CS342 Operating Systems – Spring 2018
Project 1: Processes and IPC

Assigned: Feb 13, 2018
Due date: Feb 28, 2018, **23:55** (Moodle)

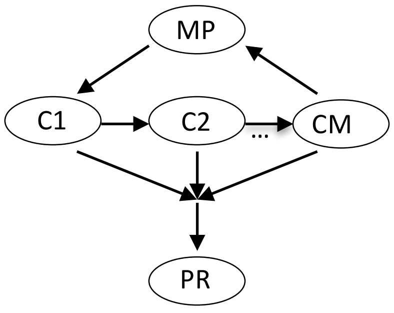
You will do this project individually. You have to program in C and Linux. You are required to use the following distribution of Linux: **Ubuntu 16.04 – 64 bit**.

Part A: Processes and Pipes [35 points]
Objective: Practicing process creation, process communication via pipes, multi-process application development.

You will implement a **multi-process** application called **prime** that will find and print all the prime numbers in an integer sequence. The program will take two parameters, N and M:

prime <N> <M>

N is the largest integer in the integer sequence, and M is the number of worker child processes that will be used in computing the primes. Let us call the worker child processes as C1, C2, …, CM. There will be **unidirectional pipes** between the processes: from Main process (MP) to C1, from C1 to C2, …, from CM-1 to CM, and from CM to Main (MP). There will be another child process created (let us call it PR, printing process), that will be responsible for collecting and printing out the identified primes. There will be an additional pipe that will be used to send primes from worker processes C1, C2, …, CM to PR (one pipe is enough). All child processes will send to this pipe when they identify a prime number. PR will receive primes from this pipe and will print them to the screen.



The primes will be computed as follows. The Main process (MP) will generate a sequence of integers, starting with 2 and ending with N, like 2, 3, 4, 5, …. N. It will send those integers one by one through the pipe between Main and C1. C1 will receive these integers, and will identify the first integer in the sequence, 2, as a prime number and will send the identified prime to PR through the respective pipe. It will send the sequence that it is receiving to C2 while **removing** the identified prime and all its **multiples** (2, 4, 6, 8,…). C1 does not have to wait until it receives the whole sequence to start sending it to C2. C2 will receive the updated sequence and will identify the first integer received as a prime (which is 3 in this case) and will send the identified prime to PR through the respective pipe. C2 will send the sequence it is receiving to C3 while removing all multiples of 3 (3, 9, 15, …). This will go on. CM will receive a sequence, will identify the first integer as a prime, will send the prime to PR, and will send the sequence to Main while removing the multiples of the identified prime. Main process will receive the sequence and will buffer the sequence in a linked list (will not remove any integer). Main process will not start sending the buffered sequence to C1 before it has finished sending the previous sequence to C1. At the end of a sequence, the Main process may send a marker integer, marking the end of the sequence. In this way, processes can understand the end of a sequence and the start of a new sequence. This will go on until all primes in range [2…N] are identified and printed out; that means until we arrive to a sequence of length 0. At the end, all processes will terminate and the pipes will be removed.

Maximum value of M is 50. Minimum value is 1. Maximum value of N is 1000000 (1 million). Minimum value is 1000. If M is 1, only one worker child will be created. Besides that there will be a main process (MP) and printing process (PR). Hence, the minimum number of processes that the application can have is 3. An example invocation of the program can be:

```
prime 100000 10
```

This will cause 10 child processes to be created. There will be one main process (MP) and a printing process (PR) as well: hence a total of 12 processes will execute concurrently. There will be a total of 12 pipes created and used.

Part B: Processes and Message Queues [35 points]

Objective: Practicing message queue creation and usage as IPC among multiple processes.

Develop a similar program that will do the same thing, but this time use Linux **message queues** instead of pipes for communication between processes. The program will be called **mqprime**.

```
mqprime <N> <M>
```

In this part, the maximum value for M can be 5, and the minimum value can be 1. Hence there will be at most 7 message queues used by the application. When the application finishes, all processes will be terminated. Make sure that the application also removes the created message queues before it terminates.

You will use **POSIX message queues**. You can use “man mq_overview” to get information about POSIX message queues. There are a lot of information available on the Internet as well. Your C code needs to include the mqueue.h header file.

```
#include <mqueue.h>
```

Note that your program using the POSIX message queue API must be compiled with **–lrt** option to link against the real-time library **librt**.

```
gcc -Wall -g -o mqprime  -lrt  mqprime.c
```

Part C: Experiments [30 points]

Objective: Practice designing and conducting experiments and applying knowledge and skills acquired in the Probability and Statistics course.

In this part you will design and conduct some experiments to evaluate the completion time of your program. Measure the completion time of your programs for various values of N and M. For each fixed values of N and M, you can repeat the experiment and use the average value for completion time. Report and plot the results. Try to draw conclusions. You will do this for both programs. Write a report for this part. Final report will be converted to PDF.

Submission

Put all your files into a **project directory** named with your ID, tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to **Moodle**. For example, a student with ID 20140013 will create a directory named 20140013, will put the files there, tar and gzip the directory and upload the file. The uploaded file will be 20140013.tar.gz. Include a **README.txt** file as part of your upload. It will have your name and ID at least. Include also a **Makefile** to compile your programs. We want to type just “make” and obtain the executables. Do not forget to put your **report** (PDF form) into your project directory.

Additional Information and Clarifications

- *Suggestion: work incrementally; step by step; implement something, test it, and when you are sure it is working move on with the next thing.*
- More **clarifications**, additional information and explanations that can be useful for you may be put to the **course website**, just near this project PDF. Check it regularly.
- You can use Piazza for questions and discussions.
- The objective in this project is not to see a speed-up by use of multiple processes. The objective is to practice use of multiple processes.