TP2 — Sistema de Mensagens Orientado a Eventos

Adler Melgaço e Ronald Davi

1 – Introdução

O trabalho consiste na implementação de troca de mensagens simples entre vários clientes e um servidor. O servidor é responsável pelo controle dessa troca, enquanto os clientes são responsáveis pela emissão e exibição das mensagens. Tanto o cliente quanto o servidor foram implementados através de sockets.

2 – Arquitetura

A arquitetura utilizada foi a orientada a eventos, auxiliada pela função *select*, que permite que os eventos sejam identificados e as funções de recebimento e emissão de mensagens possam ser utilizadas. Assim, é possível para o cliente saber exatamente se a entrada está sendo feita pelo teclado ou pelo servidor.

3 – Servidor

O servidor utiliza uma lista para guardar os sockets que estão registrados e aptos para serem lidos. Dessa maneira, ao identificá-los, o servidor pode tomar a ação necessária para cada um deles. Por exemplo, caso o socket identificado seja o próprio servidor, significa que ele deve aceitar um novo cliente, então ele identifica qual é esse cliente e, caso a conexão seja bem-sucedida, o adiciona na lista. Caso o socket identificado seja um cliente, o servidor deve identificar qual o tipo da mensagem que está sendo recebida e chamar o método correspondente.

Quando o servidor recebe uma mensagem do tipo “MSG”, primeiramente é verificado se o destino da mensagem é o mesmo de seu emissor, caso isso aconteça, uma mensagem do tipo “ERRO” é enviada e o procedimento é terminado. Caso contrário, o servidor procura qual o socket de destino e envia a mensagem quando ele é identificado, esperando por uma resposta do destinatário para propagá-la ao emissor.

Vale destacar uma decisão de implementação tomada, que foi a adição da opção de encerrar o servidor ao digitar *shutdown* no terminal, caso o usuário ache necessário. Quando esse comando é dado, o servidor envia uma mensagem customizada para cada um de seus clientes conectados, fazendo-os executarem o comando “FLW”, encerrando todas as conexões ativas com o servidor.

4 – Cliente

No programa *client.py,* o cliente também possui acesso à lista de sockets que devem ser lidos. Caso identifique que o socket que deve ser lido é o seu próprio socket, o cliente tenta realizar o recebimento da mensagem do servidor que, nessa situação, podem ser um de dois tipos: uma mensagem customizada do tipo “FLW” (extensão feita pelo grupo citada anteriormente) ou uma mensagem do tipo “MSG”. Ao receber a primeira, o cliente então inicia o procedimento de se desconectar, enquanto que na segunda o cliente faz a decodificação do texto, exibindo-o na tela do terminal, e respondendo ao servidor com uma mensagem do tipo “OK”, confirmando o recebimento da mensagem.

Se o socket identificado é o que corresponde a entrada do teclado, significa que o cliente deve se preparar para receber o comando que o usuário digitar e chamar o método necessário para fazer o tratamento adequado. Vale destacar que, além dos tipos de mensagem obrigatórios, as extensões também foram implementadas, sendo possível identificar os clientes não só por seu ID, mas também por seus apelidos, caso estejam previamente registrados. Os detalhes desse aspecto serão apresentados mais à frente.

5 – Discussão

Detalhes de implementação

Um dos aspectos mais importantes a serem destacados é em relação aos descritores utilizados e o uso da função *socket.fileno()*. Essa função retorna um inteiro que representa o descritor do arquivo, entretanto, devido a decisões de implementação do UNIX, os primeiros três números já estão reservados:

Dessa maneira, o primeiro descritor disponível para utilização é sempre o número 4. Como nos baseamos nesse valor para setar o ID do cliente e queríamos que esse ID começasse do número 1, foi implementado o esquema seguinte: para o recebimento de IDs de clientes, somamos 3 a variável *socket.fileno()*, enquanto que se é necessário enviar um ID para um cliente, subtraímos 3 nesse valor. Portanto, no lado do cliente, os IDs começam do número 1, enquanto que no lado do servidor, os mesmos IDs começam do número 4.

Dito isso, os aspectos que valem a pena serem destacados são:

- Quando um cliente é conectado, uma mensagem do tipo “OI” é enviada automaticamente, para permitir que ele já possa se comunicar e reconheça o seu ID.
- Se o cliente enviar uma mensagem do tipo “FLW”, sua conexão só é encerrada depois de receber um “OK” do servidor, caso essa mensagem de “OK” não chegue, o cliente envia uma mensagem de “ERRO”, e permanece conectado. Pelo lado do servidor, quando é recebida uma mensagem desse tipo, o cliente que a enviou é retirado da lista de clientes conectados e, caso tenha registrado um apelido, ele também é removido.
- Como o servidor não possui um IP específico, deve-se conectar em todas as interfaces da rede, por isso, o IP é 0.0.0.0.
- As mensagens de broadcast são enviadas a todos os clientes, exceto o seu emissor.

Execução

Para executar o programa *server.py*, o comando seguinte deve ser utilizado:

python3 server.py <porta>

Enquanto que no programa *client.py,* o comando é:

python3 client.py 0.0.0.0 <porta>

Aspectos extras

Como já mencionado, foi implementado uma maneira de encerrar o programa. Para isso, basta digitar *shutdown* no terminal que estiver rodando o programa *servidor.py*, desse

modo, cada um dos clientes serão desconectados e então o socket do servidor será fechado, encerrando o programa. Caso o usuário queira saber com mais detalhes o que ele pode fazer com o programa *client.py*, basta digitar *help,* que as suas opções serão apresentadas.

Além dessas duas características, o suporte ao uso de apelidos também foi feito, sendo possível associar a cada cliente um apelido criado pelo usuário. Dessa maneira, quatro novos tipos de mensagem foram adicionados:

- OIAP: Mesmo funcionamento da mensagem de tipo “OI”, mas com ela é possível registrar também o apelido que estará associado ao id do cliente.

  Ex: OIAP campeao (registra o cliente com o apelido “campeao”).

- CREQAP: Requisição de uma lista com os clientes conectados com ID e apelido correspondentes (somente serão contabilizados os clientes que possuem apelidos registrados).

- CLISTAP: Mensagem enviada pelo servidor que contém a lista com os clientes conectados, com ID e apelido correspondentes.

- MSGAP: Manda a mensagem se baseando no apelido do cliente, sendo trabalho do servidor identificar qual ID do cliente referenciado por esse apelido. Mesmo assim, após chegar no servidor, a mensagem será convertida em uma do tipo MSG para ser enviada ao seu cliente destinatário, já que é necessário ainda se comunicar com programas que não foram estendidos dessa forma, evitando comportamentos inesperados.

6 – Conclusão

O trabalho foi desenvolvido de maneira tranquila e se mostrou um bom aprendizado a respeito da comunicação entre sockets e uso da função select, muito utilizada em aplicações modernas de cliente-servidor, sendo possível adquirir um conhecimento prático mais aprofundado acerca dos conceitos aprendidos na disciplina. Além disso, a implementação

dos extras foi uma boa maneira de compreendermos e avaliarmos mais o relacionamento entre usuário e o programa, de forma a customizar e melhorar a sua experiência.