山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100300157 | 姓名：孙荣骏 | | 班级：21.3 |
| 实验编号：实验5 | | | |
| 实验题目：进程互斥实验 | | | |
| 实验学时：2 | | 实验日期：2022.5.12 | |
| 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。 | | | |
| 硬件环境：  宿主机  机型：联想拯救者r7000p2021  CPU：AMD R7 5800H  内存：16G  虚拟机  RAM：4GB | | | |
| 软件环境：  虚拟机：Ubuntu 16.04  宿主机：win10 | | | |
| 实验步骤与内容：  **独立实验：**          主要实验代码及注释如下：  #include "ipc.h"  int main()  {  Msg\_buf msg\_arg;  struct msqid\_ds msg\_inf;  // 建立一个共享内存  buff\_key = 1001;  buff\_num = 7;  shm\_flg = IPC\_CREAT | 0644;  buff\_ptr = (int \*)set\_shm(buff\_key, buff\_num, shm\_flg);  buff\_ptr[count\_Chair] = 3; // 理发椅空闲的数量  buff\_ptr[count\_Sofa] = 4; // 沙发空闲的数量  buff\_ptr[count\_Room] = 13; // 等候室空闲的数量  buff\_ptr[count\_Cash] = 1; // 现金登记册是否可用（1可用，0不可用）  // 建立一条请求消息队列  quest\_key = 2001;  quest\_flg = IPC\_CREAT| 0644;  quest\_id = set\_msq(quest\_key, quest\_flg);  // 建立一条响应消息队列  respond\_key = 2002;  respond\_flg = IPC\_CREAT|0644;  respond\_id = set\_msq(respond\_key,respond\_flg);  // 下面的消息队列专门用于收费  // 建立一条请求消息队列  quest\_key2 = 3001;  quest\_flg2 = IPC\_CREAT| 0644;  quest\_id2 = set\_msq(quest\_key2, quest\_flg2);  // 建立一条响应消息队列  respond\_key2 = 4002;  respond\_flg2 = IPC\_CREAT|0644;  respond\_id2 = set\_msq(respond\_key2,respond\_flg2);    int hairCutters\_state[3]; // 3位理发师的状态，0为睡觉，1为理发或收费中  // 等那3位理发师都到了ALTLI理发店，就开始营业（接收3位理发师上班请求）  for (int i = 0; i < 3; i++)  if (msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), WORKQUEST, 0) >= 0)  {  id[i] = msg\_arg.mid;  hairCutters\_state[i] = 0;  printf("%d 理发师到岗\n", msg\_arg.mid);  // 发给理发师，在数组中的索引，用于收费部分的处理  msg\_arg.mtype = msg\_arg.mid;  msg\_arg.mid = i;  msgsnd(respond\_id, &msg\_arg, sizeof(msg\_arg), 0);  }    // 理发店准备迎接顾客  printf("理发店开始营业！\n\n欢迎来到理发店！\n");    while (1)  {  // 当count\_Room大于0时，说明等候室有多余空闲的位置，查询街上是否有顾客想来理发  if (buff\_ptr[count\_Room] > 0)  {  quest\_flg = IPC\_NOWAIT; // 以非阻塞方式接收请求消息  if(msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), ROOMQUEST, quest\_flg) >= 0)  {  // 街上有顾客，允许ta来等候室  buff\_ptr[count\_Room]--;  msg\_arg.mtype = msg\_arg.mid;  msgsnd(respond\_id, &msg\_arg, sizeof(msg\_arg), 0);  printf("%d 顾客请求进入等候室\n", msg\_arg.mid);  }  }    // 当count\_Sofa大于0时，说明沙发有多余空闲的位置，查询等候室是否有顾客等待  if (buff\_ptr[count\_Sofa] > 0)  {  quest\_flg = IPC\_NOWAIT; // 以非阻塞方式接收请求消息  if(msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), SOFAQUEST, quest\_flg) >= 0)  {  // 等候室中有顾客，允许ta坐沙发  buff\_ptr[count\_Sofa]--;  buff\_ptr[count\_Room]++;  msg\_arg.mtype = msg\_arg.mid;  printf("%d 顾客请求坐沙发\n", msg\_arg.mid);  msgsnd(respond\_id, &msg\_arg, sizeof(msg\_arg), 0);  }  }    // 当count\_Chair大于0时，说明理发椅有多余空闲的位置，查询沙发上是否有客人等待  if (buff\_ptr[count\_Chair] > 0)  {  quest\_flg = IPC\_NOWAIT; // 以非阻塞方式接收请求消息  if(msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), CHAIRQUEST, quest\_flg) >= 0)  {  // 沙发上有顾客请求理发，允许ta理发  buff\_ptr[count\_Chair]--;  buff\_ptr[count\_Sofa]++;  for (int i = 0; i < 3; i++) // 找一个空闲的理发师理发  if (!hairCutters\_state[i])  {  msg\_arg.mtype = id[i];  hairCutters\_state[i] = 1;  break;  }  msgsnd(respond\_id, &msg\_arg, sizeof(msg\_arg), 0); // 给理发师发理发消息  msg\_arg.mtype = msg\_arg.mid;  msgsnd(respond\_id, &msg\_arg, sizeof(msg\_arg), 0); // 给顾客发理发消息  printf("%d 顾客请求理发\n", msg\_arg.mid);  }  }    // 接收到一个理发或收费完成的消息  quest\_flg = IPC\_NOWAIT; // 以非阻塞方式接收请求消息  if(msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), FINISHED, quest\_flg) >= 0)  {  // 理发店老板收到理发师发送的理发或收费完成信号  buff\_ptr[count\_Chair]++;  for (int i = 0; i < 3; i++)  if (msg\_arg.mid == id[i])  {  hairCutters\_state[i] = 0;  break;  }  }    // 接收到一个顾客付钱的消息  quest\_flg = IPC\_NOWAIT; // 以非阻塞方式接收请求消息  if(msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), MONEYQUEST, quest\_flg) >= 0)  {  // 理发店让顾客指定的理发师收钱  msg\_arg.mtype = id[msg\_arg.mid%10];  msg\_arg.mid = msg\_arg.mid/10;  hairCutters\_state[msg\_arg.mid%10] = 1;  msgsnd(quest\_id2, &msg\_arg, sizeof(msg\_arg), 0); // 给理发师发收费消息  }  }    return EXIT\_SUCCESS;  }  #include "ipc.h"  int main(int argc,char \*argv[])  {  int i;  Msg\_buf msg\_arg;  // 附加一个要读内容的共享内存  buff\_key = 1001;  buff\_num = 7;  shm\_flg = IPC\_CREAT | 0644;  buff\_ptr = (int \*)set\_shm(buff\_key, buff\_num, shm\_flg);  // 联系一个请求消息队列  quest\_flg = IPC\_CREAT | 0644;  quest\_key = 2001;  quest\_id = set\_msq(quest\_key, quest\_flg);  // 联系一个响应消息队列  respond\_flg = IPC\_CREAT | 0644;  respond\_key = 2002;  respond\_id = set\_msq(respond\_key, respond\_flg);  // 下面的消息队列专门用于收费  // 建立一条请求消息队列  quest\_key2 = 3001;  quest\_flg2 = IPC\_CREAT| 0644;  quest\_id2 = set\_msq(quest\_key2, quest\_flg2);  // 建立一条响应消息队列  respond\_key2 = 4002;  respond\_flg2 = IPC\_CREAT|0644;  respond\_id2 = set\_msq(respond\_key2,respond\_flg2);    msg\_arg.mid = getpid();    // 随机数，用于收费时随机选择一个理发师  int a;  srand((unsigned)time(NULL));  a = rand()%3; // 产生0~2范围内的随机数    // 此时说明有空余的理发椅，没有顾客等待，请求理发  if (buff\_ptr[count\_Chair] > 0 && buff\_ptr[count\_Sofa] == 4)  {  msg\_arg.mtype = CHAIRQUEST;  buff\_ptr[count\_Sofa]--;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发中...\n", msg\_arg.mid);  msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发完成\n", msg\_arg.mid);  // 向理发店付钱  msg\_arg.mtype = MONEYQUEST;  msg\_arg.mid = msg\_arg.mid\*10 + a; // 将顾客选择的理发师放在mid的最后一位  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msg\_arg.mid = getpid();  msgrcv(respond\_id2, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客离开理发店\n", msg\_arg.mid);  }  // 此时说明沙发中还有位置，请求去沙发等待  else if (buff\_ptr[count\_Chair] == 0 && buff\_ptr[count\_Sofa] > 0)  {  msg\_arg.mtype = SOFAQUEST;  buff\_ptr[count\_Room]--;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客在沙发等待中...\n", msg\_arg.mid);  // 请求理发  msg\_arg.mtype = CHAIRQUEST;  buff\_ptr[count\_Sofa]--;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发中...\n", msg\_arg.mid);  msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发完成\n", msg\_arg.mid);  // 向理发店付钱  msg\_arg.mtype = MONEYQUEST;  msg\_arg.mid = msg\_arg.mid\*10 + a; // 将顾客选择的理发师放在mid的最后一位  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msg\_arg.mid = getpid();  msgrcv(respond\_id2, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客离开理发店\n", msg\_arg.mid);  }  // 此时说明等候室中还有位置，请求去等候室等待  else if (buff\_ptr[count\_Sofa] == 0 && buff\_ptr[count\_Room] > 0)  {  msg\_arg.mtype = ROOMQUEST;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客在等候室等待中...\n", msg\_arg.mid);  // 请求去沙发等待  msg\_arg.mtype = SOFAQUEST;  buff\_ptr[count\_Room]--;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客在沙发等待中...\n", msg\_arg.mid);  // 请求理发  msg\_arg.mtype = CHAIRQUEST;  buff\_ptr[count\_Sofa]--;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发中...\n", msg\_arg.mid);  msgrcv(quest\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客理发完成\n", msg\_arg.mid);  // 向理发店付钱  msg\_arg.mtype = MONEYQUEST;  msg\_arg.mid = msg\_arg.mid\*10 + a; // 将顾客选择的理发师放在mid的最后一位  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msg\_arg.mid = getpid();  msgrcv(respond\_id2, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, 0);  printf("%d 顾客离开理发店\n", msg\_arg.mid);  }  // 此时说明人数超过20，不进入理发店（如果已经进了理发店，那么就不会判断是否人数超过20）  else  {  printf("%d 顾客去其他理发店理发\n", msg\_arg.mid);  }    return EXIT\_SUCCESS;  }  #include "ipc.h"  int main(int argc,char \*argv[])  {  int i;  int rate1, rate2; // rate1是理发速度，rate2是收费速度  Msg\_buf msg\_arg;  // 可在在命令行第一参数指定一个进程睡眠秒数，以调解进程执行速度  if(argv[1] != NULL)  rate1 = atoi(argv[1]);  else  rate1 = 3;  if (argv[2] != NULL)  rate2 = atoi(argv[2]);  else  rate2 = 1;  // 附加一个要读内容的共享内存  buff\_key = 1001;  buff\_num = 7;  shm\_flg = IPC\_CREAT | 0644;  buff\_ptr = (int \*)set\_shm(buff\_key, buff\_num, shm\_flg);  // 联系一个请求消息队列  quest\_flg = IPC\_CREAT | 0644;  quest\_key = 2001;  quest\_id = set\_msq(quest\_key, quest\_flg);  // 联系一个响应消息队列  respond\_flg = IPC\_CREAT | 0644;  respond\_key = 2002;  respond\_id = set\_msq(respond\_key, respond\_flg);  // 下面的消息队列专门用于收费  // 建立一条请求消息队列  quest\_key2 = 3001;  quest\_flg2 = IPC\_CREAT| 0644;  quest\_id2 = set\_msq(quest\_key2, quest\_flg2);  // 建立一条响应消息队列  respond\_key2 = 4002;  respond\_flg2 = IPC\_CREAT|0644;  respond\_id2 = set\_msq(respond\_key2,respond\_flg2);    msg\_arg.mid = getpid();  // 上班前给ALTLI理发店发消息，表示我开始上班  msg\_arg.mtype = WORKQUEST;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), getpid(), 0);  int flag = msg\_arg.mid;  msg\_arg.mid = getpid();  printf("%d 理发师上班\n", msg\_arg.mid);  printf("%d 理发师睡觉中...\n", getpid());    // 循环：理发、收费、睡觉  while (1)  {  if (msgrcv(respond\_id, &msg\_arg, sizeof(msg\_arg), msg\_arg.mid, IPC\_NOWAIT) >= 0)  {  printf("%d 理发师为 %d 顾客理发中...\n", getpid(), msg\_arg.mid);  sleep(rate1);  printf("%d 顾客理发完成\n", msg\_arg.mid);  // 理发师通知顾客理发完成，让顾客找一个理发师收费  msg\_arg.mtype = msg\_arg.mid;  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  sleep(1);  if (msgrcv(quest\_id2, &msg\_arg, sizeof(msg\_arg), getpid(), IPC\_NOWAIT) >= 0)  {  // 顾客令该理发师收费  while (1)  {  if (buff\_ptr[count\_Cash])  {  buff\_ptr[count\_Cash] = 0;  printf("%d 理发师收取 %d 顾客费用中...\n", getpid(), msg\_arg.mid);  sleep(rate2);  buff\_ptr[count\_Cash] = 1;  msg\_arg.mtype = msg\_arg.mid;  printf("收费完毕\n");  msgsnd(respond\_id2, &msg\_arg, sizeof(msg\_arg), 0); // 向顾客发送收费完成信号  break;  }  }  }  // 理发师通知理发店老板理发完成，请求给下一个顾客理发  msg\_arg.mtype = FINISHED;  msg\_arg.mid = getpid();  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0);  if (buff\_ptr[count\_Sofa] == 4)  printf("%d 理发师睡觉中...\n", getpid());  }  if (msgrcv(quest\_id2, &msg\_arg, sizeof(msg\_arg), getpid(), IPC\_NOWAIT) >= 0)  {  // 顾客令该理发师收费  while (1)  {  if (buff\_ptr[count\_Cash])  {  buff\_ptr[count\_Chair]--;  buff\_ptr[count\_Cash] = 0;  printf("%d 理发师收取 %d 顾客费用中...\n", getpid(), msg\_arg.mid);  sleep(rate2);  buff\_ptr[count\_Cash] = 1;  msg\_arg.mtype = msg\_arg.mid;  printf("收费完毕\n");  msgsnd(respond\_id2, &msg\_arg, sizeof(msg\_arg), 0); // 向顾客发送收费完成信号  msg\_arg.mtype = FINISHED;  msg\_arg.mid = getpid();  msgsnd(quest\_id, &msg\_arg, sizeof(msg\_arg), 0); // 向理发店发送收费完成信号  break;  }  }  }  }  return EXIT\_SUCCESS;  }  #include "ipc.h"  /\*  \* get\_ipc\_id() 从/proc/sysvipc/文件系统中获取 IPC 的 id 号  \* pfile: 对应/proc/sysvipc/目录中的 IPC 文件分别为  \*  \* msg-消息队列,sem-信号量,shm-共享内存  \* key: 对应要获取的 IPC 的 id 号的键值  \*/  int get\_ipc\_id(char \*proc\_file, key\_t key)  {  FILE \*pf;  int i, j;  char line[BUFSZ], colum[BUFSZ];  if((pf = fopen(proc\_file, "r")) == NULL)  {  perror("Proc file not open");  exit(EXIT\_FAILURE);  }  fgets(line, BUFSZ, pf);  while(!feof(pf))  {  i = j = 0;  fgets(line, BUFSZ, pf);  while(line[i] == ' ')  i++;  while(line[i] != ' ')  colum[j++] = line[i++];  colum[j] = '\0';  if(atoi(colum) != key)  continue;  j = 0;  while(line[i] == ' ')  i++;  while(line[i] !=' ')  colum[j++] = line[i++];  colum[j] = '\0';  i = atoi(colum);  fclose(pf);  return i;  }  fclose(pf);  return -1;  }  /\*  \* 信号灯上的 down/up 操作  \* semid:信号灯数组标识符  \* semnum:信号灯数组下标  \* buf:操作信号灯的结构  \*/  int down(int sem\_id)  {  struct sembuf buf;  buf.sem\_op = -1;  buf.sem\_num = 0;  buf.sem\_flg = SEM\_UNDO;  if((semop(sem\_id, &buf, 1)) < 0)  {  perror("down error ");  exit(EXIT\_FAILURE);  }  return EXIT\_SUCCESS;  }  int up(int sem\_id)  {  struct sembuf buf;  buf.sem\_op = 1;  buf.sem\_num = 0;  buf.sem\_flg = SEM\_UNDO;  if((semop(sem\_id, &buf, 1)) < 0)  {  perror("up error ");  exit(EXIT\_FAILURE);  }  return EXIT\_SUCCESS;  }  /\*  \* set\_sem 函数建立一个具有 n 个信号灯的信号量  \*  如果建立成功,返回 一个信号灯数组的标识符 sem\_id  \*  输入参数:  \* sem\_key 信号灯数组的键值  \* sem\_val 信号灯数组中信号灯的个数  \* sem\_flag 信号等数组的存取权限  \*/  int set\_sem(key\_t sem\_key,int sem\_val,int sem\_flg)  {  int sem\_id;  Sem\_uns sem\_arg;  // 测试由 sem\_key 标识的信号灯数组是否已经建立  if((sem\_id = get\_ipc\_id("/proc/sysvipc/sem", sem\_key)) < 0)  {  // semget 新建一个信号灯,其标号返回到 sem\_id  if((sem\_id = semget(sem\_key, 1, sem\_flg)) < 0)  {  perror("semaphore create error");  exit(EXIT\_FAILURE);  }  // 设置信号灯的初值  sem\_arg.val = sem\_val;  if(semctl(sem\_id, 0, SETVAL, sem\_arg) < 0)  {  perror("semaphore set error");  exit(EXIT\_FAILURE);  }  }  return sem\_id;  }  /\*  \* set\_shm 函数建立一个具有 n 个字节 的共享内存区  \*  如果建立成功,返回一个指向该内存区首地址的指针 shm\_buf  \*  输入参数:  \* shm\_key 共享内存的键值  \* shm\_val 共享内存字节的长度  \* shm\_flag 共享内存的存取权限  \*/  char\* set\_shm(key\_t shm\_key,int shm\_num,int shm\_flg)  {  int i, shm\_id;  char \* shm\_buf;  // 测试由 shm\_key 标识的共享内存区是否已经建立  if((shm\_id = get\_ipc\_id("/proc/sysvipc/shm", shm\_key)) < 0)  {  // shmget 新建 一个长度为 shm\_num 字节的共享内存,其标号返回到 shm\_id  if((shm\_id = shmget(shm\_key,shm\_num,shm\_flg)) <0)  {  perror("shareMemory set error");  exit(EXIT\_FAILURE);  }  // shmat 将由 shm\_id 标识的共享内存附加给指针 shm\_buf  if((shm\_buf = (char \*)shmat(shm\_id,0,0)) < (char \*)0)  {  perror("get shareMemory error");  exit(EXIT\_FAILURE);  }  for(i = 0; i < shm\_num; i++)  shm\_buf[i] = 0; //初始为 0  }  // shm\_key 标识的共享内存区已经建立,将由 shm\_id 标识的共享内存附加给指针 shm\_buf  if((shm\_buf = (char \*)shmat(shm\_id,0,0)) < (char \*)0)  {  perror("get shareMemory error");  exit(EXIT\_FAILURE);  }  return shm\_buf;  }  /\*  \* set\_msq 函数建立一个消息队列  \* 如果建立成功,返回 一个消息队列的标识符 msq\_id  \* 输入参数:  \*  msq\_key 消息队列的键值  \*  msq\_flag 消息队列的存取权限  \*/  int set\_msq(key\_t msq\_key,int msq\_flg)  {  int msq\_id;  //测试由 msq\_key 标识的消息队列是否已经建立  if((msq\_id = get\_ipc\_id("/proc/sysvipc/msg", msq\_key)) < 0)  {  //msgget 新建一个消息队列,其标号返回到 msq\_id  if((msq\_id = msgget(msq\_key,msq\_flg)) < 0)  {  perror("messageQueue set error");  exit(EXIT\_FAILURE);  }  }  return msq\_id;  }  int id[3]; // 3位理发师的id（进程号）  key\_t buff\_key;  int buff\_num;  char \*buff\_ptr;  int shm\_flg;  int quest\_flg;  key\_t quest\_key;  int quest\_id;  int respond\_flg;  key\_t respond\_key;  int respond\_id;  int quest\_flg2;  key\_t quest\_key2;  int quest\_id2;  int respond\_flg2;  key\_t respond\_key2;  int respond\_id2;  int get\_ipc\_id(char \*proc\_file,key\_t key);  char \*set\_shm(key\_t shm\_key,int shm\_num,int shm\_flag);  int set\_msq(key\_t msq\_key,int msq\_flag);  int set\_sem(key\_t sem\_key,int sem\_val,int sem\_flag);  int down(int sem\_id);  int up(int sem\_id);  #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <sys/sem.h>  #include <sys/msg.h>  #include <time.h>  #define BUFSZ 256  #define CHAIRQUEST 1 // 顾客 理发椅请求标识  #define SOFAQUEST 2 // 顾客 沙发请求标识  #define ROOMQUEST 3 // 顾客 等候室请求标识  #define FINISHED 4 // 理发师对理发店 理发完成标识  #define FINISHED\_2 5 // 理发师对顾客 理发完成标识  #define HAIRQUEST 6 // 理发师 请求理发标识  #define WORKQUEST 7 // 理发师到理发店上班标识  #define MONEYQUEST 8 // 收费标识  // 共享内存的索引，方便阅读  #define count\_Chair 0  #define count\_Sofa 1  #define count\_Room 2  #define count\_Cash 3  /\*信号灯控制用的共同体\*/  typedef union semuns {  int val;  } Sem\_uns;  /\* 消息结构体\*/  typedef struct msgbuf {  long mtype;  int mid;  } Msg\_buf;  extern int id[3]; // 3位理发师的id（进程号）  extern key\_t buff\_key;  extern int buff\_num;  extern char \*buff\_ptr;  extern int shm\_flg;  extern int quest\_flg;  extern key\_t quest\_key;  extern int quest\_id;  extern int respond\_flg;  extern key\_t respond\_key;  extern int respond\_id;  extern int quest\_flg2;  extern key\_t quest\_key2;  extern int quest\_id2;  extern int respond\_flg2;  extern key\_t respond\_key2;  extern int respond\_id2;  extern int get\_ipc\_id(char \*proc\_file,key\_t key);  extern char \*set\_shm(key\_t shm\_key,int shm\_num,int shm\_flag);  extern int set\_msq(key\_t msq\_key,int msq\_flag);  extern int set\_sem(key\_t sem\_key,int sem\_val,int sem\_flag);  extern int down(int sem\_id);  extern int up(int sem\_id);  hdrs = ipc.h  opts = -g -c  b\_src = barberShop.c ipc.c  b\_obj = barberShop.o ipc.o  c\_src = customer.c ipc.c  c\_obj = customer.o ipc.o  h\_src = hairCutter.c ipc.c  h\_obj = hairCutter.o ipc.o  all: barberShop customer hairCutter    barberShop: $(b\_obj)  gcc $(b\_obj) -o barberShop  barberShop.o: $(b\_src) $(hdrs)  gcc $(opts) $(b\_src)  customer: $(c\_obj)  gcc $(c\_obj) -o customer  customer.o: $(c\_src) $(hdrs)  gcc $(opts) $(c\_src)  hairCutter: $(h\_obj)  gcc $(h\_obj) -o hairCutter  hairCutter.o: $(h\_src) $(hdrs)  gcc $(opts) $(h\_src)  clean:  rm barberShop customer hairCutter \*.o | | | |
| 结论分析与体会：  我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。  我对Linux中的消息队列的认识更清晰了。实验的逻辑比较复杂，涉及到街上、等候室、沙发、理发椅、理发师、顾客、理发店，这些事物之间的关系比较复杂，但如果理清楚，也是比较清晰的。对进程互斥的相关知识也进行了巩固。 | | | |