Homework 7

余北辰 519030910245

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.

如果一个进程持有自旋锁，那么他就会禁止处理器的抢占。如果该进程在取得自旋锁后进入休眠，由于抢占被禁止，其他进程也就没有办法获得处理器资源。而信号量是一种休眠锁，一个进程执行wait()并发现信号量不为正，就会使自己休眠；而由于处理器的抢占被自旋锁给禁止了，其他进程无法获得cpu资源，也就无法唤醒自旋锁，cpu将不会响应任何操作而是陷入忙等待中。

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.

在读者-写者问题中，吞吐量主要受读者所影响，因为读者之间能够不互斥地访问临界区，从而提高吞吐量。然而，读者优先的处理方法又可能会导致写者一直处于等待状态而饿死。可以在读者进入临界区前增加一个判断条件：如果写者进程等待时间超过了一定值，则不能进入临界区，这样就保证了写者的有限等待。

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:

```c
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.

只需要在pop和push的前后加锁即可。

```c
#include <stdio.h>
#include <pthread.h>
```

```
#include "StackNode.h"

StackNode *top = NULL;
pthread_mutex_t mutex;

void push_with_mutex(int value)
{
    pthread_mutex_lock(&mutex);
    push(value, &top);
    pthread_mutex_unlock(&mutex);
}

int pop_with_mutex()
{
    pthread_mutex_lock(&mutex);
    int value;
    value = push(value, &top);
    pthread_mutex_unlock(&mutex);
    return value;
}

int main()
{
    push_with_mutex(5);
    push_with_mutex(10);
    push_with_mutex(15);

    int value = pop_with_mutex();
    value = pop_with_mutex();
    value = pop_with_mutex();

    return 0;
}

```