

# Capítulo 7

## Tópicos em gestão de tarefas

Este capítulo traz tópicos de estudo específicos, que aprofundam ou complementam os temas apresentados nesta parte do livro, mas cuja leitura não é essencial para a compreensão do conteúdo principal da disciplina. Algumas seções deste capítulo podem estar incompletas ou não ter sido revisadas.

### 7.1 Inversão e herança de prioridades

Um problema importante que pode ocorrer em sistemas com escalonamento baseado em prioridade é a *inversão de prioridades* [Sha et al., 1990]. A inversão de prioridades ocorre quando uma tarefa de alta prioridade é impedida de executar por causa de uma tarefa de baixa prioridade.

Este tipo de problema envolve o conceito de *exclusão mútua*: alguns recursos do sistema devem ser usados por uma tarefa de cada vez, para evitar problemas de consistência de seu estado interno. Isso pode ocorrer com arquivos, portas de entrada/saída, áreas de memória compartilhada e conexões de rede, por exemplo. Quando uma tarefa obtém acesso a um recurso com exclusão mútua, as demais tarefas que desejam usá-lo ficam esperando no estado suspenso, até que o recurso esteja novamente livre. As técnicas usadas para implementar a exclusão mútua são descritas no Capítulo 10.

Para ilustrar esse problema, pode ser considerada a seguinte situação: um determinado sistema possui uma tarefa de alta prioridade  $t_a$ , uma tarefa de baixa prioridade  $t_b$  e uma tarefa de prioridade média  $t_m$ . Além disso, há um recurso  $R$  que deve ser acessado em exclusão mútua; para simplificar, somente  $t_a$  e  $t_b$  estão interessadas em usar esse recurso. A seguinte sequência de eventos, ilustrada na Figura 7.1, é um exemplo de como pode ocorrer uma inversão de prioridades:

1. Em um dado momento, o processador está livre e é alocado a uma tarefa de baixa prioridade  $t_b$  que é a única tarefa na fila de prontas;
2. durante seu processamento,  $t_b$  solicita acesso exclusivo ao recurso  $R$  e começa a usá-lo;
3. a tarefa  $t_m$  com prioridade maior que  $t_b$  acorda e volta à fila de prontas;
4.  $t_b$  volta ao final da fila de tarefas prontas, aguardando o processador; enquanto  $t_b$  não voltar a executar, o recurso  $R$  permanecerá alocado a ela e ninguém poderá usá-lo;

- a tarefa de alta prioridade  $t_a$  acorda, volta à fila de prontas, recebe o processador e solicita acesso ao recurso  $R$ ; como o recurso está alocado a  $t_b$ , a tarefa  $t_a$  é suspensa até que  $t_b$  libere o recurso.

Assim, a tarefa de alta prioridade  $t_a$  não pode executar, porque o recurso de que necessita está nas mãos da tarefa de baixa prioridade  $t_b$ . Por sua vez,  $t_b$  não pode executar, pois o processador está ocupado com  $t_m$ .

Dessa forma,  $t_a$  deve esperar que  $t_b$  execute e libere  $R$ , o que configura a inversão de prioridades. A espera de  $t_a$  pode ser longa, pois  $t_b$  tem baixa prioridade e pode demorar a executar novamente, caso existam outras tarefas em execução no sistema (como  $t_m$ ). Como tarefas de alta prioridade são geralmente críticas para o funcionamento de um sistema, a inversão de prioridades pode ter efeitos graves.

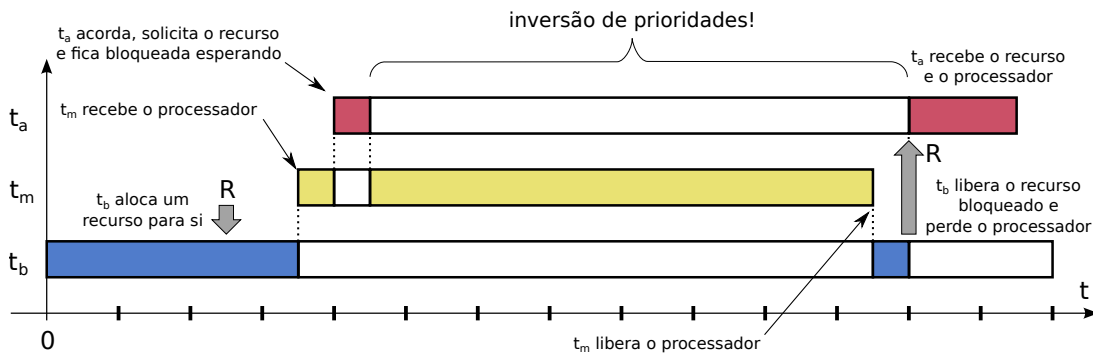


Figura 7.1: Cenário de uma inversão de prioridades.

Uma solução elegante para o problema da inversão de prioridades é obtida através de um *protocolo de herança de prioridade* [Sha et al., 1990]. O protocolo de herança de prioridade mais simples consiste em aumentar temporariamente a prioridade da tarefa  $t_b$  que detém o recurso de uso exclusivo  $R$ . Caso esse recurso seja requisitado por uma tarefa de maior prioridade  $t_a$ , a tarefa  $t_b$  “herda” temporariamente a prioridade de  $t_a$ , para que possa executar e liberar o recurso  $R$  mais rapidamente. Assim que liberar o recurso,  $t_b$  retorna à sua prioridade anterior. Essa estratégia está ilustrada na Figura 7.2.

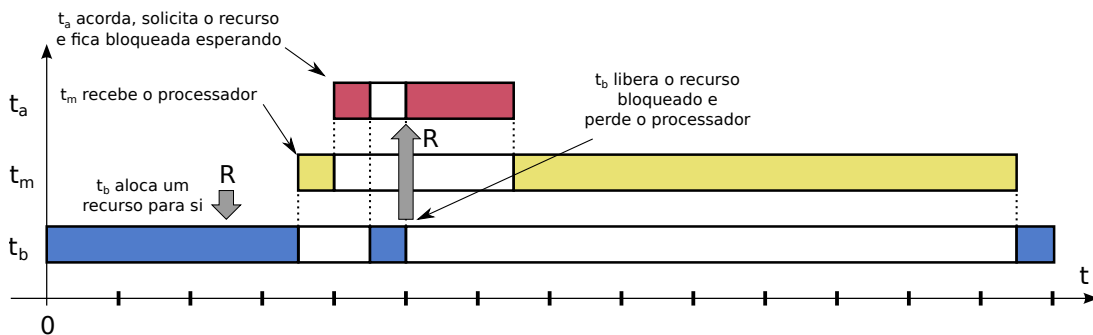


Figura 7.2: Um protocolo de herança de prioridade.

Provavelmente o mais célebre exemplo real de inversão de prioridades tenha ocorrido na sonda espacial *Mars Pathfinder*, enviada pela NASA em 1996 para explorar o solo do planeta Marte (Figura 7.3) [Jones, 1997]. O software da sonda executava sobre o sistema operacional de tempo real *VxWorks* e consistia de 97 *threads* com vários níveis de prioridades fixas. Essas tarefas se comunicavam através de uma área de transferência em

memória compartilhada (na verdade, um *pipe* UNIX), com acesso mutuamente exclusivo controlado por semáforos (semáforos são estruturas de sincronização discutidas na Seção 11.1).

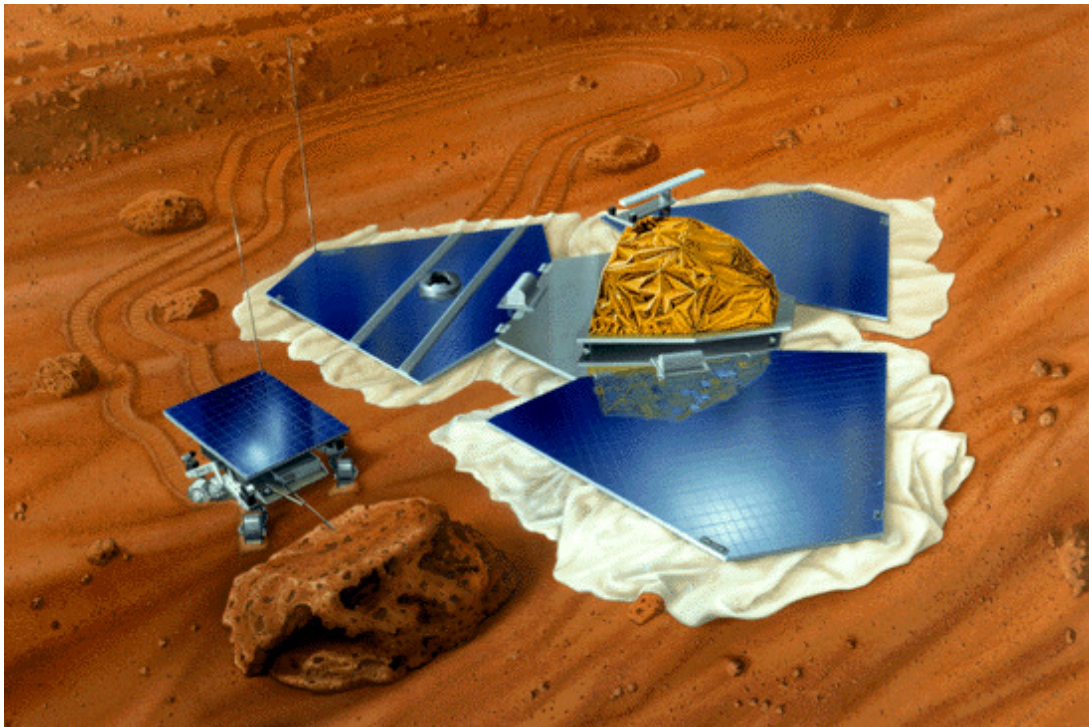


Figura 7.3: Sonda *Mars Pathfinder* com o robô *Sojourner* (NASA).

A gerência da área de transferência estava a cargo de uma tarefa  $t_{ger}$ , rápida e de alta prioridade, que era ativada frequentemente para mover blocos de informação para dentro e fora dessa área. A coleta de dados meteorológicos era feita por uma tarefa  $t_{met}$  de baixa prioridade, que executava esporadicamente e escrevia seus dados na área de transferência, para uso por outras tarefas. Por fim, a comunicação com a Terra estava sob a responsabilidade de uma tarefa  $t_{com}$  de prioridade média e potencialmente demorada (Tabela 7.1 e Figura 7.4).

tarefa	função	prioridade	duração
$t_{ger}$	gerência da área de transferência	alta	curta
$t_{met}$	coleta de dados meteorológicos	baixa	curta
$t_{com}$	comunicação com a Terra	média	longa

Tabela 7.1: Algumas tarefas do software da sonda *Mars Pathfinder*.

Como o sistema *VxWorks* usa um escalonador preemptivo com prioridades fixas, as tarefas eram atendidas conforme suas necessidades na maior parte do tempo. Todavia, a exclusão mútua no acesso à área de transferência escondia uma inversão de prioridades: caso a tarefa de coleta de dados meteorológicos  $t_{met}$  perdesse o processador sem liberar a área de transferência, a tarefa de gerência  $t_{ger}$  teria de ficar esperando até que  $t_{met}$  voltasse a executar para liberar a área. Isso poderia demorar se, por azar, a tarefa de comunicação  $t_{com}$  estivesse executando, pois ela tinha mais prioridade que  $t_{met}$ .

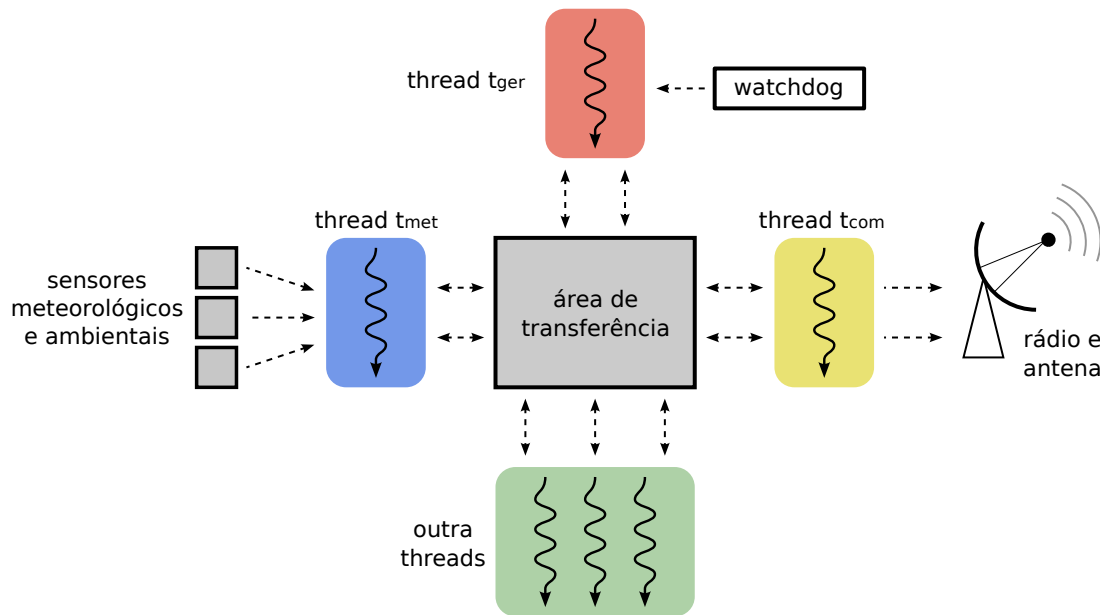


Figura 7.4: Principais tarefas do software embarcado da sonda *Mars Pathfinder*.

Como todos os sistemas críticos, a sonda *Mars Pathfinder* possui um sistema de proteção contra erros, ativado por um temporizador (*watchdog*). Caso a gerência da área de transferência ficasse parada por muito tempo, um procedimento de reinício geral do sistema (*reboot*) era automaticamente ativado pelo temporizador. Dessa forma, a inversão de prioridades provocava reinícios esporádicos e imprevisíveis no software da sonda, interrompendo suas atividades e prejudicando seu funcionamento. A solução foi obtida através de um *patch*<sup>1</sup> que ativou a herança de prioridades: caso a tarefa de gerência  $t_{ger}$  fosse bloqueada pela tarefa de coleta de dados  $t_{met}$ , esta última herdava a alta prioridade de  $t_{ger}$  para poder liberar rapidamente a área de transferência, mesmo se a tarefa de comunicação  $t_{com}$  estivesse em execução.

## Exercícios

1. Explique os conceitos de *inversão* e *herança de prioridade*.
2. O sistema operacional da sonda *Mars Pathfinder* (Seção 7.1) usa escalonamento por prioridades preemptivo, sem envelhecimento e sem compartilhamento de tempo. Suponha que as tarefas  $t_g$  e  $t_m$  detêm a área de transferência de dados durante todo o período em que executam. Os dados de um trecho de execução das tarefas são indicados na tabela a seguir (observe que  $t_g$  executa mais de uma vez).

Tarefa	$t_g$	$t_m$	$t_c$
ingresso	0, 5, 10	2	3
duração	1	2	10
prioridade	alta	baixa	média

<sup>1</sup>Fica ao leitor imaginar como pode ser depurado e corrigido um bug de software em uma sonda a 100 milhões de Km da Terra...

Desenhe o diagrama de tempo da execução sem e com o protocolo de herança de prioridades e discuta sobre as diferenças observadas entre as duas execuções.

## Referências

- M. Jones. What really happened on Mars Rover Pathfinder. *ACM Risks-Forum Digest*, 19(49), 1997.
- L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.