Proposta de Trabalho

Título: MyWhatsAPP

Sistemas Operativos

Pedro Sobral
pmsobral@ufp.edu.pt

André Ribeiro Pinto
arpinto@ufp.edu.pt

Alessandro Moreira
afmoreira@ufp.edu.pt

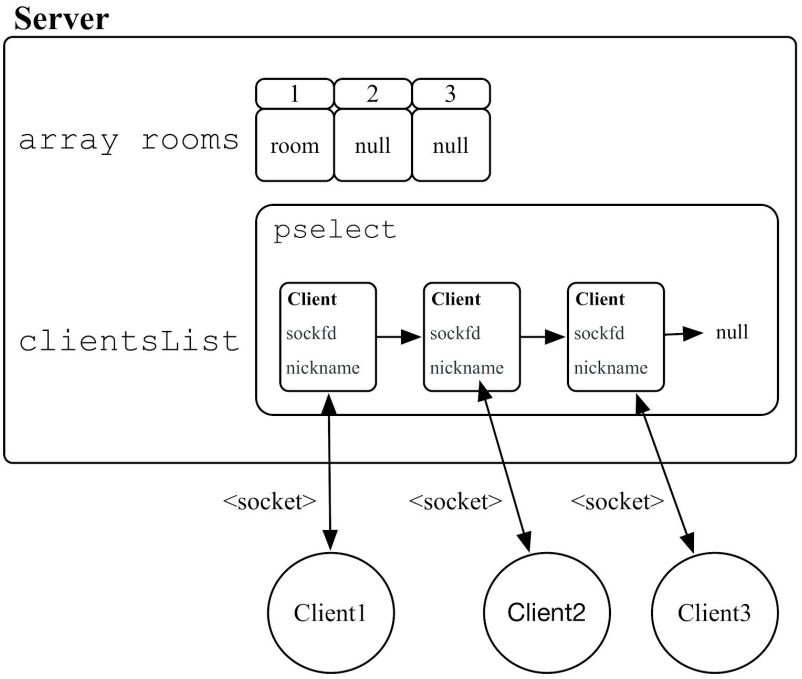
Março de 2019

Universidade Fernando Pessoa

Faculdade de Ciências e Tecnologias

Primeira Fase

Deve-se implementar uma versão do MyWhatsApp em que todos os clientes integram o mesmo grupo de conversação.



Servidor

Na primeira fase do projeto, o Servidor deverá gerir apenas uma sala de chat. Cada Cliente estabelece uma conexão com o Servidor, por meio de uma porta bem conhecida, de modo a ingressar na sala. O Servidor reconhece uma conexão inicial por meio do ***listen_sock*** nesta porta bem conhecida, que deverá estar vinculada à ***syscall* pselect()**[1]. Após esta conexão inicial, o Servidor deverá guardar o descriptor criado pela ***syscall accept*()** e guardá-lo numa estrutura de dados para gerir a comunicação com todos os Clientes, de modo a saber a que Cliente corresponde cada *socket*. Cada Cliente deverá ser identificado por um id, que será o seu *nickname* na sala. O Servidor é responsável pela gestão da fila de clientes, parsing das mensagens e encaminhamento das mesmas aos seus destinatários.

[1] Advanced Programming in the UNIX Environment Chapter 14.4

Toda comunicação entre Cliente e Servidor deve seguir um protocolo em formato JSON. Segue abaixo a estrutura inicial do protocolo a ser utilizado.

```
{
        " source_id": "user1",
        " destiny_id ": "null",
        " message_type ": "broadcast",
        "content": "Good afternoon, who's online?"
        "timestamp": "2019-02-21 15:20:47"

}
```

O campo "message_types" pode ter os seguintes valores:

- ***registration***: o cliente pede um registo no servidor. Indica ao servidor que o cliente quer entrar na sala e que está disponível para receber mensagens.
- ***broadcast***: indica ao servidor que a mensagem é destinada a todos os clientes.
- ***private***: indica ao servidor que a mensagem é destinada apenas ao cliente que possua o id mencionado no campo destiny_id.
- ***whoisonline***: O cliente que envia uma mensagem com o "*message_type"* whoisonline recebe de retorno uma lista dos Clientes que estão online.
- ***on***: Mensagem enviada a todos os clientes a indicar que um novo cliente está na sala.
- ***off:*** Mensagem enviada a todos os clientes a indicar que um cliente saiu da sala.

O *parsing* das mensagens trocadas deverá ser feito com recurso à biblioteca jsmn[2]. Será fornecido um código de exemplo de como converter uma *string* JSON para uma *struct* C.

Cliente

O Cliente conecta-se ao servidor através da porta bem conhecida do servidor.

O Cliente deverá usar a ***syscall pselect*()**, de modo a poder ler dados com origem no teclado e pelo Servidor.

O Cliente pode lançar um novo cliente (processo), cujas mensagem enviadas por esse novo processo têm um destino pré-definido e o tipo de mensagem é "private".

O Cliente deve ter uma interface gráfica a ser desenvolvida com recurso à biblioteca **GTK+**[3][4].

[2] https://github.com/zserge/jsmn
[3] https://developer.gnome.org/gtk2/stable/gtk-getting-started.html
[4] https://developer.gnome.org/gtk-tutorial/stable/

Numa primeira fase a interface gráfica poderá ser substituída por uma interface textual que permita realizar todos os testes necessários.

Primeira fase - Single Chat

Requisitos e cotações:

R1: Criar uma estrutura de dados para mensagens e utilizadores no servidor. (3 val)

R2: Implementar um algoritmo de parsing de mensagens (JSON). (2 val)

Nota: O Servidor e Clientes devem conseguir fazer o parsing de mensagens JSON de maneira a conseguir interpretar cada um dos seus campos.

R2.1 Parsing de mensagens no servidor.

R3: O cliente deve conseguir enviar mensagens a todos os utilizadores da sala (inclusive ele). (6 val)

Nota: As mensagens são enviadas para o servidor. O Servidor na posse das mensagens faz o parsing das mesmas e encaminha as mensagens aos destinatários.

R3.1 O servidor deve gerir a fila de clientes, adicionando e removendo os cliente.

R3.2 O servidor deve tratar as mensagem e encaminhar das mensagens aos clientes.

R4: O cliente deve conseguir pedir ao servidor uma lista de utilizadores presentes na sala. (2 val)

R5: O cliente deve conseguir enviar mensagens para apenas um destinatário. (2 val)

R6: Todos os participantes da sala devem ser notificados sempre que um cliente entre ou saia. (2 val)

R7: Os cliente deve ter uma janela gráfica que suporte os requisitos anteriores. (3 Val)

Segunda Fase

Deve-se implementar uma versão melhorada do MyWhatsApp em que pode haver mais do que um grupo de conversação.

Servidor

Na segunda fase do projeto, o Servidor deverá gerir múltiplas salas de chat. Cada cliente continua com apenas um socket conectado ao servidor, mas este deverá gerir uma lista de salas que cada uma delas possui uma lista de clientes.
As mensagens enviadas pelos clientes apenas podem ter como destino os clientes presentes na sala actual.

O protocolo de mensagens deverá ser estendido para suportar as seguintes operações:

- **create**: cria uma nova sala e o criador passa a ser o seu moderador
- **join**: ingressa numa sala existente
- **leave**: deixa a sala que estava presente
- **show_rooms**: lista todas as salas existentes
- **show_users**: lista todos os clientes presentes numa sala
- **ban**: exclui um cliente da sala (apenas o moderador)

O moderador de uma sala deverá ser sempre o utilizador mais antigo (inicialmente o criador).

As interações dos clientes com o servidor (conexões e mensagens) deverão ser tratadas por tarefas criadas pela main thread do servidor. Deverão ser criadas novas estruturas para suportar estas modificações.

Fase 2.1

Na fase 2.1 o servidor atende novas conexões e gere a estrutura de salas de maneira a suportar toda as funcionalidades descritas.

O servidor (*main_thread*) quando se apercebe de uma nova conexão ou, cria um novo *thread* com capacidade de aceitar essa conexão e leva o cliente para a sala solicitada. Depois de entregue o cliente a thread responsável termina.

Quando os clientes enviam mensagens, o servidor deve criar uma nova *thread* para atender esse pedido. A *thread* deve ter capacidade de interpretação mensagens. Depois da mensagem ser interpretada e atendida a *thread* deve terminar.

Fase 2.2

Na fase 2.2 as threads devem ser sincronizadas com base no esquema do Produtor/Consumidor. Nesta fase o servidor cria **M** produtores de mensagens e **N** consumidores de mensagens. O servidor quando se apercebe que há uma interação, notifica os produtores que existem mensagens para produzir. Os produtores são responsáveis por ler a mensagem com origem nos clientes e produzem essa mesma mensagem a ser consumida mais tarde. Os consumidores consomem a mensagem e são responsáveis por tratar essa mesma mensagem.

Cliente

O Cliente deve ter uma interface gráfica a ser desenvolvida com recurso à biblioteca **GTK+**[5][6], que suporte todas as funcionalidades permitidas ao cliente.

[5] https://developer.gnome.org/gtk2/stable/gtk-getting-started.html
[6] https://developer.gnome.org/gtk-tutorial/stable/

Fase 2 - Multi-room chat

Requisitos e cotações:

Fase 2.1

R1: Criar estrutura de dados para múltiplas salas. (2 Val)

R2: Estender o protocolo para suportar: (3 Val)
- **R2.1:** Criação, entrada e saída de sala
- **R2.2:** Listagem de salas e de clientes
- **R2.3:** Exclusão de clientes de uma sala (pelo criador da sala)

R3: Utilização de tarefas para o tratamento das interações com os clientes (novas conexões e mensagens). (4 Val)

Nota: Deverão criar novas estruturas para suportar a utilização de tarefas.

R4: Os clientes devem possuir uma GUI utilizando a biblioteca GTK+. (2 Val)

FASE 2.2

R5: Todas as interações do servidor com os clientes deverão ser sincronizados pelo método do Produtor/Consumidor (M Produtor/N Consumidores). (6 Val)

R6: As salas devem ter um limite de 5 clientes, caso este número seja excedido, o cliente deve ser notificado desta situação (ver sigmask do pselect*). (3 Val)

*UNIX Network Programming, Volume 1, Third Edition, The Sockets Networking API Chapter 6.9