**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)**
**IS F462 – Network Programming**
**Lab#9**

---

**Topics: Event based Select Server, Signal driven I/O, epoll**

**Writing a concurrent TCP Server with Non-blocking IO and select:**
Consider the code given in *TCPServerwithSelectNBIO.c* file. It takes port number as the input over which it listens for new connections. It contains the modified code given *TCPServerwithSelect.c* file to work with non-blocking I/O. Non-blocking I/O plays important role in event driven server designs.

It operates as per the following: server waits until it gets EXPECTED_LEN chars. Once it gets at least EXPECTED chars, it goes into processing state. It converts them into uppercase letters. then it enters into writing state. It writes at most EXPECTED_LEN chars and closes the socket.

# Q?

1. Compile the above program and run it with a port number. Connect to this server using a telnet command. `telnet hostip portno`. Connect from multiple clients. Observe the output.
2. On telnet terminal type only < EXPECTED_LEN chars, and start connecting from another telnet. Will it connect immediately?
3. Every client goes through state transitions. Here it starts with state 1 and goes upto state 3. Modify the above program so that client goes to state 1 after completing state 3.
4. In the given program, processing is very simple. Let us modify the processing function to be very intensive, say run a loop as many times as the (sum of ASCII values)$^2$. This may make server confine to only one client making others wait. Modify the program such a way that i) let there be a separate thread/ fixed set of threads (as many as no of CPU cores) to process requests. ii) When client enters state 2, the client state is put into a message Q. iii) threads read from message queue and process the request. Message Qs are used so that client state array is not in contention for the rest of the program.
5. Suppose there are two states in processing phase let us say process1() and process2(), process1() for uppercase letters, process2() for transposition that is shifting by 1 character. Modify the above program to include this. use message queues for both process1() and process2().
6. Implement the above program using poll() call instead of select().

## Signal Driven I/O:

Consider the code for a event driven server using signal driven I/O given in *sigio.c.*

```
#define _GNU_SOURCE
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
```

```c
#include <sys/types.h>
#include <unistd.h>
#include <strings.h>
#include <signal.h>
#include <sys/select.h>
#include <sys/fcntl.h>

#define LISTENQ 15
#define MAXLINE 80

char buf[MAXLINE];

int listenfd; //global var so that signal handlers can access them.
int connfd;
static void
sigioListenHandler (int sig, siginfo_t * si, void *ucontext)
{
        printf ("no:%d, for fd:%d,  event band:%ld\n", si->si_signo,
          (int) si->si_fd, (long) si->si_band);
        fflush (stdout);
        if (si->si_code==POLL_IN)
        {
          int n = accept (listenfd, NULL, 0);
          if (n > 0)
        connfd = n;
          fcntl (connfd, F_SETOWN, getpid ());
          int flags = fcntl (connfd, F_GETFL);       /* Get current flags */
          fcntl (connfd, F_SETFL, flags | O_ASYNC | O_NONBLOCK);
          fcntl (connfd, F_SETSIG, SIGRTMIN + 2);
        }
        if (sig == SIGIO)
        printf ("Real time signalQ overflow");

}

static void
sigioConnHandler (int sig, siginfo_t * si, void *ucontext)
{
        printf ("no:%d, for fd:%d,  , event code:%d,  event band:%ld\n",
          si->si_signo, (int) si->si_fd, (int) si->si_code,
          (long) si->si_band);
        fflush (stdout);
        if (si->si_code == POLL_IN)
        {                               //input available
          int n = read (si->si_fd, buf, MAXLINE);
          if (n == 0)
        {
          close (si->si_fd);
          printf ("Socket %d closed\n", si->si_fd);
        }
          else if (n > 0)
        {
          buf[n] = '\0';
          printf ("Data from connfd %d: %s %d\n", connfd, buf, n);
          write (si->si_fd, "OK", 2);
        }
          else
        {
          printf ("Socket %d error", si->si_fd);
          perror ("socket");
        }
        }
        if (si->si_code == POLL_ERR)
        {
        int err;
        int errlen=sizeof(int);
        getsockopt(si->si_fd, SOL_SOCKET, SO_ERROR, &err, &errlen);
        if(err>0)
                printf("error on socket %d : %s", si->si_fd, strerror(err));
        }
        fflush (stdout);
}
```

```
main (int argc, char **argv)
{
        socklen_t clilen;
        struct sockaddr_in cliaddr, servaddr;
        struct sigaction sa, sa1;
        memset (&sa, '\0', sizeof (sa));
        memset (&sa, '\0', sizeof (sa1));
        sigemptyset (&sa.sa_mask);
        sa.sa_flags = SA_SIGINFO;
        sa.sa_sigaction = &sigioListenHandler;//for accepting new conn
        sigaction (SIGIO, &sa, NULL);
        sigaction (SIGRTMIN + 1, &sa, NULL);

        sigemptyset (&sa1.sa_mask);
        sa1.sa_flags = SA_SIGINFO;
        sa1.sa_sigaction = &sigioConnHandler; //for reading data
        sigaction (SIGRTMIN + 2, &sa1, NULL);

        listenfd = socket (AF_INET, SOCK_STREAM, 0);
        bzero (&servaddr, sizeof (servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
        servaddr.sin_port = htons (atoi (argv[1]));
        bind (listenfd, (struct sockaddr *) &servaddr, sizeof (servaddr));
        listen (listenfd, LISTENQ);

        fcntl (listenfd, F_SETOWN, getpid ());
        int flags = fcntl (listenfd, F_GETFL);       /* Get current flags */
        fcntl (listenfd, F_SETFL, flags | O_ASYNC | O_NONBLOCK);//set signal driven IO
        fcntl (listenfd, F_SETSIG, SIGRTMIN + 1);//replace SIGIO with realtime signal

        for (;;)
        {
          pause ();
        }

}
```

# Q?

1. Compile the above program and run it with a port number. Connect to this server using a telnet command. `telnet hostip portno`. Connect from multiple clients. Observe the output.
2. Modify the server for the following requirement: Whenever client sends data, sever has to send with cumulative reply that includes all the requests that are sent earlier. This requires a state-per-client need to be maintained.
   e.g.:
   ```
   Client: Hi
   server: Hi
   client:Hello
   Server:HiHello
   ```

3. Signal driven I/O is edge triggered. Let us understand this through an example. Maximum buf size is 10. So from telnet terminal type more than 10 chars and press ENTER. POLL_IN event occurs only once although there is more data in the buffer. Send more data from telnet window. What did you observe. Modify the server so that when POLL_IN event occurs it reads all the data present in the buffer.

## epoll API

Consider the code for a event driven server using epoll given in *epollserver.c*.

```
#include <sys/epoll.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <unistd.h>
#include <strings.h>
#include <sys/select.h>
#include <errno.h>
#include <signal.h>
#include <string.h>

extern int errno;
#define LISTENQ 5
#define MAX_BUF 10         /* Maximum bytes fetched by a single read() */
#define MAX_EVENTS 5             /* Maximum number of events to be returned from
                            a single epoll_wait() call */
void
errExit (char *s)
{
  perror (s);
  exit (1);
}

int
main (int argc, char *argv[])
{
  int epfd, ready, fd, s, j, num0penFds;
  struct epoll_event ev;
  struct epoll_event evlist[MAX_EVENTS];
  char buf[MAX_BUF];
  int listenfd, clilen;

  struct sockaddr_in cliaddr, servaddr;

  listenfd = socket (AF_INET, SOCK_STREAM, 0);

  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
  servaddr.sin_port = htons (atoi (argv[1]));

  if (bind (listenfd, (struct sockaddr *) &servaddr, sizeof (servaddr)) < 0)
    perror ("bind");

  listen (listenfd, LISTENQ);


  if (argc < 2 || strcmp (argv[1], "--help") == 0)
    printf ("Uage: %s <port>\n", argv[0]);

  epfd = epoll_create (20);
  if (epfd == -1)
    errExit ("epoll_create");


  ev.events = EPOLLIN;           /* Only interested in input events */
  ev.data.fd = listenfd;
  if (epoll_ctl (epfd, EPOLL_CTL_ADD, listenfd, &ev) == -1)
    errExit ("epoll_ctl");
  for (;;)
```

```
    {
      ready = epoll_wait (epfd, evlist, MAX_EVENTS, -1);
      if (ready == -1)
       {
         if (errno == EINTR)
           continue;       /* Restart if interrupted by signal */
         else
           errExit ("epoll_wait");
       }
      //printf("nready=%d\n", ready);

      for (j = 0; j < ready; j++)
       {
         if (evlist[j].events & EPOLLIN)
           {
             if (evlist[j].data.fd == listenfd)
              {
                clilen = sizeof (cliaddr);
                char ip[128];
                memset (ip, '\0', 128);
                int connfd =
                  accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);

                if (cliaddr.sin_family == AF_INET)
                  {
                    if (inet_ntop (AF_INET, &(cliaddr.sin_addr), ip, 128) ==
                       NULL)
                     perror ("Error in inet_ntop\n");
                  }

                if (cliaddr.sin_family == AF_INET6)
                  {
                    inet_ntop (AF_INET6, &(cliaddr.sin_addr), ip, 128);
                  }

                printf ("new client: %s, port %d\n", ip,
                       ntohs (cliaddr.sin_port));


                ev.events = EPOLLIN;     /* Only interested in input events */
                ev.data.fd = connfd;
                if (epoll_ctl (epfd, EPOLL_CTL_ADD, connfd, &ev) == -1)
                  errExit ("epoll_ctl");
              }
             else
              {
                int s = read (evlist[j].data.fd, buf, MAX_BUF);
                buf[s] = '\0';
                if (s == -1)
                  errExit ("read");
                if (s == 0)
                  {
                    close (evlist[j].data.fd);
                  }
                if (s > 0)
                  write (evlist[j].data.fd, buf, strlen (buf));

              }
           }
       }
    }
}
```

# Q?

1. Compile the above program and run it with a port number. Connect to this server using a telnet command. `telnet hostip portno`. Connect from multiple clients. Observe the output.
2. Is above epoll edge-triggered or level-triggered?
3. Modify the program making all sockets non-blocking. read() or write() should not wait for data.
4. Modify the program in 3 so that server sends cumulative reply i.e. server should keep per-client state.

**===End of Lab9===**