Gerenciamento de processos

Prof. Marcos Ribeiro Quinet de Andrade
Universidade Federal Fluminense – UFF
Instituto de Ciência e Tecnologia – ICT

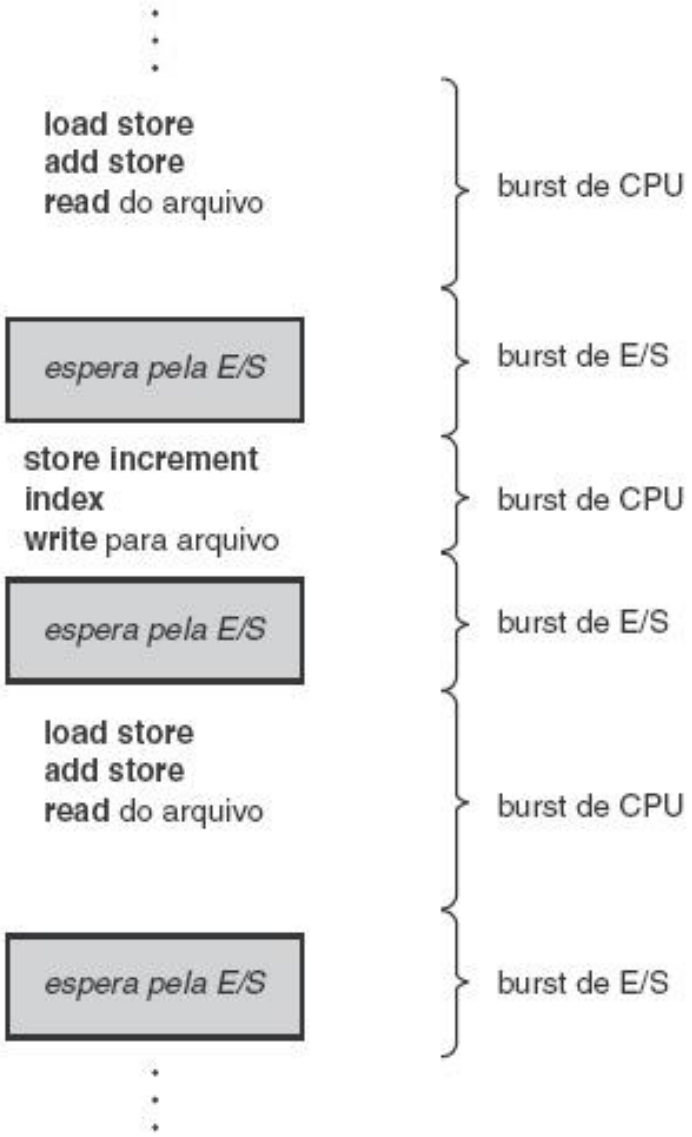
Conceitos Básicos

- Máxima utilização da UCP obtida com a **multiprogramação**;
- Ciclo do *burst* de UCP e E/S – A execução de um processo consiste em um **ciclo** de execução da UCP e espera por E/S;
- Distribuição dos *bursts* de UCP de um ou mais processos ao longo do tempo;

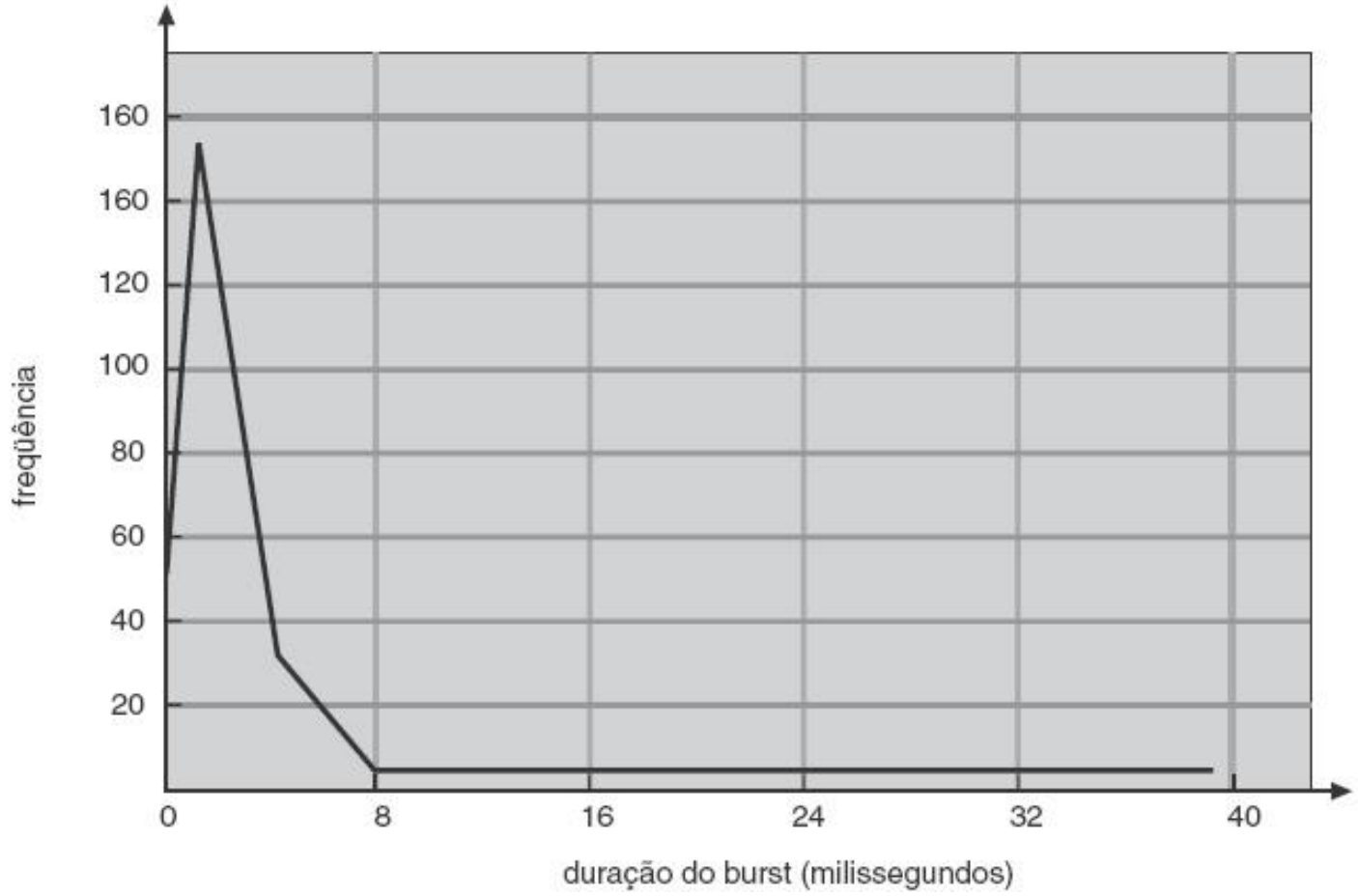
Processos

- Multiprogramação:
  - Pseudoparalelismo: coleção de processos sendo executados alternadamente na UCP;
- Um processo é caracterizado por um programa em execução, mas existe uma diferença sutil entre processo e programa:
  - Um processo pode ser composto por vários programas, dados de entrada, dados de saída e um estado (executando, bloqueado, pronto)

Alternando a Sequência de *bursts* de UCP e E/S



Histograma dos tempos de *burst* da UCP



Criando Processos

- Processos precisam ser criados e finalizados a todo o momento:
  - Na inicialização do sistema;
  - Na execução de uma chamada ao sistema de criação de processo realizada por algum processo em execução;
  - Na requisição de usuário para criar um novo processo;
  - Na inicialização de um processo em *batch* (*em mainframes* com sistemas de *batch*).

Criando Processos

- Processos podem ser:
  - Específicos para usuários específicos:
    - Leitura de um arquivo;
    - Iniciar um programa (linha de comando ou um duplo clique no mouse);
  - Com funções específicas, que independem de usuários, que são criados pelo sistema operacional e que são processados em segundo plano (*daemons*):
    - Recepção e envio de emails;
    - Serviços de Impressão;

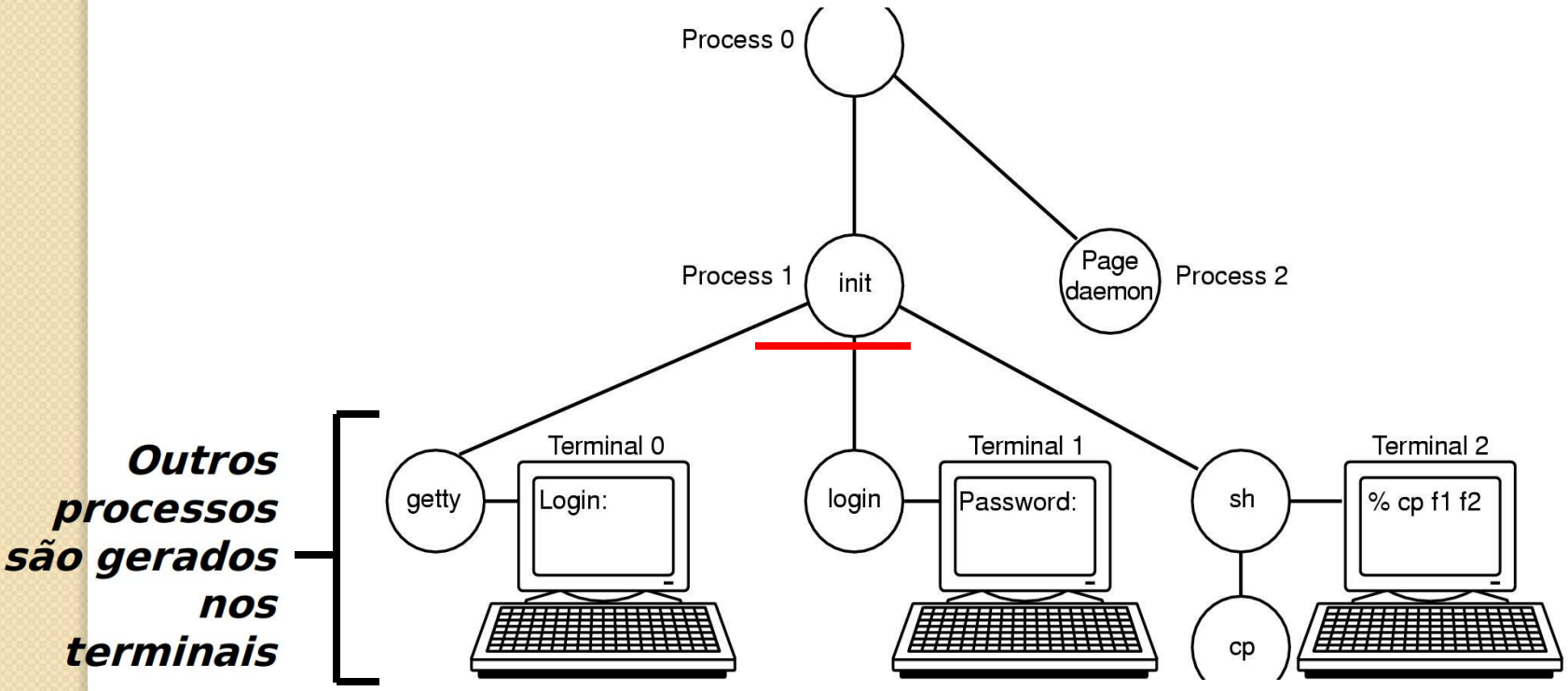
Criando Processos

- UNIX:
  - Fork;
    - Cria um processo idêntico (filho) ao processo que a chamou (pai), possuindo a mesma imagem de memória, as mesmas cadeias de caracteres no ambiente e os mesmos arquivos abertos;
    - Depois, o processo filho executa uma chamada para mudar sua imagem de memória e executar um novo programa.

- Windows:
  - CreateProcess;
    - Uma única função trata tanto do processo de criação quanto da carga do programa correto no novo processo.

Criando Processos

- Exemplo UNIX:
  - Processo init: gera vários processos filhos para atender os vários terminais que existem no sistema;



Finalizando Processos

- Condições:
  - Término normal (voluntário):
    - A tarefa a ser executada é finalizada;
    - Chamadas: *exit (UNIX) e ExitProcess (Windows).*
  - Término com erro (voluntário):
    - O processo sendo executado não pode ser finalizado, por exemplo, o comando **gcc filename.c** resultará em erro, caso o arquivo filename.c não exista;

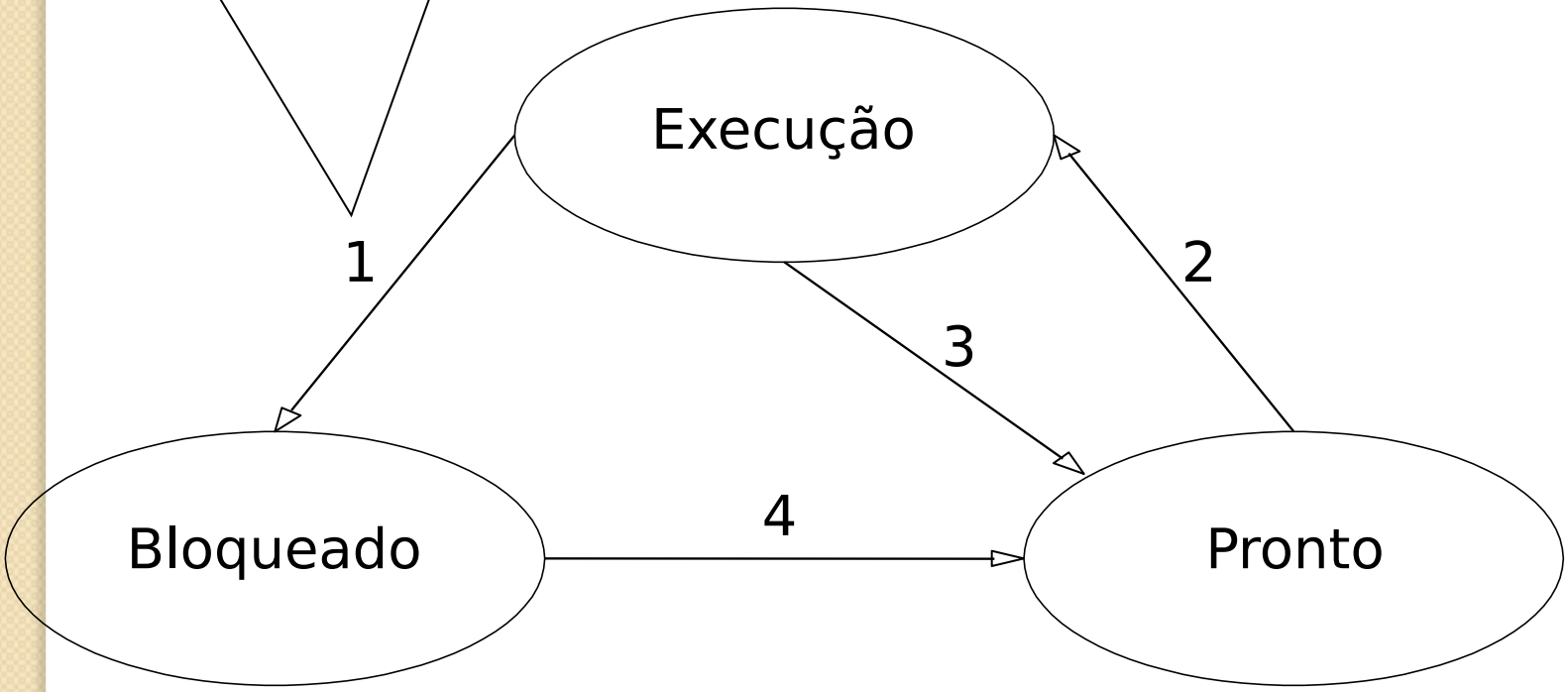
Finalizando Processos

- Condições:
  - Término com erro fatal (involuntário);
    - Erro causado por algum erro no programa (*bug*):
      - Divisão por 0 (zero);
      - Referência à memória inexistente ou não pertencente ao processo;
      - Execução de uma instrução ilegal;
  - Término causado por algum outro processo (involuntário):
    - Kill (UNIX) e TerminateProcess (Windows);

Estados de Processos

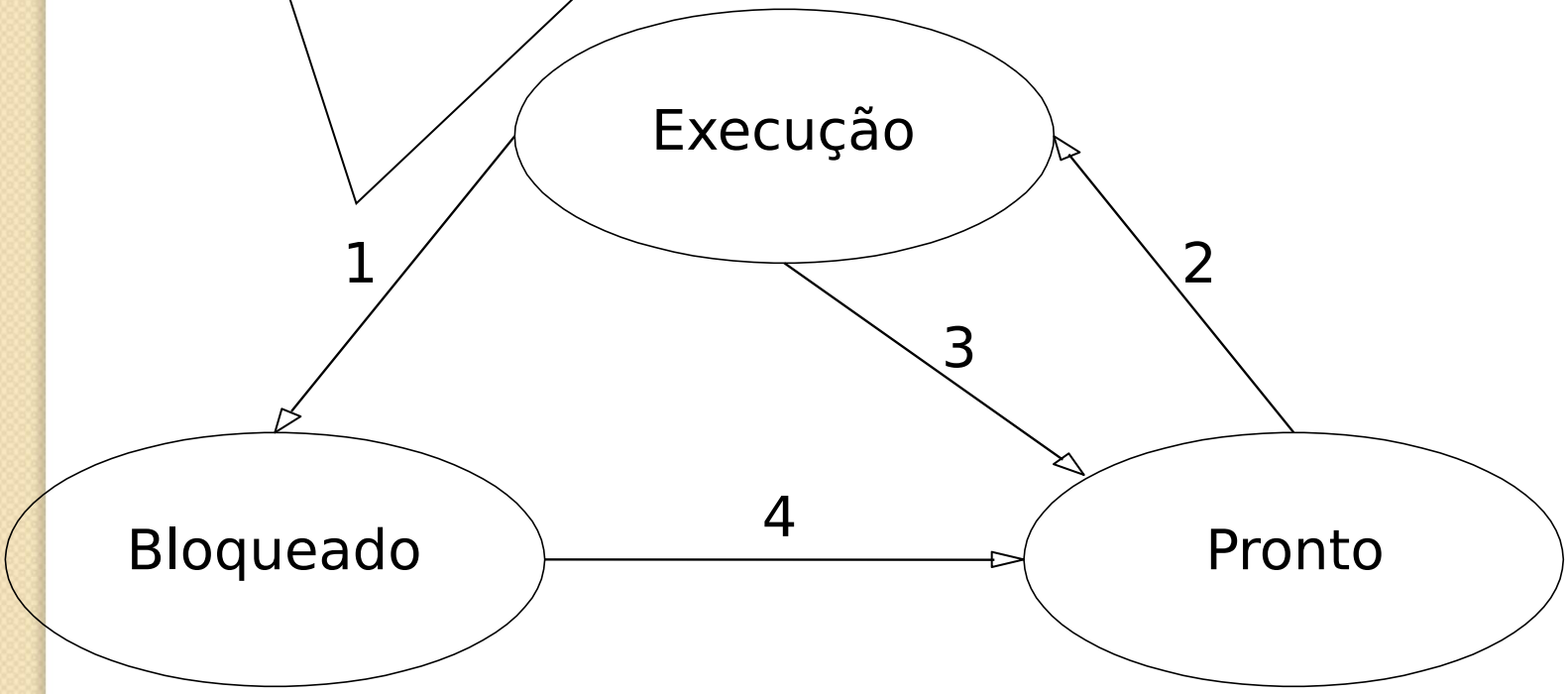
Um processo sendo executado não pode continuar sua execução, pois precisa de algum evento (E/S ou semáforo) para continuar;

Três estado básicos



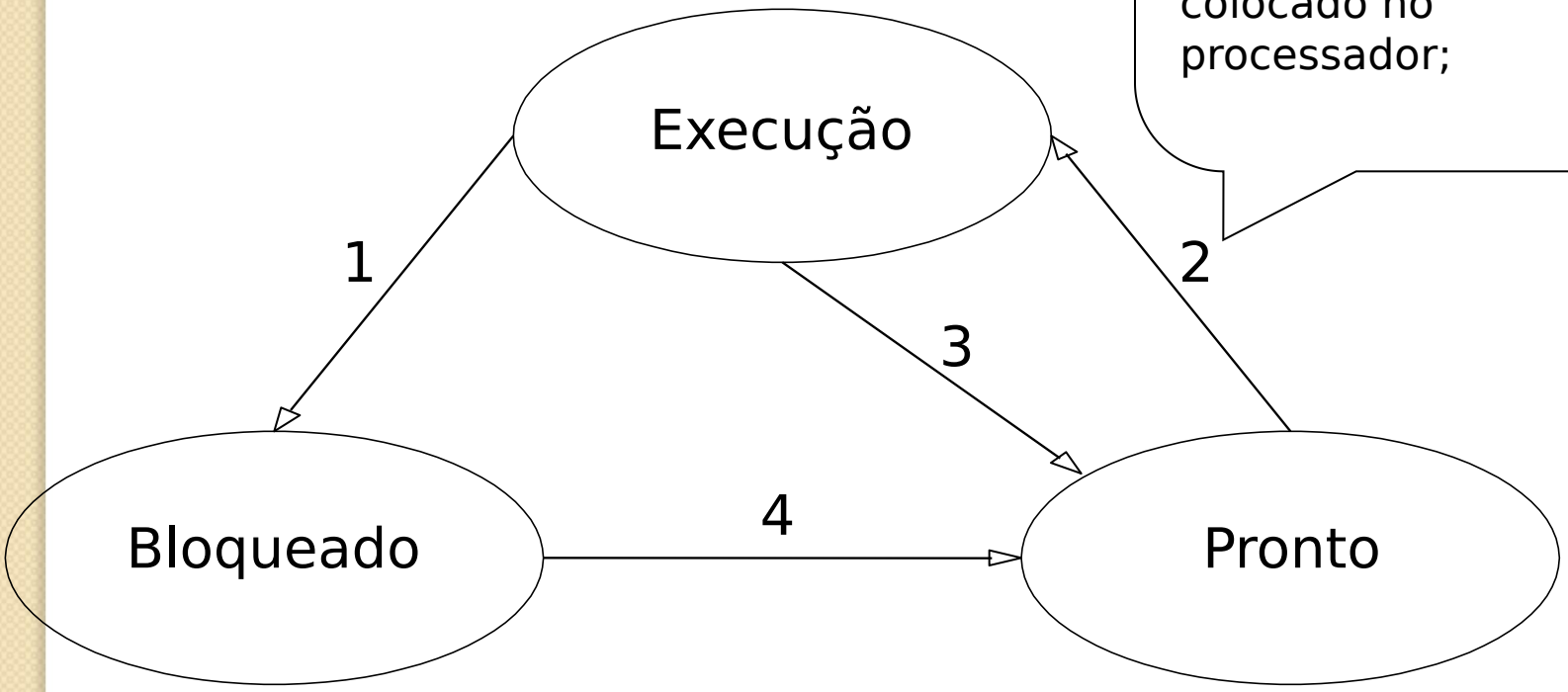
Estados de Processos

Um processo é bloqueado de duas maneiras:
- chamada ao sistema: block ou pause;
- se não há entradas disponíveis para que o processo continue sua execução;

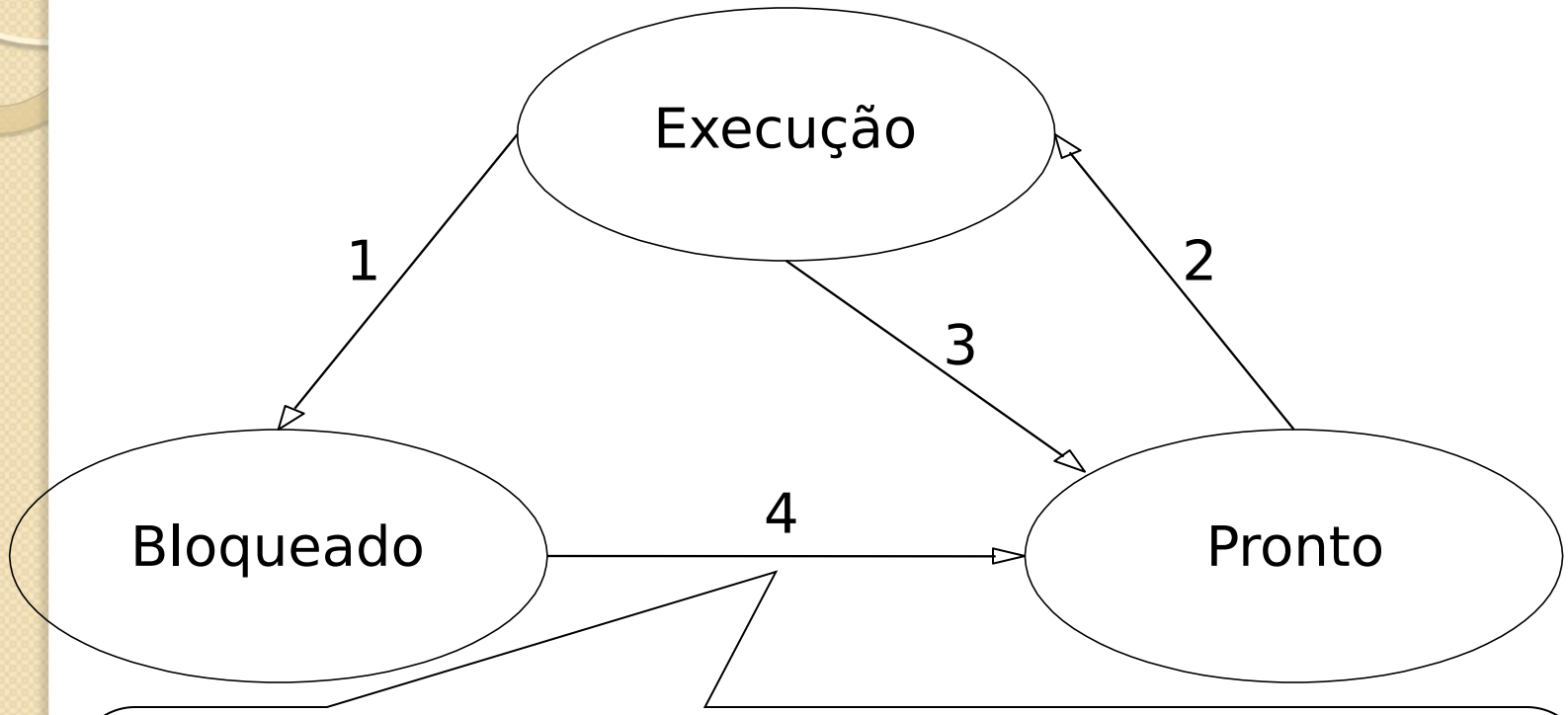


Estados de Processos

As transições 2 e 3 ocorrem durante o escalonamento de processos: o tempo destinado àquele processo acabou e outro processo é colocado no processador;



Estados de Processos



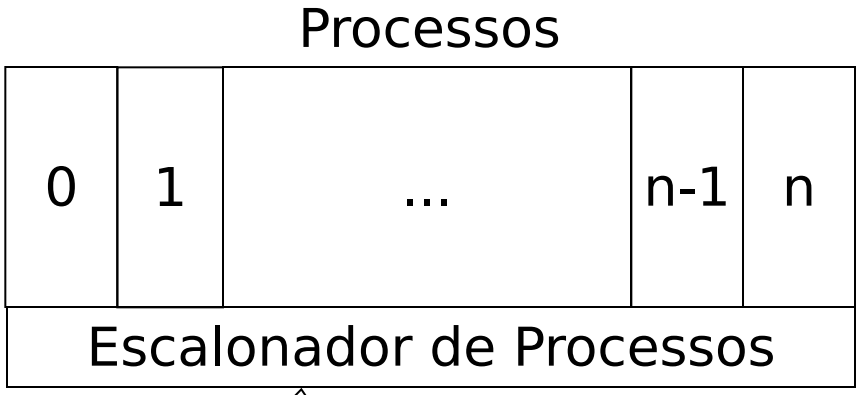
A transição 4 ocorre quando o evento esperado pelo processo bloqueado ocorre:

- se o processador está parado, o processo é executado imediatamente (2);
- se o processador está ocupado, o processo deve esperar sua vez;

Processos

- Processos *CPU-bound* (orientados à UCP): processos que utilizam muito o processador;
  - Tempo de execução é definido pelos ciclos de processador;
- Processos *I/O-bound* (orientados à E/S): processos que realizam muito E/S;
  - Tempo de execução é definido pela duração das operações de E/S;
- **IDEAL**: existir um balanceamento entre processos *CPU-bound* e *I/O-bound*;

Escalonador de Processos

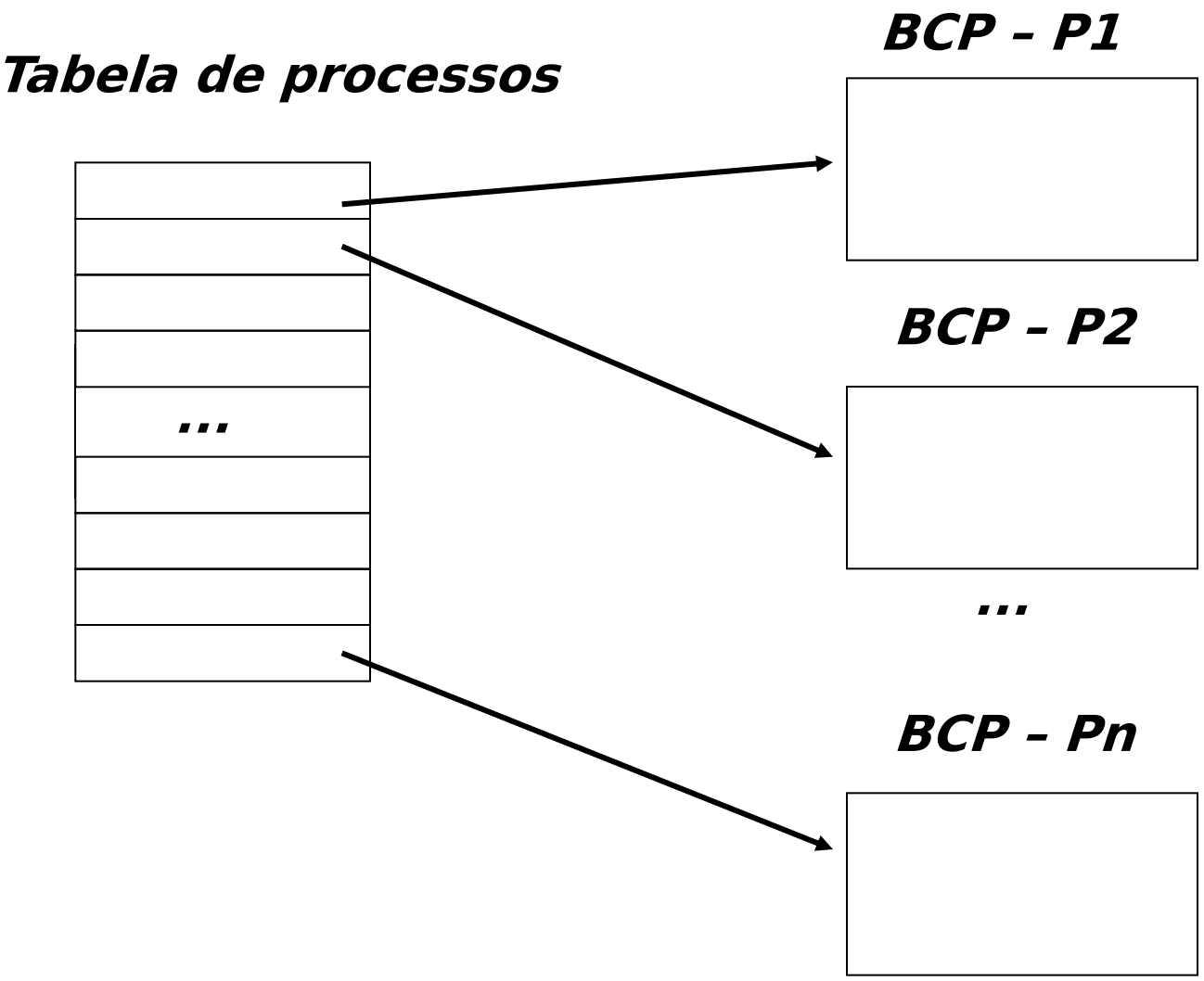


- Nível mais baixo do SO;
- Manipulação de interrupções e processos;

Implementação de Processos

- Tabela de Processos:
  - Cada processo possui uma entrada;
  - Cada entrada possui um ponteiro para o **bloco de controle de processo** (BCP) ou descritor de processo;
  - BCP possui todas as informações do processo → contextos de hardware, software, endereço de memória, etc.;

Implementação de Processos



Implementação de Processos

- Algumas das informações armazenadas no BCP:

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Escalonamento de Processos

- O **escalonador de processos** é responsável por escolher o processo que será executado pela UCP;
- O escalonamento é realizado com o auxílio do hardware;
- O escalonador deve se preocupar com a eficiência da UCP, pois o chaveamento de processos é complexo e custoso (afeta desempenho do sistema e satisfação do usuário);
- O escalonador de processo é um processo que deve ser executado quando da **mudança de contexto** (troca de processo);

Escalonamento de Processos

- Situações nas quais escalonamento é necessário:
  - Um novo processo é criado;
  - Um processo terminou sua execução e um processo pronto deve ser executado;
  - Quando um processo é bloqueado (semáforo, dependência de E/S), outro deve ser executado;
  - Quando uma interrupção de E/S ocorre o escalonador deve decidir por: executar o processo que estava esperando esse evento; continuar executando o processo que já estava sendo executado ou executar um terceiro processo que esteja pronto para ser executado;

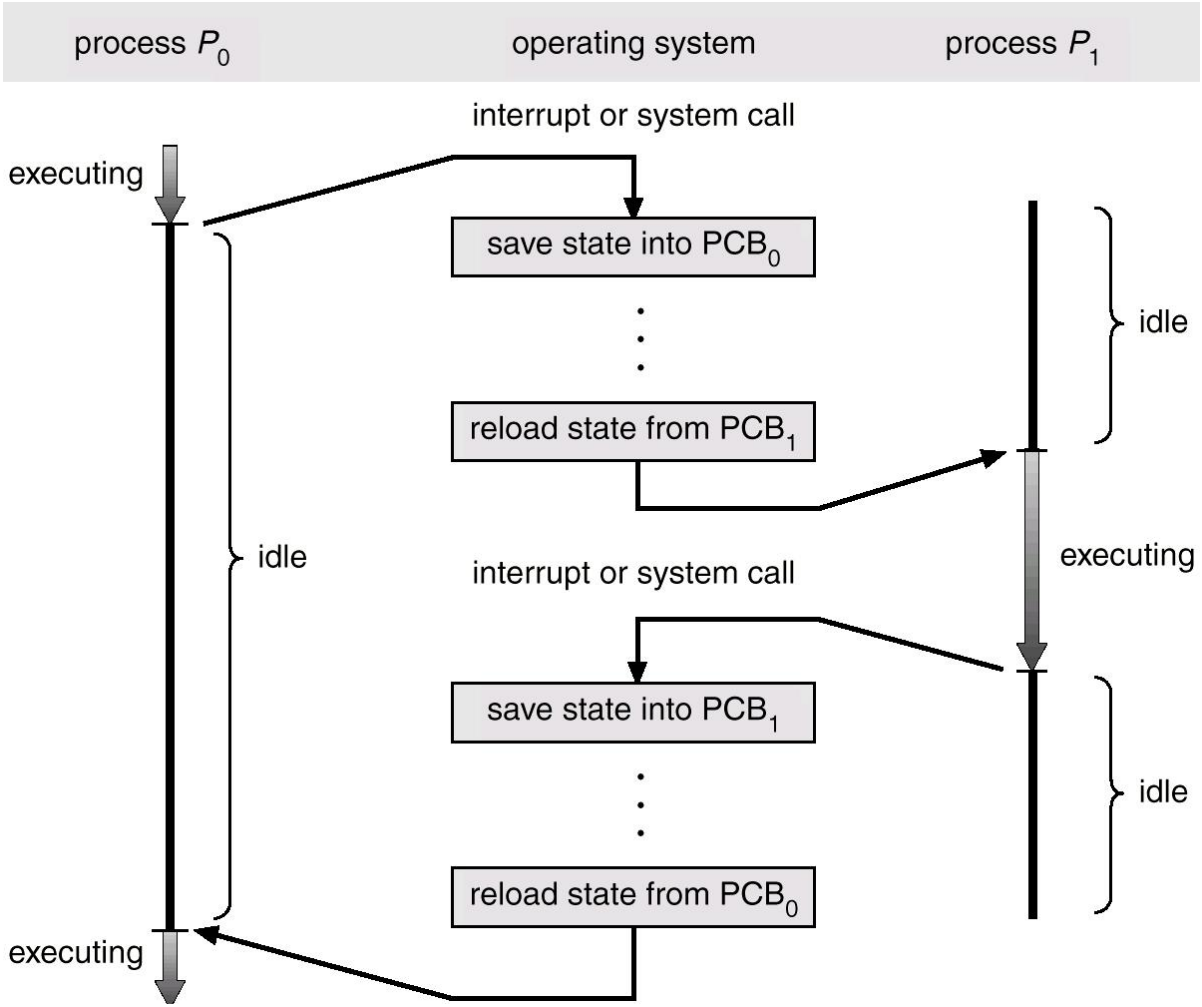
Escalonamento de Processos

- Hardware de relógio fornece interrupções de relógio e a decisão do escalonamento pode ser tomada a cada interrupção ou a cada *k* interrupções;
- Algoritmos de escalonamento podem ser divididos em duas categorias dependendo de como essas interrupções são tratadas:
  - **Preemptivo**: permite que um processo em execução seja interrompido, (por necessitar fazer uma operação de E/S, semáforos, etc.), e retome a execução do mesmo ponto que parou;
  - **Não-preemptivo**: uma vez que o processo recebe o direito de uso da UCP, ele a utiliza até o término do seu processamento; modelo padrão de antigos sistemas monoprogramados

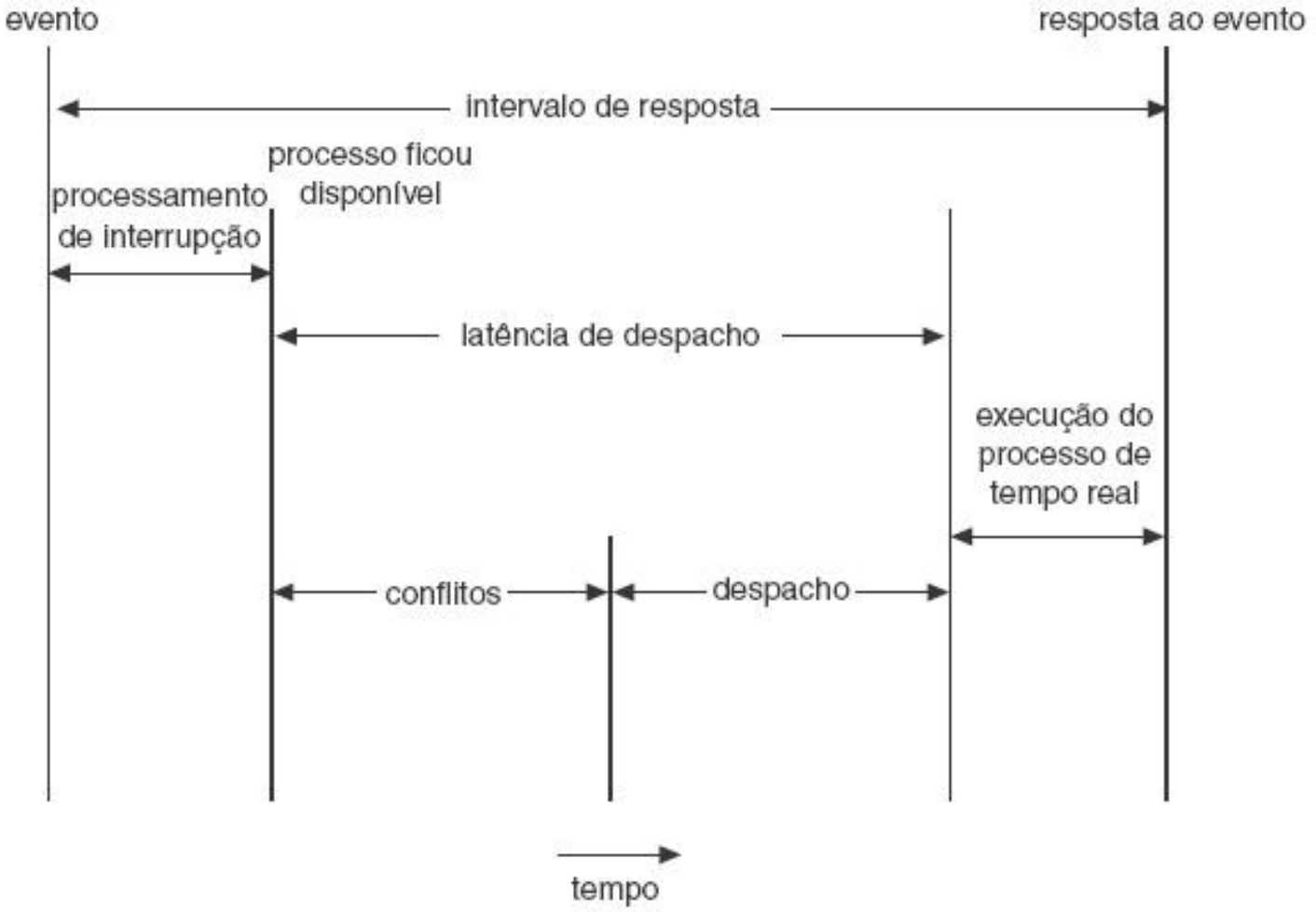
Escalonamento de Processos

- Troca de contexto:
  - *Overhead* de tempo;
  - Tarefa cara:
    - Salvar as informações do processo que está deixando a UCP em seu BCP → conteúdo dos registradores;
    - Carregar as informações do processo que será colocado na UCP → copiar do BCP o conteúdo dos registradores;

Troca de Contexto entre processos



Latência de Despacho



Escalonamento de Processos

***Antes* da Mudança de Contexto:**

BCP-P2

PC = 0BF4h
PID = 2
Estado = pronto

Próximo processo

BCP-P4

PC = 074Fh
PID = 4
Estado = executando

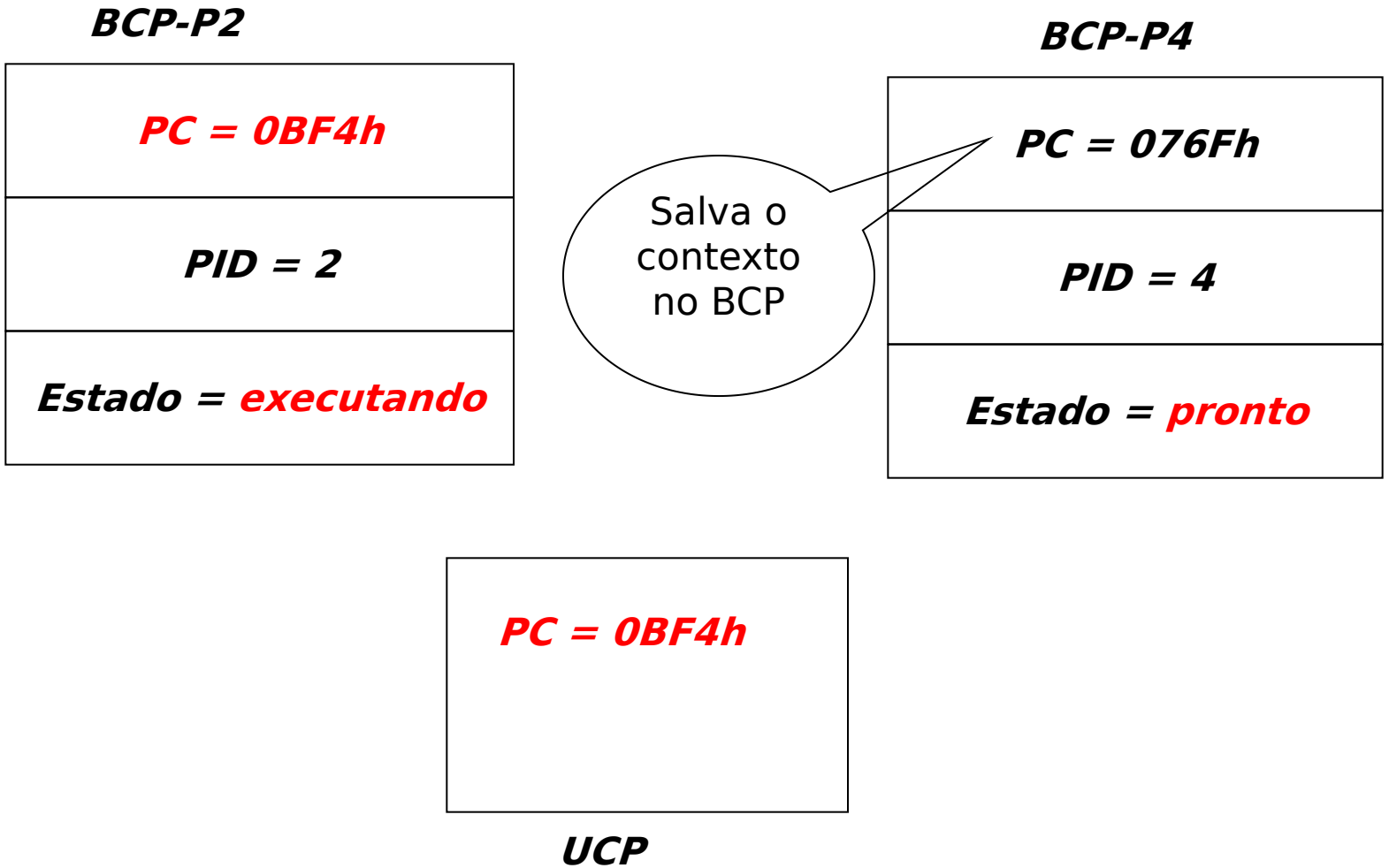
PC = 076Fh

UCP

Processo já executou algumas instruções

Escalonamento de Processos

***Depois* da Mudança de Contexto:**



Escalonamento de Processos

- Categorias de Ambientes:
  - **Sistemas em Batch**: usuários não esperam por respostas rápidas; algoritmos não-preemptivos ou preemptivos com longo intervalo de tempo;
  - **Sistemas Interativos**: interação constante do usuário; algoritmos preemptivos; Processo interativo → espera comando e executa comando;
  - **Sistemas em Tempo Real**: processos são executados mais rapidamente, pois o tempo é crucial → sistemas críticos;

Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Qualquer sistema:
    - **Justiça** (*Fairness*): cada processo deve receber uma parcela justa de tempo da UCP;
    - **Balanceamento**: diminuir a ociosidade do sistema;
    - **Políticas do sistema** – prioridade de processos;

Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Sistemas em *Batch*:
    - **Vazão** (*throughput*): maximizar o número de *jobs* executados por hora;
    - **Tempo de retorno** (*turnaround time*): tempo no qual o processo espera para ser finalizado;
    - **Eficiência**: UCP deve estar 100% do tempo ocupada;
  - Sistemas Interativos:
    - **Tempo de resposta**: tempo esperando para iniciar execução;
    - **Proporcionalidade**: satisfação do usuários;

Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Sistemas em Tempo Real:
    - **Cumprimento dos prazos**: prevenir perda de dados;
    - **Previsibilidade**: prevenir perda da qualidade dos serviços oferecidos;

Escalonador de UCP

- Seleciona um processo, entre os presentes na memória que estão prontos para execução, e o aloca à UCP;
- As decisões de escalonamento de UCP podem ocorrer quando um processo:

1. Passa do estado executando para o estado esperando ;
2. Passa do estado executando para o estado pronto;
3. Passa do estado esperando para o estado pronto ;
4. Termina.

Despachador (*dispatcher*)

- O despachador passa o controle da UCP ao processo selecionado pelo escalonador de curto prazo; isso envolve:
  - Troca do contexto;
  - Troca do modo do usuário;
  - Desvio para o local apropriado no programa do usuário a fim de reiniciar esse programa.
- *Latência de despacho* – tempo gasto para o despachante interromper um processo e iniciar a execução de outro

Critérios de Escalonamento

- **Utilização de UCP** – mantém a UCP ocupada pelo máximo de tempo possível;
- **Vazão** – número de processos que são completados por unidade de tempo;
- ***Turnaround*** – o tempo necessário para executar um determinado processo;
- **Tempo de espera** – tempo que um processo gasta esperando na fila de prontos;
- **Tempo de resposta** – tempo percorrido desde que uma requisição é submetida até a primeira resposta produzida, não até a saída (para ambiente de tempo compartilhado);

Critérios de otimização

- Utilização de UCP máxima;
- *Throughput* máxima;
- *Turnaround* mínimo;
- Tempo de espera mínimo;
- Tempo de resposta mínimo.

Escalonamento "First-Come, First-Served" (FCFS)

Processo	Burst
P_1	24
P_2	3
P_3	3

- Suponha que os processos cheguem na ordem: P_1 , P_2 e P_3. O diagrama de Gantt para o escalonamento será:

- Tempo de espera para $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo de espera médio: $T_{med} = (0 + 24 + 27)/3 = 17$ u.t.

Escalonamento "First-Come, First-Served" (FCFS)

- Suponha que os processos cheguem na ordem:

$$P_2 , P_3 , P_1$$

  - O diagrama de Gantt para o escalonamento será:

  - Tempo de espera para $P1 = 6; P2 = 0; P3 = 3$
  - Tempo de espera médio: $T_{med} = (6 + 0 + 3)/3 = 3$ u.t.
- Muito melhor do que o caso anterior
- Efeito comboio – processo curto atrás de processo longo

Escalonamento "Shortest Job First" (SJF)

- Associa a cada processo a duração do seu próximo *burst* de UCP. Usa esses tempos para escalonar o processo com o tempo mais curto

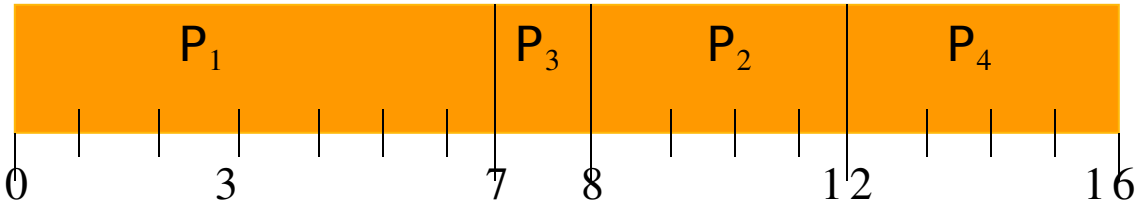
- Dois esquemas:
  - **Não-preemptivo** – uma vez que o direito de uso é associado a um processo, a UCP não pode ser escalonada até que complete seu *burst* de UCP
  - **Preemptivo** – se um novo processo chegar com tamanho de *burst* de UCP menor do que o tempo restante do processo atualmente em execução, ele é retirado. Esse esquema é conhecido como "*Shortest Remaining Time First*" (SRTF)

- O SJF é o ótimo – provê o menor tempo de espera médio para um determinado conjunto de processos

Exemplo de SJF não-preemptivo

Processo	Instante de chegada	Burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (não-preemptivo)

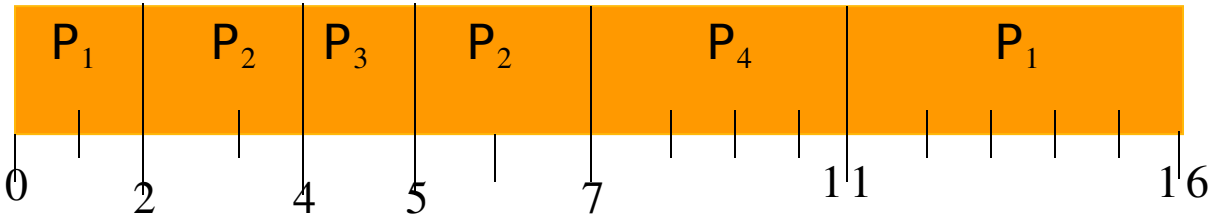


  - Tempo de espera médio = $(0 + 6 + 3 + 7)/4 = 4$ u.t.

Exemplo de SJF preemptivo

Processo	T. chegada	Burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptivo)



- Tempo de espera médio = (9 + 1 + 0 +2)/4 = 3 u.t.

Determinando o próximo *burst* de UCP

- Embora o SJF preemptivo mostre-se o algoritmo de escalonamento ideal, não há como ser implementado para o escalonador de curto prazo, por não haver uma forma de saber com precisão a duração do próximo *burst* de UCP
- Embora não possamos saber a duração do próximo *burst*, ela pode ser estimada, como uma média expoencial dos períodos dos *bursts* de UCP anteriores

Determinando o próximo *burst* de UCP

- Para isso, empregamos a fórmula;
- $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$
  - t_n: duração real do enésimo *burst* de UCP;
  - τ_{n+1}: duração prevista do próximo *burst* de UCP;
  - τ_n: última previsão de duração de *burst* de UCP;
  - α, $0 \leq \alpha \leq 1$

- Considerando $\alpha = 0$, então $\tau_{n+1} = \tau_n$, o que diz que o histórico recente não tem efeito
- Caso seja utilizado $\alpha = 1$, então $\tau_{n+1} = t_n$, e com isso apenas o burst de UCP mais recente é considerado
- Geralmente, é utilizado $\alpha = 1/2$, dessa forma tanto o histórico recente e o histórico passado têm pesos iguais

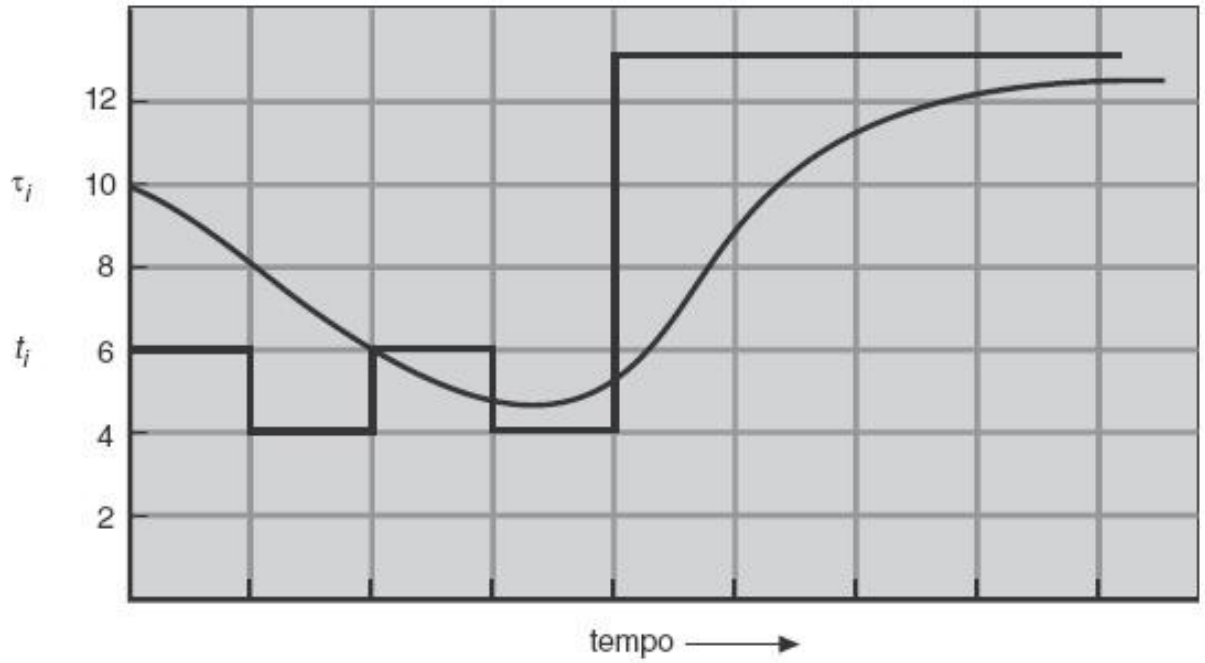
Determinando o próximo *burst* de UCP

- Por exemplo, considerando um 'chute' inicial $T_0 = 10$ (que pode ser definido como uma constante ou uma média geral do sistema) cujo tempo $t_0=6$:
- $T_1=\alpha t_0 +(1-\alpha)T_0$
- $T_1= 1/2 * 6 + (1- 1/2) 10$
- $T_1= 8$
- O burst real de t_1 foi igual a 4

- $T_2= \alpha\ t_1 + (1- \alpha)\ T_1$
- $T_2= 1/2 * 4 + (1- 1/2)\ 8$
- $T_2= 6$
- Nesta operação o burst real de t_2 foi igual a 6, o mesmo valor do "chute"

...

Previsão da duração do próximo *burst* de UCP



burst de CPU(t_i)	6	4	6	4	13	13	13	. . .
"chute"(τ_i)	10	8	6	6	5	9	11	12	. . .

Escalonamento por Prioridade

- Um valor de prioridade (inteiro) é associado a cada processo
- A UCP é alocada para o processo com a prioridade mais alta (menor inteiro = prioridade mais alta)
  - Preemptivo;
  - Não-preemptivo.

Escalonamento por Prioridade

- SJF é um escalonamento por prioridade em que a prioridade é o próximo menor tempo de *burst* de UCP previsto
- Problema → Estagnação (*starvation*) – processos de baixa prioridade podem nunca ser executados
- Solução → Envelhecimento – conforme o tempo passa, a prioridade de um processo não executado aumenta

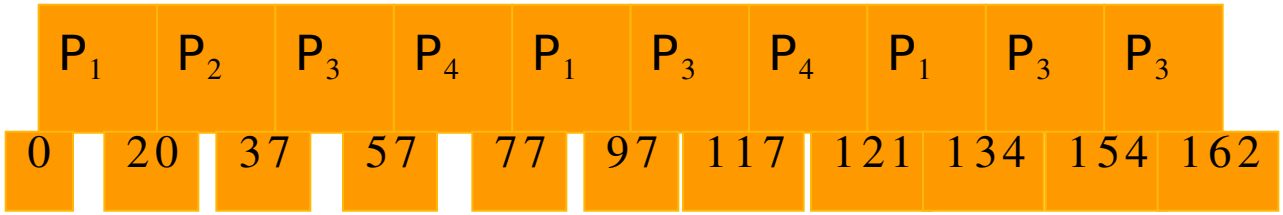
Algoritmo de escalonamento Round-Robin (RR)

- Cada processo obtém uma pequena unidade do tempo de UCP (quantum de tempo), normalmente 10-100 milissegundos. Após decorrido esse tempo, o processo é escalonado e colocado no final da fila de prontos
- Se houver *n* processos na fila de prontos e o quantum de tempo for igual a *q*, então cada processo recebe 1/n do tempo de UCP em, no máximo, *q* unidades de tempo de cada vez
- Nenhum processo espera mais do que $(n-1)*q$ unidades de tempo

Exemplo de RR com Quantum de Tempo = 20

Processo	Burst
P_1	53
P_2	17
P_3	68
P_4	24

- O diagrama de Gantt é:

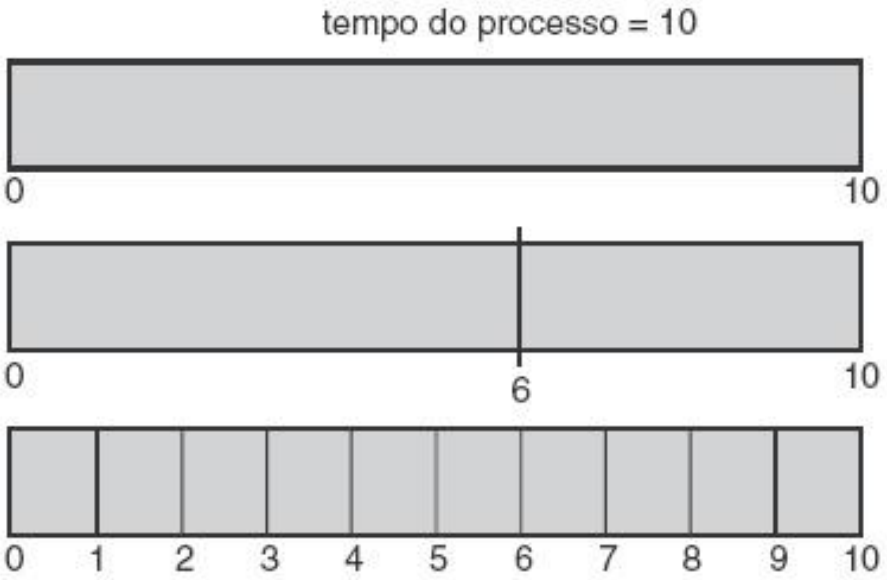


- T. de espera médio = $(0+57+24)+20+(37+40+17)+(57+60)/4 = 78$ u.t.
- Normalmente, o tempo de espera médio é mais alto do que SJF, mas melhora a resposta

Algoritmo de escalonamento Round-Robin (RR)

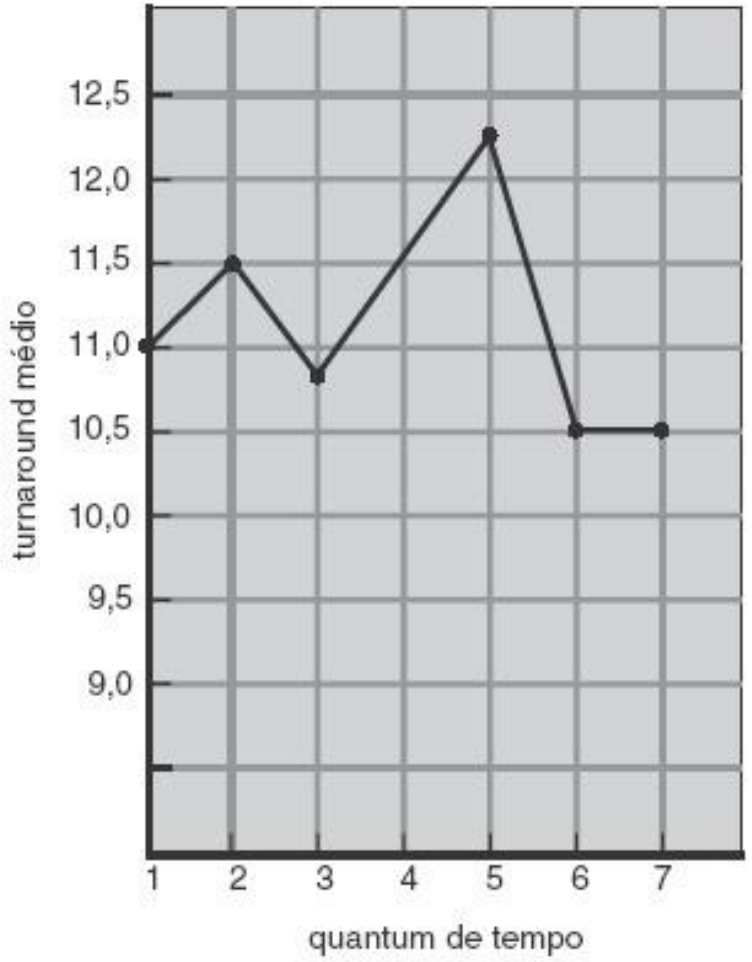
- Desempenho
  - quantum grande → FIFO
  - quantum pequeno → o fator de tempo *q* precisa ser grande com relação ao tempo de troca de contexto ou o custo adicional será muito alto

Quantum de Tempo e Tempo de Troca de Contexto



quantum	trocas de contexto
12	0
6	1
1	9

O *Turnaround Time* varia conforme o quantum de tempo



processo	tempo
P_1	6
P_2	3
P_3	1
P_4	7

Exercício

- Suponha que existam os seguintes processos para serem executados em um processador:

Processo	Tempo de execução	Tempo de chegada
A	12	0
B	6	3
C	2	5
D	5	8
E	9	11
F	8	13

Forneça o diagrama de Gantt e o tempo de espera médio para os métodos de escalonamento: FCFS, SJF (preemptivo e não-preemptivo) e Round Robin com *quantum* igual a 3.

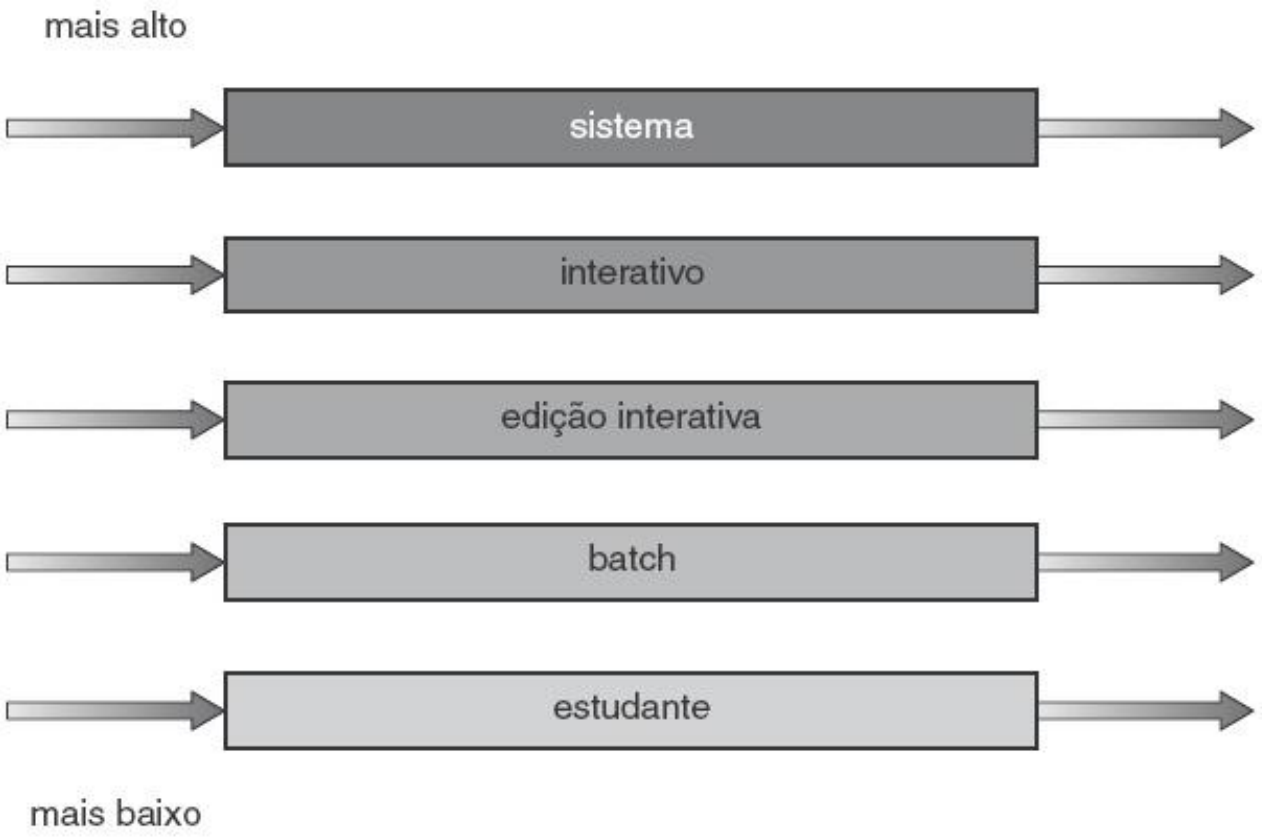
Multilevel Queue (fila multiníveis)

- Fila de prontos é dividida em filas distintas:
  - primeiro plano (interativa)
  - segundo plano (*batch*)
- Cada fila possui seu próprio algoritmo de escalonamento, por exemplo:
  - primeiro plano – RR
  - segundo plano – FCFS

Multilevel Queue

- É necessário haver escalonamento entre as filas:
  - Escalonamento de prioridade fixa; (ou seja, serve a todos a partir do primeiro plano e depois do segundo plano). Possibilidade de estagnação
  - Fatia de tempo – cada fila recebe uma certa parte do tempo de UCP, que pode ser escalonado entre seus processos; por exemplo, 80% para o primeiro plano no RR e 20% para o segundo plano no FCFS

Escalonamento *Multilevel Queue*



Multilevel *Feedback Queue*

- Um processo pode se mover entre as várias filas, permitindo que o envelhecimento pode ser implementado da seguinte forma;
- Escalonador da *multilevel feedback queue* é definido pelos seguintes parâmetros:
  - número de filas;
  - algoritmos de escalonamento para cada fila;
  - método usado para determinar quando elevar um processo;
  - método usado para determinar quando rebaixar um processo;
  - método usado para determinar em que fila um processo entrará quando esse processo precisar de atendimento;

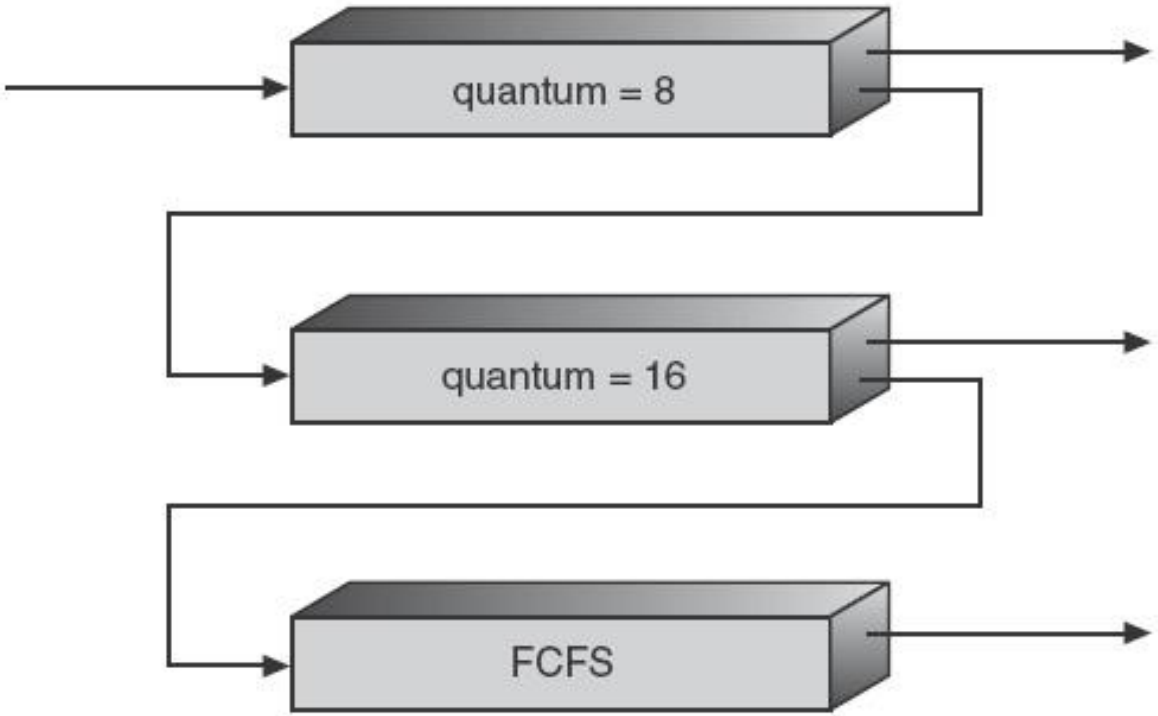
Multilevel *Feedback Queue*

- Por exemplo, considere um modelo com três filas:
  - Q_0 – quantum de tempo 8 milissegundos;
  - Q_1 – quantum de tempo 16 milissegundos;
  - Q_2 – FCFS;

- Escalonamento
  - Uma nova tarefa entra na fila Q_0, que é atendida com base no FCFS. Quando ganha a UCP, a tarefa recebe 8 milissegundos. Se não terminar nesse tempo, a tarefa é movida para a fila Q_1.

  - Em Q_1, a tarefa é atendida novamente com base no FCFS e recebe 16 milissegundos adicionais. Se ainda não estiver completa, a tarefa é apropriada e movida para a fila Q_2.

Multilevel Feedback Queue



Escalonamento em múltiplos processadores

- Escalonamento da UCP mais complexo quando várias UCPs estão disponíveis;
- Processadores homogêneos dentro de um multiprocessador;
- Compartilhamento de carga;
- Multiprocessamento assimétrico – apenas um processador acessa as estruturas de dados do sistema, reduzindo a necessidade de compartilhamento de dados;

Escalonamento em Tempo Real

- Sistemas de tempo real rígido – necessários para completar uma tarefa vital dentro de um período de tempo garantido;
- Computação em tempo real flexível – exige que processos vitais tenham prioridade sobre os menos importantes;

Avaliação de Algoritmo

- Modelagem determinística – define o desempenho de cada algoritmo para uma carga de trabalho predeterminada;
- Modelos de enfileiramento;
- Implementação;

Avaliação dos Escalonadores de CPU via Simulação