**Computer Communications and Networks**



**Lab No 13**

**Socket Programming**

**TCP Socket API (Iterative TCP server using select() call)**

Lab Manual 13

|  |
| --- |
| Objectives  * Select()   + Handling multiple client without multi-threading |

# Reference Material

**Select (): Why do we need it?**

In the basic model, server handles only one client at a time, which is a big assumption if you want to develop any scalable server model.

The simple way to handle multiple clients would be to spawn new thread for every new client connected to the server. This method is strongly not recommended because of various disadvantages, namely:

* Threads are difficult to code, debug and sometimes they have unpredictable results.
* Overhead switching of context
* Not scalable for large number of clients
* Deadlocks can occur

A better way to handle multiple clients is by using **select()** linux command.It gives you the power to monitor several sockets at the same time. It'll tell you which ones are ready for reading, which are ready for writing, and which sockets have raised exceptions. This being said, in modern times **select()**, though very portable, is one of the slowest methods for monitoring sockets. **Select()** works like an interrupt handler, which gets activated as soon as any file descriptor sends any data.

**Syntax**

|  |
| --- |
| #include <sys/time.h>  #include <sys/types.h>  #include <unistd.h>  int select(int numfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout) |

**Data structure used for select:**fd\_set. It contains the list of file descriptors to monitor for some activity.There are four functions associated with fd\_set:

|  |
| --- |
| FD\_SET(int fd, fd\_set \*set);  FD\_CLR(int fd, fd\_set \*set);  FD\_ISSET(int fd, fd\_set \*set);  FD\_ZERO(fd\_set \*set); |

**Detail of Select () command**

You populate your sets of socket descriptors using the macros, like FD\_SET(), above. Once you have the set, you pass it into the function as one of the following parameters: readfds if you want to know when any of the sockets in the set is ready to recv() data, writefds if any of the sockets is ready to send() data to, and/or exceptfds if you need to know when an exception (error) occurs on any of the sockets. Any or all of these parameters can be NULL if you're not interested in those types of events. After select() returns, the values in the sets will be changed to show which are ready for reading or writing, and which have exceptions.

The first parameter, n is the highest-numbered socket descriptor (they're just ints, remember?) plus one

The helper macros do the following:

|  |
| --- |
| FD\_SET(int fd, fd\_set \*set); // Add fd to the set.  FD\_CLR(int fd, fd\_set \*set);//Remove fd from the set.  FD\_ISSET(int fd, fd\_set \*set);//Return true if fd is in the set.  FD\_ZERO(fd\_set \*set);// Clear all entries from the set. |

**Return Value**

Returns the number of descriptors in the set on success, 0 if the timeout was reached, or -1 on error (and errno will be set accordingly.) Also, the sets are modified to show which sockets are ready.

# Lab Task

1. **Write a client server code in which server can handle one client at a time and I/O requests using select() call. [50 Marks]**

**Server:**

* **Waits for new connections**
* **Whenever a new client arrives, it will send a welcoming message in the following format**
* **Hi,**
* **Please Enter any two numbers or type EXIT to exit**
* **On providing two numbers by client, (e.g 10,5) the server will reply following result to the client**
* **Numbers : 10,5**
* **Sum: 10 + 5 = 15**
* **Sub: 10 – 5 = 5**
* **Div: 10 / 5 = 2**
* **Mul : 10 x 5 = 50**
* **Now, server will again ask the client to enter further numbers or wants to exit**
* **If a client types EXIT, the server will simply close the connection with that client**