**Q1 - Written question (5 marks)**

**a)** Use the time utility to time palindrome.py and slow-pali.cpp on files t4.txt and t3.txt. Copy/paste the output of time from the terminal window into your report.

**Time – t3.txt (palindrome.py)**

Longest palindrome: \_\_\_o.O.o\_\_\_

real 0m0.024s

user 0m0.013s

sys 0m0.007s

**Time – t3.txt (slow-pali.cpp)**

Longest palindrome:

real 0m0.006s

user 0m0.000s

sys 0m0.003s

**Time – t4.txt (palindrome.py)**

Longest palindrome: redder

real 0m0.222s

user 0m0.207s

sys 0m0.011s

**Time – t4.txt (slow-pali.cpp)**

Longest palindrome: redder

real 0m4.341s

user 0m1.621s

sys 0m2.714s

**b)** How much time did the C++ and python programs spend in kernel vs user mode?

The python programs spent 0.007s for t3.txt and 0.011s for t4.txt in kernel mode. They also spent 0.013s for t3.txt and 0.207s for t4.txt in user mode.

The C++ programs spent 0.003s for t3.txt and 2.714s for t4.txt in kernel mode. They also spent 0.000s for t3.txt and 1.621s for t4.txt in user mode.

**c)** Run ‘strace -c’ on palindrome.py and slow-pali.cpp on t4.txt and t3.txt. Copy/paste the output from the terminal window into your report.

**Strace – t3.txt (palindrome.py)**

Longest palindrome: \_\_\_o.O.o\_\_\_

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

22.08 0.000208 0 253 39 newfstatat

15.39 0.000145 9 16 getdents64

12.31 0.000116 19 6 4 execve

10.40 0.000098 1 84 18 openat

7.32 0.000069 1 46 mmap

6.05 0.000057 0 **80 read**

5.94 0.000056 0 66 rt\_sigaction

4.78 0.000045 0 69 close

3.40 0.000032 0 45 40 ioctl

2.76 0.000026 0 70 2 lseek

2.65 0.000025 5 5 munmap

1.80 0.000017 2 8 mprotect

0.96 0.000009 0 12 brk

0.85 0.000008 0 9 pread64

0.64 0.000006 1 4 3 readlink

0.42 0.000004 1 4 2 arch\_prctl

0.32 0.000003 1 2 2 access

0.32 0.000003 1 2 getcwd

0.32 0.000003 3 1 sysinfo

0.32 0.000003 1 3 getrandom

0.21 0.000002 1 2 set\_tid\_address

0.21 0.000002 1 2 prlimit64

0.11 0.000001 1 1 write

0.11 0.000001 1 1 fcntl

0.11 0.000001 0 2 futex

0.11 0.000001 0 2 set\_robust\_list

0.11 0.000001 0 2 rseq

0.00 0.000000 0 3 dup

0.00 0.000000 0 1 getuid

0.00 0.000000 0 1 getgid

0.00 0.000000 0 1 geteuid

0.00 0.000000 0 1 getegid

------ ----------- ----------- --------- --------- ----------------

100.00 0.000942 1 804 110 total

**Strace – t4.txt (palindrome.py)**

Longest palindrome: redder

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

20.17 0.000381 63 6 4 execve

17.68 0.000334 1 253 39 newfstatat

16.04 0.000303 18 16 getdents64

13.29 0.000251 2 84 18 openat

8.21 0.000155 3 46 mmap

6.35 0.000120 0 **784 read**

5.61 0.000106 1 69 close

2.54 0.000048 0 70 2 lseek

1.96 0.000037 0 45 40 ioctl

1.75 0.000033 4 8 mprotect

1.32 0.000025 5 5 munmap

1.06 0.000020 0 26 brk

0.90 0.000017 1 9 pread64

0.42 0.000008 4 2 2 access

0.42 0.000008 2 4 2 arch\_prctl

0.32 0.000006 2 3 getrandom

0.21 0.000004 0 66 rt\_sigaction

0.21 0.000004 2 2 getcwd

0.21 0.000004 1 4 3 readlink

0.21 0.000004 2 2 set\_robust\_list

0.21 0.000004 2 2 prlimit64

0.16 0.000003 1 2 futex

0.16 0.000003 1 2 rseq

0.11 0.000002 2 1 getuid

0.11 0.000002 2 1 geteuid

0.11 0.000002 2 1 getegid

0.11 0.000002 1 2 set\_tid\_address

0.05 0.000001 0 3 dup

0.05 0.000001 1 1 fcntl

0.05 0.000001 1 1 getgid

0.00 0.000000 0 1 write

0.00 0.000000 0 1 sysinfo

------ ----------- ----------- --------- --------- ----------------

100.00 0.001889 1 1522 110 total

**Strace – t3.txt (slow-pali.cpp)**

Longest palindrome:

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

50.36 0.000070 3 23 mmap

10.79 0.000015 2 7 mprotect

8.63 0.000012 2 5 openat

5.04 0.000007 1 **5 read**

5.04 0.000007 1 5 pread64

5.04 0.000007 1 6 newfstatat

3.60 0.000005 1 5 close

2.88 0.000004 4 1 munmap

1.44 0.000002 2 1 write

1.44 0.000002 0 3 brk

1.44 0.000002 2 1 set\_tid\_address

1.44 0.000002 2 1 set\_robust\_list

0.72 0.000001 0 2 1 arch\_prctl

0.72 0.000001 1 1 prlimit64

0.72 0.000001 1 1 getrandom

0.72 0.000001 1 1 rseq

0.00 0.000000 0 1 1 access

0.00 0.000000 0 1 execve

------ ----------- ----------- --------- --------- ----------------

100.00 0.000139 1 70 2 total

**Strace – t4.txt (slow-pali.cpp)**

Longest palindrome: redder

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

100.00 8.560619 1 **5767198 read**

0.00 0.000069 3 23 mmap

0.00 0.000030 6 5 openat

0.00 0.000019 2 7 mprotect

0.00 0.000011 1 6 newfstatat

0.00 0.000008 1 5 pread64

0.00 0.000005 1 5 close

0.00 0.000005 5 1 munmap

0.00 0.000004 4 1 write

0.00 0.000002 0 3 brk

0.00 0.000002 1 2 1 arch\_prctl

0.00 0.000001 1 1 set\_tid\_address

0.00 0.000001 1 1 set\_robust\_list

0.00 0.000001 1 1 getrandom

0.00 0.000001 1 1 rseq

0.00 0.000000 0 1 1 access

0.00 0.000000 0 1 execve

0.00 0.000000 0 1 prlimit64

------ ----------- ----------- --------- --------- ----------------

100.00 8.560778 1 5767263 2 total

**d)** When compared to the C++ code, why is the python program faster on some inputs, and slower on others? Try to justify your answers using the results you obtained above.

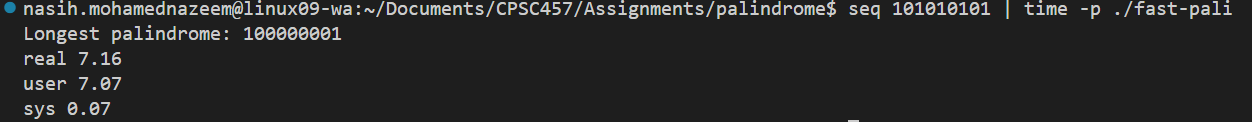
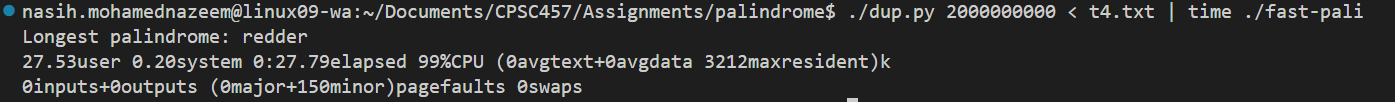
In t3.txt, C++ was faster due to how it compiles a small text file. Python also uses a garbage collector whereas C++ requires you to not include any garbage in the code. Python has a few delays due to how it executes code.

When we look at t4.txt, C++ had more delays trying to read through each iteration whilst Python was much faster.

We can see that when the code is much more simple (file was empty), C++ version has an advantage over Python, but when the task is a bit more daunting, Python is more efficient due to its optimization.

**Q2 - Programming question (15 marks)**

NICE!



**Q3 - Written question (5 marks)**

**a)**

**Time – t3.txt (fast-pali.cpp)**

**Text

Description automatically generated**

**Time – t4.txt (fast-pali.cpp)**

**Text

Description automatically generated**

**Strace – t3.txt (fast-pali.cpp)**

**Text

Description automatically generated**

**Strace – t4.txt (fast-pali.cpp)**

Text

Description automatically generated

**b)**

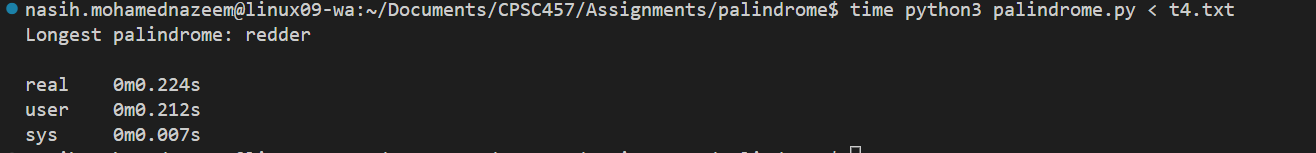
A screenshot of a computer screen

Description automatically generated

My fast-pali is much faster than slow-pali due to the changes on how I read each character. In slow-pali. In fast-pali we use a char array to act as a buffer for the characters being read. This buffer array will hold 1MB of data, and replenish whenever reset until EOF. In slow-pali we would read through each word without any buffers causing much delay in between each read. If we collect those reads and place them in an array, this will save us much more time than having to call read for every single character.

**c)**

Text

Description automatically generated 

This shows that fast-pali is much faster than palindrome.py. This is due to the compilation of C++ compared to how python executes the code, where there are a lot of optimizations.