**Operating Systems**

**Term Project**

All source codes including files you changed and those you created, and the output should be uploaded to Blackboard using the Project Link in Week 9 folder.

1. **Project 1**

**CPU Scheduling**

**Goals**

* To build and understand CPU time-sharing scheduler implementations
* To gain an insightful experience to the issues of scheduling on multiprocessors
* To gain some insight to the correlation between scheduling and other parts of the OS (e.g., I/O and memory management)
* To have an idea what the scheduler wants to achieve you might "verify" that your implementation has achieved those goals.

**Overview**

The schedulers that we have discussed so far has no fairness except maybe round-robin (RR) which is the primary Linux concept and fairness is the part of RR. In scheduling, there should be fairness as we demand fairness in our daily lives. In processes, the operating system does well to achieve a sense of fairness in the use of the CPU when there is a given priority over a period of time. One main issue of RR scheduling is that it is *per process fairness*, and not *per user fairness.*  This means that a user who is running many processes will receive a much larger CPU share than a user who is a running a few processes.

Develop a scheduler with the aim to achieve equal allocation per user. For example if there are four processes each process should receive a fair amount of CPU time.

This Project/assignment involves you to design, implement, and evaluate a fair-share scheduler in Linux.

***Obstacles:***

As part of this assignment you will decide the objectives of your scheduler and what you used to perform the scheduling.

Make all decisions that you think are appropriate and discuss that in your project

Part 1

In Part 1, please read and understand the source code to the Linux schedulers and our schedulers, by running and revisiting the code and how it functions. Design on paper how you want your fair-share scheduler to work, and implement your design and test it several times to evaluate your implementation. List all your observations and arrive at a valid conclusion for your scheduler.

**WHAT TO TURN IN FOR PART 1**

As part of your project/assignment, you need to answer the following questions:

1. What does your current scheduler do? Please consider and address the following inquiry:
   * What is your scheduling mechanism?  Be explicit and detailed.  (Description and specification of your scheduler)
   * Can other processes be starved in your scheduling?  If so, give an example of how it might occur and how starvation can be avoided.
   * Is there aging?  That is, are the priorities of processes that have low recent CPU consumption raised to avoid effective starvation?
2. What are the objectives of your fair-share scheduler? Discuss how you want your "fair-share" to mean.  Your answers to these inquiry are best answered by referring to your implementation of your scheduler (just as exactly what your scheduler does can be clear only by its implementation).
3. How to you plan to transform our schedulers’ in week 5 implementation to achieve your fair-share scheduling?
4. How will you evaluate your modified scheduler? Your evaluation should test whether each of the objectives you set for your scheduler are met, within the bounds of what is realistically possible given the time allotted for this assignment.
5. Use Gantt charts to illustrate the execution of the processes

**PART 2:**

* A short report to discuss how and what you might if done differently from your plans from Part 1
* A detail analysis of the experiments you actually ran to corroborate your implementation, their results, and an explanation of the results. (For example, if your scheduler did not behave as you had expected, some explanation for why.)

1. **Project 2**

**Kernel Module**

Computers have come a long way to stay. Studying operating systems allow us to explore architecture, organization and the software part of computers. In this project, we are going to develop our own hardware and software that will allow users to use the hardware. Device drivers are used to enable a hardware to work. Develop your own hardware and create your own device driver (kernel module).

The kernel module should use the name of your hardware and be licensed to you. Load the module and remove it.

Files to Submit

1. One Page description of your hardware and kernel
2. Source Files – including your Makefile
3. Output
4. Documentation

I hope we can invent a hardware and its device driver to warrant further research