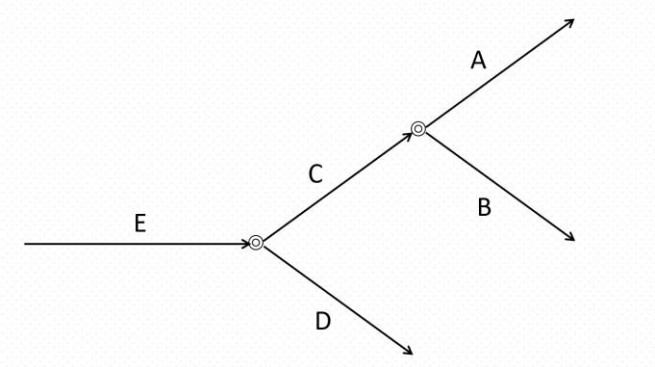
# 2021 年 2019 级《操作系统》期中测验

（2021211313-14-17-18）

班级：_____　　学号：_____ 姓名：_____

注：交卷时，本试题页填写姓名、班级，随同答题页一起上交

1. (30 分)现有 5 个进程 A、B、C、D、E，进程 C 完成之后 A 和 B 才开始执行，进程 E 完成之后 C 和 D 才开始执行。使用信号量和 wait()、signal 操作，描述上述进程间的同步关系，要求：说明信号量含义及其初值，信号量名称应有明确含义，与题意相符。
答案：




本题考查进程间同步，设计如下进程间二元同步信号量：
Semaphore （5 分）

```
syncC-A=0;      //C 完成后，通知 A
syncC-B=0;      //C 完成后，通知 B
syncE-C=0;      //E 完成后，通知 C
syncE-D=0;      //E 完成后，通知 D
```

```
A:                  B:                  C:
wait(syncC-A);      wait(syncC-B);      wait(syncE-C);
工作区              工作区              工作区
退出结束            退出结束            signal(syncC-A);
                                        signal(syncC-B)

D:                  E:
wait(syncE-D);      工作区
工作区              signal(syncE-C);
退出结束            signal(syncE-D);
```

## Example One

- Error solution for process synchronization in Fig. (C)

| $P_1$: | $P_2$: | $P_3$: |
|---|---|---|
| begin | begin | begin |
| S1; | wait(sync); | wait(sync); |
| S2; | S3; | S5; |
| signal(sync); | S4; | S6: |
| signal(sync); | end | end |
| end | | |

问题：
可能存在$P_1$执行两次signal()操作后，$P_2$连续运行两轮，执行两次wait()操作，导致$P_3$无法启动

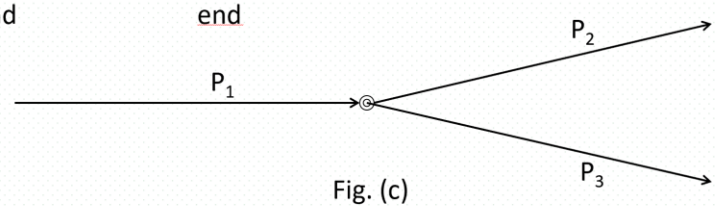
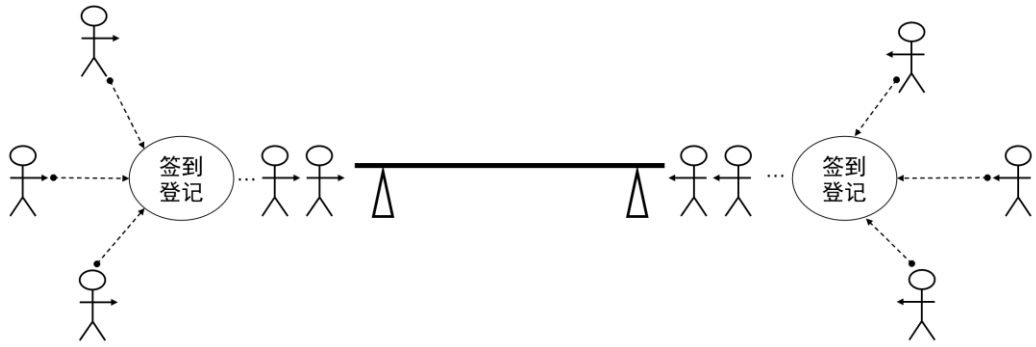



Fig. (c)

2. （30 分）两群人双向过独木桥。对左端人群，人群无序到达独木桥左端，在桥左端签到处签到登记，然后组队有序依次过桥，签到处每次只允许一个人签到登记。对右端人群，其行为类似于左端人群。

独木桥只允许人群单向通过，当左端第一个人上桥后，左端后续人群可依次上桥、过桥，此时右端人群只能等待；类似地，当右端第一个人上桥后，右端后续人群可依次上桥、过桥，此时左端人群只能等待。




设计基于信号量的解决方案，保证左右两端人群依次过桥。要求：

（1）给出信号量的定义和初值，信号量名称应有明确含义，与题意相符；

（2）采用所定义的信号量和 wait()、signal()操作，描述左、右两端人群的签到登记、过桥行为；

（3）左端人群中，最多允许 M 个同时上桥，右端人群中最多允许 N 个同时上桥。

答案：

本题是读写者问题的扩展：

（1）左右两群人相当于两队读者，无写者；

（2）利用多元信号量 emptyL、emptyR，分别控制左右两队上桥总人数，初值 emptyL=M，emptyR=N；

（3）利用二元互斥信号量 loginL、loginR 控制左右两端人群签到登记，

Semaphore

```
wrt=1;
mutexL=1;        mutexR=1;      //控制对左右两端上桥人数的互斥访问
readcountL=0;    readcountR=0;
loginL=1;        loginR=1;
emptyL=M;        emptyR=N;
```

左端人群：

```
wait(loginL);
signal(loginL);
wait(emptyL)
wait(mutexL);
readcountL++;
if (readcountL == 1)
        wait(wrt);
signal(mutexL);
                ...
Passing bridge
```

```
wait(mutexL);
readcountL--;
if (readcountL == 0)
    signal(wrt);
signal(mutexL);
signal(emptyL)
```

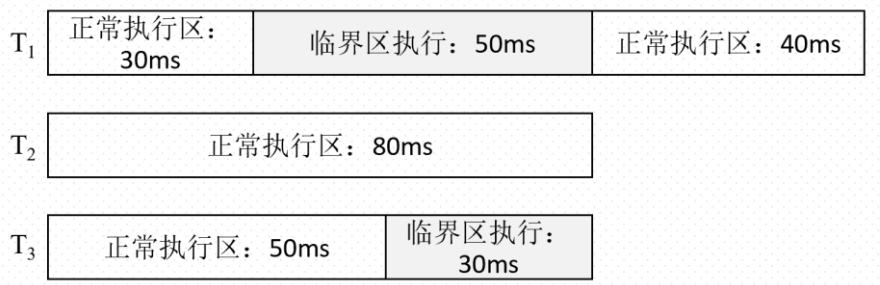
说明：“注册登记”与“判断缓冲区空、进入 buffer 临界区”要分开，执行 wait(LoginL)，注册成功后，立刻 signal(LoginL)释放注册登记控制器，运行后续进程注册登记

右端人群：

```
wait(loginR);
signal(loginR);
wait(emptyR)
wait(mutexR);
readcountR++;
if (readcountR == 1)
        wait(wrt);
signal(mutexR);
...
Passing bridge
```

```
wait(mutexR);
readcountR--;
if (readcountR == 0)
    signal(wrt);
signal(mutexR);
signal(emptyR)
```

3. （35 分）线程的执行轨迹根据其自身业务逻辑分为：（1）与资源竞争和同步互斥无关的正常执行区，（2）通过信号量互斥使用资源的临界区。线程 $T_1$、$T_2$ 、$T_3$ 在单核 CPU 上并发执行，三个线程的执行逻辑分别如下所示：




线程进入临界区之前，需要执行加锁/信号量 wait()操作，与其它线程竞争使用资源。如果资源已被其它线程占用，加锁失败，线程进入阻塞/等待态；当其它占用资源的线程释放资源后，执行 signal()操作，被阻塞的线程立即被重新唤醒，进入就绪态，并立即被 OS 重新调度，为其分配 CPU。

线程 $T_1$ 和 $T_3$ 之间竞争使用同一种资源，$T_2$ 则与 $T_1$ 和 $T_3$ 无资源竞争关系。

三个线程 $T_1$、$T_2$ 、$T_3$ 的优先级分别为 10、20、30，操作系统采用基于优先级的抢占式 CPU 调度策略，允许新进入系统的高优先级线程抢占处于正常执行区或临界区中的低优先级线程的 CPU。

假设 $T_1$、$T_2$ 、$T_3$ 进入系统的时间分别为：0ms，40ms，60ms。

要求：

（1）画出三个线程并发执行时的甘特图，计算三个线程各自的周转时间和等待时间；

（2）三个线程执行完毕的先后顺序是否与线程优先级的顺序相符，高优先级线程是否先结束，为什么？

进一步地，采用不可抢占临界区协议，改进上述线程调度方式：不允许抢占位于临界区中、正在使用临界资源的低优先级线程的 CPU。

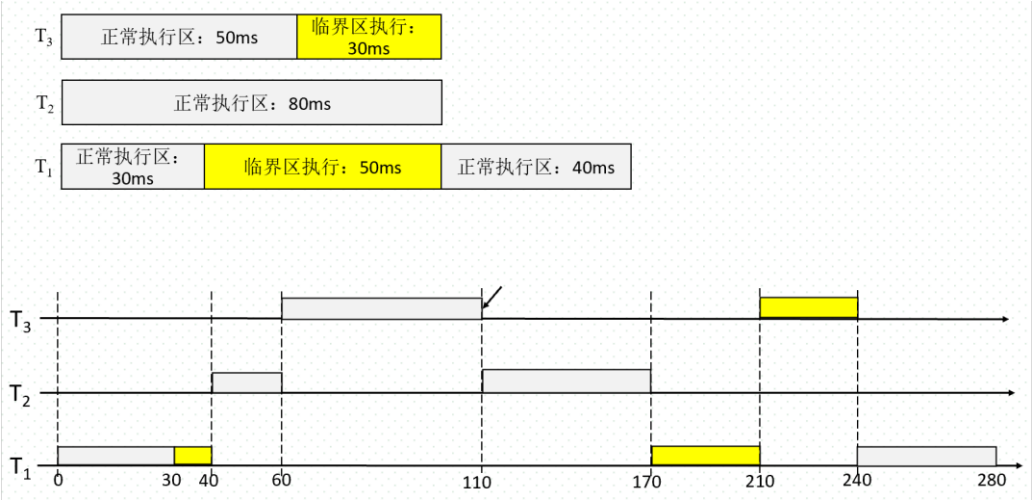
要求：

（3）画出三个线程并发执行时的甘特图；

（4）计算三个线程各自的周转时间和等待时间。

答案：

（1）三个线程执行的甘特图：




（2）各线程的周转时间、等待时间如下

T1：　周转时间=280-0=280

　　等待时间=(170-40) + (240-210)=160

T2：　周转时间=170-40=130

　　等待时间=110-60=50

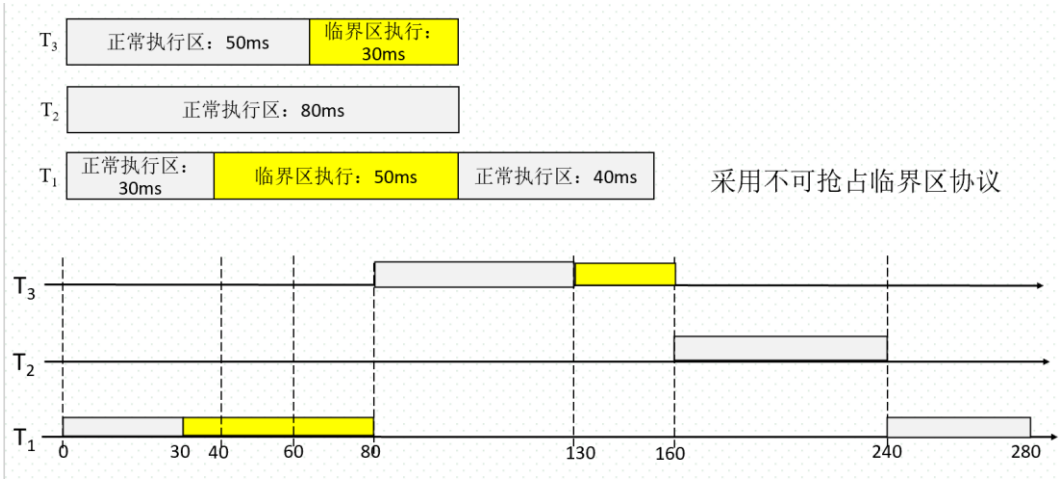
T3：　周转时间=240-60=180

　　等待时间=210-110=100

线程结束顺序与线程优先级不相符。

采用不可抢占临界区协议：

（3）甘特图




（4）各线程的周转时间、等待时间如下

T1：　周转时间=280-0=280

　　等待时间=240-80=160

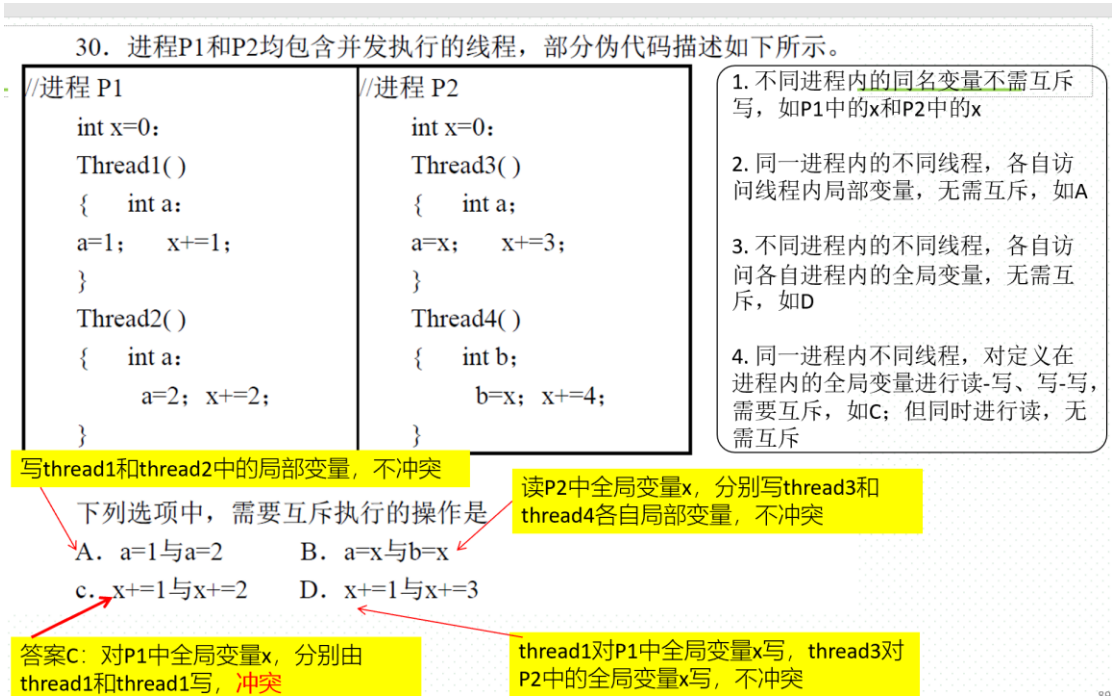
T2：　周转时间=240-40=200

　　等待时间=160-40=120

T3:　周转时间=160-60=100

等待时间=80-60=20

4. 简答（5 分）




答案:

需要互斥的操作是 C;

原因：一个进程内的全局变量 x，被本进程内的两个线程同时“写”，需要进行互斥

2. （30 分）计数（多元）信号量控制进程/线程互斥访问具有多个资源实例的临界资源。多元信号量可以用内核空间中的二元信号量和一个用户空间中的整型计数变量替代实现。针对生产者-消费者问题，给出采用“二元信号量 + 整型计数变量”替代计数信号量 empty、full 的解决方案。要求：

（1）给出信号量的定义和初值，信号量名称应有明确含义，与题意相符；

（2）采用所定义的信号量和 wait()、signal()操作，描述生产者、消费者并发访问共享缓冲区 buffer 的行为；

（3）有多个生产者、消费者，允许一个生产者和一个消费者同时进入缓冲区，分别向不同的空缓冲单元写入数据项和从满缓冲单元读取数据项；不允许多个生产者同时进入缓冲区写数据，也不允许多个消费者同时进入缓冲区读取数据。

答案：

**基于“二元信号量+整型计数变量”的生产者-消费者问题**

```
1  int empty=5;
2  int full=0;
3  semaphore empty_mutex=1;
4  semaphore suspend_P=0;
5  semaphore full_mutex=1;
   semaphore suspend_C=0;
                mutexP=1
                mutexC=1;

6   void producer(void)
7   {
8      int new_msg;
```

```
11  while (true) {
12    new_msg=producer_new();
13    wait(empty_mutex);  //加锁
14    if (empty==0)  //判断"无空缓冲区""
15        { signal(empty_mutex); wait(suspend_P) };
            //条件不满足时，挂起
16    else {empty-- ;
17        signal(empty_mutex);  //解锁
          };
18    wait(mutexP);  //竞争buffer访问权
19    buffer_add_safe(new_mag); //进入临界区写数据
20    signal(mutexP);
```

**基于“二元信号量+整型计数变量”的生产者-消费者问题**

```
21    wait(full_mutex);      //加锁
22    full++;
23    signal(suspend_C);     //唤醒被阻塞的消费者
24    signal(full_mutex);    //解锁
25    }
26 }
27
```

说明：

对多元信号量empty，用整型变量empty、二元互斥信号量empty_mutex、二元挂起信号量suspend_P替代：

(1) 用整型变量empty替代多元信号量的empty的资源计数功能；

(2) 用empty_mutex控制对整型变量empty的++、--的互斥访问；

(3) 当无空缓冲区单元时，通过wait(suspend_P)，挂起生产者

类似地，多元信号量full，用整型变量full、二元互斥信号量full_mutex、二元挂起信号量suspend_C替代

## 基于“二元信号量+整型计数变量”的生产者-消费者问题

```
28  void consumer(void)
29  {
30    int  cur_msg;
31    while (true) {
32      wait(full_mutex);    // 加锁
33      if (full==0)    //无满缓冲区
34        {signal(full_mutex); wait(suspend_C)};
                  //条件不满足，阻塞线程
35       else { full--;
36            signal(full_mutex);   //解锁
              }

37    wait(mutexC);
37    cur_msg=buffer_remove_safe();
38    signal(mutexC);
39
40    wait(empty_mutex);   //加锁
41    empty++;
42    signal(suspend_P);
       // 唤醒阻塞的生产者
43    signal(empty_mutex);

44     consume_msg(cur_msg);
45   }
46  }
```