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?

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.
Non-blocking I/O

• use `fcntl()` to set `O_NONBLOCK`:

```c
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`.

The Problem with Non-blocking 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!!!

Using alarms

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

A function you write
Alarming Problem

What will happen to the 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.

```c
int select( int maxfd,
            fd_set *readset,
            fd_set *writeset,
            fd_set *excepset,
            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.
- `excepset`: set of descriptors to watch for exceptions.
- `timeout`: maximum time select should wait.

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

**struct timeval**

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

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

```c
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);
```

**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.