A1 Report

February 4, 2025 4:05 PM

Benny Liang | 3019242

Q1)

a) Use the time utility to time palindrome.py and slow-pali.cpp on files t3.txt and t4.txt. Copy/paste the output of time from the terminal window into your report.

Using t3.txt as input

	`palindrome.py`	`slow-pali.cpp`
Real	0m0.034s	0m0.012s
User	0m0.022s	0m0.004s
Sys	0m0.011s	0m0.006s

Using t4.txt as input

	`palindrome.py`	`slow-pali.cpp`
Real	0m0.249s	0m2.295s
User	0m0.233s	0m0.361s
Sys	0m0.013s	0m1.908s

b) Using your results from (a), how much time did the C++ and python programs spend executing on the CPU, and how much time did each of them spend waiting for I/O to finish?

CPU time spent executing on `t3.txt`

- `palindrome.py` spent 0.022s + 0.011s = **0m0.033s on the CPU**
- `slow-pali.cpp` spent 0.004s + 0.006s = **0m0.010s on the CPU**

CPU time spent executing on `t4.txt`

- `palindrome.py` spent 0.233s + 0.013s = **0m0.246s on the CPU**
- `slow-pali.cpp` spent 0.361s + 1.908s = **0m2.269s on the CPU**

Time spent waiting for I/O to finish on `t3.txt`

- `palindrome.py` I/O time taken = real - user - sys = 0.034s - 0.022s - 0.011s = **0m0.001s**
- `slow-pali.cpp` I/O time taken = real - user - sys = 0.012s - 0.004s - 0.006s = **0m0.002s**

Time spent waiting for I/O to finish on `t4.txt`

- `palindrome.py` I/O time taken = real - user - sys = 0.249s - 0.233s - 0.013s = **0m0.003s**
- `slow-pali.cpp` I/O time taken = real - user - sys = 2.295s - 0.361s - 1.908s = **0m0.026s**

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

`palindrome.py`

`t3.txt` as input

Longest palindrome: ___o.O.o___

% time	seconds	usecs/call	calls	errors	syscall
16.94	0.000728	10	71	20	openat
12.73	0.000547	109	5	3	execve
12.08	0.000519	10	49		mmap
9.07	0.000390	2	141	55	newfstatat
8.63	0.000371	5	73		fstat
7.19	0.000309	6	51		close
6.82	0.000293	6	**48**		**read**

`t4.txt` as input

Longest palindrome: redder

% time	seconds	usecs/call	calls	errors	syscall
20.55	0.000995	199	5	3	execve
15.20	0.000736	15	49		mmap
14.09	0.000682	9	71	20	openat
10.45	0.000506	3	141	55	newfstatat
9.33	0.000452	6	**70**		**read**
6.98	0.000338	4	73		fstat
6.05	0.000293	5	51		close

6.58	0.000283	15	18		getdents64
6.42	0.000276	4	66		rt_sigaction
3.05	0.000131	2	45	2	lseek
2.77	0.000119	10	11		brk
1.98	0.000085	3	24	19	ioctl
1.49	0.000064	8	8		mprotect
0.86	0.000037	7	5		munmap
0.63	0.000027	9	3		getrandom
0.35	0.000015	3	4		fcntl
0.33	0.000014	7	2		futex
0.26	0.000011	5	2		rt_sigprocmask
0.26	0.000011	2	4		pread64
0.23	0.000010	5	2		getcwd
0.19	0.000008	4	2	2	access
0.14	0.000006	1	4	3	readlink
0.14	0.000006	6	1		gettid
0.12	0.000005	2	2		arch_prctl
0.12	0.000005	2	2		set_tid_address
0.12	0.000005	2	2		set_robust_list
0.12	0.000005	2	2		prlimit64
0.12	0.000005	2	2		rseq
0.09	0.000004	4	1		write
0.07	0.000003	3	1		getgid
0.05	0.000002	2	1		getuid
0.05	0.000002	2	1		geteuid
0.05	0.000002	2	1		getegid
100.00	0.004298	6	654		104 total

2.91	0.000141	7	18		getdents64
2.79	0.000135	3	45	2	lseek
2.38	0.000115	14	8		mprotect
1.45	0.000070	2	24	19	ioctl
1.28	0.000062	5	11		brk
1.22	0.000059	11	5		munmap
0.89	0.000043	10	4		pread64
0.74	0.000036	0	66		rt_sigaction
0.74	0.000036	18	2	2	access
0.41	0.000020	10	2		arch_prctl
0.41	0.000020	10	2		set_robust_list
0.41	0.000020	10	2		rseq
0.39	0.000019	9	2		set_tid_address
0.35	0.000017	5	3		getrandom
0.25	0.000012	3	4	3	readlink
0.25	0.000012	6	2		futex
0.25	0.000012	6	2		prlimit64
0.12	0.000006	1	4		fcntl
0.10	0.000005	2	2		getcwd
0.00	0.000000	0	1		write
0.00	0.000000	0	2		rt_sigprocmask
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	1		gettid
100.00	0.004842	7	676		104 total

`slow-pali.cpp`

`t3.txt` as input

Longest palindrome: ___o.O.o___

% time	seconds	usecs/call	calls	errors	syscall
37.90	0.000669	669	1		execve
23.34	0.000412	17	23		mmap
14.96	0.000264	6	**43**		**read**
7.20	0.000127	21	6		mprotect
4.31	0.000076	15	5		openat
2.61	0.000046	7	6		fstat
2.04	0.000036	7	5		close
1.76	0.000031	31	1		munmap
1.42	0.000025	12	2		pread64
0.74	0.000013	4	3		brk
0.62	0.000011	11	1		arch_prctl
0.62	0.000011	11	1		set_tid_address
0.62	0.000011	11	1		set_robust_list
0.62	0.000011	11	1		rseq
0.57	0.000010	10	1		write
0.45	0.000008	8	1		prlimit64
0.23	0.000004	4	1		futex
0.00	0.000000	0	1	1	access
0.00	0.000000	0	1		getrandom
100.00	0.001765	16	104		1 total

`t4.txt` as input

Longest palindrome: redder

% time	seconds	usecs/call	calls	errors	syscall
99.99	18.229164	3	**5767198**		**read**
0.00	0.000580	580	1		execve
0.00	0.000560	24	23		mmap
0.00	0.000099	19	5		openat
0.00	0.000068	11	6		mprotect
0.00	0.000057	9	6		fstat
0.00	0.000053	10	5		close
0.00	0.000037	12	3		brk
0.00	0.000028	14	2		pread64
0.00	0.000019	19	1	1	access
0.00	0.000012	12	1		set_tid_address
0.00	0.000012	12	1		rseq
0.00	0.000011	11	1		write
0.00	0.000011	11	1		arch_prctl
0.00	0.000011	11	1		set_robust_list
0.00	0.000009	9	1		getrandom
0.00	0.000000	0	1		munmap
0.00	0.000000	0	1		futex
0.00	0.000000	0	1		prlimit64
100.00	18.230731	3	5767259		1 total

d) Using the results from (c), why is the python program faster on some inputs than the C++ program, but slower on others?

The python program was slower on *smaller inputs* due to the fact that it had more read calls than the C++ program on the same input. Another contributor are the types of languages Python and C++ are, Python is an

interpreted language (runs line-by-line) whereas C++ is compiled before execution, helping the C++ program run faster than the Python program.
The python program was faster than the C++ program on *very large inputs*, due to the fact that the C++ program had to make so many `read` calls. When comparing with the python program it had a fraction of the C++ program's read calls.

Q3)

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.

`t3.txt` as input

`time`

```
Longest palindrome: ___o.O.o___

real    0m0.009s
user    0m0.004s
sys     0m0.004s
```

`strace -c`

```
Longest palindrome: ___o.O.o___
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 52.14    0.000573          24        23           mmap
 18.65    0.000205          34         6           mprotect
  9.01    0.000099          19         5           openat
  4.64    0.000051           8         6           fstat
  4.55    0.000050           8         6           read
  4.09    0.000045           9         5           close
  2.55    0.000028          14         2           pread64
  1.18    0.000013          13         1           arch_prctl
  1.09    0.000012          12         1           set_tid_address
  1.09    0.000012          12         1           set_robust_list
  1.00    0.000011          11         1           rseq
  0.00    0.000000           0         1           write
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           futex
  0.00    0.000000           0         1           prlimit64
  0.00    0.000000           0         1           getrandom
------ ----------- ----------- --------- --------- ----------------
100.00    0.001099          16        67         1 total
```

`t4.txt` as input

`time`

```
Longest palindrome: redder

real    0m0.111s
user    0m0.071s
sys     0m0.037s
```

`strace -c`

```
Longest palindrome: redder
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 31.24    0.001987         180        11           read
 30.58    0.001945          97        20           brk
 11.26    0.000716          23        31           mmap
 10.34    0.000658         658         1           execve
  9.50    0.000604          67         9           munmap
  2.15    0.000137          27         5           openat
  1.48    0.000094          18         5           close
  1.15    0.000073          12         6           fstat
  0.61    0.000039          19         2           pread64
  0.33    0.000021          21         1         1 access
  0.30    0.000019          19         1           set_tid_address
  0.25    0.000016          16         1           set_robust_list
  0.20    0.000013          13         1           futex
  0.20    0.000013          13         1           getrandom
  0.19    0.000012          12         1           arch_prctl
  0.19    0.000012          12         1           rseq
  0.03    0.000002           2         1           write
```

	0.00	0.000000	0	6		mprotect
	0.00	0.000000	0	1		prlimit64
	------	-----------	-----------	---------	---------	----------
	100.00	0.006361	60	105		1 total

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

Yes my `fast-pali.cpp` is much faster than `slow-pali.cpp` because of the significantly reduced number of system calls for `read`. For the `t4.txt` input; in `slow-pali.cpp` it had ***5756198*** read calls while `fast-pali.cpp` had ***11*** read calls, significant difference. Then looking at the `time` results (for `t4.txt), `fast-pali.cpp` finished much faster than `slow-pali.cpp`, again due to the significantly reduced number of `read` calls.

C) Is your program faster than palindrome.py and why?

Yes `fast-pali.cpp` is faster than `palindrome.py` for similar reasons as described in b). The results for `t4.txt` input, `palindrome.py` had ***70*** read calls whereas `fast-pali.cpp` had ***11*** read calls, then looking at the `time` results we can see that the `real` time for the python program was 0m0.249s and the C++ program was 0m0.111s, C++ program was indeed faster than the python version. Another factor that contributes to the C++ program running faster than the python program is due to the fact that C++ is a compiled language while Python is a interpreted language.