

## I/O Multiplexing

- We often need to be able to monitor multiple descriptors:
  - a generic TCP client (like telnet)
  - need to be able to handle unexpected situations, perhaps a server that shuts down without warning.
  - A server that handles both TCP and UDP

Netprog 2002 Multiplexing

1

---

---

---

---

---

---

---

---

## 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?

Netprog 2002 Multiplexing

2

---

---

---

---

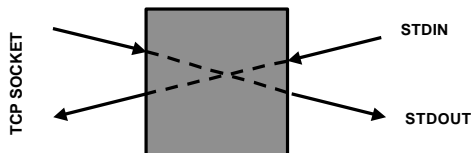
---

---

---

---

## Generic TCP Client



Netprog 2002 Multiplexing

3

---

---

---

---

---

---

---

---

## 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.

Netprog 2002 Multiplexing

4

---

---

---

---

---

---

---

---

## Non blocking I/O

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

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

(and other system calls)  
will return an error and set `errno`

Netprog 2002 Multiplexing

5

---

---

---

---

---

---

---

---

```
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,...)  
}
```

---

---

---

---

---

---

---

---

## The problem with nonblocking I/O

Using blocking I/O allows the Operating

system to schedule other processes in your program and read() (or whatever)

- With nonblocking I/O the process will chew up all available processor time!!!

---

---

---

---

---

---

---

---

## Using alarms

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

A function you write

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

---

---

---

---

---

---

---

---

## Alarming Problem

What will be the effect on response time ?

What is the 'right' value for MAX\_TIME?

---

---

---

---

---

---

---

---

## 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.

10

---

---

---

---

---

---

---

---

## select ( )

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

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

Netprog 2002 Multiplexing

11

---

---

---

---

---

---

---

---

## struct timeval

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

```
struct timeval max = {1,0};
```

Netprog 2002 Multiplexing

12

---

---

---

---

---

---

---

---

## fd\_set

- Implementation is not important
- Operations to use with 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);
```

Netprog 2002 Multiplexing

13

---

---

---

---

---

---

---

---

## 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.

Netprog 2002 Multiplexing

14

---

---

---

---

---

---

---

---