/*===THIS PAPER IS MEANT AS CODE EXPLANATION OF CODE 2 AND 3 OF LAB 2*/
//for more convenient assessment --> https://github.com/arroiyyaan/LLDM-Lab2

The peace joint of code below performs thread creation as well as repeating that creation by 100 times which is resulting 100 threads. After creation, the threads created then are activated and joint by pthread_join. Inside the loop of the thread creation below, a loop counter is also performed which counts from 0 to the 1000.

```
#define MAX_CORES 100

//SYNCHRONIZATION
  //thread creation
  pthread_t *thread_group = malloc(sizeof(pthread_t) * MAX_CORES);

  // ignite all threads
  for (i = 0; i < MAX_CORES; ++i) {
     pthread_create(&thread_group[i], NULL, start_counting, NULL);
  }

  // threads await
  for (i = 0; i < MAX_CORES; ++i) {
     pthread_join(thread_group[i], NULL);
  }
```

The below code is where the program starts making counter. As detailed version of counter process mentioned above. Basically, this code only counts from 0 to the any assigned number of COUNT_TO, in this case 1000. While sometimes, during the process, the counter program does not always run precisely. Initially when the complete code is run, the resulted number may exceed 1000 and will keep increasing. This is for sure away from what it was meant.

```
void *start_counting(void *arg) {
    for (;;) {
        if (i >= COUNT_TO) {
            return NULL;
        }
        ++i;
        printf("i = %lld\n", i);
    }
}
```

For that matter, the mutex for synchronization is necessary. This synchronization can be done by utilizing mutex by firstly assigning pthread_mutex_t as follows:

```
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

The below code is the code where mutex program related is added to the code above. The mutex is performed by running two functions which are lock and unlock. The mutex first will be locked just before the critical code of start_counting function. This is to ensure that the process of counter is initiated precisely and there should not be any random buffer withing the loop which will make a mess to the counter program. The mutex is then unlocked in the last phase of the counter loop. After

assigning mutex to the program, the resulted counter presicely gives number from 0 to 1000 and will be terminated in case the existance of random buffer or exceeded loop.

```
void *start_counting(void *arg) {
    for (;;) {
        // lock revealing
        pthread_mutex_lock(&mutex);
        if (i >= COUNT_TO) {
            pthread_mutex_unlock(&mutex);
            return NULL;
        }

        ++i;

        // lock release
        pthread_mutex_unlock(&mutex);

        printf("i = %lld\n", i);
    }
}
```