heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Romain Albasini

08.03.2106

Experiment conditions

The program shows the performance differences depending on the IO processing implementation. Writing and reading operations are compared, processing bytes in blocks, one by one, using buffered streams or not.

A quite decent amount of bytes (10'485'760) will be written/read using two different approaches to highlight the interest of using streams.

Results

This is the output produced by the program on my machine, as seen in the `.csv` file, is resumed on the following array:

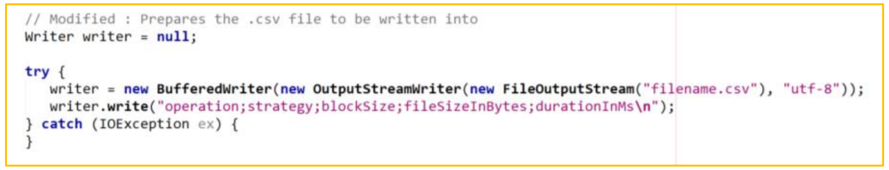
operation	strategy	blockSize	fileSizeIn Bytes	durationI nMs
WRITE	BlockByBlockWithBufferedStream	500	10485760	28
WRITE	BlockByBlockWithBufferedStream	50	10485760	32
WRITE	BlockByBlockWithBufferedStream	5	10485760	84
WRITE	ByteByByteWithBufferedStream	0	10485760	293
WRITE	BlockByBlockWithoutBufferedStream	500	10485760	77
WRITE	BlockByBlockWithoutBufferedStream	50	10485760	449
WRITE	BlockByBlockWithoutBufferedStream	5	10485760	4343
WRITE	ByteByByteWithoutBufferedStream	0	10485760	20786
READ	BlockByBlockWithBufferedStream	500	10485760	11
READ	BlockByBlockWithBufferedStream	50	10485760	20
READ	BlockByBlockWithBufferedStream	5	10485760	39
READ	ByteByByteWithBufferedStream	0	10485760	45
READ	BlockByBlockWithoutBufferedStream	500	10485760	36
READ	BlockByBlockWithoutBufferedStream	50	10485760	336
READ	BlockByBlockWithoutBufferedStream	5	10485760	3182
READ	ByteByByteWithoutBufferedStream	0	10485760	15273

Modifications

In order to export the results of the program into the .csv file, 3 functions have been slightly modified. The idea is to export useful information into our `.csv` file. A `Writer` object was used.

`main()`

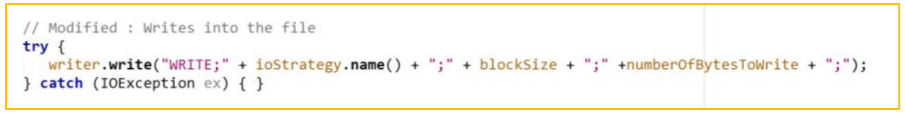
Implementation of the `Writer` to be able to write into a file.



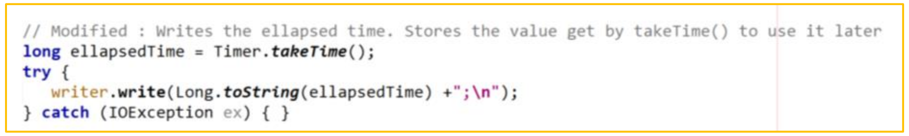
```
// Modified : Prepares the .csv file to be written into
Writer writer = null;

try {
   writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("filename.csv"), "utf-8"));
   writer.write("operation;strategy;blockSize;fileSizeInBytes;durationInMs\n");
} catch (IOException ex) {
}
```

`produceTestData()`

Use of the `writer` to export useful information into our `.csv` file



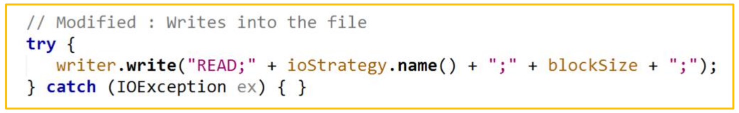
```
// Modified : Writes into the file
try {
   writer.write("WRITE;" + ioStrategy.name() + ";" + blockSize + ";" +numberOfBytesToWrite + ";");
} catch (IOException ex) { }
```



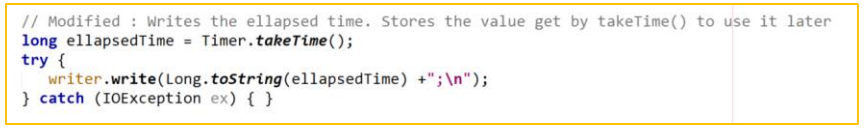
```
// Modified : Writes the ellapsed time. Stores the value get by takeTime() to use it later
long ellapsedTime = Timer.takeTime();
try {
   writer.write(Long.toString(ellapsedTime) +";\n");
} catch (IOException ex) { }
```

`consumeTestData()`

Same idea as in the `produceTestData` function. `numberOfBytesToWrite` is no longer used. Read bytes are analyzed in this case

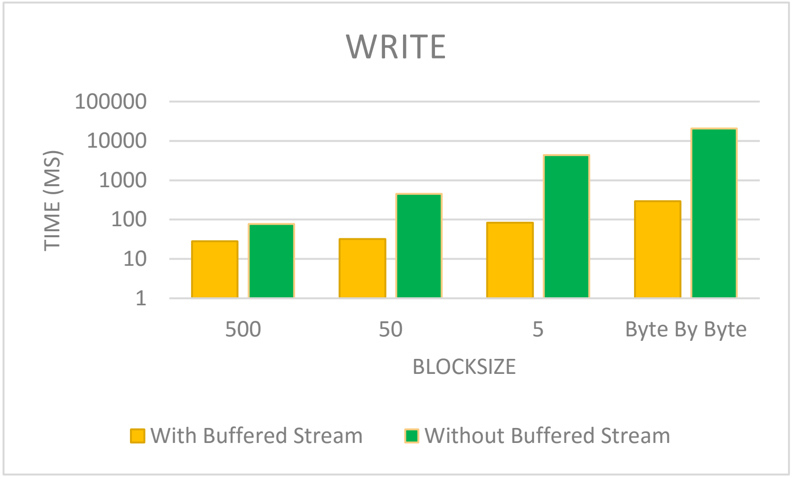


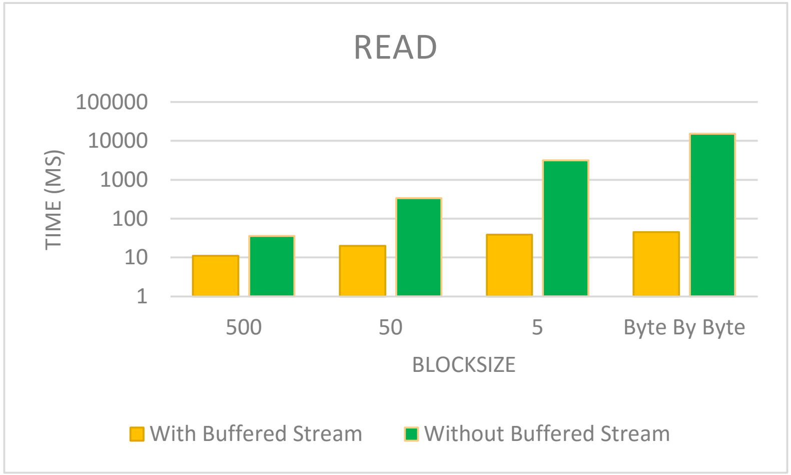
```
// Modified : Writes into the file
try {
   writer.write("READ;" + ioStrategy.name() + ";" + blockSize + ";");
} catch (IOException ex) { }
```



```
// Modified : Writes the ellapsed time. Stores the value get by takeTime() to use it later
long ellapsedTime = Timer.takeTime();
try {
   writer.write(Long.toString(ellapsedTime) +";\n");
} catch (IOException ex) { }
```

Results analysis





NB: Even if not represented on the graphics to improve readability, both functions have written/read 10’485’760 bytes.

Comments

As seen on the graphics, a buffered stream largely improves the performances of our program. The smaller the block size, the larger the difference.

The efficiency of using a stream remains the same, using the `produceTestData()` (WRITE) or the `consumeTestData()` (READ) function.

As seen during the lesson, stream improve performance. It is much more efficient to read/write big chunks of data instead of doing it byte by byte (cf the beer example). In both cases, best running time is obtained with the largest block size (500 in our case).

The performance impact of the absence of stream can be easily be seen with smaller block size. Decreasing the block size increases by about 300% the running time using streams, and by more than 1000% without.

Experiment conditions

Windows 10 x64, Intel I7-6700, 16 go RAM

GitHub link

My repo can be accessed at the following address:
https://github.com/ralbasini/Exercices/tree/master/01-BufferedIOBenchmark/BufferedIOBenchmark