

**CSIS0230A Principles of Operating Systems(Class A)**  
**Notes for Tutorial 11**  
**The select system call**

When we read from one file descriptor for I/O and write to another, we can simply use read and write in a loop like this:

```
// INFILE is a file descriptor returned by 'open'ing some device or by pipe()
while ( (n = read(INFILE, buf, BUFSIZE)) > 0)
    if (write(OUTFILE, buf, BUFSIZE) != n) {
        fprintf(stderr, "Error!\n");
        exit(1);
    }
```

This works because we only need to check for I/O in one descriptor. But what if we have to read (multiplex) from two file descriptors? If we do the same, when one file stream has data, we probably ends up waiting for the other file stream and thus can't react. In the lecture, we described a method that can be used to prevent blocking when data is not present: set the file to non-blocking. But there is a problem in this approach: our program never block, so it uses up all CPU cycles when we wait for the availability of input. Is there any way that we can wait for the availability of *any* file descriptor among a set to become ready? The *select()* system call provides an answer.

### 1. The select function

The *select()* function has the following prototype:

```
int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

The call will block for up to a time specified by *timeout* (in microseconds), or until a descriptor that it “watches” become “ready”. The call watches three sets of descriptors: *readfds*, *writefds* and *exceptfds*. These sets are implemented as bit-sets. The integer *n* is defined to be at least one larger than the largest descriptor in these sets. This helps the kernel to avoid having to search every bit in the sets of descriptors.

If *timeout* is specified to be NULL, the *select()* call will never timeout. Otherwise, it should be pointing to a *timeval* structure, which contains two members *tv\_sec* and *tv\_usec*. The call will timeout after *tv\_sec* seconds plus *tv\_usec* microseconds (1 microsecond =  $10^{-6}$  second). If both are set to 0, the select call will never block.

On success, the *select()* system call returns the number of ready descriptors. In this case, the *readfds*, *writefds* and *exceptfds* are modified so that it contains only the descriptors that are ready.

If a timeout occurs before any descriptor is ready, the return value would be zero. If the system call is interrupted by a signal, or if a descriptor set contains invalid descriptors, it returns -1.

The *timeout* value might be modified by the kernel after the system call. However, this is not portable: many OS does not do this. So it is better to consider the value of *timeout* to be unreliable after any call to *select()*, and re-initialize it for each call to *select()*.

## 2. Is this fd ready?

The condition when a file descriptor is said to be ready depends on whether it is in *readfds*, *writelfds* or *exceptfds*:

*readfds* The descriptor is ready if it is ready for read, i.e., calling *read()* on it will not block and will not cause an error *EBLOCK*.

*writelfds* The descriptor is ready if it is ready for write, i.e., calling *write()* on it will not block and will not cause an error *EBLOCK*.

*exceptfds* The descriptor is ready if there is an exception condition on the descriptor. This happens for network file descriptors when “out-of-band” data arrives, e.g., somebody press Control-C when the normal file stream is filled.

Any of these sets can be specified as *NULL*, meaning that “I’m not interested”. For our purpose, we can always use *NULL* for *exceptfds*.

## 3. Working with sets of descriptors

The *select()* call makes use of a type called *fd\_set* to represent sets of descriptors. The implementation of these sets are system dependent. Typically this is a structure containing arrays used as bitsets. To use the sets in a portable way, we can use the following functions:

*FD\_ZERO(fd\_set \*set);*

Clear *set*, so that it contains no file descriptor. Usually this is the first call for any *fd\_set* you create.

*FD\_SET(int fd, fd\_set \*set);*

Add *fd* to *set*, so that *set* contains the descriptor. This is usually called to add the *fds* you want to watch to one of *readfds* or *writelfds*.

*FD\_ISSET(int fd, fd\_set \*set);*

Check whether *fd* is contained within *set*. This can be used to check whether a file descriptor is available for reading after the *select* call returns successfully.

*FD\_CLR(int fd, fd\_set \*set);*

Remove *fd* from *set*, so that *set* no longer contains the descriptor. This is only for completeness, and is seldom needed.

## 4. Reference

The manual page of *select(2)*. It also contains the description of a function called *pselect*, which can be made safe to use with signals.