UFCG UASC/CEEI

Disciplina: Programação Concorrente Prova 1
Aluno:
Matrícula:

1. Considere uma árvore de pesquisa binária que armazena números inteiros. A árvore é representada usando nós que possuem um atributo **value** (que armazena o valor inteiro), um atributo **left** para a subárvore esquerda e um atributo **right** para a subárvore direita. Suponha que a árvore esteja inicialmente vazia e que várias threads possam acessar a árvore simultaneamente. Escreva pseudocódigo para implementar as operações de inserção e pesquisa da árvore de maneira segura para thread usando semáforos ou variáveis condicionais. Você pode usar o modelo abaixo, como referência para sua implementação.

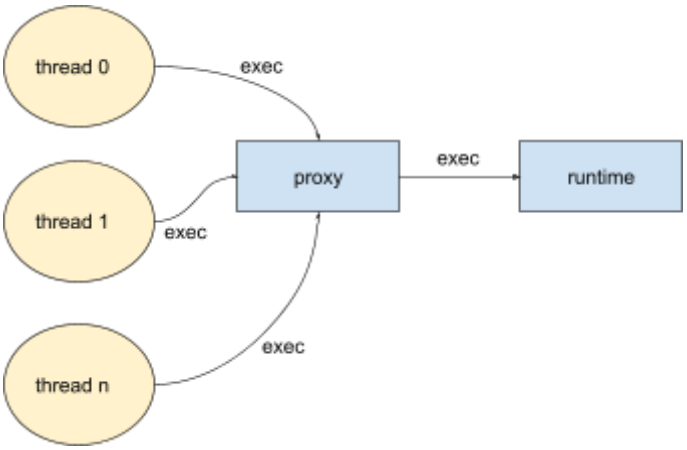
   ```
   class BinarySearchTree()

         void func insert(int valueToInsert)
         boolean func search(int valueToSearch)

         class Node(int value)
               this.value = value
               this.left = None
               this.right = None
   ```

   Você também pode modificar a pergunta pedindo aos alunos que implementem a operação de exclusão em vez de pesquisa. Como alternativa, você pode pedir aos alunos que implementem uma implementação thread-safe de uma estrutura de dados de gráfico usando semáforos ou variáveis condicionais.

2. Muitos sistemas implementam uma arquitetura que considera um componente **proxy** que intercepta chamadas para um componente que é responsável por processar requisição (vamos chamar este último de **runtime**), tal como no esquema abaixo.



Nesse esquema, threads são criadas (p.ex thread 0, 1 .. n) e estas threads executam a função **exec** do código do proxy. O **proxy** tem somente uma função: controlar o repasse de execuções para a **runtime**.

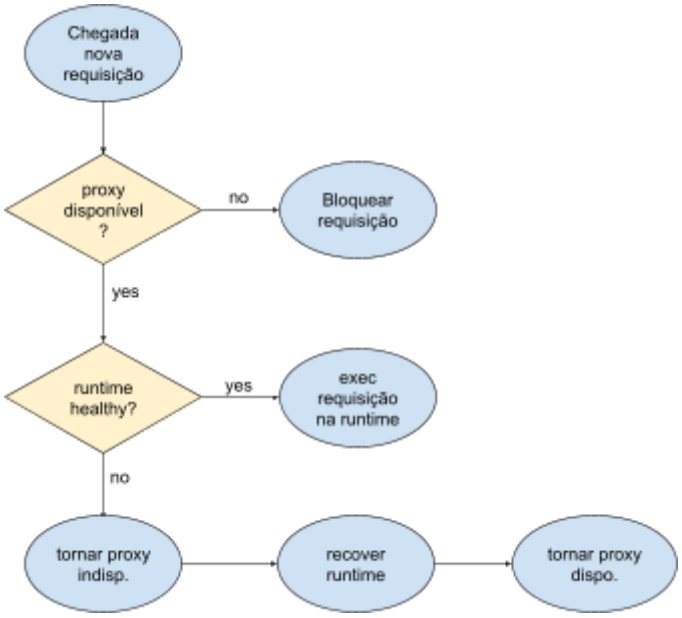
Sua implementação deve considerar no mínimo as seguintes funções. Você **NÃO** deve implementar as funções destacadas em negrito.

Proxy	Runtime
void exec(Request req)	**void exec(Request req)**
boolean isAvailable()	**void recover()**
boolean runtimeIsHealthy()	

Par Atente para os seguintes requisitos:

- Threads são criadas externamente ao seu código. Essas threads chamam a função **request** do **proxy;**
- Requests que chegam no **proxy**, devem ser repassados para a **runtime** somente se o proxy estiver disponível (função **isAvailable()**) e se o runtime estiver **healthy;**
- Se o proxy detectar que **runtime** não está **healthy,** o **proxy** mudar para seu status para **indisponível.** O valor de sua função isAvailable deve ser falso. Também, o **proxy** deve chamar a função **recover** da **runtime.** Ao fim da execução desta função, o **runtime** estará **healthy** novamente e o **proxy** volta para o estado disponível;
- Só uma thread por vez deve executar **exec** na runtime;
- Caso uma thread não possa executar **exec** na **runtime** esse fluxo não deve ser descartado. A thread deve bloquear até poder prosseguir.

Abaixo, uma versão visual de parte destes requisitos. Não necessariamente, você precisa implementar o código nesta sequência.



Você pode criar funções auxiliares para a implementação da abstração Proxy. O controle de concorrência pode ser feito tanto com semáforos quanto por variáveis condicionais. É importante inicializar esses tipos corretamente, p.ex no construtor de suas entidades ou na função **main**, se for o caso.

Caso queira que o proxy mantenha também suas threads, use a seguinte função para criar as threads.

int new_thread(function f, arg)

onde **f** é o nome da função que a thread recém criada, arg é um argumento de qualquer tipo que pode ser passado para função **f**. Além disso, a função **new_thread** retorna um inteiro que identifica a thread recém criada.

API Semáforos/Variáveis condicionais

```
Semaphore (int initialValue)
wait()
signal()


CV()
wait()
signal()
broadcast()
```