Assignment #3: Process Scheduling

1 Using the Scheduling Simulator

In this practical session, we will use a process scheduling simulator developed by Steve Robbins from the University of Texas in San Antonio.

First, download the simulator *ps.zip* from Moodle and unpack it in some directory.

To launch the simulator, go to the directory containing the unpacked files, then execute *runps.bat*.[1].

The basic functions of the simulator are described further in this section. For more information, please refer to the online documentation: `https://web.archive.org/web/20200217135539/http://classque.cs.utsa.edu/simulators/guides/ps/ps_doc.html` (archived)

The simulator uses several configuration files. For this practical session, the most important ones are *myexp.exp* and *myrun.run*

The experiment file with extension *.exp* specifies a number of experimental runs. Each run is specified by a base name and a possible list of modifications. For the moment, it should contain these lines:

```
1 name myexp
2 comment This experiment contains 2 runs
3 run myrun algorithm FCFS key "FCFS"
4 run myrun algorithm SJF key "SJF"
```

The experiment file format is as follows:

- `name` experiment_name
- `comment` experiment_description
- `run` run_name_1 optional_modification_list_1
- ...
- `run` run_name_n optional_modification_list_n

This sample experiment file executes two runs. All of the runs are based on *myrun.run*. In the first run, the *First-Come/First-Served* scheduling algorithm is used, and for the second one *Shortest Job First* is employed. The name of the run is specified using the `key` parameter.

In figure 1 the interface of the simulator is presented. To run all experiments press the *"Run Experiment"* button (mark 3 in the figure). To run only one of the experiments click on the button above the Run Experiment button (by default it displays *"Run All"*). After executing the simulation, the results of the runs are shown in the log window (marked 1).

[1] In a UNIX environment, use the following command: `java -cp ./PS.jar:./Jeli.jar PS`

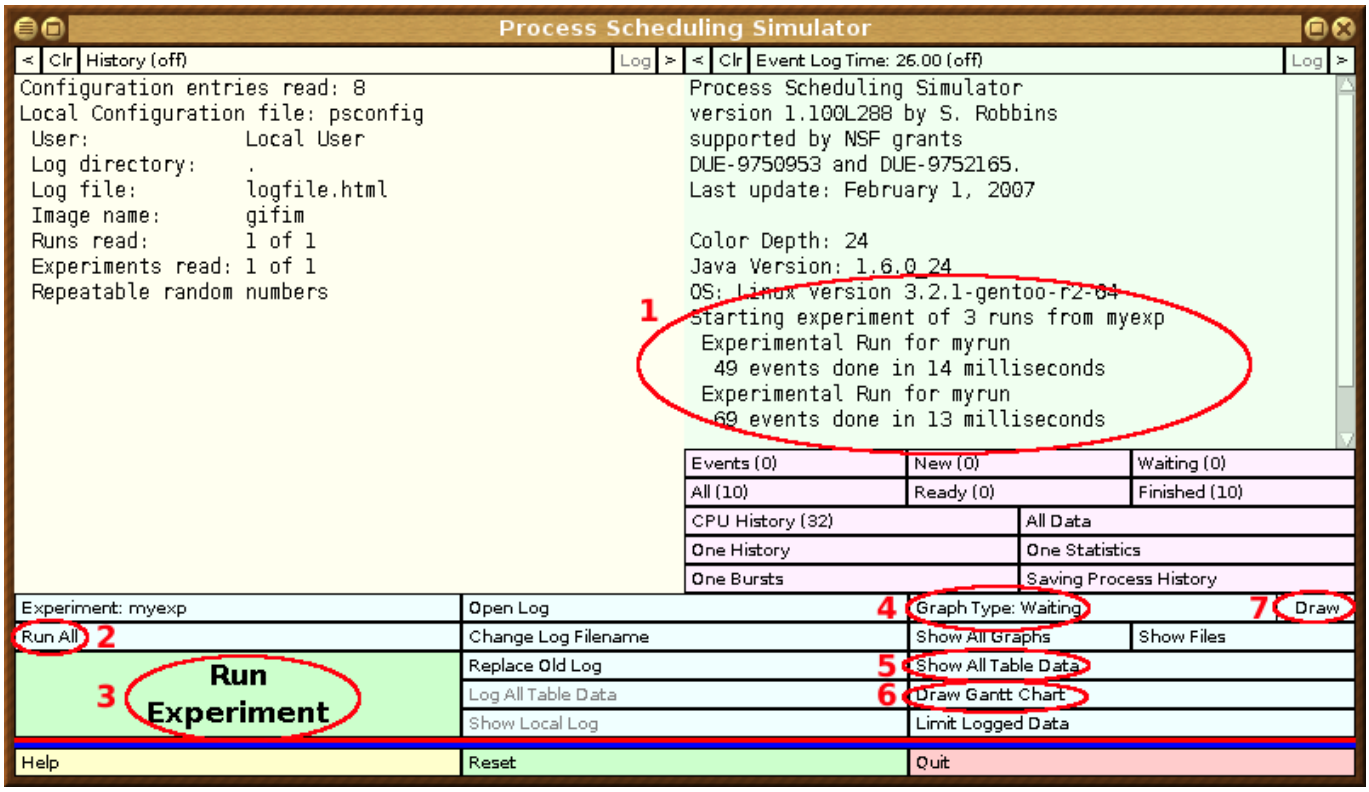

Figure 1: Interface of the scheduling simulator with main control elements marked.

The parameters of the processes to be simulated are configured in the *myrun.run* configuration file. The simulator will simulate the execution of the specified processes according to the defined scheduling algorithm. Here is an example:

```
1  name myrun
2  algorithm FCFS
3  numprocs 4
4  firstarrival 0.0
5  interarrival constant 0.0
6  duration constant 2.0
7  cpuburst constant 1.0
8  ioburst constant 10
9  basepriority 1.0
10
11 numprocs 5
12 firstarrival 0.0
13 interarrival constant 0.0
14 duration constant 6.0
15 cpuburst constant 3.0
16 ioburst constant 2.0
17 basepriority 1.0
18
19 numprocs 1
20 firstarrival 5.0
21 interarrival constant 0.0
22 duration constant 1.0
23 cpuburst constant 0.5
24 ioburst constant 1.0
25 basepriority 1.0
```

The name of the run is specified by the `name` parameter on the first line. A scheduling algorithm is specified with a string and a possible optional floating point parameter for some of the scheduling algorithms:

- `RR` *quantum* represents the Round Robin algorithm with the given quantum.
- `FCFS` represents the First-Come/First-Served algorithm.
- `SJF` represents Shortest Job First scheduling.
- `PSJF` represents Preemptive Shortest Job First scheduling.
- `SJFA` *alpha* represents the Shortest Job First Approximation by exponential averaging, using the given value of alpha (α).

The other parameters configure the following properties:

- `numprocs`: number of processes to create.
- `firstarrival`: arrival time of the first process, a floating point number.
- `basepriority`: base priority of the processes created. This base priority is not used by default.
- `interarrival`: string representing the distribution of the inter-arrival times of the processes.
- `duration`: string representing the distribution of the total CPU time used by each of the processes. Notice that a task is completed when it has executed this amount of CPU time, I/O is not counted towards the total running time.
- `cpuburst`: string representing the distribution of the CPU burst times of each of the processes.
- `ioburst`: string representing the distribution of the I/O burst times of each of the processes.

Possible string-represented distributions are the following:

- `constant`: Uses a constant value (e.g.: `constant 23.45`).
- `exponential`: Uses an exponential distribution with the specified mean (e.g.: `exponential 23.45`).
- `uniform`: An uniform distribution between two values, inclusive (e.g.: `uniform 23.45 47.89`).

Results of the simulation are presented as charts and tables. In the following example there are three runs (each one with different scheduling algorithm). Figures 2, 3, and 4 show the Gantt charts, which can be obtained by pressing the *"Draw Gantt Chart"* button (marker 6).

Several other types of charts can be generated by specifying the type of the chart (button *Graph type*, marker 4) and then pressing the *"Draw"* button (marker 7).

Figure 5 shows the tables containing statistics on the performed runs, which is generated by pressing the *"Show All Table Data"* button (marker 5).

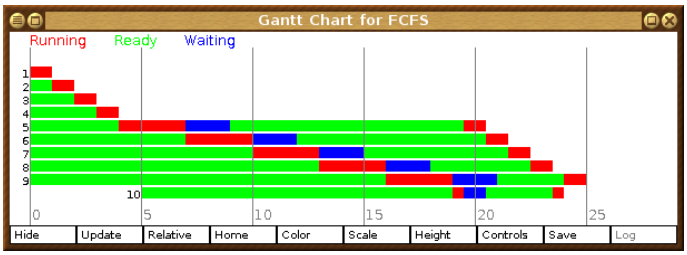

Figure 2: Example of a Gantt Chart for the FCFS algorithm

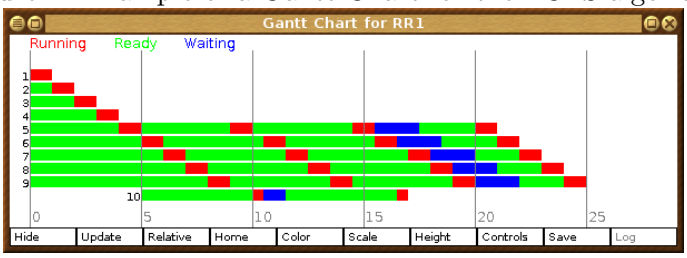

Figure 3: Example of a Gantt Chart for the RR algorithm

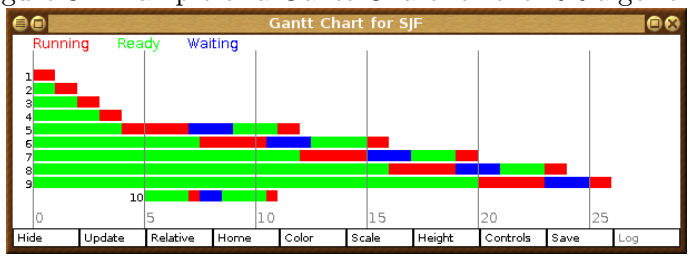

Figure 4: Example of a Gantt Chart for the SJF algorithm

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
myrun_1	FCFS	25.00	10	10	1.000000	.400000	0.00	4.24	16	6	1.56	1.83
myrun_2	RR1	25.00	10	10	1.000000	.400000	0.00	4.04	26	6	.96	1.83
myrun_3	SJF	26.00	10	10	.961538	.384615	0.00	3.00	16	6	1.56	1.83

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
myrun_1	FCFS	14.20	1.00	25.00	9.70	10.60	0.00	19.00	.75
myrun_2	RR1	13.70	1.00	25.00	9.76	10.10	0.00	19.00	.74
myrun_3	SJF	11.40	1.00	26.00	9.05	7.80	0.00	20.00	.69

Done

Figure 5: Simulated run statistics

2 Experiments

During this practical, your goal is to experiment with different scheduling algorithms under different workloads. You should submit by April 1st 2022 at 14:00. a short report (at most 2 to 3 pages are expected in **PDF** format) containing your results. For each of the tasks, the report shall contain the following items:

- A table with the performance summary of each algorithm (CPU usage, throughput, turn around time, waiting time); a template using Excel is provided on the Moodle course page.
- A concise description and analysis of the results you obtained, e.g., performance of the various algorithms in relation to the task.

The Gantt charts of the experiments may be used inside your report to detail your explanation. Apart from the report, you should also submit the modified run and exp files for each task. Those files shall be named according to the task number *N*, e.g., *N.exp*.

2.1 Setup

We first have to set-up a proper environment to run our experiments. To that end, you first have to create a *run* file for each of the workload descriptions below. The simulator loads a run file when it appears in the *psconfig* file using the `run` keyword.

- *Interactive*: Interactive applications are known for sitting mostly idle, and reacting to user input. To model this behavior, you should use a series of short CPU-bursts (reaction to program input) followed by a varying (long) waiting time (waiting for input). Use a CPU burst constant of 0.5 with IO bursts following a uniform distribution of 1 to 5 with a total CPU duration for each process of 5.
- *CPU-Intensive*: High performance applications mostly perform long CPU bursts followed by short IO bursts (usually writing results to disk). Use a constant CPU burst of 10 with exponential IO bursts with mean 1.5. Total CPU duration for each process is 40.
- *Mixed-Load*: Represents a mixed load in which neither IO nor CPU bursts predominate. Use uniform bursts for both CPU and IO in the range 1.0 to 5.0. Total CPU duration for each process is 20.

The common properties for these workloads are:

- There should be five processes for each type (`numprocs` parameter).
- They should all arrive at time 0 (`firstarrival` parameter).
- Use a seed value (`seed` parameter). A seed is required to use a *deterministic* random number generator. This way the workload simulated will be the same for all schedulers, which allows us to compare them from the obtained results. Choose any integer number for the seed value.

Once you have completed the workload definitions, you may ask the assistant to check that everything is configured properly.

In what follows, we will use tha above workloads to compare the following schedulers: FCFS, RR (with a quantum of one), SJF and SJFA (with an alpha of 0.5).

2.2 Tasks Description

T1 – Individual Workload In this task, you compare the performance of each scheduling algorithms when running a single type of workload. The following metrics are used to compare the algorithms:

- *CPU utilization*
- *Throughput*
- *Turnaround time* (average and standard deviation)
- *Waiting time* (average and standard deviation)

This data is presented in the tables (*"Show All Table Data"* button). Additionally, the Gantt graphs can be used to study in details how the scheduler is behaving (*"Draw Gantt Chart"* button).

T2 – Mixing Workloads In the second part of the report, we compare the scheduling algorithms when running two workloads in parallel. Select two out of the three types of task (Interactive, CPU intensive, Mixed) to simulate. As before, compare the performance of the schedulers, and in particular how one type of task affects the other type.

The next tasks correspond to adjustments of the scheduling algorithms. In what follows, you should simulate execution of task separately (i.e. it is not needed to analyze how scheduling performs in case of several type of tasks mixed together). Focus on how the performance changes compared with the default setup.

T3 – Varying the Round Robin Time Slice The Round Robin algorithm currently has a time-slice of 1. But what is the effect of reducing, or increasing this value? Compare the overall performance of RR when the time-slice is one, versus the values of: 5, 10, 0.5, and 0.1. By default the simulator assumes that there is no *dispatch latency* (*context-switch time* CST in terms of the current simulator) overhead. Describe in the report how would these results change if there is a dispatch latency introduced in the simulation? In the run file add a context switch time in/out of 0.1 (keywords `cstin` and `cstout`). Re-run the experiments and compare.

T4 – High Load Performance In this last experiment, we compare the performance of the schedulers using a much greater amount of tasks. Instead of having each five tasks that start at time zero for each type of task, increase the number of tasks to 500 (`numprocs` parameter), arriving at an uniform interval between:

- *Interactive:* between 4.0 and 12.5
- *CPU Intensive:* between 40 and 45
- *Mixed:* between 20 and 100