# 第 7 章作业

7.8

**7.8** The Linux kernel has a policy that a process cannot hold a spinlock while attempting to acquire a semaphore. Explain why this policy is in place.

答：In the Linux kernel, both spinlocks and mutex locks are non-recursive, which means that if a thread has acquired one of these locks, it cannot acquire the same lock a second time without first releasing the lock. Otherwise, the second attempt at acquiring the lock will block.

7.11

**7.11** Discuss the tradeoff between fairness and throughput of operations in the readers–writers problem. Propose a method for solving the readers–writers problem without causing starvation.

答：首先，在读者-写者问题中，任意多个读者可以并发访问共享对象，但一次只有一个写者可以访问共享对象，且读者与写者不能并发访问共享对象。如果使读者的优先级更高，则能够提高吞吐率，但可能导致写者的饥饿；如果使写者的优先级更高，则可能导致读者的饥饿，使吞吐率降低。可以通过让读者和写者在进入临界区前或离开临界区后等待随机时间的方式来避免饥饿。还可以设置一个最长等待时间，如果某个读者或者写者进程等待时间达到最长等待时间，则允许它抢占目前正在访问共享对象的进程。

7.16

**7.16** The C program `stack-ptr.c` (available in the source-code download) contains an implementation of a stack using a linked list. An example of its use is as follows:

```
StackNode *top = NULL;
push(5, &top);
push(10, &top);
push(15, &top);

int value = pop(&top);
value = pop(&top);
value = pop(&top);
```

This program currently has a race condition and is not appropriate for a concurrent environment. Using Pthreads mutex locks (described in Section 7.3.1), fix the race condition.

答：

在官网下载的所有源代码文件中并没有 stack-ptr.c 这个文件，所以在这里我只能靠推测来补充代码。

```
/* stack-ptr.c */
/* * Stack containing race conditions */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

pthread_mutex_t mutex;
pthread_mutex_init(&mutex,NULL);
```

```c
// Linked list node
typedef int value_t;

typedef struct Node
{
    value_t data;
    struct Node *next;
} StackNode;

// Stack function declarations
void push(value_t v, StackNode **top);
value_t pop(StackNode **top);
int is_empty(StackNode *top);

int main(void)
{
    StackNode *top = NULL;
    push(5, &top);
    push(10, &top);
    pop(&top);
    push(15, &top);
    pop(&top);
    pop(&top);
    push(20, &top);
    push(-5, &top);
    pop(&top);
    push(-10, &top);
    pop(&top);
    pop(&top);
    push(-15, &top);
    pop(&top);
    push(-20, &top);

    return 0;
}

// Stack function definitions
void push(value_t v, StackNode **top)
{
    pthread_mutex_lock(&mutex);

    StackNode *new_node = malloc(sizeof(StackNode));
    new_node->data = v;
    new_node->next = *top;
    *top = new_node;

    pthread_mutex_unlock(&mutex);
}

value_t pop(StackNode **top)
{
    pthread_mutex_lock(&mutex);

    if (is_empty(*top)) return (value_t) 0;

    value_t data = (*top)->data;
    StackNode *temp = *top;
```

```
    *top = (*top)->next;
    free(temp);

    pthread_mutex_unlock(&mutex);

    return data;
}

int is_empty(StackNode *top)
{
    pthread_mutex_lock(&mutex);

    if (top == NULL)
    {
        pthread_mutex_unlock(&mutex);
        return 1;
    }
    else
    {
        pthread_mutex_unlock(&mutex);
        return 0;
    }
}
```