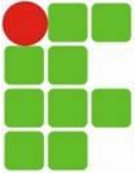
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIADO RIO GRANDE DO NORTE – IFRN
Disciplina: Fundamentos de Sistemas Operacionais e Sistemas Operacionais de Rede
Professor: Msc. Rodrigo Ronner T. da Silva
E-mail: rodrigo.tertulino@ifrn.edu.br

INTRODUÇÃO AOS SISTEMAS OPERACIONAIS

rodrigo.tertulino@ifrn.edu.br

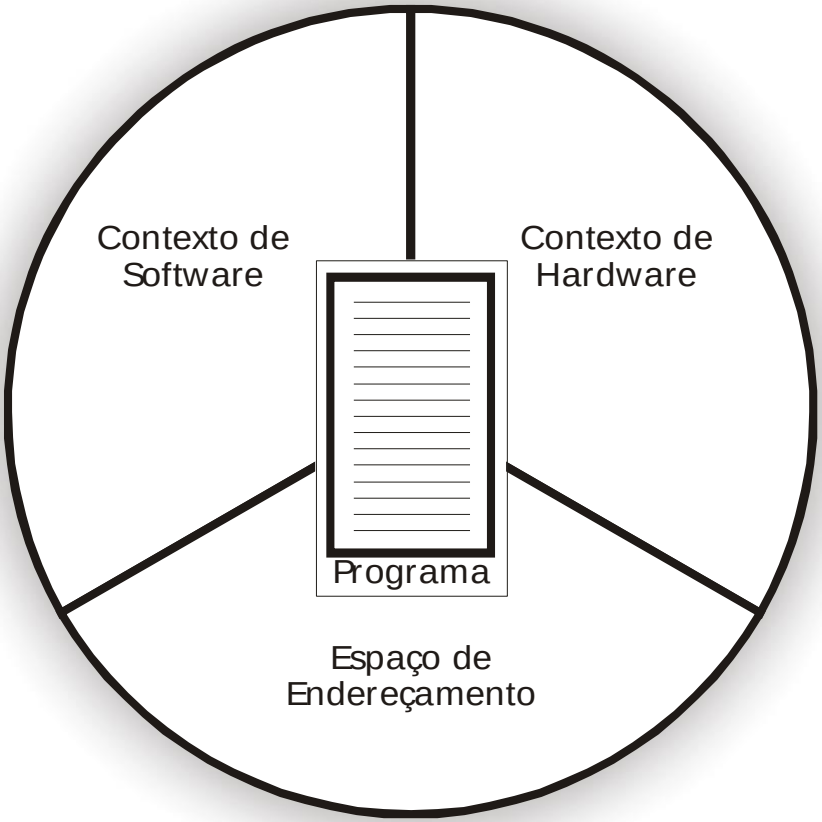


PROCESSOS

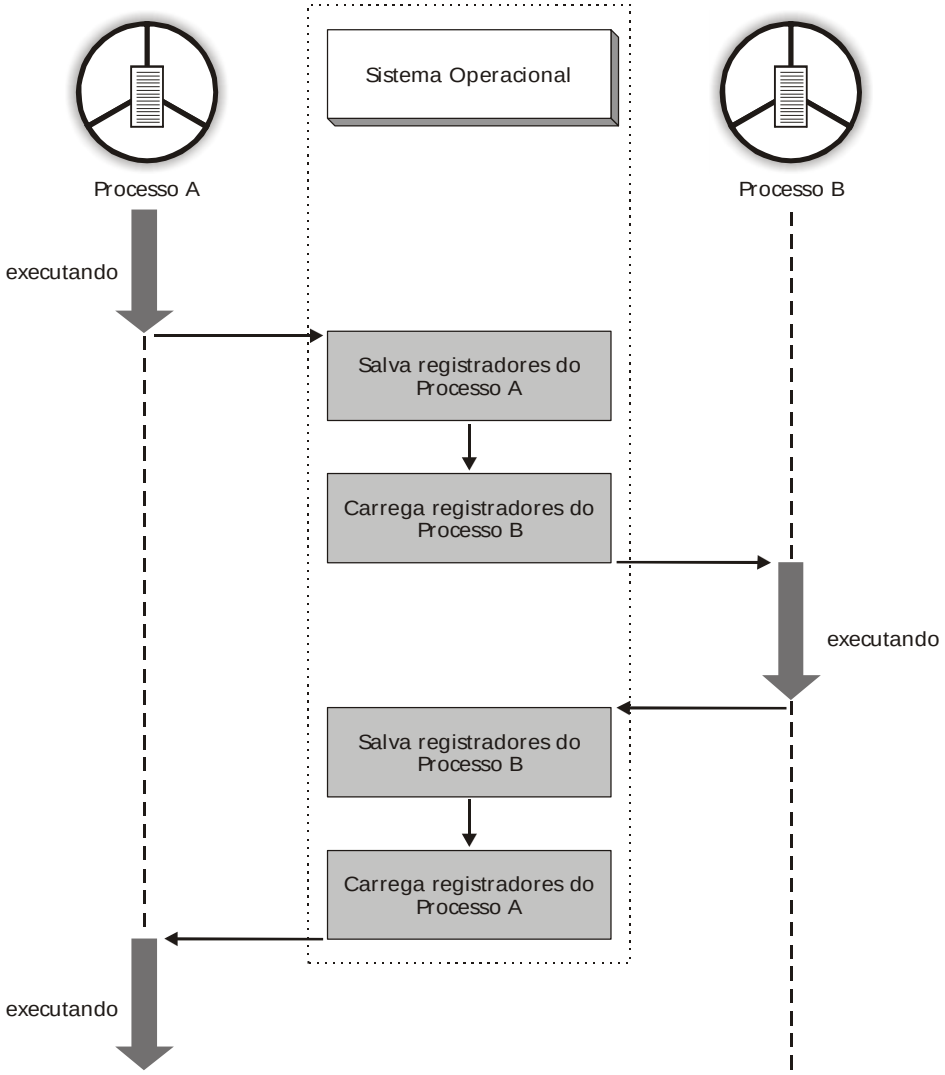
PROCESSOS

- **Programa:** Sequência de Instruções
- **Processos:** Programa em execução. Ou melhor é o ambiente onde se executa um programa.
- **A CPU executa o processo de uma tarefa por um tempo (time-slice)** e depois outro processo. **Quando a CPU** retorna a um processo já **executa**do, **é necessário recarregar as informações**.
- **PROCESSO:** Estrutura responsável pela manutenção de todas as informações necessárias para executar um programa.

Estrutura do Processo



Troca de Contexto



Contexto de Hardware

- **Armazena o conteúdo dos registradores de uso gerais e específicos da CPU**
  - **PC, IR, SP, Status Register**

- **Conteúdo destes registradores é salvo durante a troca de contexto (troca de processos) para posterior recuperação**

Contexto de Software

Especifica características e limites dos recursos alocados ao processo

Ex: Número máximo de arquivos abertos, prioridade de execução, tamanho de *buffer* de E/S, etc

Contexto de Software composto por 3 grupos:

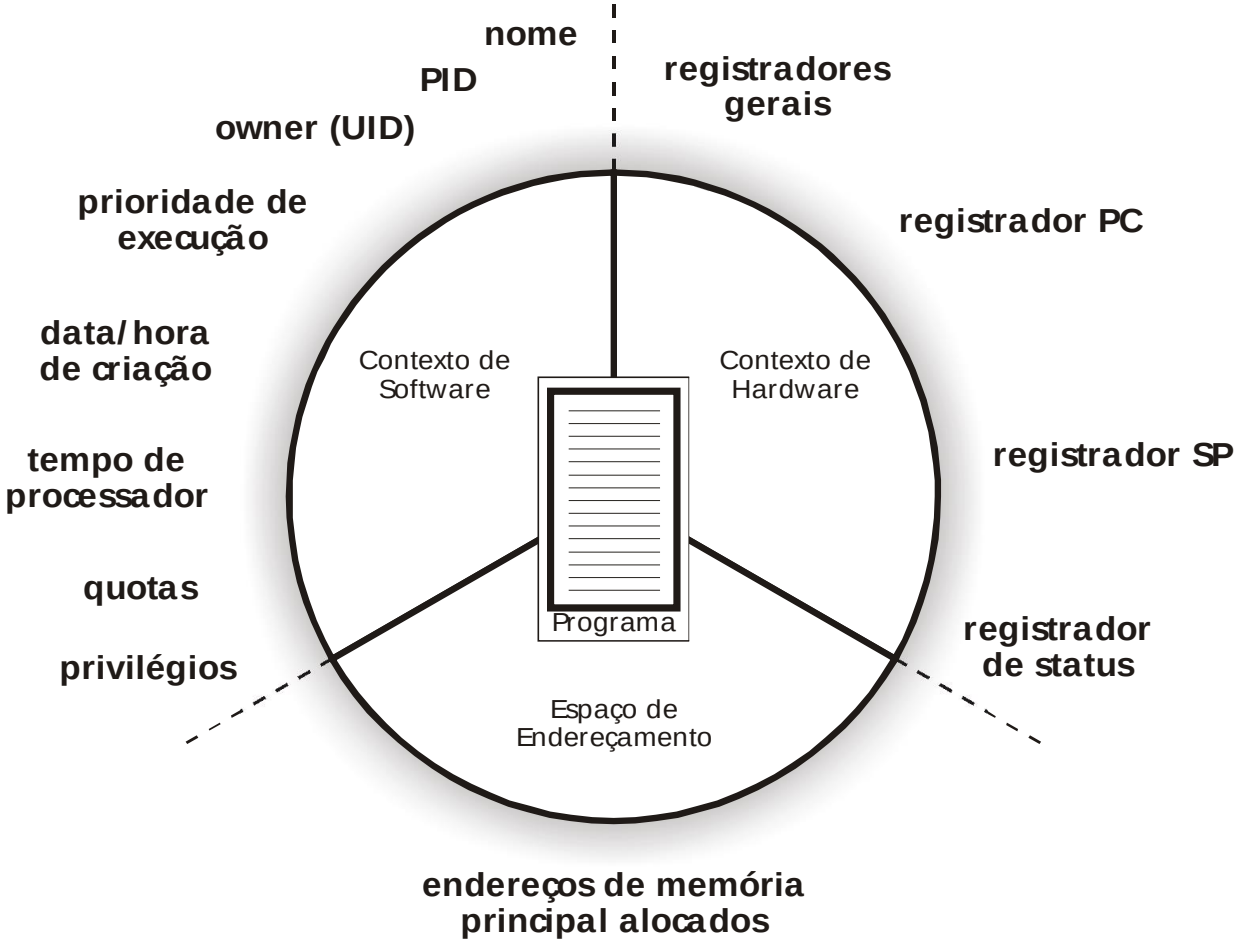
- **Identificação**
- **Quotas**
- **Privilégio**

Contexto de Software

- **Identificação** – única para cada processo e usuário
  - *Process Identification* (PID)
  - *User Identification* (UID)
- **Quotas** – limites de cada recurso p/uso do processo
  - Nmáx. de arquivos abertos, tamanho máx. de alocação de memória, Nmáx de operações de E/S, *buffer* máx p/ E/S, Nmáx de subprocessos, etc
- **Privilégios** – ações permitidas ao processo
  - Dividem-se em **privilégios** que afetam o próprio processo, afetam demais processos e afetam o próprio SO

Estrutura de um Processo

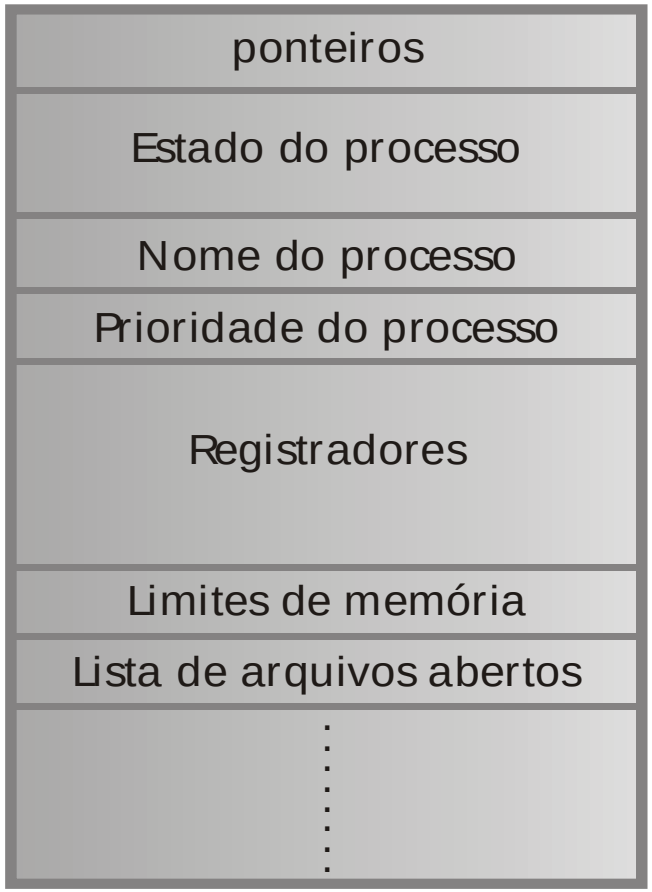
Detalhada



Bloco de Controle do Processo (PCB)

- Cada processo é implementado no SO através do *Process Control Block*
  - Cada PCB mantém todas as informações de contexto do respectivo processo
  - Todos os PCBs residem em área exclusiva da Memória Principal
    - Limitação desta área é parâmetro do SO, assim como o Nmáx de processos
  - Gerência de processos realizada exclusivamente através de *System Calls*
    - Criação, alteração, eliminação, suspensão, sincronização, etc

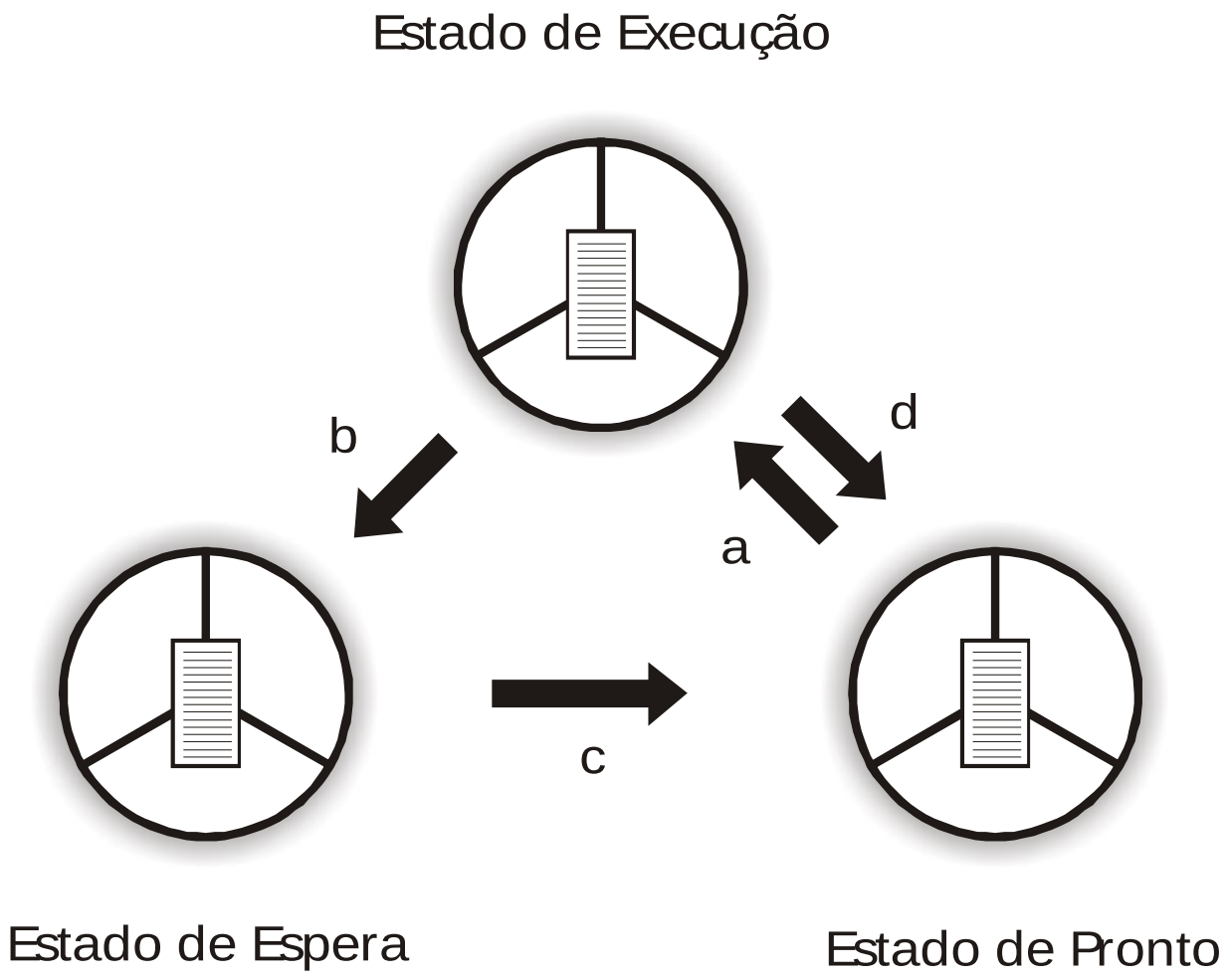
Bloco de Controle do Processo (PCB)



Estados do Processo

- **Estado de execução**
  - **Processo que está sendo executado pela CPU**
- **Estado de pronto (*ready*)**
  - **Processo aguardando para ser executado**
  - Geralmente organizados em listas encadeadas
  - **Escalonamento** da fila (lista) a **critério do SO**
- **Estado de espera (*wait* ou *blocked*)**
  - **Processo que aguarda algum evento externo ou liberação de recurso (ex: operação de E/S, relógio)**
  - Organizados em listas encadeadas

Mudanças de Estado do Processo



Mudanças de Estado do Processo

- Processos mudam de estado em função de eventos gerados por ele próprio ou pelo SO
- Processos não mudam de qualquer estado para qualquer estado. As possibilidades são:
  - De pronto para execução
  - De execução para espera
  - De espera para pronto
    - Nunca vai direto para estado de execução
  - De execução para pronto
    - Em geral, quando ocorre interrupção devido ao término da fatia de tempo alocado para o processo que estava sendo executado

Mudanças de Estado do Processo

- Processos no estado de pronto ou no estado de espera podem estar residentes ou não na MP
  - Processos não-residentes na MP ficam armazenados na memória secundária (disco)
    - Essa situação ocorre quando não há espaço suficiente na MP para todos os processos
  - Técnica para armazenamento de processos na memória secundária e reinserção dos mesmos na memória principal chamada de swapping

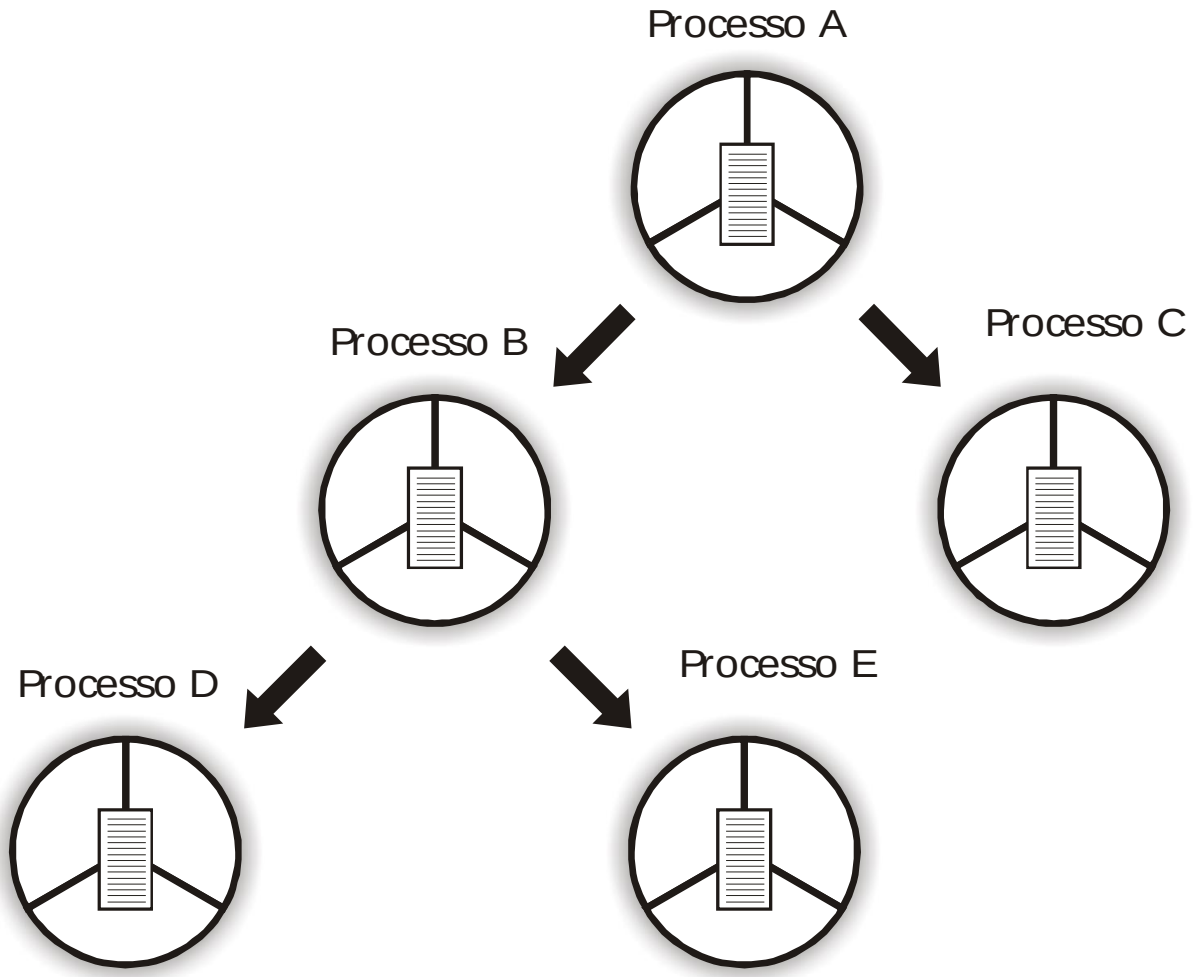
Processos Independentes, Subprocessos e Threads

- Maneiras diferentes de implementar concorrência numa mesma aplicação
- Processos Independentes
  - Maneira mais simples
  - Não há vínculo entre processo criado e processo criador
  - Exige criação de novo PCB e alocação própria de espaço de endereçamento

Processos Independentes, Subprocessos e Threads

- Subprocessos
  - Processos e subprocessos criados dentro de uma estrutura hieráquica
    - Processo pai e processo filho
  - Dependência entre processos pai e filho
    - Término do processo pai elimina respectiva estrutura de subprocessos filhos
    - Subprocesso possui PCB e espaço de endereçamento próprio, mas pode compartilhar quotas com processo pai

Estrutura de Processos e Subprocessos



Políticas de escalonamento

Políticas de escalonamento (agendamento)

- SO escolhe qual processo da fila de PRONTOS irá executar
- Despachante designa um processador a um dado processo
- Política de escalonamento (ou disciplina de escalonamento):
  - Critério do SO para escolher o processo que executará

Nível de ESCALONAMENTO	**Alto** (de admissão)	**Intermediário** (de memória)	**Baixo** (de CPU)
Prazo de atuação	Longo	Médio	Curto
Seleciona onde?	HD	Memória	PRONTOS
Qual processo?	Entra na RAM (prontos)	Sofrerá SWAP (out/in)	Irá executar

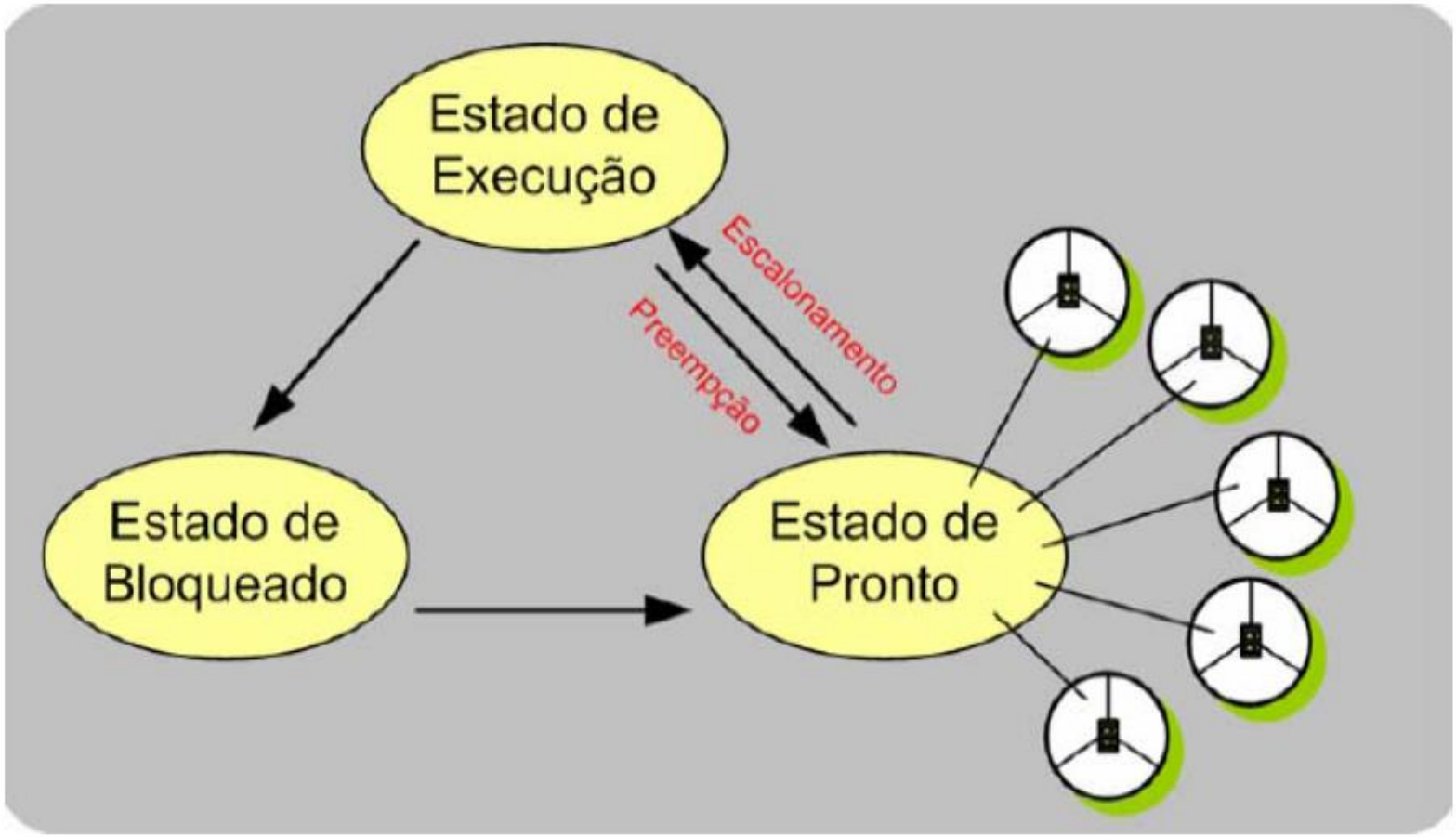
Políticas de escalonamento (agendamento)

- Forte influência no desempenho global do SO
- Deve garantir: justiça, previsibilidade e escalabilidade
- Deve considerar o comportamento de um processo
  - Processos CPU-Bound x I/O-Bound
  - Processos Em Lote x Interativo

Objetivos do escalonamento (em relação aos processos)

- Maximizar: throughput (total de processos terminados na unidade de tempo)
- Minimizar: turnaround (tempo da criação até o término do processo)
- Maximizar: taxa de utilização de CPU (tempo total de ocupação da CPU)
- Minimizar: tempo de resposta (tempo da criação até o inicio da execução)
- Minimizar: tempo de espera (soma dos tempos na fila de prontos)
- Minimizar: tempo de resposta em processos interativos
- Maximizar: utilização dos recursos (HW ou SW)
- Minimizar: a sobrecarga de gerenciamento do SO
- Favorecer: rotinas de maior prioridade e importância
- Garantir: previsibilidade no atendimento

Escalonamento



Políticas de escalonamento (agendamento)

- Poderá ser classificada como:
  - Preemptiva
  - Não-preemptiva
- Não-preemptiva (cooperativo)
  - SO não pode interromper um processo
  - Processo executa até concluir (ou parar voluntariamente)
- Preemptiva:
  - SO pode interromper um processo a qualquer instante
  - Para executar outro processo (chaveamento de contexto)
- Há vantagens e desvantagens em ambas

Políticas de escalonamento com prioridades

- Permite quantificar a importância relativa dos processos
- Mecanismos de prioridades são de dois tipos:
  - Prioridade estática
  - Prioridade dinâmica

- Prioridade estática:
  - Não varia ao longo da execução do processo
  - Fácil de implementar

- Prioridade dinâmica (mais inteligente)
  - Varia ao longo da execução do processo
  - Permite uma maior responsividade à mudanças no ambiente
  - Exemplos: Técnica de aging, algoritmo HRRN

- Inversão de prioridade:
  - SO aumenta a prioridade de um processo menos importante
  - Diminuir sobrecarga ou liberar recursos para outro processo

Políticas de escalonamento: critérios

- Tentam implementar "justiça" na escolha dos processos
- Primeiros SO eram colaborativos: ineficientes!
  - Exemplo: Windows 95 (multitarefa cooperativa)
  - Processo rodava o tempo que desejava
- Regra geral:

1º: Processo que chega vai para o final da fila pronto

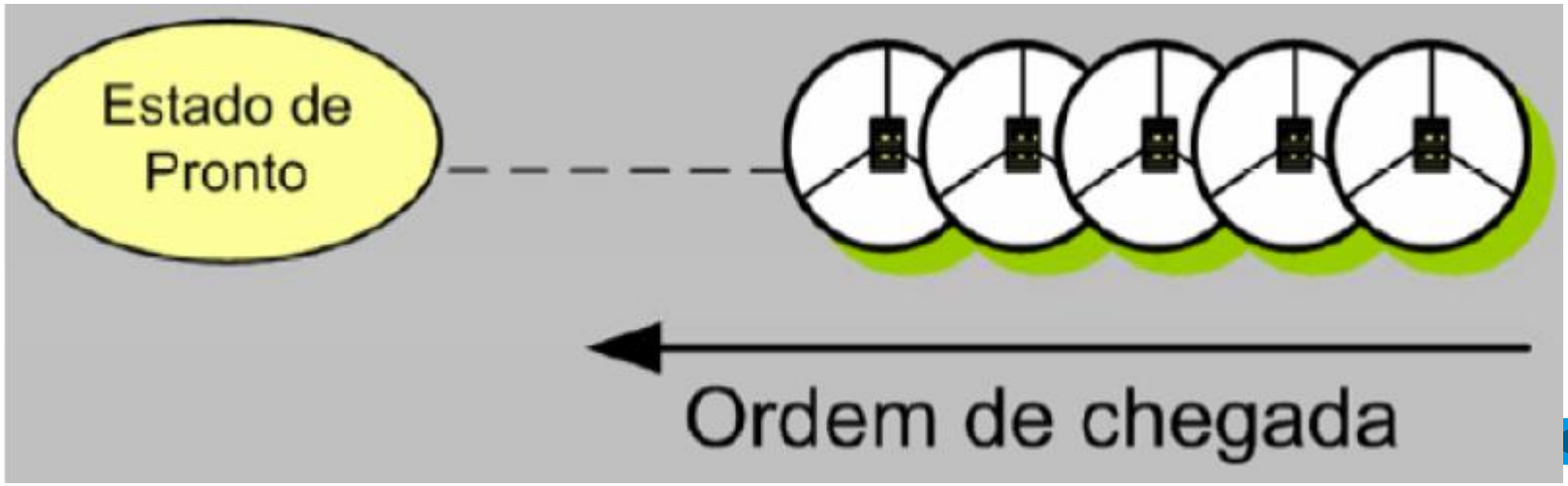
2º: Fila é rearranjada conforme o critério de escalonamento

- Atualmente, na prática, algoritmos usados nos SO:
  - Combinam dois ou mais critérios e são adaptativos
  - Variam dinamicamente conforme os estados dos processos

Políticas de escalonamento: critérios

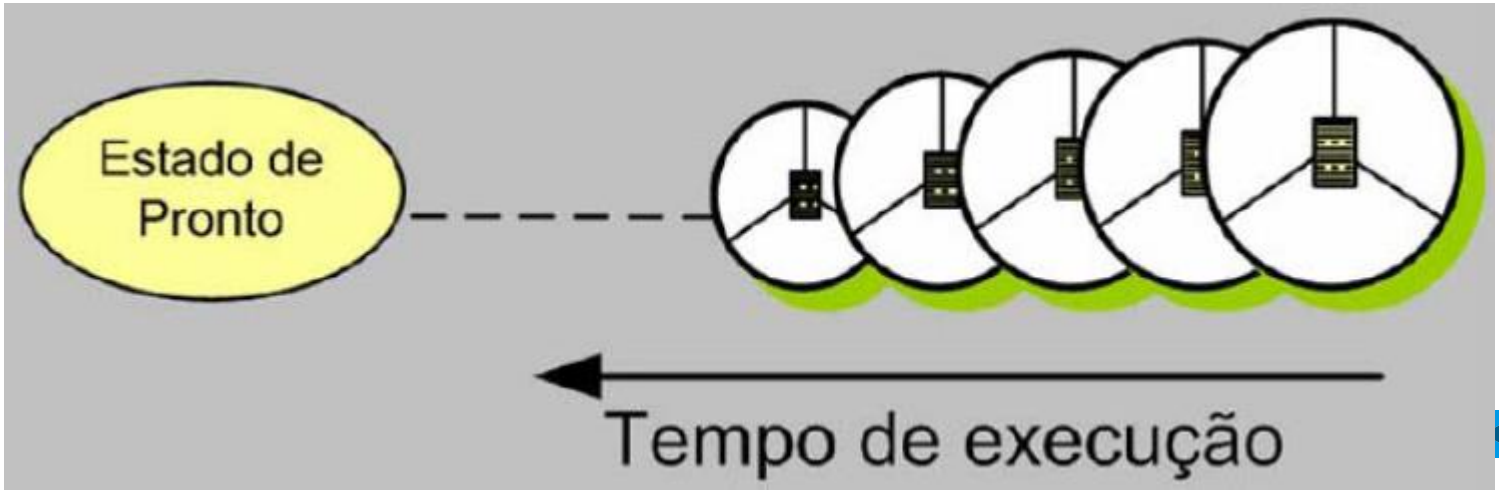
FIFO: First In First Out (First Come First Served)

- Critério: ordem de chegada
- Não-preemptivo (processo executa enquanto quiser)
- Vantagens
  - Justo: atende pela ordem de chegada
  - Impede adiamento indefinido
  - Fácil de implementar
- Desvantagens
  - Processos longos fazem os curtos esperarem muito
  - Não se mostra eficiente para processos interativos
  - Não considera a importância de uma tarefa



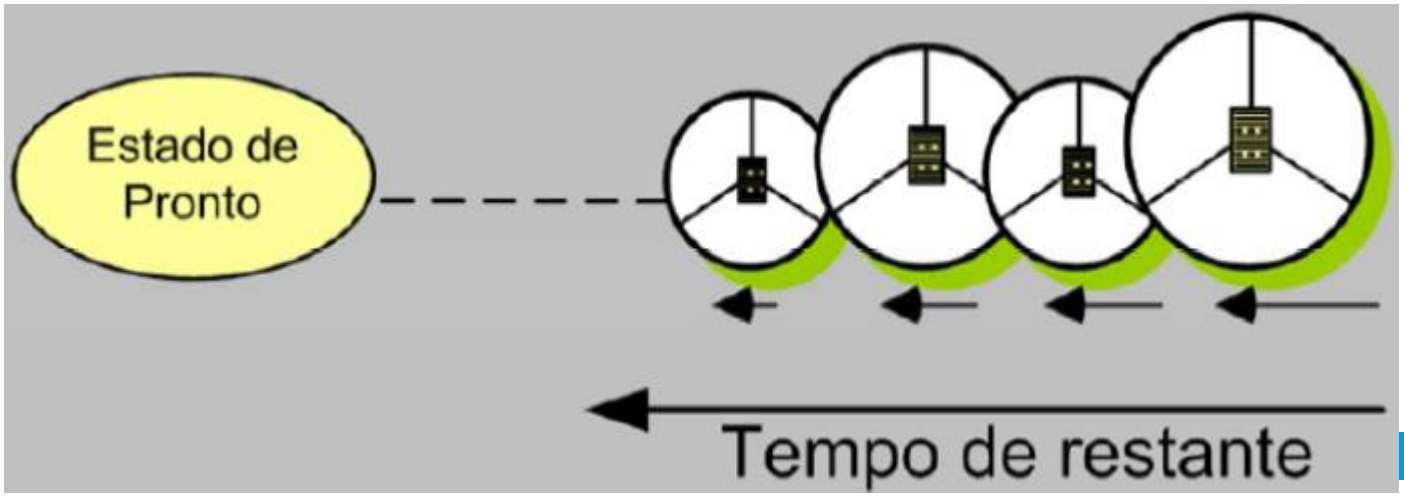
SPF: Shortest Process First (Shortest Job First)

- Critério: tempo de execução restante (burst): menor primeiro
- Não-preemptivo
- Vantagens
  - Favorece os processos mais curtos
  - Aumenta o rendimento (throughput)
  - **Menor tempo médio de espera**
- Desvantagens
  - Baseado em estimativas de tempo
  - Maior variância no tempo de espera (+ imprevisibilidade)
  - Não impede o adiamento indefinido



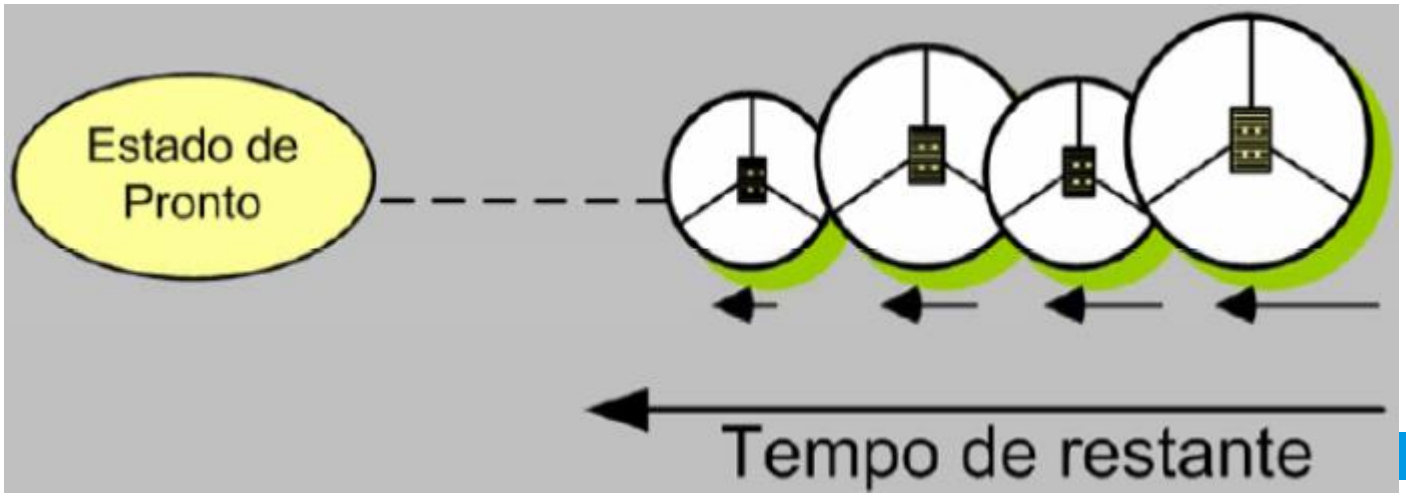
SRT: Shortest Remaining Time

- Critério: tempo de execução restante (burst): menor primeiro
- Preemptivo (se chegar processo de menor burst)
- Versão preemptiva do SPF
- Vantagens
  - Busca minimizar tempo de espera
- Desvantagens
  - Baseado em estimativas de tempo
  - Gera sobrecarga desnecessária nas troca de contexto
  - Usado em SO antigos (para processamento em lote)



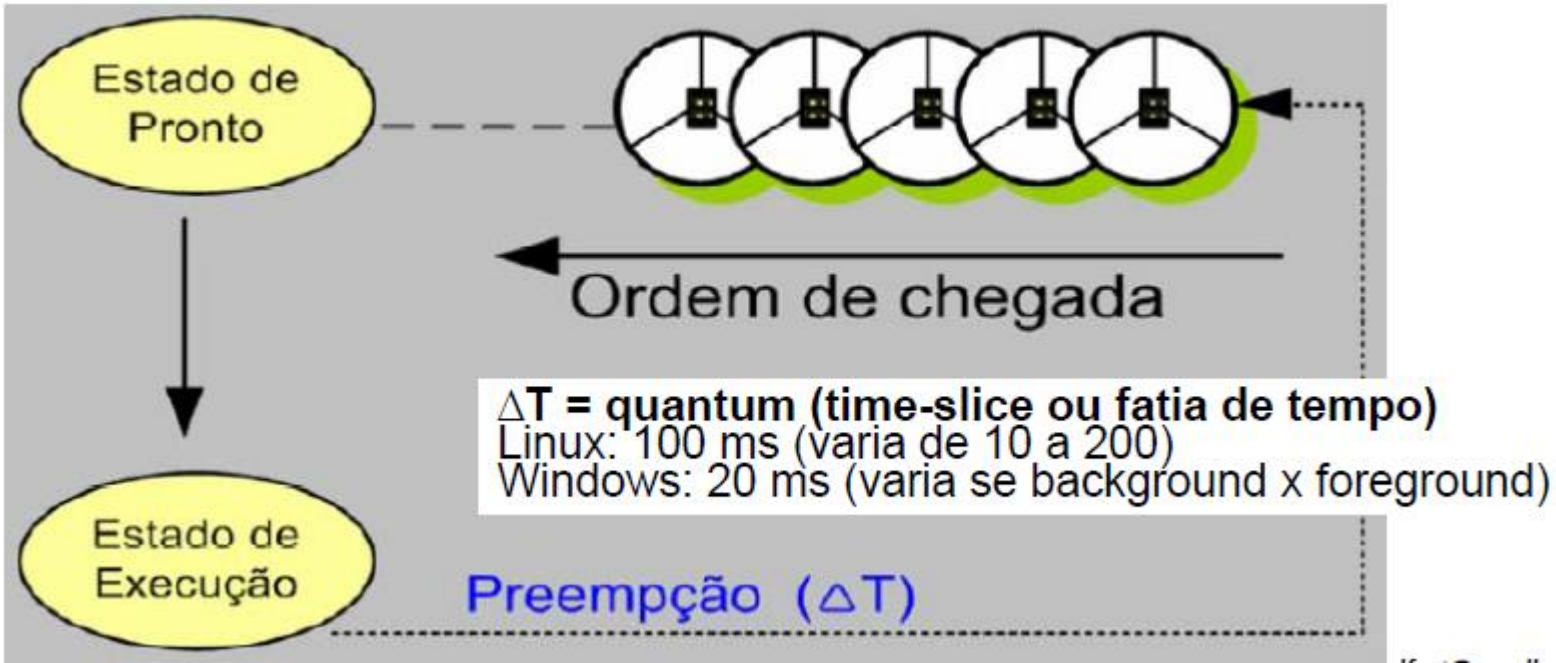
SRT: Shortest Remaining Time

- Critério: tempo de execução restante (burst): menor primeiro
- Preemptivo (se chegar processo de menor burst)
- Versão preemptiva do SPF
- Vantagens
  - Busca minimizar tempo de espera
- Desvantagens
  - Baseado em estimativas de tempo
  - Gera sobrecarga desnecessária nas troca de contexto
  - Usado em SO antigos (para processamento em lote)



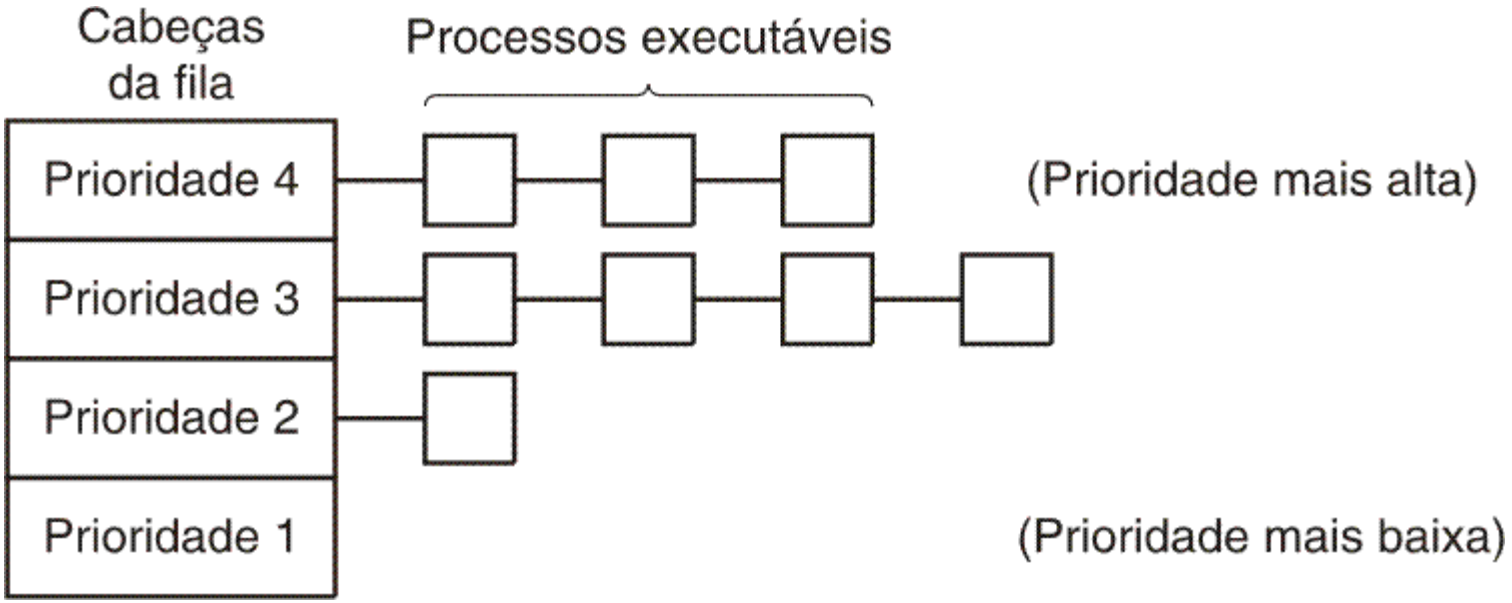
RR: Round Robin (alternância circular)

- Critério: ordem de chegada (como no FIFO)
- Preemptivo (por quantum de tempo)
- Vantagens
  - Efetivo com processos interativos
  - Impede adiamento indefinido
- Desvantagens
  - Mais complexo que FIFO
  - Adiciona sobrecarga no chaveamento de contexto



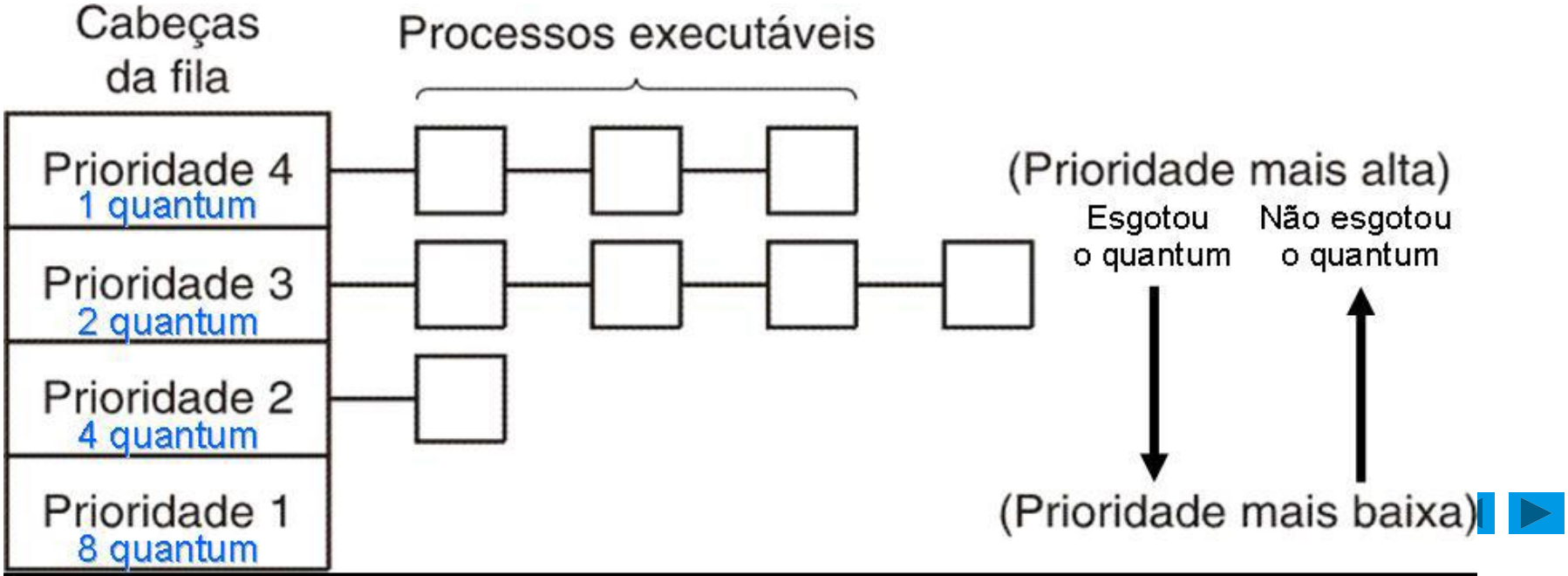
Por Prioridades (Filas de Prioridades)

- Critério: maior prioridade primeiro
- Preemptivo (por tempo e/ou prioridade de execução)
- Prioridade por ser fixa ou dinâmica
- Agrupa processos em filas de prioridades decrescentes
  - Filas mais altas são executadas primeiro (na totalidade)
  - Em cada fila, aplica-se RR (Round Robin)



Filas Múltiplas (multinível com feedback)

- **Critério:** Filas de prioridade distintas e quantum crescente
- **Preemptivo:** Em cada fila usa RR ou outro critério
- **Filas mais altas são executadas primeiro (na totalidade)**
- **Processos mudam de fila:** pelo uso do quantum
  - Se esgota, desce (menor prioridade)
  - Se não esgota, sobe (maior prioridade) ou mantém a fila
- **Vantagens**
  - Mais justo e inteligente que os algoritmos básicos
  - Prioriza processos interativos e rápidos (ou IO-Bound)



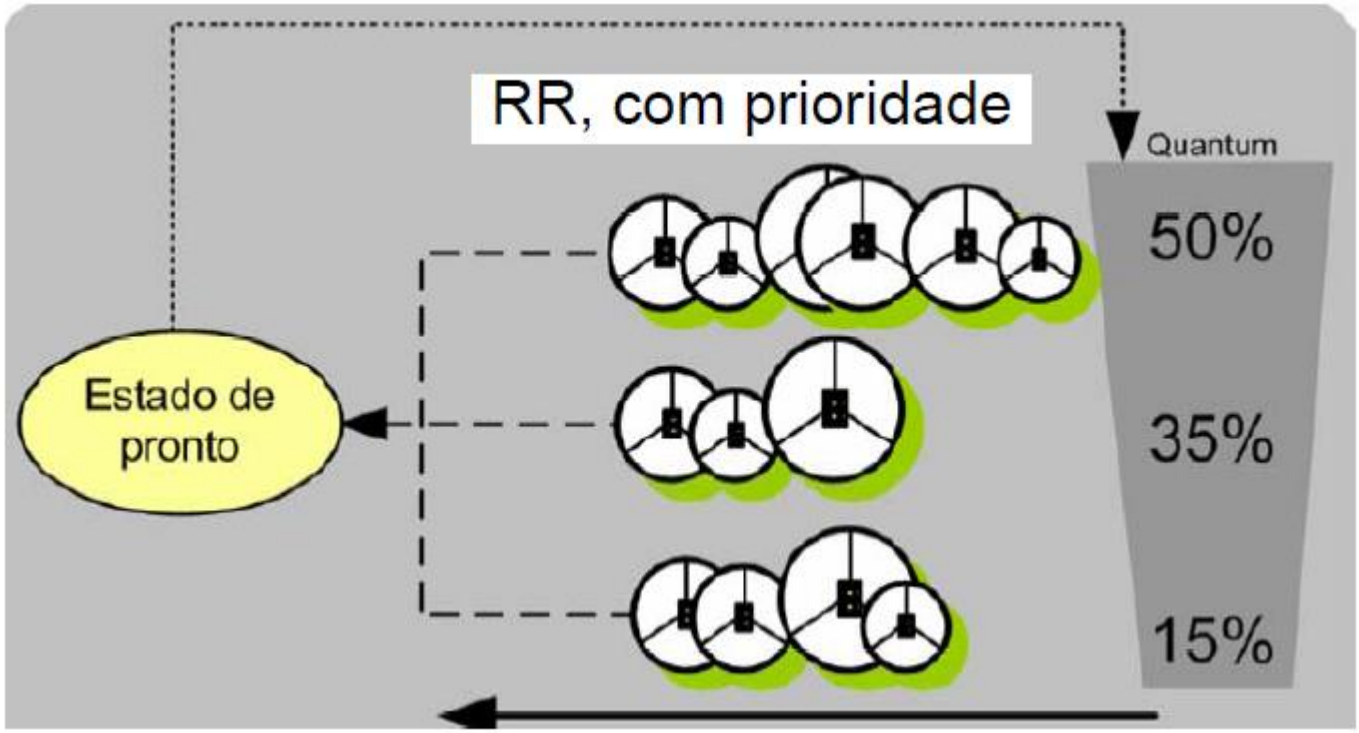
Fração Justa (Fair-share)

- Critério: escalonamento de dois níveis
  - Filas organizadas por grupos de usuários (ou de processos)
  - Cada grupo recebe um quantum diferente (fixo)
- Preemptivo (em cada fila usa RR, com prioridade)
  - Prioridade do processo, em cada fila = maior valor de:

$$\frac{\text{Tempo de uso recente}^* + \text{Tempo total de uso}^*}{\text{Tempo total de uso}^*}$$

- Processo não muda de fila, mas muda de posição na fila
- Vantagens
  - Mais justo e inteligente: prioriza processos mais importantes
  - Priorização
- Processos CPU-BOUND
- Mais recentemente executados
- Desvantagens
  - Complexidade de implementação

Fração Justa (Fair-share)

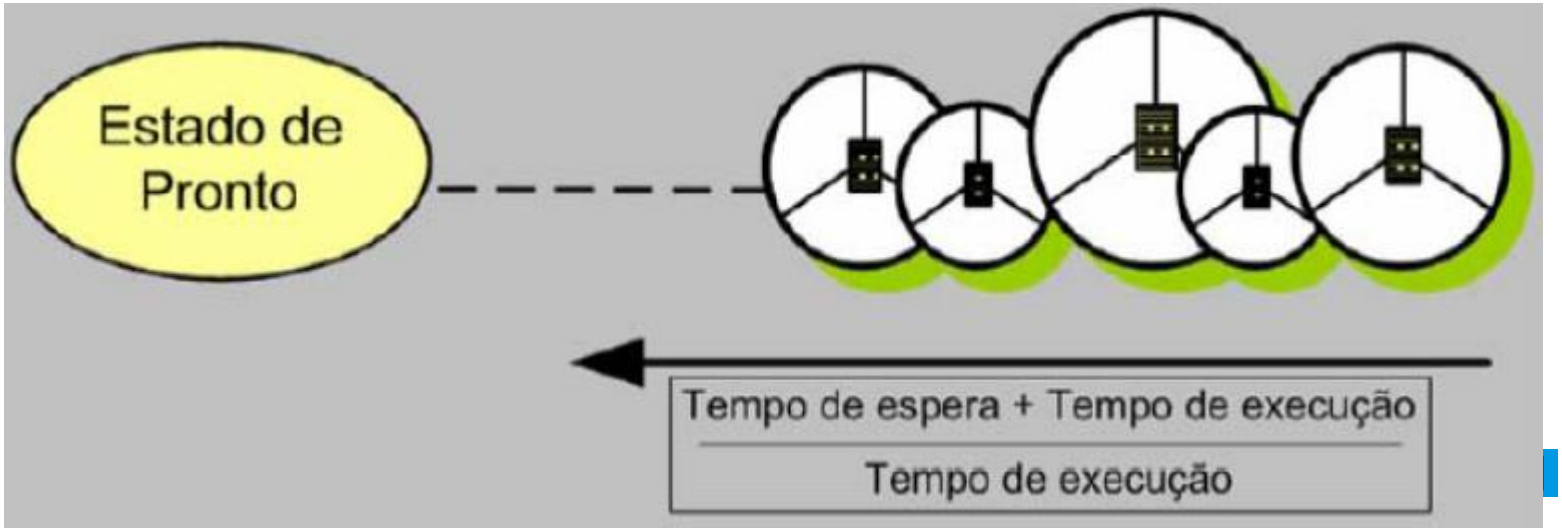


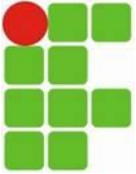
HRRN: Highest Response Rate Next

•**Critério: valor da relação abaixo (maior primeiro):**

$$\frac{\text{Tempo de espera na fila + Tempo de execução restante}}{\textbf{Tempo de execução restante}}$$

•Não-preemptivo
•Busca corrigir algumas deficiências do SPF
•Vantagens
–Prioriza processos mais antigos (maior burst): aging
–Elimina o adiamento indefinido
•Desvantagens
–Não leva em consideração a importância dos processos





Outras políticas de escalonamento

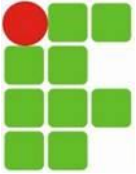
- Escalonamento Garantido
  - Cumpre promessas feitas a usuários (% alocação de CPU)
- Escalonamento por Loteria
  - SO distribui tokens (fichas) numerados entre os processos
  - Escalonador sorteia um número aleatório
  - Processos com mais tokens têm mais chance de escolha
- Vantagens
  - Altamente responsivo (ao número de tokens distribuídos)
  - Ideal para processos cooperativos (doação de tokens)

Escalonamento de Tempo Real

- Ocorre em SO de tempo real;
- Prioriza os processos em detrimento do próprio SO;
- Busca produzir resultados em tempos determinados;
- Divide-se em 2 tipos
  - Crítico: prazos devem ser rigorosamente cumpridos;
  - Não-crítico: descumprimentos de prazo são tolerados.

Resumo: Políticas de Escalonamento

Não-preemptivos	Preemptivos
FIFO (FCFS)	SRT
SPF (SJF)	RR
HRRN	por Prioridades
	Filas Múltiplas
	Fração Justa
	Tempo Real

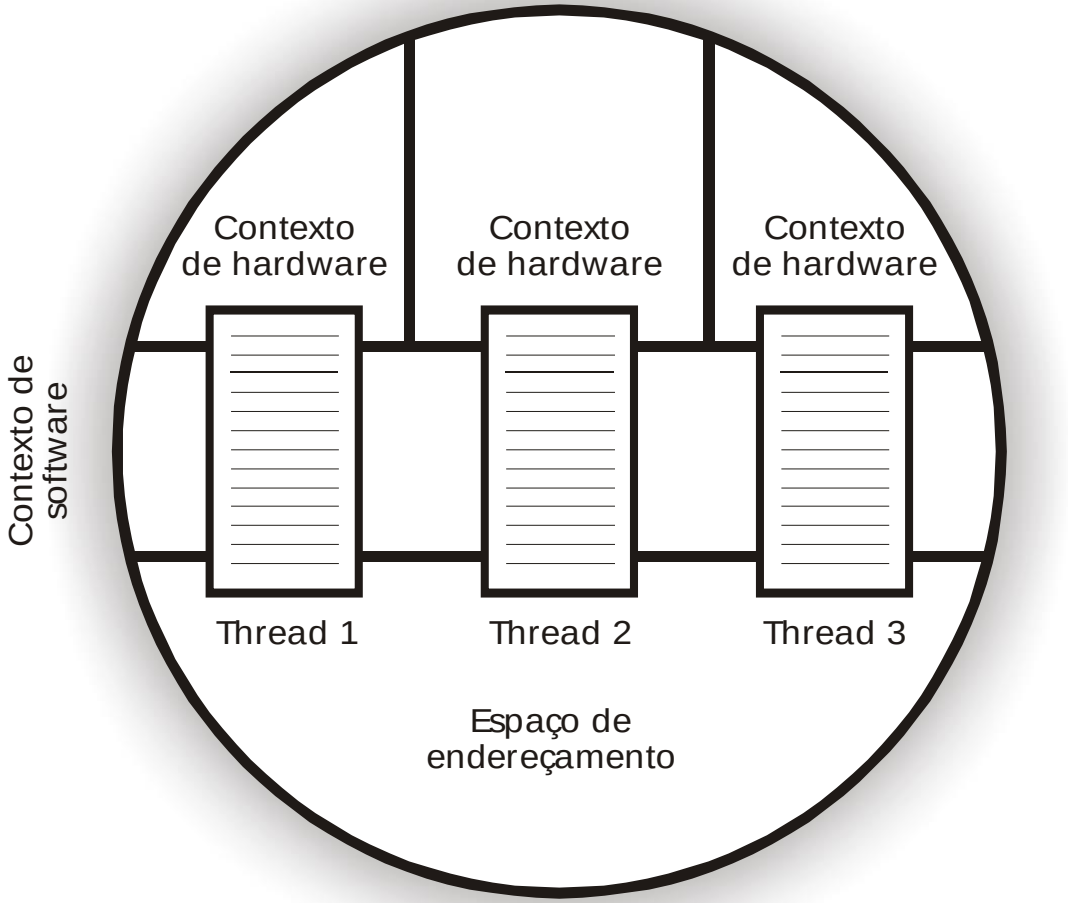


Thread

Threads

- Compartilham mesmo contexto de software e espaço de endereçamento
- Reduz tempo de criação, eliminação, comunicação e troca de contexto entre processos, economizando recursos do sistema
- Um processo pode suportar múltiplas threads, cada qual associada a uma parte do código

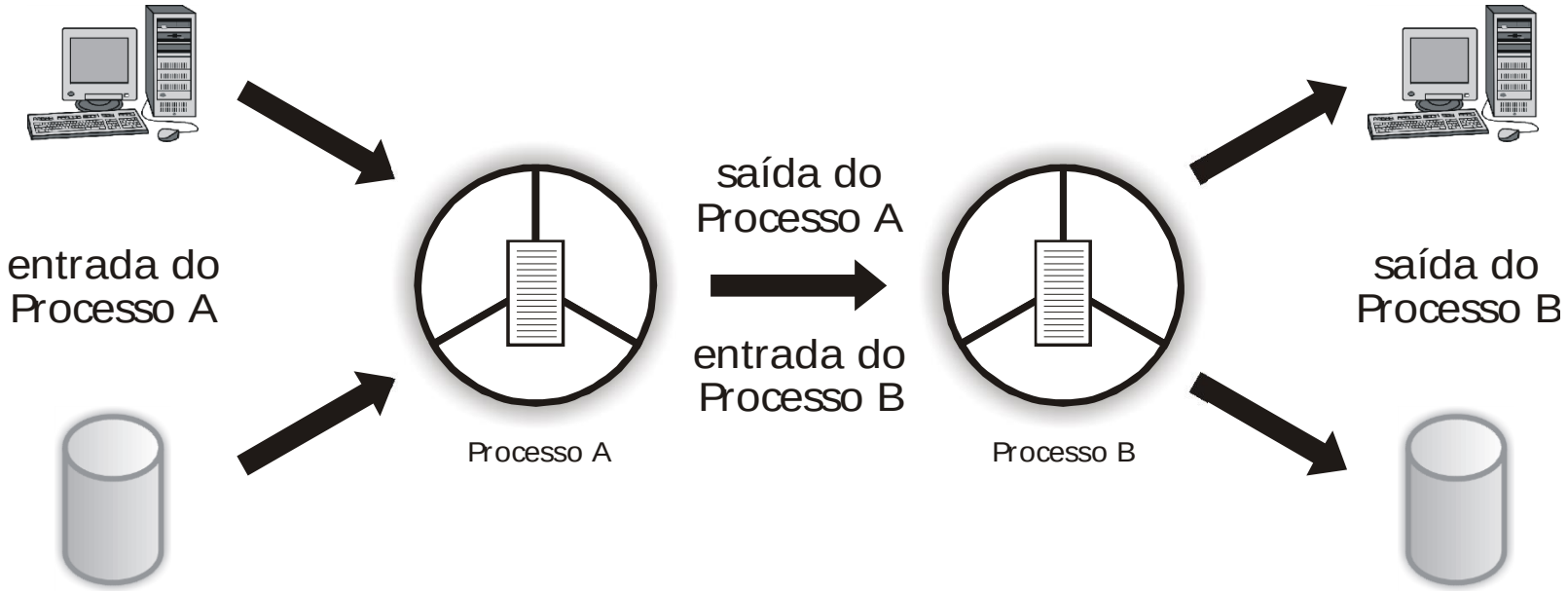
Processo Multithread



Processos Foreground e Background

- Todo processo possui pelos menos um canal de entrada e um canal de saída a ele associado
- Canais de entrada (*input*) e saída (*output*) podem estar associados a terminais, arquivos ou mesmo a outros processos
  - Canal de saída de um processo pode estar associado ao canal de entrada de outro processo através de um *pipe*
  - Processo em *foreground* possui canal de entrada em comunicação direta com usuário
  - Processos em *background* não possibilitam comunicação direta com usuário

Pipe


entrada do
Processo A
Processo A
saída do
Processo A
entrada do
Processo B
Processo B
saída do
Processo B

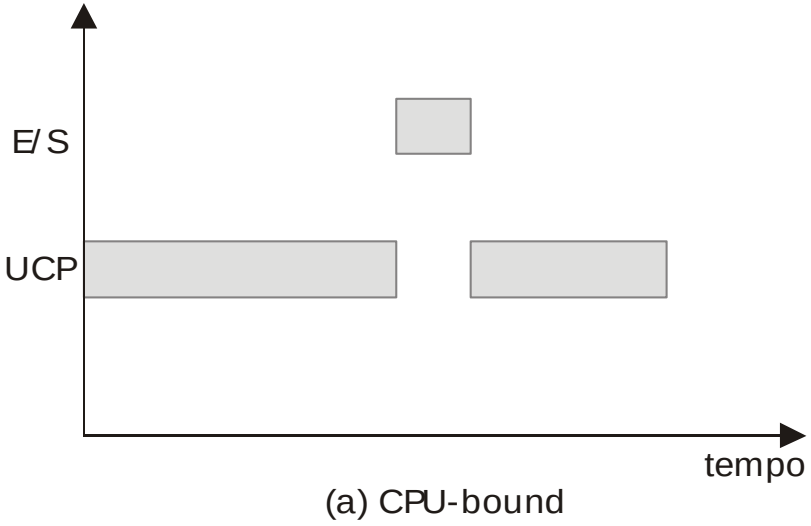
Processos do SO

- Arquitetura *microkernel* faz uso intensivo de processos que disponibilizam serviços para outros processos (aplicações e processos do SO)
- Exemplos:
  - Auditoria e segurança
  - Serviços de rede
  - Contabilização de uso de recursos
  - Gerência de impressão
  - Comunicação de eventos
  - Interface de comandos de linha (*shell*)

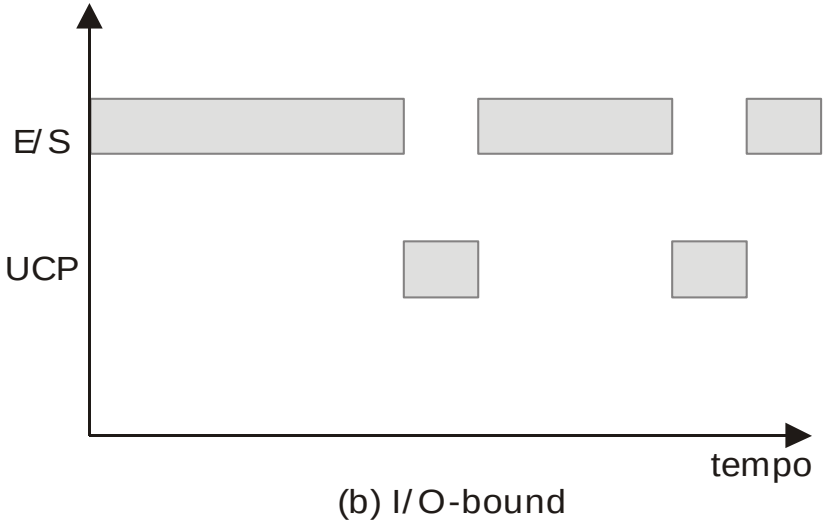
Processos CPU-bound x I/O-bound

- Processos **CPU-bound** ficam maior parte do tempo no estado de execução e pronto. Muito processamento

- Processos **I/O-bound** ficam maior parte do tempo no estado de espera. Muito IO



(a) CPU-bound



(b) I/O-bound

Sinais

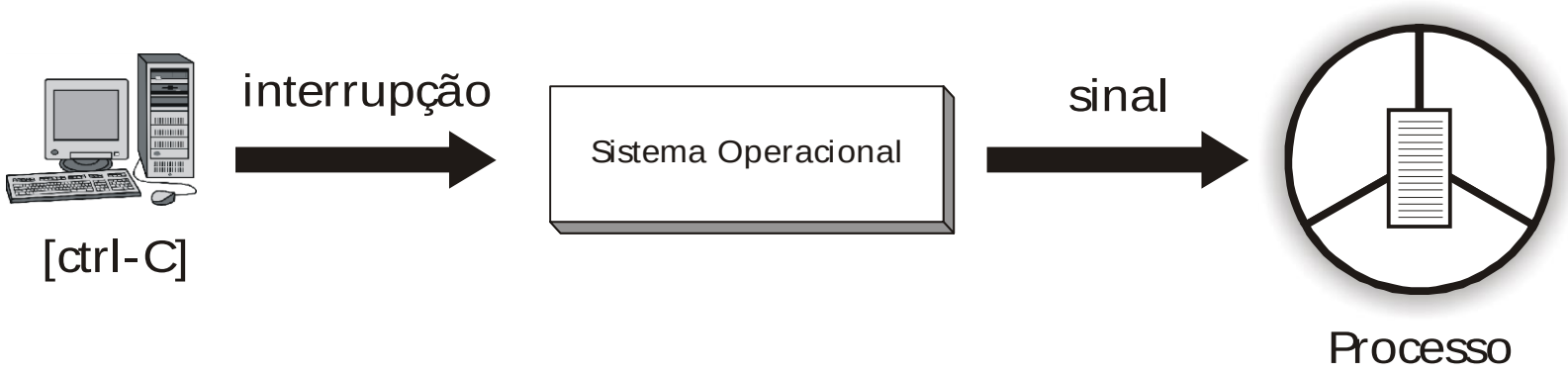
- Mecanismo que permite notificar processos da ocorrência de eventos, possibilitando a gerência, a comunicação e a sincronização entre processos

- Quando um processo identifica a chegada de um sinal, uma rotina de tratamento específica é executada

- Podem ser gerados por temporizadores, onde um sinal é enviado em decorrência de um *timeout*

- Geralmente são originados pelo *hardware* ou pelo SO, através de interrupções e exceções, ou ainda por outros processos

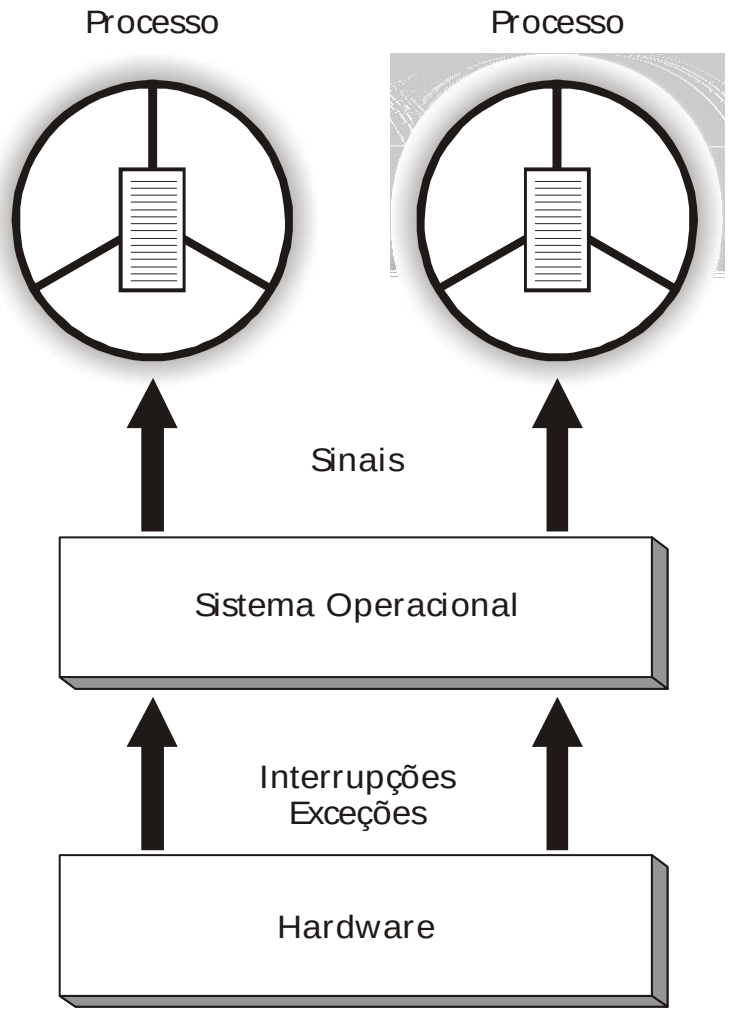
Sinais



Sinais

- A geração propriamente dita ocorre quando o SO notifica o processo através de bits de sinalização localizados no seu PCB
- Um sinal recebido fica pendente até que o processo seja escalonado para execução e possa tratá-lo
  - Ex: sinal de término (“kill -9 <PID>”) fica pendente até que o processo a ser eliminado seja escalonado para execução

Sinais, Interrupções e Exceções



Processos no Linux – Sinais

Entre os sinais existentes, tem-se os seguintes exemplos:

- STOP - esse sinal tem a função de interromper a execução de um processo e só reativá-lo após o recebimento do sinal CONT;
- CONT - esse sinal tem a função de instruir a execução de um processo após este ter sido interrompido;
- SEGV - esse sinal informa erros de endereços de memória;
- TERM - esse sinal tem a função de terminar completamente o processo, ou seja, este deixa de existir após a finalização;
- ILL - esse sinal informa erros de instrução ilegal, por exemplo, quando ocorre divisão por zero;
- KILL - esse sinal tem a função de "matar" um processo e é usado em momentos de criticidade.

Processos no Linux – Sinais

O kill também é um comando que o usuário pode usar para enviar qualquer sinal, porém, se ele for usado de maneira isolada, ou seja, sem o parâmetro de um sinal, o kill por padrão executa o sinal TERM.

A sintaxe para a utilização do comando kill é a seguinte:

kill -SINAL PID

Como exemplo, vamos supor que você deseja interromper temporariamente a execução do processo de PID 4220. Para isso, pode-se usar o seguinte comando:

kill -STOP 4220

Para que o processo 4220 volte a ser executado, basta usar o comando:

kill -CONT 4220

Se o sinal precisa ser enviado a todos os processos, pode-se usar o número -1 no lugar do PID. Por exemplo:

kill -STOP -1

Processos no Linux - Prioridades

Embora determinar a prioridade de um processo não seja uma prática comum, afinal, o próprio Linux faz muito bem essa tarefa, isso pode ser necessário em alguma situação. Para isso, utiliza-se um comando que recebe o mesmo nome do conceito: nice.

Quanto mais alto for o valor nice, mais gentil é o processo. Geralmente, o intervalo de números usados no nice são os inteiros entre -19 e 19.

A sintaxe é:

nice -n prioridade processo

Por exemplo:

nice -n -5 ntpd

No exemplo, o ntpd recebe prioridade -5. Trata-se de uma prioridade alta, quanto menor o número menor sua prioridade.

Processos no Linux - Prioridades

Se um determinado processo está em execução, isso acontece com uma prioridade já definida. Para alterar um processo nessa condição, usa-se o comando **renice**, cuja sintaxe é:

renice prioridade opção processo/destino

As opções do renice são:

-u - a alteração ocorrerá nos processos do usuário informado;
-g - a alteração ocorrerá nos processos do grupo indicado;
-p - a alteração ocorrerá no processo cujo PID for informado.

Um exemplo:

renice +19 1000 -u usuário

Neste caso, o comando renice alterou a prioridade do processo 1000, assim como a prioridade dos processos do usuário que for informado.

Processos no Linux – Verificando Processos

Verificando processos com o top

O comando ps trabalha como se tirasse uma fotografia da situação dos processos naquele momento. O comando top, por sua vez, coleta as informações, mas as atualiza regularmente. Geralmente essa atualização ocorre a cada 10 segundos.
A sintaxe do comando top é a seguinte:

top –opção

Entre as opções, tem-se as que se seguem:

-d - atualiza o top após um determinado período de tempo (em segundos). Para isso, informe a quantidade de segundos após a letra d. Por exemplo: top -d 30;
-c - exibe a linha de comando ao invés do nome do processo;
-i - faz o top ignorar processos em estado zumbi;
-s - executa o top em modo seguro.

Processos no Linux – Verificando Processos

Esse comando mostra processos relacionados em formato de árvore. Sua sintaxe é:

pstree -opção PID

Entre as opções, tem-se:

-u - mostra o proprietário do processo;
-p - exibe o PID após o nome do processo;
-c - mostra a relação de processos ativos;
-G - usa determinados caracteres para exibir o resultado em um formato gráfico.

Gerenciando Serviços

Esse comando mostra processos relacionados em formato de árvore. Sua sintaxe é:

pstree -opção PID

Entre as opções, tem-se:

-u - mostra o proprietário do processo;
-p - exibe o PID após o nome do processo;
-c - mostra a relação de processos ativos;
-G - usa determinados caracteres para exibir o resultado em um formato gráfico.

Gerenciando Serviços

Na maioria das distribuições, podemos gerencias os serviços que vão iniciar junto com SO.

Comando: **ntsysv** ou **service** nomedoprograma start/stop/restart