CPSC 457 – Spring 2022
Assignment 1 Questions

Instructor: Pavol Federl
TA: Haiyang He
Tutorial Section: T02

Student Name: Danielle Jourdain
UCID: 30114566

Question 1

a) *Timing results of using 'time' utility*

Code Used	**Text File Input**	**Terminal Output**
palindrome.py	t3.txt	Longest palindrome: ___o.O.o___ real 0m0.028s user 0m0.017s sys 0m0.006s
	t4.txt	Longest palindrome: redder real 0m0.260s user 0m0.245s sys 0m0.012s
slow-pali.cpp	t3.txt	Longest palindrome: ___o.O.o___ real 0m0.008s user 0m0.003s sys 0m0.003s
	t4.txt	Longest palindrome: redder real 0m2.721s user 0m1.253s sys 0m1.464s

b) For t3.txt, the C++ program spent about half of its time in user mode. This is because the "user" and "sys" times from the time utility make up about half of the "real" time given. The rest of the time is likely spent switching in between modes. For the python program, much more time is spent in user mode, between 60% and 90%. Once again, this can be calculated from the "real", "sys", and "user" times given by the time utility in part A. The rest of the time for the python program was most likely spent switching between user and kernel mode.

c) *Timing results of using 'strace -c' utility*

Code Used	**Text File Input**	**Terminal Output**
palindrome.py	t3.txt	Longest palindrome: ___o.O.o___ % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 26.35 0.000459 1 336 50 newfstatat 23.19 0.000404 2 196 113 openat 9.87 0.000172 7 22 getdents64 9.41 0.000164 2 73 mmap 8.50 0.000148 1 104 read 6.37 0.000111 1 86 close 4.94 0.000086 0 91 2 lseek

		3.27 0.000057 1 53 49 ioctl 2.87 0.000050 2 17 mprotect 0.86 0.000015 1 10 pread64 0.69 0.000012 0 14 brk 0.46 0.000008 1 5 munmap 0.40 0.000007 1 4 3 readlink 0.29 0.000005 0 68 rt_sigaction 0.23 0.000004 2 2 2 access 0.23 0.000004 1 3 dup 0.23 0.000004 2 2 getcwd 0.23 0.000004 1 4 2 arch_prctl 0.23 0.000004 4 1 getrandom 0.17 0.000003 3 1 rt_sigprocmask 0.17 0.000003 1 2 futex 0.17 0.000003 3 1 set_tid_address 0.11 0.000002 2 1 sysinfo 0.11 0.000002 2 1 getuid 0.11 0.000002 2 1 getgid 0.11 0.000002 2 1 geteuid 0.11 0.000002 2 1 getegid 0.11 0.000002 2 1 set_robust_list 0.11 0.000002 2 1 prlimit64 0.06 0.000001 0 3 fcntl 0.00 0.000000 0 1 write 0.00 0.000000 0 5 3 execve ------ ----------- ----------- --------- --------- ---------------- 100.00 0.001742 1 1111 224 total
	t4.txt	Longest palindrome: redder % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 23.59 0.000456 2 196 113 openat 21.37 0.000413 1 336 50 newfstatat 9.47 0.000183 8 22 getdents64 9.26 0.000179 2 73 mmap 7.76 0.000150 0 808 read 5.85 0.000113 1 86 close 5.54 0.000107 21 5 3 execve 5.02 0.000097 1 68 rt_sigaction 2.85 0.000055 0 91 2 lseek 2.38 0.000046 2 17 mprotect 2.22 0.000043 0 53 49 ioctl 1.24 0.000024 0 27 brk 0.67 0.000013 1 10 pread64 0.57 0.000011 2 5 munmap 0.41 0.000008 2 4 3 readlink 0.31 0.000006 2 3 dup

		0.21 0.000004 2 2 getcwd 0.21 0.000004 1 4 2 arch_prctl 0.21 0.000004 4 1 getrandom 0.16 0.000003 1 2 2 access 0.16 0.000003 1 3 fcntl 0.16 0.000003 3 1 prlimit64 0.10 0.000002 2 1 rt_sigprocmask 0.10 0.000002 1 2 futex 0.10 0.000002 2 1 set_tid_address 0.10 0.000002 2 1 set_robust_list 0.00 0.000000 0 1 write 0.00 0.000000 0 1 sysinfo 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 0.001933 1 1828 224 total
slow-pali.cpp	t3.txt	Longest palindrome: ___o.O.o___ % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 0.00 0.000000 0 43 read 0.00 0.000000 0 1 write 0.00 0.000000 0 5 close 0.00 0.000000 0 22 mmap 0.00 0.000000 0 9 mprotect 0.00 0.000000 0 1 munmap 0.00 0.000000 0 3 brk 0.00 0.000000 0 4 pread64 0.00 0.000000 0 1 1 access 0.00 0.000000 0 1 execve 0.00 0.000000 0 2 1 arch_prctl 0.00 0.000000 0 58 53 openat 0.00 0.000000 0 16 9 newfstatat ------ ----------- ----------- --------- --------- ---------------- 100.00 0.000000 0 166 64 total
	t4.txt	Longest palindrome: redder % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 100.00 11.404008 1 5767198 read 0.00 0.000137 2 58 53 openat 0.00 0.000079 3 22 mmap 0.00 0.000037 2 16 9 newfstatat 0.00 0.000036 4 9 mprotect 0.00 0.000012 2 5 close 0.00 0.000009 2 4 pread64

		0.00 0.000007 7 1 write 0.00 0.000006 6 1 munmap 0.00 0.000005 1 3 brk 0.00 0.000002 1 2 1 arch_prctl 0.00 0.000000 0 1 1 access 0.00 0.000000 0 1 execve ------ ----------- ----------- --------- --------- ---------------- 100.00 11.404338 1 5767321 64 total

d) In most cases, python code will be slower than C++ code. This generalization, however, does not hold when the C++ code is extremely inefficient. Since slow-pali.cpp is very inefficient, it will not be able to outperform well-written python code consistently. Looking at the strace results for t4.txt show that the C++ program makes 5767198 read system calls while the python file only makes 808 read system calls. Since system calls are expensive, requiring the CPU to switch from user to kernel mode, this causes the C++ program to take 2.7 seconds compared to the 0.26 seconds the python file takes. However, looking at the results of the time utility on t3.txt, we can see that the C++ code is 0.02 seconds faster than the python code. This is likely because the inefficient slow-pali.cpp only makes 43 read system calls while the python code makes 104 for the same file. Once again, since system calls are expensive, this will cause the python code to take longer. Since the C++ code is unoptimized, it makes more system calls than necessary which causes it to be slower than the python program in some cases. However, the opposite can also be true since the python program sometimes makes more system calls than the C++ code.

Question 3

a) *Timing results of fast-pali.cpp using 'time' utility*

Text File Input	**Terminal Output**
t3.txt	Longest palindrome: ___o.O.o___ real 0m0.008s user 0m0.002s sys 0m0.003s
t4.txt	Longest palindrome: redder real 0m0.136s user 0m0.084s sys 0m0.008s

Timing results of fast-pali.cpp using 'strace -c' utility

Text File Input	**Terminal Output**
t3.txt	Longest palindrome: ___o.O.o___ % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 31.82 0.000878 15 58 53 openat

	26.86 0.000741 741 1 execve 14.75 0.000407 18 22 mmap 8.30 0.000229 14 16 9 newfstatat 5.87 0.000162 18 9 mprotect 2.94 0.000081 13 6 read 2.10 0.000058 11 5 close 1.74 0.000048 12 4 pread64 1.59 0.000044 44 1 munmap 1.56 0.000043 14 3 brk 0.87 0.000024 24 1 1 access 0.87 0.000024 12 2 1 arch_prctl 0.72 0.000020 20 1 write ------ ----------- ----------- --------- --------- ---------------- 100.00 0.002759 21 129 64 total
t4.txt	Longest palindrome: redder % time seconds usecs/call calls errors syscall ------ ----------- ----------- --------- --------- ---------------- 48.55 0.000504 45 11 read 16.18 0.000168 2 58 53 openat 15.61 0.000162 162 1 execve 8.09 0.000084 3 22 mmap 3.95 0.000041 2 16 9 newfstatat 3.47 0.000036 4 9 mprotect 0.96 0.000010 2 5 close 0.96 0.000010 10 1 munmap 0.77 0.000008 2 4 pread64 0.67 0.000007 2 3 brk 0.39 0.000004 4 1 1 access 0.39 0.000004 2 2 1 arch_prctl 0.00 0.000000 0 1 write ------ ----------- ----------- --------- --------- ---------------- 100.00 0.001038 7 134 64 total

b) My fast-pali.cpp is faster than slow-pali.cpp. For t3.txt, the difference is minimal, with the same real time, but a 0.001s difference in the times spent in both user and kernel mode. For t4.txt the difference is much more noticeable with over 2.5 seconds saved on fast-pali.cpp. This is because fast-pali.cpp makes fewer read system calls and therefore spends less time switching between user and kernel mode and less time in user mode overall. This can be seen in the results of the strace utility where the slower program makes 5767198 read system calls for t4.txt while the fast program only makes 11 of these calls. Since system calls require that the CPU switches from user to kernel mode, they are expensive operations which is why the fast-pali.cpp program runs so much faster than the slow-pali.cpp program on the larger input.

c) Overall fast-pali.cpp is also faster than palindrome.py. The difference is not as drastic as the difference between the C++ programs. This improvement is mostly because python code is generally slower than well-written C++ programs. For t3.txt, the C++ code saves approximately 0.02 seconds overall, and for t4.txt, it saves 0.124 seconds over the python program according to the time utility. Since fast-pali.cpp has been somewhat optimized, it will run faster than equivalent python code. Looking at the strace utility for both programs, we can see that the python program makes 808 read system calls for t4.txt while fast-pali.cpp only makes 11. Similarly for t3.txt, fast-pali.cpp only makes 6 read system calls while palindrome.py makes 104. For the same reasons as part B, this decrease in system calls is the main area that fast-pali.cpp saves time, which is why it runs faster than palindrome.py.