CS39002

OS LAB ASGN-4
DESIGN DOCUMENT

GROUP :44
Proteet Paul (18CS10065)
Suryansh Kumar (18CS30043)

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.

- 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/fixed_point.h:
    - fxpt_t int_to_fxpt(int n): Convert int to fxpt_t
    - int fxpt_to_int_trunc(fxpt_t x): Convert fixed point value to int and truncate the decimal part
    - int fxpt_to_int_round(fxpt_t x): Convert fixed point value to int followed by rounding off
    - fxpt_t add_fxpt(fxpt_t x,fxpt_t y): Add 2 fixed point values
    - fxpt_t sub_fxpt(fxpt_t x,fxpt_t y): Subtract one fixed point value from another
    - fxpt_t add_int(fxpt_t x,int n): Add an int to a fixed point value
    - fxpt_t sub_int(fxpt_t x,int n): Subtract an int from a fixed point value
    - fxpt_t mul_fxpt(fxpt_t x,fxpt_t y): Multiply 2 fixed point values
    - fxpt_t mul_int(fxpt_t x,int n): Multiply a fixed point value by an integer
    - fxpt_t div_fxpt(fxpt_t x,fxpt_t y): Divide one fixed point value by another

- fxpt_t div_int(fxpt_t 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	Recent cpu			Priority			Thread to run
	A	B	C	A	B	C	
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, the scheduler specifications did not specify which thread to run next if all threads have the same priorities. We used Round Robin scheduling algorithm to determine the next running thread in such cases.
The cmp_priority( ) function ensures this as thread A is put before thread B on the list only if thread A's priority is greater than that of thread B. Otherwise it comes after thread B in the sorted list.

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

If the CPU spends too much time on calculations for recent_cpu, load_avg and priority, then it takes away most of the time from a thread before enforced preemption. Then this thread can not get enough running time as expected and it will run longer. This will cause it to get blamed for taking more CPU time, and raise its recent_cpu, and therefore lower its priority. This may disturb the scheduling decision 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:

- Contains only one ready queue instead of 64 ready queues, which cuts down the time spent in removing threads from one queue and adding them to another queue if their priority changes. Simply sorting the ready queue is sufficient

Disadvantages:

- The order of individual threads of same priority in the ready queue depends upon how the sort function is implemented and may not always guarantee the ordering between demoted threads and other threads in the level to which the threads have been demoted. In our case, a stable sorting algorithm ensures that threads with the same priority are scheduled using Round Robin scheduling.
- External interrupts are disabled as soon as we enter the thread_set_nice( ) function. This might affect performance due to the cost involved in computing the new priority. This can

be improved upon by initializing the required variables before disabling interrupts and including very few statements inside the critical section

>> 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 created a set of functions to manipulate fixed-point numbers as it:

- Reduces the complexity of the arithmetic expressions involved in thread.c and improves readability of the code
- Helps to distinguish between integer and fixed-point parameters which in turn minimizes the chances of an error