IM420 Sistemas Embarcados de Tempo Real

Notas de Aula – Semana 13

Prof. Denis Loubach
dloubach@fem.unicamp.br

Programa de Pós-Graduação em Eng. Mecânica / Área de Mecatrônica
Faculdade de Engenharia Mecânica - FEM
Universidade Estadual de Campinas - UNICAMP

1^o Semestre de 2018

Tópicos

1. Motivação
2. Preempção Segura
3. Starvation
4. Deadlock
5. Inversão de Prioridades
   - Protocolo de Herança de Prioridade
   - Protocolo de Prioridade Topo
   - Protocolo Topo de Prioridade
6. Referências

China's New Supercomputer is World's Most Powerful

For the last three years, China has topped the Top500 list of the most powerful supercomputers with its massive Tianhe-2. But today, the Top500 group announced that Tianhe-2 has been ousted by another Chinese supercomputer, the Sunway TaihuLight. The new machine, which is based at the National Supercomputing Center in Wuxi, can perform a key benchmark test called Linpack at 93 petaflops (a thousand trillion floating point operations per second)—nearly three times the speed of the Tianhe-2

Fonte: IEEE Spectrum, June, 2016
`http://spectrum.ieee.org/tech-talk/computing/hardware/china-beats-worlds-mostpowerful-supercomputer-with-a-new-one`

Preempção Segura

Muitos RTOS embarcados facilitam a programação multi-tarefa

Entretanto, vários desafios de projetos do sistema emergem com a utilização de multi-tarefa

A natureza desse ambiente é de múltiplas tarefas que compartilham e disputam um mesmo conjunto de recursos

Neste caso, o compartilhamento de recursos requer uma coordenação cuidadosa visando assegurar que cada tarefa possa acessar o recurso para continuar sua execução

Preempção Segura (cont...)

Num sistema multi-tarefa preemptivo, o compartilhamento de recursos é **função da prioridade das tarefas**

Ou seja, quanto maior a prioridade da tarefa, mais importante ela é

Tarefas de prioridade maior devem possuir precedência sobre as de menor prioridade quando acessando recursos compartilhados

Portanto, o compartilhamento de recursos não deve violar esta regra

Classificação de Recursos

Exemplos de recursos compartilhados em sistemas embarcados são:

- Dispositivos de I/O
- Registradores da máquina
- Memória

Tais recursos podem ser classificados como **preemptável** ou **não-preemptável**

Recurso Preemptável

Um recurso preemptável pode ser involuntariamente e temporariamente removido de uma tarefa sem afetar seu estado de execução e seu resultado

Os registradores da máquina são um exemplo deste tipo de recurso

Quando o escalonador preempta a tarefa corrente, o conteúdo dos registradores da máquina (incluindo o estado de execução da tarefa) é salvo na memória principal

Estes registradores são re-inicializados para executar outra tarefa

Quanto a outra tarefa termina sua execução, o estado de execução é restaurado para o conjunto de registradores e a tarefa preemptada volta a executar

Recurso Preemptável (cont...)

O escalonador garante que o conjunto de registradores contém o estado de execução de uma única tarefa, mesmo que tais registradores sejam compartilhados entre diversas tarefas ao longo da execução do sistema

Recurso Não-preemptável

Um recurso não-preemptável deve ser voluntariamente cedido pela tarefa que o possui, caso contrário resultados imprevisíveis podem acontecer

Uma memória compartilhada pertence a esta categoria

Por exemplo, uma tarefa não deve ter permissão para escrever numa memória compartilhada antes que outra tarefa complete seu procedimento de leitura ou escrita

Starvation

Por outro lado, se a regra:
Tarefas de prioridade maior devem possuir precedência sobre as de menor prioridade quando acessando recursos compartilhados

Implicar que as tarefas de maior prioridade sempre tomarão os recursos das tarefas de menor prioridade

Este esquema de compartilhamento acabendo não sendo justo e pode evitar que tarefas com prioridade baixa sejam completadas

Esta condição é conhecida como ***starvation***

Starvation (cont...)

A maximização da utilização de recursos é também um requisito conflitante

Neste contexto, dois dos problemas de projeto mais comuns enfrentados são:

- *Deadlock*
- Inversão de Prioridades

Deadlock

O tipo de recurso que uma tarefa possui/requer é importante para decidir quais soluções adotar quando uma tarefa encontra-se numa situação de *deadlock*

Deadlock

É uma situação na qual múltiplas tarefas de um sistema são bloqueadas permanentemente devido a requisitos de recursos que nunca poderão ser satisfeitos

Deadlock (cont...)

Num sistema de tempo real típico existem múltiplos tipos de recursos e múltiplas tarefas concorrentes competindo por tais recursos

Existe um potencial para ocorrência de *deadlocks* num sistema no qual o RTOS permite compartilhamento de recursos entre as várias tarefas

Deadlock (cont...)

Deadlocks ocorrem quando as quatro condições seguintes encontram-se presentes:

1. **Exclusão mútua** – um recurso que só pode ser acessado por apenas uma tarefa por vez, *e.g.,* modo de acesso exclusivo
2. **Não-preempção** – um recurso não-preemptável não pode ser forçadamente removido da tarefa que o possui. Este recurso se torna disponível apenas quando a tarefa que o possui o cede de maneira voluntária
3. ***Hold and wait*** – uma tarefa segura (*hold*) um recurso já adquirido enquanto espera (*wait*) por outro recurso se tornar disponível
4. ***Circular wait*** – uma cadeia circular de duas ou mais tarefas existem, na qual cada tarefa segura um ou mais recursos que são requisitados por uma outra tarefa nesta cadeia

Deadlock (cont...)

Dado que cada recurso é não-preemptável e suporta apenas modo de acesso exclusivo, a figura abaixo ilustra um exemplo de situação de *deadlock* entre duas tarefas

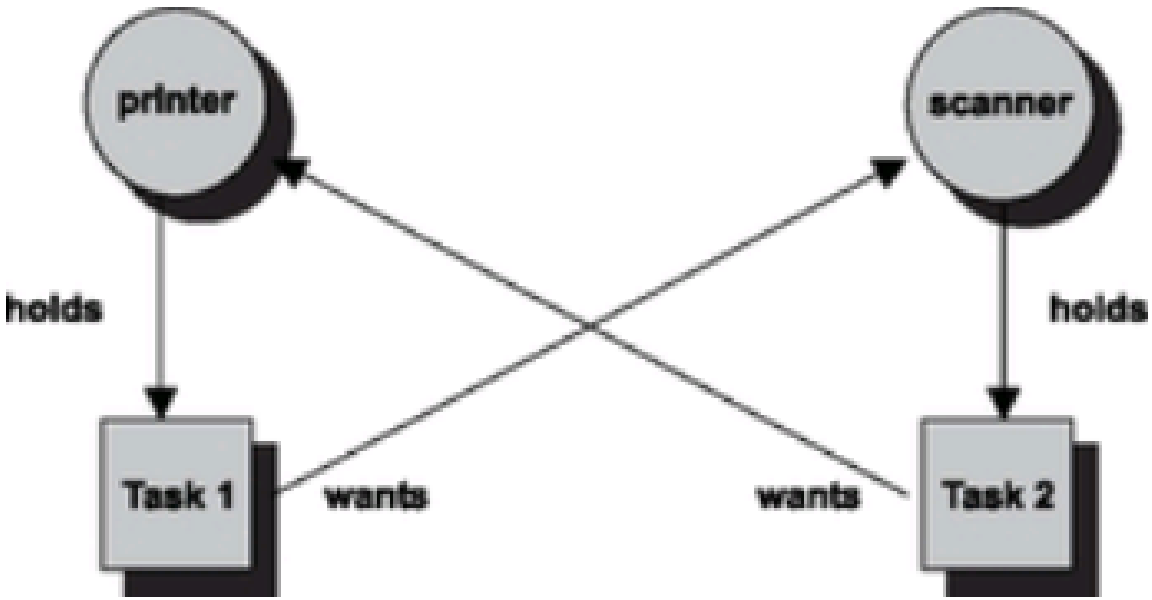

Figura: Grafo de recursos

Deadlock (cont...)

A Figura 1 ilustra um grafo de recursos

As arestas *holds* indicam que a tarefa segura ou possui um recurso

As arestas *wants* indicam que a tarefa precisa daquela recurso para continuar sua execução

Neste exemplo, a tarefa #1 ***wants*** o *scanner* enquanto segura a impressora

A tarefa #1 não pode continuar até que a impressora e o *scanner* estejam sob sua posse

Por outro lado, a tarefa #2 ***wants*** a impressora enquanto segura o *scanner*

A tarefa #2 não pode continuar até que a impressora e o *scanner* estejam sob sua posse

Uma vez que nenhuma das tarefas irá liberar o recurso já conquistado, as duas encontram-se em *deadlock* pois nenhuma pode continuar sua execução

Deadlock (cont...)

Deadlocks podem envolver mais de duas tarefas

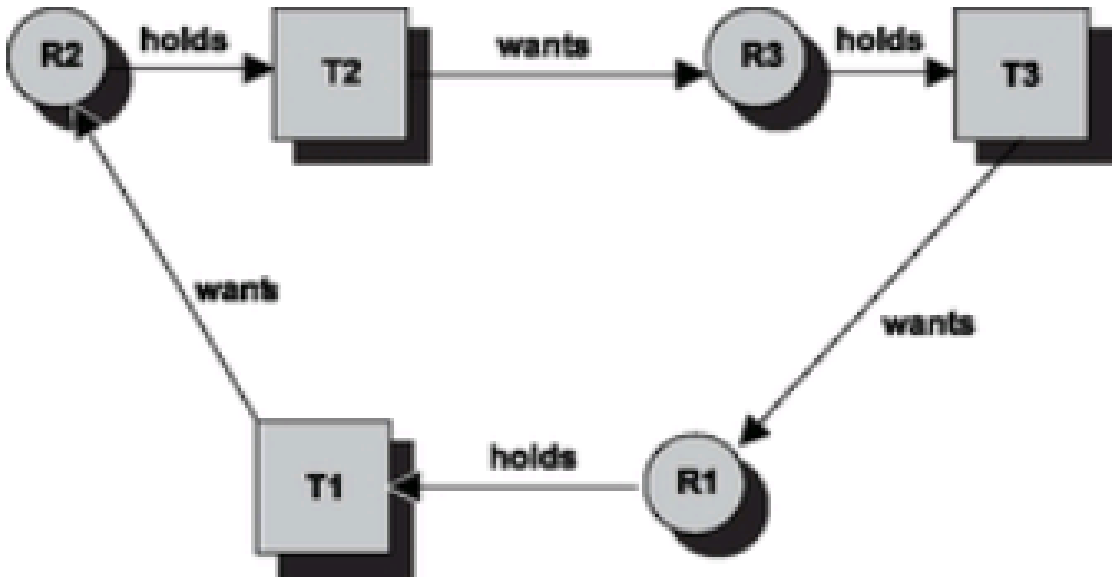

Figura: Deadlock entre 3 tarefas

Deadlock (cont...)

Conforme ilustrado na Figura 2:

- T1 possui o recurso R1
- T1 *wants* R2
- T2 possui o recurso R2
- T2 *wants* R3
- T3 possui o recurso R3
- T3 *wants* R1

Neste caso, torna-se evidente o ciclo, *i.e., circular-wait condition*

As tarefas T1, T2 e T3 e os recursos R1, R2 e R3 formam o conjunto em *deadlock*

Condição de *Deadlock*

Condição de *deadlock*

Uma condição de *deadlock* é denominada **estável** quando nenhuma tarefa no conjunto em *deadlock* espera por um limite de tempo (*timeout*) ou um aborto que possa eliminar tal *deadlock*

Um *deadlock* estável é permanente e requer influência externa para eliminá-lo

Esta influência externa é um algoritmo de detecção e recuperação de *deadlock* do RTOS

Detecção de *Deadlock*

Detecção de *deadlock*

Detecção de *deadlock* é a ativação periódica de um algoritmo pelo RTOS. Esse algoritmo examina a o estado da alocação atual de recursos e pedidos pendentes de recursos para determinar se existe algum *deadlock* no sistema, e em caso positivo, quais tarefas e recursos encontram-se envolvidos

Deadlock temporal

Um *deadlock* temporal consiste numa situação temporária de *deadlock* na qual uma ou mais tarefas no conjunto em *deadlock* ou utiliza um mecanismo de *timeout* ou aborta de maneira anormal devido a restrições temporais

Quando isso ocorre, a tarefa libera o recurso que pode ter causado o *deadlock* primeiramente, assim eliminado o *deadlock*

Recuperação de *Deadlocks*

Para recursos preemptáveis, a preempção de recursos consiste numa maneira de se recuperar de um *deadlock*

Neste caso, o conjunto em *deadlock* é transferido para o algoritmo de recuperação

De maneira bem simplificada, este algoritmo busca liberar recursos de tarefas e ceder para outras visando eliminar a condição de *deadlock*

Recuperação de *Deadlocks* (cont...)

Outra possibilidade seria definir *checkpoints* dentro da tarefa, criando um algoritmo de auto-recuperação (para recursos não-preemptáveis)

Tão logo a tarefa chegue a um *checkpoint* ela altera um estado global para refletir tal transição

Além disso a tarefa precisa definir uma função de entrada específica para ser chamada pelo algoritmo de recuperação de *deadlock* após a tarefa voltar sua execução

Em geral, esse algoritmo envolve voltar ao começo da tarefa e re-iniciar sua execução

Inversão de Prioridades

Inversão de Prioridades

É a situação na qual uma tarefa de baixa-prioridade executa enquanto uma tarefa de prioridade maior espera devido a uma contenção de recursos

Num sistema de tempo real baseado em prioridade e preemptivo, o escalonador preempta tarefas de menor prioridade em favor de tarefas de maior prioridade

Essa é basicamente a norma quando não existe comunicação inter-tarefas (tarefas independentes)

Entretanto, é praticamente inevitável a inter-dependência entre tarefas num sistema com múltiplas tarefas e múltiplos recursos (compartilhamento de recursos e sincronização)

Inversão de Prioridades (cont...)

Neste caso, a inversão de prioridades pode acontecer numa inter-dependência entre tarefas com prioridades diferentes

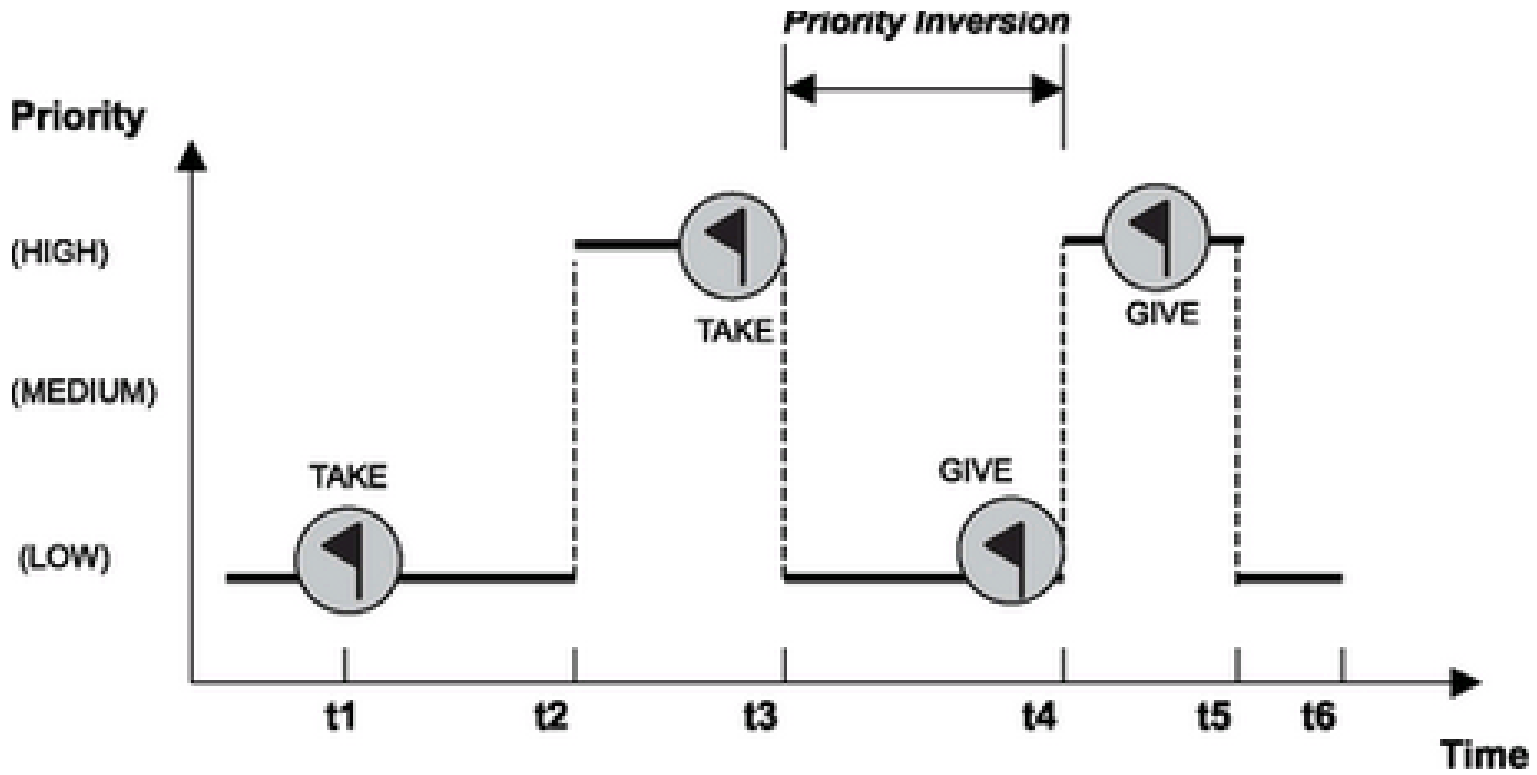

Figura: Exemplo de inversão de prioridades

Inversão de Prioridades (cont...)

A Figura 3 ilustra o caso em que uma tarefa de maior prioridade compartilha um recurso com uma tarefa de menor prioridade

A tarefa de maior prioridade precisa esperar bloqueada porque a tarefa de menor prioridade adquiriu (*locked*) um recurso que a de maior prioridade requer, isso mesmo a de maior prioridade sendo elegível para rodar

Inversão de Prioridades (cont...)

- no tempo `t1` uma tarefa de baixa-prioridade trava um recurso compartilhado (*lock*)
- essa tarefa continua até o tempo `t2` quando uma tarefa de alta-prioridade fica elegível para rodar (preempção e troca de contexto)
- a tarefa de alta-prioridade roda até o tempo `t3` quando requer o recurso e é bloqueada (*block and wait*)
- neste ponto acontece a troca de contexto para a tarefa de baixa-prioridade
- a inversão de prioridade começa em `t3`
- no tempo `t4` a tarefa de baixa-prioridade libera o recurso, que dispara a preempção trazendo a tarefa de alta-prioridade para executar novamente
- a inversão de prioridade vai até o tempo `t4`
- a tarefa de alta-prioridade completa no tempo `t5`, onde acontece a troca de contexto para a tarefa de baixa-prioridade que termina em `t6`

Inversão de Prioridades (cont...)

No exemplo da figura 3, o problema ilustrado de inversão de prioridade é denominado **inversão de prioridade limitado** (*bounded priority invertion*)

A duração do tempo que a tarefa de baixa-prioridade segura o recurso compartilhado é conhecido

Entretanto, é possível que uma tarefa de média-prioridade preempte a de baixa-prioridade por um período de tempo indeterminado

Isso causaria um **tempo de espera indefinido** para a tarefa de alta-prioridade (*unbounded priority invertion*), conforme ilustrado na Figura 4

Inversão de Prioridades (cont...)

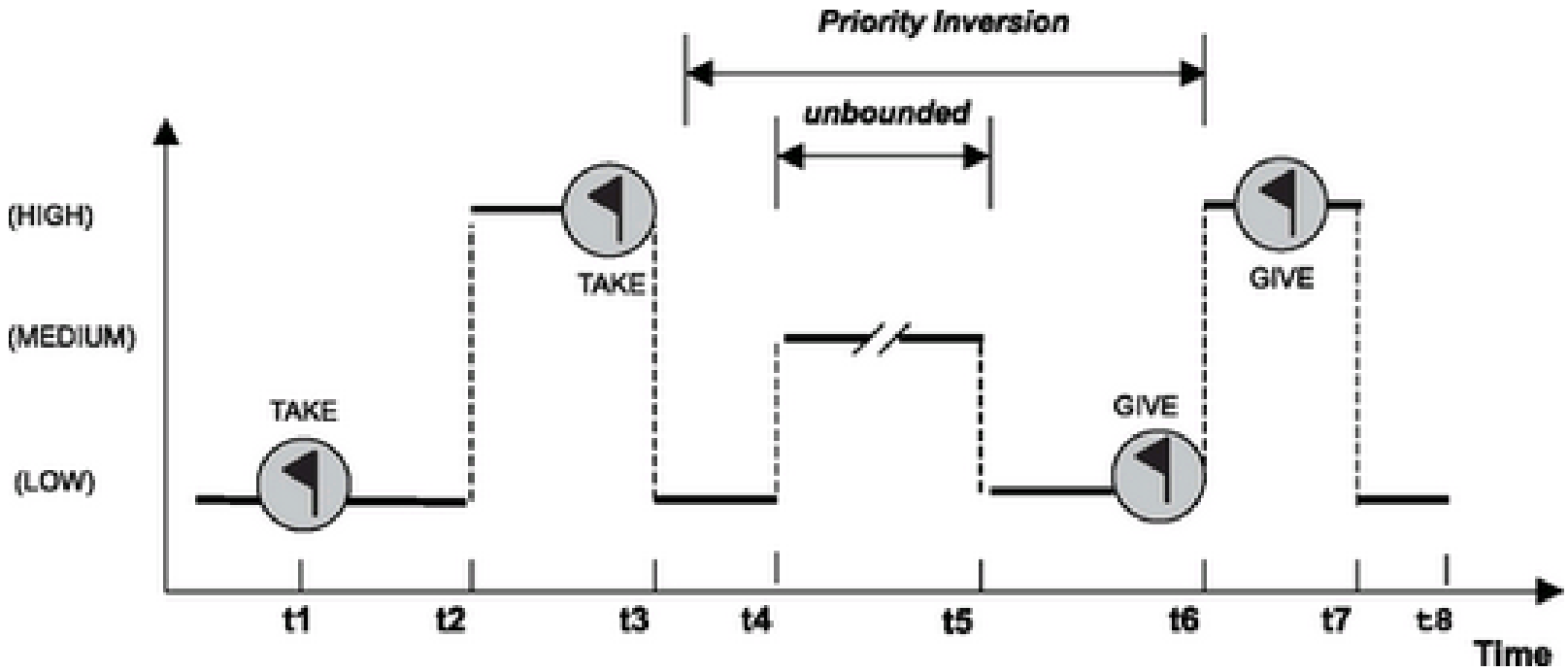

Figura: Inversão de prioridade sem limite definido

Inversão de Prioridades (cont...)

O problema de inversão de prioridades não pode ser evitado, mas pode ser minimizado via utilização de protocolos de controle para acesso a recursos

Protocolo de controle de acesso a recursos

É um conjunto de regras que define condições sob as quais um recurso pode ser fornecido para uma tarefa requisitante e governa propriedades de execução do escalonamento de uma tarefa que detém o recurso

Protocolo de Herança de Prioridade

Protocolo que **eleva** a prioridade de uma tarefa (se esta tarefa segura um recurso requisitado por outra tarefa de prioridade maior) para o mesmo nível de prioridade da tarefa de maior prioridade

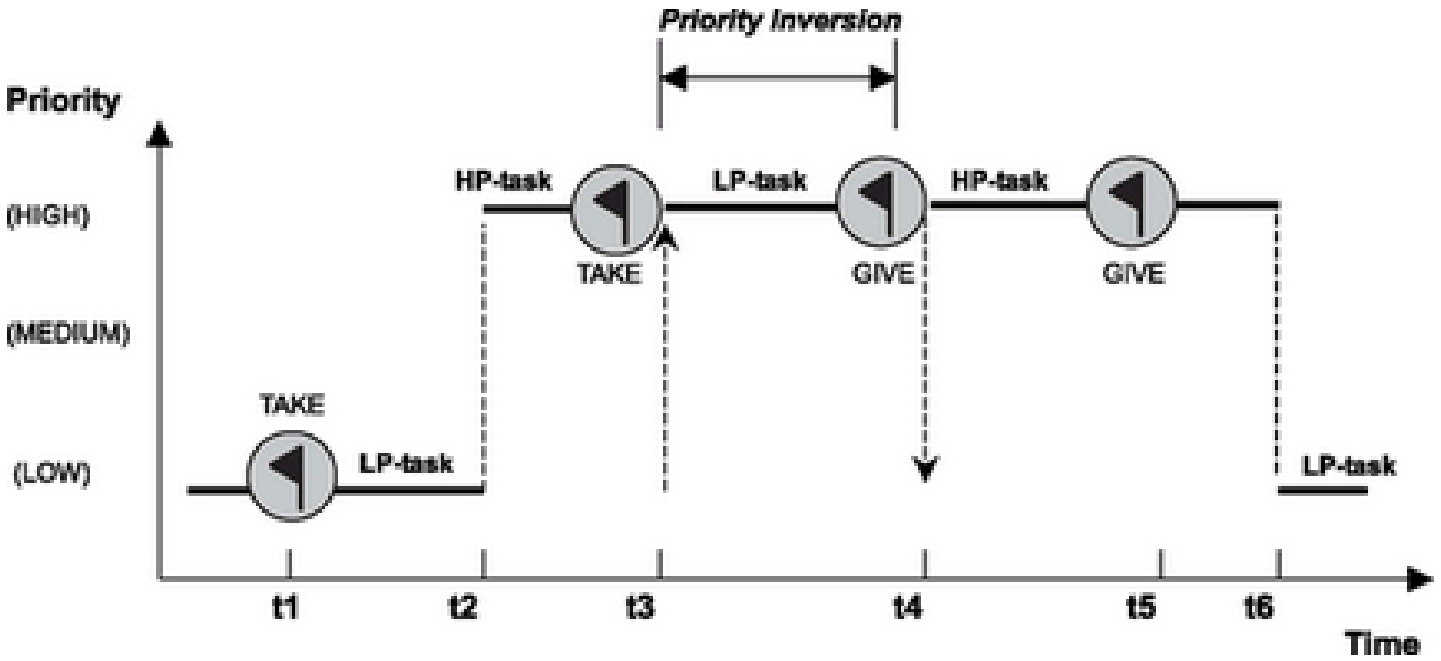

Figura: Exemplo de protocolo de herança de prioridade

Protocolo de Herança de Prioridade (cont...)

Regras:

1. Se `R` estiver em uso, `T` é bloqueada
2. Se `R` estiver livre, `R` é alocado para `T`
3. Quanto uma tarefa de maior prioridade requisita o mesmo recurso, a prioridade de `T` é elevada a mesma prioridade da tarefa requisitante
4. A tarefa retorna para o nível de prioridade anterior quando `R` é liberado

Protocolo de Prioridade Topo

Ceiling Priority Protocol

Neste protocolo, a prioridade de cada tarefa é conhecida, assim como os recursos necessários por cada tarefa

Para um dado recurso, o topo de prioridade (*priority ceiling*) é a maior prioridade entre todas as possíveis tarefas que possam requerer o recurso

Por exemplo, se um recurso `R` é requerido por 4 tarefas

- `T1` com prioridade 4
- `T2` com prioridade 9
- `T3` com prioridade 10
- `T4` com prioridade 8

O *priority ceiling* de `R` é 10

Protocolo de Topo de Prioridade

Priority Ceiling Protocol

Neste protocolo, a prioridade de cada tarefa é conhecida, assim como os recursos necessários por cada tarefa

O topo de prioridade **corrente** para um sistema em execução em qualquer instante de tempo é a maior prioridade topo de todos os **recursos em uso naquele tempo**

Por exemplo, se 4 recursos estão em uso e:

- `R1` tem topo de prioridade 4
- `R2` tem topo de prioridade 9
- `R3` tem topo de prioridade 10
- `R4` tem topo de prioridade 8

O topo de prioridade corrente (*current priority ceiling*) do sistema é 10

Informação ao leitor

Notas de aula baseadas nos seguintes textos:


Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*.
CMP books, 2003.