操作系统实验 Lab4

一、基本信息：

姓名：黄云鹏

学号：181860036

邮箱：181860036@smail.nju.edu.cn

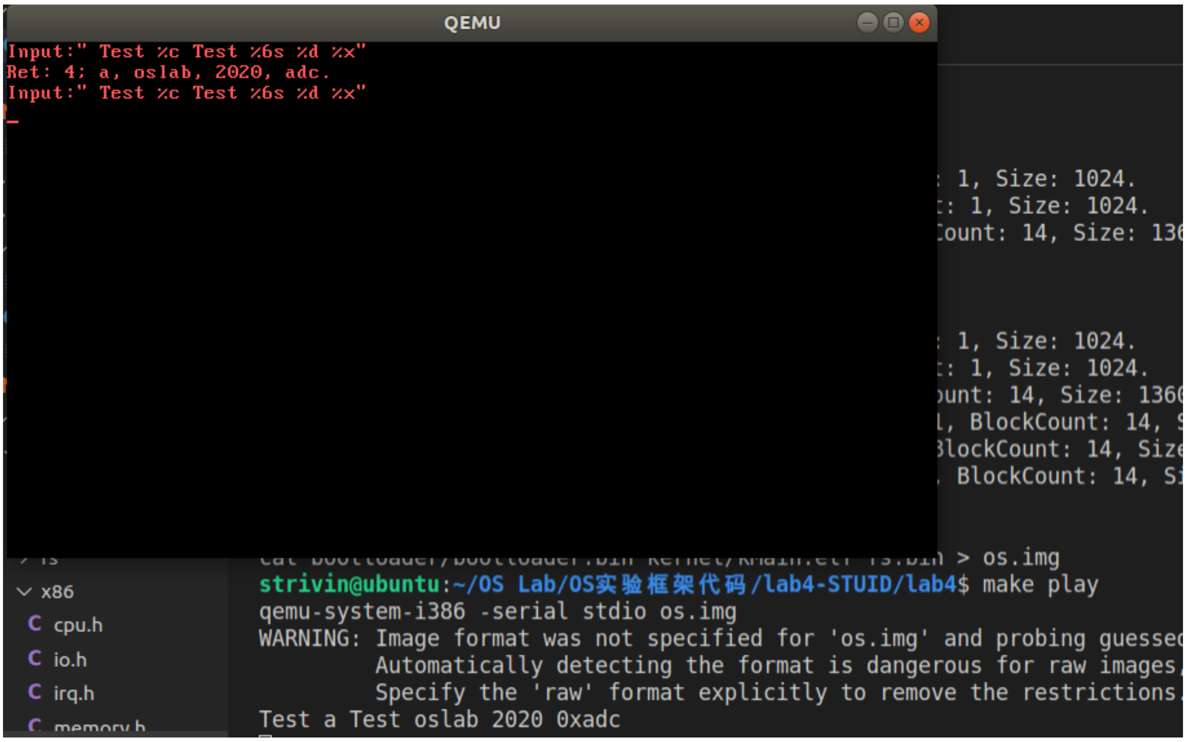
二、实验进度：

我完成了所有必做内容（除了随机函数的选做部分）

三、实验结果：

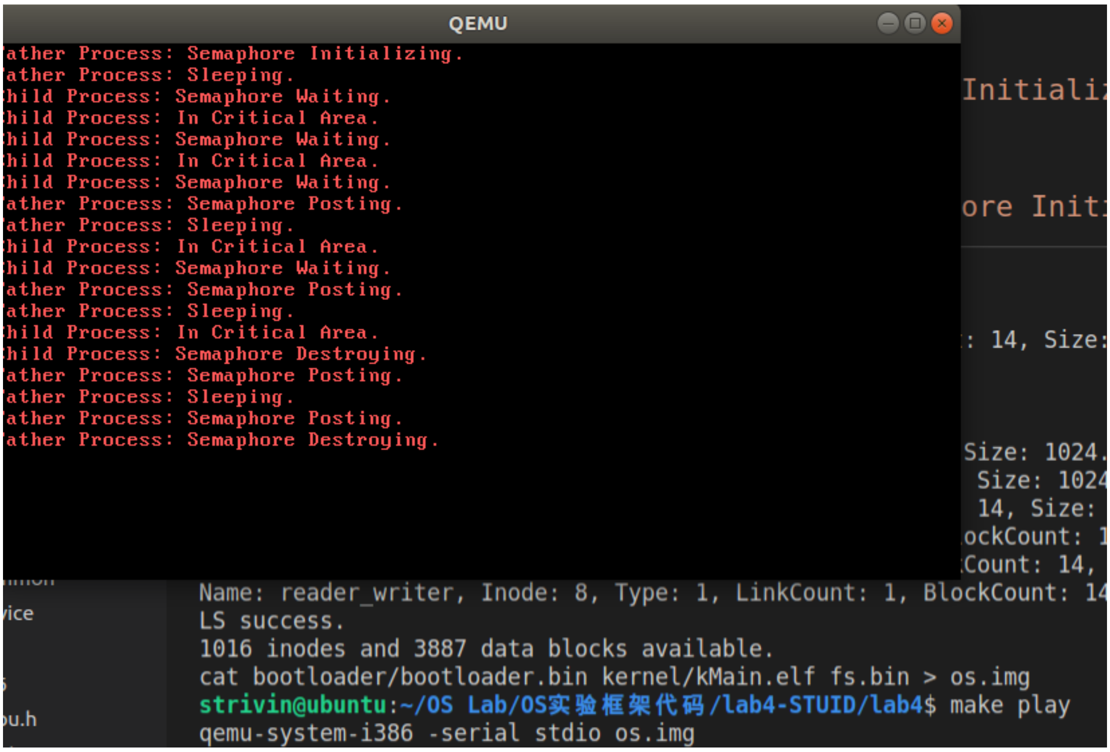
=>1. 完成了syscallReadStdIn和keyboardHandle之后，对scanf的测试样例/结果如下所示：

```c
int uEntry(void) {
    int dec = 0;
    int hex = 0;
    char str[6];
    char cha = 0;
    int ret = 0;
    while(1){
        printf("Input:\" Test %%c Test %%6s %%d %%x\"\n");
        ret = scanf(" Test %c Test %6s %d %x", &cha, str, &dec, &hex);
        printf("Ret: %d; %c, %s, %d, %x.\n", ret, cha, str, dec, hex);
        if (ret == 4)
            break;
    }
    return 0;
}
```



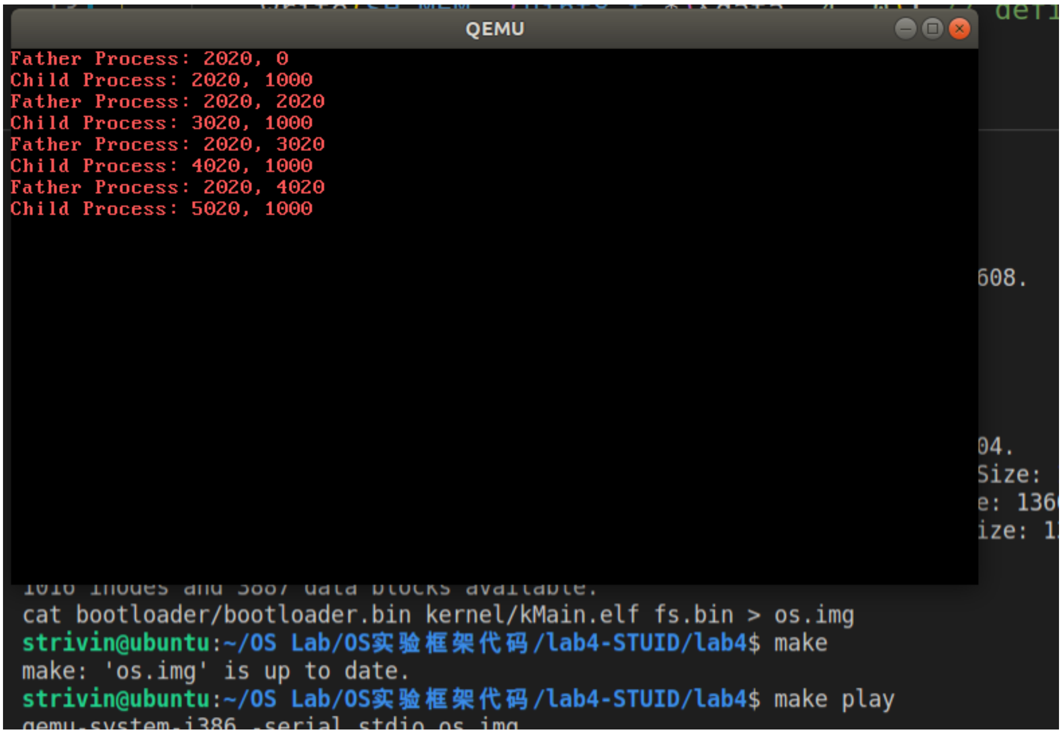
=>2. 完成了四个系统调用sem_init,sem_wait,sem_post,sem_destroy之后，测试样例/结果如下所示：

```c
int uEntry(void) {
    int i = 4;
    int ret = 0;
    int value = 2;
    sem_t sem;
    printf("Father Process: Semaphore Initializing.\n");
    ret = sem_init(&sem, value);
    if (ret == -1) {
        printf("Father Process: Semaphore Initializing Failed.\n");
        exit();
    }
    ret = fork();
    if (ret == 0) {
        while( i != 0) {
            i --;
            printf("Child Process: Semaphore Waiting.\n");
            sem_wait(&sem);
            printf("Child Process: In Critical Area.\n");
        }
        printf("Child Process: Semaphore Destroying.\n");
        sem_destroy(&sem);
        exit();
```

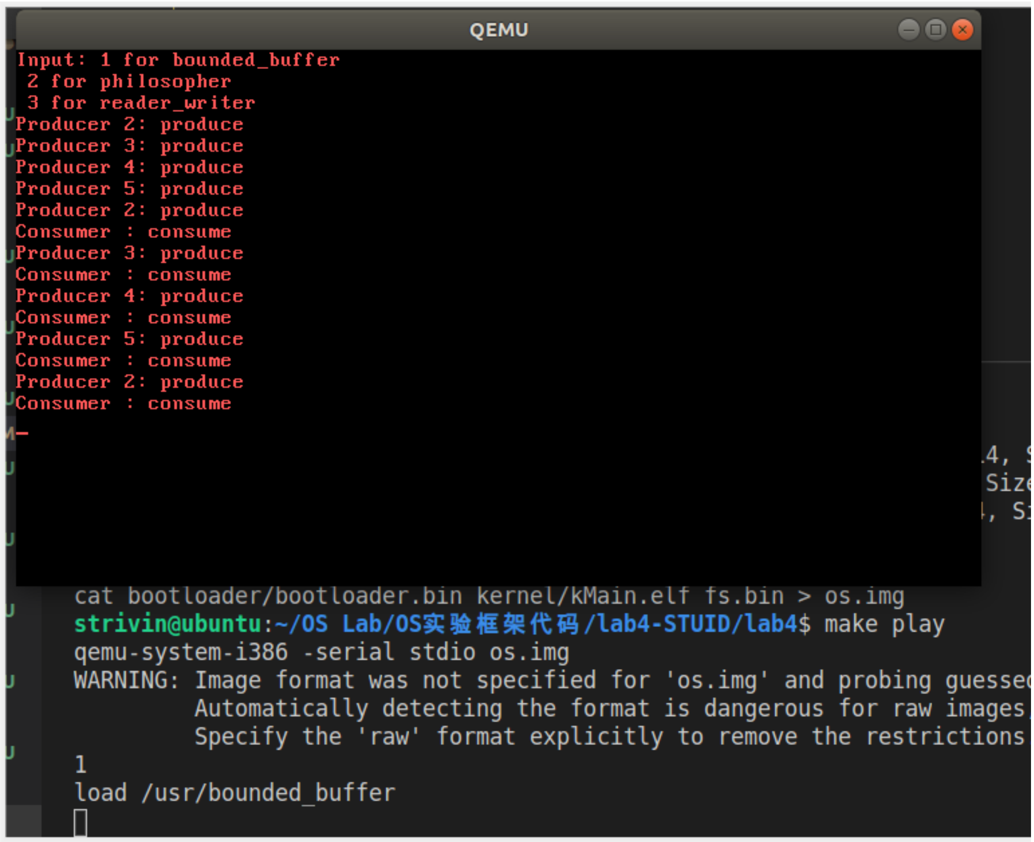


=>3. 完成了共享内存读写syscallReadShMem和syscallWriteShMem之后，测试样例/结果如下所示：

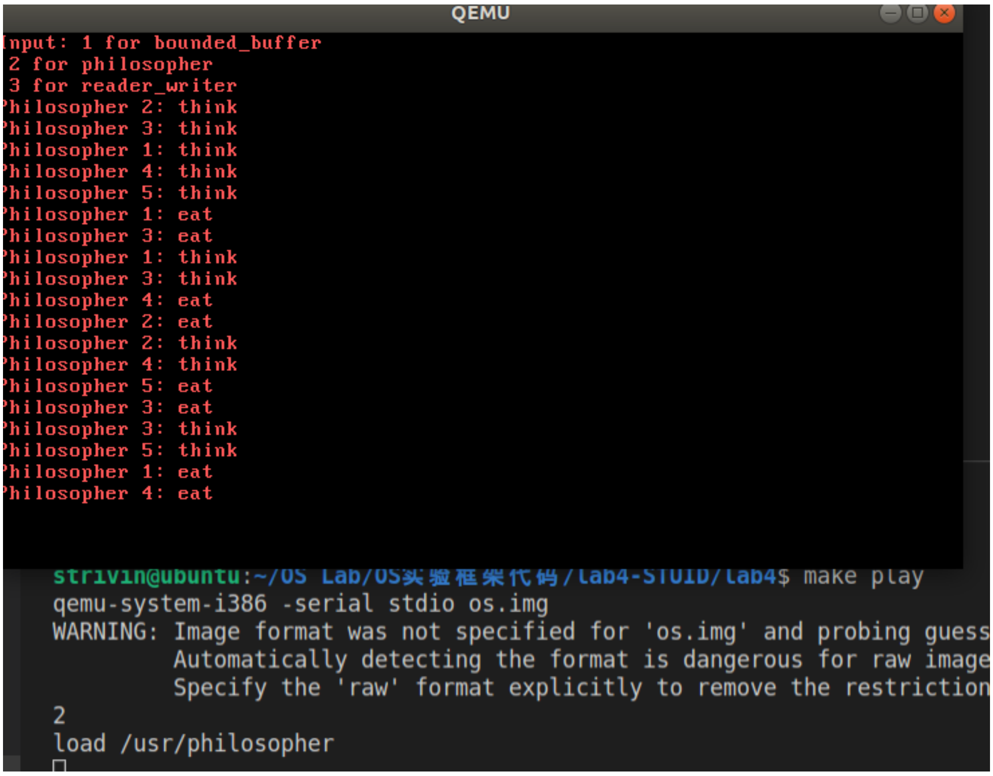
```
int uEntry(void) {
    int data = 2020;
    int data1 = 1000;
    int i = 4;
    int ret = fork();
    if (ret == 0) {
    while (i != 0) {
        i--;
        printf("Child Process: %d, %d\n", data, data1);
        write(SH_MEM, (uint8_t *)&data, 4, 0); // define SH_MEM 3
        data += data1;
         sleep(128);
        }
    exit();
    }       You, a few seconds ago • Uncommitted changes
     else if (ret != -1) {
        while (i != 0) {
            i--;
            read(SH_MEM, (uint8_t *)&data1, 4, 0);
            printf("Father Process: %d, %d\n", data, data1);
            sleep(128);
        }
```



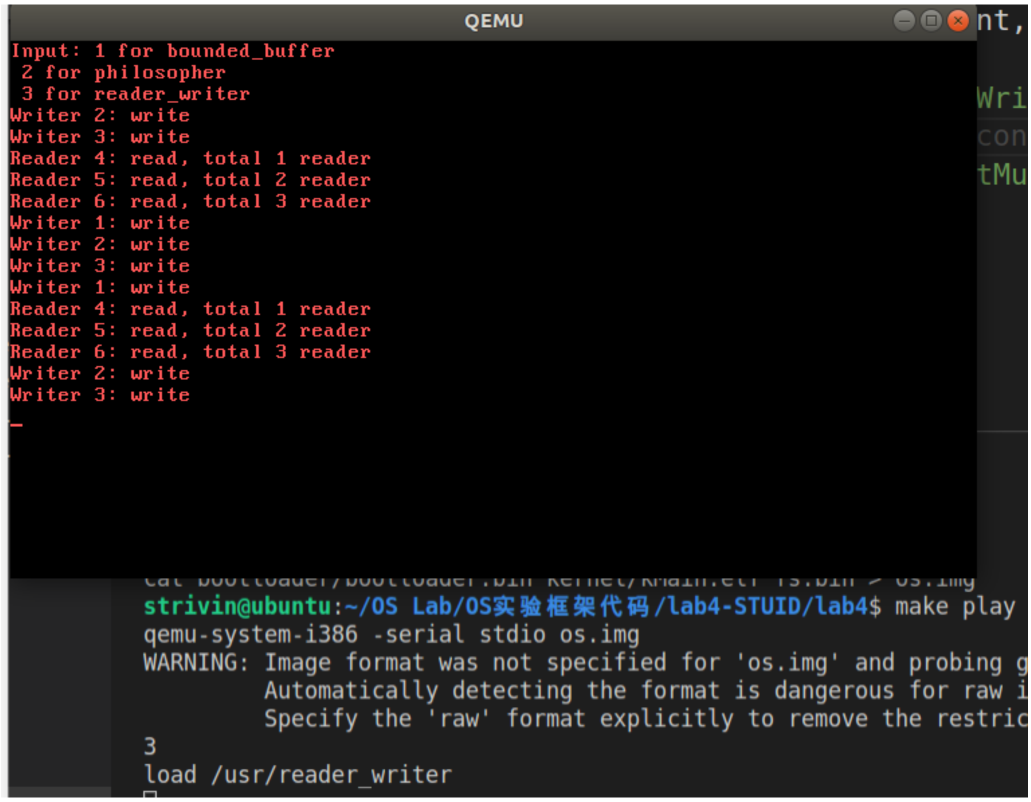
=>4. 解决了“生产者-消费者问题”之后，测试结果如下所示：



=>5. 解决了“哲学家就餐问题”之后，测试结果如下所示：



=>6. 解决了“读者写者问题”之后，测试结果如下所示：



四、修改代码：

1）在irqHandle.c中完成了syscallReadStdIn，实现了scanf对键盘设备缓冲区内容的读取功能：

```c
void syscallReadStdIn(struct TrapFrame *tf) {
    // TODO in lab4
    // if dev not busy, block itself
    if(dev[STD_IN].value == 0)
    {        You, 3 hours ago • done syscallReadStdIn
        dev[STD_IN].value -= 1;
        pcb[current].blocked.next = dev[STD_IN].pcb.next;
        pcb[current].blocked.prev = &(dev[STD_IN].pcb);
        dev[STD_IN].pcb.next = &(pcb[current].blocked);
        (pcb[current].blocked.next)->prev = &(pcb[current].blocked);

        pcb[current].state = STATE_BLOCKED;
        asm volatile("int $0x20");
```

```c
        // and read keybuffer to user str
        int sel = tf->ds;
        char *str = (char *)tf->edx;
        int readsize = (uint32_t)tf->ebx;
        int getsize = 0;
        char character;
        asm volatile("movw %0, %%es"::"m"(sel));
        for(; getsize < readsize-1 ;)
        {
            if(bufferHead != bufferTail)
            {
                character = getChar(keyBuffer[bufferHead]);
                putChar(character);
                if(character!=0)
                {
                    asm volatile("movb %0, %%es:(%1)"::"r"(character),"r"(str + getsize));
                    getsize++;
                }
                bufferHead += 1;
                bufferHead %= MAX_KEYBUFFER_SIZE;
            }
            else
                break;
        }
```

```c
        // to add '/0' to the end of str
        asm volatile("movb $0x00, %%es:(%0)"::"r"(str+getsize));
        // return the gotten size
        tf->eax = getsize;
    }
    // if dev busy, return -1
    else if(dev[STD_IN].value < 0)
    {
        tf->eax = -1;
    }
    return;
}
```

2）在irgHandle.c中完成了keyboardHandle，实现了键盘缓冲区对键盘码的存储功能：

```c
void keyboardHandle(struct TrapFrame *tf) {
    // TODO in lab4
    // read keycode from keyborad
    uint32_t keycode = getKeyCode();
    if(keycode != 0)
    {
        keyBuffer[bufferTail++] = keycode;
        bufferTail %= MAX_KEYBUFFER_SIZE;
        // if dev was busy, release the waitting process
        if(dev[STD_IN].value < 0)
        {
            dev[STD_IN].value += 1;
            ProcessTable* pt = (ProcessTable*)((uint32_t)(dev[STD_IN].pcb.prev)
            (uint32_t)&(((ProcessTable*)0)->blocked));
            dev[STD_IN].pcb.prev = (dev[STD_IN].pcb.prev)->prev;
            (dev[STD_IN].pcb.prev)->next = &(dev[STD_IN].pcb);
            pt->state = STATE_RUNNABLE;
        }
    }
    return;
```

3）在irqHandle.c中完成了syscallReadShMem和syscallWriteShMem，实现了对共享内存的读写功能：

```c
void syscallReadShMem(struct TrapFrame *tf) {
    // TODO in lab4
    // copy ShMem to str
    int sel = tf->ds;
    int size = tf->ebx;
    int index = tf->esi;
    char *str = (char *)tf->edx;
    char character;
    asm volatile("movw %0, %%es"::"m"(sel));

    for(int i=0;i<size;i++)
    {
        character = shMem[index+i];
        asm volatile("movb %0, %%es:(%1)"::"r"(character),"r"(str+i));
    }

    return;
}
```

```c
void syscallWriteShMem(struct TrapFrame *tf) {
    // TODO in lab4
    // copy str to ShMem
    int sel = tf->ds;
    int size = tf->ebx;
    int index = tf->esi;
    char *str = (char *)tf->edx;
    char character;
    asm volatile("movw %0, %%es"::"m"(sel));

    for(int i=0;i<size;i++)
    {
        asm volatile("movb %%es:(%1), %0":"=r"(character):"r"(str + i));
        shMem[index+i] = character;
    }

    return;
}
```

4）在irqHandle.c中完成了syscallSemInit,syscallSemWait,syscallSemPost,syscallSemDestroy，实现了信号量的创建、使用和销毁：

```c
void syscallSemInit(struct TrapFrame *tf) {
    // TODO in lab4
    // find an empty Semaphore in sem
    int find = -1;
    for(int i=0;i<MAX_SEM_NUM;i++)
    {
        if(sem[i].state == 0)
        {
            find = i;
            break;
        }
    }

    // if not found, return -1
    if(find == -1)
    {
        tf->eax = -1;  return;
    }

    // if found, init sem[find] and return 0
    sem[find].state = 1;
    sem[find].value = (int32_t)tf->edx;
    sem[find].pcb.next = &(sem[find].pcb);
    sem[find].pcb.prev = &(sem[find].pcb);
    tf->eax = find;
    return;
```

```c
void syscallSemWait(struct TrapFrame *tf) {
    // TODO in lab4
    int i = (uint32_t)tf->edx;
    // decrease sem[i] and block itself if sem[i] busy
    if(i>=0 && i< MAX_SEM_NUM)
    {
        sem[i].value -= 1;
        if(sem[i].value <0)
        {
            pcb[current].blocked.next = sem[i].pcb.next;
            pcb[current].blocked.prev = &(sem[i].pcb);
            sem[i].pcb.next = &(pcb[current].blocked);
            (pcb[current].blocked.next)->prev = &(pcb[current].blocked);
            pcb[current].state = STATE_BLOCKED;
            asm volatile("int $0x20");
        }
        tf->eax = 0;
    }
    else
        tf->eax = -1;
    return;
}
```

```c
void syscallSemPost(struct TrapFrame *tf) {
    // TODO in lab4
    int i = (uint32_t)tf->edx;
    // increase sem[i] and release one pcb if sem[i] busy
    if(i>=0 && i< MAX_SEM_NUM)
    {
        sem[i].value += 1;
        if(sem[i].value <= 0)
        {
            ProcessTable* pt = (ProcessTable*)((uint32_t)(sem[i].pcb.prev)
            (uint32_t)&(((ProcessTable*)0)->blocked));
            sem[i].pcb.prev = (sem[i].pcb.prev)->prev;
            (sem[i].pcb.prev)->next = &(sem[i].pcb);
            pt->state = STATE_RUNNABLE;
        }
        tf->eax = 0;
    }
    else
        tf->eax = -1;
    return;
}
```

```c
void syscallSemDestroy(struct TrapFrame *tf) {
    // TODO in lab4
    // destroy the sem
    int i = (uint32_t)tf->edx;
    if(i>=0 && i< MAX_SEM_NUM)
    {
        if(sem[i].state == 1)
        {
            sem[i].state = 0;
            tf->eax = 0;
            return;
        }
    }
    tf->eax = -1;
    return;
}
```

5）在bounded_buffer中，完成了main.c，解决了4个生产者，1个消费者，大小为6的有效缓冲区的进程互斥和同步问题：

```c
int main(void) {
    // TODO in lab4
    // init sems
    sem_t mutex, full,empty;
    sem_init(&mutex,1);
    sem_init(&full,0);
    sem_init(&empty,6);

    for(int i=0;i<4;i++)
    {
        int ret = fork();
        if(!ret)  // child
            break;
        else if(ret == -1) // fork failed
            exit();
        else  // father
            continue;
    }
```

```c
int pid = getpid();
// father as consumer
if(pid == 1)
{
    while(1)
    {
        sem_wait(&full);  // fullbuffers->P()
        sem_wait(&mutex); // mutex->P()
        printf("Consumer : consume\n"); // remove c from buffer
        sleep(128);
        sem_post(&mutex); // mutex->V()
        sem_post(&empty); // emptybuffers->V()
    }
}
```

```c
// children as producers
else if(pid<=5)
{
    while(1)
    {
        sem_wait(&empty);  // emptybuffers->P()
        sem_wait(&mutex); // mutex->P()
        printf("Producer %d: produce\n", pid); // add c to buffer
        sleep(128);
        sem_post(&mutex); // mutex->V()
        sem_post(&full); // fullbuffers->V()
    }
}

// destroy sems
sem_destroy(&mutex);
sem_destroy(&full);
sem_destroy(&empty);
exit();      You, 4 hours ago • add lab4
return 0;
```

6）在philosopher中，完成了main.c，解决了5个哲学家，5把叉子的进程互斥和同步问题：

```c
int main(void) {
    // TODO in lab4
    // init forks
    sem_t forks[5];
    sem_init(&forks[0],1);
    sem_init(&forks[1],1);
    sem_init(&forks[2],1);
    sem_init(&forks[3],1);
    sem_init(&forks[4],1);

    for(int i=0;i<4;i++)
    {
        int ret = fork();
        if(!ret)  // child
            break;
        else if(ret == -1) // fork failed
            exit();
        else  // father
            continue;
    }

    int pid = getpid();
    // think and eat
```

```c
    // think and eat
    while(1)
    {
        printf("Philosopher %d: think\n", pid);  // think
        sleep(128);
        if(pid%2==0)
        {
            sem_wait(&forks[pid]);          //P(fork[i]);
            sem_wait(&forks[(pid+1)%5]);  //P(fork[(i+1)%N]);
        }
        else
        {
            sem_wait(&forks[(pid+1)%5]);  //P(fork[(i+1)%N]);
            sem_wait(&forks[pid]);          //P(fork[i]);
        }
        printf("Philosopher %d: eat\n", pid); //eat
        sleep(128);      You, 11 minutes ago • Uncommitted changes
        sem_post(&forks[pid]);          //V(fork[i]);
        sem_post(&forks[(pid+1)%5]);  //V(fork[(i+1)%N]);
    }

    // destroy forks
    sem_destroy(&forks[0]);
    sem_destroy(&forks[1]);
    sem_destroy(&forks[2]);
    sem_destroy(&forks[3]);
    sem_destroy(&forks[4]);
    exit();
    return 0;
```

7）在reader_writer中，完成了main.c，解决了3个读者，3个写者的进程互斥和同步问题：

```c
int main(void) {
    // TODO in lab4
    // init mutex
    sem_t writemutex,countmutex;
    sem_init(&writemutex,1);
    sem_init(&countmutex,1);
    int Rcount = 0 ;

    for(int i=0;i<5;i++)
    {
        int ret = fork();
        if(!ret)  // child
            break;
        else if(ret == -1) // fork failed
            exit();
        else  // father
            continue;
    }

    int pid = getpid();
```

```c
    // writer
    if(pid<=3)
    {
        while(1)
        {
            sem_wait(&writemutex);//P(WriteMutex);
            printf("Writer %d: write\n", pid);//write;
            sleep(128);
            sem_post(&writemutex);//V(WriteMutex);
        }
```

```
// reader
else if(pid<=6)
{
    while(1)
    {
        sem_wait(&countmutex);//P(CountMutex);
        read(SH_MEM, (uint8_t *)&Rcount, 4,0);
        if (Rcount == 0)
            sem_wait(&writemutex);//P(WriteMutex);
        ++Rcount;
        write(SH_MEM, (uint8_t *)&Rcount, 4,0);
        sem_post(&countmutex);//V(CountMutex);
        printf("Reader %d: read, total %d reader\n", pid, Rcount);//reac
        sleep(128);
        // after read
        sem_wait(&countmutex);//P(CountMutex);
        read(SH_MEM, (uint8_t *)&Rcount, 4,0);
        --Rcount;
        write(SH_MEM, (uint8_t *)&Rcount, 4,0);
        if (Rcount == 0)
            sem_post(&writemutex);//V(WriteMutex);
        sleep(128);        You, a few seconds ago • Uncommitted changes
        sem_post(&countmutex);//V(CountMutex);
    }
}

// destroy mutexs
sem_destroy(&writemutex);
sem_destroy(&countmutex);
exit();
```

五、实验思考题：

1. 对于生产者-消费者问题，PV操作的操作次序有影响吗？

   答：有影响，临界区互斥量的PV操作一定要在条件条件同步互斥量的PV操作之间，否则可能会出现死锁现象；

2. 对于哲学家就餐问题，有比奇数号哲学家与偶数号哲学家拿左右叉子的顺序不同更好的就餐方式吗？

   答：有其他方案，但不能保证一定比该方案更好，比如至多允许4位哲学家同时就餐 或 每位哲学家拿到两把叉子才开始吃，否则一把都不拿；

六、实验心得：

=> 加深了对信号量实现互斥访问和条件同步的机制的理解

=> 对进程互斥和通信的几个经典问题有了更深入的思考

=> 简要了解了共享内存的实现机制