Client/Server Communication Concepts for a Data Communications Course

Sub Ramakrishnan, Mohammad B. Dadfar

Department of Computer Science
Bowling Green State University
Bowling Green, Ohio 43403
phone: (419) 372 2337 fax: 419 372 8061
email: datacomm@cs.bgsu.edu

Abstract

As the computing environment is shifting towards client-server computing, there is a vital need for people with expertise in internet applications and network programming. We feel this area will continue to attract more attention in the future. Universities are addressing this need by offering courses in computer networking and data communications.

An undergraduate course in data communications appears to be a suitable vehicle for discussing client-server concepts and internet applications due to the direct relationship between client-server concepts and computer networks. These ideas are often complemented by hands-on experience in writing client-server applications.

Traditionally, UNIX machines provide support for process to process communication within the *same* host by using a *BSD protocol stack.* The TCP/IP protocol suite is a direct extension of this idea except that it provides for communication between two hosts. In this paper, we propose a project that illustrates the interoperability of these two protocol architectures and the client-server relationships all in a networked environment. The client and the server reside on two *different* hosts. The application user is under the illusion that the communication is over the BSD suite of protocols. Basically, we employ a TCP/IP paradigm at the backend of the two hosts, and a BSD stack at the front end of the client, and the server. Thus, communication from the client goes over client-side BSD protocol stack to the server-side BSD protocol stack, via the intermediate TCP/IP protocol stack.

1. Introduction

The use and growth of the internet has revolutionized the way in which information is organized, stored, and presented to end users. The internet has created a surge in enrollment in data communications courses offered at the undergraduate level. Students realize that most employers expect computer science graduates to have a good understanding of data communications and networking concepts. The curriculum for the data communications and computer networking courses are also revised often to integrate evolving technologies in this area.

Like other computer science departments around the country, we at Bowling Green State University have offered a data communications and networks course (CS 429) in our undergraduate program since early 1980s. This course is offered as a one semester course in which both seniors and some graduate students enroll. Our CS 205 - *Advanced Programming Concepts I* (in depth introduction to C++, interactive debugging tools, elementary data structures and dynamic storage management techniques) is a prerequisite to CS 429. A majority of our students have also had a one credit hour course (Introduction to UNIX). At this point students enrolled in data communications course are familiar with the UNIX environment.

In the CS 429 course students study both the theory and application aspects of data communications. The course deals with a variety of topics such as communication hardware and software; network topologies, architecture and protocols; transmission media and impairments; local area networks; wide area networks; electrical interfaces; multiplexing; switching techniques; network management and operations. A good discussion of these topics can be found in standard textbooks [2, 4, 7, 10, 11]. We spend over two weeks on the TCP/IP protocol stack and client-server concepts.

The client-server paradigm forms a solid basis for network applications. The internet and its functionality is studied and the role of application software as a useful part of communication across an internet is investigated. In general, communication across an internet involves two cooperating application programs where one application initiates its willingness to communicate with another application on a different computer. The second application responds to the first application's request. This type of communication between network applications follows the client-server paradigm. A client application actively initiates communications by sending requests to a server application that is waiting passively to receive specific type of messages and respond to these incoming requests. From previous courses, the students have limited exposure to interprocess communication features of the UNIX environment where both processes reside on the same computer. The TCP/IP protocol suite is introduced as an extension of this type of communication where processes reside on different hosts and communicate across an internet.

Traditionally, offering of data communications courses include assigning suitable projects designed to provide students with hands-on experience in data communications and network applications. These class projects are intended to help students obtain an insight into the nature of data communications and a better understanding of the theoretical concepts as well as filling some of the gaps which may exist covering the topics. Many class projects have been proposed in the literature [1, 3, 5, 6, 8, 9].

Our CS 429 students are typically assigned three to five homeworks and three to five lab projects. The lab projects in our offering include: *i*) simulation of a simple network, *ii*) terminal emulation, *iii*) performance evaluation of heterogeneous networks using a simulation tool, and *iv*) client-server projects. This paper describes one of the client-server projects we have assigned in our course. This project illustrates the ideas of interoperability of network protocols and the impact of client-server environment on operating system services and user programs.

In the next section, we describe the project. In Section 3, we elaborate on the extensions to this project so it may be used in other institutions. Concluding remarks are given in Section 4.

2. Project Description

The socket abstraction is a general purpose mechanism for client-server communication. The socket control block stores state information about the endpoints of a network communication path. The socket abstraction can be used for a number of different protocol stacks including TCP/IP, Berkeley BSD, OSI, SNA and so on [2, 10]. The protocol stacks are similar in some ways and yet provide some unique functionalities. A specific protocol stack can be chosen by specifying it as a parameter in the socket primitive.

In this paper we describe a client-server application project that uses two different protocol stacks in a client-server application. The project brings out the interoperability issues in operating over multi protocol stacks. To keep the discussion simple, we will use the BSD and TCP/IP stacks.

The BSD stack is based on the *AF_UNIX* domain [10]; reading and writing data is just like doing a file I/O except that the data written by one process is sent directly to a buffer in the process that owns the socket at the other end. There are several reasons for using the BSD stack in our project. They are heavily used in any Berkeley derived kernel, such as in UNIX pipes and the X window system but their use is *under the cover* and most users are unaware of their presence. They are also used to pass descriptors between processes which is a powerful technique for interprocess communication. The only constraint of the BSD stack is that the client and server processes reside on the same system. The socket interface between the client and server serves as a bidirectional pipe.

The TCP/IP or Internet stack [11] is more widely known and enables the server to reside on the same system as the client or on a different system that is reachable from the client system.

The students build a client-server application using the two protocol stacks, as illustrated in Figure 1. The client and the server reside on two *different* hosts. The application user is under the illusion that the communication is over the BSD protocol stack. Basically, we employ a TCP/IP paradigm at the backend of the two hosts, and BSD stack at the front end of the client, and the server. Thus, communication from the client goes over client-side protocol stack to the server-side protocol stack, via the intermediate TCP/IP protocol stack. The application provides simple echo service; what is transmitted from the client is echoed by server and vice-versa. Communication is terminated by transmitting a control character. The project statement, as given to students, is shown in Appendix A.

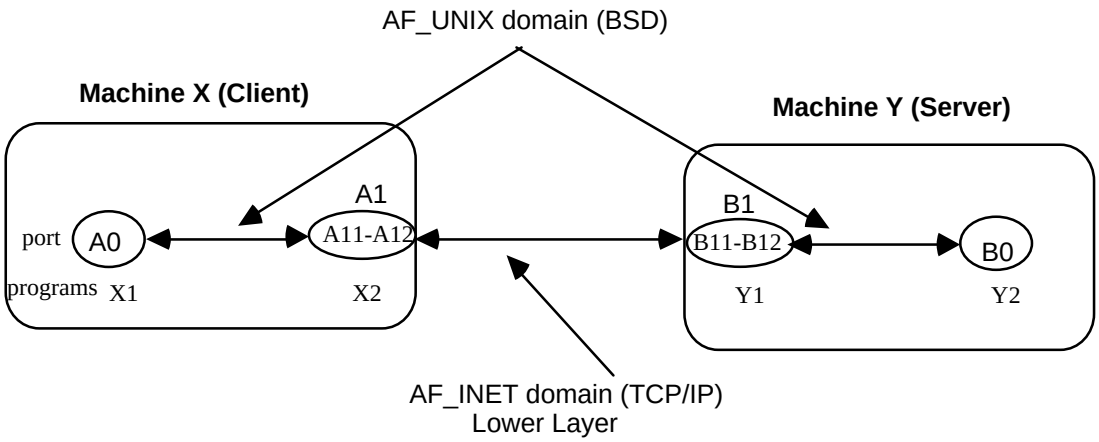

Figure 1: System View of the Project

3. Implementation Issues and Extensions

As shown in Figure 1, there are four programs, X1, X2, Y1, and Y2 that work together but are launched separately to create the illusion required in the assignment. The students quickly realize that one needs multiple clients and servers. Program X2 has two subprograms, a BSD server component (X2bsd-s) and a TCP/IP client component (X2tcp-c). Similarly, program Y1 has two subprograms, a TCP/IP server part (Y1tcp-s) and a BSD client (Y1bsd-c). Module X1 is a BSD stack client for BSD stack server X2bsd-s while module Y2 is a BSD stack server for BSD stack client Y1bsd-c. Further, X2tcp-c is a client of the TCP stack server Y1tcp-s. The modules are launched from right to left. Server Y2 is started first which provides a port number for Y1bsd-c to connect to. Then, Y1tcp-s provides a port number for X2tcp-c to connect to. Then, X2bsd-s provides a port for X1 to connect to. This is shown in Figure 2.

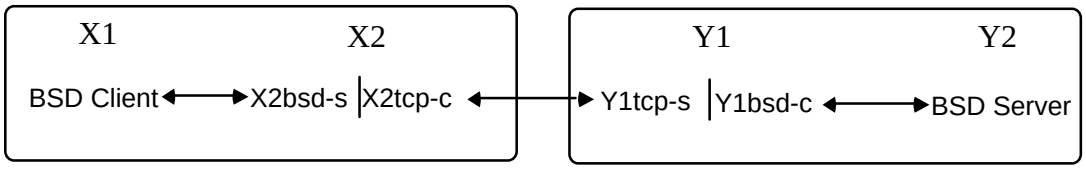

Figure 2: Components of programs

Another issue to resolve for students is how to communicate between X2bsd-s and X2tcp-c (and similarly between Y1tcp-s and Y1bsd-c). A simple approach, as noted in the previous paragraph, is to treat X2bsd-s and X2tcp-c as subprograms of program X2. (Similarly, Y1tcp-s and Y1bsd-c are subprograms of program Y1.) The advantage of this approach is that X2bsd-s and X2tcp-c *share common memory* and can pass data between them easily. X2bsd-s issues a read to receive client message from X1. Then, it hands over the data, through the shared memory, to X2tcp-c which then transmits the data over the TCP stack to Y1tcp-s and so on.

Now the implementation is straight forward. The client implementation requires the use of *socket*, *connect*, *read* and *write* calls. The server implementation requires the use of *socket*, *bind*,

listen and *accept* calls. For simplicity, there is only one instance of the client program X1. Thus, there is no need for the server side to spawn a new process (or thread) to handle a client request since there is only one client. For brevity, we do not include the solution code here. Readers may obtain the code from the authors.

Extension:

The project provides possibilities for a number of extension. We describe three simple extensions. (1) Asynchronous send and receive. The simple implementation described above requires that the client (X1) sends a message to which the server (Y2) responds. Then the client sends a second message to which the server responds and so on. However, in real life, it is not necessary that the server responds to each client message. Asynchronous implementation does not *block* on a server read. It can turn around and get the next client message if it has one. (2) allow multiple clients to call the server Y2. The server has to fork of a new process for each client connection. (3) The server is usually started first and waits for the client to call in. Integrate the server program with the internet daemon (*inetd*); the client's request is intercepted by the *inetd*, it then launches the appropriate server program. This allows for efficient use of system resources and is a common strategy for the deployment of well known server programs (such as *ftpd, fingerd* and so on).

4. Concluding Remarks

In this paper, we described a project suitable for a senior level course in data communications. The project employs multiple protocol suites. This internet based client-server application is interesting and educational since it demonstrates many ideas: how to build multi-protocol client-server applications, what is the difference between a client and a server, how can an internet application module serve both as a client and a server. We believe that projects like this which provide hands-on experience are necessary to reinforce theoretical concepts. We also outlined some extensions to this project.

Appendix: Problem Statement

Write a client-server application in C/C++ using the BSD protocol stack (*AF_UNIX* domain) for stream oriented communication. The client resides on machine X while the server resides on machine Y. Since the BSD stack requires that they both be on the same machine, you will simulate the BSD stack client-server environment as shown in Figure 1.

At the lower layer, use TCP/IP stack to establish communication between port number A1, on machine X with port number B1 on machine Y. At the user or application layer on either machine, use a BSD stack; port number A0 on machine X communicates with port number A11 on the same machine, while port number B12 on machine Y communicates with port number B0 on machine Y. Thus the user is given the illusion that the BSD stack is used for communication across two distinct machines.

You will need four modules, two on each machine. Note that the lower layers (or the TCP/IP client and server) could be written as a subprocedure of the corresponding user layer. Thus, on each machine, the two modules on that machine can have a parent-child relationship (use fork).

Submit the programs with a sample script file showing that it works. Messages from client go from A0 to B0 via A11, A12 - B11, B12 and printed on the server window. The server then echoes the message back to the client. The client terminates connection by sending a special *terminal* message.

Before you start coding, you should be familiar with a number of issues:
1. At the lower, TCP/IP, layer who is the client and why?
2. Useful system calls:
   `socket, connect, bind, listen, accept, read, write`.
3. Useful header files:
   `<sys/types.h>` and `<sys/socket.h>`

References

[1] Classen, R., Elizandro, D., and Smith, W., "An Undergraduate Data Communications Laboratory" ASEE Computers in Education Journal, Vol. IV, Number 1, January-March 1991, pp. 1-5.

[2] Comer, Douglas, "Computer Networks and Internet," Prentice-Hall, 1997.

[3] Dadfar, Mohammad and Evans, Stephen, "An Instructional Token Ring Model on the Macintosh Computer," ASEE Computers in Education Journal, Vol. IV, Number 1, January-March 1991, pp. 28-32.

[4] Halsall, Fred, "Data Communications, Computer Networks and Open Systems," (Fourth Edition), Addison-Wesley, 1996.

[5] Hughes, Larry, "Low-Cost Networks and Gateways for Teaching Data Communications," ACM SIGCSE Bulletin, Vol. 21, Number 1, February 1989, pp. 6-11.

[6] Kamel, K. and Riehl, A., "An Instructional Model to Build a Computer Network by Adding Nodes," ASEE Annual Conf. Proceedings, June 1992, pp. 1107-1111.

[7] Orfali, R., Harkey, D., and Edwards, J., "Essential Client/Server Survival Guide," Wiley & Sons, 1994

[8] Shay, William, "A Software Project for a Data Communications Course," ACM SIGCSE Bulletin, Vol. 23, Number 1, March 1991, pp. 15-20.

[9] Smith, Wayne, "The Design of An Inexpensive Undergraduate Data Communications Laboratory," ACM SIGCSE Bulletin, Vol. 23, Number 1, March 1991, pp. 273-276.

[10] Stevens, Richard, TCP/IP Illustrated Volume 3, Addison Wesley, 1994.

[11] Tanenbaum, A. S., "Computer Networks," (Third Edition), Prentice-Hall, 1996.

SUB RAMAKRISHNAN is an Associate Professor of Computer Science at Bowling Green State University. From 1985-1987, he held a visiting appointment with the Department of Computing Science, University of Alberta, Edmonton, Alberta. Dr. Ramakrishnan's research interests include distributed computing, performance evaluation, parallel simulation, and fault-tolerant systems.

MOHAMMAD B. DADFAR is an Associate Professor in the Computer Science Department at Bowling Green State University. His research interests include Computer Extension and Analysis of Perturbation Series, Scheduling Algorithms, and Computers in Education. He currently teaches undergraduate and graduate courses in data communications, operating systems, and computer algorithms. He is a member of ACM, IEEE, ASEE, and SIAM.