Programa ‘chat’

Docente: Carlos Baquero Moreno

Índice

Constituição do Grupo

	Augusto Mota (a76563) a76563@halunos.uminho.pt
	Hugo Machado (a80362) a80362@alunos.uminho.pt

Introdução

Este projeto tem como objetivo principal aplicar dos conhecimentos adquiridos na disciplina de Sistemas distribuídos, tais como programação concorrente, programação com *sockets* e controlo de concorrência.

O tema do projeto é um simples chat entre múltiplos clientes, onde cada cliente é identificado através de um *nickname*.

A comunicação é feita através do protocolo TCP, e dentro do programa é orientada á linha.

No lado do cliente, foi sugerido pelo docente a utilização do comando *NetCat.*

Resolução do problema

O primeiro desafio consistiu em fazer um servidor sempre á escuta de novas ligações e por cada ligação que recebia, iniciar um novo *thread* responsável pela ligação.

Apos ultrapassar este desafio, a solução para o problema seguinte foi ter dois *thread*'s ativos por cada ligação, um para ler o que era escrito pelo cliente e outro para escrever para o cliente. E desta forma, com uma variável partilhada, fazer com que existisse um "*broadcast*" de uma mensagem escrita por um cliente para todos os outros conectado ao mesmo servidor.

Com esta fase inicial minimamente funcional decidi avançar para o controlo de concorrência do programa, este baseia se existir um bloqueio quando alguma coisa é escrita na variável partilhada, e quando é terminada a escrita, lançar um aviso para os *thread*'s de escrita, para eles escreverem o conteúdo da variável partilhada.

E assim o principal desafio do projeto foi completado.

Para estabelecer o controlo de concorrência usei *lock*'s sempre que uma mensagem é escrita na variável partilhada, e usei os métodos *notifyAll* e *wait* para os avisos e esperas.

Classes do programa

Class Control:

Nesta classe mantemos um *lynkedHashMap* que mantem a ordem das mensagens, e como chave o utilizador. Usamos este *HashMap* como a variável partilhada de escrita porque a partir daqui conseguimos identificar o remetente da mensagem e também o numero de clientes ativos, sempre que um cliente abandona o chat a sua entrada do *HashMap* é eliminada.

É nesta classe também onde efetuamos o controlo de concorrência, com métodos personalizados, através de *locks()*(do tipo *ReentrantLock*), *Synchronized* e variáveis de condição(*wait()*,*notifyAll()*).

Class User:

Nesta classe apenas ligamos um id numéricos a um *nickname* para fazer a identificação do remetente.

Class ClientWriter:

Nesta classe é onde reside o segundo *thread* de cada ligação, que trata da escrita da mensagem.

Class Client:

É nesta classe onde a *thread* principal da ligação lê o que se encontra escrito no terminal do cliente e coloca na variável partilhada. É também aqui onde o criado o segundo *thread* e avisado o fim de sessão, sempre que um cliente se junta ou sai do chat.

Class serverDist:

Aqui mantemos a classe principal onde inicializamos uma *ServerSocket()* que esta constantemente á espera de novas ligações, é onde o objeto *Control* é inicializado.

Funcionamento do Programa

Inicialmente no nosso programa inicializamos uma *ServerSocket*() na porta 9999, para escutar essa porta, inicializamos também o objeto *Control* e um contador id(*Integer*) a zero. De seguida entramos num *while* infinito á espera de novas ligações através do método *accept*() e sempre que é recebida uma nova ligação e inicializamos um *thread* com o objeto *Client*(classe que implementa a interface *Runnable*) e fazemos *start*().

Depois na classe *Client* é inicializado os objetos *BufferedReader()* e *PrintWriter()*, para *input* e *output*, e chamamos o método *newUser()*. No método *newUser()* é questionado ao cliente qual o seu *nickname* e de seguida existe uma confirmação desse mesmo, após essa confirmação é chamada o método *sendS()*, que consiste em enviar as *strings* para a variável partilhada. No método *sendS()* em primeiro lugar é pedido *Lock()* e de seguida é verificado se a mensagem que vamos enviar é de despedida, se sim é chamado o método *removeUser()* que elimina o cliente do *HashMap*, se for de inicio de sessão é enviada a mensagem a avisar que o cliente juntou-se, se não for nenhuma dessas é porque é uma mensagem normal. Após a escrita da mensagem chamamos o método *notify()* que corresponde ao método *notifyAll()* na classe *Control*, por fim é feito o *Unlock()* que liberta o *lock* para que outra *thread* possa escrever.

Após o aviso de novo cliente o *thread* é executado.

Este novo *thread* é executado na classe *ClientWriter*(que implementa a interface *Runnable*), onde declara uma variável do tipo *Map.Entry* que corresponde a um iterador de *HashMap*, de seguida entra no ciclo *while* condicionado por uma variável partilhada entre os dois *threads* da mesma ligação(variável que avisa o *thread* para sair quando a ligação termina), e entra noutro *while* que prende o *thread* até que exista uma alteração no *HashMap*, neste ciclo também chamamos o método *Wait()*. Quando o *thread* é avisado de uma nova mensagem ele sai deste ciclo *while* entra no método *barrier()*(método de barreira desenvolvido nas aulas) para que os *threads* existentes consigam sair todos do primeiro ciclo, de seguida é extraída através do método *getMsg()* a ultima entrada do *HashMap* e verificada se o remetente da mensagem é o mesmo que o utilizador correspondente a essa *thread*, se sim a mensagem não é enviada, se não a mensagem é enviada.

Por fim, quando um cliente decide sair do chat é gerada automaticamente uma mensagem que avisa os outros clientes a saída desse utilizador.

Conclusão

No âmbito da Unidade Curricular de Sistemas Distribuídos foi-nos proposto a realização de um projeto que consta no desenvolvimento de um servidor para um chat entre clientes que liguem ao servidor.

A realização deste projeto tornou-se vários níveis um grande desafio ao qual o grupo se demonstrou á altura.

Quanto mais progredimos neste projeto mais aprendemos, visto que é sempre necessário realizar uma grande pesquisa e adotar metodologias nas diversas áreas e melhorar as nossas capacidades utilizadas para o mercado de trabalho.

Este procedimento tem custos, pois despendemos algumas horas em pesquisa, estudo de metodologias de trabalho e também o estudo de alguns guiões realizados na aula. Embora seja um processo exigente, também é muito compensador em termos de conhecimento.