Assignment 1: John Ming Ngo, 30020834

Assignment 1: John Ming Ngo, 30020834

Question 1: Pipelining

Stages: 5 stages, taking times 3ns, 6ns, 1ns, 10ns, and 5ns.

a. No Pipelining:

If the stages do not operate in parallel, then one instruction at a time falls through eachs stage, finishing with the last step before the next instruction is executed. Each and every instruction, then, takes (3 + 6 + 1 + 10 + 5)ns of time, so the total number of instructions executed per second is 1s/(3 + 6 + 1 + 10 + 5)ns, or 1 000 000 000ns / 25ns = 40 000 000 instructions per second, total.

b. With Pipelining.

If the stages operate in parallel, the issue is that each stage still depends on the results of the prior stage to operate on. From the perspective of any one instruction, it must still fall through each stage, and take 25ns as a result. However, after an instruction completes a stage, the next instruction is loaded into that stage, as opposed to waiting for the instruction to finish, even as the current instruction continues on to the next stage. From the perspective of processed stages (the end of the pipeline), once the first instruction is fully processed, the next instructions will be processed as quickly as the slowest part of the pipeline, since each stage of the pipeline is processing instructions and the slowest stage is the only thing preventing the other stages from processing even more instructions. Thus, so long as there are a lot of instructions (the process runs for a long time), and these instructions are not discarded due to other instructions, the time between each finished instruction will tend to 10ns, and the total number of instructions executed per second will tend to be 1s/10ns = 1 000 000 000ns / 10ns = 100 000 000 instructions per second, total.

Question 2: Benefits of Virtual Machines

Virtual machines are software-run emulated machines with operating systems.

a. From a company's perspective: The biggest benefit of virtual machines is the possibility of cost savings, since it's typically cheaper to buy a single massive server (system consolidation) and then run as many virtual machines and applications as your operation requires on it, as opposed to buying a separate computer for every single computer your operation may need - especially since your operation might not know how many computers they need in the first place, or there may be wasted space or processing power on each individual computer that cannot be used where in a bigger server, that space would be usable (ex. program of size 5, individual computer of size 9, you can't fit 2 programs into that computer but with a server equal to 5 such computers, you can run 9 such programs as opposed to just 5 with the individual computers), so you might run into cost issues where you buy too many, or too little and then run into severe project delays for it.

b. From a programmer's perspective: Sometimes, as a programmer, you need to do cross-platform development, where your software needs to run on operating systems other than the one you're using. Rather than getting as many different computers as operating systems you intend your code to run on, you can just install those other operating systems as virtual machines on your computer, and then test your code on those virtual machines there, saving a lot of hassle.

c. From a regular user's perspective: Sometimes, a regular user might not be sure the program they installed is safe - the program could be a virus that hijacks their system down to the boot level, for example. With a virtual machine, it's completely contained, with no possibility that malware which damages the virtual machine can actually damage the real machine. Also, sometimes you want to use an application that only exists on a different operating system. You can use a virtual machine to run that program on your own computer, without getting a new computer.

d. From a system administrator's perspective: System administrators handle the management and configuration of all the computers under their perview on a daily basis. Virtual machines make this much easier, since rather than opening up and handling countless hundreds of different machines without VMs, they can handle a single large server and make software modificaitons to all the VMs on the server as needed. For example, suppose you need to install a new program. Rather than opening up every single computer and manually installing the application on every single computer, the System Administrator can simply run a script on the server to install the application on every virtual machine as needed.

Question 3: Exceptions.

a. Interrupts: A notification the CPU that something of significance has occurred and needs to be handled as soon as possible, like knocking on the door to let an owner know that someone's outside, wishing to communicate with them. Usually used for Input/Output, hardware inputs, and other applications where input is expected, but when it occurs is unknown, and things must be done according to that input. When an interrupt occurs, the CPU must save its state, load up a fresh state corresponding to what the interrupt is all about, execute the instructions relevant to the interrupt, then load back its prior state and continue as though nothing has happened.

b. Traps: Traps are software-triggered interrupts, where in the normal execution of the code, the CPU interrupts itself. It is used to detect bad instructions and methods to switch safety to kernal mode in order to execute code which requires kernal mode to work - that is, execute a pre-defined routine in kernal mode as desired. It can be used as a safe way to utilize kernal mode.

c. Hardware Interrupts vs Traps: Hardware interrupts occur due something other than the CPU itself, such as input/output management devices, and are asynchronous to the CPU. Traps are generated by the software the CPU is currently executing, such as a bad instruction, and follow the execution of the CPU. Hardware interrupts are completely unpredictable, since they're outside the control of the code and the CPU, whilst traps can be predictable, since they're written into or implied in the code and its execution (they are an internal call or event). Both trigger the same interrupt hardware to manage interrupt problems, such as the interrupt vector table. Both save the prior state of the CPU, put the CPU in kernal mode, invoke a kernal routine, and then restore the original state of the CPU when they are done. Traps are usually utilized to execute specific code in kernal mode, whilst hardware interrupts are utilzied to communicate between the hardware and a CPU waiting on the hardware. Traps are also usually intended, since unintended ones are usually called exceptions.

d. Kernal Mode vs User Mode: Interrupts are handed in kernal mode rather than user mode for many reasons. Firstly, in user mode, one cannot guarantee that the running user program is in any state to be aware that it received an interrupt and needs to handle the interrupt now - there's no reason that user mode must pay attention, unlike kernal mode, which is always connected to the hardware. Second, user-mode on most OSes does not even have permissions to touch the hardware in the first place, for security reasons, thus one requires the mode which actually has the permission - kernal mode - to handle the interrupt for the program.

Question 4: Word Count

a. The outputs of my time command are as follows:

john.ngo@csx:~/457$ time ./simple_wc < a-tale-of-two-cities.txt

16272 138883 804335

real 0m1.419s

user 0m0.384s

sys 0m1.024s

john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt

16272 138883 804335

real 0m0.018s

user 0m0.017s

sys 0m0.001s

b. User is the amount of time the program spent in user mode, whilst sys is the amount of time the program spent waiting on kernal mode. Based off that information, we can see that the C++ program spent 0.384 seconds in user mode, and 1.024 seconds in kernal mode. Both are vastly greater than the wc command, which used only 0.017 second and 0.001 seconds in user and system mode, respectively.

c. The biggest difference is in kernal mode - the wc command is significantly faster because it needed to spend next to no time in kernal mode at all, saving a full second there. Meanwhile, based off user mode performance, the WC command's algorithm is also likely more efficient, taking less steps to achieve the same things. The kernal mode differences can be explained (as hinted to in the lecture, and show in a later question) as the difference in the number of system calls needed to record the information, where simple_wc reads one byte of the input at a time, and so wastes a lot of time trapping, waiting on kernal mode to read the byte, then doing that over and over again, and the wc algorithm does not. Similarly, since the WC algorithm calls on the system call to read input less often, it likely requires less iterations to process everything, hence the lesser time spent in user mode.

Question 5: Rewrite the program.

See attached file/file in the same folder.

Question 6:

Here are my terminal results when testing my program versus the native wc command:

Welcome to the Department of Computer Science, University of Calgary

```
This system is for use by authorized users only.

   DEPARTMENT WEBPAGE:  https://ucalgary.ca/cpsc
 NEWS & ANNOUNCEMENTS:  https://ucalgary.ca/cpsc/news
         TECH SUPPORT:  https://ucalgary.ca/cpsc/tech
                        Email: scihelp@ucalgary.ca
                        Help Desk: MS 151

Please send bugs reports and package requests to scihelp@ucalgary.ca

Last login: Tue May 19 18:10:08 2020
john.ngo@csx:~$ cd 457
john.ngo@csx:~/457$ ls
 A1.pdf                         README.txt
 Assignment1_30020834.txt       Report.pdf
 Assignment1.docx               romeo-and-juliet.txt
 a-tale-of-two-cities.txt       simple_wc.cpp
 bad_simple_wc_with_streams.cpp smalltest.txt
 makeNullByteTest.cpp           Submission_30020834.zip
 myWc.cpp                       'Table of Contents.html'
john.ngo@csx:~/457$ chmod +rwx * -R
john.ngo@csx:~/457$ ls
 A1.pdf                         README.txt
 Assignment1_30020834.txt       Report.pdf
 Assignment1.docx               romeo-and-juliet.txt
 a-tale-of-two-cities.txt       simple_wc.cpp
 bad_simple_wc_with_streams.cpp smalltest.txt
 makeNullByteTest.cpp           Submission_30020834.zip
 myWc.cpp                       'Table of Contents.html'
john.ngo@csx:~/457$ g++ -o wcCompiled myWc.cpp
john.ngo@csx:~/457$ g++ -o myWc myWc.cpp
john.ngo@csx:~/457$ printf "Testing a\00 null \00 example." > nullTest.txt
john.ngo@csx:~/457$ wc < nullTest.txt
```

0 4 26

john.ngo@csx:~/457$ ./ my-bash: _xspecs: bad array subscript

john.ngo@csx:~/457$ rm myWc

john.ngo@csx:~/457$ rm wcCompiled

john.ngo@csx:~/457$ ls

A1.pdf Assignment1.docx bad_simple_wc_with_streams.cpp myWc.cpp README.txt romeo-and-juliet.txt smalltest.txt 'Table of Contents.html'

Assignment1_30020834.txt a-tale-of-two-cities.txt makeNullByteTest.cpp nullTest.txt Report.pdf simple_wc.cpp Submission_30020834.zip

john.ngo@csx:~/457$ chmod +rwx nullTest.txt

john.ngo@csx:~/457$ rm nullText.txt

rm: cannot remove 'nullText.txt': No such file or directory

john.ngo@csx:~/457$ rm nullTest.txt

john.ngo@csx:~/457$ clear

john.ngo@csx:~/457$ ls

A1.pdf Assignment1.docx bad_simple_wc_with_streams.cpp myWc.cpp Report.pdf simple_wc.cpp Submission_30020834.zip

Assignment1_30020834.txt a-tale-of-two-cities.txt makeNullByteTest.cpp README.txt romeo-and-juliet.txt smalltest.txt 'Table of Contents.html'

john.ngo@csx:~/457$ g++ -o myWc myWc.cpp

john.ngo@csx:~/457$ printf "Testing\00 U\00ser based \00\00\00 inserted null values" > nullTest.txt

john.ngo@csx:~/457$ ls

A1.pdf Assignment1.docx bad_simple_wc_with_streams.cpp myWc nullTest.txt Report.pdf simple_wc.cpp Submission_30020834.zip

Assignment1_30020834.txt a-tale-of-two-cities.txt makeNullByteTest.cpp myWc.cpp README.txt romeo-and-juliet.txt smalltest.txt 'Table of Contents.html'

john.ngo@csx:~/457$ wc < nullTest.txt

0 6 45

john.ngo@csx:~/457$ ./myWc < nullTest.txt

0 7 45

john.ngo@csx:~/457$ printf "It appears the difference here is if the null-byte center word is a word at all. Doesn't really matter."

It appears the difference here is if the null-byte center word is a word at all. Doesn't really matter.john.ngo@csx:~/457$

john.ngo@csx:~/457$ time wc < nullTest.txt

0 6 45

```
real    0m0.002s
user    0m0.001s
sys     0m0.000s
john.ngo@csx:~/457$ time ./myWc < nullTest.txt
        0       7      45

real    0m0.003s
user    0m0.001s
sys     0m0.001s
john.ngo@csx:~/457$ time wc < romeo-and-juliet.txt
  4853  28983 178983

real    0m0.009s
user    0m0.005s
sys     0m0.001s
john.ngo@csx:~/457$ time ./m
makeNullByteTest.cpp  myWc                  myWc.cpp
john.ngo@csx:~/457$ time ./myWc romeo-and-juliet.txt
^C
real    0m6.975s
user    0m0.001s
sys     0m0.001s

john.ngo@csx:~/457$ time ./myWc < romeo-and-juliet.txt
   4853   28983  178983

real    0m0.005s
user    0m0.004s
sys     0m0.001s
john.ngo@csx:~/457$ say Small mistake there
-bash: say: command not found
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
```

```
 16272 138883 804335

real    0m0.023s
user    0m0.018s
sys     0m0.000s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.013s
user    0m0.011s
sys     0m0.002s
john.ngo@csx:~/457$ strace -c wc < nullTest.txt
 0  6 45
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 43.27    0.000318           8        36        19 openat
 17.55    0.000129           6        19           fstat
 14.15    0.000104           5        18           mmap
 11.16    0.000082           4        20           close
  3.95    0.000029           7         4           mprotect
  3.13    0.000023           4         5           read
  2.18    0.000016          16         1           munmap
  2.04    0.000015           3         4           brk
  1.36    0.000010          10         1           write
  0.68    0.000005           5         1           fadvise64
  0.54    0.000004           4         1           arch_prctl
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.000735                   112        20 total
john.ngo@csx:~/457$ strace -c ./myWc < nullTest.txt
               0       7      45
% time     seconds  usecs/call     calls    errors syscall
```

```
------ ----------- ----------- --------- --------- ----------------
 35.98    0.000118          8        14           mmap
 28.35    0.000093          9        10           mprotect
 10.06    0.000033          6         5          openat
  8.84    0.000029          5         5          read
  5.18    0.000017          3         5          close
  4.88    0.000016          2         6          fstat
  3.35    0.000011         11         1           munmap
  2.13    0.000007          2         3          brk
  1.22    0.000004          4         1          arch_prctl
  0.00    0.000000          0         1          write
  0.00    0.000000          0         1        1 access
  0.00    0.000000          0         1          execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.000328                   53         1 total
john.ngo@csx:~/457$ strace -c wc < romeo-and-juliet.txt
  4853  28983 178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 42.74    0.000427         11        36        19 openat
 16.52    0.000165          9        18           mmap
 13.41    0.000134          6        20           close
 13.41    0.000134          7        19           fstat
 10.71    0.000107          7        15           read
  2.10    0.000021         21         1          write
  1.10    0.000011         11         1          fadvise64
  0.00    0.000000          0         4          mprotect
  0.00    0.000000          0         1          munmap
  0.00    0.000000          0         4          brk
  0.00    0.000000          0         1        1 access
  0.00    0.000000          0         1          execve
  0.00    0.000000          0         1          arch_prctl
------ ----------- ----------- --------- --------- ----------------
```

```
100.00    0.000999                 122       20 total
john.ngo@csx:~/457$ strace -c ./myWc < romeo-and-juliet.txt
   4853   28983  178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         6           fstat
  0.00    0.000000           0        14           mmap
  0.00    0.000000           0        10           mprotect
  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           arch_prctl
  0.00    0.000000           0         5           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                  53         1 total
john.ngo@csx:~/457$ strace -c wc < a-tale-of-two-cities.txt
 16272 138883 804335
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.82    0.000576          16        36        19 openat
 20.68    0.000363           6        54           read
 15.21    0.000267          14        18           mmap
 11.34    0.000199          10        19           fstat
  9.06    0.000159           7        20           close
  4.50    0.000079          19         4           mprotect
  2.45    0.000043          10         4           brk
  1.48    0.000026          26         1           munmap
  0.80    0.000014          14         1         1 access
  0.63    0.000011          11         1           execve
```

```
  0.57    0.000010        10        1         fadvise64
  0.46    0.000008         8        1         arch_prctl
  0.00    0.000000         0        1         write
------ ----------- ----------- --------- --------- ----------------
100.00    0.001755               161       20 total
john.ngo@csx:~/457$ strace -c ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.58    0.000202        14       14         mmap
 25.00    0.000155        15       10         mprotect
 13.39    0.000083        16        5         openat
  7.58    0.000047         9        5         read
  6.61    0.000041         8        5         close
  5.81    0.000036         6        6         fstat
  2.90    0.000018        18        1         munmap
  2.10    0.000013        13        1       1 access
  1.61    0.000010        10        1         execve
  1.29    0.000008         8        1         arch_prctl
  1.13    0.000007         2        3         brk
  0.00    0.000000         0        1         write
------ ----------- ----------- --------- --------- ----------------
100.00    0.000620                53        1 total
john.ngo@csx:~/457$ These are all the results.
-bash: These: command not found
john.ngo@csx:~/457$
```

a. Based off the timing results, when compared to the results for simple_wc from before, myWc is SIGNIFICANTLY faster than simple_wc. Let us investigate the strace -c of simple_wc:

```
john.ngo@csx:~/457$ strace -c ./simple_wc < romeo-and-juliet.txt
   4853   28983  178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
```

```
99.97    1.839927        10    178988          read
 0.01    0.000197        14        14          mmap
 0.01    0.000116        11        10          mprotect
 0.00    0.000061        10         6          fstat
 0.00    0.000055        11         5          openat
 0.00    0.000044         8         5          close
 0.00    0.000032        32         1          write
 0.00    0.000019        19         1        1 access
 0.00    0.000013        13         1          arch_prctl
 0.00    0.000007         2         3          brk
 0.00    0.000000         0         1          munmap
 0.00    0.000000         0         1          execve
------ ----------- ----------- --------- --------- ----------------
100.00    1.840471              179036         1 total
john.ngo@csx:~/457$
```

Contrast, for myWc:

```
john.ngo@csx:~/457$ strace -c ./myWc < romeo-and-juliet.txt
   4853   28983  178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         6           fstat
  0.00    0.000000           0        14           mmap
  0.00    0.000000           0        10           mprotect
  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           arch_prctl
  0.00    0.000000           0         5           openat
```

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

100.00   0.000000   53   1 total

Consider how in a prior question, I noted that simple_wc takes a lot of time due to a large amount of time spent in kernal mode, and further its algorithm is less efficient, so it even spends more time in user mode. Here, comparing it to the strace -c of myWc, we can clearly see that it utilizes the read() call vastly more than myWc does, and as the read call is a kernal call, putting the program into kernal mode and waiting until it reads some input data to resume the program, this alone explains why myWc is vastly faster. myWc utilizes orders of magnitude less kernal calls, which means much, much less time required to wait for all the times where the CPU state is saved, waiting until other kernal calls are done and it's their turn, and then doing their kernal call, only to do it all over again later in the program.

b. Based off the 'time' results above, myWc is within a similar order of magnitude of time as the wc command, and in fact is about 50% faster than the native wc command. Based off the trace results above, this appears to be primarily due to significantly less kernal calls, also resulting in less user time processes needed to handle more kernal calls.

This similar order of magnitude is likely because my word count program utilizes relatively few kernal calls - from the strace results above, it's clear that my program uses a roughly comparable number of kernal calls, as opposed to the several orders of magnitude more kernal calls that simple_wc utilized, and as kernal calls are the primary source of time inefficiency from before, reducing it down to a number comparable to the native wc has produced similar results, especially since it appears the actual user portion of the code runs fairly fast.

To attempt to definitively state whether my program is faster or slower, we can do multiple trials:

Doing multiple trials:

Wc:

john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt

 16272 138883 804335

real   0m0.018s

user   0m0.016s

sys   0m0.001s

john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt

 16272 138883 804335

real   0m0.018s

user   0m0.016s

```
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.018s
user    0m0.016s
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.017s
user    0m0.016s
sys     0m0.001s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.018s
user    0m0.016s
sys     0m0.002s
```

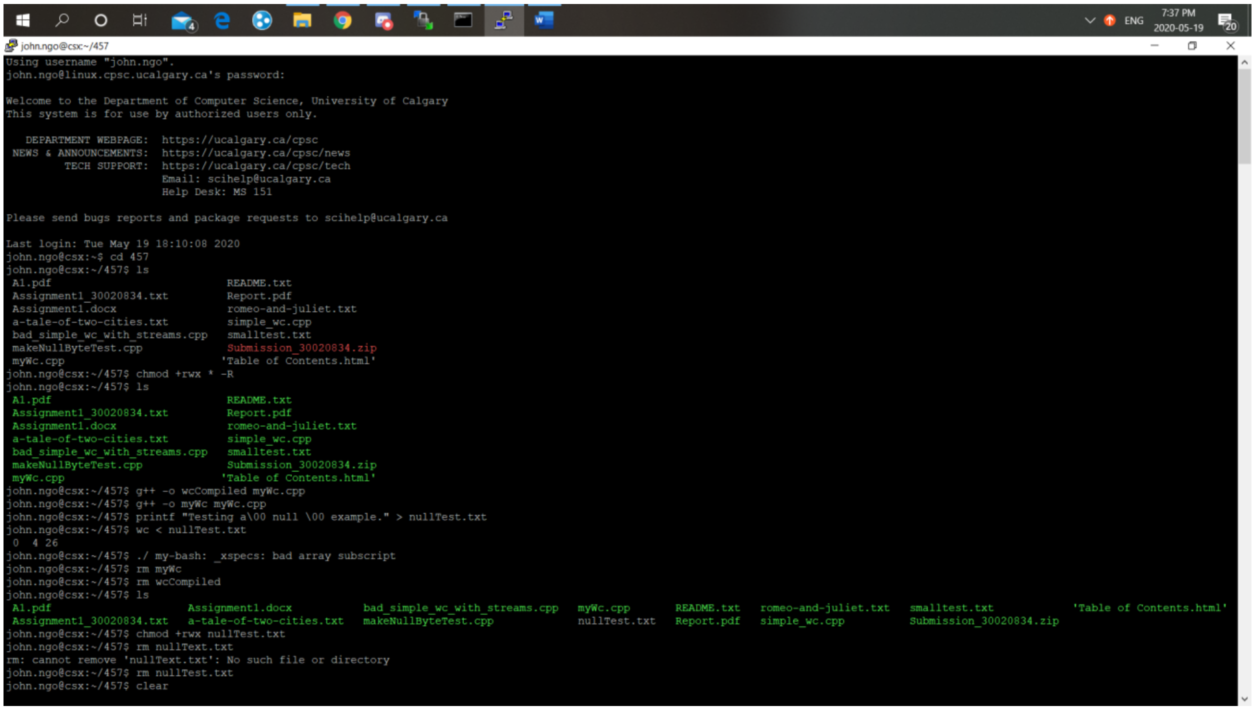
myWc:

```
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.014s
user    0m0.010s
sys     0m0.003s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.013s
user    0m0.012s
sys     0m0.001s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
```

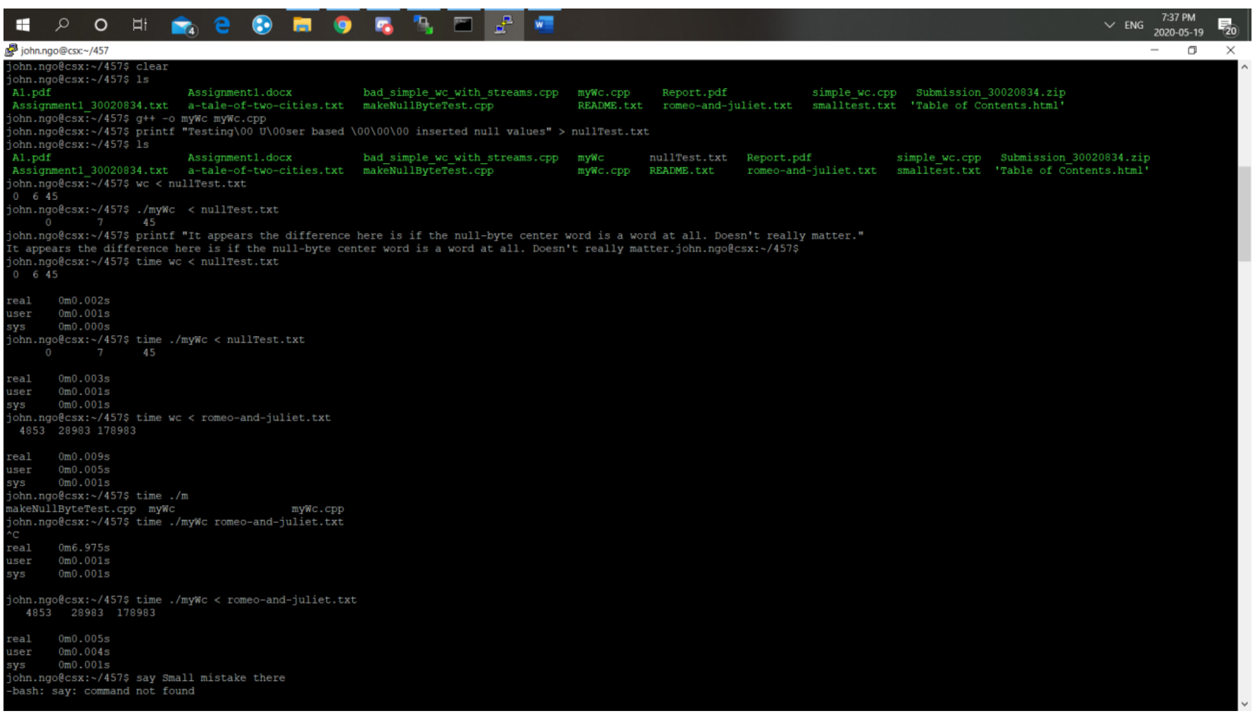
```
  16272  138883  804335

real    0m0.013s
user    0m0.011s
sys     0m0.002s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.013s
user    0m0.012s
sys     0m0.000s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.014s
user    0m0.012s
sys     0m0.002s
john.ngo@csx:~/457$
```

Eyeballing these numbers, based off the time average of these values, myWc is definitively faster for a relatively large input file, such as 'a tale of two cities', due primarily due to significantly less user processing time despite surprisingly similar amounts of kernal time.

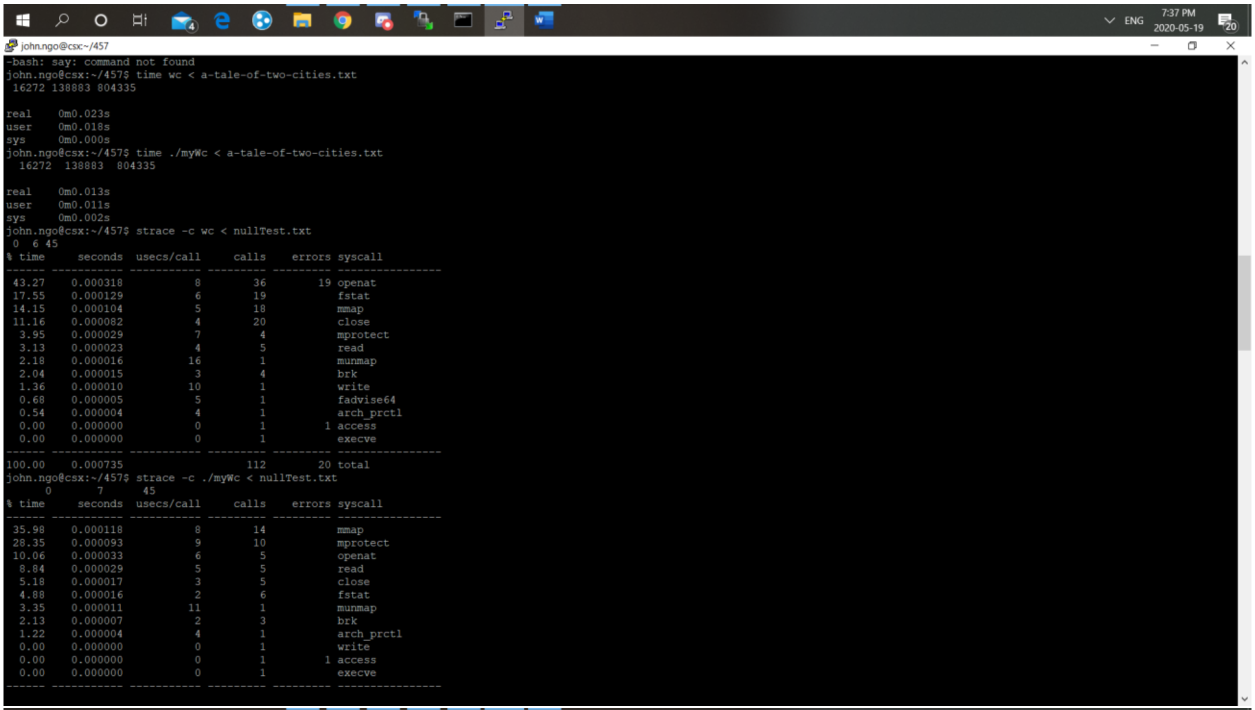
Screenshots of the above here:



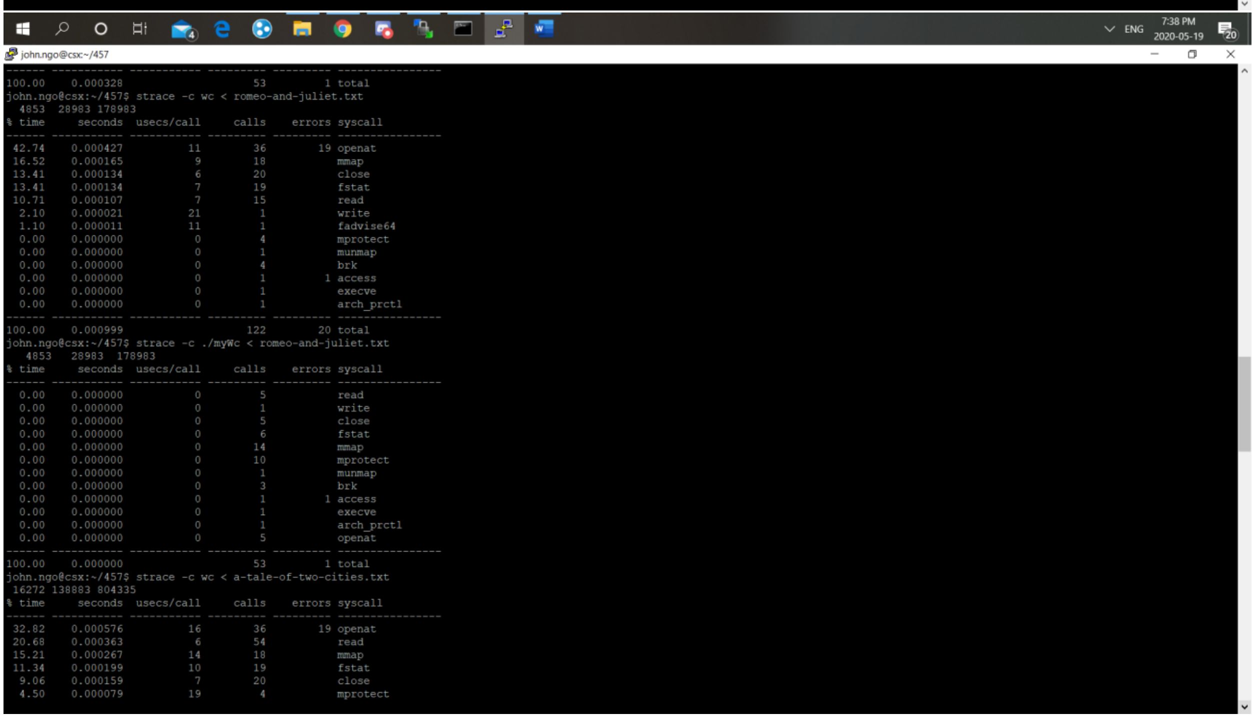
```
john.ngo@csx:~/457
Using username "john.ngo".
john.ngo@linux.cpsc.ucalgary.ca's password:

Welcome to the Department of Computer Science, University of Calgary
This system is for use by authorized users only.

   DEPARTMENT WEBPAGE:  https://ucalgary.ca/cpsc
 NEWS & ANNOUNCEMENTS:  https://ucalgary.ca/cpsc/news
         TECH SUPPORT:  https://ucalgary.ca/cpsc/tech
                        Email: scihelp@ucalgary.ca
                        Help Desk: MS 151

Please send bugs reports and package requests to scihelp@ucalgary.ca

Last login: Tue May 19 18:10:08 2020
john.ngo@csx:~$ cd 457
john.ngo@csx:~/457$ ls
 A1.pdf                          README.txt
 Assignment1_30020834.txt        Report.pdf
 Assignment1.docx                romeo-and-juliet.txt
 a-tale-of-two-cities.txt        simple_wc.cpp
 bad_simple_wc_with_streams.cpp  smalltest.txt
 makeNullByteTest.cpp            Submission_30020834.zip
 myWc.cpp                       'Table of Contents.html'
john.ngo@csx:~/457$ chmod +rwx * -R
john.ngo@csx:~/457$ ls
 A1.pdf                          README.txt
 Assignment1_30020834.txt        Report.pdf
 Assignment1.docx                romeo-and-juliet.txt
 a-tale-of-two-cities.txt        simple_wc.cpp
 bad_simple_wc_with_streams.cpp  smalltest.txt
 makeNullByteTest.cpp            Submission_30020834.zip
 myWc.cpp                       'Table of Contents.html'
john.ngo@csx:~/457$ g++ -o wcCompiled myWc.cpp
john.ngo@csx:~/457$ g++ -o myWc myWc.cpp
john.ngo@csx:~/457$ printf "Testing a\00 null \00 example." > nullTest.txt
john.ngo@csx:~/457$ wc < nullTest.txt
 0  4 26
john.ngo@csx:~/457$ ./ my-bash: _xspecs: bad array subscript
john.ngo@csx:~/457$ rm myWc
john.ngo@csx:~/457$ rm wcCompiled
john.ngo@csx:~/457$ ls
 A1.pdf                    Assignment1.docx          bad_simple_wc_with_streams.cpp   myWc.cpp       README.txt   romeo-and-juliet.txt   smalltest.txt             'Table of Contents.html'
 Assignment1_30020834.txt  a-tale-of-two-cities.txt  makeNullByteTest.cpp             nullTest.txt   Report.pdf   simple_wc.cpp          Submission_30020834.zip
john.ngo@csx:~/457$ chmod +rwx nullTest.txt
john.ngo@csx:~/457$ rm nullText.txt
rm: cannot remove 'nullText.txt': No such file or directory
john.ngo@csx:~/457$ rm nullTest.txt
john.ngo@csx:~/457$ clear
```



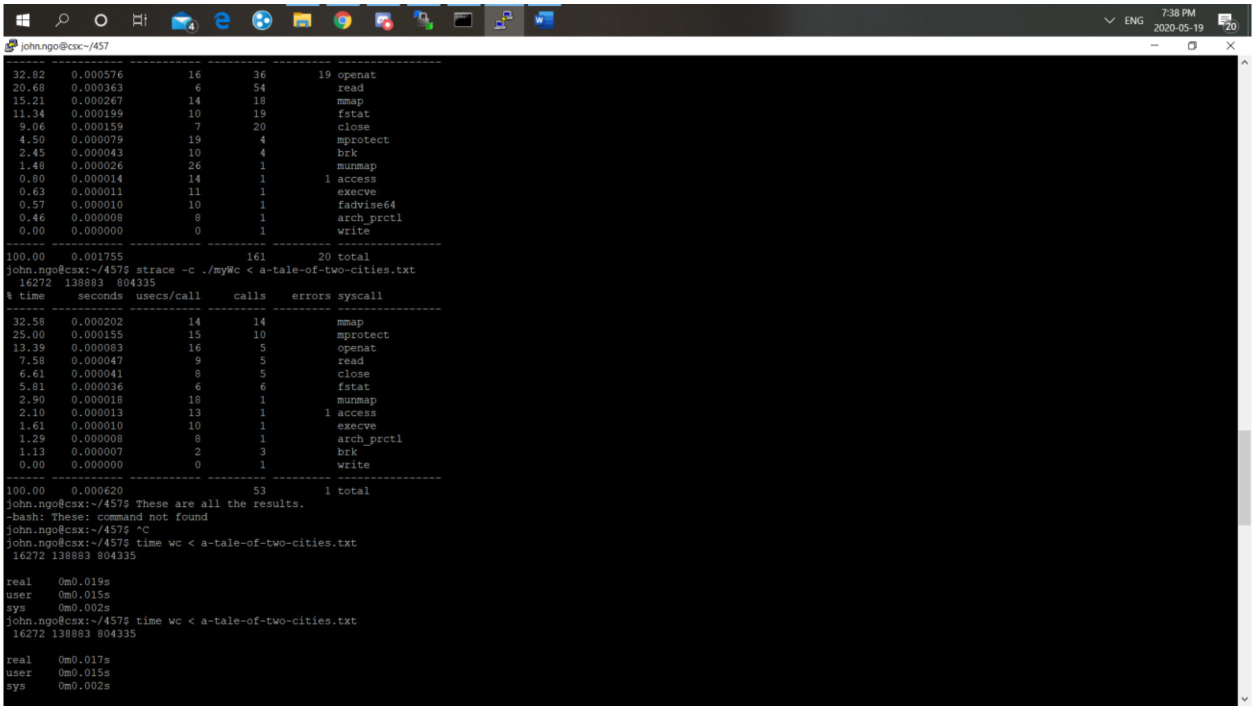
```
john.ngo@csx:~/457
john.ngo@csx:~/457$ clear
john.ngo@csx:~/457$ ls
 A1.pdf                    Assignment1.docx          bad_simple_wc_with_streams.cpp   myWc.cpp     Report.pdf             simple_wc.cpp   Submission_30020834.zip
 Assignment1_30020834.txt  a-tale-of-two-cities.txt  makeNullByteTest.cpp             README.txt   romeo-and-juliet.txt   smalltest.txt  'Table of Contents.html'
john.ngo@csx:~/457$ g++ -o myWc myWc.cpp
john.ngo@csx:~/457$ printf "Testing\00 U\00ser based \00\00\00 inserted null values" > nullTest.txt
john.ngo@csx:~/457$ ls
 A1.pdf                    Assignment1.docx          bad_simple_wc_with_streams.cpp   myWc       nullTest.txt   Report.pdf             simple_wc.cpp   Submission_30020834.zip
 Assignment1_30020834.txt  a-tale-of-two-cities.txt  makeNullByteTest.cpp             myWc.cpp   README.txt     romeo-and-juliet.txt   smalltest.txt  'Table of Contents.html'
john.ngo@csx:~/457$ wc < nullTest.txt
 0  6 45
john.ngo@csx:~/457$ ./myWc  < nullTest.txt
      0       7      45
john.ngo@csx:~/457$ printf "It appears the difference here is if the null-byte center word is a word at all. Doesn't really matter."
It appears the difference here is if the null-byte center word is a word at all. Doesn't really matter.john.ngo@csx:~/457$
john.ngo@csx:~/457$ time wc < nullTest.txt
 0  6 45

real    0m0.002s
user    0m0.001s
sys     0m0.000s
john.ngo@csx:~/457$ time ./myWc < nullTest.txt
      0       7      45

real    0m0.003s
user    0m0.001s
sys     0m0.001s
john.ngo@csx:~/457$ time wc < romeo-and-juliet.txt
  4853  28983 178983

real    0m0.009s
user    0m0.005s
sys     0m0.001s
john.ngo@csx:~/457$ time ./m
makeNullByteTest.cpp  myWc                  myWc.cpp
john.ngo@csx:~/457$ time ./myWc romeo-and-juliet.txt
^C
real    0m6.975s
user    0m0.001s
sys     0m0.001s

john.ngo@csx:~/457$ time ./myWc < romeo-and-juliet.txt
   4853   28983  178983

real    0m0.005s
user    0m0.004s
sys     0m0.001s
john.ngo@csx:~/457$ say Small mistake there
-bash: say: command not found
```



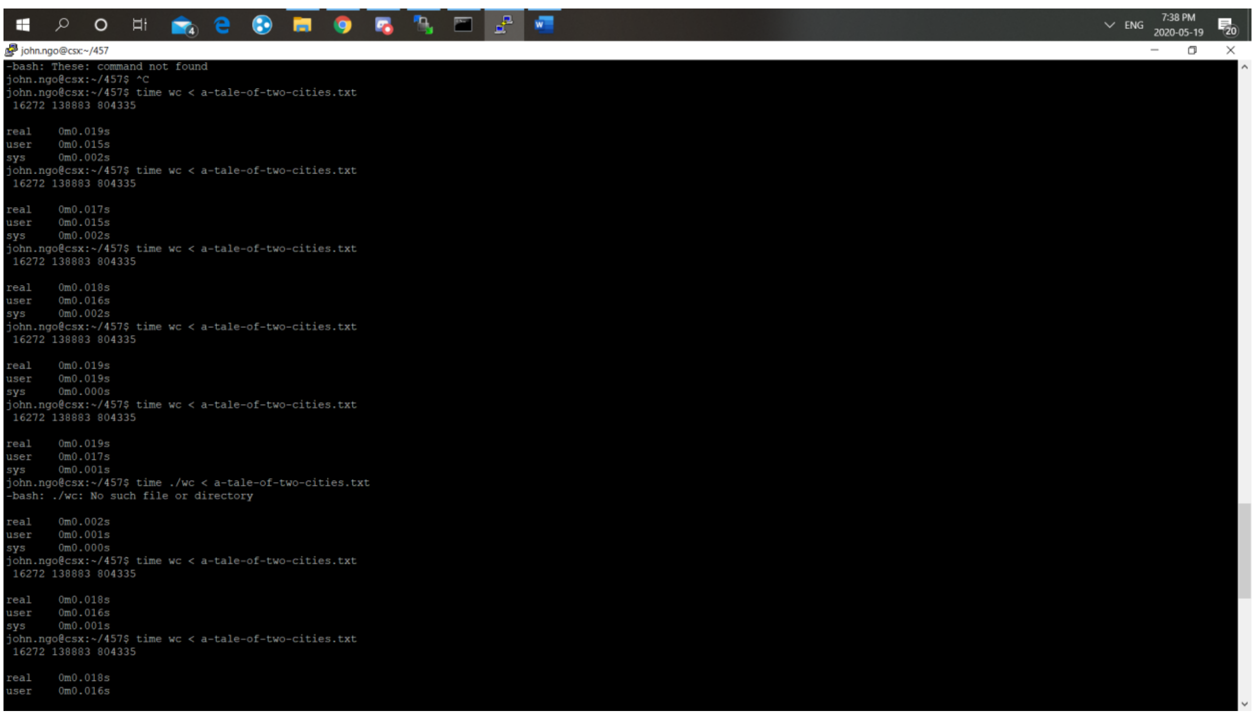
```
john.ngo@csx:~/457
-bash: say: command not found
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.023s
user    0m0.018s
sys     0m0.000s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335

real    0m0.013s
user    0m0.011s
sys     0m0.002s
john.ngo@csx:~/457$ strace -c wc < nullTest.txt
 0  6 45
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 43.27    0.000318           8        36        19 openat
 17.55    0.000129           6        19           fstat
 14.15    0.000104           5        18           mmap
 11.16    0.000082           4        20           close
  3.95    0.000029           7         4           mprotect
  3.13    0.000023           4         5           read
  2.18    0.000016          16         1           munmap
  2.04    0.000015           3         4           brk
  1.36    0.000010          10         1           write
  0.68    0.000005           5         1           fadvise64
  0.54    0.000004           4         1           arch_prctl
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.000735                   112        20 total
john.ngo@csx:~/457$ strace -c ./myWc < nullTest.txt
      0       7      45
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 35.98    0.000118           8        14           mmap
 28.35    0.000093           9        10           mprotect
 10.06    0.000033           6         5           openat
  8.84    0.000029           5         5           read
  5.18    0.000017           3         5           close
  4.88    0.000016           2         6           fstat
  3.35    0.000011          11         1           munmap
  2.13    0.000007           2         3           brk
  1.22    0.000004           4         1           arch_prctl
  0.00    0.000000           0         1           write
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
```



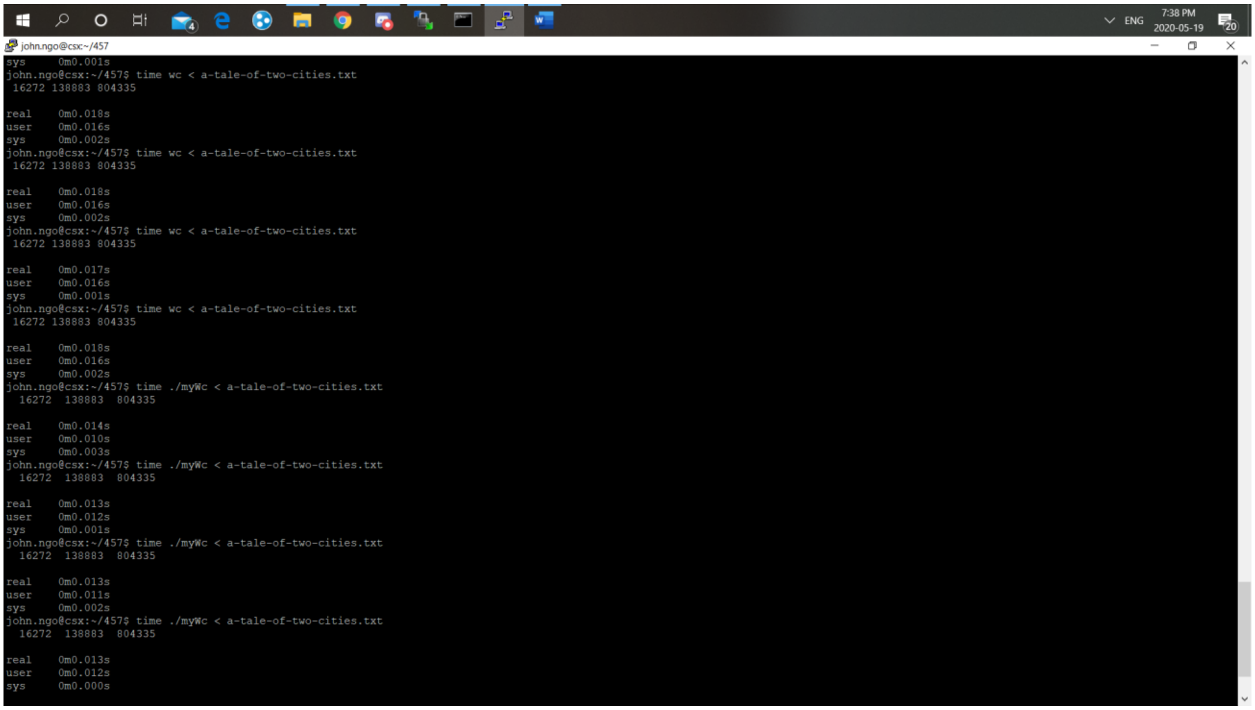
```
john.ngo@csx:~/457
------ ----------- ----------- --------- --------- ----------------
100.00    0.000328                    53         1 total
john.ngo@csx:~/457$ strace -c wc < romeo-and-juliet.txt
  4853  28983 178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 42.74    0.000427          11        36        19 openat
 16.52    0.000165           9        18           mmap
 13.41    0.000134           6        20           close
 13.41    0.000134           7        19           fstat
 10.71    0.000107           7        15           read
  2.10    0.000021          21         1           write
  1.10    0.000011          11         1           fadvise64
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         4           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.000999                   122        20 total
john.ngo@csx:~/457$ strace -c ./myWc < romeo-and-juliet.txt
   4853   28983  178983
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         6           fstat
  0.00    0.000000           0        14           mmap
  0.00    0.000000           0        10           mprotect
  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           arch_prctl
  0.00    0.000000           0         5           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    53         1 total
john.ngo@csx:~/457$ strace -c wc < a-tale-of-two-cities.txt
 16272 138883 804335
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.82    0.000576          16        36        19 openat
 20.68    0.000363           6        54           read
 15.21    0.000267          14        18           mmap
 11.34    0.000199          10        19           fstat
  9.06    0.000159           7        20           close
  4.50    0.000079          19         4           mprotect
```

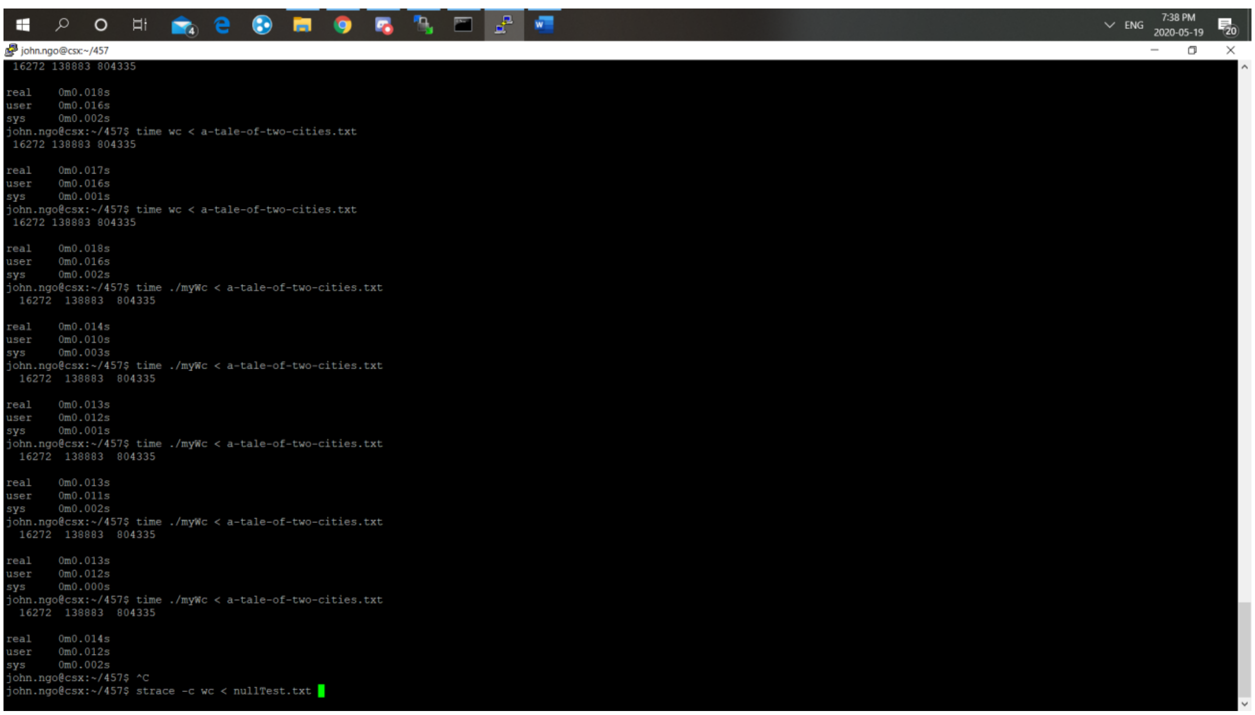


```
------ ----------- ----------- --------- --------- ----------------
 32.82    0.000576          16        36        19 openat
 20.68    0.000363           6        54           read
 15.21    0.000267          14        18           mmap
 11.34    0.000199          10        19           fstat
  9.06    0.000159           7        20           close
  4.50    0.000079          19         4           mprotect
  2.45    0.000043          10         4           brk
  1.48    0.000026          26         1           munmap
  0.80    0.000014          14         1         1 access
  0.63    0.000011          11         1           execve
  0.57    0.000010          10         1           fadvise64
  0.46    0.000008           8         1           arch_prctl
  0.00    0.000000           0         1           write
------ ----------- ----------- --------- --------- ----------------
100.00    0.001755                   161        20 total
john.ngo@csx:~/457$ strace -c ./myWc < a-tale-of-two-cities.txt
  16272  138883  804335
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.58    0.000202          14        14           mmap
 25.00    0.000155          15        10           mprotect
 13.39    0.000083          16         5           openat
  7.58    0.000047           9         5           read
  6.61    0.000041           8         5           close
  5.81    0.000036           6         6           fstat
  2.90    0.000018          18         1           munmap
  2.10    0.000013          13         1         1 access
  1.61    0.000010          10         1           execve
  1.29    0.000008           8         1           arch_prctl
  1.13    0.000007           2         3           brk
  0.00    0.000000           0         1           write
------ ----------- ----------- --------- --------- ----------------
100.00    0.000620                    53         1 total
john.ngo@csx:~/457$ These are all the results.
-bash: These: command not found
john.ngo@csx:~/457$ ^C
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.019s
user    0m0.015s
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.017s
user    0m0.015s
sys     0m0.002s
```



```
-bash: These: command not found
john.ngo@csx:~/457$ ^C
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.019s
user    0m0.015s
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.017s
user    0m0.015s
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.018s
user    0m0.016s
sys     0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.019s
user    0m0.019s
sys     0m0.000s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.019s
user    0m0.017s
sys     0m0.001s
john.ngo@csx:~/457$ time ./wc < a-tale-of-two-cities.txt
-bash: ./wc: No such file or directory

real    0m0.002s
user    0m0.001s
sys     0m0.000s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.018s
user    0m0.016s
sys     0m0.001s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
 16272 138883 804335

real    0m0.018s
user    0m0.016s
```


ENG
7:38 PM
2020-05-19
john.ngo@csx:~/457
sys 0m0.001s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.018s
user 0m0.016s
sys 0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.018s
user 0m0.016s
sys 0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.017s
user 0m0.016s
sys 0m0.001s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.018s
user 0m0.016s
sys 0m0.002s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.014s
user 0m0.010s
sys 0m0.003s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.012s
sys 0m0.001s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.011s
sys 0m0.002s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.012s
sys 0m0.000s


ENG
7:38 PM
2020-05-19
john.ngo@csx:~/457
16272 138883 804335
real 0m0.018s
user 0m0.016s
sys 0m0.002s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.017s
user 0m0.016s
sys 0m0.001s
john.ngo@csx:~/457$ time wc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.018s
user 0m0.016s
sys 0m0.002s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.014s
user 0m0.010s
sys 0m0.003s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.012s
sys 0m0.001s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.011s
sys 0m0.002s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.013s
user 0m0.012s
sys 0m0.000s
john.ngo@csx:~/457$ time ./myWc < a-tale-of-two-cities.txt
16272 138883 804335
real 0m0.014s
user 0m0.012s
sys 0m0.002s
john.ngo@csx:~/457$ ^C
john.ngo@csx:~/457$ strace -c wc < nullTest.txt