

NetCore Take Home Interview Problem:

Please create either a C or C++ program (note: if you choose C++, Greg likes “dumb” code because it’s maintainable, and thinks OO paradigm is almost certainly the wrong choice for this task) for a *nix system that, upon compilation, produces a binary that may be executed with the following arguments:

```
./your_programs_binary threads initial_port
```

Both arguments are integers. For the purposes of this exercise, threads will be an arbitrary number from 1 to 10, and initial port will be some number between 2000 and 30000.

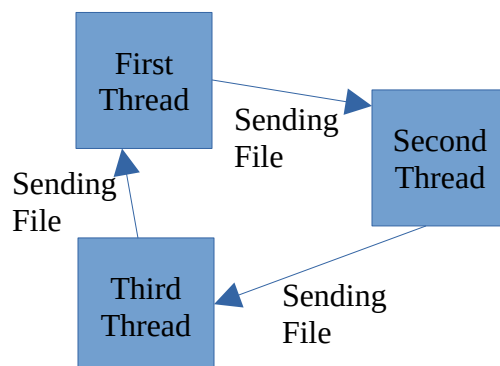
Upon executing, your program should create as many threads as is specified by the threads argument, and each thread should open a TCP socket on a different port, starting with the port number specified in the initial_port argument, and each successive thread incrementing up by one. For example, if your compiled program were run with:

```
./your_programs_binary 3 3000
```

The program should run with (at least) three threads, and there should be three threads with open TCP sockets, one listening on port 3000, another on port 3001, and a final thread on port 3002.

Upon all threads being created and now listening on ports, each thread should open a TCP connection with the next thread “up” from it until it gets to the last thread, which should open up a TCP connection with the first (ie in the above example, the thread on port 3000 should open a TCP connection with the thread on 3001, the thread on port 3001 should open a TCP connection with the thread on 3002, and the thread on 3002 should open a TCP connection with the thread on 3000).

Upon all the TCP connections being open, the first thread should read this requirements document (NetCoreInterviewProblem.pdf) from disk (you may place this file wherever on your machine, just please document the required path relative to the binary’s execution for our testing) into memory, print to stdout that it is sending the file, and then send the file over the TCP connection it has open to the next thread. It should be noted that this file is deliberately very much larger than the Maximum Transmission Unit for TCP. Upon the next thread receiving the file, it should print to stdout that it has received the file. It should then wait exactly 1 second, and then do exactly as the first thread did in that it should print that it is sending the file to stdout, and send it to the next thread. The last thread should then do the same, except sending the file to the first thread. The threads should repeatedly pass this file around until the program is manually terminated.



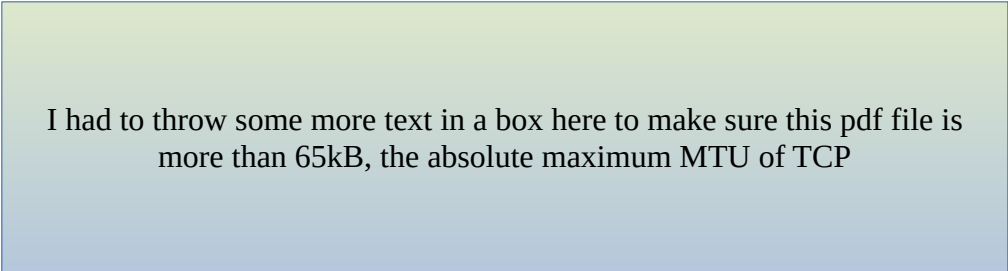
It should be noted that this program may be required to run for an arbitrary amount of time, so be sure to free memory when it is no longer needed (ie after reading the message from the socket and passing it on, it is probably very advisable to free the memory used to temporarily hold the message).

Please limit library usage to “standard” and avoid third party libraries, ie any part of the standard posix library such as pthreads.h socket.h, netinet/in.h, is naturally totally fair game, but third party libraries (such as all the boost software libraries ie netlib for C++) are off limits. It is likely the program you produce will be build-able with just a simple

```
gcc -o your_programs_binary your_program.c
```

but if this is not the case, please limit your build to gnu make, and provide a makefile and documentation.

Thank you for participating in our interview process, and we look forward to seeing what you come up with to solve this problem.



I had to throw some more text in a box here to make sure this pdf file is more than 65kB, the absolute maximum MTU of TCP



And here too...



And finally here.