This lab is concerned with special I/O (for character devices) and setting an I/O descriptor non-blocking. It is also concerned with host name lookup (both ways: name to ip-address and ip-address to name).

1. Get copies of the sample programs `terminal.c` and `gethost.c` and compile and link the programs.

2. Try running the `terminal` program. It will put you successively into:
   i. CBREAK mode (which responds to SIGINT (CTRL-C), SIGQUIT (CTRL-), but, for example, ignores other signal generating keys such as SIGSTOP (CTRL-Z); single character is input as soon as the key is pushed),
   ii. RAW mode (no interpretation of any key; single character is input as soon as the key is pushed),
   iii. INPUT via a select() loop (which will also handle timeouts, signals (errno=EINTR), and exceptions) asking you for a character at a time,
   iv. COOKED mode (the default, also called line-edited mode) asking you for a string, supporting backspacing, etc. and converting the CR that you type into an end of string character.

3. Try running the program several times, first entering printable characters, then trying tabs, the bell character (CTRL-G), the ESCape key, and then generate SIGINT (CTRL-C) and SIGSTOP (CTRL-Z). Compare backspacing in case (iii.) above with backspacing in case (iv.)

4. Examine the code for the four procedures `raw_cbreak`, `raw_noecho()`, ... `echo_noraw()`. Note that there is a local header file `specio.h` which handles some differences in special I/O with regard to terminal structure and keywords, but BSD has some crucial differences which have to be handled by C-preprocessor directives in the code. The old style of I/O control uses `ioctl()` calls which are very system dependent. The newer POSIX method of I/O control uses the simpler `tcgetattr()` and `tcsetattr()` instead. Go to the function main() and examine the setup and subsequent options after the select() loop. We can make decisions based upon timeouts, exceptions, and activity on a number of descriptors. This gives maximum flexibility and control for I/O. Also note the signal trap for SIGINT.

5. The program `gethost` does basic DNS lookup and reverse lookup. If you enter, for example
   `gethost rtfm.mit.edu < cr >`
   then it calls `gethostbyname()` and returns the IP address in a standard hostent structure. This assumes that the responsible name server has a `hosts.db` record of the form

   `rtfm.mit.edu A 18.181.0.29`

   If you enter, for example
   `gethost 18.181.0.29 < cr >`
   then it calls `gethostbyaddr()` and returns the host-domainname in a standard hostent structure. This assumes that the responsible name server has a `hosts.rev` record of the form

   `29.0.181.18.in-addr.arpa PTR rtfm.mit.edu`

   In both cases it shows any aliases that it found for the host. These calls are the standard ways to use the Domain Name Service (DNS) (for translation of addresses both ways) in a program.

**Assignment** Write a short program which will exactly simulate a pegasus login. That is, the program should give the Digital UNIX (pegasus.cs.csubak.edu) (`ttypx`) login:
   prompt, echoing characters for the username being typed, then prompt for the password with password:
   but not echo what is being typed. The program should end by printing out the (username, password) pair which was typed. The `main()` code should be your own but you may incorporate any of the includes (e.g. `specio.h`, `diagost.h`, etc.), utility, or diagnostic routines (e.g. `raw_cbreak()`, `echo_noraw`, etc) from the sample programs; do not use calls like `system("setty -echo")` to do terminal control.

When you are satisfied your program is working correctly, e-mail me and tell me the path to the source file (that’s all I need).