# CPSC 416: Distributed Systems, 2013 W2

Assignment #2, Due Feb 24, 2014 (midnight)

Change log:

Feb. 5, 2014: corrected `generator_state_t *s` use inside function.
Feb. 6, 2014: corrected `load_generator()` function signature.
Feb. 21, 2014: corrected typo on `load_generator()` return type.

1. Write Peterson's mutual-exclusion algorithm in LTSA.

http://en.wikipedia.org/wiki/Peterson%27s_algorithm

2. Concurrency Experiment

Write program to benchmark the relationship between the number of processes and the access time to data stored in a shared queue. Input to the program should be the number of processes to create and runtime of the program.

```
./assign2 -p <1-100> -t <1-300>
```

The parameters are "-p" for the number of processes, which can be in range {1-100}, and " -t" for the time to execute, which can be in the range {1-300} seconds. For example

```
./assign2 -p 5 -t 30
```

runs the program with 5 processes for 30 seconds.

Use **Pthreads** to create processes that enqueue and dequeue positive integer value to/from a queue object. You will be provided a function that has the following signature and provides the type of operation to perform on that queue. You are to implement the queue and then use this function to generate a workload to test the operation of the queue.

```
typedef struct queue_operation
{
    int operation;  // 1 to insert an item, 0 to remove
    int value;  // value should be ignored if operation=0
} operation_t;

typedef struct generator_state {
{
    int state[4];  // for internal use
} generator_state_t;
```

```
// Example
int load_generator(operation_t *op, generator_state_t **s)
{
   if ( *s == NULL )
      {
      *s = (generator_state_t *) malloc(sizeof(generator_state_t));
      // initialize state
      }
   // set value for op->operation and op->value
   return 1; // 1 success -- return 0 for failure
}

// use the function as follows

operation_t myops;
generator_state_t *state = NULL;
while ( load_generator(&myops, &state) ) { do stuff };
```

Return value of the function is integer, 1 for success and 0 for failure.

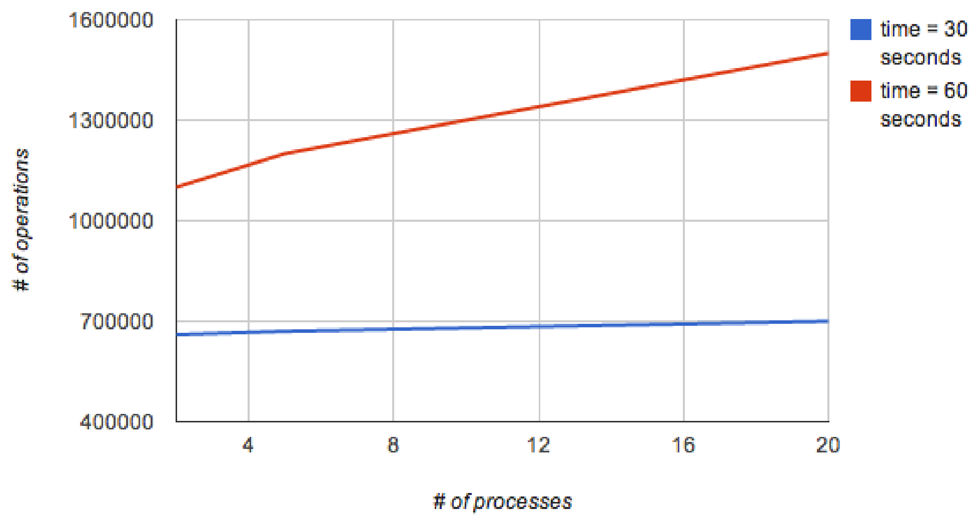
- *operation = 1* indicates enqueue *value* into the shared queue;
- *operation = 0* indicates dequeue item from the shared queue (*value* can be ignored);

Enqueue/Dequeue items to/from queue is, as expected, *first in first out* (FIFO) order. Your program should output “result.txt” in following format.

```
# of threads=2 time (seconds)=30 total number of operations=660000
# of threads=5 time (seconds)=30 total number of operations=670000
# of threads=10 time (seconds)=30 total number of operations=680000
# of threads=15 time (seconds)=30 total number of operations=690000
# of threads=20 time (seconds)=30 total number of operations=700000

# of threads=2 time (seconds)=60 total number of operations=1100000
# of threads=5 time (seconds)=60 total number of operations=1200000
# of threads=10 time (seconds)=60 total number of operations=1300000
# of threads=15 time (seconds)=60  total number of operations=1400000
# of threads=20 time (seconds)=60  total number of operations=1500000
```

Based on these evaluations write a benchmarking report that should include a similar graph based on your data.



The report should be as most 1 page (letter format, font Arial, size 11) to explain your observations. Include answers to the following two questions.

1. Briefly describe the access mechanism used to protect the shared queue from concurrent access by the different threads. What trade-offs did you consider in deciding on that mechanism to ensure the correct operation of the queue.

2. How does the number of operations (enqueue and dequeue) depend on the number of threads you created? Could you double the number of operations performed (for constant runtime) by doubling number of threads? If not, why does it behave in that manner?

Feel free to include other observations you thought to be interesting.

**Deliverables:**

The following file should be gzipped and submitted as a single file by the handin command.

1. FSP source code for Peterson's algorithm (as **peterson_mutex.lts**).
2. Source code for benchmark study (as **assign2.c**) and **Makefile** for compile.
3. Benchmark report file (as **report.pdf**).

Please make sure to follow the naming conventions for the files and parameters for the program assign2.c.

**Good luck!**