

semaphore mutex = 1;      *// 互斥锁，保护共享变量*

semaphore a = N;          *// A物品的剩余可入库量（初始为N）*

semaphore b = M;          *// B物品的剩余可入库量（初始为M）*

semaphore empty\_A = N;    *// A物品的剩余仓库容量（初始为N）*

semaphore empty\_B = N;    *// B物品的剩余仓库容量（初始为N）*

semaphore full\_A = 0;     *// A物品的已入库数量（初始为0）*

semaphore full\_B = 0;     *// B物品的已入库数量（初始为0）*

cobegin

*// A物品入库进程*

    procedure A {

        while (1) {

            P(empty\_A);           *// 检查A物品仓库是否有空位*

            P(a);                 *// 检查A-B≤N是否满足*

            P(mutex);

*// 执行A物品入库操作*

            V(mutex);

            V(b);                 *// 允许B物品入库（影响B-A≤M）*

            V(full\_A);            *// 通知消费者A物品已入库*

        }

    }

*// B物品入库进程*

    procedure B {

        while (1) {

            P(empty\_B);           *// 检查B物品仓库是否有空位*

            P(b);                 *// 检查B-A≤M是否满足*

            P(mutex);

*// 执行B物品入库操作*

            V(mutex);

            V(a);                 *// 允许A物品入库（影响A-B≤N）*

            V(full\_B);            *// 通知消费者B物品已入库*

        }

    }

*// 消费者进程（组装C）*

    procedure C {

        while (1) {

            P(full\_A);            *// 等待A物品可用*

            P(full\_B);            *// 等待B物品可用*

            P(a);                 *// 检查A-B≤N是否允许取出A*

            P(b);                 *// 检查B-A≤M是否允许取出B*

            P(mutex);

*// 执行组装操作：取出1个A和1个B*

            V(mutex);

            V(a);                 *// 释放A物品的约束*

            V(b);                 *// 释放B物品的约束*

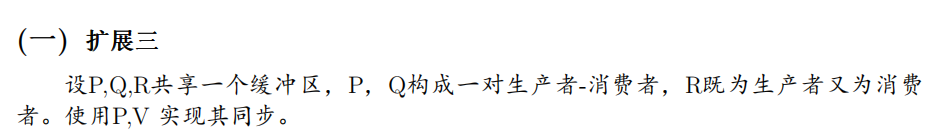
            V(empty\_A);           *// 释放A物品的仓库空位*

            V(empty\_B);           *// 释放B物品的仓库空位*

        }

    }

coend



semaphore mutex = 1;  *// 互斥信号量，保护缓冲区访问*

semaphore full = 0;   *// 初始时缓冲区无产品*

semaphore empty = 1;  *// 初始时缓冲区可放入一个产品*

cobegin

*// 纯生产者 P*

    process P {

        while (true) {

            P(empty);      *// 检查是否有空位*

            P(mutex);     *// 进入临界区*

            生产一个产品;   *// 生产动作*

            V(mutex);     *// 离开临界区*

            V(full);      *// 增加可消费产品数*

        }

    }

*// 纯消费者 Q*

    process Q {

        while (true) {

            P(full);       *// 检查是否有产品*

            P(mutex);      *// 进入临界区*

            消费一个产品;   *// 消费动作*

            V(mutex);     *// 离开临界区*

            V(empty);     *// 增加空位*

        }

    }

*// 既是生产者又是消费者的 R*

    process R {

        while (true) {

*// R 作为生产者*

            if (empty == 1) {  *// 检查是否有空位*

                P(empty);      *// 预留空位*

                P(mutex);      *// 进入临界区*

                生产一个产品;    *// 生产动作*

                V(mutex);     *// 离开临界区*

                V(full);       *// 增加可消费产品数*

            }

*// R 作为消费者*

            if (full == 1) {  *// 检查是否有产品*

                P(full);     *// 预留产品*

                P(mutex);   *// 进入临界区*

                消费一个产品;  *// 消费动作*

                V(mutex);    *// 离开临界区*

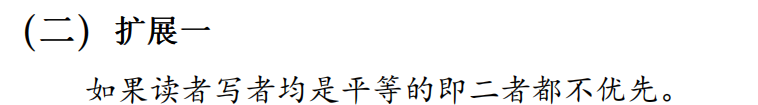
                V(empty);    *// 增加空位*

            }

        }

    }

coend



int count = 0;          *// 当前正在读取的读者数量*

semaphore mutex = 1;    *// 保护对count变量的互斥访问*

semaphore rw = 1;       *// 保证读者和写者互斥访问共享资源*

semaphore w = 1;        *// 实现"写优先"（防止写者饥饿）*

*// 写者进程*

void writer() {

    while(1) {

        P(w);          *// 获取写优先锁*

        P(rw);         *// 获取读写锁，独占访问*

*// 写入操作*

        writing();

        V(rw);         *// 释放读写锁*

        V(w);         *// 释放写优先锁*

    }

}

*// 读者进程*

void reader() {

    while(1) {

        P(w);          *// 获取写优先锁（防止写者无限等待）*

        P(mutex);      *// 互斥访问count变量*

        count++;       *// 增加读者计数*

        if (count == 1) {  *// 如果是第一个读者*

            P(rw);      *// 阻止写者访问*

        }

        V(mutex);      *// 释放count锁*

        V(w);         *// 释放写优先锁*

*// 读取操作*

        reading();

        P(mutex);      *// 互斥访问count变量*

        count--;       *// 减少读者计数*

        if (count == 0) {  *// 如果是最后一个读者*

            V(rw);      *// 允许写者访问*

        }

        V(mutex);      *// 释放count锁*

    }

}

读者优先：

*// 定义信号量*

semaphore mutex = 1;      *// 保护readcount的互斥信号量*

semaphore rw = 1;         *// 读写互斥的信号量*

int readcount = 0;        *// 当前读者数量*

*// 读者进程*

Process reader(int i) {

    while (true) {

*// 进入区*

        P(mutex);             *// 获取readcount的互斥访问*

        readcount = readcount + 1;

        if (readcount == 1) { *// 如果是第一个读者*

            P(rw);           *// 阻止写者访问*

        }

        V(mutex);            *// 释放readcount的互斥访问*

*// 临界区 - 读取数据*

        perform\_read\_operation(i);

*// 退出区*

        P(mutex);            *// 获取readcount的互斥访问*

        readcount = readcount - 1;

        if (readcount == 0) { *// 如果是最后一个读者*

            V(rw);           *// 允许写者访问*

        }

        V(mutex);            *// 释放readcount的互斥访问*

*// 其他处理*

        do\_other\_things();

    }

}

*// 写者进程*

Process writer(int j) {

    while (true) {

*// 进入区*

        P(rw);               *// 获取读写互斥锁*

*// 临界区 - 写入数据*

        perform\_write\_operation(j);

*// 退出区*

        V(rw);              *// 释放读写互斥锁*

*// 其他处理*

        do\_other\_things();

    }

}

*// 主程序*

Process main() {

    cobegin

        reader(1); reader(2); reader(3); *// 启动多个读者进程*

        writer(1); writer(2);            *// 启动多个写者进程*

    coend

}

写者优先：

*// 定义信号量*

semaphore rw = 1;         *// 读写互斥信号量*

semaphore mutex = 1;      *// 保护readcount的互斥信号量*

semaphore w = 1;          *// 写者优先控制信号量*

int readcount = 0;        *// 当前读者数量*

int writecount = 0;       *// 当前写者数量*

semaphore mutex\_w = 1;    *// 保护writecount的互斥信号量*

*// 读者进程*

Process reader(int i) {

    while (true) {

        P(w);               *// 检查是否有写者等待*

        P(mutex);           *// 获取readcount的互斥访问*

        readcount = readcount + 1;

        if (readcount == 1) { *// 如果是第一个读者*

            P(rw);          *// 尝试获取读写锁*

        }

        V(mutex);          *// 释放readcount的互斥访问*

        V(w);              *// 释放写者优先锁*

*// 临界区 - 读取数据*

        perform\_read\_operation(i);

*// 退出区*

        P(mutex);          *// 获取readcount的互斥访问*

        readcount = readcount - 1;

        if (readcount == 0) { *// 如果是最后一个读者*

            V(rw);          *// 释放读写锁*

        }

        V(mutex);          *// 释放readcount的互斥访问*

*// 其他处理*

        do\_other\_things();

    }

}

*// 写者进程*

Process writer(int j) {

    while (true) {

        P(mutex\_w);        *// 获取writecount的互斥访问*

        writecount = writecount + 1;

        if (writecount == 1) { *// 如果是第一个写者*

            P(w);          *// 阻止后续读者进入*

        }

        V(mutex\_w);        *// 释放writecount的互斥访问*

        P(rw);             *// 获取读写锁*

*// 临界区 - 写入数据*

        perform\_write\_operation(j);

        V(rw);             *// 释放读写锁*

        P(mutex\_w);        *// 获取writecount的互斥访问*

        writecount = writecount - 1;

        if (writecount == 0) { *// 如果是最后一个写者*

            V(w);          *// 允许读者进入*

        }

        V(mutex\_w);        *// 释放writecount的互斥访问*

*// 其他处理*

        do\_other\_things();

    }

}

*// 主程序*

Process main() {

    cobegin

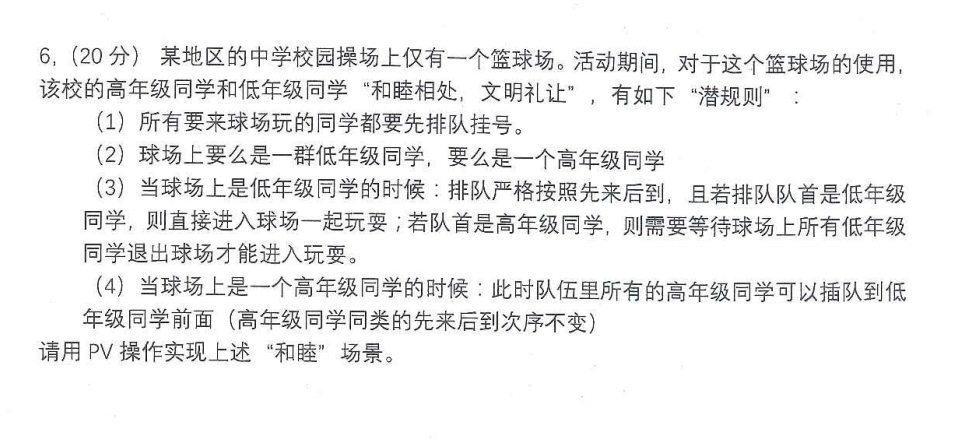
        reader(1); reader(2); reader(3); *// 启动多个读者进程*

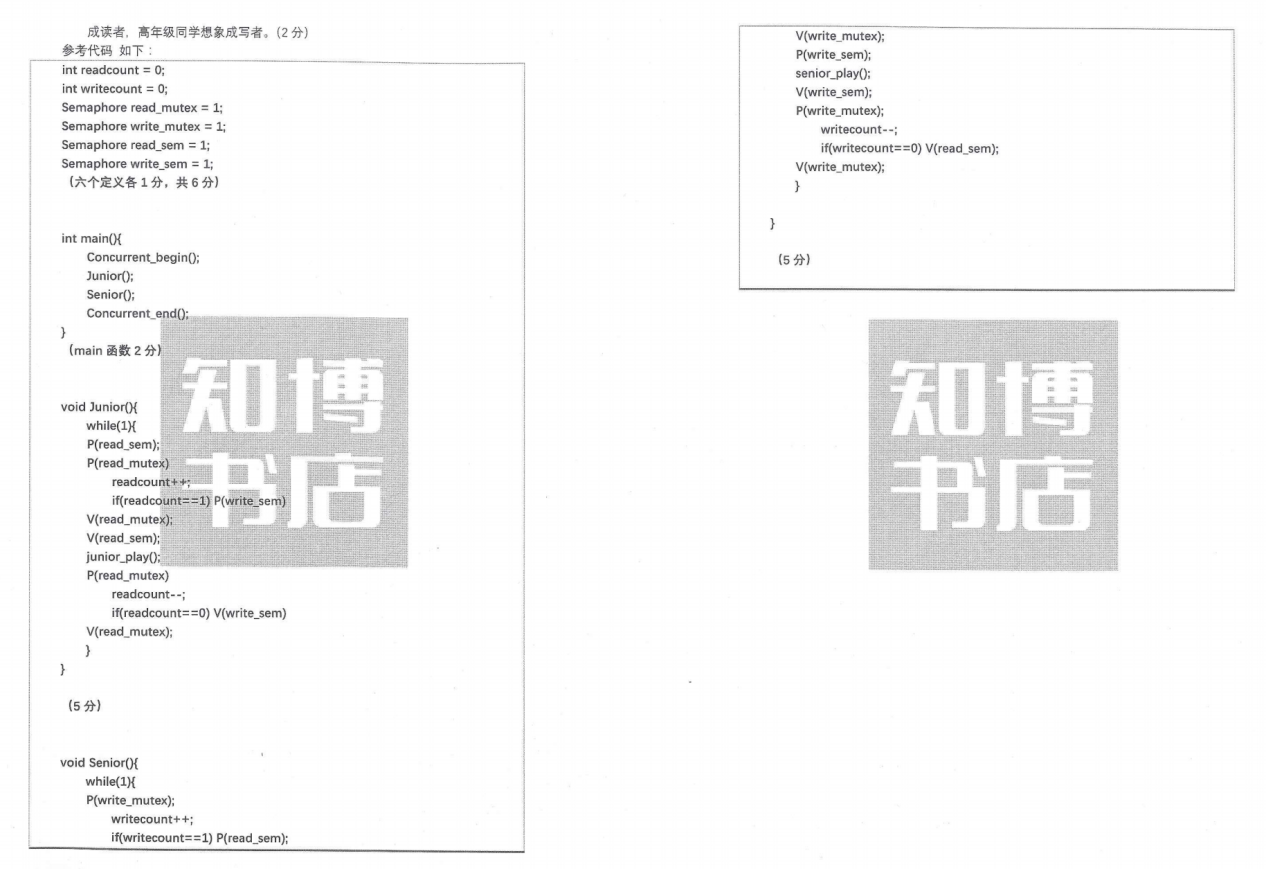
        writer(1); writer(2);            *// 启动多个写者进程*

    coend

}

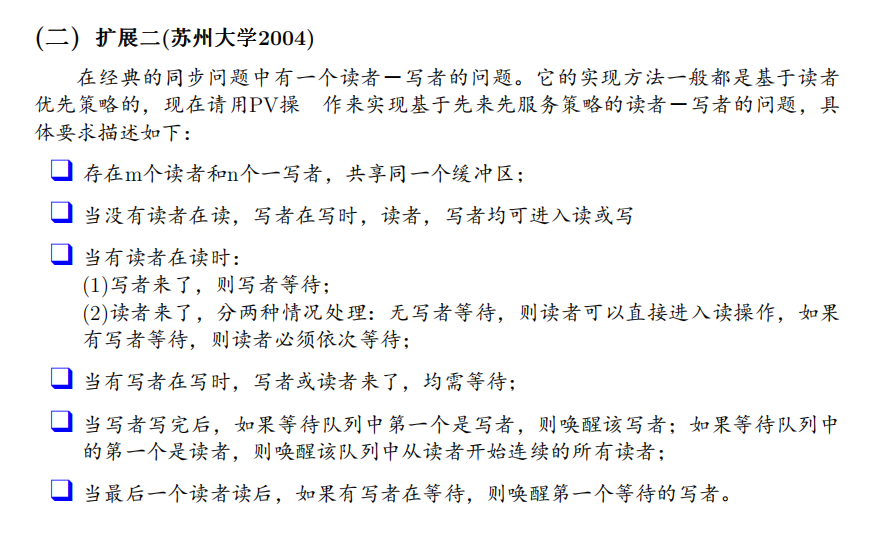
写者优先例题：





### ****读者-写者问题（限制最多 m 个读者）:****

****再原来的基础上弄一个semaphore capacity=m;PV一下即可。****



*// 信号量定义*

semaphore mutex = 1;      *// 保护共享变量和队列的互斥访问*

semaphore rw = 1;         *// 控制读写操作的互斥性*

semaphore order = 1;      *// 保证先来先服务的顺序*

int read\_count = 0;       *// 当前活跃的读者数量*

queue waiting\_queue;      *// 等待队列（记录读者/写者的到达顺序）*

*// 读者进程*

process Reader {

    P(order);            *// 申请进入顺序队列*

    P(mutex);            *// 保护 read\_count 和 waiting\_queue*

    if (read\_count == 0) { *// 如果是第一个读者*

        P(rw);           *// 尝试获取读写锁*

    }

    read\_count++;        *// 增加读者计数*

    V(mutex);            *// 释放 mutex*

    V(order);            *// 释放顺序锁*

*// 执行读操作*

    读取数据;

    P(mutex);            *// 保护 read\_count*

    read\_count--;        *// 减少读者计数*

    if (read\_count == 0) { *// 如果是最后一个读者*

*// 检查等待队列*

        if (!waiting\_queue.empty() && waiting\_queue.front() == WRITER) {

            V(rw);       *// 唤醒第一个写者*

        }

    }

    V(mutex);            *// 释放 mutex*

}

*// 写者进程*

process Writer {

    P(order);            *// 申请进入顺序队列*

    P(rw);               *// 申请读写锁（独占访问）*

    P(mutex);            *// 保护 waiting\_queue*

    waiting\_queue.push(WRITER); *// 加入等待队列*

    V(mutex);            *// 释放 mutex*

    V(order);            *// 释放顺序锁*

*// 执行写操作*

    写入数据;

    P(mutex);            *// 保护 waiting\_queue*

    waiting\_queue.pop(); *// 移除自己*

*// 检查等待队列*

    if (!waiting\_queue.empty()) {

        if (waiting\_queue.front() == READER) {

*// 唤醒连续的读者*

            while (!waiting\_queue.empty() && waiting\_queue.front() == READER) {

                V(rw);  *// 允许读者进入*

                waiting\_queue.pop();

            }

        } else {

            V(rw);       *// 唤醒第一个写者*

        }

    } else {

        V(rw);           *// 释放读写锁*

    }

    V(mutex);            *// 释放 mutex*

}

*// 主程序（启动读者和写者线程）*

cobegin

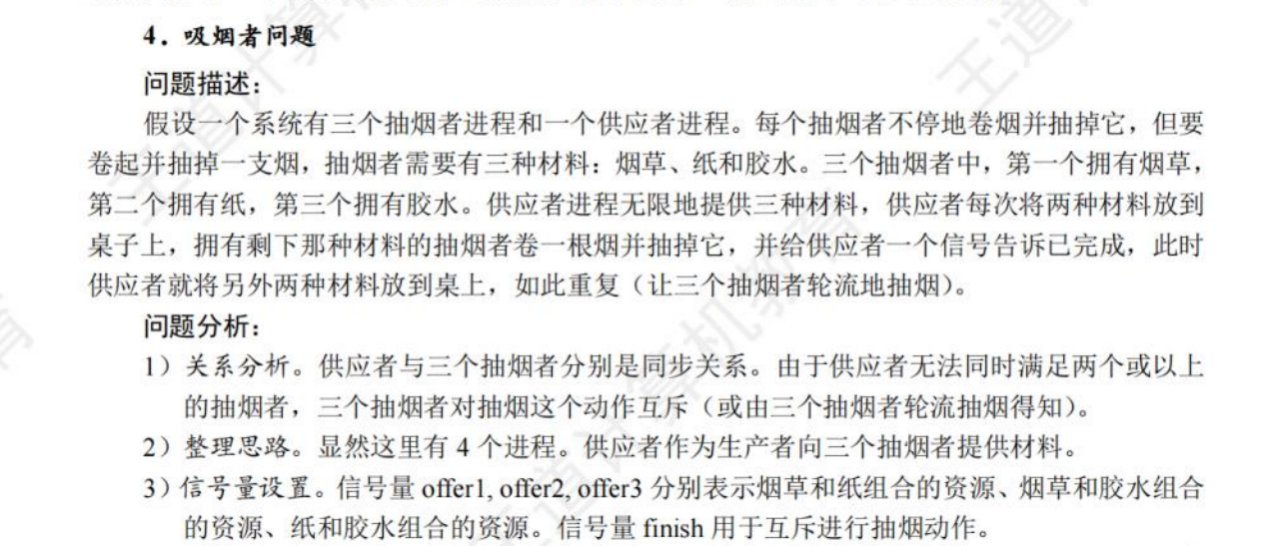
*// 启动 m 个读者和 n 个写者*

    for (int i = 0; i < m; i++) Reader();

    for (int i = 0; i < n; i++) Writer();

coend

吸烟者问题：



int num = 0;                *// 材料组合选择计数器*

semaphore offer1 = 0;       *// 信号量：烟草+纸就绪*

semaphore offer2 = 0;       *// 信号量：烟草+胶水就绪*

semaphore offer3 = 0;       *// 信号量：纸+胶水就绪*

semaphore finish = 0;       *// 信号量：抽烟完成同步*

*/\* 材料供应商进程 \*/*

process P1() {

    while (1) {

        num++;

        num = num % 3;      *// 循环选择0/1/2三种材料组合*

        if (num == 0)

            V(offer1);      *// 释放烟草和纸*

        else if (num == 1)

            V(offer2);      *// 释放烟草和胶水*

        else

            V(offer3);      *// 释放纸和胶水*

        放置材料到桌子；    *// 模拟材料放置动作*

        P(finish);         *// 等待吸烟完成*

    }

}

*/\* 吸烟者A（拥有烟草）\*/*

process P2() {

    while (1) {

        P(offer3);          *// 等待纸+胶水*

        拿到纸和胶水制作并抽烟；        *// 使用已有烟草+获取的材料*

        V(finish);         *// 通知供应商*

    }

}

*/\* 吸烟者B（拥有纸）\*/*

process P3() {

    while (1) {

        P(offer2);          *// 等待烟草+胶水*

        拿到烟草和胶水制作并抽烟；

        V(finish);

    }

}

*/\* 吸烟者C（拥有胶水）\*/*

process P4() {

    while (1) {

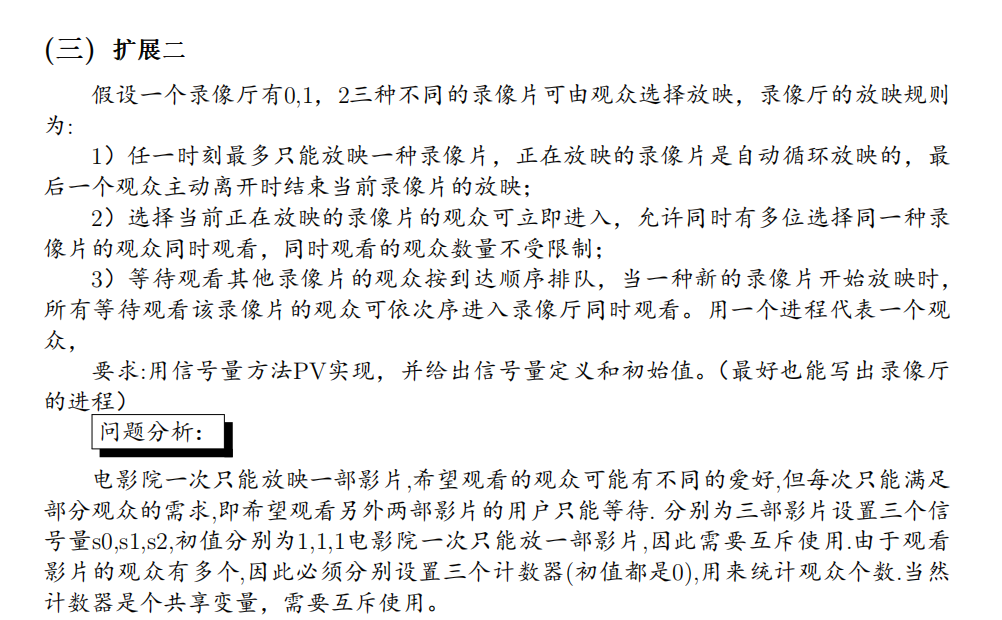
        P(offer1);          *// 等待烟草+纸*

        拿到烟草和纸制作并抽烟；

        V(finish);

    }

}



*// 信号量定义*

semaphore s = 1;       *// 控制录像厅互斥使用（初始值为1）*

semaphore s0 = 1;      *// 控制影片0的计数器访问（初始值为1）*

semaphore s1 = 1;      *// 控制影片1的计数器访问（初始值为1）*

semaphore s2 = 1;      *// 控制影片2的计数器访问（初始值为1）*

*// 计数器定义*

int count0 = 0;        *// 当前观看影片0的观众数（初始值为0）*

int count1 = 0;        *// 当前观看影片1的观众数（初始值为0）*

int count2 = 0;        *// 当前观看影片2的观众数（初始值为0）*

*// 观众进程（影片0）*

process Audience0 {

    while (true) {

        P(s0);                  *// 申请访问影片0计数器*

        count0++;               *// 增加影片0观众计数*

        if (count0 == 1) {      *// 如果是第一个观看影片0的观众*

            P(s);              *// 申请录像厅使用权*

        }

        V(s0);                  *// 释放影片0计数器*

*// 观看影片0*

        观看影片0;

        P(s0);                  *// 申请访问影片0计数器*

        count0--;               *// 减少影片0观众计数*

        if (count0 == 0) {      *// 如果是最后一个观看影片0的观众*

            V(s);              *// 释放录像厅使用权*

        }

        V(s0);                  *// 释放影片0计数器*

    }

}

*// 观众进程（影片1）*

process Audience1 {

    while (true) {

        P(s1);                  *// 申请访问影片1计数器*

        count1++;               *// 增加影片1观众计数*

        if (count1 == 1) {      *// 如果是第一个观看影片1的观众*

            P(s);              *// 申请录像厅使用权*

        }

        V(s1);                  *// 释放影片1计数器*

*// 观看影片1*

        观看影片1;

        P(s1);                  *// 申请访问影片1计数器*

        count1--;               *// 减少影片1观众计数*

        if (count1 == 0) {      *// 如果是最后一个观看影片1的观众*

            V(s);              *// 释放录像厅使用权*

        }

        V(s1);                  *// 释放影片1计数器*

    }

}

*// 观众进程（影片2）*

process Audience2 {

    while (true) {

        P(s2);                  *// 申请访问影片2计数器*

        count2++;               *// 增加影片2观众计数*

        if (count2 == 1) {      *// 如果是第一个观看影片2的观众*

            P(s);              *// 申请录像厅使用权*

        }

        V(s2);                  *// 释放影片2计数器*

*// 观看影片2*

        观看影片2;

        P(s2);                  *// 申请访问影片2计数器*

        count2--;               *// 减少影片2观众计数*

        if (count2 == 0) {      *// 如果是最后一个观看影片2的观众*

            V(s);              *// 释放录像厅使用权*

        }

        V(s2);                  *// 释放影片2计数器*

    }

}

*// 主程序*

int main() {

*// 启动录像厅管理进程和多个观众进程*

    cobegin

        VideoHall();

        Audience0();  *// 影片0观众*

        Audience1();  *// 影片1观众*

        Audience2();  *// 影片2观众*

    coend

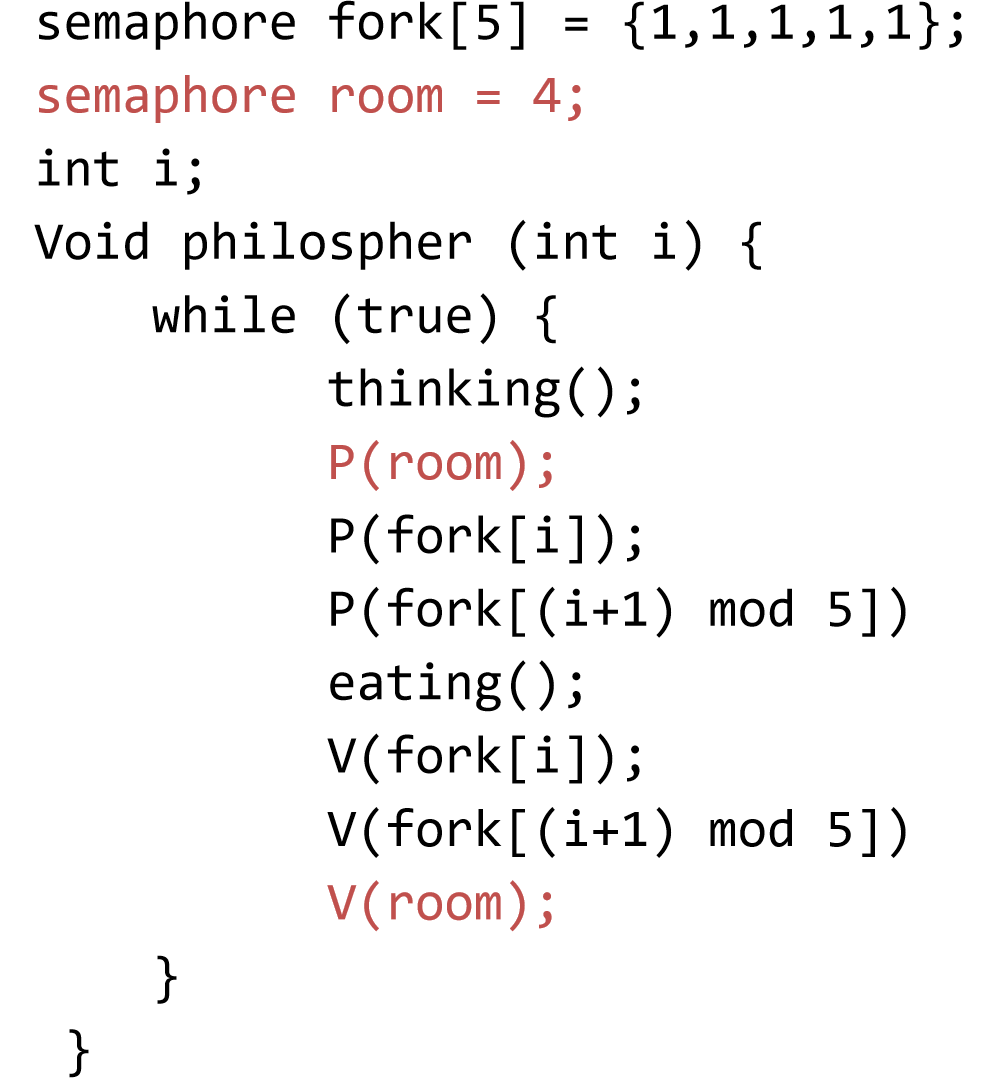
    return 0;

}

哲学家进餐问题

避免死锁的办法：

1.最多允许四个哲学家同时进食



2.序号为偶数，先右后左；奇数，先左后右。

semaphore chopstick[5] = {1,1,1,1,1}; *// 定义信号量数组 chopstick[5]，并初始化*

Pi() {    *// i 号哲学家的进程*

    do {

        if (i % 2 == 0) { *// 偶数标号哲学家*

            P(chopstick[(i+1)%5]);    *// 先拿右边筷子*

            P(chopstick[i]);          *// 再拿左边筷子*

        } else {          *// 奇数标号哲学家*

            P(chopstick[i]);          *// 先拿左边筷子*

            P(chopstick[(i+1)%5]);    *// 再拿右边筷子*

        }

        eat;                          *// 进餐*

        V(chopstick[i]);              *// 放回左边筷子*

        V(chopstick[(i+1)%5]);        *// 放回右边筷子*

        think;                        *// 思考*

    } while(1);

}

3.当哲学家左右两边的筷子都可用的时候，才允许拿筷子，要么不拿筷子。

semaphore chopstick[5] = {1,1,1,1,1}; *// 筷子信号量*

semaphore mutex = 1;                  *// 互斥信号量，控制拿筷子过程*

Pi() {    *// i 号哲学家的进程*

    do {

        P(mutex);                    *// 拿筷子前加锁（互斥）*

        P(chopstick[i]);             *// 拿左边筷子*

        P(chopstick[(i+1)%5]);       *// 拿右边筷子*

        V(mutex);                    *// 拿完筷子后解锁*

        eat;                         *// 进餐*

        V(chopstick[i]);             *// 放回左边筷子*

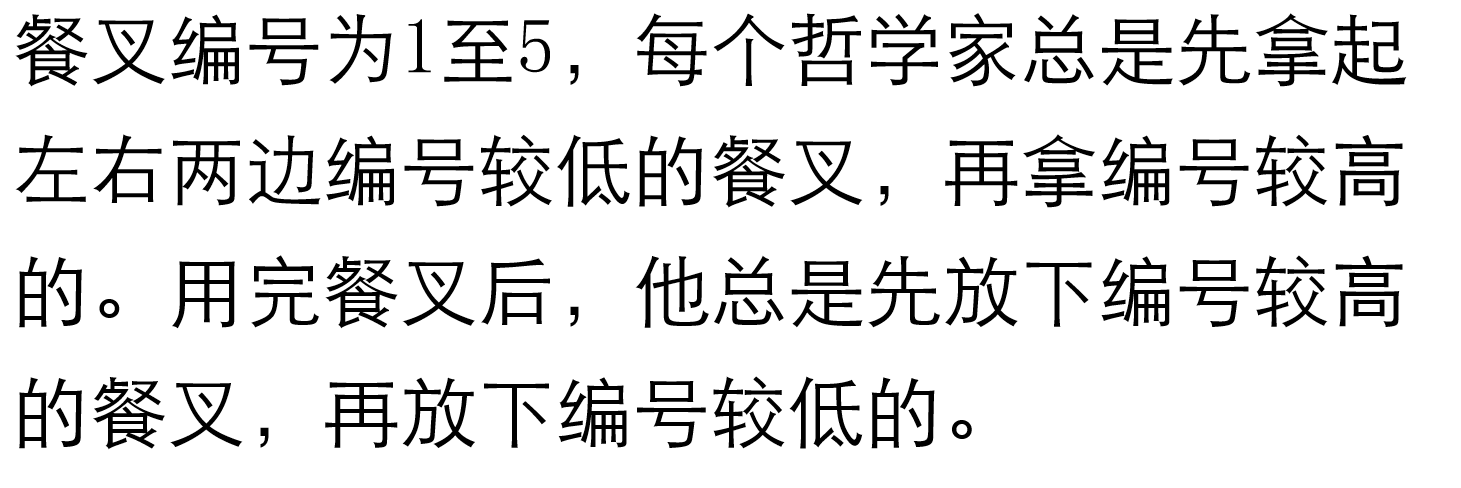
        V(chopstick[(i+1)%5]);        *// 放回右边筷子*

        think;                       *// 思考*

    } while(1);

}

4.



semaphore fork[6] = {1,1,1,1,1,1};  *// 餐叉 1~5，fork[0] 未使用*

Pi() {  *// i ∈ {1, 2, 3, 4, 5}*

    do {

        int left = i;

        int right = i % 5 + 1;  *// 关键修正：计算右餐叉*

*// 拿餐叉：先拿编号较低的*

        if (left < right) {

            P(fork[left]);

            P(fork[right]);

        } else {

            P(fork[right]);

            P(fork[left]);

        }

        eat;

*// 放餐叉：先放编号较高的*

        if (left > right) {

            V(fork[left]);

            V(fork[right]);

        } else {

            V(fork[right]);

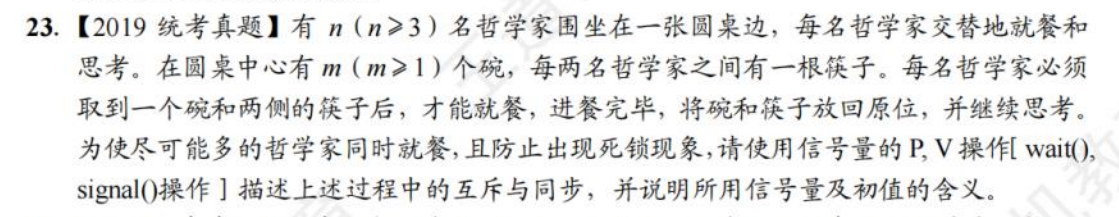
            V(fork[left]);

        }

        think;

    } while(1);

}





答案不是很容易想到，但是将碗作为普通资源，在原来的限制就餐人数、奇偶限制和添加一个mutex互斥锁基础上pv一下bowl也是可行的，只是会拿到70-80%的分数。

2. 何谓临界资源？若采用以下算法解决哲学家就餐问题，是否会导死锁？为什么？

semaphore fork[5] = {1, 1, 1, 1, 1};

void main()

{

cobegin {

philosopher(0);

philosopher(1);

philosopher(2);

philosopher(3);

philosopher(4);

} coend;

}

void philosopher(int i)

{

while(true) {

thinking();

if (i != 4) {

P(fork[i]);

P(fork[(i+1)%5]);

} else {

P(fork[(i+1)%5]);

P(fork[i]);

}

eating();

if (i != 4) {

V(fork[i]);

V(fork[(i+1)%5]);

} else {

V(fork[(i+1)%5]);

V(fork[i]);

}

}

}

答：所谓临界资源，是指必须互斥使用的资源。即在一段时间内，只允许一个进程使用的资源。（1分）

结论：该算法不会导致死锁。（3分）

原因：它破坏了产生死锁的四个充要条件之一——循环等待条件（3分）。上述算法实质上是对每个临界资源——餐刀进行编号，保证每个哲学家必须按照从小到大的次序依次申请资源，从而不会产生循环等待现象。（3分）

(更详细地说，加入0-3拿到了左边的筷子，4要先右后左，但是右已经被0号取走了，4被阻塞，拿不到左边的筷子，自然3号就能拿到右边的筷子，不会死锁)