A Framework for Inteligent Remote Learning System

유 영동 (동국대학교 경영정보학과 교수)

1. INTRODUCTION

Communication technology is being exploited to create competitive advantage in the marketplace, increase productivity, and enable new forms of organizational design. Many organizations now have found that information networking is a management-level issue with the same importance as information systems applications and the data resource.

Training and educational entities are no exceptional in terms of the degree of innovation when deems as a business organization. How well the organization currently uses electronic communications technologies is a significant subject.

An training and educational institute in their nature should be obliged to the order to be an academic and educational capacity.

(1) Bulletin board and consultation services.
(2) Instructional, or computer-based learning and training.
(3) Informational database services and delivery.
(4) Data communications.
(5) Storage technology with hypermidea equipment.
(6) Specific application software with input/output devices.
(7) A knowledge-based environment for management and research units.
(8) End-user computing support that includes operating system and level routines and utilities.

Intelligent remote learning system(IRLS) is a system that incorporate communication technology. The opportunities rest on the advantages of modern communications technology but it will also be weighted against their costs. Fortunately it is still feasible when we considered the existing available system resources.

2. ARCHITECTURE OF IRLS

2.1 System Configuration of IRLS

IRLS works under an network environment: a network configuration of machines that exchange information among themselves. The information originating at the instructor's side can be transmitted along a communication line and delivered to the intended participant in an intelligent form.

Implementation of IRLS is devided into two phases as follows:

Phases 1: It is more than an experimental system that a real sophisticated system while it still has the function of communication capabilities centered within a single host machine. It include most of the instructional facilities and a prototype of database system which can be accessed by the legal parties.

Phase 2: In order to make IRLS to be more reasonable remote learning system, consideration has been taken for the feasibility study of a heterogeneous machine environment on which the IRLS can work. Another characteristic of IRLS is the build-up of an integral database subsystem that is used to keep track of all the student's grade history and other academic records. Besides, program routines are incorporated for the attainment of a powerful editing capability which is developed to encompass more sophisticated editing functions.

Nowadays window operation has been one of the main topic of software engineering, it is something short to be a complete piece of work without introducing the asset of window functioning. Prospects for IRLS include a good computer-based learning system, a database engine, a self-contained communication entity, and a tutorial system. All of these capabilities contribute to it's name of an "Intelligent Remote Learning System."

2.2 Machine / Network Configuration for IRLS

IRLS implementation relies heavily on network services and it is written in C on Ultrix/VAX 8550, which supports TCP/IP.

TCP/IP was initially developed by the United States Department of Defense to run on the ARPANET. ARPANET is a packet switching wide area network which was firstly demonstrated in 1972. Today the ARPANET is a part of a wide area network known as the DoD (Department of Defense) Internet, or the Internet, for short. Presently the term Internet is used to describe both the protocol family and the wide area network.

Every time when network is being referred to network protocol is sure to set in behind the scenes. Internet protocol family included transmission Control (TCP), User Datagram Protocol (UDP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), and Internet Control Message Protocol (ICMP). The entire family is popularly termed as TCP/IP, reflecting the names of the two main protocols.

TCP/IP provides services to many different types of host machines connected to heterogeneous networks. These networks may be wide area networks, such as X.25-based networks, but they can also be local area networks, such the one installed in a single building.

The TCP/IP protocol structure can be conceptualized as a series of layers as shown in the following figure 2.1:

Layer	Network Services
Application	Telnet, FTP, TFTP
Transport	TCP, UDP
Network	IP, ICMP
Data Link	ARP, RARP, device driver (such as Ethernet)
Physical	Cable or other devices (such as an Ethernet board)

Figure 2.1 Layers of TCP/IP

The following figure 2.2 shows an interaction of sender and receiver.

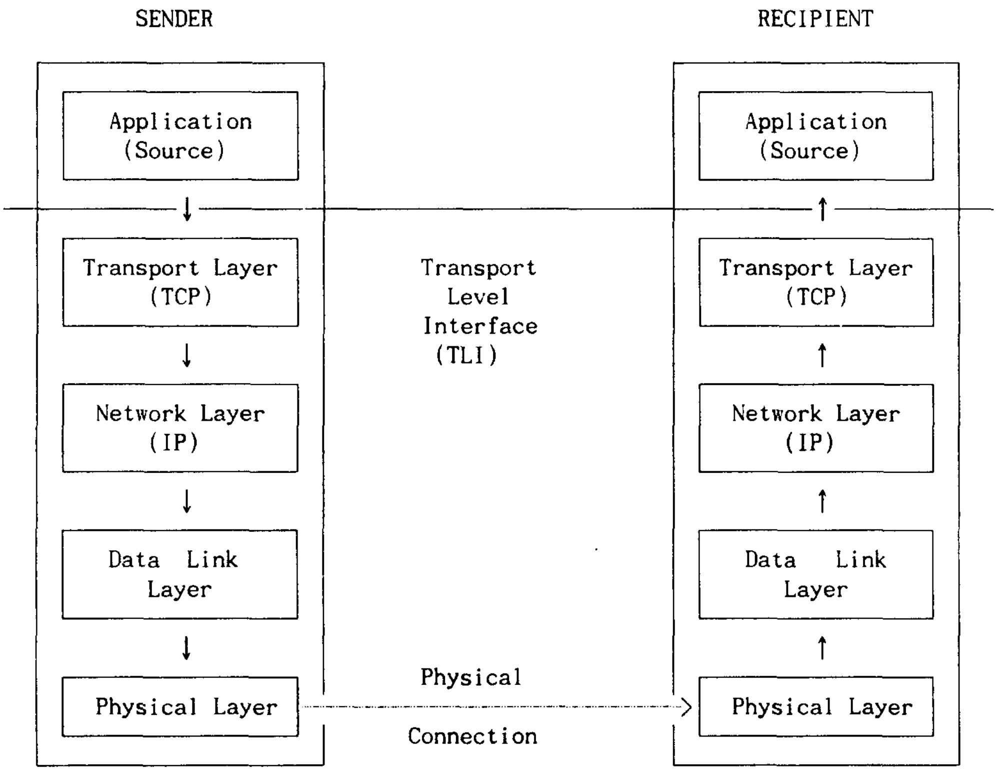

Figure 2.2 Sender/Receiver Interaction

2.3 STREAMS in UNIX communication service

IRLS implementation relies heavily on UNIX's existing network services. STREAMS was incorporated in UNIX system VRelease 3 to augment the character input/output (I/O) mechanism and to support development of communication services. A well defined and contrived structure of STREAMS makes it easy for a system programmer to have a good command in dealing with system communication tasks.

STREAMS defines standards interface for character I/O within the kernel, and between the kernel and the rest of the UNIX system. It consists of a set of system calls, kernel resources, and kernel routine. As STREAMS does not impose any specific network architecture, so its standard interface and mechanism enable module, portable development and easy integration of high performance network service and their components. Upward compatibility with the character

I/O user level functions like open, close, read, and write is one of its many benefits.

In one word STREAMS is a set of system calls, kernel resources, and kernel routines. STREAMS provideds a flexible, portable, and reusable set of tools for development of UNIX system communication services (Figure 2.3).

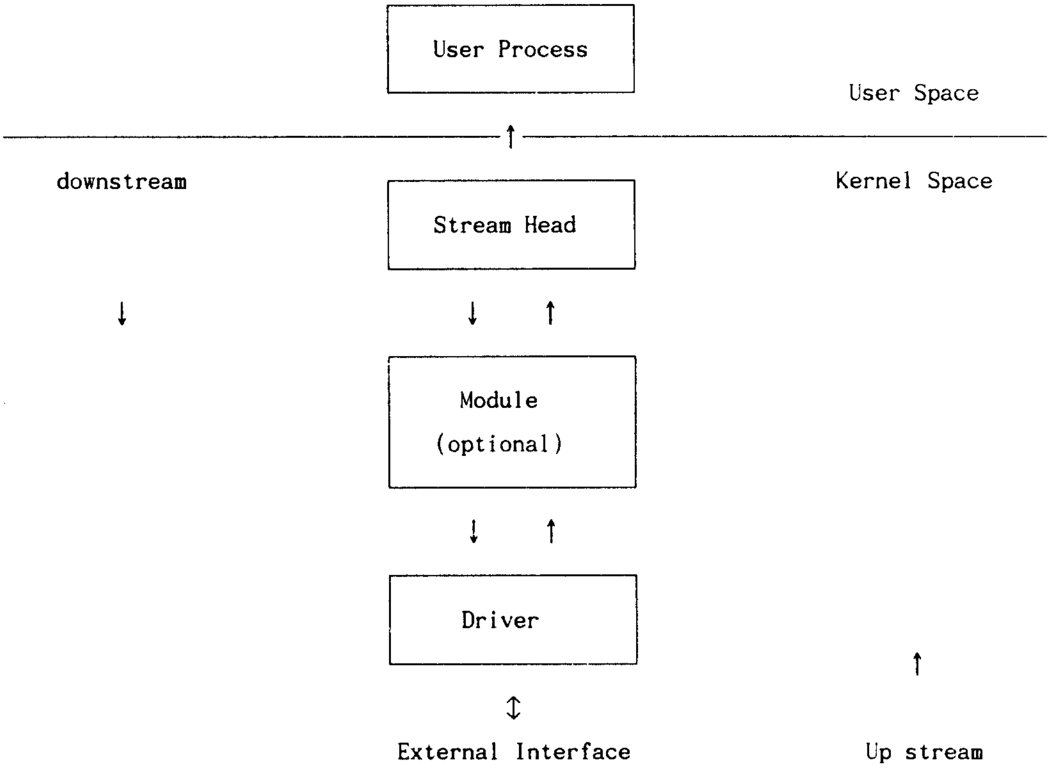

Figure 2.3 STREAMS

STREAMS are composed of four components mainly: queues, messages, modules and drivers. The components of STREAMS are briefly discussed in the following.

2.3.1 Queues

Queues structure constitute an essential part in IRLS through which interface between a STREAMS driver or module and the rest of the Stream can be achieved. A queue's service routine is invoked to process messages on the queue. It usally removes sucessive message from the queue, processes them, and calls the put routine of the next module in the stream to give the processed message to the next queue.

A queue's put routine is invoked be the preceding queue's put and or service routine to add a message to the current queue. If a module does not need to enqueue messages, its put routine can call the neighboring queue's put routine.

Each queue also has a pointer to an open and close routine. The open routine of a driver is called when the driver is first opened and on every sucessive open of the Stream. The close routine of the module is called when the module is popped (removed) off the Stream. The close routine of the driver called when the last reference to the Stream is given up and the Stream is dismantled.

2.3.2 Messages

Message is a set of data structure used to pass data, status, and control information between user processes, modules, and drivers. All STREAMS message are assigned message types to indicate their intended use by modules and drivers and to determine their handling by the Stream head. A driver or module can assign most types to a message it generates, and module can modify a message type during specified message types and sent\d them down-streams, and it will respond to other calls by coping the contents of certain message types that were sent upstream.

In some cases, messages may contain urgent information (such as BREAK or ALARM conditions) which must pass through the Stream very quickly. To deal with these cases, STREAMS provides multiple classes of message queuing priority. All messages have an associated priority field.

Message priority is defined by the message type; once a message is created, its priority cannot be changed. Certain message types common in equivalent high priority/ordinary pairs, so that a module or device driver can choose between the two priorities when sending information.

2.3.3 Modules

A module perform intermediate transformations on messages passing between a Stream head a driver. They may be zero or more modules in a Stream.

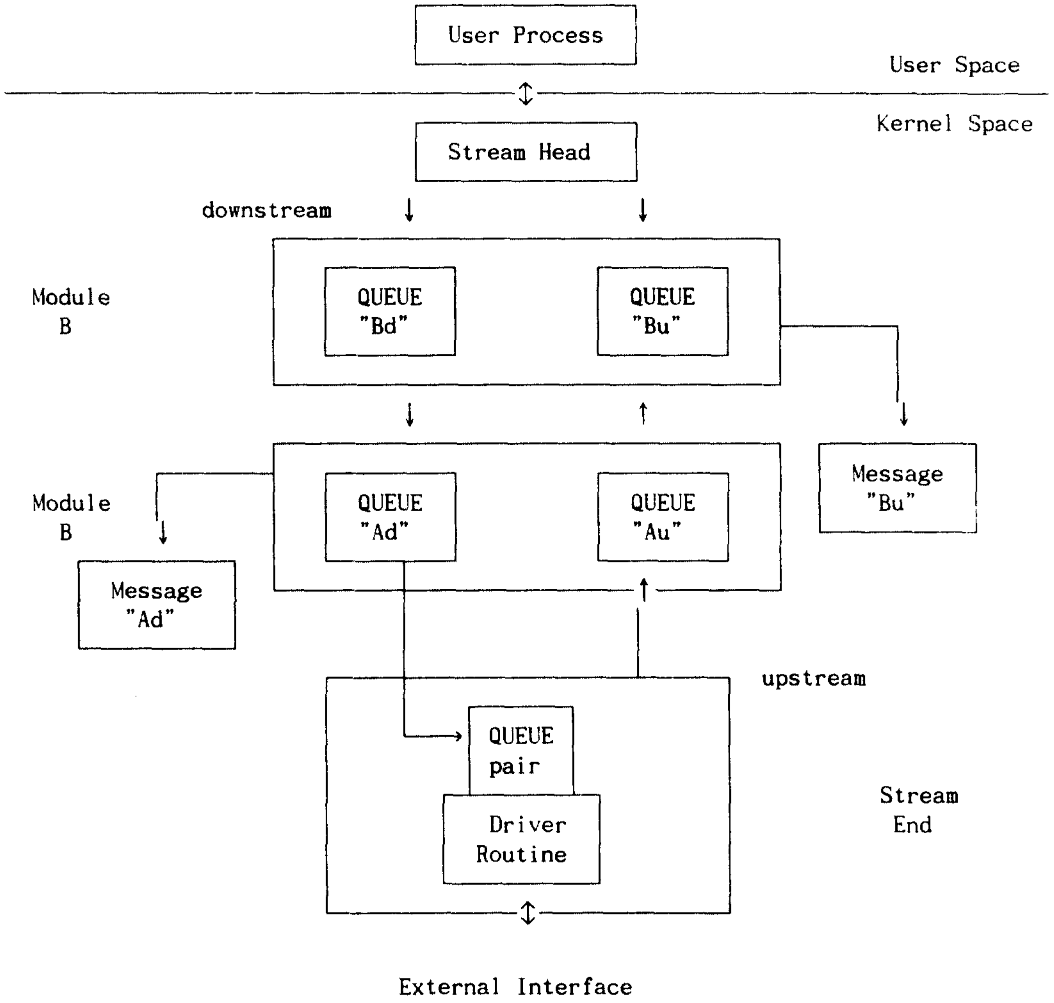

Figure 2.4 A Stream in More Detail

Each module is constructed from a pair of queue structures. One queue performs functions on messages passing upstream through the module. The other set performs another set of functions on downstream messages.

Each of the two queues in a module has distinct functions, that is, unrelated processing procedures .and data. The queues operate independently and "Au" (Figure 2.4) will not know if a message passes through "Ad" (Figure 2.4) unless "Ad" is programmed to inform it. Messages and data can be shared only if the developer specially programs the module functions to perform the sharing.

Each queue can directly access the adjacent queue in the direction of message flow (for example, "Au" to "Bu" or "Bd" to "Ad"). In addition, within a module, a queue can readily located its mate and access its messages and data.

Each queue in a module points tp messages, processing procedures, and data as follows:

1) Messages : These are dynamically attached to the queue on a linked list ("Message queue", see "Ad" and "Bu" in Figure 2.4) as they pass through the module.

2) Data : Developers may use a private field in the queue to reference private data structures (for example, state information and translation tables).

3) Processing procedures : A put procedure processes messages and must be incorporated in each queue. An optional service procedure can also be incorporated. According to their function, the procedure can send messages upstream and/or downstream, and they can also modify the private data in their module.

2.3.4 Drivers

Drivers that are to be included are services of an external I/O device, or a software driver (pseudo-device driver). IRLS uses driver to handle transfer between the kernel and the devices and does little or no processing of data other than conversation between data structure used by STREAMS mechanism and data structures that the device understands.

Note : Basically there three differences between modules and drivers in IRLS implementation.

1. A driver is able to handle interrupts from the device.
2. A driver can multiple Stream connected to it.

3. A driver is initialized/deinitialized via "open" and "close" system calls.

Similarities are :

Drivers and modules can pass signals, error codes, and return values to processes via message types provided for that prupose.

2.4 Multiplexing in IRLS

Multiplexing is suitable for many applications, IRLS is capable of multiplexing Streams in a variety of configurations. Typical multiplexing examples in IRLS are terminal window facilities.

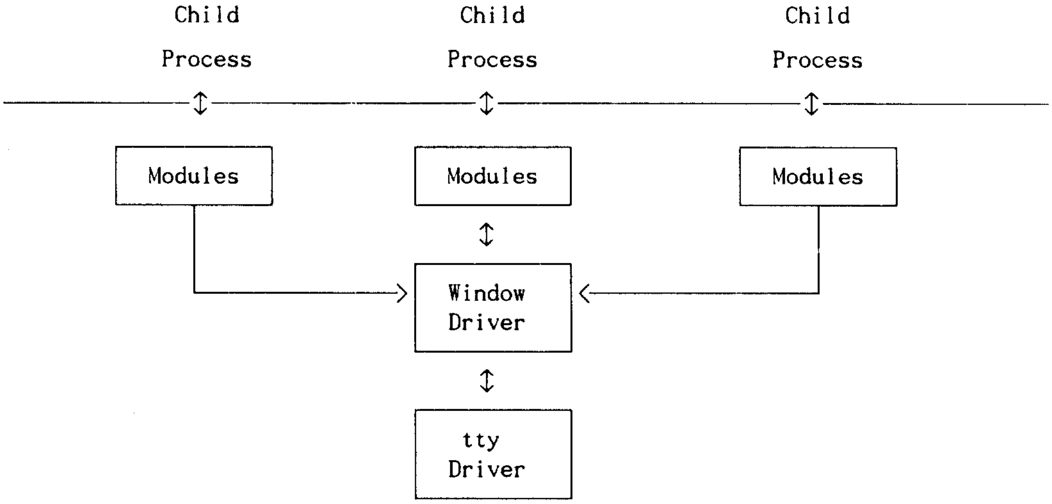

Figure 2.5 Window Multiplexing Streams

2.5 Interprccess Communication in IRLS

Without the using Interprocess Communication mechanism which is implemented as a part of the UNIX system utilities IRLS will never work properly as we expect. System V provides three seperate forms of IPC: they are semaphores, shared memory, and message queues. Each of these mechanism, while powerful in its own area, tends to rather restrictive in the types of uses to Which it can be put. The Berkely UNIX method, called "sockets," provides and interface that is a generalization of the pipe mechanism already familiar to most UNIX programmers. The pipe mechanism is actually implemented in Berkely UNIX as a pair of connected sockets.

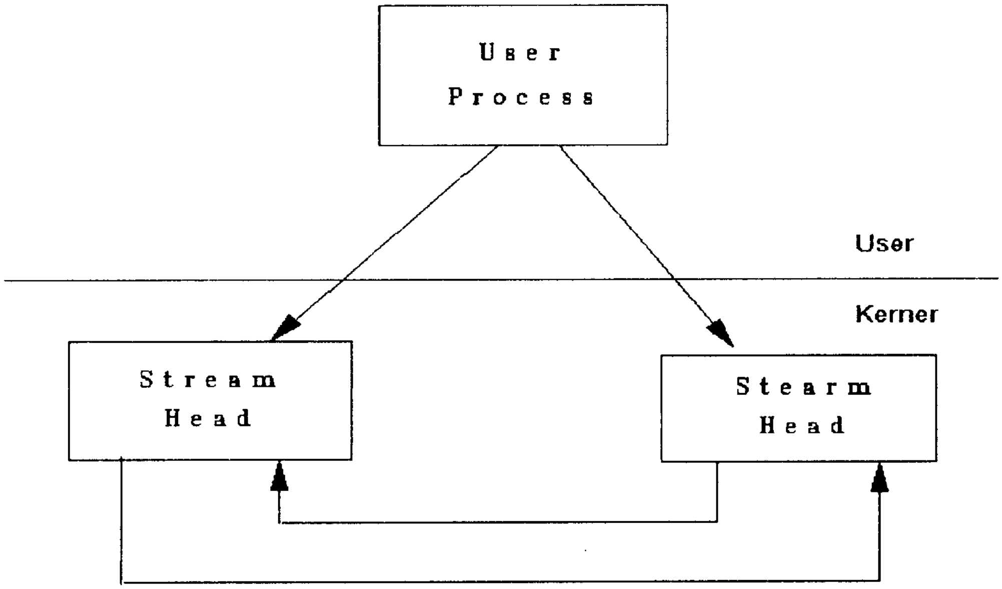

Figure 2.6 Pipe (STREAMS - based)

In the above "Pipe" mechanism there is no need for a driver. Ordinarily a driver is required for a STREAMS, while the module is not necessarily a required component. A STREAMS can be one without any module like the "pipe" above. (Figure 2.6)

3 Philosophy of IRLS in terms of IPC

IPC in IRLS is achived by a model as belows :

One process is called the server; it is responsible for satisfying requests put to it by the other process, the client. Normally when a server program is invoked (like instructor's Demon in IRLS), it asks the operating system for a socket. When it gets one, it assigns a well -known name to that socket, so that other programs can ask the operating system to talk to that name (the actual integer value will not be knownto other programs). After naming the socket, the serverlistens on the socket for connectionrequests form client process to come in. When a connection request arrives, the server may accept or reject the connection. If it accepts the connection, the operating system joins the client and server together at the socket, and the server may read and write data to andfrom the socket just as if it were a pipe to the client.

The client starts the process by asking the operating system for a socket, and then requesting that the socket be connected to some other socket that has a given name. The operating system tries to find a socket with the given name, and if it succeeds, it will send the process which is listening to that socket a connection, the operating system joins the two processes together at the socket, and the client can read and write data to and from that socket just like there exists a pipe to the server.

4 Summary

Intelligent remote learning system is a system that incorporate communication technology and others : a database engine, an intelligent tutorial system. Learners can study by themeselves through the intelligent tutorial system. The existence of a communication, database and artificial intelligence enhance the capability of IRLS.

According to Parsaye, an intelligent databases should have the following features :

1) Knowledge discovery
2) Data integrity and quality control
3) Hypermedia management
4) Data presentation and display
5) Decision support and scenario analysis
6) Data format management
7) Intelligent system design tools

I hope that this research of framework for IRLS paves for the future research. As mentioned in the above, the future work will include an intelligent database, self-learning mechanism using neural network.

REFERENCES

Ambrosch, W. D., Maher, and Sasscer, B. (1989). The Intelligent Network. NY: Springer-Verlag.

Burrows, B. C. (1986). Planning Information Technology and the Post-Industrial Society, Long Range Planning, vol. 19, no.2, pp. 79-89.

Kochan, S. G., and Wood, P. H. (1987). Topics in C programming. Hayden Books UNIX system library.

______ (1990), UNIX system V Release 4, Network User's and Administration's Guide. Unix Software Operation.

Parsaye, K., et al (1989). Intelligent Databases, Objected-oriented, Deductive Hypermedia Technologies.

Rao, K. R., and Srinivasan (1985). Teleconferencing. NY: Van Nostrand Reinhold Company.

Sullivan, C. H., and Smart, J. R. (1987). Planning for Information Networks, Information Technology Planning Corporation AT&T, vol. 28, no.2.

Touretzky, D. S. (1990). Advances in Neural information Processing System 2. Morgan Kaufmann.

Yoo, Y. D. (1991). An Expert Training System Loosely Coupled to External Database Systems, Proceedings, IEEE/ACM International Conference on Developing and Management Expert System Programs, Washington, D.C., Sept. pp. 24-28.