Especificação, Modelação e Projecto de Sistemas Embutidos

Conceitos básicos de Tempo-Real

Paulo Pedreiras, Luís Almeida

{pbrp,lda}@ua.pt

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Parcialmente baseado nos materiais pedagógicos da disciplina de "Sistemas Tempo-Real", Luís Almeida, DETI, Universidade de Aveiro

V1.0 Novembro/2008

Definição

- **Computação** de Tempo-Real:
  - Os resultados das computações devem ser:
    - **Logicamente correctos** e **Produzidos a tempo**

Pontualidade

Correcção lógica
(Stankovic, 1988)

- **Funcionalidade ou serviço** de Tempo-Real
  - **Funcionalidade ou serviço** que tem de ser desempenhada ou prestado dentro de **intervalos de tempo finitos** impostos por um **processo físico**

- **Sistema** de Tempo-Real
  - Aquele que desempenha **pelo menos uma funcionalidade** de tempo-real ou que presta **pelo menos um serviço de tempo-real** (PDC, 1990)

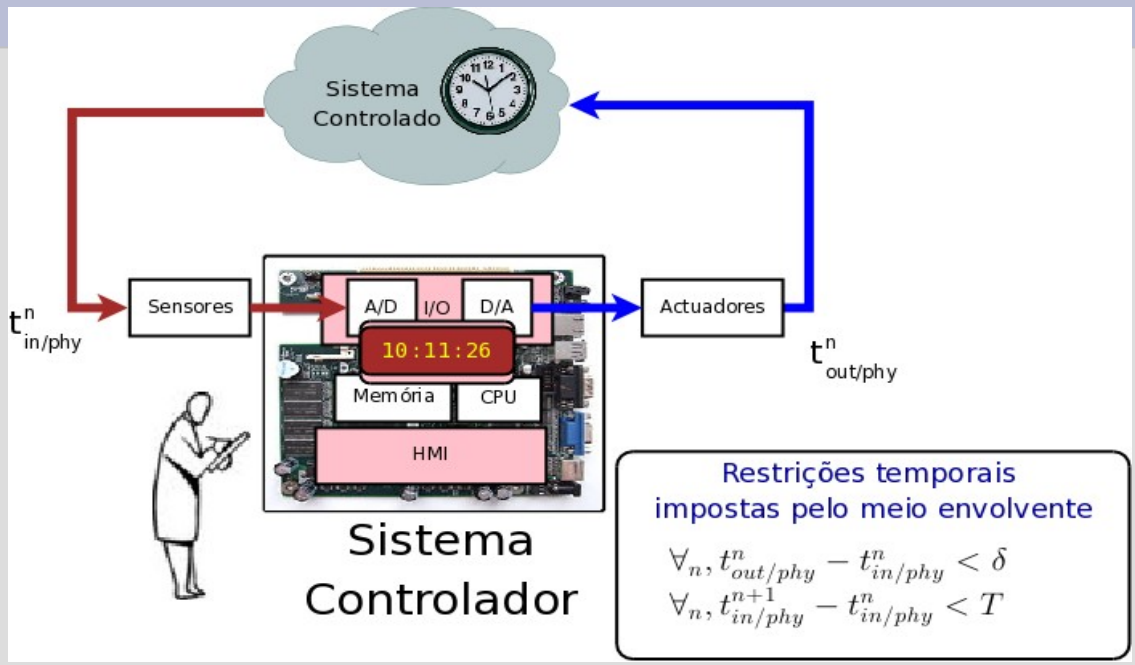
Requisitos temporais

Origem dos requisitos temporais:

- Normalmente advêm da **dinâmica** do **processo físico** que se pretende controlar
- Impõem **restrições** aos **instantes** em que as acções desempenhada num sistema são **executadas**.
  - No sistema de travagem de um automóvel não é suficiente dizer que após carregar-se no respectivo pedal se desencadeia a travagem! É necessário **garantir** que tal acontece no **tempo correcto** (e.g. ordem não pode demora mais de 50ms a ser executada).
- Estas restrições têm de ser cumpridas em **todas as instâncias** (incluindo o pior caso) e não apenas em termos médios

Exemplo de ES com requisitos temporais

- Exemplo de ES: controlo de um processo.



Algoritmo de controlo

Configurar um Timer para gerar uma **interrupção** em **intervalos** de **T**;
Em **cada interrupção** do Timer
do
    Conversão A/D dos **parâmetros** relevantes do sistema
        e.g. temperatura, pressão, humidade, ...
    Cálculo do sinal de controlo
    Escrita do sinal de controlo nos **actuadores** (e.g. válvula a 30%)
end do

Exemplo de ES com requisitos temporais

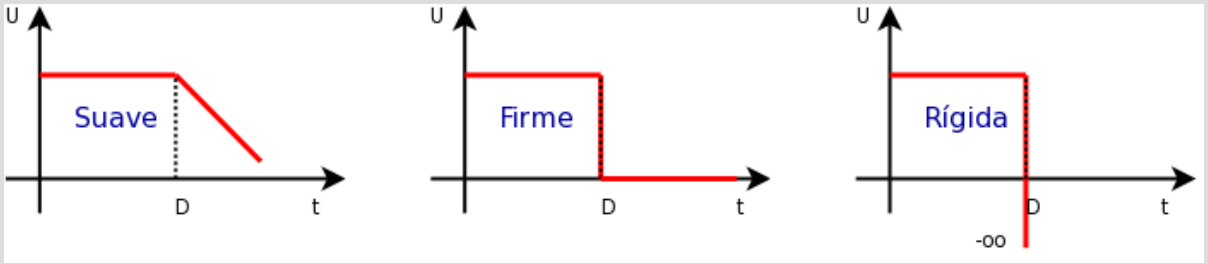
- Há muitos outros SE que possuem requisitos de tempo-real.
- E.g. **Sistemas multimédia**
  - A *Set-top box* recebe, para **cada *frame*** (imagem), vídeo e áudio comprimido em f**ormato digital** (e.g. MPEG-2)
  - A **taxa** de recepção de *frames* é e.g. **25 frames/sec**
    - i.e. são mostradas 25 imagens distintas em cada segundo
  - Cada *frame* tem de ser **processada** em 1/25=40ms
    - Em paralelo pode ter de ser efectuada desencriptação, recuperação de erros, ...
    - Quando tal não acontece pode acontecer uma degradação da qualidade

Muitas outras classes de SE possuem requisitos de tempo-real: sist. transporte, controlo de tráfego, equip. médico, equip./sistemas militares, automação industrial, equip. telecomunicações, ...

Classificação das restrições temporais

Classificação das restrições temporais

- De acordo com a utilidade do resultado para a aplicação podem classificar-se em:
  - ***Suave (Soft)*** **-** Restrição temporal em que o resultado que a ela está associado mantém **alguma utilidade** para a aplicação mesmo depois de um limite D embora haja uma degradação da qualidade de serviço.
  - ***Firme (Firm)*** **-** Restrição temporal em que o resultado que a ela está associado **perde qualquer utilidade** para a aplicação depois de um limite D.
  - ***Rígida (Hard)*** **-** Restrição temporal que, quando não cumprida, pode originar uma **falha catastrófica.**



Classificação dos sistemas tempo-real

Classificação do Sistemas de Tempo-Real:

- De acordo com o tipo das restrições temporais os STR podem classificar-se em:
  - **Soft Real-Time**
    - O sistema apenas apresenta restrições temporais do tipo ***firm*** **ou** ***soft*** (e.g., simuladores, sistemas multimédia)
  - **Hard Real-Time**
    - O sistema apresenta **pelo menos uma** restrição temporal do tipo ***hard***. São sistemas de segurança crítica (e.g. controlo de voo de aviões, de mísseis, de centrais nucleares, de fábricas de produtos perigosos)

Classificação das tarefas

Os ES são habitualmente estruturados como um **conjunto de tarefas concorrentes** que executam funções específicas.

Tipos de tarefas:

- **Periódicas**
  - *Time-driven*, activadas regularmente em intervalos fixos
  - Tipicamente especificadas por $\{C_i , T_i , D_i\}$
    - C_i = tempo de execução de pior caso
    - T_i = período da tarefa
    - D_i = *deadline* (relativa) da tarefa

Exemplo: amostragem periódica de um sensor

Classificação das tarefas

- **Tarefas esporádicas**
  - *Event-driven*, **activadas** por uma **entidade externa** ou **alteração** no ambiente
  - Tipicamente especificadas por $\{C_i, mit_i, D_i\}$
    - $\{C_i, D_i\}$ significado idêntico ao anterior
    - mit_i representa o tempo mínimo entre activações

      Exemplo: detecção de passagem de um objectos numa linha de montagem

- **Tarefas aperiódicas**
  - *Event-driven*, activadas por uma entidade externa ou alteração no ambiente
  - Chegada de **eventos** com **propriedades desconhecidas** (múltiplos eventos, eventualmente simultâneos ou muito próximos)

Normalmente não é possível garantir o cumprimento de restrições temporais deste tipo de tarefas! Porquê?

Conceitos básicos de escalonamento

O **tempo** que uma tarefa demora a **terminar** depende:

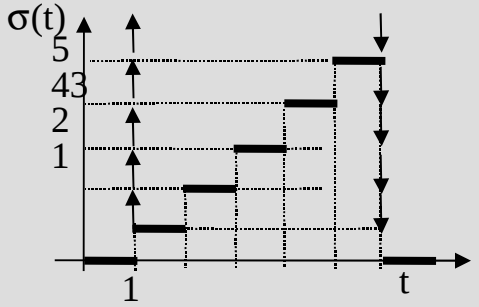
- **Tempo de execução** (próprio)
  - Tempo de execução das instruções que compõem a tarefa
  - Depende da complexidade da tarefa
  - É em geral difícil de estimar, sendo influenciado por factores como:
    - Estrutura do código (linguagem, condicionais, ciclos)
    - DMA, caches, pipeline
    - Sistema operativo ou *kernel* (*system calls*)
- **Interferência**
  - Tempo de espera motivado pela execução de tarefas com maior prioridade – **Depende do algoritmo de escalonamento**
- **Bloqueio**
  - Execução de tarefas de menor prioridade
  - E.g. devido a acesso a recursos partilhados (*buses*, discos, portos de comunicação,...)

Noção de escalonamento

Noção de **escalonamento** de tarefas:

- **Dados**:
  - um conjunto de tarefas
  - restrições que lhe estão associadas (ou função de custo)
- Encontrar uma **atribuição de tempo de processador às tarefas** que lhes permita :
  - executar as **tarefas completamente**
  - **cumprir as suas restrições** (ou minimizar a função de custo)

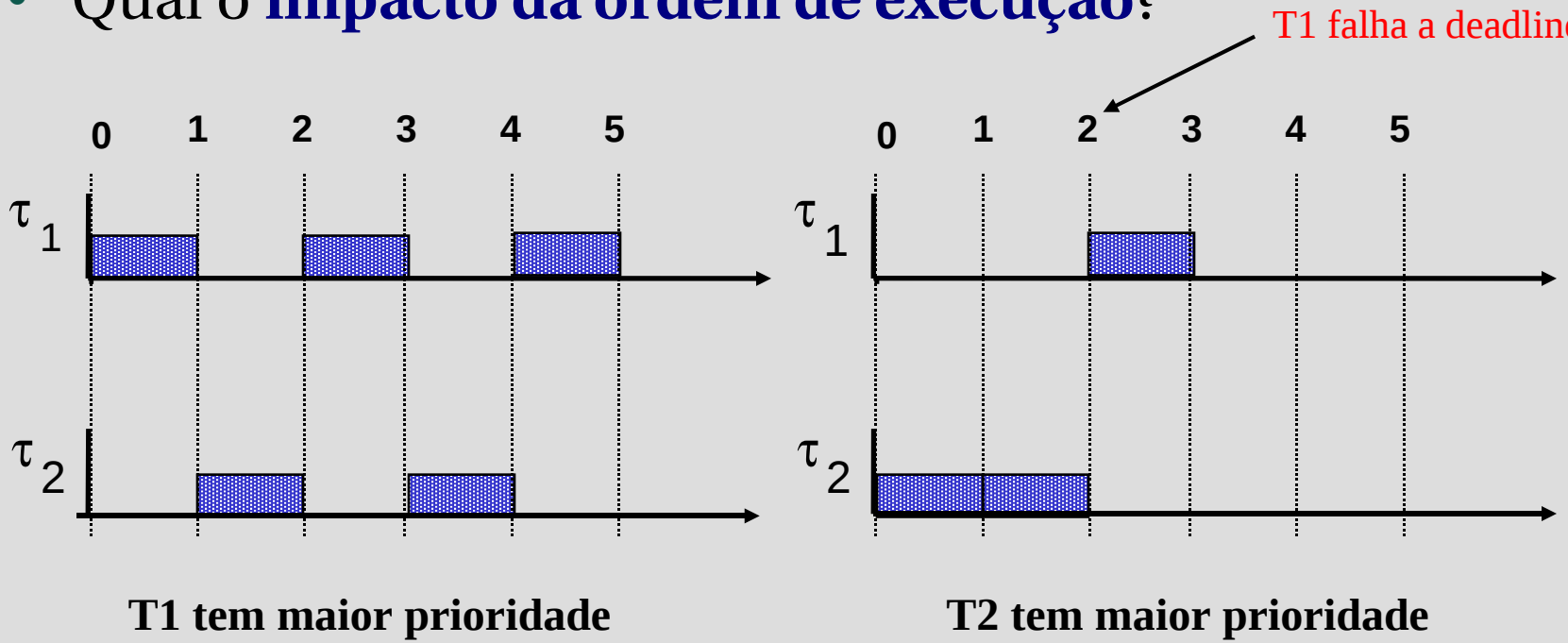
e.g. $J = \{J_i\ (C_i=1, a_i=1, D_i=5, i=1..5)\}$ ->



Noção de escalonamento

Ilustração do **problema** de escalonamento

- Considere as duas tarefas seguintes, **activadas sincronamente** no instante t=0:
  - T1 (1,2,2)
  - T2 (2,5,5)
- Qual o **impacto da ordem de execução**?

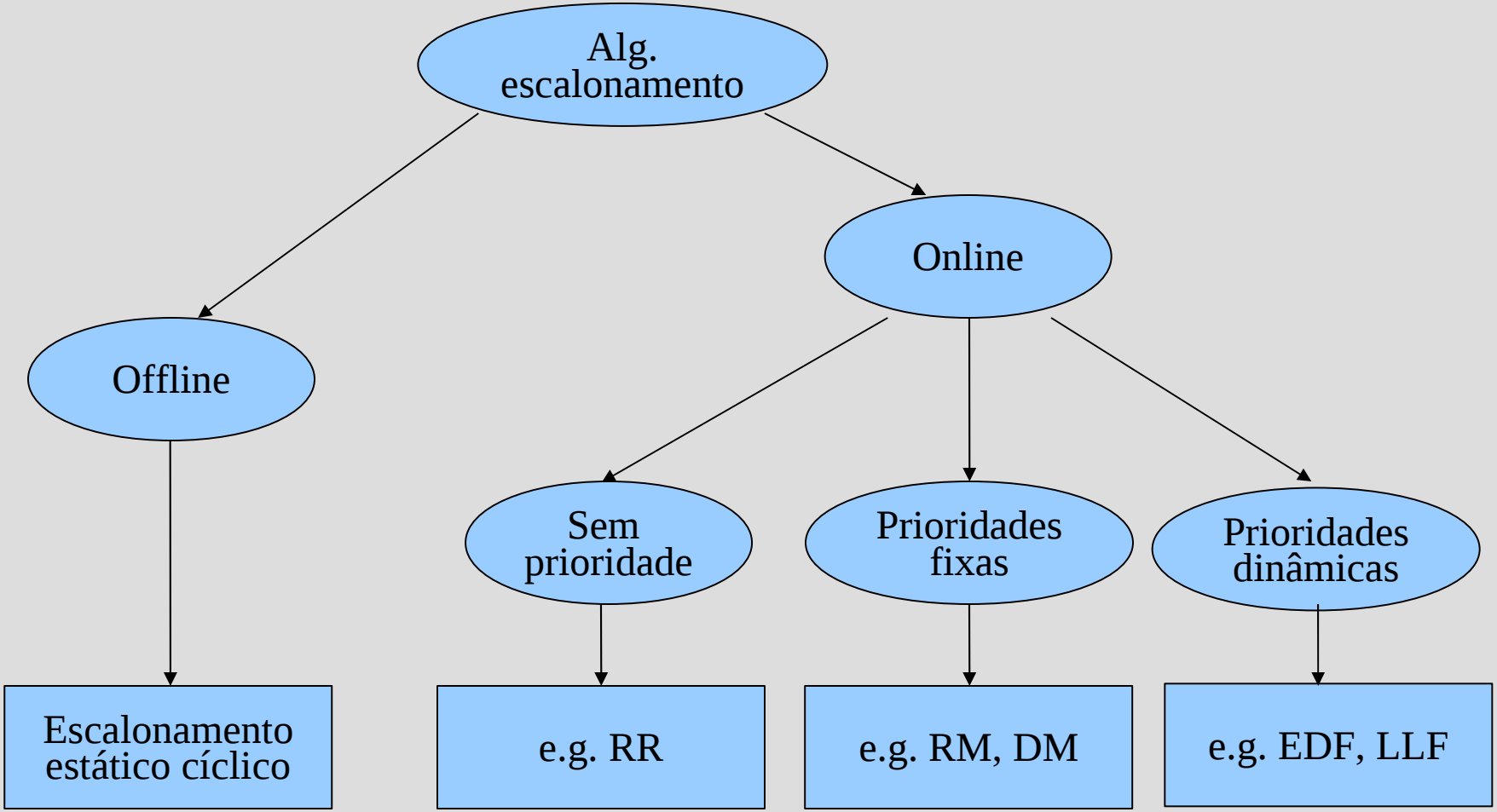


T1 tem maior prioridade

T2 tem maior prioridade

Taxonomia dos algoritmos de escalonamento

Taxonomia dos algoritmos de escalonamento



Algoritmos de escalonamento

Escalonamento **estático cíclico**

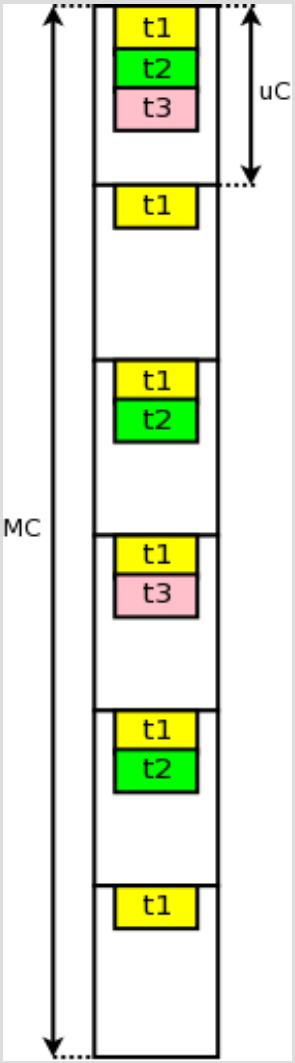
- A tabela é organizada em **micro-ciclos** (uC) de **duração fixa** para que, quando varrida, se obtenha o carácter periódico das tarefas.
- Os micro-ciclos são **disparados** por um ***timer***.
- O varrimento contínuo da tabela resulta num **padrão cíclico global** chamado **macro-ciclo** (MC)

$$\Gamma = \{ \tau_i (C_i, \Phi_i, T_i, D_i, \ i=1..n)\}$$

uC = MDC(T_i) (*GCD*)
MC = MMC(T_i) (*LCM*)

E.g.

$\Phi_i = 0$,
$C_i = 1ms$,
$T_1 = 5ms$
$T_2 = 10ms$
$T_3 = 15ms$



Algoritmos de escalonamento

Algoritmos sem prioridade (exemplos):

- First-In-First-Out (**FIFO**) ou First-Come-First-Served (**FCFS**)
  - Tarefas prontas são inseridas numa **lista**
  - As tarefas **são despachadas** da lista por **ordem de chegada**
  - Não há preempção
  - Não existe o conceito de prioridade

- **Round-Robin** (Preemptivo)
  - As tarefas prontas são despachadas "à vez"
  - Cada tarefa recebe periodicamente um **parte fixa equitativa** (*fair*) do tempo de CPU
  - A tarefa em execução é **suspensa** (*preempted*) no **final** de cada intervalo de execução
  - Não existe o conceito de prioridade

Algoritmos de escalonamento

Algoritmos baseados em **prioridades fixas**

- Cada tarefa recebe um nível de **prioridade específico**
- Algumas tarefas têm **maior importância** que outras
- O nível de prioridade **não** pode ser **alterado** em *run-time*

- **Prioridades arbitrárias**
  - Alocadas pelo projectista do sistema de acordo com um critério arbitrário (e.g. importância)

- **Rate Monotonic**
  - Período mais curto -> maior prioridade
  - Tarefas mais "rápidas" escalonadas em primeiro lugar

- **Deadline Monotonic**
  - *Deadline* (relativa) mais curta -> maior prioridade
  - As tarefas com prazos de execução mais "apertados" são escalonadas em primeiro lugar

Algoritmos de escalonamento

Algoritmos baseados em **prioridades dinâmicas**

- Algumas tarefas têm **maior importância** que outras
- O nível de prioridade **pode** ser **alterado** em *run-time*

- **Earliest Deadline First (EDF)**
  - As tarefas prontas recebem uma prioridade inversamente proporcional à distância à *deadline* absoluta respectiva.
- **Least Slack Time (LST)**
  - Maior prioridade às tarefas com menor tempo livre (*slack*, *laxity*)
- **Shortest Completion Time (SCT)**
  - Tarefas com menor tempo de computação restante são escalonadas em primeiro lugar

Análise de escalonabilidade

Análise de escalonabilidade – prioridades fixas

- Em diversos sistemas é necessário saber à priori se é possível **garantir o cumprimento dos requisitos temporais** de um certo conjunto de tarefas, ou seja, se este é escalonável
- Metodologias para determinação da escalonabilidade
  - **Baseados em utilização**
    - Computacionalmente mais simples (complexidade O(n))
    - Menos rigoroso (e.g. para prioridades fixas é apenas suficiente)
  - **Baseados em tempo de respostas**
    - Computacionalmente mais complexo
    - Condição necessária e suficiente quer para prioridades fixas e dinâmicas (preempção, tarefas independentes e *release* síncrono). Fornece o tempo de resposta de cada tarefa.

Análise de escalonabilidade

Testes para **RM** baseados na **taxa de utilização**

- com preempção, n tarefas independentes e D=T

- **Menor majorante** de Liu&Layland (1973)

$$U(n) = \sum_{i=1}^{n} \frac{C_i}{T_i} \leqslant n(2^{\frac{1}{n}} - 1) \Rightarrow \textit{Uma activação por período garantida}$$

- **Majorante hiperbólico** de Bini&Buttazzo&Buttazzo (2001)

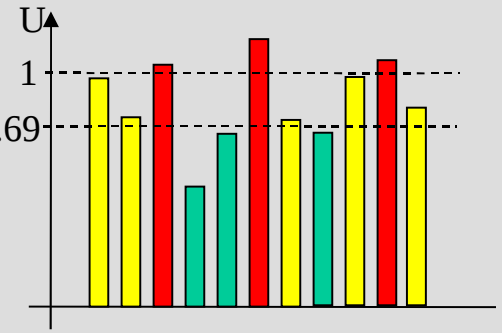
$$\prod_{i=1}^{n} (\frac{C_i}{T_i} + 1) \leqslant 2 \Rightarrow \textit{Uma activação por período garantida}$$

Análise de escalonabilidade

Significado do teste de utilização de Liu&Layland

- **U(n) > 1**
  - conjunto **não escalonável** (*overload*) - **condição necessária**
- **U(n)≤Majorante**
  - conjunto escalonável – **condição suficiente**
- **Majorante ≤U(n)≤1**
  - situação **indeterminada**

U(1) = 1.0
U(2) = 0.828
U(3) = 0.779
U(4) = 0.756
U(5) = 0.743
U(6) = 0.734
U(7) = 0.728
U(8) = 0.724
U(9) = 0.720



À medida que o número de tarefas tende para **infinito** o *bound* tende para **ln(2) = 0.693 (~69%)**

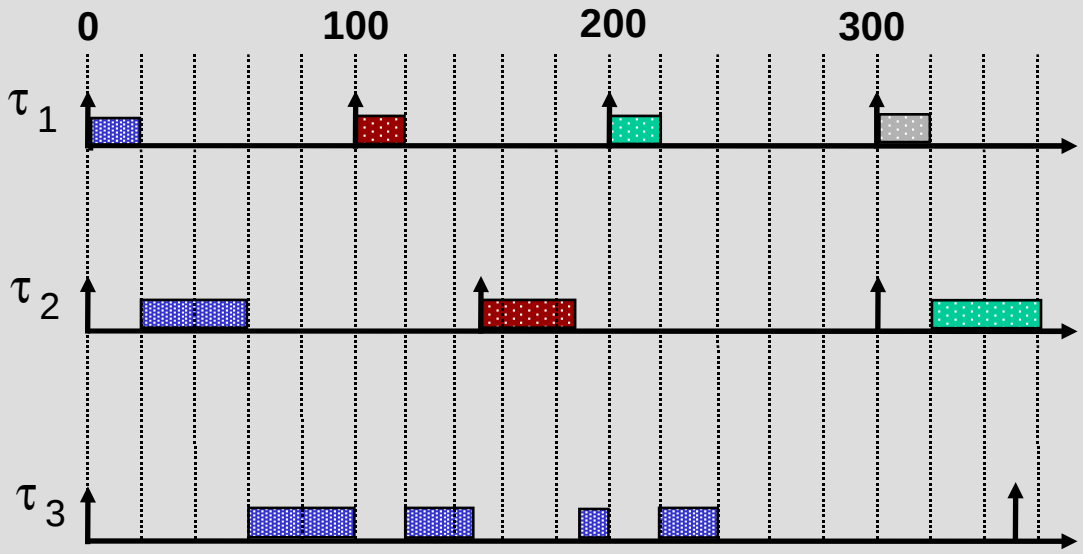
Análise de escalonabilidade

Exemplo de aplicação do teste de Liu&Layland

Task	C	T	U
τ1	20	100	0.200
τ2	40	150	0.267
τ3	100	350	0.286

- *Utilização das tarefas*
  - $U_1 + U_2 + U_3 =$ $.200 + .267 + .286 = .753$
- *Majorante*
  - $U(3) = .779$

– Logo $U_1 + U_2 + U_3 < U(3)$, e o conjunto de tarefas é **escalonável**



Análise de escalonabilidade

Teste de tempo de resposta

- Teorema
  - Para um conjunto de tarefas **independentes** e **periódicas**, se a **primeira instância** de cada tarefa cumpre a sua ***deadline***, com fase de pior caso, então **todas as instâncias** seguintes cumprirão as *deadlines*
- Seja a_n o tempo de resposta da tarefa *i*, onde

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j, com\, a_0 = \sum_{j=1}^{i} C_j$$

  - O teste termina quando há convergência ($a_{n+1} = a_n$) ou a *deadline* é violada ($a_{n+1} > D_i$)
  - O teste tem de ser aplicado a cada tarefa que se queira testar!

Análise de escalonabilidade

Exemplo de aplicação do teste Tempo de Resposta

Task	C	T	U
τ_1	40	100	0.400
τ_2	40	150	0.267
τ_3	100	350	0.286

O teste de utilização é **inconclusivo**!
(verificar)

Teste à tarefa 3

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1 = C_i + \sum_{j=1}^{i-1} \lceil \frac{a_0}{T_j} \rceil C_j = C_3 + \sum_{j=1}^{2} \lceil \frac{a_0}{T_j} \rceil C_j =$$

$$100 + \lceil \frac{180}{100} \rceil 40 + \lceil \frac{180}{150} \rceil 40 = 100 + 80 + 80 = 260$$

$a_1 <> a_0$ e $a_1 < D_3 = 350$

Nova iteração!

Análise de escalonabilidade

(cont.)

$$a_2 = C_3 + \sum_{j=1}^{2} \lceil \frac{a_1}{T_j} \rceil C_j = 100 + \lceil \frac{260}{100} \rceil 40 + \lceil \frac{260}{150} \rceil 40 = 300$$

$$a_3 = C_3 + \sum_{j=1}^{2} \lceil \frac{a_1}{T_j} \rceil C_j = 100 + \lceil \frac{300}{100} \rceil 40 + \lceil \frac{300}{150} \rceil 40 = 300$$

$a_3 = a_2 = 300$ Stop! O teste convergiu!

$a_3 = 300 < D_3 = 350$: tarefa 3 é **escalonável**,

e o seu **tempo de resposta de pior caso** é 350!

Análise de escalonabilidade

Análise de escalonabilidade – prioridades dinâmicas

- Metodologias para determinação da escalonabilidade:
  - **Baseados na taxa de utilização do CPU**
    - Computacionalmente simples (complexidade O(n))
    - Para D<T é suficiente, apenas
  - **Análise de carga imposta ao CPU / Tempo de resposta**
    - Não apresentadas!
      - Sugere-se a consulta e.g. de **Buttazzo, G., "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications",2nd Ed., Springer, 2004**

Análise de escalonabilidade

Testes para EDF baseados na taxa de utilização
(com preempção e n tarefas independentes)

- **D=T**
  - $U(n)=\sum_{i=1}^{n}\frac{C_i}{T_i}\leqslant 1 \Leftrightarrow Conjunto\ escalonável$
    - Condição **necessária e suficiente**. Permite usar 100% do CPU mantendo as garantias temporais
- **D<T**
  - $U'(n)=\sum_{i=1}^{n}\frac{C_i}{D_i}\leqslant 1 \Rightarrow Conjunto\ escalonável$
    - Condição **suficiente**, apenas

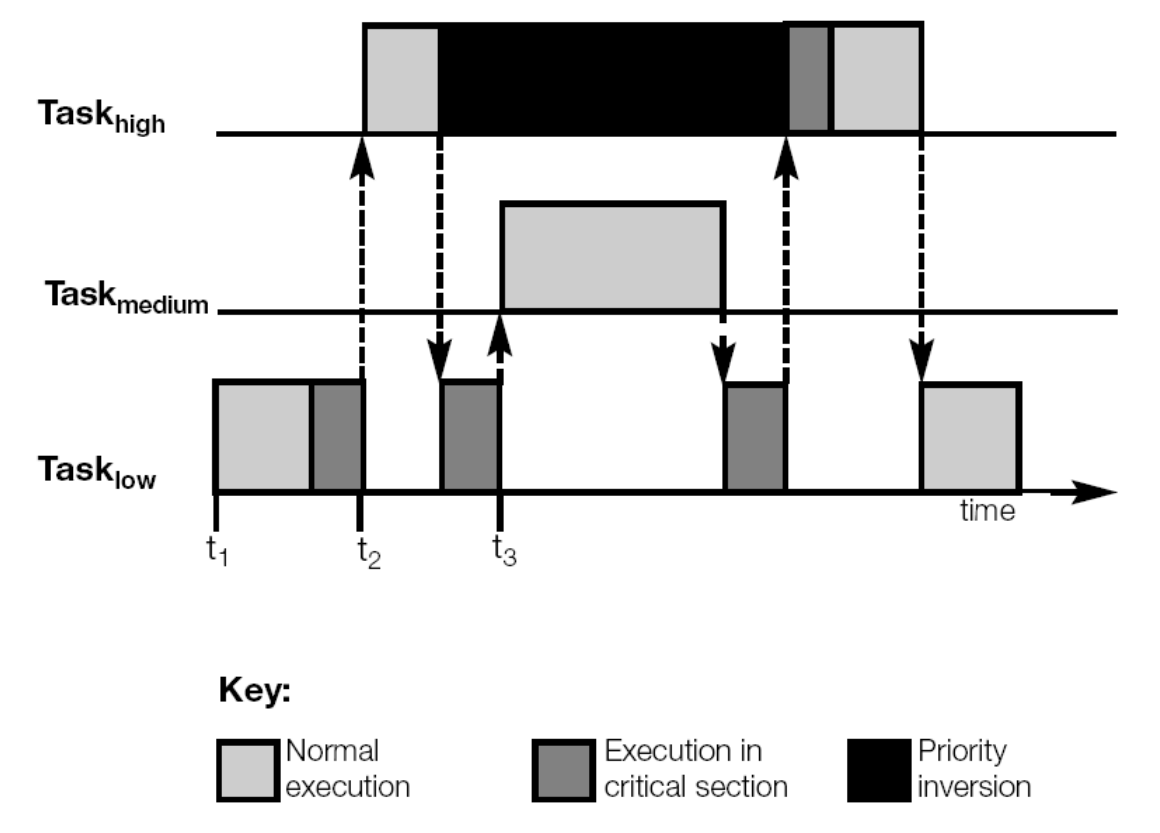
Acesso a recursos partilhados

O Problema da Partilha de recursos

- Nas análises anteriores admitiu-se que as tarefas **não partilhavam** recursos.
- Em muito **sistemas reais** as tarefas **partilham recursos** como estruturas de dados, portos de comunicação, ...
- A partilha de recursos pode causar dois problemas:
  - ***Deadlocks*** (de uma forma semelhante aos sistemas comuns)
  - ***Inversões de prioridade***
- Este fenómenos podem acontecer em sistemas "normais", i.e. não tempo-real. Todavia nos **sistemas tempo-real** podem levar ao **não cumprimento** de **restrições temporais**!!!

Acesso a recursos partilhados

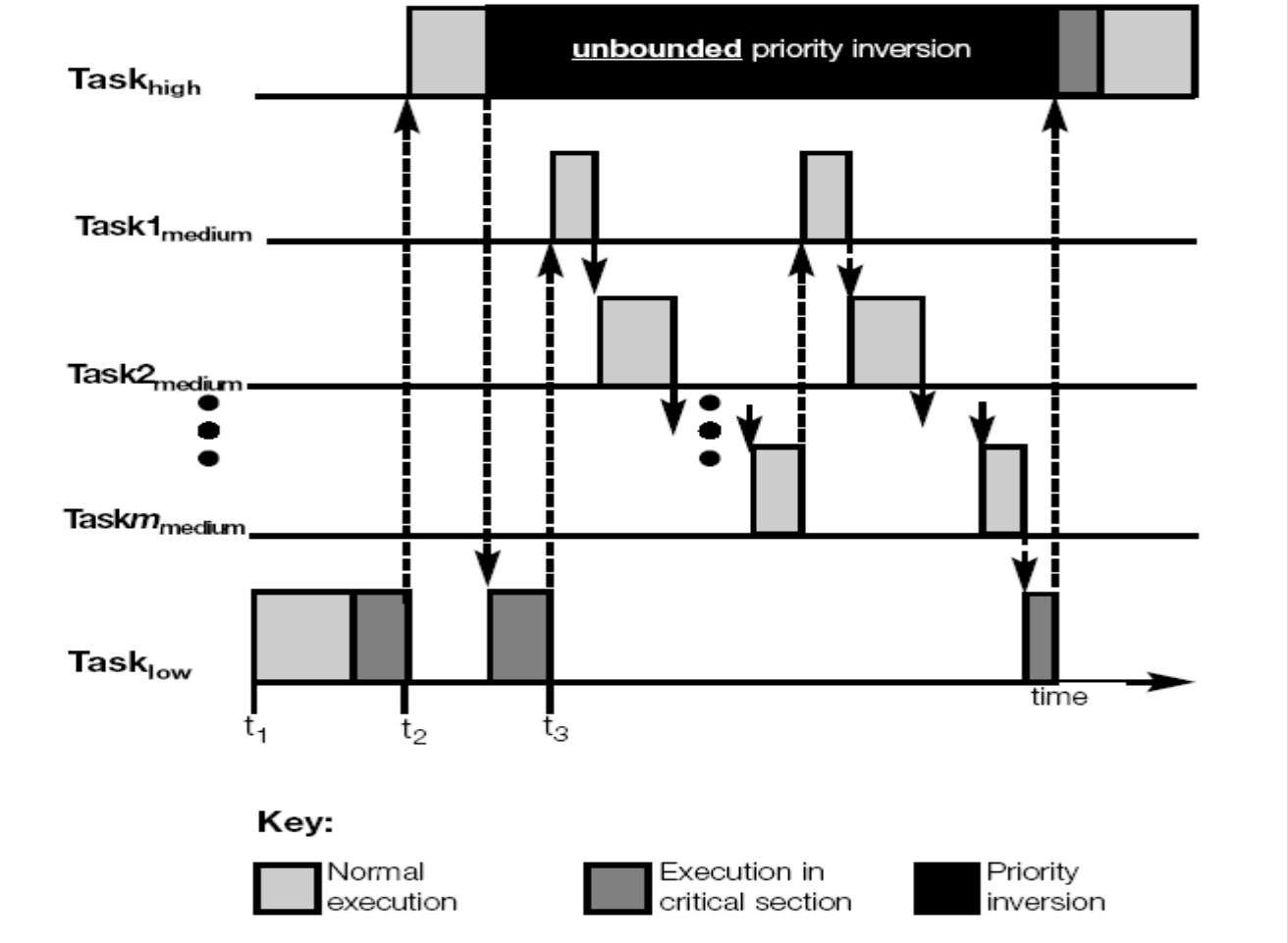
- Inversão de prioridades – exemplo I



Fonte: Handbook of Real-time Systems

Acesso a recursos partilhados

- Inversão de prioridades – exemplo II



Fonte: Handbook of Real-time Systems

Acesso a recursos partilhados

- A **inversão de prioridades** é um **fenómeno inevitável** na presença de **recursos partilhados** de **acesso exclusivo** (intrínseco ao bloqueio)

- Contudo, é fundamental **limitar** e **quantificar** o seu impacto no pior caso, para que se possa raciocinar sobre a **escalonabilidade** do conjunto de tarefas.

- Assim, as técnicas usadas para garantir o acesso exclusivo a cada recurso (primitivas de sincronização) deverão permitir **limitar** a zona de inversão de prioridades e ser **analisáveis**, i.e. permitir quantificar o pior bloqueio que cada tarefa poderá sofrer.

Acesso a recursos partilhados

Técnicas de sincronização

- **Inibição de interrupções** (disable / enable ou cli / sti)
  - **Todas** as tarefas são **bloqueadas**, mesmo as que **não usam recursos partilhados**.
  - Pode **interferir** com actividades do **kernel**, ***device-drivers***, etc.
  - Cada tarefa só **bloqueia uma vez** em cada recurso e pela duração da **maior região critica** das tarefas de menor prioridade;
  - Fácil de implementar
- **Inibição de preempção** (no_preemp / preempt)
  - Propriedades semelhantes à inibição de interrupções, excepto que **apenas afecta outras tarefas** (transparente para *kernels, device-drivers, ...*)

Acesso a recursos partilhados

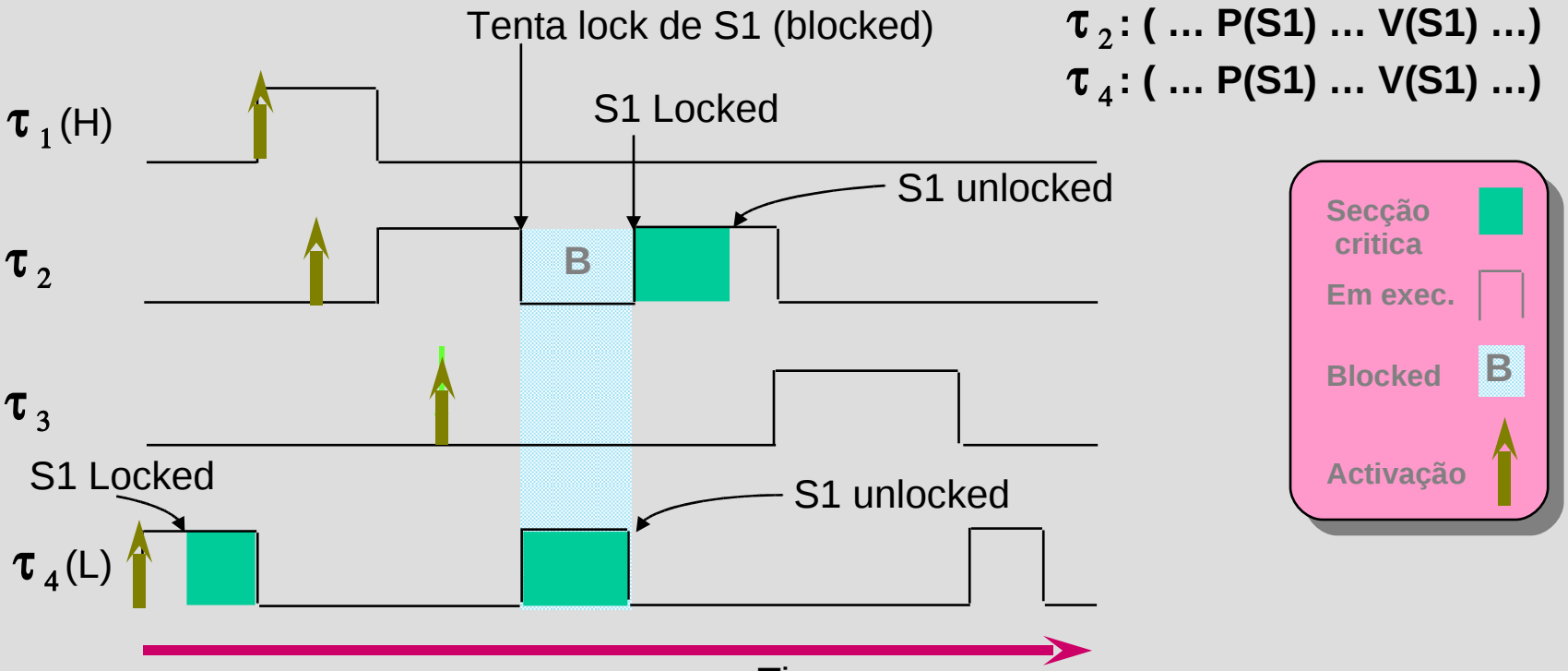
Técnicas de sincronização

- Utilização de ***locks* ou semáforos**
  - Apenas **afectam** as **tarefas envolvidas** na **partilha** de recursos.
  - Implementação **mais custosa** mas mais eficiente (causa bloqueios apenas às tarefas que acedem a recursos)
  - Contudo, a **duração dos bloqueios** é bastante **dependente** do **protocolo específico** utilizado para gerir os semáforos
  - Neste caso é particularmente importante que o referido protocolo permita evitar:
    - Bloqueios indeterminados
    - Bloqueios em cadeia
    - *Deadlocks*

O Protocolo de Herança de Prioridades

Protocolo de Herança de Prioridades (PIP)

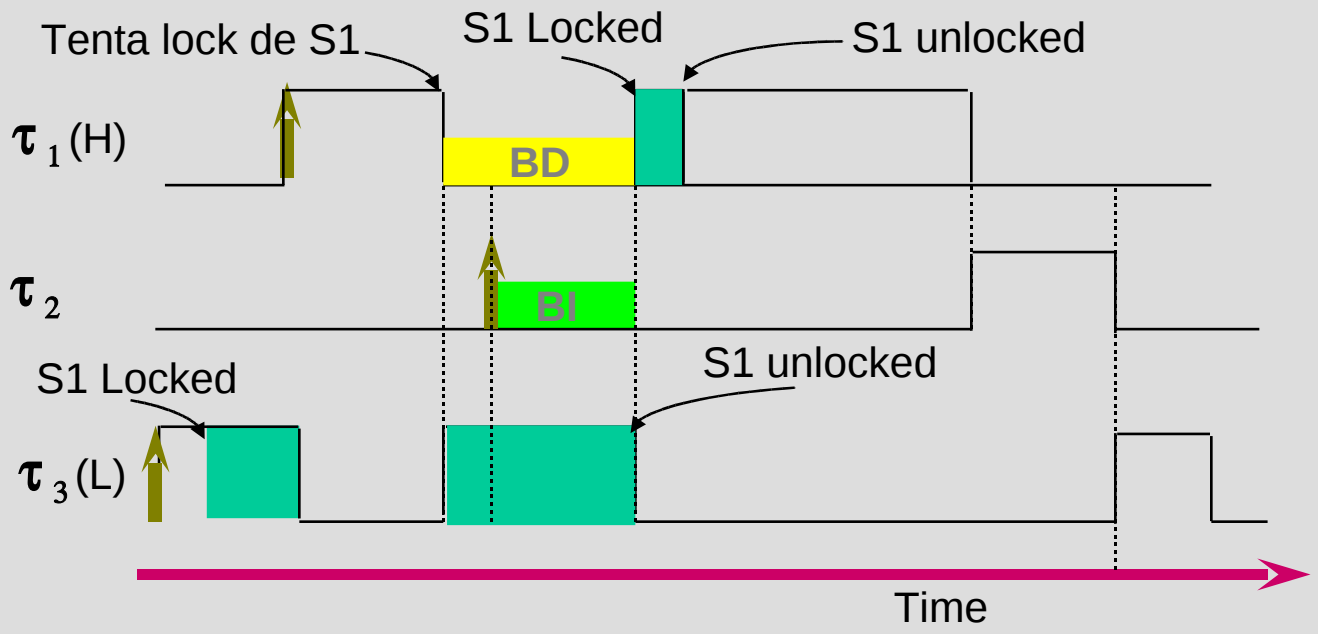
- Uma tarefa executa com a sua própria prioridade até que **bloqueie** uma **tarefa** de **maior prioridade**. Neste caso a prioridade da tarefa bloqueante é **temporariamente elevada** à da tarefa de **maior prioridade** que se encontre **bloqueada** nesse recurso



O Protocolo de Herança de Prioridades

Protocolo de Herança de Prioridades (PIP) (cont)

- Exemplo: Semáforo S1 partilhado entre as tarefas τ_1 e τ_3.



O Protocolo de Herança de Prioridades

Protocolo de Herança de Prioridades (PIP) (cont)

- Para majorar o tempo de bloqueio (B) note-se que uma tarefa pode ser bloqueada por qualquer tarefa de menor prioridade:
  - com a qual **partilhe** um recurso
    - **Bloqueio directo**
  - que possa **bloquear** uma **tarefa** de maior **prioridade**
    - **Bloqueio indirecto**
- Note-se ainda que, na ausência de acessos encadeados:
  - cada **tarefa** só pode bloquear **outra uma vez**
  - cada tarefa só pode ficar bloqueada **uma vez** em **cada semáforo**

O Protocolo de Herança de Prioridades

Protocolo de Herança de Prioridades (PIP) (cont)

- Análise de escalonabilidade
  - As **técnicas estudadas** podem ser **modificadas** por forma **incluir** o factor de **bloqueio**.
    - B_i = máximo bloqueio que qualquer instância da tarefa *i* pode sofrer
    - B_n=0 (τ_n é a tarefa de menor prioridade)

- **Utilização**: Para i=1,2, ... n $\frac{C_1}{T_1}+\frac{C_2}{T_2}+...+\frac{C_i+B_i}{T_i}\leq U(i)$

- **Tempo de resposta**:

$$a_{n+1}=B_i+C_i+\sum_{j=1}^{i-1}\lceil\frac{a_n}{T_j}\rceil C_j, com\, a_0=B_i+\sum_{j=1}^{i}C_j$$

O Protocolo de Herança de Prioridades

Protocolo de Herança de Prioridades (PIP) (cont)

- Relativamente **fácil** de concretizar
  - requer mais um campo no TCB, a prioridade herdada
- É **transparente** para o **programador**
  - cada tarefa só usa informação local
- Todavia, sofre de bloqueio em **cadeia** e, principalmente, **não evita *deadlocks***

- Há outros protocolos que evitam estes problemas, e.g.
  - **Protocolo de Tecto de Prioridades (PCP)**
  - **Política de Pilha de Recursos (SRP)**
    - **Livre de bloqueios em cadeia, *deadlocks***
    - **Complexo**, **não transparente** para o programador

Sumário

- **Definição** de tempo-real
- **Requisitos** temporais e **classificação** dos sistemas
- **Tipos** de tarefas e sua **caracterização**
- **Escalonamento**
  - **Noções** básicas, **Taxonomia**
  - Alguns **algoritmos** de escalonamento
    - Estático cíclico,
    - Sem prioridades (FIFO, RR)
    - Com prioridades fixas (RM, DM)
    - Prioridades dinâmicas (EDF, LST, SCF)
  - **Análise**
    - Utilização e tempo de resposta
- **Acesso a recurso partilhados**
  - Exemplos,
  - Técnicas de sincronização, Protocolo de Herança de Prioridades