You are to write and test C or C++ programs that implement the following client-server configuration with $M$ Clients and 1 Server:

The server repeats the following sequence until determining that all clients have finished sending requests: 1) get a number $N$ from a well-known FIFO; 2) fork a child to compute the $N$th prime; and 3) wait for the next request.

Request messages contain three fields: 1) an op-code of 1, 2, or 3; 2) the PID of the client; and 3) the $N$-value; i.e.,

```c
struct {
    short  op;    // op code
    pid_t  pid;   // PID of client
    long   N;
};
```

An op-code of 1 indicates that the server should fork a child and compute the $N$th prime number (i.e., EVAL). An op-code of 2 indicates that the client will not be sending any more requests (i.e., END). An op-code of 3 indicates that the server should cleanly shutdown (i.e., QUIT). Any client can send a QUIT request. When quitting, all requests that have not been started should be flushed, and all clients should be told to quit.

A client sends the server a sequence of messages (spaced approximately 1 second apart). The responses from the server (actually, child processes) are sent back to the client through reply FIFOs (described below). The client then sends the result to stdout. Thus, clients ”simulate” request generators that send requests to the server through a FIFO at approximately 1 second intervals.
Guidelines

- A shell script (or shell sequence in a Makefile) controls the overall initiation of the client and server programs. The call lines of the shell script (A.sh), clients (Ac), and server (As) are shown below where M is a non-negative integer (default value of 3), and n is a non-negative integer between 0 and M-1.

\[
\begin{align*}
\text{A.sh} & \ [M] \\
\text{As} & \ [-v] \ [-m \ M] \ > \ \text{Log} \\
\text{Ac} & \ [-v] \ < \ \text{Requests.m} \ > \ \text{Replies}
\end{align*}
\]

- Each client should read requests from stdin. These requests contain the opcode and N-values only. It should also create its client-specific FIFO ReplyFifo.m where m is its process ID.

- The server should create the well-known FIFO RequestFifo and read requests from it until it sees that the M clients are done sending requests. If there is a -v flag, it should log all requests (with a newline character separating each message) to stdout. It forks a child to handle each request to compute a prime number. This child computes the prime number and sends the result back to the client using the client-specific FIFO named ReplyFifo.PID where PID is the process ID of the client.

- The server should allow at most only one outstanding request from each client.

- The reply messages sent by the server (actually a child) to the client over the reply FIFO should have a format similar to the request messages. But the op-code should be 4 (i.e., REPLY); the PID field should be the PID of the child process that computed the prime number; and the N-field should contain the prime number;

- The parent does not exit until it gets and handles the QUIT request. Furthermore, before quitting, the server should report in tabular form on stdout the accumulated user and system times consumed by the child processes on each clients behalf.

- Remove all FIFOs when they are no longer needed.

What to Submit

Hardcopy documentation (described below) should be submitted in class. Also, the documentation, the source code, Makefile, and test cases should be submitted electronically (see below).

1) A summary of the status of your program (e.g., limitations, test failures) and documentation.

2) An overview section that describes and discusses the structure of your program. In particular, describe how you handle each of the requests.

3) A discussion of why you think your program is correct. Include a discussion of test results.

4) Describe any features that you have added to your program that are beyond the requirements.

Demonstration

You must demonstrate your program to a grader/consultant. Details of this demonstration will be provided in class and on the course Web page. Also, see the section Late Policy.

Electronic Submission
The end result should be that you mail to kenw@arl.wustl.edu a single shar (shell archive) file containing your files. Do NOT submit object code or executables. The following command will create a shar file named A.shar containing the files Makefile, run1, ..., A.sh, As.c, and Ac.c:

```
shar Makefile run1 ... A.sh As.c Ac.c > A.shar
```

I should be able to enter the following to unpack your shar file, compile your code (using gcc, g++, or CC), and run A.sh for $M = 4$:

```
sh A.shar
make compile       # compile the client and server code
make run           # run the main test
```

If your program does not work in its entirety, supply the best test case that works, submit a description of the test as part of the documentation, and indicate how to run this test case(s). But also supply a listing of code that shows work in progress and describe how it is different than your working code.

**Late Policy**

All hardcopy documentation must be submitted in class by the due date. However, you can receive an upgrade of your score by improving your implementation and demonstrating the improved version. Demonstrations can be given within two weeks after the due date. At the time of the demonstration, submit only the revised Status section of your documentation and state clearly the improvements that were made and any remaining bugs and deficiencies.

**EXTRA CREDIT**

Implement the reasonable scheduler described in Homework 4, Problem 2 but use the signal SIGJUSR1 to tell a child to record its usage statistics in the file with name equal to its usage.PID where PID is the PID of the child. You should read about how to implement a signal handler (see sigaction(2)), but there will be a code fragment on the Web that contains an alarm signal handler. Also, you will need to read about select(2) which will give you a semantic equivalent to the waitfor() system call described in Homework 4. Also, the course Web page will contain a short code fragment demonstrating the use of select(2).

Describe fully your implementation, supply output and an explanation of that output that demonstrates that your implementation is working properly. You should only submit one set of documentation; i.e., do not submit a separate document for the main part of the Project.