**Operating Systems Lab: CS39002**

**Assignment 4:** Implementation of multilevel feedback queue scheduling (MLFQS) in PintOS

**Group 31: Design Document**

**Group Details**

**Name:** Animesh Jain **Roll No:** 18CS10004 **Email ID:** animeshjain99@iitkgp.ac.in

**Name:** Abhinav Bohra **Roll No:** 18CS30049 **Email ID:** abhinavbohra@iitkgp.ac.in

**---- PRELIMINARIES ----**

**ADVANCED SCHEDULER**

**==================**

**---- DATA STRUCTURES ----**

>> C1: Copy here the declaration of each new or changed `struct' or

We added the following variables in struct thread

**1.) int64\_t wakeup\_at; /\* var to store wake up time of thread \*/**

**2.) int nice; /\* var to store nice value \*/**

**3.) int recent\_cpu; /\* var to keep track of recent CPU usage. \*/**

1. Addition of variable int load\_avg in threads.h to hold load average in fixed point notation
2. Addition of variable int nice in threads.h to store nice value
3. Addition of variable int recent\_cpu to store recent cpu in fixed point notation
4. Addition of function update\_priority(struct thread \*) to update priority for a given thread in
5. mlfqs
6. Addition of comparator function cmp\_priority(list\_elem\*,list\_elem\*) to return true if thread belonging to 1st parameter has higher priority than second
7. Addition of following functions in threads/fixed\_point.h:
   1. fxpt\_t int\_to\_fxpt(int n): Convert int to fxpt\_t
   2. int fxpt\_to\_int\_trunc(fxpt\_t x): Convert fixed point value to int and truncate the decimal part
   3. int fxpt\_to\_int\_round(fxpt\_t x): Convert fixed point value to int followed by rounding off
   4. fxpt\_t add\_fxpt(fxpt\_t x,fxpt\_t y): Add 2 fixed point values
   5. fxpt\_t sub\_fxpt(fxpt\_t x,fxpt\_t y): Subtract one fixed point value from another
   6. fxpt\_t add\_int(fxpt\_t x,int n): Add an int to a fixed point value
   7. fxpt\_t sub\_int(fxpt\_t x,int n): Subtract an int from a fixed point value
   8. fxpt\_t mul\_fxpt(fxpt\_t x,fxpt\_t y): Multiply 2 fixed point values
   9. fxpt\_t mul\_int(fxpt\_t x,int n): Multiply a fixed point value by an integer
   10. fxpt\_t div\_fxpt(fxpt\_t x,fxpt\_t y): Divide one fixed point value by another
   11. fxpt\_t div\_int(fxpt\_t x,int n): Divide a fixed point value by an int

**---- ALGORITHMS ----**

>> C2: Suppose threads A, B, and C have nice values 0, 1, and 2. Each

>> has a recent\_cpu value of 0. Fill in the table below showing the

>> scheduling decision and the priority and recent\_cpu values for each

>> thread after each given number of timer ticks:

* Modified thread\_tick() to calculate load\_avg and recent cpu. Priority of each thread is

also updated

* Modified thread\_create() to pre-empt execution of current thread if new thread has

higher priority

* Modified thread\_yield() to sort ready\_list according to priority
* Modified thread\_get\_recent\_cpu() to return recent\_cpu of currently running thread
* Modified thread\_get\_nice() to return nice value of currently running thread
* Modified thread\_get\_load\_avg() to return load\_avg of system
* Modified set\_nice(int) to update the nice value of current thread. The function calls

update\_priority( ) in current thread and subsequently calls thread\_yield( )

* Modified thread\_set\_priority() to modify priority of currently running thread. It

subsequently calls thread\_yield()

timer recent\_cpu priority thread

ticks A B C A B C to run

----- -- -- -- -- -- -- ------

0

4

8

12

16

20

24

28

32

36

>> C3: Did any ambiguities in the scheduler specification make values

>> in the table uncertain? If so, what rule did you use to resolve

>> them? Does this match the behavior of your scheduler?

>> C4: How is the way you divided the cost of scheduling between code

>> inside and outside interrupt context likely to affect performance?

**---- RATIONALE ----**

>> C5: Briefly critique your design, pointing out advantages and

>> disadvantages in your design choices. If you were to have extra

>> time to work on this part of the project, how might you choose to

>> refine or improve your design?

>> C6: The assignment explains arithmetic for fixed-point math in

>> detail, but it leaves it open to you to implement it. Why did you

>> decide to implement it the way you did? If you created an

>> abstraction layer for fixed-point math, that is, an abstract data

>> type and/or a set of functions or macros to manipulate fixed-point

>> numbers, why did you do so? If not, why not?

**We just did it directly, because it was the most straightforward way**

**to do it. Implementing a library might have made the code more readable**

**but I only had to use fixed-point arithmetic in like three functions**

**so it didn't seem that worth it. By the time the thought even came**

**to me to do it the other way, I had already written those functions.**