Assignment T02

Data Structure:

Code:

```
struct thread
 {
   struct list_elem sema_elem; /* Used to make list of threads in
semaphore waiters list*/
   struct list holding_locks;          /* list of acquired locks by this
thread*/
   struct lock *seeking;               /* seeking lock for the current
thread*/
   struct semaphore *seeking_sema;   /* seeking semaphore for the current
thread*/
 };

struct lock
 {
   struct list_elem elem;      /* This is used to make a list of locks in
thread struct */
   int priority;               /* Maximum of priorities of the threads
which are seeking the lock */
 };

struct semaphore_elem
 {
   int priority;                       /* This priority is used to sort
conditional variables*/
 };
```

Explanation :

Struct threads{}

Sema_elem is list element for a semaphore waiters list.
We can use the list holding locks to get a held lock which has max priority and that priority will be the next donation when this thread releases a lock.
When we change a thread priority after it receives a donation then,we must change the thread position in all list in which it exists so for that identifying that list seeking_sema is used.
Seeking gives the lock which this thread wanted to acquire but was blocked by that lock.

Struct lock{}

Priority is added in **struct lock{}** to identify the donation benefit which may be received by a thread holding the lock.
elem is the element of lock which is inserted in a list of locks.

Algorithm:

Func added : donate_my_priority(), change_priority_after_lock_release(),
Func modified: thread_set_priority(), sema_down(), sema_up(), lock_acquire(), lock_release(), cond_wait()

For **task 1** we modified the function thread_set_priority which sets thread->priority = new priority. Also preemption is taken care of. I.e if the priority of the current thread is reduced and it’s priority has less value than next thread in ready list, current thread is preempted.

For **task 2** ,when a semaphore block a thread in sema_down() and insert it in its waiters list ,we insert the thread in list in order of priority(thread with higher priority is at head of list), so that when a thread is unblocked in sema_up() it is done in order of priority.

For **task 3**, we did same thing as above, when a thread is inserted in cond->waiters list we insert it in order of priority.

For **task 4** we added new members in struct thread{} and struct lock{}.

Mainly in lock we added priority and in thread we added holding_lock which will be useful in task 5.

For **task 5**,when a thread calls lock_acquire() on a lock, then we call func donate_my_priority(), this func donates the priority to thread holding the lock in recursive manner i.e. the donor donates to donee then donee again donates to another thread which holds the lock donee->seeking and so on. Note: donation only occurs if donor->priority > donee->priority.

For lock_release()
when a thread calls lock release for a lock L,the current threads calls func change_priority_after_lock_release().
In change_priority_after_lock_release() either thread again acquires a donated priority or resets to original priority. If the priority of lock having max priority among all locks in list holding_locks is greater than before_donate_priority of thread then thread gets a new donations otherwise it resets to original priority.
Then sema_up is called which unblocks a thred.
Now the lock L also needs to change its priority .
L->priority = priority of thread with highest priority in waiters list
If waiters list ==NULL then priority of l will be PRI_MIN.

Rationale

We achieved the task of unblocking thread with highest priority by inserting thread in ordered manner in a list. Because otherwise every time for unblocking searching for a thread with highest priority takes O(n) time which is wastage. Preemption is also taken care of in functions like thread_create() and thread_set_priority which was not earlier. Also priority donation is done recursively. Priority of lock is changed after sema_up is called and a thread is unblocked.