理论题

5.2

a.

```
void read(){
    FILE *cardfile, *tempfile;
    char c;
    char blank = " ";
    cardfile = fopen("cardfile_name", "r");
    tempfile = fopen("tempfile_name", "w");
    while((c = fgetc(cardfile)) != EOF){        //一个字符一个字符地读取
        fputs(c, tempfile);
        fputs(blank, tempfile);
    }
    fclose(cardfile);
    fclose(tempfile);
}

void transform(){
    FILE *tempfile1, *tempfile2;
    char c;
    char blank = " ";//替换的字符
    tempfile1 = fopen("tempfile1_name", "r");
    tempfile2 = fopen("tempfile2_name", "w");
    while((c = fgetc(tempfile1)) != EOF){
        if(c != "*"){
            fputs(c, tempfile2);
        }else{
            char next = fgetc(tempfile1);
            if(next == "*"){
                fputs(blank, tempfile2);
            }else if(next == EOF){
                fputs(c, tempfile2);
                return;
            }else{
                fputs(c, tempfile2);
                fputs(next, tempfile2)
            }
        }
    }
    fclose(tempfile1);
    fclose(tempfile2);
}

void print(){
    FILE *fp;
    char c;
    int count = 0;
    fp = fopen("file_name", "r");
    while((c = fgetc(fp)) != EOF){
        if(count == 125){
            printf("\n");       //满125个之后换行
```

```
            printf("%c", c);
            count = 1;
        }else{
            printf("%c", c);
            count++;
        }
    }
    fclose(fp);
}
```

b.

read协程读取卡片，利用rs将字符传递给squash协程，并且随后添加一个空格在该字符后面。squash协程查找双星号”**“,若传来的字符不是星号，则利用sp缓冲区直接将其传给print协程存入输出流；若是星号，则再继续读，下一个仍为星号时，利用sp来修改字符，传给print，否则直接传给print上一个星号以及本次读入的字符。最后，print将接收来的字符流打印为每行125个字符的序列。

c.

```
void read(){
    while(true){
        if(READCARD (inbuf)){  //如果可以读入完整一行
            for(int i = 0; i < 80; i++){
                rs = inbuf[i];
                RESUME squash;
            }
            rs = " ";
            RESUME squash;
        }else{  //如果读完了
            rs ='\0';   //将最后一个字符设置为字符串结束字符
            RESUME squash;  //传给协程
        }
    }
}
void squash(){
    while(true){
        if(rs == "\0"){     //如果读入字符结束标志，则不再恢复read协程
            sp = rs;
            RESUME print;
        }else if(rs != "*"){
            sp = rs;
            RESUME print;
            RESUME read;
        }else{
            RESUME read;
            if(rs == "*"){
                sp = " ";
                RESUME print;
                RESUME read;
            }else{
                sp = "*";
                RESUME print;
                sp = rs;
                RESUME print;
                RESUME read;
            }
```

```
        }
    }
}
void print(){
    while(true){
        for(int j = 0; j < 125 ; j++){
            if(sp == '\0'){     //如果是结束符，则直接将outbuf打印后返回
                outbuf[j] = sp;
                OUTPUT (outbuf);
                return;
            }
            else{
                outbuf[j] = sp;
                RESUME squash;
            }
        }
        OUTPUT (outbuf);
    }
}
```

d.

可以通过三个并发进程来实现。

read：读取卡片并简单地通过一个有限缓冲区（inbuf）将字符（带有额外的尾随空格）传递给一个进程squash　squash：查找双星号，并通过第二个有限缓冲区（outbuf）将一个修改过的字符流传递给一个进程print　　print:从第二个缓冲区中提取字符，并将它们打印出来。

```
const int sizeofinbuf = 80;
const int sizeofoutbuf = 125;
semaphore inempty = sizeofinbuf / 2, outempty = sizeofoutbuf, infull = 0,
outfull = 0;
semaphore mutex1 = 1, mutex2 = 1;
int in, out;
char inbuf[sizeofinbuf], outbuf[sizeofoutbuf]

void read(){
    while(true){
        semWait(inempty);
        semWait(mutex1);
        //读字符，加空格，放入输入缓冲区
        readcard();
        semSignal(mutex1);
        semSignal(infull);
    }
}

void squash(){
    while(true){
        semWait(infull);
        semWait(mutex1);
        //取出输入缓冲区的字符，作变换
        transform();
        semSignal(mutex1);
        semSignal(inempty);

        semWait(outempty);
```

```
        semWait(mutex2);
        //放入输出缓冲区
        toprint();
        semSignal(mutex2);
        semSignal(outfull);
    }
}

void print(){
    while(true){
        semWait(outfull);
        semWait(mutex1);
        //取出并打印
        take_and_print();
        semSignal(mutex2);
        semSignal(outempty;
    }
}

void mian(){
    parbegin(read, squash, print);
}
```

5.7

a. 进程在进入临界区之前会被分配一个票号。

boolean 数组表示进程是否在取号，number数组中存放对应进程得到的票号。

分配票号的规则是：在当前已经拿到票号的进程（即已经进入临界区和还在排队等待的进程）中，最大票号数再加1为下一进程的票号。

票号最小的进程在进入临界区时具有最高的优先级。如果多个进程拥有相同的票号，则最小数字号的进程将进入临界区。当进程退出临界区时，其票号重置为零。

b. 每一个进程都被分配了唯一的进程号同时也拥有票号，这保证任何时候都有一个唯一且严格的进程顺序进入临界区，所以死锁不会发生。

c.

说明该算法实施了互斥，即说明：当有一个进程进入临界区执行代码时，其余进程不能进入。也就是说明

(number([i],i)) < (number([k],k)) 可以保证上述要求成立。

即对于已经处于临界区的进程i，而另一个获得票号的进程k（即number[k]已知）试图要进入其临界区时，关系(number([i],i)) < (number([k],k)) 成立

为了证明上述命题，我们定义以下时间节点：
Ti1： 进程i最后一次访问choosing[k]，对于j=k，在它的第一次等待，我们在Ti1时刻有choosing[k]=false。
Ti2 ： 进程i开始最终执行，对于j=k，还在它的第二个while循环中。因此我们在Ti2时刻有Ti1<Ti2。

Tk1 进程k进入重复循环。
Tk2 进程k完成计算number[k]。
Tk3 进程k将choosing[k]设置为false。我们有Tk1<Tk2<Tk3。

Ti1时刻，choosing[k]=false，所以我们得到Ti1<Tk1或Tk3<Ti1。

第一种情况，由于进程i在进程k之前被分配了它的票号，所以我们得到了number[i]<number[k]，所以此时k进入不了临界区，这满足我们要证明条件。
第二种情况，我们有Tk2<Tk3<Ti1<Ti2，因此Tk2<Ti2。这意味着在Ti2处，进程i已经读取了number[k]的值，而此时，由于Ti2是j=k时的第二个while循环，我们有（number[i]，i）<（number[k]，k）。

所以说明该算法实施了互斥。

5.13

a.

第29行直接must_wait = false有问题。原因有两点：第一，在第10行一个刚刚解除阻塞的进程要再次进入互斥时，若此时新来了一个进程，它有可能抢占原进程优先使用资源。第二，等待队列中的进程解除阻塞，与将它们唤醒重新开始执行并且更新active和waiting两个计数器，这两个过程之间存在一定的时间延迟。假设当前阻塞队列中有3个进程，当最后一个占用资源的进程离开时会解除那3个进程的阻塞，但这时如果没有立刻更新计数器，并且那3个进程没有立刻恢复执行，又恰巧新来了一个进程也要获取资源，就会出现临界区资源被多于三个进程访问的情况。

b.

可以解决，但是会有可能产生饥饿。将if换成while可以使解除阻塞的进程在进入临界区时重新再检测一遍状态变量，来判断他们是否可以进入。但是同时会出现后来的程序比先来的程序优先获得资源的可能，如果一直不断地有新进程加入，先来的进程将一直被阻塞，那么就会出现饥饿的情况。

5.14

a.

该版本代码在最后一个使用资源的进程离开时就更新了active和waiting计数器，并且对第30行的must_wait变量赋值修改，这些都使得在新进程进入临界区之前两个计数器反映的都是系统当前的真实状态，同时新来的进程也不用重新进入临界区。这消除了上一题a问中的时间延迟情况。

b.

当一个进程执行完第8行的 semSignal(mutex);但还没有执行第9行的semWait(block);时出现错误（例如其时间片被消耗完），有可能会有一个新的进程获取mutex，并阻塞在block上，这使得该新来的进程比原来的进程先访问资源。但是这种情况的可能性不大。

c.

这种模式由最后离开的进程来修改全局变量，如计数器以及状态信息，并且还会唤醒阻塞中的进程。这样使得被唤醒的进程不用再次判断是否满足条件。

5.23

分析题目：该问题共有三种并发进程：驯鹿reindeer、小精灵elves、圣诞老人santa

驯鹿

1. 他们要在热带待到最后的时刻
2. 只有最后一头，也就是第九头驯鹿回来，才能唤醒圣诞老人
3. 提前回来的驯鹿，要进入队列中等待（温暖的棚子），一直被阻塞
4. 9头驯鹿回来后，等待圣诞老人要为他们准备的雪橇——临界区

小精灵

1. 三只精灵才会唤醒圣诞老人，也即临界区只有三个小精灵进程可以进入
2. 唤醒圣诞老人之后会问他问题——临界区

圣诞老人

1. 等待被唤醒
2. 对于圣诞老人来说，处理驯鹿的事情更重要
3. 当第九只驯鹿全部回来后，圣诞老人要做的事：

   更新驯鹿计数器，将其置为0

   将原来排在队列中的八只驯鹿叫回来

   给九只驯鹿准备雪橇
4. 处理完3后，当第三个小精灵到来时，圣诞老人要做的事：

   更新来问问题的小精灵数目为0

   将原来2个正在等待的小精灵叫回来

   为三个小精灵解决问题

```
#define REINDEER 9
#define ELVES 3

/* Semaphores */
only_elves = 3, /* 可以有3只小精灵问问题 */
emutex = 1, /* 保护精灵个数的互斥量 */
rmutex = 1, /* 保护驯鹿计数器的互斥量 */
rein_semWait = 0, /* 阻塞驯鹿，直到第九只回来 */
sleigh = 0, /* 雪橇是否做好 */
elf_semSignal = 0, /* 阻塞小精灵，直到凑够3只 */
santa = 0, /* 圣诞老人是否被唤醒 */
problem = 0, /* 小精灵是否可以向圣诞老人提问 */
ask = 0,/*是否允许提问*/
reply = 0; /* 是否得到回答 */

/* Shared Integers */
rein_ct = 0; /* 回来的驯鹿个数 */
elf_ct = 0; /* 要问问题的小精灵 */

/* Reindeer Process */
while(1) {
     return until Christmas is close
     semWait (rmutex)
     rein_ct++       /*保护全部变量*/
     if (rein_ct == REINDEER) {      /*如果九只驯鹿全部回来 */
         semSignal (rmutex)      //注意这里才恢复互斥量，防止判断驯鹿数时出错
         semSignal (santa)       /*唤醒圣诞老人*/
     }
     else {
         semSignal (rmutex)
         semWait (rein_semWait)      /* 阻塞提前回来的驯鹿 */
}

/* 全部驯鹿回来了 */
     semWait (sleigh)  /* 等待雪橇制作 */
} /* end "forever" loop */
```

```
//小精灵的代码结构和驯鹿类似
/* Elf Process */
while(1) {
     semWait (only_elves) /* 允许三个小精灵进入临界区 */
     semWait (emutex)
     elf_ct++
     if (elf_ct == ELVES) {
         semSignal (emutex)
         semSignal (santa) /* 满三个小精灵即可唤醒圣诞老人 */
     }
     else {
         semSignal (emutex)
         semWait (elf_semSignal) /* 不满三只则被阻塞 */
     }

     semWait (problem)  /* 等待问问题 */
     ask question
     semSignal(ask)              //---------------问完问题，发送信号量给圣诞老人，可以回
答了!
     semWait (reply)             //等待回答
     semSignal (only_elves)      //恢复信号量
} /* end "forever" loop */

/* Santa Process */
while(1) {
 semWait (santa) /* 圣诞老人被唤醒 */
     //优先处理驯鹿
     if (rein_ct == REINDEER) {
         semWait (rmutex)
         rein_ct = 0 /* 计数器置0 */
         semSignal (rmutex)
         for (i = 0; i < REINDEER - 1; i++)
             semSignal (rein_semWait)       /*将8只提前回来的驯鹿唤醒*/
         for (i = 0; i < REINDEER; i++){
             make sleigh
             semSignal (sleigh)             /*准备雪橇*/
         }
     }
     else { /* 来了第3只精灵 */
         for (i = 0; i < ELVES - 1; i++)
             semSignal (elf_semSignal)  /*将前两只唤醒*/
         semWait (emutex)
         elf_ct = 0 /*计数器置0*/
         semSignal (emutex)
         for (i = 0; i < ELVES; i++) {
            semSignal (problem)     //可以问问题了!
            semWait(ask)
            answer that question
            semSignal (reply)       //回答完问题了!
        }
    }
} /* end "forever" loop */
```

5.25

可能出现多个读者一直在读取，而写者进程一直得不到调度而出现饥饿的情况

实验题

利用信号量实现有界缓存的生产者/消费者问题，要求有结果截屏和解释

编写该代码时，我做了如下设定;

1. 一共有5个生产者线程和5个消费者线程，且分别为其编号为1~5
2. 有界缓冲区的大小为5
3. 生产者和消费者每次都只能生产一份或消费一份资源，该资源用A~Z26个大写字母表示，随机生成

全局变量说明

```
#define BufferSize 5 //缓冲区大小
//始终有 empty + full = buffersize

sem_t empty;     //该信号量会被初始化为 缓冲区大小，若缓冲区满，则其变为0，阻塞后续线程
sem_t full;      //该信号量和empty配合以达到同步效果，即缓冲区有资源才可消费，缓冲区满不可生
产
sem_t s;         //该信号量用于给生产和消费过程加锁，实现互斥

int in = 0;      //控制生产出的item置放位置
int out = 0;     //控制被消费的item所处位置
char buffer[BufferSize];     //可以存放A~Z的缓冲区
```

生产者代码块

```
void *producer(void *arg)
{
    while(1){
        char item = rand() % 26 + 'A'; // Produce a random item
        sem_wait(&empty);   //如果empty>0，表示有位置供生产者生产
        sem_wait(&s);
        buffer[in] = item;
        printf("Producer %d produces item %c at %d position.\n", *
((int*)arg),buffer[in],in);
        in = (in+1)%BufferSize;
        sem_post(&s);
        sem_post(&full);    //缓冲区资源加一
    }
}
```

消费者代码块

```
void *consumer(void *cno)
{
    while(1){
        sem_wait(&full);     //full>0,表示缓冲区中有资源可以使用，消费者线程进入缓冲区
        sem_wait(&s);
        int item = buffer[out];
        printf("Consumer %d takes item %c from %d position.\n",*((int
*)cno),item, out);
        out = (out+1)%BufferSize;
        sem_post(&s);
        sem_post(&empty);
    }
}
```

主函数

```
int main()
{
    pthread_t pro[5],con[5];    //存放线程
    // 初始化信号量
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);
    sem_init(&s, 0, 1);

    int a[5] = {1,2,3,4,5}; //用来给线程分配线程号

    //创建生产者消费者线程各5个
    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }

    //销毁信号量
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&s);

    return 0;
}
```

说明

1. 本程序由pthread以及信号量实现，pthread用于创建多线程，由于其pthread-create函数中要求其代表线程所执行函数的参数是指针类型，所以producer和consumer函数都设置成了void*的返回类型。
2. 至于producer和consumer的参数，其代表着被分配的线程

3. 对于信号量的真实应用，需要注意到声明时为sem_t类型，并且需要用专门的sem_init函数来进行初始化，最后使用完毕还需要用sem_destroy函数来销毁，而且要注意信号量的函数传入形式都是&semaphore

结果

由于该程序是一直执行的程序，我们只截取从运行一开始的一部分结果来说明

```
liyu@liyu-VirtualBox:~/os/hw5$ ./producer_consumer
Producer 5 produces item N at 0 position.
Producer 5 produces item L at 1 position.
Producer 5 produces item R at 2 position.
Consumer 3 takes item N from 0 position.
Consumer 2 takes item L from 1 position.
Consumer 1 takes item R from 2 position.
Producer 4 produces item W at 3 position.
Producer 4 produces item M at 4 position.
Producer 4 produces item Q at 0 position.
Producer 4 produces item B at 1 position.
Consumer 2 takes item W from 3 position.
Consumer 3 takes item M from 4 position.
Consumer 4 takes item Q from 0 position.
Consumer 5 takes item B from 1 position.
Producer 4 produces item H at 2 position.
Producer 4 produces item C at 3 position.
Producer 4 produces item D at 4 position.
Producer 4 produces item A at 0 position.
Consumer 4 takes item H from 2 position.
Consumer 1 takes item C from 3 position.
Consumer 3 takes item D from 4 position.
Producer 3 produces item B at 1 position.
Producer 3 produces item Z at 2 position.
Consumer 4 takes item A from 0 position.
Consumer 2 takes item B from 1 position.
Consumer 5 takes item Z from 2 position.
Producer 5 produces item B at 3 position.
Producer 5 produces item W at 4 position.
Producer 5 produces item K at 0 position.
Consumer 4 takes item B from 3 position.
Producer 4 produces item R at 1 position.
Producer 4 produces item Y at 2 position.
Consumer 3 takes item W from 4 position.
Consumer 3 takes item K from 0 position.
Consumer 1 takes item R from 1 position.
Consumer 2 takes item Y from 2 position.
Producer 3 produces item O at 3 position.
```

由图，程序运行开始后，5号生产者分别生产了N，L，R，依次置于缓冲区中，接着3号消费者消费0号缓冲区中资源，即N，2号消费者消费1号缓冲区中资源……以此类推

我们可以看到每一次生产者和消费者号是随机的，生成的资源也是随机的，只有放入缓冲区和从缓冲区取走是有序的。