Alexandria University
Faculty of Engineering

Semester-long Project
CS333: Operating Systems

Pintos Projects

Phase 1: Threads

GROUP

Abanob Maher (02)
abanobmaher1994@gmail.com

Mohamed Ahmed Abdel-Twab (52)
Mohamed.Ahmed.Heeba@gmail.com

Mohamed Samir Shaaban (55)
Samircode95@gmail.com

PRELIMINARIES

If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.

Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.

ALARM CLOCK

DATA STRUCTURES:

A1: 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.

In timer.c

static struct list sleeping_list;
/*list to add the waiting (sleeping) threads*/

In thread.h

int64_t wake_up;
/*time to wake up a sleeping thread*/

ALGORITHMS:

A2: Briefly describe what happens in a call to timer_sleep(), Including the effects of the timer interrupt handler.

In timer_sleep()

disable interrupts
calculatr the wakeup time for current thread
wake_up = timer_ticks () + ticks
insert the current thread in the sleeping list sorted by wakeup time
block the thread until get it's wake up time
enable interrupts.

In timer_interrupt()

iterate on the sleeping list while it's not empty
get the 1st element (sleeping thread)
if it's time to wake up → wake it up
if not → break (as all elements are sorted by their wakeup time).

A3: What steps are taken to minimize the amount of time spent in The timer interrupt handler?

Insert orderd in sleeping list → save the time in each interrupt time happen for each tick, as we not iterate on whole sleeping list, we just get the top elements which we got their wakeup time, once we reach to 1st element which it's wakeup time is less than ticks, we break the iteration.

SYNCHRONIZATION:

A4: How are race conditions avoided when multiple threads call timer_sleep () simultaneously?

By disableing interrupts brfore getting the current thread
and enabling interrupts after insert the current thread in the sleeping list
Note: we can use lock and unlock instead of disable and enable interrupts.

A5: How are race conditions avoided when a timer interrupt occurs during a call to timer_sleep ()?

By disableing interrupts brfore getting the current thread and enabling interrupts after insert the current thread in the sleeping list in timer_sleep()
Note: we can't use lock and unlock instead of disable and enable interrupts.

RATIONALE:

A6: Why did you choose this design? In what ways is it superior to another design you considered?

Consider a design that insert unorderd and when calling timer_interrupt() loop on the whole sleeping list and get the threads that their wakeup time come. So in calling timer_sleep() it costs O(1) , but in calling

timer_interrupt() it costs O(n). assume that number in calling timer_sleep() is S and number of calling timer_interrupt() is I = #ticks. So I must be greater than or equal ti S, then total cost of considered design = O(1)*S+O(n)*I
In our design total cost = cost of insert orderd(O(lgn) or O(n))*S+O(1)*I this means that → our design total cost <= considered design total cost.

PRIORITY SCHDULING

DATA STRUCTURES:

B1: 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.

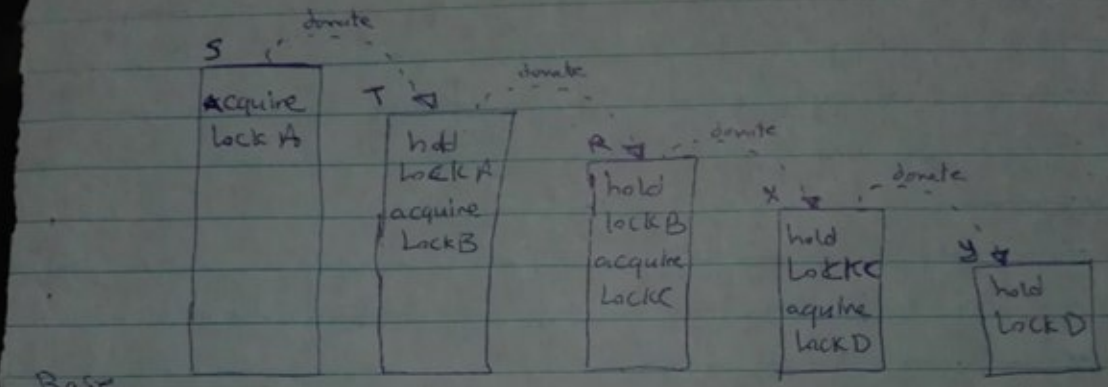
In thread.h

```
int base_priority;
/*the priority that the thread created with*/
struct lock *mylock;
/*a held lock by this thread*/
struct list lock_list;
/*all locks this thread hold*/
```

In synch.h

```
int max_donate;
/*maximum thread priority acquired the lock*/
struct list_elem lock_elem;
/* List element point to this lock */
```

B2: Explain the data structure used to track priority donation.
Use ASCII art to diagram a nested donation. (Alternately, submit a .png file.)



Base priority	50	40	30	20	10
priority	50	50	50	50	50

thread y run → release lock D → blocked after update priority = 10

thread x run → release lock C → blocked after ~ ~ = 20

thread R run → release lockB → ~ ~ ~ ~ = 30

~ T run → release lockA → ~ ~ ~ ~ = 40

thread S run → the first tick → update periorities and rescheduling and choose the max priority thread to run. and so on.

ALGORITHMS:

B3: How do you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable wakes up first?

By getting the thread which have the max priority in the list of waiters every time of calling cond_signal ,sema_up and look_release.

B4: Describe the sequence of events when a call to lock_acquire () Causes a priority donation. How is nested donation handled?

1- check if this lock is held by other thread
2- setting thread lock
/*
go deeper until reach to a thread that not acquire a held lock or not acquire a lock, which thread priority is greater than the holder priority
*/
3- while (above comment)
4- set the holder priority with the current priority
5- update the holder priority with max priority from the held locks in the lock_list the holder have
6- get next step in iteration
7- block the current thread OR enter critical section
/*NOW the blocked thread had been freed OR exit critical section*/
8- disable interrupts
9- current thread no longer point to this lock
10- setting the priority of the lock by the current thread donate priority
11- insert this lock to the lock_list of the current thread
12- current thread will be the holder of this lock
13- enable interrupt.

Deep nested donation is handled in 3-6

B5: Describe the sequence of events when lock_release() is called on a lock that a higher-priority thread is waiting for.

1- remove the lock from the lock_list in the thread hold it
2- update the current thread priority with max priority from the held locks
3- indecate that this lock no longer is held
4- unblock and check yeild

SYNCHRONIZATION:

B6: Describe a potential race in thread_set_priority () and explain how your implementation avoids it. Can you use a lock to avoid this race?

that happen when call thread_set_priority() to set new prirority and at the same time the interrput handler want to update the priority in this case will happen race condition so we use disable and enable interrupt.

in thread_set_priority() function we update the base_priority in each time we call the function but update the donation priority if the new priority is greater than the donation priority or there is no donation that happen

we think no , we can't use lock.

RATIONALE:

B7: Why did you choose this design? In what ways is it superior to another design you considered?

There is two design.
1st design : keep waiters list orderd.
2nd design : keep waiters list unorderd.
In case of semaphoores and condition variables neerly two design have the same cost, so the indecator is the case of locks with donations as we get the max priority in each time we update the priority, and we insert the element only one time, so to keep it sorted is better in time, but we kept it unsorted because of complexity.

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.

In thread.h

```
int nice;
```

```
/*the thread's current nice value determines how "nice" the
thread should be to other thread*/

int recent_cpu;
/*the thread's current recent_cpu value, to measure how
 much CPU time each process has received "recently."*/
```

In thread.c

```
int load_avg;
/*gloal variable , known as the system load average, estimates
 the average number of threads ready to run over the past minute*/
```

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:

TIME_SLICE = 4;
in each tick we incrment the current thread's recent_cpu by 1
in each four we update (all) threads 's priority
thread's priority = PRI_MAX - (thread's recent_cpu) - (thread's nice*2)
then the scheduling select the highest thread's priority to run

timer ticks	recent_cpu A	B	C	priority A	B	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 , if there is two or more threads have equal priority which thread should be selected by the schedule to run next :

the running thread has the highest priority and the ready threads have priority less than the priority of the running thread if at any time the schedule want to select next thread to run and there are more than ready thread
have the same priority which is equal the higest priority the schedule will choise the thread which least runing , which i placed it in the first of the ready list.

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

-inside the interrupt every 4 ticks we update the priority for all threads and every second we update load average for the system and update all recent_cpu for all threads and this if very expensive for the system have many threads.

-outside the interrupt only one operation which we set new nice to the current thread and update the priority to ther current thread.

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:
-the code is simplicity which we used in the design and the complexity.

Disadvantages:
-we didn't use complex data structures in the project such as tree or skiplist to insert in it in O(log n) and remove from it in (log n) which not need to

sort as we did in the orderd list , we took O(n log n) to sort the list in each time we need to add thread in it.
-we used interrupt to make sure the is no two thread enter the critical region at the same time and do problems

To refine our design:
-we will use the complex data structure in the project such as tree or skiplist to improve the insertion and deletion time
-we will use lock and unlock(use mutex , sema or condition varriable) to make
sure there is no two thread enter the critical section in the same time instead of using disable interrupt and enable it.

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 create a header file called fixed-point.h and define a set of macros to manipulate fixed-point numbers , which we use this macros that define in the header file to convert betweent integers and fixed-point and the arithmetic equation.
-we use this macros which we define in the header file in the thread.c class to make the code more readable and more Organizer and to be clear for understanding.

SURVEY QUESTIONS

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

Q1: In your opinion, was this assignment, or any one of the three problems in it, too easy or too hard? Did it take too long or too little time?

Q2: Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?

Q3: Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?

Q4: Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects?

Any other comments?