Sample solution

Quiz 4

(1) Answer:
No. Take IO for example, the performance still gets improved as there is always I/O operations that block the execution of a process

(2) Answer:
Reader–writer locks are similar to mutexes, except that they allow for higher degrees of parallelism. Three states are possible with a reader–writer lock: locked in read mode, locked in write mode, and unlocked.

Compare to mutex, reader–writer locks are also called shared–exclusive locks. When a reader–writer lock is read locked, it is said to be locked in shared mode. When it is write locked, it is said to be locked in exclusive mode. Therefore, reader–writer locks are more efficient than mutex since it allows multiple threads access the critical section when read mode.

(3) Answer:

```
#include <pthread.h>

struct msg {
    struct msg *m_next;
    /* ... more stuff here ... */
};

struct msg *workq;

pthread_cond_t qready = PTHREAD_COND_INITIALIZER;

pthread_mutex_t qlock = PTHREAD_MUTEX_INITIALIZER;

void
process_msg(void)
{
    struct msg *mp;

    for (;;) {
        pthread_mutex_lock(&qlock);
        while (workq == NULL)
            pthread_cond_wait(&qready, &qlock);
        mp = workq;
        workq = mp->m_next;
        pthread_mutex_unlock(&qlock);
        /* now process the message mp */
    }
}

void
enqueue_msg(struct msg *mp)
{
    pthread_mutex_lock(&qlock);
    mp->m_next = workq;
    workq = mp;
    pthread_mutex_unlock(&qlock);
    pthread_cond_signal(&qready);
}
```

The bug is that if we didn't check the condition (workq) in a loop, all the thread will wait for the signal. If there is a new work added to the empty queue and processed by one thread, the other threads will access the same workq which leads to a problem. The solution is put the checking procedure in a while loop.