Universidade de Coimbra
Faculdade de Ciências e Tecnologias
Departamento de Engenharia Informática

Proxytel

Introdução às Redes e Serviços de Comunicação
2000/01

Relatório Técnico

Paulo José dos Santos Guilhoto
Luís Filipe Lopes Rosa Dinis

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia da
Universidade de Coimbra

Relatório Técnico do Proxytel

Trabalho realizado para a disciplina de
Introdução às Redes e Serviços de Comunicação
no âmbito da
Licenciatura da Engenharia Informática

Paulo José dos Santos Guilhoto
guilhoto@student.dei.uc.pt - N.º 985011444

Luís Filipe Lopes Rosa Dinis
ldinis@student.dei.uc.pt - N.º 985011476

Dezembro de 2000

Conteúdo

Introdução

Nas organizações ligadas à Internet, é cada vez mais frequente a utilização de sistemas de firewall como garante da segurança dos sistemas informáticos, face às ameaças vindas do exterior do organização.

É neste ambiente que se insere o Proxytel. Com o objectivo de filtrar as comunicações com o exterior da rede e as entradas na rede interna, são utilizados como intermediários servidores de comunicações e *proxys*, sendo estes dedicados a restringir determinado serviço.

O Proxytel é mais um destes *proxys* que se encontra no seio da *firewall*, com o intuito de autenticar as entradas e/ou saídas de uma rede protegida exclusivamente para aplicações de Telnet. Sendo assim, era pedido que fosse construído um servidor que, quando colocado a funcionar numa máquina, obrigasse à autenticação dos utilizadores para estes poderem entrar ou sair da rede interna.

Especificação interna

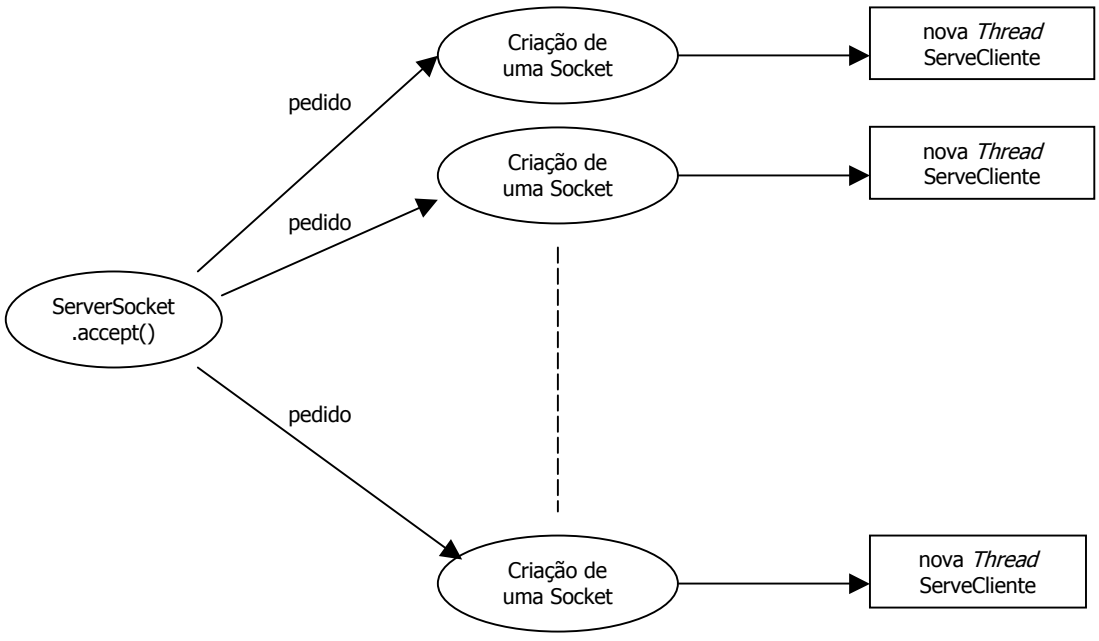
O servidor de Proxytel começa por ser criado na classe *Proxytel* através de uma *ServerSocket* num determinado porto da máquina. Esse porto é especificado aquando da execução do servidor através da adição do parâmetro *–p*. Segue um exemplo:

```
java Proxytel –p 5959
```

De seguida, o servidor fica continuamente à espera de ligações de clientes à máquina onde está a correr e ao respectivo porto. Este mecanismo é posto em prática através da função *ServerSocket.accept()* . De cada vez que chega um cliente, é criada uma nova *Socket* para estabelecer a comunicação directa entre o servidor Proxytel e este cliente, dando assim a noção que existe um canal físico directo de comunicação entre as duas máquinas, o equivalente a um *pipe* bidireccional.

A *Socket* criada é então enviada para a classe *ServeCliente,* que é uma *Thread*, onde irá ser feita a autenticação dos utilizadores e criação de um ambiente onde este poderá aceder a determinadas funções do sistema. Através da criação de uma *Thread* para cada ligação ao servidor, possibilita-se assim o atendimento a vários clientes em simultâneo.

```
while(true) {
  //Bloqueia até haver uma ligação
  Socket socket = s.accept();
  try {
    new ServeCliente(socket,admin,files);
  } catch(IOException e) {
      System.out.println("Erro na criacao do ServeCliente: "+e);
     socket.close();
  }
```



Ao entrar na classe *ServeCliente,* a primeira coisa a fazer é desactivar o *echo* local da máquina origem. Optou-se por proceder à desactivação do *echo* local e ao consequentemente uso do *echo* gerado pelo Proxytel por motivos de força maior (neste caso, de sistemas operativos). De facto, a aplicação de Telnet do sistema operativo Windows recusa-se a negociar os parâmetros de *echo,* obrigando assim ao uso incondicional do *echo* gerado pelo servidor Proxytel. Para a desactivação do *echo* local, utilizou-se a sequência *IAC WILL ECHO*, indicando assim que seria o servidor Proxytel a fazer *echo*. Também se procede à indicação de que o cliente deverá permitir a visualização dos caracteres à medida que os recebe do *echo* do servidor Proxytel, e não somente quando é recebido o caractere *CR*. Para isso, usou-se a sequência *SUPPRESS GO AHEAD*.

```
private void desactiva_echo_local() {
  char c;
  try{
    //IAC WILL ECHO
    writeChar((char)255);
    writeChar((char)251);
    writeChar((char)1);
    c = (char)in.read();
    c = (char)in.read();
    c = (char)in.read();
    //SUPPRESS GO AHEAD
    writeChar((char)255);
    writeChar((char)251);
    writeChar((char)3);
    c = (char)in.read();
    c = (char)in.read();
    c = (char)in.read();
  }
  catch(IOException e){}
}
```

Verifica-se então se o cliente faz parte da lista de máquinas origem autorizadas, contida no ficheiro *clientes.aut.* Tal como para o caso das máquinas destino autorizadas (que iremos ver mais à frente), a lista de máquinas cliente é uma estrutura do tipo *Vector,* ou seja uma estrutura dinâmica que, para além de permitir pesquisas e eliminações rápidas, cresce à medida que lhe vamos adicionando itens. Este *Vector* é lido do ficheiro apenas quando se inicia o servidor e é guardado em ficheiro sempre que lhe são feitas alterações, quer ao nível da inserção de novas máquinas origem, quer ao nível da sua eliminação.

Depois de validada a origem da ligação, chega-se então à parte da identificação do utilizador. A lista de utilizadores está organizada sobre a forma de uma *Hashtable*, permitindo assim pesquisas muito rápidas consoante o valor de uma chave, que neste caso é o login do utilizador. Sendo assim, basta carregar a *Hashtable* de utilizadores que está guardada no ficheiro *utilizadores.aut* apenas aquando da iniciação do servidor Proxytel. Resta apenas agora esclarecer como foi feito o processo de encriptação da password do utilizador. O algoritmo de codificação utilizado foi o *SHA*, um dos vários algoritmos disponibilizados pela Sun Microsystems na package *java.security*. Por isso, não irá aqui ser descrito qual o funcionamento do algoritmo, mas sim como é que este foi utilizado para responder às existências deste projecto. Apenas iremos salientar como é que é criado e inicializado o algoritmo de compressão e como é feita a encriptação de uma *String* password.

```
try{
  //cria uma nova instância do algoritmo de compressão SHA
  sha = MessageDigest.getInstance("SHA");
} catch(NoSuchAlgorithmException e){
    System.out.println("Erro no algoritmo de compressão");
}
```

```
//encripta a password ...
sha.update(password.getBytes());
//...e coloca o resultado num array de bytes
byte[] encrypt = sha.digest();
```

Mais acima, referimos que a *Hashtable* tinha como chave o login do utilizador. Basta apenas referir que o elemento associado a cada chave é a respectiva password do utilizador já encriptada. Porém, uma das características do algoritmo de encriptação utilizado é que não é possível obter a password (como ela era inicialmente) depois de esta estar encriptada. Sendo assim, uma forma de comparar a password da *Hashtable* com a password introduzida aquando do login, é simplesmente encriptando a password recebida no login e compará-la byte a byte com a password da *hashtable.*

```
public synchronized boolean identifica_user(String login, String password){
  sha.reset();
  //vai buscar a password da Hashtable correspondendo a este utilizador
  byte[] pass_read = (byte[]) hash.get(login);
  //encripta a password passada como parâmetro para a poder comparar com a
  // password já encriptada existente na Hashtable
  sha.update(password.getBytes());
  byte[] encrypt = sha.digest();
  if ((pass_read==null) || (!comparaPass(pass_read,encrypt)))
    return false;
  return true;
}

public boolean comparaPass(byte[] pass1, byte[] pass2){
  if (pass1.length != pass2.length)
  return false;
  for (int i=0;i<pass1.length;i++){
    if (pass1[i] != pass2[i])
      return false;
  }
  return true;
}
```

Falta-nos agora referir como funciona o processo de leitura de uma linha da *Socket* de comunicação entre o servidor Proxytel e o cliente. A leitura do terminal da máquina origem processa-se caractere a caractere (por causa do servidor ter que fazer o *echo*). Depois basta apenas verificar se o caractere lido é ou não um caractere dito 'especial'. Isto é, se for pressionada a tecla *Backspace* (caractere 8 no sistema operativo Windows e 127 no sistema operativo Unix), deveremos voltar ao caractere anterior, inserir um caractere *Space* e deslocar de novo o cursor para a esquerda. Todo este processo está de acordo com o especificado nos RFC 's e está retratado na função que se segue:

```
private String readLine(){
  char c, b;
  String str = new String ();

  try{
    do{
      c = (char) in.read();
      if (echoo==1){
        if (((int)c == 8) || ((int)c == 127)){     //backspace
          if (str.length()!=0){
            writeChar((char)8);
            writeChar((char)32);
            writeChar((char)8);
          }
        }
        else{
```

```
          if (((int)c != 13) && ((int)c != 10))
            writeChar(c);
          else{
             if ((int)c == 13){
               b = (char) in.read();
               writeChar((char)13);
               writeChar((char)10);
             }
          }
        }
        else   //sem echo
          if ((int)c == 13)
            b = (char) in.read();

      if (((int)c != 10) && ((int)c != 13) && ((int)c != 8) && ((int)c != 127))
        str = str + c;
      if ((((int)c == 8) || ((int)c == 127)) && (str.length()!=0))
        str = str.substring(0, str.length()-1);
    }while (((int)c!=13) && ((int)c!=10));
  } catch(IOException e){
      System.out.println("Erro ao ler caracteres do cliente");
  }
    return(str);
  }
```

Apenas é necessário lembrar que cada linha lida termina com os caracteres *CR LF*, o que quer dizer que, a partir do momento em que se lê o caractere *CR*, a linha está completa. Mas, é necessário ler o próximo caractere *LF* de forma a não armazenar lixo no buffer para a leitura da próxima linha.

As funções que têm aparecido no meio das linhas de código *writeChars()* e *writeChar()* apenas colocam os dados na stream de saída da *Socket.*

```
private void writeChars(String s){
try{
   out.write(s,0,s.length());
   out.flush();
  } catch(IOException e) {
      System.err.println("IO Exception");
  }
}

private void writeChar(char c){
try{
   out.write(c);
   out.flush();
  } catch(IOException e) {
      System.err.println("IO Exception");
  }
}
```

Voltemos então à parte da validação do utilizador. Depois de feita, o utilizador tem acesso a comandos do sistema, que não interessam para já referir. Salienta-se apenas que o administrador do Proxytel (utilizador *root*) tem acesso a um conjunto muito mais vasto de comandos, permitindo-lhe assim configurar o sistema em termos de utilizadores, máquinas origem e máquinas destino das ligações.

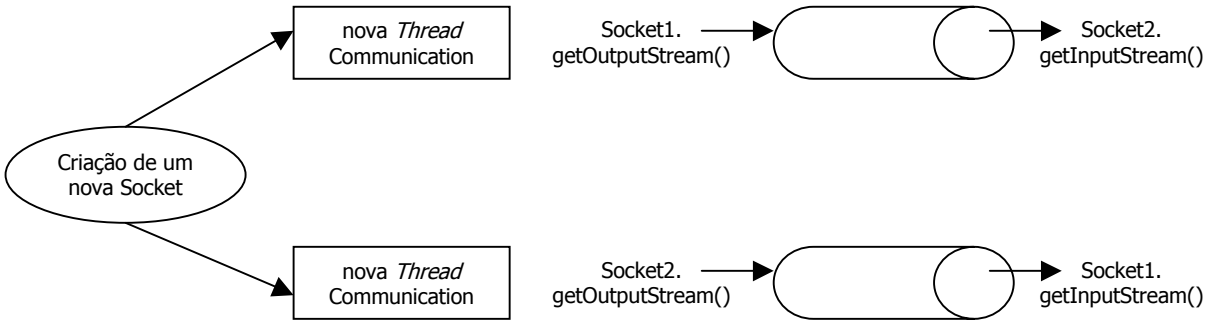
Vamos centrar-nos no comando mais importante: *connect* (ou simplesmente *c*). Através deste comando, é permitido estabelecer uma ligação com uma máquina remota e concatenar essa ligação com a ligação que já existia com a máquina cliente.

A primeira coisa a fazer é verificar se a máquina destino escolhida existe. Para isso, basta transformar o nome da máquina num endereço único, resultado da combinação do nome da máquina e do seu IP, e verificar se disparou alguma excepção.

```
try {
  addr = InetAddress.getByName(machine);
} catch(java.net.UnknownHostException e){
  writeChars("   host unknown\n\r");
    host_unknown = true;
}
```

Depois de ter a certeza que a máquina existe, é necessário verificar se a máquina destino faz parte da lista de máquinas remotas autorizadas. O processo é exactamente o mesmo que sucedeu anteriormente aquando da verificação da máquina cliente. De novo, temos uma estrutura *Vector* que vai conter os endereços de todas as máquinas destino autorizadas. Essa estrutura é guardada no ficheiro *remotos.aut* sempre que é feita alguma alteração na lista de máquinas pelo administrador e apenas é lida aquando da inicialização do servidor Proxytel.

Se a máquina fizer parte da lista de máquinas destino autorizadas, passa-se então à fase seguinte, que consiste na criação de uma nova *Socket* para fazer a comunicação entre o servidor Proxytel e a máquina destino, e de duas novas *Threads* que irão redireccionar a comunicação, cada uma no seu sentido. Isto é, uma das *Threads* irá buscar dados à stream de saída da nova *Socket* e colocar-los-à na stream de entrada da *Socket* que fazia a comunicação entre o servidor Proxytel e a máquina origem (ou cliente); enquanto que a outra *Thread* faz precisamente o contrário, colocando os dados à saída da stream já existente na stream de entrada da *Socket* mais recente. Consegue-se assim uma ligação virtual directa entre a máquina cliente que se ligou ao servidor Proxytel e a máquina remota.




```
socket2 = new Socket(addr,porto);

com1 = new Communication(socket1.getInputStream(),socket2.getOutputStream());
com2 = new Communication(socket2.getInputStream(),socket1.getOutputStream());
try{
  com1.join();
  com2.join();
}catch(InterruptedException e){
  System.out.println("error");
}
```

As duas instruções *join()* servem para garantir que a *Thread* fica à espera que as duas outras *Threads* morram para esta poder prosseguir. Desta forma, possibilita-se que a ligação entre o servidor Proxytel e a máquina origem continue activa mesmo depois de se ter terminado a ligação a uma máquina remota.

Um problema com o qual nos deparámos foi a impossibilidade de efectuar duas ligações durante a execução do servidor a determinadas máquinas destino. Este problema verifica-se no entanto em situações bem delimitadas, sendo uma das condicionantes a máquina origem estar a funcionar num terminal do sistema operativo Unix.

Resta então apenas explicar o que cada *Thread* da classe *Communication* faz enquanto está a executar. O processo é bastante simples e já foi descrito mais acima. São lidos, caractere a caractere, os dados da stream de entrada até ocorrer o caractere 65535, que indica o fim da ligação, e são escritos na stream de saída.

Uma eventual alteração em futuras versões seria passar de uma comunicação constituída pelo envio de pacotes de caracteres para uma comunicação baseada no envio de pacotes com linhas completas.

```
try{
    while(true){
     c = (char) in.read();
     //indica o fim da ligação
      if (c==65535){
         return;
       }
       out.write(c);
       out.flush();
    }
}catch(IOException e){
}
```

Manual do Utilizador

O utilizador deverá utilizar uma aplicação de Telnet e ligar-se ao servidor Proxytel no porto onde este foi ligado.

De seguida, o utilizador deverá introduzir o seu login e a sua password para se identificar. O utilizador dispões de três tentativas para inserir a password correcta, caso contrário, a aplicação será fechada automaticamente. Cada vez que o utilizador introduz uma password errada, aparece a mensagem *'Login incorrect'*. Se o utilizador estiver acreditado para entrar no Proxytel, deverá aparecer a prompt, sinal de que o servidor está pronto a receber comandos.

É importante salientar que existem dois tipos de utilizadores: o administrador (utilizador *root*) e o utilizador comum. Estes têm um leque diferente de comandos à sua disposição, bastante mais alargado para o administrador devido às suas funções de gestor.

De referir também que para auxiliar qualquer dúvida do utilizador, este poderá recorrer em qualquer altura ao comando ***'help'***, o que permitir-lhe-á visualizar a listagem de todos os comandos aos quais tem acesso, assim como a sua respectiva sintaxe de utilização.

Segue uma listagem de todos os comandos disponíveis para cada tipo de utilizador.

Administrador (root)

- ***connect*** *<remote> [port]*

  Estabelece uma ligação Telnet à máquina especificada em *<remote>*. Opcionalmente é possível especificar o porto de ligação no parâmetro *[port]* caso este seja diferente do porto 23.

  Se a máquina *<remote>* não existir na rede, é devolvida a mensagem *'host unknown'*.

  Se a máquina *<remote>* não fizer parte das máquinas destino autorizadas, é devolvida a *mensagem 'access not allowed: type 'listremotes' to see a list of available remotes'.*

- ***c*** *<remote host> [port]*

  Semelhante ao comando ***connect***.

- ***mkuser*** *<login> <password>*

  Cria um novo utilizador com o login especificado por *<login>* e a password indicada em *<password>*.

  Caso já exista um utilizador com o mesmo login, é devolvida a mensagem *'not succeed'.*

- ***rmuser*** *<login>*

  Remove o utilizador com o login indicado em *<login>*.

  Caso não exista nenhum utilizador com esse login, é devolvida a mensagem *'not succeed'*.

- ***listusers***
  Mostra todos os utilizadores registados no servidor Proxytel.
- ***mkremote*** *<remote>*
  Adiciona uma nova máquina destino com o nome ou o IP *<remote>* à lista existente.
  Caso já exista uma máquina destino com esse nome ou IP, é devolvida a mensagem *'not succeed'.*

- ***rmremote*** *<remote>*
  Remove a máquina destino com o nome ou IP indicado por *<remote>*.
  Caso não exista nenhuma máquina destino com esse nome ou IP, é devolvida a mensagem *'not succeed'.*

- ***listremotes***
  Mostra todos os máquinas destino registadas no servidor Proxytel.

- ***mkclient*** *<client>*
  Adiciona uma nova máquina origem com o nome ou o IP *<client>* à lista existente.
  Caso já exista uma máquina origem com esse nome ou IP, é devolvida a mensagem *'not succeed'.*

- ***rmclient*** *<client>*
  Remove a máquina origem com o nome ou IP indicado por *<client>*.
  Caso não exista nenhuma máquina origem com esse nome ou IP, é devolvida a mensagem *'not succeed'.*

- ***listclients***
  Mostra todos os máquinas origem registadas no servidor Proxytel.

- ***passwd*** *<old> <new>*
  Altera a password do utilizador actual. Deve ser especificada a password antiga em *<old>* assim como a nova em *<new>*.
  Caso a password *<old>* não corresponda à password actual do utilizador, é devolvida a mensagem *'not succeed'.*

- ***help***
  Mostra a lista de todos os comandos disponíveis para o utilizador actual.

- ***exit***
  Sai do Proxytel cortando a ligação com o servidor.

Utilizador comum

- ***connect*** *<remote> [port]*
  Estabelece uma ligação Telnet à máquina especificada em *<remote>*. Opcionalmente é possível especificar o porto de ligação no parâmetro *[port]* caso este seja diferente do porto 23.
  Se a máquina *<remote>* não existir na rede, é devolvida a mensagem *'host unknown'*.

Se a máquina *<remote>* não fizer parte das máquinas destino autorizadas, é devolvida a *mensagem 'access not allowed: type 'listremotes' to see a list of available remotes'.*

- ***c*** *<remote> [port]*
  Semelhante ao comando ***connect***.

- ***passwd*** *<old> <new>*
  Altera a password do utilizador actual. Deve ser especificada a password antiga em *<old>* assim como a nova em *<new>*.
  Caso a password *<old>* não corresponda à password actual do utilizador, é devolvida a mensagem *'not succeed'.*

- ***listremotes***
  Mostra todos os máquinas destino registadas no servidor Proxytel.

- ***help***
  Mostra a lista de todos os comandos disponíveis para o utilizador actual.

- ***exit***
  Sai do Proxytel cortando a ligação com o servidor.

Anexos

Classe Administration

```
import java.util.*;
import java.security.*;
import java.io.*;
import java.net.*;

/**
 * Permite fazer a gestão do proxy em termos de utilizadores, destinos e origens
 * das ligações.
 * @version 1.0
 * @author Paulo Guilhoto
 * @author Luis Dinis
 */
public class Administration{
  private Hashtable hash = new Hashtable();
  private Vector remotes = new Vector();
  private Vector clients = new Vector();
  private MessageDigest sha;
  private Files files;

  public Administration(){
    try{
      //cria uma nova instância do algoritmo de compressão SHA
      sha = MessageDigest.getInstance("SHA");
    } catch(NoSuchAlgorithmException e){
       System.out.println("Erro no algoritmo de compressão");
    }
  }

  /**
   * Cria um novo utilizador.
   * @param login o login do novo utilizador
   * @param password a password do novo utilizador
   * @return true se o utilizador foi criado com sucesso
   * @return false caso contrário
   */
  public synchronized boolean create_user(String login, String password){
    Object obj;
    //verifica se ja' existe esse utilizador
    obj = hash.get(login);
    if (obj != null)
      return false;
    if ((login!=null) && (password!=null)){
      //encripta a password
      sha.update(password.getBytes());
      byte[] encrypt = sha.digest();
     obj = hash.put(login,encrypt);
    }
    //guarda as alterações
   files.saveUtilizadores();
    return true;
  }

  /**
   * Remove um utilizador da lista de utilizadores autorizados.
   * @param login o login do utilizador a remover
   * @return true se o utilizador foi removido com sucesso
   * @return false caso contrário
   */
  public synchronized boolean delete_user(String login){
    Object obj;
    //verifica se o utilizador existe
    obj = hash.get(login);
    if ((obj == null) || (login.equals("root")))
      return false;
    if (login!=null){
     obj = hash.remove(login);
    }
    //guarda as alterações
   files.saveUtilizadores();
    return true;
```

```
}

/**
 * Efectua uma enumeração de todos os utilizadores autorizados a entrar no Proxytel.
 * @param out o Stream do cliente para o qual deve ser enviada a lista
 */
public void list_users(BufferedWriter out)
{
  String s = "";
  try{
    Enumeration enum = hash.keys();
    //percorre a Hashtable
    while (enum.hasMoreElements()){
      s = "   "+(String)enum.nextElement()+"\n\r";
      out.write(s,0,s.length());
      out.flush();
    }
  } catch(IOException e) {
      System.err.println("IO Exception");
  }
}

/**
 * Verifica se um determinado utilizador faz parte da lista de utilizadores
 * autorizados.
 * @param login o login do utilizador
 * @param password a password do utilizador
 */
public synchronized boolean identifica_user(String login, String password){
  sha.reset();
  //vai buscar a password da Hashtable correspondendo a este utilizador
  byte[] pass_read = (byte[]) hash.get(login);
  //encripta a password passada como parâmetro para a poder comparar com a password
  //já encriptada existente na Hashtable
  sha.update(password.getBytes());
  byte[] encrypt = sha.digest();
  if ((pass_read==null) || (!comparaPass(pass_read,encrypt)))
    return false;
  return true;
}

/**
 * Compara duas passwords encriptadas.
 * @param pass1
 * @param pass2
 * @return true se as passwords coincidem
 * @return false caso contrário
 */
public boolean comparaPass(byte[] pass1, byte[] pass2){
  if (pass1.length != pass2.length)
  return false;
  for (int i=0;i<pass1.length;i++){
  if (pass1[i] != pass2[i])
    return false;
  }
  return true;
}

/**
 * Altera a password de um utilizador.
 * Remove o utilizador e volta a criá-lo mas agora com uma nova password.
 * @param login o login do utilizador
 * @param pass1 a password antiga
 * @param pass2 a password nova
 * @return true se o foi alterada a password do utilizador com sucesso
 * @return false caso contrário
 */
public synchronized boolean alteraPass(String login, String pass1, String pass2){
  Object obj;
  //verifica se ja' existe esse utilizador
  obj = hash.get(login);
  if ((obj == null) || (!identifica_user(login, pass1)))
    return false;
```

```
    delete_user(login);
    create_user(login,pass2);
    //guarda as alterações
   files.saveUtilizadores();
    return true;
  }

  /**
   * Adiciona uma nova máquina remota autorizada como destino das ligações.
   * @param remote a máquina remota
   * @return true se foi adicionada a nova máquina destino com sucesso
   * @return false caso contrário
   */
  public boolean add_remote(String remote){
    try{
      InetAddress host = InetAddress.getByName(remote);
      int index = remotes.indexOf(host);
      if (index!=-1)
        return false;
      remotes.addElement(host);
      //guarda as alterações
      files.saveRemotos();
      return true;
    }catch(java.net.UnknownHostException e){
      return false;
    }
  }

  /**
   * Remove uma máquina remota autorizada como destino das ligações.
   * @param remote a máquina remota
   * @return true se foi removida a máquina destino com sucesso
   * @return false caso contrário
   */
  public boolean delete_remote(String remote){
    try{
      InetAddress host = InetAddress.getByName(remote);
      boolean index = remotes.removeElement(host);
      if (!index)
        return false;
      //guarda as alterações
      files.saveRemotos();
      return true;
    }catch(java.net.UnknownHostException e){
      return false;
    }
  }

  /**
   * Efectua uma enumeração de todas as máquinas remotas autorizadas como destino
   * das ligações.
   * @param out o Stream do cliente para o qual deve ser enviada a lista
   */
  public void list_remotes(BufferedWriter out){
    String s = "";
    try{
      Enumeration enum = remotes.elements();
      //percorre todo o Vector de máquinas remotas
      while (enum.hasMoreElements()){
        s = "   "+(InetAddress)enum.nextElement()+"\n\r";
        out.write(s,0,s.length());
        out.flush();
      }
    } catch(IOException e) {
        System.err.println("IO Exception");
    }
  }

  /**
   * Verifica se a máquina especificada faz parte da lista de máquinas remotas
   * autorizadas como destino das ligações.
   * @return true se é uma máquina autorizada
   * @return false caso contrário
```

```
 */
public boolean identifica_remote(InetAddress host){
  int index = remotes.indexOf(host);
  if (index==-1)
    return false;
  return true;
}

/**
 * Adiciona uma nova máquina cliente autorizada para efectuar ligações.
 * @param client a máquina cliente
 * @return true se foi adicionada a máquina origem com sucesso
 * @return false caso contrário
 */
public boolean add_client(String client){
  try{
    InetAddress host = InetAddress.getByName(client);
    int index = clients.indexOf(host);
    if (index!=-1)
      return false;
    clients.addElement(host);
    //guarda as alterações
    files.saveClientes();
    return true;
  }catch(java.net.UnknownHostException e){
    return false;
  }
}

/**
 * Remove uma máquina cliente autorizada para efectuar ligações.
 * @param client a máquina cliente
 * @return true se foi removida a máquina origem com sucesso
 * @return false caso contrário
 */
public boolean delete_client(String client){
  try{
    InetAddress host = InetAddress.getByName(client);
    boolean index = clients.removeElement(host);
    if (!index)
      return false;
    //guarda as alterações
    files.saveClientes();
    return true;
  }catch(java.net.UnknownHostException e){
    return false;
  }
}

/**
 * Efectua uma enumeração de todas as máquinas cliente autorizadas para efectuar
 * ligações.
 * @param out o Stream do cliente para o qual deve ser enviada a lista
 */
public void list_clients(BufferedWriter out){
  String s = "";
  try{
    Enumeration enum = clients.elements();
    while (enum.hasMoreElements()){
      s = "   "+(InetAddress)enum.nextElement()+"\n\r";
      out.write(s,0,s.length());
      out.flush();
    }
  } catch(IOException e) {
      System.err.println("IO Exception");
  }
}

/**
 * Verifica se a máquina especificada faz parte da lista de máquinas cliente
 * autorizadas para efectuar ligações.
 * @return true se é uma máquina autorizada
 * @return false caso contrário
```

```
   */
  public boolean identifica_client(InetAddress client){
    int index = clients.indexOf(client);
    if (index==-1)
      return false;
    return true;
  }

  /**
   * @param files
   */
  public void setFiles(Files files){
    this.files = files;
  }

  /**
   * @param hash
   */
  public void setHash(Hashtable hash){
    this.hash = hash;
  }

  /**
   * Devolve a Hashtable de utilizadores autorizados a se ligarem ao Proxytel.
   * @return a Hashtable de utilizadores autorizados
   */
  public Hashtable getHash(){
    return hash;
  }

  /**
   * @param remotes
   */
  public void setRemotes(Vector remotes){
    this.remotes = remotes;
  }

  /**
   * Devolve o Vector de máquinas remotas autorizadas como destino das ligações.
   * @return o Vector de máquinas remotas
   */
  public Vector getRemotes(){
    return remotes;
  }

  /**
   * @param clients
   */
  public void setClients(Vector clients){
    this.clients = clients;
  }

  /**
   * Devolve o Vector de máquinas cliente autorizadas a efectuar ligações.
   * @return o Vector de máquinas cliente
   */
  public Vector getClients(){
    return clients;
  }
}
```

Classe Communication

```
import java.io.*;
import java.net.*;

/**
 * Concatena as ligações entre as duas sockets.
 * O InputStream de uma é redireccionado para o OutputStream da outra, repetindo-se
 * o processo mas desta vez ao contrário, criando assim um pipe bidireccional.
 * @version 1.0
 * @author Paulo Guilhoto
 * @author Luis Dinis
 */
public class Communication extends Thread{
  private ServeCliente serveCliente;
  private Socket socket;
  private BufferedReader in;
  private BufferedWriter out;

  /**
   * @param i a InputStream de uma das sockets
   * @param o a OutputStream da outra socket
   */
  public Communication(InputStream i, OutputStream o){
    in = new BufferedReader(new InputStreamReader(i));
    out = new BufferedWriter(new OutputStreamWriter(o));
    start();
  }

  /**
   * Lê caractere a caractere do InputStream e coloca o caractere no OutputStream.
   */
  public void run() {
  String line;
  char c;

  try{
      while(true){
       c = (char) in.read();
       //indica o fim da ligação
        if (c==65535){
           return;
         }
         out.write(c);
         out.flush();
      }
  }catch(IOException e){
  }
  }
}
```

Classe Files

```
import java.io.*;
import java.util.*;
import java.net.*;

/**
 * Trata da leitura e escrita nos ficheiros 'utilizadores.aut', 'remotos.aut' e
 * 'clientes.aut'.
 * @version 1.0
 * @author Paulo Guilhoto
 * @author Luis Dinis
 */
public class Files{
  private Administration admin;

  public Files(Administration admin){
    this.admin = admin;
  }

  /**
   * Leitura de uma Hashtable do ficheiro 'utilizadores.aut'.
   */
  public void readUtilizadores(){
    String login;
    String pass;

    try{
      File f = new File("utilizadores.aut");
      if (f.exists()){
        FileInputStream dataFile = new FileInputStream("utilizadores.aut");
        ObjectInputStream file_in = new ObjectInputStream(dataFile);

        Hashtable hash = (Hashtable)file_in.readObject();
        admin.setHash(hash);

       file_in.close();
      }
      else
       //não existe o ficheiro, logo é criado por defeito um primeiro utilizador:
       //o administrador do proxy (root)
       admin.create_user("root","root");
    } catch(Exception e) {
        System.out.println("Erro na leitura do ficheiro utilizadores.aut "+e);
    }
  }

  /**
   * Escrita da Hashtable contendo todos os utilizadores acreditados no ficheiro
   * 'utilizadores.aut'.
   */
  public void saveUtilizadores(){
    try{
      FileOutputStream dataFile = new FileOutputStream("utilizadores.aut");
      ObjectOutputStream file_out = new ObjectOutputStream(dataFile);

      file_out.writeObject(admin.getHash());
      file_out.close();
    } catch(Exception e) {
        System.out.println("Erro na escrita do ficheiro utilizadores.aut "+e);
    }
  }

  /**
   * Escrita de um registo de log no ficheiro 'logs.txt'.
   * @param login login do utilizador
   * @param origem máquina a partir do qual se ligou o cliente
   * @param destino máquina à qual o cliente efectuou ligação
   * @param time_inio momento em que se iniciou a ligação
   * @param time_fim momento em que se fechou a ligação
   */
```

```
  public void saveLogs(String login, InetAddress origem, InetAddress destino, int[]
time_inicio, int[] time_fim){
    try{
      String logEntry;
      String tempo_inicio = "", tempo_fim = "";
      BufferedWriter logFile = new BufferedWriter(new FileWriter("logs.txt",true));

      for(int i=0; i<6; i++){
        switch (i){
          case 2: tempo_inicio = tempo_inicio + time_inicio[i] + "/";
                  tempo_fim = tempo_fim + time_fim[i] + "/";
                  break;
           case 5: tempo_inicio = tempo_inicio + time_inicio[i];
                  tempo_fim = tempo_fim + time_fim[i];
                  break;
            default:tempo_inicio = tempo_inicio + time_inicio[i] + ":";
                  tempo_fim = tempo_fim + time_fim[i] + ":";
        }
      }

      logEntry = login+" "+origem+" "+destino+" "+tempo_inicio+" "+tempo_fim;
      logFile.write(logEntry);
      logFile.newLine();
      logFile.close();
    }
    catch(IOException e){
      System.out.println("Erro na escrita do ficheiro de logs");
    }
  }

  /**
   * Leitura de um Vector de máquinas remotas autorizadas do ficheiro 'remotos.aut'.
   */
  public void readRemotos(){
    try{
      File f = new File("remotos.aut");
      if (f.exists()){
      FileInputStream dataFile = new FileInputStream("remotos.aut");
      ObjectInputStream file_in = new ObjectInputStream(dataFile);

        Vector remotes = (Vector)file_in.readObject();
        admin.setRemotes(remotes);

       file_in.close();
      }
      else
        System.out.println("O ficheiro remotos.aut não existe");
    } catch(Exception e) {
        System.out.println("Erro na leitura do ficheiro remotos.aut "+e);
    }
  }

  /**
   * Escrita de um Vector de máquinas remotas autorizadas para o ficheiro
   * 'remotos.aut'.
   */
  public void saveRemotos(){
    try{
      FileOutputStream dataFile = new FileOutputStream("remotos.aut");
      ObjectOutputStream file_out = new ObjectOutputStream(dataFile);

      file_out.writeObject(admin.getRemotes());
      file_out.close();
    } catch(Exception e) {
        System.out.println("Erro na escrita do ficheiro remotos.aut "+e);
    }
  }

  /**
   * Leitura de um Vector de máquinas autorizadas a efectuar ligação ao Proxytel
   * do ficheiro 'clientes.aut'.
   */
  public void readClientes(){
```

```
    try{
      File f = new File("clientes.aut");
      if (f.exists()){
        FileInputStream dataFile = new FileInputStream("clientes.aut");
        ObjectInputStream file_in = new ObjectInputStream(dataFile);

        Vector clients = (Vector)file_in.readObject();
        admin.setClients(clients);

       file_in.close();
      }
      else{
         System.out.println("O ficheiro clientes.aut não existe");
         //como o ficheiro não existe, é essencial adicionar como cliente a própria
         //máquina onde se encontra a correr o servidor Proxytel
         admin.add_client("127.0.0.1");
      }
    } catch(Exception e) {
        System.out.println("Erro na leitura do ficheiro clientes.aut "+e);
    }
  }

  /**
   * Escrita de um Vector de máquinas autorizadas a efectuar ligação ao Proxytel
   * no ficheiro 'clientes.aut'.
   */
  public void saveClientes(){
    try{
      FileOutputStream dataFile = new FileOutputStream("clientes.aut");
      ObjectOutputStream file_out = new ObjectOutputStream(dataFile);

      file_out.writeObject(admin.getClients());
      file_out.close();
    } catch(Exception e) {
        System.out.println("Erro na escrita do ficheiro clientes.aut "+e);
    }
  }
}
```

Classe Proxytel

```
import java.io.*;
import java.net.*;

/**
 * Criação do servidor.
 * O servidor fica à espera no porto especificado como parâmetro.
 * De cada vez que chega um cliente, é criada uma nova socket para servir cada
 * cliente, permitindo assim atender vários pedidos de origens.
 * O resto do processo é então reencaminhado para a classe ServeCliente.
 * @version 1.0
 * @author Paulo Guilhoto
 * @author Luis Dinis
 */
public class Proxytel {
  private static final int PORT = 7080;
  private static int porto;
   private static Files files;
   private static Administration admin;

  public static void main(String[] args) throws IOException{
  //Tratamento dos parâmetros de entrada
    if (args.length == 2 ) {
      if (args[0].equals("-p"))
      try{
      porto = Integer.parseInt(args[1]);
      }catch(NumberFormatException e) {
        porto = PORT;
    }
      else
      porto = PORT;
    }
    else
       porto = PORT;

    ServerSocket s = new ServerSocket(porto);
    System.out.println("Servidor iniciado no porto "+porto);

    admin = new Administration();
    files = new Files(admin);
    admin.setFiles(files);
    files.readUtilizadores();
    files.readRemotos();
    files.readClientes();

    try {
      while(true) {
        //Bloqueia até haver uma ligação
        Socket socket = s.accept();
        try {
          new ServeCliente(socket,admin,files);
        } catch(IOException e) {
            System.out.println("Erro na criacao do ServeCliente: "+e);
           socket.close();
        }
      }
    }
    finally {
      System.out.println("Closing server");
      s.close();
    }
  }
}
```

Classe ServeCliente

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * Trata de fazer a comunicação entre o servidor Proxytel e o cliente.
 * Disponibiliza uma variada lista de comandos ao utilizador, com destaque
 * para os comandos de configuração apenas acessiveis pelo administrador do
 * proxy (root).
 * @version 1.0
 * @author Paulo Guilhoto
 * @author Luis Dinis
 */
public class ServeCliente extends Thread {
  private Socket socket1, socket2;
  private BufferedReader in;
  private BufferedWriter out;
  private int echoo;
  private Files files;
  private String[] cwd =
{"help","c","connect","mkuser","rmuser","listusers","passwd","mkremote","rmremote","listr
emotes","mkclient","rmclient","listclients","exit"};
  private Communication com1, com2;
  private Administration admin;
  private Date date = new Date();
  private int[] time_inicio = new int[6];
  private int[] time_fim = new int[6];
  private String login;

  /**
   * @param s socket de comunicação entre o servidor Proxytel e o cliente
   * @param admin instancia da classe
   * @param f instância da classe
   */
  public ServeCliente(Socket s, Administration admin, Files f) throws
java.io.IOException{
  this.admin = admin;
  files = f;
    socket1 = s;
    try{
      in = new BufferedReader(new InputStreamReader(socket1.getInputStream()));
      out = new BufferedWriter(new OutputStreamWriter(socket1.getOutputStream()));

    } catch(IOException e){
       System.out.println("Erro nos streams dos sockets");
    }
    start();
  }

  /**
   * Trata de identificar o utilizador e de responder a todos os comandos
   * introduzidos por este.
   */
  public void run() {
  boolean match = false;
  int tentativas = 0;
    String pass;
    String line, cwdz;
    int index;
    boolean cwd_ok = false;

    try {
      desactiva_echo_local();

      writeChars("                    ****************************************\n\r");
      writeChars("                            Bemvindo ao Proxytel do DEI     \n\r");
      writeChars("                                                            \n\r");
      writeChars("                                 versao 1.0                 \n\r");
      writeChars("                    ****************************************\n\r");
```

```
//verifica se o cliente está ligado de uma máquina autorizada
if (admin.identifica_client(socket1.getInetAddress())){
  echo(1);
  //identificação do utilizador
  do {
     tentativas++;
    writeChars("\n\rLogin: ");
    login = readLine();
    writeChars(login+"'s Password: ");
    echo(0);
    pass = readLine();
    echo(1);
    match = admin.identifica_user(login,pass);
    if (!match)
      writeChars("\n\rLogin incorrect\n\r");
  } while ((!match) && (tentativas<3));

  if ((!match) && (tentativas==3))
  return;
  else
  writeChars("\n\r\n\rFor some help, type 'help'\n\r");

  //prompt
  do {
  cwd_ok = false;
    writeChars("proxytel> ");
    line = readLine();

    if (!line.equals("")){
     StringTokenizer token = new StringTokenizer(line);
     cwdz = token.nextToken();
       index = indexOf(cwdz);

     //-------------- comandos comuns a todos os utilizadores  -------------

       switch (index){
         case 0: //ajuda
                  if (login.equals("root"))
                   roothelp();
                 else
                   help();
                cwd_ok=true;
                break;
         case 1:
         case 2: //ligação a um servidor remoto
                 if (token.countTokens() == 2)
                   connect(token.nextToken(),token.nextToken());
                 else
                   if (token.countTokens() == 1)
                     connect(token.nextToken(),"23");  //porto por defeito de Telnet

                 cwd_ok=true;
                 break;
         case 6: //alteração da password
                 if (token.countTokens() == 2){
                 if (!admin.alteraPass(login,token.nextToken(),token.nextToken()))
                      writeChars("   not succeed\n\r");
                   else
                     writeChars("   successfull\n\r");
               }
                     cwd_ok=true;
                     break;
         case 9: //listagem de todos os servidores remotos
                 admin.list_remotes(out);
                   cwd_ok=true;
                   break;
         case 13://exit
                 cwd_ok=true;
       }


       //------------ comandos relativos ao administrador do proxy ------------
```

```
      if (login.equals("root"))
         switch (index){
         case 0: //ajuda
         case 3: //criação de um novo utilizador
                  if (token.countTokens() == 2){
                  if (!admin.create_user(token.nextToken(), token.nextToken()))

                    writeChars("   not succeed\n\r");
                  else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 4: //eliminação de um utilizador
                  if (token.countTokens() == 1){
                  if (!admin.delete_user(token.nextToken()))
                  writeChars("   not succeed\n\r");
               else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 5: //listagem de todos os utilizadores
                    admin.list_users(out);
                    cwd_ok=true;
                    break;
           case 7: //adição de um novo servidor remoto
                  if (token.countTokens() == 1){
                      if (!admin.add_remote(token.nextToken()))
                  writeChars("   not succeed\n\r");
               else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 8: //eliminação de um servidor remoto
                   if (token.countTokens() == 1){
                      if (!admin.delete_remote(token.nextToken()))
                  writeChars("   not succeed\n\r");
               else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 10://adição de um novo cliente
                  if (token.countTokens() == 1){
                      if (!admin.add_client(token.nextToken()))
                  writeChars("   not succeed\n\r");
               else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 11://eliminação de um cliente
                   if (token.countTokens() == 1){
                      if (!admin.delete_client(token.nextToken()))
                  writeChars("   not succeed\n\r");
               else
                    writeChars("   successfull\n\r");
                  }
                  cwd_ok=true;
                  break;
          case 12://listagem de todos os clientes
                    admin.list_clients(out);
                    cwd_ok=true;
                    break;
         }
      if(!cwd_ok)
         writeChars("   command not understood\n\r");
    }
    else
     cwdz = "";
  } while(!cwdz.equals("exit"));
```

```
    }

    //cliente não autorizado
    else
     writeChars("\n\rCliente nao autorizado!\n\r");
   }
   finally {
     try {
       socket1.close();
     } catch(IOException e) {
         System.out.println("Socket1 not closed");
     }
   }
}


/**
 * Permite desligar ou ligar o echo efectuado pelo servidor dos dados enviados
 * pelo cliente
 * @param n Se 0, desliga o echo. Se 1, liga o echo.
 */
private void echo(int n) {
echoo = n;
}

/**
 * Permite negociar os parâmetros da ligação com o cliente.
 * É primeiro indicado que será o servidor Proxytel a fazer o echo dos dados, e
 * depois é pedido ao cliente para mostrar os caracteres à medida que estão a
 * ser introduzidos pelo utilizador.
 */
private void desactiva_echo_local() {
   char c;

try{
    //IAC WILL ECHO
    writeChar((char)255);
    writeChar((char)251);
    writeChar((char)1);
    c = (char)in.read();
    c = (char)in.read();
    c = (char)in.read();
    //SUPPRESS GO AHEAD
    writeChar((char)255);
    writeChar((char)251);
    writeChar((char)3);
    c = (char)in.read();
    c = (char)in.read();
    c = (char)in.read();
  }
  catch(IOException e){}
  }

/**
 * Lê uma string da socket, caractere a caractere, e faz o echo caso seja
 * necessário.
 * @return a string lida
 */
private String readLine(){
char c, b;
String str = new String ();

try{
    do{
    c = (char) in.read();
    if (echoo==1){
    if (((int)c == 8) || ((int)c == 127)){     //backspace
          if (str.length()!=0){
        writeChar((char)8);
            writeChar((char)32);
            writeChar((char)8);
          }
      }
```

```
        else{
          if (((int)c != 13) && ((int)c != 10))
            writeChar(c);
          else{
           if ((int)c == 13){
           b = (char) in.read();
               writeChar((char)13);
               writeChar((char)10);
            }
           }
         }
    }
      else   //sem echo
      if ((int)c == 13)
        b = (char) in.read();

    if (((int)c != 10) && ((int)c != 13) && ((int)c != 8) && ((int)c != 127))
      str = str + c;
    if ((((int)c == 8) || ((int)c == 127)) && (str.length()!=0))
      str = str.substring(0, str.length()-1);
    }while (((int)c!=13) && ((int)c!=10));
} catch(IOException e){
    System.out.println("Erro ao ler caracteres do cliente");
}
  return(str);
}

/**
 * Permite efectuar a ligação a um servidor remoto.
 * É criada uma nova socket para efectuar a comunicação entre o servidor Proxytel
 * e o servidor remoto.
 * @param machine servidor remoto ao qual se está a tentar fazer a ligação
 * @param port porto do servidor remoto (habitualmente 23)
 */
private void connect(String machine, String port){
int porto;
InetAddress addr = null;
boolean host_unknown = false;

  try {
    addr = InetAddress.getByName(machine);
  } catch(java.net.UnknownHostException e){
    writeChars("   host unknown\n\r");
      host_unknown = true;
  }

  if (!host_unknown)
    //verifica se a máquina faz parte da lista de remotos acessíveis
    if (admin.identifica_remote(addr)){
      writeChars("Connecting...\n\r");

      try{
        try{
          porto = Integer.parseInt(port);
        }catch(NumberFormatException e) {
          porto = 23;
        }
        socket2 = new Socket(addr,porto);

         //guarda informação útil para registo dos logs
         time_inicio = getTime(time_inicio);

        com1 = new Communication(socket1.getInputStream(), socket2.getOutputStream());

        com2 = new Communication(socket2.getInputStream(), socket1.getOutputStream());
        try{
          com1.join();
          com2.join();
        }catch(InterruptedException e){
        System.out.println("error");
        }
        writeChars("\n\r");
        //guarda informação útil para registo dos logs
```

```
          time_fim = getTime(time_fim);
          files.saveLogs(login,socket1.getInetAddress(),addr,time_inicio,time_fim);
        }
        catch(IOException e){
        System.out.println("Erro na comunicacao");
        }
        finally {
          try {
            socket2.close();
          } catch(IOException e) {
            System.out.println("Socket2 not closed");
          }
        }
      }
      else
       writeChars("   access not allowed: type 'listremotes' to see a list of available
remotes\n\r");
  }

  /**
   * Listagem de todos os comandos disponiveis para o utilizador comum.
   */
  private void help(){
    writeChars("   connect <remote> [port]: makes a telnet link connection to remote host
\n\r");
    writeChars("   c <remote> [port]      : same usage as 'connect'  \n\r");
    writeChars("   passwd <old> <new>     : changes password of the current user \n\r");

    writeChars("   listremotes            : shows a list of all the remotes
available\n\r");
    writeChars("   help                   : displays this list\n\r");
    writeChars("   exit  \n\r");
  }

  /**
   * Listagem de todos os comandos disponiveis para o administrador.
   */
  private void roothelp(){
    writeChars("   connect <remote> [port]: makes a telnet link connection to a remote
host \n\r");
    writeChars("   c <remote host> [port] : same usage as 'connect'  \n\r");
    writeChars("   mkuser <login> <pass>  : creates a new user  \n\r");
    writeChars("   rmuser <login>         : removes an existing user  \n\r");
    writeChars("   listusers              : shows a list of all the users \n\r");
    writeChars("   mkremote <remote>      : adds a new remote host which can be a string
or a IP\n\r");
    writeChars("   rmremote <remote>      : removes an existing host  \n\r");
    writeChars("   listremotes            : shows a list of all the remotes
available\n\r");
    writeChars("   mkclient <client>      : adds a new client which can be a string or a
IP\n\r");
    writeChars("   rmclient <client>      : removes an existing client  \n\r");
    writeChars("   listclients            : shows a list of all the clients
available\n\r");
    writeChars("   passwd <old> <new>     : changes password of the current user \n\r");

    writeChars("   help                   : displays this list\n\r");
    writeChars("   exit  \n\r");
  }

  /**
   * Coloca num array todos os dados relativos ao tempo corrente.
   * @param tempo o array de inteiros a preencher
   * @return o array de inteiros preenchido
   */
  private int[] getTime(int[] tempo){
    tempo[0] = date.getYear()+1900;
    tempo[1] = date.getMonth()+1;
    tempo[2] = date.getDate();
    tempo[3] = date.getHours();
    tempo[4] = date.getMinutes();
    tempo[5] = date.getSeconds();
     return tempo;
```

```
  }

  /**
   * Escreve uma cadeia de caracteres para a socket.
   * @param s a string a escrever
   */
  private void writeChars(String s){
  try{
    out.write(s,0,s.length());
    out.flush();
    } catch(IOException e) {
        System.err.println("IO Exception");
    }
  }

  /**
   * Escreve um caractere para a socket.
   * @param c o caractere a escrever
   */
  private void writeChar(char c){
  try{
    out.write(c);
    out.flush();
    } catch(IOException e) {
        System.err.println("IO Exception");
    }
  }

  /**
   * Verifica se o comando introduzido faz parte da lista de comandos disponiveis.
   * @param cwdz comando
   * @return a posicao do comando introduzido no array de comandos
   */
  private int indexOf(String cwdz){
  int i=0;
  while ((i<cwd.length) && (!cwdz.equals(cwd[i]))){
    i++;
  };
  return i;
  }
}
```