CSE 2431 **LAB 2** **AUTUMN 2015**

Electronic Submission: 11:59 pm Wednesday, Oct 28th

1. Goal

1) Learn how to create and execute threads using the pthread library.
2) Learn how to use synchronization mechanisms
3) Learn how to test multithreaded code

2. Introduction

In this project, we will implement a bounded buffer (see Chapter 6), and use it to solve the producer/consumer problem. I suggest you to implement it in three steps:

1) Implement a bounded buffer correctly, without considering synchronization
2) Add synchronization to protect the buffer
3) Create producer and consumer threads to test your buffer

3. Implement a non-synchronized bounded buffer

The following two functions are core to the bounded buffer:
Push(…): it adds an item to the tail of the buffer, and if the buffer is already full, print "error".
Pop(): it returns the head of the buffer, and if the buffer is empty, print "error".

Apart from them, you will also need to implement a initilization function and a destroy function. You can either use the circular buffer approach, which is taught in Chapter 6, or use a linked list approach to implement the above functions.

You should thoroughly test your code before you move to the next step.

4. Add synchronization

You should add synchronization code to protect your bounded buffer when multiple threads are accessing it concurrently.
Push(…): it adds an item to the tail of the buffer, and if the buffer is already full, it waits until the buffer is not full.
Pop(): it returns the head of the buffer, and if the buffer is empty, it waits until the buffer is not empty.

You can use either the semaphore approach, which is taught in Chapter 6, or use the conditional variable approach, which I taught at end of Chapter 6, to solve this problem.

I have already shown example codes about mutex and conditional variables. Here is an example for semaphore:

The pthread package provides two types of semaphores - named and unnamed. For this project, we use unnamed semaphores. The code below illustrates how a semaphore is created:

```
#include <semaphore.h>
sem_t sem;

/* create the semaphore and initialize it to 7*/
sem_init(&sem, 0, 7);
```

The *sem_init*(…) creates a semaphore and initializes it. This function is passed three parameters:

1. A pointer to the semaphore
2. A flag indicating the level of sharing
3. The semaphore's initial value

In this example, by passing the flag 0, we are indicating that this semaphore can only be shared by threads belonging to the same process that created the semaphore. A nonzero value would allow other processes to access the semaphore as well. In this example, we initialize the semaphore to the value 7.

For the semaphore operations wait (or down, P) and signal (or up, V) discussed in class, the pthread package names them sem_wait(…) and sem_post(…), respectively. The code example below creates a binary semaphore mutex with an initial value of 1 and illustrates its use in protecting a critical section: (Note: The code below is only for illustration purposes. Do not use this binary semaphore for protecting critical section. Instead, you are required to use the mutex locks provided by the pthread package for protecting critical section.)

```
#include <semaphore.h>
sem_t sem_mutex;

/* Create the semaphore */
sem_init(&sem_mutex, 0, 1);
/* Acquire the semaphore */
sem_wait(&sem_mutex);

/*** critical section ***/

/* Release the semaphore */
sem_post(&mutex);
```

5. Create producer/consumer threads

You should create producer and consumer threads to test your buffer. I have shown you examples about how to create threads.

Your producer thread should generate a number of messages and "push" them into a bounded

buffer one by one.
Your consumer thread should “pop” from the buffer continuously and print the message.
You should check the followings:

1) If there are only producers, do they behave as you expected?
2) If there are only consumers, do they behave as you expected?
3) If there are both producers and consumers, have all “produced” messages been “consumed”?
4) Is any “produced” message “consumed” more than once?

6. Compilation

I have provided a Makefile.
If you need to use semaphores, you also need to add the library –lrt to the gcc command in Makefile.

7. Submission

Please make a lab2 directory and submit all file for that lab using the **submit** command. Please include a README file for the grader with your full name and email address. Include any instructions needed to compile and run your program. Include your own tests. We will add our tests.

A late penalty of 10% **per hour** will be assessed, for instance, if a lab is submitted at 12:00 am or 12:59 am there is a 10% penalty. Labs must be the student’s own work.