CPSC457 Fall 2021 - Assignment 1

Due date is posted on D2L.
Individual assignment. Group work is NOT allowed.
Weight: 15% of the final grade.

Motivation for this assignment:

- Python programs usually run slower than C++ programs. But a well written Python code can run faster than a badly written C++ code.
- A common reason for badly performing C++ is the use of unnecessary system calls.
- In this assignment you will improve the performance of a badly written C++ program, by changing it to use less system calls.

Start by cloning the following repository:

```
$ git clone https://gitlab.com/cpsc457f21/palindrome.git
$ cd palindrome
```

The repository contains:

`palindrome.py`	Python 3 program that reads in text from standard input and reports the longest palindrome to standard output.
`slow-pali.cpp`	Not a very good C++ implementation of palindrome.py. Feel free to re-use any part of this code in your solution.
`fast-pali.cpp`	This is the file you will need to modify and submit for grading.
`Makefile`	Makes compilation a bit easier.
`t1.txt … t6.txt`	Some test files.
`dup.py`	Python3 script that can generate big data (see appendix).

For this assignment we will use the following definitions:

Standard input	Please read http://www.linfo.org/standard_input.html .
White space	Whatever `isspace()` reports as white-space. Read the man page for `isspace()` or https://www.cplusplus.com/reference/cctype/isspace/ for more information.
Word	Non-zero-length sequence of non-white-space characters delimited by white space, or beginning of file, or end of file.
Palindrome	Any word that remains the same after reversing it and ignoring the case. Examples of palindromes: "`Did`", "`01-!-10`", "`x`"
Longest palindrome	If there are multiple palindromes, the longest one is reported. If multiple palindromes have the same maximum length, report the first one.

Q1 - Written question (5 marks)

For this question you will compare the performance of the python program (`palindrome.py`) to the C++ program (`slow-pali.cpp`) by using `time` and `strace` utilities. For example, to time `palindrome.py` on the `t5.txt` file, execute this command:

```
$ time python3 palindrome.py < t5.txt
Longest palindrome: DetartrateD
real    0m0.023s
user    0m0.016s
sys     0m0.007s
```

To get a summary of all system calls made by `palindrome.py,` run this:

```
$ strace -c ./palindrome.py < t5.txt
Longest palindrome: DetartrateD
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 23.43    0.000422           2       189       105 openat
 12.49    0.000225           1       189        54 stat
 10.72    0.000193           2        78           mmap
  9.38    0.000169           8        20           getdents64
  7.77    0.000140           1       119           fstat
  7.22    0.000130          18         7         5 execve
  6.72    0.000121           1        68           rt_sigaction
  6.50    0.000117           1        87           close
  5.22    0.000094           1        90           read
  2.67    0.000048           3        15           mprotect
...
```

The results above indicate that the `read()` system call was executed 90 times.

Here is how you can compile the C++ code by using the included `Makefile`:

```
$ make
g++ -O2 -Wall slow-pali.cpp -o slow-pali
```

If you wish, you can also compile it by hand:

```
$ g++ -O2 -Wall slow-pali.cpp –o slow-pali
```

Now you can `time` the resulting executable `slow-pali`:

```
$ time ./slow-pali < t4.txt
Longest palindrome: redder
real    0m2.929s
user    0m1.477s
sys     0m1.450s
```

And this is how you run `strace` on it:

```
$ strace -c ./slow-pali < t5.txt
Longest palindrome: DetartrateD
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 33.58    0.000231           1       117           read
```

```
24.13    0.000166         166         1           execve
16.72    0.000115           2        48        43 openat
...
```

Answer the following questions:

a) Time the `palindrome.py` and `slow-pali.cpp` on `t4.txt` and `t3.txt` using the `'time'` utility. Copy/paste the output from the terminal window into your report.

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

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

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.

Q2 - Programming question (15 marks)

Your job is to improve `slow-pali.cpp` by writing a new implementation called `fast-pali.cpp`. Your new implementation should be faster than `slow-pali.cpp` and at least as fast as `palindrome.py` for all possible inputs! Your new implementation must match the output of the slow implementation and the Python implementation. You may re-use any code from the `slow-pali.cpp` file.

Hints

The slow-pali.cpp makes too many `read()` system calls, as it calls `read()` for every single character. You need to find a way to reduce the number of calls to `read()`. I suggest you refactor the slow code so that `read()` is called with a buffer size of 1MB, i.e. you will read about 1 million bytes per system call, which should dramatically speed up your program.

The GitLab repository below contains a similar problem and solution. Feel free to reuse any parts of this code in your own solution, but please include citations for the parts you reuse.

https://gitlab.com/cpsc457/public/longest-int-my-getchar

Valid input

Your program must be able to handle any text input of up to 2GiB in size. You may assume that no word will be longer than 1024 bytes. The files may or may not include a new line at the end.

Small number of test files are available in the GitLab repository, but it is expected that you create your own test files to help you validate your solutions. Your TAs will grade your code on inputs that are not published to you.

Requirements

- Your program must read input from standard input.
- You are only allowed to use the `read()` system call wrapper. You cannot use any other APIs, such as `mmap()`, `fopen()`, `fread()`, `fgetc()`, or C++'s streams.
- Do not store the entire input in memory. You need to write your code so that it can handle any input size, even if it is bigger than the available memory.

- Your program must run on `linuxlab.cpsc.ucalgary.ca`. Use SSH to test your code before you submit it!

Marking

Your code needs to be both correct, and efficient. Programs that output wrong results, or run very slowly, will receive 0 marks. On 2GB input your program should finish under 30s on `linuxlab` machines. Below are some timings I obtained using my own solution, to give you an idea of what you should be aiming for.

```
$ ./dup.py 2000000000 < t4.txt | time ./fast-pali
Longest palindrome: redder
28.79user 0.25system 0:29.52elapsed 98%CPU (0avgtext+0avgdata
4076maxresident)k
0inputs+0outputs (0major+382minor)pagefaults 0swaps

$ seq 101010101 | time -p ./ fast-pali
Longest palindrome: 100000001
real 10.18
user 9.87
sys 0.09
```

Please check the D2L page for this assignment periodically, as it may contain additional hints and answers to common questions.

Q3 - Written question (5 marks)

a) Run your `fast-pali.cpp` on `t3.txt` and `t4.txt` files using `’time’` and `’strace -c’`. Copy/paste the output from the terminal window into your report.
b) Is your `fast-pali.cpp` faster than `slow-pali.cpp`? Why do you think that is?
c) Is your program faster than `palindrome.py` and why?

Justify your answers for (b) and (c) by comparing the outputs of `’time’` and ‘`strace -c’`.

Submission

Submit two files for this assignment to D2L:

- Answers to the written questions Q1 and Q3 combined into a single file called `report.[pdf|docx|txt]`. Do not use any other file formats.
- Your solution to Q2 in a file called `fast-pali.cpp`.

Submit these files as two separate files, i.e. **do not** submit an archive, such as ZIP or TAR. If you submit an archive, you will receive a penalty.

General information about all assignments

1. All assignments are due on the date listed on D2L. Late submissions will not be marked.
2. Extensions may be granted only by the course instructor.

3. After you submit your work to D2L, verify your submission by re-downloading it.
4. You can submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It is better to submit incomplete work for a chance of getting partial marks, than not to submit anything. Please bear in mind that you cannot re-submit a single file if you have already submitted other files. Your new submission would delete the previous files you submitted. So please keep a copy of all files you intend to submit and resubmit all of them every time.
5. Assignments are likely going to be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA, then you can contact your instructor.
6. All programs you submit must run on `linuxlab.cpsc.ucalgary.ca`. If your TA is unable to run your code on `linuxlab`, you will receive 0 marks.
7. Unless specified otherwise, you must submit code that can finish on any valid input under 10s on linuxlab.cpsc.ucalgary.ca, when compiled with `-O2` optimization. Any code that runs longer than this may receive a deduction, and code that runs for too long (about 30s) will receive 0 marks.
8. **Assignments must reflect individual work**. Here are some examples of what you are not allowed to do for individual assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to collaborate with anyone; you are not allowed to share your solutions (code or pseudocode) with anyone; you are not allowed to sell or purchase a solution; you are not allowed to make your code available publicly (e.g. via public git repositories). This list is not exclusive. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
9. We will use automated similarity detection software to check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.

Appendix – dup.py utility (aka. testing your code on large inputs)

Many of you probably do not have enough storage quota to store 2Gib test files in your accounts. I created a python script `dup.py` to make it possible to test your program on large inputs, without having to store big files.

`dup.py` is a simple python program that accepts a single command line argument "N", which indicates the number of bytes that the script will generate on standard output. `dup.py` reads in data from stdin, byte by byte, and outputs the data to stdout. It always outputs N bytes. If the data on stdin is bigger than N bytes, only the first N bytes are copied. If the data on stdin is shorter than N, the script will repeat the input data, until N bytes are generated. Example:

```
$ echo "hello." | ./dup.py 10
hello.
hel
```

Here is an example of how to feed 2GB of data to your program, generated by repeating `t3.txt`:

```
$ ./dup.py 2000000000 < t3.txt | ./fast-pali
```

Here is how you can `time` your code on the same data:

```
$ ./dup.py 2000000000 < t3.txt | time ./fast-pali
```

Here is how to run `strace` in combination with dup.py:

```
$ ./dup.py 2000000000 < t3.txt | strace -c ./fast-pali
```

Warning: This assignment is quite simple, as it requires minimum amount of coding. Please do not assume that future assignments will be this simple.