```
+--------------------+
|        CS 140      |
| PROJECT 1: THREADS |
| ADVANCED SCHEDULER |
|   DESIGN DOCUMENT  |
+--------------------+
```

----- GROUP 34 -----

Gajula Sai Chaitanya 18CS30018 <chaitanya6575@gmail.com>
Gunda Maneesh Kumar 18CS10020 <manishkumargunda@gmail.com>

---- PRELIMINARIES ----
We have closely followed these links to understand what must be done,
(i) https://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html
(ii) https://web.stanford.edu/class/cs140/projects/pintos/pintos_7.html

## 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.

(i) In threads/thread.h
```
    struct thread
       ...
       int nice;                          /* nice value of a thread */
       int recent_cpu;                    /* recent cpu usage */
```

These two fields represent 'nice value' which determines how "nice" the thread should be to other threads and 'recent_cpu' measures the amount of CPU time a thread has received "recently". These values are useful to update the priority of each thread when thread_tick() is called by timer_interrupt().

(ii) In threads/thread.c
```
     static int load_avg;
```

'load_avg' estimates the average number of threads ready to run over the past minute.

(iii)In threads/thread.h
```
     /* Nice values */
     #define NICE_MAX 20
     #define NICE_DEFAULT 0
     #define NICE_MIN -20
```

A 'nice' of zero does not affect thread priority. A positive 'nice', to the maximum of 20, decreases the priority of a thread and causes it to give up some CPU time it would otherwise receive. On the other hand, a negative 'nice', to the minimum of -20, tends to take away CPU time from other threads.

(iv)In threads/thread.c
```
    static struct thread* wakeup_thread;  /*Thread which is used for waking up sleeping threads*/
```

    This thread wakes up each thread according to their wakeup times. This thread blocks itself before waking up the sleeping threads(wait_elem_list).

---- 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 ticks | recent_cpu A | recent_cpu B | recent_cpu C | priority A | priority B | priority C | thread 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 (Round Robin for threads with same priority) |
| 12 | 8 | 4 | 0 | 61 | 60 | 59 | A |
| 16 | 12 | 4 | 0 | 60 | 60 | 59 | B (Round Robin for threads with same priority) |
| 20 | 12 | 8 | 0 | 60 | 59 | 59 | A |
| 24 | 16 | 8 | 0 | 59 | 59 | 59 | C (Round Robin for threads with same priority) |
| 28 | 16 | 8 | 4 | 59 | 59 | 58 | B (Round Robin for threads with same priority) |
| 32 | 16 | 12 | 4 | 59 | 58 | 58 | A |
| 36 | 20 | 12 | 4 | 58 | 58 | 58 | C (Round Robin for threads with same priority) |

>> 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?

The ambiguity lies in the recent_cpu calculation. Because we did not consider the time taken to calculate the updated 'priority', 'load_avg' and 'recent_cpu' values and as well re-sorting the ready_list based on new priorities during each TIME_SLICE of 4 ticks. Thus in each TIME_SLICE, the effective real ticks that is added to 'recent_cpu'(recent_cpu is incremented by 1 for every tick, implemented in timer_interrupt() of devices/timer.c) is not really 4. As we can't exactly know how much does this additional calculations require, we incremented the 'recent_cpu' by 4 for current running thread for each TIME_SLICE while filling up the above table.

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

While interrupts are on, If the CPU spends most of its time updating the priority, recent_cpu and load_avg values then the effective time slice that a thread receives would be less than the predefined timeslice of 4 ticks. So if the code for updating these values is inside the interrupt context (i.e.,when interrupts are on) then performance is usually lower.

---- 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:
(i) We have implemented the mlfqs scheduling using only one ready queue instead of using 64 queues for 64 priorities, as it would reduce the space complexity but still providing the same functionality. We have used the 'priority' field of 'struct thread' to achieve this.
(ii) We also made sure that Round-Robin scheduling occurs if there are multiple threads in the highest priority queue, using an appropriate thread comparator based on priority values(thread_insert_desc_end()).

Disadvantages:
(i) As we used one ready queue for mlfqs, inserting a thread into ready queue requires us to traverse through the entire queue to look for a correct position for that thread.

>> 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?

For fixed-point math, we have followed exactly what is mentioned in B.6 section of the the official pintos documentation (fixed-point). We have created a 'fixed-point.h' header file, where we defined all of those helper functions for fixed-point arithmetic as macros. We used macros instead of inline functions as macros are quiet simple and are usually faster than the inline functions.