操作系统第六章作业

计试 81 白思雨 2186123935

6.4 请解释：为什么在单处理器系统上通过禁止中断来实现同步原语的方法不适用于用户级程序

答：如果一个用户级程序具有停止中断的能力，那么它能够停止计时器中断，防止上下文切换的发生，从而允许它使用处理器而不让其他进程执行。

6.9 考虑如何使用原子硬件指令实现互斥锁。假设定义互斥锁的结构如下：

```
typedef struct{
    int available;
}lock;
```

当 available 为 0 时，表示锁可用；当 available 为 1 时，表示锁不可用。通过这个 struct，说明如何采用指令 test_and_set()和 compare_and_swap()来实现如下函数，一定包括任何可能必要的初始化：

- void acquire(lock *mutex)
- void release(lock *mutex)

答：

```
void acquire(lock *mutex){
    while(test_and_set(*mutex.available));
}
```

```
void acquire(lock *mutex){
    while(compare_and_swap(*mutex.available, 0, 1) != 0);
}
```

```
void release(lock *mutex){
```

```
2.      *mutex.available = 0;
3.  }
```

6.11 假设一个系统有多个处理核。针对下面的各个场景，请讨论一下：哪一个是更好的加锁机制，是自旋锁？还是互斥锁？这里要求等待进程在等待可用锁时应睡眠。

- **用锁的时间很短。**
- **用锁的时间很长。**
- **线程在拥有锁时可能处于睡眠。**

答：如果锁被占用的时间很短，自旋等待的效果就非常好；反之，锁占用的时间很长，那么自旋线程只会白白浪费处理器资源，因此，如果自旋超过了限定的次数仍然没有获得锁，就应将线程挂起。互斥锁属于 sleep-waiting 类型的锁。例如在一个双核的机器上有两个线程（线程 A 和线程 B），它们分别运行在 Core0 和 Core1 上。假设线程 A 想要通过 pthread_mutex_lock 操作去得到一个临界区的锁，而此时这个锁正被线程 B 所持有，那么线程 A 就会被阻塞，所以，1 适用于自旋锁，2 和 3 适用于互斥锁。