Problem 1
Consider the client-server configuration depicted below:

![Client-server diagram]

A client sends requests spaced approximately 1 second apart to the server by depositing each request in a well known FIFO named `/requestFifo`. Upon receiving each request, the server forks a child which computes the Nth prime number, and waits for the next client request. Here are some more details:

- Each request is an 8-byte message containing two longs: the client process’ PID, and a value of N.
- The child process returns a 12-byte reply message by writing to a FIFO whose name is the PID passed in the request message to the server. The message contains three longs: the prime process PID, the value of N, and the Nth prime number.
- The client signals the server that it is done by sending an N-value of 0. The server should clean up and exit when the client is done.
- After the client is done sending all of its requests, it should read its reply FIFO and display the value of N and the prime number on stdout.
- Note that the server should create/delete the request FIFO, and the client should create/delete its reply FIFO.

Guidelines
- The client and server are separate C or C++ programs with the following call lines:

  ```
  server [-v] > server.log
  client [-v] > client.out
  ```

The optional -v flag indicates verbose mode which means that the process should display on stdout detailed messages about its computational state that can be used for debugging purposes. For example, a verbose server should display each message it receives and the PID of each child that it creates. A verbose child process should display each message it sends and a message with its PID when it starts execution. A verbose client should display each request that it sends. Note: The Web page contains code for the `parseargs()` function that you can use for processing command line arguments.
• The client should send the 4 N-values shown on the CS422 Web page to the server; i.e., you
  can just initialize a 4-element array to these values.

Submit the documentation described below and demonstrate your program to one of the graders.
The demonstration should be scheduled in advance with a grader/consultant and be done on or

Submit the following documentation:

• A clear statement of existing bugs and deficiencies. If none, state that there are no known bugs.

• Listings of the server code, client code, and the demo shell script.

• Include pseudo-code for the client and server programs. You may provide additional expla-
nation or data structure descriptions to clarify your pseudo-code but try to limit this to no
more than two pages.

• The output and an explanation of one or more of your test runs that indicates that your
  program is functioning properly. Your explanation should describe the test and explain why
  the output shows a properly functioning program. Try to limit this part to no more than two
  pages.

Problem 2
Suppose that I want to extend the program described above to support the following features:

• Limit the number of concurrent prime computations to M. M is an optional command line
  argument in the form -m M and is a positive integer with a default value of M=2.

• A reasonable request is one that can be finished using approximately 1 second of CPU time or
  less. All new requests are assumed to be reasonable. An unreasonable request is one that has
  already consumed more CPU time than is reasonable (you have to monitor its CPU usage to
determine if it is reasonable).

• Allow any reasonable request to always get executed unless there are already M reasonable
  requests being handled. In this case, the most unreasonable request should be put to sleep to
allow a reasonable request to execute; it can be resumed after more reasonable requests are
finished.

  a) List the major issues that will be addressed by your design.

  b) Summarize (no more than two pages) your basic approaches to handling the issues listed in
Part a. (C/C++ Code is not necessary and discouraged.)

SOME HINTS:

• Assume that you can put a process to sleep using kill(pid,SIGSTOP) and wake it up using
  kill(pid,SIGCONT).

• Assume that a process (the server) can wait on the arrival of a message from one of two
FIFOs with file descriptors fd1 and fd2 by calling fd=waitfor(fd1,fd2). fd is either fd1 or
fd2 and indicates which file has a message. If there are no messages, waitfor() blocks.

• A parent process can not get a child process' resource usage until the child has terminated,
  but a child process can get its own resource usage using getrusage(2) while it is running.