CS450/599: Parallel Programming

Assignment 3: Programming with PThreads #2

Preliminaries

You are expected to do your own work on all homework assignments. You may (and are encouraged to) engage in discussions with your classmates regarding the assignments, but specific details of a solution, including the solution itself, must always be your own work. See the academic dishonesty policy in the course syllabus.

Submission Instructions

You should turn in an electronic archive (.zip, .tar., .tgz, etc.). The archive must contain a single top-level directory called CS450_aX_NAME, where "NAME" is your NAU username and "X" is the assignment number (e.g., CS450_a1_mg1234). Inside that directory you should have all your code (no binaries and other compiled code) and requested files, named exactly as specified in the questions below. In the event that I cannot compile your code, you may (or may not) receive an e-mail from me shortly after the assignment deadline. This depends on the nature of the compilation errors. If you do not promptly reply to the e-mail then you may receive a 0 on some of the programming components of the assignment. Because I want to avoid compilation problems, it is crucial that you use the software described in Assignment 0. Assignments need to be turned in via BBLearn.

Turn in a single pdf document that outlines the results of each question. For instance, screenshots that show you achieved the desired program output and a brief text explanation. If you were not able to solve a problem, please provide a brief write up (and screenshots as appropriate) that describes what you tried and why you think it does not work (or why you think it should work). You must provide this brief write up for each programming question in the assignment. The questions may provide test scripts to (try) and validate your programs. When test scripts are used, show the output of both the program and the testing script (this is 2 executions, because the script takes your program output as input). I will demonstrate this in class.

This pdf should be independent of the source code archive, but feel free to include a copy in the top level of that archive as well. Let me know if there are problems uploading multiple files to BBLearn.

Problem Overview

In this exercise, you must implement the following "simulation". In a bakery that exclusively bakes chocolate chip cookies we have several bakers (i.e., threads). Each baker has its own oven that is always hot, its own set of supplies and work area. Each baker follows the following pseudo-code:

```
// Baker thread
for (int i=0; i < 10; i++) {
work (i.e., prepare dough, cut cookies, put them on a baking sheet)
get oven mitts from the oven mitt rack
put cookies in the oven
wait for cookies to be baked
remove cookies from the oven
put oven mitts back onto the oven mitt rack
}
```

In the pseudo-code above the "work" and "wait for cookies to be baked" operations simply print a message to standard out and then sleep a random number of microseconds (using `usleep`) between 0.2 and

0.5 seconds. The “put cookies in the oven” and “remove cookies from the oven” operations simply print a message to the console. See the sample output. The point of the exercise is to implement the “get oven mitts” and “put back oven mitts” operations, which should also print messages, as seen in the sample output below.

Problem Details

The bakery is dysfunctional in that the owner does not want to buy oven mitts for each baker. Instead, there are:

- 3 left-handed oven mitts
- 3 right-handed oven mitts

We have three kinds of bakers:

- Left-handed bakers who require only one left-handed oven mitt
- Right-handed bakers who require only one right-handed oven mitt
- Cautious bakers who require both a left-handed and a right-handed oven mitt. A cautious baker first takes a left-handed mitt, and then takes a right-handed mitt.

The bakers are named/numbered as follows:

- Left-handed baker 0, Left-handed baker 1, Left-handed baker 2, ...
- Right-handed baker 0, Right-handed baker 1, Right-handed baker 2, ...
- Cautious baker 0, Cautious baker 1, Cautious baker 2, ...

All bakers must be able to bake, and they cannot use the same oven mitts at the same time. In other words, the shared resources are the oven mitts and the bakers contend for them to do their job. Hint: this is similar to the dining philosophers problem.

Question #1 [10/10]

Implement a program called bakery.c that takes four integer command-line arguments:

- The number of left-handed bakers
- The number of right-handed bakers
- The number of cautious bakers
- A seed for the random number generator. This is so that we control “randomness” and can reproduce runs.

Important guidelines:

- Your program MUST use locks and condition variables
- Your program should not do busy waits
- Your program should not contain global variables. E.g., you create your locks in main and then pass pointers to them as arguments to functions. Since you can only send 1 arg, you need to pack multiple arguments into a struct.

- Each baker (of any type) must perform its operations 10 times.
- **If you get stuck and are finding that you can't implement the program with the above guidelines, then include in your documentation that you broke one of the constraints above.**

Below is a sample execution of the program. You must make sure that the formats of your output messages, all printed to standard output (not standard error), matches the format below exactly, because it will be used as input to a test script like the last assignment.

```
./bakery
Usage: ./bakery <# left-handed bakers> <# right-handed bakers> <# cautious bakers> <seed>
./bakery 3 0 10 42
Invalid command-line arguments... Aborting
./bakery 2 4 3 42
[Left-handed baker 0] is working...
[Left-handed baker 1] is working...
[Right-handed baker 0] is working...
[Right-handed baker 2] is working...
[Right-handed baker 1] is working...
[Right-handed baker 3] is working...
[Cautious baker 0] is working...
[Cautious baker 1] is working...
[Cautious baker 2] is working...
[Left-handed baker 0] wants a left-handed mitt...
[Left-handed baker 0] has got a left-handed mitt...
[Left-handed baker 0] has put cookies in the oven and is waiting...
[Right-handed baker 3] wants a right-handed mitt...
[Right-handed baker 3] has got a right-handed mitt...
[Right-handed baker 3] has put cookies in the oven and is waiting...
[Right-handed baker 2] wants a right-handed mitt...
[Right-handed baker 2] has got a right-handed mitt...
[Right-handed baker 2] has put cookies in the oven and is waiting...
[Right-handed baker 1] wants a right-handed mitt...
[Right-handed baker 1] has got a right-handed mitt...
[Right-handed baker 1] has put cookies in the oven and is waiting...
[Cautious baker 1] wants a left-handed mitt...
[Cautious baker 1] has got a left-handed mitt...
[Cautious baker 1] wants a right-handed mitt...
[Left-handed baker 1] wants a left-handed mitt...
[Left-handed baker 1] has got a left-handed mitt...

...

[Cautious baker 2] has put back a right-handed mitt...
[Right-handed baker 0] wants a right-handed mitt...
[Right-handed baker 0] has got a right-handed mitt...
[Right-handed baker 0] has put cookies in the oven and is waiting...
[Cautious baker 1] wants a left-handed mitt...
[Cautious baker 1] has got a left-handed mitt...
[Cautious baker 1] wants a right-handed mitt...
[Right-handed baker 2] has taken cookies out of the oven...
[Right-handed baker 2] has put back a right-handed mitt...
```

```
[Cautious baker 1] has got a right-handed mitt...
[Cautious baker 1] has put cookies in the oven and is waiting...
[Cautious baker 0] has taken cookies out of the oven...
[Cautious baker 0] has put back a left-handed mitt...
[Cautious baker 0] has put back a right-handed mitt...
[Cautious baker 1] has taken cookies out of the oven...
[Cautious baker 1] has put back a left-handed mitt...
[Cautious baker 1] has put back a right-handed mitt...
[Right-handed baker 0] has taken cookies out of the oven...
[Right-handed baker 0] has put back a right-handed mitt...
```

Testing: To check the correctness of your program I provide you with a Python script (Python 2): `check_bakery.py`. This script checks the bakery output for bugs. Each You can run it as follows:

```
gcc bakery.c -o bakery
./bakery 5 4 3 128 | python check_bakery.py
Checking the the output is well-formatted...
Detected 5 left-handed bakers
Detected 4 right-handed bakers
Detected 3 cautious bakers
Checking that every baker does its required number of operations...
Every thread does what it needs to do 10 times.
Checking that no more mitts are taken than there are available...
No more mitts are used than are available.
Checking that bakers are able to bake at the same time...
Bakers can bake concurrently.
NO ERRORS DETECTED
```

Submission: I provide you with `bakery_starter.c`. This starter code is to use as a starting point. Download it, rename it `bakery_NAME.c`, implement the bakery simulation, and turn in an archive with this file.

Question #2 [1 Bonus Point]

Augment your code so that each baker thread keeps track of how long it has spent waiting for oven mitts, in seconds. Here is code fragment that shows you how to measure time:

```
#include <sys/time.h>

struct timeval before, after;
gettimeofday(&before,NULL);

.... // code to time

gettimeofday(&after,NULL);
double elapsed_time = ( 1000000.0*(after.tv_sec - before.tv_sec) +
                        (after.tv_usec - before.tv_usec))/1000000.0;
fprintf(stdout,"Elapsed time: %.3lf seconds\n",elapsed_time);
```

Before terminating, each thread should print its total waiting time. All time-related output should be printed to **standard error (not standard output)**. Here is a (filtered) sample output:

```
./bakery 4 5 3 42 | grep WAIT
[Right-handed baker 0] WAIT-TIME = 1.3950...
[Right-handed baker 4] WAIT-TIME = 1.5421...
[Left-handed baker 0] WAIT-TIME = 1.6960...
[Right-handed baker 1] WAIT-TIME = 1.5728...
[Right-handed baker 3] WAIT-TIME = 1.6163...
[Left-handed baker 3] WAIT-TIME = 1.8637...
[Right-handed baker 2] WAIT-TIME = 2.1344...
[Cautious baker 1] WAIT-TIME = 2.5780...
[Left-handed baker 2] WAIT-TIME = 2.2367...
[Cautious baker 0] WAIT-TIME = 2.4980...
[Left-handed baker 1] WAIT-TIME = 2.2593...
[Cautious baker 2] WAIT-TIME = 2.4725...
```

Run your code with 5 threads of each type, 10 threads of each type, and 50 threads of each type (use seed 42 as above). Report in the pdf you turn in the average wait times for each thread type.

Are left-handed bakers and right-handed bakers treated fairly? Why do you think that is? Include your answer and brief discussion in the pdf.