

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?).

Netprog 2002 Multiplexing

1

Example - generic TCP client

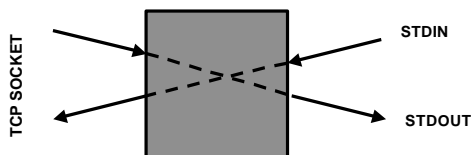
- Input from standard input should be sent to a TCP socket.
- Input from a TCP socket should be sent to standard output.

- 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;  
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`.

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,...)  
}
```

Netprog 2002 Multiplexing

6

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!!!

Netprog 2002 Multiplexing

7

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

Netprog 2002 Multiplexing

8

Alarming Problem

What will happen to the response time ?

What is the 'right' value for MAX_TIME?

Netprog 2002 Multiplexing

9

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.

Netprog 2002 Multiplexing

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 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);
```

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
