

1. Considere a seguinte situação. Em um determinado stand de uma feira, um demonstrador apresenta um filme sobre a vida de Hoare. Quando 10 pessoas chegam, o demonstrador fecha o pequeno auditório que não comporta mais do que essa platéia. Novos candidatos a assistirem o filme devem esperar a próxima exibição. Esse filme faz muito sucesso com um grupo grande de fãs (de bem mais de 10 pessoas), que permanecem na feira só assistindo o filme seguidas vezes. Cada vez que um desses fãs consegue assistir uma vez o filme, ele vai telefonar para casa para contar alguns detalhes novos para sua mãe. Depois de telefonar ele volta mais uma vez ao stand para assistir o filme outra vez.

Usando semáforos, modele o processo *fã* e o processo *demonstrador*, lembrando que existem muitos fãs e apenas um demonstrador. Como cada fã é muito ardoroso, uma vez que ele chega ao stand ele não sai dali até assistir o filme. Suponha que há muitos telefones disponíveis na feira, e portanto que a tarefa de telefonar para casa não impõe nenhuma necessidade de sincronização.

Atenção: Observe que o demonstrador só pode começar a exibir o filme quando há 10 pessoas no stand, e que as pessoas que chegam durante uma exibição têm que esperar a próxima. E *importante:* observe que um fã só pode ir telefonar para a mãe depois que acaba a exibição do filme! Isso tem que estar modelado na sincronização entre os processos demonstrador e fãs.

2. O que é um escalonador *preemptivo*? Dê exemplo(s) de motivo(s) para ocorrência da preempção.
3. Foram discutidas em sala quatro condições necessárias para que ocorram *deadlocks* em um sistema. Explique cada uma das condições abaixo, discutindo como elas podem ser negadas.
 - (a) espera circular
 - (b) hold & wait
4. Considere o problema do jantar comunal discutido em sala: Suponha que um grupo de N canibais come jantares a partir de uma grande travessa que comporta M porções. Quando alguém quer comer, ele(ela) se serve da travessa, a menos que ela esteja vazia. Se a travessa está vazia, o canibal acorda o cozinheiro e espera até que o cozinheiro coloque mais M porções na travessa. Desenvolva código para as ações dos canibais e do cozinheiro. Use semáforos para sincronização. A solução deve evitar *deadlock* e deve acordar o cozinheiro apenas quando a travessa estiver vazia. (Suponha um longo jantar, onde cada canibal continuamente se serve e come, sem se preocupar com as demais ações na vida do canibal...)

Considere a seguinte solução, por monitor, para este problema:

<pre>monitor jantarComunal { int comida = 0; cond cozinheiro, travessa; void seServe() { while (comida==0) { signal(cozinheiro); wait(travessa); } comida--; } void encheTravessa() { wait(cozinheiro); comida = MUITA; signalAll(travessa); } }</pre>	<pre>processo canibal executa: void canibal() { while(1) { seServe() come...; } } processo cozinheiro: void cozinheiro() { while(1) { cozinha... encheTravessa(); } }</pre>
--	--

Suponha que esse monitor funciona com a disciplina “signal como dica” dada em sala (mas observe que ele admite filas separadas por variáveis de condição!). Explique por que essa solução não funciona corretamente e sugira uma forma de corrigi-la.

obs: Caso você não saiba que disciplina é essa, escolha uma disciplina compatível com a sintaxe apresentada e faça a questão usando essa disciplina.