UDK 004.254:004.052(045)
Melnyk V.M., Pekh P.A., Melnyk K.V., Zhyharevych O.K.
Lutsk national technical university

SIGNIFICANCE OF THE SOCKET PROGRAMMING FOR THE LABORATORY WITH INTENSIVE DATA COMMUNICATIONS

Melnyk V.M., Pekh P.A., Melnyk K.V., Zhyharevych O.K. Significance of the socket programming for the laboratory with intensive data communications. Many courses based on data communications are connecting with no programming content. They are designed for computer science topics and should include programming. A lot of data communication courses with a programming component make use of serial ports on PCs while some of them deal with detailed network layer projects. UNIX socket programming allows the learners to deal with the same issues and problems, but in a context that is more to be useful and interesting. In addition, if classes with sockets are used with C++, only as much detail of socket operation as desired need be presented.
Keywords: data communications, socket programming, contention, C++ socket classes, server.

Мельник В.М., Пех П.А., Мельник К.В. Жигаревич О.К. Важливість сокетного програмування для лабораторій з інтенсивним обміном даних. Багато курсів, засновані на передачі даних, підключаються без програмного контенту. Вони створені для тем з області комп□ютерних наук і повинні включати програмування. Багато курсів на базі комунікації даних з компонентом програмування можуть використовувати послідовні порти на ПК, в той час як деякі з них взаємодіють з конкретними мережевими рівнями проектів. Програмування UNIX-сокетів дозволяє вивчаючим займатися тими ж питаннями і проблемами, але в контексті, що може бути більш корисним і цікавим. Крім того, представлення сокетних класів з використанням C++ повинно настільки деталізувати операції сокетів, скільки це необхідно.
Ключові слова: обмін даними, сокетне програмування, зв□язок, класи сокетів C++, сервер.

Мельник В.М., Пех П.А., Мельник К.В. Жигаревич О.К. Важность сокетного программирования для лабораторий с интенсивным обменом данных. Многие курсы, основанные на передаче данных, подключаются без содержания программирования. Они предназначены для информатики темы и должен включать программного контента. Много курсов на основании коммуникации данных с компонентом программирования могут использовать последовательные порты на ПК, в то время как некоторые из них взаимодействуют с конкретными слоями сетевых проектов. Программирование UNIX-сокетов позволяет учащимся заниматься теми же вопросами и проблемами, но в контексте, который является более полезным и интересным. Кроме того, представление сокетных классов с использованием C++ должно настолько детализировать операции сокетов, сколько это необходимо.
Ключевые слова: обмен данными, сокетное программирование, связь, классы сокетов C++, сервер.

Introduction. Data communications is a common part of most Multi Interface Socket (MIS) and Client Socket (CS) programs. As evidenced, the actual implementation of the course varies widely by the available text books variety. Many texts, oriented on toward MIS or CS, provide little or absolutely no laboratory activity. MIS programs tendency to emphasize on a data communications and networks management. Recent news lists postings indicate an emphasis on using data communications and investigations of the types and available communication styles. National or international cooperative projects are too popular and CS programs may use very technical texts or a broad text, such as used in [1], where principles, design approaches and standards are going to be highlighted. Obviously, an engineering program would have a much more extensive and detailed course/courses to investigate the physical and structural data communication aspects. The course taught by the author requires for all CS majors. Students or outside users in the course maybe in any of the specialized tracks (artificial intelligence, business information systems, graphics or scientific programming) as well as more generic CS major.

The possible laboratory experiences types are also wide-ranging. The ″global cooperation model″ teaches how data communications works by forcing students to use sophisticated communication mechanisms and provides a basis for explaining how these systems function. It is possible to consider design alternatives, based on the available resources, by allowing learners to explore different physical or logical communication types.

At the other extreme are exercises that emphasize low-level physical understanding of data communications almost an engineering approach. A typical example could be the use of serial ports on PCs. In addition to writing code with aim to manipulate the physical hardware can be studied many more complicated concepts. In material the author has used in the past [2], file transfer assignments using a modified BiSynch protocol and implemented token rings. An alternative low-level approach is modelled by the NetCp software [3]. This laboratory tactic involves a large scale project based on the 0SI IS0 data

link layer developing. None of these approaches provide practical hands-on hardware experience. In addition to the exercises described in this work, the author assigns a project involving installation of hardware and software to add a PC to a network. A server can be installed and configured for extra credit. Such a project was continued after the socket model was adopted. Users placed in practice or in their first job to consider how the data communications course is important and they have typically not believed that PC serial port programming is to be so important.

The approach presented here is designed to provide a broad overview of data communication and network issues to students. The goals for the laboratory part of the data communication course also were presented later in publications.

UNIX sockets. Simply say, sockets area mechanism by which messages may be sent between processes on the same or different machines. If the processes are located on the same machine, the sockets may be used as *pipe*s. Internet sockets allow communication between processes running on different machines. The system calls are the same as file input/output. A typical attitude to socket programming is to create a process that opens a server socket port and listens for another process to attempt connection. A client can open a socket with the same port number as the server socket, requesting connection to the service. A connection is established when server hears the request from the client. Communication can now operate using the read() and write() system calls.

There are many types of standard protocols. Two of the most common are UDP (user datagram protocol) and TCP (transmission control protocol). Both protocols transmit packages of information between processes via socket. UDP does not put a guarantee, that data will be received or that a transmission of multiple packets will be received in order. TCP is a stream protocol that is reliable and sequenced. Practically, input and output to the programmer on a TCP socket appears as a byte stream from a terminal or a file. If TCP data cannot be successfully transmitted within a reasonable amount of time, then will be indicated an error. There is less overhead involved in UDP, but programming must be much more sophisticated if there is important orderly message receipt.

The socket connection between two processes usually is a connection between host-port pairs where the port number indicates a particular available service that is made. Many of the services commonly available via TCP sockets are recognizable acronyms: SMTP (Simple Mail Transport Protocol), NNTP (Network News Transport Protocol) and FIT (File Transport Protocol). *Telnet* and *rsh* are also additional socket services. UNIX provides a mechanism whereby the name of an available service is translated to a port number. Sockets are also used for the interprocess communication necessary in concurrent or parallel processing. Therefore, parallel processing assignments as well as data communications projects can be built on the same framework.

Advantages of sockets. One obvious disadvantage of using socket programming for the data communications lab is that they are less direct hardware interaction than with PC serial ports. However, most graduates will not be in situations where such detailed knowledge will be important. Even with the serial port approach the concepts have remained to some extent abstract to many students. The socket based approach has the advantage that the abstract sockets concepts (and practical uses such as mail, telnet, etc.) become much more concrete.

One advantage that PC based labs have had in the past is that they were inexpensive. However, there are at least two factors that balance this advantage. One is that UNIX workstations are now commonly available. PC labs can be converted to workstations by installing free UNIX versions. Most of socket assignments, given in a data communications course, are not compute intensively and do not require a graphical interface; workstations have not to be dedicated to the course as it would be true for PCs. Another factor is that even though PCs are relatively inexpensive, what happens practically is that older, less reliable, machines are assigned to a dedicated project such as a data communications lab. Our experience says that the machines we could use were quite unreliable.

Although the higher level nature of socket programming has been stressed as an advantage. It is possible to make assignments so much detailed as desired. Socket programming without any support software can require a great deal of low level understanding and manipulation. One simple modification

would be to base assignments on UDP packets rather than TCP one. Much additional programming (error checking via CRCS, sequence numbers, acknowledgment of receipt, negative acknowledgment for receipt of a bad packet) would be also necessary. With either UDP or TCP packets, properly designed handshaking mechanisms may be necessary for applications like file transfer.

With serial port assignments, lecture time was devoted to such low level concepts as control, status and data registers, and parallel to serial conversion. With a socket based approach can be discussed analogous concepts such as packet headers and network and machine byte order. If looked-for, many of the topics appropriate for serial port communication can be required for socket programs and many of the same assignments can be given. Even if high level applications are assigned, the learners must still understand the differences between streams and buffers.

Advantages of C++ socket classes. Many references provide details of socket communication [4,5,6]. These references provide examples and ideas for assignments. All of the establishing communications details, converting the communication into a buffered stream and error checking, can be done with UNIX system calls. Much low level understanding may be required to write applications that are stable. A well designed set of C++ classes can be constructed which will provide the full power of sockets with simple semantics requiring. It is possible to write up clients to established servers, event driven servers, polling servers, etc.

The author [7] provided the students with a set of C++ socket classes written and copyright by G. Swaminathan from Electrical Engineering Department. These routines have been written to work with GNU *libg++* and appear the same as the *iostream* library. These classes have functioned very well for the given assignments. The interface is the same as the *iostream* library and provides type-safe input and output. There are *sockstream* classes in the UDP and TCP domains as well as a *pipestream* class. The *sockbuf* classes are derived from the *streambuf* class of the *libg++ iostream* library. Thus, learners must learn about streams and buffers for non-socket input and output.

Sockbuf classes include error functions: ready checks, flush operations, and overflow, underflow, and *timeout* functions. There can also be set socket options, such as message routing, reuse of local address, broadcast etc. Therefore, socket detail may be included as it may be desired.

In this particular prospectus a side benefit of using these C++ classes is that clients (students) are required to use C++ in a junior/senior level course to help them retain skills gained at the freshman/sophomore level.

Assignments. In choosing assignments to give hour course (during a 1,5 year), several goals were desired. It was hoped to design a set of assignments which would require the student to write a client application, a server application a peer-to-peer application and also provide experience with some standard applications such as electronic mail and file transfer. In addition, the assignments should begin simply and become more complicated during the semester. They outlined below met these criteria.

The assignments were very well received by the students. They were perceived to be of practical interest and, at the same time, to be fun projects. Some of them, who don't have a history of applying themselves well to project assignments spent much effort on these assignments and produced good results. Specifically, we are giving five assignments here:

Assignment 1 □ socket client to SMTP server. Write a client program to connect with an SMTT server on a local or remote machine and send a mail message to a *userid*. The user need not be on either the local or remote machine. For example, the program might be named *smtp* and have two arguments: *hostname* and *userid*. A simple command line interface is required but learners were free to develop much more elaborate e-mail style interfaces. There must be used commands understood by SMTP. A subset are follows:

HELO `localname`	Identifies connecting machine □ `localname` is not needed □ some servers do not need HELO, but include it before do something.
HELP	Sends commands list.
MAIL FROM `name`	May be anything you wish □ it is not checked for validity.

RCPT TO: `name`	Recipient of mail □ need not be local `name`.
DATA	Allows entry of message □ terminate message with □.□ as only character on line.
QUIT	Disconnect.

This assignment, as well as others, teaches students how to do improper activities. The following warning was provided to them: *«Obviously one could do ill-mannered things with this program. For example, it could be sent a bunch of messages from Dafi Duck to some administrator. It would take some work, but the sender of these messages could be traced. Please, do not involve in such juvenile behavior»*. Some of them would argue that such assignments are too «dangerous». However, the students are learning how to manipulate sockets and could figure out how to send such mail on their own. For this it need to be ready to acknowledge the problem and announce a warning.

Assignment 2 □ simple network information server. Write a network server program which will behave as follows: accept commands from the input socket; interpret the commands and gather the information; and send the commands output to the output socket.

You will no need to write a client program for this assignment as the standard telnet client will provide the necessary functions. Telnet allows you to send information over a client to a server process and handles the return information printing. A selection of information providing system commands such as domain name, who, etc. was chosen. The system functions can be executed from within a C++ program. The difficult part is to take the commands output and send the output to the socket connected to the client. The output of the commands should be connected directly to the socket. Two approaches are suggested: using the *pipe*stream class and using a traditional C fork() to execute the system function which is connected by a user-constructed *pipe*.

Assignment 3 – peer-to-peer socket communications. Write a "chat" program which will execute as two idential programs. It should allow users to type information that will appear as output on the connected process. The two processes have to be connected via socket. The program will allow the user to connect to a certain process or to listen for another process or trying to connect to it.

The same program runs on both machines. Topics necessary for this assignment include: timeout on listen, creating a child process as done by many server programs, closing sockets and killing child processes. A finite-state transition model could be presented to help in this program design.

Assignment 4 □ file transfer □ client and server. Write a pair or programs to transfer files over a TCP/IP network socket connection. The first program should operate in much the same way as an FIT server. It should be run in the background and wait for a connection on a specified port. The second program will function in much the same way as an FIT client. Therefore, a user interface will be needed. Commands will be entered and sent to the server with noted responses and files may be transferred in either direction. The client program should accept the following commands with corresponding actions:

`ls`	list files on server
`put`	transfer file from client to server
`get`	transfer file from server to client
`quit`	disconnect from server
`:<command>`	execute `<command>` on client useful for `ls` on client

The capabilities (and therefore, protocol) of this client/server pair are much simpler than FIT. SFTP (Simple File Transfer Protocol) is closer to what is required. FTP, for example, uses 2 TCP connections: a telnet-like connection for control and a second one for data transfer. SFTP uses a single TCP connection and supports user access control, directory listing and changing, file renaming and tile erasure. There, only directory listing is required here for these commands. FIT also supports *lcd*, input, message*get*, etc. A hand-shaking protocol was required for this assignment.

Assignment 5 – three options were given:

Assignment 5A – FIT file transfer using UDP. Implement the file transfer programs of assignment 4 built on UDP sockets rather than TCP sockets. The programs will need to assemble data packets, provide CRC error checking and provide packet sequencing.

Packets may arrive in different order, duplicated or missing. They may need to be re-requested and/or rearranged. Each packet should be acknowledged (positively or negatively). A protocol will need to be adopted, which will describe the packet format error messages, acknowledgments, etc. In order to test the robustness of the protocol used, allow the user to specify transmissions fractions that will (randomly) be done in error.

Assignment 5B – two channel bidirectional file transfer. Implement of the file transfer programs of assignment 4 modified to have two sockets open: one for control information and other for data transfer. In addition, it need to allow the two programs to send files back and forth at the same time. A transfer can be aborted by using the control channel. It may be helpful to fork multiple processes (a finite state machine approach would be a good idea). FTP works in a similar manner. It has two socket connections, but does for different reasons since it is a true client-server protocol rather than the peer-to-peer protocol implemented here.

Assignment 5C – multi-user chat program. Assignment 2 involved a peer-to-peer chat program. This assignment requires a multiplexing chat server program creation which can handle multiple socket connections. There is no need to write a client program because telnet can be used here.

The server should accept input lines from any connected socket and output them to the rest of the connected sockets. When a user connects to the chat server, the server should prompt for a user name. This name should be broadcast to the rest of the users as ″user <name> has just joined the session″. Similarly, a message should be broadcast when the user leaves the conference. When a user's message is sent to the other connected users, the user name should be included for identification.

Conclusions. The goals of the laboratory component redesigning of the data communications course were to provide assignments that are: more meaningful and practical to the students; more enjoyable and, therefore, will be done better; more modern but are still oriented toward understanding, what is happening rather than simply using data communications; increasingly complex and build on previous assignments; based on more reliable hardware that the discarded PCs used previously.

Once the socket paradigm was chosen, goals were to create assignments which required students to write code that: makes use of C++ classes; provides simple client access to a well-defined server; provides simple server functionality; provides peer-to-peer communication; provides multiplexing server functionality; functions in a manner similar to a well-known network service; requires students to be concerned with unreliable communications; uses some form of fork() and programming for interprocess communications. The use of C++ socket classes and the assignments chosen meet all of the above goals if the optional forms of assignment 5 are chosen.

1. W. Stallings. Data and Computer Communications. / 4-th edition, MacMillan, New York, 1994.
2. W. D. Smith. The Design of An Inexpensive Undergraduate Data Communications Laboratory, SIGC'SE Bulletin. 1991. Vol 23, № 1. p. 273-276.
3. D. Finkel, S. Chandra, NetCp A Project Environment for an Undergraduate Computer Networks Course, SIGCSE Bulletin. 1994. Vol 26, № 1. p. 174-177.
4. A Socket-Based Interprocess Communications Tutorial. Chapter 10 of SunOS Network Programming Guide.
5. An Advanced Socket-Based InterProcess Communications Tutorial. Chapter 11 of SunOS Network Programming Guide.
6. R. Quinton. An Introduction to Socket Programming, Computing and Communication Skrvices. The University of Western Ontario, London, Ontario.
7. G. Swaminathan. C++ Socket Classes, Electrical Engineering Department, University of Virginia.