Contents

1 README

```
akiva_s, roigreenberg
Akiva Sygal (305277220), Roi Greenberg (305571234)
EX: 1

FILES:
osm.c -- a file that implements the given osm.h
makefile

REMARKS:
no

ANSWERS:

Task 1:
    the program create a file inside folders' write the username(after
    milion spaces) then the argument at the next line. then close and
    delete everything.

        the system calls:


        mkdir - create "os" folder (with define permossions)
        mkdir - create "2015" folder inside "os" (with define permossions)

        open - (create and) open "info" file from "os/2015/" and the FD is 3

        write - write the username(after    milion spaces) then the argument at the
                next line

        close - closes the FD in 3(the given parameter) so it's not refer to any
                file and can be reuse

        unlink - delete 'info' file-name from the filesystem. the file will be deleted
                if nothing use it.

        rmdir - deletes the folder(if they are empty)

        exit_group - exit all of the threads in the prosses

Task 2:
 1. a.    if 2 programs write to the same file and one is saving the current
    state while the other continue to write, when the first program finish
    with the interrupt, it could start write from the saved state and
    override what the second program does.

    b.    if the program is time-based, while stopping for an interrupt it's
    may save (among other things) the current time and then restore it
    even-though it the wrong time since time comtinue to move while the
    interrupt was in action.

 2.      that a very bad solution. It can paralized the computer. while 1
    program is running, no one can access the hard drive, the I/O devices
    and etc.

Task 3:
    we implemented the iterarions in each function with loop unrolling of 10s. each time we've measured the
    just before and after the loop (using 2 calls to "gettimeofday"), and divided the difference by the number of
    iterations. for that we've used he function "time_difference_msec" that calculates the differences in nanoseconds
    between 2 imstances of the struct "timeval". also, we've added the function "fixIterations" to deal with invalid
```

input, including rounding it up to a multiplication of 10 and handling 0 input. also we've defined the variables in the functions as volatile so they could not be optimized out, and recommended the compiler to store them on registers to save time.

2 Makefile

```
CC=gcc
RANLIB=ranlib

LIBSRC=osm.c
LIBOBJ=$(LIBSRC:.c=.o)

INCS=-I.
CFLAGS = -Wall -g $(INCS)
LOADLIBES = -L./

OSMLIB = libosm.a
TARGETS = $(OSMLIB)

TAR=tar
TARFLAGS=-cvf
TARNAME=ex1.tar
TARSRCS=$(LIBSRC) Makefile README

all: $(TARGETS)

$(TARGETS): $(LIBOBJ)
    $(AR) $(ARFLAGS) $@ $^
    $(RANLIB) $@

clean:
    $(RM) $(TARGETS) $(OSMLIB) $(OBJ) $(LIBOBJ) *~ *core

depend:
    makedepend -- $(CFLAGS) -- $(SRC) $(LIBSRC)

tar:
    $(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
```

3 osm.c

```c
#include "osm.h"
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

#define SUCCESS 0
#define FAIL -1
#define ITERATIONS 50000
#define REPEAT 10

/* Initialization function that the user must call
 * before running any other library function.
 * Returns 0 uppon success and -1 on failure.
 */
int osm_init()
{
    return SUCCESS;
}


/*
 * If number of iterarions is not valid, this will fix it.
 */
unsigned int fixIterations (unsigned int osm_iterations)
{

    if (osm_iterations == 0)
        {
        return (ITERATIONS);
        }
    if (osm_iterations % REPEAT != 0)
        {
            return (((osm_iterations / REPEAT) + 1) * REPEAT);
    }
    else
    {
            return (osm_iterations);
    }
}

/*
 * Gets the difference in Nano-seconds between the timevals t1 and t0.
 */
unsigned long long time_difference_msec(struct timeval t0, struct timeval t1)
{
    return (t1.tv_sec - t0.tv_sec) * 1000000000 + (t1.tv_usec - t0.tv_usec) * 1000;
}

void func(){}

/* Time measurement function for an empty function call.
   returns time in nano-seconds upon success,
   and -1 upon failure.
   Zero iterations number is invalid.
   */
double osm_function_time(unsigned int osm_iterations)
{
    struct timeval tv1, tv2;
    volatile register int i;
```

```
60      if (gettimeofday(&tv1, NULL) != SUCCESS )
61          return FAIL;
62      for (i = 0; i < osm_iterations; i = i + REPEAT)
63      {
64          func();
65      func();
66          func();
67          func();
68          func();
69          func();
70          func();
71          func();
72          func();
73          func();
74      }
75      if (gettimeofday(&tv2, NULL) != SUCCESS )
76          return FAIL;
77      return ((double)time_difference_msec(tv1, tv2) / osm_iterations);
78  }
79
80
81  /* Time measurement function for an empty trap into the operating system.
82     returns time in nano-seconds upon success,
83     and -1 upon failure.
84     Zero iterations number is invalid.
85     */
86  double osm_syscall_time(unsigned int osm_iterations)
87  {
88      struct timeval tv1, tv2;
89      volatile register int i;
90      if (gettimeofday(&tv1, NULL) != SUCCESS )
91          return FAIL;
92      for (i = 0; i < osm_iterations; i = i + REPEAT)
93      {
94          OSM_NULLSYSCALL;
95          OSM_NULLSYSCALL;
96          OSM_NULLSYSCALL;
97          OSM_NULLSYSCALL;
98          OSM_NULLSYSCALL;
99          OSM_NULLSYSCALL;
100         OSM_NULLSYSCALL;
101         OSM_NULLSYSCALL;
102         OSM_NULLSYSCALL;
103         OSM_NULLSYSCALL;
104     }
105     if (gettimeofday(&tv2, NULL) != SUCCESS )
106         return FAIL;
107     return ((double)time_difference_msec(tv1, tv2) / osm_iterations);
108 }
109
110 /* Time measurement function for a simple arithmetic operation.
111    returns time in nano-seconds upon success,
112    and -1 upon failure.
113    Zero iterations number is invalid.
114    */
115 double osm_operation_time(unsigned int osm_iterations)
116 {
117     struct timeval tv1, tv2;
118     volatile register int i, a;
119     a = 0;
120     if (gettimeofday(&tv1, NULL) != SUCCESS )
121         return FAIL;
122     for (i = a; i < osm_iterations; i = i + REPEAT)
123     {
124         a = 0;
125         a = 0;
126         a = 0;
127         a = 0;
```

```
128             a = 0;
129             a = 0;
130             a = 0;
131             a = 0;
132             a = 0;
133             a = 0;
134         }
135         if (gettimeofday(&tv2, NULL) != SUCCESS )
136             return FAIL;
137         return ((double)time_difference_msec(tv1, tv2) / osm_iterations);
138     }
139
140
141     /*
142      * the function that meseaurs all the needed times.
143      */
144     timeMeasurmentStructure measureTimes (unsigned int osm_iterations)
145     {
146         timeMeasurmentStructure result;
147         if (gethostname(result.machineName, HOST_NAME_MAX) != SUCCESS)
148             result.machineName[0] = '\0';
149             osm_iterations = fixIterations(osm_iterations);
150         result.numberOfIterations = osm_iterations;
151         result.instructionTimeNanoSecond = osm_operation_time(osm_iterations);
152         result.functionTimeNanoSecond = osm_function_time(osm_iterations);
153         result.trapTimeNanoSecond = osm_syscall_time(osm_iterations);
154         result.functionInstructionRatio = result.functionTimeNanoSecond /
155                 result.instructionTimeNanoSecond;
156         result.trapInstructionRatio = result.trapTimeNanoSecond /
157                     result.instructionTimeNanoSecond;
158
159         return result;
160     }
```