Application of Client/Server Paradigm and Web Technologies in a Networking Course

Mohammad B. Dadfar, Jeffrey A. Francis, Sub Ramakrishnan

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

Commercial organizations realize the importance of providing information over the internet to customers about the services they provide. Today, the world wide web serves as the primary vehicle to get this information to the customer while the back end database and client-server technologies are used to process the information and deliver them to the web client. Therefore, the need for people with expertise in the areas of client-server technology and web foundations is becoming increasingly more important.

An undergraduate course in data communications and networking can provide students with conceptual information of the client-server paradigm along with providing students with some hands-on experience. By building web enabled client-server applications as part of their classroom projects, students will be able to bridge the relationship between client-server concepts and delivery of information over the internet. We believe that an application to query a well known server, say, *finger* using a web based client will provide them the necessary hands-on experience.

A project that implements the above ideas begins with building a client form on the world wide web in HTML code. Next, a Common Gateway Application (CGI) is developed in *perl* language to extract the form data. A network module is then used to transmit the client data to the server process, whether the server process resides on the same computer or on a different computer. The server, which is not implemented by the student, receives the client data and processes it and then generates a response in HTML code. The response is then transmitted back to the client via the same network module that carried the client data to the server processes. Finally, the web client renders the response on the screen.

This simple project helps students to visualize the flow of information in a real world application scenario. It also integrates nicely with the concepts introduced in class: TCP/IP protocol suite, client-server paradigm, and functional details of well known application servers such as *finger*. By requiring that the student implementation of the client software communicates with the server software from the OS manufacturer (traditional services such as *finger* are usually bundled with

the OS), the students are exposed to interface and multivendor interoperability issues. Finally, the project is modular and is assigned as a group project which can help students in promoting interpersonal, leadership and project management skills.

1. Introduction

As the computer environment is shifting towards network based computing, there is a vital need for people with expertise in data communications and network based applications. Computer Science departments around the country have recognized this need and have offered a course in this area at the undergraduate level. Our department like other CS departments elsewhere has traditionally offered a data communications course (CS 429) that covers protocols and internet software and hardware architectures. Details of this course and some hands-on project activities are described elsewhere [2, 5].

In this paper we take a look at the impact of the changing facets of communications technology, the impact of web technology on businesses and propose a *new* course offering at the undergraduate level for a discussion of these concepts. We then describe two projects suitable for this new course offering. The new course will serve as a *prerequisite* to more advanced courses such as CS 429.

Today's commercial organizations realize the importance of providing information about their company to potential customers. The increase in the use and availability of the world wide web has changed the process of delivering this information to their consumers. Client-server technologies along with company's back end databases are being utilized to both process and provide information to web clients. Therefore the need for individuals with web skills and client-server communication skills is becoming increasingly more important. Recognizing this need, we feel that all computer science graduates should have a basic understanding of computer networks, client-server computing and delivery of application services over the web. Building such applications requires not only networking skills but also awareness of operating system issues and process management.

Starting Fall 1996, we are offering a new course, Operating Systems and Networks (CS 327). Unlike CS 429, this new course is *mandatory* for all of our students and is typically taken at the Junior level. The course introduces them to the basics of the operating systems and networks and gets them excited in these fields with some hands-on exercises. However, students wishing to know more of operating systems and/or networking concepts can take more advanced offerings such as CS 429 or a course on operating systems. CS 327 is a prerequisite for these advanced courses.

CS 327 is not an in-depth introduction to operating systems or networks. Instead, it attempts to introduce some concepts of operating systems and networking. In the operating systems area, topics covered include process management, concurrent processes and process scheduling (see Chapter 2 of [7]). In the networking area, we cover protocol architecture, TCP/IP suite, brief overview of broadband services, client-server communication and web enabling applications [1, 4].

In this paper, we discuss two projects used recently in CS 327. The first one is a client-server application while the second project provides a web interface to the client. The projects are implemented on a UNIX platform and use C/C++ for the first assignment, and HTML and *perl* for the second project. The students are allowed to work in groups of up to four.

The objective of the two assignments are the following: (1) The project should integrate conceptual and theoretical information presented in the classroom and actual application implementation. (2) The project should enhance the students knowledge of network use and communication protocols. (3) The project should provide the opportunity to build both leadership and interpersonal communication skills along with teaching them the value of working together as a team to complete a project.

In the next two sections, we describe these two projects and the solution phase. Section 4 provides certain extensions to the projects followed by the concluding remarks.

2. A Client-Server Project

This project deals with the client-server paradigm. The students write a client-server communication application. The client communicates a message to the server and the server sends a response back to the client. A formal project description is given in Figure 1.

The project is simple for three reasons: (i) This is the first time the students have had any experience doing something like this over the network. (ii) UNIX socket primitives are a bit challenging for beginners. (iii) Because the course is an overview on networking and operating systems only the *functional* details of the primitives are necessary and more details on these primitives might be confusing. Hence, the instructor supplied some routines callable by the student implementation. The instructor routines shield some of the intricacies of some network communication primitives. (In a follow up course, CS 429, students may be asked to write such applications *without* using the instructor supplied routines.) This gives them the ability to quickly get a working version instead of getting involved in complex issues such as address structures, host and network formats. The instructor supplied primitives *connectToHost*, and *buildServer* that handle all these complex issues in a manner transparent to the user (see Figure 1).

The instructor provides the students with explanations of client and server roles in the client-server paradigm along with some sample code that students may study and dissect for their own learning. The instructor also stresses the importance of *man* pages to the students. In the class we discuss at length the various socket primitives in the assignment sheet. Since they already know host and port number concepts they can appreciate what is needed at the client and server ends. Students were able to get a working version within a matter of hours. They were excited about being able to communicate across machines.

A possible solution to the client side of this problem could be handled as follows. The client first needs to establish a socket so the server knows where to respond to. Then after successfully connecting to the server, communication can begin. To get successful connection though, the server must be started first to establish a port number that the client can connect to. The client then asks the user to enter the port number, machine name of the server and the number of

messages to send to the server. The client then connects to the server and enters a loop. Each time through the loop, the client sends a message to the server and receives a response which is then printed out on the screen. The client terminates once the specified number of messages are sent and received.

The server is invoked in a similar fashion. Once the server is invoked it asks the user to enter the number of messages that it will receive from the client. Since that client is the one establishing the connection, the server does *not* need to know the machine where the client exists nor the port number of the client. Then the server establishes a socket to allow data communication to take place. Once this is established, it needs to determine a port number for itself. This number is very valuable to the client since the client has to know this port number the server is listening for incoming connections. The server prints its port number following a call to *buildServer* . Following a call to *listen*, the server then enters a loop. During the loop, the server either receives and displays information from the client or it sends messages in response to the client message.

3. A Web Enabled Client

This project exposes students to writing web enabled clients. Again, we keep the task simple since students enrolled in the course are juniors and have just been exposed, for the first time, to OS and networking concepts. In this project, the students communicate with a well known server such as the *finger* daemon. The client sends a user name to be fingered to the server. The client then receives the reply. The client, however, is web enabled, and is launched using a browser. The user fills a form (that asks for machine and user to be fingered). Upon submit, the request is sent through a CGI application to the finger daemon. Students create a client form in HTML for the world wide web [3, 8]. The CGI is written in perl [9]. Once the CGI has started receiving data from the web server (*http*), it packages the data into HTML code and renders it to the user's display. A formal statement of this project is given in Figure 2.

This application first extracts the data from the HTML form. Once it gets this data, it can determine which machine it needs to connect to along with the port number of the desired server. It then establishes a TCP connection to that port. Once a successful connection is made, the network module can pass on the user name(s) to get information on specific users or it can pass a carriage return to get all the current users on the machine desired. The response is returned to the client via the same network module that passed the data to the server. A loop in the CGI perl program reads the data from the server until it reads a termination or EOF character from the server. The received data is then packaged into HTML code and finally given to the web client for display.

Like the first project simplicity was our primary goal. The instructor supplied the complete code for a simple application wherein the server just echoes the client data. It consists of a sample HTML module, a corresponding CGI module and a server module. The HTML form data is shown in Figure 3 and the corresponding CGI code is quite similar to the one described elsewhere (see Figure 3 of [6]). The server application is similar to the one given in the client-server project of Section 2.

4. Discussions and Concluding Remarks

In this paper, we have described two UNIX-based projects for a core CS course. The course provides an overview of operating systems and networking. We feel that the overall outcomes of these projects were both interesting and beneficial to the students. The students were able to bind concepts to implementations. The projects are extremely simple and yet they provide a new experience to the students. The projects help them to get their feet wet on these technologies and promote an interest so that they can learn more about these topics possibly by enrolling in more advanced courses. With more companies entering the client-server world, we believe that these skills are essential and add to their marketability. For brevity we do not include the complete code for the instructor supplied routines, and the student solutions. They can be obtained from the authors.

The projects can be enhanced and tailored to offerings of a similar course elsewhere. A simple extension to the client-server project is for the client to accept the basic parameters as command line arguments instead of reading from as input. The parameters are: *server port number*, *machine name* and the number of messages to be passed from the client to the server. The students know how to do this from basic C/C++ programming courses. Another extension relates to incorporating OS issues such as signal handlers in the client environment. Instead of terminating the application after exchanging a fixed number of messages entered from the command line, user terminates the application by generating a signal, such as pressing the interrupt key; the client catches the signal, enters a student written signal handler and sends a good-bye message to server and then closes the socket connection gracefully.

The web enabled application can also be modified in some ways. For example, one can assign even a simpler project than this; the client talks to the daytime server (instead of finger daemon). Note that the daytime server simply returns the *date* to the client. This project would be a straight forward extension of the instructor supplied solution. Implementation of the finger based application is a bit more complicated since the server may send multiple lines of message (such as user name, last login, mail outstanding status and so on) in response to fingering just one user.

Establish a *client/server* communication. Each time through the loop, the client (server) sends a message to the server (client) and displays the message received from the other side. Since some UNIX client/server calls are not easy to use, we provide you with some routines which are easier to use.

The messages displayed on the screen are as follows:

- Client message: Message #X from Client to Server will appear on the server's display.
- Server Message: Message #Y from Server to Client will appear on the client's display.

where #X, #Y are the loop index values.

The client module reads in three data items, the number of times the loop is executed, the machine the server is running on, and port number the server is running on. The server module reads in the number of times the loop is executed. Run server first to print the server port number.

Client routines:

- `int socket (AF_INET, SOCK_STREAM, 0)`: Allocate a **socket** and return the socket number. The returned value is used as the first parameter in the next set of routines.

- `int connectToHost (int socketNO, char * serverMachineName, int serverPortNO)`: connect to the server host given by serverMachineName (e.g. "alpha") which is on port number serverPortNO. The call blocks. Returns < 0 upon error.
- `int read (socketNO, FromServer, maxLength)`: Read (up to maxLength bytes) from the server and store it in FromServer. Actual number of bytes read is returned.
- `int write (int socketNO, char *ToServer, int lengthOfToServer)`: Write to the server lengthOfToServer bytes from the array ToServer. Returns < 0 upon error.
- `int close (int socketNO)`: close the connection. Returns < 0 upon error.

Server routines:

- `int socket (AF_INET,SOCK_STREAM,0)`: See above.
- `int buildServer (int socketNO, int serverPortNO)`: Request creation of the server at port serverPortNO (if 0, the system assigns the next available port number (preferred option)). Returns < 0 upon error, else returns the port number the server can be reached at! Prints server port number.
- `int listen (int socketNO, int maxNOClients)`: **block** until a client calls. maxNOClients (set it to, say, 2) is the limit for the number of clients this server is prepared to entertain. Unblock when a client calls. Also returns the number of clients trying to reach you.
- `int matchWithClient (socketNO)`: Now you have matched up with a client, and bingo, you can talk to that client using **read/write**. Returns a **new** socket number for talking to this client. Make sure you use this socket number as the first argument of the read/write calls.
- `int close(...)`: same as for client. Returns < 0 upon error.

Notes:

- Compile client as `cxx client.c clientMod.o`
  server as `cxx server.c serverMod.o`
  where client.c and server.c are the names of your programs. Please compile and run them on different machines. Use the following include files.
  ```
  #include  <sys/socket.h>;
  #include  <sys/types.h>;
  #include  "instructorHeaderFile.h";
  ```
- Create a *readme* file that explains how to run the program and any other features about your program for the instructor.
- Run the program for a variety of inputs.
- Hand-in a printed copy of readme file, program listing, input, output.
- Do *man* on `atoi`, `close`, `listen`, `read`, `socket`, and `write`.

Figure 1. Client-server project description

Many commercial institutions value the wealth of information in their customer and product databases. Today, companies are realizing that money can be made by on-line dissemination of their products and services over the web to their customers. The web is attractive because it provides a uniform interface across platforms and can integrate all of the services in a common framework.

A web client fills in a form which is initially processed by an application program known as Common Gateway Interface(CGI). This application then interacts with a service-process, possibly on another computer. This process services the clients' request and sends the result back to the CGI. The CGI delivers the data, in HTML form, to the client browser which then renders the document on the screen.

The Problem

Provide a web interface to the UNIX *finger* daemon server (RFC 1288). It is helpful to recap your implementation of the client-server project.

To Do List

1. HTML code for client form
2. CGI module to extract client data (*Perl* Language)
3. Network module to interact with server process.
4. Server code to process client data and generate response. You do NOT write this; instead, use a well known server implementation (*fingerd*).
5. Transmit HTML response back to client.

Notes:

- Turn in the URL, listing of both documents and a sample output.
- Resources (Sample **Echo** Application)
  - Echo Application - Form Interface — echo.html
  - Echo Application - CGI Module — echo.cgi
  - Echo Application - Server Module — server.c
  - Want to run Echo application? — echo.html
  - Web/HTML Documentation
  - Perl Documentation

Figure 2. The problem statement for web enabled clients

```
<HTML>
<HEAD>
<TITLE>Sample Form </TITLE>
</HEAD>

<H1><Center>Simple Echo Application </center></H1>

<!-- This form accepts i) Your name, ii) Server machine name, iii) server port number, iv) Message.
     It then sends Message through the CGI application to server at that machine/port #.
     Any response from server is rendered at the client browser.
-->

<FORM METHOD="POST" ACTION="CGI location of echo application">

<DL>
<DD>Your Name <INPUT TYPE="text" NAME="Name" SIZE=15 VALUE=""><BR>
<DD>Host to Connect to <INPUT TYPE="text" NAME="server_host_name"
    SIZE=25 VALUE="">
<DD>Port No <INPUT TYPE="text" NAME="server_port_no" SIZE=5
    VALUE="1100">
<DD>Message to Send <INPUT TYPE="text" NAME="Message" SIZE=25 VALUE="">

<P><H4>Start <B>Server</B> before hitting Submit</H4>
<P><INPUT TYPE="submit" VALUE="Submit Document">
</FORM>

<HR>
</BODY>
</HTML>
```

Figure 3. Web HTML form

References

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

[2] 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.

[3] Musciano, M. and Kennedy, B., "HTML: The Definite Guide," O'Reily, 1996.

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

[5] Ramakrishnan, Sub and Dadfar, Mohammad B. ,"Client/Server Communication Concepts for a Data Communications Course," ASEE 1997 Annual Conference. (submitted for publication)

[6] Ramakrishnan, Sub and Rao, Madhu, "Classroom Projects on Database Connectivity and the Web," 28th ACM SIGCSE Conference Proceedings, February 1997.

[7] Tanenbaum, A. S., "Modern Operating Systems," (Chapters 1 and 2), Prentice-Hall, 1992.

[8] NCSA, "The NCSA Beginner's Guide to HTML", on the web with URL:
"http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html"

[9] "perl Documentation,", on the web with URL:
"http://www.cs.bgsu.edu/~rama/perl_doc/index.html"

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.

JEFFREY A. FRANCIS received his Master's degree in Computer Science from Bowling Green State University in December, 1996. He is a local area network coordinator/programmer analyst for Variable Life Systems of Nationwide Insurance company, Columbus, Ohio. His responsibilities are creating standards for intranet use, develop intranet interfaces and active forms using HTML, perl and Java, and provide server maintenance.

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.