CS5243 Advanced UNIX Programming
Assignment 9 (4 pts)
Group 4

(1) Screenshot of the code

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

// Mutex
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;


void* new_pthread_barrier_wait(){

    // Lock
    pthread_mutex_lock(&mutex);

    // Critical Section
    printf("Thread %u running\n", (unsigned int)pthread_self());
    //

    // Unlock
    pthread_mutex_unlock(&mutex);

}

int main(void){

    int num_thread = 5;
    pthread_t tid[num_thread];
    void *tret;


    // Lock, Waiting for creating all threads
    pthread_mutex_lock(&mutex);
    for (int i = 0; i < num_thread; i++)
    {
        printf("Starting thread %d\n", i);
        if (pthread_create(&tid[i], NULL, new_pthread_barrier_wait, NULL) != 0)
```

```
36          {
37              printf("Starting thread %d error !\n", i);
38              return 0;
39          }
40      }
41
42      // Unlock
43      pthread_mutex_unlock(&mutex);
44
45
46      // Waiting for thread termination
47      for(int i = 0;i < num_thread;i++){
48          if(pthread_join(tid[i], &tret) != 0){
49              printf("Thead %d exit error !\n", i);
50              return 0;
51          }
52      }
53
54      return 0;
55  }
```

(2) Screenshot of the result

```
freebsd@generic:~/Advanced-UNIX-Programming_Student/assignment9 % ./assignment9
Starting thread 0
Starting thread 1
Starting thread 2
Starting thread 3
Starting thread 4
Thread 2219012864 running
Thread 2219007488 running
Thread 2219009280 running
Thread 2219011072 running
Thread 2219005696 running
```

(3) Explanation:

```c
// Mutex
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;


void* new_pthread_barrier_wait(){

    // Lock
    pthread_mutex_lock(&mutex);

    // Critical Section
    printf("Thread %u running\n", (unsigned int)pthread_self());
    //

    // Unlock
    pthread_mutex_unlock(&mutex);

}
```

To implement the pthread_barrier_wait function, we first create a global mutex
This mutex will block the execution in the critical section for 5 threads, and only allow one thread enter the critical section at the same time.

```c
// Lock, Waiting for creating all threads
pthread_mutex_lock(&mutex);
for (int i = 0; i < num_thread; i++)
{
    printf("Starting thread %d\n", i);
    if (pthread_create(&tid[i], NULL, new_pthread_barrier_wait, NULL) != 0)
    {
        printf("Starting thread %d error !\n", i);
        return 0;
    }
}

// Unlock
pthread_mutex_unlock(&mutex);
```

In the main function, the creation thread first locks the mutex, before creating threads. Hence, every thread will await the completion of the creation process for all threads, and then start to execute the routine function.