Contents

1 README

```
ransha
Ran Shaham (203781000)
EX: 1

FILES:
README  -- This file
osm.cpp -- a file with some code
Makefile -- a makefile that creates the library (make with no arguments, or
            'all')

NOTES -
There are A LOT of magic numbers in my code, I'm hoping that you'll believe me
when I say this is for the sake of accuracy in time measurements and not
because of laziness or bad coding practice.

TASK 1:
The program takes an argument. It then creates a directory called 'Welcome',
inside it it creates another directory called 'To', and inside it creates a
file name OS with write attributes (open O_CREAT| O_WRONLY).
It then writes some text to this file
(prompts us to read the course guidelines).
I also noticed that the number of bytes written to the file is dependant by
the length of the arguments I supplied.
Namely, with 1 letter argument the number of bytes written is 178 and each
added letter adds another byte. Then, the program closes and deletes the file
and remove both directories.

I learned it by 'man'ing the sys calls:
mkdir - creates a directory, returns 0 upon success
open - opens a file with the attributes set by the flags give
file descriptor
write - writes to a file given by its file descriptor from a given pointer a
given number of
bytes.
close - closes an open file
unlink - deletes a file (deletes a name from the filesystem?) given by its
descriptor.
rmdir - removes an empty directory

TASK 2:
osm_operation_time - in this function I used loop unrolling to perform a lot
of basic operations on random numbers (they were randomized before the
measurement start).
I then divided by the number of total operations performed, hoping that the
loop operations weren't significant compared to the operations.
Note that I didn't assign the result of the operations to a variable so I
won't count the assignment time, I'm aware this could be problematic - if the
compiler decides that an unassigned rhs expression should be ignored, couldn't
think of a better way.
The loop unrolling, as suggested in the ex description, is to get average time
bigger than 0 because basic operations are performed quickly.

osm_function_time - I created two empty functions in my cpp file and called
them one by one a lot of times in each iteration (~500 iterations in each
main for iteration, in each one of those I made 12 function calls).
Again, this is to avoid manipulating very small numbers, so i'd get more
accurate results.

osm_syscall_time - in each iteration I made 5 empty system calls. This is
```

-3/-15 Wrong anwser(s) in the README. (code='README_general_err') Partial explanation of the system calls including parameters and return value

```
much fewer than the ones above because these functions take much longer time
so the measured numbers will be reasonable to handle.

osm_disk_time - Most of my time was spent HERE. I tried to write and read from
files that I opened with O_SYNC and O_DIRECT flags to avoid caching and
buffering. It was very hard since the writing and reading failed - after
reading a lot about it I figured it's because of alignment issues, couldn't
understand it, not for a bit.
So I removed the flags and measured the times without them. I'm aware this
measurement isn't the best, but it got reasonable results (~900 ns which is
much higher than all the others) so I left it at that.
I used two files hoping that accessing them alternatively will prevent the
disk from optimizing the access to their locations.

Note that the files I handled were created and deleted by the osm_init,
osm_finalizer functions.
```

2 Makefile

You may use built-in variable CXX which defaults to g++.

```
1  CC=g++
2  RANLIB=ranlib
3
4  LIBSRC=osm.cpp
5  LIBOBJ=$(LIBSRC:.cpp=.o)
6
7  INCS=-I.
8  CFLAGS = -Wall -g $(INCS)
9  LOADLIBES = -L./
10
11 OSMLIB = libosm.a
12 TARGETS = $(OSMLIB)
13
14 TAR=tar
15 TARFLAGS=-cvf
16 TARNAME=ex1.tar
17 TARSRCS=$(LIBSRC) Makefile README
18
19 all: $(TARGETS)
20
21
22 $(TARGETS): $(LIBOBJ)
23     $(AR) $(ARFLAGS) $@ $^
24     $(RANLIB) $@
25
26 clean:
27     $(RM) $(TARGETS) $(OSMLIB) $(OBJ) $(LIBOBJ) *~ *core
28
29 depend:
30     makedepend -- $(CFLAGS) -- $(SRC) $(LIBSRC)
31
32 tar:
33     $(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
```

3 osm.cpp

```cpp
#include "osm.h"
#include <vector>
#include <sys/time.h>
#include <cstdlib>
#include <unistd.h>
#include <fcntl.h>

#define FINISH_SUCCESS 0
#define FINISH_ERROR -1
#define INVALID_ITER 0
#define DEFAULT_ITER 1000
#define MICRO_TO_NANO(x) ((x) * 1000)
#define SEC_TO_MICRO(x) ((x) * 1000000)
#define FILE_FLAGS O_CREAT | O_RDWR
#define FILE_NAME_1 "/tmp/.ransha"
#define FILE_NAME_2 "/tmp/.ransha1"
#define MAX_NAME_LEN 64

// g for global
static int gfd1 = 0, gfd2 = 0; // fd for file-descriptor
static char *gmachineName;

/* 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()
{
    // Attempt to open/create two files, if one fails the other isn't opened
    // (logical or)
    if (((gfd1 = open(FILE_NAME_1, FILE_FLAGS)) == FINISH_ERROR) ||
        ((gfd2 = open(FILE_NAME_2, FILE_FLAGS)) == FINISH_ERROR))
    {
        return FINISH_ERROR;
    }
    // Allocate memory for hostname
    gmachineName = new char[MAX_NAME_LEN];
    return FINISH_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()
{
    // Free allocated memory
    delete gmachineName;

    // Bitwise or to ensure both files are at least attempted to be closed
    if ((close(gfd1) == FINISH_ERROR) | (close(gfd2)) == FINISH_ERROR)
    {
        return FINISH_ERROR;
    }
    // Remove files.
```

```
60      if ((unlink(FILE_NAME_1) == FINISH_ERROR) |
61          (unlink(FILE_NAME_2) == FINISH_ERROR))
62      {
63          return FINISH_ERROR;
64      }
65      return FINISH_SUCCESS;
66  }
67
68  /*
69   * Empty functions for the function-call time measurements.
70   */
71  void emptyFunc() {}
72  void emptyFunc2() {}
73
74  /*
75   * Subtracts the second timeval struct from the first to get the difference
76   * between them in microseconds, then converts to nano-seconds.
77   */
78  double timeDiffInNano(timeval a, timeval b)
79  {
80      return MICRO_TO_NANO(SEC_TO_MICRO(a.tv_sec - b.tv_sec) \
81                                      + a.tv_usec - b.tv_usec);
82  }
83
84  /* Time measurement function for a simple arithmetic operation.
85   * returns time in nano-seconds upon success,
86   * and -1 upon failure.
87   */
88  double osm_operation_time(unsigned int iterations)
89  {
90      // Ensure non-zero iterations
91      if (iterations == INVALID_ITER)
92      {
93          iterations = DEFAULT_ITER;
94      }
95
96      std::srand(0);
97      int a, b, c, d, j;
98      struct timeval st, et;
99      double tv, total = 0;
100
101     for (int i = 0; i < iterations; ++i)
102     {
103         // Get random numbers
104         a = std::rand();
105         b = std::rand();
106         c = std::rand();
107         d = std::rand();
108         if (gettimeofday(&st, NULL) != 0) // Start measure
109             return FINISH_ERROR;
110         /* Do a lot of basic operations on random numbers */
111         for (j = 0; j < DEFAULT_ITER; ++j) {
112             //
113             a + b;
114             a & b;
115             a | b;
116             b + d;
117             b & d;
118             b | d;
119             a + c;
120             a & c;
121             a | c;
122             b + c;
123             b & c;
124             b | c;
125             a + d;
126             a & d;
127             a | d;
```

```
128         }
129         if (gettimeofday(&et, NULL) != 0) // End measure
130             return FINISH_ERROR;
131         // Calculate the time took (and convert to double)
132         tv = timeDiffInNano(et, st);
133         tv /= DEFAULT_ITER * 15; // 15 is the number of operations in
134                                  // each inner loop iteration
135         total += tv; // accumulative measured time
136     }
137     return total / iterations; // Average time
138 }
139 
140 /* Time measurement function for an empty function call.
141  * returns time in nano-seconds upon success,
142  *  and -1 upon failure.
143  */
144 double osm_function_time(unsigned int iterations)
145 {
146     // Ensure non-zero iterations
147     if (iterations == INVALID_ITER)
148     {
149         iterations = DEFAULT_ITER;
150     }
151 
152     int j;
153     struct timeval st, et;
154     double tv, total = 0;
155 
156     for (int i = 0; i < iterations; ++i)
157     {
158         if (gettimeofday(&st, NULL) != 0)
159             return FINISH_ERROR;
160         for (j = 0; j < DEFAULT_ITER / 2; ++j)
161         {
162             emptyFunc();
163             emptyFunc2();
164             emptyFunc();
165             emptyFunc2();
166             emptyFunc();
167             emptyFunc2();
168             emptyFunc();
169             emptyFunc2();
170             emptyFunc();
171             emptyFunc2();
172             emptyFunc();
173             emptyFunc2();
174         }
175         if (gettimeofday(&et, NULL) != 0)
176             return FINISH_ERROR;
177         tv = timeDiffInNano(et, st);
178         tv /= DEFAULT_ITER / 2 * 12; // 12 for number of function calls in each
179                                      // iteration, DEFAULT_ITER / 2 iterations
180         total += tv;
181     }
182     return total / iterations;
183 }
184 
185 /* Time measurement function for an empty trap into the operating system.
186  * returns time in nano-seconds upon success,
187  * and -1 upon failure.
188  */
189 double osm_syscall_time(unsigned int iterations)
190 {
191     // Ensure non-zero iterations
192     if (iterations == INVALID_ITER)
193     {
194         iterations = DEFAULT_ITER;
195     }
```

```
196
197     struct timeval st, et;
198     double tv, total = 0;
199
200     for (int i = 0; i < iterations; ++i)
201     {
202         if (gettimeofday(&st, NULL) != 0)
203             return FINISH_ERROR;
204         OSM_NULLSYSCALL;
205         OSM_NULLSYSCALL;
206         OSM_NULLSYSCALL;
207         OSM_NULLSYSCALL;
208         OSM_NULLSYSCALL;
209         if (gettimeofday(&et, NULL) != 0)
210             return FINISH_ERROR;
211         tv = timeDiffInNano(et, st);
212         tv /= 5; // 5 empty traps were made
213         total += tv;
214     }
215     return total / iterations;
216 }
217
218 /* Time measurement function for accessing the disk.
219  * returns time in nano-seconds upon success,
220  * and -1 upon failure.
221  */
222 double osm_disk_time(unsigned int iterations)
223 {
224     // Ensure non-zero iterations
225     if (iterations == INVALID_ITER)
226     {
227         iterations = DEFAULT_ITER;
228     }
229     int buffSize = 64;
230     struct timeval st, et;
231     double tv, total = 0;
232     char oneByte[buffSize];
233
234     for (int i = 0; i < iterations; ++i)
235     {
236         if (gettimeofday(&st, NULL) != 0)
237             return FINISH_ERROR;
238         // Try to write to both files
239         if ((write(gfd1, oneByte, 1) == FINISH_ERROR) ||
240                 (write(gfd2, oneByte, 1) == FINISH_ERROR) ||
241                 (read(gfd1, oneByte, 1) == FINISH_ERROR) ||
242                 (read(gfd2, oneByte, 1) == FINISH_ERROR))
243         {
244             return FINISH_ERROR;
245         }
246         if (gettimeofday(&et, NULL) != 0)
247             return FINISH_ERROR;
248         tv = timeDiffInNano(et, st);
249         tv /= 4; // 4 is the number of "disk access"es performed
250         total += tv;
251     }
252     return total / iterations;
253 }
254
255 /*
256  * Measure all times and return a struct with the relevant data.
257  * Eeach field is set to -1 upon error or the correct value otherwise,
258  * except machineName which is set to the null-char '\0' upon failure.
259  */
260 timeMeasurmentStructure measureTimes(unsigned int operation_iterations,
261                                      unsigned int function_iterations,
262                                      unsigned int syscall_iterations,
263                                      unsigned int disk_iterations)
```

```
{
    timeMeasurmentStructure retVal;
    // Get host name
    if (gethostname(gmachineName, MAX_NAME_LEN) != 0)
    {
        gmachineName = '\0';
    }
    retVal.machineName = gmachineName;

    // Measure times
    retVal.instructionTimeNanoSecond = \
    osm_operation_time(operation_iterations);
    retVal.functionTimeNanoSecond = osm_function_time(function_iterations);
    retVal.trapTimeNanoSecond = osm_syscall_time(syscall_iterations);
    retVal.diskTimeNanoSecond = osm_disk_time(disk_iterations);

    // If basic op time is 0 for some reason, all ratios are
    // invalid (divided by 0), therefore set to -1
    if (retVal.instructionTimeNanoSecond == 0 ||
        retVal.instructionTimeNanoSecond == FINISH_ERROR)
    {
        retVal.functionInstructionRatio = retVal.trapInstructionRatio = \
                                          retVal.diskInstructionRatio = -1;
    }
    else
    {
        retVal.functionInstructionRatio = retVal.functionTimeNanoSecond / \
        retVal.instructionTimeNanoSecond;
        retVal.trapInstructionRatio =  retVal.trapTimeNanoSecond / \
        retVal.instructionTimeNanoSecond;
        retVal.diskInstructionRatio = retVal.diskTimeNanoSecond / \
        retVal.instructionTimeNanoSecond;
    }
    return retVal;
}
```