

Sistemas Operacionais

Conceito de Processos

Processos

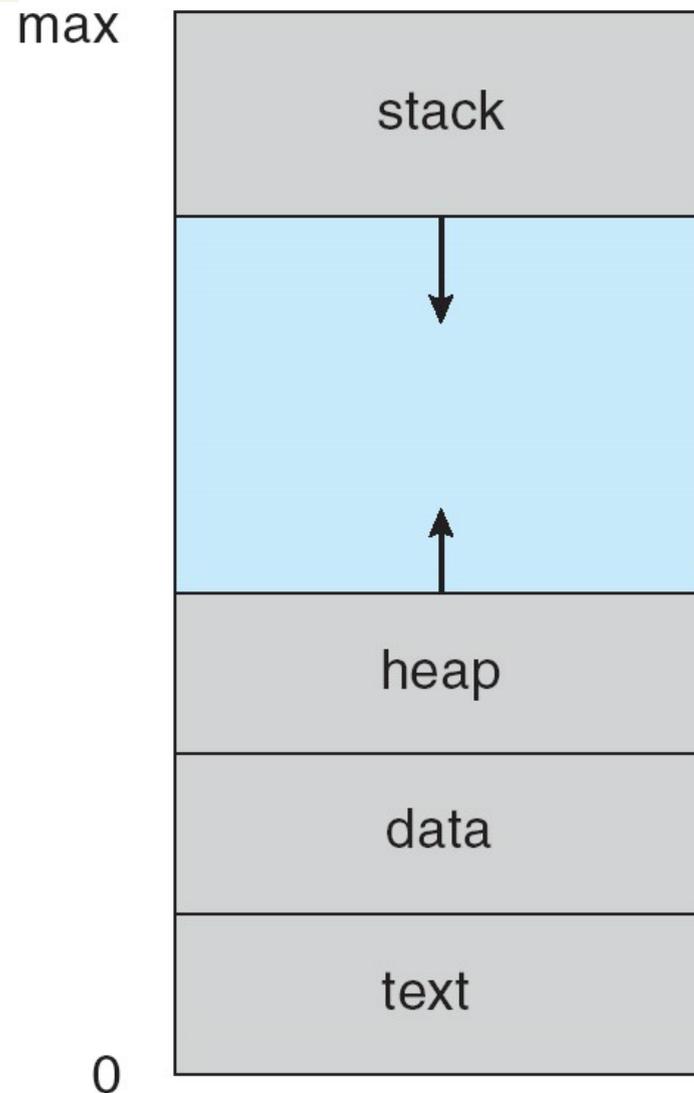
- Conceito de processo
- Estado de um processo
- Troca de contexto e PCB
- Fila de Processo
- Escalonador
- Comunicação entre processos

Processo

- Um sistema operacional executa diversos programas
 - Sistemas batch – jobs
 - Sistemas compartilhados no tempo – programas ou tarefas do usuário
- Normalmente job e processo são sinônimos
- Processo – um programa em execução; a execução do processo deve progredir de modo sequencial
- Um processo inclui, entre outras coisas:
 - contador de programa
 - pilha
 - seção de dados



Na memória

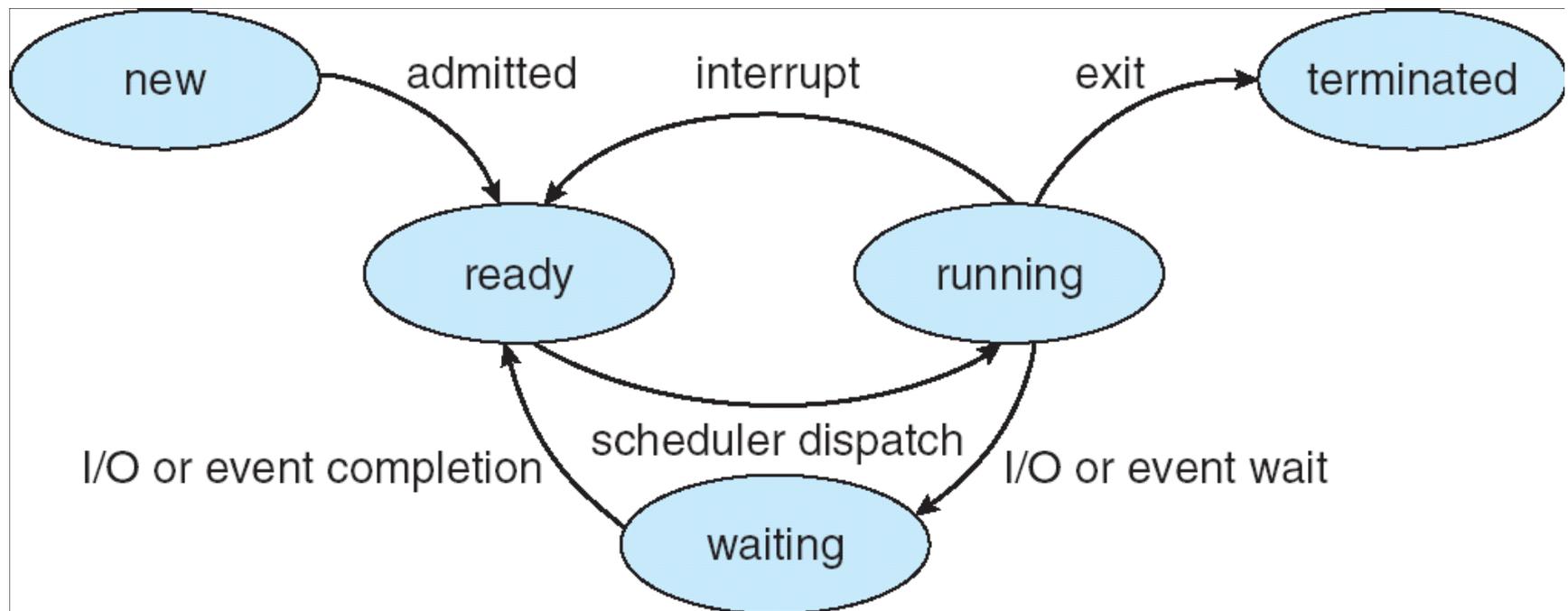


Estados de um processo

- O processo pode estar no estado:
 - Novo
 - O processo está sendo criado
 - Executando
 - Instruções estão sendo executadas
 - Esperando
 - O processo está esperando que ocorra algum evento
 - Pronto
 - O processo está esperando para ser atribuído a um processo
 - Terminado
 - O processo terminou a execução



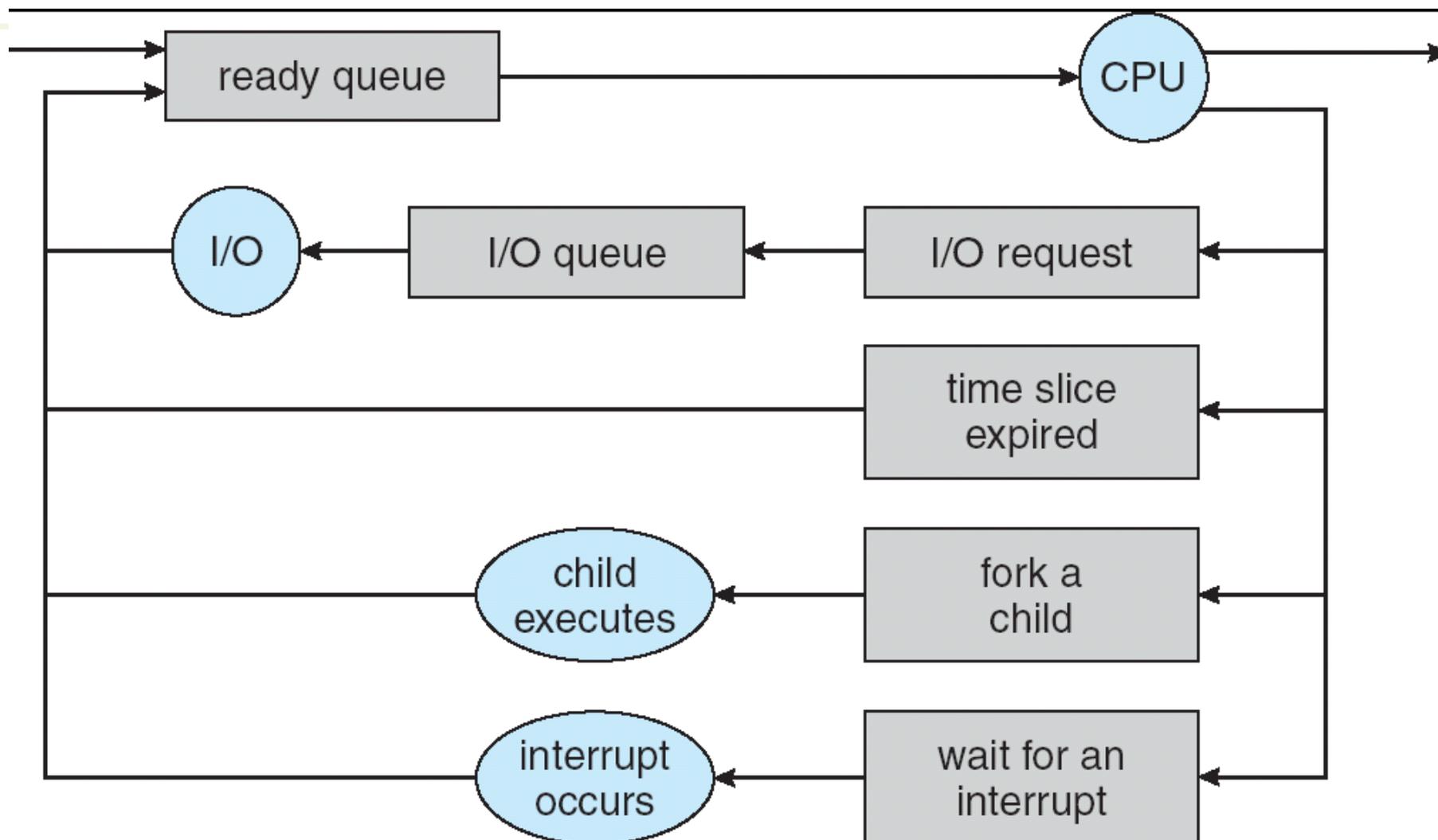
Estados de um processo



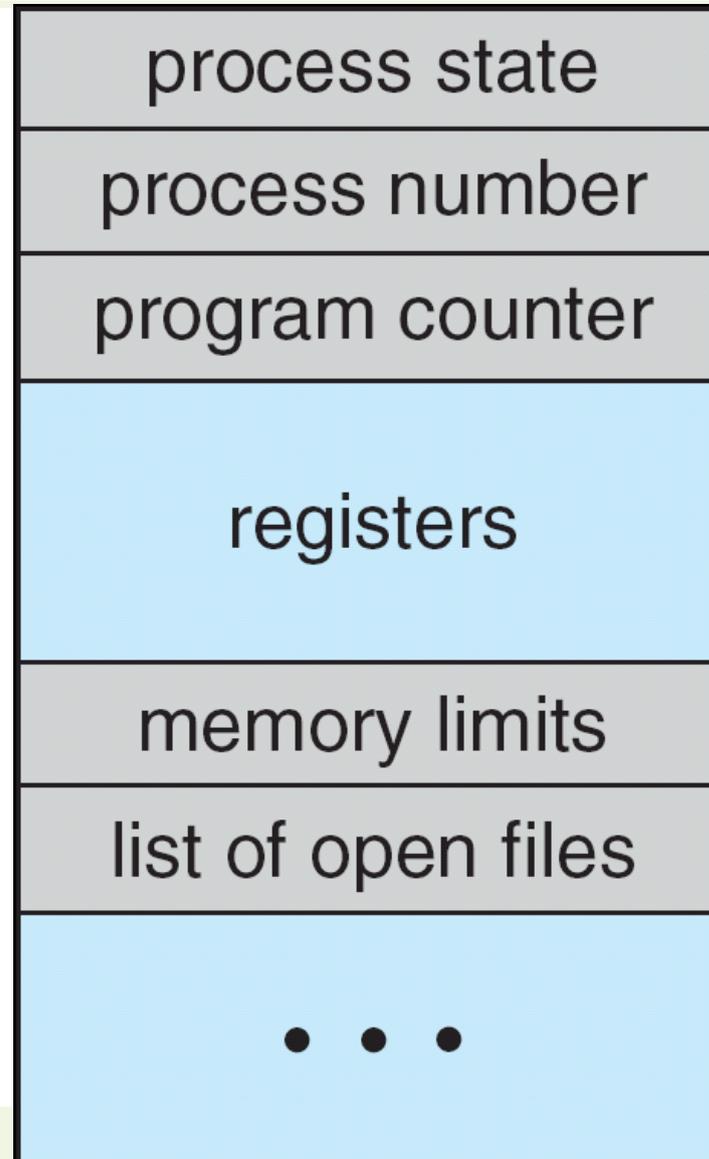
Troca de Contexto

- Quando a CPU passa para outro processo, o sistema deve salvar o estado do processo antigo e carregar o estado salvo para o novo processo
- O tempo de troca de contexto é *overhead*; o sistema não realiza trabalho útil enquanto faz a troca
- Tempo dependente do suporte do hardware

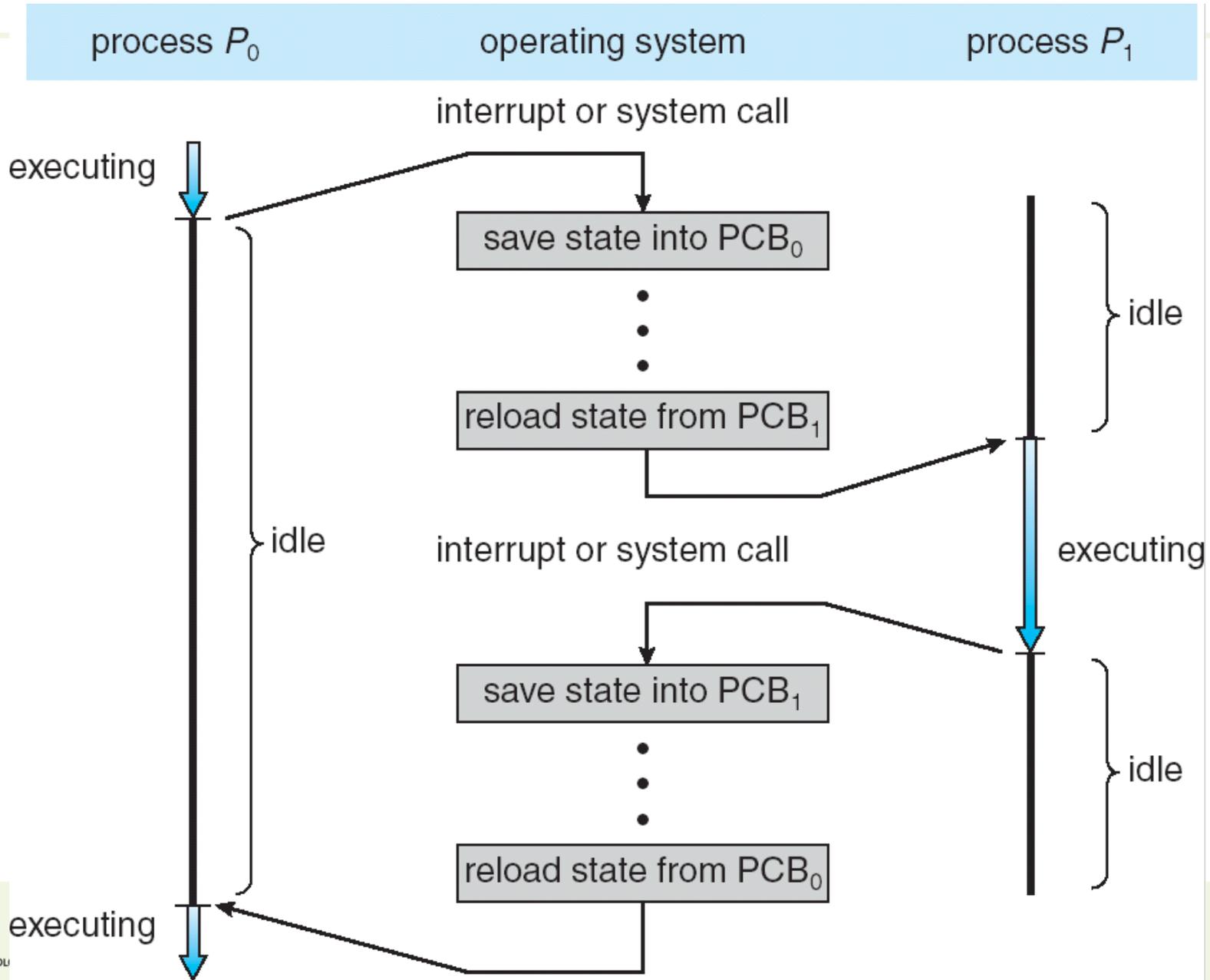
Troca de Contexto quando?



Process Control Block



Troca de Contexto



Process Control Block

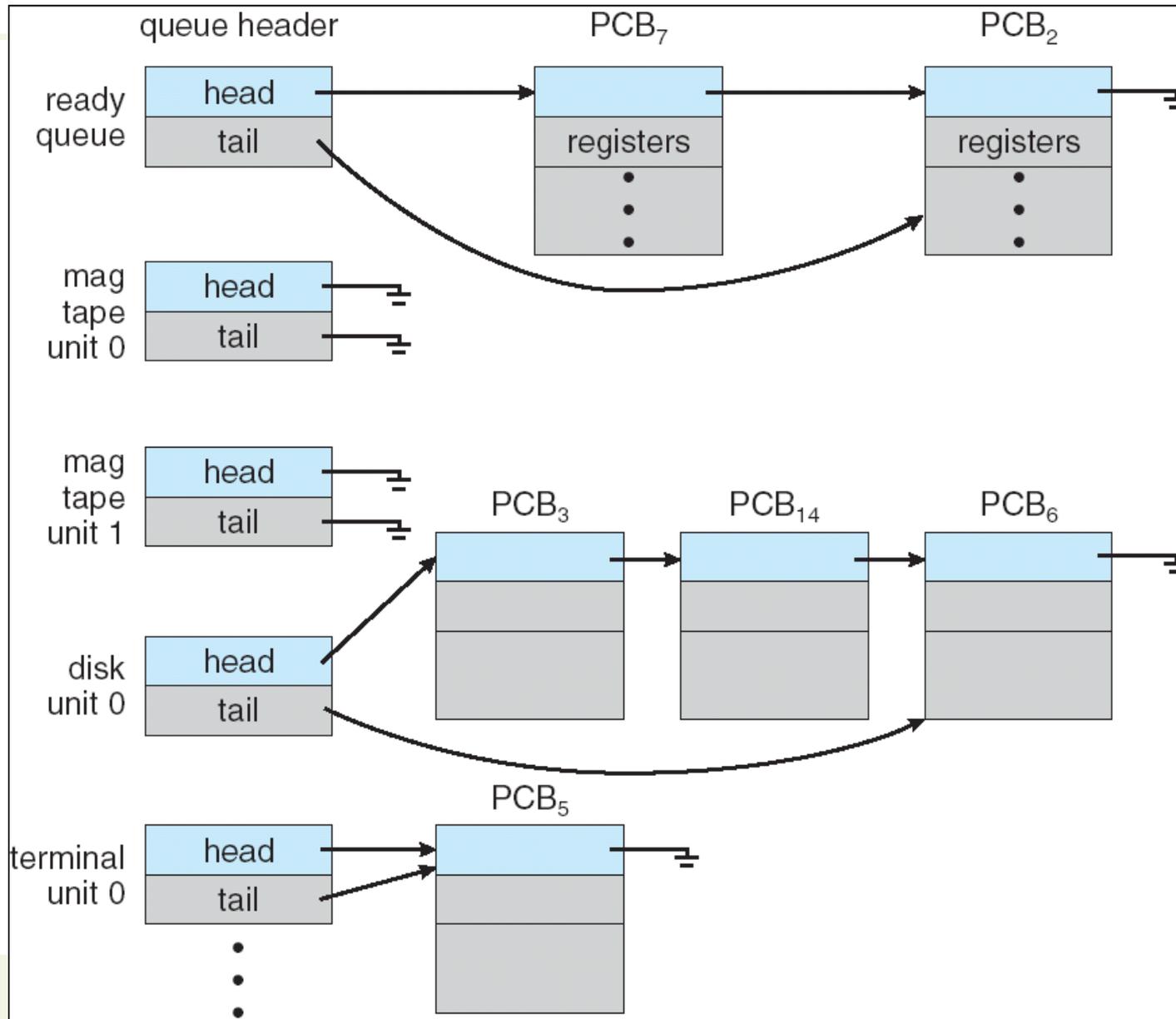
- Informações associadas a cada processo
 - Estado do processo
 - Contador de programa
 - Registradores da CPU
 - Informação de escalonamento da CPU
 - Informação de gerenciamento de memória
 - Informação de contabilidade
 - Informação de status de E/S

Filas de Processos

- Fila de job – conjunto de todos os processos no sistema
- Fila de pronto – conjunto de todos os processos residindo na memória principal, prontos e esperando para execução
- Filas de dispositivo – conjunto de processos esperando por um dispositivo de E/S
- Processos migram entre as diversas filas



Fila de Processos



Escalonador

- Escalonador é um programa/algoritmo responsável pela ordenação das “filas” de processos
- Tipos de escalonadores
 - Escalonador a longo prazo (ou escalonador de job) – seleciona quais processos devem ser trazidos para a fila de pronto
 - Escalonador a curto prazo (ou escalonador de CPU) – seleciona qual processo deve ser executado em seguida e aloca CPU



Escalonador

- O escalonador a curto prazo é invocado muito freqüentemente (milissegundos)
- O escalonador a longo prazo é invocado muito infreqüentemente (segundos, minutos)
- O escalonador a longo prazo controla o grau de multiprogramação

Escalonador

- Os processos podem ser descritos como:
 - Processos voltados para E/S
 - Gasta mais tempo realizando E/S do que cálculos, com bursts de CPU muito curtos
 - Processos voltados para CPU
 - Gasta mais tempo realizando cálculos; poucos bursts de CPU muito longos

Criação de Processos

- Processo pai cria processos filho que, por sua vez, criam outros processos, formando uma árvore de processos
 - Compartilhamento de recursos
 - Pai e filhos compartilham todos os recursos
 - Filhos compartilham subconjunto dos recursos do pai
 - Pai e filho não compartilham recursos
- Execução
 - Pai e filhos executam simultaneamente
 - Pai espera até que filhos terminem

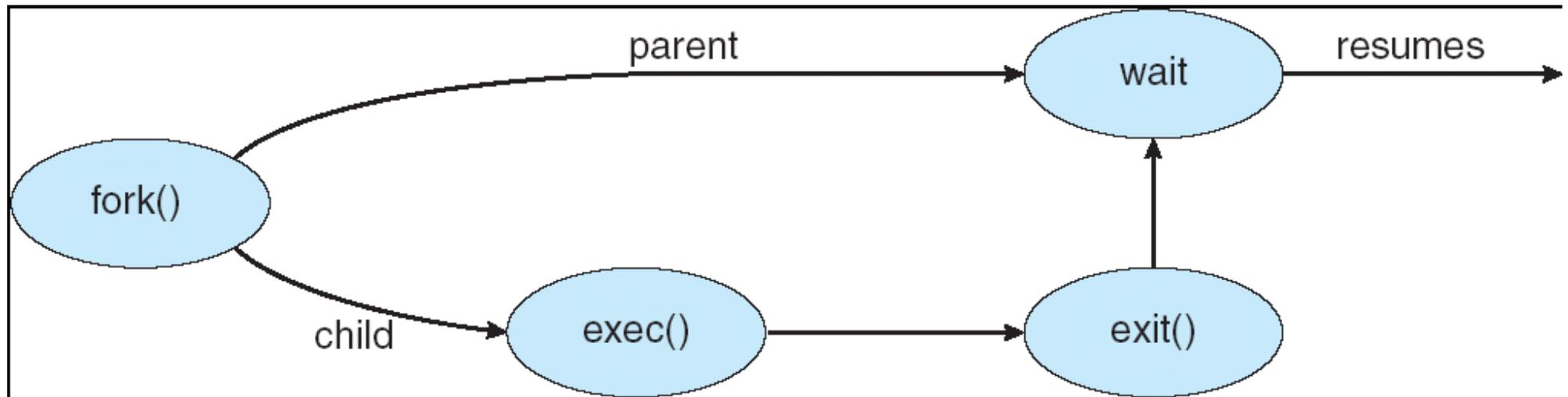


Criação de Processos

- Espaço de endereços
 - Filho duplica do pai
 - Filho tem um programa carregado
- Exemplos do UNIX
 - Chamada do sistema *fork* cria novo processo
 - Chamada do sistema *exec* usada após um *fork* para substituir o espaço de memória do processo por um novo programa



Criação de Processos



Criar processos POSIX

```
#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");
        exit(-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");
        exit(0);
    }
}
```



Criar Processos 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 line
        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);
}
```



Criar Processos em Java

```
import java.io.*;

public class OSProcess
{
    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.err.println("Usage: java OSProcess <command>");
            System.exit(0);
        }

        // args[0] is the command
        ProcessBuilder pb = new ProcessBuilder(args[0]);
        Process proc = pb.start();

        // obtain the input stream
        InputStream is = proc.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        // read what is returned by the command
        String line;
        while ( (line = br.readLine()) != null)
            System.out.println(line);

        br.close();
    }
}
```



Término de um processo

- Processo executa última instrução e pede ao sistema operacional para excluí-la (exit)
 - Dados de saída do filho para o pai (via wait)
 - Recursos do processo são desalocados pelo sistema operacional
- Pai pode terminar a execução dos processos dos filhos (abort)
 - Filho excedeu recursos alocados
 - Tarefa atribuída ao filho não é mais exigida
- Se o pai estiver saindo
 - Alguns sistemas operacionais não permitem que o filho continue se o pai terminar
 - Todos os filhos terminam – término em cascata



Atividade

- Para próxima semana
 - Comentar o código para criação de processos em Java e no Windows
 - Linha a linha!
 - Compilar e Executar os códigos
 - Entregar códigos comentados
 - Descrever:
 - Roteiro para compilação
 - Execução dos códigos
 - Ferramentas usadas
- Endereço para entrega: <https://goo.gl/NzQsR6>**

