Servidor de Mensagens

Execução: Individual
Data de entrega: 25 de junho de 2021

Introdução

Desde o início de 2020 o mundo vive um período de pandemia com o surgimento do vírus coronavírus (COVID-19) e suas variantes, o que mudou muitos dos aspectos das nossas vidas. Os cientistas do mundo têm trabalhado incansavelmente para termos vacinas e outras ações que nos permitam voltar à nossa antiga normalidade. Algumas vacinas surgiram e os países vivem diferentes momentos de vacinação onde pessoas de diferentes faixas etárias e profissões estão vacinadas. Com o intuito de colaborar para que o cidadão saiba para onde se dirigir quando for o seu momento de vacinação, colocaremos em prática os conceitos aprendidos na disciplina de redes de computadores. Assim, neste trabalho iremos implementar um **sistema modelo servidor e clientes** para troca de mensagens tal que as pessoas possam achar um local de vacinação a partir da sua posição geográfica num dado momento.

O sistema possui 4 tipos de mensagens para comunicação clientes-servidor que foram divididos, para uma melhor compreensão, em dois grupos: SAÚDE e CIDADÃO. O primeiro grupo é composto por três tipos de mensagens que são usados para controle e configuração do servidor pelas autoridades de saúde que indicarão os locais de vacinação. O último grupo é composto por apenas um tipo de mensagem que é usado para consulta e atendimento às solicitações dos clientes propriamente dito. **No trabalho, você deve implementar os quatro tipos de mensagens.**

Protocolo

Com os tipos de mensagens do grupo SAÚDE, os clientes podem adicionar, remover e consultar locais de vacinação. Já com o único tipo de mensagem do grupo CIDADÃO, os clientes enviam para o servidor sua localização para conhecer o local mais próximo de vacinação. O servidor recebe as mensagens dos clientes, faz a verificação da localização atual e em seguida responde ao cliente qual é o local mais próximo. Um local de vacinação é definido através de uma posição geográfica (X, Y), com $0 \leq X \leq 9999$ e $0 \leq Y \leq 9999$.

O servidor e clientes trocam mensagens curtas de até 500 bytes usando o protocolo TCP. Mensagens carregam texto codificado segundo a tabela ASCII. Apenas letras, números e espaços podem ser transmitidos (caracteres acentuados não podem ser transmitidos).

Na sequência, são definidos o formato de cada tipo de mensagem, bem como a resposta esperada para cada uma deles.

Grupo SAÚDE:

- **Adicionar local de vacinação:** clientes enviam uma mensagem no formato "**add X Y**" para o servidor. O servidor, por sua vez, deve confirmar a adição do local de vacinação com a mensagem "**X Y added**". Caso o local **X Y** já tenha sido adicionado anteriormente, o servidor deve retornar a mensagem "**X Y already exists**" para o cliente. Serão

adicionados no máximo 50 locais de vacinação. Caso o limite seja atingido, o servidor deve retornar a mensagem "**limit exceeded**".

- **Remover local de vacinação:** clientes enviam uma mensagem no formato "**rm X Y**" para o servidor. O servidor, por sua vez, deve confirmar que o local de vacinação foi removido com a mensagem "**X Y removed**". Caso o local **X Y** não tenha sido adicionado anteriormente, o servidor deve retornar a mensagem "**X Y does not exist**".
- **Consultar locais de vacinação:** clientes enviam uma mensagem no formato "**list**" para o servidor. O servidor, por sua vez, retorna **todos** os locais de vacinação que foram adicionados para o cliente no formato "$\mathbf{X_1\ Y_1\ X_2\ Y_2\ \ldots\ X_n\ Y_n}$". Caso nenhum local de vacinação tenha sido adicionado ainda, o servidor deve retornar a mensagem "**none**".

Grupo CIDADÃO:

- **Consultar o local de vacinação mais próximo:** clientes enviam uma mensagem contendo sua localização atual para o servidor no formato "**query X Y**". O servidor, por sua vez, deve retornar a coordenada do local de vacinação mais próximo ao local informado pelo cliente no formato "**X Y**". Caso nenhum local de vacinação tenha sido adicionado ainda, o servidor deve retornar a mensagem "**none**".

Abaixo seguem dois exemplos de comunicação do cliente com o servidor, onde linhas começando com > são enviadas pelo cliente e linhas começando com < foram recebidas do servidor:

Exemplo 1

```
> add 111 111
< 111 111 added
> add 111 111
< 111 111 already exists
> add 222 222
< 222 222 added
> list
< 111 111 222 222
> rm 111 111
< 111 111 removed
> rm 111 111
< 111 111 does not exist
```

Exemplo 2

```
> add 10 15
< 10 15 added
> add 89 100
< 89 100 added
```

```
> query 90 90
< 89 100
> query 5 4
< 10 15
```

Detalhes de implementação do protocolo:

- As mensagens são terminadas com um caractere de quebra de linha `'\n'`. O caractere nulo `'\0'` para terminação de strings em C *não* deve ser enviado na rede.
- O servidor deve descartar mensagens com caracteres inválidos ou com um formato desconhecido. O servidor pode desconectar o cliente que enviou a mensagem, mas precisa continuar a operação sem impacto para outros clientes.
- Para funcionamento do sistema de correção semi-automática (descrito abaixo), seu servidor deve fechar todas as conexões e terminar sua execução ao receber uma mensagem contendo apenas "**`kill`**" de qualquer um dos clientes.

Como especificado acima, mensagens podem ter até 500 bytes e o fim de uma mensagem é identificado com um caractere `'\n'`. Uma mensagem não pode ultrapassar 500 bytes (i.e., um caractere `'\n'` deve aparecer entre os primeiros 500 bytes). Caso essas condições sejam violadas, o servidor pode inferir que há um *bug* no cliente e desconectá-lo.

Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor.

Implementação

O aluno deve implementar uma versão do servidor e uma versão do cliente. O servidor e o cliente devem utilizar o protocolo TCP, criado com [socket(AF_INET, SOCK_STREAM, 0)] ou com [socket(AF_INET6, SOCK_STREAM, 0)], para comunicação. Deve ser possível utilizar tanto IPv4 quanto IPv6.

Seu cliente deve receber mensagens do teclado e imprimir as mensagens recebidas na tela.

O servidor deve imprimir na saída padrão todas as mensagens recebidas de clientes. **Não é necessário** que seu servidor aceite mais de um cliente simultaneamente.

Seu servidor deve receber, **estritamente nessa ordem**, o tipo de endereço que será utilizado (**v4** para IPv4 ou **v6** para IPv6) e um número de porta na linha de comando especificando em qual porta ele vai receber conexões. Seu cliente deve receber, **estritamente nessa ordem**, o endereço IP e a porta do servidor para estabelecimento da conexão. Exemplo de execução dos programas em dois terminais distintos:

IPv4:

```
no terminal 1: ./servidor v4 51511
```

```
no terminal 2: ./cliente 127.0.0.1 51511
```

IPv6:

```
no terminal 1: ./servidor v6 51511
no terminal 2: ./cliente ::1 51511
```

O servidor pode dar bind em todos os endereços IP associados às suas interfaces usando a constante INADDR_ANY para IPv4 ou in6addr_any para IPv6.

Limites:

- Cada mensagem possui no máximo 500 bytes.
- Serão adicionados no máximo 50 locais de vacinação.
- As coordenadas X e Y devem ser valores inteiros entre 0 e 9999 ($0 \leq X \leq 9999$ e $0 \leq Y \leq 9999$).

Materiais para consulta:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle.
- Playlist de programação com soquetes.

Avaliação

Este trabalho deve ser realizado individualmente e **deve ser implementado em C** utilizando apenas a biblioteca padrão (interface POSIX de soquetes de rede). Seu programa deve rodar no sistema operacional Linux e, em particular, **não deve utilizar bibliotecas do Windows,** como o winsock. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo (você pode testar com as implementações dos seus colegas). Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor. O seu servidor será testado por um cliente implementado pelo professor com funcionalidades adicionais para realização dos testes. Os testes avaliam a aderência do seu servidor ao protocolo de comunicação inteiramente através dos dados trocados através da rede (a saída do seu servidor na tela, e.g., para depuração, não impacta os resultados dos testes).

O cliente implementado pelo professor para realização dos testes bem como um teste de exemplo estão disponíveis no Moodle para que você possa testar a compatibilidade de seu servidor com o ambiente de testes.

Pelo menos os seguintes testes serão realizados:

- Adicionar locais de vacinação.
- Remover locais de vacinação.
- Listar todos os locais de vacinação.
- Consultar o local de vacinação mais próximo.
- Recebimento de mensagens particionadas em múltiplas partes (mensagem recebida parcialmente no primeiro [recv]).
    - Este teste cobre o caso de uma mensagem com, por exemplo, 20 caracteres, que é enviada em dois pacotes. Neste teste, o servidor pode receber os primeiros 10 caracteres da mensagem em um `recv` e receber os últimos 10 caracteres da mensagem em um `recv` subsequente. É tarefa do servidor detectar que os primeiros 10 caracteres não possuem o `'\n'`, determinar que a mensagem ainda não chegou por completo, e chamar `recv` novamente até terminar de receber a mensagem para então processá-la.
- Recebimento de múltiplas mensagens em uma única chamada ao [recv].
    - Note que a sequência de bytes
      “`add 111 111`**\n”**
      contém uma mensagem, enquanto a sequência de bytes
      “`add 111 111`**\n**`add 222 222`**\n”**
      contém duas mensagens. Este teste cobre o segundo caso acima. Uma sequência de bytes, que pode ser lida em um único `recv`, contendo duas mensagens será enviada ao servidor. O servidor deve processar as duas mensagens corretamente.

Note que apesar do programa cliente não ser avaliado no conjunto de testes, ele ainda será avaliado manualmente pelo monitor.

Entrega

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.
Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar o código fonte em C e um Makefile para compilação do programa. Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado “`cliente`” e o servidor em um binário chamado “`servidor`”.
- Seu programa deve ser compilado ao se executar apenas o comando “`make`”, ou seja, sem a necessidade de parâmetros adicionais.

Desconto de nota por atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2^{d-1} \div 0.32\ \%$$

onde *d* é o atraso em dias úteis. Note que após 5 dias, o trabalho não pode ser mais entregue.