```
+---------------------------------+
|             CS 140              |
|       PROJECT 2: MLFQS          |
|      DESIGN DOCUMENT            |
+---------------------------------+
```

---- GROUP 9 ----

Aditya Singh <aditya390402@gmail.com>
Itish Agarwal <itishagarwal2000@gmail.com>

---- PRELIMINARIES ----

>> References

https://web.stanford.edu/class/cs140/projects/pintos/pintos.html
https://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html
https://web.stanford.edu/class/cs140/projects/pintos/pintos_7.html
https://practice.geeksforgeeks.org/problems/busy-waiting
https://www.ccs.neu.edu/home/skotthe/classes/cs5600/fall/2015/notes/pintos-project1.pdf
http://www.pvpsiddhartha.ac.in/dep_it/lecture%20notes/OS/unit3.pdf

ADVANCED SCHEDULER

======================

---- DATA STRUCTURES ----

>> C1: Copy here the declaration of each new or changed `struct' or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.

Here are the declaration of each new or changed ‘struct’ or ‘struct’ member, global or static variable, `typedef', or enumeration.

time_wkup : Time for a thread to wake up

nice : Nice value of a thread

thread_recent_cpu : CPU time each process has received “recently”

mng_thread_func : Function that unblocks all threads which have to be woken up. It is called when the managerial thread is running

thread_sleep : Function that puts the current thread to sleep

tick_cmp and priority_cmp : Comparator functions that compare no. of ticks and priority respectively.

set_next_wakeup : Function that wakes the next sleeping thread if its wakeup time is same as current running thread

earliest_wkup : Earliest wake up time among all sleeping threads

thread_sleep_list : List of sleeping threads

mng_thread : Managerial thread responsible to wake up sleeping threads

load_avg : Used for tracking load average

---- 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:

timer	recent_cpu			priority			thread
ticks	A	B	C	A	B	C	to run
0	0	0	0	63	61	59	A
4	4	0	0	62	61	59	A
8	8	0	0	61	61	59	B
12	8	4	0	61	60	59	A
16	12	4	0	60	60	59	B
20	12	8	0	60	59	59	A
24	16	8	0	59	59	59	C
28	16	8	4	59	59	58	B
32	16	12	4	59	58	58	A
36	20	12	4	58	58	58	C

>> 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?

We did run into some ambiguities in the scheduler specification. There were many cases where 2 or more threads had the same priority, hence, the ambiguity to choose which thread to run arose. To combat this issue, we used the **round robin scheduling algorithm** rule in determining which thread to run first.

Yes, this matches the behaviour of our scheduler. Additionally, we have also taken into account the time taken to compute the recent_cpu, load_avg and updating the priorities. We are providing the cpu for less than **4 time ticks** to a thread (keeping the time to compute the above aside).

>> C4: How is the way you divided the cost of scheduling between code inside and outside interrupt context likely to affect performance?

As we are calculating the thread_recent_cpu and the load_avg every second and are updating the priority of each thread every 4 timer ticks, it is likely that performance will be relatively low if the number of threads is large. (Note : This is all happening in the interrupt context)

---- 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?

Advantages :-

1. Easy implementation since we are using list data structures

2. Contains only one ready queue instead of 64 , which cuts down the time spent in removing and adding threads

3. By turning interrupts on/off, we can ensure simple synchronizations

Disadvantages :-

1. Performance can be affected when turning interrupts off

2. Inserting in a sorted list can take O(n) worst case time complexity. Hence, updating the list can take up a lot of time.

We can improve our design by :-

1. Using more efficient data structures instead of lists to store the sleeping threads (heap or hash table)

2. To prevent multiple threads from accessing the same variables, we can use locks instead of interrupts

>> 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?

Yes, we have created a set of functions to manipulate fixed-point numbers because :-

1. Helps in distinguishing between fixed-point parameters and integers

2. Improves readability

3. Reduces complexity of arithmetic expressions