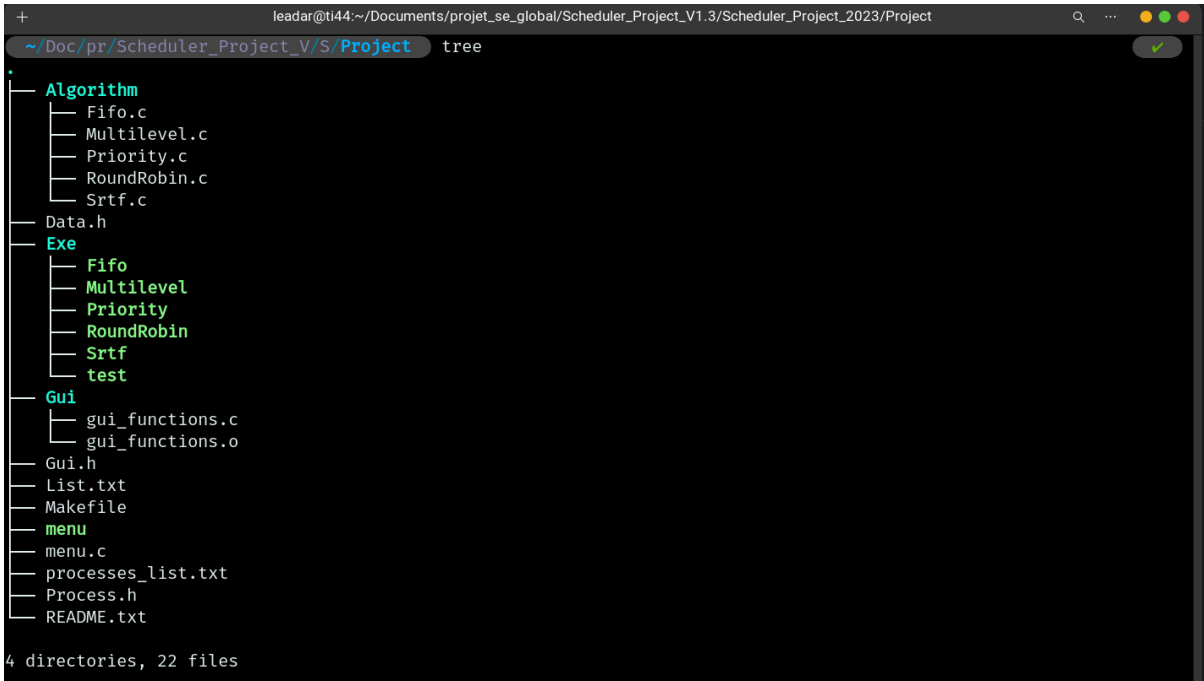


Explanation Document

1. Project plan

Our project architecture is presented in the following diagram:

A terminal window with a dark background showing the output of the 'tree' command. The path is ~/Doc/pr/Scheduler_Project_V/S/Project. The tree structure is as follows:

```
Algorithm
├── Fifo.c
├── Multilevel.c
├── Priority.c
├── RoundRobin.c
└── Srtf.c
Data.h
Exe
├── Fifo
├── Multilevel
├── Priority
├── RoundRobin
├── Srtf
└── test
Gui
├── gui_functions.c
└── gui_functions.o
Gui.h
List.txt
Makefile
menu
menu.c
processes_list.txt
Process.h
README.txt
4 directories, 22 files
```

Project: is the main folder where we organized the files and the rest of folders

Algorithm: this folder includes all the algorithm files such as Fifo, Srtf, Sjf, Round Robin and Priority.

- ✓ Fifo (First In First Out)
- ✓ Multilevel
- ✓ Priority (Ordering by high priority)
- ✓ RoundRobin (each process is assigned a fixed time slot in a cyclic way)
- ✓ Srtf (Shortest Remaining Time First)

Exe: this folder contains the executable program of all the algorithm.

Makefile: contains a list of rules. These rules tell the system what commands we want to be executed.

The code of the makefile was shown perfectly in the next capture:

```

1  #Variables Declaration
2  SRC_DIR := Algorithm
3  GUI_DIR := Gui
4  BUILD_DIR := Exe
5  #Get a list of all the C source files in a directory
6  SRC_FILES := $(wildcard $(SRC_DIR)/*.c)
7  SOURCE:= $(SRC_FILES:$(SRC_DIR)/%.c=$(BUILD_DIR)/%)
8  #Compilation Options
9  CFLAGS := -Wall -g -w
10 CGUIFLAGS := `pkg-config --cflags --libs gtk+-3.0`
11 all:gui $(SOURCE) menu
12     chmod 755 $(SOURCE) menu
13
14 #Generate the executable of menu
15 menu: menu.c
16     gcc $(CFLAGS) menu.c -o menu
17
18 gui: $(GUI_DIR)/gui_functions.c
19     gcc $(CGUIFLAGS) -c $(GUI_DIR)/gui_functions.c -o $(GUI_DIR)/gui_functions.o
20
21 #Generate the executable of algorithms
22 $(BUILD_DIR)/%: $(SRC_DIR)/%.c
23     gcc $(CGUIFLAGS) $(CFLAGS) $< $(GUI_DIR)/gui_functions.o -o $@
24
25 clean:
26     rm -f Exe/*      #Remove all executables files of Algorithms
27     rm -f menu       #Remove the executable file of menu
28     rm -f List.txt   #Remove the text file List
29     rm -f Gui/gui_functions.o

```

In the first part, we declare the necessary variables to use them later, and CFLAGS is the list of flags to pass to the compilation command.

Note: One use of the wildcard function is to get a list of all the C source files in a directory, like this: \$(wildcard *.c)

Options:

-g : adds debugging information to the executable file.

-Wall : turns on most, but not all, compiler warnings.

In the next part, we put four rules in the makefile:

The rules are in two parts:

The first line is called a **dependency line** and the subsequent line(s) are called **actions or commands**. The action line(s) must be indented with a tab.

RULE: DEPENDENCY LINE

[tab]**ACTION LINE(S)**

The dependency line is made of two parts. The first part (before the colon) are **target files** and the second part (after the colon) are called **source files**. It is called a dependency line because the first part depends on the second part. Multiple target files and source files must be separated by a space.

DEPENDENCY LINE: **TARGET FILES**: **SOURCE FILES**

Rule 1:

```
all:gui $(SOURCE) menu
      chmod 755 $(SOURCE) menu
```

Typing 'make' will invoke the first target entry in the file (in our case the **all** target entry). We can name this target entry anything, but "default" or "all" are the most commonly used names by convention. This uses Suffix Replacement within a macro: \$(SRC_FILES: \$(SRC_Dir)/%.c=\$(BUILD_DIR)/%) For each word in SRC_FILES replace the .c files with their executables.

Rule 2:

```
menu: menu.c
      gcc $(CFLAGS) menu.c -o menu
```

To create the executable file **menu** we need the source file menu.c .

Rule 3:

```
gui: $(GUI_DIR)/gui_functions.c
      gcc $(CGUIFLAGS) -c $(GUI_DIR)/gui_functions.c -o $(GUI_DIR)/gui_functions.o
```

this is a suffix replacement rule for building .exe's from .c's it uses automatic variables \$<: the name of the prerequisite of the rule(.c file) and \$@: the name of the target of the rule (.exe file)

Rule 4:

```
clean:
      rm -f Exe/*      #Remove all executables files of Algorithms
      rm -f menu       #Remove the executable file of menu
      rm -f List.txt   #Remove the text file List
      rm -f Gui/gui_functions.o
```

To start over from scratch, type 'make clean' removes all executables files and the text file, so that the next make rebuilds them. **rm -f** : remove with force.

Data.h: contains the data type used in our application, and the implementation of the Extract function which is used to extract data from the configuration file into an array of records.

Menu.c: is the main program which displays

the list of scheduling algorithms dynamically, give the choice to the user to choose an algorithm from the list displayed on the console and launch directly executable of this algorithm. By the system call, we assure to generate a dynamic menu

```
system("ls Algorithm/ | sed 's/.c//g'> List.txt");
```

ls: to list the content of Algorithm directory.

Sed -n: to remove the .c

> : to redirect the result to the List.txt file.

Menu.exe: is the executable of the menu.c.

List.txt: contains the list of the algorithms generated automatically.

Gui.h: contains the void functions to create the Graphical HMI to follow the progress of the Loading simulation dynamic functions whose source code would be placed in the directory dedicated to scheduling policies.

Gui: this folder includes all the functions files uses in Gui.h.

2. Pilotage Scrum

Our Scrum project organization

Roles

The Scrum Master: Ms. Yosra Najjar

The development team: Nidhal Ghazouani

Maintains detailed product specifications up to date.

Product Backlog:

N'Task	Sprint	Task	priority
1.1	1	Préparation de l'environnement de travail	Haute
1.2	1	Menu file in C	Haute
1.3	1	Makefile	Moyenne
2.1	2	Fifo	Haute
2.2	2	Srt	Haute

2.3	2	Round-robin	Haute
3.1	3	Multi-Level	Haute
3.2	3	Priority	Haute
3.3	3	Graphical HMI	Moyenne
3.4	3	Gant diagram and other functions in header	Moyenne
4.1	4	Generate and save / extract processes	Moyenne
4.2	4	work report + guide	Moyenne

Sprint Backlog

Sprint 1 (October 25 - November 7):

- Préparation de l'environnement de travail
- Menu file in C
- Makefile

Sprint 2 (November 8 - November 15):

- Fifo (First In First Out)
- Srt (Shortest Remaining Time)
- Round-robin

Sprint 3 (November 16 - November 26):

- Multi-Level
- Priority (Ordering by high priority)
- Graphical HMI
- Gant diagram and other functions in the header

Sprint 4 (November 27 - December 6):

- Generate and save/extract processes
- Work report + guide