Componentes: André Felipe de Paiva Cardozo e Gabriel Garcia de Almeida
DREs: 110144913 e 110093284

Relatório e Documentação do Trabalho 2

Ideia geral:

Após iniciada cada Thread, elas iram entrar numa barreira, implementada em monitor, que irá moderar a sincronização por condição de todas. Saindo da barreira, todas as threads liberadas chamaram o método board() e só a última a sair executaria o método rowBoard(), verificando se ainda é possível liberar o monitor para o próximo grupo de Threads.

Sobre a execução:

A nome classe que deve ser executada (após sua compilação) com o comando 'java' no terminal é 'main', sem nenhum argumento. O programa irá esperar pelas quantidades de programadores C e Java, nessa ordem, da entrada padrão e então disparará suas threads respectivas de forma alternada entre os dois tipos de programadores e com um delay inicial aleatório de no máximo 200 milisegundos.
Durante a execução, o programa irá exibir um log do o que está ocorrendo internamente no monitor, como que threads estão esperando as condições para embarcar, quem liberou essa fila, quantos barcos ainda faltam etc. No final, será exibido quantas threads de Programadores C e Java, respectivamente, não conseguiram embarcar, o total que era esperado e se o resultado final foi igual ao previsto.

Implementação:

- **Classe abstrata Programador:**

Subclasse de Thread. Recebe como parâmetro no construtor apenas seu ID, a referência para o recurso compartilhado (nesse caso é um objeto da classe Barco) e um delay. A implementaçao do método run() da classe Thread é bem simples: primeiro a thread 'dorme' o delay recebido no construtor, em seguida ela entra na fila do Barco. Após liberada, a thread executa o método board() do recurso para embarcar, que ira retornar verdadeiro se ela deverá ser a remadora. Finalmente, o método rowBoat() irá ser chamado ou não, dependendo do resultado anterior.

- **Classe ProgramadorJava e ProgramadorC:**

Classes concretas de Programador. São usadas apenas para diferenciar um tipo de programador do outro, usando a palavra reservada 'instanceof' do Java.

- **Classe Barco:**

Recurso usado pelas threads de Programador, sincronizado por monitor. Todas as sincronizações feitas nessa classe usam 'this' como variável de condição.

Seu construtor recebe apenas a quantidade total de Programadores de cada tipo que viram a usar o recurso.

O método filaEspera() bloqueia as threads que acessam o recurso e as libera de forma coordenada para poderem usar o método board(), sem infligir as regras. Funciona da seguinte forma: Se o barco estiver sendo carregado ou cheio, trava todos que tentarem entrar posteriormente, senão a entrada do programador é registrada e ele espera o barco encher com uma configuração válida, indicada pelo método barcoCheio().
Resumidamente, o método barcoCheio() verifica se já existe uma condição válida qualquer e libera a viagem. Caso ocorra um dos dois casos não permitidos, é esperado um quinto programador, que definirá um dos tipos de programador para esperar até uma viagem

posterior. Essa lógica é suscetível a starvation, porém é necessária em, por exemplo, um caso em que só haja um programador Java e 200 programadores C, esse programador Java deve ser bloqueado constantemente.

Quando um Programador sai da fila, ela chamará board(), que simplesmente registra sua saída da fila e retorna 'true' se foi último a sair da fila. O último ira usar rowBoard(), que libera a fila para a próxima viagem e verifica se ainda é possível haver viagens com a quantidade atual de programadores.

Dificuldades encontradas:

Após alguns problemas tentando usar semáforos, tais como a falta de integração com blocos synchronized, optou-se por usar apenas as diretivas wait() e notifyAll(). Outra dificuldade foi relativa a complexidade nas condições de sincronização, que não são triviais e problemas de deadlock e inconsistencia eram frequentes durante a implementação e geralmente ocorriam quando uma thread que havia "ficado para trás", esperando a viajem seguinte, voltava num momento indevido, fazendo com que o barco atual não fosse liberado e ocasionando um deadlock ou o barco partia com 5 pessoas. Isso foi resolvido fazendo com que essa thread obrigue todos a verificarem novamente sua condições, inclusive essa thread problemática.

Relatório das Execuções:

Como podem ser previstas quantas viagens haverão com aquelas configurações (1), e o total de threads que não embarcarão (2), a verificação da corretude é bem simples, pois os principais problemas são fáceis de se identificar: deadlock, no qual o programa não termina, e barcos saindo com condições inválidas, onde o total de viagens é diferente do que era previsto e/ou o total das que não viajaram não é coerente, por que alguma thread foi num momento indevido.

(1) (cTotal+ javaTotal)%4
(2) (cTotal%4 +javaTotal%4)%4
Obs.: Essas formulas não verificam o caso em que o resultado final é do tipo 3 programadores C e 1 Java, mas a rotina de verificação do programa em si verifica.

Com isso em mente, o programa foi adaptado para indicar se tudo ocorreu como o esperado e foi executado várias vezes com valores digitados a esmo em dois computadores diferentes, exibindo corretude em todas as vezes, excluso os casos em que haviam muitas threads para serem criadas e a JVM ficou sem memória, ocasionando em um deadlock.

A avaliação do quesito desempenho não faz muito sentido, pois como cada thread pode ter um delay antes de começar a executar de até 200 milisegundos para ocorrer as mais diversas situações possíveis, o tempo de execução também será muito aleatório e irrelevante para a aplicação.