CptS/EE 455
Spring 2002
Notes from lecture on select()
March 29, 2002

Regardless of whether you choose to use threads or select() in project 3 you are responsible for the material in this lecture.

In project 3 the program for each router will have to be able to receive a message from any of the other routers or the control program or a timeout, any time that it has finished its immediate work. In the work we've done so far, we've seen the recv() procedure for waiting for and then receiving a message from a single source. The select() system call allows a program to wait for input from any number of sources (file descriptors) or a timeout. When select() returns, it provides enough information so that your program can perform a recv() operation on a file descriptor with the assurance that it will not block – it will return immediately.

> Note 1: Consider what would have to happen if select() didn't exist: your program would have to continuously test each socket descriptor to see if it had input waiting. This is called "busy waiting" and is not good practice as it continuously uses cpu resources that might be put to use in a different program. Even to implement busy waiting you would need something more than the form of recv() we've talked about thus far: you would need to set the socket descriptor to be "non-blocking" which causes recv() calls to return immediately when no data is available.
>
> Note 2: Some Unix systems have an alternative system call for waiting for multiple descriptors. It is called poll() and accomplishes the same result using a different argument and result arrangement. All modern Unix systems will have either select() or poll(), or, like our lab systems, both.

select() is available on Unix platforms and in the winsock library for Windows platforms. On Unix, select() operates over any file descriptors (sockets, real files, teletype input, …) while on Windows it operates over only socket descriptors. For this project, that is adequate.

Be sure to read the documentation in both the Pocket Guide and the man page for select(), or the winsock documentation if using Windows. The system documentation dominates anything I say here (though please let me know if you find mistakes so that I can correct them.)

Headers and declarations for using select()

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

struct timeval timeout;
```

```
fd_set in_fds;
int res;
```

An fd_set is a bit-vector with a bit corresponding to each possible file descriptor. A set of macros is provided for setting, testing and clearing the bits of an fd_set. The struct timeval you have seen before in project 1.

Macros for manipulating fd_set

```
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

Using select()

To use select you create up to fd_sets: descriptors you want to become ready for read, descriptors you want to become ready for write, and descriptors you want to test for exceptional conditions. In this project we will only be waiting for input descriptors, so we'll pass NULL for the other two sets. You also need to tell select how long to wait if none of the descriptors become ready – the timeout. Finally (but first in the calling list ☺), you tell select how many bits of the fd_sets it should concern itself with by passing it 1+(the highest numbered descriptor in any of the sets). Thus a call to select looks like this:

```
res = select(maxDescPlus1, &in_fds, NULL, NULL, &timeout);
```

About the timeout parameter: notice that it is a pointer to a struct timeval. You may pass NULL for this pointer in which case select will not time out. You may also pass a pointer to a struct timeval contain zero in both its seconds and microseconds fields in which case the select() call will always return immediately.

When select returns, the value returned is the number of ready file descriptors, or –1, in which case the global errno variable indicates the cause of the error. Notice that if `res==0` no file descriptors are ready, so a timeout must have occurred.

When select returns the fd_sets also have been updated so that the bits are set only for the descriptors that are ready. So your program has to loop testing, using FD_ISSET, each bit in turn to learn which descriptors it can call recv() on. Note that this means the fd_sets must be reinitialized for each call to select().

On some Unix systems, including those in our lab, the timeout structure will be updated to reflect the amount of time remaining in the specified timeout period. You cannot count on this behavior: the timeout structure should be initialized for each call to select() just as the fd_sets are.

Program structure using select

```
while (...) {
```

```
        /* initialize in_fds and maxDescPlus1 */
        /* initialize timeout */
        res = select( maxDescPlus1, &in_fds, NULL, NULL, &timeout );
        if ( res < 0 ) {
              /* handle error */
        } else if ( res == 0 ) {
              /* handle timeout */
        } else {
              int fd;
              for ( fd=0; fd<maxDescPlus1; fd++ ) {
                    if FD_ISSET(fd, &in_fds) {
                          /* actions for this file descriptor
                           * including recv(fd, ...)
                           */
                    }
              }
        }
}
```

Note on initializing the timeout: in this project your program needs to send its routing table to its neighbors every 30 seconds. Thus the timeout interval you provide to select must be such that if nothing else happens between the time select is called and the next 30 second boundary a wakeup will occur. What I suggest is that outside the loop you calculate the time of the first wakeuptime as (now+30 seconds) and then inside the loop, to initialized the timeout calculate (now – wakeuptime) and in the handle timeout action calculate a new wakeuptime (now+30 seconds).

Note on calculating with struct timevals: a struct timeval has two fields, seconds and microseconds. When you add or subtract struct timevals you must make sure that the microseconds value is in [0,1000000) by appropriately carrying or borrowing to the seconds field.