Capítulo 5

Escalonamento de Tarefas em sistemas monoprocessados

Em sistemas de tempo-real o objetivo principal é de que cada processo produza resultados a tempo de serem úteis. Assim, é importante que se tenha um controle eficiente de qual processo ocupa a CPU a cada instante. Isso é garantido através de algoritmos de escalonamento especializados. Ocorre, como já visto, que STR são compostos por tarefas de vários tipos, o que torna impraticável a utilização de uma solução geral para o escalonamento. Para atender aos vários "tipos" de STR foram propostos diversos algoritmos diferentes, atendendo a propriedades distintas dos processos em execução, segundo diferentes métricas de avaliação. Ao longo desse capítulo examinaremos primeiro esses conceitos fundamentais e depois um conjunto de algoritmos de escalonamento adequados para sistemas monoprocessados.

5.1 Conceitos fundamentais

Para melhor compreender a aplicabilidade de cada algoritmo é importante revisar as diferentes propriedades e tipos de tarefas (processos). Temos então os seguintes tipos de tarefas:

- **Periódicas:** ocorrem em intervalos predefinidos, tendo que ser atendidas dentro de seus deadlines;
- **Aperiódicas:** são disparadas por eventos não-periódicos, tendo normalmente um deadline não-rígido (**soft deadline**);
- **Esporádicas**, que se comportam como as aperíodicas, porém com deadline rígido (**hard deadline**);
- **Não tempo-real**, que não têm restrição de tempo, tipicamente não-periódicas.

Em STR a preocupação maior é com tarefas periódicas e esporádicas, por serem de certa forma críticas dentro do sistema. Independente desse fato é importante observar que o objetivo de cada política de escalonamento é evitar a perda de deadlines das tarefas. Isso, entretanto, só é possível dentro de condições determinadas e, no caso da perda de deadline ser inevitável, é preciso que o escalonador procure fazer com que a tarefa que perder o deadline seja a menos crítica para o sistema.

Essas condições indicam que escalonamentos podem ser válidos mesmo com eventuais perdas de deadline. Formalmente, um escalonamento válido é aquele que pode ser executado, mesmo que não seja factível do ponto de vista de atendimento dos deadlines. Assim, para um escalonamento ser válido, independente da política usada, deve atender às seguintes condições:

1. Todo processador é alocado a no máximo uma tarefa por vez;
2. Toda tarefa é atribuída a no máximo um processador em qualquer instante;
3. Nenhuma tarefa é alocada antes de sua ocorrência;
4. Todas as condições de precedência ou uso são atendidas;
5. O tempo alocado para cada tarefa é igual ao seu tempo de execução.

Com isso temos as seguintes características para escalonamentos:

1. Um escalonamento é factível se todas as tarefas terminam dentro de seus deadlines;
2. Um conjunto de tarefas é escalonável por um dado algoritmo se o escalonador sempre produz escalonamentos factíveis para ele;
3. Um algoritmo de escalonamento é dito ótimo se para um conjunto de tarefas que tenham um escalonamento factível, então ele sempre produz escalonamento factíveis;
4. Se um algoritmo ótimo não obtém um escalonamento factível para um conjunto de tarefas, então esse conjunto não é escalonável por qualquer algoritmo;
5. O desempenho de um escalonador pode ser medido por várias métricas, tais como:
   a) **Tardiness:** indica o atraso relativo das tarefas, importando seus valores médio e máximo;

b) **Lateness:** indica o atraso ou o adiantamento das tarefas, sendo negativo para tarefas adiantandas;

c) **Makespan (ou tempo de resposta):** indica o tempo necessário para executar todas as tarefas;

d) **Miss Rate:** é a taxa de tarefas que perdem seu deadline;

e) **Loss Rate:** é a taxa de tarefas que são descartadas pois levariam a perdas de deadlines de outras tarefas mais prioritárias.

Finalmente, para que as métricas definidas possam ser calculadas é preciso medir alguns parâmetros da execução de um conjunto de tarefas. Esses parâmetros são:

- Tempo de chegada (release ou arrival time), r_i, é o instante em que a tarefa passa a ser escalonavel, podendo variar dentro de um intervalo $[r_i^-, r_i^+]$;
- Deadline relativo, D_i, é o tempo máximo permitido de resposta para uma tarefa;
- Deadline absoluto, d_i, é o instante em que a tarefa deve estar concluída, sendo igual ao seu tempo de chegada mais deadline relativo;
- Tempo de execução (ou carga), c_i, é a quantidade de tempo necessária para completar uma tarefa, podendo estar no intervalo $[c_i^-, c_i^+]$;
- Período, P_i, é o intervalo mínimo entre chegadas consecutivas de uma tarefa periódica.

5.2 Abordagens para escalonamento

Um algoritmo de escalonamento pode ser definido a partir de diferentes condições. Entre elas temos a forma como o escalonamento é atualizado ou em que momento ocorrem alterações na ocupação da CPU. Assim, podemos definir as seguintes abordagens para escalonamento:

- Dirigida por tempo, em que as decisões são feitas em momentos específicos, escolhidos antes da execução. Um exemplo é o round-robin ponderado, em que a cada ciclo as tarefas recebem ω_t fatias de execução, sendo ω_t determinado pela prioridade da tarefa;
- Dirigida por prioridade, em que as decisões são feitas a partir de eventos como chegada de tarefas ou término de execução, compreendendo portanto uma gama bastante ampla de algoritmos de escalonamento;

- Escalonamento estático, em que as tarefas são alocadas estaticamente aos processadores, sem possibilidade de migração, exceto na reconfiguração do sistema;
- Escalonamento dinâmico, em que as tarefas são alocadas aos processadores livres, com possibilidade de migração se houver necessidade;
- Escalonamento *on-line*, em que o escalonador atua, isto é, determina que tarefas ocuparão cada processador, durante a execução do sistema, sendo que necessariamente são dinâmicos;
- Escalonamento *off-line*, em que a determinação do escalonamento é feita *a priori*, antes de iniciar a execução do sistema.

Antes de iniciarmos o estudo de algoritmos específicos é importante identificar algumas propriedades derivadas das abordagens aqui descritas. A primeira delas é que sistemas dirigidos por prioridade podem apresentar comportamento anômalo em virtude de condições que levem a ótimos locais em momentos específicos. Isso ocorre principalmente quando existe compartilhamento de recursos de uso exclusívo entre tarefas. Pelo mesmo motivo também é difícil determinar a previsibilidade de execuções quando o escalonamento é feito por prioridade.

Existem dezenas de algoritmos de escalonamento propostos para tarefas de tempo-real. Assim, trataremos aqui apenas um pequeno subconjunto definido pela representatividade de cada algoritmo e pela cobertura de diferentes características das tarefas a serem escalonadas. Iniciamos com alguns algoritmos básicos, que consideram tarefas como sendo periódicas e independentes umas das outras (EDF, LST e Taxa Monotônica). Depois trataremos algoritmos para tarefas esporádicas e, por último, algoritmos para tarefas com regiões críticas.

5.3 Algoritmos para tarefas periódicas independentes

5.3.1 Earliest Deadline First (EDF)

Aloca tarefas com deadline mais próximo primeiro. É um algoritmo ótimo para tarefas independentes e se preempção é permitida.

5.3.2 Least-Slack-Time First (LST)

Aloca primeiro a tarefa com menor folga para cumprir o seu deadline. Essa folga (*laxity*) é dada pelo tempo que ainda sobraria antes de seu *deadline* caso a

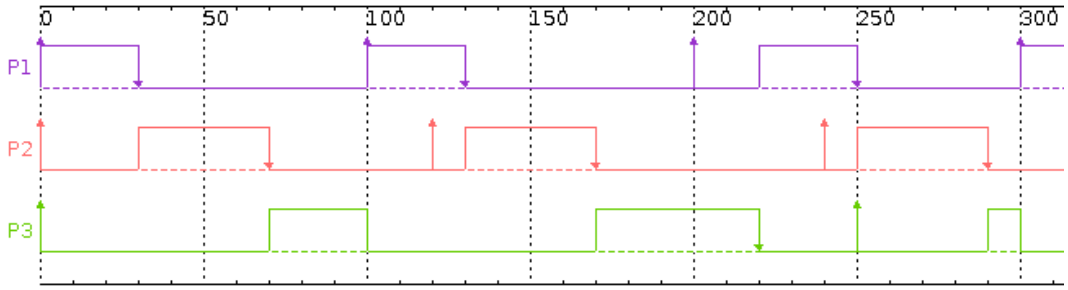

Figura 5.1: Escalonamento para o EDF produzido pelo RTsim

execução da tarefa ocorresse até seu final a partir do momento atual. Isso pode ser representado pela equação:

$$laxity = deadline - t_{atual} - carga\_ainda\_não\_executada \tag{5.1}$$

Deve-se observar que o LST também é um algoritmo ótimo para tarefas independentes e com preempção.

Exemplo:

Apenas para comparação, a aplicação destes algoritmos produz resultados diferentes (Figuras 5.1 e 5.2) se considerado o seguinte conjunto de tarefas:

Tarefa	Carga	Período/Deadline
P_1	30	100
P_2	40	120
P_3	80	250

A diferença surge já em $t = 100$, quando o EDF irá escalonar a tarefa P_1, que ocorreu em $t = 100$ e tem *deadline* em $t = 200$, enquanto o LST irá escalonar P_3, que havia ocorrido em $t = 0$ e terá, naquele momento, folga igual a $LS = 0,667$ (contra $LS = 0,700$ de P_1).

5.3.3 Algoritmo Taxa Monotônica (TM)

O algoritmo Taxa Monotônica, assim, como o EDF e LST, é um algoritmo ótimo para tarefas periódicas independentes e com possibilidade de preempção. A

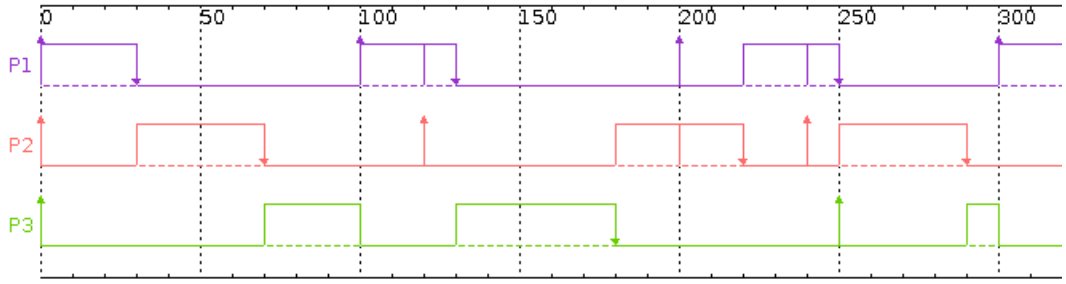

Figura 5.2: Escalonamento para o LST produzido pelo RTsim

diferença entre ele e os algoritmos anteriores é sua estabilidade, isto é, independente do momento do escalonamento a tarefa a ser alocada será sempre a mesma, dado um mesmo conjunto de tarefas escalonáveis. Isso ocorre pois o critério adotado é o do período da tarefa, que obviamente não muda de um instância da tarefa para outra.

Para a aplicação do Taxa Monotônica (TM) considera-se que as tarefas têm seu *deadline* igual ao seu período e que tarefas de maior prioridade têm períodos menores. Com essas condições é possível demonstrar que o TM garante, em situações de sobrecarga, que as tarefas que perderão o *deadline* serão as tarefas menos prioritárias.

Essa previsibilidade do TM permite a definição de testes de escalonabilidade mais precisos. O primeiro deles é o teste de escalonabilidade limite, definido por Liu e Layland, que diz que um conjunto de n tarefas periódicas será sempre escalonável se a utilização da CPU atender ao seguinte teste:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n.(2^{1/n} - 1) \tag{5.2}$$

Essa expressão, quando considerando um número infinito de tarefas, resulta em uma utilização máxima dada por $U = \ln 2 \approx 0.693$.

Isso implica em que qualquer conjunto de tarefas cuja utilização for menor que 69% será sempre escalonável, independente do número de tarefas. Para valores de utilização maiores é necessário aplicar o teste. Assim, considerando um conjunto de tarefas dado por:

Tarefa	Carga	Período	Uso
τ_1	20	100	0.200
τ_2	40	150	0.267
τ_3	100	350	0.286

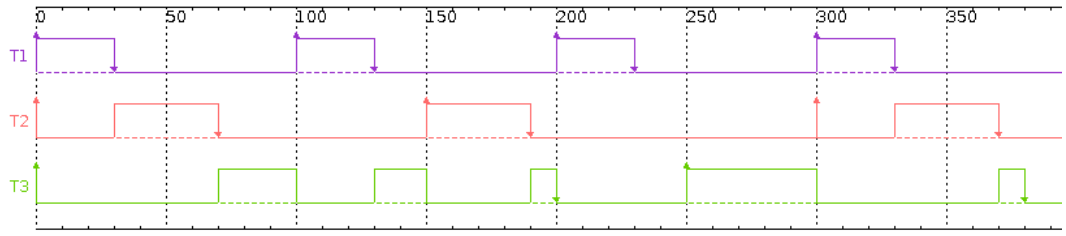

Figura 5.3: Gráfico de escalonamento produzido pelo RTsim

Temos que o limite máximo dado pela Equação 5.2 é de 0.779, que é suficiente para escalonar o conjunto de tarefas, que tem uso total de 0.753. Logo o conjunto de tarefas é escalonável.

Entretanto, se a carga de τ_1 passar para 40, teremos $U = 0.953$, que é bem maior que o valor de teste. Entretanto, mesmo com essa nova carga o conjunto continua escalonável (como visto na Figure 5.3) e isso pode ser confirmado pelo chamado **Teste de Escalonabilidade Exato**.

Teste de Escalonabilidade Exato

Liu e Layland provaram que um conjunto de tarefas periódicas é escalonável se a primeira instância de ocorrência de cada tarefa atende ao seu *deadline* quando todas as tarefas iniciarem em $t = 0$.

Esse critério pode ser verificado através dos chamados pontos de escalonamento, isto é, pontos em que uma nova instância de alguma tarefa se torna disponível (ocorre). Nesses pontos calcula-se a soma da carga teoricamente consumida por todas as tarefas que já ocorreram antes desse ponto e, se a soma for menor do que o valor do ponto de escalonamento, então as tarefas serão escalonáveis, ou seja:

Dado $\tau_1, ..., \tau_p$, com pontos de escalonamento $P_1, ..., P_k$, então o escalonamento será factível se:

$$\sum_{i=1}^{p} n_i . C_i \leq P_n$$

Como exemplo, a aplicação desse teste no conjunto de tarefas anteriormente definido resulta em:

$$P_i = \{100, 150, 200, 300, 350\}$$

$$P_1 = C_1 + C_2 + C_3 = 40 + 40 + 100 > 100$$
$$P_2 = 2.C_1 + C_2 + C_3 = 80 + 40 + 100 > 150$$
$$P_3 = 2.C_1 + 2.C_2 + C_3 = 80 + 80 + 100 > 200$$
$$P_4 = 3.C_1 + 2.C_2 + C_3 = 120 + 80 + 100 = 300 \ \checkmark$$
$$P_5 = 4.C_1 + 3.C_2 + C_3 = 160 + 120 + 100 > 350$$

Desse teste é possível identificar que o conjunto de tarefas é escalonável, uma vez que em $t = 300$ todas as tarefas que terão ocorrido até aquele momento terão atendido seus *deadlines*, mesmo não existindo folga nesse atendimento.

5.4 Algoritmos para tarefas não-periódicas

O tratamento de tarefas esporádicas e aperiódicas não pode ser feito do mesmo modo que se tratam tarefas periódicas. A impossibilidade de saber quando e de que forma ocorrerão os eventos que disparam uma tarefa desse tipo, faz com que os escalonadores já apresentados tenham pouca eficiência. Para tarefas não-periódicas existem três modos básicos de escalonamento:

- **Por execução em *background***, em que o escalonador trabalha as tarefas periódicas usando algoritmos como o TM. Para as demais tarefas o escalonador usa os intervalos em que a CPU fica ociosa.

  A eficiência dessa política depende da porcentagem de ocupação da CPU pelas tarefas periódicas e da frequência de ocorrência das não-periódicas. Uma característica importante dessa estratégia de escalonamento é a de não alterar o atendimento das tarefas periódicas.

- **Por interrupção**, em que se interrompe a periódica em execução toda vez que ocorrer uma não-periódica. Evidentemente isso prioriza tais tarefas mas pode causar problemas desnecessários no atendimento das periódicas.

- **Por *polling***, em que se cria uma tarefa periódica virtual, chamada servidor aperiódico, cuja carga é aproveitada para executar as tarefas não periódicas.

  A eficiência do *polling* depende do projeto do servidor, ou mais precisamente do dimensionamento de seu período, carga e política de reabastecimento da carga.

Destas políticas examinaremos aqui os algoritmos de *polling* denominados Servidor Esporádico (*Sporadic server*) e Servidor por Deferência (*Deferrable server*).

5.4.1 Deferrable Server - DS

Usa um servidor de *polling* com carga C_{DS} e período T_{DS}, sendo que a carga é reabastecida em todo instante múltiplo de T_{DS}. Tarefas não-periódicas são atendidas sempre que a carga C_{DS} for maior que zero, ocupando a CPU por um período igual à carga, ou até um novo reabastecimento.

5.4.2 Sporadic Server - SS

Usa um servidor com carga C_{SS} e período T_{SS}, atendendo tarefas não-periódicas até que consumam um tempo de CPU equivalente a C_{SS} a cada T_{SS} unidades de tempo.

O reabastecimento de uma carga consumida C_i ocorrerá apenas depois de T_{SS} unidades de tempo do início do consumo, e reabastecerá apenas o total consumido C_i.

5.4.3 Projeto do servidor de *polling*

A eficiência dos mecanismos de *polling* depende do projeto correto da carga e período do servidor aperiódico. Para tanto seu projeto deve levar em consideração as características do sistemas em que será empregado, segundo diferentes ponderações. Os principais parâmetros a serem considerados são:

- Maior período das tarefas periódicas;
- Folga calculada pelo Teste de Escalonabilidade Exato;
- Cargas e *deadlines* das tarefas não-periódicas;
- Intervalo mínimo entre chegadas das tarefas não-periódicas;
- Prioridade relativa entre periódicas e não-periódicas;

Em particular, para o RTsim o projeto do servidor de *polling* segue uma abordagem baseada no teste de escalonabilidade exato (TEE) e pelos períodos das periódicas. Isso implica nas seguintes equações para carga e período do servidor de *polling* usado para a simulação dos servidores esporádicos e por deferência:

$$Período = Menor\ Período\ das\ Periódicas \tag{5.3}$$

$$Carga = \frac{Maior\ folga\ pelo\ TEE}{Maior\ Período\ das\ Periódicas/Período} \tag{5.4}$$

Exemplo:

Considerando o conjunto de tarefas periódicas dado a seguir é possível determinar o servidor de *polling* para as aperiódicas. É importante notar que esse servidor terá as mesmas características tanto para o DS como para o SS.

Tarefa	C	P
t_1	30	100
t_2	40	200
t_3	120	400

Teste de escalonabilidade exato:

$P = \{40, 80, 100, 120, 160, 200\}$

$P_1(100) = 30 + 40 + 120 = 190 > 100$

$P_2(200) = 60 + 40 + 120 = 220 > 200$

$P_3(300) = 90 + 80 + 120 = 290 < 300$

$P_4(400) = 120 + 80 + 120 = 320 < 400$ (maior folga = 80)

Pela aplicação das Equações 5.3 e 5.4 chega-se aos valores de carga e período para o servidor:

$TDS = TSS = 100, CDS = CSS = 20$

Supondo ocorrencias de $Ap(C = 30, D = 300)$ em $t_1 = 60$ e $t_2 = 150$ temos como resultado os gráficos apresentados nas Figuras 5.4 e 5.5. Neles se percebe que para o servidor por deferência ocorre o escalonamento antecipado das tarefas aperiódicas em $t = 100$, $t = 150$ e $t = 200$, logo após o reabastecimento do servidor. Já para o servidor esporádico a execução dessas tarefas é postergado para $t = 160$ e $t = 260$, que são os instantes em que ocorrerá o reabastecimento do servidor a partir da carga consumida em $t = 60$.

5.5 Escalonamento de tarefas com compartilhamento de recursos

Os algoritmos já vistos consideram que as tarefas são independentes, inclusive quanto ao compartilhamento de recursos. Esta, entretanto, é uma hipotese

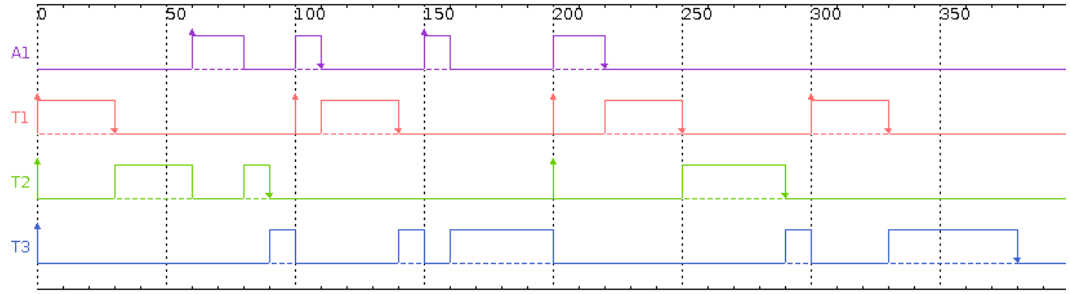

Figura 5.4: Gráfico de escalonamento produzido pelo Servidor por Deferência

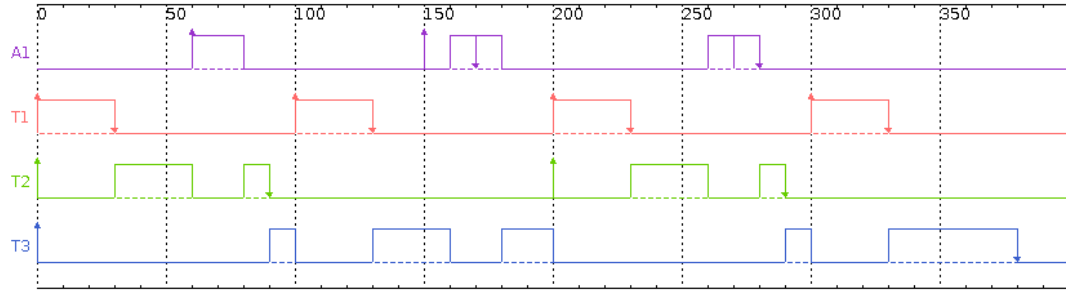

Figura 5.5: Gráfico de escalonamento produzido pelo Servidor Esporádico

frequentemente inválida pela necessidade de troca de informações entre tarefas. Quanto isso ocorre é preciso usar regiões críticas com exclusão mútua, o que complica a garantia de escalonabilidade das tarefas.

Quanto temos tarefas com diferentes prioridades compartilhando recursos pode ocorrer inversão de prioridade. Isso ocorre quando uma tarefa de alta prioridade é bloqueada ao acessar um recurso em posse de alguma tarefa de baixa prioridade, que não consegue acesso a CPU pois tarefas de prioridade intermediárias ficam executando.

Assim, essas tarefas de prioridade intermediária acabam atrasando a tarefa de prioridade mais alta, podendo resultar na perda de deadline daquela tarefa. Essa situação é o que se chama inversão de prioridade, que pode ser resolvida se a tarefa de baixa prioridade, que possui o recurso, tiver sua prioridade elevada temporariamente, até sair da região crítica.

Isso se chama Herança de Prioridade, sendo que escalonadores que a apliquem são conhecidos como protocolos de herança de prioridade. Desses algoritmos examinaremos aqui os protocolos Troca de Prioridade (priority exchange) e Topo de Prioridade (priority ceiling).

5.5.1 Troca de Prioridade

Neste algoritmo o escalonamento ocorre normalmente, usando políticas como TM, DS, etc, até que uma tarefa seja bloqueada ao tentar acesso a uma RC. Nesse caso, se a tarefa com posse do recurso guardado pela RC tiver prioridade **menor** do que a tarefa bloqueada, então recebe a prioridade dessa tarefa até liberar a RC, quando voltará a ter sua prioridade original.

5.5.2 Topo de Prioridade

O problema com a Troca é que se existirem tarefas ainda mais prioritárias, elas assumirão a CPU quando ocorrerem. Eventualmente essas tarefas podem ser bloqueadas na mesma RC (se usarem o mesmo recurso). Isso implica numa cadeia de trocas, retardando a liberação da RC pela tarefa com prioridade herdada. Isso é corrigido pelo Topo de Prioridade ao fazer com que a tarefa de baixa prioridade herde a prioridade da tarefa de maior prioridade (topo) entre as que podem acessar a RC que causou o bloqueio.

Exemplo:

Neste exemplo se considera o conjunto de tarefas periódicas listadas na tabela a seguir, com informações sobre uso das regiões críticas (semáforos) e o instante de sua primeira ocorrência no sistema.

Tarefa	Carga	Período	Prioridade	1ª chegada	Semáforo	T_{RC}	I_{RC}
T_1	30	120	1	40	S_1	10	7
T_2	40	250	3	10	S_1 S_2	10 20	9 3
T_3	30	300	2	15	S_2	15	12
T_4	40	300	4	0	S_1	20	5

Para essas tarefas, a atuação dos escalonadores Troca de Prioridade e Topo de Prioridade ocorrerá de modo distinto, especialmente na prioridade herdada pela tarefa T_4 quando bloquear a tarefa T_2. As saídas produzidas pelo RTsim podem ser vistas nas Figuras 5.6 e 5.7. Pode ser visto em 5.6 que o Troca de Prioridade faz com que em $t = 31$ a tarefa T_4 herde a prioridade de T_2, permitindo a execução de T_1 a partir de $t = 40$. Com isso, haverá nova herança de prioridade quando T_1 tentar entrar na RC guardada por S_1.

Já para o algoritmo Topo de Prioridade, em $t = 31$ a tarefa irá receber a prioridade de T_1, fazendo com que assuma a cpu, sem interrupções por tarefas de

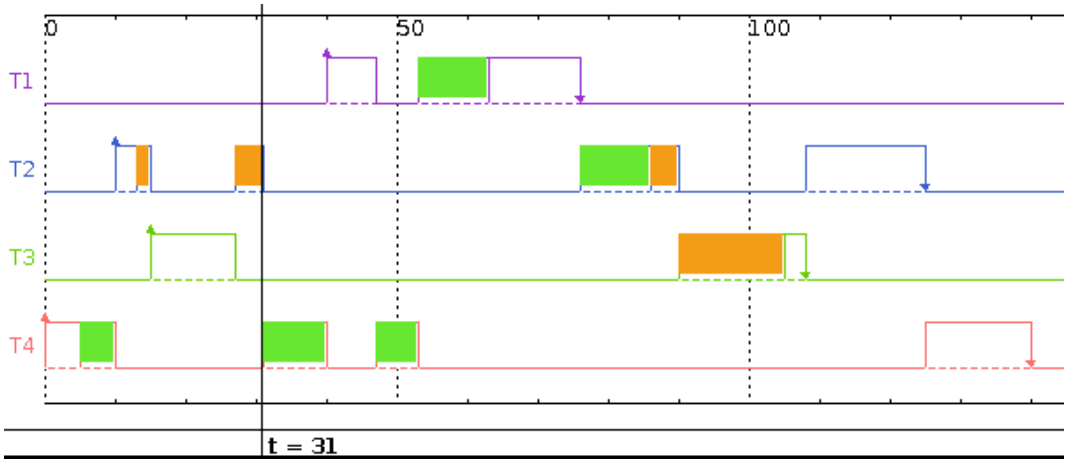

Figura 5.6: Gráfico de escalonamento produzido pelo Troca de Prioridade

prioridade menor ou igual à prioridade de T_1, até que saia da região crítica, em $T = 46$.

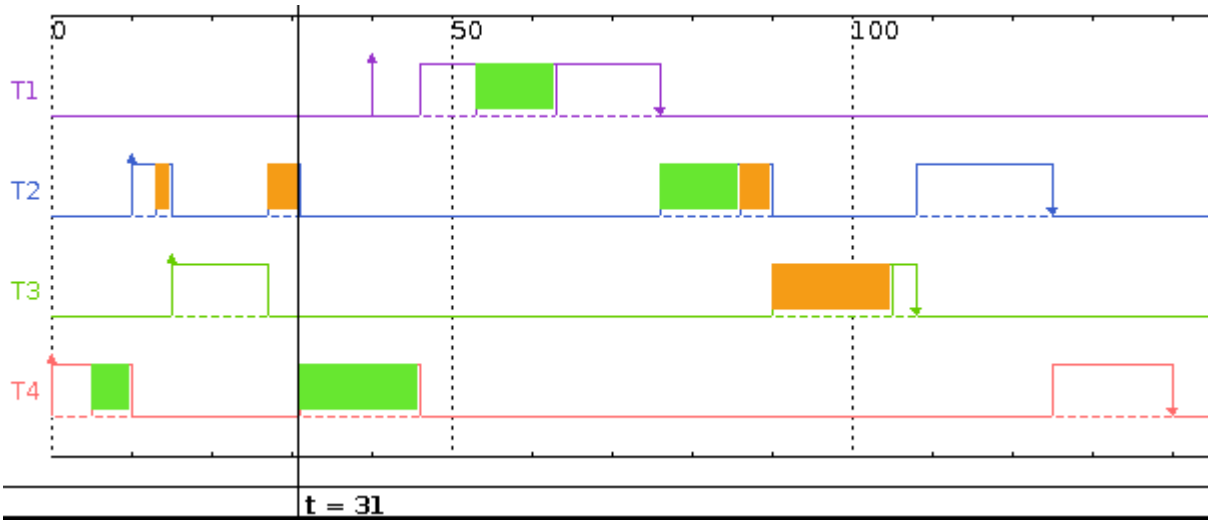

Figura 5.7: Gráfico de escalonamento produzido pelo Topo de Prioridade