CS 333: Operating Systems Lab, Autumn 2018

15th October 2018

Lab Quiz #3, 30 marks

1. **semaphores with xv6** (15 marks)

   The following kernel functions are provided as part of xv6 kernel implementation, and need to be used in the corresponding system call implementations.

   - `spinlock_acquire(spinlock *s)` A kernel function to acquire a spinlock.
   - `spinlock_release(spinlock *s)` A kernel function to release a spinlock.
   - `cv_sleep(void *variable, spinlock *s)` A kernel function to put a process to sleep on a condition variable.
   - Implement the semaphore functionality by implementing the following systems calls:

     `sem_init(index, value)` ... initializes the i^{th} semaphore to `value` and initialize rest of semaphore struct/object.
     `sem_up(index)` ... increments value of semaphore and issues wake up.
     `sem_down(index)` ... if possible, decrements value of semaphore else sleeps.

     Print index and value of sempahore, and pid of process when value of semaphore is updated. The semaphore objects are stored in the kernel and accessed via the the above system calls from user-processes. Using a structure to represent the semaphore would be a good idea, the structure can hold the value, required locks, variables etc. Note that the semaphore value is never negative.

     Look up test programs to test semaphore implementation.
   - Using the semaphore implementation build a producer-consumer application. Implement the `producer` and `consumer` functions in the template code provided in the file `prodcon.c`.
     The program takes as input the maximum size of the buffer (for production), number of producers, number of items produced by each producer, number of consumers, and number of item consumed per consumer.
     e.g., `./prodcon 10 5 4 4 5` ... 5 producers producing 4 items each, and 4 consumers consuming 5 items each.
   - Look up the `README` file for usage details.

2. **just a barrier away!** (15 marks)

   - This part of the quiz requires you to use the *pthreads* library.
   - Implement the *barrier* functionality using the following functions.
     - `barrier_init(struct barrier*, int nthreads)` initializes the barrier structure variable, `nthreads` is number of threads on which the barrier should hold.
     - `barrier_wait(struct barrier*)` wait on barrier if all n threads have not reached this point of execution.
       support of the `pthreads` library.
     - Implementation files: `barrier.c` and `barrier.h`
     - Test cases in: `barrier1.c`, `barrier2.c` ...
     - Look up the `README` file for usage details and also the output directory for sample outputs.

Refer to the `README` file for more details. Check the tar file for sample testcases and outputs.

Deadline: 15^{th} October 5.00 p.m.
Submission via moodle.

Submission Guidelines

- All submissions via moodle. Name your submission as: **<rollno-labquiz3>.tar.gz**
- Execute the following command in the xv6 directory to clean the source code.
  `make clean`
- Use the following command to create the submission archive.
  `tar -cvzf ~/rollno-labquiz3.tar.gz ~/rollno-labquiz3`
- The tar should contain the following files in the following directory structure:

  <rollno-labquiz3>/
  |__part1
  |____xv6/
  |_____<the xv6 directory with all files>
  |____sample-outputs/
  |_____<sample outputs>
  |__part2/
  |____barrier.h
  |____barrier.c
  |____barrier-testcase<1,2,3>.c

- Verify submission archive exists in the home directory.
- Upload on moodle.