Digital Communication in the Modern World

I/O Models
Select command

http://www.cs.huji.ac.il/~com1
com1@cs.huji.ac.il

Blocking Vs. Non-Blocking I/O

Everything in Unix is a File

– Unix programs do any sort of I/O, they do it by reading or writing to a file descriptor.
– A file descriptor is simply an integer associated with an open file.
– The file can be:
  - Network connection (socket).
  - Pipe.
  - A real type on-the-disk file.
  - Just about anything else.

Blocking

- When you first create the socket descriptor with `socket()`, the kernel sets it to blocking. If you don't want a socket to be blocking, you have to make a call to `fcntl()`:

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/socket.h>
sockfd = socket(AF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
```

- If you try to read from a non-blocking socket and there's no data there, it's not allowed to block--it will return -1 and `errno` will be set to `EWOULDBLOCK`

Blocking Vs. Non Blocking

- If you put your program in a busy-wait looking for data on the socket, you'll suck up CPU time.
- You can execute multi-threaded
- A more elegant solution for checking to see if there's data waiting to be read comes in the synchronous mechanism of `select()`.

`select()` Synchronous I/O Multiplexing

- `select()` gives you the power to monitor several sockets at the same time. It'll tell you which ones are ready for reading, which are ready for writing, and which sockets have raised exceptions.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int numfds, fd_set *readfds,
  fd_set *writefds, fd_set *exceptfds,
  struct timeval *timeout);
```

`select()`

- The parameter `numfds` should be set to the values of the highest file descriptor plus one.
- In order to manipulate `fd_set` use the following macros:
  - `FD_ZERO(fd_set *set)`
    //clears the set
  - `FD_SET(int fd, fd_set *set)`
    //adds fd to the set
  - `FD_CLR(int fd, fd_set *set)`
    //removes fd from the set
  - `FD_ISSET(int fd, fd_set *set)`
    //tests to see if fd is in the set.

`select()`

- If the time specified in `struct timeval` is exceeded and `select()` still hasn't found any ready file descriptors, it'll return so you can continue processing.

```
struct timeval {
  int tv_sec; //seconds
  int tv_usec; //microseconds
};
```

- If you set the fields in your `struct timeval` to `0`, `select()` will timeout immediately, effectively polling all the file descriptors in your sets.
- If you set the parameter `timeout` to `NULL`, it will never timeout, and will wait until the first file descriptor is ready.

`select()`

- if you don't care about waiting for a certain set, you can just set it to `NULL` in the call to `select()`.
- if you have a socket that is `listen()`'ing, you can check to see if there is a new connection by putting that socket's file descriptor in the readfds set.
- On success, `select()` returns the number of descriptors contained in the descriptor sets, which may be zero if the timeout expires before anything interesting happens. On error, `-1` is returned, and *`errno`* is set appropriately;

Example

```
int main() {
  struct timeval tv;
  fd_set readfds;
  tv.tv_sec = 2;
  tv.tv_usec = 500000;
  FD_ZERO(&readfds);
  FD_SET(STDIN, &readfds);
  select(STDIN+1, &readfds,
     NULL, NULL, &tv);
  if (FD_ISSET(STDIN, &readfds))
     printf("A key was pressed!\n");
  else
     printf("Timed out.\n");
  return 0;
}
```