+-----------------------------------+
| OS LAB 4 |
| PROJECT 1: THREADS |
| DESIGN DOCUMENT |
+-----------------------------------+

---- GROUP ----

Rahul Aditya (18C30032)
Abhishek Srivastava (18CS10068)

---- PRELIMINARIES ----

- Added a line in **devices/shutdown.c** to avoid shutdown problem caused due to new release of qemu
- Added a header file **threads/fixed_point_calc.h** with macros defined for various fixed point calculations used in advanced scheduling.

## 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.

**A.**
**threads/thread.h**

```
struct thread
  int nice;              /* Thread nice value */
  int recent_cpu;        /* Thread recent CPU */
  Int wakeTime           /* Thread wakeup time */
```

**threads/thread.c**

```
#define NICE_MIN -20        /* Minimum possible nice value */
#define NICE_DEFAULT 0      /* Default nice value */
#define NICE_MAX 20         /* Maximum possible nice value */

fixed_t  load_avg;          /* load_avg for BSD scheduling. Fixed-point number */
```

**threads/fixed_point_calc.h**

```
typedef int fixed_t;      /* For fixed point calculations */
```

---- 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:

**A.**

| | recent_cpu | | | priority | | | |
|---|---|---|---|---|---|---|---|
| timer ticks | A | B | C | A | B | C | running |
| 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 | B |
| 28 | 16 | 12 | 0 | 59 | 58 | 59 | C |
| 32 | 16 | 12 | 4 | 59 | 58 | 58 | A |
| 36 | 20 | 12 | 4 | 58 | 58 | 58 | B |

**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?

**A:** The scheduler specification does not define what is to be done if the priority of 2 or more threads is the same. Different handling protocols will result in different outcomes. We have used Round Robin to schedule threads with the same priority.

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

**A:** The recent_cpu of each thread is recalculated every second, and the priorities of all threads are updated every 4 timer ticks. These involve various calculations including fixed point calculation. They do take a finite amount of time and slightly deteriorate the performance of our scheduler.

---- 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?

**A:** Our design is fairly straightforward and easy to understand. The advantage of our design is that we used a single ready queue to implement the multiple levels of the feedback queue. This makes it easy to reorder that ready list in case the priority of threads change. If instead we used multiple queues to implement the scheduler, the OS would have to do some work in migrating the thread element from one queue to another.

The disadvantage is that we have to sort the ready list every 4 timer ticks in order to keep the multilevel feature working. Also, when inserting a preempted process in the ready queue, we have to ensure that the insertion is ordered. These take some extra time, and deteriorate system performance.

**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?

**A:** Fixed points calculations were implemented because measuring recent_cpu and load_avg are floating point operations, which are not available in pintos. Hence, to perform precise calculations to ensure fair scheduling we implemented it.

We created a new header file with macros defined for different types of fixed point calculations. This makes the code in threads/threads.c cleaner and more readable.