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

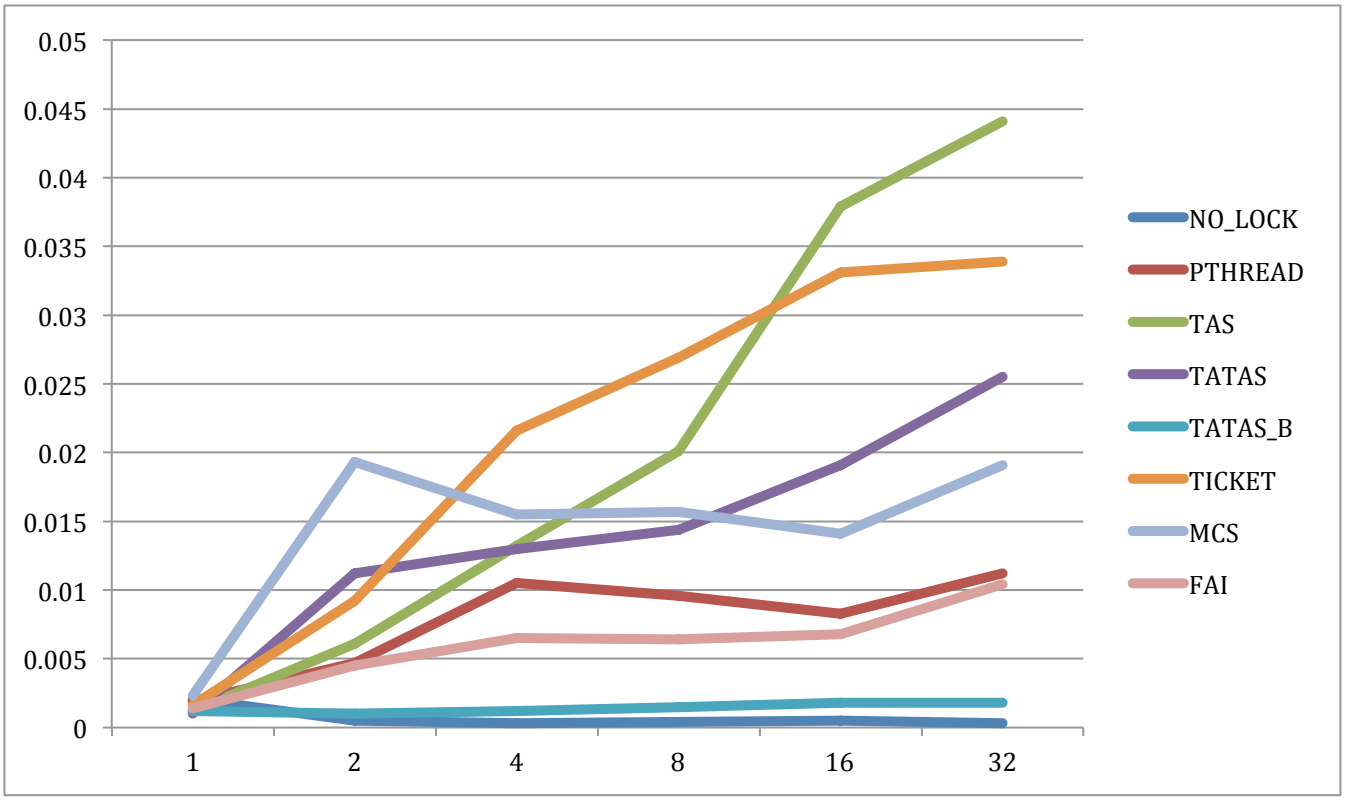
Where m followed by the index of the barrier mode, m:

- ✧ 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)

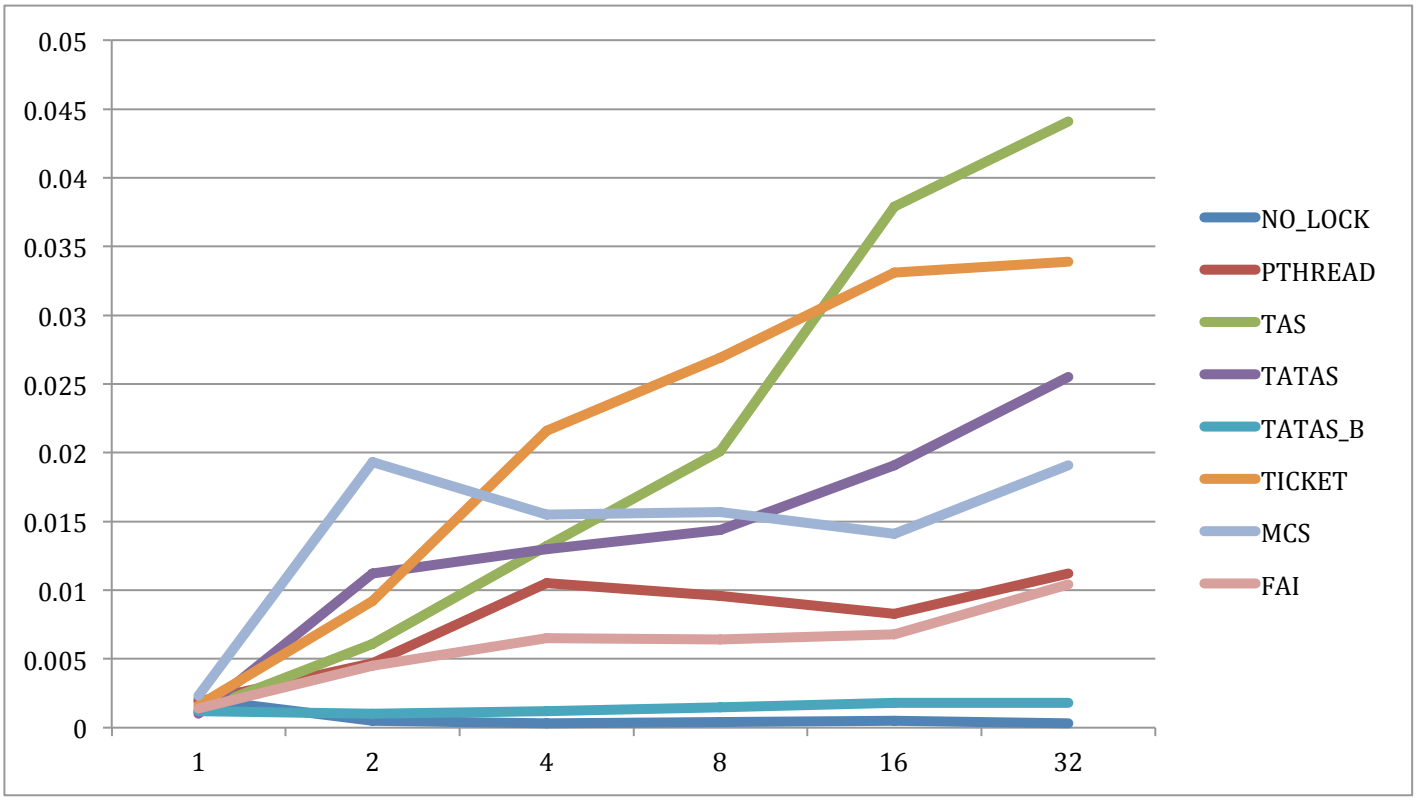
	TYPE							
EXE_TIME								
NUM_TH	NO_LOCK	PTHREAD	TAS	TATAS	TATAS_B	TICKET	MCS	FAI
1	0.0021	0.0019	0.001	0.001	0.001	0.002	0.002	0.001
2	0.0005	0.0047	0.006	0.011	0.001	0.009	0.019	0.005
4	0.0003	0.0105	0.013	0.013	0.001	0.022	0.016	0.007
8	0.0004	0.0096	0.02	0.014	0.002	0.027	0.016	0.006
16	0.0005	0.0083	0.038	0.019	0.002	0.033	0.014	0.007
32	0.0003	0.0112	0.044	0.026	0.002	0.034	0.019	0.01
64	0.0277	0.0378	0.1	0.049	0.032	NOTAV/	NOTAV/	0.018



	TYPE							
COUNTER								
NUM_TH	NO_LOCK	PTHREAD	TAS	TATAS	TATAS_B	TICKET	MCS	FAI
1	40000	40000	40000	40000	40000	40000	40000	40000
2	24593	40000	40000	40000	40000	40000	40000	40000
4	11102	40000	40000	40000	40000	40000	40000	40000
8	6179	40000	40000	40000	40000	40000	40000	40000
16	2927	40000	40000	40000	40000	40000	40000	40000
32	1488	40000	40000	40000	40000	40000	40000	40000
64	8378	40000	40000	40000	40000	40000	40000	40000

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

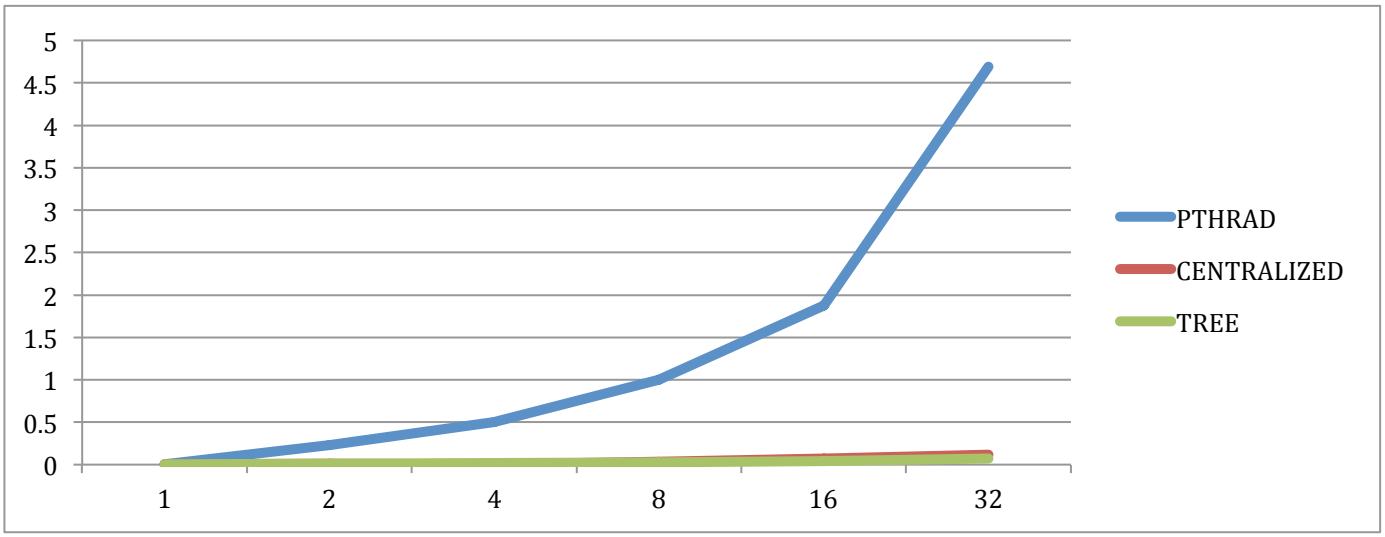
		TYPE							
	EXE_TIME								
NUM_TH		NO_LOCK	PTHREAD	TAS	TATAS	TATAS_B	TICKET	MCS	FAI
	1	0.0002	0.002	0.001	0.001	0.0009	0.002	0.002	0.001
	2	0.0007	0.0086	0.008	0.017	0.0022	0.006	0.037	0.009
	4	0.0014	0.0399	0.034	0.034	0.0043	0.061	0.055	0.023
	8	0.0022	0.0534	0.093	0.087	0.0085	0.133	0.097	0.054
	16	0.0051	0.0829	0.308	0.162	0.0206	0.363	0.149	0.106
	32	0.0176	0.1473	1.172	0.443	0.0749	0.763	0.25	0.168
	64	0.0323	0.2961	4.331	1.725	0.2231	NOTAV	NOTAV	0.279



		TYPE							
	COUNTER								
NUM_TH		NO_LOCK	PTHREAD	TAS	TATAS	TATAS_B	TICKET	MCS	FAI
	1	40000	40000	40000	40000	40000	40000	40000	40000
	2	50212	80000	80000	80000	80000	80000	80000	80000
	4	46561	160000	160000	160000	160000	160000	160000	160000
	8	47941	320000	320000	320000	320000	320000	320000	320000
	16	50460	640000	640000	640000	640000	640000	640000	640000
	32	59051	1E+06	1280000	1280000	1280000	1280000	1280000	1280000
	64	50047	3E+06	2560000	2560000	2560000	2560000	2560000	2560000

2.3 Barriers(time:s)

		TYPE		
	EXE_TIME			
NUM_TH		PTHRAD	CENTRALIZED	TREE
	1	0.0031	0.0004	0.0015
	2	0.2302	0.0086	0.0095
	4	0.4987	0.0063	0.0185
	8	1.0019	0.0377	0.0297
	16	1.8756	0.0685	0.0453
	32	4.6896	0.1194	0.071
	64	11.7758	NOTAVAILABLE	NOTAVAILABLE



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.

```
cpu_set_t cpuset;
int setted, set_aff;
setted = task_id % num_cpu;
thread_id = pthread_self(); // get the ID of calling thread
CPU_ZERO(&cpuset);
CPU_SET(setted, &cpuset); // set specified CPU in a set

set_aff = pthread_setaffinity_np(thread_id, sizeof(cpu_set_t), &cpuset); // set thread I to core I
assert(set_aff == 0);
```

4.2 TAS&TATAS locks

```
for(i = 0; i < num_count; i++){
    while (tas(&flag) == 1);  //acquire tas lock, either get the lock or spin
    val_counter++;
    flag = 0;  //release tas lock
}
```

```
void tatas_lock(volatile unsigned long* flag){
    while (1){
        while(*flag != 0);
        if(tas(flag) == 0)
            return;
    }
}

void tatas_unlock(volatile unsigned long* flag){
    *flag = 0;  //reset the flag
}
```

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.)

```
void InitializeBarrier(barr *B) {
    int i;
    node* root = B->leaf[0];
    root->count = 2;
    root->parent = NULL;
    root->sense = 0;

    for (i = (2 * (num_th - 1)); i >= (num_th - 1); i--) {
        B->leaf[i]->count = 1;
        B->leaf[i]->parent = B->leaf[(i-1)/2];
        B->leaf[i]->sense = 0;
    }

    for(i = 1; i < (num_th - 1); i++){
        B->leaf[i]->count = 2;
        B->leaf[i]->sense = 0;
        B->leaf[i]->parent = B->leaf[(i-1)/2];
    }

    for (i = 0; i < num_th; i++){
        B->leaf[i]->k = B->leaf[i]->count;
    }
}
```

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.