CS342 Fall 2015 - Project 2
IPC and Multi-Threaded Programs

Assigned: 11.10.2015
Due date: 25.10.2015, Sunday**,** 23:55

In this project you will develop a client-server application running in a Linux system (both client and server will run in the same machine). You will implement a multi-process, multi-threaded server program (server.c) and a single-threaded client program (client.c). Your server will be running all the time and one or more clients will be getting service from it. Clients and server will be communicating using Linux POSIX message queues. There will be a separate child process serving each client and a child process will create multiple threads to process multiple files (one thread per file).

The service that the server will provide is word-counting service. A client will give a set of filenames and a keyword to the server. This will be a *request* from the client to the server. The request will be passed through a message queue that the server has created earlier. All requests from clients will be passed through this message queue (request-queue) to the server. The server will serve requests from this queue. For each request, server will create a new child process that will serve the request. The child process will create a separate thread for each filename appearing in the request.

A thread will search the respective file for the given keyword and will find out at which lines the keyword appears and how many times in total the keyword appears. Each occurrence of a keyword will be counted. Consider only exact-matches. That means, for example, a keyword ”hello” matches to word “hello”, but does not match to word “helloo” or “hellohello” in a file. If a keyword appears multiple times in a line (for example, as in line “hello hello how are you”), the line will appear multiple times in the output.

If there are N files to search, the child process will create N threads. When all created threads finish execution, the child process will have the complete information about in which files and where in those files the keyword appears. This result will be passed to the respective client using a reply-queue. The reply-queue is to be created by the client. The client passes the name of this queue to the server together with the request. In this way the server and child process will know which reply-queue to use for the client. Each client will use its own reply queue to get the result back. After sending the result, a child process will terminate.

The server is to be run first. It will create the request message queue and will wait for incoming requests. It will run all the time (possibly in the background; you can start a process in the background by using the “&” sign at the end command line) until terminated with ^D. The server will be invoked as follows:

server <req-qname>

Here, <req-qname > is a filename corresponding to a message queue that is to be created by the server. This filename is the identification (name) of the queue; there is no file content associated with it. The queue will be identified with this name by the server and by all clients.

A client program will be invoked as follows:

client <req-qname> <word> N <fname1> …<fnameN>

<req-qname> is the name of the request queue through which the client will send a request to the server. The <word> parameter is the keyword to search for. <N> is the number of files to search through. <file1>…<fileN> are the names of those files. We assume those files are accessible by the server.

When invoked, the client program will form a request (message) using the given parameters and will send the request to the server process. The request will be sent through the request queue. Before sending the request, the client will also create a message queue, a reply queue, that it will use to get the result back. The name of this reply queue must be unique and this can be achieved by using the pid of the client as part of the name corresponding to the queue (see getpid() function). The name is nothing but a filename, but there will be no file associated with this name. The name will correspond to the reply queue. This queue name will also be sent to the server together with the request message that client is sending to the server. In this way, the server child process can access (open) this reply queue and can send the result back (inside one or more messages) to the client after threads compute the result. A client gets the result through its reply queue and prints it to the screen. Then the client can destroy its reply message queue and terminate. A client will issue just one request per invocation.

Each line of output will have the following format:
<filename> [<wordcount>]: <linenumber> … <linenumber>

An example output printed by a client can be:

<f1> [3]: 34 90 123
<f2> [5]: 1 6 6 10 10
<f3> [10]: 5 5 6 6 6 14 14 14 14 23

That means, the client has provided 3 filenames (f1, f2, and f3) to the server to search for a given keyword. After server computes the result, the client prints this out. In f1, the keyword appears 3 times and at lines 34, 90, and 123. In f2, the keyword appears 5 times, at lines 1, 6, and 10. The keyword appears twice in line 6 and in line 10.

Experiments: You will also do some experiments and will write a report. Design your experiments. Measure, for example, how much time it takes to search though one input file, two input files, …, M input files, and so on. Run multiple clients concurrently (in different terminal windows) and measure their run time.

Submission: Submit through Moodle. Put your server.c, client.c, Makefile, report.pdf, and a README file into a project directory; tar and gzip the directory; and upload your project2.tar.gz file.

Clarifications:

- Use POSIX message queues and POSIX Pthreads.
- The maximum number of concurrent clients: 5.
- Maximum filename length: 128.
- Maximum number of input files specified by a client: 10.
- A keyword can contain only alphanumeric characters.
- Files are text files (ascii text files).
- You will do the project individually.
- Use C programming language.