Assignment 1

UCID: 30067857

Q1 - Written question

a. Time the python code and C++ code on `t4.txt` and `t3.txt` using the `time` utility. Copy/paste the outputs from the terminal output to your report.

C++ `slow-pali.cpp`

```
$ time ./slow-pali < t3.txt
Longest palindrome: ___o.O.o___

real    0m0.011s
user    0m0.003s
sys     0m0.001s
```

```
$ time ./slow-pali < t4.txt
Longest palindrome: redder

real    0m2.773s
user    0m1.422s
sys     0m1.347s
```

Python `palindrome.py`

```
$ time python3 palindrome.py < t3.txt
Longest palindrome: ___o.O.o___

real    0m0.024s
user    0m0.019s
sys     0m0.005s
```

```
$ time python3 palindrome.py < t4.txt
Longest palindrome: redder

real    0m0.293s
user    0m0.289s
sys     0m0.003s
```

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

Field `user` represents the time spend on user mode and `sys` for kernel mode. C++ program has similar amount of time on both modes, and python program has most of the time on user mode.

c. Why is the python program faster on some inputs when compared to C++ code? Why is the python program slower on other inputs?

Compare the time spend on `t3.txt` and `t4.txt`, C++ program is faster for small text input, and Python program is faster on larger.
For `t4.txt`, Python is faster only because the input is huge and C++ made huge amount of `read` calls, namely one character per `read`. For `t3.txt`, C++ is faster because: a. Python has no advantage on `read` calls, the file is small, b. Python is an interpreted language, it compiles the script everytime it runs; C++ program is pre-compiled, it only compile once, so it has better performance than Python.

Q2 - Programming questio

`fast-pali.cpp` is attached.

Q3 - Written question

a. Is your `fast-pali.cpp` faster than `slow-pali.cpp`? Why do you think that is?

Yes, it is faster. Because by having a larger buffer, `fast-pali` reads a block of text at a time, and reduce the times invoking `read` function, there is not as much waiting time by I/O. `fast-pali` only made `13` calls of `read`, while `slow-pali` made `5597763` calls which is almost the length of the text, as it only reads a single character per `read`. And the split word function is simplified, no vector is involved, no extra memory and fewer for loops.

```
 $  strace -c ./slow-pali < t4.txt
Longest palindrome: redder
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00   10.739006           1   5597763           read
  0.00    0.000096          13         7           mprotect
  0.00    0.000082          27         3           brk
  0.00    0.000023           1        22           mmap
  0.00    0.000010          10         1           munmap
  0.00    0.000008           8         1           write
  0.00    0.000003           0         5           close
  0.00    0.000003           0         6           fstat
  0.00    0.000003           1         2         1 arch_prctl
  0.00    0.000000           0         8         7 stat
  0.00    0.000000           0         7           lseek
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0        48        43 openat
------ ----------- ----------- --------- --------- ----------------
100.00   10.739234           1   5597875        52 total
```

```
 $ strace -c ./fast-pali < t4.txt
Longest palindrome: redder
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 86.71    0.003098         238        13           read
  7.58    0.000271         135         2           munmap
  1.85    0.000066           9         7           mprotect
  1.76    0.000063           1        48        43 openat
  1.20    0.000043           1        23           mmap
  0.36    0.000013           1         7           lseek
  0.20    0.000007           1         6           fstat
  0.11    0.000004           0         5           close
  0.11    0.000004           0         5           brk
  0.06    0.000002           2         1           write
  0.06    0.000002           1         2         1 arch_prctl
  0.00    0.000000           0         8         7 stat
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.003573          27       129        52 total
```

b. Is your program faster or slower than `palindrome.py` and why?

Yes, it is. Compare the calls of `read` field, `palindrome.py` reads one line at a time, and though it did not make that huge amount of `read` calls as `slow-pali`, it still made `767` calls, while `fast-pali.cpp` reads 10M of text which is way more than one line, the times invoking I/O is much fewer and less much time waiting. And because python is a interpreted language, as a trade off, it does more things than what we actually need: there are `175` calls on `stat` which takes even more time than `read`, and other more operations like `sysinfo`.

```
 $ strace -c python3 palindrome.py < t4.txt
Longest palindrome: redder
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 17.40    0.000436           2       175        47 stat
 17.36    0.000435           3       141        76 openat
 14.60    0.000366           0       767           read
  8.78    0.000220           3        59           mmap
  7.86    0.000197           1       100           fstat
  7.14    0.000179          11        16           getdents64
  6.58    0.000165           2        68           close
  6.38    0.000160           2        68           rt_sigaction
  4.71    0.000118         118         1           execve
  2.63    0.000066           1        42         2 lseek
  1.52    0.000038           3        11           mprotect
  0.96    0.000024           1        18        11 ioctl
  0.92    0.000023           0        66           brk
  0.52    0.000013           4         3           fcntl
  0.36    0.000009           3         3           munmap
  0.32    0.000008           8         1           lstat
  0.28    0.000007           2         3         2 readlink
  0.20    0.000005           5         1           set_robust_list
  0.16    0.000004           4         1         1 access
  0.16    0.000004           1         3           dup
  0.16    0.000004           4         1           getcwd
  0.16    0.000004           2         2         1 arch_prctl
  0.12    0.000003           3         1           getpid
  0.12    0.000003           1         2           futex
  0.08    0.000002           2         1           sysinfo
  0.08    0.000002           2         1           getuid
  0.08    0.000002           0         3           sigaltstack
  0.08    0.000002           2         1           set_tid_address
  0.08    0.000002           2         1           getrandom
  0.04    0.000001           1         1           rt_sigprocmask
  0.04    0.000001           1         1           getgid
  0.04    0.000001           1         1           geteuid
  0.04    0.000001           1         1           getegid
  0.04    0.000001           1         1           prlimit64
  0.00    0.000000           0         1           write
------ ----------- ----------- --------- --------- ----------------
100.00    0.002506           1      1566       140 total
```