Contents

1 Basic Test Results

```
1  g++ -c -g -Wall -Wno-unused-value -std=c++11 osm.cpp
2  ar rcs libosm.a osm.o
3  rm -f ex1.tar osm.o libosm.a
```

2 README

```
itamakatz, liavst2
Itamar Katz (555792977) , Liav Steinberg (203630090)
EX: 1

===================================
=             FILES               =
===================================

- README - this file.
- makefile
- osm.cpp - implementation of osm.h

===================================
=            ANSWERS              =
===================================

Task 1:
======
The program "WhatIDo" creates a directory "Welcome" with the syscall
"mkdir" and inside of it another directory named "To", also with "mkdir".
Then,  the program opens a file inside of "To", named "OS", for write only
(O_WRONLY). The returned value 3 is the file descriptor used to "identify"
the file, and used later in "write" to write to the file a message that says
"If you didn't read the course guidelines, please do it now!". Then the
program removes the file and the directories (using "unlink" and "rmdir").

Task 2:
======
The osm_init and finalizer are called before and after every other library
function. We implemented them to open and close one file for the disk
access timing function. In the four main functions of the library we used
the gettimeofday function to measure the time differences before and after
the loop. In the loop we used loop unrolling, meaning in each iteration we
added 5 more independent operation, excluding the one that we wanted to time.
The reason for that is decreasing the assembly-language code meant to check
the loop condition, so it would not affect the time measuring of the wanted
operation. The 6 operation inside the loop are independent, so they would,
as well, not affect the time measuring by decreasing the delay on the machine
pipline. We returned the time differnce between the two calls of gettimeofday,
divided by the number of the iterations to get the average time.

*Notice that we counted on the user to use osm_init and osm_finalizer before
 and after using the library, and hence we started with "fclose" operation
 and ended with "fopen" in the osm_disk_time loop , knowing that the file
 will be open and closed properly.
```

3 Appeal

```
itamakatz, liavst2
Itamar Katz (555792977) , Liav Steinberg (203630090)
EX: 1

=====================================
=              DESCRIPTION               =
=====================================

1.  In the function measureTimes, we didn't allocate memory for
    "machineName", so the program terminated with segmentation fault.
    (it worked on Code::Blocks, but on Clion it threw seg fault)
    therefore, we added the following line:
    "times.machineName = new char[HOST_LEN]; " where
    HOST_LEN = 1024 (on line 208 in osm.cpp).
    That fixed the problem.

2.  Also, there were "operation has no effect" warnings in the
    makefile compilation (because of the operation_time function).
    So we added the specific "-Wno-unused-value" flag to suppress them.
     The compilation now is clean of warnings.

3. Line 234 exceeded, fixed.

Those 2 lines are the only lines that were changed.
```

4 Makefile

```
CFLAGS = -g -Wall -Wno-unused-value -std=c++11
TAR_NAME = ex1.tar
SOURCES = osm.cpp
HEADERS = osm.h
OSMLIB = libosm.a
OBJS = $(SOURCES:.cpp=.o)
EXTRA_FILES = README Makefile
TAR_FILES = $(SOURCES) $(EXTRA_FILES)

.DEFAULT_GOAL = $(OSMLIB)


all: $(OSMLIB) tar


$(OBJS): $(SOURCES) $(HEADERS)
    $(CXX) -c $(CFLAGS) $<


$(OSMLIB): $(OBJS)
    ar rcs $@ $^

tar:
    tar -cvf $(TAR_NAME) $(TAR_FILES)


clean:
    rm -f $(TAR_NAME) $(OBJS) $(OSMLIB)


.PHONY:
    all tar clean
```

5 osm.cpp

```cpp
#include <sys/time.h>
#include <cmath>
#include <sys/unistd.h>
#include <stdio.h>

#include "osm.h"

#define NANO_ADJUST_TH 1000
#define NANO_ADJUST_BIL 1000000000
#define FAILURE -1
#define SUCCESS 0
#define DEFAULT 1000
#define INSTRUCTION_LOOP_GROUP 6
#define HOST_LEN 1024

FILE* file;

/**
* Initialization function that the user must call
* before running any other library function.
* The function may, for example, allocate memory or
* create/open files.
* Returns 0 uppon success and -1 on failure
*/
int osm_init()
{
    file = fopen("/tmp/liavst2.txt", "w");
    return (file ==  NULL) ? FAILURE: SUCCESS;
}


/**
* finalizer function that the user must call
* after running any other library function.
* The function may, for example, free memory or
* close/delete files.
* Returns 0 uppon success and -1 on failure
*/
int osm_finalizer()
{
    if (fclose(file) == EOF)
    {
        return FAILURE;
    }

    return remove("/tmp/liavst2.txt") ? FAILURE: SUCCESS;
}



/**
* some void function to calculate the function call time
*/
void someFunction()
{}


/**
* returns the elapsed time calculated by gettimeofday
```

```
60  * @param timeBegin - the beginning time
61  * @param timeEnd - the final time
62  */
63  double elapsedTime(timeval* timeBegin, timeval* timeEnd)
64  {
65      return (timeEnd->tv_sec - timeBegin->tv_sec) * NANO_ADJUST_BIL +
66             (timeEnd->tv_usec - timeBegin->tv_usec) * NANO_ADJUST_TH;
67  }
68
69
70  /**
71  * returns the average time it takes to perform a single operation
72  * @param iterations - the number of times to compute the time
73  */
74  double osm_operation_time(unsigned int iterations)
75  {
76      iterations = (!iterations) ? DEFAULT: iterations;
77
78      timeval timeBegin, timeEnd;
79      int loopJumps = std::floor(iterations/INSTRUCTION_LOOP_GROUP);
80
81      if (gettimeofday(&timeBegin, NULL) == FAILURE)
82      {
83          return FAILURE;
84      }
85
86      for (int i = 0; i < loopJumps; i++) // with loop unrolling
87      {
88          1 + 1;
89          1 + 1;
90          1 + 1;
91          1 + 1;
92          1 + 1;
93          1 + 1;
94      }
95
96      return (gettimeofday(&timeEnd, NULL) == FAILURE) ?
97             FAILURE: (double)(elapsedTime(&timeBegin, &timeEnd) / iterations);
98  }
99
100
101 /**
102 * returns the average time it takes to perform a function call operation
103 * @param iterations - the number of times to compute the time
104 */
105 double osm_function_time(unsigned int iterations)
106 {
107     iterations = (!iterations) ? DEFAULT: iterations;
108
109     timeval timeBegin, timeEnd;
110     int loopJumps = std::floor(iterations/INSTRUCTION_LOOP_GROUP);
111
112     if (gettimeofday(&timeBegin, NULL) == FAILURE)
113     {
114         return FAILURE;
115     }
116
117     for (int i = 0; i < loopJumps; i++) // with loop unrolling
118     {
119         someFunction();
120         someFunction();
121         someFunction();
122         someFunction();
123         someFunction();
124         someFunction();
125     }
126
127     return (gettimeofday(&timeEnd, NULL) == FAILURE) ?
```

```
128             FAILURE: (double)(elapsedTime(&timeBegin, &timeEnd) / iterations);
129 }
130
131 /**
132 * returns the average time it takes to perform a null system call operation
133 * @param iterations - the number of times to compute the time
134 */
135 double osm_syscall_time(unsigned int iterations)
136 {
137     iterations = (!iterations) ? DEFAULT: iterations;
138
139     timeval timeBegin, timeEnd;
140     int loopJumps = std::floor(iterations/INSTRUCTION_LOOP_GROUP);
141
142     if (gettimeofday(&timeBegin, NULL) == FAILURE)
143     {
144         return FAILURE;
145     }
146
147     for (int i = 0; i < loopJumps; i++) // with loop unrolling
148     {
149         OSM_NULLSYSCALL;
150         OSM_NULLSYSCALL;
151         OSM_NULLSYSCALL;
152         OSM_NULLSYSCALL;
153         OSM_NULLSYSCALL;
154         OSM_NULLSYSCALL;
155     }
156
157     return (gettimeofday(&timeEnd, NULL) == FAILURE) ?
158             FAILURE: (double)(elapsedTime(&timeBegin, &timeEnd) / iterations);
159 }
160
161
162 /**
163 * returns the average time it takes to perform a disk access operation
164 * @param iterations - the number of times to compute the time
165 */
166 double osm_disk_time(unsigned int iterations)
167 {
168     iterations = (!iterations) ? DEFAULT: iterations;
169
170     timeval timeBegin, timeEnd;
171     int loopJumps = std::floor(iterations/INSTRUCTION_LOOP_GROUP);
172
173     if (gettimeofday(&timeBegin, NULL) == FAILURE)
174     {
175         return FAILURE;
176     }
177
178     for (int i = 0; i < loopJumps; i++) // with loop unrolling
179     {
180         if (fclose(file) == EOF) return FAILURE;
181         if ((file = fopen("/tmp/liavst2.txt", "w")) == NULL) return FAILURE;
182         if (fclose(file) == EOF) return FAILURE;
183         if ((file = fopen("/tmp/liavst2.txt", "w")) == NULL) return FAILURE;
184         if (fclose(file) == EOF) return FAILURE;
185         if ((file = fopen("/tmp/liavst2.txt", "w")) == NULL) return FAILURE;
186     }
187
188     return (gettimeofday(&timeEnd, NULL) == FAILURE) ?
189             FAILURE: (double)(elapsedTime(&timeBegin, &timeEnd) / iterations);
190 }
191
192
193 /**
194 * returns a struct containing all the information about the
195 * different timing
```

```
196  * @param operation_iterations - for a simple operation timing
197  * @param function_iterations - for function call timing
198  * @param syscall_iterations - for system call timing
199  * @param disk_iterations - for disk access timing
200  */
201 timeMeasurmentStructure measureTimes (unsigned int operation_iterations,
202                                       unsigned int function_iterations,
203                                       unsigned int syscall_iterations,
204                                       unsigned int disk_iterations)
205 {
206     timeMeasurmentStructure times;
207     times.machineName = new char[HOST_LEN];
208     if (gethostname(times.machineName, HOST_LEN) == FAILURE)
209     {
210         times.machineName = NULL;
211     }
212
213     times.instructionTimeNanoSecond = osm_operation_time(operation_iterations);
214     times.functionTimeNanoSecond = osm_function_time(function_iterations);
215     times.trapTimeNanoSecond = osm_syscall_time(syscall_iterations);
216     times.diskTimeNanoSecond = osm_disk_time(disk_iterations);
217
218     // checking for errors
219     if (times.instructionTimeNanoSecond &&
220         times.instructionTimeNanoSecond != FAILURE)
221     {
222         if (times.diskTimeNanoSecond != FAILURE)
223         {
224             times.diskInstructionRatio = (double)(times.diskTimeNanoSecond /
225                                                   times.instructionTimeNanoSecond);
226         } else {
227             times.diskInstructionRatio = FAILURE;
228         }
229
230         if (times.functionTimeNanoSecond != FAILURE)
231         {
232             times.functionInstructionRatio = (double)(times.functionTimeNanoSecond /
233                                                 times.instructionTimeNanoSecond);
234         } else {
235             times.functionInstructionRatio = FAILURE;
236         }
237
238         if (times.trapTimeNanoSecond != FAILURE)
239         {
240             times.trapInstructionRatio = (double)(times.trapTimeNanoSecond /
241                                                   times.instructionTimeNanoSecond);
242         } else {
243             times.trapInstructionRatio = FAILURE;
244         }
245
246     } else {// there was a problem in the instruction time
247         times.diskInstructionRatio = FAILURE;
248         times.functionInstructionRatio = FAILURE;
249         times.trapInstructionRatio = FAILURE;
250     }
251
252     return times;
253 }
```