CS342:Operating System

Project 1 : Threads

Submitted By:

Pradeep Bansal

10010143

ASSIGNMENT 1b: **Implement Priority Scheduling and Priority Donation.**

CHANGED FILES

1. ~/pintos/src/threads/thread.h

Added Functions

a. bool thread_change_priority(struct thread * ,struct thread *);

It assigns the donor priority to the running thread and if it is in the ready_list then reinserts it as per
its current priority and return true; else false

2. ~/pintos/src/threads/thread.c

Changed Functions

a. Void thread_unblock (struct thread

*t).

b. Void thread_yield (void).

c. void thread_set_priority (int

new_priority).

d. static void init_thread(struct thread *t,const

char *name,int priority);

Added Functions

a. bool comp_priority(const struct list_elem *,const struct list_elem *,void *);
Returns true if priority of Ist thread's priority is 2nd than second's

3. ~/pintos/src/threads/synch.c

Changed Functions

a. void sema_down (struct semaphore

*sema) .

b. Void sema_up (struct semaphore *sema)

c. Void lock_acquire (struct lock *lock).

d. void lock_release (struct lock *lock).

e. void cond_signal (struct condition *cond, struct lock *lock).

Added Functions

a. bool priority_comp(const struct list_elem *,const struct list_elem *,void *);
   It is used to insert a thread into the semaphore's waiter's list in the descending order of priority.

b. bool prior(const struct list_elem * e1,const struct list_elem *e2,void *aux)

DATA STRUCTURES

/*Added to thread structure*/

int actual_priority; /* Priority before donation. */

bool donated; /*true if the priority is donation received */

struct lock *waiting_lock; /* Lock on which this thread is waiting.*/

struct semaphore * waiting_on_sema; /*lock on which thread is waiting*/

struct list received_priority; /*the list of received priorities */

/*Added structure*/

/*list threads to from which this thread has received the priority*/

struct received_from{

struct thread *donor;

struct lock *donated_for_lock;

int received_priority;

struct list_elem elem;

};

ALGORITHMS

Priority Scheduling:

Whenever a thread come from waiting state to ready state or when it is newly created, it calls thread_unblock() the algorithm first inserts the thread into the ready list in order of, then it checks wheather the threads which called thread_unblock() has higher prioirity than the current executing thread.

If Yes, Then we preempt the current running thread and push it into the ready list again using priority as a parameter, and call schedule(), to schedule this job(which called thread_unblock()).

If No,we proceed.

A check specially for the idle thread is required because we don't insert the idle thread into the ready list apart from once, in which case we just preempt the idle thread and call schedule().

For the case of thread modifying it's own priority (Implementing thread_set_priority())

Since it affects the thread's base priority (actual_priority in the submitted code) we change it, and we see if the base priority has risen above the donated priority(priority in the submitted code)in that case we raise the donated priority to that base priority.
But in case the new priority is less than the thread's current base_priority then we don't change the thread's donated priority.

After the modifications to the priority are done then we check whether the effective priority of the thread is not less than any priority of any process in the ready_list and then we apply the algorithm of priority scheduling as stated above.

Priority Donation:
Since now it is priority scheduler changes in the original source code done are:

```
Void sema_up (struct semaphore *sema)
{
  enum intr_level old_level;
  ASSERT (sema != NULL);
/*sema->value++ is moved above since if the thread that is unblocked has higher priority than
the current thread than the current thread will not be able to up the semaphore and hence
the thread which gets unbloked will not be able to down the semaphore.*/
sema->value++;
  old_level = intr_disable ();
  if (!list_empty (&sema->waiters))
    thread_unblock (list_entry (list_pop_front

    (&sema->waiters), struct thread, elem));

    intr_set_level (old_level);

    }
```

As a thread call lock_aquire it check whether lock is help by some other thread or not, if not it will aquire lock. If lock is acquired by some other thread then it will check the priority of its thread(one who has lock say a), if priority of thread a is more than current threads priority it will siple wait, If not then it will donate its priority to a. And check if a is waiting on some lock or semaphore by checking its waiting list, if it is waiting then to every waiting thread it will donate its priority till there is no wait either on some other lock or semaphore, and insert that in ready list . This way multiple donatin, donate sema, donate one, all are taken care of.

For condvar, everytime max priority thread from waiting list of condition is woken up and sema_up is called.

Synchronization

Interrupts are disabled before the thread is blocked and inserted to the ready_list and are set back to the old level afterwards. Also the interrupts are disabled during the priority change and donation.

RATIONALE

Advantages

1.The lock_acquire() code that checks for the possibility of nested donations is implemented fast enough so that it executed well before the next timer interrupt so no synchronization was needed in this case.