操作系统(D)

第 7 次作业

Log Creative

2021 年 6 月 15 日

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: 一个进程拥有旋转锁时不能获取信号量，是因为旋转锁处于忙等待状态的同时，获取信号量也有可能会导致忙等待，该进程所使用的CPU已经处于忙状态时不应该再叠加一次忙状态，可能会显著增加等待时间，甚至宕机。

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.

解. 如果为了保持公平，读者和作者一次只能执行一个，但这会导致吞吐量的显著降低。而如果设置读者优先级高于作者，就可以保持可以多个进程可以同时读取，作者与读者互斥。

无饥饿的解法：可以设置两个信号量

```
semaphore writer = 1;
semaphore reader = 1;
```

作者进程如下：

```
do{
    wait(reader);
    wait(writer);

    /* writing is performed */

    signal(writer);
    signal(reader);
} while (true);
```

读者进程如下：

```
do{
    wait(reader);
    wait(mutex);
    read_count++;
    if(read_count==1)
        wait(writer);
    signal(mutex);
    signal(reader);

    /* reading is performed */

    wait(mutex);
    read_count--;
    if(read_count==0)
        signal(writer);
    signal(mutex);
} while (true);
```

当一个作者在做写操作时，第一个读者会在 writer 上等待，其余的读者会在reader 上等待。作者完成写操作时，读者都可以进行读操作，reader被解开，writer被锁上。只有当所有的读操作都结束后，writer会被解开。

这种方式不会发生死锁。如果作者在reader或者writer上等待，就一定会有读者在临界区，而且这些读者同时在临界区，并足够将 read_count 重置为 0，解开writer以及通过 signal(reader) 解开 reader。如果读者在reader上等待，就一定有作者在临界区，而且此时 writer 一定处于锁上状态，否则 reader 将会被解开，那么作者一定会完成，以解开 writer 和 reader。

这种解法作者不会饿死。这个时候作者将会共同占用 reader 导致其不会被无限阻塞，但仍然满足作者和读者互斥。一旦 reader被解开，作者就有可能准备进入临界区解锁，其余读者的数量将会有限，一旦最后一个读者完成读操作，writer 就会被解开，作者的等待将是有限的。

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.

解.

```c
/* Before all processes start. */
#include <pthread.h>

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

```c
/* For every process. */
int main(){
    pthread_mutex_lock(&mutex);
    StackNode *top = NULL;
    push(5, &top);
    push(10, &top);
    push(15, &top);

    int value = pop(&top);
    value = pop(&top);
    value = pop(&top);
    pthread_mutex_unlock(&mutex);
}
```