Operating Systems Assignment: Scheduling

1. Introduction

This assignment has the objective of developing your understanding of CPU process scheduling through the construction of simulations of 'First-Come, First-Served' (FCFS) and 'Round-Robin' (RR) scheduling strategies.

You will construct two simulator programs, one for FCFS, and one for RR, then you will experiment with various simulated workloads to verify known characteristics of the strategies.

1.1 Simulation principle

The principle is that your programs simulate execution of a set of pseudo programs from the point of view of process scheduling.

These pseudo programs are abstractions, as illustrated by the following example:

```
# firefox.dat
# CPU <burst duration>
# IO <burst duration> <device id>
CPU 2
IO 2 1
CPU 3
IO 5 2
CPU 1
IO 4 1
CPU 2
```

- A line beginning with '`CPU`' is a 'process instruction'. It is an abstract description of a block of program code that is executed on the system CPU. To put it another way, it describes a CPU burst – a period of time during which the program only uses the CPU.
- A line beginning with '`IO`' is an 'I/O instruction'. It is an abstract description of a block of program code representing I/O activity. To put it another way, it describes an IO burst – a period of time during which the program waits for I/O.

A simulator workload is described in a configuration file such as follows:

```
# I/O Devices attached to the system.
# DEVICE <device id> <type>
DEVICE 1 DISK
DEVICE 2 CDROM
# Programs
# PROGRAM <arrival time> <priority> <program file name>
PROGRAM 8 0 firefox.dat
PROGRAM 5 0 primescalculator.dat
PROGRAM 11 0 spice.dat
```

1.2 Framework design

On the Vula assignment page you will find a (i) Java framework within which you will develop your simulations, and (ii) a document describing the framework design.

The framework defines five types of component: kernel, CPU, system timer, I/O device, and simulation configurator.

- The kernel simulates aspects of kernel behaviour relating to scheduling:
  - Processes are represented by Process Control Blocks (PCB)
  - A 'ready queue' holds processes waiting to be executed.
  - One or more device queues hold processes waiting for I/O to complete.
  - As events unfold, processes are moved between queues and onto and off the CPU.
- The CPU simulates the processing of program instructions.
- The system timer records the current system time, the time spent in user space, the time spent in kernel space, and the time spent idle.
  The timer is advanced when:
  - the CPU processes a program instruction, or
  - kernel code is executed, either through system call or interrupt.
- An I/O device simulates the infrastructure needed for an I/O call: the process joins a queue to access the resource, and is released when the operation is completed.

Two kinds of component are defined in abstract: Kernel and ProcessControlBlock. These are interface declarations.

To develop a simulator, you must construct:

(i) a concrete type of ProcessControlBlock,
(ii) a concrete type of Kernel, and
(iii) a driver program that can, using the simulation configurator, set up and run a simulation: read in configuration data from a file, 'run' the programs, and write simulation data out to the screen.

Further framework details are available in the aforementioned document.

2. Tasks

A precise description of your tasks follows.

2.1 First-Come, First-Served Simulator [35 marks]

Using the supplied framework, construct an FCFS simulator called 'SimulateFCFS' that uses a kernel class called 'FCFSKernel'.

The program will exhibit the following behaviour:

```
*** FCFS Simulator ***
Enter configuration file name: Test2C/config.cfg
Enter cost of system call: 1
Enter cost of context switch: 3
Enter trace level: 0

*** Results ***
System time: 69
Kernel time: 26
User time: 35
```

```
Idle time: 8
Context switches: 6
CPU utilization: 50.72
```

The trace level specifies the degree of detail shown of the simulated execution. In this example, no trace information is requested.

By way of a further example, a trace level of 1 outputs a trace of context switches:

```
*** FCFS Simulator ***
Enter configuration file name: Test2C/config.cfg
Enter cost of system call: 1
Enter cost of context switch: 3
Enter trace level: 0

*** Trace ***
Time: 0000000001 Kernel: Context Switch {Idle}, process(pid=1, state=READY,
name="Test2C/programA.prg")).
Time: 0000000016 Kernel: Context Switch process(pid=1, state=WAITING,
name="Test2C/programA.prg"), process(pid=2, state=READY,
name="Test2C/programB.prg")).
Time: 0000000025 Kernel: Context Switch process(pid=2, state=WAITING,
name="Test2C/programB.prg"), {Idle}).
Time: 0000000037 Kernel: Context Switch {Idle}, process(pid=1, state=READY,
name="Test2C/programA.prg")).
Time: 0000000052 Kernel: Context Switch process(pid=1, state=TERMINATED,
name="Test2C/programA.prg"), process(pid=2, state=READY,
name="Test2C/programB.prg")).
Time: 0000000066 Kernel: Context Switch process(pid=2, state=TERMINATED,
name="Test2C/programB.prg"), {Idle}).

*** Results ***
System time: 69
Kernel time: 26
User time: 35
Idle time: 8
Context switches: 6
CPU utilization: 50.72
```

Tracing capability is built into the supplied framework. (See the simulator.TRACE class for details.)

In the framework '`.zip`' file you will find a directory called '`tests`' that contains sub directories containing test workloads and associated trace output ('`.log`' files). You can use these to check your FCFS simulator functions correctly.

On the Vula assignment page you will find skeleton code for the 'FCFSKernel ' class.

2.2 FCFS Convoy [15 Marks]

Referring to the course text ('Operating Systems Concepts'), in the chapter on CPU/Process scheduling the authors describe a phenomenon that they call the 'convoy effect'. Devise a simulator workload that demonstrates this effect.

We suggest the following strategy:

- Devise a large number of I/O bound programs (programs with generally small CPU bursts).
- Devise a program that is initially also I/O bound but that becomes CPU bound.
- Study a trace of the resulting cycles of scheduling behaviour for a transition to 'convoy-like' behaviour.

Provide a written justification as to how your chosen workload demonstrates the effect.

- Put the justification in a document called 'Scheduling[.txt|.doc]' i.e. a text or word document called 'Scheduling'.
- Your justification can refer to events in the event trace that the workload generates.
- If you wish, you can use the TRACE facility to instrument your kernel so that it prints out a more concise trace of events. Use trace codes of '64', '128', '256', ...
- Provided along with the framework code is a 'program generator' that you may wish to use to automate workload construction.

2.3 Round-Robin Simulator [30 marks]

Using the framework, construct a Round-robin simulator called 'SimulateRR' that uses a kernel class called 'RoundRobinKernel'. (You should be able to reuse your concrete ProcessControlBlock class and chunks of your FCFSKernel and SimulateFCFS classes.)

The program will exhibit the following behaviour:

```
*** RR Simulator ***
Enter configuration file name: Test1F/config.cfg
Enter slice time: 80
Enter cost of system call: 1
Enter cost of context switch: 3
Enter trace level: 0

*** Results ***
System time: 103
Kernel time: 13
User time: 90
Idle time: 0
Context switches: 3
CPU utilization: 87.38
```

Note that the interaction is much the same as for SimulateFCFS. The key difference is that the program asks for the slice time (time quantum) as well as system call and context switch costs.

In the framework '`.zip`' file you will find a directory called '`tests`' that contains sub directories containing test workloads and associated trace output ('`.log`' files). You can use these to check your RR simulator functions correctly.

2.4 Round-Robin rule of thumb [20 marks]

Referring to the course text, 'Operating Systems Concepts', in the chapter on CPU scheduling the authors state that, as a rule of thumb, 80% of the CPU bursts [in a workload] should be shorter than the time quantum.

Use your simulator to conduct a rudimentary investigation of this rule of thumb. We suggest you:

- Develop a mixed workload: I/O bound programs, CPU bound programs, and balanced.
- Experiment by varying time slice length to identify the optimum.

Write up your findings in your 'Scheduling[.txt|.doc]' documents.

3. Marking and submission

You will submit your materials for assessment via the automatic marker within a single '.zip' file bearing your student number.

The file should contain the following:

- A folder called '`src`' that contains your code and the '`simulator`' package folder i.e.
  `src/SimulateFCFS.java`
  `src/SimulateRR.java`
  …
  `src/simulator/Config.java`
  `src/simulator/CPUInstruction.java`
  …
  It should be possible to compile your code from the `src` folder.
- A folder called FCFS-Convoy that contains the configuration file and pseudo programs that demonstrate the convoy effect (your solution to task 2).
- A folder called Thumb that contains the configuration file and pseudo programs used in your investigation of the 80% rule of thumb (task 4).
- Your 'Scheduling[.txt|doc]'.

The automatic marker will assess your solutions for tasks 1 and 3. [If you have instrumented your code so that it produces additional trace output, make sure that it does not interfere with the built-in tracing.]

END