This lab concerns processes and communication (over a socket) between two processes (possibly on different hosts).

1. Get copies of the sample program `process.c` and the makefile (Makefile) in one of your subdirectories. Compile and link the program by typing:
   ```
   make process < cr >
   ```

2. Try running the program (you just need to type the name of the executable, `process`). It will prompt you for the info-flags you wish to set and you can enter the octal code with leading zero (or you can put the info-flags as the first argument on the command line). You may wish to copy the output to a file via:
   ```
   process [info-flags] 2>process.out < cr >
   ```
   Note that the output of the program is written to stderr (descriptor 2) not stdout (descriptor 1). This is so that you can run it interactively (with stdin and stdout) while it writes to stderr. Note that some information (such as usernames) might not be obtained with “-na-” appearing. This is due to the fact that the password file is shadowed.

3. Get copies of the two sample programs `vcsend.c` and `vcrec.c` and the makefile (Makefile) in one of your subdirectories. These programs require some files which handle platform differences: `header.h`, `startup.h`, `startup.c`, and `cleanup.c`. The programs do some diagnostic dumps so the following files are also needed: `diagnost.h` and `diagnost.c`. Compile and link both programs by typing:
   ```
   make tcp < cr >
   ```
   Make printouts of the source for both of these programs.

4. The program `vcrec` calls `socket()` to get a socket descriptor, calls `bind()` to bind the socket to the Internet name space (AF_INET), calls `getsockname()` to find out which port was assigned, and finally calls `listen()` and blocks waiting for a connection request. The program should be started with:
   ```
   vcrec [optional_buf_size] [%i] < cr >
   ```
   You should give it time to print out its results including, most importantly, the tcp port number.

5. The program `vcsend` calls `socket()` to get a socket descriptor, and calls `connect()` to try to connect to the host and tcp port that `vcrec` is listening at. The program should be started with:
   ```
   vcsend pegasus.cs.csubak.edu [portnumber] < cr >
   ```
   assuming, of course, that `vcrec` is running on pegasus (although these programs will work from different hosts). When `vcsend` calls `connect()`, `vcrec` unblocks from `listen()` and calls `accept()` which allocates a new descriptor for the dialog between `vcsend` and `vcrec` (a more realistic program would do some authentication before deciding to accept). Thereafter, the programs use `send()`’s and `recv()`’s to communicate, the session ending when the user enters a line starting with a period. Under Unix `write()` and `read()` would work just as well, but some systems require `send()` and `recv()`. Both programs close the sockets opened before exit. While the session is active you can use the network utility `1sof -i tcp:<port number>` to check which processes have these ports open.

Assignment. Answer the following questions and e-mail your answers to me as an ascii text file. Please do not e-mail attachments to me. I will just send them back. It is completely unnecessary and wasteful of both network bandwidth and spooling space to send attachments which are just text files. It also slows me down because I use scripts to process my e-mail.

   a. What is the effect of using a very small buffer size in `vcrec` (e.g., 4 characters)? What does this tell you about keeping track of message boundaries when you communicate with a socket?

   b. Note that, except for the initial handshake, `vcsend` only writes and `vcrec` only reads. Is it possible to make simple modifications to the code so that one has full duplex communication (i.e., each process sending and receiving) between the two processes? If your answer is “yes” explain the changes specifically. If your answer is “no” explain why not.