Lab 4

1. Reference Code: lab3.c

2. Lab Practices

Practice ***spinlock***, ***mutex*** and binary/count ***semaphores***.

Related APIs

pthread_mutex_t mutex;

- **pthread_mutex_init(&mutex, NULL);**
  // 2nd argument: const pthread_mutexattr_t *attr
- **pthread_mutex_lock(&mutex);**
- **pthread_mutex_unlock(&mutex);**

#include <semaphore.h>
sem_t sem;

- **sem_init(&sem, 0, 1);**
  // 2nd argument: 0 means shared between threads, non-zero means shared between processes (requires shared memory) // 3rd argument: initial value
- **sem_wait(&sem);**
- **sem_post(&sem);**

3. Lab Assignment: Submission date on the moodle (**Mar. 2, Mon. 11 am**),
Online Grading (28/2 and 2/3, 1 pm during lecture sessions) or Offline grading using the submitted files.

- In the main thread: create two threads (one is **odd_thread** the other is **even_thread**).
  - But, explicitly **odd thread MUST** be created firstly (e.g., give 1 second sleep between the creation of the two threads).

- The even_thread: prints 0 2 4 6 8 10 and the odd_thread: prints [1] [3] [5] [7] [9]
  - But, the sequential synchronization **MUST** be done with the critical section problem in the shared datum by adding one and print (i.e., ***num*** global variable with **the initialized value 0** (int num = 0)) increase
  - Constraints: **MUST** use **mutex(es)** and **semaphore(s)**.
  - **Assumption**: For convenience, create and test only 10 threads, (but **MUST** be scalable up to any N threads).

- You **MUST** implement the synchronization techniques **inside the created threads** (not the main thread) except initialization.

- You **MUST** follow the ***pthread_join()*** calls, after executing all the ***pthread_create()***.

- Lastly, the expected **Output** could be like below:

Output : (might depend on the machine, but the BLUE sequential order MUST be the same.)
jurn@ZBookG4:~/work/cs332/thread$ **./oddeven**
0 [1] 2 [3] 4 [5] 6 [7] 8 [9] 10