# Lab 3– Process Scheduling

# Due date: 22-Nov-2016

## Introduction

Modern operating system employ scheduling algorithms that are based on the round-robin concept as described in class. The scheduling policy is also based on ranking processes according to their priority. Complicated algorithms are used to calculate the current priority of a process and scheduling decisions are based on the value of each process's priority.

For this lab, you are to implement three scheduling algorithms:

a) Multi-level Feedback Queue Scheduler (MFQS)
b) Real-Time Scheduler (RTS)
c) Windows Hybrid Scheduler (WHS)

You are also required to write a technical paper to evaluate the performance of each algorithm. The paper must comply with the IEEE paper format.

## Program Efficiency

Your program must be efficient, i.e., it must not take a long time to compute the result. The following is a breakdown of the grading of this project:

Stress Tests Efficiency: 40%
Correctness: 40%
Analysis Paper: 15%
Documentation: 5%

## MFQS

You are to implement the scheduler described in class. Listed below are some of the parameters that must be considered in your simulation.

0 Number of queues:
variable, upper bound = 5 (ask user to input number)

1 Scheduling algorithms for each queue:
a. Round Robin for all except last queue (FCFS).
b. Time quantum: doubled for each subsequent queue below it.

2 Method used to determine when to demote a process:
Processes that used up their time quantum and still cannot complete are demoted.

3 Aging: when a process waits in a queue for more than some specified value (value is prompted). Apply this all but the first queue.

If a process is running RR in the third queue and a new process arrives in the first queue, you allow the process in the third queue to finish its time quantum before handling the new arrival. This is applicable to all schedulers in this lab assignment.

Note that whenever a tie has to be resolved, the priority of the process is considered, if the tie still exists, then the smaller PID of a process is used to break the tie. An exception to this is when a process finishes its time quantum and is demoted to the queue below at the same time as a another process which is to age up to the same queue, the process that ages up will be inserted into the queue before the demoted process. For example, if we have three queues, A (top), B (middle) and C (bottom), and process 1 just finished its time quantum in queue A and is to be demoted to queue B. At the same time process 2 is supposed to age up to queue B. Process 2 will enter queue B before process 1.

For MFQS, processes need not do I/O.

Processes do ***not*** age up to the top queue. Is it a good idea to permit processes in all but the top queue to age up? Discuss this in your paper.

## Real-Time Scheduler (RTS)

Implement the algorithm described in class.

a) Account for both soft and hard RT environments.
b) Report any process that cannot be scheduled (in soft RT only)

You can assume that processes do not do I/O for this scheduler. When a process cannot meet its deadline, report it and take it out of the CPU and schedule the next one. Processes that do not meet their deadlines are not allowed to run. The time spent in the CPU of the failed process should be included in the calculation of the average turnaround time and waiting time for the scheduler.

## Windows Hybrid Scheduler (WHS)

Process priorities in HWS are dynamic. HWS keeps track of what processes are doing and adjusts their priorities periodically. Processes that have been denied the use of the CPU for a long time interval are boosted by dynamically increasing their priority and those running for a long time are penalized by decreasing their priority.

There are basically two bands of priorities, i.e., the high band has priorities between 50 to 99, and the low band with priorities from 0 to 49. Assume that all user processes are in the low band and kernel-type processes are in the high band. Processes belonging to one band cannot move to the other band. If a process whose priority is 45 is boosted by 10, it is not allowed to move to the higher band (50-99). The priority of that process will be maxed at 49. Similarly, a process in the high band cannot be demoted to the lower band, i.e., at worst, it will get the lowest priority in the high band of 50.

For both user and kernel type processes, the system boosts the dynamic priority of a process to enhance its responsiveness as follows:

When a process is scheduled, it is given a quantum of time in which to run. The quantum for processes in this exercise is in clock ticks and you will have to prompt the user for it. In this exercise, time quantum values may be any number.

**Promotion**

A process is promoted when (a) it does I/O and (b) its aging timer expires. When a process does I/O, it is preempted and sent to a wait queue. Its priority is then boosted based on the number of clock ticks it takes to do the I/O, and is subsequently sent to the appropriate priority queue after the I/O is done. For example, a process (priority = 38) does I/O at clock tick# 24 for 3 clock ticks will be put at the end of the queue with priority = 41 at clock tick# 27. Multiple processes doing I/O close together will not have to spend time in the wait queue for more than the I/O time, i.e., other processes doing I/O do not affect each other waiting in an I/O queue. They just wait for as long as their I/O burst lasts, boost their priority and join their appropriate ready queues.

If a process does an I/O, it is done in the second to last clock tick of the time quantum, i.e., I/O is initiated at clock tick# 7 if the time quantum for that process is 8. E.g., a process with a time quantum of 10 and a burst of 6 in does not do I/O because it did not reach its 9$^{th}$ clock tick. Not every process will do I/O, i.e., if you encounter a line in the input file that has zero I/O, it is considered CPU-bound.

Every process that is demoted automatically reset its age timer. E.g., if a process gets demoted at clock tick# 38, its aging time will expire at clock tick# 138 and it will be promoted. Its new priority will be its current priority plus 10. Every process must watch its timer and age once its timer expires.

All scheduling is done strictly by priority. The scheduler chooses the highest priority process which is ready to run. Processes are put into their respective run queues based on their priorities. Hint: use a red-black tree to represent all active priority run queues.

**Demotion**

A process is demoted only when its time quantum expires. A process's priority cannot be decreased pass its initial base priority, i.e., if a process starts out with a priority of 25, it cannot be demoted to a value < 25, while at the same time, it cannot be increased to a value > 49. In essence, any process, regardless of whether it is a user or kernel process, adheres to this rule.

When a process encounters a clock interrupt (time-quantum expired), its dynamic priority is decremented by the amount of clock ticks it spent in the CPU prior to the interrupt and is placed in its corresponding priority queue. A process's priority cannot go below its original base priority. If a priority change occurs, then the scheduler locates the highest priority process which is ready to run. Otherwise, the process is placed at the end of the run queue for its priority allowing processes of equal priority to be "round robin" scheduled.

## Statistics For All Algorithms:

0. Process information for Gannt chart construction, e.g., start/end time spent in CPU, etc.
1. Average Waiting Time (AWT)
2. Average Turnaround Time (ATT)
3. Total number of processes scheduled (NP)

Note: The use of `ifdef DEBUG` to turn debugging prints off for stress tests is required. Statistics for AWT, ATT, and NP must be printed at the end of each simulation for all cases.

## Process Characteristics as Input Parameters

The following is an example of the information that will be provided to you in your project demonstration phase:

| P_ID | Burst | Arrival | Priority | Deadline | I/O |
|---|---|---|---|---|---|
| 1 | 8 | 0 | 23 | 10 | 3 |
| 2 | 4 | 2 | 17 | 8 | 0 |

I will be providing you with a file that has the information described above during the demo (in that order). The values in a row are tab delimited. Obviously, not all columns are appropriate for every scheduling algorithm. When sanitizing the input do not eliminate rows automatically just because it has a negative value which is not affecting the said scheduler, i.e., if you see a row with a negative value for deadline, eliminate the row only for the Real-Time scheduler and not for the other schedulers. That row is valid because the other schedulers do not make use of the deadline.

## Project Requirements

Your program should be menu-driven, i.e., the user should have the option of selecting any of the scheduling algorithms to run first. It is also advisable to allow users to enter interactively input values for process ID, burst times, priority, total number of processes, real-time deadline (if appropriate) and arrival times of each process.

If you elect to provide a GUI for the simulation, you will be eligible for extra credit. The GUI interface should also be able to handle output in the form of a Gannt chart.

What do I look for in the output?

1. Display the Gantt chart for each scheduling algorithm.
2. A chronological set of statistics - in the absence of a Gannt chart - displaying the entire simulation run of any particular algorithm. It is best to output all process characteristics in one clock tick intervals.
3. Statistics for the average waiting and average turnaround time. This applies to every scheduling algorithm.

## Project Submission

Project is only considered completely submitted, i.e., to get a grade, if **ALL** items mentioned below are accomplished.

1. W drive:
   A directory consisting of the source code, makefile, and README files tar'ed or zipped and put under the cs452/lab3 directory on the W drive. Name your directory with both your email names, e.g., wagnerpj_tanjs_lab1.tar.

2. Email as attachment (tar'ed or zipped project) and percentage workloads.

3. Technical paper focusing on your ***analysis***. Failure to follow prescribed IEEE paper format will result in points being deducted.

4. Print, fill, and submit the **lab cover page** and **honor code** shown below.

5. Make an appointment to demo your project.

# CS 452 – Operating Systems Lab Assignment Cover Page

Name(s) _____ Lab Assignment No.: 2

1. Include a copy of your source code, a design document (if required), and a sample run
2. Fill in the table of contents including the names of routines and the corresponding pages.
3. Indicate the status of your program by checking one of the boxes.
4. Submit the assignment at the beginning of the class on the due date.

**Program Status:** (check one box)
□ Program runs with user-defined test cases. □ Program runs with some errors.
□ Program compiles and runs with no output. □ Program does not compile.

| **Workload:** | Written By | | | |
|---|---|---|---|---|
| Design document | | | | |
| Paper | | | | |
| Test cases | | | | |
| Program modules (classes) | Module name | Written by | Module name | Written by |
| List only major modules | | | | |
| | | | | |
| that are typically more | | | | |
| than 50 lines of code | | | | |
| | | | | |

| **Grading:** | Points | Score |
|---|---|---|
| Stress Tests | 40 | |
| Correctness | 40 | |
| Technical Paper | 15 | |
| Documentation | 5 | |
| Late Penalty (15% off per day) | | |
| Total | 100 | |

**Honor Code**

**Department of Computer Science**

**University of Wisconsin – Eau Claire**

As members of the University of Wisconsin – Eau Claire community and the computer science discipline, we commit ourselves to act honestly, responsibly, and above all, with honor and integrity in all areas of campus life. We are accountable for all that we say, read, write, and do. We are responsible for the academic integrity of our work. We pledge that we will not misrepresent our work nor give or receive illicit aid. We commit ourselves to behave in a manner which demonstrates concern for the personal dignity, rights and freedoms of all members of the community and those that depend on the expertise we possess.

For all course work in the Department of Computer Science, students will write and sign (if printed) the following: "**I have abided by the Department of Computer Science Honor Code in this work**."

I accept responsibility to maintain the Honor Code at all times.

Name _____

Signature _____

Date _____