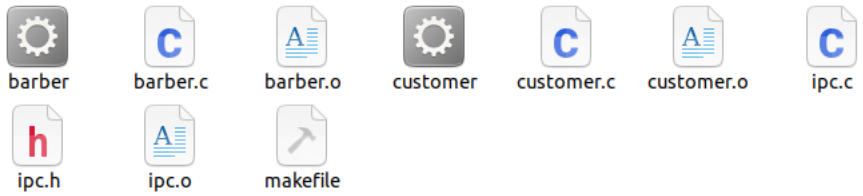
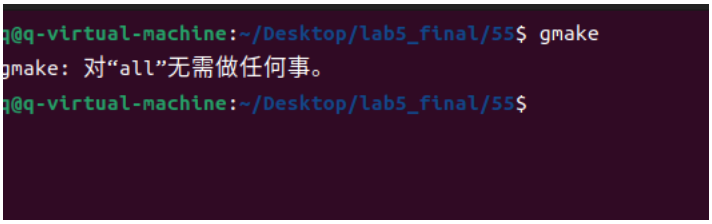
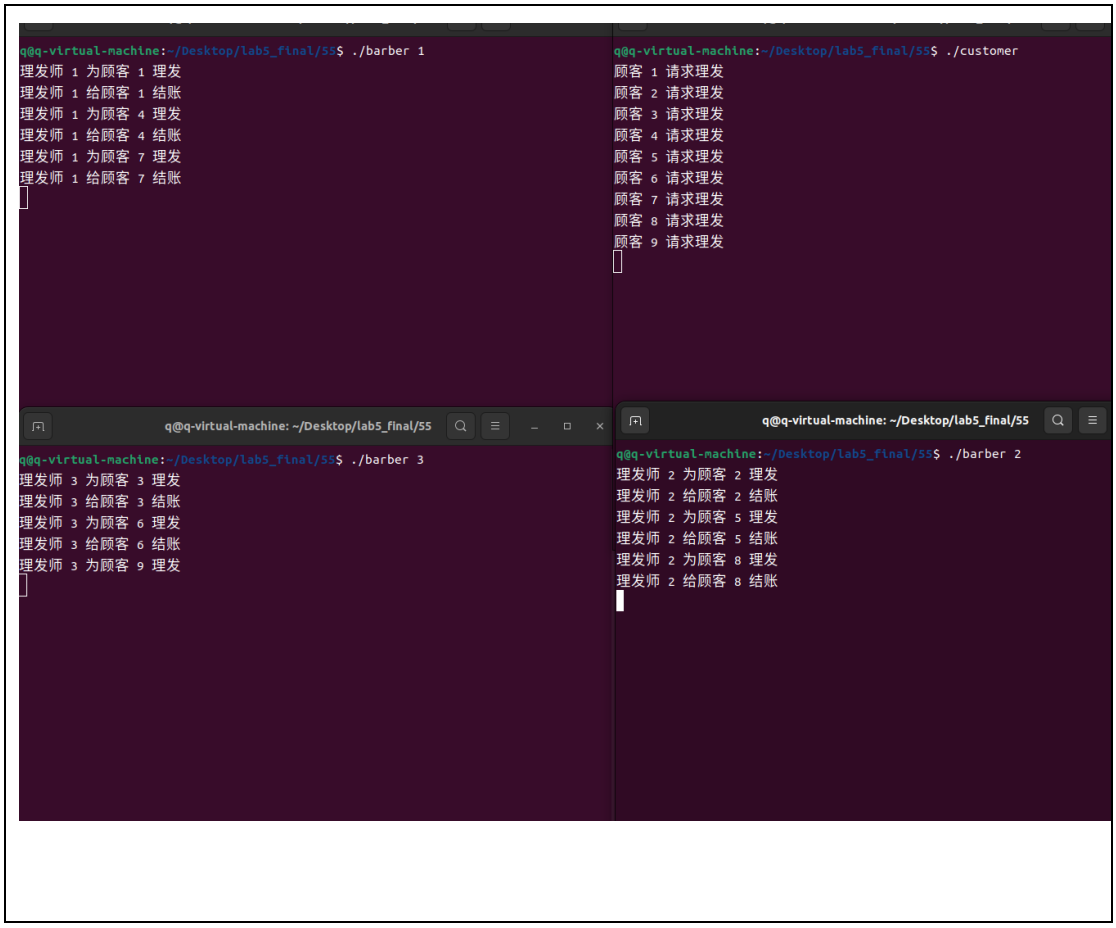
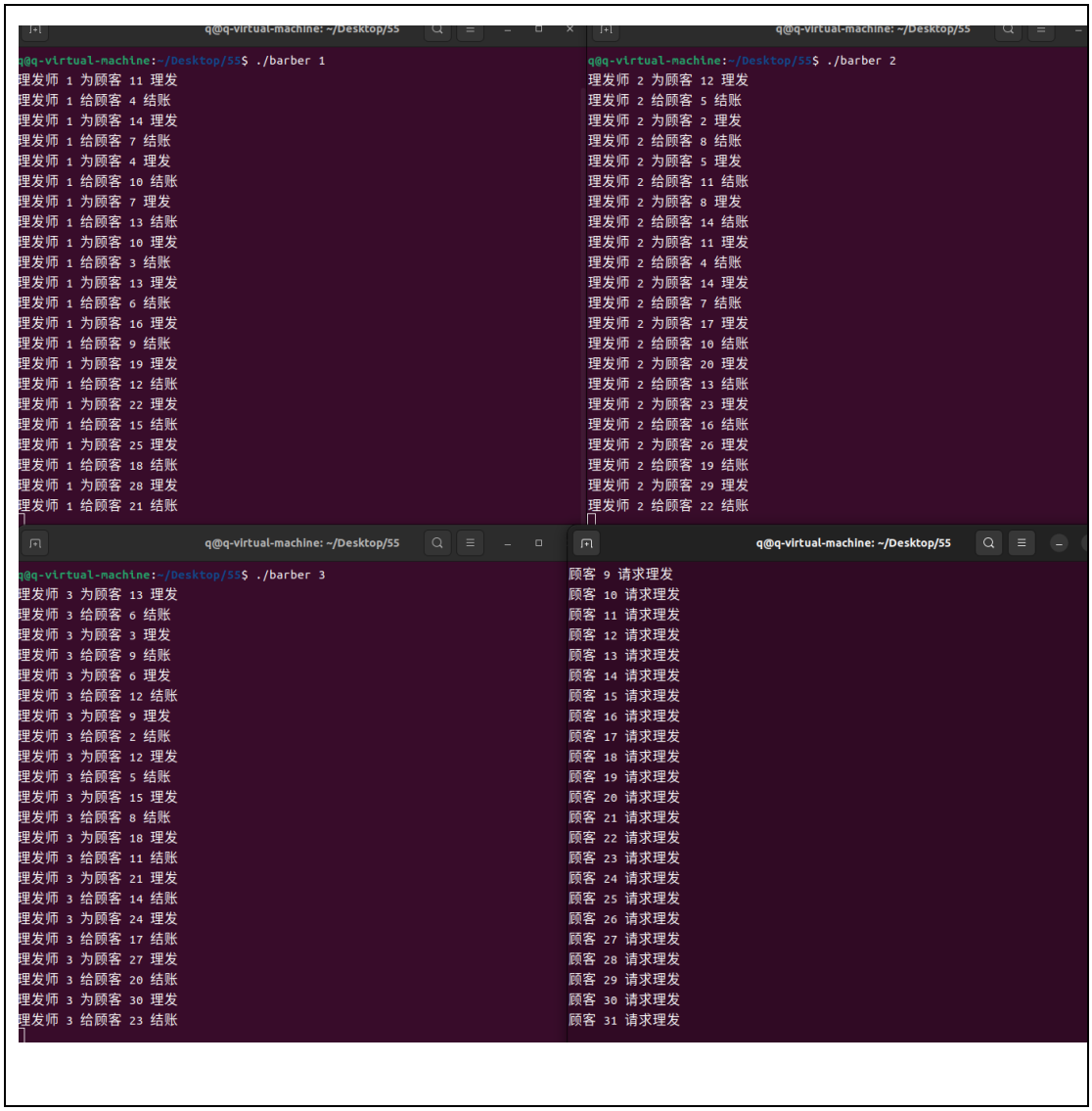
山东大学 计算机科学与技术 学院
操作系统 课程实验报告

学号：202200130223	姓名：李洋永	班级：4 班
实验题目：实验 5 进程互斥		
实验学时：2	实验日期：2024.11.6	
实验目的：进一步研究和实践操作系统中关于并发进程同步与互斥操作的一些经典问题的 解法，加深对于非对称性互斥问题有关概念的理解。观察和体验非对称性互斥问题 的并发控制方法。进一步了解 Linux 系统中 IPC 进程同步工具的用法，训练解决对 该类问题的实际编程、调试和分析问题的能力。）		
实验环境： Ubuntu		
源程序清单： 		
编译及运行结果： 		


q@q-virtual-machine:~/Desktop/lab5_final/55$ ./barber 1
理发师 1 为顾客 1 理发
理发师 1 给顾客 1 结账
理发师 1 为顾客 4 理发
理发师 1 给顾客 4 结账
理发师 1 为顾客 7 理发
理发师 1 给顾客 7 结账
q@q-virtual-machine:~/Desktop/lab5_final/55$ ./customer
顾客 1 请求理发
顾客 2 请求理发
顾客 3 请求理发
顾客 4 请求理发
顾客 5 请求理发
顾客 6 请求理发
顾客 7 请求理发
顾客 8 请求理发
顾客 9 请求理发
q@q-virtual-machine: ~/Desktop/lab5_final/55
q@q-virtual-machine:~/Desktop/lab5_final/55$ ./barber 3
理发师 3 为顾客 3 理发
理发师 3 给顾客 3 结账
理发师 3 为顾客 6 理发
理发师 3 给顾客 6 结账
理发师 3 为顾客 9 理发
q@q-virtual-machine: ~/Desktop/lab5_final/55
q@q-virtual-machine:~/Desktop/lab5_final/55$ ./barber 2
理发师 2 为顾客 2 理发
理发师 2 给顾客 2 结账
理发师 2 为顾客 5 理发
理发师 2 给顾客 5 结账
理发师 2 为顾客 8 理发
理发师 2 给顾客 8 结账


q@q-virtual-machine: ~/Desktop/55
q@q-virtual-machine:~/Desktop/55$ ./barber 1
理发师 1 为顾客 11 理发
理发师 1 给顾客 4 结账
理发师 1 为顾客 14 理发
理发师 1 给顾客 7 结账
理发师 1 为顾客 4 理发
理发师 1 给顾客 10 结账
理发师 1 为顾客 7 理发
理发师 1 给顾客 13 结账
理发师 1 为顾客 10 理发
理发师 1 给顾客 3 结账
理发师 1 为顾客 13 理发
理发师 1 给顾客 6 结账
理发师 1 为顾客 16 理发
理发师 1 给顾客 9 结账
理发师 1 为顾客 19 理发
理发师 1 给顾客 12 结账
理发师 1 为顾客 22 理发
理发师 1 给顾客 15 结账
理发师 1 为顾客 25 理发
理发师 1 给顾客 18 结账
理发师 1 为顾客 28 理发
理发师 1 给顾客 21 结账
q@q-virtual-machine: ~/Desktop/55
q@q-virtual-machine:~/Desktop/55$ ./barber 2
理发师 2 为顾客 12 理发
理发师 2 给顾客 5 结账
理发师 2 为顾客 2 理发
理发师 2 给顾客 8 结账
理发师 2 为顾客 5 理发
理发师 2 给顾客 11 结账
理发师 2 为顾客 8 理发
理发师 2 给顾客 14 结账
理发师 2 为顾客 11 理发
理发师 2 给顾客 4 结账
理发师 2 为顾客 14 理发
理发师 2 给顾客 7 结账
理发师 2 为顾客 17 理发
理发师 2 给顾客 10 结账
理发师 2 为顾客 20 理发
理发师 2 给顾客 13 结账
理发师 2 为顾客 23 理发
理发师 2 给顾客 16 结账
理发师 2 为顾客 26 理发
理发师 2 给顾客 19 结账
理发师 2 为顾客 29 理发
理发师 2 给顾客 22 结账
q@q-virtual-machine: ~/Desktop/55
q@q-virtual-machine:~/Desktop/55$ ./barber 3
理发师 3 为顾客 13 理发
理发师 3 给顾客 6 结账
理发师 3 为顾客 3 理发
理发师 3 给顾客 9 结账
理发师 3 为顾客 6 理发
理发师 3 给顾客 12 结账
理发师 3 为顾客 9 理发
理发师 3 给顾客 2 结账
理发师 3 为顾客 12 理发
理发师 3 给顾客 5 结账
理发师 3 为顾客 15 理发
理发师 3 给顾客 8 结账
理发师 3 为顾客 18 理发
理发师 3 给顾客 11 结账
理发师 3 为顾客 21 理发
理发师 3 给顾客 14 结账
理发师 3 为顾客 24 理发
理发师 3 给顾客 17 结账
理发师 3 为顾客 27 理发
理发师 3 给顾客 20 结账
理发师 3 为顾客 30 理发
理发师 3 给顾客 23 结账
q@q-virtual-machine: ~/Desktop/55
顾客 9 请求理发
顾客 10 请求理发
顾客 11 请求理发
顾客 12 请求理发
顾客 13 请求理发
顾客 14 请求理发
顾客 15 请求理发
顾客 16 请求理发
顾客 17 请求理发
顾客 18 请求理发
顾客 19 请求理发
顾客 20 请求理发
顾客 21 请求理发
顾客 22 请求理发
顾客 23 请求理发
顾客 24 请求理发
顾客 25 请求理发
顾客 26 请求理发
顾客 27 请求理发
顾客 28 请求理发
顾客 29 请求理发
顾客 30 请求理发
顾客 31 请求理发

```
while (1) {
    sleep(rate);
    id++;
    msg_arg.mid = id;
    msg_arg.mtype = id + 1;
    if (sofa > 0) {//沙发有空位
        if (room == 13) {//等候室没人，直接进入沙发
            sofa--;
        } else if (room > 0) {//进入等候室等候
            room--;
            msg_arg.mtype = SOFAQUEST;//请求沙发
            msgsnd(sofa_quest_id, &msg_arg, sizeof(msg_arg), 0);
        } else {//等候室满
            printf("理发店满员，顾客%d 离开\n", id);
        }
    }
    else if (room > 0){//沙发满了,等待室有空位
        room--;
        msg_arg.mtype = SOFAQUEST;//请求沙发
        msgsnd(sofa_quest_id, &msg_arg, sizeof(msg_arg), 0);
    } else//等候室满
        printf("理发店满员，顾客%d 离开\n", id);

    if (sofa > 0) {//沙发有空位，处理沙发请求
        if (msgrcv(sofa_quest_id, &msg_arg, sizeof(msg_arg), SOFAQUEST, IPC_NOWAIT) >= 0) {
            sofa--;
            room++;
        }
    }

    if (chair > 0) {//理发椅有空位，发出请求
        printf("顾客%d 请求理发\n", msg_arg.mid);
        sofa++;
        chair--;
        msg_arg.mtype = HAIRQUEST;
        msgsnd(barber_quest_id, &msg_arg, sizeof(msg_arg), 0);
    }

    if (msgrcv(barber_respond_id, &msg_arg, sizeof(msg_arg), CUT_FINISHED, IPC_NOWAIT) >= 0) {//理发完成，发出结帐请求
        chair++;
        msg_arg.mtype = ACCOUNTQUEST;
        msgsnd(account_quest_id, &msg_arg, sizeof(msg_arg), 0);
    }

    if (msgrcv(account_respond_id, &msg_arg, sizeof(msg_arg), ACC_FINISHED, IPC_NOWAIT) >= 0) {//结帐完成
        //printf("顾客 %d 结帐\n", msg_arg.mid);

    }
}
```

顾客函数，每隔几秒产生一位顾客，依次判断沙发、等待室空位，等待室无空位则离开，若理发椅有空位则将其加入理发队列尾，向理发师发出请求，理发师会对队列头的顾客进行处理，接受到理发完成信息后请求结账。

```
account_sem = set_sem(account_key, sem_val, sem_flg);
while (1) {
    if (msgrcv(barber_quest_id, &msg_arg, sizeof(msg_arg), HAIRQUEST, 0) >= 0) {
        sleep(rate);
        printf("理发师%d 为顾客%d 理发\n", barber_num, msg_arg.mid);
        msg_arg.mtype = CUT_FINISHED;
        msgsnd(barber_respond_id, &msg_arg, sizeof(msg_arg), 0);
    }
    if (msgrcv(account_quest_id, &msg_arg, sizeof(msg_arg), ACCOUNTQUEST, 0) >= 0) {
        sem_wait(account_sem);//收到结帐信号
        printf("理发师%d 给顾客%d 结账\n", barber_num, msg_arg.mid);
        msg_arg.mtype = ACC_FINISHED;
        msgsnd(account_respond_id, &msg_arg, sizeof(msg_arg), 0);
        sem_signal(account_sem);
    }
}

return EXIT_SUCCESS;
```

理发师收到请求处理完毕后返回完成信号。

11）请按与以上不同的启动顺序、不同的延迟时间，启动更多的读写者。观察和分析是否仍能满足我们要求的读写者问题的功能。

12) 请修改以上程序，制造一个读者或写者的饥饿现象。观察什么是饥饿现象，说明为什么会发生这种现象。

(1) 以不同的启动顺序、不同的延迟时间、更多的读写者都能满足要求
(2) 将进行写操作时的 count 值不复原为 100，则会由于 count 值不满足要求，读写都无法进行。饥饿现象是指在并发环境中某个进程或线程由于长时间得不到所需的资源而无法执行或完成任务的情况。饥饿现象的主要原因包括不公平的资源分配策略、缺乏抢占机制等

问题及收获：

非对称性互斥操作是指某些进程（例如理发师）有优先权，而另一些进程（例如顾客）则在某些条件下等待资源。在本实验中，理发师有优先处理顾客的能力，但顾客也需要等待理发师。饥饿现象：发出请求后后一直有别的优先级更高的请求，导致发出的请求一直不被处理。在本实验中，顾客进店时，如果排队等待的顾客过多，会长时间无法得到理发服务。可以使用调度算法来保证每个进程都有机会执行。例如通过轮询队列或优先级队列来确保每个顾客都能得到处理。消息传递可以用来实现进程之间的通信。在本实验中，理发师和顾客之间的操作可以通过消息传递进行协调。通过消息传递的方式，进程之间不需要直接共享数据，只通过传递消息来进行交互，可以有效地避免数据冲突和资源竞争。