Chapter 13b
Clients with Multiple Input

Motivation: The chat client has two sources of input it must watch.
Input from keyboard and input from network.

We can't ignore the keyboard while waiting for network input
The user couldn't type until after a message arrived.

We can't ignore the network while waiting for keyboard input.
A message couldn't be displayed except after the user typed a message.

Extension: If we were talking to multiple chat servers there could be more than one network input.

Solution: Watch all possible sources of input and read the one with input waiting.

Chat Client Algorithm

1. Get the socket (and other descriptors).
2. Use select to wait for input on all open descriptors.
3. Arrival on socket, print to screen.
4. Arrival on keyboard, write to socket
5. Repeat 2–4

Useful fact, the keyboard is descriptor 0.

Details (no code)

1) Open the network socket

2) Clear the file descriptor set.

3) Set the socket descriptor in the descriptor set.

4) Set the keyboard descriptor in the descript set.

5) Set nfds to be the number of the socket descriptor+1.

6) `while(1)` loop

7) `rfds=afds` (memcpy)

8) select (wait for input)

9) if input from keyboard
`fgets`
`send` (`strlen`)

10) if input from socket
`recv` (`read`)
Turn into a string `buf[n]='\0'`, then `print`/`fputs`
or raw-write to the screen `write(1,...)`

Timed Wait

Wait for a certain amount of time (could be 0) for input, then process.

```
int n;
struct timeval waitTime;
waitTime.tv_sec = 10;
waitTime.tv_usec = 0;
n = select(nfds,&rfds,NULL,NULL,&waitTime);
if (n < 0) printf("it's an error");
if (n==0)
{
  printf("No input, do something useful");
}
else
{
  // input, do some input handling.
}
```

Wait at the select for 10 sec; if input arrives in that time, handle the input. If not, do something else.

Graceful Termination

If you close and the other process has just done a write, that write will fail.

Good behavior:
1) announce you are leaving.

2) consume any pending input.

3) exit.

`shutdown`: a partial close of a socket. You can shutdown output, input or both.

output: the other process will see end of file.
input: any writes by the other process will fail.
both: a close

Style:
1) shutdown output (announcing you are done)

2) read anything in the socket from the other processes (until end of file)

3) close/exit

The shutdown System Call

Example: the program will no longer be writting to the socket (the other end will see an EOF).

```
shutdown(s,SHUT_WR);
```

Other client issues

When handling an EOF from the keyboard, don't forget the close/clear.

Omitting this will cause the client to spin (busy wait) consuming lots of CPU. This will not be visible if your program is the only thing running on the machine; but by using excessive CPU, it will slow down any other program that is attempting to run.

Sample code:

```
close(0);
FD_CLR(0,&afds);
```