# Lab 3

**1. Starting Code: The lab practice, lab2_ex.c**

Add the ‘sum’ operation into the created thread, and make it as a critical section

```
printf("thread_arg = %d was created\t", arg);
int sum = 0;
for(int i=0; i<=arg; i++)
        sum += i;
printf("sum = %d\n", sum);
//printf("and sleeping for 60 secs\n");
```

**Output**:

```
jurn@ZBookG4:~/work/cs332/thread$ thread number = 10
thread_arg = 0 was created      sum = 0   //(MUST be atomic operation, and correct results)
thread_arg = 1 was created      sum = 1   //(0+1)
thread_arg = 3 was created      sum = 6   //(0+1+2+3)
thread_arg = 2 was created      sum = 3   //(0+1+2)
thread_arg = 4 was created      sum = 10  ...
thread_arg = 5 was created      sum = 15
thread_arg = 6 was created      sum = 21
thread_arg = 7 was created      sum = 28
thread_arg = 9 was created      sum = 45
thread_arg = 8 was created      sum = 36
```

**2. Lab Practices**

Instead of the ***spinlock***, change it to ***mutex***.

**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 (**26/2, Wed. 11 am**),
Online Grading (26/2, 1 pm  during lecture session) or Offline grading using the submitted files.

- Using mutex(s) and semaphore(s), **synchronize  the order of the threads** using **the mutual exclusion problem fixing (atomic)**  (i.e., according to the ***thread_arg***, the threads **MUST be executed/printed in order (sequentially) using the corresponding format** (i.e., **"**thread_arg = 0 was created                    sum = 0").

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

```
for(int i=0; i<num_th; i++){
        pthread_create(...);
        ...
}
for(int i=0; i<num_th; i++){
        pthread_join(...);
        ...
}
```

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

**Output : (some results depend on the machine, but the BLUE sequential order MUST be the same.)**

```
jurn@ZBookG4:~/work/cs332/thread$ ./lab3.out 10 &
jurn@ZBookG4:~/work/cs332/thread$ thread number = 10
thread_arg = 0 was created    sum = 0
thread_arg = 1 was created    sum = 1
thread_arg = 2 was created    sum = 3
thread_arg = 3 was created    sum = 6
thread_arg = 4 was created    sum = 10
thread_arg = 5 was created    sum = 15
thread_arg = 6 was created    sum = 21
thread_arg = 7 was created    sum = 28
thread_arg = 8 was created    sum = 36
thread_arg = 9 was created    sum = 45
thread_arg 0 was completed  // (the below parts might be different from machines)
thread_arg 0  is joined well
thread_arg 1 was completed
thread_arg 2 was completed
thread_arg 3 was completed
thread_arg 4 was completed
thread_arg 1  is joined well
thread_arg 2  is joined well
thread_arg 5 was completed
thread_arg 3  is joined well
thread_arg 6 was completed
thread_arg 4  is joined well
thread_arg 5  is joined well
thread_arg 6  is joined well
thread_arg 7 was completed
thread_arg 7  is joined well
thread_arg 8 was completed
thread_arg 8  is joined well
thread_arg 9 was completed
thread_arg 9  is joined well
the main thread exits
```