

Lab 1: Report

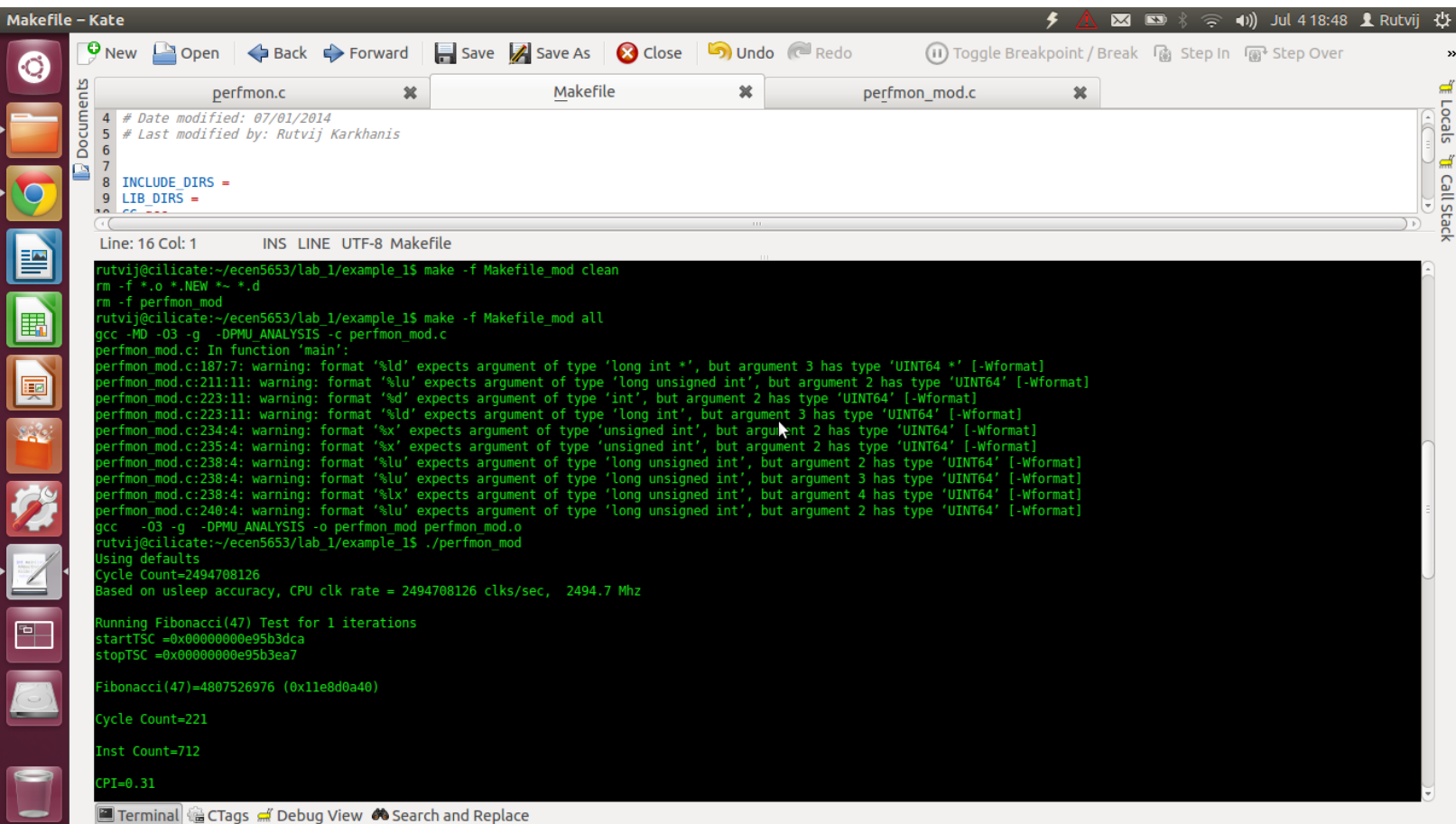
Rutvij Girish Karkhanis

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:



The screenshot shows a Kate editor window with three tabs: `perfmon.c`, `Makefile`, and `perfmon_mod.c`. The `Makefile` tab is active, showing a file with 9 lines. The first two lines are comments: `# Date modified: 07/01/2014` and `# Last modified by: Rutvij Karkhanis`. Lines 8 and 9 are `INCLUDE_DIRS =` and `LIB_DIRS =` respectively. The terminal output shows the execution of `make -f Makefile_mod clean` and `make -f Makefile_mod all`. The compilation process uses `gcc -MD -O3 -g -DPMU_ANALYSIS -c perfmon_mod.c`. The output includes several warnings about format specifiers and the final result: `Fibonacci(47)=4807526976 (0x11e8d0a40)`, `Cycle Count=221`, `Inst Count=712`, and `CPI=0.31`.

```
4 # Date modified: 07/01/2014
5 # Last modified by: Rutvij Karkhanis
6
7
8 INCLUDE_DIRS =
9 LIB_DIRS =
```

Line: 16 Col: 1 INS LINE UTF-8 Makefile

```
rutvij@cilicate:~/ecen5653/lab_1/example_1$ make -f Makefile_mod clean
rm -f *.o *.NEW *~ *.d
rm -f perfmon_mod
rutvij@cilicate:~/ecen5653/lab_1/example_1$ make -f Makefile_mod all
gcc -MD -O3 -g -DPMU_ANALYSIS -c perfmon_mod.c
perfmon_mod.c: In function 'main':
perfmon_mod.c:187:7: warning: format '%ld' expects argument of type 'long int *', but argument 3 has type 'UINT64 *' [-Wformat]
perfmon_mod.c:211:11: warning: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type 'UINT64' [-Wformat]
perfmon_mod.c:223:11: warning: format '%d' expects argument of type 'int', but argument 2 has type 'UINT64' [-Wformat]
perfmon_mod.c:223:11: warning: format '%ld' expects argument of type 'long int', but argument 3 has type 'UINT64' [-Wformat]
perfmon_mod.c:234:4: warning: format '%x' expects argument of type 'unsigned int', but argument 2 has type 'UINT64' [-Wformat]
perfmon_mod.c:235:4: warning: format '%x' expects argument of type 'unsigned int', but argument 2 has type 'UINT64' [-Wformat]
perfmon_mod.c:238:4: warning: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type 'UINT64' [-Wformat]
perfmon_mod.c:238:4: warning: format '%lu' expects argument of type 'long unsigned int', but argument 3 has type 'UINT64' [-Wformat]
perfmon_mod.c:238:4: warning: format '%lx' expects argument of type 'long unsigned int', but argument 4 has type 'UINT64' [-Wformat]
perfmon_mod.c:240:4: warning: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type 'UINT64' [-Wformat]
gcc -O3 -g -DPMU_ANALYSIS -o perfmon_mod perfmon_mod.o
rutvij@cilicate:~/ecen5653/lab_1/example_1$ ./perfmon_mod
Using defaults
Cycle Count=2494708126
Based on usleep accuracy, CPU clk rate = 2494708126 clks/sec, 2494.7 Mhz

Running Fibonacci(47) Test for 1 iterations
startTSC =0x00000000e95b3dca
stopTSC =0x00000000e95b3ea7

Fibonacci(47)=4807526976 (0x11e8d0a40)

Cycle Count=221

Inst Count=712

CPI=0.31
```

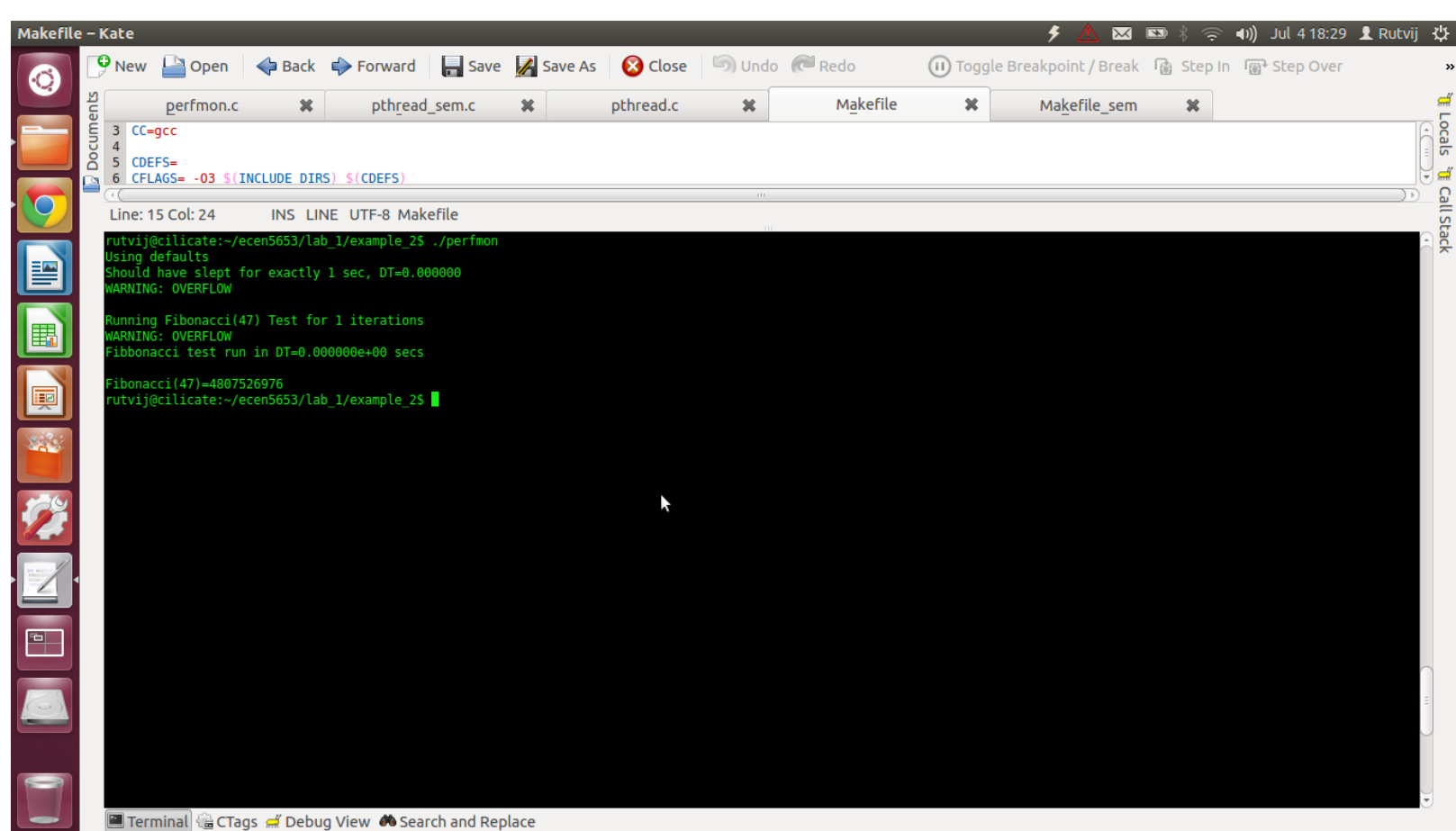
Terminal CTags Debug View Search and Replace

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



Makefile - Kate

Documents

perfmon.c x pthread_sem.c x pthread.c x Makefile x Makefile_sem x

Line: 15 Col: 24 INS LINE UTF-8 Makefile

```
rutvij@cilicate:~/ecen5653/lab_1/example_2$ ./perfmon
Using defaults
Should have slept for exactly 1 sec, DT=0.000000
WARNING: OVERFLOW

Running Fibonacci(47) Test for 1 iterations
WARNING: OVERFLOW
Fibonacci test run in DT=0.000000e+00 secs

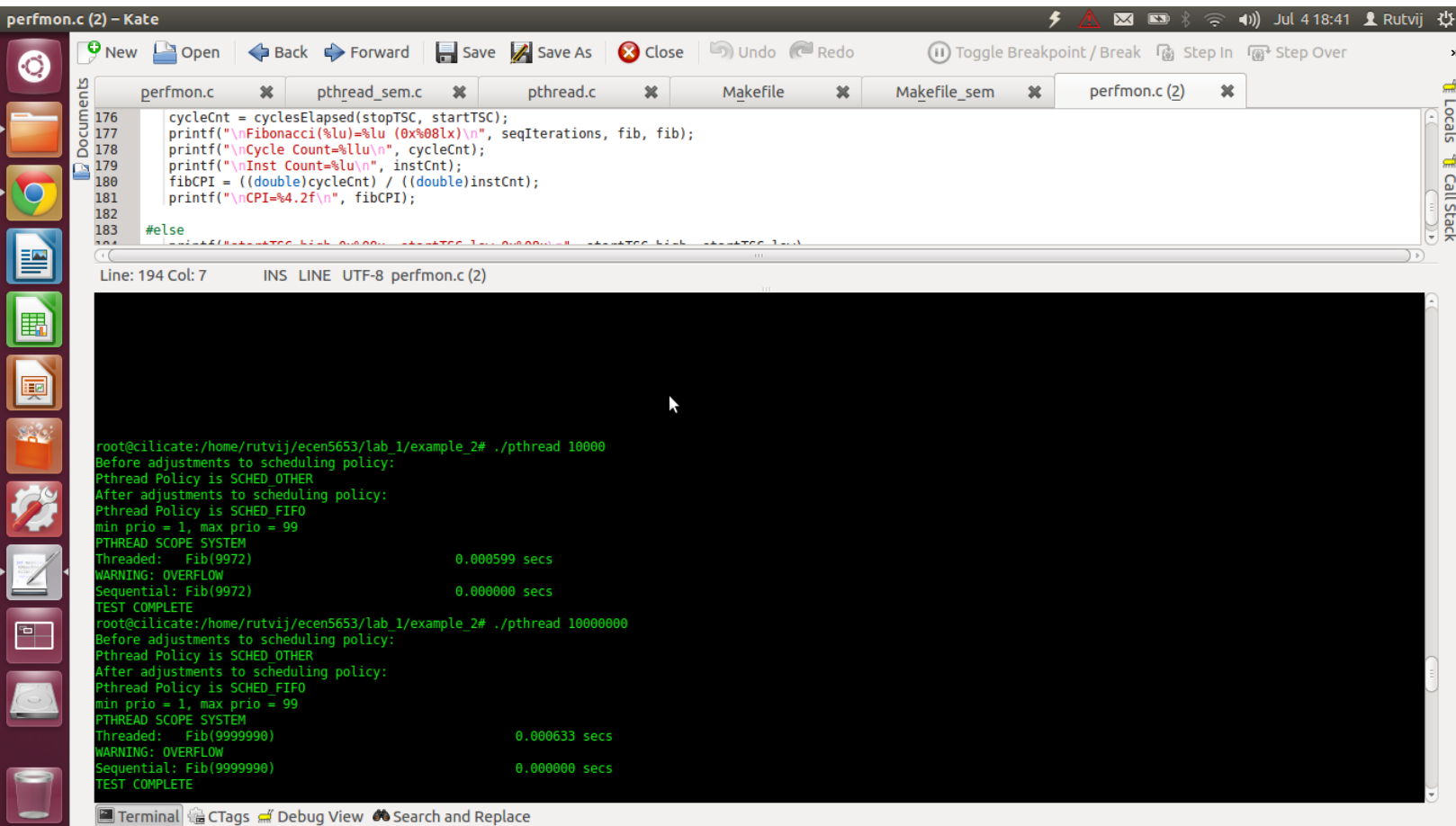
Fibonacci(47)=4807526976
rutvij@cilicate:~/ecen5653/lab_1/example_2$
```

Terminal CTags Debug View Search and Replace

-
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



The screenshot shows a KDE Kate editor window titled "perfmon.c (2) - Kate". The editor has several tabs open: "perfmon.c", "pthread_sem.c", "pthread.c", "Makefile", "Makefile_sem", and "perfmon.c (2)". The "perfmon.c" tab is active, showing C code for a Fibonacci sequence calculation using pthreads. The code includes functions for timing and printing results. Below the editor is a terminal window showing the output of running the program. The terminal output shows the results of two runs: one with 10000 threads and one with 1000000 threads. The output includes the pthread policy (SCHED_OTHER), the min and max priority (1 and 99), the pthread scope (SYSTEM), and the time taken for the threaded and sequential versions of the Fibonacci calculation. The threaded version is significantly faster than the sequential version.

```
176 cycleCnt = cyclesElapsed(stopTSC, startTSC);
177 printf("\nFibonacci(%lu)=%lu (0x%08lx)\n", seqIterations, fib, fib);
178 printf("\nCycle Count=%llu\n", cycleCnt);
179 printf("\nInst Count=%lu\n", instCnt);
180 fibCPI = ((double)cycleCnt) / ((double)instCnt);
181 printf("\nCPI=%4.2f\n", fibCPI);
182
183 #else
184 ...
```

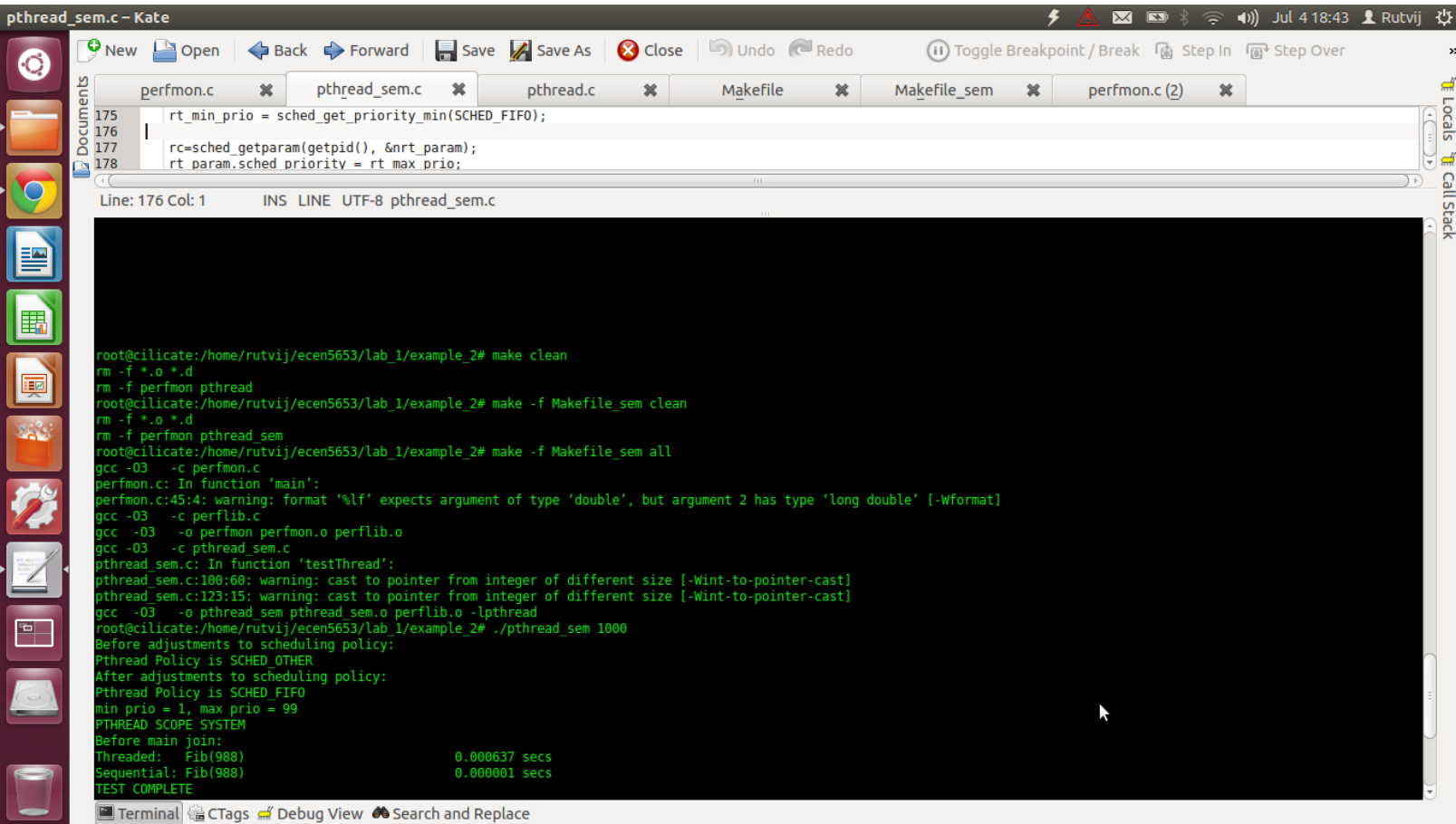
```
root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# ./pthread 10000
Before adjustments to scheduling policy:
Pthread Policy is SCHED_OTHER
After adjustments to scheduling policy:
Pthread Policy is SCHED_FIFO
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM
Threaded:  Fib(9972)                0.000599 secs
WARNING: OVERFLOW
Sequential: Fib(9972)                0.000000 secs
TEST COMPLETE

root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# ./pthread 1000000
Before adjustments to scheduling policy:
Pthread Policy is SCHED_OTHER
After adjustments to scheduling policy:
Pthread Policy is SCHED_FIFO
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM
Threaded:  Fib(9999990)              0.000633 secs
WARNING: OVERFLOW
Sequential: Fib(9999990)              0.000000 secs
TEST COMPLETE
```

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

Codefile: pthread_sem.c

makefile: Makefile_sem



pthread_sem.c - Kate

Line: 176 Col: 1 INS LINE UTF-8 pthread_sem.c

```
175 rt_min_prio = sched_get_priority_min(SCHED_FIFO);
176
177 rc=sched_getparam(getpid(), &nrt_param);
178 rt_param.sched_priority = rt_max_prio;
```

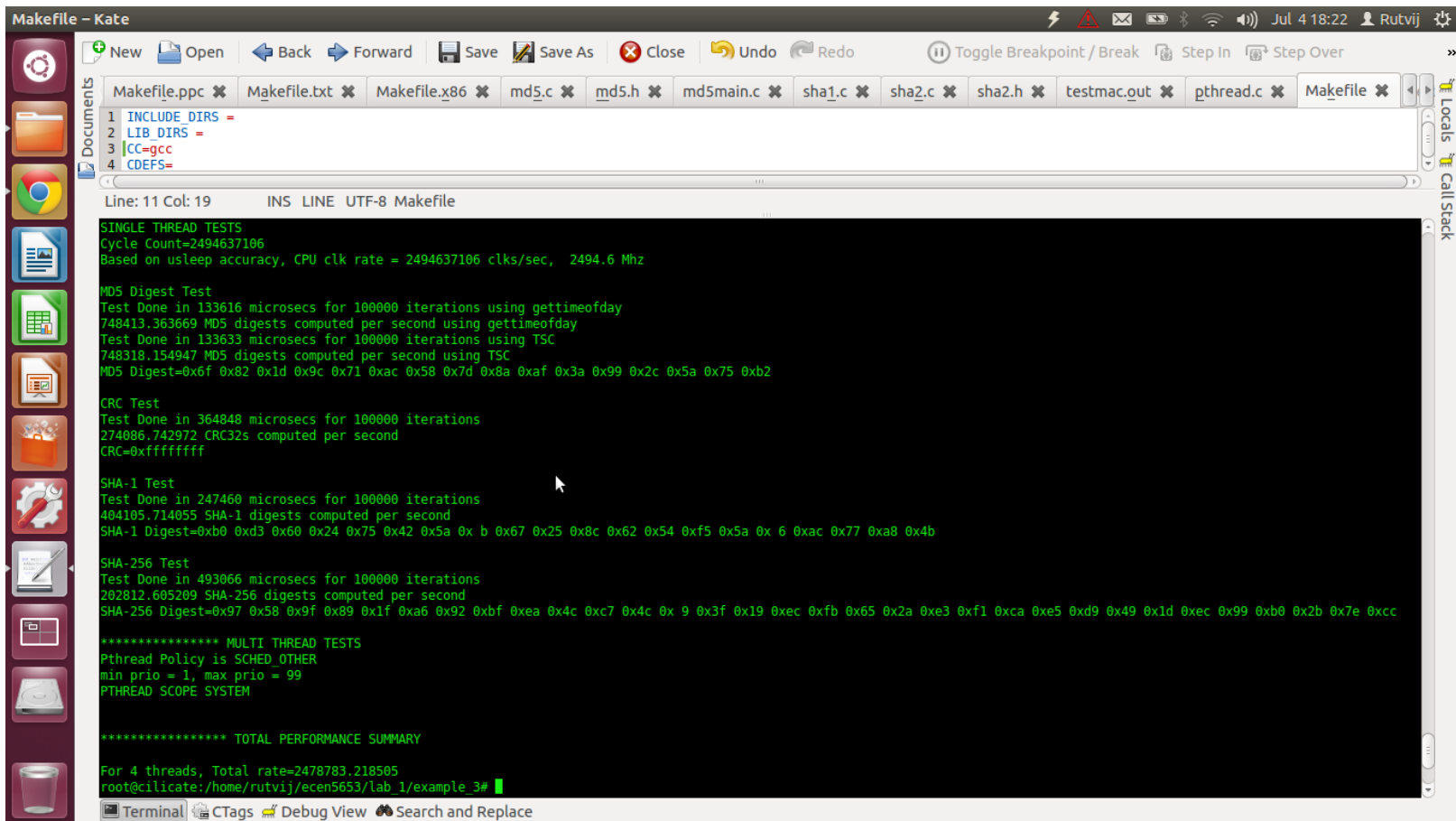
```
root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# make clean
rm -f *.o *.d
rm -f perfmon pthread
root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# make -f Makefile_sem clean
rm -f *.o *.d
rm -f perfmon pthread_sem
root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# make -f Makefile_sem all
gcc -O3 -c perfmon.c
perfmon.c: In function 'main':
perfmon.c:45:4: warning: format '%lf' expects argument of type 'double', but argument 2 has type 'long double' [-Wformat]
gcc -O3 -c perflib.c
gcc -O3 -o perfmon perfmon.o perflib.o
gcc -O3 -c pthread_sem.c
pthread_sem.c: In function 'testThread':
pthread_sem.c:100:60: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
pthread_sem.c:123:15: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
gcc -O3 -o pthread_sem pthread_sem.o perflib.o -lpthread
root@cilicate:/home/rutvij/ecen5653/lab_1/example_2# ./pthread_sem 1000
Before adjustments to scheduling policy:
Pthread Policy is SCHED_OTHER
After adjustments to scheduling policy:
Pthread Policy is SCHED_FIFO
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM
Before main join:
Threaded: Fib(988) 0.000637 secs
Sequential: Fib(988) 0.000001 secs
TEST COMPLETE
```

Q6).

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

Code file: testdigest.c

makefile: Makefile



```
Makefile - Kate
New Open Back Forward Save Save As Close Undo Redo Toggle Breakpoint / Break Step In Step Over
Makefile.ppc x Makefile.txt x Makefile.x86 x md5.c x md5.h x md5main.c x sha1.c x sha2.c x sha2.h x testmac.out x pthread.c x Makefile x
1 INCLUDE_DIRS =
2 LIB_DIRS =
3 CC=gcc
4 CDEFS=
Line: 11 Col: 19 INS LINE UTF-8 Makefile
SINGLE THREAD TESTS
Cycle Count=2494637106
Based on usleep accuracy, CPU clk rate = 2494637106 clks/sec, 2494.6 Mhz
MD5 Digest Test
Test Done in 133616 microsecs for 100000 iterations using gettimeofday
748413.363669 MD5 digests computed per second using gettimeofday
Test Done in 133633 microsecs for 100000 iterations using TSC
748318.154947 MD5 digests computed per second using TSC
MD5 Digest=0x6f 0x82 0x1d 0x9c 0x71 0xac 0x58 0x7d 0x8a 0xaf 0x3a 0x99 0x2c 0x5a 0x75 0xb2
CRC Test
Test Done in 364848 microsecs for 100000 iterations
274086.742972 CRC32s computed per second
CRC=0xffffffff
SHA-1 Test
Test Done in 247460 microsecs for 100000 iterations
404105.714055 SHA-1 digests computed per second
SHA-1 Digest=0xb0 0xd3 0x60 0x24 0x75 0x42 0x5a 0x b 0x67 0x25 0x8c 0x62 0x54 0xf5 0x5a 0x 6 0xac 0x77 0xa8 0x4b
SHA-256 Test
Test Done in 493066 microsecs for 100000 iterations
202812.605209 SHA-256 digests computed per second
SHA-256 Digest=0x97 0x58 0x9f 0x89 0x1f 0xa6 0x92 0xbf 0xea 0x4c 0xc7 0x4c 0x 9 0x3f 0x19 0xec 0xfb 0x65 0x2a 0xe3 0xf1 0xca 0xe5 0xd9 0x49 0x1d 0xec 0x99 0xb0 0x2b 0x7e 0xcc
***** MULTI THREAD TESTS
Pthread Policy is SCHED_OTHER
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM
***** TOTAL PERFORMANCE SUMMARY
For 4 threads, Total rate=2478783.218505
root@cilicate:/home/rutvij/ecen5653/lab_1/example_3#
```

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

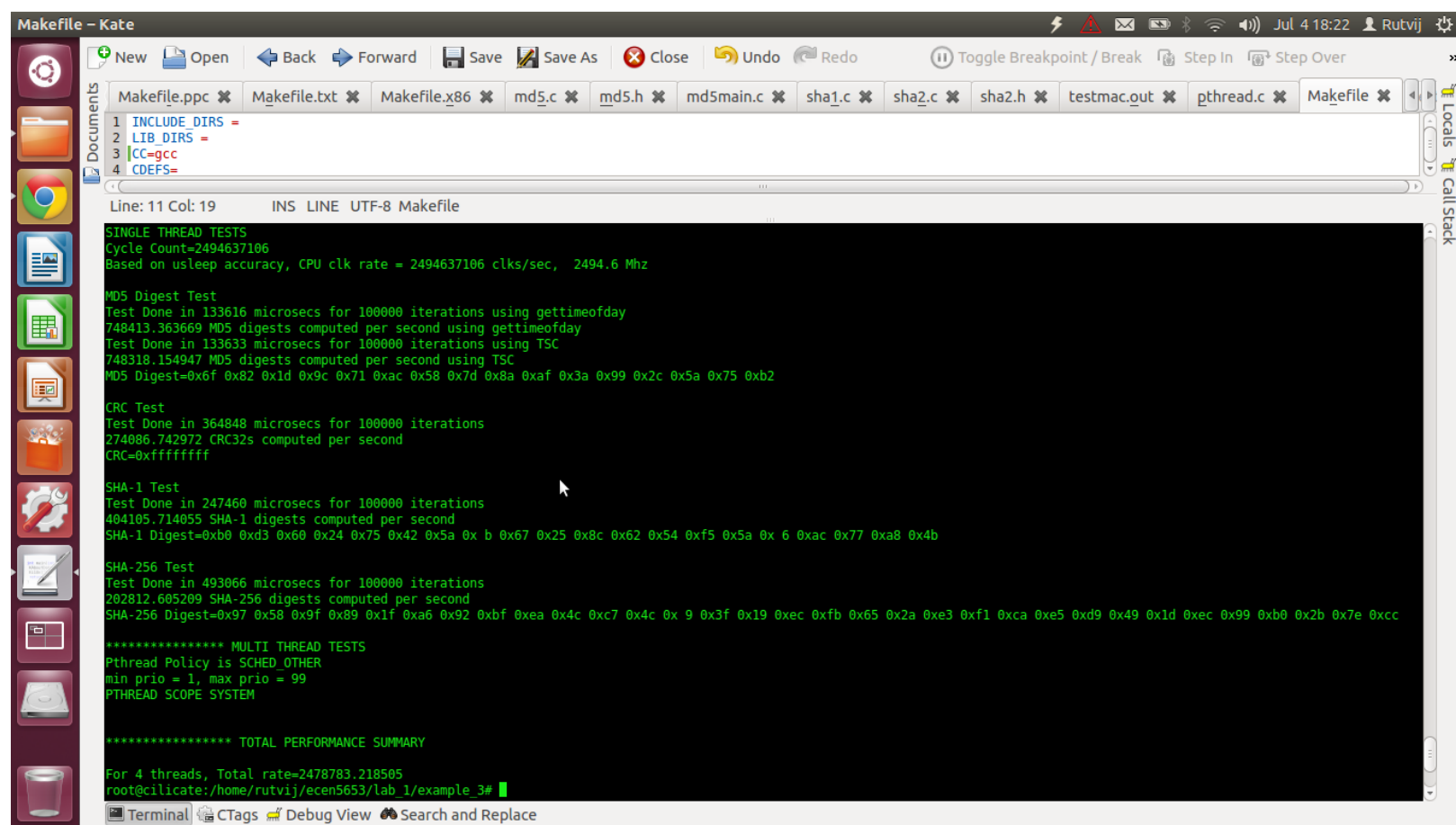
Codefile: testdigest.c

makefile: Makefile

- 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



```
Makefile - Kate
New Open Back Forward Save Save As Close Undo Redo Toggle Breakpoint / Break Step In Step Over
Makefile.ppc Makefile.txt Makefile.x86 md5.c md5.h md5main.c sha1.c sha2.c sha2.h testmac_out pthread.c Makefile
1 INCLUDE_DIRS =
2 LIB_DIRS =
3 CC=gcc
4 CDEFS=

Line: 11 Col: 19 INS LINE UTF-8 Makefile

SINGLE THREAD TESTS
Cycle Count=2494637106
Based on usleep accuracy, CPU clk rate = 2494637106 clks/sec, 2494.6 Mhz

MD5 Digest Test
Test Done in 133616 microsecs for 100000 iterations using gettimeofday
748413.363669 MD5 digests computed per second using gettimeofday
Test Done in 133633 microsecs for 100000 iterations using TSC
748318.154947 MD5 digests computed per second using TSC
MD5 Digest=0x6f 0x82 0x1d 0x9c 0x71 0xac 0x58 0x7d 0x8a 0xaf 0x3a 0x99 0x2c 0x5a 0x75 0xb2

CRC Test
Test Done in 364848 microsecs for 100000 iterations
274086.742972 CRC32s computed per second
CRC=0xffffffff

SHA-1 Test
Test Done in 247460 microsecs for 100000 iterations
484105.714055 SHA-1 digests computed per second
SHA-1 Digest=0xb0 0xd3 0x60 0x24 0x75 0x42 0x5a 0x b 0x67 0x25 0x8c 0x62 0x54 0xf5 0x5a 0x 6 0xac 0x77 0xa8 0x4b

SHA-256 Test
Test Done in 493066 microsecs for 100000 iterations
282812.605209 SHA-256 digests computed per second
SHA-256 Digest=0x97 0x58 0x9f 0x89 0x1f 0xa6 0x92 0xbf 0xea 0x4c 0xc7 0x4c 0x 9 0x3f 0x19 0xec 0xfb 0x65 0x2a 0xe3 0xf1 0xca 0xe5 0xd9 0x49 0x1d 0xec 0x99 0xb0 0x2b 0x7e 0xcc

***** MULTI THREAD TESTS
Pthread Policy is SCHED_OTHER
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM

***** TOTAL PERFORMANCE SUMMARY
For 4 threads, Total rate=2478783.218505
root@cilicate:/home/rutvij/ecen5653/lab_1/example_3#
```

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

The screenshot shows the Kate IDE interface with a terminal window displaying the output of a program. The terminal output includes performance metrics for single-threaded tests and a total performance summary for 4 threads.

```
Makefile - Kate
New Open Back Forward Save Save As Close Undo Redo Toggle Breakpoint / Break Step In Step Over
Makefile.ppc Makefile.txt Makefile.x86 md5.c md5.h md5main.c sha1.c sha2.c sha2.h testmac_out pthread.c Makefile
1 INCLUDE_DIRS =
2 LIB_DIRS =
3 CC=gcc
4 CDEFS=
Line: 28 Col: 30 INS LINE UTF-8 Makefile

SINGLE THREAD TESTS
Cycle Count=2494656504
Based on usleep accuracy, CPU clk rate = 2494656504 clks/sec, 2494.7 Mhz

MD5 Digest Test
Test Done in 250766012 microsecs for 100000 iterations using gettimeofday
398.778125 MD5 digests computed per second using gettimeofday
Test Done in 250799576 microsecs for 100000 iterations using TSC
398.724757 MD5 digests computed per second using TSC
MD5 Digest=0x64 0x87 0xda 0xf5 0x 1 0xe0 0x33 0xfe 0x83 0x67 0x 4 0x3f 0xf1 0x26 0x64 0x69

CRC Test
Test Done in 451134 microsecs for 100000 iterations
221663.629875 CRC32s computed per second
CRC=0xffffffff

SHA-1 Test
Test Done in 305840 microsecs for 100000 iterations
326968.349464 SHA-1 digests computed per second
SHA-1 Digest=0xb0 0xd3 0x60 0x24 0x75 0x42 0x5a 0x b 0x67 0x25 0x8c 0x62 0x54 0xf5 0x5a 0x 6 0xac 0x77 0xa8 0x4b

SHA-256 Test
Test Done in 599033 microsecs for 100000 iterations
166935.711388 SHA-256 digests computed per second
SHA-256 Digest=0x97 0x58 0x9f 0x89 0x1f 0xa6 0x92 0xbf 0xea 0x4c 0xc7 0x4c 0x 9 0x3f 0x19 0xec 0xfb 0x65 0x2a 0xe3 0xf1 0xca 0xe5 0xd9 0x49 0x1d 0xec 0x99 0xb0 0x2b 0x7e 0xcc

***** MULTI THREAD TESTS
Pthread Policy is SCHED_OTHER
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM

***** TOTAL PERFORMANCE SUMMARY
For 4 threads, Total rate=2437118.175042

Terminal CTags Debug View Search and Replace
```


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

```
testdigest_ntsc_concurrent.c - Kate
New Open Back Forward Save Save As Close Undo Redo Toggle Breakpoint / Break Step In Step Over
Makefile.x86 md5.c md5.h md5main.c sha1.c sha2.c sha2.h testmac_out Makefile pthread.c testdigest_ntsc_concurrent.c
299 startTSC = readTSC();
300 usleep(1000000);
301 stopTSC = readTSC();
302
Line: 192 Col: 63 INS LINE UTF-8 testdigest_ntsc_concurrent.c

SINGLE THREAD TESTS
Cycle Count=2494636557
Based on usleep accuracy, CPU clk rate = 2494636557 clks/sec, 2494.6 Mhz

MD5 Digest Test
Test Done in 262752921 microseconds for 100000 iterations using gettimeofday
380.585683 MD5 digests computed per second using gettimeofday
Test Done in 262788089 microseconds for 100000 iterations using TSC
380.534751 MD5 digests computed per second using TSC
MD5 Digest=0x64 0x87 0xda 0xf5 0x 1 0xe0 0x33 0xfe 0x83 0x67 0x 4 0x3f 0xf1 0x26 0x64 0x69

CRC Test
Test Done in 452314 microseconds for 100000 iterations
221085.352211 CRC32s computed per second
CRC=0xffffffff

SHA-1 Test
Test Done in 305172 microseconds for 100000 iterations
327684.060137 SHA-1 digests computed per second
SHA-1 Digest=0xb0 0xd3 0x60 0x24 0x75 0x42 0x5a 0x b 0x67 0x25 0x8c 0x62 0x54 0xf5 0x5a 0x 6 0xac 0x77 0xa8 0x4b

SHA-256 Test
Test Done in 599665 microseconds for 100000 iterations
166759.774207 SHA-256 digests computed per second
SHA-256 Digest=0x97 0x58 0x9f 0x89 0x1f 0xa6 0x92 0xbf 0xea 0x4c 0xc7 0x4c 0x 9 0x3f 0x19 0xec 0xfb 0x65 0x2a 0xe3 0xf1 0xca 0xe5 0xd9 0x49 0x1d 0xec 0x99 0xb0 0x2b 0x7e 0xcc

***** MULTI THREAD TESTS
Pthread Policy is SCHED_OTHER
min prio = 1, max prio = 99
PTHREAD SCOPE SYSTEM

***** TOTAL PERFORMANCE SUMMARY
For 4 threads, Total rate=3821.253786
```