Extend Lock::acquire() and Lock::release() in Figure 5.17 to implement priority donation.

Assume:

* runningThread is a global pointer to the calling thread with data type of TCB \*
* TCB has a public member named “priority” with type int (assume public to avoid get/set)
* the necessary extra member variables have been added to Lock
* the waiting list for a Lock object is changed to a priority-ordered queue (already done)
* waiting.remove() returns the TCB on the waiting list with the highest priority

class Lock {

private:

int value = FREE;

SpinLock spinLock;

PriorityQueue waiting; // changed to priority queue

TCB \*lockOwner; // added

int lockOwnerOriginalPriority; // added

public:

void acquire();

void release();

}

Code segments to add:

A:

lockOwner = next;

lockOwnerOriginalPriority = next->priority;

B:

lockOwner = runningThread;

lockOwnerOriginalPriority = runningThread->priority;

C:

// defensive programming

lockOwner = NULL;

lockOwnerOriginalPriority = NOT\_A\_PRIORITY;

D:

runningThread->priority = lockOwnerOriginalPriority;

E:

if ( runningThread->priority > lockOwner->priority ) {

lockOwner->priority = runningThread->priority;

}

Mark the code segment letter in the appropriate space in the code on the left.

Lock::acquire() {

spinLock.acquire();

if (value != FREE) {

//SEGMENT E

if ( runningThread->priority > lockOwner->priority ) {

lockOwner->priority = runningThread->priority;

}

waiting.add( runningThread );

scheduler.suspend( &spinlock );

// scheduler releases spinLock

} else {

//SEGMENT B

lockOwner = runningThread;

lockOwnerOriginalPriority = runningThread->priority;

value = BUSY;

spinLock.release();

}

}

void Lock::release() {

TCB \*next;

spinLock.acquire();

//SEGMENT D

runningThread->priority = lockOwnerOriginalPriority;

if ( waiting.notEmpty() ) {

next = waiting.remove();

//SEGMENT A

lockOwner = next;

lockOwnerOriginalPriority = next->priority;

scheduler.makeReady( next );

} else {

value = FREE;

//SEGMENT C

// defensive programming

lockOwner = NULL;

lockOwnerOriginalPriority = NOT\_A\_PRIORITY;

}

spinLock.release();

}