Escalonamento de Processos

Ciclo 3 – AT1

Prof. Hermes Senger

Objetivos

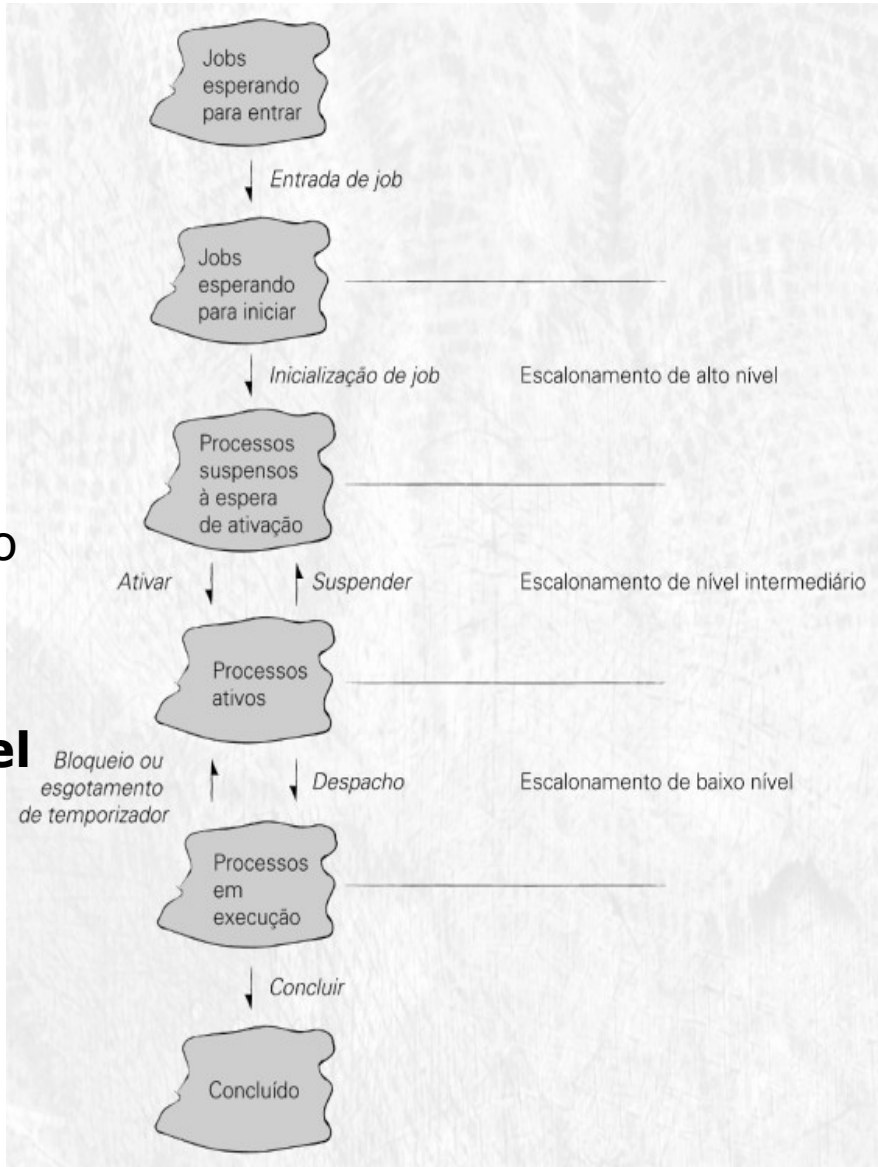
- No ciclo anterior aprendemos que existe uma fila de processos.
  - Mas afinal, quem decide em que **ordem** os processos são executados?
  - Todos os processos têm a mesma **prioridade**?
  - Por **quanto tempo** o processador permanece executando cada processo?
- Todas essas questões (e mais algumas) são tratadas pelos **algoritmos** ou **políticas de escalonamento** do processador.
- Neste ciclo estudaremos algoritmos de escalonamento relevantes, incluindo exemplos de algoritmos **preemptivos** e **não-preemptivos**, tratando questões de prioridades e aspectos da seleção de processos em ambientes com restrições de tempo real.
- Por fim veremos (resumidamente) como funciona o escalonamento de **threads** em Java.

Introdução

- O **escalonamento** de **processos** (ou **escalonamento do processador**):
  - Trata da decisão que qual processo será executado em um determinado instante
  - e por qual processador (caso exista mais de um).
    - Isto vale também para processadores com vários núcleos (*dual e quad core, por exemplo*).
- O **escalonador** é um componente típico dos SOs.
- Ele implementa uma **Política de escalonamento**.
- Implementação do escalonador pode mudar significativamente de um sistema operacional para outro.

Níveis de escalonamento

- *Mainframes* geralmente têm 3 níveis de escalonamento
- **Escalonamento de alto nível** trata da **admissão** de serviços ou *jobs* no sistema:
  - Quantos? Quais?
  - Visa a evitar a sobrecarga.
- Depois que jobs são admitidos (carregados), o **escalonamento de nível intermediário** decide quais poderão iniciar sua execução
  - Também pode adiar ou pausar processos temporariamente para aliviar carga a carga do sistema.
- O **escalonamento de baixo nível** (ou escalonador de curto prazo) atribui prioridades, escolhe a ordem, e é ele quem designa processadores a processos.



Preempção e Prioridades

- **No escalonamento preemptivo os processos que estão em execução podem "perder" o processador (o escalonador pode transferir o uso do processador para outro processo).**
  - **Importante em ambientes interativos: pode melhorar o tempo de resposta.**
  - **O processo que sofre preempção será retomado mais tarde.**
- **No escalonamento não preemptivo os processos são executados até o fim ou até que passem o controle ao escalonador, para dar chance de outros processos executarem.**
- **Alguns escalonadores estabelecem prioridades aos processos. Existem dois tipos de prioridades:**
  - **A prioridade estática de um processo é um valor permanente, que não se altera durante sua execução. Geralmente é atribuída pelo próprio usuário, que pode diferenciá-la de acordo com a importância ou a urgência de um processo, ou job.**
  - **Já a prioridade dinâmica pode aumentar ou diminuir, procurando adequar o peso de um processo para melhorar o tempo de resposta e a interatividade do sistema.**

Objetivos do escalonamento

- Escalonadores podem ter **objetivos muito diferentes,** dependendo do sistema. Ex.:
  - Maximizar a **vazão** do sistema como um todo. Ex: num. de jobs concluídos por hora, ...
  - Maximizar o **número de processos interativos** cujo tempo de resposta seja aceitável.
  - Maximizar a **utilização** de recursos
    - Deixa contente quem pagou pelo sistema, que quer que ele seja bem ocupado.
  - Obedecer as **prioridades** dos processos na hora de executá-los.
  - etc

- Em geral, a maioria dos escalonadores prima por **justiça** (i.e., tratar igualmente os processos semelhantes), **previsibilidade** e **escalabilidade**, que é a capacidade de suportar maiores cargas à medida que os recursos aumentam.

Critérios de escalonamento

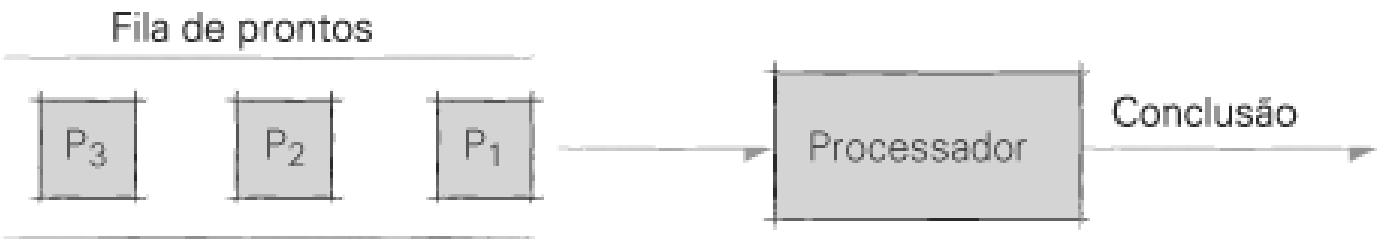
- Afinal, quais critérios são levados em conta pelo algoritmo de escalonamento?
  - Um possível critério pode ser o perfil dos processos

- Exemplos de perfis:
  - **Processos orientados a processador (*CPU-bound*)**
    - Usam todo o tempo disponível do processador.
  - **Processos orientados a E/S (*IO-bound*)**
    - Usam o processador por pouco tempo – geralmente ganham a CPU e emitem solicitação de E/S logo em seguida (devolvendo o processador).
  - **Processo em lote**
    - São aqueles que o sistema executará sem interagir com o usuário.
  - **Processos interativos**
    - Exigem continuamente informações do usuário.

- Qual seria uma forma "justa" para distribuir o tempo da CPU entre 2 processos, sendo um CPU-*bound* e outro *IO-bound*?
  - Daria o mesmo *time-slice para ambos?*

Escalonamento FIFO

- **Escalonamento FIFO (*first-in first-out*)**
  - É o esquema mais simples e mais antigo.
  - Também chamado de FCFS (*first-come-first-served* – primeiro a chegar, primeiro a ser servido)
  - Processos são executados até o fim, na mesma ordem de chegada.
  - Não preemptivo
  - Raramente é usado como algoritmo de escalonamento principal.



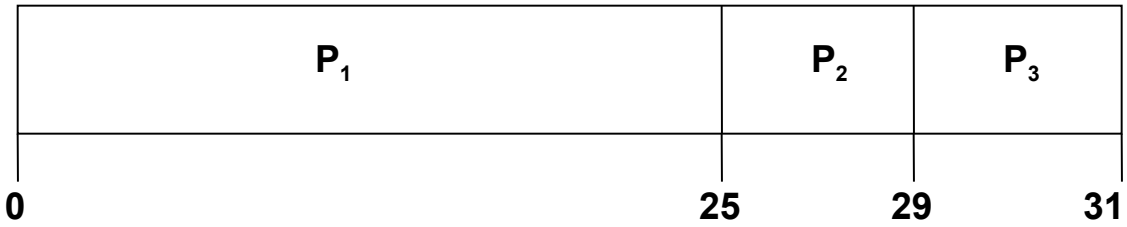
Escalonamento FIFO: Exemplo

Processo	Duração
P_1	25
P_2	4
P_3	2

Suponha que os processos cheguem na ordem: P1 , P2 , P3.

O diagrama de Gantt(1) para o escalonamento será:

- **Tempo de espera para P1 = 0; P2 = 25; P3 = 29**
- **Tempo de espera médio: (0 + 25 + 29)/3 = 18**
- **Tempo de *turnaround* para P1 = 25; P2 = 29; P3 = 31**

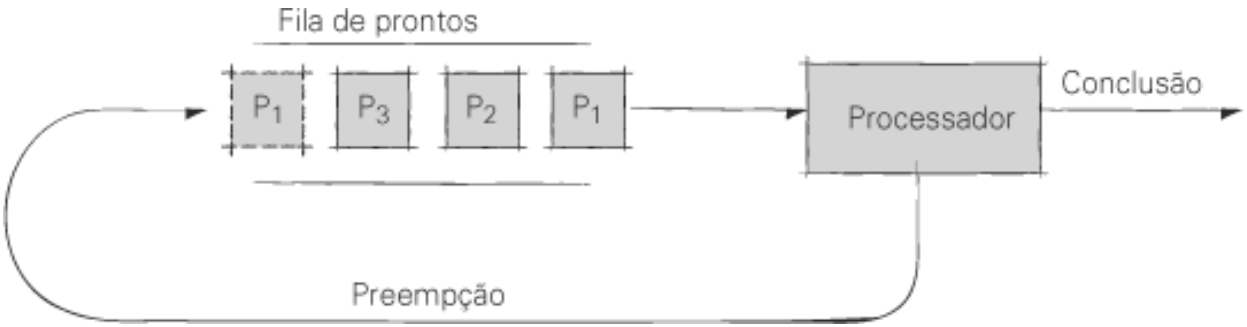


- **Recalcule os tempos supondo que a ordem de chegada fosse P3, P2 e P1.**

(1) *Gant chart* é um diagrama de tempo, que mostra atividades com seus instantes iniciais e

Escalonamento circular (RR)

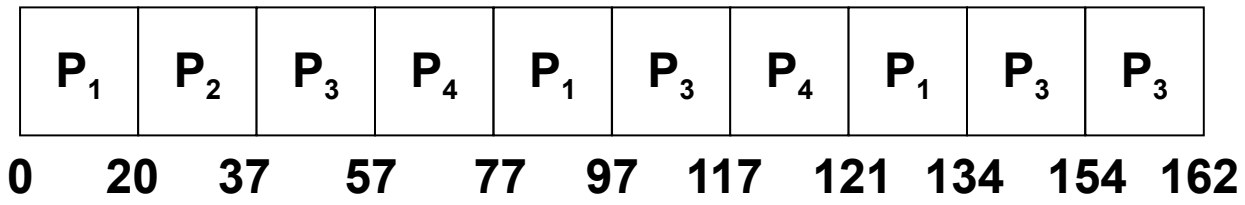
- **Escalonamento circular (Round-Robin — RR)**
  - Inspirado no FIFO
  - Preemptível: os processos são executados apenas durante um ***quantum*** de tempo
  - Quando o *quantum* expira, o processo volta para o fim da fila de processos prontos
  - Os processos devem permanecer carregados na memória, pois em breve receberão um novo *quantum*.
  - É usado comumente como parte de algoritmos mais complexos.



Escalonamento circular (RR): Exemplo

Processo	Duração
P1	53
P2	17
P3	68
P4	24

- Quantum = 20 ms (milissegundos)
- O diagrama de Gantt é:



Escalonamento circular (RR)

- **Qual seria o tamanho do quantum ideal ?**
  - Isso depende ...
  - Imagine um quantum **grande**.
    - Como isso influenciaria a responsividade dos processos interativos?
      - Suponha um quantum maior que a duração do processo – o RR passaria a se comportar como FIFO!
  - Que tal um quantum **bem pequeno**?
    - Nesse caso, o sistema gastaria mais tempo chaveando o contexto do que executando os processos.

- O ideal seria um tamanho "médio"
  - Longo o bastante para que os processos interativos façam solicitações de E/S (boa responsividade).
  - Assim, os processos em lote continuam consumindo maior parte do tempo processado.

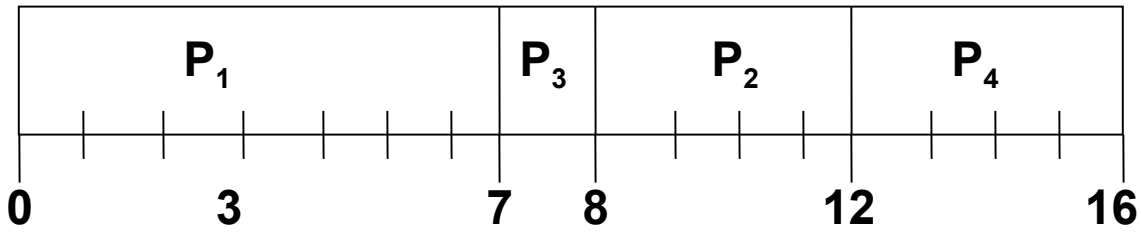
Escalonamento por processo-mais-curto-primeiro (SPF)

- Em uma fila, processos longos podem gerar aumento do tempo médio de espera
- **Para isso foi criado o escalonamento SPF (***Shortest Process First***)**
  - Algoritmo não-preemptivo
  - **Escolhe primeiro o processo com menor tempo de finalização.**
  - Baseia-se na estimativa do **tempo de execução** até a conclusão.
    - Estimativa pode ser imprecisa ou estar completamente errada.
  - Também chamado de *shortest-job-first* (SJF)
  - Efeitos no desempenho:
    - Proporciona tempo de espera médio **menor** que no FIFO.
    - Mas a **variação** nos tempos de espera tende a ser **maior**.
    - Como é não-preemptivo, a velocidade do **tempo de resposta** é **pior** para as solicitações interativas que estão chegando.
  - Não é adequado para os modernos sistemas interativos.
- Mesmo princípio do "caixa rápido" de um supermercado

SPF: Exemplo

Processo	chegada	Burst
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SPF (não-preemptivo)



Tempo de espera médio = (0 + 6 + 3 + 7)/4 = 4

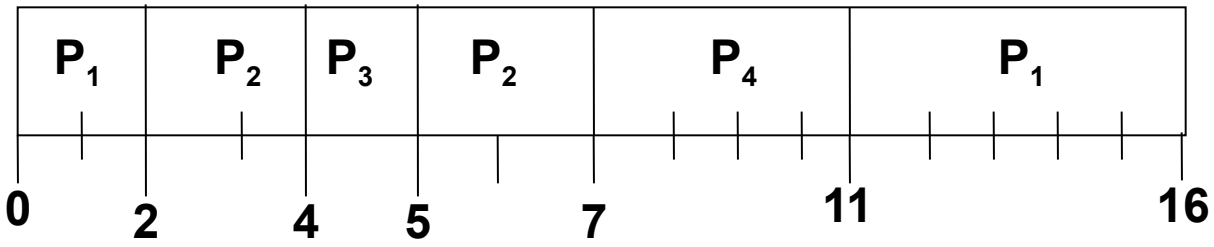
Escalonamento por menor-tempo-de-execução-restante (SRT)

- **Escalonamento *shortest remaining time* (SRT)**
  - Seleciona o processo com **menor** tempo de execução **restante**
- É uma versão melhorada (preemptiva) do SPF
  - Permite que os processos mais curtos que chegam causem a preempção de um processo em execução.
  - Desempenho:
    - Tempo **médio** de espera **menor** que do SPF
    - Mas a **variação** dos tempos de resposta é **grande**: o tempo de espera dos processos longos é ainda maior que no SPF.
  - Nem sempre é ideal.
    - A chegada de um processo curto pode provocar preempção em um processo que está perto de concluir sua execução.
    - A sobrecarga de chaveamento de contexto pode tornar-se significativa.

SRT: Um SPF preemptivo

Processo	chegada	Duração
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SRT (preemptivo) – Diagrama de Gantt

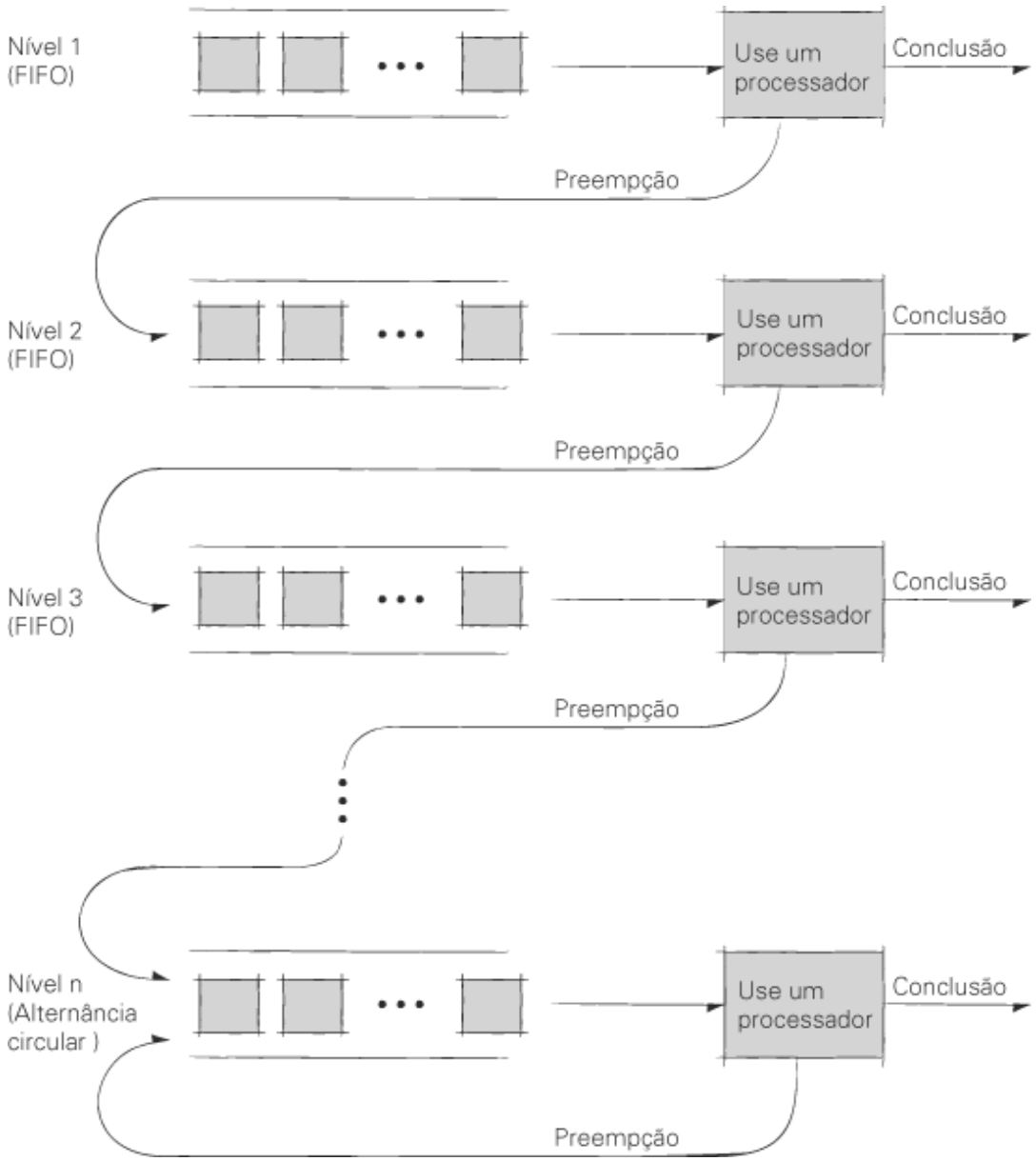


Tempo de espera médio = (9 + 1 + 0 +2)/4 = 3

Filas multiníveis com retorno

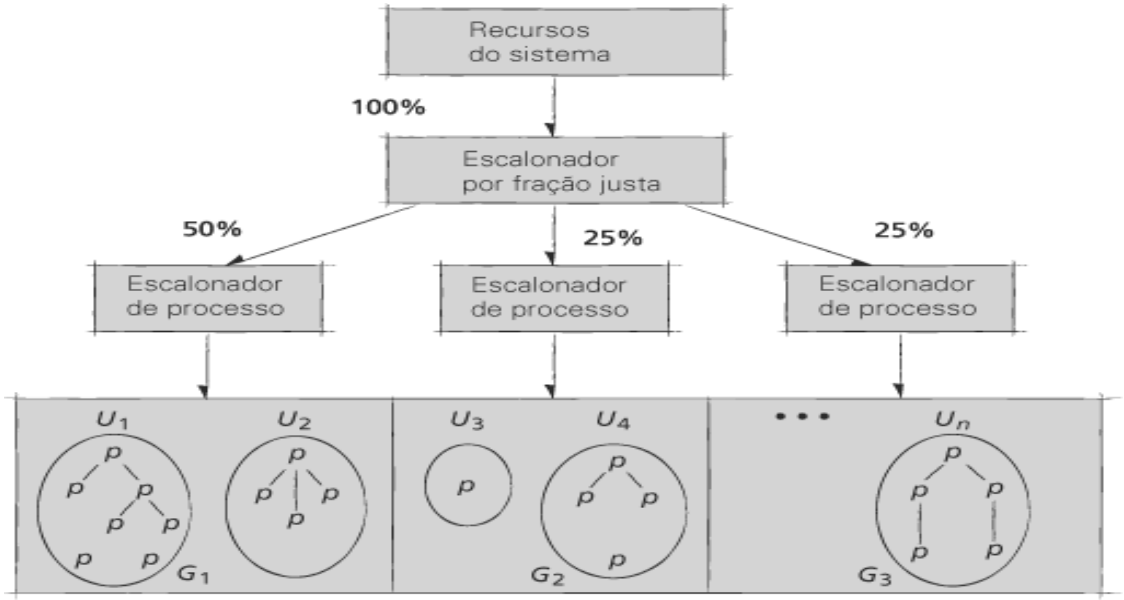
- **Processos diferentes têm necessidades diferentes.**
  - Os processos **interativos** orientados a E/S geralmente devem ser executados **antes** dos processos em lote orientados a processador.
  - Os **padrões** de comportamento não são evidentes e também podem se alterar.
- **Filas multiníveis com retorno**
  - Os processos que chegam entram na fila de nível superior
    - Filas **superiores** possuem prioridade de execução **mais alta**
    - Porém, filas **inferiores** possuem um ***quantum*** de tempo **maior**
  - Processos longos são penalizados e transferidos para os níveis inferiores.
  - Mobilidade entre as filas:
    - Processos ("ruins") que usam todo o seu quantum são **rebaixados**
    - Processos ("bonzinhos") que requisitam E/S antes de expirar o seu quantum são promovidos
  - Dentro de uma mesma fila, os processos são atendidos por meio do escalonamento RR
    - Os processos que entram na fila de nível superior provocam preempção nos processos em execução.
- Trata-se, portanto, de um esquema adaptativo!!!

Filas multiníveis de retorno



Escalonamento por fração justa

- **Escalonamento FSS (*fair share scheduling*)**
- Permite estabelecer como será dividido o tempo do processador entre os usuários.
  - Imagine uma situação em que alguns grupos usuários são mais importantes que outros.
    - O FSS garante que os grupos menos importantes não monopolizem os recursos
  - Se um certo grupo não estiver utilizando seus recursos:
    - Os recursos que ele não está usando são distribuídos entre os outros grupos
    - E além disso sua prioridade sobe
  - O sistema UNIX implementa um escalonamento-padrão de processos e um escalonador do tipo FSS



Escalonamento de tempo real

- **Escalonamento de tempo real**
  - Até certo ponto é semelhante ao escalonamento por prazo.
  - Os processos também têm **restrição** de tempo.
  - Mas engloba também tarefas que são executadas **periodicamente**. Ex:
    - Monitorar temperatura e pressão de um processo químico
- **Há duas categorias:**
  - Escalonamento de tempo real **não crítico**
    - Não garante que as restrições de tempo serão cumpridas.
    - Utilizado por exemplo, para aplicações multimídia.
  - Escalonamento de tempo real **crítico**
    - As restrições de tempo **devem** ser cumpridas – sempre!
    - O não cumprimento do prazo pode ter resultados catastróficos.
    - Por exemplo, o ILS (“piloto automático” de avião).

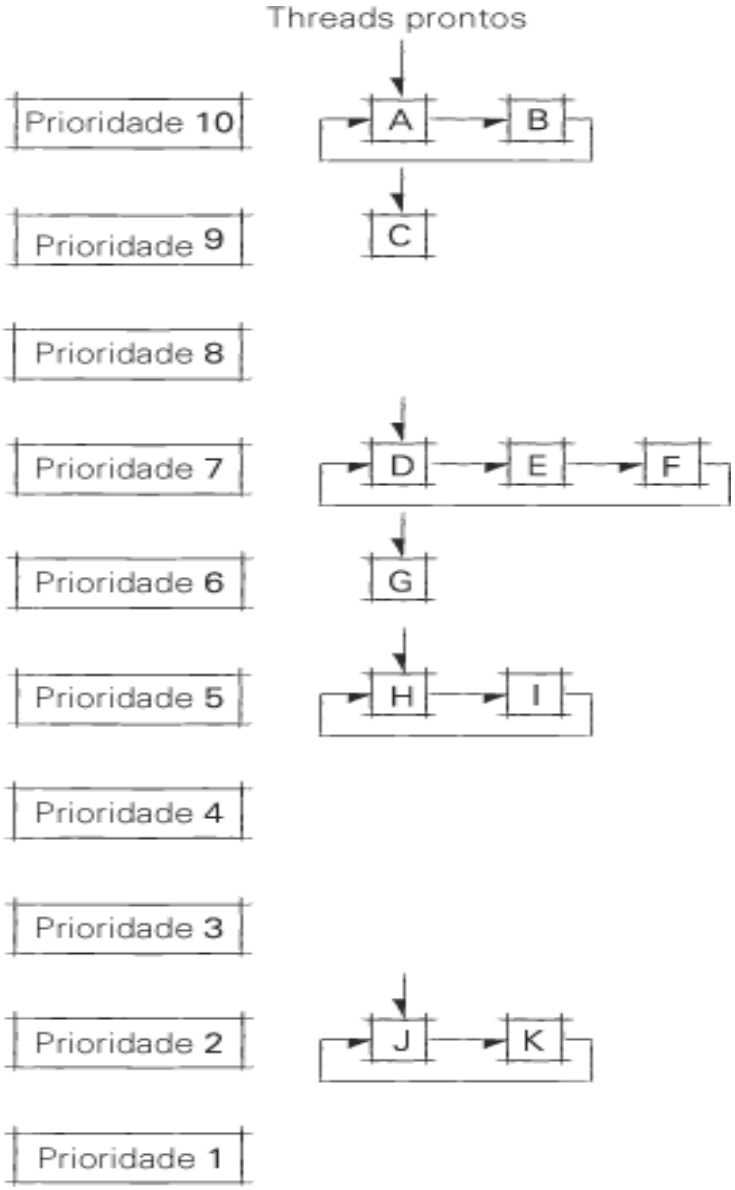
Escalonamento de threads

- **Os sistemas operacionais oferecem uma variedade de suportes ao escalonamento de threads.**
  - Threads de núcleo
    - São implementados no **núcleo** do SO
    - São chamados de threads nativos
    - O próprio escalonador subdivide o tempo do processador para os threads do processo.
  - Threads de usuário
    - São implementados no nível de usuário (fora do *kernel* do SO)
    - A própria **aplicação** (linguagem de programação ou biblioteca) implementa o seu independentemente.
    - O sistema operacional não "enxerga" esse tipo de thread – ou seja, ele "entrega"o processador ao processo, que possui um escalonador para gerenciar os seus próprios threads

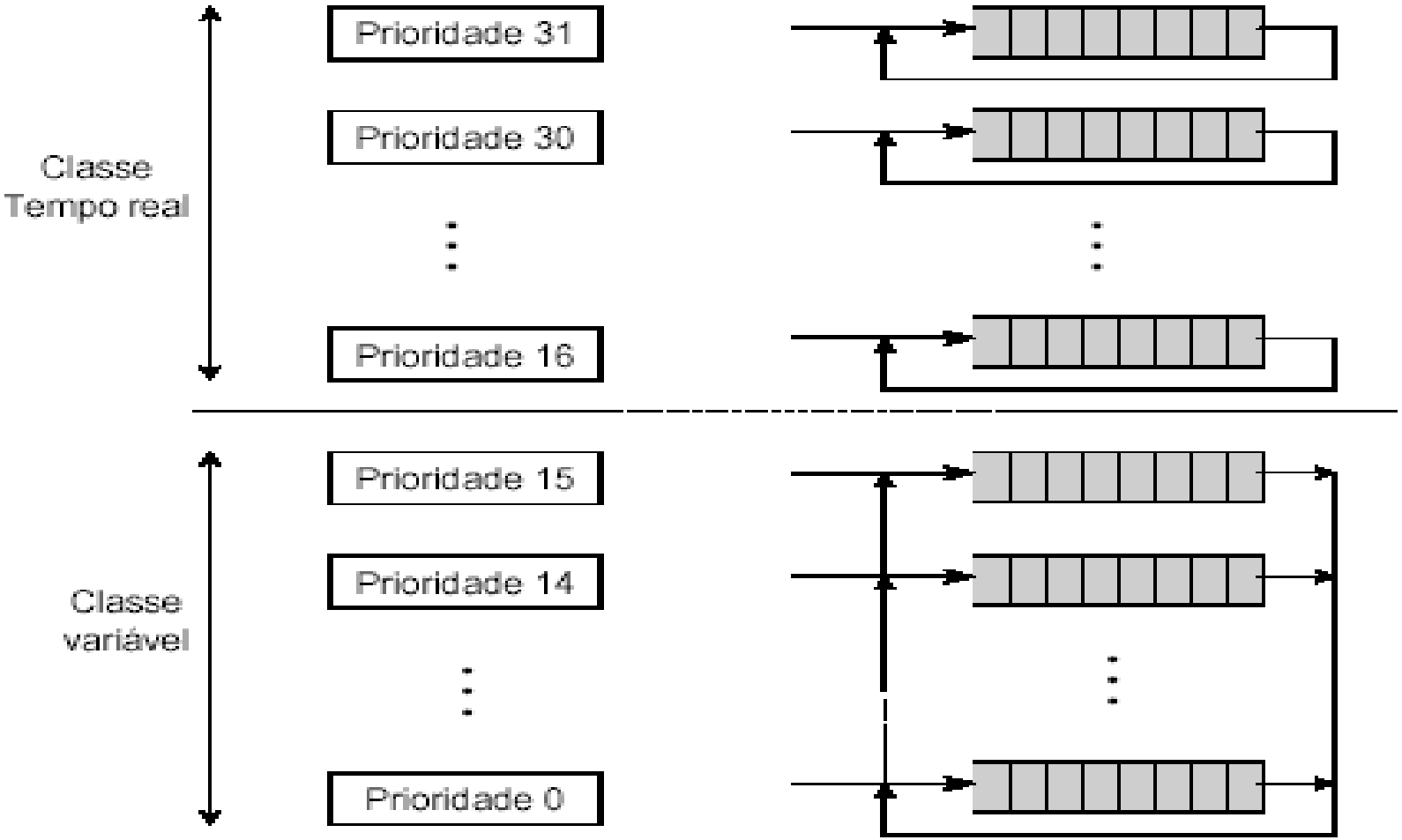
Escalonamento de threads Java

- **A JVM (Java Virtual Machine) possui o seu próprio escalonador de threads Java**
  - Se o SO tiver threads de núcleo, então a JVM os utilizará
    - **Mas a JVM não precisa disso!**
  - Os threads podem usar a chamada `yield()` para ceder o processador a outros de igual prioridade.
    - Isso é necessário apenas em sistemas que não dispõem de intervalo de tempo.
    - Os threads que estão aguardando execução são chamados de estado de espera, adormecidos ou bloqueados.

Escalonamento de threads Java



Escalonamento no Windows XP



Escalonamento no UNIX

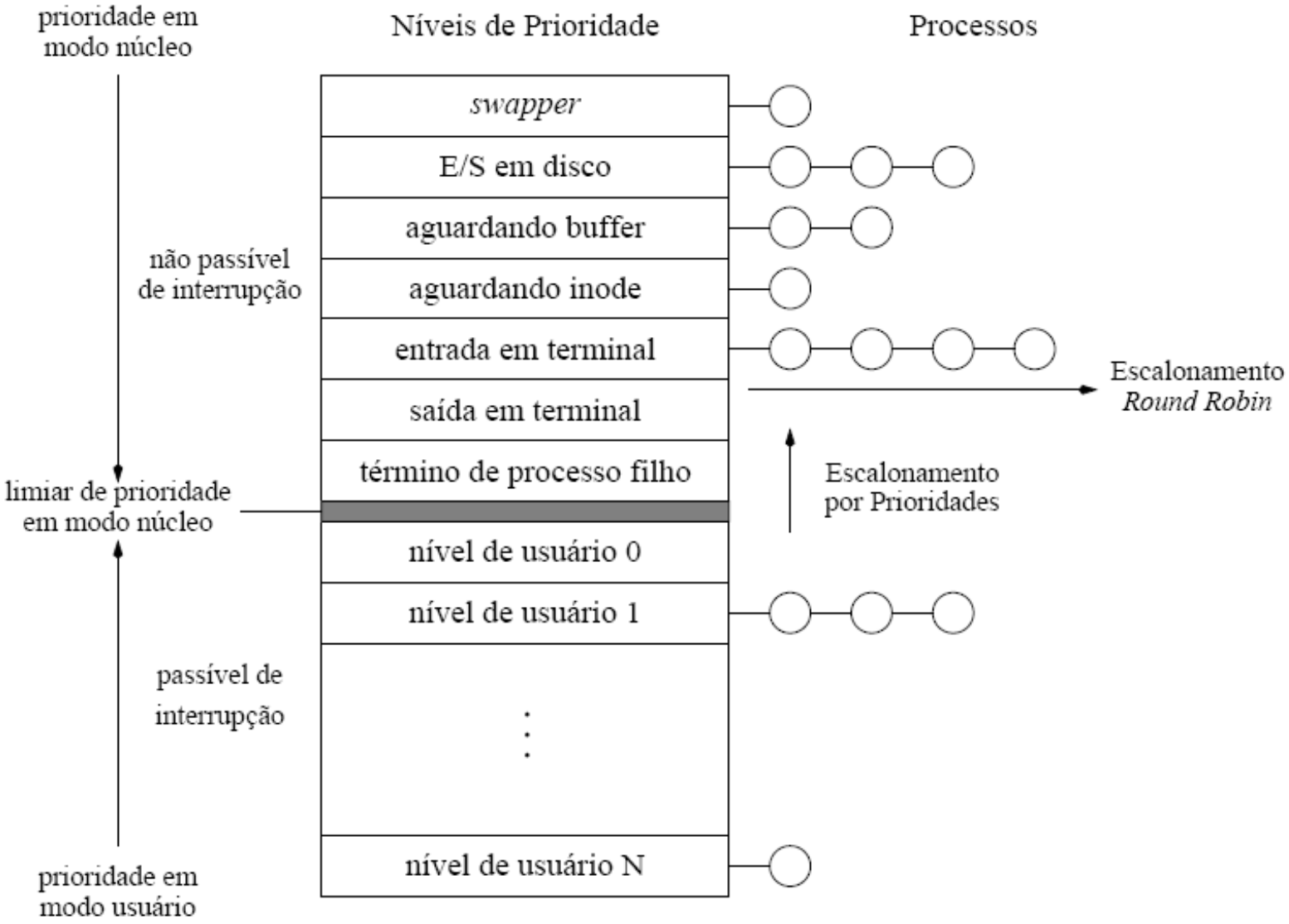


Figura de (Cardozo, Magalhães e Faina, 2002)