Servidor de Mensagens

Pokédex

Execução: Individual
Data de entrega: 17 de novembro de 2021

INTRODUÇÃO

Pokémon é uma conhecida franquia de videogames desenvolvida desde 1996, no Japão, pela Nintendo. Os Pokémons são monstrinhos semelhantes a animais que permeiam o mundo imaginário da franquia. O objetivo principal do jogador é capturar, treinar e batalhar com seus Pokémons para adquirir experiência e evoluir dentro do jogo. Para ajudar o treinador em sua jornada, cada jogador possui um dispositivo conhecido como Pokédex, onde ele consegue armazenar diversas informações sobre os Pokémons encontrados e capturados. Com esta motivação, o objetivo deste trabalho prático é implementar um **sistema modelo servidor e clientes** para simular a interação de um treinador com sua Pokédex.

O sistema deve possuir 4 mensagens principais, sendo elas: adicionar um novo Pokémon, remover um Pokémon existente no servidor, listar a relação de Pokémons salvos no servidor e realizar uma troca entre dois Pokémons. Para ajudar na implementação e nos testes, você pode consultar o site https://www.pokemon.com/br/pokedex/ para obter os nomes e informações dos 891 Pokémon existentes. Neste trabalho, **você deve implementar os quatro tipos de mensagens propostos**.

PROTOCOLO

O **cliente** será o treinador Pokémon e o **servidor** atuará como a Pokédex**.** Um treinador poderá executar as seguintes ações: solicitar que seja adicionado ou removido um Pokémon do banco; requisitar a lista de Pokémons cadastrados até então; ou então solicitar que seja feita a troca de um Pokémon por outro que seja ofertado.

O servidor e o cliente trocam mensagens curtas de até 500 bytes utilizando o protocolo TCP. As mensagens carregam textos codificados segundo a tabela ASCII. Apenas letras, números e espaços podem ser transmitidos. Caracteres acentuados e especiais não devem ser transmitidos.

A seguir, estão descritas as ações que podem ser performadas, bem como a formatação de cada tipo de mensagem e a resposta desejada:

- **Adicionar Pokémon:** o cliente poderá solicitar a adição de um novo Pokémon encontrado. Isso deve ser feito através do comando **“add Pokémon”.** Após adicionado, o servidor deve retornar a resposta **"Pokémon added”**. Não deve ser possível adicionar novamente um Pokémon que já exista no banco. Caso isso ocorra, o servidor deve retornar uma mensagem **"Pokémon already exists”**. Deve ser possível adicionar somente 40 Pokémons no servidor, e caso o limite seja ferido, o servidor deve retornar a mensagem **“limit exceeded”.**
- **Remover Pokémon:** o cliente poderá solicitar a remoção de um Pokémon do banco. Essa ação deve ser feita através do comando **“remove Pokémon”**. Após removido, o servidor deve retornar a resposta **"Pokémon removed”**. Caso o cliente tente remover

um Pokémon que não exista, o servidor deve retornar a mensagem "**Pokémon does not exist”**.

- **Consultar Pokedex:** o cliente pode pedir ao servidor que envie a relação dos Pokémons que foram cadastrados até o momento. Para isso, deve usar o comando **“list”**. O servidor, então, retorna todos os Pokémons que já foram adicionados pelo cliente no formato **“Pokémon$_1$ Pokémon$_2$ Pokémon$_3$ … Pokemon$_n$”**. Caso a Pokédex esteja vazia, o servidor deve retornar **“none”**.
- **Trocar Pokémon:** o cliente também deve ser capaz de solicitar que seja realizada uma troca em sua Pokedex. Neste cenário, ele deve enviar o comando **“exchange Pokémon$_1$ Pokémon$_2$”,** indicando que deseja trocar o Pokémon$_1$, que já deve existir na Pokédex, pelo Pokémon$_2$. O servidor retorna, então, uma mensagem **“Pokémon$_1$ exchanged”**. Caso o cliente solicite a listagem da Pokédex após essa operação, o nome do Pokémon$_1$ deve ter sido trocado pelo nome do Pokémon$_2$.
    - Caso o cliente tente trocar um **Pokémon$_1$** que ainda não exista no banco, o servidor deve retornar a mensagem "**Pokémon does not exist”**.
    - Caso o cliente tente trocar por um **Pokémon$_2$** que já existe no banco, o servidor deve retornar a mensagem **“Pokémon already exists”**.

Exemplo Exchange:

```
> add charmander pikachu squirtle rattatta
< charmander added pikachu added squirtle added rattatta added
> exchange charmander vileplume
< charmander exchanged
> list
> vileplume pikachu squirtle rattatta
< exchange eevee hypno
> eevee does not exist
< exchange pikachu squirtle
> squirtle already exists
```

Exemplo 1:

```
> add charmander
< charmander added
> add charmander
< charmander already exists
> add squirtle
< squirtle added
> list
< charmander squirtle
> remove charmander
< charmander removed
> list
< squirtle
> remove charmander
```

```
< charmander does not exist
```

Exemplo 2:

```
> add pikachu raichu
< pikachu added raichu added
> exchange snorlax bulbasaur
< snorlax does not exist
> exchange pikachu bulbasaur
< pikachu exchanged
> list
< bulbasaur raichu
> add ho-oh
< invalid message
> add eevee bulbasaur
< eevee added bulbasaur already exists
```

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.
- Você deve ser capaz de **adicionar** mais de um Pokémon por vez com o comando **"**`add`**"**, lembrando que o fim da mensagem é identificado pelo `'\n'`. Os nomes devem ser separados por um espaço neste caso. Para efeitos de simplificação, considere adicionar no máximo 4 Pokémons por vez com este comando (sem necessidade de tratamento). Dica: faça o tratamento de strings com a biblioteca `<string.h>`.
    - Nesse caso , o servidor deve enviar todas as respostas em uma mesma linha, conforme mostrado nos exemplos acima.
- O servidor pode desconectar o cliente caso receba uma mensagem com um comando desconhecido (exemplo: **"**`ad`**"** em vez de **"**`add`**"**), mas não precisa retornar mensagem inválida.
- Os nomes de cada Pokémon devem ter no máximo 10 caracteres. Caso exceda, retorne um erro **"invalid message"**.
- Mantenha os comandos e os nomes dos Pokémons enviados nas mensagens com caracteres em **letra minúscula**. Os nomes dos pokémons podem possuir apenas **letras ou números**. Caso receba algo diferente disso, o servidor deve enviar uma mensagem de erro **"invalid message"**.
- Para Pokémons que possuam o caractere especial hífen "-" no nome, apenas desconsidere o caractere. Por exemplo, `Ho-oh` deve ser enviado como `hooh`. Caso contrário o servidor deve enviar uma mensagem de erro **"invalid message"**.
- 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.

IMPLEMENTAÇÃO

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

O **cliente** deve receber mensagens do teclado e imprimir as mensagens recebidas na tela. O **servidor** deve imprimir na saída padrão todas as mensagens recebidas dos clientes. **Não é necessário** que o 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 (Sugestão: utilize a porta 51511 para efeitos de padronização do trabalho). Seu cliente deve receber, **estritamente nessa ordem**, o endereço IP e a porta do servidor para estabelecimento da conexão.

A seguir, um exemplo de execução dos programas em dois terminais distintos:

IPv4:

```
no terminal 1: ./server v4 51511
no terminal 2: ./client 127.0.0.1 51511
```

IPv6:

```
no terminal 1: ./server v6 51511
no terminal 2: ./client ::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 40 Pokémons à Pokédex.
- Os nomes dos Pokémons devem ter no máximo 10 caracteres.

Materiais para Consulta:

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

AVALIAÇÃO

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve**

utilizar bibliotecas 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:

- Testar em IPv6 e IPv4 - o correto funcionamento de cada endereço corresponde a 50% da nota total do código
- Adicionar Pokémon - 20%
- Remover Pokémon - 10%
- Listar todos os Pokémons - 10%
- Realizar uma troca entre dois Pokémons - 20%
- Testar casos de mensagem inválida - 10%
- Cliente envia kill para o servidor e fecha a execução - 10%
- Recebimento de mensagens particionadas em múltiplas partes (mensagem recebida parcialmente no primeiro [recv]) - 20%
    - 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.

Obs.1: Considere para cada cenário acima todas as possibilidades possíveis (adicionar um Pokémon que já existe, remover um Pokémon que não existe, etc).
Obs.2: Não é necessário fazer tratamento para overflow de mensagens.

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. A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.

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, além da documentação, 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 "`client`" e o servidor em um binário chamado "`server`".
- Seu programa deve ser compilado ao se executar apenas o comando "`make`", ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP1_MATRICULA.zip
- O nome dos arquivos deve ser padronizado:
    - `server.c`
    - `client.c`
    - `common.c, common.h (se houver)`

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.