UFMG Engenharia de Controle e Automação

Automação em Tempo Real

Módulo 9:
Escalonadores de Tempo Real

Professor: Constantino Seixas Filho

sexta-feira, 26 de maio de 2006

1

Sistema de Tempo Real

UFMG

- Um sistema é classificado como de tempo real quando a execução de toda e qualquer tarefa deve se dar dentro de uma faixa de tempo estipulada a priori. Isto é importante para que o sistema possa realizar tarefas periódicas ou reagir a estímulos do meio sempre dentro de um tempo previsível
- No caso da automação industrial esse tempo depende das constantes de tempo do processo a ser controlado

2

Hard Real Time e Soft Real Time

- **Sistema *hard real time:***
  Não é permitido que um limite de tempo especificado (*deadline*) seja ultrapassado em nenhuma situação. Caso este *deadline* não seja respeitado a atuação do sistema resultaria ruinosa ou de graves conseqüências para a segurança das pessoas envolvidas, para o processo, para os ativos industrias ou para o meio ambiente

- **Sistema soft real time:**
  Se o sistema falhar em responder dentro do tempo estipulado o seu desempenho cairá, mas os danos são considerados suportáveis

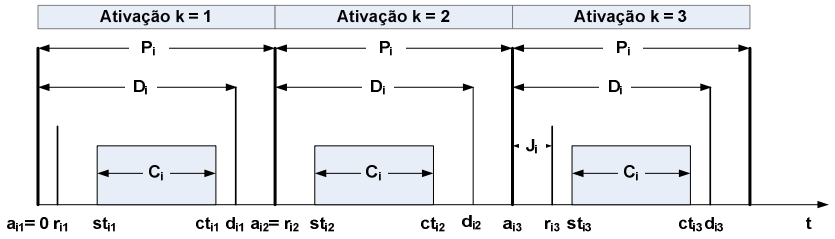
3

Características de um RTOS

- Ser multitarefas e multithreading
- Possuir um escalonador preemptivo baseado num mecanismo de prioridades
- Possuir um grande número de prioridades (> 128) a fim de que threads diferentes não tenham que executar em uma mesma prioridade
- Possuir objetos para implementar a exclusão mútua tais como semáforos.
- Possuir primitivas de sincronismo entre as threads como eventos, temporizadores, filas
- Possuir diretivas para a comunicação síncrona e assíncrona entre threads e entre processos (IPC = *Inter Process Communication*)
- Possuir filas organizadas por prioridade nos objetos de sincronização, ao invés de FIFOs
- Possuir temporizadores precisos na casa de 0.1 ms assíncronos com o relógio do processador
- Possuir mecanismos para impedir que haja inversão de prioridades.
- Possuir baixo tempo de latência de interrupção
- Possuir mecanismo de interrupções aninhadas em que uma Rotina de Serviço de Interrupção (ISR) consegue preemptar outra ISR de menor prioridade
- Apresentar boa performance temporal tal como: baixo tempo de chaveamento de contexto, baixo tempo para criação de threads, baixo overhead nas chamadas ao sistema operacional para sincronismo, etc

4

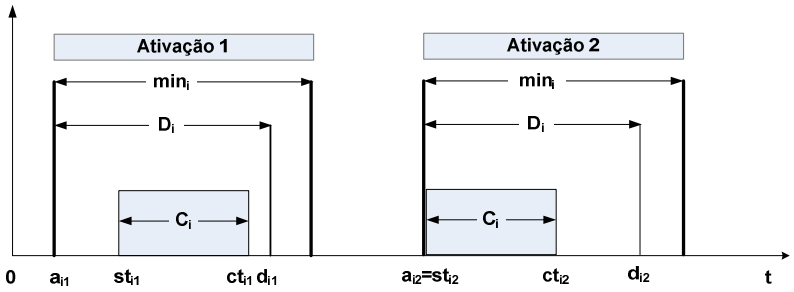
Requisitos de execução de um conjunto de tarefas
Tarefas periódicas



- Ci corresponde ao tempo de computação (*computation time*) da tarefa de ordem i.
- Pi é o período de execução da tarefa.
- Di é o *deadline* ou data limite da tarefa.
- si é o tempo de início (*start time*) da tarefa naquela ativação.
- ci é o tempo de término (*completion time*)
- ai é o tempo de chegada (*arrival time*) da tarefa. É neste instante que o escalonador toma conhecimento da ativação da tarefa.
- ri é o tempo de liberação (*release time*) que corresponde ao instante de inserção da tarefa na fila de pronto
- Ji é o Jitter ou pior caso do tempo de atraso de liberação da tarefa
- Ri ou tempo de resposta Ri é o pior caso da diferença entre o tempo de chegada da tarefa i e o seu tempo de término

5

Tarefas Assíncronas



- São caracterizadas pela tripla (Ci, Di, mini):
- Ci é o tempo de computação da tarefa
- Di o *deadline*
- Mini o intervalo mínimo entre duas requisições da tarefa

6

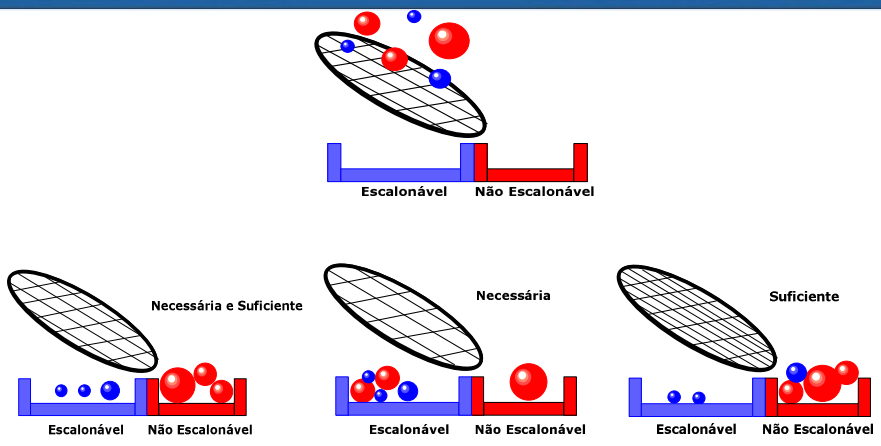
Mecanismos de Escalonamento

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	20	20
T2	500	50	10
T3	200	40	20
T4	100	30	30

- Necessidades de um conjunto de tarefas (*task set*)
- Ui = Ci / Pi

7

Testes a priori



8

Taxa Monotônica (*Rate Monotonic* – RM)

- As condições para utilizar este algoritmo são:
  - As tarefas são periódicas e independentes
  - O tempo de computação de todas as tarefas é conhecido e não varia entre execuções da tarefa. Na verdade tomamos a situação de pior caso como o tempo padrão.
  - O tempo de troca de contexto é nulo
  - O *deadline* coincide com o período de cada tarefa *(Di = Pi)*
- O RM atribui prioridades estaticamente na ordem inversa aos períodos de solicitação das tarefas
- A condição **suficiente** para que um dado conjunto de tarefas seja exeqüível é dada por:

$$\sum_i U_i \le n*(2^{\frac{1}{n}}-1) \qquad \lim_{n\to\infty} n*(2^{\frac{1}{n}}-1) = \ln 2 \approx 0,69$$

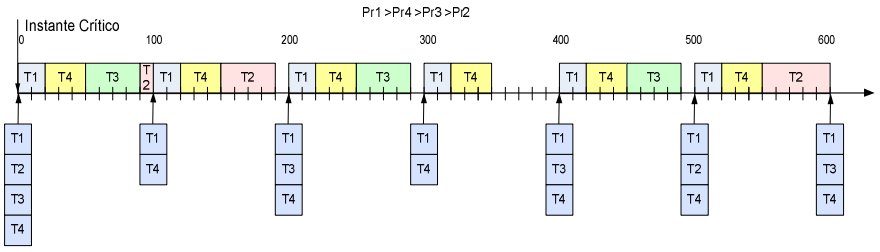
9

Diagrama de escalonamento com RM

- No nosso exemplo, temos n=4

$$\sum_i \frac{C_i}{P_i} = 0.80 \qquad n*(2^{\frac{1}{n}}-1) = 0,757$$

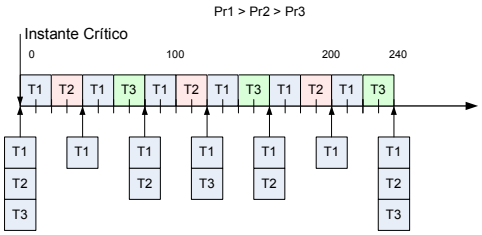
- Atribuição de prioridades: Pr1 > Pr4 > Pr3 > Pr2



10

Períodos múltiplos do menor período

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	40	20	50
T2	80	20	25
T3	120	20	16.67

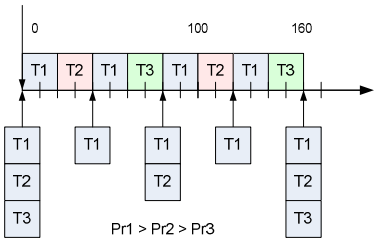


- A Utilização total do sistema = 91.67 %
- Condição suficiente é 78%
- O sistema se mostrou factível

11

Períodos Múltiplos entre si

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	40	20	50
T2	80	20	25
T3	160	20	25

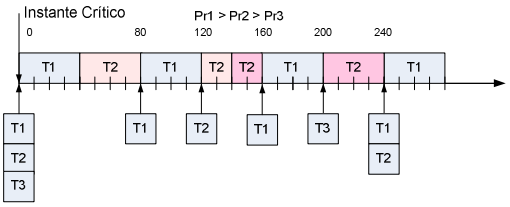


- Fator de utilização total = 1
- Esta é a condição de melhor utilização do processador para um escalonador RM.

12

Robustez em condições de sobrecarga permanente

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	80	40	50
T2	120	60	50
T3	200	50	25

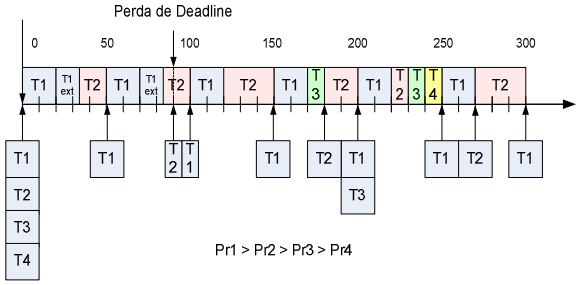


- As tarefas de menor prioridade podem ser simplesmente bloqueadas e não conseguem executar
- A tarefa T3 simplesmente não consegue ser escalonada e nunca é executada

13

Robustez em condições de sobrecarga transitória

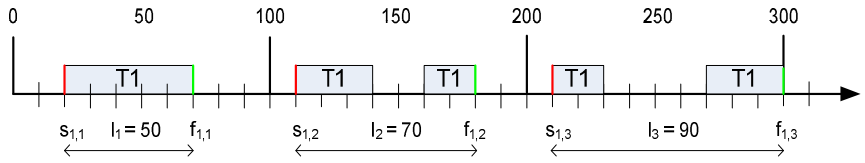
Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	50	20..35	40..70
T2	90	30	30
T3	200	10	5
T3	300	10	3



- Se o sistema se torna sobrecarregado, apenas a tarefa de maior prioridade (menor período) garantidamente não perde seu *deadline*. Nada se pode dizer sobre as demais tarefas

14

Latência de Entrada-Saída



- Tempo máximo entre o tempo de finalização e de começo de uma tarefa considerando que a tarefa Ti adquire as entradas no início e define as saídas no final da sua execução

$$L_i = \max_k (f_{i,k} - s_{i,k})$$

- L1 = max(50, 70, 90) = 90 ms

15

Menor Data Limite Primeiro (*Earliest Deadline First*)

- Ela receberá uma prioridade de acordo com o seu *deadline* e será inserida na fila de pronto que é ordenada por prioridades
- A alocação de prioridades portanto não é fixa e sim dinâmica e continua *on-line*, isto é não depende de escolhas e definições de um operador humano
- Um determinado *task set* será factível se e somente se:

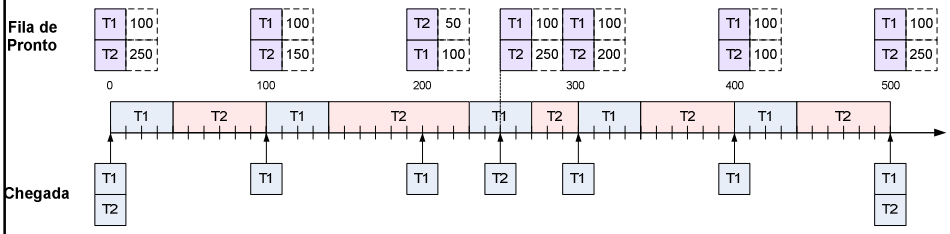
$$\sum_i U_i \leq 1$$

16

Exemplo com EDF

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	40	40
T2	250	150	60

$$\sum_i \frac{C_i}{P_i} = 1 \leq 1$$



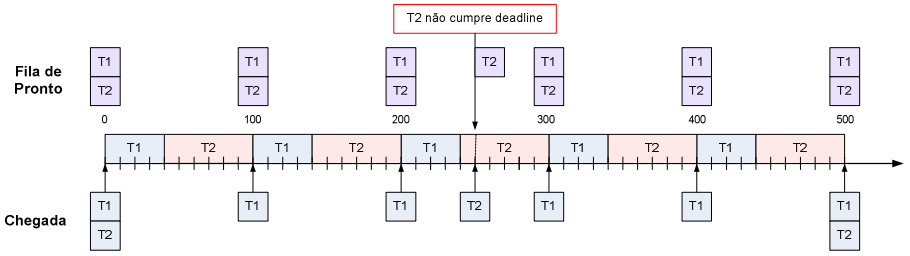
17

Exemplo com RM

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	40	40
T2	250	150	60

$$\sum_i \frac{C_i}{P_i} = 1$$

$$n * (2^{\frac{1}{n}} - 1) = 0,8284$$



18

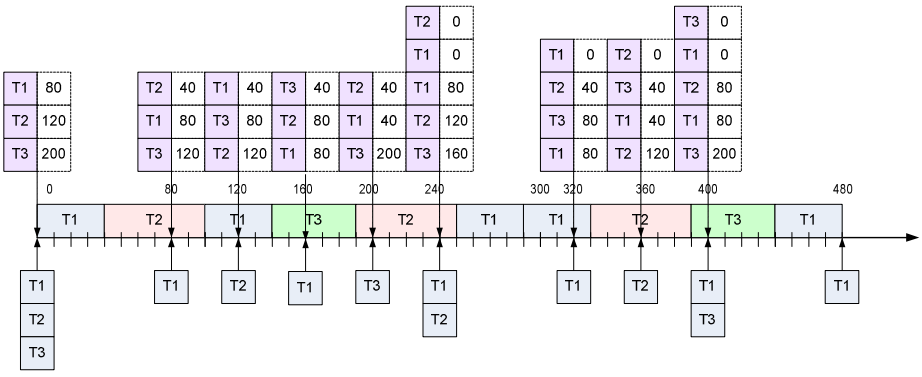
Robustez em condições de sobrecarga com EDF

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	80	40	50
T2	120	60	50
T3	200	50	25

- Em condições de sobrecarga, o algoritmo EDF tem um comportamento melhor que o RM correspondente
- O teorema de Cevin, 2002 estabelece que quando o fator de utilização total U > 1, então o período médio de cada tarefa i é dado por $\overline{T}_i = T_i U$
- Isto significa que todas as tarefas são escalonadas, exibindo um comportamento médio mais lento, como se o período sofresse uma dilatação

19

Robustez em condições de sobrecarga



- A tarefa T3 agora volta a ser executada. T1 por sua vez perde seu *deadline* e fica sem executar no terceiro período.
- T1 executou 5 vezes em 6 períodos o que dá um período médio de 480/5=96.
- Período médio calculado pelo teorema = 1,25*80=100

20

Tempo de latência de entrada-saída

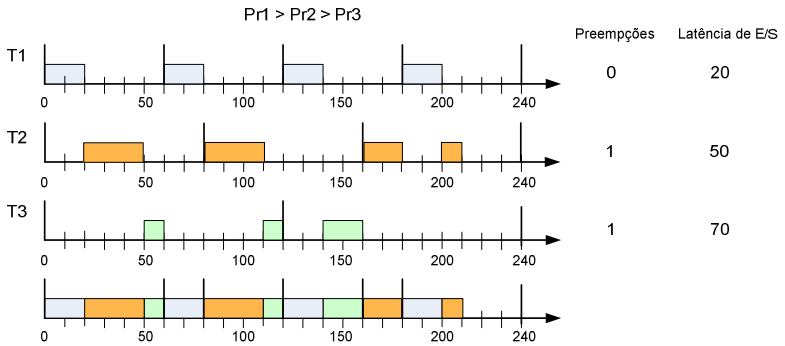
- Teorema de Cervin [2003]
  Dados um conjunto de tarefas periódicas realizando entrada no início de sua execução e saída ao final da execução, o tempo máximo de latência de cada tarefa sob EDF é menor ou igual ao tempo de latência máximo sob RM

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	60	20	33.33
T2	80	30	37,50
T3	120	20	16,17

21

Resultados com RM

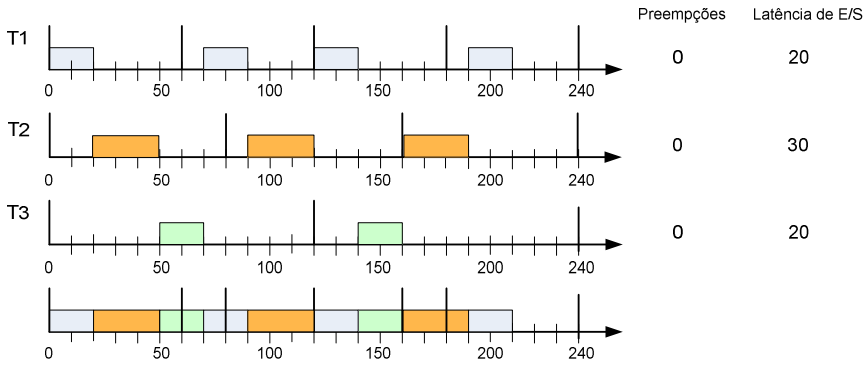
Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	60	20	33.33
T2	80	30	37,50
T3	120	20	16,17



22

Resultados com RM

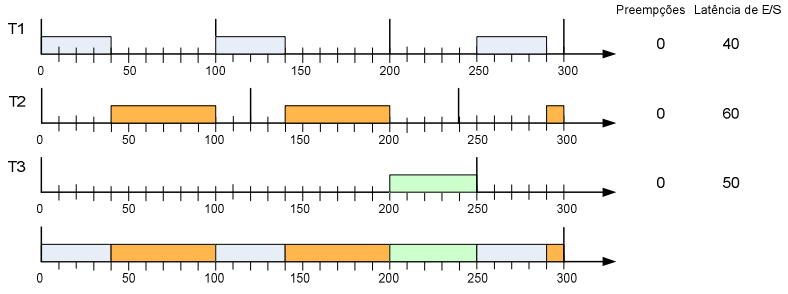
Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	60	20	33.33
T2	80	30	37,50
T3	120	20	16,17



23

EDF – Task Set não factível

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	40	40
T2	120	60	50
T3	250	50	20

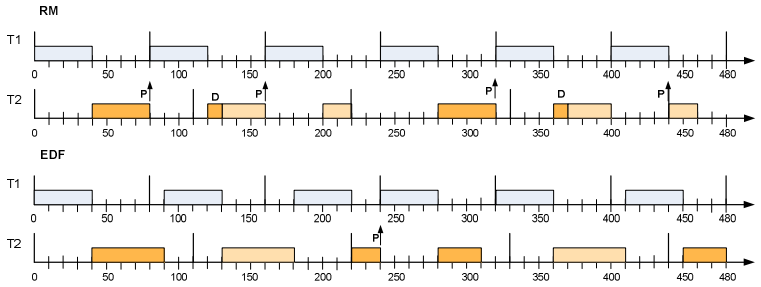


- Observe que em t=200, T3 é quem tem o menor *deadline* e é escalonado
- T1 apenas sofre um adiamento no seu início e nunca é preemptado

24

Task set factível sob EDF mas não factível sob RM

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	80	40	50
T2	110	50	45



- Ocorrem 4 preempções sob RM e apenas uma sob EDF. A preempção no algoritmo EDF ocorre apenas quando T2 (tarefa de maior período) está executando e T1 apresenta um *deadline* menor
- Sob RM T1 deixa de cumprir seu *deadline* duas vezes no trecho de programa examinado

25

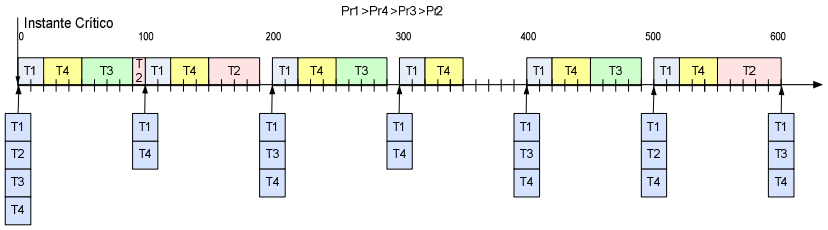
Deadline Monotônico ecarga com EDF

- As condições para utilizar este algoritmo são:
  - As tarefas são periódicas e independentes
  - O tempo de computação de todas as tarefas é conhecido e não varia entre execuções da tarefa. Na verdade tomamos a situação de pior caso como o tempo padrão
  - O tempo de troca de contexto é nulo
  - O *deadline* é igual ou menor que o período de cada tarefa *(Ci <= Di <= Pi)*
- O algoritmo possui atribuição de prioridades estática. A prioridade será escolhida de forma a que a tarefa de menor data limite tenha a maior prioridade

26

Deadline Monotônico
Deadline Monotonic - DM

Tarefa	P_i (ms)	D_i (ms)	C_i (ms)	Prioridade	Utilização (%)
T1	100	80	20	0	20
T2	500	450	50	3	10
T3	200	180	40	2	20
T4	100	100	30	1	30



- P1>P4>P3>P4

27

Tarefas esporádicas

- Tarefa periódica cujo período é igual à Min, o menor intervalo entre solicitações sucessivas da tarefa esporádica
- Os testes de exeqüibilidade são os mesmos de um algoritmo DM onde o valor do *deadline* é escolhido tal que Ci < Di < Min

28

Teste de escalonabilidade necessários e suficientes

- É dado um conjunto de n tarefas periódicas T1..Tn, com cargas computacionais C1 a Cn e períodos iguais a P1..Pn , com P1 ≤ P2 ≤ ... ≤ Pn. Ao invés de chegar todas no instante crítico, assume-se que cada tarefa chega numa fase Ii em relação a zero. Logo a tarefa T2 irá chegar nos instantes: I2 + kP2, k ≥ 0. A tarefa Ti receberá prioridade i onde um menor valor de i indica uma maior prioridade
- O pior caso acontece quando Ii=0, ou seja todas as tarefas chegam no instante 0
- A demanda cumulativa do processador feitas pelas tarefas de prioridade maior ou igual a i disponíveis para execução no intervalo [0, t] é dada por:

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil . C_j$$

29

Teste de escalonabilidade necessários e suficientes

- Máximo número de ocorrências da tarefa Tj no intervalo [0, t]

$$\left\lceil \frac{t}{P_j} \right\rceil$$

- Necessidade de processador da tarefa Tj no intervalo [0, t]

$$\left\lceil \frac{t}{P_j} \right\rceil . C_j$$

- Ui(t) é a percentagem de utilização do processador no intervalo [0, t] considerando apenas as tarefas de prioridade maior ou igual a Pi

$$U_i(t) = \frac{W_i(t)}{t} \qquad U_i = \min_{0 \le t \le T_i} U_i(t) \qquad U = \max_{1 \le i \le n} U_i$$

30

Teste de escalonabilidade necessários e suficientes

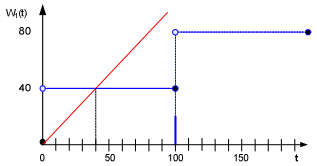
- Wi(t) é constante durante o intervalo entre ativações., o que faz de Ui(t) uma função monotônica decrescente nestes intervalos
- Uma tarefa Ti só conseguirá completar durante sua primeira ativação (0, Pi) se e somente se todas as demais tarefas de prioridade maior (período menor) completarem e se Ci, o requerimento de tempo de Ti, tiverem sido completados
- **Teorema:**
  A tarefa Ti será escalonável para qualquer fase I, usando o algoritmo RM, se e somente se Ui ≤ 1. Todo o *task set* será escalonável sob RM se e somente se U ≤ 1.

31

Exemplo

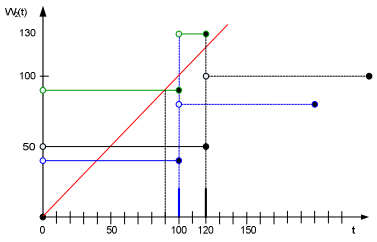
Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	40	40
T2	120	50	42

$$W_1(t) = \left\lceil \frac{t}{100} \right\rceil * 40$$



- W1(0) = 0 W1(20) = 40 W1(120)= 80

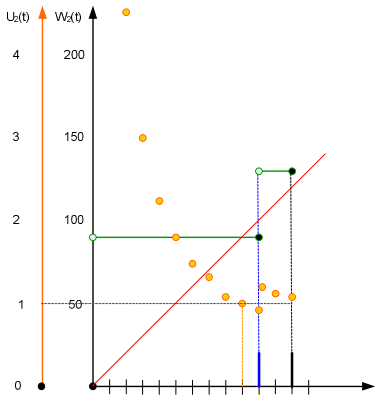
$$W_2(t) = \left\lceil \frac{t}{100} \right\rceil * 40 + \left\lceil \frac{t}{120} \right\rceil * 50$$



- W2(0) = 0 W2(20) = 90 W2(110) = 130

32

Exemplo



- Vemos que a tarefa 1 é factível pois para t= 40, W1(t) = t
- A tarefa T2 também é factível pois para t=90, W2(t) = t
- Logo, todo o *task set* é escalonável

33

Melhorando o Teorema

- Se uma determinada tarefa for escalonável no seu primeiro período ela também o será nos períodos subsequentes, logo temos que examinar cada tarefa apenas no intervalo (0..Pi]. Na verdade a função Wi(t) só faz sentido para t ∈ (0, Pj) onde Tj é a tarefa de maior período
- Dado o período Pi da tarefa sendo examinada, seja Si o conjunto de todos os pontos de escalonamento (troca de contexto) dentro do período Pi

$$S_i = \{kP_j \mid j = 1..i,\ k = 1..\left\lfloor \frac{P_i}{P_j} \right\rfloor\}$$

- No exemplo dado temos:
  S1={100} e S2={100, 120}
- **Teorema revisado:**
  A tarefa Ti será escalonável para qualquer fase I, usando o algoritmo RM, se e somente se Ui = min {t ∈ Si } Ui(t) ≤ 1. Todo o *task set* será escalonável sob RM, se e somente se U = max {1 ≤ i ≤ n} Ui(t) ≤ 1

34

Exemplo

Tarefa	$P_i = D_i$ (ms)	C_i (ms)	Utilização (%)
T1	100	40	40
T2	150	40	26.7
T3	350	100	28.6

Para T1:

$$W_1(t) = \left\lceil \frac{t}{100} \right\rceil * 40 \quad S_1 = \{100\}$$

$U_1(100) = 40 / 100 = 0.40 < 1$ Logo T1 é escalonável.

$$W_2(t) = \left\lceil \frac{t}{100} \right\rceil * 40 + \left\lceil \frac{t}{150} \right\rceil * 40 \quad S_2 = \{100, 150\}$$

$U_2(100) = (40 + 40) / 100 = 0.80$, logo T2 é escalonável.

$$W_3(t) = \left\lceil \frac{t}{100} \right\rceil * 40 + \left\lceil \frac{t}{150} \right\rceil * 40 + \left\lceil \frac{t}{350} \right\rceil * 100 \quad S_3 = \{100, 200, 300, 150, 350\}$$

$U_3(100) = (40 + 40 + 100)/100 = 1.8 > 1$

$U_3(150) = (80 + 80 + 100) / 100 = 2.6 > 1$

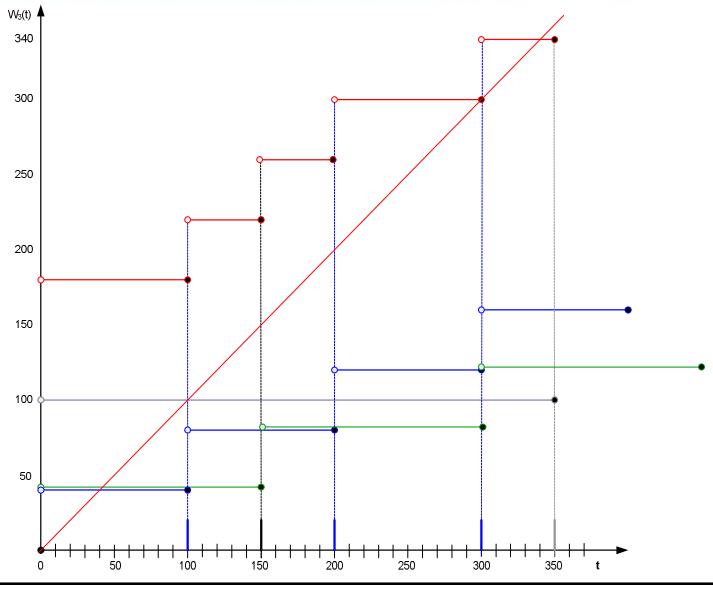
$U_3(200) = (80 + 80 + 100) / 200 = 1.3 > 1$

$U3(300) = (120 + 80 + 100) / 300 = 1$

Logo T3 é escalonável e todo o *task set* é escalonável.

35

Exemplo



36

UFMG

Perguntas?

Constantino Seixas Filho

constantino.seixas@task.com.br

37