FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

T569 –SISTEMAS DE TEMPO REAL

Aula 15

Prof. Marcelo Sousa

Agenda

- Características importantes PCP

- Comparação:
  - PIP - Priority Inheritance Protocol
  - HLP - Highest Locker Protocol
  - PCP - Priority Ceiling Protocol

- Gerenciamento de dependências das tasks

Características Importantes PCP

- **Teorema 2:** *Tasks* são bloqueadas apenas uma vez quando sob PCP.

- **Corolário 1:** Quando sob PCP uma *task* sofre no máximo uma inversão de prioridade durante sua execução.

- Este protocolo evita a ocorrência de *deadlocks*, inversão ilimitada de prioridade e *chain blocking*.

Características Importantes PCP

- Como o *deadlock* é evitado?
  - Deadlocks ocorrem apenas quando *tasks* diferentes travam parte dos recursos necessários ao mesmo tempo e tentam acessar recursos previamente travados.
  - Solução: Quando uma *task* é executada com alguns recursos, qualquer outra *task* não pode travar um recurso que pode ser necessário a essa *task*. Quando uma task solicita um recurso, todos os necessários devem estar livre. Isso previne a possibilidade de *deadlock.*

Características Importantes PCP

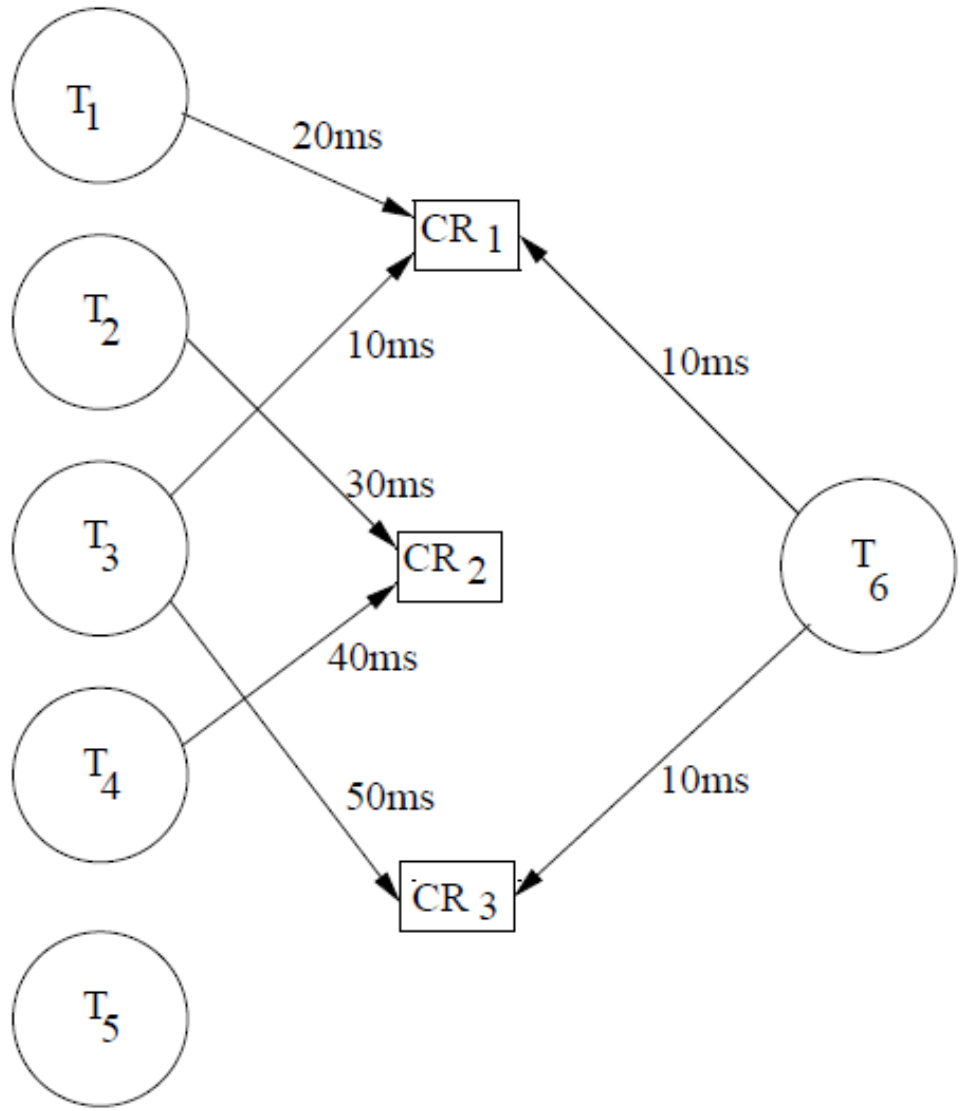
- Como a inversão ilimitada é evitada?
  - Uma *task* sofre deste tipo de inversão quando está aguardando por uma *task* de menor prioridade liberar o recurso necessário para ela e, enquanto isso, outra *task de* prioridade intermediária utiliza o processador.
  - Conclusão: Essa situação não ocorre quando em PCP, já que existe herança de prioridade e a tarefa de menor prioridade receberá a prioridade da *task* de maior prioridade.

Características Importantes PCP

- Como o *chain blocking* é evitado?
  - A partir do teorema 2, as *tasks* sob PCP só podem ser bloqueadas uma única vez.

Analise de Inversões

- Exemplo:
  - Defina os tipos de inversões de prioridades para um sistema que está sendo gerenciado por PCP e que a relação de utilização dos recursos está mostrado na figura ao lado.



Analise de Inversões

Task	Direct					Inheritance					Avoidance				
	T_2	T_3	T_4	T_5	T_6	T_2	T_3	T_4	T_5	T_6	T_2	T_3	T_4	T_5	T_6
T_1	x	10	x	x	10	x	x	x	x	x	x	x	x	x	x
T_2	x	x	40	x	x	x	10	x	x	10	x	10	x	x	10
T_3	x	x	x	x	10	x	x	40	x	10	x	x	40	x	10
T_4	x	x	x	x	x	x	x	x	x	10	x	x	x	x	10
T_5	x	x	x	x	x	x	x	x	x	10	x	x	x	x	x

Tempo de bloqueio

- Tempo de Bloqueio

$$bt_i = max_j^{i-1}\{(b_{idj}), (b_{iij}), (b_{iaj})\}$$

  - bti = tempo de bloqueio da tarefa Ti

  - bidj = tempo de bloqueio por inversão direta
  - biij = tempo de bloqueio por inversão por herança
  - biaj = tempo de bloqueio por *avoidance*

Prioridades dinâmicas

- Todas as considerações realizadas até então levam em consideração que as *tasks* possuem **prioridade estática**. A prioridade da *task* não muda desde sua chegada até sua finalização.
- Em sistemas de **prioridade dinâmica**, a prioridade pode mudar com o tempo.
  - Como consequência, os tetos de todos os recursos devem ser recalculados quando qualquer *task* modificar sua prioridade.
  - O valor do CSC e prioridade por herança também devem ser atualizados.

Comparação

Protocolo	Suporte do SO	Previne	Sofre com
PIP	Baixo	Inversão Ilimitada	Deadlock e Chain Blocking
HLP	Moderado	Chain Blocking e Deadlock	Inversão relacionada a herança
PCP	Alto	Deadlock e Chain Blocking	Pouco com Inversão relacionada a herança e *Avoidance*

Gerenciando de dependências das *tasks*

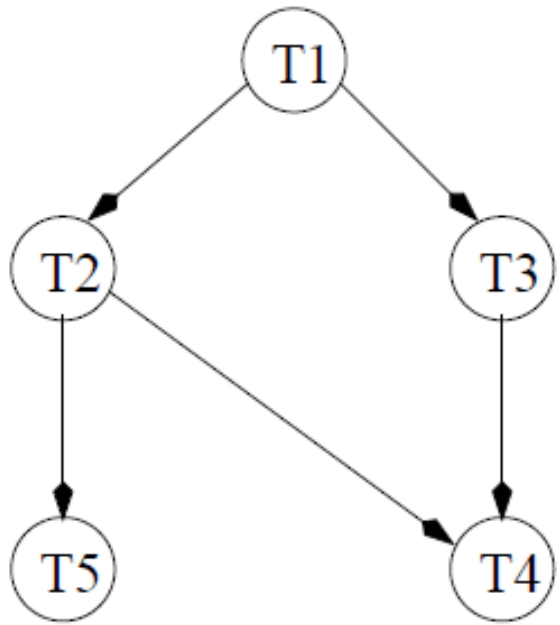
- Os algoritmos de escalonamento de tarefas estudados no capítulo 2 assumem que as tarefas são independentes, mas em sistemas reais isso quase nunca é verdadeiro.
- Em sistemas reais, as *tasks* necessitam de saídas ou devem ser executas em uma ordem predeterminada.
- Os escalonadores estudados no Capítulo 2 são insuficientes caso estas afirmativas acima sejam verdadeiras.

Gerenciando de dependências das *tasks*

- Table-Driven:
  - Passo 1
    - Organize as tarefas em ordem crescente de acordo com seus *deadlines*, sem violar qualquer restrição de precedência e armazene em uma lista ligada (*linked list*).
  - Passo 2
    - Repetir:
      - Pegue a task de maior deadline e que não tanha sido escalonada ainda.
      - Escalone esta tarefa o mais tarde possível.
    - Até que todas as tarefas sejam escalonadas.
  - Passo 3:
    - Mova o agendamento de todas as tarefas para o mais cedo possível sem modificar as posições relativas no escalonador.

Gerenciando de dependências das *tasks*

- Exemplo:
  - Determine um escalonador factível para o conjunto de tasks
    - {T1, T2, T3, T4, T5};
  - Dependência entre *tasks* ao lado;
  - Tempos de execução:
    - {7,10,5,6,2}
  - Deadlines:
    - {40,50,25,20,8}



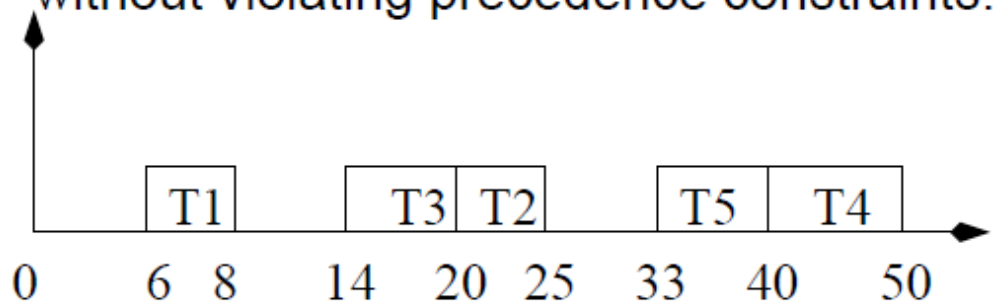
Gerenciando de dependências das *tasks*

- Solução:

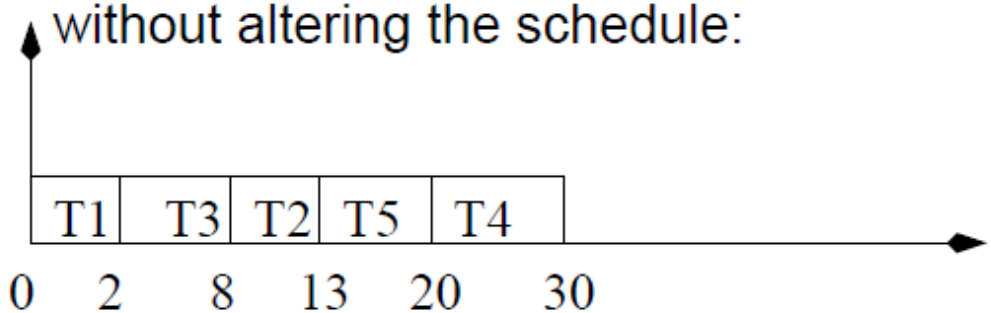
Step1: Arrangement of tasks in ascending order:

T1 T3 T2 T5 T4

Step 2: Schedule tasks as late as possible without violating precedence constraints:



Step 3: Move tasks as early as possible without altering the schedule:



Exercícios

- Próxima Aula:
  - Prática – Gerenciamento de Interrupção no FreeRTOS