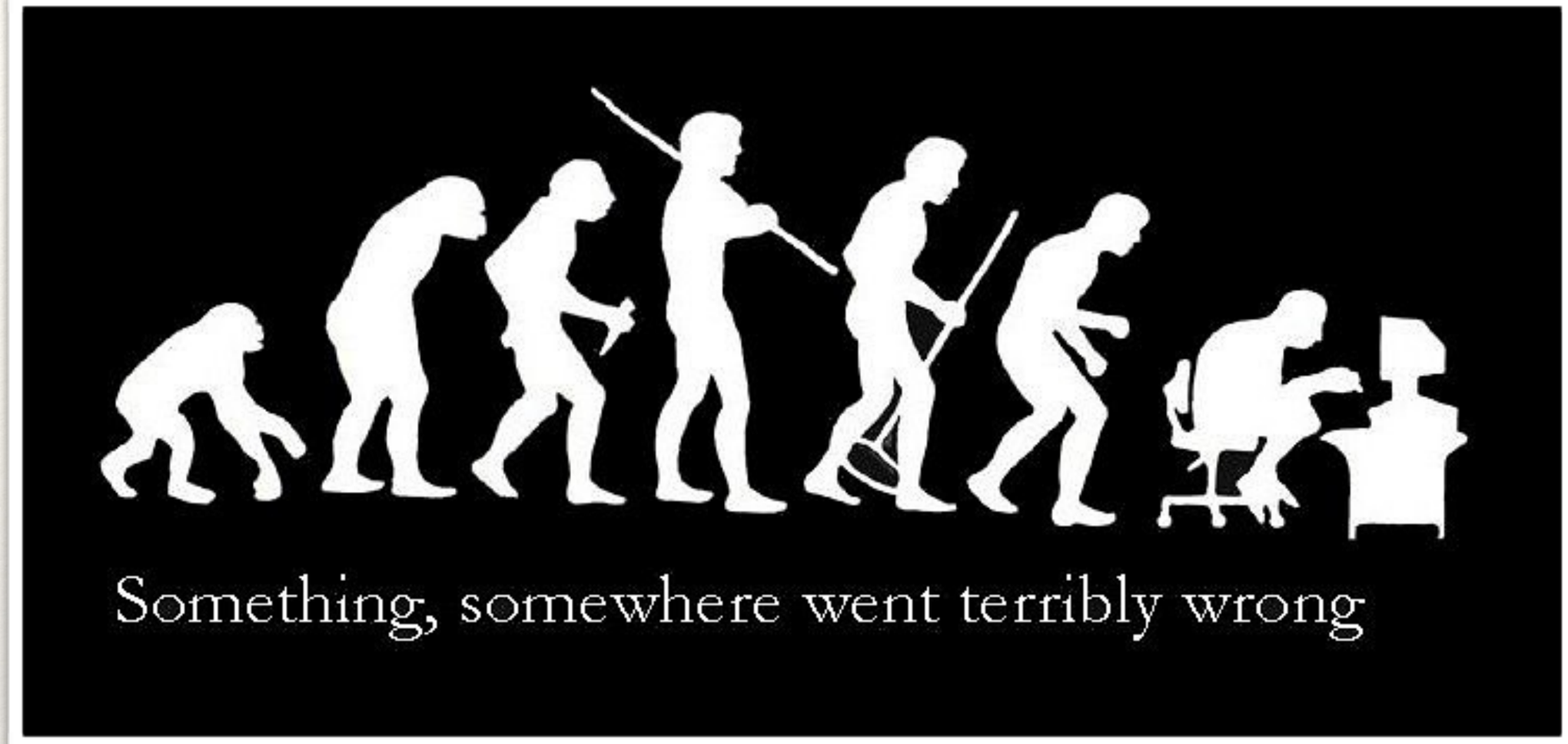
ULBRA
CAMPUS CACHOEIRA DO SUL



Processos

Professor
Wagner Gadêa Lorenz
wagnerglorenz@gmail.com

Disciplina: Sistemas Operacionais
Curso de Sistemas de Informação

Cachoeira do Sul, 12 de Março de 2015.

Processos

Todos os computadores modernos são capazes de fazer várias coisas ao mesmo tempo. Enquanto executa um programa do usuário, um computador também pode ler os dados de um disco e mostrar um texto na tela ou enviá-lo para uma impressora.

Em um sistema multiprogramado, a CPU também salta de programa para programa, executando cada um deles por dezenas ou centenas de milissegundos.

Processos

Estritamente falando, enquanto a cada instante a CPU executa somente um programa, no decorrer de um segundo ela pode trabalhar sobre vários programas, dando aos usuários a ilusão de paralelismo.

Algumas vezes, nesse contexto, fala-se de **pseudoparalelismo** para contrastar com o verdadeiro paralelismo de hardware dos sistemas **multiprocessadores** (que têm duas ou mais CPUs que compartilham simultaneamente a mesma memória física).

Processos

Ter controle sobre múltiplas atividades em paralelo é algo difícil para as pessoas. Contudo, projetistas de sistemas operacionais vêm desenvolvendo ao longo dos anos um modelo conceitual (processos sequenciais) que facilita o paralelismo.

Modelo de Processo

Nesse modelo, todos os softwares que podem executar em um computador - inclusive, algumas vezes, o próprio sistema operacional - são organizados em vários **processos sequenciais** (ou, para simplificar, **processos**).

Um processo é apenas um programa em execução acompanhado dos valores atuais do contador de programa, dos registradores e das variáveis.

Modelo de Processo

Conceitualmente, cada processo tem sua própria CPU virtual. É claro que na realidade a CPU troca, a todo momento, de um processo para outro, mas para entender o sistema, é muito mais fácil pensar em um conjunto de processos executando (pseudo) paralelamente do que tentar controlar o modo como a CPU faz essa alternância.

Esse mecanismo de trocas rápidas é **chamado de multiprogramação**.

Modelo de Processo

Pode-se observar na Figura 1(a), um computador **multiprogramado** com quatro programas na memória.

Na Figura 1 (b) estão quatro processos, cada um com seu próprio fluxo de controle (isto é, seu próprio contador de programa lógico) e executando independentemente dos outros.

Modelo de Processo

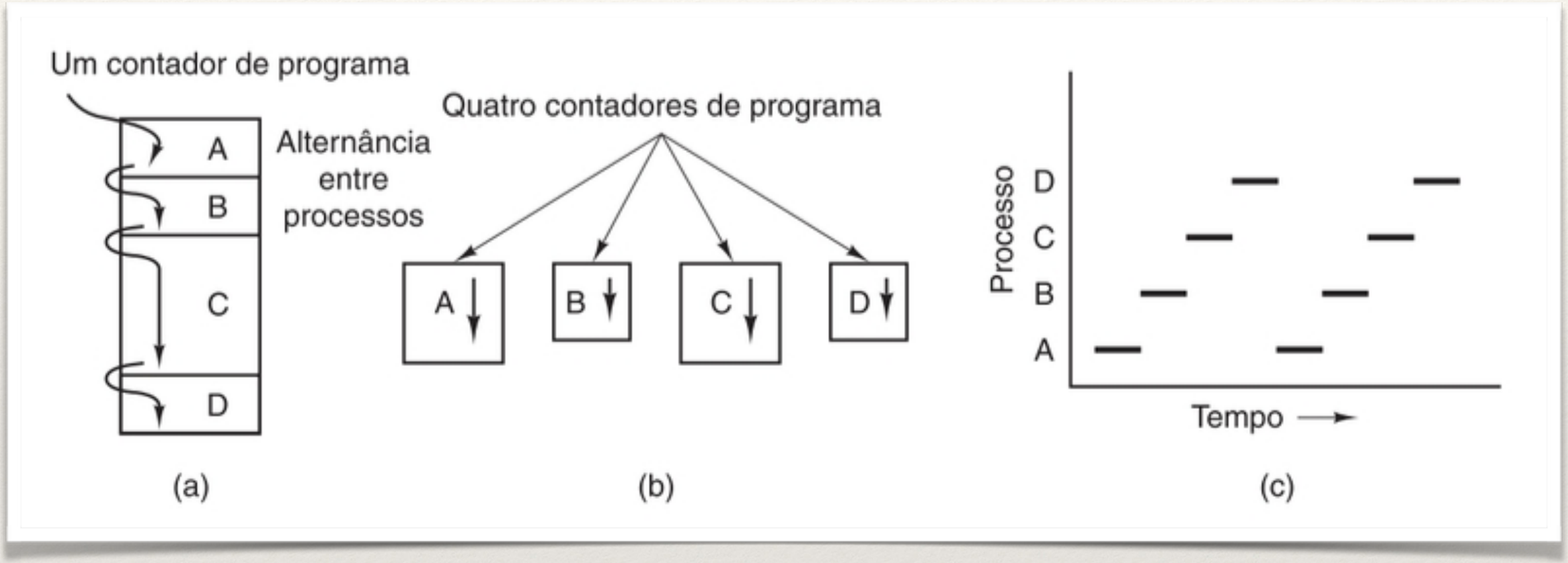


Figura 1. (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Somente um programa está ativo a cada momento.

Modelo de Processo

É claro que existe apenas um contador de programa físico. Assim, quando cada processo executa, seu contador de programa lógico é carregado no contador de programa real.

Quando acaba o tempo de CPU alocado para um processo, o contador de programa físico é salvo no contador de programa lógico do processo na memória.

Na Figura 1 (c) vemos que, por um intervalo de tempo suficientemente longo, todos os processos estão avançando, mas, a cada instante, apenas um único processo está realmente executando.

Modelo de Processo

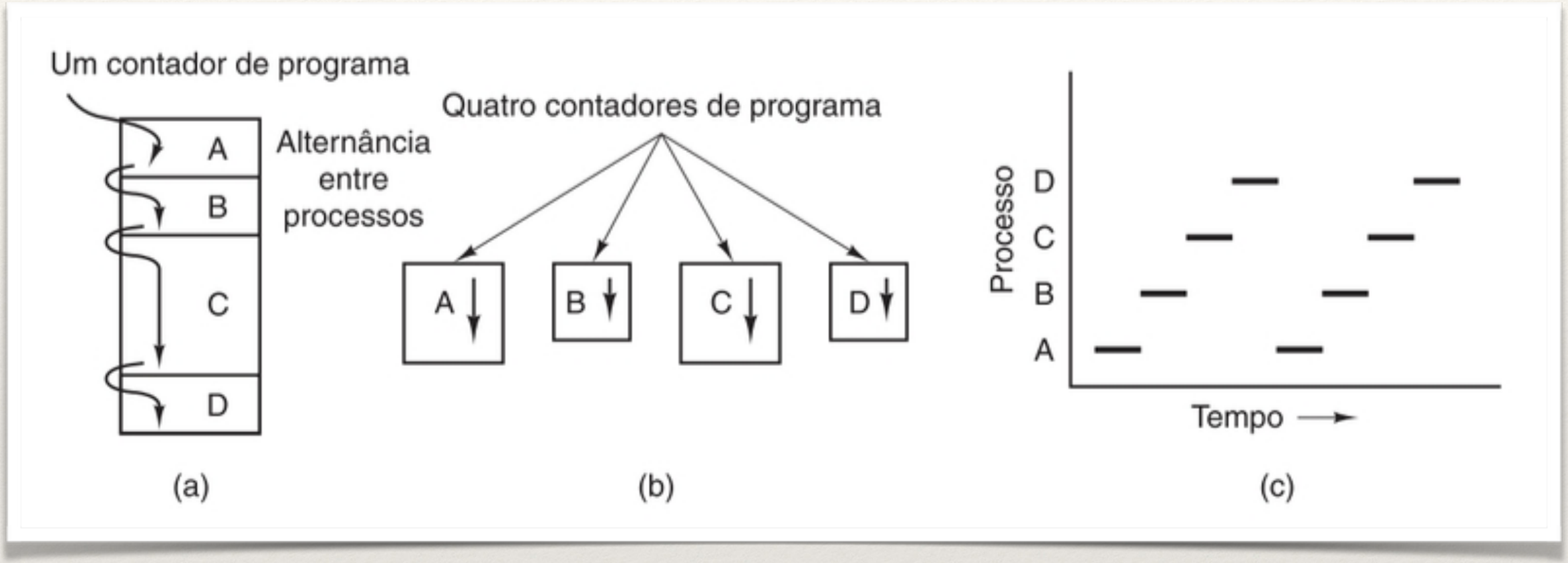


Figura 1. (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Somente um programa está ativo a cada momento.

Modelo de Processo

Com a alternância da CPU entre os processos, a taxa na qual o processo realiza sua computação não será uniforme e provavelmente não será nem reproduzível se os mesmos processos executarem novamente.

Desse modo os processos não devem ser **programados** com **hipótese predefinidas** sobre a **temporização**.

Modelo de Processo

Considere, por exemplo, um processo de E/S que inicia uma fita magnética para que sejam restaurados arquivos de backup; ele executa dez mil vezes em um laço ocioso para aguardar que uma rotação seja atingida e então emite um comando para ler o primeiro registro.

Se a CPU decidir alternar para um outro processo durante a execução do laço ocioso, o processo da fita não executará enquanto a cabeça de leitura não chegar ao primeiro registro.

Quando um processo tem restrições críticas de tempo real como essa - isto é, eventos específicos devem ocorrer dentro de um intervalo de tempo, prefixado em milissegundos - é preciso tomar medidas especiais para que esses eventos ocorram.

Contudo, em geral a maioria dos processos não é afetada pelo aspecto inerente de multiprogramação da CPU ou pelas velocidades relativas dos diversos processos.

Modelo de Processo

A diferença entre um processo e um programa é sutil, mas crucial.

Vamos utilizar uma analogia:

Imagine um cientista de computação com dotes culinários e que está assando um bolo de aniversário para sua filha. Ele tem um a receita de bolo de aniversário e uma cozinha bem suprida, com todos os ingredientes: farinha, ovos, açúcar, essência de baunilha, entre outros.

Nessa analogia, a receita é o programa (isto é, um algoritmo expresso por uma notação adequada), o cientista é o processador (CPU) e os ingredientes de bolo são os dados de entrada. O processo é a atividade desempenhada pelo confeiteiro de ler a receita, buscar os ingredientes e assar o bolo.

Modelo de Processo

Agora imagine que o filho do cientista chegue chorando dizendo que uma abelha o picou. O cientista registra onde ele estava na receita (o estado atual do processo é salvo), busca um livro de primeiros socorros e começa a seguir as instruções contidas nele.

Nesse ponto, vemos que o processador está sendo alternado de um processo (assar bolo) para um processo de prioridade mais alta (fornecer cuidados médicos), cada um em um programa diferente (receita versus livro de primeiros socorros).

Quando a picada de abelha tiver sido tratada, o cientista voltará ao seu bolo, continuando do ponto em que parou.

Modelo de Processo

A **ideia principal** é que um **processo** constitui uma **atividade**. Ele possui **programa**, **entrada**, **saída** e um **estado**.

Um único **processador** pode ser compartilhado entre os **vários processos**, com algum **algoritmo** de **escalonamento** usado para **determinar quando parar o trabalho sobre um processo e servir um outro.**

Criação de Processo

Os sistemas operacionais precisam assegurar de algum modo a existência de todos os **processos necessários**.

Em sistemas muito simples, ou em sistemas projetados para executar apenas uma **única aplicação** (por exemplo, o controlador do forno de microondas), é possível ter **todos** os **processos** que **serão necessários presentes** quando o sistema é **ligado**.

Criação de Processo

Contudo, em **sistemas** de **propósito geral**, é **necessário** algum modo de **criar** e **terminar processos** durante a operação, quando for preciso.

Criação de Processo

Há **quatro eventos principais** que fazem com que processos sejam criados:

1. Início do sistema;

2. Execução de uma chamada ao sistema de criação de processo por um processo em execução;

3. Uma requisição do usuário para criar um novo processo;

4. Início de um job em lote.

Criação de Processo

Quando um sistema operacional é carregado, em geral criam-se vários processos.

Alguns deles são processos em primeiro plano, ou seja, que interagem com usuários (humanos) e realizam tarefas para eles.

Outros são processos de segundo plano, que não estão associados a usuários em particular, mas que apresentam alguma função específica.

Criação de Processo

Por exemplo, um processo em segundo plano pode ser designado a aceitar mensagens eletrônicas sendo recebidas, ficando inativo na maior parte do dia, mas surgindo de repente quando uma mensagem chega.

Outro processo em segundo plano pode ser destinado a aceitar solicitação que chegam para páginas Web hospedadas naquela máquina, despertando quando uma requisição chega pedindo o serviço.

Processos que ficam em segundo plano com a finalidade de tratar alguma atividade como mensagens eletrônicas, páginas Web, notícias, impressão, entre outros são chamados de **daemons**.

Criação de Processo

É comum aos grandes sistemas lançarem mão de dezenas deles.

No Unix, o programa *ps* pode ser usado para relacionar os processos que estão executando.

No Windows o gerenciador de tarefas pode ser usado.

Criação de Processo

Além dos processos criados durante a **carga** do **sistema** operacional, **novos processos** podem ser criados **depois** disso.

Muitas vezes, um **processo** em **execução emitirá chamadas** ao **sistema (*system calls*)** para **criar** um ou **mais novos processos** para ajudá-lo em seu trabalho.

Criar **novos processos** é particularmente útil quando a tarefa a ser executada pode facilmente ser formulada com base em vários processos relacionados, mas interagindo de maneira independente.

Criação de Processo

Por exemplo, se uma grande quantidade de dados estiver sendo trazida via rede para que seja subsequentemente processada, poderá ser conveniente criar um processo para trazer esses dados e armazená-los em um local compartilhado da memória, enquanto um segundo processo remove os dados e os processa.

Em um sistema **multiprocessador**, permite que cada processo execute em uma CPU diferentes, tornando o trabalho mais rápido.

Criação de Processo

Em sistemas interativos, os usuários podem iniciar um programa digitando um comando ou clicando (duas vezes) um ícone.

Cada uma dessas ações inicia um novo processo e executa nele o programa selecionado.

Existe sistemas Unix baseados em comandos que executam o X, o novo processo toma posse da janela na qual ele foi disparado.

No Windows, quando iniciado, um processo não tem uma janela, mas ele pode criar uma (ou mais de uma), e a maioria deles cria.

Nos dois sistemas os usuários podem ter múltiplas janelas abertas ao mesmo tempo, cada uma executando algum processo. usando o mouse, o usuário seleciona uma janela e interage com o processo - por exemplo, fornecendo a entrada quando necessário.

Criação de Processo

A última situação na qual processos são criados aplica-se somente a sistemas em lote encontrados em computadores de grande porte.

Nesses sistemas, usuários podem submeter (até remotamente) jobs em lote para o sistema.

Quando julgar que tem recursos para executar outro job, o sistema operacional criará um novo processo e executará nele o próximo job da fila de entrada.

Criação de Processo

Tecnicamente, em todos esses casos, um novo processo é criado por um processo existente executando uma chamada as sistema de criação de processos.

Esse processo pode estar executando um processo de usuário, um processo de sistema invocado a partir do teclado ou mouse ou um processo gerenciador de lotes.

O que o processo faz é executar uma chamada ao sistema para criar um novo processo e assim indica, direta ou indiretamente, qual programa executar nele.

Término de Processo

Depois de criado, um processo começa a executar e faz seu trabalho.

Contudo, nada é para sempre, nem mesmo os processos. Mais cedo ou mais tarde o novo processo terminará, normalmente em razão de alguma das seguintes condições:

1. Saída normal (voluntária);
2. Saída por erro (voluntária);
3. Erro fatal (involuntário);
4. Cancelamento por um outro processo (involuntário).

Término de Processo

Na maioria das vezes, os processos terminam porque fizeram seu trabalho.

Quando acaba de compilar o programa atribuído a ele, o compilador executa uma chamada ao sistema para dizer ao sistema operacional que ele terminou.

Essa chamada é a ***exit*** no Unix e a ***ExitProcess*** no Windows.

Programas baseado em tela suportam também o término **voluntário**.

Processadores de texto, visualizadores da Web (browsers) e programas similares sempre têm um ícone ou um item de menu que o usuário pode clicar para dizer ao processo que remova quaisquer arquivos temporários que ele tenha aberto e então **termine**.

Término de Processo

O segundo motivo para término é que o processo descobre um erro fatal.

Por exemplo, se o usuário digita o comando

cc foo.c

par compilar o programa foo.c e esse arquivo não existe, o compilador simplesmente emite uma chamada de saída ao sistema.

Processos interativos com base na tela geralmente não fecham quando parâmetros errados são fornecidos. Em vez disso, uma caixa de diálogo emerge e pergunta ao usuário se ele quer tentar novamente.

Término de Processo

A terceira razão para o término é um erro causado pelo processo, muitas vezes por um erro de programa.

Entre vários exemplos estão a execução de uma instrução ilegal, a referência à memória inexistente ou a divisão por zero.

Em alguns sistemas (por exemplo, Unix), um processo pode dizer ao sistema operacional que deseja, ele mesmo, tratar certos erros. Nesse caso, o processo é sinalizado (interrompido) em vez de finalizado pela ocorrência de erros.

Término de Processo

A quarta razão pela qual um processo pode terminar se dá quando um processo executa uma chamada ao sistema dizendo ao sistema operacional para cancelar algum outro processo.

No Unix, essa chamada é a *kill*.

A função Win32 correspondente é a *TerminateProcess*.

Em ambos os casos, o processo que for efetuar o cancelamento deve ter a autorização necessária para fazê-lo.

Em alguns sistemas, quando um processo termina, voluntariamente ou não, todos os processos criados por ele são imediatamente cancelados.

Contudo, nem o Unix nem o Windows funcionam dessa maneira.

Hierarquias de Processos

Em alguns sistemas, quando um processo cria outro processo, o processo pai e o processo filho continuam, de certa maneira, associados.

O próprio processo filho pode gerar mais processos, formando uma hierarquia de processos.

Observer que isso é diferente do que ocorre com plantas e animais, que utilizam a reprodução sexuada, pois um processo tem apenas um pai (mas pode ter nenhum, um, dois, ou mais filhos).

Hierarquias de Processos

No Unix, um processo, todos os seus filhos e descendentes formam um grupo de processo. Quando um usuário envia um sinal do teclado, o sinal é entregue a todos os membros do grupo de processo associado com o teclado (normalmente todos os processos ativos que foram criados na janela atual).

Individualmente, cada processo pode capturar o sinal, ignorá-lo ou tomar uma ação predefinida - ser cancelado pelo sinal.

Hierarquias de Processos

Outro exemplo da atuação dessa hierarquia pode ser observado no início do Unix quando o computador é ligado. Um processo especial, chamado *init*, está presente na imagem de carga do sistema.

Quando começa a executar, ele lê um arquivo dizendo quantos terminais existem. Então ele se bifurca em um novo processo para cada terminal.

Esses processos esperam por alguma conexão de usuário. Se algum usuário se conectar, o processo de conexão executará um interpretador de comandos para aceitar comandos do usuário.

Esses comandos podem iniciar mais processos e assim por diante. Desse modo, todos os processos em todo o sistema pertencem a uma única árvore, como o *init* na raiz.

Hierarquias de Processos

Por outro lado, o Windows não apresenta nenhum conceito de hierarquia de processos.

Todos os processo são iguais.

Algo parecido com uma hierarquia de processo ocorre somente quando um processo é criado. Ao pai é dado um identificador especial (chamado **handle**), que ele pode usar para controlar o filho.

Contudo, ele é livre para passar esse identificador para alguns outros processos, invalidando assim a hierarquia.

Os processos no Unix não podem deserdar seus filhos.

Estado dos Processos

Embora cada processo seja uma entidade independente, com seu próprio contador de programa e estado interno, muitas vezes os processos precisam interagir com outros.

Um processo pode gerar uma saída que outro processo usa como entrada. No interpretador de comandos, o comando:

cat chapter1 chapter2 chapter3 | grep tree

Estado dos Processos

cat chapter1 chapter2 chapter3 | grep tree

O primeiro processo, que executa *cat*, concatena e emite para a saída-padrão três arquivos.

O segundo processo, que executa *grep*, seleciona todas as linhas contendo a palavra “tree”.

Dependendo das velocidades relativas dos dois processos (atreladas tanto à complexidade relativa dos programas quanto ao tempo de CPU que cada um deteve), pode ocorrer que o *grep* esteja pronto para executar, mas não haja entrada para ele.

Ele então deve bloquear até que alguma entrada esteja disponível.

Estado dos Processos

Um processo bloqueia porque obviamente não pode prosseguir - em geral porque está esperando por uma entrada ainda não disponível.

É possível também que um processo conceitualmente pronto e capaz de executar esteja bloqueado porque o sistema operacional decidiu alocar a CPU para outro processo por algum tempo.

Estado dos Processos

Essa duas condições são completamente diferentes.

No primeiro caso a suspensão é inerente ao problema (não se pode processar a linha de comando do usuário enquanto ele não digita nada).

O segundo é uma tecnicalidade do sistema (não há CPUs suficientes para dar a cada processo um processador exclusivo).

Estado dos Processos

Pode-se observar na Figura 2, um diagrama de estados mostrando os três estados de um processo:

1. Em execução (realmente usando a CPU naquele instante);

2. Pronto (executável; temporariamente parado para dar lugar a outro processo);

3. Bloqueado (incapaz de executar enquanto um evento externo não ocorre).

Estado dos Processos

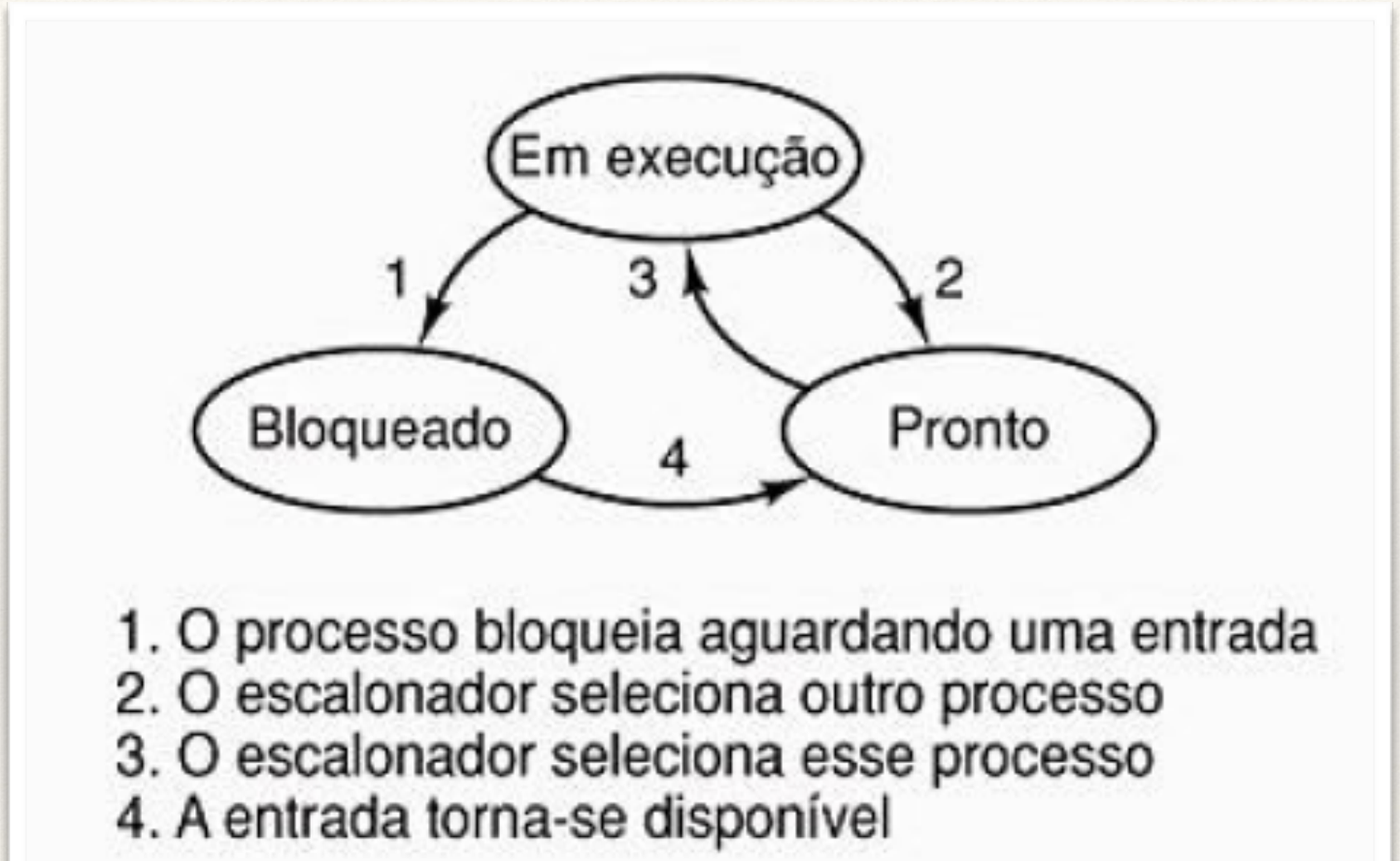


Figura 2. Um processo pode estar em execução, bloqueado ou pronto. As transições entre esses estados aparecem ilustradas.

Estado dos Processos

Logicamente, os **dois primeiros estados** são **similares**. Em ambos os casos o processo vai executar, só que no segundo não há, temporariamente, CPU disponível para ele.

O **terceiro estado** é diferente dos dois primeiros, pois o processo não pode executar, mesmo que a CPU não tenha nada pra fazer.

Estado dos Processos

Quando transições são possíveis entre esses três estados, conforme se vê na figura.

A **transição 1** ocorre quando um processo descobre que ele não pode prosseguir.

Em alguns sistemas, o processo precisa executar uma chamada ao sistema, como *block* ou *pause*, para entrar no estado bloqueado.

Em outros sistemas, inclusive no Unix, quando um processo lê de um pipe ou de um arquivo especial (por exemplo, um terminal) e não há entrada disponível, o processo é automaticamente bloqueado.

Estado dos Processos

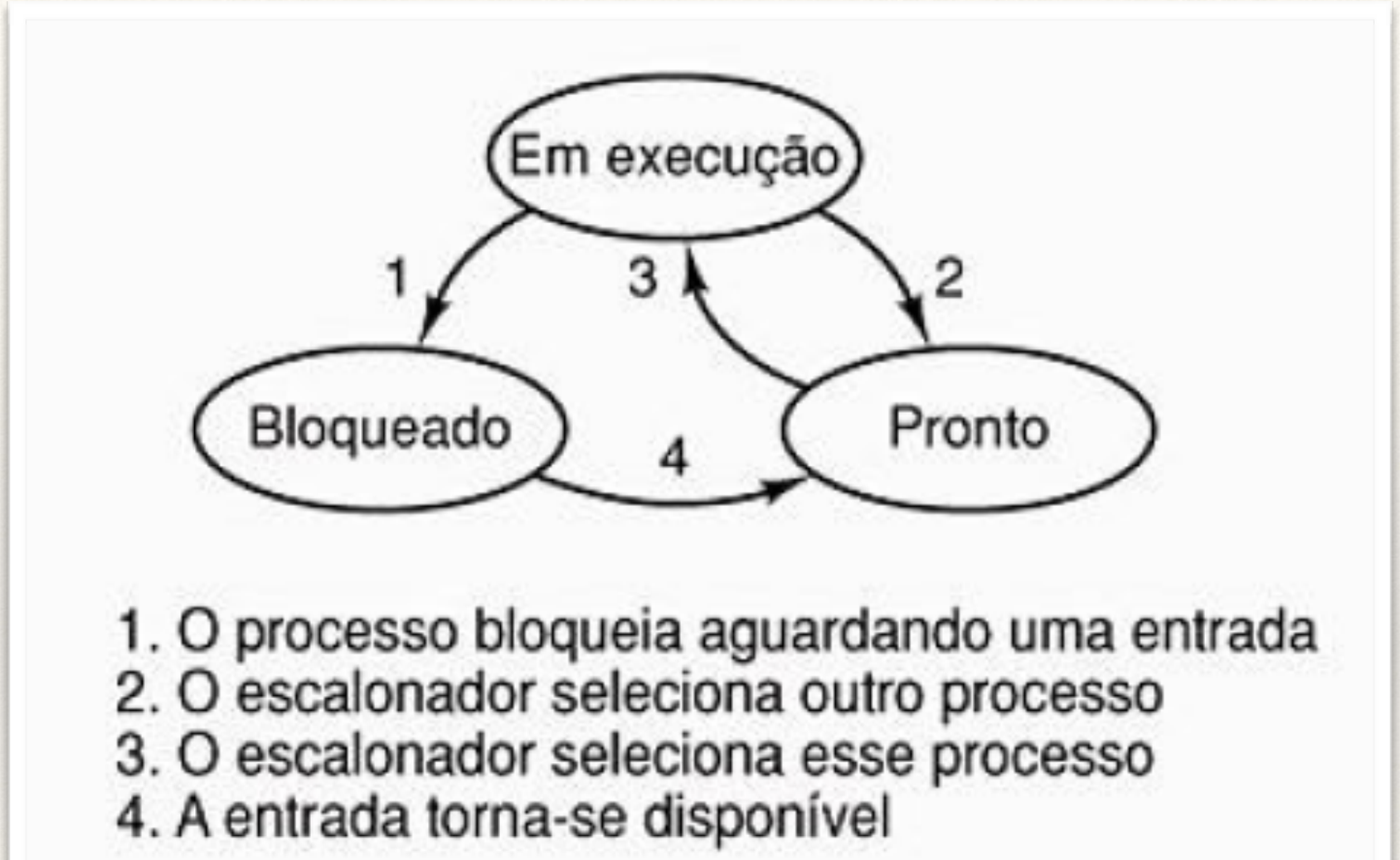


Figura 2. Um processo pode estar em execução, bloqueado ou pronto. As transições entre esses estados aparecem ilustradas.

Estado dos Processos

As **transições 2** e **3** são causadas pelo **escalonador** de **processos** - uma parte do sistema operacional - sem que o processo saiba disso.

A **transição 2** ocorre quando o escalonador decide que o processo em execução já teve tempo suficiente de CPU e é momento de deixar outro processo ocupar o tempo de CPU.

Estado dos Processos

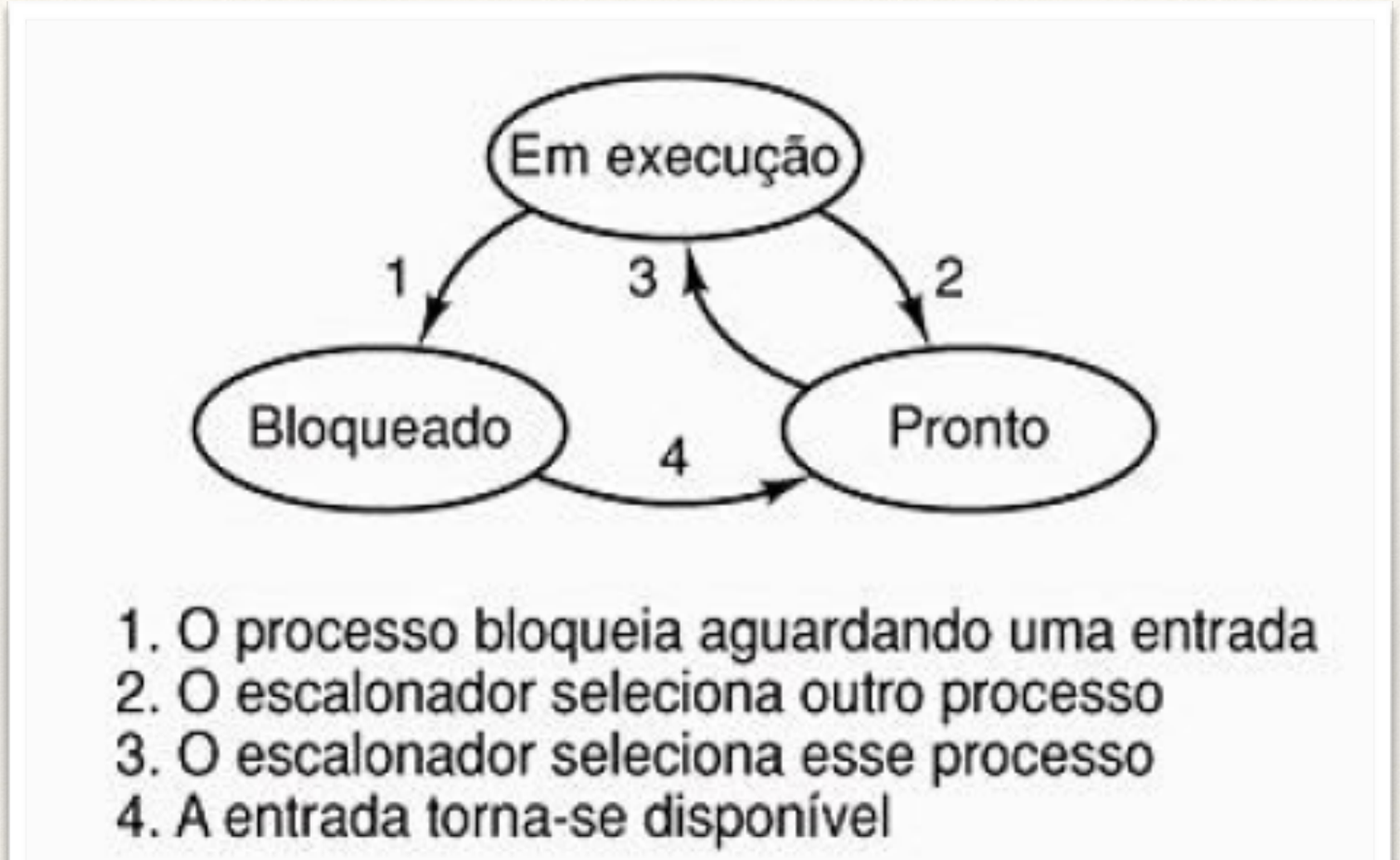


Figura 2. Um processo pode estar em execução, bloqueado ou pronto. As transições entre esses estados aparecem ilustradas.

Estado dos Processos

A **transição 3** ocorre quando todos os outros processos já compartilham a CPU, de uma maneira justa, e é hora de o primeiro processo obter novamente a CPU.

O escalonador - isto é, a decisão sobre quando e por quanto tempo cada processo deve executar.

Estado dos Processos

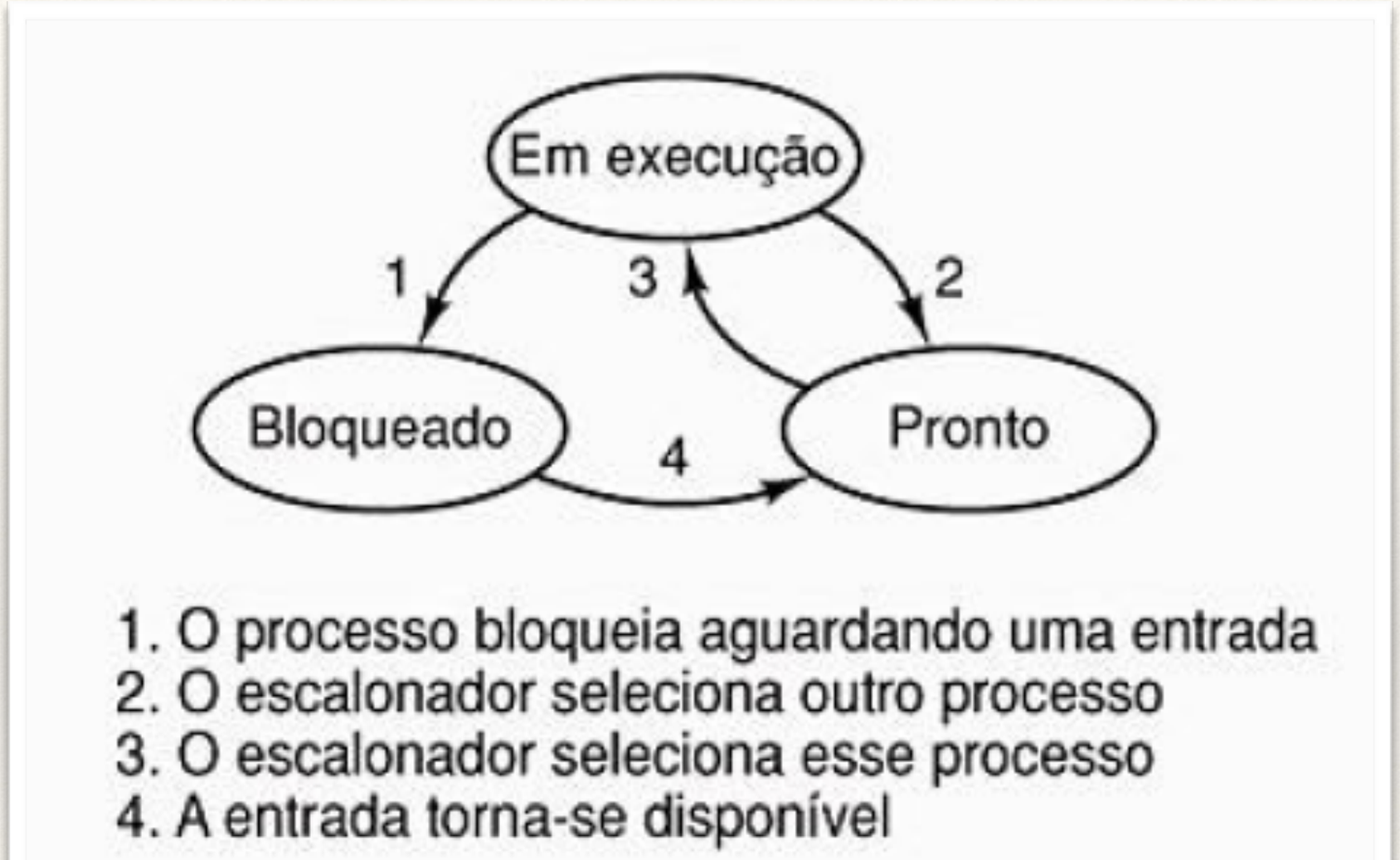


Figura 2. Um processo pode estar em execução, bloqueado ou pronto. As transições entre esses estados aparecem ilustradas.

Estado dos Processos

A **transição 4** ocorre quando acontece um evento externo pelo qual um processo estava aguardando (como a chegada de alguma entrada).

Se nenhum outro processo estiver executando naquele momento, a transição 3 será disparada e o processo executará.

Caso contrário, ele poderá ter de aguardar um estado de *pronto* por um pequeno intervalo de tempo, até que a CPU esteja disponível e sua vez chegue.

Estado dos Processos

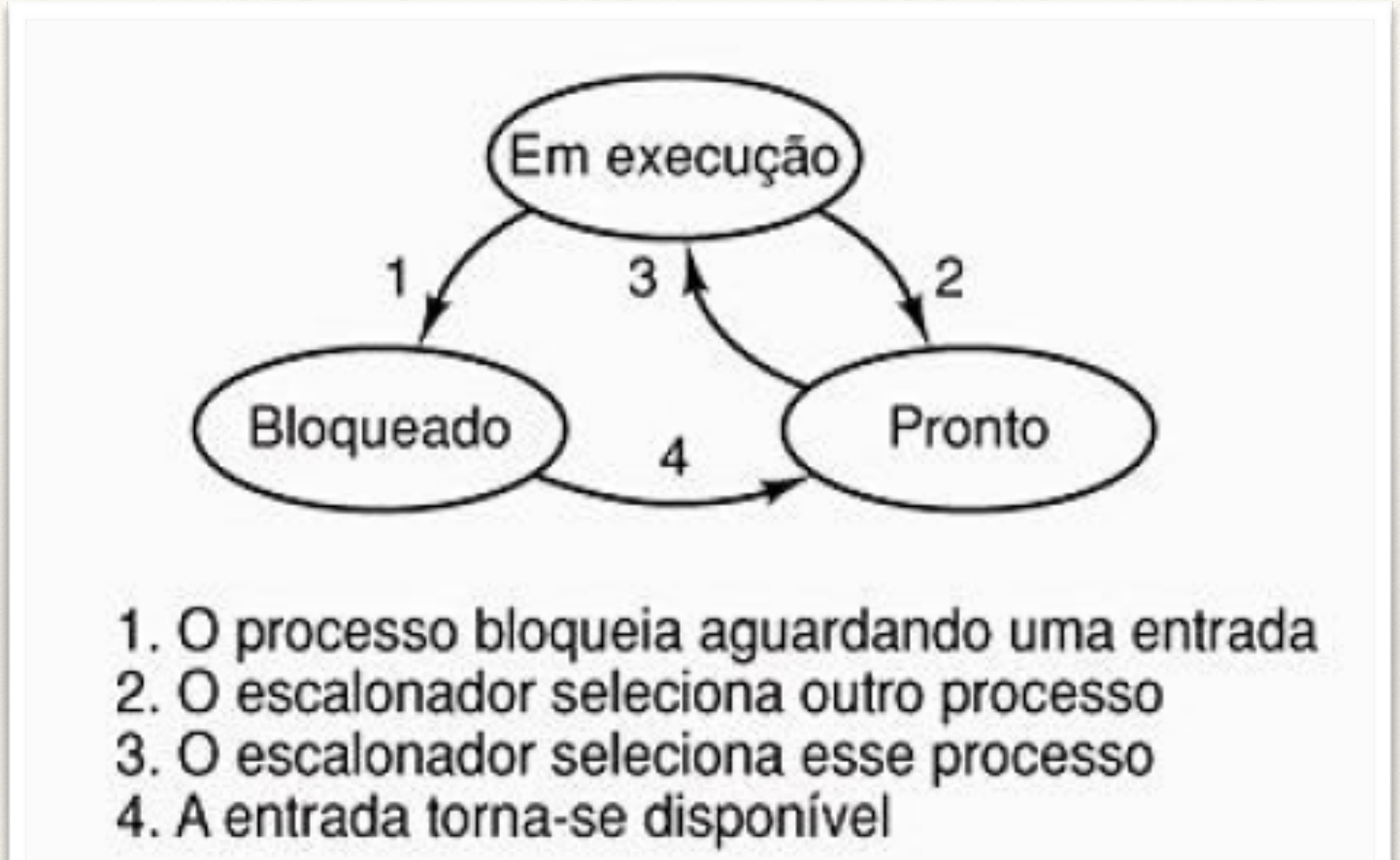


Figura 2. Um processo pode estar em execução, bloqueado ou pronto. As transições entre esses estados aparecem ilustradas.

Estado dos Processos

Com o modelo de processo, torna-se muito mais fácil saber o que está ocorrendo dentro do sistema.

Alguns dos processos chamam programas que executam comandos digitados por um usuário.

Outros processos são parte do sistema e tratam tarefas como fazer requisições por serviços de arquivos ou gerenciar os detalhes do funcionamento de um acionador de disco.

Estado dos Processos

Quando ocorre uma interrupção de disco, o sistema toma a decisão de parar de executar o processo atual e retomar o processo do disco que foi bloqueado para aguardar essa interrupção.

Assim, em vez de pensar em interrupções, podemos pensar em processos de usuários, processo de disco, processos de terminais ou outros que bloqueiam quando estão a espera de que algo aconteça.

Finalizada a leitura do disco ou a digitação de um caractere, o processo que aguarda por isso é desbloqueado e torna-se disponível para executar novamente.

Estado dos Processos

Essa visão dá origem ao modelo mostrado na Figura 3. Nele, o nível mais baixo do sistema operacional é o escalonador, com diversos processos acima dele.

Todo o tratamento de interrupção e detalhes sobre a inicialização e o bloqueio de processos estão ocultos naquilo que é chamado aqui de escalonador, que, na verdade, não te muito código.

O restante do sistema operacional é bem estruturado na forma de processos. Contudo, poucos sistemas reais são tão bem estruturados como esses.

Estado dos Processos

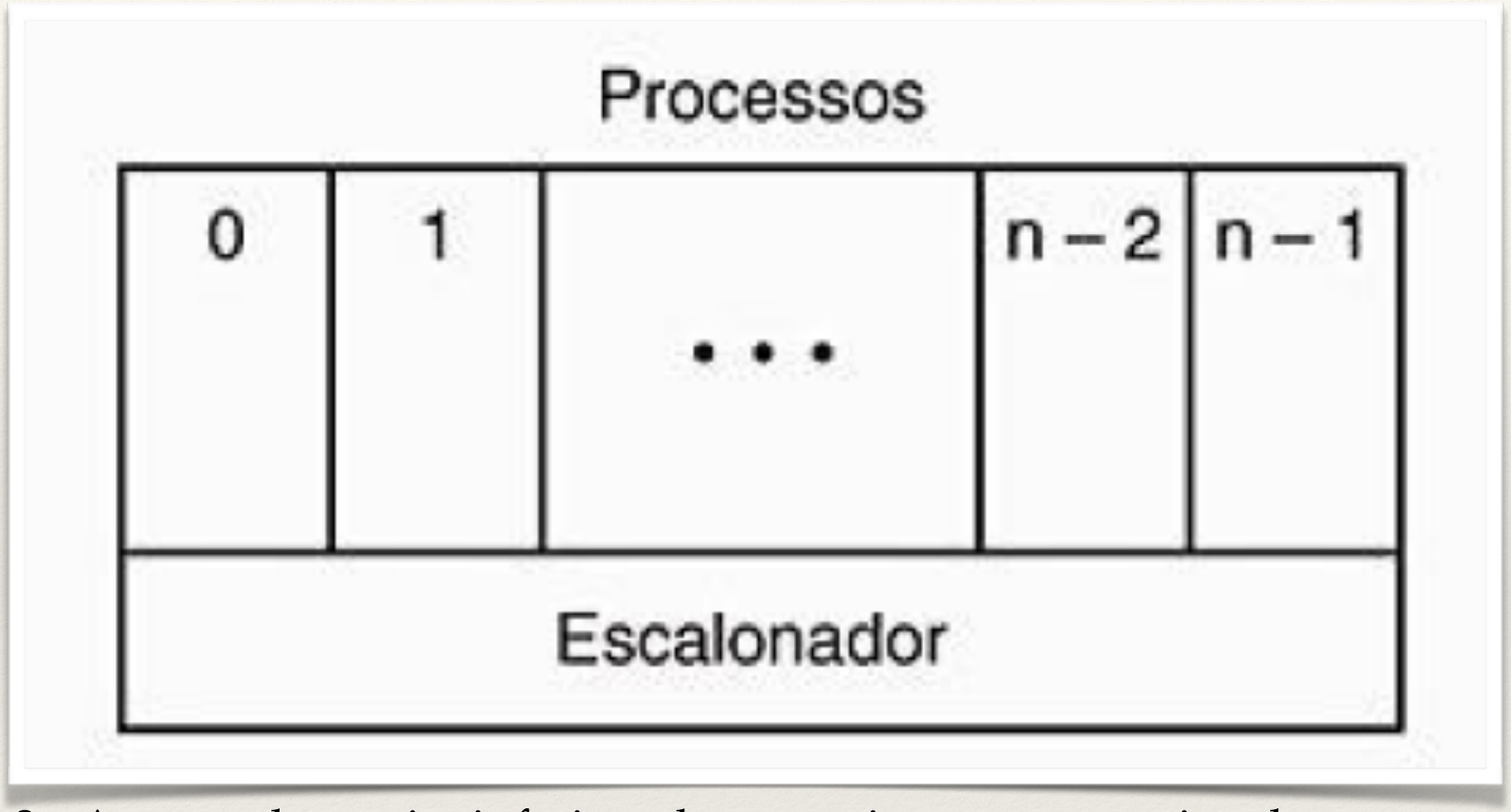


Figura 3. A camada mais inferior de um sistema operacional estruturado por processos trata as interrupções e o escalonamento. Acima daquela camada estão os processos sequenciais.

Próxima Aula

- Processos e Deadlock.

Dúvidas

- Conteúdo
  - Moodle
  - (http://wagnerglorenz.com.br/moodle/)
- Dúvidas
  - wagnerglorenz@gmail.com

Referências Bibliográficas

- TANENBAUM, A. Sistemas Operacionais Modernos. São Paulo: Prentice Hall, 2003.
- Sistemas Operacionais Modernos - 3ª edição Tanenbaum, Andrew S. http://ulbra.bv3.digitalpages.com.br/users/publications/9788576052371