CSE 438/598: Embedded Systems Programming

Device Drivers for Shared Message Queues

Profiling Report

Project 1

Author: Ankit Rathi

ASU ID: 1207543476

1 TABLE OF CONTENTS

2 INTRODUCTION

The purpose of this project is to implement a device driver for shared message queues. In this project I have two programs. First, is the Device Driver, which performs the basic functionality of en-queue and de-queue on a Circular Buffer. It makes use of the helper file (header file) to perform the functionalities. Driver is also used to calculate the time spent by the message in the queues. Second, is a test program to control the threads in the project and generate random messages of random length for sending over the device. The test programs uses 3 sender threads which generate data and add the data into the device queue "*bus_in_q*". Bus Daemon thread is used to read messages from "*bus_in_q*" device and add it into the respective receiver queues. i.e "*bus_out_q1*", "*bus_out_q2*", and "*bus_out_q3*". The messages that have been queued into the receiver buffers are then picked up by the corresponding receiver threads. The driver implements the functionality of queuing time by making use of the TSC Counter. And in the last part of the project, the code has been analyzed by making use of Linux perf tool to check for number of CPU cycles, number of instructions executed in user and kernel space and memory used by the code during its execution.

Perf is a profiler tool for Linux 2.6+ based systems that abstracts away CPU hardware differences in Linux performance measurements and presents a simple commandline interface. Perf is based on the perf_events interface exported by recent versions of the Linux kernel. This article demonstrates the perf tool through example runs. Output was obtained on a Ubuntu 11.04 system with kernel 2.6.38-8-generic results running on an HP 6710b with dual-core Intel Core2 T7100 CPU). For readability, some output is abbreviated using ellipsis ([...]).

3 PROFILING REPORT FOR DYNAMICALLY ALLOCATED MESSAGE BUFFERS

3.1 CPU CYCLES

The instruction cycle of CPU consists of:

1) fetch the next instruction to execute from the memory;
2) decode it;
3) run it.

This is known as the fetch-decode-execute cycle. My test CPU(Laptop) runs at 2.4GHz, meaning that it can run this cycle 2.4 billion times per second. Simpler CPUs, the instruction cycle is executed sequentially, i.e each instruction is processed completely before it begins to start with the next instruction. But, now a days CPUs, instruction cycle is executed simultaneously in parallel. The next instruction is fetched and being processed even before the previous instruction has finished executing. This has been made possible by techniques such as pipelining.

3.1.1 CPU Cycles in User Space

Command: *sudo perf stat -e cycles:u ./main_1.o*

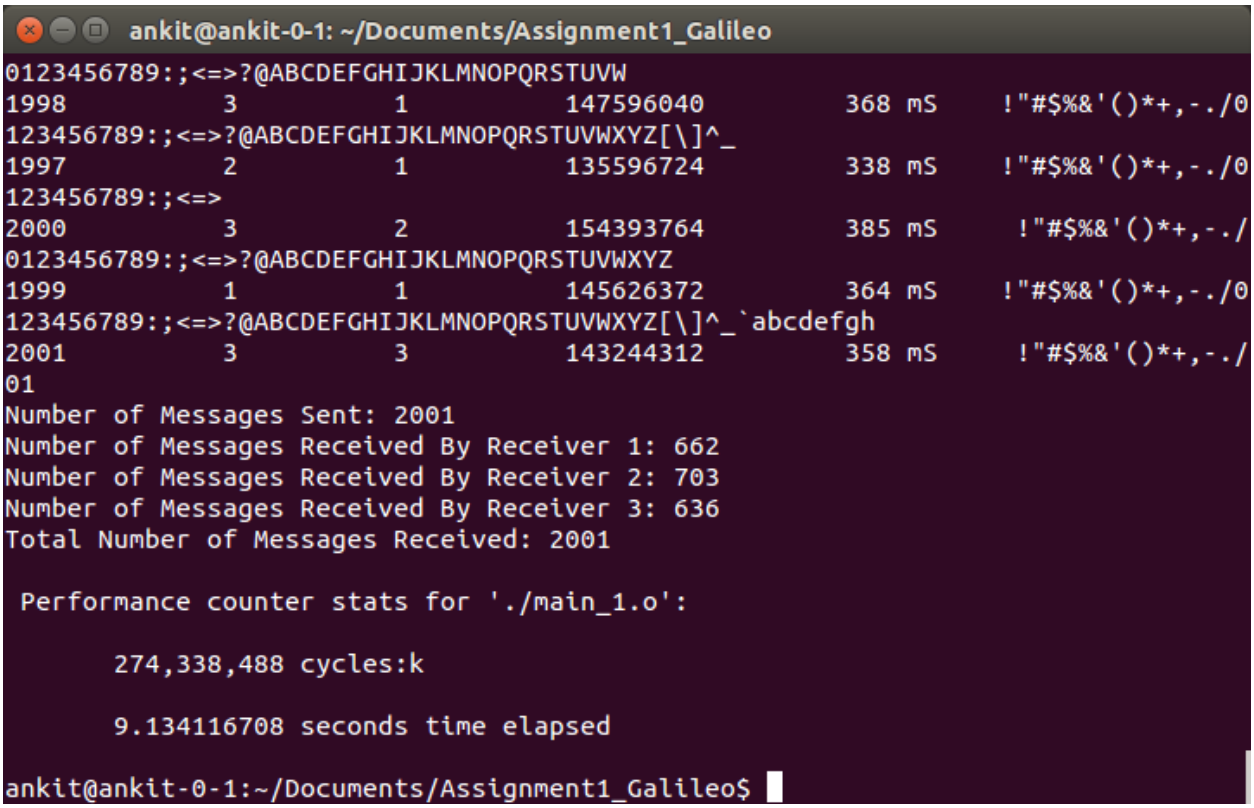
80,316,723 cycles are used by the user program.

```
ankit@ankit-0-1: ~/Documents/Assignment1_Galileo
2047          1          2          89619780         224 mS     !"#$%&'()*+,-.
2052          1          1          87603192         219 mS    !"#$%&'()*+,-./01
23456789:;<=>?@
2051          2          1          94564404         236 mS    !"#$%&'()*+,-./01
23456789
2050          3          2          115151184         287 mS     !"#$%&'()*+,-./
0123456789:;<=>?@ABC
2046          2          2          126806016         317 mS     !"#$%&'()*+,-./
0123456789:;<=>?@ABCDEFGHIJ
2053          1          2          117000792         292 mS     !"#$%&'()*+,-./
0123456789:;<=>?@AB
Number of Messages Sent: 2053
Number of Messages Received By Receiver 1: 681
Number of Messages Received By Receiver 2: 682
Number of Messages Received By Receiver 3: 690
Total Number of Messages Received: 2053

 Performance counter stats for './main_1.o':

        80,316,723 cycles:u

       9.422337929 seconds time elapsed

ankit@ankit-0-1:~/Documents/Assignment1_Galileo$
```

3.1.2 CPU Cycles in Kernel Space

Command: *sudo perf stat -e cycles:k ./main_1.o*

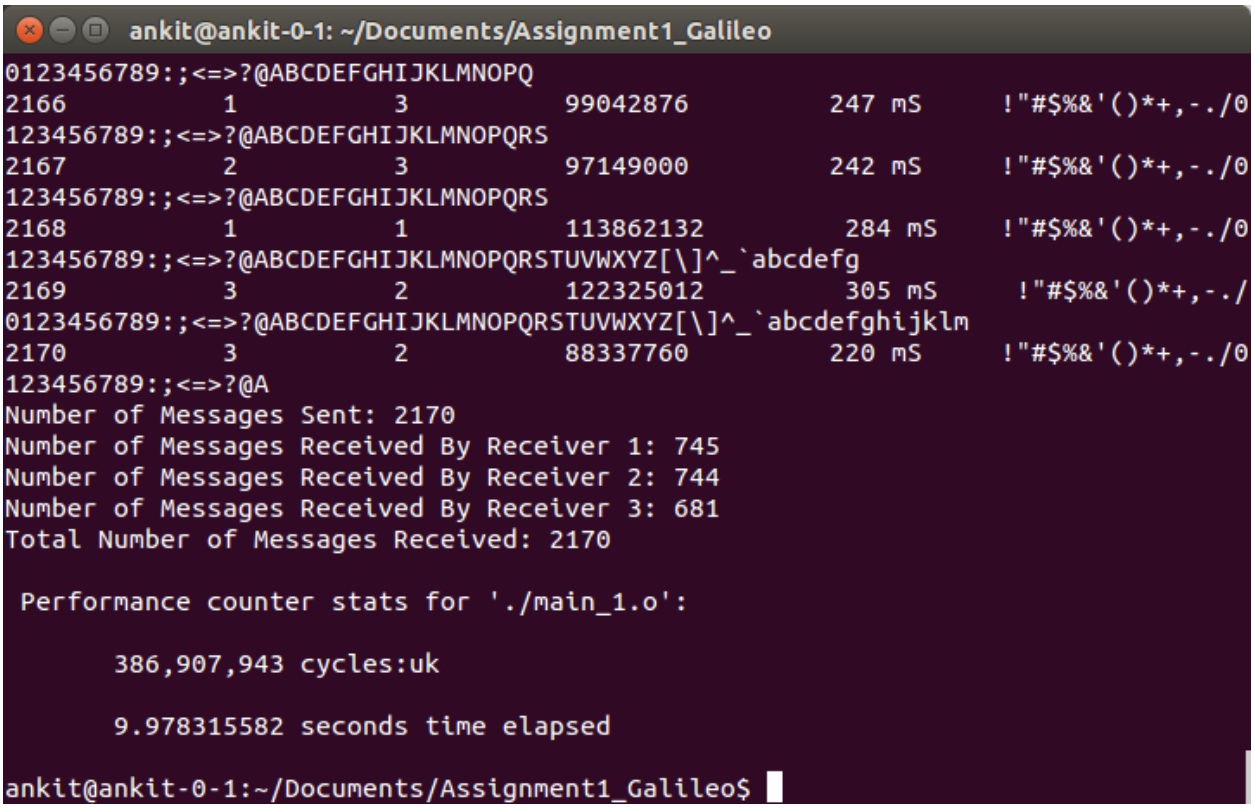
274,338,448 cycles are used by the kernel program.



3.1.3 CPU Cycles in User and Kernel Space

Command: *sudo perf stat -e cycles:uk ./main_1.o*

386,907,943 cycles are used by the user and kernel program combined.



3.2 NUMBER OF INSTRUCTIONS

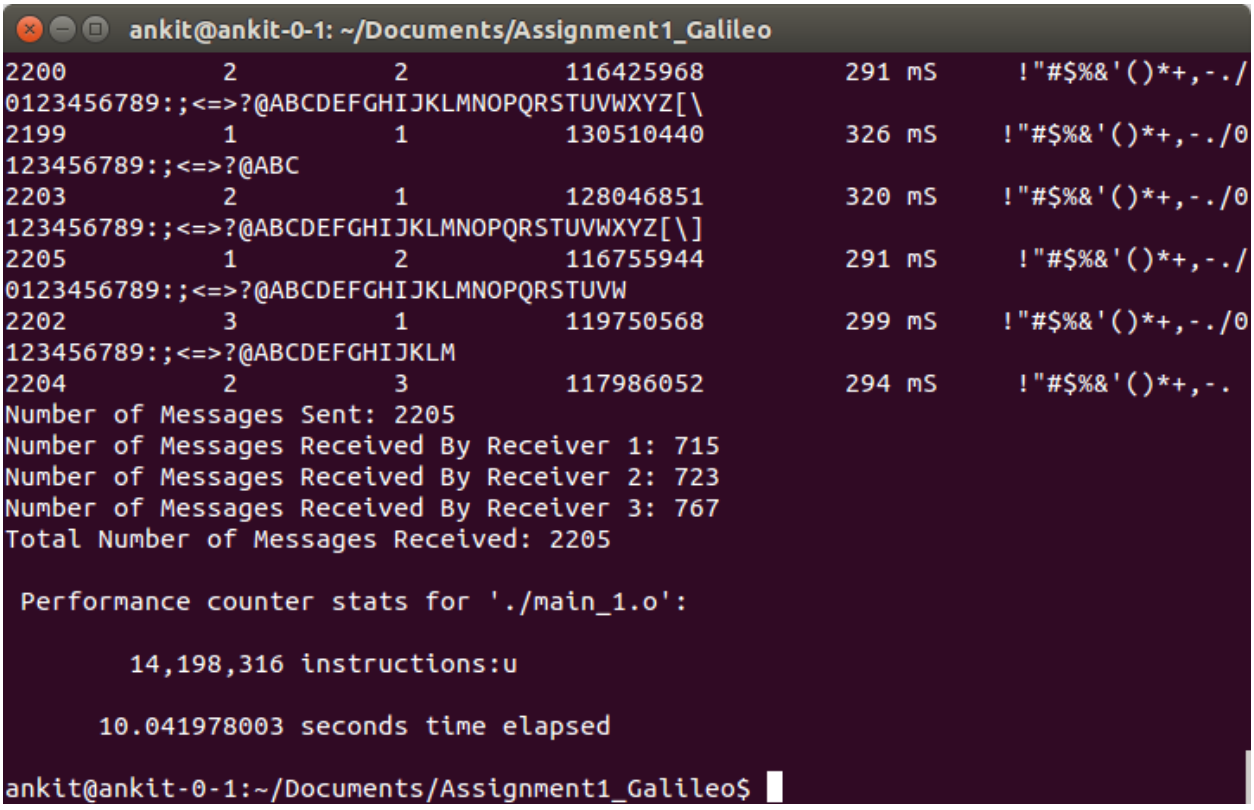
Here, I have analyzed how much instructions are executed in total for the program in user and kernel space program.

3.2.1 Number of Instructions in User Space

Command: *sudo perf stat -e instructions:u ./main_1.o*

Nearly, 14 million instructions are executed by the user space program.

Number of Instructions / Clock Cycles = 14,198,316 / 80,316,723 = 0.17 instructions per cycle

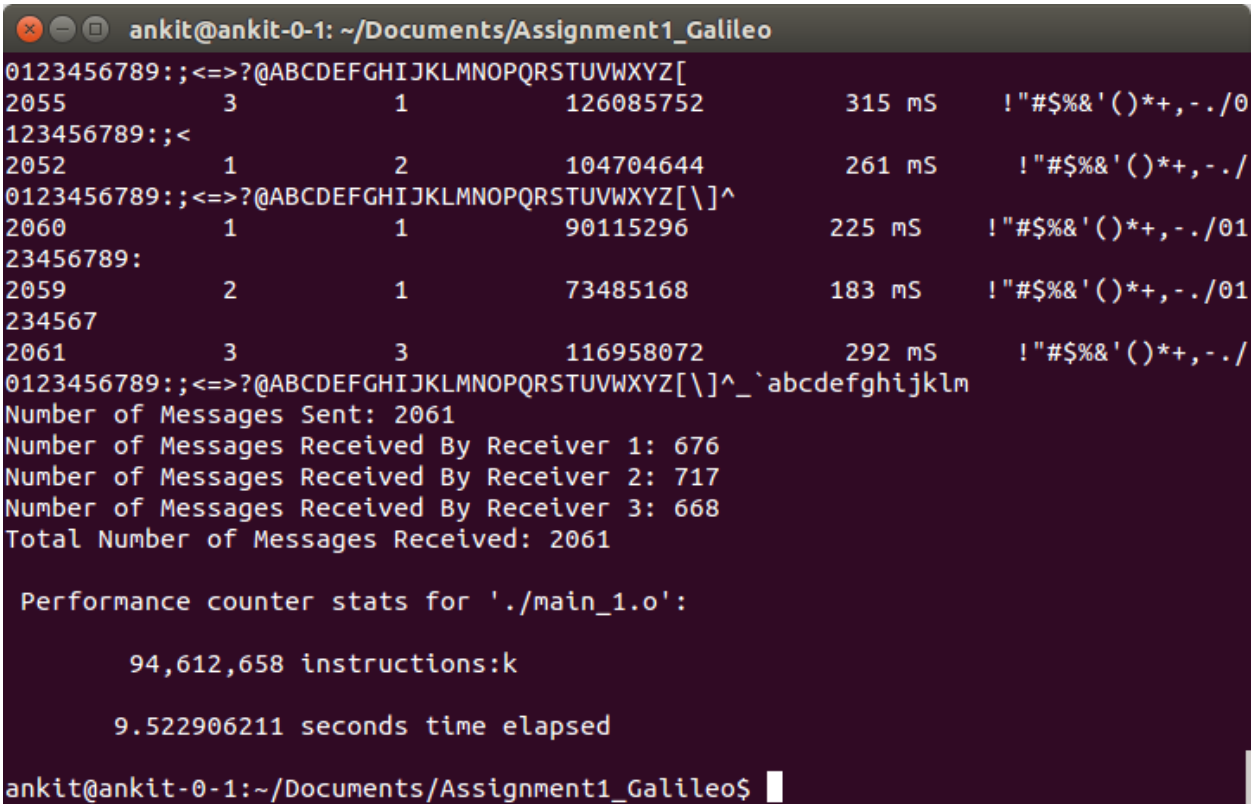


3.2.2 Number of Instructions in Kernel Space

Command: *sudo perf stat -e instructions:k ./main_1.o*

Nearly, 94 million instructions are executed by the kernel space program.

Number of Instructions / Clock Cycles = 94,612,658 / 274,338,448 = 0.344 instructions per cycle

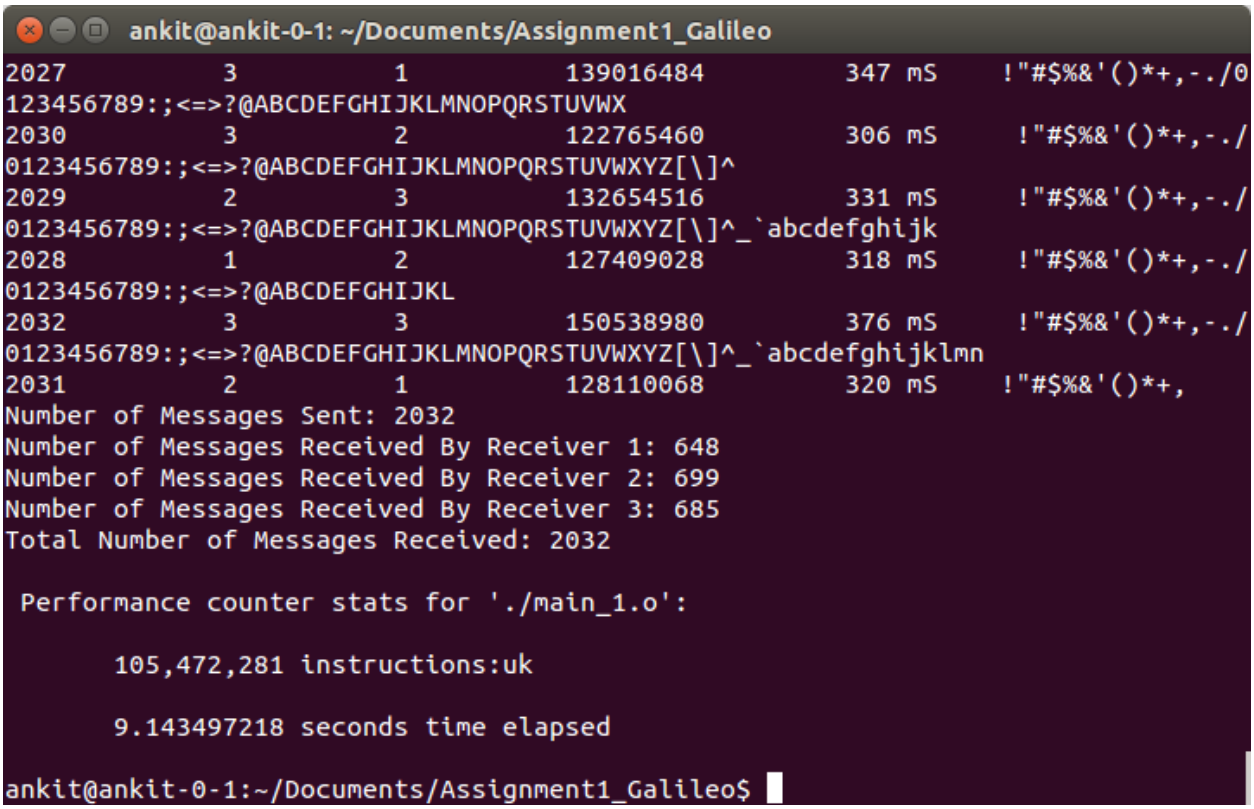


3.2.3 Number of Instructions in User and Kernel Space

Command: *sudo perf stat -e instructions:uk ./main_1.o*

Nearly, 105 million instructions are executed by the user and kernel space program combined.

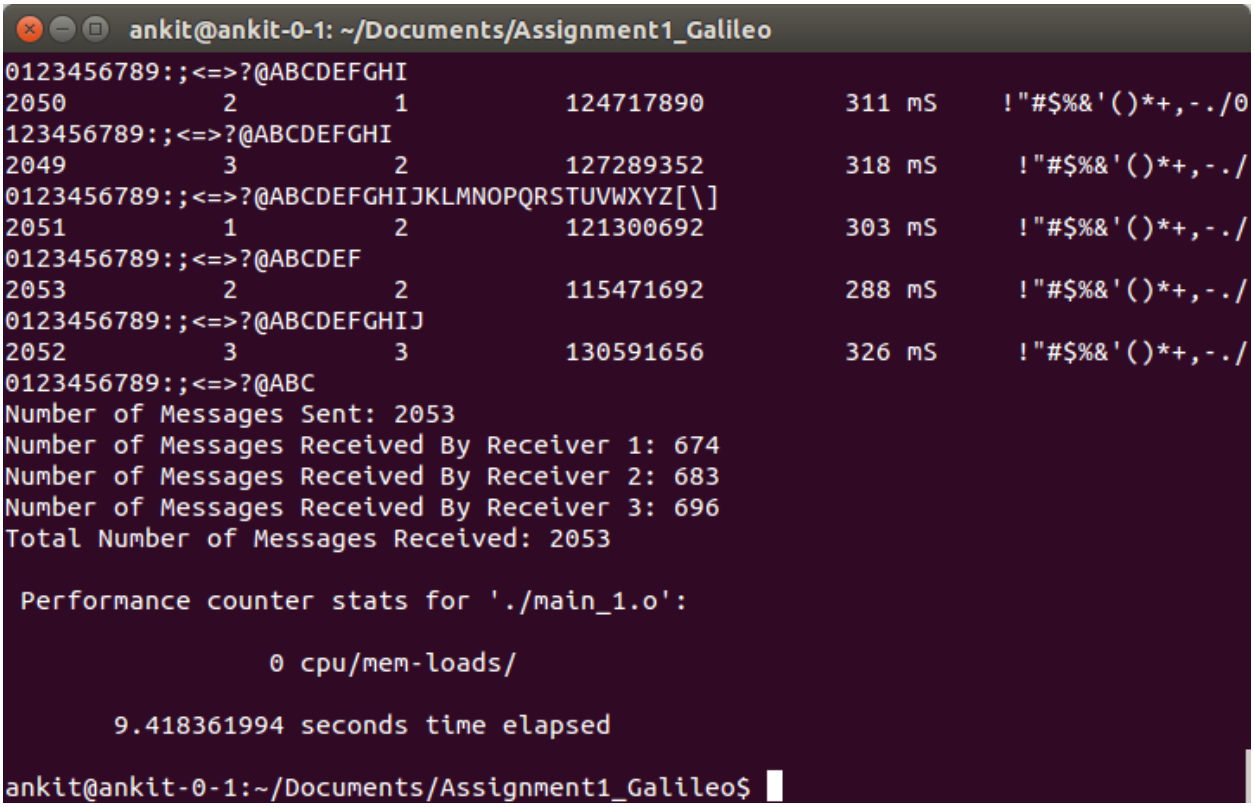
Number of Instructions / Clock Cycles = 105,472,281 / 386,907,943 = 0.272 instructions per cycle



3.3 MEMORY USAGE

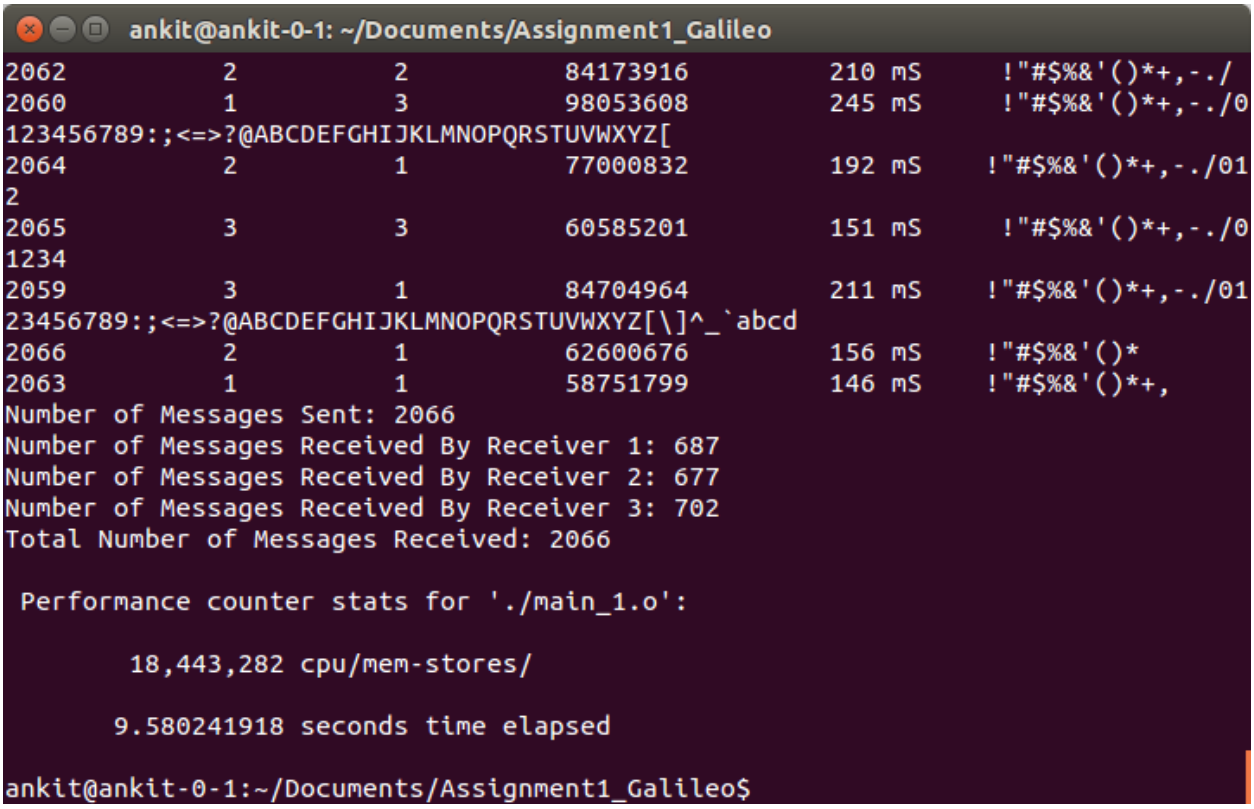
3.3.1 Memory Loads

Command: *sudo perf stat -e mem-loads ./main_1.o*



3.3.2 Memory Stores

Command: *sudo perf stat -e mem-stores ./main_1.o*



4 PROFILING REPORT FOR STATICALLY ALLOCATED MESSAGE BUFFERS

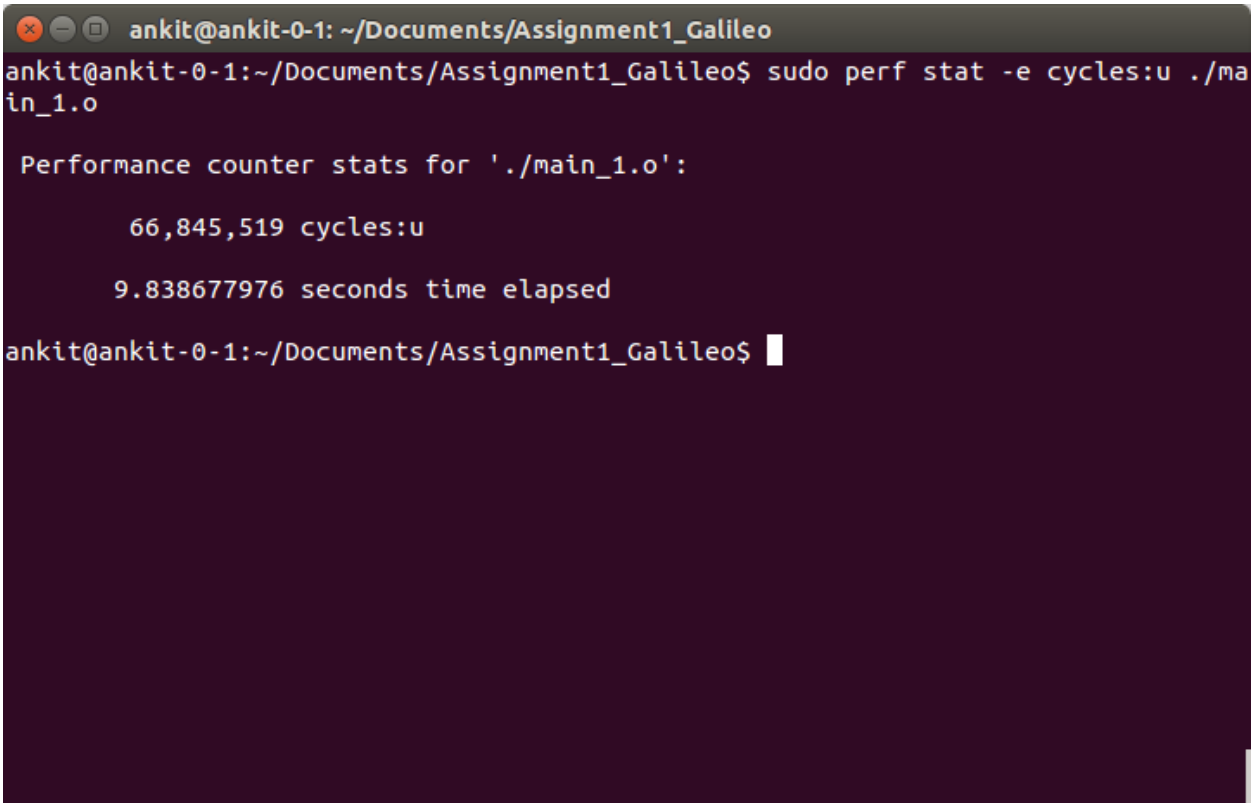
This is the second way in which the code in CircularBuffer.h has been designed. Here the message buffers have been allocated for all 10 buffers at one go.

4.1 3.1 CPU CYCLES

4.1.1 CPU Cycles in User Space

Command: *sudo perf stat -e cycles:u ./main_1.o*

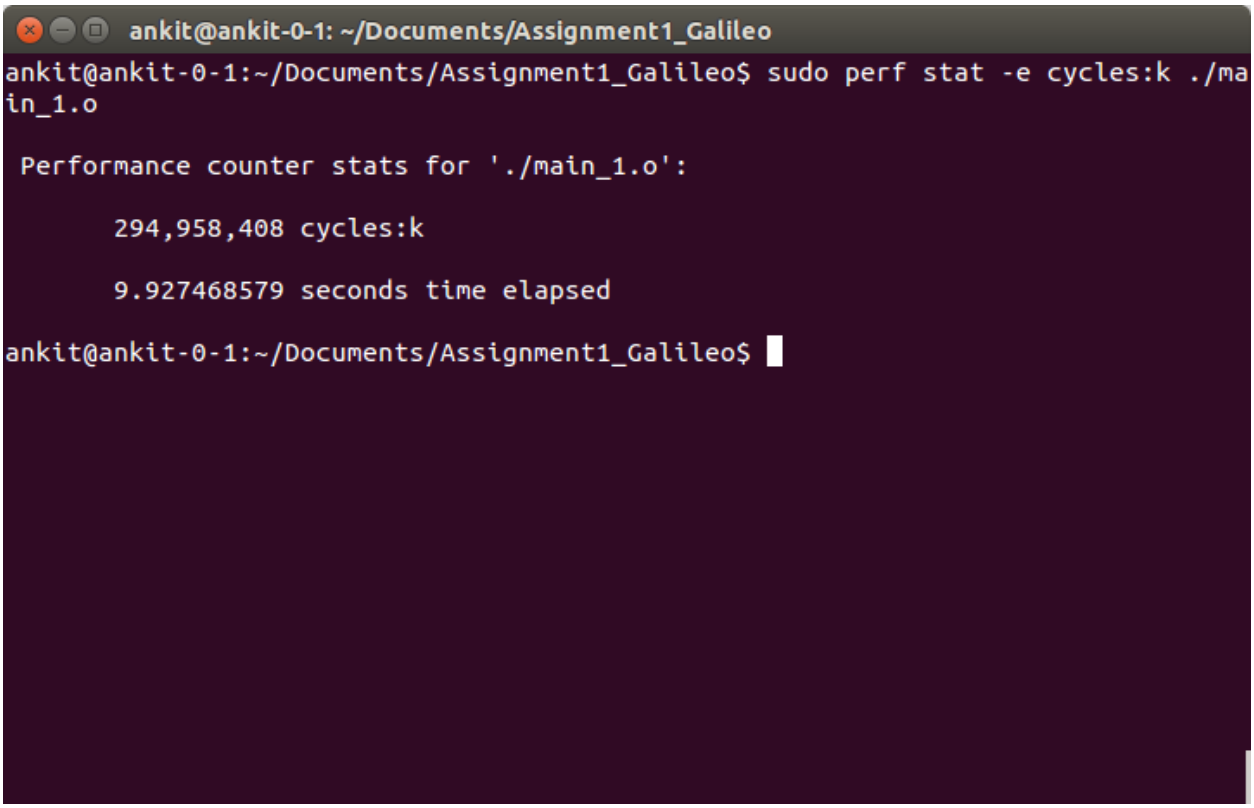
66,845,519 cycles are used by the user program.



4.1.2 CPU Cycles in Kernel Space

Command: *sudo perf stat -e cycles:k ./main_1.o*

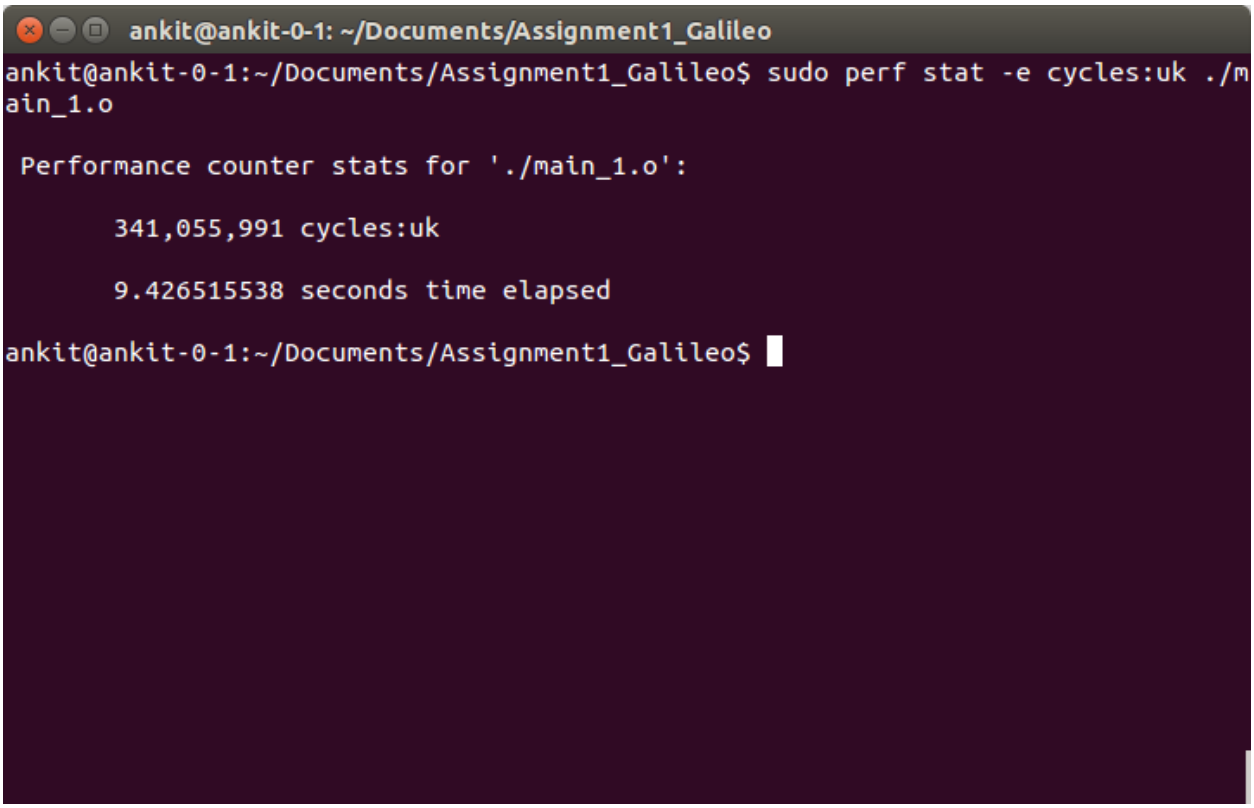
294,958,408 cycles are used by the kernel program.



4.1.3 CPU Cycles in User and Kernel Space

Command: *sudo perf stat -e cycles:uk ./main_1.o*

341,055,991 cycles are used by the user and kernel program combined.



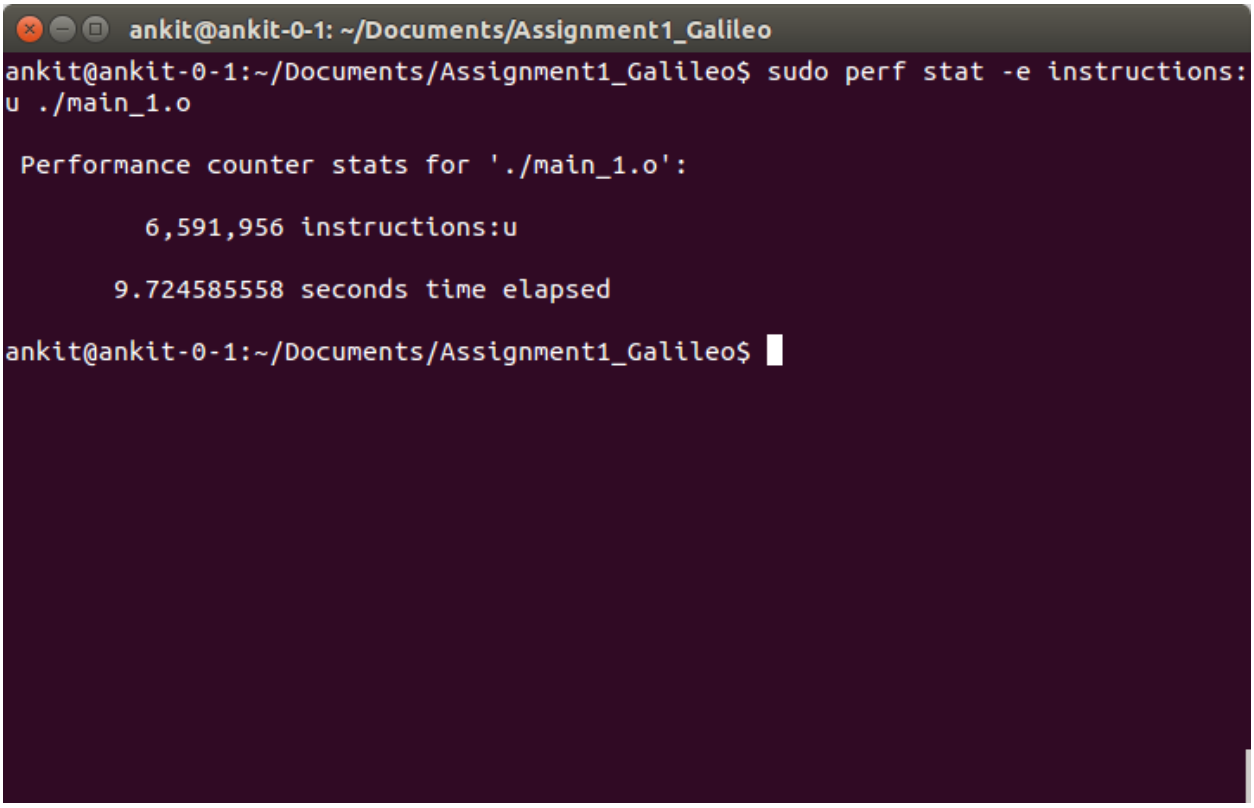
4.2 NUMBER OF INSTRUCTIONS

4.2.1 Number of Instructions in User Space

Command: *sudo perf stat -e instructions:u ./main_1.o*

Nearly, 6.5 million instructions are executed by the space program.

Number of Instructions / Clock Cycles = 6,591,956 / 66,845,519 = 0.098 instructions per cycle

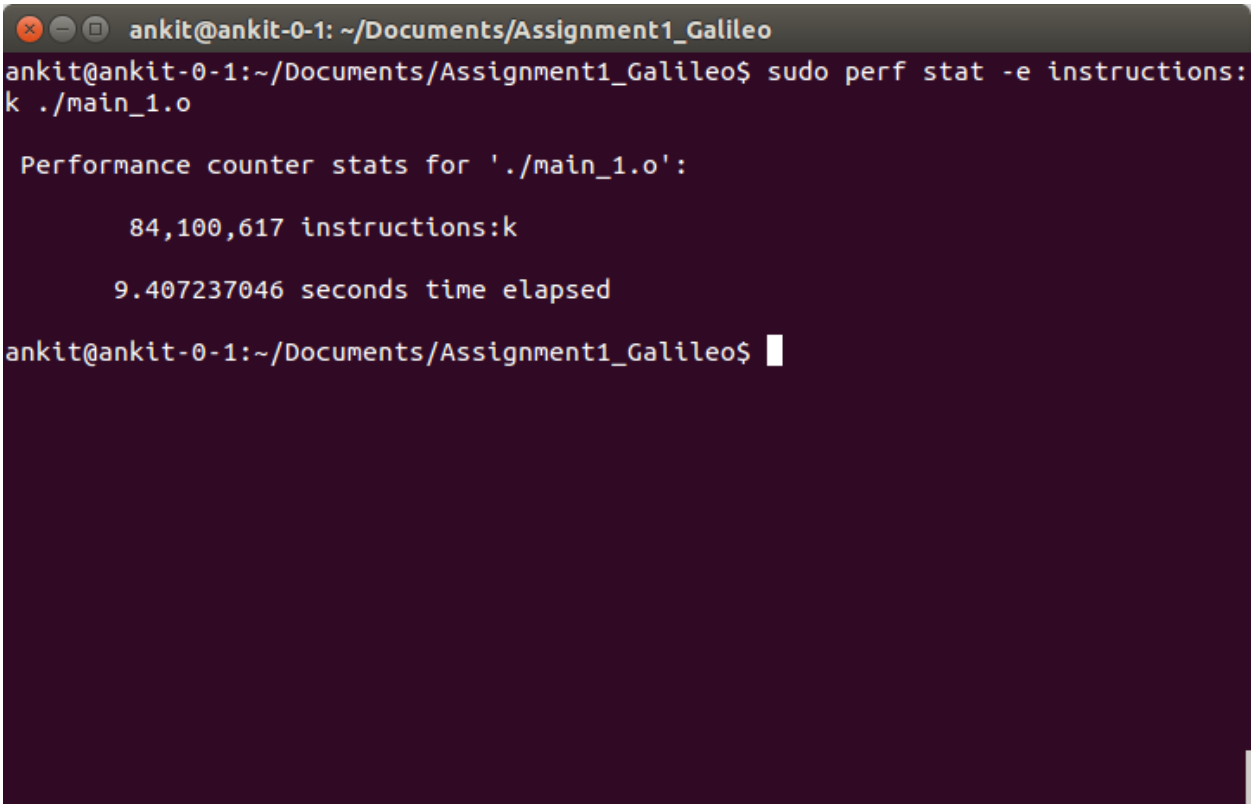


4.2.2 Number of Instructions in Kernel Space

Command: *sudo perf stat -e instructions:k ./main_1.o*

Nearly, 84 million instructions are executed by the kernel space program.

Number of Instructions / Clock Cycles = 84,100,617 / 294,958,408 = 0.285 instructions per cycle

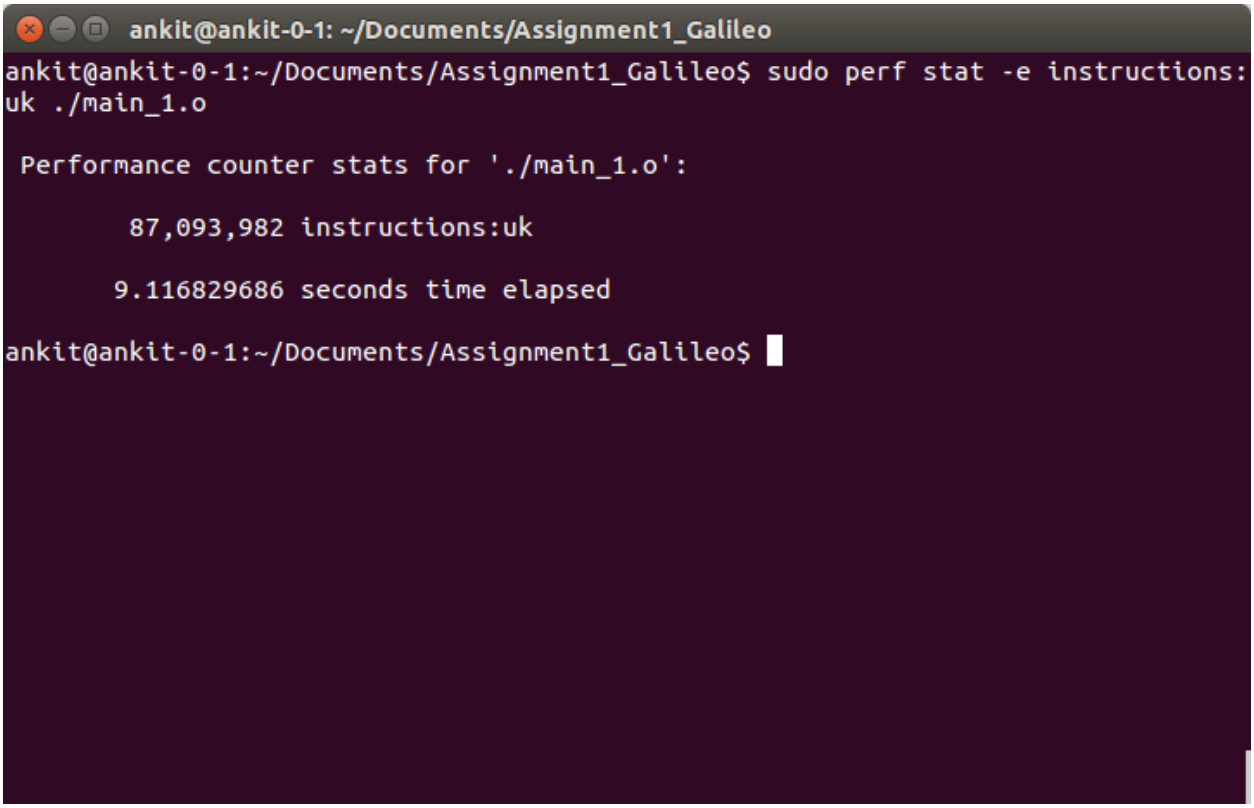


4.2.3 Number of Instructions in User and Kernel Space

Command: *sudo perf stat -e instructions:uk ./main_1.o*

Nearly, 87 million instructions are executed by the user and kernel space program combined.

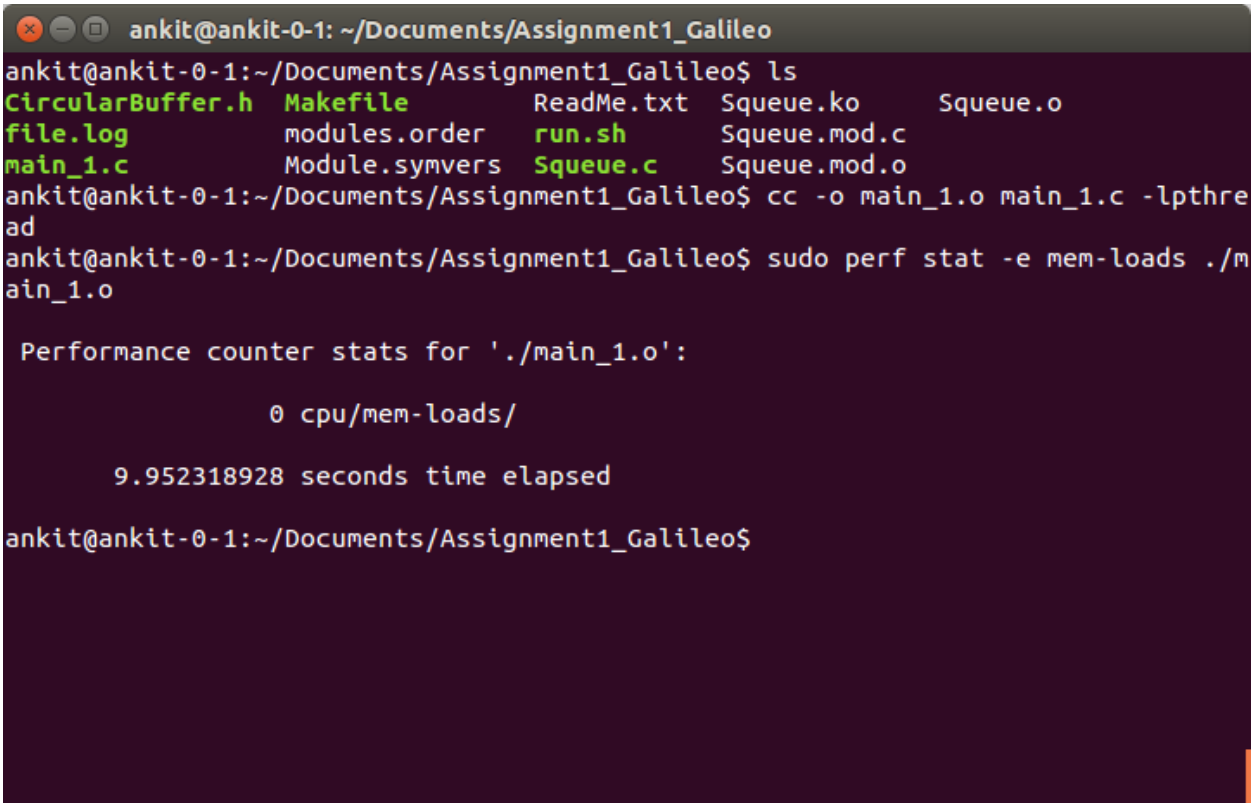
Number of Instructions / Clock Cycles = 87,093,982 / 341,055,991 = 0.255 instructions per cycle



4.3 MEMORY USAGE

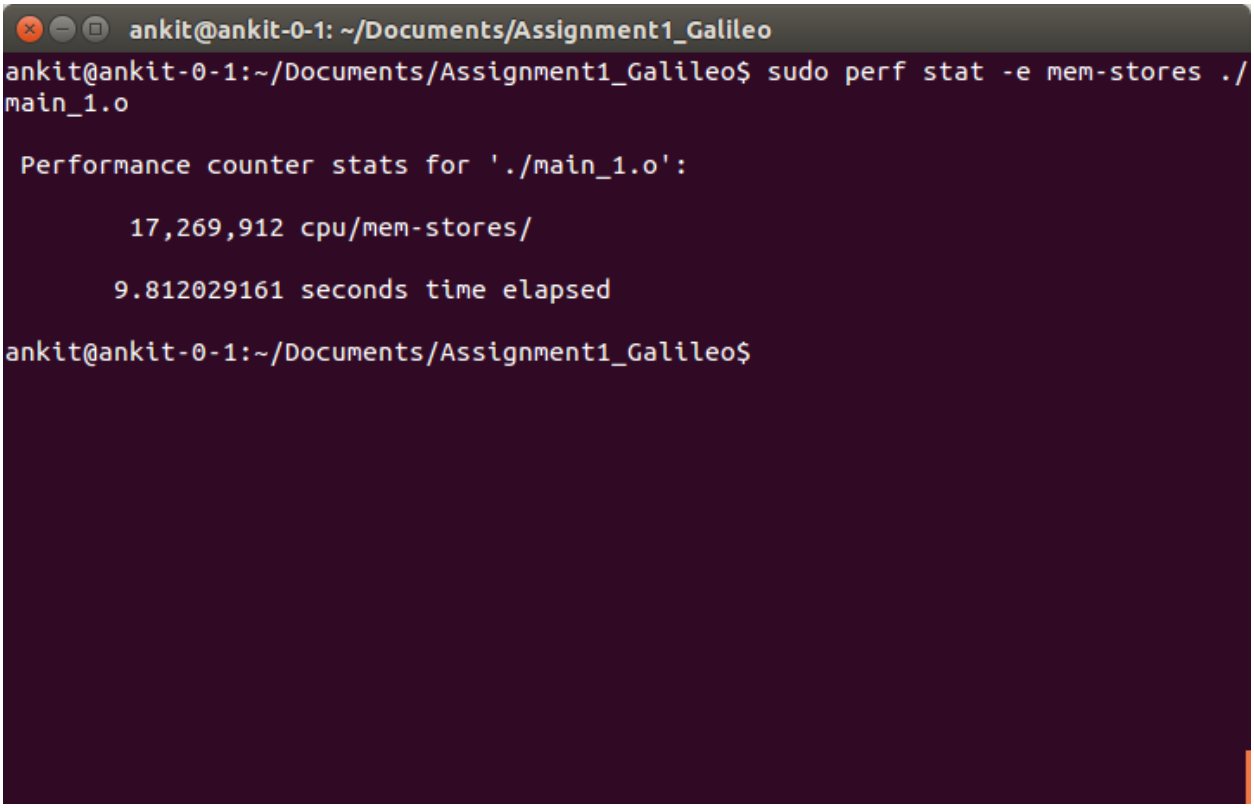
4.3.1 Memory Loads

Command: *sudo perf stat -e mem-loads ./main_1.o*



4.3.2 Memory Stores

Command: *sudo perf stat -e mem-stores ./main_1.o*



5 BIBLIOGRAPHY

1. https://perf.wiki.kernel.org/index.php/Tutorial
2. http://paolobernardi.wordpress.com/2012/08/07/playing-around-with-perf/
3. http://en.wikipedia.org/wiki/Instruction_cycle