CMSC 421 – Operating Systems
Krishna Sivalingam and Dennis Frey
Spring 2006
Due Date: Wednesday, **Mar 2, 9PM**. On-Line Submission via *submit* command
NO LATE SUBMISSIONS

1 Project description

The objective of this project is to implement solutions to the following synchronization problem. The implementation will use: multiple processes (obtained using fork()) and shared memory; and multiple threads using the *Linux pthreads API* (see Section 4.3.1) of the textbook.

The implementation will be done in Linux/C/C++ platform and MUST run on one of the Linux machines in the ITE 240 Lab. Please DO NOT develop or run the programs on servers such as linux1.gl, linux2.gl, linuxserver1.cs, linuxserver2.cs. You can use one of the machines in the ITE 240 lab or your own Linux system.

1.1 Factoring problem

The objective is to implement a multi-threaded solution to the prime factoring problem.

INPUT: An integer, $N > 1$.

OUTPUT: Factorization of N expressed in terms of powers of prime numbers. For example, $12 = 2^2 * 3^1$; and $1050^1 = 2^1 * 3^1 * 5^2 * 7^1$.

The main program will read the numbers N, P and Q from the command line, as follows:

```
% ./pf -n N -p P -q Q
```

It will then:

- ▷ Spawn a set of P prime number determination (PND) threads. The numbers from 1 to $\sqrt{N}$ will be partitioned among the PND threads so that two threads do not work on the name number. That is, each thread will given be a subset of numbers from the set $\{1, 2, 3, \ldots, \sqrt{N}\}$ and the thread will determine whether the numbers in its allocated subset are prime or not. When a prime number is determined, the PND threads adds the number to a common data structure shared among the main thread, the PND threads and the FAT threads (see below).

  The primality testing can be done using simple brute force logic – we do not plan to test with extremely large numbers.

- ▷ Spawn a set of Q factoring threads (FAT). Given N and a prime number (X) from the common data structure:

  - ⋆ If X evenly divides N, this thread will determine the value f, where $f > 0$, is an integer and is given by $N = X^f * Y$. Note that $Y = 1$ or Y will not be evenly divisible by X. For example, for $N = 120$ and $X = 2$, $24 = 2^3 * 15$ and $f = 3$; $N = 16$ and $X = 2$, $f = 4$.
  - ⋆ If X does not evenly divide N, then $f = 0$.

The value f is then written to sent to the main thread, which will collect all the (N, X, f) values and produce the output as described below.

The output from the program will be in the format:

$$1650^1 = 2^1 * 3^1 * 5^2 * 11^1$$

with the prime numbers that are factors of N listed in increasing order.

1.2 Doonut Eating Contest

Consider a Junking Doonut shop with the following specifications.

Doonut Makers:

- ▷ There are $M \geq 5$ doonut makers, who will make doonuts and place them on doonut shelves.
- ▷ There are a set of S shelves, each with capacity to hold up to D doonuts.
- ▷ The doonut maker is in two states: making a batch and napping.
  - ⋆ The doonut maker naps for a time Z milliseconds; and then
  - ⋆ enters "making doonuts" state, where he/she makes $B \geq 10$ doonuts per batch. It takes C milliseconds to make a doonut.
- ▷ Every doonut made by a maker will have an identifier that is unique to the doonut maker given by (Maker ID, Batch ID, Doonut ID). The Doonut ID gets set to zero for each batch.
- ▷ The initial state of each doonut maker is selected randomly (with equal probability).

System Constraints:

- ▷ Two doonut makers cannot add to the same shelf at the same time. When a doonut maker selects a shelf, but finds that it is in use by another maker, it waits until the shelf becomes free.
- ▷ When the first doonut of a maker's batch is ready, it will add it to a randomly selected doonut shelf.

  You will use the functions *random* and *srandom* functions. Run `'man random'` for more information. The function *srandom* will be called once and initialized with the seed value (N), that will be obtained from the command line.

  It will continue to add to this shelf until either 10 doonuts have been added to this shelf in succession or the shelf is full. At this time, the doonut maker selects a different (random) shelf, which might be the same as the last one.
- ▷ If there is no empty shelf when a doonut-maker has made a doonut, s/he waits until a shelf becomes available. It does not make a doonut while so waiting.

Hungry Students:

- ▷ There are $U \geq 7$ hungry undergraduate students who will remove and eat the doonuts (whether they pay for it or not is our concern) from the shelves.
- ▷ It takes E milliseconds for a student to eat a doonut.
- ▷ The student will be in two states: hungry and studying.
  - ⋆ After entering "hungry" state, a student be in this state until he/she has consumed a certain number of doonuts (H).
  - ⋆ Once a student's hunger is satisfied, they will spend R ms in "studying" state, before becoming hungry again.
  - ⋆ Note: Students do not have "sleeping" state.
- ▷ The initial student will be: hungry, for all students.
- ▷ Every doonut eaten by a student will have an identifier that is unique to the student given by (Student ID, Doonut ID). The Doonut ID gets set to zero for each student at the start of the program.

Some constraints:

- ▷ A student can take a doonut from any available shelf.
- ▷ Two students cannot take a doonut from the same shelf at the same time.
- ▷ A student can take a doonut from a shelf at the same time that a maker is adding a doonut to that shelf, providing that there is at least one doonut on the shelf when the student looked at it.

What to Do: Your task is to implement the synchronization mechanisms for this problem using **semaphores** and/or **locks**. In addition, you will only use processes and shared memory, i.e. no threads. Each student and maker will be represented by a unique process.

You will also implement driver code that will take as input as follows:

```
% ./dm -m M -s S -d D -z Z -c C -b B -u U -e E -h H  -r R -n N
```

Print an error message if the input values of variables do not meet the minimum constraints above.

The driver code will generate a random number of maker threads (between 5 and M), and a random number of student threads (between 7 and U). The maker IDs will range from 1 through maximum; student IDs will range from 1 through maximum.

N.B.:

1. Note that you can simulate doonut-makers making doonuts and students eating doonuts by having them sleep using a suitable sleep function.
2. Please DO NOT develop or run the programs on linux1.gl, linux2.gl, linuxserver1.cs, linuxserver2.cs, etc. You can use one of the machines in the ITE 240 lab or your own Linux system.

What to Output: There will be one output file per process, with output filename in the format **out_maker_1.txt**, etc. for makers and **out_student_1.txt** for students. Any other output messages can go to the screen or to a separate file **out_other.txt**.

Each process will print its type (maker or student) and respective ID initially. It will also output a message when the following events take place: a doonut is produced, a doonut is put on a shelf, waiting for a shelf to be free, a doonut is removed and eaten, or a student is waiting for a doonut to be produced. Each event message will contain the corresponding doonut ID, if appropriate for that event.

2 Starting Point

Sample programs, in C, with invocations of basic pthreads, fork(), shared memory and semaphores, will be made available from the class website.

3 Sample Session

Assume that you have created the necessary files and the corresponding executables in your PROJECT1 directory.

```
% cd PROJECT1
% make pf
% ./pf -n 647 -p 1 -q 1
..
% ./pf -n 647 -p 3 -q 2
...

% cd PROJECT1
% make dm
% ./dm  -m 5 -s 4 -d 15 -z 25 -c 10 -B 12 -u 7 -e 5 -h 10 -r 30 -n 10000
.. Output goes to various files (and screen if so designed)
..
% cat out_maker_1.txt
..
% cat out_student_2.txt
..
```

4 What to Submit

Now log into one of the campus GL machines. Change to your project directory. Make sure that ONLY files related to this project are in this directory - we do not want any temporary files, music files, etc.

Submit your project using the *submit* command. For help with submitting, click on the *Miscellaneous* link on the course homepage.

```
submit cs421 proj1 <list of files>
```

You can test the program execution after submission using *submitmake* and *submitrun* routines that are provided in Mr. Frey's public directory (`/afs/umbc.edu/users/d/e/dennis/pub/CMSC421`).

Submit the following files:

- ▷ Source Files
- ▷ Makefile
  Typing command 'make' at the Linux command prompt **MUST** generate all the required executables. The executables **MUST** be named **pf** and **dm** as in the samples above.
- ▷ A Script file obtained by running UNIX command *script* which will record the way you have finally tested your program. The script file will contain at least ONE run each for the prime factoring problem and the doonut problem.
- ▷ a README file for the TA. The README should document known error cases and weaknesses with the program. You should also document if any code used in your submission has been obtained/modified from any other source, including those found on the web (e.g. Prime determination algorithm). If you helped any other 421 student and/or took help from/discussed with any other student, please describe it here.
- ▷ a COMMENTS file which describes your experience with the project, suggestions for change, and anything else you may wish to say regarding this project. This is your opportunity for feedback, and will be very helpful.

5 Help

1. WARNING ABOUT ACADEMIC DISHONESTY: Do not share or discuss your work with anyone else. The work YOU submit SHOULD be the result of YOUR efforts. The academic conduct code violation policy and penalties, as discussed in the class website, will be applied.
2. Ask questions EARLY. Do not wait until the week before. This project is quite time-consuming.
3. Implement the solutions, step by step. Trying to write the entire program in one shot, and compiling the program will lead to frustration, more than anything else.
4. List of Useful URLS:
   - ▷ http://www.llnl.gov/computing/tutorials/pthreads/

6 Grading

- ▷ Prime Factoring Problem: 25 points
- ▷ Doonut problem: 75 points

No README/COMMENTS: -5 points;   No Script File: -10 points;   Incomplete Compilation: -10 points