Lab 07: Multiple File copying

The handling of I/O from multiple sources is an important problem that arises in many different forms. For example, a program may want to overlap terminal I/O with reading input from a disk or with printing. Another example occurs when a program expects input from two different sources, but it doesn't know which input will be available first. If the program tries to read from source A, and in fact, input was only available from source B, the program blocks. To solve this problem, we need to block until input from either source becomes available. Blocking until at least one member of a set of conditions becomes true is called OR synchronization. The condition for the case described is "input available" on a descriptor.

One method of monitoring multiple file descriptors is to use a separate process for each one.

While using separate processes to monitor two file descriptors can be useful, the two processes have separate address spaces and so it is difficult for them to interact.

The `select` call provides a method of monitoring file descriptors from a single process. It can monitor for three possible conditions—a read can be done without blocking, a write can be done without blocking, or a file descriptor has error conditions pending. Older versions of UNIX defined the `select` function in `sys/time.h`, but the POSIX standard now uses `sys/select.h`.

The `nfds` parameter of `select` gives the range of file descriptors to be monitored. The value of `nfds` must be at least one greater than the largest file descriptor to be checked. The `readfds` parameter specifies the set of descriptors to be monitored for reading. Similarly, `writefds` specifies the set of descriptors to be monitored for writing, and `errorfds` specifies the file descriptors to be monitored for error conditions. The descriptor sets are of type `fd_set`. Any of these parameters may be `NULL`, in which case `select` does not monitor the descriptor for the corresponding event. The last parameter is a timeout value that forces a return from `select` after a certain period of time has elapsed, even if no descriptors are ready. When `timeout` is `NULL`, `select` may block indefinitely.

On successful return, select clears all the descriptors in each of readfds, writefds and errorfds except those descriptors that are ready. If successful, the select function returns the number of file descriptors that are ready. If unsuccessful, select returns –1 and sets errno.

```
SYNOPSIS

#include <sys/select.h>

int select(int nfds,fd_set *restrict readfds,fd_set *restrict writefds,
        fd_set *restrict errorfds,struct timeval *restrict timeout);

   void FD_CLR(int fd, fd_set *fdset);
   int FD_ISSET(int fd, fd_set *fdset);
   void FD_SET(int fd, fd_set *fdset);
   void FD_ZERO(fd_set *fdset);                        POSIX
```

Historically, systems implemented the descriptor set as an integer bit mask, but that implementation does not work for more than 32 file descriptors on most systems. The descriptor sets are now usually represented by bit fields in arrays of integers.

The `FD_SET` macro sets the bit in `*fdset` corresponding to the `fd` file descriptor, and the `FD_CLR` macro clears the corresponding bit. The `FD_ZERO` macro clears all the bits in `*fdset`. Use these three macros to set up descriptor masks before calling `select`. Use the `FD_ISSET` macro after `select` returns, to test whether the bit corresponding to the file descriptor `fd` is set in the mask.

Task 1: Write a program that copies two files sequentially in a single process.

Task 2: Write a program that monitors two files using 'select'.

Task 3: Write a program that monitors N files using 'select'.

Task 4: Write a program that creates two child processes, both child read from the same source and writes to two separate destination, make sure you use select to monitor if the source is ready to be read from.

Task 5: Write a function that creates a delay of N seconds using select function. Pass N as an argument to the function.