Contents

1 Basic Test Results

```
['brahan', 'danielabayev']
make test
passed basic make test
compile test
passed basic compile test
```

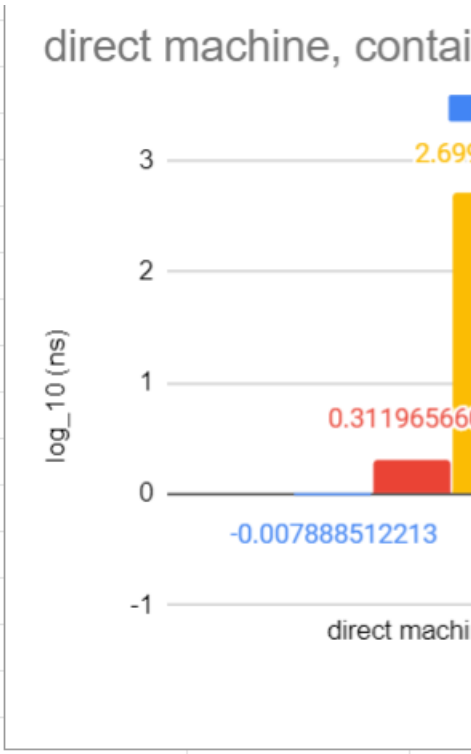
2 README

```
brahan, danielabayev
Brahan Wassan(320455116), Daniel Abayev (206224396)
EX: 1

FILES:
osm.c -- a static library with the programming part of the ex
makefile -- a makefile for the program
os_ex1_part_b.png

REMARKS:
No remarks

ANSWERS:

Assignment 1 - Using strace to understand what a progam is doing:

If you run the command without execute permission you get :
    strace: exec: Permission denied\n", 32strace: exec: Permission denied
And the program exits
After granting the program execute permission you get:

If you running the program any number of arguments that different from one you get:
    "Error. The program should receive a single argument. Exiting.\n: Success\n"
Meaning the program writing an error message and then exits

If you run the program with exec one argument you get:
    mkdir("Welcome", 0775)                  = 0
    mkdir("Welcome/To", 0775)               = 0
    openat(AT_FDCWD, "Welcome/To/OS2020", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
    fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
    write(3, "brahan\nIf you haven't read the course guidelines yet --- do it right now!\nempty", 79) = 79
    close(3)                                = 0
    unlink("Welcome/To/OS2020")             = 0
    rmdir("Welcome/To/")                    = 0
    rmdir("Welcome/")                       = 0
    exit_group(0)

Meaning the program creates a directory named "Welcome" with mkdir
Inside of it it creates another directory named To
Then its create a file called OS2020 (if its already created then it's just open it)
Inside the new file it writes:
 "USER_NAME\nIf you haven't read the course guidelines yet --- do it right now!\nGIVEN_INPUTFILE"
Then we delete the OS2020 file with unlink and the directory with rmdir and exit the program.


```

3 Graph.png

	log scale		
	direct machine	container	vm
op	-0.007888512213	0.00346053211	0.7891574919
empty_function	0.3119656604	0.3121773564	1.444138518
sys_call	2.699594071	2.706518934	2.918626296

	direct machine	container	vm
op	0.982	1.008	6.154
empty_function	2.051	2.052	27.806
sys_call	500.719	508.767	829.137



4 Makefile

```
CC=g++
CXX=g++
RANLIB=ranlib

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

INCS=-I.
CFLAGS = -Wall -std=c++11 -g $(INCS)
CXXFLAGS = -Wall -std=c++11 -g $(INCS)

OSMLIB = libosm.a
TARGETS = $(OSMLIB)

TAR=tar
TARFLAGS=-cvf
TARNAME=os_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) Graph.png
```

5 osm.cpp

```cpp
#include <sys/time.h>
#include "osm.h"

#define ERR -1
#define NOT_VALID 0
#define UNROLL_FACTOR 5
struct timeval start, end;

unsigned int loop_unrolling_factor(const unsigned int &iterations) {
    unsigned int real_iterations = iterations;
    unsigned int reminder = iterations % UNROLL_FACTOR;
    if (reminder != 0) {
        real_iterations += reminder; // round factor
    }
    return real_iterations;
}

double calculate_time_taken() {
    double time_taken = (end.tv_sec - start.tv_sec) * 1e6;
    return (double) (time_taken + (end.tv_usec - start.tv_usec)) * 1e3;
}

double osm_operation_time(unsigned int iterations) {
    if (iterations == NOT_VALID) {
        return ERR;
    } else {
        int a = 0, b = 0, c = 0;
        unsigned int real_iterations = loop_unrolling_factor(iterations);
        int is_valid = gettimeofday(&start, nullptr);
        if (is_valid != ERR) {
            for (unsigned int i = 0; i < real_iterations; i = i + UNROLL_FACTOR) {
                a = a + 1; //loop unrolling , factor 5
                b = b + 1;
                c = c + 1;
                a = a + 1;
                b = b + 1;
            }
            is_valid = gettimeofday(&end, nullptr);
            if (is_valid != ERR) {
                return calculate_time_taken() / (double) (real_iterations);
            }
        }

        return ERR;
    }
}

void empty_function() {}

double osm_function_time(unsigned int iterations) {
    if (iterations == NOT_VALID) {
        return ERR;
    } else {
        unsigned int real_iterations = loop_unrolling_factor(iterations);
        int is_valid = gettimeofday(&start, nullptr);
        if (is_valid != ERR) {
            for (unsigned int i = 0; i < real_iterations; i = i + UNROLL_FACTOR) {
                empty_function();
                empty_function();
```

```
                empty_function();
                empty_function();
                empty_function();
            }
            is_valid = gettimeofday(&end, nullptr);
            if (is_valid != ERR) {
                return calculate_time_taken() / (double) (real_iterations);
            }
        }

        return ERR;
    }
}

double osm_syscall_time(unsigned int iterations) {
    if (iterations == NOT_VALID) {
        return ERR;
    } else {
        unsigned int real_iterations = loop_unrolling_factor(iterations);
        int is_valid = gettimeofday(&start, nullptr);
        if (is_valid != ERR) {
            for (unsigned int i = 0; i < real_iterations; i = i + UNROLL_FACTOR) {
                OSM_NULLSYSCALL;
                OSM_NULLSYSCALL;
                OSM_NULLSYSCALL;
                OSM_NULLSYSCALL;
                OSM_NULLSYSCALL;
            }
            is_valid = gettimeofday(&end, nullptr);
            if (is_valid != ERR) {
                return calculate_time_taken() / (double) (real_iterations);
            }
        }
        return ERR;
    }
}
```