**CPSC 457 A1 REPORT**

* 1. Calling the time utility on palindrome.py using t3.txt as input gives us the following:

real    0m0.032s

user    0m0.025s

sys    0m0.005s

Similarly, calling the time utility on palindrome.py using t4.txt as input gives us the following:

real    0m0.264s

user    0m0.256s

sys    0m0.005s

Calling the time utility on slow-pali.cpp using t3.txt as input gives us:

real    0m0.002s

user    0m0.000s

sys    0m0.001s

And calling the time utility on slow-pali.cpp using t4.txt as input gives us:

real    0m1.573s

user    0m0.259s

sys    0m1.310s

* 1. By taking the time spent in kernel mode to be real – user, we get that when using t3.txt as input the python program and C++ program spent 0.007s and 0.002s in kernel mode respectively. When the input was t4.txt we see these numbers change to 0.008s and 1.314s respectively.
  2. By calling the strace utility on the python program using file t3.txt as input, we get the following output:

% time seconds  usecs/call calls errors syscall

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

 26.50 0.000961       2   373   245 openat

 20.95 0.000760       1   496    56 newfstatat

 12.99 0.000471       2   174       mmap

  8.13 0.000295       1   165       read

  7.20 0.000261      10    24       getdents64

  6.40 0.000232       1   132       close

  4.82 0.000175       1   127     2 lseek

  3.72 0.000135      27     5     3 execve

  2.89 0.000105       2    48       mprotect

  2.01 0.000073       1    71    67 ioctl

  1.19 0.000043       2    17       brk

  0.58 0.000021       5     4     3 readlink

  0.55 0.000020       0    68       rt\_sigaction

  0.41 0.000015       1    10       pread64

  0.28 0.000010       1     8       munmap

  0.25 0.000009       4     2       getcwd

  0.11 0.000004       0     5     4 access

  0.11 0.000004       1     4     2 arch\_prctl

  0.11 0.000004       2     2       futex

  0.08 0.000003       3     1       rt\_sigprocmask

  0.08 0.000003       3     1       sysinfo

  0.08 0.000003       3     1       gettid

  0.08 0.000003       3     1       set\_tid\_address

  0.08 0.000003       3     1       set\_robust\_list

  0.08 0.000003       3     1       prlimit64

  0.06 0.000002       2     1       write

  0.06 0.000002       0     3       fcntl

  0.06 0.000002       2     1       getuid

  0.06 0.000002       2     1       getrandom

  0.03 0.000001       1     1       getgid

  0.03 0.000001       1     1       geteuid

  0.03 0.000001       1     1       getegid

  0.00 0.000000       0     3       dup

  0.00 0.000000       0     1       uname

  0.00 0.000000       0     2     2 statfs

  0.00 0.000000       0     6     4 prctl

  0.00 0.000000       0     1       epoll\_create1

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

100.00 0.003627       2 1763   388 total

And the following output when the file t4.txt is used on the python program:

% time seconds  usecs/call calls errors syscall

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

 25.89 0.000831       2   373   245 openat

 16.04 0.000515       1   496    56 newfstatat

 15.26 0.000490       2   174       mmap

 10.93 0.000351       0   869       read

  7.98 0.000256       3    68       rt\_sigaction

  4.58 0.000147       3    48       mprotect

  4.36 0.000140      28     5     3 execve

  4.24 0.000136       1   132       close

  2.74 0.000088       0   127     2 lseek

  2.68 0.000086       3    24       getdents64

  1.74 0.000056       0    71    67 ioctl

  0.93 0.000030       3    10       pread64

  0.72 0.000023       0    29       brk

  0.50 0.000016       2     8       munmap

  0.22 0.000007       1     4     3 readlink

  0.19 0.000006       6     1       write

  0.16 0.000005       1     4     2 arch\_prctl

  0.12 0.000004       0     5     4 access

  0.12 0.000004       2     2       getcwd

  0.09 0.000003       3     1       rt\_sigprocmask

  0.09 0.000003       1     3       dup

  0.09 0.000003       1     2       futex

  0.09 0.000003       3     1       prlimit64

  0.09 0.000003       3     1       getrandom

  0.06 0.000002       2     1       set\_tid\_address

  0.06 0.000002       2     1       set\_robust\_list

  0.00 0.000000       0     1       uname

  0.00 0.000000       0     3       fcntl

  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

  0.00 0.000000       0     2     2 statfs

  0.00 0.000000       0     6     4 prctl

  0.00 0.000000       0     1       gettid

  0.00 0.000000       0     1       epoll\_create1

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

100.00 0.003210       1  2479   388 total

The output of the strace command for the slow-pali.cpp program using t3.txt as input is as follows:

% 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

And finally, the output for the strace of slow-pali.cpp using t4.txt as input:

% time seconds  usecs/call calls errors syscall

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

 99.98   16.421723      2   5767198       read

  0.01 0.001035      17    58    53 openat

  0.00 0.000733     733     1       execve

  0.00 0.000460      20    22       mmap

  0.00 0.000275      17    16     9 newfstatat

  0.00 0.000143      15     9       mprotect

  0.00 0.000069      13     5       close

  0.00 0.000057      14     4       pread64

  0.00 0.000027      13     2     1 arch\_prctl

  0.00 0.000023      23     1     1 access

  0.00 0.000014       4     3       brk

  0.00 0.000006       6     1       write

  0.00 0.000000       0     1       munmap

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

100.00   16.424565      2   5767321    64 total

* 1. From the above results, we can see that the python program makes more system calls than slow-pali.cpp on smaller inputs. I believe this is because the python code makes many different system calls, whereas for the slow-pali code the majority of the system calls made are read calls. So, on smaller input files where there are fewer read calls to be made, the slow-pali code will outperform the python code because it isn’t making as many different system calls, but once the input file becomes bigger, slow-pali must make many read calls which slows it down. This can be seen through the results of the strace and the time functions, if we compare the total number of system calls that the python code made compared to the slow-pali code, we can see that there were 1763 calls made for the python code, compared to the 166 calls made by slow-pali when t3.txt is input. This difference is reflected in the time as well, as the python code spent 0.007s in kernel mode compared to the 0.002s the slow-pali spent. The change when a bigger input file is used is quite drastic; As we can see that the slow-pali ends up making 5767321 total system calls, 5767198 of which are read calls, when t4.txt is given as input as opposed to the 2479 system calls made by the python code. This affects the time difference massively, as the python code spends only 0.008s in kernel mode, compared to the 1.314s spent by slow-pali.

3. a. the output of using the time utility on fast-pali.cpp with t3.txt as input is:

real    0m0.006s

user    0m0.001s

sys    0m0.004s

and with t4.txt as input:

real    0m0.086s

user    0m0.081s

sys    0m0.002s

the output of the strace utility using t3.txt is:

% time seconds  usecs/call calls errors syscall

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

 28.25 0.000087       3    22       mmap

 24.03 0.000074       1    58    53 openat

 16.56 0.000051       5     9       mprotect

 14.29 0.000044       2    16     9 newfstatat

  4.22 0.000013       2     6       read

  3.57 0.000011       2     4       pread64

  2.60 0.000008       1     5       close

  2.27 0.000007       2     3       brk

  1.95 0.000006       6     1       munmap

  1.62 0.000005       5     1       write

  0.65 0.000002       1     2     1 arch\_prctl

  0.00 0.000000       0     1     1 access

  0.00 0.000000       0     1       execve

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

100.00 0.000308       2   129    64 total

And using t4.txt as input we get:

% time seconds  usecs/call calls errors syscall

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

  0.00 0.000000       0    11       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   134    64 total

b. Yes, this program is faster than slow-pali.cpp. This is due to the drastic decrease in the number of system calls made. By looking at the strace results, with t4.txt as input the number of read calls made drops from 5767198 to 11. This changes the amount of time spent in kernel mode from 1.314s to 0.005s.

c. Yes, this program is faster than palindrome.py, again due to the drastic decrease in the number of system calls. The total number of system calls for palindrome.py with t4.txt as input is 2479, compared to 134 system calls made by fast-pali. The amount of time spent in kernel mode drops from 0.008s to 0.005s.