Lab 1: Report

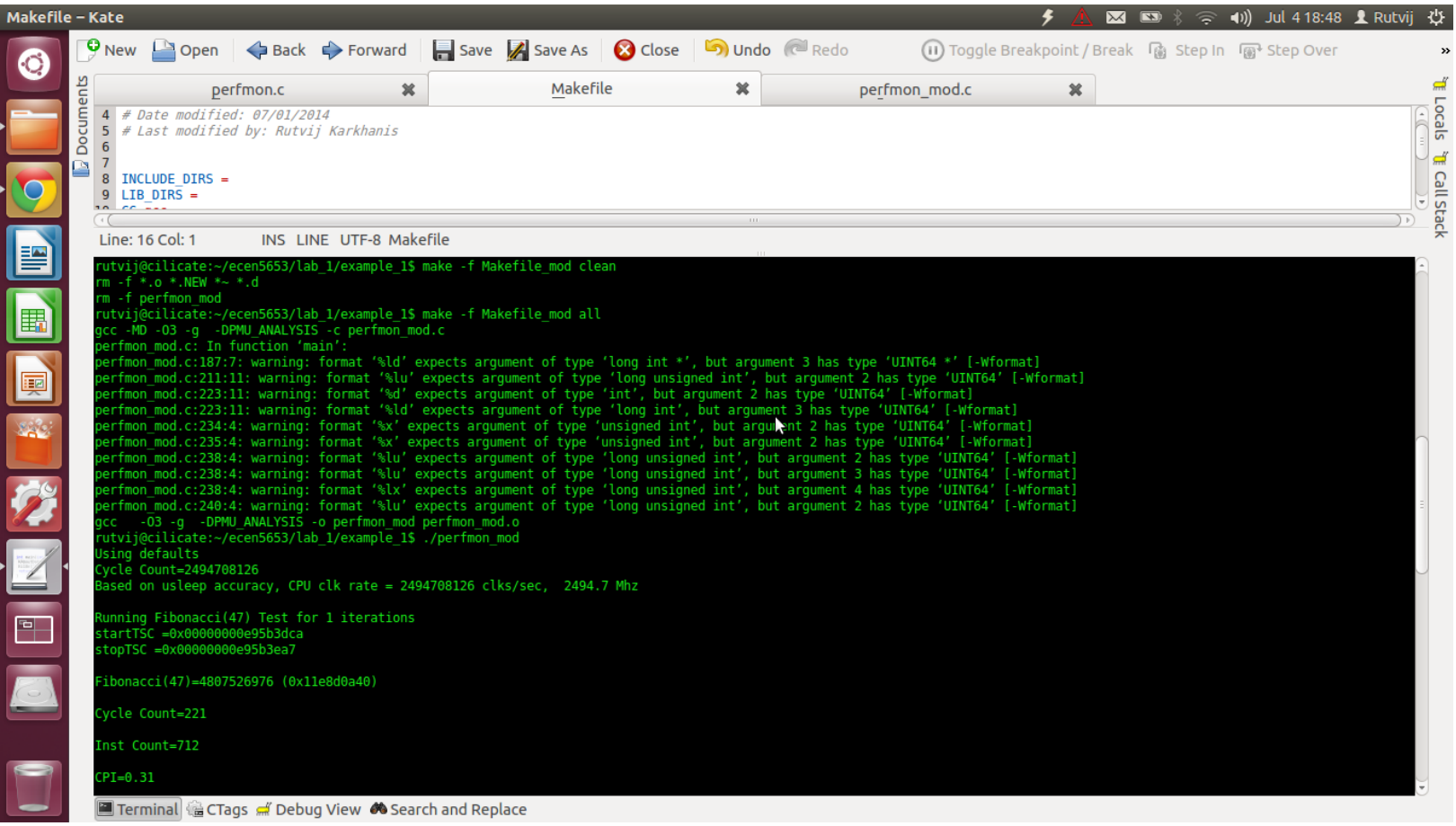
Rutvij Girish Karkhanis

Q1) Q2), and Q3) are included in the zip file

Q4).

1. No, the results obtained by directly applying example one code were not consistent with the /proc/cpuinfo results. After modification, the CPu frequency calculated by the TSC is 2494.6 Mhz while the frequency shown by /proc/cpuinfo is 2501.0 Mhz
2. The code in example one attempts to calculate the CPU frequency by using the cycles elapsed in one second (using usleep()) read by the TSC (Time Stamp Counter).
3. The code does work on my Corei5 2450M dual core machine with modifications to suit the 64 bit architecture of the processor.
4. Yes the code can be modified by using threads by setting core affinity for each thread. Each thread tied to the core accesses the TSC of that particular core. This ensures that the Linux SMP wouldn't perform a context switch and the values for the TSC would be calculated concurrently by the threads. This would ensure correct computation of the cycles elapsed.
5. The instructions in the outer and the inner loop are counted and multiplied with the iteration numbers, and stored in a variable. The cycles elapsed are calculated for the Fibonacci computations and are divided by the instruction count to get the cycles per instruction.

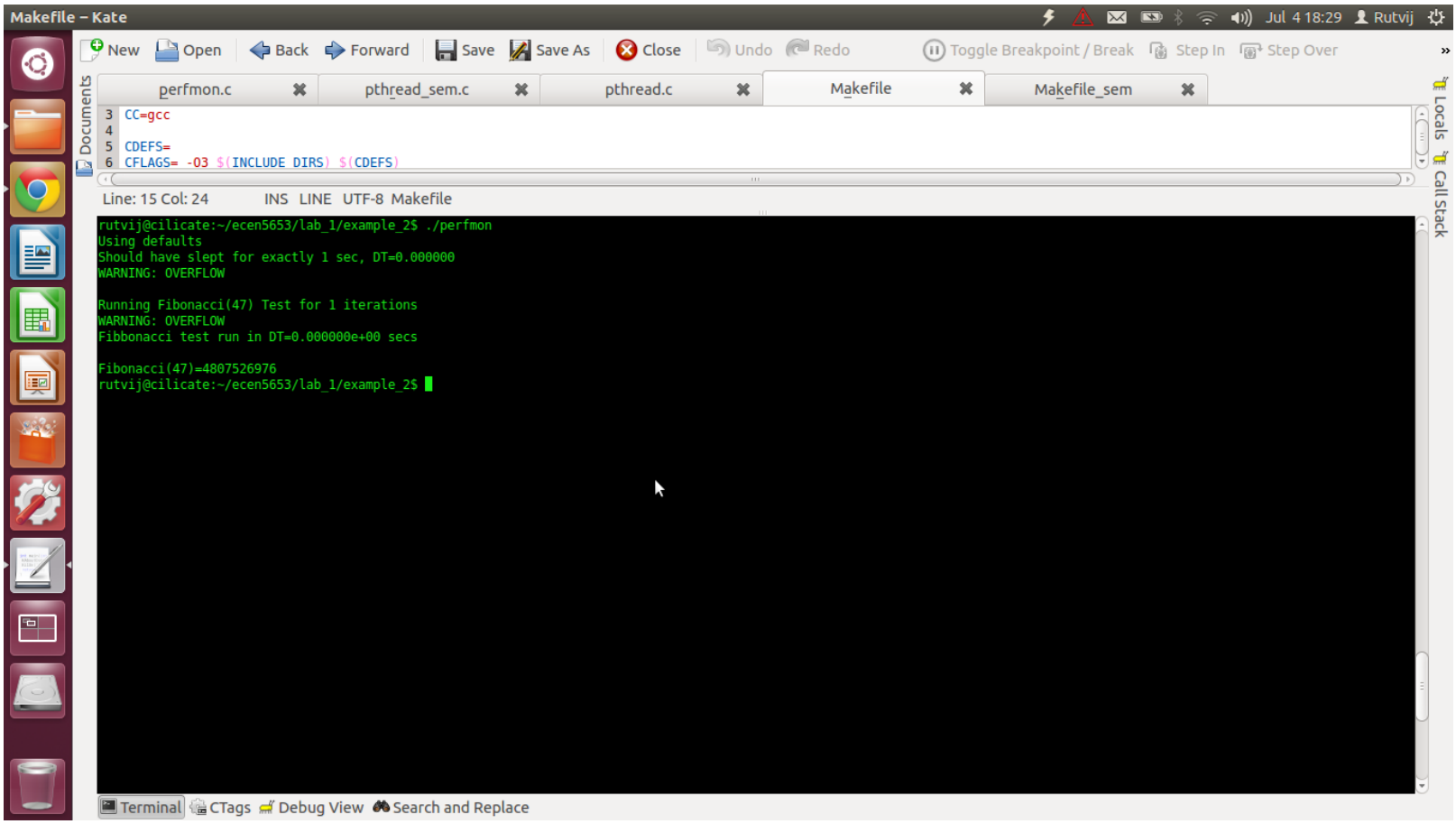
6. The CPI for my machine is 0.31 because of a deep pipeline and a fairly large L1 cache for my CPU evident as follows:



Q5).

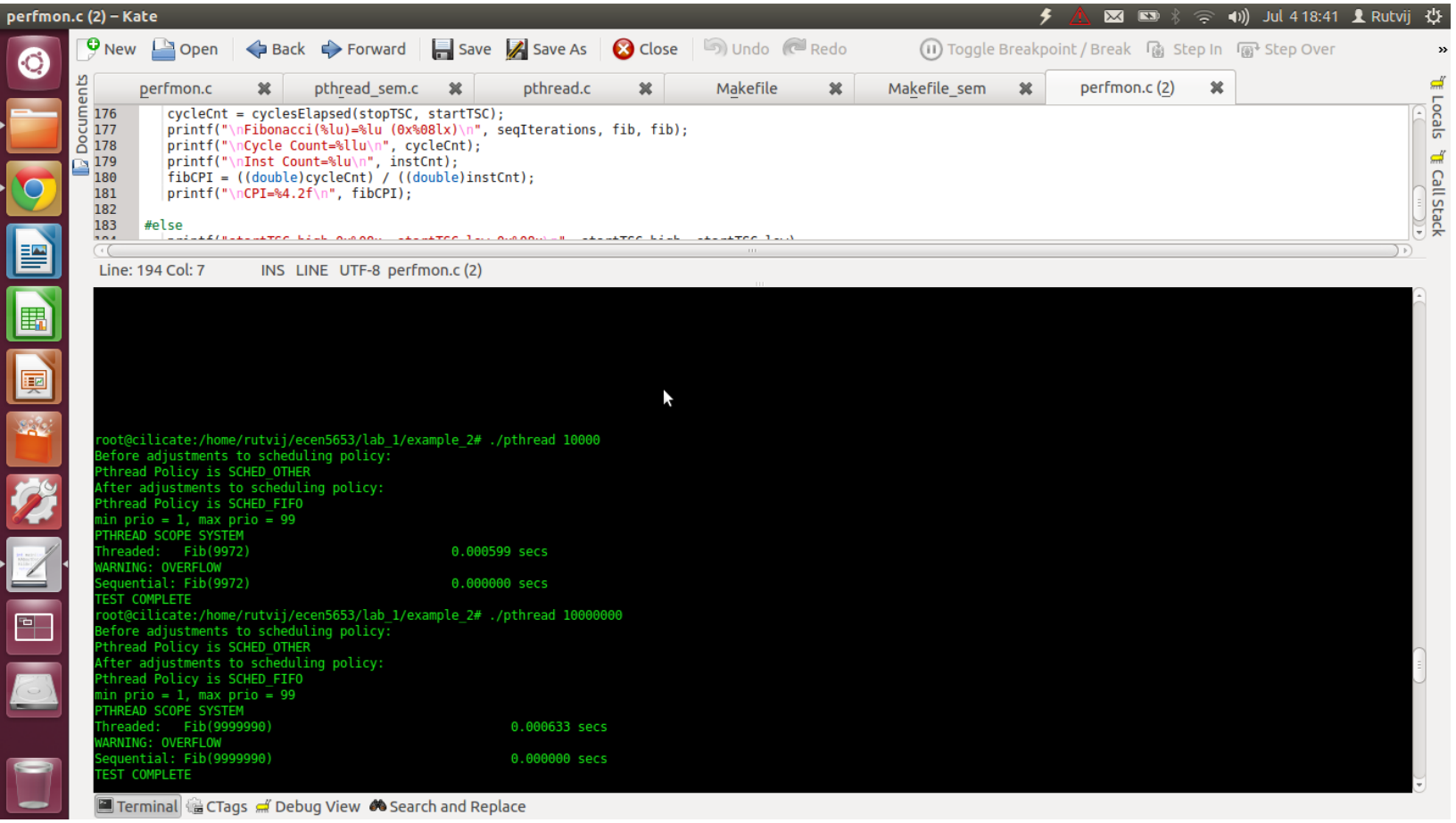
1. The parallel computation of the Fibonacci series never becomes faster than the sequential computation. The reason for this is possibly because the pthreads concurrently compute the Fibonacci series independently. So there is a lot of context switching between the threads by the Linux SMP. The context switch takes a lot of CPU cycles.

Codefile: perfmon.c
makefile: Makefile



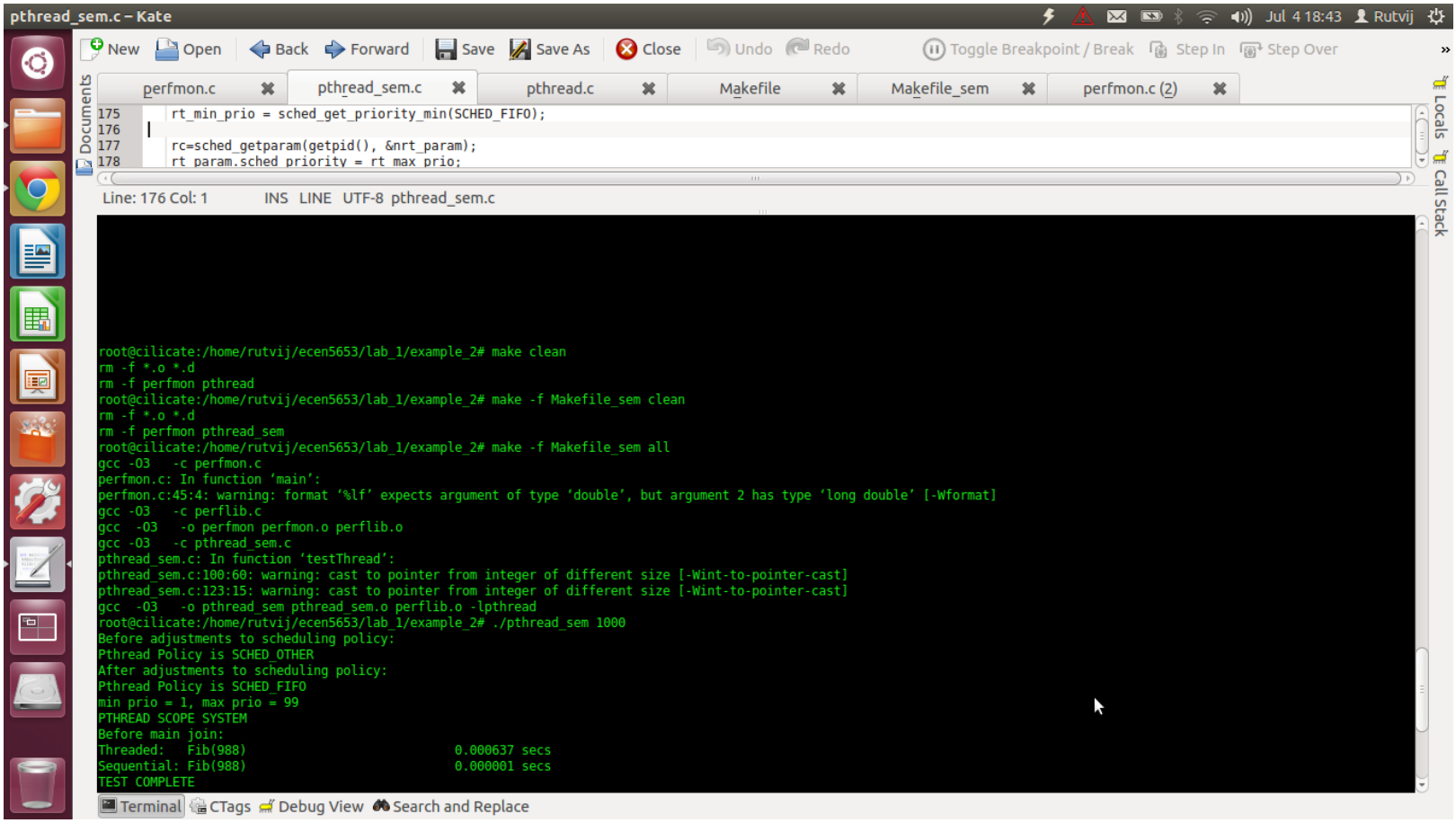
2. The priorities assigned to the pthreads are within a range of 1 to 99. We do not assign specific priorities to the threads hence scheduler assigns random priorities to the threads and schedules the first thread in the FIFO. The Linux SMP then performs context switches between these threads.

Codefile: pthread.c
Makefile: Makefile



3. The code is included in the submission zip. The specific code file is pthread_sem.c

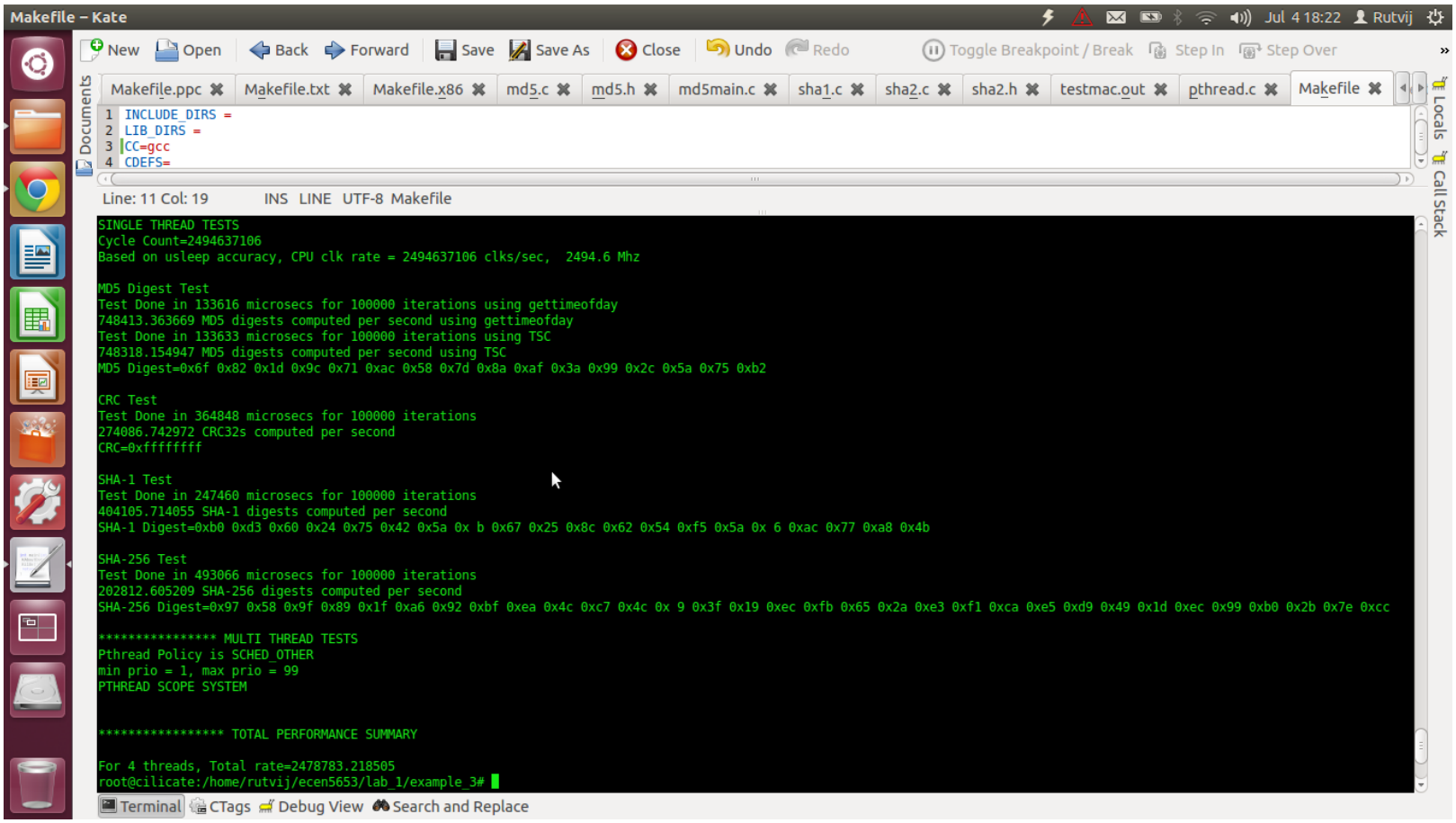
Codefile: pthread_sem.c
makefile: Makefile_sem



Q6).

1. For test buffer MD5 digest is the fastest sequentially. It is evident as follows:

Code file: testdigest.c
makefile: Makefile

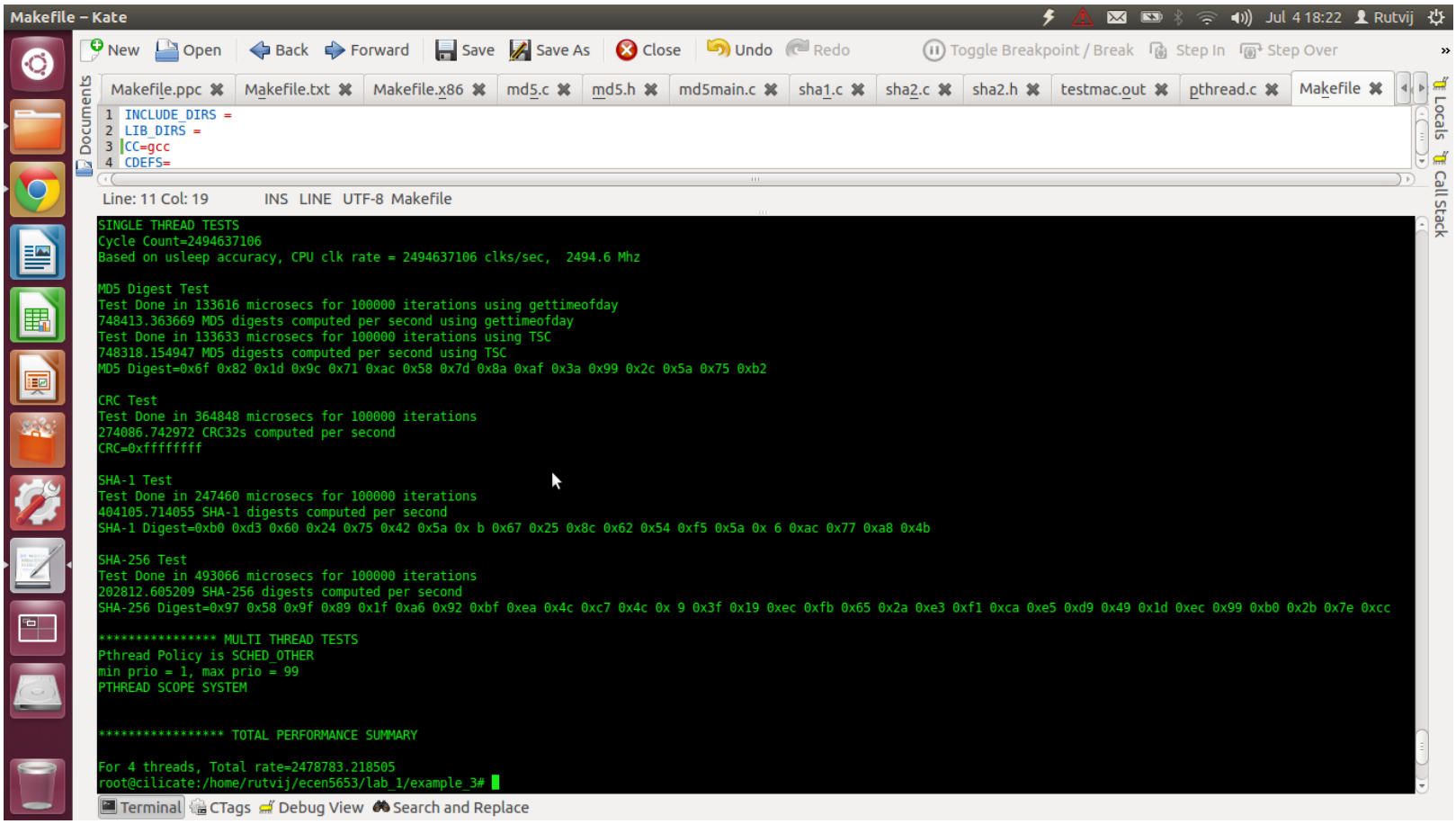


2. It takes an average of 3349 cycles to compute one digest on my machine.

Codefile: testdigest.c
makefile: Makefile

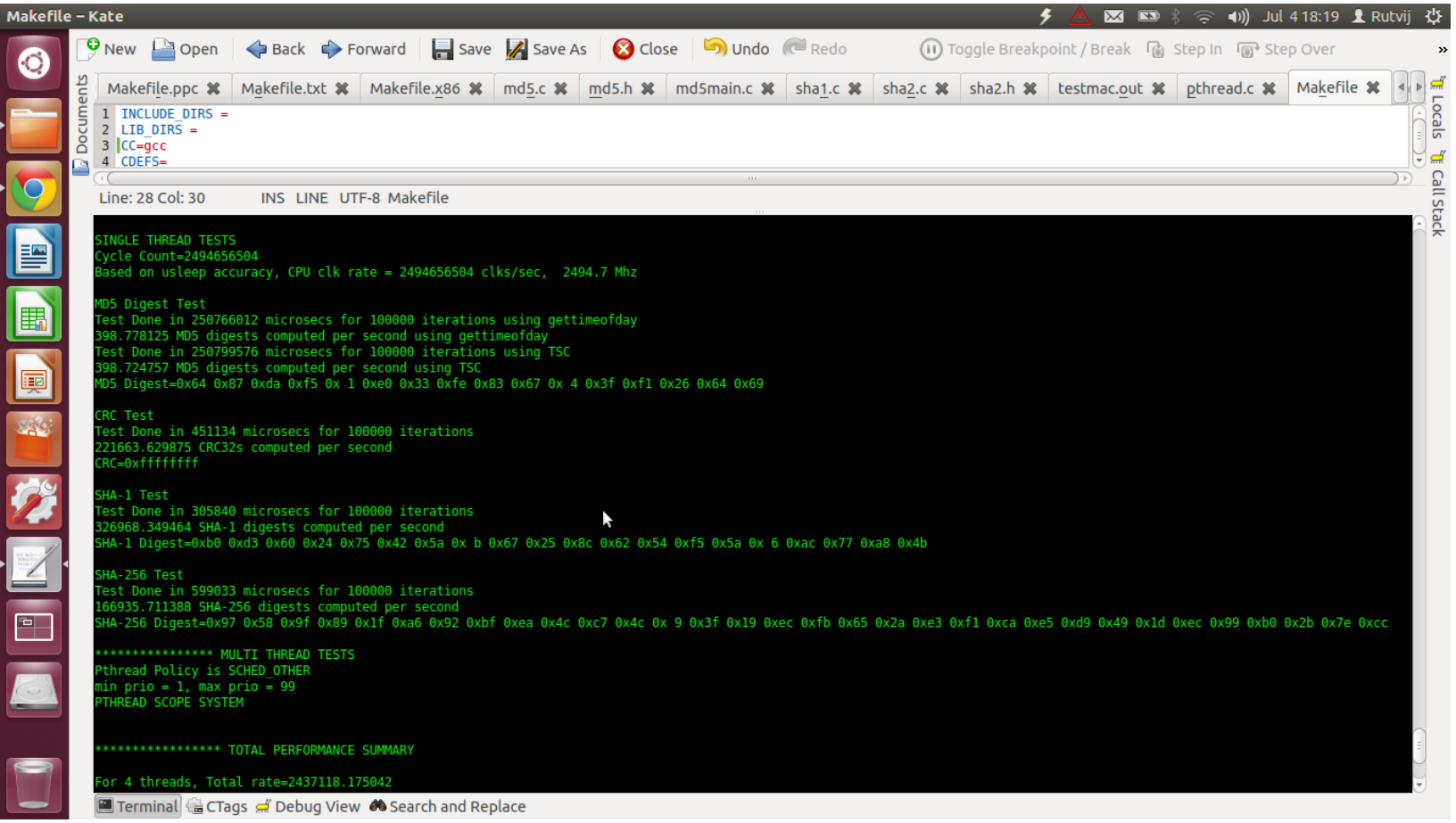
3. No, on my system the TSC is less accurate than gettimeofday because usleep terminates earlier than stipulated. This causes to profile the CPU with a lesser clock speed as evident in the following screenshot.

Codefile: testdigest_ntsc.c
makefile: Makefile_ntsc



4. For a 24 bit 720x480 resolution ppm image, about 399 MD5 digests are computed per second that is one digest takes 2.5 milliseconds computation. Thus this would easily keep up with the frame rate slightly less than 30 fps.

Codefile: testdigest_ntsc.c
makefile: Makefile_ntcs



5. The code for the same is included in the zip. The screenshot of the output is as follows:

Codefile: testdigest_ntsc_concurrent
makefile: Makefile_ntsc_concurrent