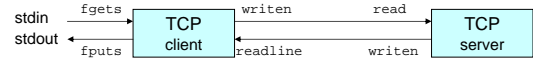


CSCE 515: Computer Network Programming ----- Select

Wenyuan Xu

Department of Computer Science and
Engineering
University of South Carolina

TCP Client/Server Example



2007

CSCE515 – Computer Network Programming

Echo client

```

int sockfd;
struct sockaddr_in server;

socket = Socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(SERV_PORT);
Inet_pton(AF_INET, argv[1], &server.sin_addr);

Connect(sockfd, (sockaddr *)&server, sizeof(servaddr));

str_cli(stdin, sockfd);

exit(0);
  
```

2007

CSCE515 – Computer Network Programming

Echo client (cont.)

```

str_cli(FILE *fp, int sockfd) {

    char sendline[MAXLINE], recvline[MAXLINE];
    while (Fgets(sendline, MAXLINE, fp) != NULL) {

        Writen(sockfd, sendline, strlen(sendline));

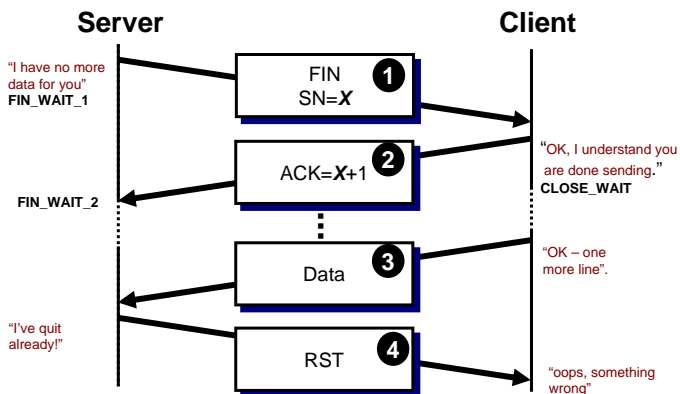
        if (Readline(sockfd, recvline, MAXLINE) == 0
            err_quit("str_cli: server terminated
            prematurely");

        Fputs(recvline, stdout);
    }
}
  
```

2007

CSCE515 – Computer Network Programming

TCP Termination



2007

CSCE515 – Computer Network Programming

Problem

- Server sends FIN
- Client TCP responds with ACK
- After that:
 - Server: FIN_WAIT2
 - Client: CLOSE_WAIT
- The client process is blocked in fgets when FIN arrives on the socket
- The client is working with two descriptor, while it should not block on one of them:
 - Socket
 - User input

2007

CSCE515 – Computer Network Programming

I/O Multiplexing

- We often need to be able to monitor multiple descriptors:
 - a generic TCP client (like telnet)
 - A server that handles both TCP and UDP
 - Client that can make multiple concurrent requests (browser?).

2007

CSCE515 – Computer Network Programming

Example - generic TCP client

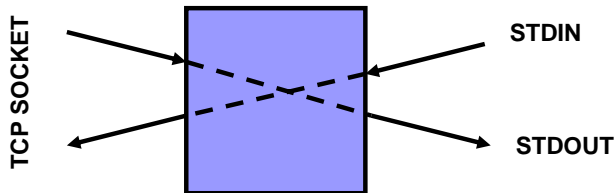
- Input from standard input should be sent to a TCP socket.
- Input from a TCP socket should be sent to standard output.

- How do we know when to check for input from each source?

2007

CSCE515 – Computer Network Programming

Generic TCP Client



2007

CSCE515 – Computer Network Programming

Options

- Use nonblocking I/O.
 - use `fcntl()` to set `O_NONBLOCK`
- Use alarm and signal handler to interrupt slow system calls.
- Use multiple processes/threads.
- Use functions that support checking of multiple input sources at the same time.

2007

CSCE515 – Computer Network Programming

Non blocking I/O

- Use `fcntl()` to set `O_NONBLOCK`:

```
int flags;
flags = fcntl(sock, F_GETFL, 0);
fcntl(sock, F_SETFL, flags | O_NONBLOCK);
```

- Now calls to `read()` (and other system calls) will return an error and set `errno` to `EWOULDBLOCK`.

2007

CSCE515 – Computer Network Programming

```
while (! done) {
    if ( (n=read(STDIN_FILENO,...)<0))
        | if (errno != EWOULDBLOCK)
          | /* ERROR */
        else write(tcpsock,...)

    if ( (n=read(tcpsock,...)<0))
        | if (errno != EWOULDBLOCK)
          | /* ERROR */
        else write(STDOUT_FILENO,...)
}
```

2007

CSCE515 – Computer Network Programming

The problem with nonblocking I/O

- Using blocking I/O allows the Operating System to put your process to sleep when nothing is happening (no input). Once input arrives, the OS will wake up your process and read() (or whatever) will return.
- With nonblocking I/O, the process will chew up all available processor time!!!

2007

CSCE515 – Computer Network Programming

Using alarms

```
signal(SIGALRM, sig_alm);  
alarm(MAX_TIME);  
read(STDIN_FILENO,...);  
...  
signal(SIGALRM, sig_alm);  
alarm(MAX_TIME);  
read(tcpsock,...);  
...
```

A function you write

2007

CSCE515 – Computer Network Programming

Alarming Problem

What will happen to the response time ?

What is the 'right' value for MAX_TIME?

2007

CSCE515 – Computer Network Programming

Select ()

- The select() system call allows us to use blocking I/O on a set of descriptors (file, socket, ...).
- For example, we can ask select to notify us when data is available for reading on either STDIN or a TCP socket.

2007

CSCE515 – Computer Network Programming

Select ()

- Return when
 - Any of the descriptors in the set {1,4,5} are ready for reading
 - Any of the descriptors in the set {2,7} are ready for writing
 - Any of the descriptors in the set {1,4} have an exception condition pending
- Specify what descriptors we are interested in and how long to wait

2007

CSCE515 – Computer Network Programming

select ()

```
int select( int maxfd,  
           fd_set *readset,  
           fd_set *writeset,  
           fd_set *exceptset,  
           const struct timeval *timeout);
```

maxfd: highest number assigned to a descriptor.
readset: set of descriptors we want to read from.
writeset: set of descriptors we want to write to.
exceptset: set of descriptors to watch for exceptions.
timeout: maximum time select should wait

2007

CSCE515 – Computer Network Programming

struct timeval

```
struct timeval {
    long tv_sec;    /* seconds */
    long tv_usec;  /* microseconds */
}
```

```
struct timeval max = {1,0};
struct timeval forever = NULL;
struct timeval polling = {0,0}
```

2007

CSCE515 – Computer Network Programming

fd_set

- Implementation is not important
- Operations you can use with an `fd_set`:

```
void FD_ZERO( fd_set *fdset);
void FD_SET( int fd, fd_set *fdset);
void FD_CLR( int fd, fd_set *fdset);
int FD_ISSET( int fd, fd_set *fdset);
```

2007

CSCE515 – Computer Network Programming

Using select()

- Create `fd_set`
- Clear the whole thing with `FD_ZERO`
- Add each descriptor you want to watch using `FD_SET`.
- Call `select`
- when `select` returns, use `FD_ISSET` to see if I/O is possible on each descriptor.

2007

CSCE515 – Computer Network Programming

Errors -- errno

- **EBADF**
 - An invalid file descriptor was given in one of the sets.
- **EINTR**
 - A non blocked signal was caught.
- **EINVAL**
 - *n* is negative or the value contained within *timeout* is invalid.
- **ENOMEM**
 - `select` was unable to allocate memory for internal tables.

2007

CSCE515 – Computer Network Programming

shutdown()

```
int shutdown( int sockfd, int howto);
```

`sockfd` is the TCP socket

howto:

- **SHUT_RD**: close the **read** half of the connection
- **SHUT_WR**: close the **write** half of the connection
- **SHUT_RDWR**: close both the **read and write** half of the connection.

`shutdown()` returns -1 on error (otherwise 0).

2007

CSCE515 – Computer Network Programming

shutdown() vs close()

- Reference counter:
 - `close()` decrements the descriptor's reference count and close the socket only if the count reaches 0.
 - `shutdown()` initiate TCP connection termination sequence regardless of the reference count.
- Directions:
 - `close()` terminate both directions of data transfer, reading and writing.
 - `shutdown()` can close one-half of the TCP connection, either reading or writing.

2007

CSCE515 – Computer Network Programming



Assignment & Next time

- Reading:

- UNP 5.12, 6.3-6.9**

- Next Lecture:

- thread