Getting Started

1. Download the os-coursework1.zip (I'll provide this) archive and unzip it into your workspace.

2. Open the project with IntelliJ. The project should contain several *.java* files stored under the *src/main/java* directory.

3. Compile and run the code. This should work without problems. Note that the source code contains two classes with main methods,
namely *Simulator* and *InputGenerator*.

Simulator

The simulation framework that is contained in *os-coursework1.zip* consists of two components:

- input generator
- simulator

The flow of using these two tools is as follows:

Input Generator

The input generator can be run either through IntelliJ or, directly, from the command line.

As mentioned above, the project contains two main Java classes, therefore you will have to create two separate *run configurations* and set the arguments discussed below*. IntelliJ will automatically create a default configuration if you right-click --> run a specific main class.*

Running the input generator through the command line can be done in different ways; the simplest would be to generate a .jar file through IntelliJ. You are working with maven projects, therefore you can execute the *mvn package* goal which will, by default, generate a .jar file named *os-coursework1-1.0-SNAPSHOT.jar* under the *target* folder. Subsequently you can run the *InputGenerator* with the following command from the base project folder (os-coursework1):

```
java -cp target/os-coursework1-1.0-SNAPSHOT.jar InputGenerator experiment1/input_p
arameters.prp experiment1/inputs.in
```

This will generate an input data file in *../experiment1/inputs.in* based on the parameters provided in the property file *../experiment1/input_parameters.prp*.

The property file contains several parameters to specify (1) the number of processes, (2) their static priority, (3) the mean inter-arrival time for processes, (4) the average duration of CPU and (5) the average duration of I/O bursts and (6) the number of these bursts, as well as (7) a **seed** for the simulator. The total number of parameters is 7.

You will have to run your simulations (see *Simulator* section below) using different seeds (e.g. 5 different seeds would be a good choice) to make sure that your conclusions are robust. A seed is used to initialise (i.e. seed) the pseudorandom number generator which is integral in running a simulation. If you run a simulation with the same seed and input data, you will always get the same output. You will have to change the seed, in order to influence the pseudonumber generator and, therefore, the input data and simulation

The generated input data file contains a line for each process with the first value being the static priority and the second value the arrival time. The remaining values are the durations of alternating CPU and I/O bursts (the number of these values is always odd because we always start and finish with a CPU burst). For example:

```
0 0 15 5 15 5 15
0 10 50
0 25 5
```

Note that you can write such input files manually in order to understand the behaviour of the simulator and to test your implementation.

Simulator

As with the input generator, the simulator can be run through IntelliJ (remember to add the arguments in the run configuration) or the command line. The command to run the simulator using the *jar* file created as discussed above is:

```
java -cp target/os-coursework1-1.0-SNAPSHOT.jar Simulator experiment1/simulator_pa
rameters.prp experiment1/output.out experiment1/inputs.in
```

where *experiment1* is the subfolder included in the .zip file.

This will generate an output file in *../experiment1/output.out* (1) based on the parameters provided in the property file *experiment1/simulator_parameters.prp* and (2) using the input data file e*xperiment1/inputs.in*. Note that you can supply several input files. The contents of **all** supplied files will be considered when running the simulation.

The property file contains parameters that define how the experiment is run:

1. the *scheduler* class to use
2. a potential time limit (*timeLimit*) on the duration of the simulation
3. the duration of interrupts, including scheduler invocations (interruptTime)
4. parameters that are needed for the scheduling algorithms. These are *timeQuantum*, *initialBurstEstimate*, *alphaBurstEstimate* and will be defined below in the specification of the schedulers.

The most important classes are the *Process class* and the *AbstractScheduler* classes which concrete scheduler implementations extend. You are provided with the implementation of a First-Come, First-Served scheduler (see *FcfsScheduler.java*) in the provided IntelliJ project.

An output file looks as follows:

You can copy and paste the content of this file into any spreadsheet program in order to perform further analysis of this output data or use an appropriate language for that (e.g. R, Matlab, Python etc). We assume that the unit of time is *ms*. The simulator logs the events that it executes to the terminal as follows (the number before the colon is the point in time when the event happens):

```
0: CREATE process 1
10: CREATE process 2
15: BLOCK process 1
20: UNBLOCK process 1
25: CREATE process 3
65: TERMINATE process 2
80: BLOCK process 1
85: UNBLOCK process 1
85: TERMINATE process 3
100: TERMINATE process 1
```

Your Implementation

As part of this project you will need to write Java code for:

1. calculating the performance metrics by completing the corresponding functions in the file *Process.java*.

- turnaround time of the process: *getTurnaroundTime*()
- waiting time: *getWaitingTime*()
- response time: *getResponseTime*()

Remark: These functions are called by the simulator when a process terminates to produce the output file. You will be able to compute CPU utilisation and throughput, separately, by analysing the output data.

2. the following scheduling algorithms by completing the corresponding *.java* files. You will have to override some methods from the *AbstractScheduler* class -- read carefully their documentation in the source code:

- Round Robin (*RRScheduler.java*) - Read the *timeQuantum* from the parameters. The scheduler is non-preemptive*.
- Ideal Shortest Job First (IdealSJFScheduler.java) - You can use the *getNextBurst()* method to get the duration of the next burst for each process. The scheduler is non-preemptive.
- Multi-level feedback queue with Round Robin (*FeedbackRRScheduler.java*) - The easiest way to compute a multi-level queue is to use a priority queue where priorities correspond to the levels (lower number means higher priority). Implement the following feedback: *a process is demoted if it used its full time slice*. The scheduler is preemptive.
- Shortest Job First using exponential averaging (*SJFScheduler.java*) - Read the *initialBurstEstimate* (τ0) and *alphaBurstEstimate* (α) from the parameters. For *each process*, use exponential averaging to estimate the

duration of the process' next burst (which will then define the priority of the process) from its previous burst durations. You can use the *getRecentBurst()* method to get the duration of the most recent CPU burst of a process. The scheduler is non-preemptive.

You may add debug output to your implementation. Make sure that you print to **System.out** only.

Remark: Note that there are placeholders, as *TODO* comments, in the code, where you are expected to add code. You may have to create new or override existing methods as well - all abstract methods in the AbstractScheduler class must be overridden, otherwise your code won't compile. Do NOT alter the structure of the given classes but only add code where deemed necessary.

***Important:** here we say that the RR scheduler is non-preemptive. This is because the simulator considers as preemptive a scheduler that will preempt a running process only when a process (new or previously blocked) appears in the *ready* queue, but not when the allocated time quantum is consumed by a process. Dealing with completed time quanta is done at a different point in the code (*setRunning*() in the *BurstProcess* class). The RR scheduler will therefore be dealt with as non-preemptive here, as described above.

Your Experiments

Using the simulator and the schedulers you developed, set up **three** experiments to investigate three different aspects of scheduling algorithms. You are free to choose which aspects you target - it is important that you clearly explain in your report what the specific purpose of each experiment is and which conclusions your draw from the experimental data that you gather.

General questions of interest are for instance:

- How does the process characteristics affect the choice of a good scheduling algorithm?
- What is the influence of the workload?
- What is the effect of the scheduling algorithm parameters?
- How does the cost for running the scheduler affect performance?

Remarks: You will have to adjust the workload (CPU utilisation) of your input data by finding appropriate combinations of parameter values for the input generator. Hint: The CPU time of the *idle process* (process ID 0) tells you something about the CPU utilisation. It is important to present averaged results over several input data sets created using different random seed values. Note that we ask you to investigate specific aspects related to the given schedulers, so the experiments should be designed in a way that each experiment includes all schedulers. Running one experiment (e.g. by changing a specific parameter) using three schedulers does not mean that you are running three experiments!

Your Report

The report should have the following format:

Introduction.

Describe what you are trying to do in this experiment.

Methodology. Describe the experimental setup.

- Describe the experiments you have performed. For each experiment, clearly state:
    - the parameters you used to generate the input data (all of them, even if you only alter a subset of these for an experiment)
    - the parameters you used to run the simulator (all of them, even if you only alter a subset of these for an experiment)
    - the corresponding names of input and output files in your submission
- Discuss the metrics you have chosen to use and the rationale behind your selection
- Discuss how you validated that your experiments produce reasonable results

Results. Present your results in such a way that the reader can draw direct comparisons between schedulers. Use graphs to visualise your results. Tables are also acceptable, but graphs are preferable.

Discussion. Explain how your experimental results support or challenge your hypotheses. Every claim must be supported by evidence - explicitly refer to graphs and tables in the *Results* section.

Threats to validity. Mention any reservations, caveats or biases that may have affected the outcome of your experiments.

Conclusions. Summarise what you have achieved and which insights you gained.

There is no lower or upper word limit. As a rough guide,
the *Introduction* and *Methodology* sections should be one page long.
The *Results* section should be 2 - 4 pages long (most of which should be graphs and tables). The remaining sections should be around one page in total.

IN THE END, I NEED:

- report.pdf - the report as discussed above (in pdf format)
- os-coursework1/
    - *experimentk (*for $k = 1 ... 3)
        - all parameter property files for the simulator used in experiment *k*
        - all parameter property files for the input generator used in experiment *k*
        - all input files (*.in*) generated for experiment *k*
        - all output files (*.out*) produced in experiment *k*
    - src/
        - all *.java* files that were contained in *os-coursework1.zip*, including the 5 files that you modified.
        - *run.sh* or *run.bat*: a script to automatically reproduce the files in *output/* from the files in *input/* and corresponding *parameters.prp* files for each experiment