4CS6456 (F2015) Operating Systems
Project 4: Concurrent Mergesort
Chenxi Guo

0.Notes:

(1) Since I use recursion method to deal with the merge sort, the memory needed is larger than 2048 MB, so you need to change the memory into 3064 to run the program. Specifically, in the VM VirtualBox, choose setting->system->memory size.
(2) The program runs perfectly in my VirtualBox, however, it fails sometime in my classmate's VirtualBox. Just in case,if there is some problem, you can contact me and I will show you the running process of my program.
(3) Thanks for reading!

1. how you implemented the barrier

I wrote a function named barrier_wait(struct barrier_t *barrier, int round) to realize barrier, the parameter round represent the which round that the current thread is in. There is a global Integer Array subs[] representing the number of threads in each round. the function barrier_wait() can get the number of the threads in current round though subs[round]. The barrier_wait() can minus this number can get the value of this number to decide whether the thread should wait or broadcast its siblings.

2.how you tested your barrier

For the case of (i) all integers are distinct; (ii) all integers are identical; (iii) all integers are already sorted in nondecreasing order; and (iv) all integers are already sorted in ascending order. I wrote the test case into the indata.txt. Then I can make sure that the program can handle the numbers of every orders.
The I wrote a random.cpp to random 4096 Integers between 0 and 4096 and put them into the indata.txt. In the mergesort.cpp, main function read all the number and save them in Integer Array a[]. The I'm sure that the program can handle at least 4096 integers.

3. whether or not your main thread participates in comparisons

No, the main function create the first thread and the first thread creates the threads in round 1, the threads in round 1 create the threads in round 2,…the the integers can be sorted and merged.

4. how you organized (in memory) the intermediate results

```
for (i = 0; i < (high-low+1) ; i++)
    a[low+i] = b[i];
```

I use this way to save the intermediate result into the original integer Array a[]. So that a[] is update in each merge, then we can get correct sorted a[] when all threads finish.