CS433A : Report | Design Exercise 2

> ***Submitted on : April 3, 2022 by Group No. 24***
>
> ***Dipanshu Garg | 190306***
>
> ***Gurbaaz Singh Nandra | 190349***

Deliverables

```
.
├── omp_main_barrier.c
├── omp_main_lock.c
├── pthread_main_barrier.c
├── pthread_main_lock.c
├── report.pdf
└── sync_library.c

0 directories, 6 files
```

Question 1

Usage

```
gcc -O0 -pthread pthread_main_lock.c -o pthread_main_lock
./pthread_main_lock <number-of-threads>

gcc -O0 -fopenmp omp_main_lock.c -o omp_main_lock
./omp_main_lock <number-of-threads>
```

Results

These results have been obtained on an`8-core` machine.

Thread	i. Lamport's Bakery	ii. Spin-lock	iii. Test-and-test-and-set	iv. Ticket lock	v. Array lock	vi. POSIX mutex	vii. Binary Semaphore	viii. #pragma omp critical	Best Locking Technique
1	722730	103551	115329	115131	162515	164680	185999	156432	ii.
2	3206033	1088492	996153	1663097	2328109	999836	2951201	660225	viii.
4	6504397	5934378	4207946	3622779	6906580	2395333	8172353	2390785	viii.
8	21963826	28552022	7951185	10479153	13671661	8955384	29304285	6267495	viii.
16	> 6e+8	> 6e+8	> 6e+8	> 6e+8	> 6e+8	22554756	60494359	25957631	vi.

> ***NOTE*** *: Time is in microseconds*

- Lamport's Bakery is performing poorly in all the cases, which is as expected because it has too many instructions overhead to implement just a lock.
- In case of single thread, Spin-lock performs the best. TTS and Ticket lock also have similar performances as they are all based on hardware support, and there is no issue of coherence in case of single thread.
- In case of two threads, #pragma omp critical performs the best, followed closely by TTS, POSIX mutex and Spin-lock.
- As we further increase the threads, #pragma omp critical continues to dominate until 16 threads, where POSIX performs slightly better.
- On increasing number of threads, TTS starts to outperform Spin-lock, which is as expected, because TTS only tries to acquire the lock when it is free. This reduces the cache coherence overhead in case of TTS.
- Although we expected Array Lock to perform better than Ticket Lock, but the results for us were otherwise. This could be because the array lock acquiration required modulo (%) and mutliplication (*) operation per access per every busy waiting access. The cost of these computations could have exceeded the gain obtained by reducing bus transactions.
- Binary semaphores are performing inferior for upto 8 threads because of the overhead of thread scheduling and descheduling. However, for 16 threads they perform relatively better as we are running our programs on 8-core machine.

Question 2

Usage

```
gcc -O0 -pthread pthread_main_barrier.c -o pthread_main_barrier
./pthread_main_barrier <number-of-threads>

gcc -O0 -fopenmp omp_main_barrier.c -o omp_main_barrier
./omp_main_barrier <number-of-threads>
```

Results

These results have been obtained on an `8-core` machine.

Thread	i. Centralized sense-reversing barrier	ii. Tree barrier using busy-wait on flags	iii. Centralized barrier using POSIX condition variable	iv. Tree barrier using POSIX condition variable	v. POSIX barrier interface (pthread_barrier_wait)	vi. #pragma omp barrier	Best Barrier Technique
1	32112	6725	32233	11177	253844	239071	ii.
2	207018	130021	2464271	4979723	2110571	275993	ii.
4	1419658	320566	5460099	11757788	4049400	359919	ii.
8	4874075	541181	18689335	23339640	10233316	390497	vi.
16	> 6e+8	> 6e+8	43973611	52925339	21773587	26601924	v.

> ***NOTE*** *: Time is in microseconds*

- Tree barrier using busy-wait on flags and #pragma omp barrier are performing significantly better upto 8 threads than the remaining 4 barriers. For tree barrier, the reason is due to the decentralised synchronisation which has low overhead.
- Tree barrier using POSIX condition variable also performs well on single thread, however on increasing threads, its performance starts to deteriorate (roughly doubling) due to overhead of scheduling and descheduling.
- Centralised barrier using reverse-sense performs worse than tree barriers because of the sequential critical section. Centralised barrier using POSIX condition variable performs even worse due to the added cost of scheduling and descheduling (upto 8 threads). However on 16 threads it outperforms all the busy-waiting barriers because now scheduling is cheaper as we are running on an 8-core machine.
- As we further increase the threads, #pragma omp barrier continues to dominate over POSIX barrier until 16 threads, where POSIX performs slightly better.