

**Relatório de Segurança e Confiabilidade**

Trabalho 1

Engenharia Informática

Grupo 51

João David n49448

João Marques n49038

Luís Moreira n49531

# Objectivos concretizados

# Organização do Software

Na implementação do projecto, foram criados dois programas, o MsgFile (cliente) que recorre a uma linha de comandos para comunicar com o outro programa, o MsgFileServer (servidor). Para estabelecer um protocolo de comunicação, foi criada uma class Message que “carrega” o conteúdo das mensagens pela rede (explicada em maior detalhe mais à frente). De forma a organizar todas as classes, foram criados três packages, o package ‘client’ que contem as classes que implementam o MsgFile (inclusive), o package ‘server’ que contem todas as classes que implementam o MsgFileServer(inclusive), nestes dois packages, as classes contidas não “conhecem” outras classes fora do seu package, excepto as classes do terceiro package, ‘communication’, que contém a class Message referida anteriormente e o enumerado OpCode que define os códigos das operações.

Package client

* Client
* ClientConst
* MsgFile

O ‘MsgFile’ corresponde ao programa cliente, o objecto ‘Client’ é responsável por comunicar com o servidor, após o utilizador se ter conectado com sucesso é disponibilizado a linha de comandos, o parsing é feito pelo ‘MsgFile’. No ‘ClientConst’ temos os caminhos dos directórios definidos para guardar os ficheiros transferidos e mensagens colectadas do servidor.

Package communication

* Message
* OpCode

O ‘Message’ corresponde ao objecto que é enviado pela rede entre o cliente e o servidor, sempre que é enviada uma mensagem, um dos atributos de Message é sempre inicializado com um ‘OpCode’, trata-se de um enumerável que permite ao receptor da mensagem saber que tipo de dados estão disponíveis na mensagem, e no caso do servidor saber que operação o cliente está a tentar executar.

Package server

* ConcurrentManager
* Manager
* MsgFileServer
* ServerConst
* Skel

A classe ConcurrentManager é responsável por gerir os semáforos que permitem fazer exclusão mútua de ficheiros que podem ser acedidos em simultâneo por dois utilizadores conectados ao servidor. Na class Manager estão os métodos responsáveis por concretizar as operações requisitadas pelo cliente (escrever,ler,criar ficheiros). O MsgFileServer é o programa servidor, sempre que um cliente se conecta, é criado um thread para satisfazer os pedidos desse cliente. O ServerConst possui constantes com os nomes dos directórios definidos para guardar os ficheiros no lado do servidor. O Skel implementa a função invoke, que ao receber uma Message do cliente, com base no OpCode que vem lá dentro, chama a função respectiva implementada pelo ‘Manager’, com base no sucesso ou insucesso dessa operação, o invoke constrói uma nova Message de resposta para o cliente com o OpCode respectivo.

# Formato das mensagens

Para concretizar o envio de mensagens foi criada uma class objecto ‘Message’ com diversos atributos que são usados dependendo do propósito da mensagem, para definir que conteúdo a mensagem contêm efectivamente foi criado um enum ‘OpCode’, sempre que uma mensagem é criada é definido um OpCode para que o receptor da mensagem saiba que atributos da mensagem foram inicializados, para que os possa aceder e concretizar a operação requerida, em certos casos a mensagem apenas contêm o OpCode inicializado visto não ser necessário mais informação para que seja feita a operação, por exemplo a operação “list” vinda do cliente para o servidor apenas leva o opCode com ‘LIST\_FILES’.

O OpCode também tem como função na resposta do servidor para o cliente definir se as operações foram bem concretizadas, ou em caso de erro informar o cliente o que aconteceu, por exemplo, ERR\_NOT\_REGISTERED, quando o utilizador introduzido como input numa operação não esteja registado no servidor.

Atributos de Message

* OpCode opCode
* OpCode[] arrCode
* String str
* String[] arrStr
* ArrayList<String> arrListStr
* ArrayList<Byte[]> arrListArrByte
* Byte[] arrByte

Operation Codes (OpCode)

* STORE\_FILES, LIST\_FILES, REMOVE\_FILES, USERS, TRUST\_USERS, UNTRUST\_USERS, DOWNLOAD\_FILE, SEND\_MSG, COLLECT\_MSG, END\_CONNECTION
* OP\_SUCCESSFUL: operação sobre no servidor bem-sucedida
* OP\_RES\_ARRAY: os resultados da operação vêm num array para cada parâmetro enviado para o servidor respectivamente, ex:”trusted fernando carlos rita” retorna um vetor de OpCode com length 3
* OP\_ERROR: operação no servidor não foi bem-sucedida, exception por exemplo
* ERR\_NOT\_REGISTERED: user enviado como parâmetro não está registado no servidor
* ERR\_ALREADY\_EXISTS: parâmetro introduzido já existe, por exemplo fazer “trusted fernando”, quando o fernando já tinha sido trusted anteriormente
* ERR\_NOT\_TRUSTED: o utilizador local não está na lista trusted do utilizador enviado como parâmetro
* ERR\_YOURSELF: utilizador enviado como parâmetro é o utilizador local
* ERR\_NOT\_FOUND: parâmetro enviado não encontrado, por exemplo, ao fazer download de um ficheiro inexistente, ou fazer untrusted de um utilizador que não foi trusted anteriormente

|  |  |  |
| --- | --- | --- |
| **Comando Utilizador** | **Mensagem de pedido** | **Mensagem de resposta** |
| store <files> | STORE\_FILES <arrListStr> <arrListArrBytes> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| list | LIST\_FILES | OP\_SUCCESSFUL <arrStr> |
|  |  | OP\_ERROR |
| remove <files> | REMOVE\_FILES <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  |  |
| users | USERS | OP\_SUCCESSFUL <arrStr> |
|  |  | OP\_ERROR |
| trusted <trustedUserIDs> | TRUST\_USERS <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| untrusted <untrustedUserIDs> | UNTRUST\_USERS <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| download <userID> <file> | DOWNLOAD\_FILE <arrStr> | OP\_SUCCESSFUL <byteArray> |
|  | [0]=owner, [1]=fileName | ERR\_YOURSELF or ERR\_NOT\_FOUND or  ERR\_NOT\_TRUSTED or ERR\_NOT\_REGISTERED |
| msg <userID> <msg> | SEND\_MSG <arrStr> | OP\_SUCCESSFUL |
|  | [0]=receiver, [1]=textMsg | OP\_ERROR or ERR\_NOT\_TRUSTED  or ERR\_NOT\_REGISTERED |
| collect | COLLECT\_MSG <str> | OP\_SUCCESSFUL <arrListStr> |
|  |  | OP\_ERROR |
| exit | END\_CONNECTION | OP\_SUCCESSFUL |
|  |  |  |

A seguinte tabela apresenta os comandos e respectivas sintaxes, que os clientes podem requerer e a que o servidor pode responder.

Nas mensagens de pedido, a segunda linha corresponde à informação colocada no índex do array, enquanto que nas mensagens de resposta corresponde à mensagem de erro (a primeira linha corresponde à mensagem de resposta quando não há erro).

# Segurança

Para impedir múltiplos logins no servidor pelo mesmo utilizador, foi realizado o seu bloqueio, ou seja, apenas pode permanecer conectado ao servidor um utilizador para aquele “user name”.

O username e password são enviados pela rede como objectos String, isto é muito inseguro, alguém que consiga interceptar os pacotes enviados do cliente para o servidor quando faz login, consegue roubar os dados de acesso da conta.

APONTAR MAIS INSEGURANÇAS DO NOSSO PROGRAMA

# Configuração da sandbox

server.policy – Ficheiro de policy do servidor

As seguintes permissões permitem ao programa ler e escrever nas directorias que contêm os utilizadores registados no servidor, e os seus respectivos ficheiros e mensagens

permission java.io.FilePermission "${user.home}/MsgFileG51", "read,write";

permission java.io.FilePermission "${user.home}/MsgFileG51/server", "read,write";

Esta permissão também tem o parâmetro de delete, de forma a que os clientes ao usarem o comando remove, o servidor consiga apagar o respectivo ficheiro

permission java.io.FilePermission "${user.home}/MsgFileG51/server/-", "read,write,delete";

Corresponde ao endereço e porto que o servidor usa para estar à escuta de clientes que se queiram conectar, neste caso escolhemos o porto 23456, e o endereço de ip local

permission java.net.SocketPermission "127.0.0.1:23456", "listen,resolve";

Corresponde à permissão para aceitar uma conecção de clientes, tal como na permissão anterior, usa o endereço de ip local, e usa todas as portas a partir da 1024, corresponde as portas de aceitação de conecção

permission java.net.SocketPermission "127.0.0.1:1024-", "accept,resolve";

Como se pretende que o programa de servidor funcione em todos os sistemas operativos que suportem JVM, foram utilizadas propriedades do java para obter a home directory e o line separator, se isto não fosse utilizado, e fosse hardcoded, poderia ocorrer que em certos sistemas operativos, não fosse possível criar os directórios essências ao bom funcionamento do programa

permission java.util.PropertyPermission "user.home", "read";

permission java.util.PropertyPermission "line.separator", "read";

client.policy – Ficheiro de policy do cliente

As seguintes permissões permitem ao programa ler e escrever nas directorias que contêm os downloads efectuados do servidor e mensagens que foram colectadas

permission java.io.FilePermission "${user.home}/MsgFileG51", "read,write";

permission java.io.FilePermission "${user.home}/MsgFileG51/client", "read,write";

Foi também dada a permissão de delete, visto que quando um utilizador faz download de um ficheiro do servidor com o mesmo nome de um outro ficheiro que já se encontre na sua directoria de download, este ficheiro é substituído pelo novo ficheiro acabado de transferir do servidor

permission java.io.FilePermission "${user.home}/MsgFileG51/client/-", "read,write,delete";

A aplicação cliente apenas tem permissão para se conectar ao ip correspondente à máquina local, e ao porto 23456, de igual forma ao que foi definido no servidor

permission java.net.SocketPermission "127.0.0.1:23456", "connect,resolve";

Analogamente ao que já foi explicado anteriormente no server.policy, estas permissões permitem que o programa funcione em diferentes sistemas operativos

permission java.util.PropertyPermission "user.home", "read";

permission java.util.PropertyPermission "line.separator", "read";

De forma a que o cliente possa fazer store no servidor de qualquer ficheiro na sua máquina, é necessário dar permissão de leitura a todos os ficheiros

permission java.io.FilePermission "<<ALL FILES>>", "read";