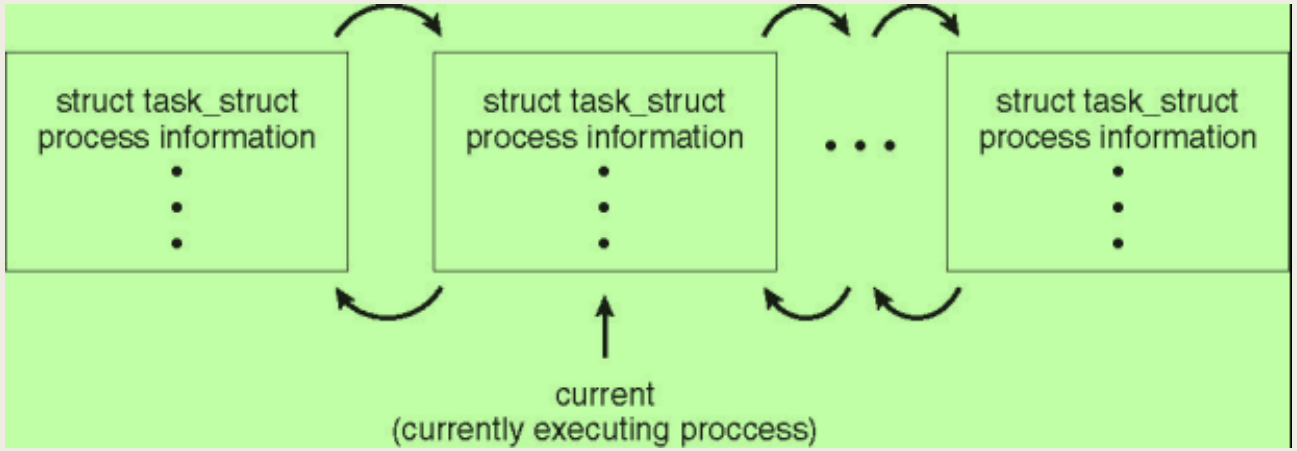


Representação do Bloco de Controle de Processo no Linux

O PCB no Linux é representado pela estrutura C task_struct

- Linux prefere o termo tarefa/task em vez de processo

```
estado longo;                    /* estado do processo */
estrutura sched_entity se;       /* informações de agendamento */
struct task_struct *pai;         /* pai deste processo */
struct list_head filhos;         /* os filhos deste processo */
struct arquivos_struct *arquivos; /* lista de arquivos abertos */
estrutura mm_struct *mm;         /* espaço de endereço deste processo */
```



Fila pronta e Fila de espera

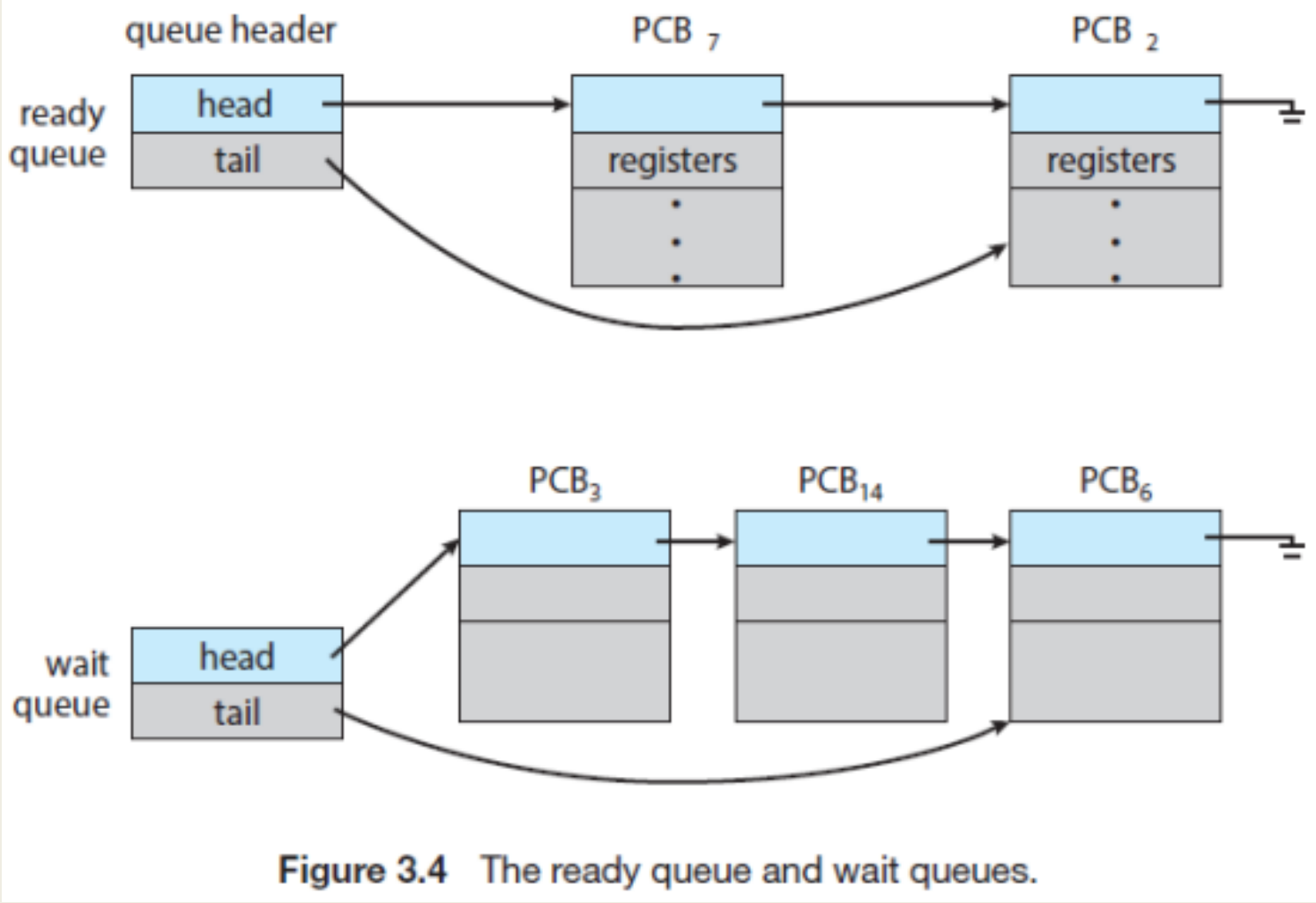
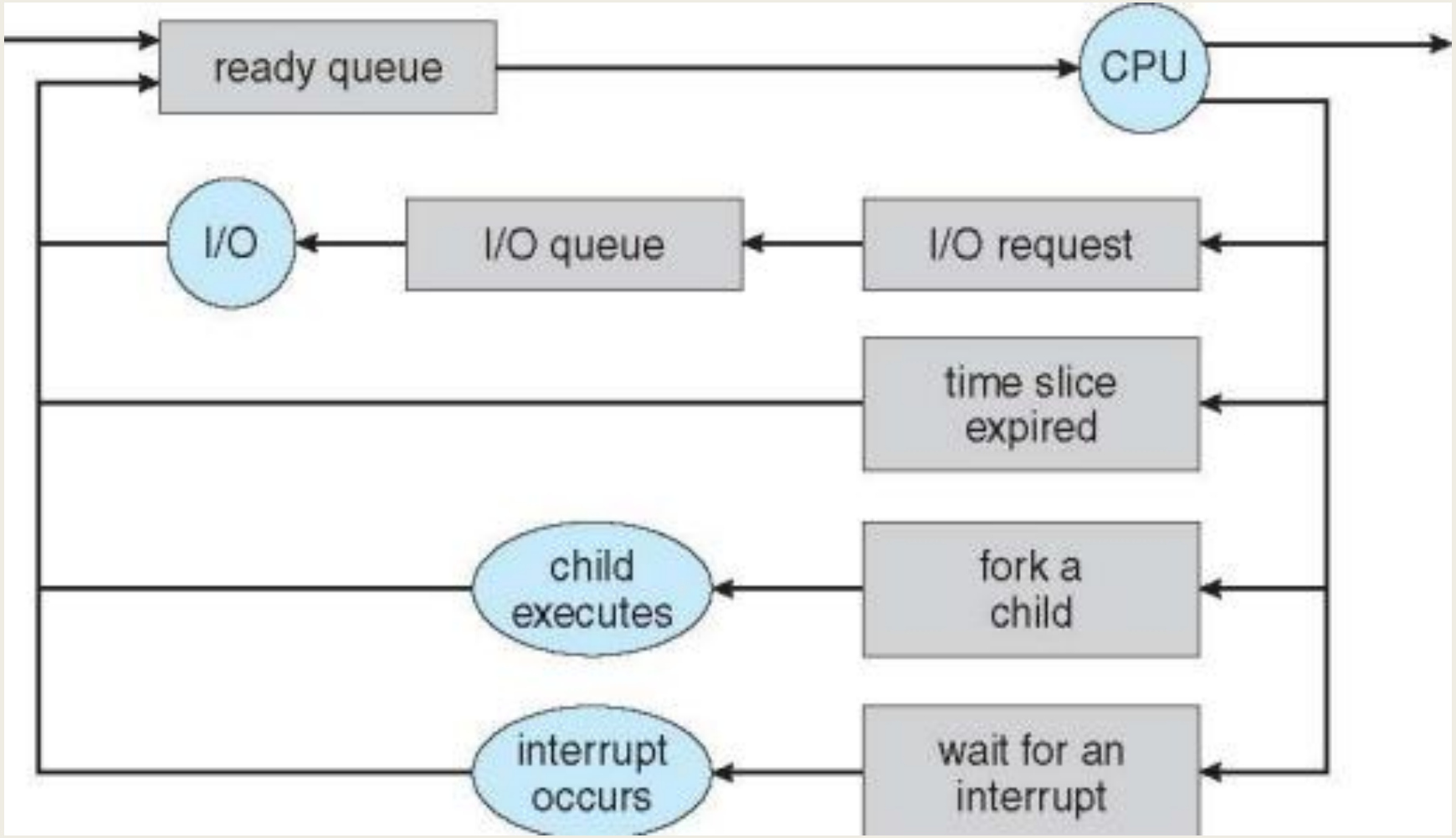


Figure 3.4 The ready queue and wait queues.

Representação do Agendamento de Processos

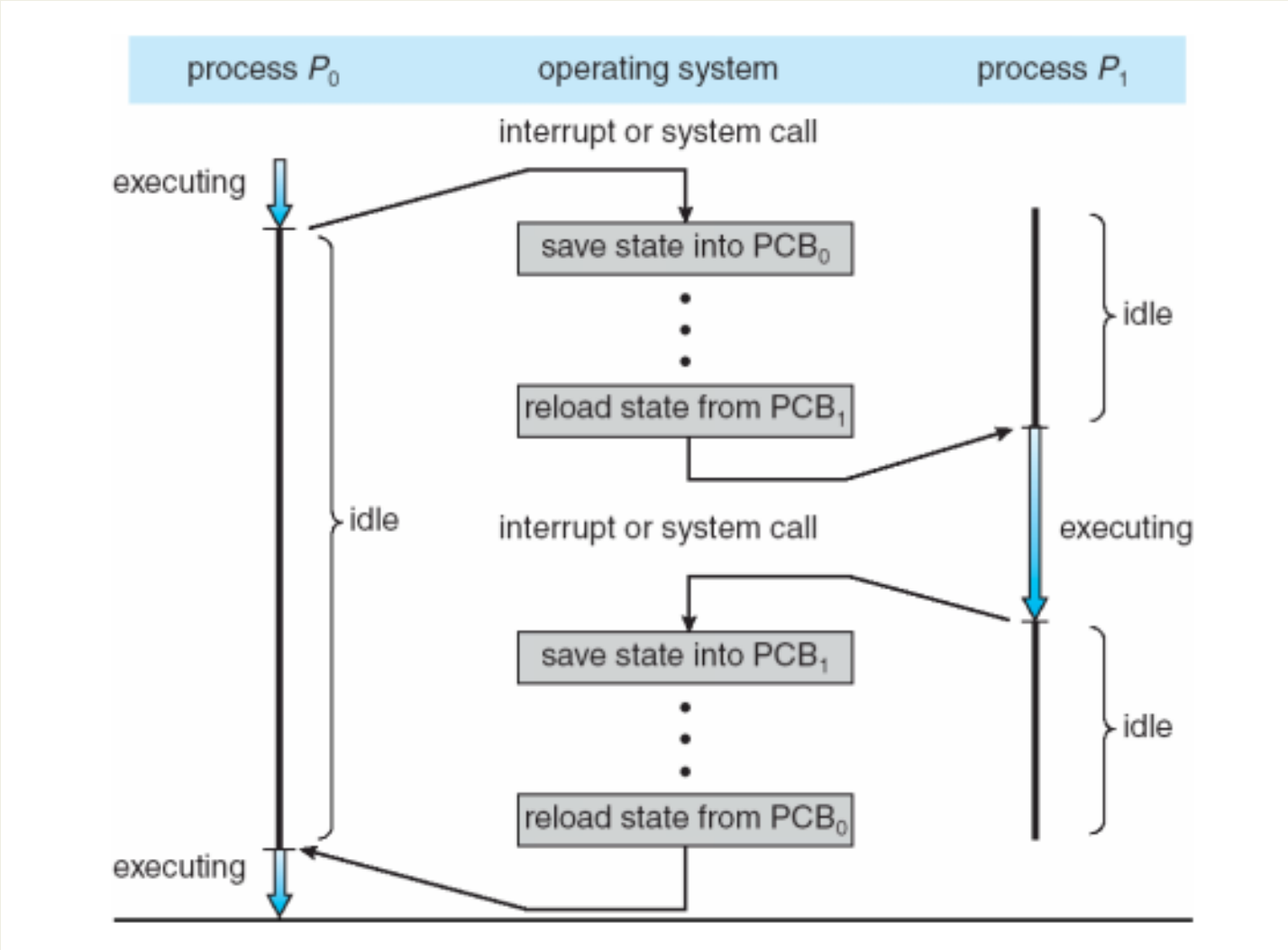
O **diagrama de filas** representa filas, recursos, fluxos



Mudança de contexto / Context Switch

- Quando a CPU muda para outro processo, o sistema deve salvar o estado do processo antigo e carregar o estado salvo para o novo processo por meio de uma troca de contexto
- Contexto de um processo representado no PCB
- O tempo de troca de contexto está sobrecarregado; o sistema não faz nenhum trabalho útil durante a comutação
  - Quanto mais complexo o sistema operacional e o PCB ➔ mais longa será a troca de contexto
- Tempo dependente do suporte de hardware
  - Alguns hardwares fornecem vários conjuntos de registros por CPU ➔ vários contextos carregados de uma só vez

Mudança de contexto / Context Switch



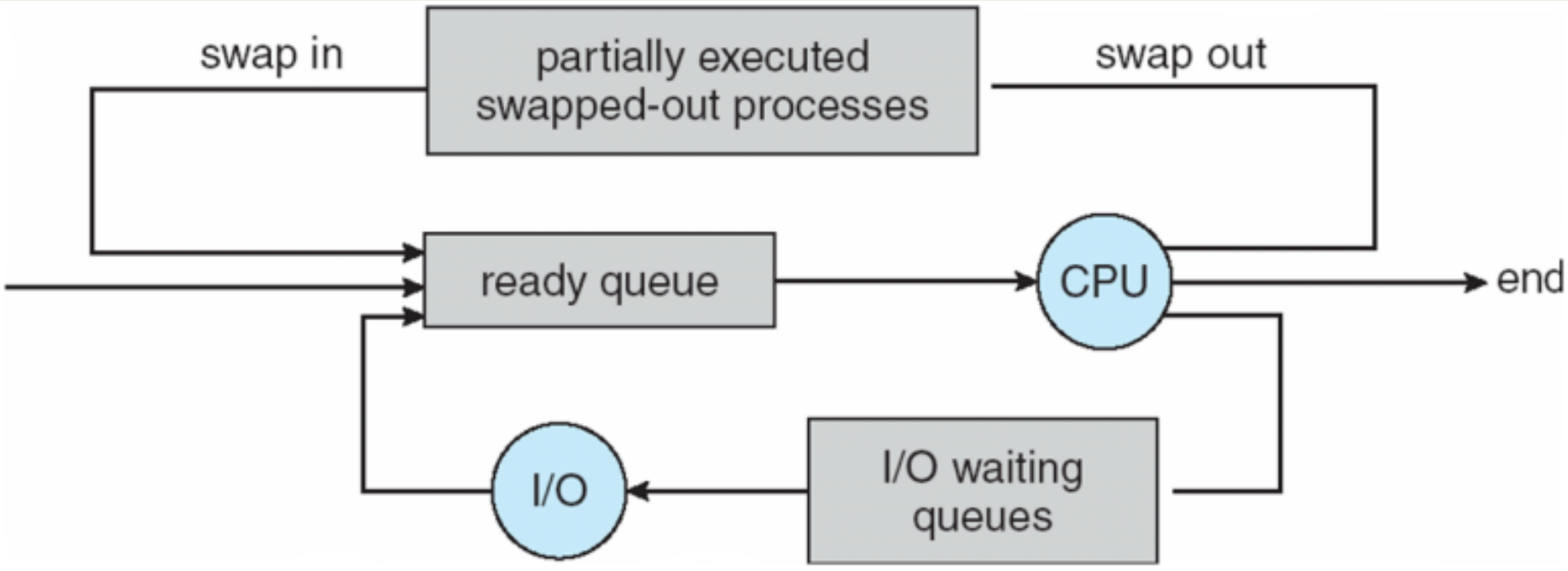
Schedulers/Agendadores

Agendadores / Schedulers

- Agendador de curto prazo (ou agendador de CPU) – seleciona qual processo deve ser executado em seguida e aloca CPU
  - Às vezes, o único agendador em um sistema
  - O escalonador de curto prazo é invocado frequentemente (milissegundos) ☐ (deve ser rápido)
- Agendador de longo prazo (ou agendador de tarefas) – seleciona quais processos devem ser colocados na fila de prontidão
  - O escalonador de longo prazo é invocado com pouca frequência (segundos, minutos) ☐ (pode ser lento)
  - O escalonador de longo prazo controla o grau de multiprogramação
- Os processos podem ser descritos como:
  - Processo vinculado a E/S – gasta mais tempo fazendo E/S do que computando, muitas rajadas curtas de CPU
  - Processo vinculado à CPU – gasta mais tempo fazendo cálculos; algumas rajadas de CPU muito longas
- O escalonador de longo prazo busca um bom mix de processos

Adição de Agendamento de Médio Prazo

- O escalonador de médio prazo pode ser adicionado se o grau de programação múltipla precisar diminuir
  - Remover processos da memória, armazená-los em disco, trazê-los de volta do disco para continuar a execução: troca (“*swapping*”)

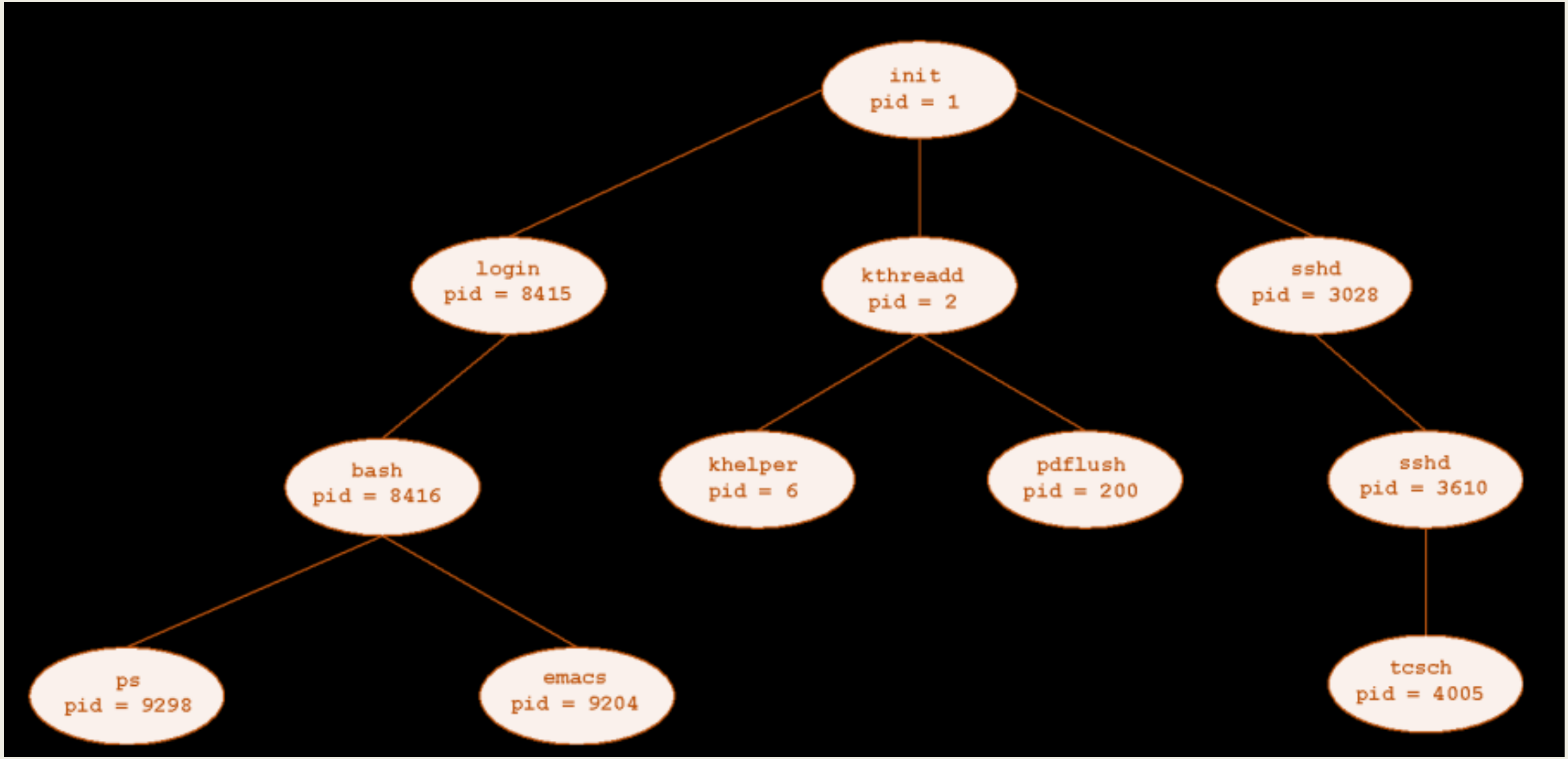


Operações em Processos

Criação de Processo

- O processo pai cria processos filhos, que, por sua vez, criam outros processos, formando uma árvore de processos
- Geralmente, processo identificado e gerenciado por meio de um identificador de processo (pid)
  - Opções de compartilhamento de recursos
  - Pais e filhos compartilham todos os recursos
- Os filhos compartilham um subconjunto de recursos dos pais
- Opções de execução
  - Pai e filhos executam simultaneamente
  - O pai espera até que os filhos terminem

Uma Árvore de Processos em UNIX/Linux



Programa C – Processo Separado de Bifurcação

```
#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;
}
```

Figure 3.8 Creating a separate process using the UNIX `fork()` system call.

Programa C – Processo Separado de Bifurcação

- Espaço de endereço
  - Duplicata filha do pai
  - A criança tem um programa carregado nela
- Exemplos UNIX
  - A chamada de sistema fork() cria um novo processo
  - A chamada de sistema exec() é usada após um fork() para substituir o espaço de memória do processo por um novo programa

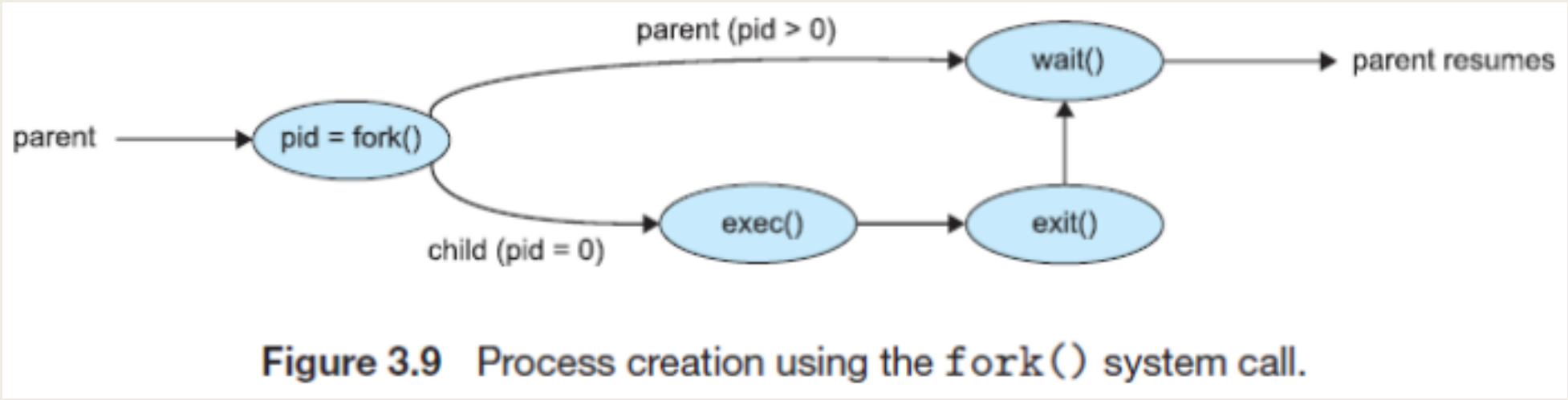


Figure 3.9 Process creation using the `fork()` system call.

Rescisão do Processo

- O processo executa a última instrução e então pede ao sistema operacional para excluí-la usando a chamada de sistema exit().
  - Retorna dados de status (normalmente um número inteiro) do filho para o pai (via chamada de sistema wait())
  - Os recursos do processo são desalocados e recuperados pelo sistema operacional (memória física e virtual, arquivos abertos, buffers de E/S)
- O pai pode encerrar a execução dos processos filhos usando a chamada de sistema abort(). Algumas razões para fazer isso:
  - A criança excedeu os recursos alocados
  - A tarefa atribuída ao filho não é mais necessária
  - O pai está saindo e os sistemas operacionais não permitem que um filho continue se o pai terminar

Rescisão do Processo

- Alguns sistemas operacionais não permitem a existência de filho se seu pai tiver terminado. Se um processo terminar, todos os seus filhos também deverão ser finalizados.
  - **Rescisão em cascata**: Todos os filhos, netos, etc. são rescindidos.
  - O encerramento é iniciado pelo sistema operacional.
- O processo pai pode aguardar o encerramento de um processo filho usando a chamada de sistema wait(). A chamada retorna informações de **status** e o pid do processo encerrado

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

- Se nenhum processo pai esperando (não invocou wait()) é um **zumbi**
- Se o pai for encerrado sem invocar wait , o processo será **órfão** (“*orphan*”)

Prática:
Comandos UNIX/Linux

Pico-Dicionário de comandos UNIX / Linux

/* DESENVOLVIMENTO C */

gcc	Compilador C e C++
gdb	Depurador C e C++

/* PROCESSOS */

ps -ef	status do processo
pstree	árvore de processo
top	processa informações

/* INFORMAÇÃO DO SISTEMA */

uname	informação do sistema
df	informações do sistema de arquivos

/* MANUSEIO DE ARQUIVOS E DIRETÓRIOS */

pwd	caminho para o diretório de trabalho
ls	listar arquivos e diretórios
cd	mudar diretório
mkdir	criar diretório
rmdir	excluir diretório
touch	criar arquivo
cat	leia o arquivo vi editor de texto
cp	copiar arquivos, diretórios
rm	excluir arquivo

/* DISCOS */

du	uso de disco

/* REDE */

ifconfig	configuração de interface

/* DOCUMENTATION */

man	manuais de referência do sistema

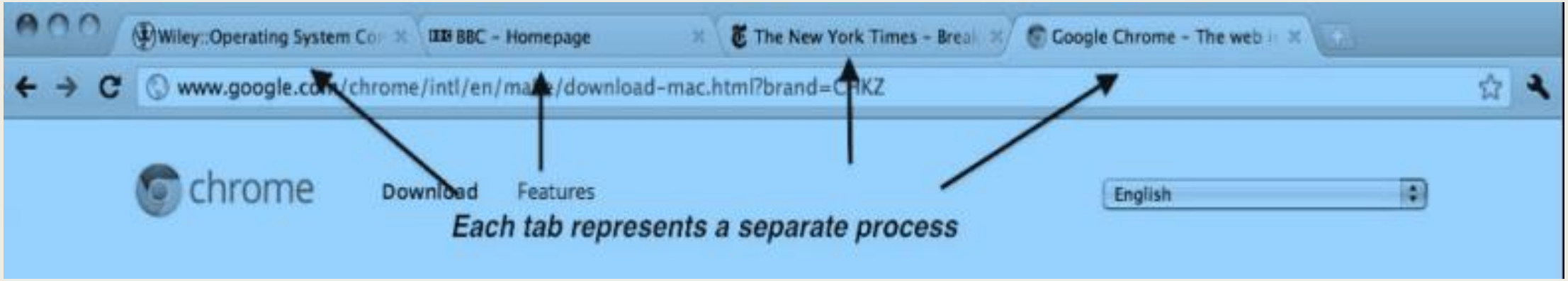
/* Special content */
/* Conteúdo especial */

Multitarefa em Sistemas Móveis

- Alguns sistemas móveis (por exemplo, versões anteriores do iOS) permitem a execução de apenas um processo, outros suspendem
- Devido ao espaço da tela, os limites da interface do usuário, o iOS fornece uma
  - Processo de primeiro plano único ("*foreground*") controlado por meio da interface do usuário
  - Vários processos em segundo plano ("*background*") – na memória, em execução, mas não na tela, e com limites que incluem tarefa única e curta, recebimento de notificação de eventos, tarefas específicas de longa duração, como reprodução de áudio
  - Versões mais recentes do iOS suportam funcionalidades mais ricas para multitarefa com menos restrições devido a melhorias no hardware, capacidade de memória, processamento multicore e duração da bateria
- O Android suporta multitarefa (primeiro e segundo plano), com menos limites
  - O processo em segundo plano usa um **serviço** para executar tarefas
  - O serviço pode continuar em execução mesmo se o processo em segundo plano for suspenso
  - O serviço não tem interface de usuário, usa pouca memória

Arquitetura Multiprocesso – Navegador Chrome

- Muitos navegadores da web rodavam como um processo único (alguns ainda funcionam)
  - Se um site causar problemas, todo o navegador poderá travar ou travar
- O navegador Google Chrome é multiprocessado com 3 tipos diferentes de processos:
  - O processo do navegador gerencia a interface do usuário, disco e E/S de rede
  - Processo de renderização renderiza páginas web, lida com HTML, Javascript. Um novo renderizador criado para cada site aberto
    - É executado em sandbox, restringindo E/S de disco e rede, minimizando o efeito de explorações de segurança
  - Processo de plug-in para cada tipo de plug-in



Visualização/Preview: Comunicação entre processos

Comunicação entre Processos

- Os processos dentro de um sistema podem ser **independentes** ou **cooperantes**
- Processo **independente** não pode afetar ou ser afetado pela execução de outro processo
- O processo de **cooperação** pode afetar ou ser afetado por outros processos, incluindo o compartilhamento de dados
- Vantagens dos processos cooperativos:
  - Compartilhamento de informações
  - Aceleração de computação
  - Modularidade
- Os processos de cooperação precisam de comunicação entre processos (IPC)
- Dois modelos de IPC
  - Memoria compartilhada (“*Shared memory*”)
  - Passagem de mensagens (“*Message passing*”)

Modelos de Comunicação

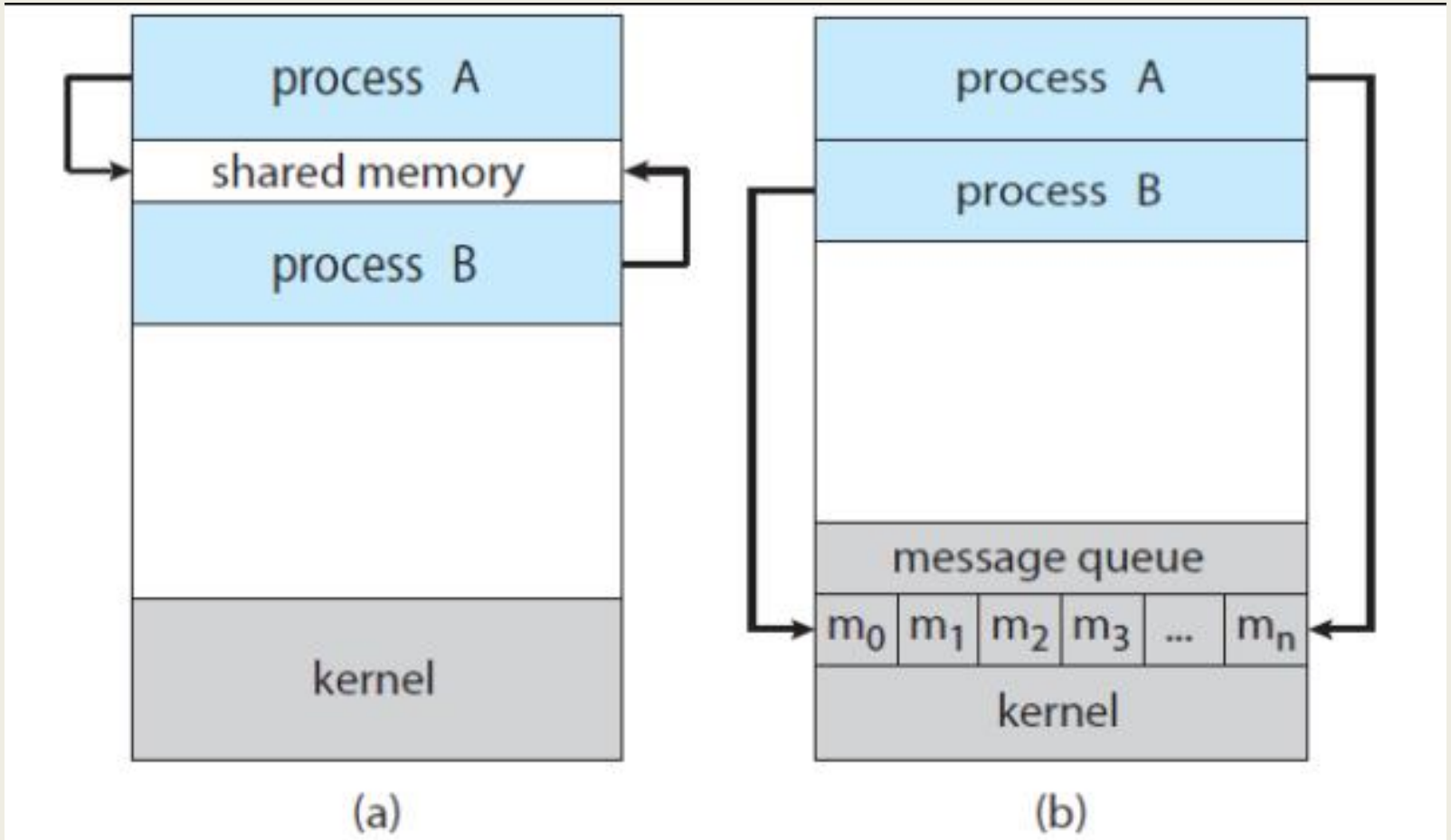


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

Problema Produtor-Consumidor

- Paradigma para processos cooperativos, o processo produtor produz informações que são consumidas por um processo consumidor
- O buffer ilimitado (“*Unbounded-buffer*”) não impõe limites práticos ao tamanho do buffer
- O buffer limitado (“*Bounded-buffer*”) assume que existe um tamanho de buffer fixo

Resumo

Resumo

- Um processo é um programa em execução, que forma a base de toda computação
- Foram introduzidas operações com processos:
  - Criação (“*Creation*”)
  - Terminação (“*Termination*”)
- Pratique em código C realizando operações em processos usando chamadas de sistema