# Assignment2 Report

--Shaowei Su

## 1. Specification

1.1 Environment

All the tests are finished on CSUG cycle3 machine.

1.2 List of files

* Lock.cc
* Barrier.c
* hrtimer\_x86.c
* hrtimer\_x86.h
* Makefile
* atomic\_ops.h

1.3 Compile input

g++ lock.cc hrtimer\_x86.c -lpthread -o lock;

g++ barrier.c hrtimer\_x86.c -lpthread -o barrier

1.4 Input format

./lock –m1 –t1 –i1000

Where m followed by the index of lock mode, m:

* 1 : No sync;
* 2 : With pthread lock;
* 3 : With TAS lock;
* 4 : With TATAS lock;
* 5 : With backoff included in TATAS lock;
* 6 : With Ticked locks;
* 7 : With MCS locks;
* 8 : With FAI;

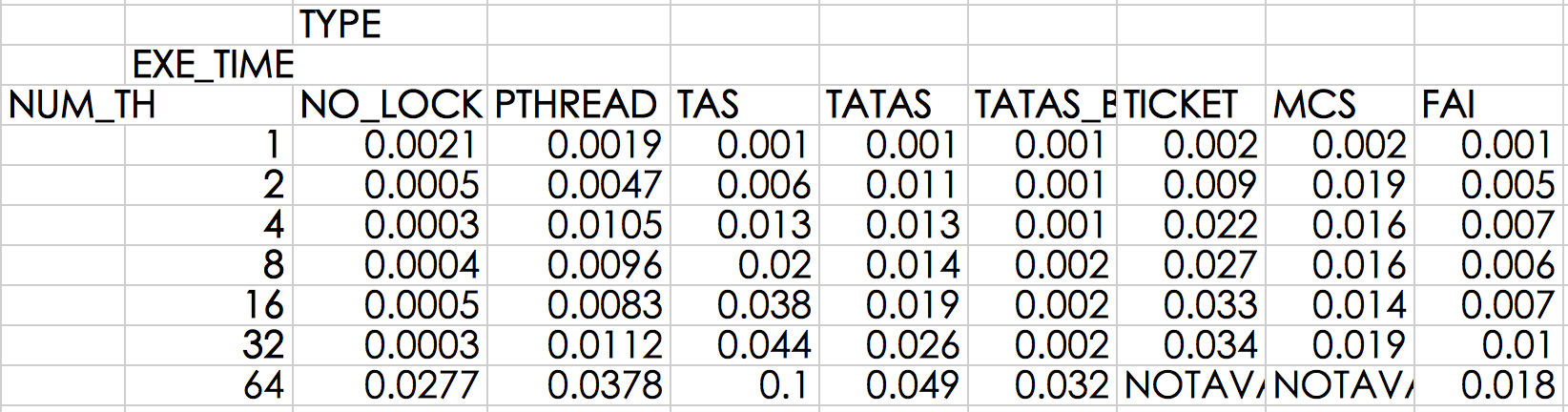
./barrier –m1 –t1 –i1000

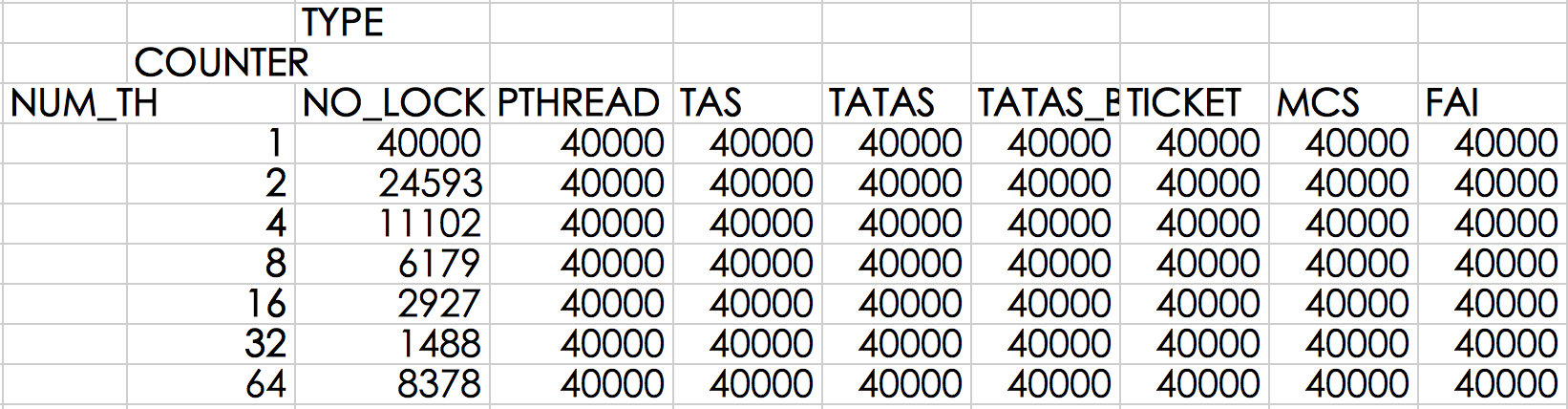
Where m followed by the index of the barrier mode, m:

* 1 : Pthread-based barrier;
* 2 : Centralized sense-reversing barrier;
* 3 : Tree-based barrier;

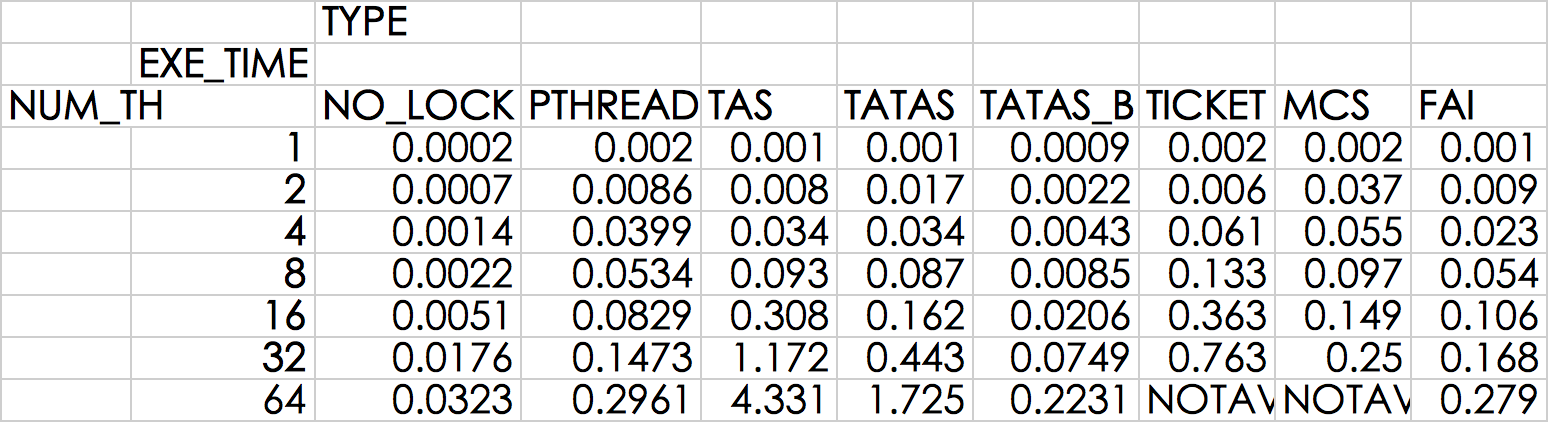
## 2. Test Results

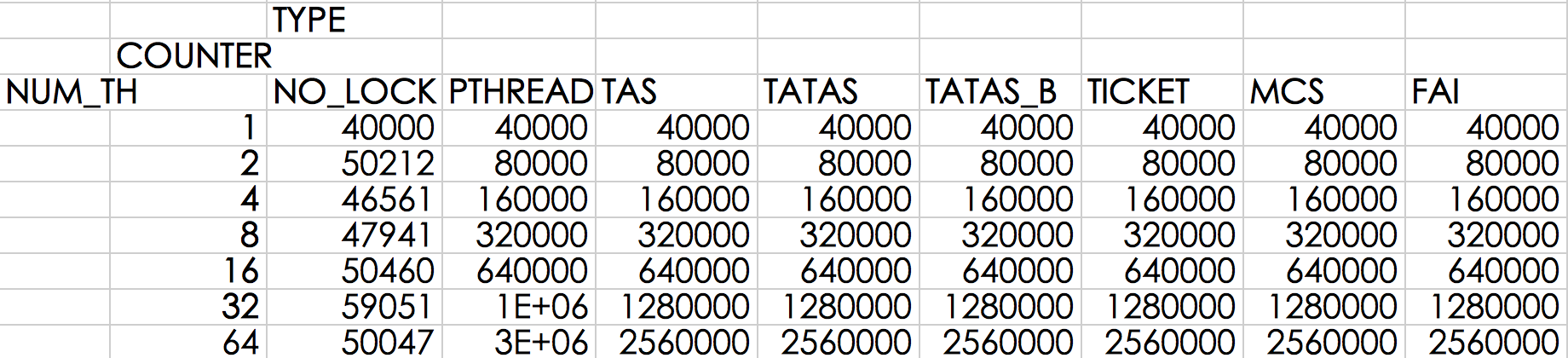
2.1 Fixed total count times as 40000(time:s)



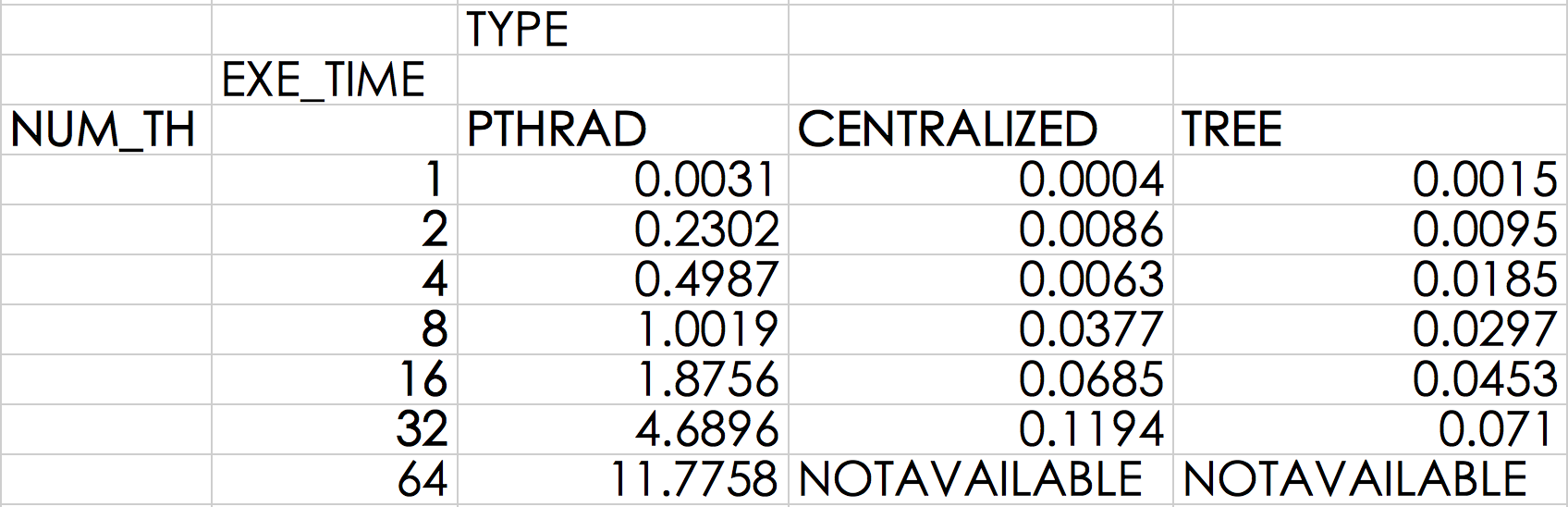


2.2 Fixed count times per thread as 40000(time:s)





2.3 Barriers(time:s)



## 3. Analysis

3.1 TAS locks

This is the easiest one to realize but it has some shortcomings: poor fairness and high latency;

3.2 TATAS locks

The benefit of TATAS locks is that spins happen at cache so that improves the scalability; while there is still a large expense for the traffic when many processors go to do the test&set at same time;

3.3 TATAS with backoffs

This lock turns out to be the best one in our experiment. The basic idea is that when one processor fails, it will delay for a while for the next attempt instead of retry immediately. In this way, it reduces the network traffic dramatically. However, exponential backoff may cause starvation for high contention locks;

3.4 Ticket locks

The guaranteed FIFO queue make ticket lock keep fairness and at the same time it reduce the traffic and latency, but it does not guarantee the traffic complexity; Since every thread reads one global variable “now serving”, it will increase the cost;

3.5 MCS locks

To deal with the issue with ticket locks, MCS lock read one local variable instead of a global one.

As a list-based lock, MCS lock also make sure the FIFO order and keeps low storage than the array-based locks, but he CAS atomic function is not easy to build; besides, MCS generally is the most scalable locks and keeps low latency;

3.6 Pthread-based condition synchronization

Apparently, this barrier suffers from the poor scalability. But it is quite easy to implement.

3.7 Centralized sense-reversing barrier

This barrier is quite scalable and simple, but produce high traffic. And its critical path is O(p).

3.8 Software combining tree barrier

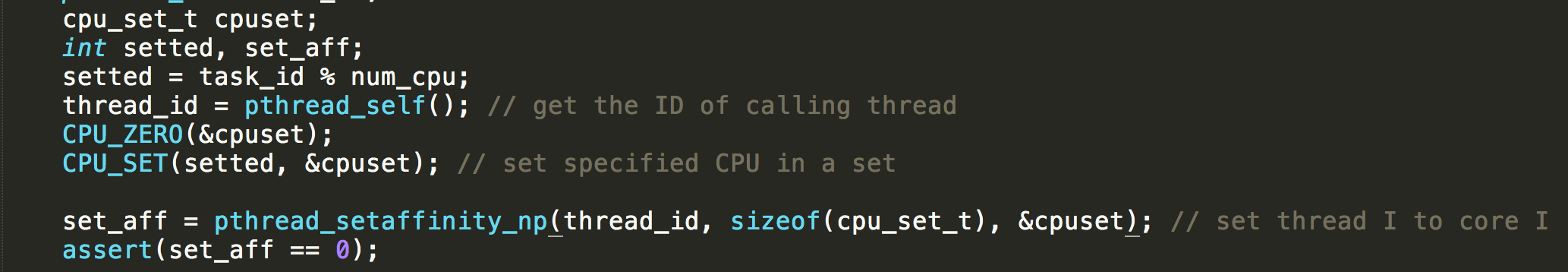
With critical path O(logp), and sense reversal to distinguish consecutive barriers, SCT is also good for scalability. On cache coherence machines, it may spin locally and otherwise on remote locations.

## 4. Major work

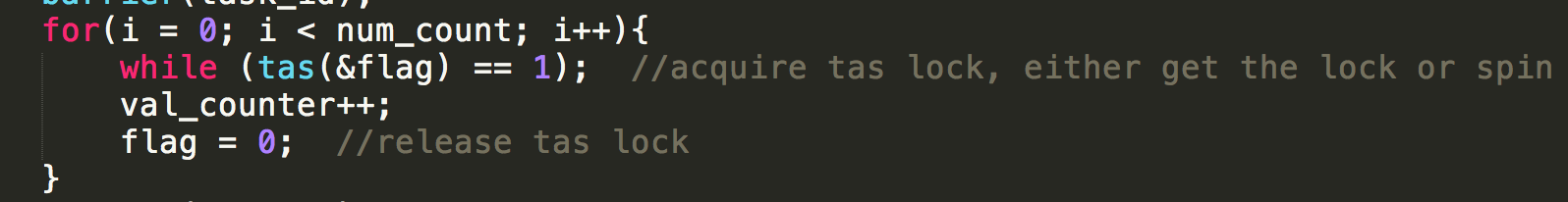
This part describes what I’ve done with effort.

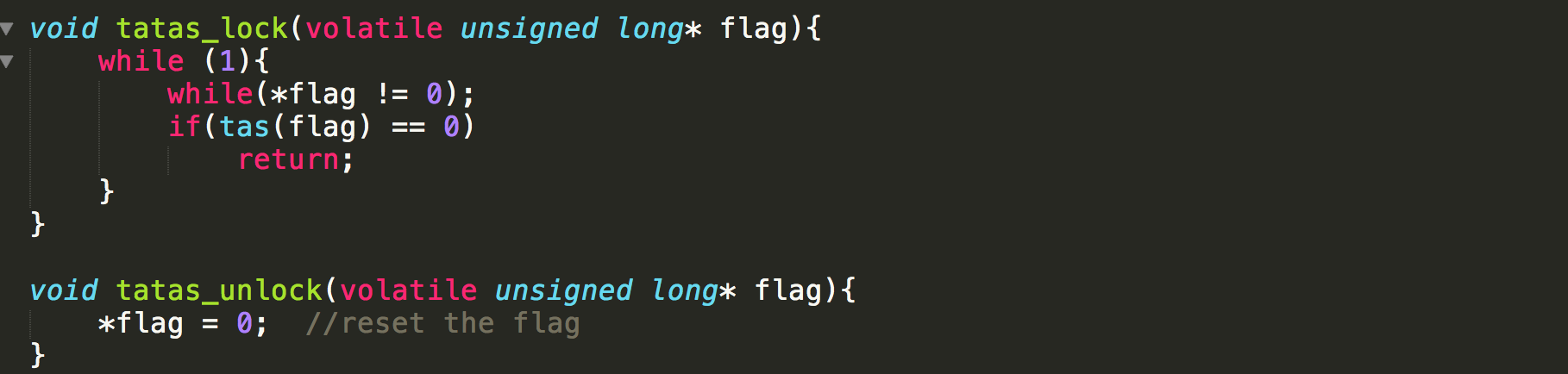
4.1 Set affinity

As required in the assignment, it is better to affix certain thread to one physical processor.



4.2 TAS&TATAS locks

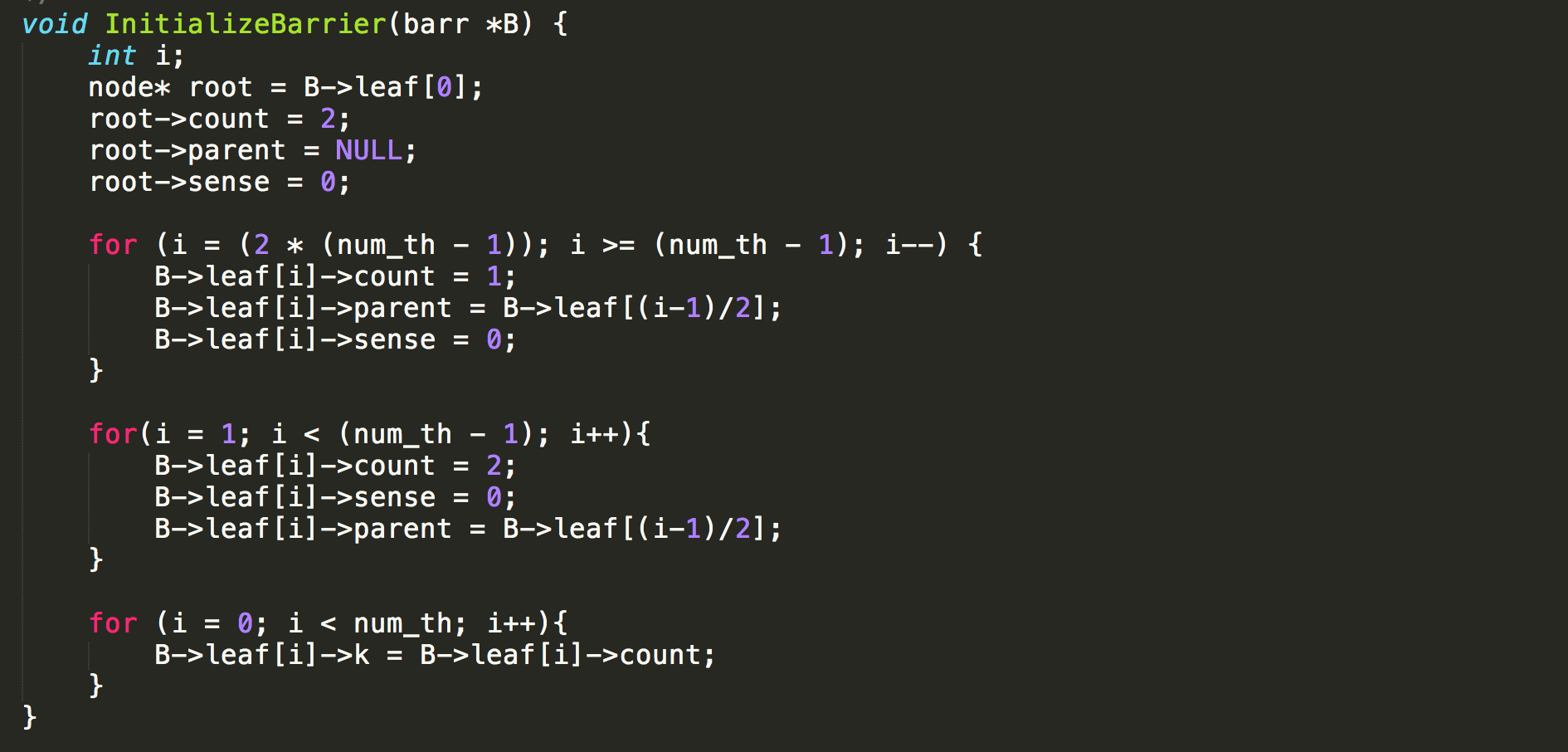




4.3 Combining tree barrier

For me this is the most challenging part in this assignment and so I’d like to explain the idea in detail.

(Discussed with my classmates, the initialization of barrier tree was actually inspired by Jingwei’s idea.)



Here the barrier tree is initialized with MAX\_THREAD as 128 nodes. Once we get the input number of threads, we are going to affix each thread to certain node to format the combining tree. The idea is that thread with id TID will be attached to tree node (numth – 1 + TID) so that every node before (num\_th – 1) will be count as 2; every node affixed with thread will be count as 1;

The rest of the combining tree strictly follows the pseudocode at: www.cs.rochester.edu/research/synchronization/pseudocode/ss.html.