Jaswinder Kaur

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.

```
Int recent_cpu;  // A thread’s recent cpu values
int nice;            //A thread’s nice value
```

---- 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	A
12	12	0	0	60	61	59	B
16	12	4	0	60	60	59	B
20	12	8	0	60	59	59	A
24	16	8	0	59	59	59	A
28	20	8	0	58	59	59	C
32	20	8	4	58	59	58	B
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?

Yes, there is the ambiguity of two threads possibly having the same priority. To resolve this issue, we used two rules: One that if the running thread and a ready thread have the same priority, the running thread continues running. The second rule is that if there are multiple threads that have the same priority waiting in the ready queue, the thread that was placed in first gets executed first. These rules match the behavior of our scheduler.

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

A majority of the computations will be done within the interrupt handler with the exception of the computation of the nice value being calculated outside of the interrupt handler. However, while computing the nice value, interrupts must be turned off. Because of the way the cost of scheduling was divided up, the performance will decrease.

---- RATIONALE ----

>> C5: 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 decided to implement it this way because our implementation ensures that threads with the same need for the resources gets the similar access to the cpu time. Also, we have decided to use fixed-point in order to represent the recent_cpu values. As we know, recent_cpu is a real number. So it is better to use fixed-point because pintos disables real numbers in system.
We think using macros to manipulate fixed - point numbers would determine that the processor can resolve the macro into integers in a simple way.