

Gaulthier Gain & Jonathan Francart

INFO0940 OPERATING SYSTEMS

Project #1


Academic year 2020-2021



SOME WORDS ABOUT PROJECTS

In total, 2 projects and 2 labs (test on the reference VM):

	<u>Project1</u>	<u>Lab1</u>	<u>Project2</u>	<u>Lab2</u>
<i>Type</i>	user space	user space	kernel space[1]	user or kernel space
<i>Release Month</i>	Feb.	March	March/April	April/May
<i>Details</i>	a bit of research	via Discord (mandatory lab)	a bit more of research	via Discord (mandatory lab)

Difficulty 

- ❖ Projects will require that you perform some research from your side.
- ❖ Labs will allow you to put into practice the theoretical concepts.
- ❖ Online tutorials and practical sessions can significantly help you.

[1] for this part, you need kernel sources (see next sessions)

YOUR FIRST PROJECT

It is a (custom) debugger...

1. Also called “tracer”, this program will gather and display different information about a particular running process.
2. Not the same as previous years (so do not take old projects).
3. You will understand taxonomy of executable file and process creation (a bit useful for the oral exam).

TRACER VS TRACEE (1)

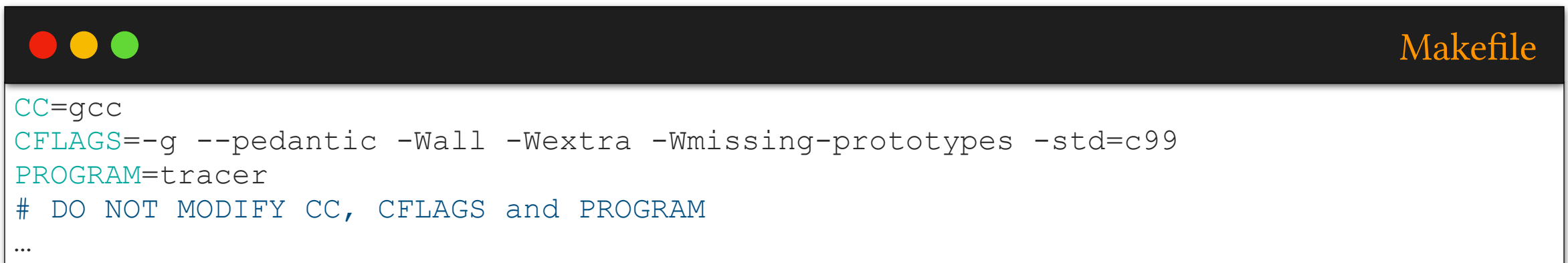
There are two different entities to consider:

1. The **tracer**: the program that you need to implement. This one must be compiled with the *make* command (you need to complete the given *Makefile*).
2. The **tracee**: it represents the process which will be traced by your tracer. You can write your own but this one must be **compiled statically**[1]. We will provide you an example file (called “*tracee.c*”).

[1] see practical session2

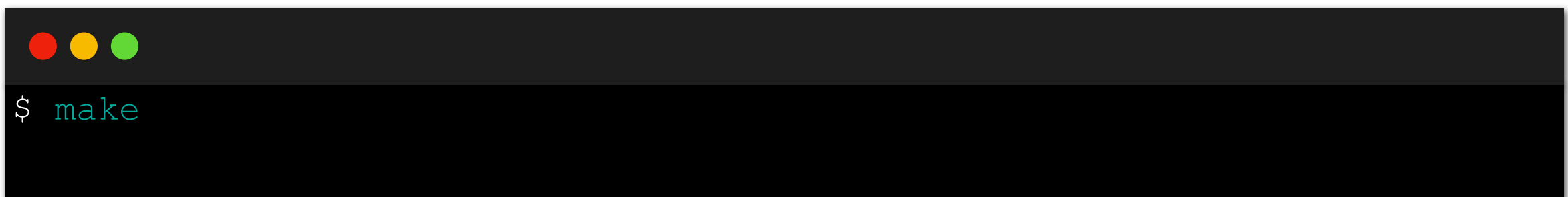
TRACER VS TRACEE (2)

1. The **tracer**: compile it with the *make* command:



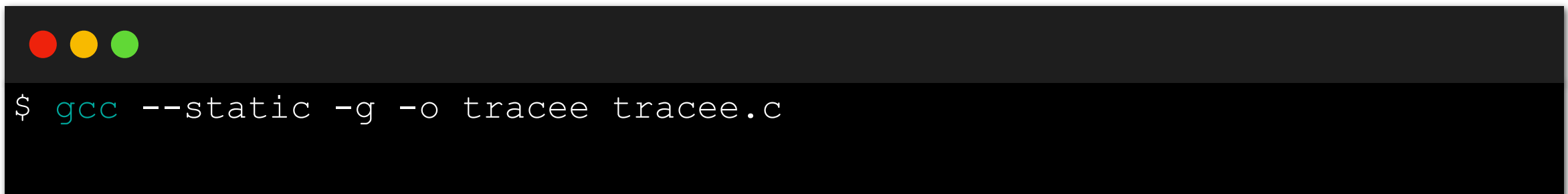
```
CC=gcc
CFLAGS=-g --pedantic -Wall -Wextra -Wmissing-prototypes -std=c99
PROGRAM=tracer
# DO NOT MODIFY CC, CFLAGS and PROGRAM
...
```

Makefile



```
$ make
```

2. The **tracee**: compile it **statically** with GCC:



```
$ gcc --static -g -o tracee tracee.c
```

THE TRACER:

MAIN FEATURES (1)

The **tracer** must rely on the *ptrace* API and takes two arguments:

1. The 1st argument is a string which represents the mode. There are two different modes:
 - A. Specifying `-p` to use the “*profiler*” mode;
 - B. Specifying `-s` to use the “*syscall*” mode.
2. The 2nd argument is also a string which represents the process which will be traced (the tracee). This string can either be a relative or an absolute path to the executable of the tracee.

THE TRACER:

MAIN FEATURES (2)

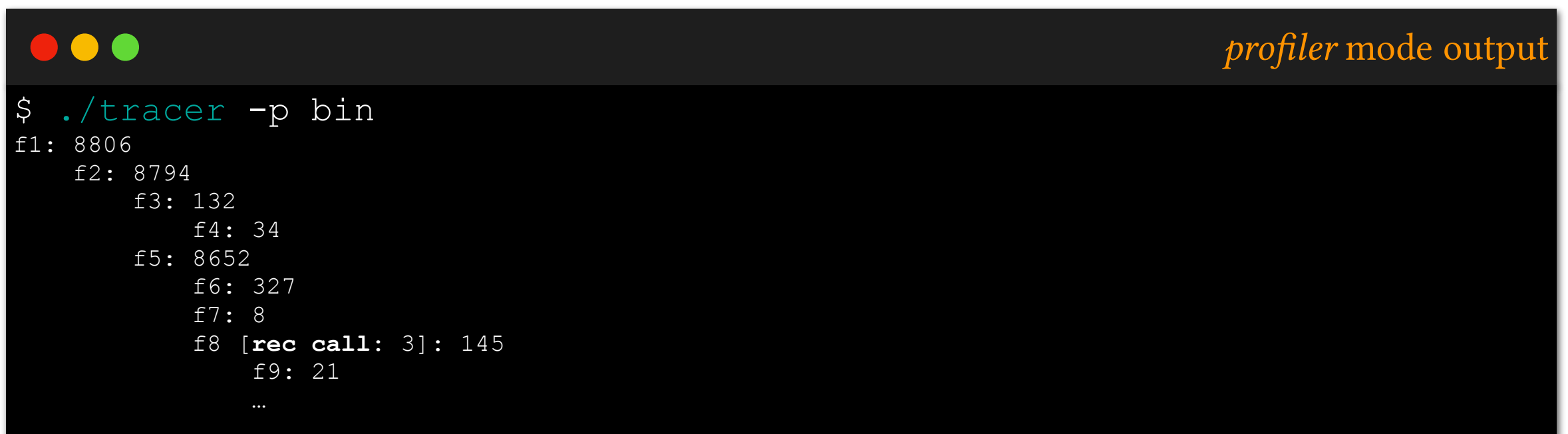
According to its first parameter, the tracer will have a different behaviour:

1. The “*profiler*” mode will display all functions called by the tracee as well as the number of instructions for each of them.
2. The second mode “*syscall*” will display all system calls performed by the tracee during its execution.

THE TRACER:

PROFILER MODE (1)

- ❖ When the tracer is called with the `-p` argument (*profiler* mode), it must detect and record all **relative** *calls* and *returns* (instructions) of the tracee.
- ❖ With this information, your program must construct a call tree/graph which contains the number of instructions executed by each function.
- ❖ *Hint*: Do not use any external libraries to find functions name. You can find them by thinking a bit about it[1].

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text "profiler mode output" is written in orange in the top-right corner. The terminal shows a command prompt "\$" followed by the command `./tracer -p bin`. Below the command, there is a list of function frames (f1 through f9) with their respective instruction counts. f1: 8806, f2: 8794, f3: 132, f4: 34, f5: 8652, f6: 327, f7: 8, f8: [rec call: 3]: 145, f9: 21. The list ends with an ellipsis "...".

```
$ ./tracer -p bin
f1: 8806
  f2: 8794
    f3: 132
      f4: 34
    f5: 8652
      f6: 327
      f7: 8
      f8 [rec call: 3]: 145
        f9: 21
      ...
```


THE TRACER:

PROFILER MODE (2)

- ❖ The number of instructions executed by a function is equal to the number of instructions executed by all its called functions + the number of instructions made by this function.
- ❖ The instruction *call* must not be counted as the first instruction of the called function. Nevertheless the *ret* instruction must be counted as the last instruction of the function.

C code

```
void print(){
    syscall(SYS_write,1,"Hello
world\n", 12);
    return;
}

int main(){
    print();
    exit(1);
}
```

Asm code

```
.global main
.text
main:
    call print
    mov $1,%eax
    mov $0,%ebx
    int $0x80
print:
    mov $4,%eax
    mov $1,%ebx
    mov $hello,%ecx
    mov $12,%edx
    int $0x80
    ret
.data
hello:
    .ascii "Hello wolrd\n"
```

```
$ ./tracer -p bin
main: 10
    print: 6
```

THE TRACER:

PROFILER MODE (3)

- ❖ If there is a recursive function, for each recursive calls: you must count the number of recursive calls without adding a new level in the call tree/graph.
- ❖ If a recursive function calls other functions during its execution, those functions must add a level in the call tree/graph.

C code

```
void print(){
    syscall(SYS_write,1,"Hello
world\n", 12);
    return;
}

void rec(int i){
    if(i != 0)
        rec(i - 1);
    print();
    return;
}

int main(){
    rec(4);
    exit(0);
}
```

Asm code

```
.global main
.text
main:
    mov $4,%eax
    push %eax
    call rec
    pop %eax
    mov $1,%eax
    mov $0,%ebx
    int $0x80
rec:
    cmp $0,%eax
    je .if_true
    push %eax
    dec %eax
    call rec
    pop %eax
.if_true:
    call print
    ret
print:
    mov $4,%eax
    mov $1,%ebx
    mov $hello,%ecx
    mov $12,%edx
    int $0x80
    ret
.data
hello: .ascii "Hello wolrd\n"
```

```
$ ./tracer -p bin
main: 73
    rec [rec call: 4]: 66
        print: 6
        print: 6
        print: 6
        print: 6
        print: 6
```

THE TRACER: SYSCALL MODE

- ❖ When the tracer is called with the `-s` argument (*syscall* mode), it will intercept and display all system calls which are called by the tracee process.
- ❖ We provide you a text file (called “*syscall.txt*”) which associates the system call number to its respective name.



```
$ ./tracer -s bin
syscall: uname
syscall: set_thread_area
syscall: readlink
syscall: brk
syscall: access
syscall: write
...
```

syscall mode output

THE TRACER: PTRACE

Your tracer must be based on the *ptrace* system call[1] in order to gather system calls, functions and generate the call tree/graph.

- ❖ For this part, we let you investigate and do some research on your side in order to meet the requirements of the project.
- ❖ The *ptrace* interface is quite easy to use and documentation can be found at <https://man7.org/linux/man-pages/man2/ptrace.2.html>

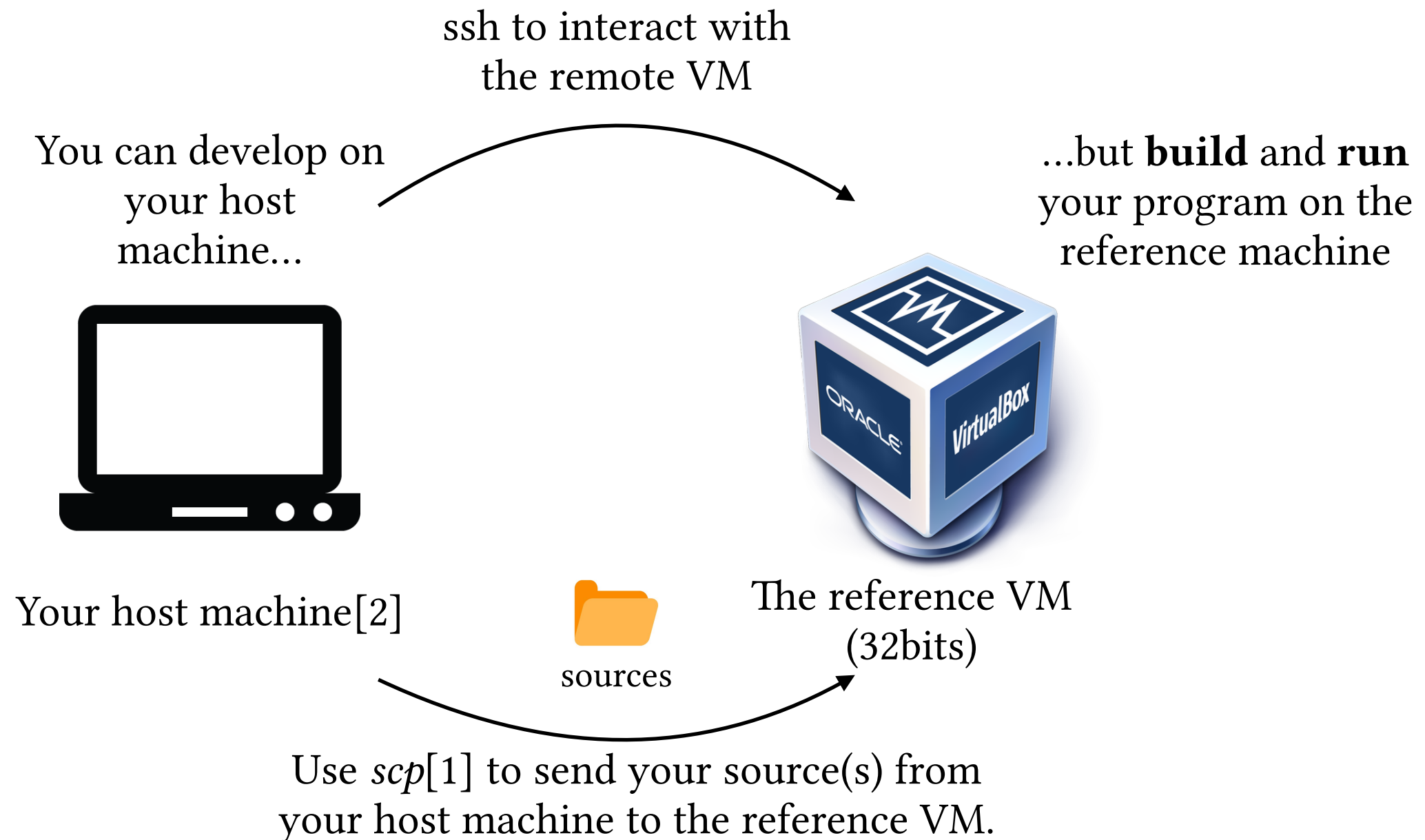
[1] see practical session2

THE 32BITS REFERENCE MACHINE

Use the provided reference machine (32bits) to test
your project.

- ❖ Testing your tracer on a 64bits machine will have a different behaviour than on the reference machine (different registers name, etc.).
- ❖ So your program may fail some tests on the submit platform...
- ❖ Please refer to the online [tutorial](#) to setup the required environment.

HOW TO TEST YOUR CODE ON THE REFERENCE MACHINE



[1] `scp <source_file> -P <vm_port> student@localhost:/home/student`

[2] Linux/Windows/Mac

THE TRACER: REQUIREMENTS

Additional Information:

- Group of **two** that you will **keep** the whole semester (see last two slides).
- Submit a *tar.gz* archive on the submission platform (code, *Makefile* & report).
- You are asked to write a very short report (max 2 pages) in which you briefly explain your implementation and answer to three specific questions.
- Further information in the statements available on eCampus.

Do not forget: We want clean code, without error, warning and memory leak (use `valgrind`[1]).

Do not forget too: We detect **plagiarism** so don't try...

Plagiarism = **0 for the course!**

[1] <https://www.valgrind.org>

Deadline: 31th March 2021

Happy Coding!

Teams	Member1	Member2
<i>Team01</i>	s173169 Cédric Meyers	s170859 Jean-Lorys Ossohou
<i>Team02</i>	s171384 Guillaume Lodrini	s171714 Thomas Demoulin
<i>Team03</i>	s180521 Maxime Goffart	s182113 Olivier Joris
<i>Team04</i>	s141543 Quentin Polet	s161917 Asad Bakija
<i>Team05</i>	s165516 Jan Held	s180444 Adil Ayman
<i>Team06</i>	s174809 Lorenzo Pagliarello	s172987 Davan Chiem Dao
<i>Team07</i>	s162756 Elias Piette	s165357 Clément Grodent
<i>Team08</i>	s152601 Guilian Deflandre	s174777 Loslever Terry
<i>Team09</i>	s172429 Wauthoz Julien	s162694 Peeters Thomas
<i>Team10</i>	s174817 Esteban Hernandez	s171282 Kamal El Feniche
<i>Team11</i>	s143467 Francesco Mirisola	s170962 Arnaud Crucifix
<i>Team12</i>	s151517 Maxence Calixte	s151777 Robin Perski
<i>Team13</i>	s181506 Ilias Sebati	s180821 Alex-Manuel Rosca
<i>Team14</i>	s180337 Julien Gustin	s181539 Houyon Joachim
<i>Team15</i>	s180290 Jakub Duchateau	s180383 Emilien Wansart
<i>Team16</i>	s170679 Jean-David Mukolonga	s157349 Christal Mangolopa
<i>Team17</i>	s202208 Alain Eros Prestige	s206176 Karim Khadro
<i>Team18</i>	s208007 Mohammad El Sakka	s208006 Clément Alexandre

Teams	Member1	Member2
<i>Team19</i>	s175603 Guilherme Madureira	s174231 Julien Carion
<i>Team20</i>	s140212 Florian Mataigne	s153143 Manon Latour
<i>Team21</i>	s172873 Meng Lin	s170818 François Cubelier
<i>Team22</i>	s161284 Lionel La Rocca	s197073 Maxime Binot
<i>Team23</i>	s171830 Maxime Amodei	s170940 Martin Peeters
<i>Team24</i>	s205441 Briec Saille	s208000 Zakat Regragui
<i>Team25</i>	s204708 Alexandre Debart	s131797 Adrien Dauvister
<i>Team26</i>	s197070 Ramiamanana	s197206 Pacome Désire Dandji
<i>Team27</i>	s180162 Lev Malcev	s182281 Cyril Lambert
<i>Team28</i>	s175615 Jordi Hoorelbeke	s175610 Marco Naa
<i>Team29</i>	s172244 Stephan Bulut	s175202 Cédric Siboyabasore
<i>Team30</i>	s165306 Lisa Bueres y	s175676 Ilias Koutroumpas
<i>Team31</i>	s154371 Corentin Van Putte	s165391 Gaël Di Raimo
<i>Team32</i>	s207273 Marc-evy Misseke	s150234 Antoine Dekyvere
<i>Team33</i>	s152481 Antoine Verdonck	s151229 William Belvaux
<i>Team34</i>	s171472 Adrien Guilliams	s161942 Laurent Habiyaremye
<i>Team35</i>	s160923 Robin Douhard	s160733 Thomas Lucas
<i>Team36</i>	s175068 Boris Courtoy	s175004 Loïc Meunier