FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Informática

Licenciatura em Engenharia Informática
1º TESTE– Redes de Computadores
1º Semestre, 2006/2007 (25/Outubro/2006)

NOTAS:

Leia com atenção cada questão antes de responder. A interpretação do enunciado de cada pergunta é um factor de avaliação do teste.

A duração do teste é 1 hora e 15 minutos.

O enunciado contém **7 páginas** que devem ser entregues com a resposta ao teste.

NOME:_____ Nº Aluno: _____

1. Suponha que pretende desenvolver um sistema cliente/servidor para disponibilizar ficheiros combinando os protocolos TCP e UDP da seguinte forma.

O servidor, ao iniciar a execução, cria um *socket* UDP para o qual os clientes podem enviar mensagens com o nome do ficheiro pretendido. O servidor, caso o ficheiro exista, cria um servidor auxiliar que usa o protocolo TCP para efectuar a transferência do ficheiro e envia ao cliente o porto em que o servidor auxiliar aguarda conexões. Caso o ficheiro não exista, a resposta do servidor é a palavra "NAO".

O cliente, ao iniciar a sua execução, envia uma mensagem ao servidor contendo o nome do ficheiro a transferir. Ao receber a resposta do servidor, caso a resposta seja "NAO", o cliente apresenta no ecrã a mensagem "Ficheiro não existente". Caso a resposta seja positiva, vai iniciar a transferência criando um canal de comunicação TCP para o servidor e lendo o conteúdo do ficheiro enviado pelo servidor. O ficheiro recebido deve ser apresentado no ecrã (à medida que é transferido).

O servidor auxiliar limita-se a transmitir o conteúdo do ficheiro para o cliente, após o que termina a sua execução. Assim, as mensagens transmitidas são as seguintes:

INTERACÇÃO COM SERVIDOR PRINCIPAL
CLT->SRV: *string* (nome_do_ficheiro)
SRV->CLT: *string* (**NÃO** ou **porto_do_servidor_auxiliar**)

INTERACÇÃO COM SERVIDOR AUXILIAR
SRV->CLT: sequência de bytes (com conteúdo do ficheiro)

a. Complete o anexo **A.1** e **A.2** com o código da função do cliente que interage com o servidor auxiliar e o código do servidor auxiliar, respectivamente.
b. Complete o anexo **B.1** e **B.2** com o código do cliente e servidor principal.
c. Complete o anexo **C** com as modificações necessárias para que os servidores auxiliares executem concorrentemente com o tratamento de novos pedidos de clientes.

2. Os computadores da Alice e do Bob estão ligados através da Internet. Com o programa *ping* a Alice chegou à conclusão que o tempo de trânsito de ida e volta entre o computador dela e o do Bob seguia o seguinte padrão:

```
alice@localhost$ ping bob.domain.com
PING bob.domain.com (64.233.187.99) 56(84) bytes of data.
64 bytes from 64.233.187.99: icmp_seq=1 ttl=235 time=169 ms
64 bytes from 64.233.187.99: icmp_seq=2 ttl=235 time=167 ms
64 bytes from 64.233.187.99: icmp_seq=3 ttl=235 time=170 ms
64 bytes from 64.233.187.99: icmp_seq=4 ttl=235 time=161 ms
64 bytes from 64.233.187.99: icmp_seq=5 ttl=235 time=186 ms
64 bytes from 64.233.187.99: icmp_seq=6 ttl=235 time=165 ms
64 bytes from 64.233.187.99: icmp_seq=7 ttl=235 time=237 ms
64 bytes from 64.233.187.99: icmp_seq=8 ttl=235 time=168 ms
64 bytes from 64.233.187.99: icmp_seq=9 ttl=235 time=161 ms
64 bytes from 64.233.187.99: icmp_seq=10 ttl=235 time=169 ms
64 bytes from 64.233.187.99: icmp_seq=11 ttl=235 time=172 ms
64 bytes from 64.233.187.99: icmp_seq=12 ttl=235 time=166 ms
64 bytes from 64.233.187.99: icmp_seq=13 ttl=235 time=161 ms
64 bytes from 64.233.187.99: icmp_seq=14 ttl=235 time=170 ms
64 bytes from 64.233.187.99: icmp_seq=15 ttl=235 time=171 ms

--- bob.domain.com ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14017ms
rtt min/avg/max/mdev = 161.176/173.327/237.329/18.052 ms
```

Indique, neste contexto, qual dos seguintes valores de "playout delay" seriam mais adequados de usar pelos programas de IP phone da Alice e do Bob para poderem realizar, via a Internet, uma chamada telefónica de um para o outro. Por hipótese, o CODEC usado não tolera percas de pacotes superiores a 1% sem degradação significativa da qualidade.

7 ms 35 ms 200 ms

3. Admita, por hipótese, que o RTT médio dentro da rede interna da FCT é de 2 ms (mili segundos), que a capacidade dos canais que ligam todos os computadores da FCT à rede interna é de 100 Mbps e que essa rede está bem dimensionada para o tráfego existente, não se formando em geral filas de espera significativas nos routers internos.

a) Qual dos seguintes valores é o tempo médio para receber a resposta a um pedido HTTP, feito por um PC da FCT a um servidor interno através do protocolo HTTP 1.0, sabendo que a resposta ao pedido tem um total de 105 bits, assumindo um tempo de processamento do pedido pelo servidor desprezável?

5 ms 7 ms 20 ms 100 ms

b) Admita, por hipótese, que o RTT médio entre um computador da rede da FCT e um servidor HTTP externo é de 200 ms. Decidiu-se instalar um servidor proxy / cache HTTP na FCT que todos os computadores internos passaram a usar. Constatou-se que em 50% dos casos, uma página HTTP solicitada estava já no proxy (cache hit ratio = 50%). Admita que todas as páginas pedidas tinham 105 bits e que os PCs internos e o proxy só usam o protocolo HTTP 1.0. Qual dos seguintes valores passou a ser o tempo médio que durava cada acesso a uma página HTTP.

402 ms 202 ms 705 ms 505 ms 480 ms 1005 ms

ANEXO A.1 – CLIENTE

```
import                ;
import                ;

public class Client {
    /* … outro código omitido … */

    /* Função para obter o conteúdo do ficheiro a partir do servidor usando TCP*/
    public static void getFile(                serverAddress, int port) {
        try {

            =  new                                                    );

            InputStream in =

            byte[] buf = new byte[1024];
            int len;
            while( (len =                 > 0) {
                System.out.                (                , 0,                );
            }
            System.out.flush();

            ;

        } catch(                ) {
            e.printStackTrace();
            System.err.println( "Erro na transmissão do ficheiro");
        }
    }
}


/* classe com métodos auxiliaries */
public class FileUtils
{
   /* Devolve true se o ficheiro indicado existir */
   static boolean exists( String  filename) {
      return new File( filename).exists();
   }
   /* Devolve input stream para o fiheiro indicado */
   static InputStream getFile( String  filename) throws IOException {
      return new FileInputStream( filename);
   }
}
```

ANEXO A.2 – SERVIDOR

```
import ...


class FileSender
{
   static int SERVER_PORT_SEED = 31000;
   int serverPort;
   String filename;

   FileSender( String filename) {
      this.filename = filename;
      serverPort = SERVER_PORT_SEED++;
   }

   public void doit() throws IOException {
      [                                ] = new [                                ];

      [            ] socket = server.[            ]();

      InputStream in = FileUtils.getFile( filename);
      [                            ] out = socket.[                            ]();

      byte[] arr = new byte[1024];
      int len;
      while( (len = in.[        ]( arr)) > 0) {
         out.[        ]( arr, 0, len);
      }
      out.close();
      socket.[                ]();
      server.[                ]();
   }

}
```

ANEXO B.1 – CLIENTE

```
import ...
import ...

public class Client {

    public static void main(String[] args ) throws Exception {
       if( args.length != 3 ) {
          System.out.printf("Use: java Cliente maquina_do_servidor porto nome_do_ficheiro\n") ;
          System.exit(0);
       }

       [                    ] serverAddress = [                    ].getByName( args[0]) ;
       int port = Integer.parseInt( args[1] ) ;
       String filename = args[2];

       [                                                                                ];

       byte[] filenameArr = filename.getBytes();
       [                    ] request ;
       request = new [                    ]( filenameArr, filenameArr.length ) ;
       request.[                ]( [                    ] ) ;
       request.[            ]( [      ] ) ;

       socket.[        ]( request ) ;

       byte[] buffer = new byte[1024] ;
       [                                                                                ]

       String resultado ;
       resultado = new String( [                                                        ]);

       if( resultado.equals( "NAO" )) {
          System.out.println( "Ficheiro " + filename + " nao existe no servidor");
       } else {
          System.out.println( "Ficheiro " + filename + " existe no servidor");
          int portNum = Integer.parseInt( resultado);
          getFile( [                    ], portNum);
       }
    }

    public static void getFile( InetAddress serverAddress, int port) {
       /* como no anexo B.1 */
    }
}
```

ANEXO B.2 – SERVIDOR

```
import ...
import ...

public class Server {
   public static void main(String[] args) throws Exception {
      if( args.length == 1) {
         new Server( Integer.parseInt( args[0])).doit();
      } else {
         System.err.println( "Use: java Server porto");
   } }
   int serverPort;

   Server( int serverPort) {
      this.serverPort = serverPort;
   }

   void doit() throws IOException {
      
                                                                                ;

      byte[] buffer = new byte[1024];
      for( ; ; ) {
         
                                                                                
                                                                                
                                                                                
                                                                                


         String resultado = null;
         FileSender fs = null;

         if( FileUtils.exists( filename )) {
            fs = new FileSender( filename);
            resultado = "" + fs.serverPort;
         } else
            resultado = "NAO";

         byte[] sendMsg = resultado.getBytes();

         
                                                                                
                                                                                
                                                                                
                                                                                


         if( ! resultado.equals( "NAO")) fs.doit();
      }
   }
}

class FileSender { /* como definida no anexo A.2 */ }
```

ANEXO C – SERVIDOR CONCORRENTE

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

public class FileUtils { /* como definida no anexo A.2 */ }
```

```
public class Server {
   ... /* como definida no anexo A.2 */

   void doit() throws IOException {
      ... /* como definida no anexo A.2 */

         if( ! resultado.equals( "NAO"))









      }
   }
}

   /* como definida no anexo A.2 excepto: */
class FileSender
{
```

```
}
```