COP4600.001C11: Operating Systems
Project Two: Synchronization

June 6, 2011

1 Instructions

Obtain a copy of *The Little Book of Semaphores* and read about other advanced synchronization objects like turnstiles, barriers, and lightswitches, each of which can be built using semaphores and mutexes. Study them, understand them, and consider implementing them in C, because a future project may require them. This one does not.

Write a program that takes a single non-zero integer argument, n, from the command line. As always, check to make sure the correct number of correct arguments are passed on the command line and exit gracefully if the commmand line usage is not correct. The program must create four threads, each of which needs to access and modify a shared integer variable, call it x, initialized to n. Each thread performs a single arithmetic operation on x. Use semaphores and/or mutexes to enforce the following requirements.

1. Each thread must modify x one thousand (1000) times, and with each modification print only the updated value of x.
2. Thread 1 increments x, printing the new value on a newline.
3. Thread 2 decrements x, printing the new value on a newline, followed by a blank line.
4. Thread 3 doubles x, printing the new value on a newline.
5. Thread 4 halves x, printing the new value on a newline, followed by a blank line.
6. Thread 2 must perform its operation immediately after thread 1, and is the only thread which may immediately follow thread 1 (decrementation always follows incrementation).
7. Thread 4 must perform its operation immediately after thread 3, and is the only thread which may immediately follow thread 3 (halving always follows doubling).
8. After thread 2 finishes its operation, either thread 1 or thread 3 may proceed.
9. After thread 4 finishes its operation, either thread 1 or thread 3 may proceed.
10. The first thread to execute must be either thread 1 or thread 3, and your code may not explicitly specify which.
11. No thread may poll the value of x either before or after modifying it.
12. Other than these requirements you may neither assume nor enforce any scheduling of the threads.

After all four threads finish execution, your program should print the final value of x. Note that if you provide a correct solution, the following properties will hold:

- x still will equal n after the four thousand operations.
- x will always remain an integer, despite the halving operation.
- Except in the case $n = -1$, x will never change sign.

2 Conventions

Your submission will receive severe penalties if it does not adhere to the following conventions.

2.1 Required Files

Please submit a zip file named `lastname-firstname-proj2.zip` (all letters lowercase!) containing the following files.

```
Makefile
proj2.c
```

2.2 Requirements of the makefile

Your makefile should support the following commands when run from the directory containing your source files:

```
$ make all
$ make clean
```

The `all` command should produce your executable file named `proj2`. The `clean` command should remove all temporary and compiled files, leaving only the source files required of you for this project.

2.3 Output Format

Please see the following sample output to see how to structure your printf() format strings. Test commands similar to these will be run after executing `make all`. If the user does not supply your program with a valid argument, simply print `incorrect arguments`, and quit.

```
$ make all
$ ./proj2 3
  6
  3

  6
  3

  4
  3

  6
  3

  ...

  4
  3

  final value: 3

$ ./proj2 -101
  -100
  -101

  -202
  -101

  ...

  final value: -101
```

```
$ ./proj2 0
  incorrect arguments

$ ./proj2 "derp"
  incorrect arguments

$ ./proj2 5 19
  incorrect arguments
```