

**The University of Queensland  
School of Information Technology and Electrical Engineering  
Semester 2, 2008**

COMP3301 COMP7308

**ASSIGNMENT 1 – Scheduling**

Due: 5pm Friday 28th August

Weighting: 25% (25 marks COMP3301, 30 marks COMP7308)

**Objective**

As part of the assessment for this course, students are required to complete three assignments which will assess your understanding of the Minix operating system and ability to program at a low level in C. This particular assignment will assess your ability to understand the scheduling of processes in Minix and the implementation of a different scheduling algorithm for Minix.

**The Assignment**

The Minix scheduling algorithm will be explained in lectures briefly. Your task in this assignment is to replace the current algorithm with an **lottery-based implementation of Fair-share scheduling**. Your implementation of this scheduling method should not break current system calls and programs packaged with Minix – for example nice(1).

You should base your version on that explained in the textbook (pages 107-109). In your implementation, root processes are always run before any non-root processes, but are otherwise unaffected by the new scheduling algorithm (i.e. root processes are given first priority and are not fair-shared). After all root processes are run, each user (other than root) is allocated an equal share of the CPU time, regardless of the number of processes that user has running.

Each non-root user is then able to define the priority of its own processes relative to other processes it owns. There will be 3 priority levels for non-root processes (instead of Minix's typical 15). How you implement the new levelling scheme is up to you (e.g. new library calls, re-interpreting nice, etc...).

The ratio for levels 1,2,3 are 4:2:1 respectively. (i.e. for every 4 quanta a priority 1 process gets, a priority 2 process will get 2 quanta and a priority 3 process will get 1 quanta).

Consider the following examples, where users U1 and U2 create a number of processes P1..Pn.

In Example 1, U1:P1 and U1:P2 each get 44%, and U1:P3 gets 11%. In Example 2, U1:P1 (33%), U1:P2(16%), U2:P1(28%), U2:P2(14%) and U2:P3(7%).

**Example 1.**

Priority Ratio	Priority	User Processes
4	1	U1:P1 U1:P2
2	2	
1	3	U1:P3

**Example 2.**

Priority Ratio	Priority	User Processes
4	1	U1:P1 U2:P1
2	2	U1:P2 U2:P2
1	3	U2:P3

You are required to alter the operating system source appropriately and generate a patch for a default kernel image. Your code should be appropriately commented to describe your code. In addition to this, you will be required to submit a user application(s) which will demonstrate the correct functionality of your code. This code should include a README file, describing your solution and how the user application demonstrates the scheduling algorithm.

## Marking

Your submission will include the following four files:

- **s123456\_ass1\_final.patch** (where 123456 is replaced with your student number)
- **build.sh** (a shell script which will rebuild the kernel appropriately)
- **demo\_program.tar**, containing
  - application code for a program or programs to demonstrates the operation of your scheduling algorithm
  - A Makefile to build the demo program(s)
- a **README** file containing two sections (plain text or PDF), 1-3 pages in 12 point font.
  - The first section will describe your design and implementation of lottery/fair-share scheduling
  - The second section will describe how your demo program(s) exercise the functionality, and how it is to be tested.
- **COMP7308 only** - Write a 3rd section in your README (about 300-400 words) , discussing
  - 3 ways in which your implementation might be improved (design, efficiency etc), and
  - outline an alternative approach to the design and implementation of the lottery / fair share scheduler.

In order to mark your assignment, we will use a default Minix image (which we will provide for your own testing purposes). The patch file will be applied in the /usr/src directory with the following command:

```
patch -p0 < s123456_ass1_final.patch
```

The kernel will then be built using your supplied shell script and rebooted. Your programs will be tested by extracting them to /home and following the instructions in your README. Ensure that your submission will work correctly according to this procedure – particularly ensure that the patch file applies cleanly to the provided image.

## Hints

- Process scheduling is implemented in the kernel, while the user IDs associated with each process are only known to the process manager (PM). You will need to work out a way to alter scheduling behaviour based on information managed by the PM.
- There are many ways of implementing this assignment – some of which will be more difficult than others. Examine the existing code *carefully* and do plenty of thinking *before* you start coding!

## Help sources

Help for this assignment is available from a number of sources. Help in understanding the Minix code is best obtained either from the textbook, or from the course newsgroup, uq.itee.comp3301. Help in understanding the algorithm is best obtained either from the internet or other textbooks.

The tutor is available to provide assistance during the scheduled practical sessions in 78-110. If none of these methods provide the answers you need, the lecturers will be available for consultation - please email for an appointment.

### Submission Details

The due date for the assignment is 5pm Friday 28th August. The assignment must be submitted via the ITEE online submission system. Assignments may be resubmitted numerous times, but only the last submission will be marked.

### Late Submission

Late submissions will **not be accepted except with a doctor's note and even then by permission of the course coordinator.**

### Notification of Results

Students will be notified of their results via the course homepage. Students will be able to view their individual results only.

### Allocation of Marks

Students will be marked on their ability to implement changes in the kernel code and the demonstration of how those changes can be seen in the operation of the kernel.

	Marks available
<b>Coding and Testing (17 marks)</b>	
Lottery scheduling mechanism design/execution	0-5
Fair-share scheduling policy design/execution	0-5
Suitable scheme for assigning task priorities	0-2
Demonstration Program(s)	0-5
<b>Documentation and style (8 marks)</b>	
Kernel Design/Implementation documentation	0-3
Demo Program Design/Implementation documentation	0-3
Coding style and commenting	0-2
Design analysis and alternative design discussion <b>(COMP7308 ONLY)</b>	0-5
<b>Caps</b>	
System fails to compile	Capped at 10
System fails to boot / is unusable	Capped at 13

### Academic Merit, Plagiarism, Collusion and Other Misconduct

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. You should note that this is an **individual assignment**.

### Reference

Tanenbaum and Woodhull, Operating Systems: Design and Implementation (3<sup>rd</sup> Edition), Prentice Hall, 2006.