SchedSim v2

Tomás Eduardo Droppelmann Pidal

December 6, 2021

Contents

1 Project data

- Project supervisor(s): Federico Reghenzani
- Describe in this table the group that is delivering this project:

Last and first name	Person code	Email address
Droppelmann Tomás	10843872	tomaseduardo.droppelmann@mail.polimi.it

- Describe here how development tasks have been subdivided among members of the group:
  - I worked on the complete project (there are no more members).
- Links to the project source code: `https://github.com/tedroppelmann/schedsim_v2`

2 Project description

SchedSim is an application created by Franceso Ratti as a HEAPLab project in Politecnico di Milano. The application is a real-time schedulers simulator, developed using Python and PyQt (GUI tool). The goal of this project is, given a set of tasks (with some attributes) imported in an XML file, to calculate a list with the schedule of all of the tasks imported and sorted through the scheduling algorithm selected. The application displays the schedule through a simple interface to allow the user to visualize the result of the process. Now, the application provides only one scheduling algorithm (Deadline Monotonic algorithm) as an example of its functionality.

The idea of SchedSim v2 is to rebuild from scratch the previous script from SchedSim. In particular, there is no more interest in the GUI, so it seeks to change the graphical interface for an output in text format with all the information about the result of the scheduling algorithm. Besides, the main goal is to implement all the scheduling algorithms seen in classes, which are: First-In-First-Out (FIFO), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Highest Response Ratio Next (HRRN), and Round Robin (RR). To develop this project it will still be used Python but PyQt will no longer be necessary. It should be noted that part of the code of SchedSim is reused in this implementation. It is correctly cited in the code itself.

Task scheduling is one of the main topics in operating systems. Scheduling is the action to assign resources to perform tasks (threads, processes, etc.). To do this, there exist different types of scheduling algorithms, where each one has particular characteristics to focus on some objectives, like maximizing CPU utilization, the throughput, and the fairness, or minimizing the turnaround time, the waiting time, the completion time, and the overhead. Tools like SchedSim and SchedSim v2 could be very useful to understand and compare the performance of the different scheduling algorithms with the same file of tasks.

2.1 Design and implementation

The application contains the next modules:

- **`main.py`**:

  It is the principal module of the project and the executable file. This module verifies if the user fills the correct numbers of arguments to start the program (input path and the output path). Then, it creates the scheduler by calling the `import_file` method of `SchedIO.py`. Finally, executes the scheduler.

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/main.py`

- **`SchedIO.py`**:

  It is the module in charge to read the import file and generate the output file. It contains the function `import_file`, which is in charge to read the XML input file and with this information creates the scheduler (depending on the scheduling algorithm defined in the XML) calling methods from `Scheduler.py` and generates each task of the file using methods from `Task.py`.

  In the case that it is required to add more scheduling algorithms, it is necessary to add in this function the code to read and create the new type of scheduler.

  Besides, it contains the class `SchedulerEventWriter`, which is in charge of creating the output file and generates each line when a new event occurs in the simulation. Each line of the text output file is composed of 6 fields:

  `timestamp,task,job,processor,type_of_event,extra_data`

  where:

  - **timestamp**: the elapsed time since the epoch of measurements (usually 0). This can be measured in different time units (e.g. seconds, milliseconds, clock cycles, etc.). This may not be unique in the file, multiple events may happen at the same time.
  - **task**: the task id. If the event refers to a processor-only event, this value is 0.
  - **job**: the job id. If the event refers to a processor-only event, this value is 0.
  - **processor**: the processor id where the event happened. If the event is not processor-related, this value is 0.
  - **type of event**: the identificator for the event (see later)
  - **extra data**: additional data depending on the event type, this value is 0 if not used.

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/SchedIO.py`

- **`Task.py`**:

  This module contains the class `Task`, which contains all the task attributes read from the XML file. Besides, it contains two attributes that facilitate the implementation of the preemptive algorithms.

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/Task.py`

- **`Scheduler.py`**:

  `Scheduler.py` contains the core of the system: all the scheduling algorithms are located there. It contains a parent class named `Scheduler` with all the common attributes for each particular scheduler, the abstract methods (`execute`, `find_finish_events`) and other common methods (`get_all_arrivals`, `find_arrival_event`, `find_deadline_events`).

  There are two classes that inherit from `Scheduler`: `NonPreemptive` and `Preemptive`. It was decided to divide these two types of schedulers because its implementation is pretty different in some cases, especially when it is necessary to write in the output file a finish event.

  From these two classes inherit all the scheduling algorithms developed, depending on their characteristics (if it is preemptive or not). All of these scheduling algorithms classes follow the same structure and similar methods. Some of these have extra methods to help to calculate necessary values to choose the next event, like `calculate_responsive_ratio` in `HRRN` or `calculate_remaining_time` in `SRTF`.

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/Scheduler.py`

- **`SchedEvent.py`**:

  This module contains the class `ScheduleEvent`, which creates an event of a specific task when it is required. Each event contains all the necessary data to write a line in the output file (see `SchedIO.py`). Besides, it contains additional attributes that facilitate the implementation of the HRRN and SRTF algorithm.

  It also contains the class `EventType` (which inherits from `Enum` class), that defines all the possible event types, which are the following:

  - activation = 'A'
  - deadline = 'D'
  - worst_case_finish_time = 'W'
  - start = 'S'
  - finish = 'F'
  - deadline_miss = 'M'

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/SchedEvent.py`

- **`Cpu.py`**:

  This module contains the class CPU, which creates the cores (or processors) when the XML file is read. In this implementation, the multiprocessor algorithm is not used, but if in the future is necessary, developers could add attributes and methods in this module and code in Scheduler.py how the program chooses a core for each task.

  Link to module: `https://github.com/tedroppelmann/schedsim_v2/blob/main/Cpu.py`

To make the structure of the application clearer, the class diagram is as follows:

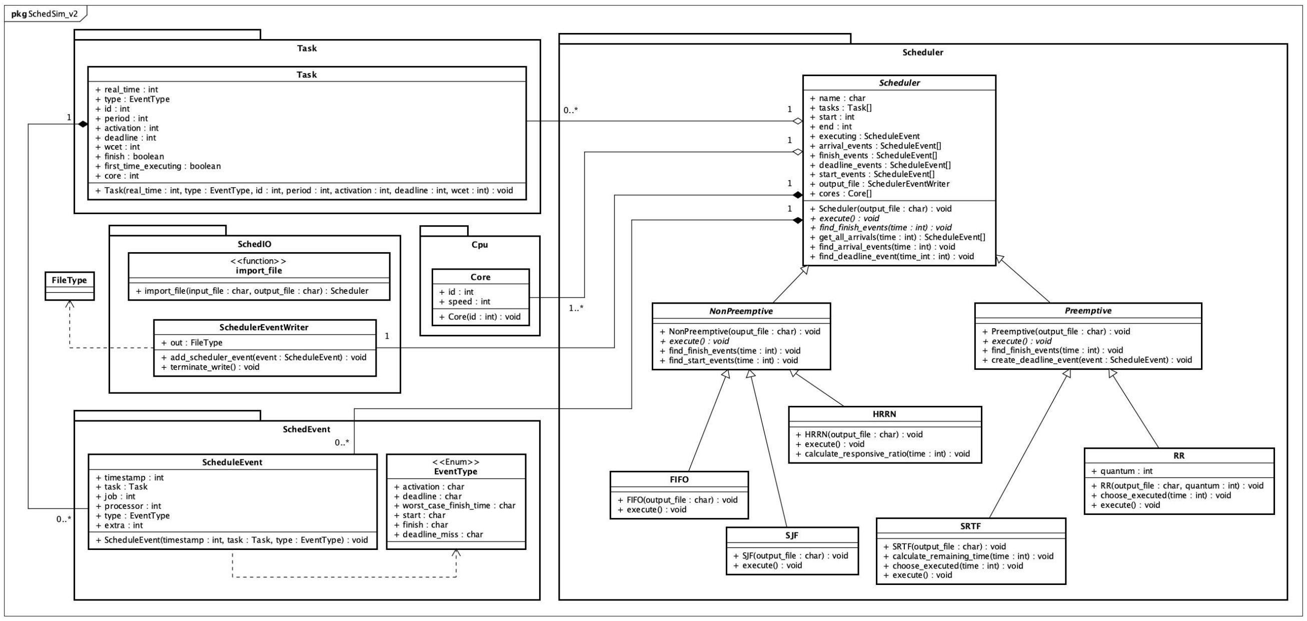

pkg SchedSim_v2
Task
Task
+ real_time : int
+ type : EventType
+ id : int
+ period : int
+ activation : int
+ deadline : int
+ wcet : int
+ finish : boolean
+ first_time_executing : boolean
+ core : int
+ Task(real_time : int, type : EventType, id : int, period : int, activation : int, deadline : int, wcet : int) : void
1
0..*
SchedIO
<<function>>
import_file
+ import_file(input_file : char, output_file : char) : Scheduler
SchedulerEventWriter
+ out : FileType
+ add_scheduler_event(event : ScheduleEvent) : void
+ terminate_write() : void
1
FileType
Cpu
Core
+ id : int
+ speed : int
+ Core(id : int) : void
1..*
SchedEvent
0..*
ScheduleEvent
+ timestamp : int
+ task : Task
+ job : int
+ processor : int
+ type : EventType
+ extra : int
+ ScheduleEvent(timestamp : int, task : Task, type : EventType) : void
0..*
<<Enum>>
EventType
+ activation : char
+ deadline : char
+ worst_case_finish_time : char
+ start : char
+ finish : char
+ deadline_miss : char
Scheduler
Scheduler
+ name : char
+ tasks : Task[]
+ start : int
+ end : int
+ executing : ScheduleEvent
+ arrival_events : ScheduleEvent[]
+ finish_events : ScheduleEvent[]
+ deadline_events : ScheduleEvent[]
+ start_events : ScheduleEvent[]
+ output_file : SchedulerEventWriter
+ cores : Core[]
+ Scheduler(output_file : char) : void
+ execute() : void
+ find_finish_events(time : int) : void
+ get_all_arrivals(time : int) : ScheduleEvent[]
+ find_arrival_events(time : int) : void
+ find_deadline_event(time_int : int) : void
1
1
1
1
NonPreemptive
+ NonPreemptive(ouput_file : char) : void
+ execute() : void
+ find_finish_events(time : int) : void
+ find_start_events(time : int) : void
Preemptive
+ Preemptive(output_file : char) : void
+ execute() : void
+ find_finish_events(time : int) : void
+ create_deadline_event(event : ScheduleEvent) : void
HRRN
+ HRRN(output_file : char) : void
+ execute() : void
+ calculate_responsive_ratio(time : int) : void
FIFO
+ FIFO(output_file : char) : void
+ execute() : void
SJF
+ SJF(output_file : char) : void
+ execute() : void
SRTF
+ SRTF(output_file : char) : void
+ calculate_remaining_time(time : int) : void
+ choose_executed(time : int) : void
+ execute() : void
RR
+ quantum : int
+ RR(output_file : char, quantum : int) : void
+ choose_executed(time : int) : void
+ execute() : void

Figure 1: SchedSim v2 class diagram

In addition, to clarify the sequence of events within the application, the sequence diagram is attached using the FIFO programming algorithm. To reduce the length of the sequence, `alt` CombinedFragments of the application were omitted.

The sequence diagram is in the next link: `https://github.com/tedroppelmann/schedsim_v2/blob/main/docs/diagrams/Sequence%20diagram.jpg`

3 Project outcomes

3.1 Concrete outcomes

To run the application, it is necessary to run the next line in the terminal (located in the application folder). The line receives two arguments: the first is the name of the XML file with the tasks to be scheduled and the second is the name of the text file that will be generated with the final schedule.

Listing 1: Initialize application

```
python3 main.py <input_name.xml> <output_name.txt>
```

The format of the XML input file is the following:

Listing 2: Input file example

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation>
        <time start="0" end="100" />
    <software>
        <tasks>
            <task real-time="true" type="periodic" id="1"
                period="50" deadline="50"  wcet="25" />
            <task real-time="true" type="periodic" id="2"
                period="100" deadline="30" wcet="20" />
            <task real-time="true" type="sporadic" id="3"
                activation="50" deadline="50"  wcet="20" />
            <task real-time="false" type="sporadic" id="4"
                activation="50" wcet="20" />
        </tasks>
        <scheduler algorithm="FIFO"/>
    </software>
    <hardware>
        <cpus>
                <pe id="0" speed="1" />
        </cpus>
    </hardware>
</simulation>
```

It is important to say that for this implementation it was not important the hardware part (number of cores and the speed of each one).

For comparison, it will show the execution of two XML files with the same tasks but different scheduling algorithms: FIFO (non preemptive) and Round Robin (preemptive). We are using the following files as the input files:

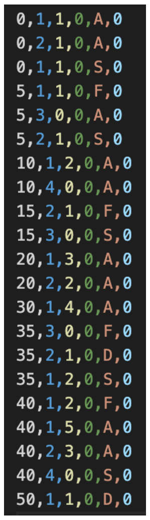
- FIFO input file: `https://github.com/tedroppelmann/schedsim_v2/blob/main/examples/Inputs/example_fifo.xml`
- RR input file: `https://github.com/tedroppelmann/schedsim_v2/blob/main/examples/Inputs/example_rr.xml`

with the following task:

- `<task real-time="true" type="periodic" id="1" period="10" deadline="50" wcet="5" />`
- `<task real-time="true" type="periodic" id="2" period="20" deadline="30" wcet="10" />`
- `<task real-time="true" type="sporadic" id="3" activation="5" deadline="50" wcet="20" />`
- `<task real-time="false" type="sporadic" id="4" activation="10" wcet="15" />`

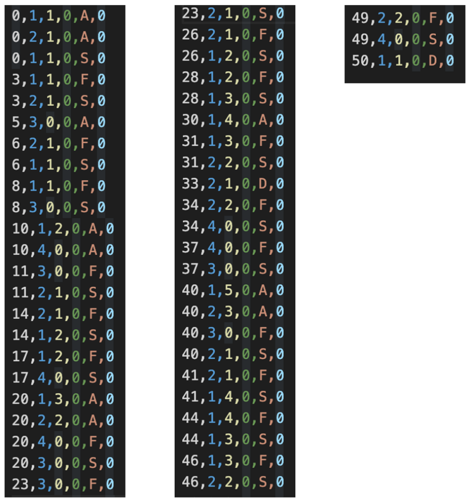
Besides, the quantum for the Round Robin algorithm is 3. The results of the implementation are the following:

- FIFO output file: `https://github.com/tedroppelmann/schedsim_v2/blob/main/examples/Outputs/out_fifo.txt`
- RR output file: `https://github.com/tedroppelmann/schedsim_v2/blob/main/examples/Outputs/out_rr.txt`



```
0,1,1,0,A,0
0,2,1,0,A,0
0,1,1,0,S,0
5,1,1,0,F,0
5,3,0,0,A,0
5,2,1,0,S,0
10,1,2,0,A,0
10,4,0,0,A,0
15,2,1,0,F,0
15,3,0,0,S,0
20,1,3,0,A,0
20,2,2,0,A,0
30,1,4,0,A,0
35,3,0,0,F,0
35,2,1,0,D,0
35,1,2,0,S,0
40,1,2,0,F,0
40,1,5,0,A,0
40,2,3,0,A,0
40,4,0,0,S,0
50,1,1,0,D,0
```

Figure 2: Output file (FIFO example)



```
0,1,1,0,A,0
0,2,1,0,A,0
0,1,1,0,S,0
3,1,1,0,F,0
3,2,1,0,S,0
5,3,0,0,A,0
6,2,1,0,F,0
6,1,1,0,S,0
8,1,1,0,F,0
8,3,0,0,S,0
10,1,2,0,A,0
10,4,0,0,A,0
11,3,0,0,F,0
11,2,1,0,S,0
14,2,1,0,F,0
14,1,2,0,S,0
17,1,2,0,F,0
17,4,0,0,S,0
20,1,3,0,A,0
20,2,2,0,A,0
20,4,0,0,F,0
20,3,0,0,S,0
23,3,0,0,F,0
```

```
23,2,1,0,S,0
26,2,1,0,F,0
26,1,2,0,S,0
28,1,2,0,F,0
28,1,3,0,S,0
30,1,4,0,A,0
31,1,3,0,F,0
31,2,2,0,S,0
33,2,1,0,D,0
34,2,2,0,F,0
34,4,0,0,S,0
37,4,0,0,F,0
37,3,0,0,S,0
40,1,5,0,A,0
40,2,3,0,A,0
40,3,0,0,F,0
40,2,1,0,S,0
41,2,1,0,F,0
41,1,4,0,S,0
44,1,4,0,F,0
44,1,3,0,S,0
46,1,3,0,F,0
46,2,2,0,S,0
```

```
49,2,2,0,F,0
49,4,0,0,S,0
50,1,1,0,D,0
```

Figure 3: Output file (RR example)

3.2 Learning outcomes

- I learned to implement a time simulator. Previously, I had done some simulators but based on events. In the current implementation, the time always runs and in each unit of time the system checks if there are new events to analyze, so it is more similar to the functionality in the real world. This knowledge could be very useful for other projects in the future.
- I deeply learned the functionality of different scheduling algorithms and how they differ from each other, for example in their performance in several aspects (throughput, fairness, waiting time, etc). This knowledge could be useful because it can be extrapolated to other branches of engineering, like logistics, supply chain, etc.
- I polished my Python skills. I had not programmed in Python and in object-oriented programming for a long time, so this project was useful to remind me of the paradigm of this type of programming (it is especially useful now since Python has gained a lot of followers).
- A question that is still open is how would be the performance of the application if it had implemented priority-based scheduling and multiprocessor scheduling. Maybe if somebody else takes this project in the future he/she could improve it with these interesting functionalities.
- I improved my knowledge of Github functionalities. I used Github to save the state of the implementation and order with commits when adding a new feature. It was not necessary to use pull requests because I was working alone. If the project had been in a group, I would have recommended the use of pull requests for the implementation.

3.3 Existing knowledge

I am a double degree student and this is my first year here in Politecnico di Milano. So, the courses of my home university that helped me in the implementation of this project were the following:

- **Advanced programming**: this course gave me the tools to understand object-oriented programming. All projects that I did in this course were coded in Python, so it was very useful for this project.
- **Stochastic systems**: this course is not part of the faculty of Informatics but I did in this course a simulation project, which was very useful to choose the best type of simulation for this project.

- **Software engineering**: in this course, I saw the different UML diagrams that are useful to show a third party the flow and structure of the implementation. I used in this project the class diagram and the sequence diagram.

As I said before, I am a double degree student, so sometimes it is difficult for me to follow the class because some aspects of it are references to previous courses of the bachelor in Politecnico di Milano. My suggestion is that maybe it would be possible to add to lectures or as extra material some topics that would be useful for the people who don't take the previous courses. For example, I have not taken a course in Computer Architectures before, so sometimes when in the slides was code in RISC-V I didn't understand what it means. I think that this extra material would be really useful especially for foreign students.

3.4 Problems encountered

I think that the biggest problem was to think of a way to model the scheduling algorithms such that the code already used was useful when you want to add more scheduling algorithms to the application. I think that separating between preemptive and non preemptive algorithms was a really good choice because the structure of the algorithms inside the same category is pretty similar. Besides, the use of different lists to save each type of event was a really good choice too, because it is useful for almost all scheduling algorithms developed. In this way, it will be easier for the next developers to add more scheduling algorithms following the same logic of the algorithms already developed.

4 References

- Ratti, Francesco. (2021). *SchedSim*. `https://github.com/HEAPLab/schedsim/blob/master/docs/SchedSim.pdf`

5 Honor Pledge

(**This part cannot be modified and it is mandatory to sign it**)

I/We pledge that this work was fully and wholly completed within the criteria established for academic integrity by Politecnico di Milano (Code of Ethics and Conduct) and represents my/our original production, unless otherwise cited.

I/We also understand that this project, if successfully graded, will fulfill part B requirement of the Advanced Operating System course and that it will be considered valid up until the AOS exam of Sept. 2022.

Group Students' signatures