**ASSIGNMENT 1 – CPSC 457**

**SOLUTION 1:**

***1a****.* $ time python3 palindrome.py < t3.txt

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

real 0m0.024s

user 0m0.019s

sys 0m0.005s

$ time ./slow-pali < t3.txt

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

real 0m0.003s

user 0m0.001s

sys 0m0.002s

$ time python3 palindrome.py < t4.txt

Longest palindrome: redder

real 0m0.309s

user 0m0.300s

sys 0m0.007s

$ time ./slow-pali < t4.txt

Longest palindrome: redder

real 0m2.859s

user 0m1.451s

sys 0m1.403s

***1b****.* The C++ file for **t3.txt** spent *0.002 sec in kernel mode* performing system calls and *0.001 sec in user mode* while the python file spent *0.005 sec in kernel mode* and *0.019 sec in user mode*. The C++ spent *1.403 sec in kernel mode* and *1.451 in user mode* for **t4.txt,** while python spent *0.007 sec in kernel mode* and *0.300 sec in user mode***.**

***1c****.* The python is faster on t4.txt, because it makes significantly less number of system calls hence spending very less time in kernel mode. Which means the python code reads more bytes at a time, reducing the number of times the read function is called, while the C++ reads one character and thus calling read function as many times as there are characters in the file. The python is slower on t3.txt, because it has to first run an interpreter and then the program, unlike C++ which is compiled and run separately. Also when the file is small, the time interpreter takes has significant impact, but when the file size is big, the interpreting time is insignificant.

**Solution 3:**

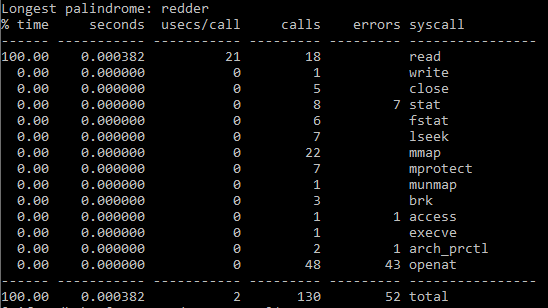
**Results:**

$ time ./fast-pali < t4.txt

Longest palindrome: redder

real 0m0.097s

user 0m0.095s

sys 0m0.002s

Strace (-c) result:

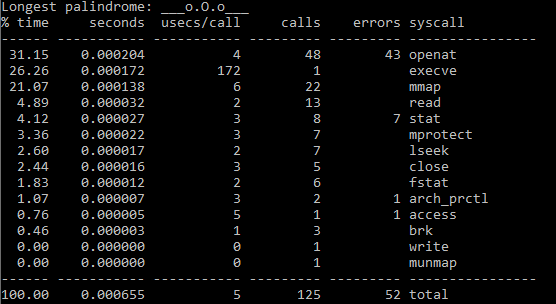
$ time ./fast-pali < t3.txt

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

real 0m0.003s

user 0m0.002s

sys 0m0.001s

Strace –c results:

***3(a)*** *For t3.txt***:** For slow-pali, the fast-pali and slow-pali seems to take almost the same amount of time. Slow-pali makes 50 read calls vs fast-pali, also there are other system calls made. In total slow-pali and fast-pali makes 112 calls in addition to read calls. So the fast-pali spends less time in kernel mode, but it spends more time in the user mode and slow-pali spends less time in user and more in kernel. So maybe because of the implementation and accessing and changing variable’s values, fast-pali spends more time in user mode and since slow-pali has less variable to handle, it spends less in user mode.

*For t4.txt:*Fast-pali is really faster than the slow-pali program. Slow-pali reads every byte one by one, so it makes as many read call as there are characters. To be precise, slow-pali calls the read system call 5767205 times vs 18 times in our fast-pali program. Since so many system calls are made it takes longer time and thus slow-pali is slower than fast-pali.

Overall, the fast pali will probably be faster in big files (because of less system calls made), but I believe in shorter files the slow-pali will be faster (because of its implementation).

***3(b)*** *For t3.txt:*Again the fast-pali is faster than palindrome.py. This is because Python has to run an interpreter first and then run. And also it makes more system calls where there are 84 read calls vs 13 read calls of fast-pali and the total system calls made in python was 835 while fast-pali makes system calls only 125 times. So again because fast-pali makes less system calls and therefore is faster.

*For t4.txt:*This program is faster than the python firstly because first the interpreter is run for python and secondly it makes significantly a lot more read calls (i.e. 788) compared to 18 read calls of this program. Since more system calls are made, it is slower than this fast-pali program.

Overall, in almost all the cases fast-pali will be faster than the python version (manily due to less system calls being made in fast-pali).