Technical report of scheduling simulator **UNIST**

Advanced Operating Systems Vicente Adolfo Bolea Sánchez (20175319)
Due Date: 21/06/17 <vicente.bolea@gmail.com>

Requirements

Business requirements

The business requirements for this final projects are to implement an scheduler simulator which should be able to support at least the following scheduler algorithms:

1. Shortest job first (SJF).
2. Round Robin (RR).
3. Rate monotonic (RM).
4. Earliest deadline first (EDF).
5. Lottery scheduler (LT).

The simulator must take command line arguments to select in run-time the specified scheduler and its options. In addition to that, the scheduler must read the input from an specified file and print the output to the standard output.

Technical requirements

The definition of technical requirements is an important milestone in the design of a project as it gives the guidelines to implement the high level business requirements into something tangible.

Note that those technical requirements are not given and were chosen by me. I trusted my criteria to chose optimal requirements which allowed me to finish this project successfully. The list of technical requirements goes as follows:

1. It should be generic code, so that, after finishing this project I can reuse some of the components.
2. It should support UNIX platforms with a minimal amount of dependencies.
3. It should be writen in C++14 flavor with GNU/Autotools.
4. The development should be done using *mini-XP* workflow. This is a custom *Pseudo-XP workflow* which I normally use for projects of about 1000 lines of code. [2]

Mini-XP workflow

Mini-XP workflow is a *Pseudo-XP workflow* which I normally use for projects of about 1000 lines of code. During my years of college, I realized that standard development workflows were not suitable for a typical one person project of less than 1000 lines and less than 20 hours of work. To maximize the developing speed and quality I decided to create a workflow which adapts to this specific situation. [2]

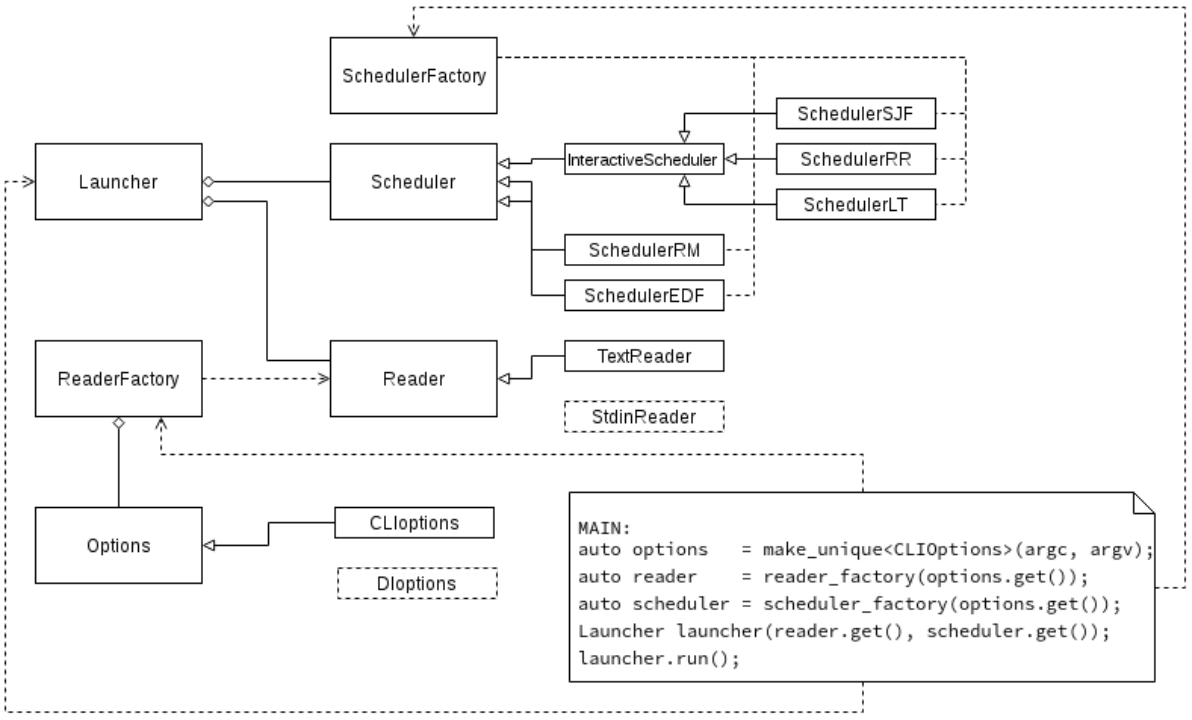
Mini-XP workflow consists in the following items.

1. Top-down approach, minimal requirements and design first.
2. Unit or integrations tests before writing the code (TDD).
3. Using those tests as acceptance tests to prove that the features is implemented.
4. If the tests are passed, refactor the code (Technical debt).
5. Iterate until all the features are completed.

Design

As the first step of the *mini-XP* workflow I created the following class diagram (figure 1).

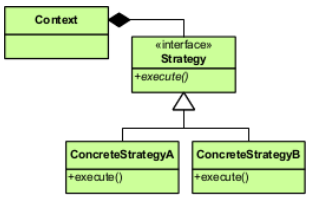
Figure 1: Class diagram



The class diagram tries to impose *Dependency injection* as much as it can so that the components depend on abstractions. This design allows to easily extract components of the simulator into other projects. [1]

Strategy pattern (Figure 2) is used in between the launcher and scheduler class; and the launcher and the *Reader* class. In the former case, Algorithm assumes the role of *Strategy* and each algorithm a concrete strategy. In the later case, *Reader* assumes the role of *Strategy* and each of the reader would be the concrete strategy. In both of the cases Launcher assumes the role of Context. [4]

Figure 2: Strategy pattern



Factory method pattern is also used twice in the project to allow run-time creation of different readers or scheduler algorithms. Note

that I used an public non-member non-static function to implement the factory method pattern, it is not a mistake but a way that allows to add functionality to a class without changing the class invariant and adding dependencies. [3]

As one might interpret from the class diagram, depending of the options we will have different readers and schedulers which are then passed to the launcher object. The launcher object can interact with any of those readers and schedulers since they all share the same interface respectively. Everything else in the code are abstractions to organize the different responsibilities among the classes.

The implementation code (*.cc file) are different for each of the algorithms, explaining each of the algorithm's implementation is out of the scope of this report and can be easily understood following the comments of the source code files.

Lastly, the stub of a main function usually helps to visualize how the framework will be used (Figure 1).

Test Driven Development

As part of the *mini-XP*, I used Test Driven development to implement each of the algorithms. For that matter I created an small python script called `tests/integration-test-runner.py` which evaluates every algorithm with a sample input and output. That script is called from the GNU/Autotools generated *Makefile* rule `make check`. In that manner, I could easily check if the modified code still satisfies the tests by running `make check` at any time. It was truly helpful during the refactoring process to ensure that the refactor does not breaks the code. [5]

How to build the simulator

This simulator uses GNU/Autotools as a building systems. Autotools brings many advantages over other building systems. Maybe the best of them is to be able to distribute you software without requiring to your user to have Autotools.
The building and installation steps described at the listing 1 are the standard Autotools steps.

```
# Get the file from github if you dont have it yet (It will be opened after 22th of June).
wget -O scheduling-simulator-0.0.2.tar.gz goo.gl/2NKfM6

# untar
tar xf scheduling-simulator-0.0.2.tar.gz

# configure the project, prefix value is where the binary will be installed.
cd scheduling-simulator
./configure --prefix `pwd`/build

# Optional, check the tests
make check

# Compile and install it
make install
```

Listing 1: How to build it

How to run the simulator

The previous installation steps will result in the copy of a binary file named `scheduler_sim` in the `prefix/bin` directory. `scheduler_sim` can be call with set of options described in the listing 2. It is important to mention the `-` option for the input file option which allows to read the input from `stdin` instead of a given file, this was needed to make possible to run the integration tests.

```
simulator_sim -s SCHEDULER [OPS] -i FILE
    -s SCHEDULER: SJF,RR,RM,EDF,LT
    -i FILE:      If file is -, it will read the input from stdin
    [OPS]:
      -q #:         Quantum (For RR and LT).
      -e #:         Ending time (For RM and EDF).
      -f:           For LT to use most probable process.
```

Listing 2: How to run it

Note that for checking the unit tests you need to run `make check` instead of `make install`.

Tools

For this assignment I used the following tools:

- Vim, Tmux and bash for editing source code and run it.
- GCC5 and GNU/Autotools for compiling the source code.
- LaTeX and overleaf to create this report.

Additional Information

- The source code can be found at: https://github.com/vicentebolea/scheduling-simulator. Which will be open tentatively by midnight 22/06/2017.
- The implementation of the scheduling algorithms considers the cases of the given sample inputs and outputs. For other cases, there have not been tested. This is for large inputs or corner cases.

References

[1] Wikipedia. *Dependency injection — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2017]. 2017. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Dependency_injection&oldid=786707674%7D.

[2] Wikipedia. *Extreme programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Extreme_programming&oldid=784493371.

[3] Wikipedia. *Factory method pattern — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2017]. 2017. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Factory_method_pattern&oldid=778626347%7D.

[4] Wikipedia. *Strategy pattern — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Strategy_pattern&oldid=781945506.

[5] Wikipedia. *Test-driven development — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Test-driven_development&oldid=785687789.