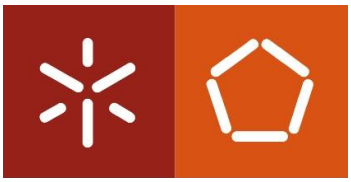

Universidade do Minho

Departamento de Informática

Sistemas Distribuídos

Alarme Covid

Grupo Nº 8

20 de janeiro de 2021

Ariana Lousada (A87998)

Carlos Gomes (A77815)

Márcia Teixeira (A80943)

Tiago Sousa (A67674)

Conteúdo

1 Introdução **3**

2 Arquitetura e Solução do Projeto **4**
2.1 Servidor 4
2.2 Cliente 4
2.3 Comunicação Cliente - Servidor 4
2.3.1 Pedidos e respostas aos pedidos 4
2.4 Gestão de estado partilhado 5

3 Conclusão **6**

Capítulo 1

Introdução

A saúde da população sempre foi um tópico fortemente explorado no sentido de melhorar a qualidade de vida de todos a todo o momento. Numa altura em que o mundo passa por uma crise relacionada com a saúde especificamente com a propagação do COVID-19, um vírus altamente contagioso, é necessário ter este assunto em ainda maior consideração.

De modo a poder ser dada à população uma forma intuitiva e eficaz de se manterem informados acerca de possíveis surtos a que possa ter estado sujeitos, foi desenvolvida a aplicação STAYAWAY COVID.

A respeito da unidade curricular de Sistemas Distribuídos, foi-nos proposta a elaboração de um sistema com funcionalidades semelhantes ao da aplicação supra-mencionada.

De forma a aplicar os conceitos abordados em contexto de sala de aula, o sistema a desenvolver (Alarme Covid), terá uma arquitetura que segue a lógica Cliente - Servidor com recurso a Sockets TCP e Threads.

É então necessário implementar um sistema que permita a utilização concorrente de um servidor por vários clientes, assegurando a consistência de toda a informação introduzida no sistema para qualquer cliente que esteja a fazer uso/manipulação da mesma.

Ao longo do presente relatório, será descrita a nossa abordagem ao problema apresentado.

Capítulo 2

Arquitetura e Solução do Projeto

2.1 Servidor

No módulo Servidor são tratadas todas as conexões e pedidos dos clientes, garantindo a capacidade de suportar vários clientes em simultâneo.

Para além disso, é tratada a gestão da informação relativa a logins/registos e localizações de forma a que seja possível um acesso concorrente entre as diversas Threads lançadas.

A receção, envio e armazenamento de dados é portanto tratada neste módulo, tendo em vista uma boa gestão de recursos, tanto a nível de memória como a nível de CPU.

De forma a conseguir um servidor capaz de lidar com vários clientes em simultâneo, este é dividido em várias Threads, sendo que é criada uma nova para tratar de cada cliente que se tente conectar.

2.2 Cliente

No módulo Cliente é efetuada a ponte entre o utilizador e o servidor, sendo o cliente capaz de efetuar pedidos ao servidor e este por sua vez apresentar ao utilizador a resposta aos pedidos efetuados.

2.3 Comunicação Cliente - Servidor

2.3.1 Pedidos e respostas aos pedidos

Para trocar informação entre o cliente e o servidor fixamos um protocolo de comunicação que especifica o formato em que os pedidos e as respetivas respostas devem ser construídos.

Este formato foi inspirado em CSV, em que o primeiro elemento corresponde à especificação do pedido, e os restantes correspondem aos argumentos deste, quando se trata do fluxo de informação Cliente-Servidor.

Um exemplo deste protocolo encontra-se representado abaixo com um pedido de registo que possui o seguinte formato:

register;username;password;latitude;longitude

Aplicando isto a um exemplo concreto, temos:

register;jose;123456;10;10

Figura 2.1: Formato pedido de registo

Em relacão ao fluxo Servidor-Cliente, caso o pedido não tenha sido executado com sucesso, o primeiro elemento corresponde ao sinalizador de erro, e o seguinte à especificação do mesmo. Caso tenha sido completado com sucesso, o primeiro representa uma resposta de sucesso, ou o identificador do pedido, e o segundo representa o output do mesmo ou um atributo relevante do pedido a que esta comunicação responde.

Um exemplo deste protocolo encontra-se representado abaixo com uma resposta a um pedido de registo efetuada com sucesso.

ok;registered

Figura 2.2: Formato de resposta a um registo (sucesso)

Caso já exista um utilizador registado com o username dado a resposta passa a ser a seguinte:

err;username_already_exists

Figura 2.3: Formato de resposta a um registo (insucesso)

2.4 Gestão de estado partilhado

Tendo presentes no código várias estruturas de dados acedidas por várias threads em simultâneo, nomeadamente os Utilizadores registados no sistema, as suas localizações e uma listagem daqueles que se encontram em espera para saber quando uma localização se encontra vazia, o código encontra-se vulnerável à ocorrência de erros derivados do acesso concorrente a memória.

No sentido de evitar este problema é necessário garantir que não existe mais do que uma Thread a executar código que acede a memória ao mesmo tempo. Para tal, tornou-se crucial etiquetar estes acessos à memória, fosse para ler valores como para os alterar, como sendo linhas de código atómicas ou indivisíveis, introduzindo-se a Exclusão Mútua através do uso de ReentrantLocks, em que delimitamos estas zonas críticas através do uso dos métodos lock() e unlock().

O uso desta funcionalidade permite-nos garantir a segurança, impedindo que duas ou mais Threads executem o mesmo código de acesso a memória ao mesmo tempo.

Capítulo 3

Conclusão

Para concluir, procuramos cumprir todos os requisitos propostos, incluindo os que foram identificados como adicionais. Para tornar a interface intuitiva para o utilizador foi alterado o esquema de introdução de pedidos ao sistema, sendo este esquema convertido para o protocolo por nós instituído internamente para que seja processado pelo servidor:

1. **Autenticação e registo de utilizadores:**

   ```
   register --username user --password pass --latitude lat
                       --longitude long

   login --username user --password pass --latitude lat
                       --longitude long
   ```

2. **Informar o servidor da localização atual:**

   ```
   location --latitude lat --longitude long
   ```

3. **Requisitar o número de pessoas que se encontram numa determinada localização**

   ```
   number-people --latitude lat --longitude long
   ```

4. **Requisitar notificação de quando uma localização se encontrar vazia**

   ```
   empty --latitude lat --longitude long
   ```

5. **Comunicar ao servidor está infetado**

   ```
   sick
   ```

Adicionalmente foram implementadas as funcionalidades adicionais.

1. **Notificar utilizadores potencialmente infetados**
2. **Utilizador especial requisitar um mapa de informações**

   ```
   vip-info
   ```

Consideramos que o sistema responde da melhor forma ao pedido, não descartanto imensas possíveis futuras melhorias ao mesmo.