Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Classes: M 16:40-18:30, W 16:40-17:20, Labs: W 17:25-18:55, all @ FAB 88-10.

Lab 2: Shared-Memory Programming with Pthreads

1 Condition Variables (`condvar-pthd.c`)

The file `condvar-pthd.c` contains an incomplete Pthread program. The `main()` routine creates two threads, one sender and one receiver. When executed, the sender sends out a signal and the receiver waits and receives the signal.

Exercise

1. Complete the program by adding the missing signal and wait code in the two routines, `sender()` and `receiver()`. Compile and run the program.
2. Switch the order of the following two statements in `main()`:

   ```
   pthread_create(&tid2, NULL, (void *)receiver, NULL);
   pthread_create(&tid1, NULL, (void *)sender, NULL);
   ```

   Compile and run the modified program. What happens? Can you explain?
3. Add a second receiver to the program by creating a third thread in `main()` to run the `receiver()` routine:

   ```
   pthread_create(&tid3, NULL, (void *)receiver, NULL);
   ...
   pthread_join(tid3, NULL);
   ```

   Don't change the `sender()` and `receiver()` routines themselves. Compile and run the modified program. What happens? If it doesn't work, fine a simple way to fix it.

2 Barrier Synchronization (`barrier-pthd.c`)

Read and understand the program in file `barrier-pthd.c`.

Exercise

1. What do you expect the program to print? Compile and run the program. Does it produce the expected result? Can you explain the program's output?
2. Insert barrier synchronization into the program, so that each of the statements in the `worker()` routine is guaranteed to have completed across all worker threads before the next statement starts. Compile and run the program. Does the output meets your expectation?
3. Replace the `pthread_join()` loop with a barrier synchronization. Compile and run the modified program.

3 Producer and Consumer Problem (`prodcons-pthd.c`)

The producer-consumer problem is a classic example of multi-threaded synchronization problem. The base version of the problem describes two threads, the producer and the consumer, who share a common fixed-size buffer. The producer repeatedly adds items to the buffer, and concurrently, the consumer repeatedly removes items from the buffer. The producer blocks and waits if the buffer is full and the consumer blocks and waits if the buffer is empty.

The file `prodcons-pthd.c` contains a naive implementation of this problem. The buffer is implemented as an array, and there is no waiting and blocking.

Exercise

1. Compile this program, and run it multiple times. Do you observe any problems? Try to explain the results.
2. Now insert a simple "busy-waiting" synchronization into the `producer()` routine:

   ```
   while (idx == BUFSIZE)
       ;  // busy waiting
   ```

   and a similar one into the `consumer()` routine:

   ```
   while (idx == 0)
       ;  // busy waiting
   ```

   Compile and run the modified program. Does the program work now? Do you observe any instance where the array index value is out of the buffer bounds?
3. Replace the busy-waiting synchronization with signal-and-wait synchronization using condition variables.
4. (**Extra**) If you still have time left, you may try to convert this program into a C++ or a Java program using their thread library.

4 Submission

Write a short report, in plain text or pdf, summarize your experience with this lab. Zip your write-up with the last modified version of program from each section into a single zip file, and submit it through the "Lab2" submission folder on D2L (under the "Activities/Assignments" tab). You should submit your work before the week-end, *i.e.* Sunday 1/19.