Low level tools to implement structured parallel/distributed programming frameworks

Marco Danelutto
Dept. Computer Science
Univ. of Pisa

Master Degree (Laurea Magistrale) in
Computer Science and Networking
Academic Year 2009-2010

POSIX TCP/IP

- **Widely available platform**
- **POSIX**
  - *Standard for processes, threads, command line tools (Unix, Linux, BSD, Windows (with libraries), MAC OS X)*
- **TCP/IP**
  - *Standard for interconnecting machines*
    - Connection and connectionless modes
    - Reliable and unreliable modes

Background & perspectives

- **From Computer Network courses and labs**
  - *Socket concepts API*
- **From Operating Systems courses and labs**
  - *Thread, Process concepts and API*
- **From Programming labs**
  - *Tools concepts and usage (e.g. ssh/scp)*
- **Perspective**
  - *Use of these tools to support structured parallel programming*
    - We act as system programmers rather than application programmers
    - To provide high level programming frameworks to application programmers

Briefly recalling APIs

- **API for**
  - *TCP sockets (connection oriented)*
  - *UDP sockets (connectionless)*
  - *Processes*
  - *Threads*
- **Command line suite**
  - *ssh, scp, process management (ps, kill, ^Z, fg/bg)*
  - *uptime, …*

Socket TCP

- **Asymmetric calls**
  - *Server side*
    - Open a socket
    - Accept connections
    - Read and write from connected sockets
  - *Client side*
    - Open a socket
    - Connect to server socket
    - Read and write to connected socket

Java Socket (Server side)

- **ServerSocket ss = new ServerSocket(int port);**
- **Socket cs = ss.accept();**
- **InputStream is = cs.getInputStream();**
- **OutputStream os = cs.getOutputStream();**
- **cs.close(); ss.close();**

- *with Exceptions to be handled*
- *Read and write to In/Out putStream are low level calls*
- *Input and Output stream may be used to create higher level stream objects (e.g. Buffered, Reader, …)*
  - But this introduces buffering and therefore more asynchronous behaviour (KEEP IT IN MIND!)

Java Sockets (client side)

- **Socket cs = new Socket (InetAddress ia, int port);**
- **InputStream is = cs.getInputStream();**
- **OutputStream os = cs.getOutputStream();**
- **cs.close();**
  - *As before (input/output streams)*

Sample code : Server

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

public class StupidServer {

    public static void main(String [] args) {

     int porta;
try {
    porta = Integer.parseInt(args[0]);
} catch(ArrayIndexOutOfBoundsException e) { porta = 43000; }

ServerSocket ss = null;
try {
    ss = new ServerSocket(porta);
} catch(BindException e) { System.out.println("busy port!");
} catch(IOException e) { System.out.println(e);}

Socket s  = null;
try {
    s = ss.accept();
} catch(IOException e) {  System.out.println(e);}
```

Server (2)

```
InputStream is = null;
try {
    is = s.getInputStream();
} catch(IOException e) {System.out.println(e);}

final int MAX = 1024;
buffer = new byte[MAX];
int letti = 0;
try {
    letti = is.read(buffer,0,MAX);
} catch(IOException e) { System.out.println(e);}
try {
    s.close();
} catch(Exception e) {System.out.println(e);}

String mesg = new String(buffer,0,letti);
System.out.println(">>> "+mesg);
return;
}
```

Sample code: client

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

public class SimpleClient {

    public static void main(String [] args) {

     String stringa = new String("msg");
     String host;
     try {
         host = args[0];
     } catch(ArrayIndexOutOfBoundsException e) { host = "localhost"; }
     int porta;
     try {
         porta = Integer.parseInt(args[1]);
     } catch(ArrayIndexOutOfBoundsException e) { porta = 43000; }

     Socket s = null;
     try {
         s = new Socket(host,porta);
     } catch(UnknownHostException e) {
         System.out.println("cannot resolve hostname!"); } catch(IOException e) {
         System.out.println(e);}
```

Client (2)

```
OutputStream os = null;
try {
    os = s.getOutputStream();
} catch(IOException e) { System.out.println("getting output stream " + e);
}

byte[] buffer = stringa.getBytes();
try {
    os.write(buffer,0,buffer.length);
    } catch(IOException e) {
        System.out.println("writing message " + e);
    }
    try {
        s.close();
    } catch(Exception e) {
        System.out.println("closing socket " + e);
    }
    String mesg = new String(buffer,0,letti);
    System.out.println(stringa+" "+mesg);
    }

}
```

C sockets (server side)

- int s = socket(int domain, int type, int protocol);
- int retcode = bind(int s, const struct sockaddr *address, socklen_t address_len);
- int acceptedSocket = accept(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);
- read(acceptedSocket, buffer, len);
  write(acceptedSocket, buffer, len);
- close(acceptedSocket);

C sockets (client side)

- **int s = socket(int domain, int type, int protocol);**
- **connect(int socket, const struct sockaddr *address,**
  socklen_t address_len);
- **read(s,buffer,len); write(s,buffer,len);**
- **close(s);**
  - *as all POSIX system calls*
    - return -1 in case of error, error king in global **errno** variable

Sample usage: server

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>

#define MAXBUF     8192
#define SERVERPORT 3005
#define NOMESERVER "servername.di.unipi.it"

int main(int argc, char * argv[])
{
  int server_socket,connect_socket;
  unsigned int client_addr_len;
  int retcode;
  struct sockaddr_in client_addr, server_addr;
  char line[MAXBUF];

  server_socket = socket(AF_INET,SOCK_STREAM,0);
  if(server_socket == -1) { perror("opening server socket"); return(-1);}

  server_addr.sin_family = AF_INET;
  server_addr.sin_port   = htons(SERVERPORT);
```

Server (2)

```
memcpy(&server_addr.sin_addr,(gethostbyname(NOMESERVER)->h_addr),
       sizeof(server_addr.sin_addr));
retcode = bind(server_socket,
               (struct sockaddr *)&server_addr,
               sizeof(server_addr));
if(retcode == -1) { perror("publishing socket");return(-1); }
listen(server_socket,1);
client_addr_len = sizeof(client_addr);

while((connect_socket =
          accept(server_socket,
                 (struct sockaddr *) & client_addr,
                 &client_addr_len)) != -1) {
    while((retcode = read(connect_socket,line,MAXBUF)) != 0) {
    write(fileno(stdout),line,retcode);
    if(line[retcode]=='\0')
      break;
    }
    printf("\nEnd of message. Closing connection\n");
    close(connect_socket);
    sleep(20);
  }

  printf("End work ... \n");
  close(server_socket);
  return(0);
}
```

Sample usage: client

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>

#define MAXBUF     8192
#define SERVERPORT 3005
#define NOMESERVER "servername.di.unipi.it"

int main(int argc, char * argv[])
{
  int client_socket;
  int retcode;
  struct sockaddr_in  server_addr;
  char message[MAXBUF];

  printf("Client (%d): initialization\n",getpid());
  client_socket = socket(AF_INET,SOCK_STREAM,0);
  if(client_socket == -1) {
    perror("opening client socket"); return(-1);}
```

Client (2)

```
server_addr.sin_family = AF_INET;
server_addr.sin_port   = htons(SERVERPORT);
memcpy(&server_addr.sin_addr,(gethostbyname(NOMESERVER)->h_addr),
       sizeof(server_addr.sin_addr));
retcode = connect(client_socket,
                  (struct sockaddr *)&server_addr,
                  sizeof(server_addr));
if(retcode == -1) { perror("connecting to socket");return(-1); }

  sprintf(message,"Message from client process %d: ciao server!",getpid());
  retcode = write(client_socket,message,strlen(message));
  if(retcode == -1) { perror("writing message"); return(-3); }
  // retcode = read(client_socket,message,strlen(message));
  // message[retcode-1] = '\0';
  // printf("--> %s \n", message);
  close(client_socket);
  return(0);
}
```