1. Get a copy of the sample program `daemon.c`. Compile and link the program by typing:

   ```
   make daemon < cr >
   ```

   Make printouts of the source for this program. If you have a Windows NT/2000/XP system available you should try compiling it with the Microsoft `nmake` (for "new make"). To do this, you will have had to have enabled the command line compiler under Microsoft Visual C++ version 5 or higher. This makes a batch file named `vcvars32.bat` which you may have to run before compilation to set your environment for a (32-bit) MSDOS command prompt session. Make a subdirectory for the files, copy over the (sub)subdirectories ./arpa, ./inet, and ./netms, and do

   ```
   copy makefile.msv Makefile < cr >
   nmake daemon.exe < cr >
   ```

   which will make an executable `daemon.exe` which you can then run.

2. The program `daemon` is the skeleton of a (dumb) daemon which will listen at a non-privileged port and allow up to `MAX_CHILDREN` (currently set at 4) concurrent dialogs (which it will log to a file if you use the "-log" option). To test it, first open two xterms. In the first, start the program with

   ```
   daemon -log < cr >
   ```

   Write down the tcp portnumber (say it is 1295) and then go to the other xterm and invoke `telnet` as follows

   ```
   telnet localhost 1295 < cr >
   ```

   On the telnet xterm you should see a "Ready" prompt and on the daemon xterm you should see a notation that the daemon has fork'ed and created an offspring which will now dialog with your telnet. In the case of Windows NT a fork is not possible and the daemon calls `beginthread()` to create a (non-main) thread which will dialog with the telnet.

   Type a few lines (which the daemon will log to a file whose name is of the form `CHILD_PID.log`). To end the connection, break to the `telnet>` prompt with CTRL] and type "close". On the daemon screen you should see an indication that the offspring terminated when the connection was broken. On the death of the child process a SIGCLD was sent to the parent by the operating system indicating the death of a child, the parent broke to the (previously set signal handler) `offspring_terminate()`, which called `wait()`, updated the parent’s tables, printed the message, and reset the handler. To kill the program `daemon` hit CTRL-C. This is handled gracefully by trapping the CTRL-C with the `parent_terminate()` handler. Examine the log file to see that it correlates with what you typed.

   Note: there will be some differences depending upon which `telnet` you use. Most newer Unix and Linux versions start in (cooked) line mode, even without any telnet arbitration; many older Unix ports and PC versions will grumble that no arbitration was offered and will be in (raw, no-echo) character mode, i.e. you will not see the characters you are typing but they will be in the log file. Some `telnet` implementations (such as the windows command line telnet) are braindead and will only give you character mode. On reasonable systems you can switch by breaking to the `telnet>` prompt (with CTRL-]) and
by typing either “mode character” or “mode line”. Make sure you test both modes. Try connecting from a variety of hosts, if possible.

3. Run the program again but try multiple connections from 2–3 other xterms. Note that the log files are separate since the offspring have different PID’s. Try to make more than MAX_CHILDREN concurrent connections. What happens?

Assignment Modify the procedure dialog_with_telnet() in daemon.c program so that the following additional modifications are operational:

   a. Instead of just printing “Ready..” the offspring of the daemon prints out the Digital Unix prompt:

   Digital UNIX (pegasus.cs.csubak.edu)

   Note: you should use both a carriage return and line feed at the end of line on output since the telnet versions you use may be in raw mode.

   b. Assume that the user connecting will be typing lines of printable ascii text of not more than 128 characters (including the end of line). Assemble the characters being received into a string variable line[]. Warning: you cannot assume that you are receiving a complete line on every read of the socket and your program must work in both character and line mode. You must allow for the possibility that a line will be received a few characters at a time. As you assemble the line, check to see if the substring “exit” appears anywhere in what you have so far. As soon as “exit” is found, write the following message “"exit" token found” to the log file, clean up the connection and exit (refer to the code after the label finish: in dialog_with_telnet()).

If you have any questions about the program’s behavior there is a compiled version lab5_sol in /usr/local/bin/ on pegasus which you can run to test.