Silesian University of Technology Gliwice

Faculty of Automatic Control,
Electronics and Computer Science

Academic year 2008/2009

Macrocourse, sem. 8

Programming in Assembler
Project

SHA1 hashing algorithm.

A report by:
Szymon Sobik
September 21, 2009

1 Project requirements and assumptions.

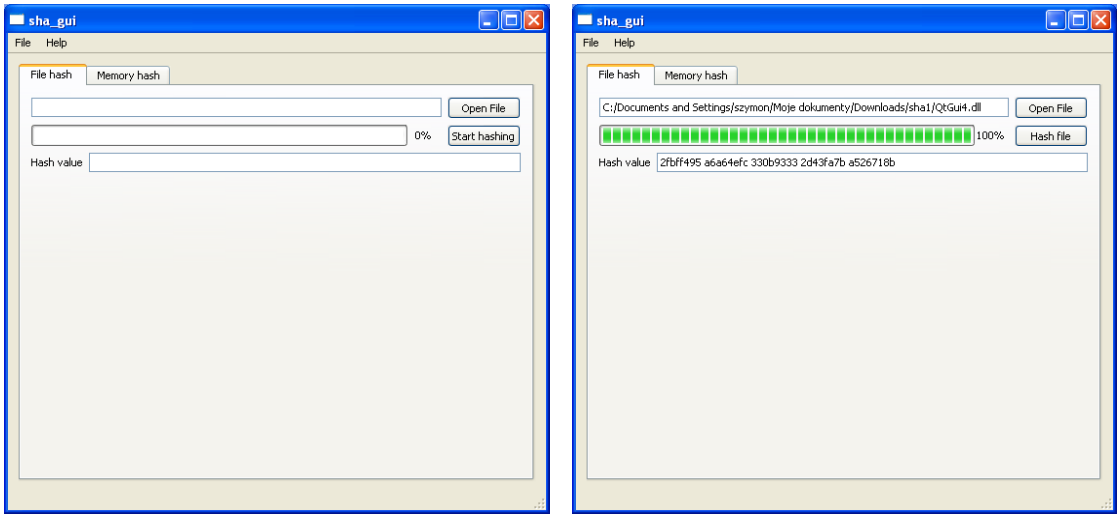
Project will be created as a DLL library. It will be used in a c/c++ project. Memory hashing will require a continious block of data, the function will take 1 pointer (beggining of the block) and block size (in bytes). The memory hashing will be constrained by a 4GB limit. File hashing will take a path to the file.

2 External specification

The program consists of 2 panels. One is for hashing files, the other for hashing memory (inputted text in this case).

2.1 File hashing

To hash a file one needs to select a file with *Open file* button, and then press *Start hashing*.



(a) the hashing window (b) finished hahsing

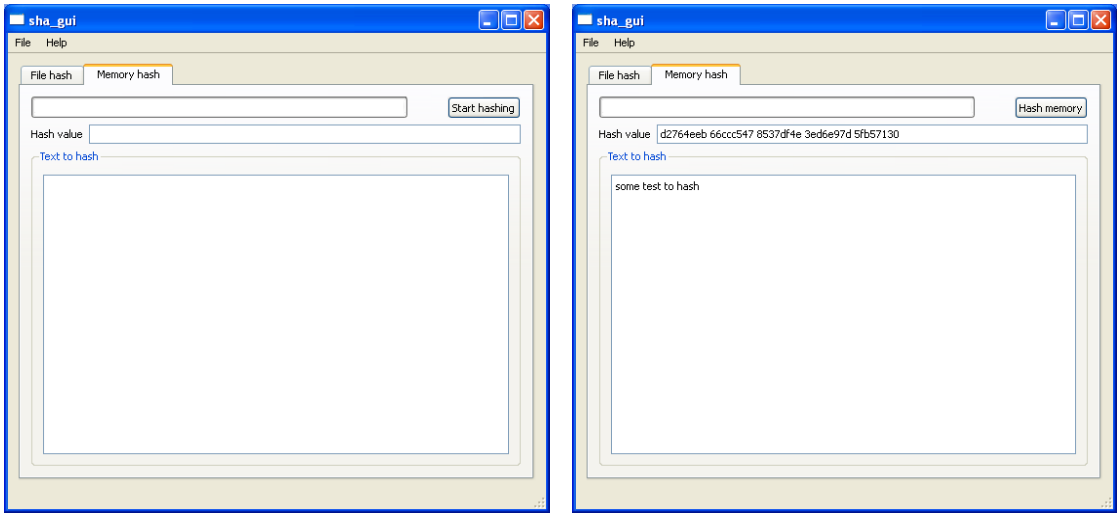
Figure 1: File hashing tab

The button changes to *Abort*, one can use it to cancel long hashing operation. The progress of hashing is displayed on the bar. Once hashing is finished the resulting number is displayed in the field labelled with *Hash value*

2.2 Memory hashing

Memory hashing is similar to file hashing.

The text to hash should be inputted in the *Text to hash* area. The progress bar display is a bouncing rectangle (displaying operation in progress).



(a) the hashing window (b) finished hahsing

Figure 2: Memory hashing tab

3 Internal specification

The library requires two include files.

Code 1 Include files

```
INCLUDE \masm32\include\windows.inc
INCLUDE \masm32\include\kernel32.inc
```

There are 3 global variables and one constant (text macro). The text macro *FBUFFSIZE* describes the file buffer size. It needs to be a multiply of 64. The 3 variables are

1. progress – used for storing progress of hashing.
2. w – internal use of proc512 function
3. tt – internal use for storing hash value

The two macros are used for 64bit*32bit multiply operation (*mul64*) and 64bit+32 (*add64*)adding operation.

Code 2 Costants varialbes and macros

```
FBUFFSIZE TEXTEQU <512>

progress QWORD 0;
w DWORD 80 dup(0);
tt DWORD 5 dup(0);

mul64 MACRO qvar, val
add64 MACRO qvar, var
```

There are 5 functions in the assembly, 3 of which are for outside use. *DllEntry* is used for loading library (system use).
hashFile is used for hashing files, it accepts pointer to an5*32bit array and pointer to string describing file location.
hashMem is used for hashing memory, is accepts pointer to an5*32bit array, base address of emory to hash and size to hash.
readProgress is used for retrieing the value of progress variable, it returns a 64bit value (*edx:eax*).
proc512 is an internal function used for hashing 512bit blocks of data, is accepts pointer to the block and pointer to the hash array 5*32bit

Code 3 Functions

```
DllEntry PROC hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
hashFile PROC USES eax ecx ebx edx, hash:PTR DWORD, fName:PTR BYTE
hashMem PROC USES eax ecx ebx edx, hash:PTR DWORD, membase:PTR BYTE, s:
    DWORD
readProgress PROC USES ecx
proc512 PROC USES ecx ebx edx esi edi, chptr:PTR DWORD, ht:PTR DWORD
```

4 Testing

The testing was fairly easy as there are multiple C libraries available. I used the code of one of them and compared the intermediate results. The most time consuming and error prone was development of *proc512* function so it was testes the most.

The *hashFile* and *hashMem* could be used from the gui and result compared with output of many CLI hashing programs.

4.1 Memory hashing test case

The phrase ”The quick brown fox jumps over the lazy dog” can be a good example for testing memory hashing. The resulting hash is ”*2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12*” [Wikipedia[1]]

4.2 File hashing test case

For a file test one can use the supplied QtGui4.dll file. The UNIX command sha1sum[2] run on QtGui4.dll returns ”*2fbff495a6a64efc330b93332d43fa7ba526718b QtGui4.dll*” The result is the same as in my program.

5 Compiling

The program uses Qt4 for GUI and thread handling, one has to have the development libraries for the compiler. As there are provided binaries and source for mingw, one needs to compile

[1] http://en.wikipedia.org/wiki/Examples_of_SHA_digests
[2] http://en.wikipedia.org/wiki/Sha1sum

the source by hand for MSVC. Alternatively one can compile the library in Visual Studio and gui in mingw. The latter method is less time consuming.

One has to download the needed library – Qt4[3]. Install the library, and mingw if needed. In order for this to work ensure that the bin folder of Visual Studio
(`<Visual Studio installation path>/VC/bin`)
, Qt4
(usually `C:/Qt/4.5.2/bin` or similar)
and optionally mingw
(`C:/MinGw/bin`)
are set in the path system variable.

5.1 Visual Studio

1. To prepare the library use command line (cmd).
2. Navigate to main Qt4 directory (where configure.exe is)
3. Run configure.exe select open source licence and agree to the licence.
4. Run make. **The process takes a really long time.**
5. Download Visual Studio Qt plugin[4].
6. Configure the plugin. (menu Qt -> Qt options)
7. Select the build type and build the application.

5.2 mingw

Since I had trouble with release application on Visual Studio I used mingw compiler for gui and MSVC for library.

1. Turn off compilation of gui. (right click on project -> Properties -> Configuration properties, tick off build on sha_gui)
2. Build the dll.
3. Navigate to project sha_gui directory.
4. run ”qmake -project” (make a project file)
5. ”qmake” (prepare makefiles)
6. ”make -f Makefile.Release” for release or ”make -f Makefile.Debug” for debugging
7. Copy the .exe to the directory where the library is.

The second method has one drawback – debugging in single environment is impossible.

[3] `http://qt.nokia.com/downloads` – LGPL/Free – 4.5 for Windows
[4] `http://qt.nokia.com/downloads` – LGPL/Free – Visual Studio add-in

6 Conclusions

Coding hashing (or other CPU intensive) algorithms may be really beneficial performance wise, but it also has some drawbacks. The code is very error prone. similar C code would be cleaner and much more readable. Also optimizing is more difficult with assebly. The *proc512* function is not well optimized in my code. It is more difficult to do manual loop unwinding and variable exchanege similar to high level languages.