Lab 7
Blocking System Calls and Timeouts

Prof. Kredo

Due: Start of lab Friday, March 13

Name:	
Name:	

Introduction

In this lab you will accomplish several goals:

- Learn about blocking system calls
- Develop simple timers using `select(2)`
- Write a program that closes if no user input arrives within a time period
- Modify the stream-talk programs to use `select(2)`

Work in pairs for this lab using the equipment at your desk. Distribute the work evenly to make sure both group members know the material, as you will be required to know the material for evaluation.

Some information that can help in this lab:

- Remember that standard output is buffered, so nothing is normally printed with `printf` until you print a newline (`'\n'`). If you wish to print something to the user without a newline, you can use `fflush(3)` with `stdout` as an argument.
- Predefined file descriptor numbers and file pointers exist for standard input and standard output. The `FILE*` variable for standard input is `stdin` and for standard output is `stdout`. The file descriptor (int) definition for standard input is `STDIN_FILENO` and for standard output is `STDOUT_FILENO`; they are defined in `unistd.h`.
- `select_tut(2)` has useful information, but you should skip the example code.

1 Prelab [10 Points]

System calls normally wait until they complete their operation before returning to the calling program. This behavior is called blocking. However, waiting is sometimes not desired. For example, if your program is attempting to receive a packet from a host that has crashed, you probably only want to wait a reasonable amount of time before giving up.

Several methods exist to prevent blocking system calls from waiting forever. A simple and efficient method that works for system calls that handle file descriptors is to use the `select(2)` system call before any potentially blocking system calls. `select` works by waiting until the file descriptor you specify becomes available, but it only waits for a specified time. Upon the return of `select`, you will know which file descriptors are available or if the maximum waiting time expired.

Read the man page for `select` and answer the following questions.

1. Write a segment of code that adds the standard input file descriptor to a file descriptor set and checks if the standard output file descriptor is in the file descriptor set.

2. Write a segment of code that initializes a `timeval` structure to store 2.05 seconds.

2 Simple Timers with `select` [20 Points]

To begin using `select`, write a short program that prints out a message every second. You can do this by setting the timeout value appropriately and by telling `select` to monitor no file descriptors.

Demo your program for credit.

3 User Input with `select` [40 Points]

Now expand your use of `select` to monitor a file descriptor. Write a program that:

- echoes user input (from standard input) back to the user (to standard output)
- closes when a user doesn't enter any input for 2 seconds

1. When a timeout occurs, is the standard input file descriptor in the file descriptor set? Is the standard input file descriptor in the set when a timeout does not occur?

2. When a timeout occurs, what is the time in `timeout`? What is the time in `timeout` when a timeout does not occur?

Demo your program for credit.

4 `simplex-talk` with `select` [30 Points]

Round out your exploration of `select` by modifying the `stream-talk` example programs on Learn. Expand the client program so that it closes when no user input arrives for 2 seconds. Expand the server program so that it closes when no clients connect for 5 seconds after program start.

Submit your `stream-talk` programs, with a Makefile, through Learn.