Considere uma festa onde, devido a uma falha de organização, 3 convidados sentados em uma mesma mesa devem compartilhar um mesmo copo. Quando o copo é enchido por um garçom, um dos convidados pega o copo e bebe todo o seu conteúdo, deixando-o vazio para que o garçom encha novamente após dar uma volta.

Apenas 3 tipos de bebidas são oferecidas na festa e cada um dos 3 convidados em uma mesa têm preferências distintas:

* O convidado 1, mais "chato", bebe apenas a bebida do tipo 1.
* O convidado 2, mais "de boa", aceita bebidas do tipo 1 e do tipo 2.
* O convidado 3, "pé de cana", aceita qualquer tipo de bebida.

Implemente um monitor que permita a sincronização entre uma thread representando o garçom e 3 threads representando os clientes de uma mesa. O monitor deve obedecer aos seguintes critérios:

* O garçom traz uma bebida aleatória e, se o copo estiver vazio, enche o copo e vai embora. Caso contrário, ele espera que o conteúdo do mesmo seja totalmente consumido para servir a dose seguinte.
* Nenhum convidado tem a preferência, ou seja, se a bebida 1 é trazida, beberá o que pegar o copo primeiro.
* Não pode haver desperdício, assim, todas as doses servidas devem ser consumidas.
* As funções servir e beber devem retornar apenas quando seu objetivo for cumprido.
* Nenhuma das threads deve ficar bloqueada no final da execução. DICA: Quando o garçom terminar de servir todas as doses, faça com que o mesmo sinalize para os convidados que a festa acabou (essa é a única exceção para o critério acima, ou seja, a função beber pode terminar sem que o convidado tenha bebido caso todas as doses já tenham sido servidas).

Um esboço para as threads garçom e convidado é exibido abaixo:

garçom:

loop {

int b=rand()%3+1; //busca uma bebida aleatória

bebida.serve(b); //serve

}

convidado c:

loop {

bebida.bebe(c);

}