# Assignment #1: Threads and Synchronization

CS 416
Professor - Ricardo Bianchini
TA – Cheng Li

Group sizes: exactly 3 people per group

Due date is Oct 14th, 2011 at 11:55 PM. Late submissions will not be accepted.

Submission: Electronic, via Sakai.

## 1. Overview

This assignment will give you the opportunity to gain some experience with threads and synchronization. All students will have to solve the first 2 problems. Students in the honor section will have to solve the latter 2 problems as well. Students who are not in the honor class can do these latter problems for extra credit.

## 2. Assignment

### 2.1 Hungry Hamster Problem

Use threads to implement the following problem (you can only use pthread condition variables, semaphores, and/or mutexes. Any other synchronization primitives will not be allowed).

Problem statement:
There are n baby hamsters and one parent hamster. The baby hamsters eat out of a dish that initially contains a total of x beans (x is equal to the number of baby hamsters). Each baby hamster repeatedly eats one bean for some time (process_time), sleeps for a while (think_time), and then comes back to eat. When the dish becomes empty, the next baby hamster will start to wait for the parent hamster. The parent hamster refills the dish with x (x is equal to the number of baby hamsters) beans at some interval (parent hamster's think_time). While the parent hamster is refilling the dish, the baby hamsters are not allowed to eat. In addition, when the baby hamsters are eating, the parent hamster is not allowed to refill the dish. This pattern repeats forever. Represent the hamsters as threads and develop code that simulates their behaviors.

The number and behavior of baby hamsters and the parent hamster will be determined by the lines of the input file (called input1.txt, which will be provided by the TA). The input file will have the following format, "id role think_time process_time", where "R" stands for baby hamster, "W" stands for parent hamster. Example of input file:

```
1 W 8 1
2 R 2 1
3 R 1 1
…
```

Line 1 indicates that the parent hamster (id=1) will be thinking for 8 ms and refilling beans for 1 ms.
Line 2 indicates that a baby hamster (id=2) will be thinking for 3 ms

and eating one bean for 2 ms.

Two constraints for this problem:
1. Parent-baby exclusion: cannot have parent hamster refilling and any baby hamster eating at the same time.
2. Baby-baby allowed: multiple baby hamsters can eat at the same time.

Use appropriate mutex/condition variables for synchronization to implement this system. Print out information to another file (called output1.txt) about the behavior of your system. Each line of the output should be printed out before each baby/parent hamster is thinking, eating, or refilling. Also, print out message when baby/parent acquired/released the lock. If they are holding different types of locks, then mark them as lock?. A sample output would be:

…
Parent (id=1), thinking, think_time=8
Baby (id=2), acquired lock1
Baby (id=2), eating, process_time=1
Baby (id=2), released lock1
Baby (id=3), thinking, think_time=3
…

**2.2 Refilling Problem**

There will be multiple parent hamsters. Each parent hamster will fill the dish with x beans at some interval (parent's think_time). When one parent finds the total number of the beans to be more than half of the baby hamsters, then she will increase her think-time at an exponential rate (1, 2, 4, ..., 2^n). If she finds that the number of beans is smaller than half of the baby hamsters, she will decrease her think-time at an exponential rate. Assume each time a parent hamster refills the same amount of beans as the number of the baby hamsters. Assume the minimal and maximal think-time for each parent is between [1, 256] and that no two parents can be refilling the beans at the same time (in addition to the constraints in the previous question). Finally, make sure that the following parent-priority strategy holds: when there are both parent hamsters and baby hamsters waiting to do their duties, baby hamsters can only start eating after no parent hamster is waiting to refill.

Parameters for the baby hamsters and parent hamsters are read from an input file with the same format as before.

For this problem, we will provide several sample input files (called input2-?.txt). You need to print the same format of information as in the previous problem to show your implementation behaves as intended. The output files shall be called output2-?.txt.

**2.3 Better Back-off Scheme (Honors section)**

Can you come up with a better back-off scheme that (1) maximizes the throughput of the system, meaning that baby hamsters would eat more beans than the previous system? (2) Would that be different than minimizing the average starving time for the baby hamster, meaning that the time that baby hamsters have to wait for a parent to refill the dish is shorter than the previous system? Can you achieve both at the same time or only one of them?

For this problem, we expect you to explain the idea of your scheme

first, and then your implementation. Finally, your evaluation must show that your scheme is able to solve this problem.

### 2.4 Better Locking Scheme (Honors section)

Can you come up with a better locking scheme in which baby hamsters will not be blocked by parent hamsters? What about the parent hamsters not being blocked by the baby hamsters? Here not blocked means baby hamsters would be allowed to eat when the parent hamsters are refilling the dish and vice versa. But the parent-parent exclusion constraint should still hold.

For this problem, we expect you to explain the idea of your scheme first, and then your implementation. Finally, your evaluation must show that your scheme is able to reduce the overhead of locking used in previous problem.

## 3. Report

Write a report as specified for each problem (with both text, numbers, and/or graphs) relating the experiences you had, and the analysis of your solutions (see each problem). The report will be a non-trivial part of your grade, so please prepare it carefully.

NOTE: the length of the report won't help your grade, but its content will. Be concise, write only what you need to convey.

## 4. Hint and Recommended Procedure

Some of you might not know where to start. For that reason, I recommend that you (a) think hard about your model/code before you start coding and compiling, and (b) make modifications incrementally, not all at once. If the program fails to produce the correct result, it's a good guess that the last set of changes you made is the problem. (It's not certain, of course.) Making sure that the set of changes is small can help find the problem. Additionally, you might find that incrementing the code (print's, or similar sorts of things) helps you, and even that building some additional tools (e.g., a new test case, and a scenario that involves the corner cases) are worth the effort.

You might need to first figure out what is/are the shared resource/s, the beans, the dish, the parent hamster, the output file, etc? Then you need to use some techniques (locks, conditional variable) to protect this/these shared resource/s? Then make sure all the constraints are respected. For example, we can interpret the baby hamster eating for 2 ms as the thread acquiring and holding a lock for 2 ms.

## 4. Submitting your Assignment

* What? Your report and all source files that you have modified or created. Use the tar command as follows:

```
$ tar cfz yourname.tar.gz report.txt file.1 file.2 file.3 .... file.n
```

* How? Hand in a file called yourname.tar.gz on Sakai https://sakai.rutgers.edu/portal

## 5. Grading

The project will be graded as follows:

| | |
|---|---|
| Problem 1 | (40 points) |
| Problem 2 | (40 points) |
| Report | (20 points) |
| Problem 3 | (20 points) |
| Problem 4 | (40 points) |

## 6. Miscellaneous

### 6.1 Useful links

* Thread programming tutorial:
https://computing.llnl.gov/tutorials/pthreads/

## 7. Supplement

1. For each group (exactly 3 people), please fill out your group member information (team leader and team members names) to the wiki page on our sakai course paper. I will create an entry for this assignment. Please do this before next recitation (Oct 5th).

2. The sample input files we provide are intended to let you run your program and observe the output. When grading your assignment, we will be using a batch of test input files to check the correctness and robustness of your program. You can assume the maximum number of threads in your system is 256. Our suggestion is that you test your program extensively before you hand it in.

3. To input data from an input file into your program, you can use fscanf() or anything you like. There should be exactly one space between each symbol. You can hard code this, if you'd like. You can also test your program using our sample input file.

4. Some students have asked if c++ is allowed. Yes, it is. But you are only allowed to use pthread to implement your locking/synchronization scheme.

5. To input the file names, we recommend that you use argv[1] for the input file name and argv[2] for the output file name. To input how many seconds your program should run for, you can use argv[3]. You can put all your code into one source code file, but you are recommend to create one source file for each problem. Also, you can create a Makefile and generate different binaries for answering each problem in the assignment.

6. Your programs need to print out the important statistics about each thread at the end of your program to STDOUT. The formats for baby hamster and parent hamster threads are:

REF=5073 (parent hamster thread)

TP=641.000000, WT=539ms (baby hamster thread)

TP is the throughput (how many beans were consumed by the baby hamster during the execution), WT is the average starving time (how much time the baby hamster waited for food), and REF is the number of times the parent hamster refilled the dish.

7. For question 3 (better back-off scheme), you should always increase the parent think time when the number of beans is larger than half the number of baby hamsters, and decrease the parent think time when the number of beans is smaller than half the number of baby hamsters. Think of this question in an economic and green way, which means refilling always with the shortest interval is not a good idea.

8. Sample of output files, input files, and statistics have been uploaded to sakai. Item means the number of beans currently in the dish, print this before parent hamsters refill the dish.
“parent(id=1), refilling, process_time=1, item=19”

9. The system starts with x beans(x is the number of baby hamsters). When refilling, parent hamsters refill the same amount of beans as the number of the baby hamsters( also x).