Aaron Goldsmith
Lab 4 - COEN 177

Screenshot of 100KB test

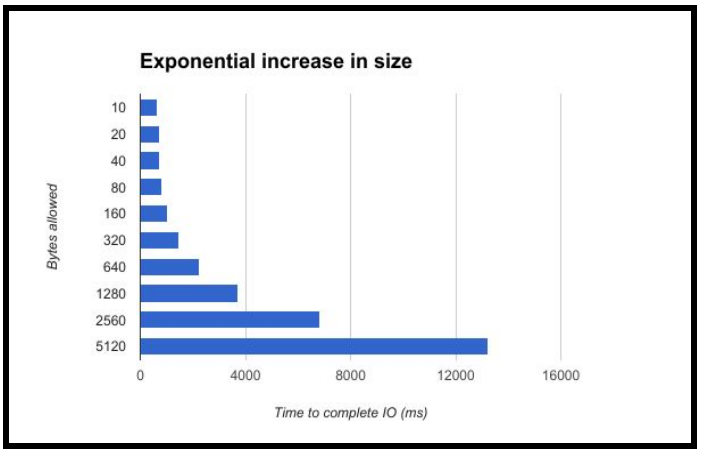
Test 1

In the image above, I captured both the properties panel and the text document (output)

In order to show how the characters that we see amount for physical space in memory.

In this example the size of the file was **100,000 Bytes = 100 KB = 0.1 MB**.

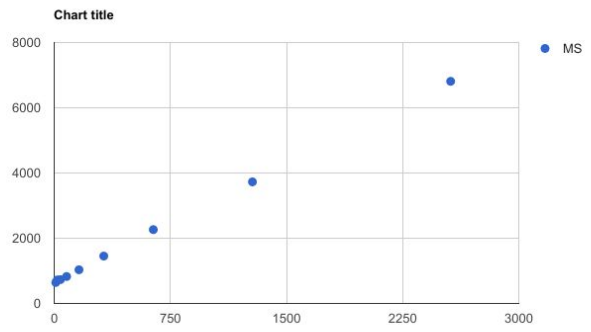
```
dcmac23:Documents agoldsmi$ python charStrm.py 10
643.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 20
721.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 40
731.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 80
826.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 160
1033.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 320
1453.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 640
2264.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 1280
3728.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 2560
6811.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 5120
13250.0 ms
dcmac23:Documents agoldsmi$
```

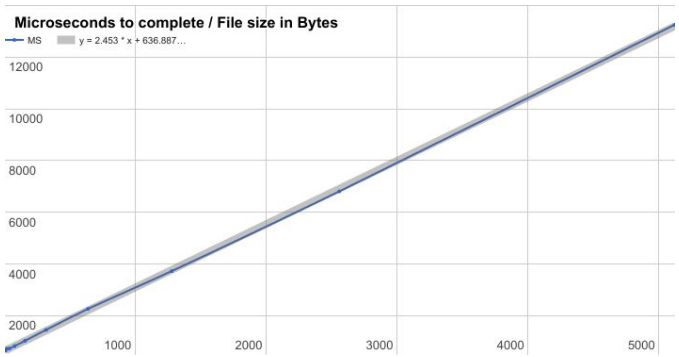


Test 2: running code in CLI

In this test I began with a character stream of 10 bytes, and and doubled it's size each iteration.

I wanted to see how similar the write speed was for small and large streams of data.





By increasing the file size **exponentially**, and viewing the graph logarithmically, the linear relationship between the two variables becomes more easily recognized.
Leading us to find an equation which maps each file size to a discrete time:
f(x) = 2.453(x) + 636.8

Responsive output

In this example, I wanted to demonstrate how the program will output answers in a more ‘legible’ way by changing the units when the data needs. I to print out the write speed in terms of microseconds as long as the write time took under a minute long. Otherwise, the data will be printed in ‘seconds’.

```
dcmac23:Documents agoldsmi$ python charStrm.py 50000000
121.60082  sec
dcmac23:Documents agoldsmi$ python charStrm.py 50000000
124.226222  sec
dcmac23:Documents agoldsmi$ python charStrm.py 50000000
123.436662  sec
dcmac23:Documents agoldsmi$ python charStrm.py 50000000
124.632681  sec
dcmac23:Documents agoldsmi$ python charStrm.py 50
804.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50
748.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50
614.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50
742.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50
753.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50
578.0 ms
dcmac23:Documents agoldsmi$ python charStrm.py 50000000
120.959571  sec
```

---

char stream	SIZE (bytes)	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial Averages
python charStrm.py 50000000	50,000,000 bytes	121.60 sec	124.22 sec	123.43 sec	124.63 sec	120.95 sec	**122.97 sec**
python charStrm.py 100000	5,000,000 bytes	12.41 sec	12.19 sec	12.16 sec	12.03 sec	12.20 sec	**12.20 sec**
python charStrm.py 100000	100,000 bytes	248186.0 ms	240617.0 ms	246873.0 ms	250385.0 ms	247889.0 ms	**246790.0 ms**
python charStrm.py 10000	10,000 bytes	25349.0 ms	25947.0 ms	26149.0 ms	25130.0 ms	24959.0 ms	**25506.8 ms**
python charStrm.py 1000	1,000 bytes	3000.0 ms	2970.0 ms	3174.0 ms	3073.0 ms	3100.0 ms	**3063.4 ms**
python charStrm.py 99	99 bytes	881.0 ms	841.0 ms	766.0 ms	824.0 ms	867.0 ms	**835.8 ms**
python charStrm.py 50	50 bytes	804.0 ms	748.0 ms	614.0 ms	742.0 ms	753.0 ms	**732.2 ms**

lab4.py	Number of bytes	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial Averages
`python stringwrite.py 50000000`	50,000,000 bytes	113.62 sec	113.41 sec	111.27 sec	114.37 sec	110.42 sec	**112.18 sec**
`python stringwrite.py 100000`	100,000 bytes	221584.0 ms	212352.0 ms	223387.0 ms	221952.0 ms	219581.0 ms	**219771.2 ms**
`python stringwrite.py 10000`	10,000 bytes	23432.0 ms	23119.0	22834.0 ms	22246.0 ms	22064.0 ms	**22644.0 ms**
`python stringwrite.py 1000`	1,000 bytes	2712.0 ms	2751.0 ms	2682.0 ms	2864.0 ms	2676.0 ms	**2737.0 ms**
`python stringwrite.py 99`	99 bytes	723.0 ms	861.0 ms	780.0 ms	826.0 ms	894.0 ms	**816.8 ms**
`python stringwrite.py 50`	50 bytes	726.0 ms	696.0 ms	578.0 ms	729.0 ms	752.0 ms	**696.2 ms**

From this data we can compare the average run times of the implementation of the write process, by running the programs *lab4.py* & *chrStrm.py* to see which process is able to write faster. From the data the write time for a string input was *on average* faster than that of a character stream input.

README:

In order to run this experiment determine
if you want to read or write first

1. Writing large String:
To test the speed of the String write call:

```
` python lab4.py BYTES`
```

NOTE: (BYTES = integer > 0)

2. Reading file given through stdin
To call the functions fastReadPipe() or line_reader()
Uncomment the respective line below in the code:

```
# fastReadPipe()
```

In the terminal type:

```
python lab4.py
"FILENAME.txt"
```

lab4.py

```
from datetime import date
import random
import sys
import os
import io
import time

def makeFile(length,fileExt):

        index = 0;     # keeps track of bytes written
        seed = ""      # an empty array is initialized to hold the generated string


        # Each iteration a single character written
        # By default, the `write()` function uses a UTF-8 character encoding which
        # holds the property of reserving 8-bits per character.

        with open('test-(%s)_%dbytes' % (fileExt,length), "w+") as file:

                while (index < length):
                        val = random.randint(65,125)   # ascii table range for {A-Z, a-z, 0-9}

                        seed += chr(val)
                                # string concatenation (variable to hold chars until finished )
                        index += 1
                file.write(seed)

# NOTE:  the command above, `with open` automatically handles closing the file after
#          completion

def reciever(byteSeq):
        newstr = str(byteSeq.decode()) # decodes to UTF-8 and casts to string
        print(newstr)

def fastReadPipe():
        fileName = str(sys.argv[1] if len(sys.argv) >= 1 else "")
        timerStart = time.time()
        with open(fileName) as f:
                data_in = f.read()
```

```
        print(time.time()-timerStart)

#  Important for driving program to isolate the write function as to minimize conflicts
#  i.e  if the user calls main(), and wants a verbose output, data won't be affected

def main(verbose=False):
        seedLen = int(sys.argv[1]) if len(sys.argv) >= 1 else 0
        fileCreation  = time.strftime('%S') # used only for file naming

        timerStart = time.time()              # the time in UTC seconds right now
        sstr = makeFile(seedLen,fileCreation) # call the function makeFile
        timerEnd = time.time()                # capture the time again right now

        if (verbose==True): reciever(sstr)    # call function if verbose is set (default =
False)

        diff = (timerEnd-timerStart) # change in time from t0 to t1 ( in seconds)

        #   NOTE: Remaining code doesn't serve function to the evaluation process
        #   - It was included to provide clarity for the person running the experiment

        MICRO_SECONDS = diff * (10**6)
        if(diff>1): print(str(diff) + '  sec')
        else: print(str(MICRO_SECONDS) + ' ms')


# fastReadPipe()    ## USE FOR TESTING READ SPEEDS
main()              ## USE FOR TESTING / DRIVING GENERATIVE WRITE FUNCTION
```

charStrm.py

```
import time
import random
import sys
import os
import io

def makeCHFile(length,fileExt):
   index = 0;
   # close after done writing

   with open('test-(%s)_%dbytes' % (fileExt,length), "w+") as file:
        while (index < length):
           val=random.randint(65,125)
           file.write(chr(val))
           index += 1

seedLen = int(sys.argv[1]) if len(sys.argv) >= 1 else 0

time.tzset()
now = time.strftime('%X')

timerStart = time.time()
sstr = makeCHFile(seedLen,now)
timerEnd = time.time()
diff = (timerEnd-timerStart) # t2 - t1

               # it is important to use the end timer as our point of reference
               # because we want that to be the same for both sec or ms,
               # regardless of CPU time to print either statement.
```

```
if(diff>1): print(str(diff) + '  sec')
else: print(str(diff * (10 ** 6)) + ' ms') # (10^6) * diff
```