Verslag opdracht 1

Bestanden lezen en schrijven

Ik maak gebruik van C++ en om bestanden te lezen en te schrijven maak ik gebruik van de iostream en fstream library's.
voor het inlezen van een bestand gebruik ik dus volgende code:

```
char * memblock;
streamsize size;
ifstream file (filename,ios::in | ios::ate | ios::binary);
  if (file.is_open()){              //voer geen code uit op de file als ze niet open is
        size = file.tellg();
        file.seekg (0, ios::beg);
        memblock = new char[size];
        file.read(memblock, size);      //steek de file in de memblock buffer
        file.close();
  }
```

de ifstream constructor neemt als argumenten een filename en een aantal voorgedefinieerde flags. Deze flags kunnen met elkaar gecombineerd worden met de | operator.
de 'ate' flag is hier belangrijk omdat deze onze get pointer aan het einde van de file staat. Dit doe ik om gemakkelijk met 'tellg()' de grootte van de file uit te lezen.

Het schrijven van files doe ik met volgende code:

```
ofstream wfile (filename,ios::out | ios::trunc | ios::binary);
  if(wfile.is_open()){
        wfile.write(memblock, size);
        wfile.close();
  }
```

Hier geldt hetzelfde principe als bij de vorige fstream constructor dat er flags kunnen doorgegeven worden. Nu is de trunc flag een belangrijk gegeven omdat deze ervoor zorgt dat we de file gaan overwriten.

Gegevens inlezen van de command line

In C++ kan je in de main functie 2 parameters toevoegen, een int en een char * .
in de char * komt een 2d array van type char of een array van strings. De eerste string in deze array is het commando waarmee de code werd opgeroepen, het 2de element is de eerste parameter die wordt meegegeven aan het commando …
met deze parameters kunnen variabelen worden ingelezen als volgt:

```
int main(int argc, char* argv[]){
  string filename = "testing.bin";
  int fileSize = 1024;
  if ( argc > 1 ){                    //het 1ste argument wordt altijd de filename
        filename = argv[1];
  }
  if ( argc > 2 ){                    //het 2de argument wordt altijd de file size
        fileSize = stoi (argv[2],nullptr); //string to int functie standard lib
  }

  mkFile(fileSize,filename);
  return 0;
}
```

Tijdmeting

Voor de tijdmeting maak ik gebruik van std::chrono::high_resolution_clock::now() om een tijdswaarde te bekomen bij start en finish deze waarden worden dan vergeleken met volgende functie:

chrono::duration_cast<chrono::nanoseconds>(finish - start).count()

Deze functie heeft als return value een waarde in nanoseconden die de tijd weergeeft tussen start en finish waarden.
De chrono lib is beschikbaar vanaf C++11

Compileren

Voor het compileren van de applicatie maak ik gebruik van de make utility met bijhorende makefile (zie bijlage hieronder)

De executable die gemaakt wordt kan nu gewoon worden uitgevoerd op de standaart manier met ./filename.

Resultaten

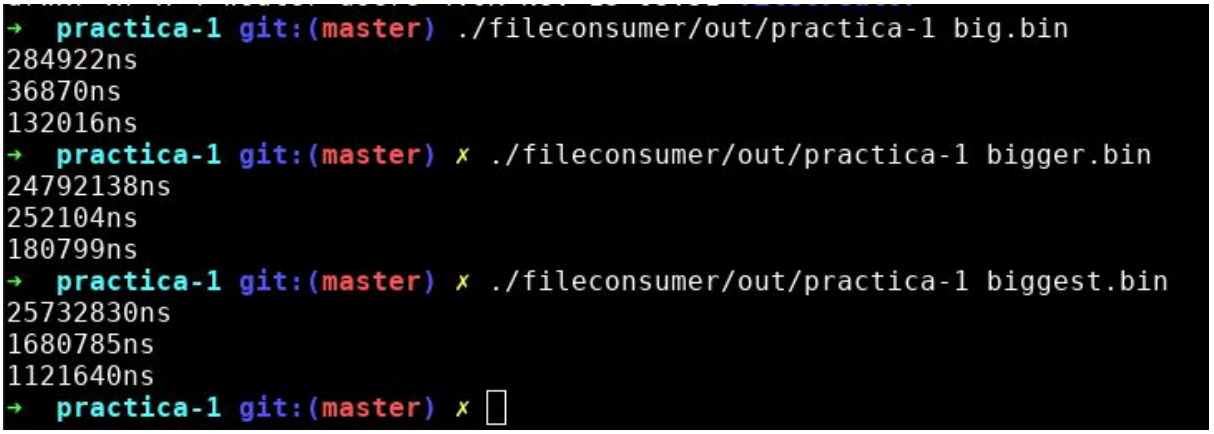
systeem

Alle tests zijn uitgevoerd op een Dell xps13 9360 met 16gig ram en processor:
Intel(R) Core(TM) i7-7560U CPU @ 2.40GHz.
De harde schijf is een PCIe SSD.

tests

Ik heb 3 files aangemaakt met de filecreator tool van toenemende grootte
-rw-r--r-- 1 wouter users 10K Nov 13 04:39 big.bin
-rw-r--r-- 1 wouter users 100K Nov 13 04:39 bigger.bin
-rw-r--r-- 1 wouter users 1.0M Nov 13 04:39 biggest.bin

nu ga ik over de files met de fileconsumer tool en krijg volgende resultaten:



De applicatie print steeds eerst de tijd nodig om de file in te lezen dan de processortijd dan de tijd nodig om de file te schrijven.

Conclusie

Het is zichtbaar door de tests dat lees en schrijf operaties trager zijn dan cpu operaties zelfs op een systeem met een zeer snelle SSD. de schrijf operatie is ook aanzienlijk sneller waarschijnlijk dankzij cashing.

Bronnen

https://en.cppreference.com/
http://www.cplusplus.com/

Broncode

FileCreator

main.cpp

```
 1 #include <iostream>
 2 #include <fstream>
 3
 4 using namespace std;
 5
 6
 7 int mkFile(int size,string filename){
 8   char* memblock;
 9   memblock = new char[size];
10   for( int i = 0; i < size; i++){ //fill up some space with random data
11     memblock[i] = rand()%255;
12   }
13   ofstream file (filename, ios::out | ios::trunc | ios::binary); //file writing sequence
14   if (file.is_open()){
15     file.write(memblock, size);
16     file.close();
17   }
18   delete[] memblock;
19   return 0;
20 }
21
22
23 int main(int argc, char* argv[]){
24   string filename = "testing.bin";//default file name and size if none are given
25   int fileSize = 1024;
26   if ( argc > 1 ){      //first argument becomes filename
27     filename = argv[1];
28   }
29   if ( argc > 2 ){      //second argument becomes filesize
30     fileSize = stoi (argv[2],nullptr); // string to int function
31   }
32
33   mkFile(fileSize,filename);//see defenition above
34   return 0;
35 }
36
37
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
ValueError: Still no compile flags, no completions yet.                    37,0-1        All
```

FileConsumer

main.cpp

```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <chrono>
 4
 5 using namespace std;
 6
 7 // little function to do some processor work
 8 int cpuAddOneToArray(char* array, int size){
 9   for (int i = 0; i < size; i++){
10     array[i]++;
11   }
12   return 0;
13 }
14
15
16 int main(int argc, char* argv[]){
17   char * memblock;
18   streamsize size;
19   string filename = "testing.bin";    //default filename
20   if ( argc > 1 ){
21     filename = argv[1];
22   }
23
24   auto start = chrono::high_resolution_clock::now();  //take starttime
25   ifstream file (filename,ios::in | ios::ate | ios::binary); // file reading sequence
26   if (file.is_open()){
27     size = file.tellg();
28     file.seekg (0, ios::beg);
29     memblock = new char[size];
30     file.read(memblock, size);
31     file.close();
32   }
33   auto finish = chrono::high_resolution_clock::now(); //take endtime
34   //print the time it took to the console
35   cout << chrono::duration_cast<chrono::nanoseconds>(finish - start).count() << "ns\n";
36   start = chrono::high_resolution_clock::now();
37   cpuAddOneToArray(memblock,size); //room heating :)
38   finish = chrono::high_resolution_clock::now();
39   cout << chrono::duration_cast<chrono::nanoseconds>(finish - start).count() << "ns\n";
40   start = chrono::high_resolution_clock::now();
41   ofstream wfile (filename,ios::out | ios::trunc | ios::binary); //file writing sequence
42   if(wfile.is_open()){
43     wfile.write(memblock, size);
44     wfile.close();
45   }
46   finish = chrono::high_resolution_clock::now();
47   cout << chrono::duration_cast<chrono::nanoseconds>(finish - start).count() << "ns\n";
48   return 0;
49 }
50
51
~
~
~
-- INSERT --                                                    51,1          All
```

Makefile

```
 1 #OBJS specifies wicht files to compile
 2 OBJS = src/main.cpp
 3
 4 #CC spesifies compiler
 5 CC = g++
 6 #CW is compiler for windows
 7 CW = i586-mingw32msvc-g++
 8 #COMPILER_FLAGS aditional options to use
 9 #-w suppress all warnings
10 COMPILER_FLAGS = -Wall -std=c++11
11
12 #LINKER_FLAGS linked libraries
13 LINKER_FLAGS =
14 LINKER_FLAGS_WIN =
15
16 #OBJ_NAME name of the compiled file
17 OBJ_NAME = out/practica-1
18
19 #compile to executable
20 all : $(OBJS)
21   $(CC) $(OBJS) $(COMPILER_FLAGS) $(LINKER_FLAGS) -o $(OBJ_NAME)
22
23
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"filecreator/Makefile" 23L, 483C                                23,0-1        All
```