**Write-up for project 4 Operating system.**

**Author:** Chengjun Yuan cy3yb@virginia.edu

This program use mutex lock and conditional variable to synchronize the multi-threads in paralleling of merge sort. The global variables are listed below:

```
#define numThreads 2048
int array[numThreads*2];                    // array to be sorted.
int N=0;                                    // synchronization condition.
typedef struct
{
        int left;
        int right;
}parameters;
parameters *para[numThreads];               // parameters for each merge sort thread.
int arraySize=0;                            // the size of array to be sorted.
pthread_cond_t cv;                          // conditional variable.
pthread_mutex_t lock;                       // mutual exclusion lock to protect cv.
```

There are two major subroutines as shown below. First one is to read array data from "indata.txt" and count the number of data (saved in variable 'arraySize'). The second one is to merge sort for each single thread.

```
void readArray(char *fileName);
void *mergeSort(void *para);
```

## How to implement barrier?

In the end of subroutine "void *mergeSort(void *para)", there are four lines of codes as shown below:

```
pthread_mutex_lock(&lock);
N--;
if(N==0)pthread_cond_broadcast(&cv);        // If N==0, wake up the sleeping thread.
pthread_mutex_unlock(&lock);
```

We can see that the mutex lock is used to avoid race condition on variable 'N'. Every time one merge sort is finished, the value of N will be subtract 1. If N==0, which means that all threads have finished. Then the sleeping main thread will be waked up by "pthread_cond_broadcast(&cv)".

In the main routine, the codes used to wait completion of all threads are listed below:

```
pthread_mutex_lock(&lock);
if(N!=0){                                   // wait for broadcast when N==0.
        while(N!=0) pthread_cond_wait(&cv, &lock);
```

```
}
pthread_mutex_unlock(&lock);
```

If N does not equal 0, which means that there are at least one merge sort thread being running, then the main thread will be set to sleep and wait for the broadcast of conditional variable 'cv'.

## How to test the barrier?

In order to validate that my barrier implement works correctly, two printf debugging codes are inserted respectively to the subroutine "mergeSort" and main function.

```
printf("thread 1_ %d \n",arraySize/(right-left+1)); // in mergeSort
printf("main_thread %d \n",i);                      // in main function
```

For example, there are 32 numbers in indata.txt as the input file. Then the output is shown below:

```
arraySize = 32
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
thread 1 in 16
main_thread 16
thread 1 in 8
thread 1 in 8
thread 1 in 8
thread 1 in 8
thread 1 in 8
thread 1 in 8
thread 1 in 8
thread 1 in 8
main_thread 8
thread 1 in 4
thread 1 in 4
thread 1 in 4
thread 1 in 4
main_thread 4
thread 1 in 2
```

| | |
|---|---|
| | ***thread 1 in 2***<br>***main_thread 2***<br>***thread 1 in 1***<br>***main_thread 1***<br>***1 2 3 3 4 5 7 8 12 12 21 23 32 33 34 43 45 45 53 54 57 61 71 72 73 75 78 87 89 90 90 97*** |

We can see that the "main_thread" is printed totally after all threads are finished. If there are one "thread 1 in n" printed after the "main_thread n", then it means that my barrier is not implemented correctly.

**Attention: in my submitted version of codes, this test codes and output are not included.**

## Whether or not your main thread participates in comparisons?

Yes, the main thread will examine the value of N. If N==0, then all threads are finished, and the program can go to next level of multi-threads merge sort. If not, then sleep and wait for wake up by broadcast.

## How you organized (in memory) the intermediate results?

In my program codes, each merge sort thread change the order of numbers and save it in the **original array**. So there are **no intermediate results**.