CS6240 Assignment 1 Section 1 **Priyanka Shenoy**

Notice : All computations have been performed on i7 processor with quad cores, hence at any given time 4 threads should be running simultaneously.

Answering questions asked as a part of Assignment 1 -

Without Fibonacci →

Program	Average Time	Min Time	Max Time
Coarse Lock	1001.9	955	1075
Sequential Lock	954.6	694	1865
No Shared Lock	378.6	362.0	439
Fine Lock	365	345	419
No Lock	356.4	340	397

With Fibonacci →

Program	Average Time	Min Time	Max Time
Sequential Lock	981.3	681	1561
Coarse Lock	844	887	966
No Shared Lock	391	367	444
Fine Lock	370.4	350	464
No Lock	363.8	335	404

Speed Up for the following locks are →

Coarse Lock = 1.163
No Shared Lock = 2.509
Fine Lock = 2.65
No Lock = 2.698

---

1. Which program would you expect to complete fastest and why? Do experiments confirm the results ? Explain with reasons

A. My hypothesis for the same is, if any give set of threads are allowed to run with all resources and without any locks/holds, the threads will start wait and stop as and when it is convenient for them. When the processor feels a thread is taking lot of time, it gives priority to another one, hence effectively reducing time required for completion of code. The same principle is applied when we create NO_LOCK, hence it should be the fastest.

I have run my program several times and observed NO-LOCK code is running the fastest which confirms my hypothesis.

2. Which program would you normally expect to complete the slowest and why? Do experiments confirm your your observation?

A. I expect SEQUENTIAL to complete the slowest because it is like there is only one thread in the system and entire execution is supposed to be completed by it. This will definitely be slower than multiple threads working on the same problem to find average of max temperature.
The results from the program assure the same observation, SEQ does take max time for execution(only for case B).

3. Compare temperatures, report if anyone is incorrect or if programs crashed because of concurrent processes.

A. Temperatures remain the same for all scenarios except NO_LOCK which is expected since multiple threads access and make changes to objects without proper completion of process/snippet.
I also faced crash while running the code in NO_LOCK i.e. NullPointerException which was because, my thread one had already access to object(ID) but had not updated in the accumulated data structure values(max temp and count), but second thread went ahead, accessed the ID but did not get any value in the data structure which cause the Exception
//*Add temperatures here *//

4. Compare run times of SEQ and COARSE_LOCK. Why is one slower than other?

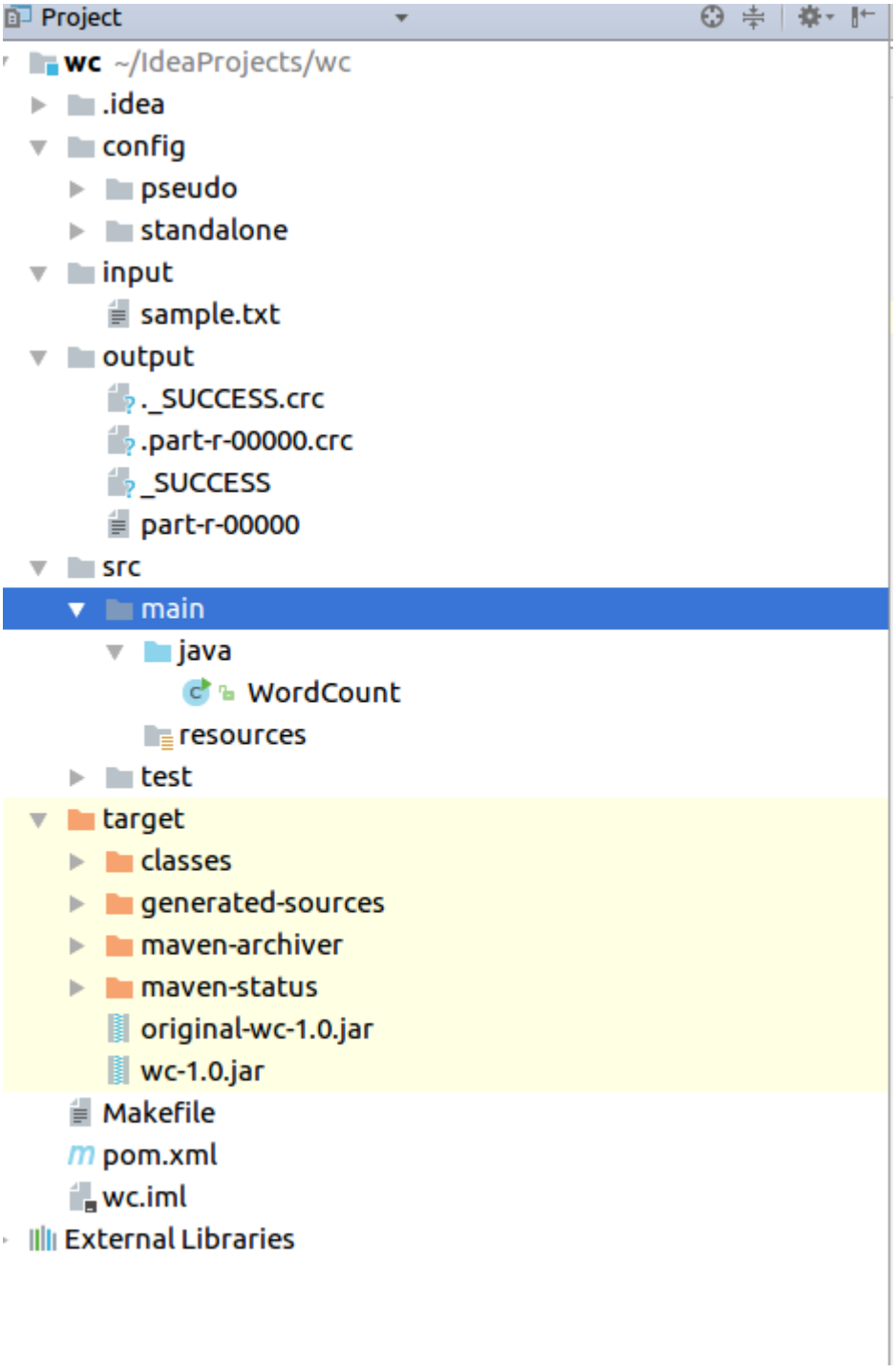
A. Logically speaking, SEQUENTIAL should be slowest *without Fibonacci* and COARSE_LOCK slowest *with Fibonacci*.
The execution of sequential is like there is only one thread in the system and it executes program from start to end. This is definitely slower than multiple threads running the same code. Hence, without Fibonacci we have Sequential as slowest, which is observed through results as well.
When we have Fibonacci, in a coarse lock, it so happens that a thread locks the object, hence no other threads have access to the object. In addition to this, Fibonacci slows the execution even more. This in turn according to me may become slower than sequential.
My COARSE_LOCK code with Fibonacci 3/5 times is slower than sequential but other times sequential is slower. This could be possible because the lock on the objects is released quickly and hence runs faster than sequential, but I feel if the programming task is bigger, COARSE_LOCK would end up being slower than SEQUENTIAL.

5. How does higher cost computation affect difference between COARSE_LOCK and FINE_LOCK?

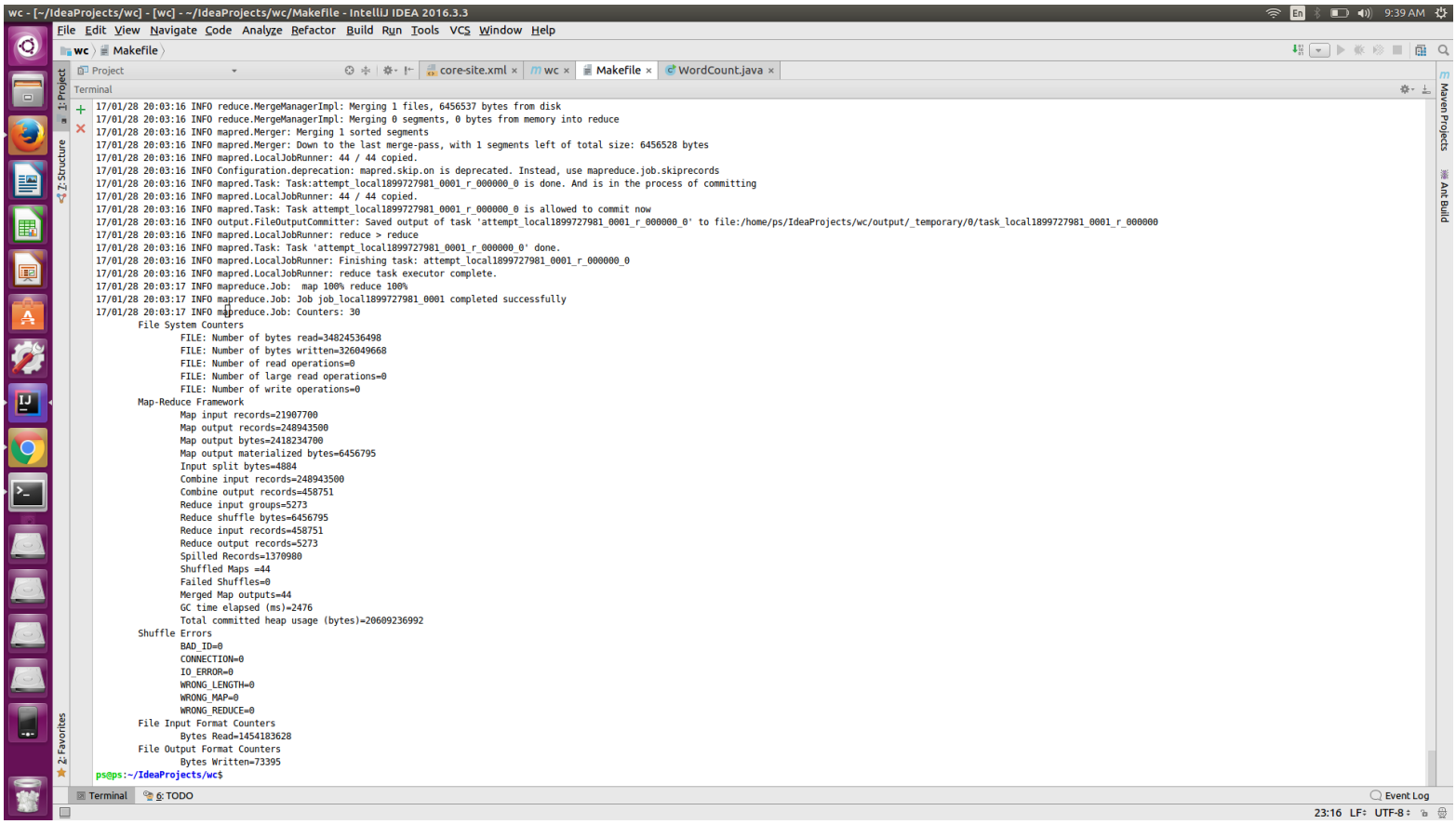
A. The computation remains the same for both COARSE_LOCK and FINE_LOCK. Fibonacci does not affect computation of COARSE_LOCK and FINE_LOCK since both ways before we update the values in accumulated data structure we ask the threads to wait. There is no change in the ratio of execution times observed.
The same is observed while running the code.

CS6240 Assignment 1 Section 1 **Priyanka Shenoy**

Standalone Screen shot →



CS6240 Assignment 1 Section 1 Priyanka Shenoy



CS6240 Assignment 1 Section 1 Priyanka Shenoy