MC823 Atividade 5 — Chamada de Procedimentos Remotos

Gustavo Sverzut Barbieri <ra008849@ic.unicamp.br>

1 Introdução

Este trabalho tratou de chamadas de procedimentos remotos usando duas técnicas comumente utilizadas, o RPC — *Remote Procedure Call* — elaborado em meados da década de 70 e utilizado amplamente a partir da década de 80 com serviços como NFS e NIS, e o RMI — *Remote Method Invocation* — elaborado também pela Sun Microsystems para sua nova arquitetura, o Java. Como exercício foi implementado um servidor de arquivos, bem básico.

A implementação foi idêntica em ambas as linguagens, a fim de testar o desempenho comparativo do sistema de chamadas, não explorando peculiaridades de cada técnica. Esta não utilizou guardar estado no servidor, a cada chamada o cliente envia o nome do arquivo, um deslocamento a partir do inicio deste e a quantidade de bytes a serem lidas; o servidor abre o arquivo, desloca-se para a posição requerida e tenta ler a quantidade desejada, fecha o arquivo o retorna a quantidade que ele conseguiu ler, seguido de um estado que pode indicar sucesso, erro ou fim de arquivo e então os dados. Note que para cada requisição faz-se a abertura, reposicionamento e fechamento do arquivo, deteriorando a performance[1]. Note também que devido a um limite de 8Kb dos pacotes RPC/UDP, utilizamos um tamanho de dados de 8000 bytes em ambas as implementações.

2 Comparativo de Performance

Para medir a diferença de performance entre os sistemas, foi medido o tempo para transportar um arquivo grande e outro pequeno usando o RPC e o RMI, os tempos e taxas estão nas tabelas de 2 a 2. Todas a comunicação foi feita na mesma maquina, usando a placa de rede, eliminando assim o gargalo da rede, porem assegurando que o RPC ou RMI não utilizasse algum artificio para o caso da interface *loopback*.

A principio cogitou-se a comparação com um sistema real e otimizado para troca de arquivos, wget/http, porem o resultado foi muito distante (90Mb/s). Isso se deve ao fato do servidor http (apache) transferir todos os dados de forma sequencial (*stream*) e não precisar abrir-procurar-fechar o arquivo a cada requisição e as implementações RPC e RMI enviarem e esperarem (*stop and wait*) e ainda executar operações de arquivo supra citadas. No entanto as tabelas

[1]Em geral o sistema operacional mantém um cache para os arquivos abertos, além disso existe o custo de reposicionamento da cabeça de leitura e o cache do disco que em geral tem 2Mb e nos testes era utilizados apenas 8Kb.

servem de comparação entre ambos os sistemas, sendo o RMI cerca de 24 vezes mais lento[2].

Para certificar o quanto as operações de arquivos e de alocação de memória (no Java) influenciavam no desempenho, criei uma versão "boba" dos programas as quais transmitiam 10Mb sem ler do disco e usando apenas variáveis estáticas no servidor. O resultado foi ainda mais surpreendente: 170Mb/s para o RPC, 11Mb/s para o RMI, sendo este 15 vezes mais lento.

Tentativa	**Tempo (ms)**	**Taxa (MB/s)**
1	< 1	—
2	< 1	—
3	< 1	—
4	< 1	—
5	< 1	—
Media	< 1	—

Tabela 1: RPC com arquivo pequeno 1Kb.

Tentativa	**Tempo (ms)**	**Taxa (MB/s)**
1	887.0	13.6
2	717.0	16.8
3	729.0	16.6
4	886.0	13.6
5	777.0	15.5
Media	799.2	15.2

Tabela 2: RPC com arquivo grande 12.1Mb.

Tentativa	**Tempo (ms)**	**Taxa (MB/s)**
1	9	0.111
1	9	0.111
1	43	0.023
1	5	0.2000
1	42	0.024
Media	21.6	0.093

Tabela 3: RMI com arquivo pequeno 1Kb.

3 Apêndice A

[2]Um dos fatores que deteriora a implementação RMI eh o fato do Java alocar o buffer todas as vezes, enquanto que em C este eh uma variavel estática.

Tentativa	Tempo (ms)	Taxa (MB/s)
1	19815	0.605
2	18799	0.638
3	18574	0.646
4	18471	0.649
5	19243	0.623
Media	18980	0.632

Tabela 4: RMI com arquivo grande 12.1Mb.

Listing 1: fs.x — Descrição de dados para o `rpcgen`.

```
/* fs.x: file server using RPC */

%#define MAX_FILENAME_SIZE  255
%#define MAX_BLOCK_SIZE    8192

struct fs_get_ret_t
{
  int     status;
  opaque buffer< MAX_BLOCK_SIZE >;
};

struct fs_get_param_t
{
  long   offset;
  long   count;
  string name< MAX_FILENAME_SIZE >;
};

struct fs_list_t
{
  int     status;
  long   offset;
  string name< MAX_FILENAME_SIZE >;
};



program FSPROG
{
  version FSVERS
    {
      string       CRED()                 = 1;

      fs_get_ret_t GET( fs_get_param_t ) = 2;

      fs_list_t    LIST( fs_list_t )     = 3;

      int          EXIST( string )       = 4;

    } = 1;
} = 0x20000001;
```

Listing 2: fs_server.c — Implementação das funções do servidor RPC.

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */
```

```c
#include "fs.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define MAX_FILENAME_SIZE 255
#define MAX_BLOCK_SIZE 8192

static char buffer[ MAX_BLOCK_SIZE ];
extern int errno;



char **
cred_1_svc(void *argp, struct svc_req *rqstp)
{
  static char * result = "Gustavo Sverzut Barbieri <ra008849@ic.unicamp.br>";
  return &result;
}



int *
exist_1_svc(char **file, struct svc_req *rqstp)
{
  static int result;
  struct stat buf;

  result = 0;
  assert( file || *file );

  if ( ( stat( *file, &buf ) == 0 ) &&  S_ISREG( buf.st_mode ) )
    result = 1;

  return &result;
}





fs_get_ret_t *
get_1_svc(fs_get_param_t *file, struct svc_req *rqstp)
{
  static fs_get_ret_t  result;
  FILE *fd = NULL;
  errno = 0;

  assert( file && file->name && file->count );

  result.status            = 0;
  result.buffer.buffer_len = 0;
  result.buffer.buffer_val = buffer;

  if ( file->count >= MAX_BLOCK_SIZE )
    file->count = MAX_BLOCK_SIZE;

  if ( ( fd = fopen( file->name, "r" ) ) == NULL )
    {
      result.status = - errno;
      return &result;
    }

  if ( file->offset )
    fseek( fd, file->offset, SEEK_SET );

  result.buffer.buffer_len = fread( buffer, 1, file->count, fd );

  if ( result.buffer.buffer_len != file->count )
    {
      if ( feof( fd ) )
        result.status = 0;
      else if ( ferror( fd ) )
        {
          result.status = - errno;
          result.buffer.buffer_len  = 0;
        }
    }
  else
    result.status = 1;

  fclose( fd );

  return &result;
}
```

```
fs_list_t *
list_1_svc(fs_list_t *dir, struct svc_req *rqstp)
{
  static fs_list_t  result;
  DIR *d;
  struct dirent *f;
  struct stat fstat;
  int dlen, len;

  errno = 0;

  assert( dir && dir->name );

  result.status   = 0;
  result.offset   = 0;
  result.name     = buffer;
  buffer[ 0 ]     = '\0';

  if ( ( d = opendir( dir->name ) ) == NULL )
    {
      result.status = -errno;
      return &result;
    }

  if ( dir->offset )
    seekdir( d, dir->offset );

  dlen = strlen( dir->name );

  while ( ( f = readdir( d ) ) != NULL )
    {
      len = strlen( f->d_name );
      if ( len + dlen + 2 >= MAX_BLOCK_SIZE )
        {
          result.status = - ENOMEM;
          buffer[ 0 ] = '\0';
          return &result;
        }

      strcpy( buffer, dir->name );
      buffer[ dlen ] = '/';
      buffer[ dlen + 1 ] = '\0';
      strcat( buffer, f->d_name );

      stat( buffer, &fstat );
      if ( S_ISREG( fstat.st_mode ) )
        {
          result.offset = telldir( d );
          result.status = 1;
          break;
        }
    }
  closedir( d );

  return &result;
}
```

Listing 3: fs_client.c — Implementação das funções do cliente RPC.

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "fs.h"
#include <assert.h>
#include "interface.c"
#include <sys/time.h>
#include <errno.h>

#define MAX_FILENAME_SIZE 255
#define MAX_BLOCK_SIZE 8192

#define BLOCK_SIZE 8000

extern int errno;
static CLIENT *client = NULL;

void cred()
{
  char **cred = NULL;
  char *arg;

  assert( client != NULL );

  while ( 1 )
    {
      cred = cred_1( (void*) &arg, client );
      if ( cred == NULL )
        clnt_perror( client, "Call failed!" );
      else
        break;
    }

  printf( "CREDITS: %s\n", *cred );

  if ( clnt_freeres( client, (xdrproc_t) xdr_wrapstring, (caddr_t) cred ) ==0 )
    fprintf( stderr, "ERROR: Could not free RPC/XDR result 'cred' (%p)\n",
             cred );
}


void  exist( char *file )
{
  int *exist = NULL;

  while ( ! exist )
    {
      exist = exist_1( &file, client );
      if ( exist == NULL )
        {
          clnt_perror( client, "Call failed!" );
          continue;
        }

      printf( "%s, file '%s' %s.\n",
              (*exist) ? "Yes" : "No",
              file,
              (*exist) ? "does exist" : "doesn't exist" );

      if ( clnt_freeres( client,
                         (xdrproc_t) xdr_int, (caddr_t) exist ) == 0 )
        fprintf( stderr,
                 "ERROR: Could not free RPC/XDR result 'exist' (%p)\n",
                 exist );
    }
}

void list( char *dir )
{
  fs_list_t  *file;
  fs_list_t  arg;

  assert( client != NULL && dir != NULL );

  arg.name   = dir;
  arg.offset = 0;
  while ( 1 )
    {
      file = list_1( &arg, client );
      if ( file == (fs_list_t *) NULL )
        {
          clnt_perror( client, "Call failed!" );
          continue;
        }

      if ( file->status == 1 )
        puts( file->name );
      else if ( file->status < 0 )
        printf( "ERROR: Could not read '%s': %s\n",
                arg.name,
                strerror( - file->status ) );

      arg.offset = file->offset;

      if ( clnt_freeres( client,
                         (xdrproc_t) xdr_fs_list_t, (caddr_t) file ) == 0 )
        fprintf( stderr, "ERROR: Could not free RPC/XDR result 'file' (%p)\n",
                 file );

      if ( arg.offset == 0 )
        break;
    }
}
```

```
void get( char *src, char *dst, int statistics )
{
  fs_get_ret_t  *gr;
  fs_get_param_t  gp;
  FILE *fd;
  int *exist = NULL;
  double size = 0.0f;
  char  unity[3] = "b";
  unsigned long long int ts, count=0;
  double rate;
  char ru[ 3 ] = "";
  int status = 0;
  struct timeval t0, t1, t2, tr;

  assert( client != NULL  && src != NULL && dst );

  gp.offset = 0;
  gp.count  = BLOCK_SIZE;
  gp.name   = src;

  while ( ! exist )
    {
      exist = exist_1( &src, client );
      if ( exist == NULL )
        {
          clnt_perror( client, "Call failed!" );
          continue;
        }
    }

  if ( *exist == 0 )
    {
      fprintf( stderr, "ERROR: File '%s' does not exit.\n", src );
      return;
    }

  /* Free XDR 'exist' */
  if ( clnt_freeres( client,
                     (xdrproc_t) xdr_int, (caddr_t) exist ) == 0 )
    fprintf( stderr,
             "ERROR: Could not free RPC/XDR result 'exist' (%p)\n",
             exist );

  if ( ( fd = fopen( dst, "w+" ) ) == NULL )
    {
      fprintf( stderr,
               "ERROR: Could not open file for writing '%s': %s\n",
               dst, strerror( errno ) );
      return;
    }

  gettimeofday( &t0, NULL );
  while ( 1 )
    {
      if ( statistics )
        gettimeofday( &t1, NULL );

      gr = get_1( &gp, client );
      if ( gr == (fs_get_ret_t *) NULL )
        {
          clnt_perror( client, "Call failed!" );
          continue;
        }

      if ( statistics )
        {
          gettimeofday( &t2, NULL );
          timersub( &t2, &t1, &tr );
          ts = (unsigned long long) (tr.tv_sec * 1000000 + tr.tv_usec);
          if ( gr->buffer.buffer_len > ( 1024 * 1024 ) )
            {
              strncpy( unity, "Mb", 3 );
              size = (double)1.0f * gr->buffer.buffer_len / 1024.0f / 1024.0f;
            }
          if ( gr->buffer.buffer_len > 1024 )
            {
              strncpy( unity, "Kb", 3 );
              size = (double)1.0f * gr->buffer.buffer_len / 1024.0f;
            }
          else
            {
              strncpy( unity, "b", 3 );
              size = (double)1.0f * gr->buffer.buffer_len;
            }

          rate = (double) gr->buffer.buffer_len / (double)ts;

          fprintf( stderr, "STAT: %0.3f %s in %lld us. [%.1f b/us]\n",
                   size, unity, ts, rate );

        }
      count += gr->buffer.buffer_len;

      status = gr->status;

      if ( status >=0 )
        {
          fwrite( gr->buffer.buffer_val,
                  1, gr->buffer.buffer_len, fd );

          gp.offset += gr->buffer.buffer_len;

          /* Free XDR 'gr' */
          if ( clnt_freeres( client,
                             (xdrproc_t) xdr_fs_get_ret_t, (caddr_t) gr ) ==0)
            fprintf( stderr,
                     "ERROR: Could not free RPC/XDR result 'gr' (%p)\n",
                     gr );

          if ( status == 0 )
            break;
        }
      else
        {
          printf( "ERROR: Could not get file '%s': %s\n",
                  gp.name, strerror( - status ) );

          /* Free XDR 'gr' */
          if ( clnt_freeres( client,
                             (xdrproc_t) xdr_fs_get_ret_t, (caddr_t) gr ) ==0)
            fprintf( stderr,
                     "ERROR: Could not free RPC/XDR result 'gr' (%p)\n",
                     gr );

          break;
        }
    }

  gettimeofday( &t2, NULL );
  timersub( &t2, &t0, &tr );
  ts = (unsigned long long)((double)tr.tv_sec * 1000.0f + tr.tv_usec / 1000.0f);
  if ( count > ( 1024 * 1024 ) )
    {
      strncpy( unity, "Mb", 3 );
      size = 1.0f * count / 1024.0f / 1024.0f;
    }
  else if ( count > 1024 )
    {
      strncpy( unity, "Kb", 3 );
      size = 1.0f * count / 1024.0f;
    }
  else
    {
      strncpy( unity, "b", 3 );
      size = 1.0f * count;
    }
  {
    double tv;
    char tu[ 3 ] = "";

    if ( ts > 1000 * 60 * 60 )
      {
        strncpy( tu, "H", 3 );
        tv = (double) ts / 1000 * 60 * 60;
      }
    else if ( ts > 1000 * 60 )
      {
        strncpy( tu, "M", 3 );
        tv = (double) ts / 1000 * 60;
      }
    else if ( ts > 1000 )
      {
        strncpy( tu, "s", 3 );
        tv = (double) ts / 1000;
      }
    else
      {
        strncpy( tu, "ms", 3 );
        tv = (double) ts;
      }

    rate = (double) count / ( ts / 1000.0f );

    if ( rate > ( 1024.0f * 1024.0f ) )
      {
        rate /= 1024.0f * 1024.0f;
        strncpy( ru, "Mb", 3 );
      }
    else if ( rate > 1024.0f )
      {
        rate /= 1024.0f;
        strncpy( ru, "Kb", 3 );
      }
    else
        strncpy( ru, "b", 3 );

    fprintf( stderr, "Got file '%s' (%0.1f %s) in %0.1f %s. [%0.1f%s/s]\n",
             gp.name, size, unity, tv, tu,
             rate, ru );
  }

  gettimeofday( &t1, NULL );
  fclose( fd );
}


void
fsprog_1( char *host )
{
  char *s, *line;
#ifndef DEBUG
  if ( client )
    clnt_destroy( client );
  client = clnt_create( host, FSPROG, FSVERS, "udp" );
  if ( client == NULL )
    {
      clnt_pcreateerror( host );
      exit( EXIT_FAILURE );
    }
#endif  /* DEBUG */

  ui_init();

  while ( ui_stop == 0 )
    {
      if ( ( line = readline( "> " ) ) == NULL )
        {
          ui_comm_quit( "" );
          break;
        }

      s = stripwhite( line );

      if ( *s != '\0' )
        {
          add_history( s );
          ui_execute_line( s );
        }

      free( line );
    }

#ifndef DEBUG
  clnt_destroy( client );
#endif   /* DEBUG */

}

int main( int argc, char *argv[] )
{
  char *host;

  if ( argc < 2 )
    {
      printf( "usage: %s server_host\n", argv[ 0 ] );
      exit( EXIT_FAILURE );
  }
  host = argv[ 1 ];
  fsprog_1( host );
  exit( EXIT_SUCCESS );
}
```

4 Apêndice B

Listing 4: GetFileResult.java — Estrutura de dados para o retorno do método `getFile()`.

```
import java.io.Serializable;

public class GetFileResult implements Serializable
{
    public byte      buffer[];
    public int       count  = 0;
    public int       status = 0;
    public Exception e;

    public GetFileResult( int buffer_count )
    {
        buffer = new byte[ buffer_count ];
    }
}
```

Listing 5: FileServerInterface.java— Interface a ser utilizada para relacionamento com o servidor.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface FileServerInterface extends Remote
{
    /**
     * show credits
     *
     * @return credits
     */
    public String cred()
        throws RemoteException;



    /**
     * get file chunck from server.
     *
     * @param offset chunck init pos
     * @param count  how many bytes to read (maximum)
     *
     * @return status=0 on EOF, status=-1 on Failure, status=1 on success.
     *         If failure, exception is kept in e, otherwise count bytes are
     *         stored in buffer.
     */
    public GetFileResult getFile( String name, long offset, int count )
        throws RemoteException;



    /**
     * list dir contents
     *
     * @param name directory name/path
     *
     * @return list of strings
     */
    public String[] list( String name )
        throws RemoteException;



    /**
     * test file existence of given file
     *
     * @param name file to test
     *
     * @return true if exists, false otherwise
     */
    public boolean exists( String name )
        throws RemoteException;
}
```

Listing 6: FileServer.java— Implementação do servidor.

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

public class FileServer
    extends UnicastRemoteObject
    implements FileServerInterface
{
    public String cwd;

    /**
     * show credits
     *
     * @return credits
     */
    public String cred()
        throws RemoteException
    {
        return new String("Gustavo Sverzut Barbieri <ra008849@ic.unicamp.br>");
    }

    /**
     * get file chunck from server.
     *
     * @param offset chunck init pos
     * @param count  how many bytes to read (maximum)
     *
     * @return status=0 on EOF, status=-1 on Failure, status=1 on success.
     *         If failure, exception is kept in e, otherwise count bytes are
     *         stored in buffer.
     */
    public GetFileResult getFile( String name, long offset, int count )
        throws RemoteException
    {
        GetFileResult gr = new GetFileResult( count );
        try {
            int i;
            FileInputStream file = new FileInputStream( new File( cwd, name ));

            file.skip( offset );
            // JAVA SUX !!!
            // Implement it this way or loose a year reading the docs...
            gr.count = 0;
            while ( true )
                {
                    i = file.read();
                    if ( i == -1 )
                        {
                            gr.status =  0;
                            break;
                        }
                    else
                        {
                            gr.status = 1;
                            gr.buffer[ gr.count ] = (byte)i;
                            gr.count ++;
                        }
                    if ( gr.count >= gr.buffer.length )
                        break;
                }
            file.close();
        }
        catch( Exception e ) {
            gr.status = -1;
            gr.e      = e;
        }

        return gr;
    }


    /**
     * list dir contents
     *
     * @param name directory name/path
     *
     * @return list of strings
     */
    public String[] list( String name )
        throws RemoteException
    {
        try {
            File dir = new File( cwd, name );
            return dir.list();
        } catch ( Exception e ) {
            System.out.println( "Exception: " + e );
            return null;
        }
    }



    /**
     * test file existence of given file
     *
     * @param name file to test
     *
     * @return true if exists, false otherwise
     */
    public boolean exists( String name )
        throws RemoteException
    {
        File file = new File( cwd, name );
        return file.exists();
    }


    public FileServer() throws RemoteException {
        super();
        cwd = System.getProperty( "user.dir" );
    }



    public static void main( String args[] )
    {
        String hostname;
        String name;

        if ( System.getSecurityManager() == null )
            System.setSecurityManager( new RMISecurityManager() );

        try {
            hostname = InetAddress.getLocalHost().getHostName();
        } catch ( java.net.UnknownHostException e ) {
            hostname = "localhost";
        }

        name = "//" + hostname + "/FileServer";
        try {
            FileServer fs = new FileServer();
            Naming.rebind( name, fs );
            System.err.println( "Server Launched." );

        } catch ( Exception e )
            {
                System.err.println( "ComputeEngine exception: " +
                                    e.getMessage() );
                e.printStackTrace();
            }
        }
}
```

Listing 7: FileClient.java— Implementação do Cliente.

```
import java.io.*;
import java.rmi.*;

public class FileClient
{
    static int BLOCK_SIZE     = 8000;
    FileServerInterface fs;

    public FileClient( String host )
    {

        if ( System.getSecurityManager() == null )
            System.setSecurityManager( new RMISecurityManager() );

        try {
            String name = "//" + host + "/FileServer";
            fs = (FileServerInterface) Naming.lookup( name );
        } catch (Exception e) {
            System.err.println( "Exception: " + e );
        }
    }


    public void cred()
    {
        while ( true )
            {
                try {
                    String cred = fs.cred();
                    System.out.println( "CRED: " + cred );
                    break;
                } catch ( RemoteException e ) {
                    System.err.println( "Connection Error: " + e );
                    System.err.println( "Trying Again..." );
                }
            }
    }



    public void getFile( String src, String dst, boolean statistics )
    {
        long offset = 0;
        GetFileResult gr;
        FileOutputStream file;
        long count = 0;
        long t0, t1, t2;

        try {
            file = new FileOutputStream( dst );
        } catch ( FileNotFoundException e ) {
            System.err.println( "ERROR: " + e );
            return;
        }

        t0 = System.currentTimeMillis();
        while ( true )
            {
                while ( true )
                    {
                        try {
                            t1 = System.currentTimeMillis();
                            gr = fs.getFile( src, offset, BLOCK_SIZE );
                            count += gr.count;
                            if ( statistics )
                                {
                                    t2 = System.currentTimeMillis();
                                    System.out.println( "STAT: Got " +
                                                        gr.count +
                                                        " bytes in "+
                                                        (t2 - t1) + " ms." );
                                }

                            break;
                        } catch ( RemoteException e ) {
                            System.err.println( "Connection Error: " + e );
                            System.err.println( "Trying Again..." );
                        }
                    }



                if ( gr.status >= 0 )
                    {
                        offset += gr.buffer.length;
                        try {
                            file.write( gr.buffer, 0, gr.count );
                            if ( gr.status == 0 )
                                break;

                        } catch ( IOException e ) {
                            System.err.println( "ERROR: " + e );
                            break;
                        }
                    }
                else
                    {
                        System.err.println( "ERROR: " + gr.e );
                        break;
                    }
        }

        t2 = System.currentTimeMillis();
        String unity;
        double rate;
        if ( count > 1024 * 1024 )
            {
                count /= ( 1024 * 1024 );
                unity = " Mb";
            }
        else if ( count > 1024 )
            {
                count /= 1024;
                unity = " Kb";
            }
        else
            unity = " b";

        rate = (float)count * 1000.0 / (t2 - t0);

        System.out.println( "Got '" + src + "' " +
                            count + unity + " in " +
                            (t2 - t0) + "ms  [" + rate + unity + "/s]" );


        try {
            file.close();
        } catch ( IOException e ) {
            System.err.println( "ERROR: " + e );
        }
    }

    public void list( String name )
    {
        int i;
        String dir[];
        while ( true )
            {
                try {
                    dir = fs.list( name );
                    break;
                } catch ( RemoteException e ) {
                    System.err.println( "Connection Error: " + e );
                    System.err.println( "Trying Again..." );
                }
            }

        if ( dir == null )
            System.err.println( "Could not list dir '" + name + "'" );
        else
            for ( i=0; i < dir.length; i++ )
                System.out.println( dir[ i ] );
    }

    public void exists( String name )
    {
        String result = new String();

        boolean r;
        while ( true )
            {
                try {
                    r =  fs.exists( name );
                    break;
                } catch ( RemoteException e ) {
                    System.err.println( "Connection Error: " + e );
                    System.err.println( "Trying Again..." );
                }
            }
        result = "File '" + name + "' does ";
        if ( ! r )
            result += "NOT ";

        result += "exist.";

        System.out.println( result );
    }


    public static String getLine() {
        String inputLine = "";


        return inputLine;
    }


    public static void main( String[] args ) {

        if ( args.length < 1 )
            {
                System.err.println( "You must provide the server name!" );
                return;
            }
        FileClient fc = new FileClient( args[ 0 ] );

        BufferedReader in = new BufferedReader( new InputStreamReader( System.in ), 1 );

        String line = "";
        String a[];
        while ( true )
            {
                System.out.print( "> " );
                try {
                    line = in.readLine();
                } catch ( IOException e ) {
                    System.err.println( "Exception: " + e );
                    continue;
                }
                line.trim();
                a = line.split( " " );

                if ( a.length < 1 )
                    continue;

                if ( ( a[ 0 ].compareTo( "quit" ) == 0 ) ||
                     ( a[ 0 ].compareTo( "exit" ) == 0 ) )
                    break;
                else if ( a[ 0 ].compareTo( "list" ) == 0 )
                    {
                        String dir = ".";
                        if ( a.length > 1 )
                            dir = a[ 1 ];

                        fc.list( dir );
                    }
                else if ( a[ 0 ].compareTo( "exist" ) == 0 )
                    {
                        if ( a.length < 2 )
                            {
                                System.out.println( "Filename required!" );
                                continue;
                            }
                        fc.exists( a[ 1 ] );
                    }
                else if ( a[ 0 ].compareTo( "get" ) == 0 )
                    {
                        String src;
                        String dst;
                        if ( a.length == 2 )
                            {
                                src = a[ 1 ];
                                dst = a[ 1 ];
                            }
                        else if ( a.length > 2 )
                            {
                                src = a[ 1 ];
                                dst = a[ 2 ];
                            }
                        else
                            {
                                System.out.println("Missing source filename!");
                                continue;
                            }
                        fc.getFile( src, dst, false );
                    }
                else if ( a[ 0 ].compareTo( "getstat" ) == 0 )
                    {
                        String src;
                        String dst;
                        if ( a.length == 2 )
```

```
                                    {
                                         src = a[ 1 ];
                                         dst = a[ 1 ];
                                    }
                               else if ( a.length > 2 )
                                    {
                                         src = a[ 1 ];
                                         dst = a[ 2 ];
                                    }
                               else
                                    {
                                         System.out.println("Missing source filename!");
                                         continue;
                                    }
                               fc.getFile( src, dst, true );
                          }
                     else if ( a[ 0 ].compareTo( "cred" ) == 0 )
                          fc.cred();

                }

     }
}
```