Sean Hinchee

Section G

3/1/18

Lab 4

1.2

```
% ./a.out
After both threads are done executing, `shared` = 1213533
```

Expected: 2,000,000

Calculated: 1,213,533

The discrepancy is due to there not being a lock on shared and thus, the two threads running over each other on reading/writing values to 'shared.'

1.2.1

```
% ./a.out
After both threads are done executing, `shared` = 2000000
```

Yes, the mutex now guarantees that both threads can cleanly could to 2,000,000 as they will both alternate cleanly all the way through their allocated data ranges.

Optimally, the code should probably be a counter to one million rather than two million.

1.3.1

There will need to be a signal to show that the producer has relented and C1 can lock in. Then, there will need to be a signal to show that C1 has relented and C2 can lock in. Then, there will need to be a signal to show that C2 has relented and P can produce again.

There are n conditions where n is the number of actors.

1.3.2

While locking is being done on the semaphore, this does not change the fact that the changes can occur out of order due to the nature of scheduling on the processor. As a result, the threads can operate safely, but out of order.

2.x

All relevant changes were constrained to the main.c file as to improve backwards compatibility.

Notable changes:

- #include sys/time.h and errno.h
- Addition of pthread_cond_t cv to print_job_list
- Addition of the -l, -t flags (logging and consumer timeout respectively)
- Addition of cond/mutex destruction
- Queue usage for print_job_list
- Mutex/semaphore usage for print_job_list for thread safety
- Thread joining for consumers and producers

In general, the problem of announcing to consumers to consume was solved by the addition of an event to the print_job_list structure and the usage of pthread_cond_signal call rather than the broadcast equivalent so that only one consumer was consuming at a time, but multiple consumers could still be used in sync (if load demanded this). Additionally, the exit problem where consumers would perpetually block was solved by using the pthread_cond_timedwait function with a timeout of 100ms (configurable via the -t flag) at which point the control loop cycles (and thus checks if the exit flag is set).

Note: If the log flag (-l filename) flag is not set, logging will be performed on stdout by default. It is recommended to run main with a log file specified. To avoid file production or printing, it is recommended to redirect stdout to /dev/null: `./test3.sh | ./main > /dev/null`