**CS 351**
**Week 10**

**Reading:**

1. The C Programming Language, Dennis M Ritchie, Kernighan, Brain.W
2. Network Programming for Windows,Second Edition,Anthony Jones & Jim Ohlund

**Objectives:**

1. An Introduction to Network Programming.
2. To learn about Socket Programming.
3. To learn basic concepts of Interprocess Communication.

**Concepts:**

1. Introduction to Windows Sockets
2. Windows Sockets Concepts
3. Interprocess Communication
4. Overview Of Lab

**Outline:**

1. Introduction to Windows Sockets
    a. What is Windows Sockets?
    b. What are its Benefits?
2. Windows Socket Concepts
    a. The OSI Network Model
    b. The Winsock Network Model
    c. OSI Layers and Winsock
3. Interprocess Communication
    a. Definition
    b. Send and Receive operators
    c. Performance Criteria
    d. Messages
    e. Possible Failures
4. Overview of Lab

**Reference:**


1. http://www.sockets.com/toc2.htm#Chapter1
2. http://www.cs.mcgill.ca/~cs577/lectures/577-communication.pdf
3. http://en.wikipedia.org/wiki/Interprocess_Communication
4. http://www.cafeaulait.org/books/jnp3/index.html
5. Stevens, Richard UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications. Prentice Hall, 1999. ISBN 0-13-081081-9

**CS 351: Week 10 – Lecture Notes**

1. **Introduction to Windows Sockets**

A. What is Windows Socket?

1. **WinSock** is the standard sockets programming API for the Windows operating system

2. WinSock has been the standard sockets library shipped with all versions of Windows starting with Windows 95.

3. WinSock was created to allow different Microsoft Windows TCP/IP software applications to communicate.

B. What are its Benefits?

- An Open Standard
- Provides Source Code Portability
- Supports Dynamic Linking
- Benefits Summary

2. **Windows Socket Concepts**

A. The OSI Network Model

- Services & Interfaces

B. The WinSock Network Model

- Information and Data
- Application Protocols

C.OSI layers and WinSock

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer

What is TCP/IP?

Transport Services

- Connection-less Services: UDP

- Connection-oriented Services: TCP
- Deciding on a Transport: UDP versus TCP

Network services

- IP Services
- ICMP Services

Support Protocols & Services

- Domain Name Service (DNS)
- Address Resolution Protocol (ARP)
- Other Support Protocols

## 3. Interprocess Communication

A. Definition

1. a set of techniques for the exchange of data among two or more threads in one or more processes

2. Processes may be running on one or more computers connected by a network. IPC techniques are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC)

3. The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

4. IPC may also be referred to as *inter-thread communication* and *inter-application communication.*

B. Send and Receive Operators

One communication "unit" consists of two primitives

- The **send** primitive is called by the sending process (caller, sender)
- A corresponding **receive** primitive must be called by the receiving process (callee, receiver)
- Basic assumption: Non-blocking send / blocking receive
  Determines the behavior upon calling **send/receive**
- Non-blocking send: sending process is allowed to proceed as soon as the underlying layer has received the message
- Blocking receive: The receive primitive blocks until a message arrives

C. Performance Criteria

Latency (response time):

- Delay between sending of a message by one process and its receipt by another process

- Time for the first bit to be transmitted through the network
- Delay in accessing the network
- Marshalling and send time at sender (CPU time!)
- Receive and unmarshalling time at receiver (CPU time)
- Bandwidth (throughput)
    - Total amount of information that can be transmitted in a given time

D. Messages

Mapping Data Structures and Data Items to Messages

- Messages are sequential -> data must be flattened
- Agreement of external data format

- XML, Corba Common Data Representatin, Java object serialization
- Marshalling Messages (serialization)
- Unmarshalling Messages (unserialization)
- Destination
- Internet address (=host) + port (location dependent)
- Port is a message destination within a computer,
- process can have several ports from which to receive messages.
- Any process who knows the number of a port can send a message to it
- Servers generally publicize their port number for use by clients.
- Service (location independent);
    - Service name is translated at runtime to server location

E. Synchronous Vs Asynchronous

Synchronous

- each message is transmitted within a known bounded time
- The time to execute each step of a process has known lower and upper bounds
- Each process has a local clock whose drift rate from real time has a known bound

Asynchronous

- Message may need an arbitrary time to be transmitted
- Each step of a process can take an arbitrary time
- Clocks drift rates are arbitrary

## 4. Overview of Lab

```
/* a server in the unix domain.  The pathname of
   the socket address is passed as an argument */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
void error(char *);
int main(int argc, char *argv[])
{
   int sockfd, newsockfd, servlen, clilen, n;
   struct sockaddr_un  cli_addr, serv_addr;
   char buf[80];

   if ((sockfd = socket(AF_UNIX,SOCK_STREAM,0)) < 0)
      error("creating socket");
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sun_family = AF_UNIX;
   strcpy(serv_addr.sun_path, argv[1]);
   servlen=strlen(serv_addr.sun_path) +
                   sizeof(serv_addr.sun_family);
   if(bind(sockfd,(struct sockaddr *)&serv_addr,servlen)<0)
      error("binding socket");

   listen(sockfd,5);
   clilen = sizeof(cli_addr);
   newsockfd = accept(
       sockfd,(struct sockaddr *)&cli_addr,&clilen);
   if (newsockfd < 0)
       error("accepting");
   n=read(newsockfd,buf,80);
   printf("A connection has been established\n");
   write(1,buf,n);
   write(newsockfd,"I got your message\n",19);
}

void error(char *msg)
{
    perror(msg);
    exit(0);
}
```

```
/* a client in the unix domain */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
void error(char *);

void main(int argc, char *argv[])
{
  int sockfd, servlen,n;
  struct sockaddr_un  serv_addr;
  char buffer[82];

  bzero((char *)&serv_addr,sizeof(serv_addr));
  serv_addr.sun_family = AF_UNIX;
  strcpy(serv_addr.sun_path, argv[1]);
  servlen = strlen(serv_addr.sun_path) +
            sizeof(serv_addr.sun_family);
  if ((sockfd = socket(AF_UNIX, SOCK_STREAM,0)) < 0)
     error("Creating socket");
  if (connect(sockfd, (struct sockaddr *)
                    &serv_addr, servlen) < 0)
     error("Connecting");
  printf("Please enter your message: ");
  bzero(buffer,82);
  fgets(buffer,80,stdin);
  write(sockfd,buffer,strlen(buffer));
  n=read(sockfd,buffer,80);
  printf("The return message was\n");
  write(1,buffer,n);
}

void error(char *msg)
{
   perror(msg);
   exit(0);
}
```

**Exercise**

1. Setup a two-way pipe between parent and child processes in a C program. i.e. both can send and receive signals.
2. Write a Program to create a Socket and Communicate between and a Client and a Server through Sockets.