Evaluation of CPU Scheduling Algorithms

Operating Systems

Coursework 1

The aim of this assignment is to investigate the performance of different CPU scheduling algorithms. You will use a discrete event simulator to conduct experiments on different processor loads and schedulers, and analyse the results to determine in what situations each scheduling algorithm works most effectively. You will then write a report on your experiments, communicating your findings in an effective manner.

1 Learning Outcomes

In this assignment, you will show that you understand the implementation of CPU scheduling, and can demonstrate through simulation the characteristics of different scheduling algorithms. You will show that you can write a report on an experiment and can demonstrate the organisation of experimental data into a format suitable for comparison.

2 Getting Started

1. Download the `os-coursework1.zip` archive from Canvas and unzip it into your workspace.

2. Start NetBeans and open the project. The project should contain several .java files in the `src` directory.

3. Compile and run the code. This should work without problems.

3 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:

$$\begin{tikzcd} \boxed{\text{input generator}} \arrow[r] & \text{input data} \arrow[r] & \boxed{\text{simulator}} \arrow[r] & \text{output data} \end{tikzcd}$$

3.1 Input Generator

The input generator is run as follows, for example:

```
java InputGenerator ../experiment1/input_parameters.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 the number of processes, their static priority, and the means ($1/\lambda$) of the exponential distributions used to draw values for the arrival time, the duration of CPU and I/O bursts and the number of these bursts.
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.

3.2 Simulator

The simulator is run as follows, for example:

```
java Simulator ../experiment1/simulator_parameters.prp
    ../experiment1/output.out ../experiment1/inputs.in
```

This will generate an output file in `../experiment1/output.out` based on the parameters provided in the property file `../experiment1/simulator_parameters.prp` and using the input data file `../experiment1/inputs.in`.

Note that you can supply several input files. The contents of all files supplied will be considered.

The property file contains parameters that define how the experiment is run: which `scheduler` class to use, a potential `timeLimit` for the simulation duration, the `interruptTime` (duration of interrupts, including scheduler invocations), and parameters that are needed for the scheduling algorithms (e.g. `timeQuantum`, `initialBurstEstimate`, `alphaBurstEstimate`).

The most important classes are `Process`, and `AbstractScheduler` that scheduler implementations extend. You are provided with the implementation of a First-Come First-Serve scheduler (`FcfsScheduler.java`).

An output file looks as follows, for instance:

id	priority	createdTime	startedTime	terminatedTime	cpuTime	blockedTime	turnaroundTime	waitingTime	responseTime
2	10	10	15	65	50	0	55	5	5
3	10	25	65	70	5	0	45	40	40
1	14	0	0	105	45	10	105	50	0
0	0	0	0	105	5	0	105	100	0

You can copy and paste the content of this file into any spreadsheet program in order to perform further analysis of this output data.

We assume that the unit of time is *ms* throughout this coursework.

Moreover, the simulator logs the events that it executes to the terminal (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
```

4 Your Implementation

1. Implement the calculation of 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.

Note: You will be able to compute CPU utilisation and throughput by analysing the output data.

2. Implement 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 otherwise.
- Ideal Shortest Job First `IdealSJFScheduler.java`:
  You can use `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` (τ_0) and `alphaBurstEstimate` (α) from the parameters. *For each process*, use exponential averaging to estimate *its* next burst duration (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.

5 Your Experiments

Using your simulator implementation, 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.
- Consider averaging your results over several input data sets with different random seed values.

6 Your Report

The report should have the following format:

Introduction What you are trying to do in this experiment

Methodology What experimental setup you have utilised

- Which experiments you performed
- Clearly state for each experiment
  - which parameters you used to generate the input data
  - which parameters you used to run the simulator
  - the corresponding names of input and output files in your submission
- Which metrics you have chosen to use and why
- 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. Graphs or tables are mandatory.

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. Be as concise as possible and as verbose as necessary. As a rough guidance, Introduction and Methodology will be one page, Results 2–4 pages (most of it graphs and tables), and the remaining sections one page in total.

7 Hand-In

Submit a zip file with the following contents via Canvas:

`report.pdf`

`os-coursework1/`

`experiment`k`/` for $k = 1 \ldots n$

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`k`/`

You will receive 0 marks for your submission if:

- it does not respect the above structure, or
- your code does not compile, or
- your `run.sh` or `run.bat` cannot reproduce your output files.

Avoid academic misconduct: Make sure that your submission is genuinely your own work. Do not collude with your fellow students.

The due date for this assignment is 21 March 2019, 4 PM - in any case, do always make sure to

double-check the deadline published on Canvas / Sussex Direct.