Escalonamento de tarefas

Prof. Paulo Roberto O. Valim
Universidade do Vale do Itajaí

Escalonamento de tarefas

- Um dos componentes mais importantes da gerência de tarefas é o escalonador (*task scheduler*), que decide a ordem de execução das tarefas prontas.

- O algoritmo utilizado no escalonador define o comportamento do sistema operacional, permitindo obter sistemas que tratem de forma mais eficiente e rápida as tarefas a executar, que podem ter características diversas: aplicações interativas, processamento de grandes volumes de dados, programas de cálculo numérico, etc.

Escalonamento de tarefas

- Antes de se definir o algoritmo usado por um escalonador, é necessário ter em mente a natureza das tarefas que o sistema irá executar. Existem vários critérios que definem o comportamento de uma tarefa; uma primeira classificação possível diz respeito ao seu comportamento temporal.

Tarefas: classificação pelo comportamento temporal

- **Tarefas de tempo real** : exigem previsibilidade em seus tempos de resposta aos eventos externos, pois geralmente estão associadas ao controle de sistemas críticos, como processos industriais, tratamento de fluxos multimídia, etc. O escalonamento de tarefas de tempo real é um problema complexo.

- **Tarefas interativas** : são tarefas que recebem eventos externos (do usuário ou através da rede) e devem respondê-los rapidamente, embora sem os requisitos de previsibilidade das tarefas de tempo real. Esta classe de tarefas inclui a maior parte das aplicações dos sistemas *desktop* (editores de texto, navegadores Internet, jogos) e dos servidores de rede (e-mail, web, bancos de dados).

- **Tarefas em lote (*batch*)** : são tarefas sem requisitos temporais explícitos, que normalmente executam sem intervenção do usuário, como procedimentos de *backup*, varreduras de anti-vírus, cálculos numéricos longos, renderização de animações, etc.

Tarefas: classificação pelo uso do processador

- Além dessa classificação, as tarefas também podem ser classificadas de acordo com seu comportamento no uso do processador:
  - Tarefas orientadas a processamento (CPU-bound tasks)
  - Tarefas orientadas a entrada/saída (IO-bound tasks)

Objetivos e métricas

- Tempo de execução ou de vida (*turnaround time*, *tt)*
- Tempo de espera (*waiting time*, *tw*)
- Tempo de resposta (*response time*, *tr*)
- Justiça
- Eficiência

Escalonamento preemptivo e cooperativo

- Sistemas preemptivos : nestes sistemas uma tarefa pode perder o processador caso termine seu ***quantum* de tempo (t_q)**, execute uma chamada de sistema ou caso ocorra uma interrupção que acorde uma tarefa mais prioritária (que estava suspensa aguardando um evento). A cada interrupção, exceção ou chamada de sistema, o escalonador pode reavaliar todas as tarefas da fila de prontas e decidir se mantém ou substitui a tarefa atualmente em execução.

- Sistemas cooperativos (não-preemptivos) : a tarefa em execução permanece no processador tanto quanto possível, só abandonando o mesmo caso termine de executar, solicite uma operação de entrada/saída ou libere explicitamente o processador, voltando à fila de tarefas prontas (isso normalmente é feito através de uma chamada de sistema sched_yield() ou similar). Esses sistemas são chamados de *cooperativos* por exigir a cooperação das tarefas entre si na gestão do processador, para que todas possam executar.

Escalonamento FCFS (First-Come, First Served)

- A forma de escalonamento mais elementar consiste em simplesmente atender as tarefas em sequência, à medida em que elas se tornam prontas (ou seja, conforme sua ordem de chegada na fila de tarefas prontas). Esse algoritmo é conhecido como FCFS – *First Come - First Served* – e tem como principal vantagem sua simplicidade.

- Como um exemplo vamos considerar as tarefas na tabela abaixo na fila de tarefas prontas:

tarefa	t_1	t_2	t_3	t_4
ingresso	0	0	1	3
duração	5	2	4	3

Escalonamento FCFS (First-Come, First Served)

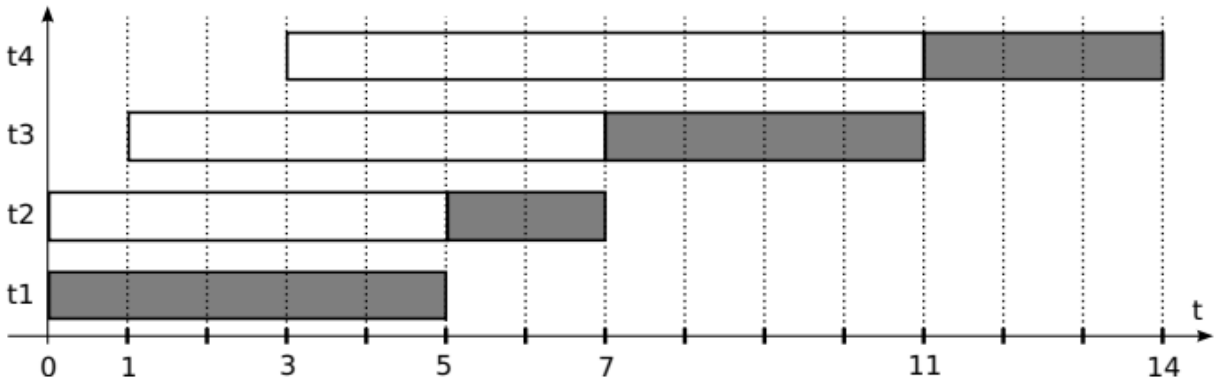


Figura 2.13: Escalonamento FCFS.

Calculando o tempo médio de execução (T_t, a média de $t_t(t_i)$) e o tempo médio de espera (T_w, a média de $t_w(t_i)$) para o algoritmo FCFS, temos:

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(5-0) + (7-0) + (11-1) + (14-3)}{4}$$
$$= \frac{5+7+10+11}{4} = \frac{33}{4} = 8.25s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(0-0) + (5-0) + (7-1) + (11-3)}{4}$$
$$= \frac{0+5+6+8}{4} = \frac{19}{4} = 4.75s$$

Escalonamento SJF (Shortest Job First)

- O algoritmo de escalonamento que proporciona os menores tempos médios de execução e de espera é conhecido como *menor tarefa primeiro*, ou SJF (*Shortest Job First*). Como o nome indica, ele consiste em atribuir o processador à menor (mais curta) tarefa da fila de tarefas prontas. Pode ser provado matematicamente que esta estratégia sempre proporciona os menores tempos médios de espera.

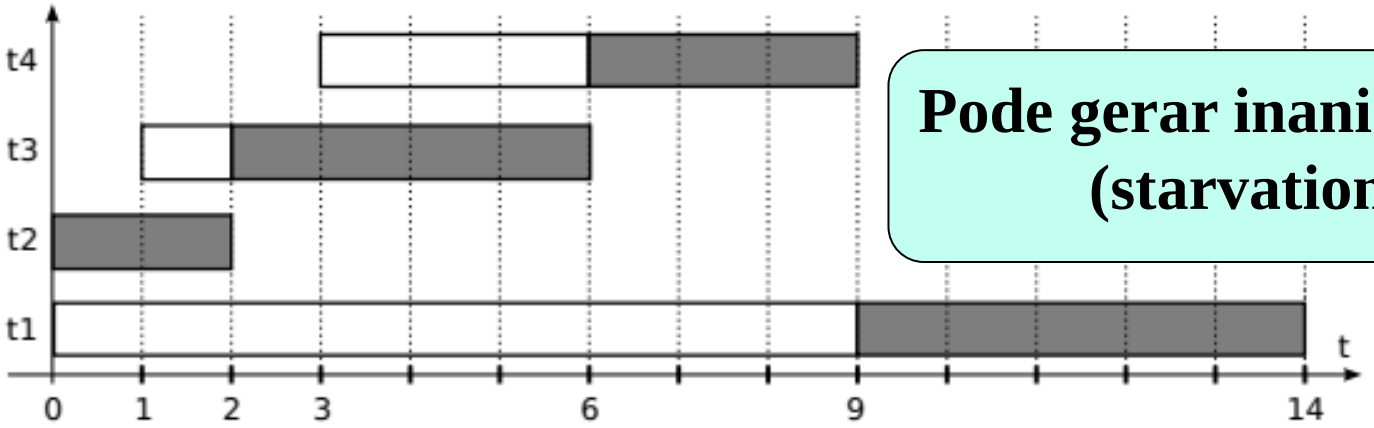
- Tomando o mesmo exemplo:

tarefa	t_1	t_2	t_3	t_4
ingresso	0	0	1	3
duração	5	2	4	3

Escalonamento SJF (Shortest Job First)*

* Cooperativo

- Exemplo de SJF aplicado a tabela de tarefas anterior



Pode gerar inanição (starvation)

Figura 2.16: Escalonamento SJF.

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(14-0) + (2-0) + (6-1) + (9-3)}{4}$$

$$= \frac{14+2+5+6}{4} = \frac{27}{4} = 6.75s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(9-0) + (0-0) + (2-1) + (6-3)}{4}$$

$$= \frac{9+0+1+3}{4} = \frac{13}{4} = 3.25s$$

Escalonamento RR (Round-Robin)*

* Com preempção

- Considerando as tarefas definidas na tabela anterior e quantum t_q = 2s teríamos:

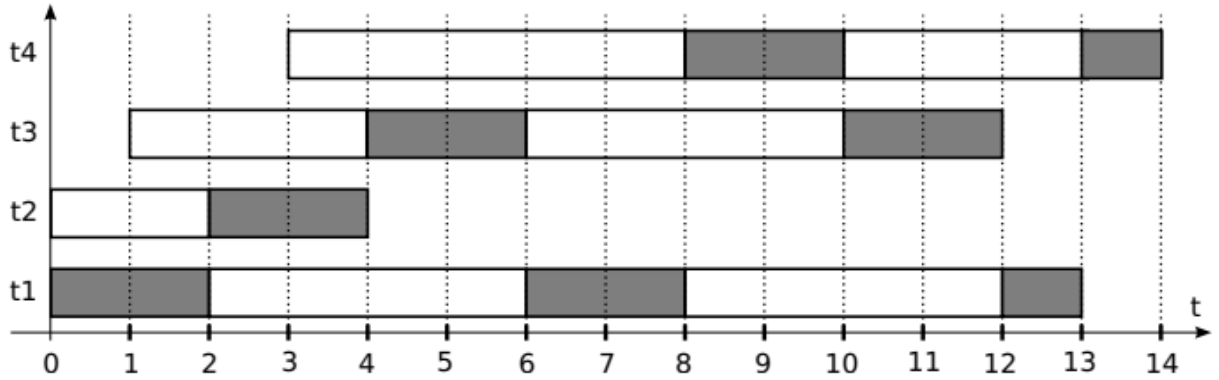


Figura 2.14: Escalonamento *Round-Robin*.

- E os cálculos de tempos seriam:

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(13-0) + (4-0) + (12-1) + (14-3)}{4}$$

$$= \frac{13 + 4 + 11 + 11}{4} = \frac{39}{4} = 9.75s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{8 + 2 + 7 + 8}{4} = \frac{25}{4} = 6.25s$$

Impacto das trocas de contextos

- A quantidade e a duração das trocas de contextos entre tarefas pode ter impacto negativo na eficiência da sistema operacional.
- É possível definir uma medida de eficiência E do uso do processador, em função das durações médias do quantum de tempo t_q e da troca de contexto t_{tc}.

$$\mathcal{E} = \frac{t_q}{t_q + t_{tc}}$$

Impacto das trocas de contextos

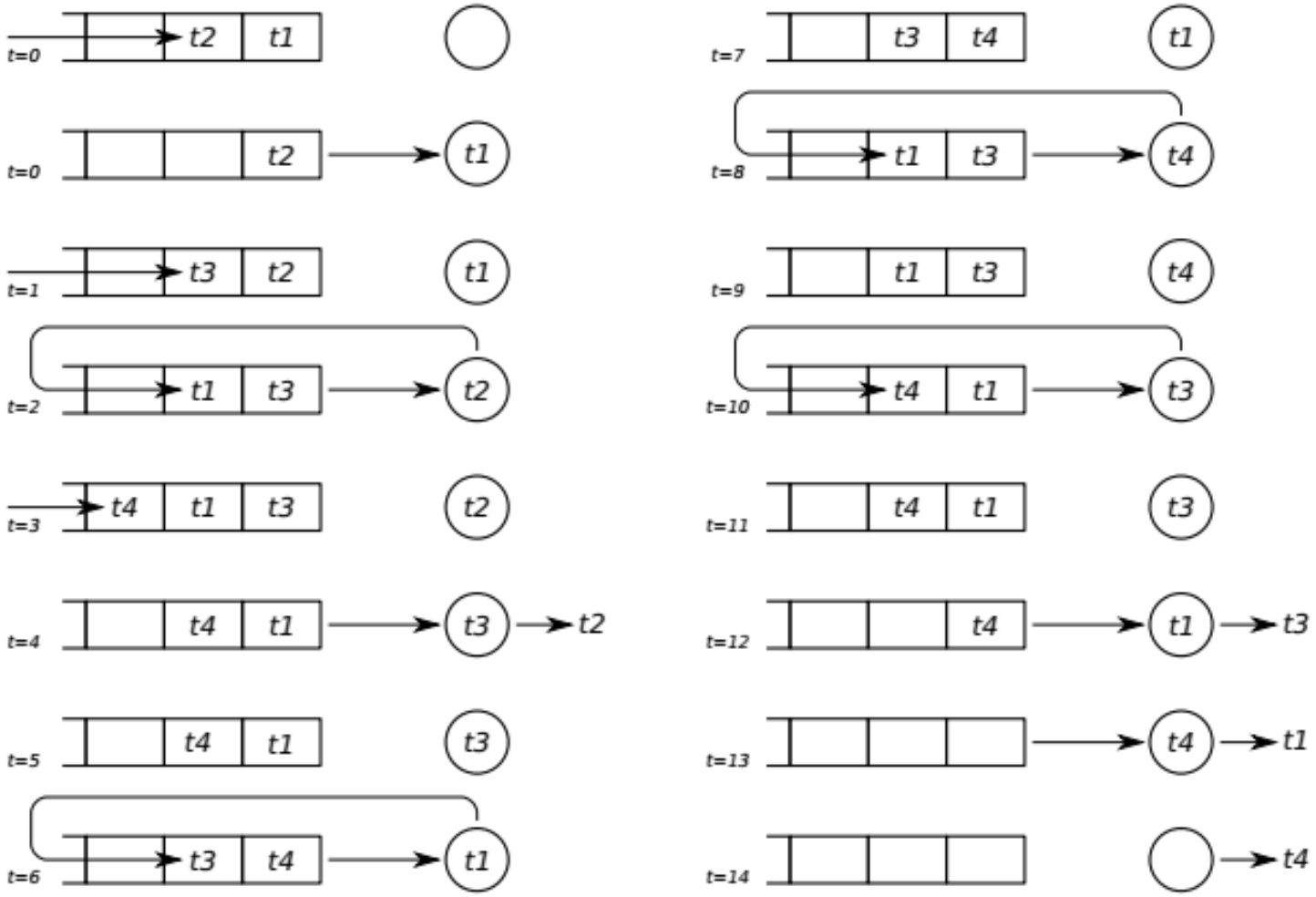


Figura 2.15: Evolução da fila de tarefas prontas no escalonamento *Round-Robin*.

Escalonamento por prioridades

- No escalonamento por prioridades, a cada tarefa é associada uma prioridade, geralmente na forma de um número inteiro. Os valores de prioridade são então usados para escolher a próxima tarefa a receber o processador, a cada troca de contexto.

- Exemplo (obs.: prioridade positiva= valores maiores indicam maior prioridade):

tarefa	t_1	t_2	t_3	t_4
ingresso	0	0	1	3
duração	5	2	4	3
prioridade	2	3	1	4

Escalonamento por prioridades

*** Cooperativo**

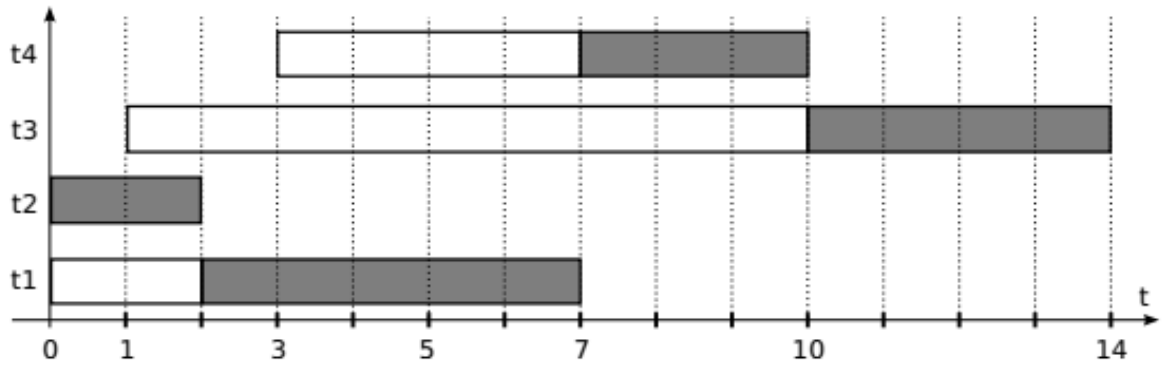


Figura 2.17: Escalonamento por prioridades (cooperativo).

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(7-0) + (2-0) + (14-1) + (10-3)}{4}$$
$$= \frac{7+2+13+7}{4} = \frac{29}{4} = 7.25s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(2-0) + (0-0) + (10-1) + (7-3)}{4}$$
$$= \frac{2+0+9+4}{4} = \frac{15}{4} = 3.75s$$

Escalonamento por prioridades

*** preemptivo**

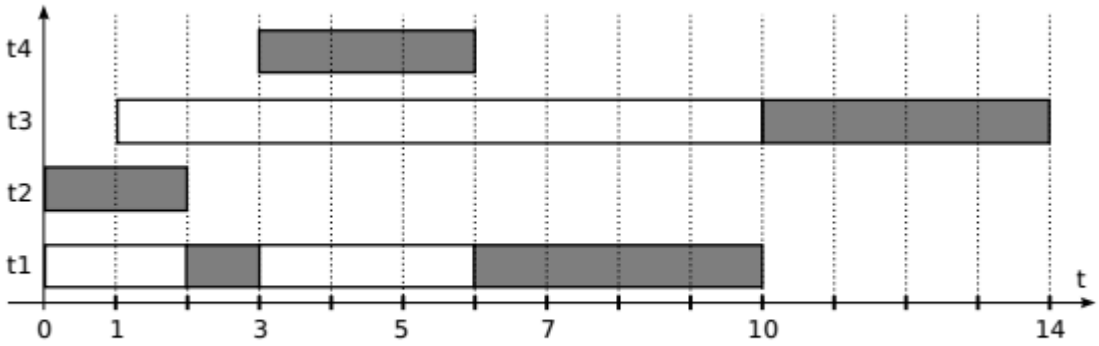


Figura 2.18: Escalonamento por prioridades (preemptivo).

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(10-0) + (2-0) + (14-1) + (6-3)}{4}$$

$$= \frac{10+2+13+3}{4} = \frac{28}{4} = 7s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{5+0+9+0}{4} = \frac{14}{4} = 3.5s$$

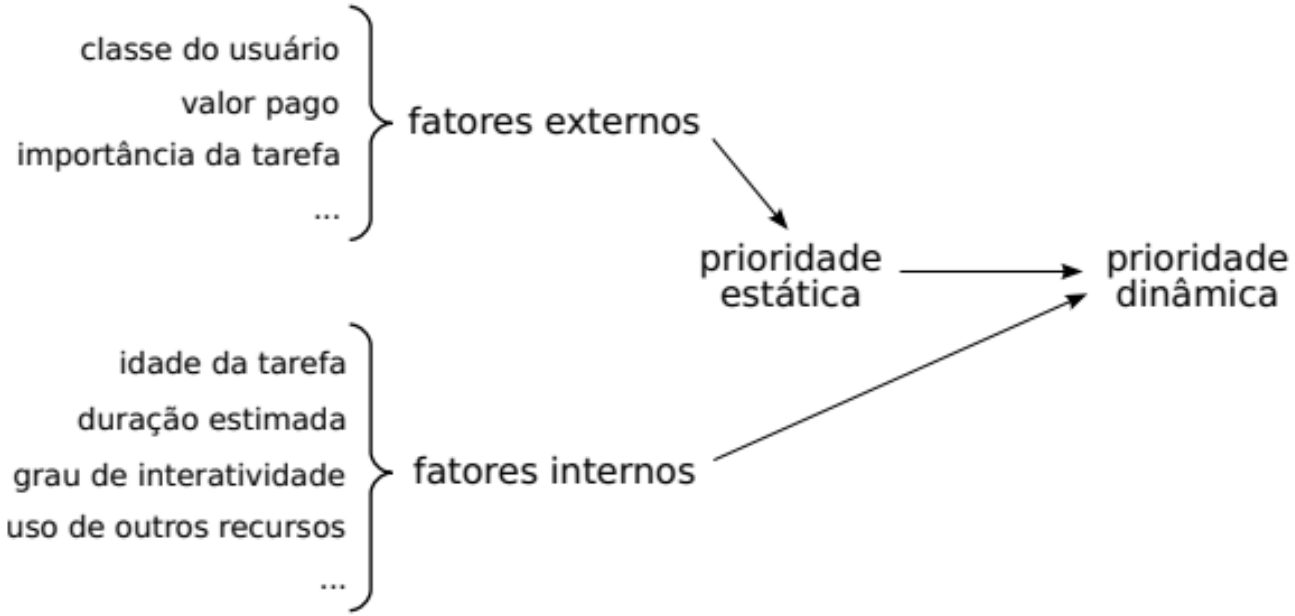
Definição de prioridades

- A definição da prioridade de uma tarefa é influenciada por diversos fatores, que podem ser classificados em dois grandes grupos:

- **Fatores externos** : são informações providas pelo usuário ou o administrador do sistema, que o escalonador não conseguiria estimar sozinho. Os fatores externos mais comuns são a classe do usuário (administrador, diretor, estagiário) o valor pago pelo uso do sistema (serviço básico, serviço *premium*) e a importância da tarefa em si (um detector de intrusão, um *script* de reconfiguração emergencial, etc.).

- **Fatores internos** : são informações que podem ser obtidas ou estimadas pelo escalonador, com base em dados disponíveis no sistema local. Os fatores internos mais utilizados são a idade da tarefa, sua duração estimada, sua interatividade, seu uso de memória ou de outros recursos, etc.

Definição de prioridades



Windows 2000 e sucessores

- processos e *threads* são associados a *classes de prioridade* (6 classes para processos e 7 classes para *threads*); a prioridade final de uma *thread* depende de sua prioridade de sua própria classe de prioridade e da classe de prioridade do processo ao qual está associada, assumindo valores entre 0 e 31. As prioridades do processos, apresentadas aos usuários no *Gerenciador de Tarefas*, apresentam os seguintes valores *default*:
- 4: *baixa* ou *ociosa*
- 6: *abaixo do normal*
- 8: *normal*
- 10: *acima do normal*
- 13: *alta*
- 24: *tempo-real*

Linux (núcleo 2.4 e sucessores)

Duas escalas de prioridades:

- *Tarefas interativas*: a escala de prioridades é negativa: a prioridade de cada tarefa vai de -20 (mais importante) a +19 (menos importante) e pode ser ajustada através dos comandos *nice* e *renice*. Esta escala é padronizada em todos os sistemas UNIX.

- • *Tarefas de tempo-real*: a prioridade de cada tarefa vai de 1 (mais importante) a 99 (menos importante). As tarefas de tempo-real têm precedência sobre as tarefas interativas e são escalonadas usando políticas distintas. Somente o administrador pode criar tarefas de tempo-real.

Inanição e envelhecimento de tarefas

- No escalonamento por prioridades básico, as tarefas de baixa prioridade só recebem o processador na ausência de tarefas de maior prioridade. Caso existam tarefas de maior prioridade frequentemente ativas, as de baixa prioridade podem sofrer de inanição (*starvation*), ou seja, nunca ter acesso ao processador.

- Exemplo: t1 com prioridade 3, t2 com prioridade 2 e t3 com prioridade 3

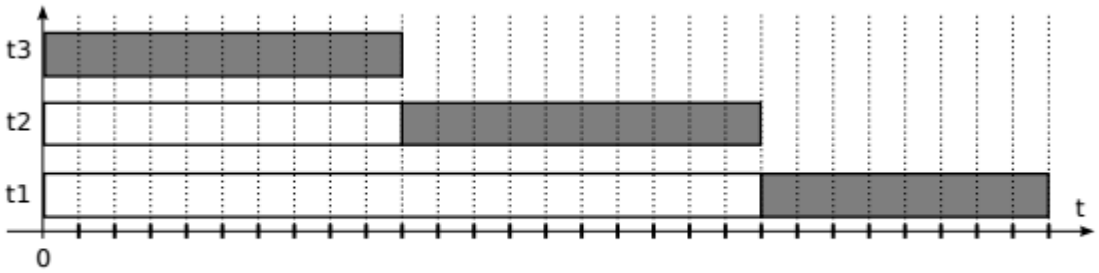


Figura 2.20: Escalonamento por prioridades.

Inanição e envelhecimento de tarefas

- Para evitar a inanição e garantir a proporcionalidade expressa através das prioridades estáticas, um fator interno denominado **envelhecimento** (*task aging*) deve ser definido. O envelhecimento indica há quanto tempo uma tarefa está aguardando o processador e aumenta sua prioridade proporcionalmente. Dessa forma, o envelhecimento evita a inanição dos processos de baixa prioridade, permitindo a eles obter o processador periodicamente.

- Exemplo de algoritmo:

Definições:

t_i : tarefa i
pe_i : prioridade estática de t_i
pd_i : prioridade dinâmica de t_i
N : número de tarefas no sistema

Quando uma tarefa nova t_n ingressa no sistema:

$pe_n \leftarrow$ *prioridade inicial default*
$pd_n \leftarrow pe_n$

Para escolher a próxima tarefa a executar t_p:

escolher $t_p \mid pd_p = max_{i=1}^{N}(pd_i)$
$pd_p \leftarrow pe_p$
$\forall i \neq p : pd_i \leftarrow pd_i + \alpha$

Inanição e envelhecimento de tarefas

- A Figura 2.21 ilustra essa proporcionalidade na execução das três tarefas *t*1, *t*2 e *t*3 com $p(t1) < p(t2) < p(t3)$, usando a estratégia de envelhecimento. Nessa figura, percebe-se que todas as três tarefas recebem o processador periodicamente, mas que *t*3 recebe mais tempo de processador que *t*2, e que *t*2 recebe mais que *t*1.

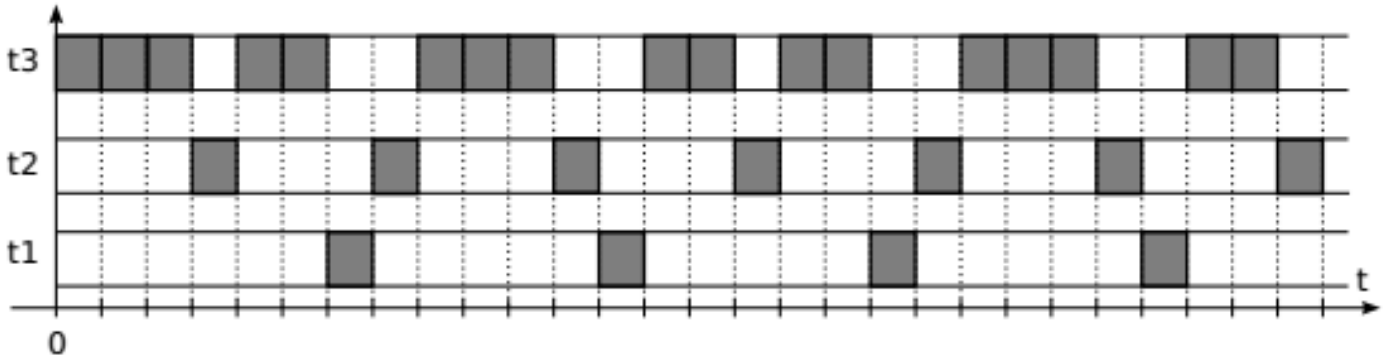


Figura 2.21: Escalonamento por prioridades com envelhecimento.

Inversão e herança de prioridades

- A inversão de prioridades consiste em processos de alta prioridade serem impedidos de executar por causa de um processo de baixa prioridade.

- Exemplo

- um determinado sistema possui um processo de alta prioridade p_a, um processo de baixa prioridade p_b e alguns processos de prioridade média p_m. Além disso, há um recurso *R* que deve ser acessado em **exclusão mútua**; para simplificar, somente p_a e p_b estão interessados em usar esse recurso.

Inversão e herança de prioridades

1. Em um dado momento, o processador está livre e é alocado a um processo de baixa prioridade p_b;
2. durante seu processamento, p_b obtém o acesso exclusivo a um recurso *R* e começa a usá-lo;
3. p_b perde o processador, pois um processo com prioridade maior que a dele (p_m) foi acordado devido a uma interrupção;
4. p_b volta ao final da fila de tarefas prontas, aguardando o processador; enquanto ele não voltar a executar, o recurso *R* permanecerá alocado a ele e ninguém poderá usá-lo;
5. Um processo de alta prioridade p_a recebe o processador e solicita acesso ao recurso *R*; como o recurso está alocado ao processo p_b, p_a é suspenso até que o processo de baixa prioridade p_b libere o recurso.

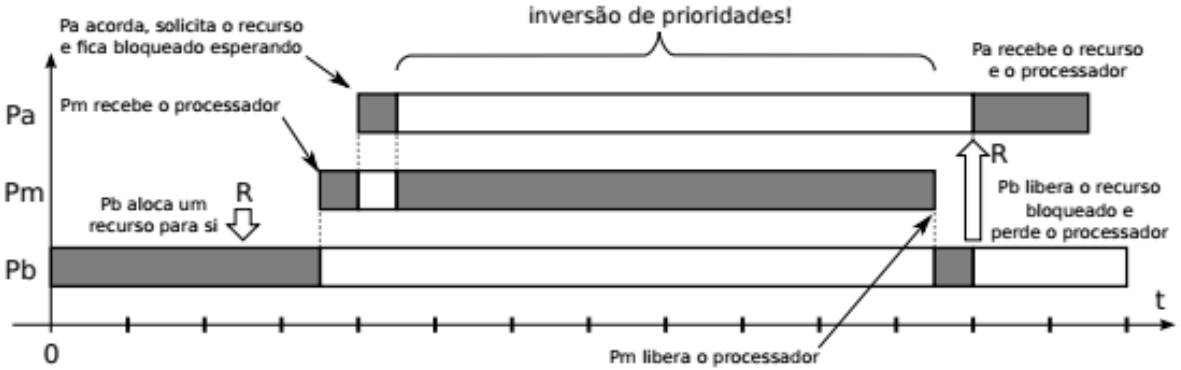


Figura 2.22: Cenário de uma inversão de prioridades.

Inversão e herança de prioridades

- Uma solução elegante para o problema da inversão de prioridades é obtida através de um *protocolo de herança de prioridade* [Sha et al., 1990]. O protocolo de herança de prioridade mais simples consiste em aumentar temporariamente a prioridade do processo *pb* que detém o recurso de uso exclusivo *R*. Caso esse recurso seja requisitado por um processo de maior prioridade p_a, o processo p_b "herda" temporariamente a prioridade de p_a, para que possa voltar a executar e liberar o recurso *R* mais rapidamente. Assim que liberar o recurso, p_b retorna à sua prioridade anterior.

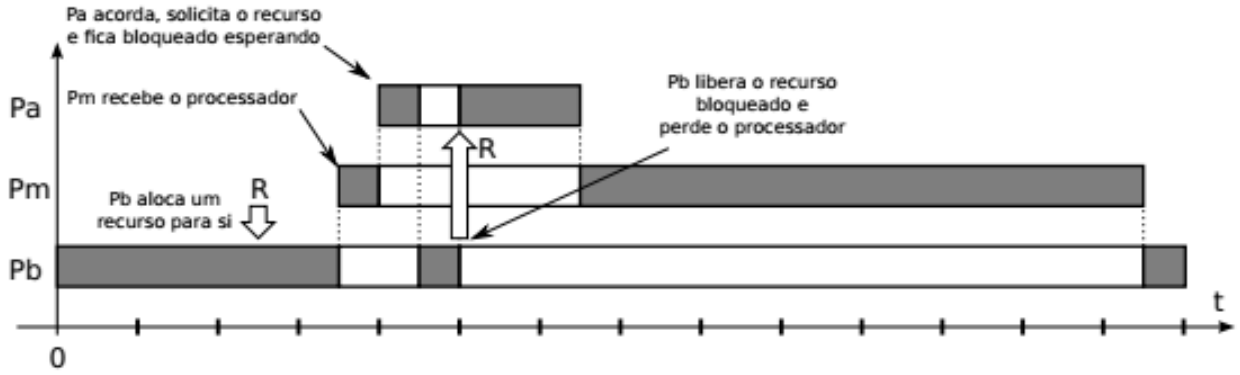


Figura 2.23: Um protocolo de herança de prioridade.

Atividade

- Pesquisar e ler sobre algoritmos de escalonamento de tempo real
- Resolução lista de exercícios

Escalonamento

- **Referências:**
  - Tanenbaum, A. – “Sistemas Operacionais – Projeto de Implementaçãoi” – terceira edição – cap 2 tópico 4; (disponível em formato digital na biblioteca a).
  - Maziero, C – “Sistemas Operacionais: Conceitos e Mecanismos” – cap 6; (disponível na internet - Versão compilada em 22 de setembro de 2023).
  - Silberschatz, A. – “Operating System Concepts”; 9ª. Edição; editora Wiley – cap 6.