Exercise 2:UNIX system programming

Mr. Roee Leon
Shenkar – 2016-2017C term.

Bonus date: August 28th 2017
Due date: August 30th 2017

Instructions

1. Please try the POSIX threads examples given in the lecture before starting this exercise
2. Please try at least one example with *mutexes & condition-variables* before starting this exercise
3. Please send the compiling project in 1 ZIP or tarred gzipped file containing only. The zip file name should be your “your ID number” i.e. a filename should be 0123456789.tgz or 9876543210.zip
Do not submit RAR or 7z etc.
3. Please ensure your project compiles with make all instruction
4. Please ensure make clean deletes your binary and any cruft.
5. Please add any additional instructions and comments to README.TXT file.
6. All dates refer to 23:59:59 Israel standard time on the date.
7. Submission by or before the bonus date will result in 10% grade BONUS
8. Submission by the due date is allowed.
9. Submission after the due date will be penalized by 25% for late submission (automatic) + 10% per day after the second day. (ie. Late submission by 1 minute will be penalized by 25%. Late submission by 4 days will result by 55% penalty) – PLEASE you are responsible adults. Manage your time and meet the deadline.
10. You need to implement this exercise on your own. You are allowed idea exchange ideas with your fellow students but not to exchange code. You are allowed to use web resources though. If you use any code you found on the Internet SPECIFY that in the README.TXT file.
If two students submit identical work a disciplinary action will be taken. (Even if both copied the same source from the net)
11. It is your responsibility to ensure your work compiles and run on the first submission. 2nd submission (and 3rd and 4th etc.) will be penalized by 10% penalty per “re-submission.” exercises that do not compile will receive 0%.
12. Please follow KNF, 1TBS or BSD code style. Exceptionally good code will receive 10% BONUS. Other coding standards (Allman, GNU etc.) are allowed as long as they are consistent but will not result in bonus. Kludge will receive 10% penalty. Students coding in Hungarian notation ~~will be shot. Twice.~~ will not be eligible for code quality bonus.

13. The code should be self-documented. You may comment critical parts of the code. Excessive commenting does not consist of good code and will be penalized.

Learning objectives

1. Grok POSIX threads
2. Grok thread synchronization using mutexes & condition variables
3. Simple system administration tasks on UNIX
4. Experience some parallel algorithms implementation

Cover Story

In this exercise, you will build your own thread pool manager.
The system will be composed of three main components:

1) Thread Pool Manager – Responsible of distributing tasks amongst its available threads
2) Task Feeder – Responsible of loading new tasks into the Thread Pool Manager (Bonus)
3) Main application – The main application (thread) is responsible for approximating PI using the Monte-Carlo method.

Exercise 1 – Thread Pool Manager (70%)

The thread pool manager has *N* available threads (*N* is given as a parameter to the program). The thread pool manager always waits for new tasks (separate thread) to be added by the Task Feeder (Bonus) or simply by assigning it tasks from main. These tasks should then be **distributed** between the available worker threads.

Note that the tasks are completely oblivious to the thread pool manager.

Hints:

1. Represent a task by the following base structure (you may add more fields):
```
struct Task
{
        void* (*task_f)(void* arg);
        void* arg;
};
```

2. Keep two data structures within the thread pool manager:
    a. A task queue (You may use STL's std::queue or just implement a simple linked-list based queue)
    b. An array of available threads (of size *N*)

3. Do not forget synchronizing between the working threads (e.g. access to the task queue) – *mutex_lock/mutex_unlock/pthread_cond_signal/pthread_cond_wait*

Exercise 2 – Main Application (30%)

The main application is responsible for approximating PI using the Monte-Carlo method. The Monte-Carlo method for PI approximation may be found here (Along with the algorithm):
http://www.eveandersson.com/pi/monte-carlo-circle

1) The main application receives as a parameter the number of iterations to be done by the Monte-Carlo method to estimate PI.
2) The main application is the **only** component familiar with the calculation.
3) The main application **must** define a set of tasks (Divide and Conquer) to be handed to the Task Feeder (Bonus) **or** simply hands it to the Thread Pool Manager by itself. Each task is responsible for solving its relative part of the PI approximation calculation.
   The main application needs to combine the sub-solutions to an overall solution

Exercise 3 –Task Feeder (10% - Bonus)

The main application, the only one familiar with the actual “work”, defines a set of tasks to be handed to the Task Feeder. The task feeder is then responsible to load these tasks to the Thread Pool Manager.

Bonus

1. (10%) Task log – Support a log that contains the list of all tasks given during the lifetime of the program along with the executing thread ID
2. (10%) Keep alive time – In case there are no available threads in the thread-pool (and there are pending tasks), allow setting of a Keep-Alive time on which threads that did not finish within the given Keep-Alive time will be terminated (thus, will be available to work on other tasks)