操作系统期中作业

1. 使用条件变量需要配合wait和signal原语。一种更通用的同步形式是只用一条同步原语waituntil，它以任意的布尔谓词作为参数。例如： waituntil x < 0 or y+z < n。 其意义为等待直到所需要的条件达成，这样就不需要signal原语，也更加通用。然而其并没有被现代操作系统所采用。尝试回答可能的原因是什么?

2.互斥锁的两种实现：不断自旋（spin）和基于阻塞（block），什么情况下用spin好一点，什么时候基于阻塞的实现好一点?

3. 考虑可以购买一台每秒执行五亿条指令的单处理器，或者购买一台拥有8个核心的多处理器，其中每个核心每秒执行一亿条指令。根据阿姆达尔定律(Amdahl's Law)，解释什么样的程序应该使用第一种CPU，什么样的程序应该使用第二种CPU。

4. 考虑如下哲学家就餐问题的变种：所有筷子都放在桌子的中间，哲学家吃饭的时候只要从中任意拿两个筷子即可，吃完了再放回中间。有一种避免死锁的方式是提供足够多的资源。那么如果有N个哲学家，至少要提供多少筷子即可一定避免死锁问题？为什么?

5. 对于一个真实的计算机系统，可用的资源和进程的资源不再一成不变，而是在动态的变化。资源会损害和替换、新的进程也来来去去，新的资源会被购买并加入到系统。如果采用银行家算法控制死锁，那么下面哪些变化是安全的（不会导致死锁），哪些是变化可能导致不安全，为什么?

a. 增加可用资源 （新的资源加入到系统）
b. 减少可用资源 （资源被系统永久移除）
c. 增加一个进程的最大需求
d. 减少一个进程的最大需求
e. 增加进程的数量
f. 减少进程的数量

6. 下面的代码展示了一个栈的实现

```
typedef struct __Node {
     struct __Node *next;
     int value;
} Node;

void push(Node **top_pter, Node *n){
     n->next = *top_ptr;
     *top_ptr = n;
}

Node *pop(Node **top_ptr){
     if (*top_ptr == NULL)
          *return NULL;
     Node *p = *top;
     *top_ptr = (*top_ptr)->next;
     return p;
}
```

a）请分析该实现是否线程安全并说明原因

b）请尝试使用互斥锁来保证该代码片段的安全

c）除了基本的push和pop操作以外，栈通常还提供一个top操作，用于查询栈顶元素的值。该查询不会移出该元素（与pop不同）。如果我们实现该查询和上述代码片段中pop函数相似（除了不改变top_ptr指针），实现过程也不用互斥锁，而是直接读取top的指针的值，那么该实现会出现错误吗?

d）如果需要同步原语来保护top操作，用什么比较好？为什么?

7. 考虑我们给出了一种n个线程互斥的实现Flaky Lock，代码如下（注：假设我们是SC的内存模型，且编译器没有乱序优化）

```
#include <stdbool.h>
// 假设 ThreadID.get() 返回当前线程的 ID
int ThreadID_get() {
    // 实现获取线程 ID 的代码
}

typedef struct {
    int turn;
    bool busy;
} Flaky;

void Flaky_init(Flaky *lock) {
    lock->turn = 0;
    lock->busy = false;
}
```

```
void Flaky_lock(Flaky *lock) {
    int me = ThreadID_get();
    do {
        do {
            lock->turn = me;
        } while (lock->busy);
        lock->busy = true;
    } while (lock->turn != me);
}

void Flaky_unlock(Flaky *lock) {
    lock->busy = false;
}
```

a）这个实现是否满足互斥性?

b）这个实现是否无饥饿?

c）这个实现是否无死锁？如果没有死锁请解释为什么，如果有死锁，同样给出解释，并给出一个改进版本，使其不会出现死锁

8. 为什么在用户态不能关中断，给出两个理由