Universidade Federal de Uberlândia
Faculdade de Computação
Sistemas Operacionais

Gerenciamento de Processos

Prof. Dr. Marcelo Zanchetta do Nascimento

Sumário

- Conceito de processo
- Ciclo de Vida
- Bloco de controle de processo
- Escalonamento
- Criação de processo
- Término de processo
- Cooperação entre processos
- Leituras Sugeridas

Introdução

- Um SO executa muitos programas:
  - Sistema Batch – jobs;
  - Sistema *Time-sharing* – programas de usuário ou tarefa.
- Permite que múltiplos programas sejam carregados na memória e executados de forma concorrente (SO multiprogramado).

Torna mais eficiente o aproveitamento dos recursos do computador

Processo

- É uma abstração que representa um programa em execução;
- É uma entidade ativa que utiliza um conjunto de recursos, como processador e registradores especiais, para executar uma função;
- É também referenciado como "tarefa" (task) ou "job".

Exemplo: cópia de arquivo ou execução de uma rotina de um programa.

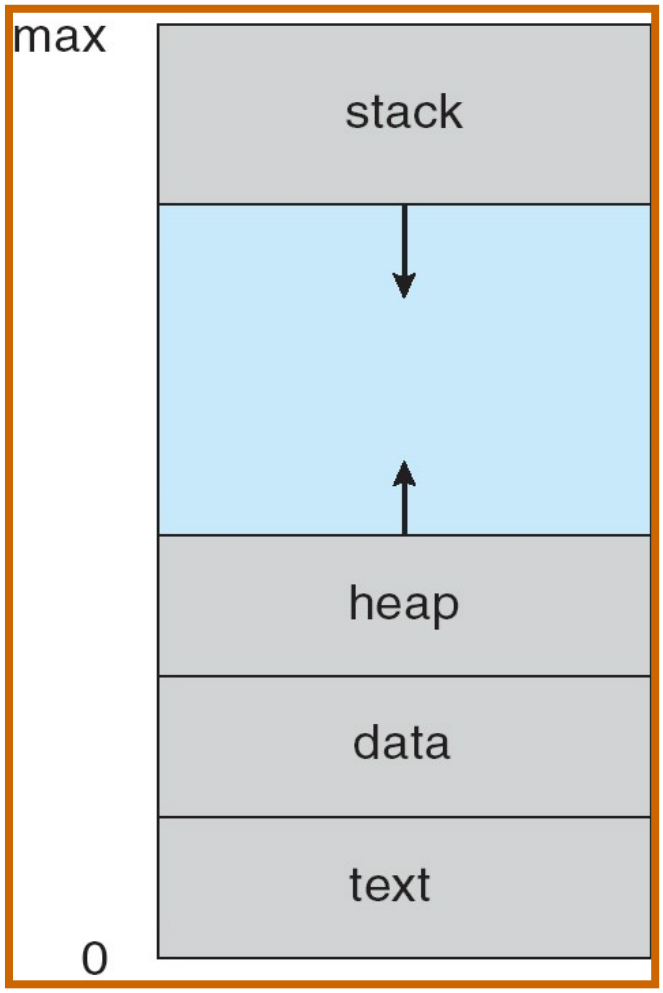


Figura 1: Estrutura de um Processo

Um programa na memória

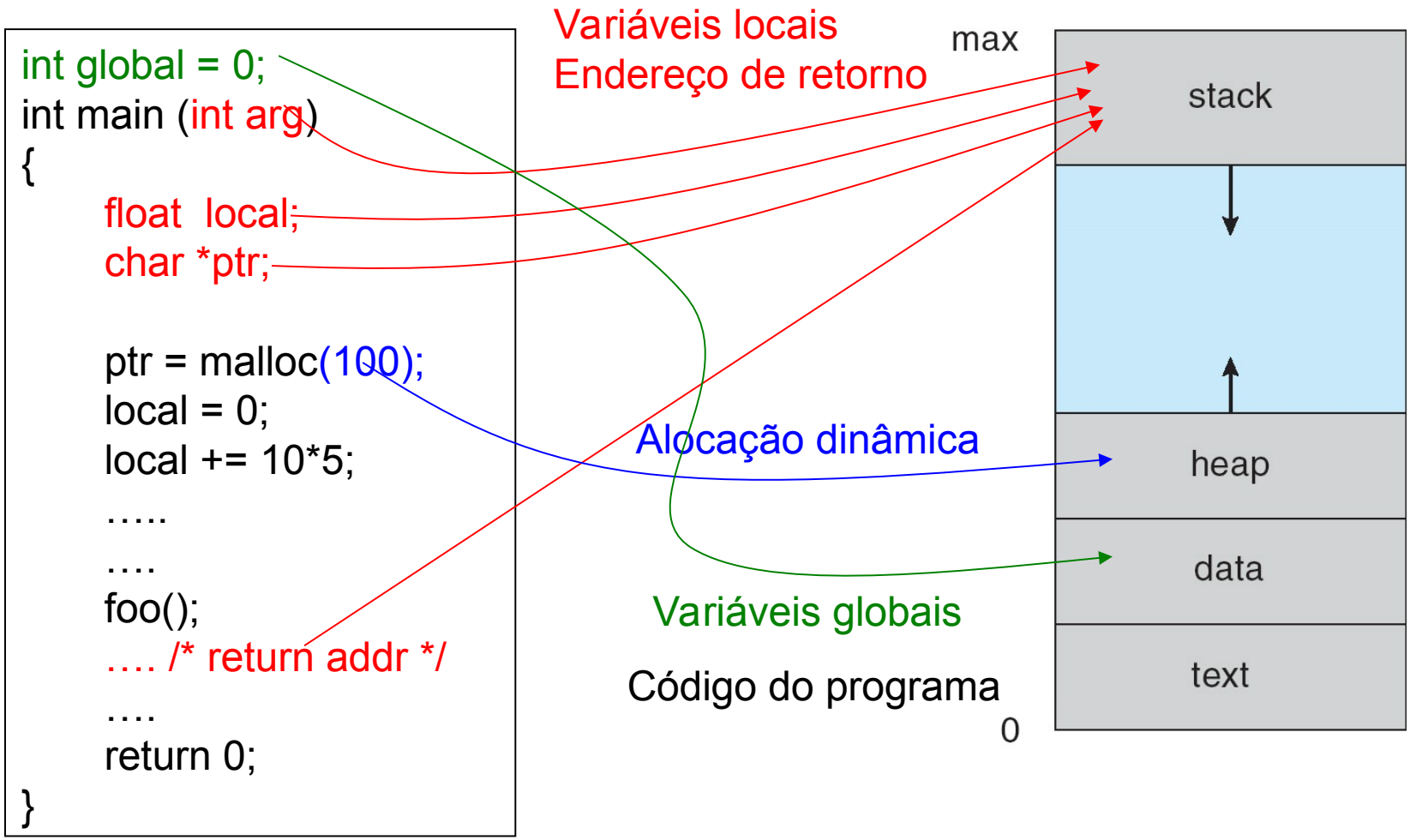


Figura 2: Exemplo de um programa e suas estruturas em um processo

CPU e o Processo

Na CPU, um processo executa instruções do seu repositório em alguma sequência ditada pelos valores do registrador CP (contador de programa)

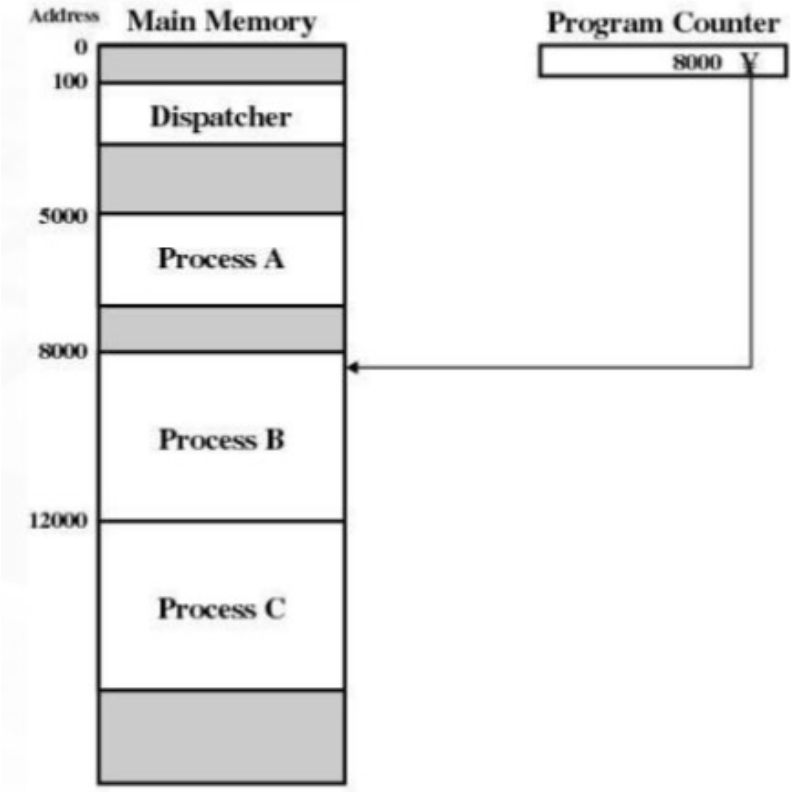


Figura 3: Exemplo de um programa e instruções em parte de uma CPU

Ciclo de vida do processo

Estado: Durante a sua execução, um processo passa por diversos estados, refletindo o seu comportamento dinâmico, isto é, a sua evolução no tempo.

- **Novo:** o processo esta sendo criado;
- **Execução:** o processo esta utilizando um processador;
- **Pronto:** o processo está apto a utilizar o processador quando este estiver disponível;
- **Bloqueado:** o processo está esperando ou utilizando um recurso qualquer de E/S;
- **Encerrado:** processo termina sua execução.

Diagrama de Estado de um Processo

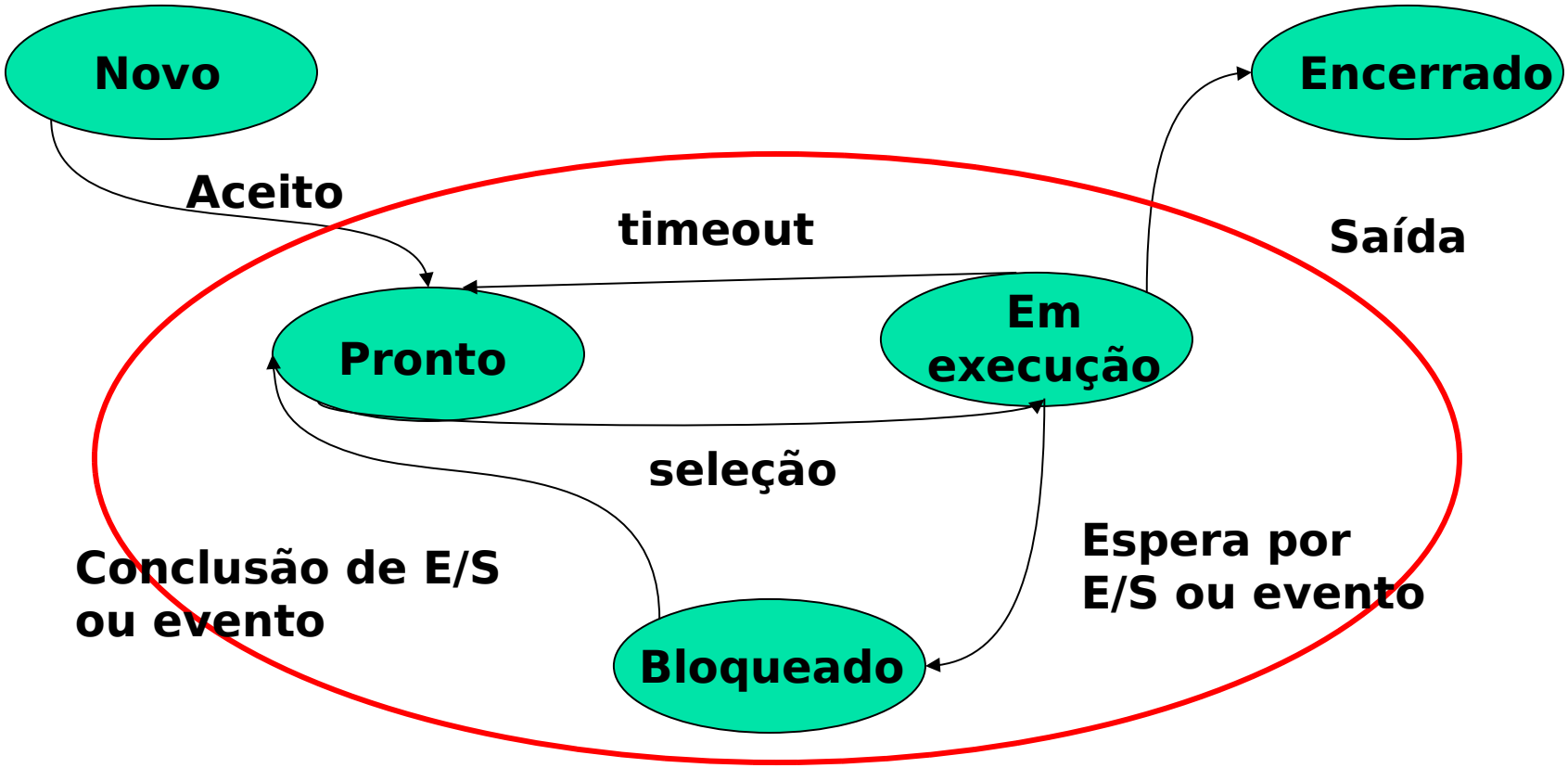


Figura 4: Ciclo de vida de um processo

Ciclo de vida do processo

- **Null → New**: Um novo processo é criado para executar o programa.
  - Novo job.
  - Login interativo (usuário se conecta ao sistema).
  - SO cria processo para prover um serviço (ex: impressão).
  - Processo cria um outro processo.

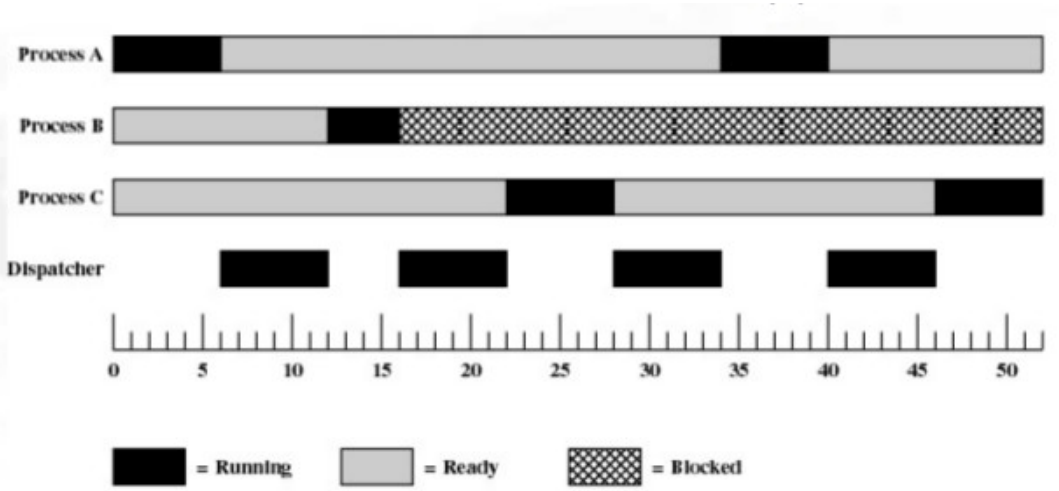
Ciclo de vida do processo

- **New → Ready:** No estado New (novo), recursos foram alocados pelo SO mas não existe um compromisso de que o processo será executado.
  - Número de processos já existentes;
  - Quantidade de memória virtual requerida;
  - Manter um bom desempenho do sistema é o fator limitante da criação de novos processos.

Ciclo de vida do processo

- **Ready → Running:**
  - Definido pela política de escalonamento de processos adotada pelo SO;
- **Running → Exit:**
  - Processo terminou as suas atividades ou abortado;
  - Término normal;
  - Término do processo pai (em alguns sistemas);
  - Excedeu o limite de tempo;
  - Memória não disponível;
  - Execução de instrução inválida ou de instrução privilegiada no modo usuário;
  - Intervenção do SO(Ex: ocorrência de deadlock).

Diagrama de tempo de execução de processos



1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
----------	Time out	
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
----------	I/O request	
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
----------	Time out	
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
----------	Time out	
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
----------	Time out	

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figura 4: Ciclos dos processos em troca de contexto na CPU

Bloco de controle de processo

- Bloco de controle do processo (BCP) (process control block)
  - é a representação de um processo num SO (repositório de informações).
- Contém informações necessárias para a execução do processo: Iniciada, Interrompida e Retomada.
- Normalmente, há uma grande estrutura no kernel:
  - Exemplo: Linux: **struct task_struct;**
  - **Usado para gerenciar processos:** em comutação da CPU.

Bloco de controle de processo

- **Estado do processo:** novo, pronto, em execução, parado, etc

- **Contador de programa:** endereço da próxima instrução.

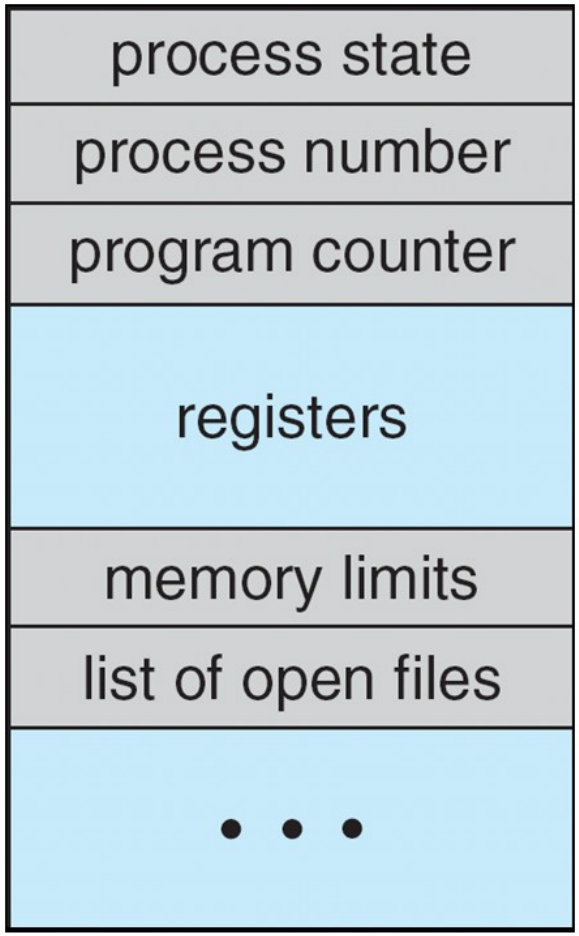
- **Registradores da CPU:**

  - variam em número e em tipo, dependendo da arquitetura de computadores.

  - Incluem acumuladores, registradores e pilha.

  - Quando ocorre uma interrupção, essa informação deve ser salva juntamente com o contador do programa.

Bloco de controle de processo



- estado corrente do processo;
- identificação do processo (PID);
- ponteiro para o processo-pai (*parent process*);
- prioridade do processo;
- lista de ponteiros para as regiões alocadas de memória;
- conteúdo dos registradores do processador.

Essas informações são importantes para a troca de processos que ocorre na CPU

Figura 5: Espaço de um Processo

BCP - Comutação de Contexto

- A troca de contexto (processos) pode gerar overhead.

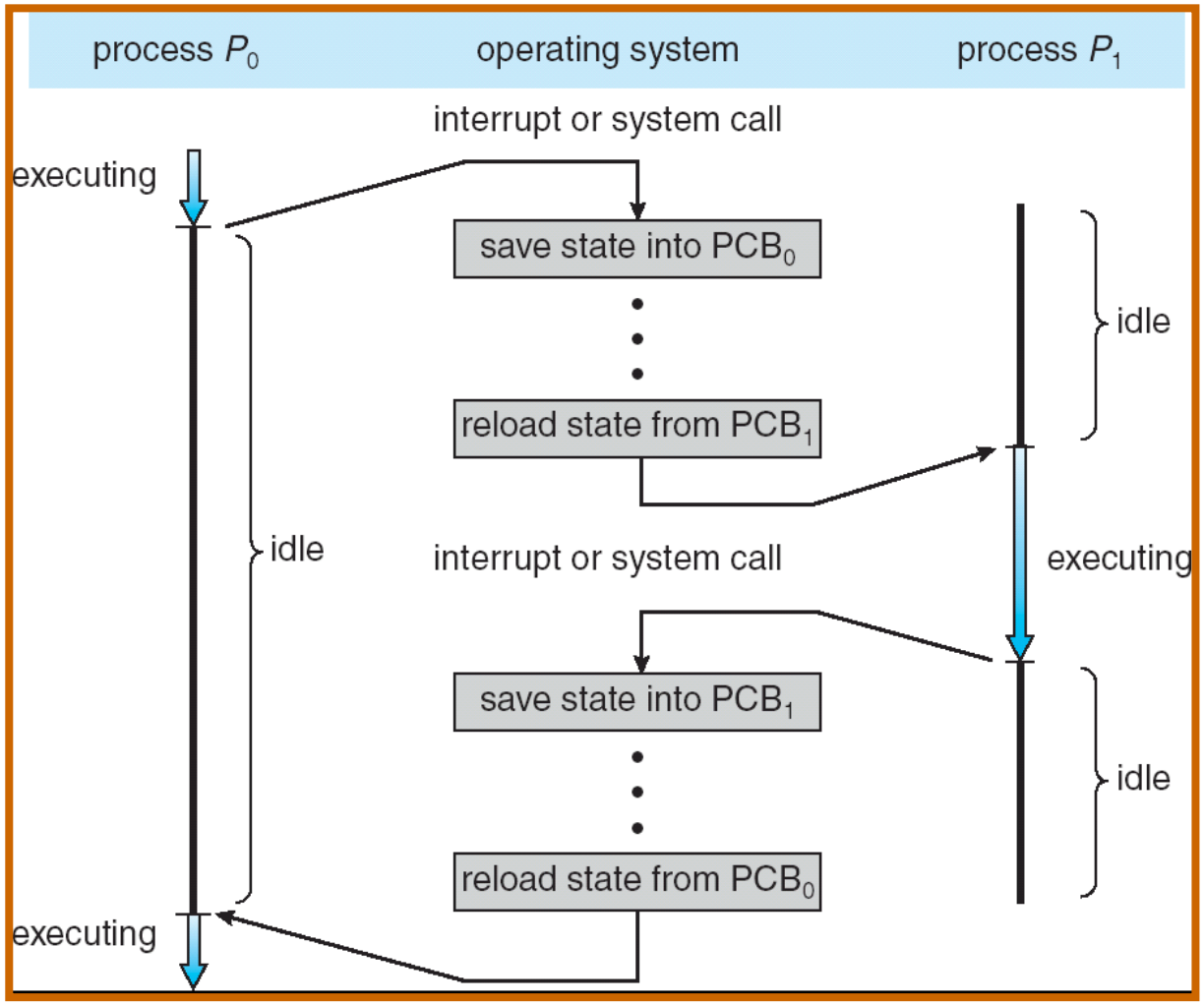


Figura 6: Exemplo de troca de contexto entre 2 processos em um sistema

BCP - Comutação de Contexto

- O tempo de comutação de contexto pode ser pura sobrecarga;
- **Nenhum trabalho será realizado;**
- O tempo de comutação depende bastante do **suporte de hardware**;
- **O Escalonador (“Scheduler”):**
  - Módulo do S.O. responsável pelo controle do recurso “processador”, o qual divide o tempo da UCP entre os processos do sistema.

Tipos de Escalatores

- **Scheduler de longo prazo** (job scheduler) – seleciona qual processo deverá ser carregado da memória para execução.
- **Scheduler de curto prazo** (or CPU scheduler) – seleciona um dos processos prontos para execução e aloca a CPU para ele.
- A distinção entre estes dois é a frequência de sua execução.
- Em alguns S.O.s, o **Scheduler de longo prazo** pode não existir ou ser mínimo (ex. UNIX).

Escalonadores

- **Escalonadores de longo prazo (jobs)**
  - Selecionar processos a serem inseridos na fila de pronto;
  - Controla o grau de multiprogramação;
  - Pode ser mais lento e usar mais informação.

- **Escalonadores de curto prazo (CPU)**
  - Selecionar os processos a receber a CPU a cada instante;
  - Chamadas em milesegundos.

Filas dos Escalonadores

- **Fila de Jobs** – consiste em todos os processos que estão no sistema.

- **Fila Pronta** – conjunto de processos residente na memória principal que estão prontos e em espera para entrar em execução. Implementado na forma de lista encadeada.

- **Fila de Dispositivos** – conjunto de processos esperando por um dispositivo de E/S;

- Processos migram entre as várias filas

Fila Prontos e Várias Filas de Dispositivos de I/O

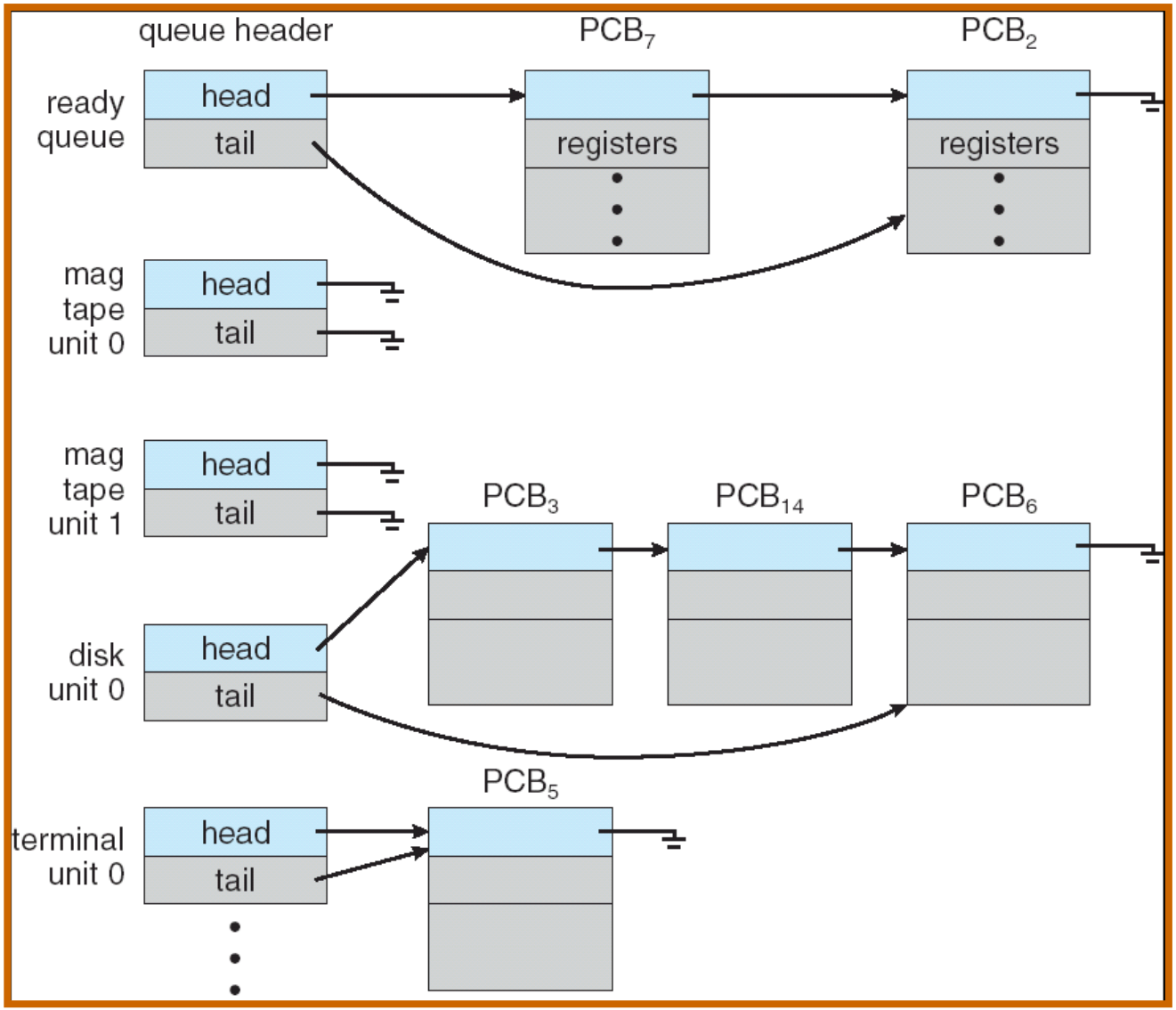


Figura 7: Fila de prontos e várias filas de dispositivos de I/O

Representação do Escalonador de Processos

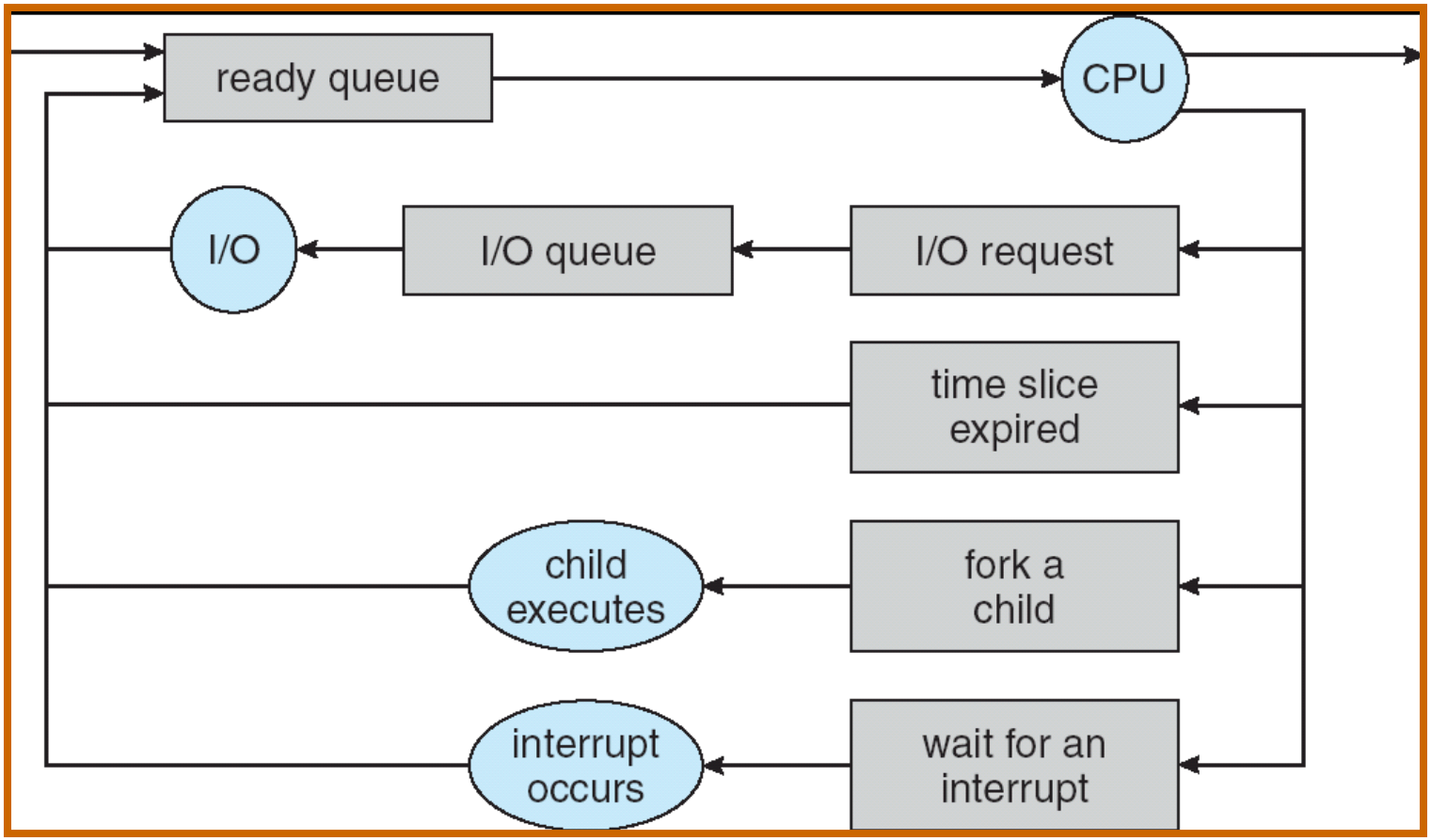


Figura 8: Representação do processo e o escalonador de processos em diagrama de enfileiramento

Tipo de Processos

- Os processos ou tarefas podem ser descritos da seguinte forma:
  - **Processo I/O-bound** – gasta mais tempos fazendo E/S que cálculo, muito pouco tempo de ocupação de CPU:
    - Devolve deliberadamente o controle da CPU.
  - **Processo CPU-bound** – gasta mais tempo fazendo cálculo, longo tempo de CPU:
    - Pode monopolizar a CPU, dependendo do algoritmo de escalonamento.

Operações sobre Processos

- **Unix:** o processo pai cria o processo filho, o qual, pode criar outros processos formando uma árvore de processos;
- Execução pode ser:
  - Pai e filho executam concorrentemente;
  - Pai espera até que o filho termine;
- Recursos compartilhados podem ser tratados:
  - Pai e filho **compartilham todos** os recursos;
  - Filho **compartilha um sub conjunto** dos recursos do pai;
  - Pai e filho **não compartilham** recursos.

Uma Árvore de Processes em Linux

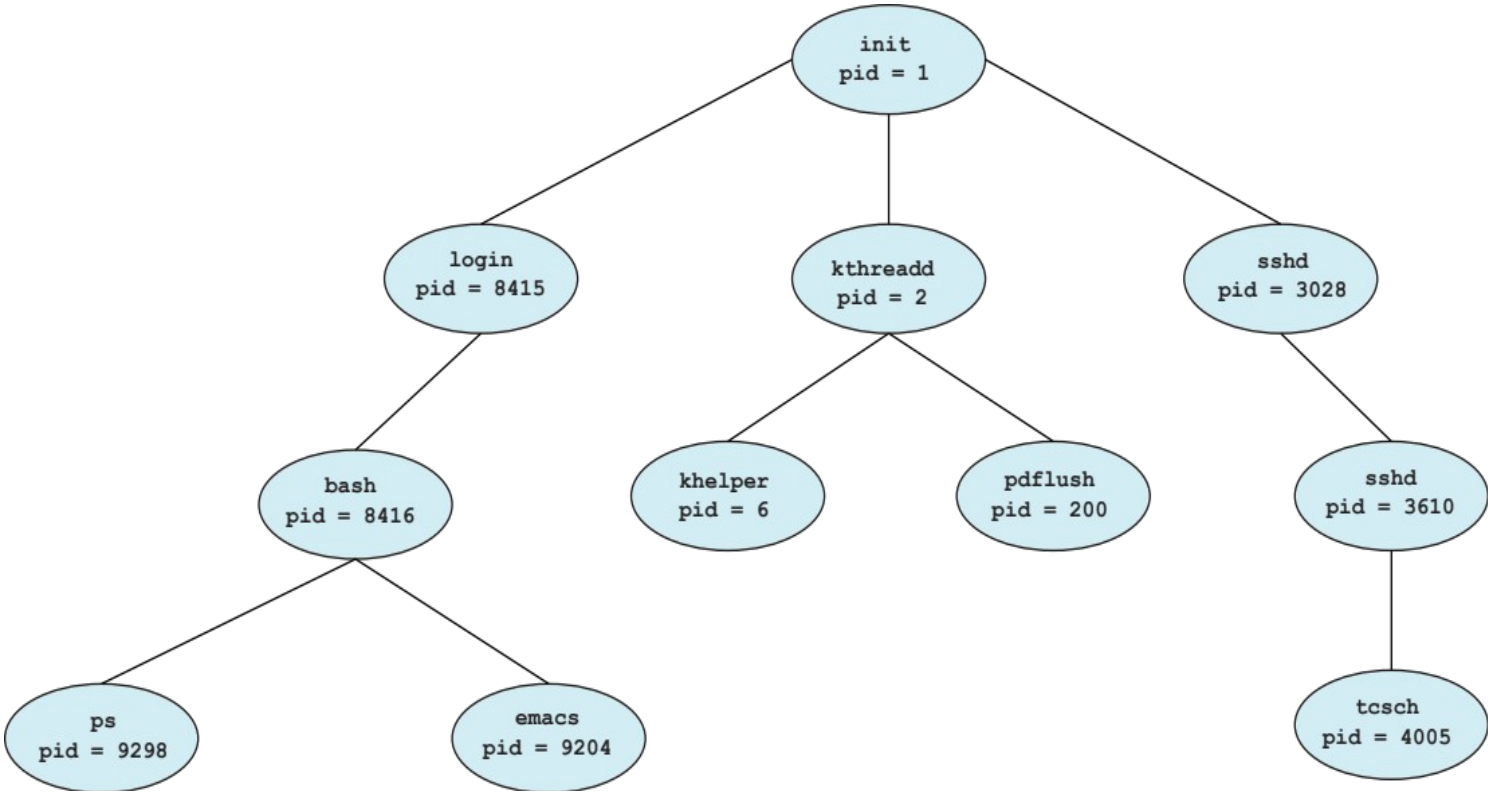


Figura 9: Representação de diversos processos dentro de uma estrutura de um sistema operacional

Criação do Processo: Unix

- Processo cria outro processo (filho) usando a chamada de sistema **fork**
  - Filho é uma cópia do pai;
  - Basicamente, o filho carrega outro programa dentro de seu espaço de endereço usando a chamada **exec;**
  - Pai espera seu filho terminar.

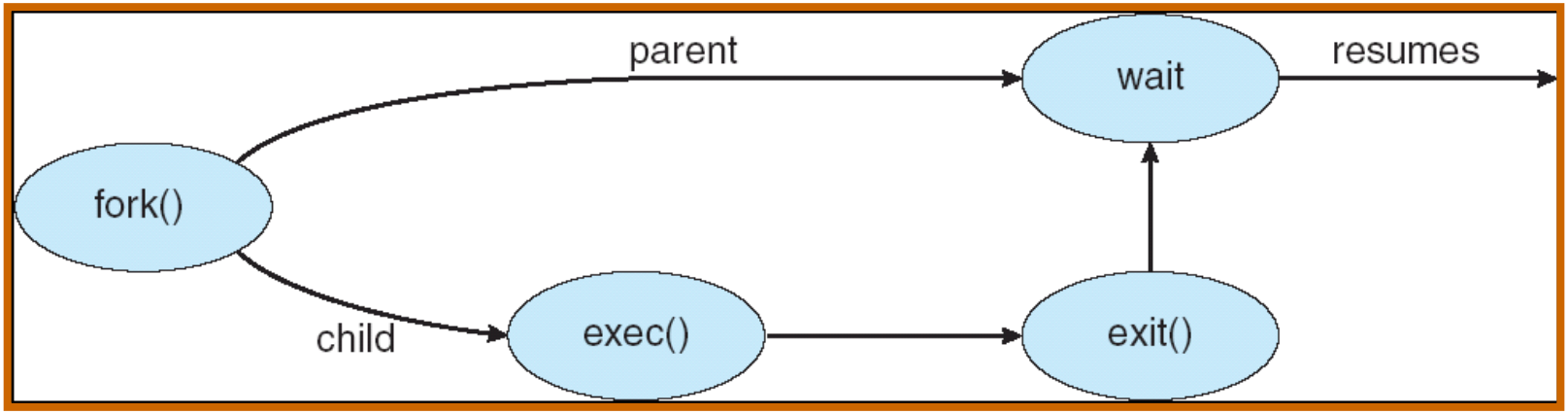


Figura 10: Criação de um processo usando o system call fork.

Criação do Processo: Unix

- fork ( )
  - O SO cria um novo BCP;
  - Cria uma cópia da memória do processo pai;
  - Recursos de E/S são compartilhados;
- Exec ( )
  - O SO busca um programa da memória e o carrega sobre a área do programa que fez a chamada;
  - Execução passa para o ínicio do programa principal carregado.

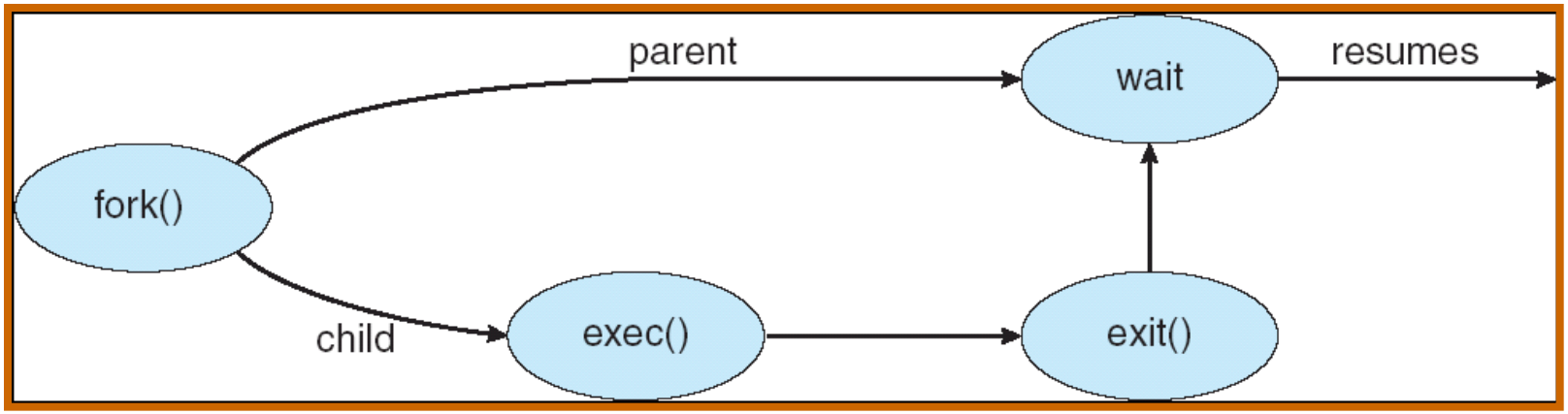


Figura 10: Criação de um processo usando o system call fork.

Criação do Processo: Unix

- wait ( )
  - Se um processo filho do processo Pai ainda não encerrou pela chamada de sistema exit( ), então essa chamada suspende a execução do processo até o filho encerrar;
  - O status de encerramento do filho é retornado no status do argumento de wait( ).
- exit ( )
  - Essa função encerra o processo, retornando todos os recursos (memória, arquivos abertos, etc) usado pelo processo para realocação pelo kernel.

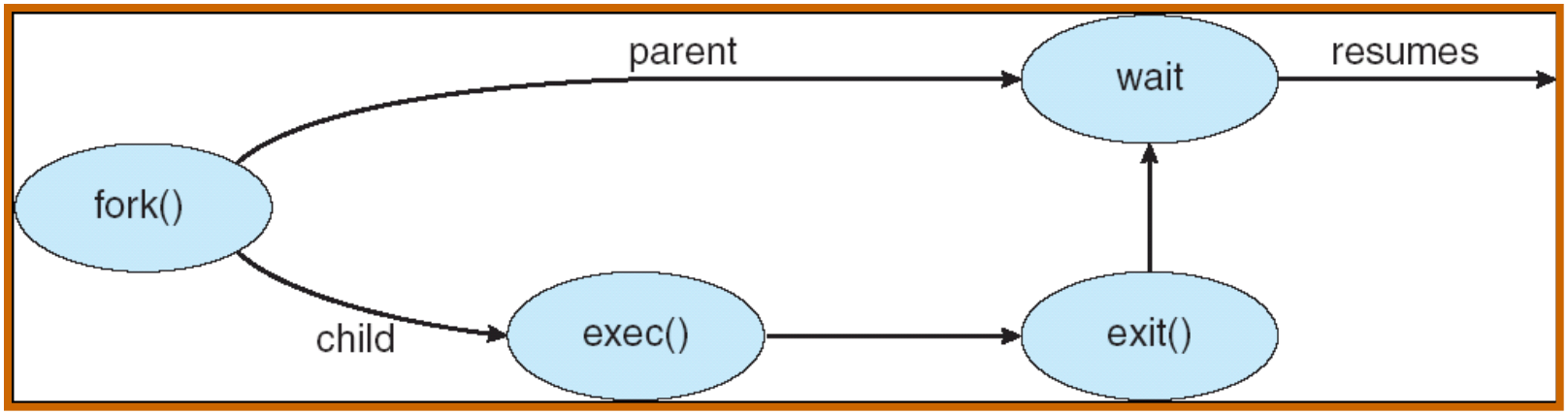


Figura 10: Criação de um processo usando o system call fork.

Programa em C: fork()

```
int main()
{
   pid_t  pid; //variável para representar o ID do processo

   /*returna o ID do processo do filho com sucesso ou –1 um erro*/
   pid = fork(); /* cria um processo */

   if (pid < 0) { /* ocorreu erro */
      fprintf (stderr, "Fork Falhou");
      exit(-1);
   }
   else if (pid == 0) { /* processo filho */
       //subst. a imagem especificada pelo programa ls
       execlp ("/bin/ls", "ls", NULL);}
   else { /* processo pai */
       /* pai espera pelo filho */
       wait (NULL);
       printf ("Filho completo");
       exit(0);
   }
}
```

Programa em C: fork()

- **Unix:** Vamos usar alguns exemplos de códigos para analisar o funcionamento do fork.
- Quais as principais funções dessa chamada de sistema?
- O número de processos criados?

Criação do Processo: Windows

- CreateProcess – biblioteca Win32 possui funções para gerenciamento do processo.
- Pai cria um novo processo filho;
- Requer a carga de um programa especificado no espaço de endereço do processo filho na criação do processo diferente do fork do Unix;
- Necessidade de pelo menos 10 parâmetros para sua ativação.

Criação de um processo via Windows API

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

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

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    // Start the child process.
    if( !CreateProcess( NULL,   // No module name (use command line).
        "C:\\WINDOWS\\system32\\mspaint.exe", // Command line.
        NULL,             // Process handle not inheritable.
        NULL,             // Thread handle not inheritable.
        FALSE,            // Set handle inheritance to FALSE.
        0,                // No creation flags.
        NULL,             // Use parent's environment block.
        NULL,             // Use parent's starting directory.
        &si,              // Pointer to STARTUPINFO structure.
        &pi )             // Pointer to PROCESS_INFORMATION structure.
    )
```

Criação de um processo separado via Windows API

```
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return -1;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}
```

Encerramento de um processo

- Processo é encerrando quando a última chamada do seu programa solicita ao S.O. que o exclua usando a chamada **exit( ):**
  - Retorna um valor de status do filho para o pai (via **wait()**)
  - Todos os recursos do processo são desalocados pelo S.O.
- Processo pai pode terminar a execução de um processo filho usando a chamada **abort( )**. Algumas razões para fazer isso:
  - Filho excedeu o uso de recursos alocados;
  - A tarefa atribuida ao filho não é mais requerida;
  - O Pai esta sendo encerrado e o S.O. não permite que o filho continue sem seu pai terminar.

Encerramento de um processo

- Alguns S.O.s não permitem que o filho continue quando o pai finalizou. Se um processo pai terminar, todos os filhos devem ser finalizados.
    - **Encerramento em cascata:** todos os filhos são encerrados.
- Se o processo filho completou a execução (via chamada de sistema exit) mas ainda possui uma entrada na tabela de processos pode ser um processo zumbi.
- O processo Pai pode esperar para o encerramento do processo filho usando uma *system call* **wait().**
    - A chamada retorna a informação de status e o pid do processo encerrado. **pid = wait(&status).**
    - Se o pai terminar sem usar o **wait,** o processo filho é um processo orfão.

Processo – UNIX e WINDOWS

UNIX	Win32	Descrição
fork	CreateProcess	Cria um novo processo
waitpid	WaitForSingleObject	Espera que um processo termine
execve	-	Substitui a imagem de um processo
exit	ExitProcess	Conclui a execução

Arquitetura multiprocessos – Chrome Browser

Muitos web browsers executam um simples processo

- Se um browser causar problemas, o navegador inteiro pode travar ou falhar.

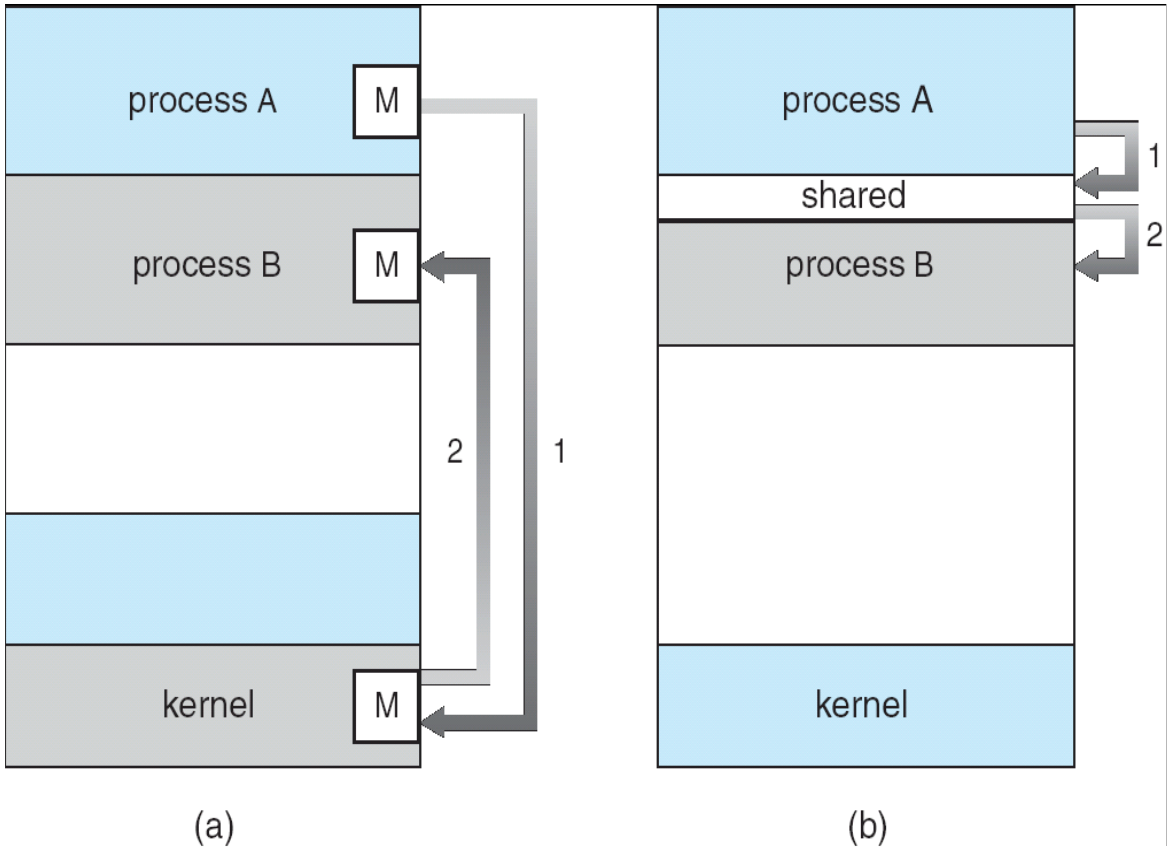
O Google Chrome é uma sistema multiprocessos com 3 tipos:

- **Processo Browser** gerencia interface usuário, disco e acesso a rede;
- **Processo Renderer** processa páginas da web, lida com HTML, Javascript. Um novo processo criado para cada website aberto;
- **Processo Plug-in** para cada tipo de plug-in.

Cooperação entre processos

- Um processo é cooperativo se puder afetar ou ser afetado pelos outros processos em execução no sistema;
- O que motiva essa cooperação?
  - Compartilhamento de informações
  - Agilidade de computação
  - Modularidade
  - Conveniência
- Mecanismo Comunicação interprocessos (IPC)
  - Memória compartilhada
  - Passagem de mensagem

Cooperação entre processos



Passagem de mensagem **Mensagem compartilhada**

Figura 11: Modelos de Comunicação entre Processos.

IPC: Memória compartilhada

- Comunicação entre processos por meio de espaço de memória compartilhada;
  - Um processo cria um espaço de memória;
- Outros processos acessam a memória compartilhada de seu próprio endereço:
  - A memória compartilhada é tratada como área compartilhada;
  - Sincronização é necessário para garantir o acesso concorrente a essa área.
- Vantagens:
  - Rápido (velocidade de acesso a memória)
  - Chamadas realizadas apenas para estabelecer memória compartilhada.
- Desvantagem
  - Precisa gerenciar conflitos de espaço compartilhado;

Problema do Produtor e Consumidor

- Um paradigma para cooperação entre processos:
  - Processo Produtor produz informação;
  - Processo Consumidor consume informação.
- Comunicação por meio de buffer:
  - Buffer com tamanho limitado impõe restrições;
  - Mesmo um buffer com tamanho ilimitado deve caber na memória.

Buffer – Memória Compartilhada

Informações compartilhadas

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

item buffer[BUFFER_SIZE];
int in = 0; %posição livre do buffer
int out = 0; %posição preenchida do buffer
```

O Buffer está cheio quando ((in+1)%BUFFER_SIZE)==out

Só pode usar quando BUFFER_SIZE-1 posições

Buffer – Memória Compartilhada

```
item next_produced;

while (true) {

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

}
```

Buffer – Memória Compartilhada

```
item next_consumed;


while (true) {

    while (in == out); /* do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    /* consume the item in next consumed */

}
```

IPC: Passagem de Mensagem

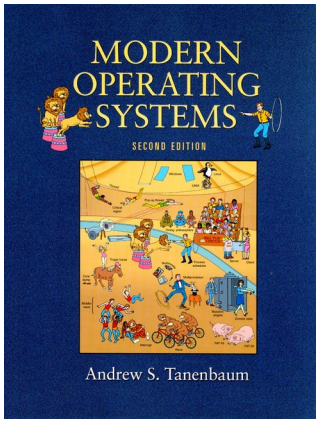
- Se os processos A e B querem se comunicar e precisam:
  - Estabelecer um canal de comunicação entre eles
  - Troca de mensagem por meio de:
    - **send** – tamanho de mensagem fixa ou variável
    - **receive –** mensagem (*message*)
- Vantagem:
  - Não há conflitos: ótimo para troca de mensagens especialmente em sistema distribuído.
- Desvantagens
  - Sobrecarga (cabeçalho da mensagem)
  - Kernel: ocorre vários *system calls* para troca de mensagem

IPC: Passagem de Mensagem

- **O canal de comunicação pode ser**
  - **Direto**: precisa nomear explicitamente o destinatário ou emissor:
    - **send** (*P, mensagem*) – envia uma mensagem para o processo P;
    - **receive** (*Q, mensagem*) – recebe uma mensagem do processo Q.
  - **Indireto**: comunicação via mailboxes (ou portas)
    - Messagem enviada e recebida de mailboxes
    - Cada mailbox tem um único ID
      - **Send** (*A, mensagem)* – envia mensagem para mailbox A
      - **Receive** (*A, mensagem*) – recebe mensagem do mailbox A

Leituras Sugeridas

- Andrew S. Tanembaum. Sistemas Operacionais. Modernos. 2ª Ed. Editora Pearson, 2003.
- Capítulo 2



- Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg. **Fundamentos de sistemas operacionais.** 6 ed. Rio de Janeiro: LTC, 2009.
- Capítulo 3

- Lista de Exercícios no Moodle: 40 - 56