Suponha que diversos threads estejam disputando um recurso que requer acesso exclusivo. A Figura mostra um monitor simples para manipular a alocação e desalocação desse recurso.

A linha 4 declara a variável de estado inUse que monitora se um recurso está ou não em uso. A linha 5 declara uma variável condicional na qual espera um thread que encontrou o recurso não disponível e é sinalizada por um thread que está devolvendo o recurso (tornando-o, portanto, disponível). As linhas 7-17 e 19-25 declaram duas rotinas de entrada no monitor. Para indicar que essas rotinas são de entrada no monitor (e não rotinas privadas do monitor), cada uma recebe um prefixo com a palavra-chave em pseudocódigo monitorEntry.

A linha 8 inicia o método getResource que um thread chama para requisitar o recurso associado à variável condicional available. A linha 10 testa o estado da variável inUse para verificar se o recurso está em uso. Se esse valor for true, significando que o recurso foi alocado a um outro thread, o thread que está chamando deverá esperar na variável condicional available (linha 12). Após chamar wait, o thread sai do monitor e é colocado na fila associada à variável condicional available. Como veremos, isso permite que o thread que está usando o recurso entre no monitor e libere o recurso sinalizando a variável condicional available. Quando o recurso não estiver em uso, o thread que o requisitar executará a linha 15 que coloca inUse em true, concedendo ao thread acesso exclusivo ao recurso.

A linha 20 inicia o método returnResource que um thread chama para liberar o recurso. A linha 22 põe o valor de inUse em false indicando que o recurso não está mais em uso e pode ser alocado a um outro thread. A linha 23 chama signal na variável condicional available para alertar quaisquer threads que estejam à espera que o recurso agora está livre. Se houver threads à espera na fila associada à available (como resultado da execução da linha 12), o próximo thread que estiver esperando entrará novamente no monitor e excutará a linha 15 obtendo acesso exclusivo ao recurso. Se não houver threads esperando em available, signal não terá nenhum efeito.

A beleza do monitor da Figura é que ele funciona exatamente como um semáforo binário: o método getResource age como a operação *P*; o método returnResource age como a operação *V*. Como o monitor simples de um só recurso pode ser usado para implementar semáforos, monitores são, no mínimo, tão poderosos quanto semáforos. Note que a inicialização do monitor (linhas 3-5) é executada antes que o thread comece a usar o monitor; nesse caso inUse é colocada em false indicando que o recurso está inicialmente disponível.

Chegando agora à resposta: se a linha 23 fosse omitida todas as threads que se bloquearem na variável condicional *available* sofreriam deadlock.