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

ADVANCED SCHEDULER
====================

---- DATA STRUCTURES ----

>> C1: Copy here the declaration of each new or changed `struct' or
We added the following variables in struct thread

In thread.h
int64_t sleep_endtick;
- The tick after which the thread should awake (if the thread is in sleep)

int recent_cpu;

- Reflects recent CPU acquisition of this thread

int nice;

- Nice-ness of this thread, ability to release CPU to other threads

A global variable:

int load_avg;

- Moving average of the number of threads ready to run

- Addition of variable int load_avg in threads.h to hold load average in fixed point notation
- Addition of variable int nice in threads.h to store nice value
- Addition of variable int recent_cpu to store recent cpu in fixed point notation
- Addition of function update_priority(struct thread *) to update priority for a given thread in
- mlfqs
- Addition of comparator function cmp_priority(list_elem*,list_elem*) to return true if thread belonging to 1st parameter has higher priority than second
- Addition of following functions in threads/thread.c for fixed-point calculations:

- int convert_to_fxpt(int n): Convert int to fixed-point
- int convert_to_intfloor(int x): Convert fixed point value to int and truncate the decimal part
- int convert_to_intround(int x): Convert fixed point value to int followed by rounding off
- int add_fx(int x,int y): Add 2 fixed point values
- int sub_fx(int x,int y): Subtract one fixed point value from another
- int add_in(int x,int n): Add an int to a fixed point value
- int sub_in(int x,int n): Subtract an int from a fixed point value
- int mul_fx(int x,int y): Multiply 2 fixed point values
- int mul_in(int x,int n): Multiply a fixed point value by an integer
- int div_fx(int x,int y): Divide one fixed point value by another
- int div_in(int x,int n): Divide a fixed point value by an int

---- ALGORITHMS ----

- 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()**

>> 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	R(A)	R(B)	R(C)	P(A)	P(B)	P(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
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?

Yes, because if the threads have the same priority, it's unconspicuous to choose which threads to execute.
So, we choose two strategies to solve them:
1. If one of the threads in the ready list have the highest priority, it will execute first obviously.
2. If the threads in ready list have the same priority, then choose the thread has the most least run-time recently.

Yes, these rules match with the behaviour 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?

As per the specification of this project, all calculations were done in interrupt context i.e.
1. Re-calculating load average and recent_cpu every second.
2. Updating priority of each thread and sorting the ready list every 4 timer ticks.

Recalculating priority every 4 ticks and sorting the ready list will become very costly if we have large number of threads.

If the CPU spends too much time on calculations for recent_cpu, load_avg and priority, then it takes away most of the time that a thread before enforced preemption. Then this thread can not get enough running time as expected and it will run longer. This will cause itself got blamed for occupying more CPU time, and raise its load_avg, recent_cpu, and therefore lower its priority. This may disturb the scheduling decisions making. Thus, if the cost of scheduling inside the interrupt context goes up, it will lower 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?

Advantages:
1. Simple to implement and comprehend
2. Efficient use of lists
3. Simple synchronizations by turning interrupts on/off

Disadvantages:

1. General list operations require O(n) and sorting a list requires O(nlogn). Therefore efficient data structures should have been employed (e.g hash tables, binary tree)
2. Turning the interrupts off is brute force and affects performance.

Potential improvements:

1. Use more efficient data structures
2. Use synchronization primitives other than turning interrupts on/off.

>> 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 we 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 us to do it the other way, we had already written those functions.