

计算机学院_操作系统_课程实验报告

实验题目：实验 5 进程互斥		学号：202300130183
日期：2025/4/21	班级：23 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		
<p>实验方法介绍：</p> <p>使用物理机上的 Ubuntu 24.04 来编写编译相应的代码。</p>		
<p>实验过程描述：</p> <p>首先是示例实验，我们先完成实验给的代码的拷贝：</p> <p>1. contro.c</p> <pre>#include "ipc.h" #include <stdio.h> 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 main(int argc, char* argv[]) { int i; int rate; int w_mid; int count = MAXVAL; Msg_buf msg_arg; struct msqid_ds msg_inf; // 建立一个共享内存先写入一串 A 字符模拟要读写的内容 buff_key = 101; buff_num = STRSIZ + 1; shm_flg = IPC_CREAT 0644; buff_ptr = (char*)set_shm(buff_key, buff_num, shm_flg); for (i = 0; i < STRSIZ; i++) buff_ptr[i] = 'A'; buff_ptr[i] = '\0'; // 建立一条请求消息队列</pre>		

```

quest_flg = IPC_CREAT | 0644;
quest_key = 201;
quest_id = set_msq(quest_key, quest_flg);
// 建立一条响应消息队列
respond_flg = IPC_CREAT | 0644;
respond_key = 202;
respond_id = set_msq(respond_key, respond_flg);
// 控制进程准备接收和响应读写者的消息
printf("Wait quest \n");
fflush(stdout);
while (1)
{
    // 当 count 大于 0 时说明没有新的读写请求, 查询是否有任何新请求
    if (count > 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), FINISHED,
quest_flg) >= 0)
        {
            // 有读者完成
            count++;
            printf("%d reader finished\n", msg_arg.mid);
            fflush(stdout);
        }
        else if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg), READERREQUEST,
quest_flg) >= 0)
        {
            // 有读者请求, 允许读者读
            count--;
            msg_arg.mtype = msg_arg.mid;
            msgsnd(respond_id, &msg_arg, sizeof(msg_arg), 0);
            printf("%d quest read\n", msg_arg.mid);
            fflush(stdout);
        }
    }
    // 当 count 等于 0 时说明写者正在写, 等待写完成
    if (count == 0)
    {
        // 以阻塞方式接收消息.
        msgrcv(quest_id, &msg_arg, sizeof(msg_arg), FINISHED, 0);
        count = MAXVAL;
        printf("%d write finished\n", msg_arg.mid);
        fflush(stdout);
        if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg), READERREQUEST,

```

```

quest_flg) >=
    0)
    {
        // 有读者请求, 允许读者读
        count--;
        msg_arg.mtype = msg_arg.mid;
        msgsnd(respond_id, &msg_arg, sizeof(msg_arg), 0);
        printf("%d quest read\n", msg_arg.mid);
        fflush(stdout);
    }
}
}
return EXIT_SUCCESS;
}

```

2. ipc.h

```

/*
*Filename: ipc.h
*copyright: (C) 2006 by zhonghonglie
* Function
: 声明 IPC 机制的函数原型和全局变量
*/
#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 <unistd.h>
#define BUFSZ 256
#define MAXVAL 100
#define STRSIZ 8
#define WRITERQUEST 1
#define READERQUEST 2
#define FINISHED 3
/*信号量控制用的共同体*/
typedef union semuns
{
    int val;
} Sem_uns;

// 读写完成标识
/* 消息结构体 */
typedef struct msgbuf

```

```

{
    long mtype;
    int mid;
} Msg_buf;
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;
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);

```

3. ipc.c

```

#include "ipc.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>

int get_ipc_id(char *proc_file, key_t key) {
    FILE *pf;
    int i, j;
    char line[BUFSZ], colum[BUFSZ];
    pf = fopen(proc_file, "r");
    if (pf == 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] != ' ') column[j++] = line[i++];
        column[j] = 0;
        if (atoi(column) != key) continue;
        j = 0;
        while (line[i] == ' ') i++;
        while (line[i] != ' ') column[j++] = line[i++];
        column[j] = 0;
        i = atoi(column);
        fclose(pf);
        return i;
    }
    fclose(pf);
    return -1;
}

char *set_shm(key_t shm_key, int shm_num, int shm_flag) {
    int shm_id;
    void *shm_buf;
    shm_id = get_ipc_id("/proc/sysvipc/shm", shm_key);
    if (shm_id < 0) {
        shm_id = shmget(shm_key, shm_num, shm_flag);
        if (shm_id < 0) {
            perror("ShareMemory create failed");
            exit(EXIT_FAILURE);
        }
        shm_buf = shmat(shm_id, 0, 0);
        if (shm_buf < 0) {
            perror("ShareMemory attach failed");
            exit(EXIT_FAILURE);
        }
        memset(shm_buf, 0, shm_num);
        return shm_buf;
    }
    shm_buf = shmat(shm_id, 0, 0);
    if (shm_buf < 0) {
        perror("ShareMemory attach failed");
        exit(EXIT_FAILURE);
    }
    return shm_buf;
}

int set_msq(key_t msq_key, int msq_flag) {
    int msq_id;
    msq_id = get_ipc_id("/proc/sysvipc/msg", msq_key);
    if (msq_id < 0) {

```

```

        msq_id = msgget(msq_key, msq_flag);
        if (msq_id < 0) {
            perror("MessageQueue create failed");
            exit(EXIT_FAILURE);
        }
    }
    return msq_id;
}

int set_sem(key_t sem_key, int sem_val, int sem_flag) {
    int sem_id;
    Sem_uns sem_arg;
    sem_id = get_ipc_id("/proc/sysvipc/sem", sem_key);
    if (sem_id < 0) {
        sem_id = semget(sem_key, 1, sem_flag);
        if (sem_id < 0) {
            perror("Semaphore create failed");
            exit(EXIT_FAILURE);
        }
        sem_arg.val = sem_val;
        if (semctl(sem_id, 0, SETVAL, sem_arg) < 0) {
            perror("Semaphore set failed");
            exit(EXIT_FAILURE);
        }
    }
    return sem_id;
}

int P(int sem_id) {
    struct sembuf buf;
    buf.sem_num = 0;
    buf.sem_op = -1;
    buf.sem_flg = SEM_UNDO;
    if (semop(sem_id, &buf, 1) < 0) {
        perror("Operation P failed");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}

int V(int sem_id) {
    struct sembuf buf;
    buf.sem_num = 0;
    buf.sem_op = 1;
    buf.sem_flg = SEM_UNDO;
    if (semop(sem_id, &buf, 1) < 0) {
        perror("Operation V failed");
    }
}

```

```

        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}

```

4. reader.c

```

#include "ipc.h"
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 main(int argc, char* argv[])
{
    int i;
    int rate;
    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)
        rate = atoi(argv[1]);
    else
        rate = 3;
    // 附加一个要读内容的共享内存
    buff_key = 101;
    buff_num = STRSIZ + 1;
    shm_flg = IPC_CREAT | 0644;
    buff_ptr = (char*)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 = 201;
    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 = 202;
    respond_id = set_msq(respond_key, respond_flg);
    // 循环请求读
    msg_arg.mid = getpid();
    while (1)
    {

```

```

    // 发读请求消息
    msg_arg.mtype = READERQUEST;
    msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
    printf("%d reader quest\n", msg_arg.mid);
    fflush(stdout);
    // 等待允许读消息
    msgrcv(respond_id, &msg_arg, sizeof(msg_arg), msg_arg.mid, 0);
    printf("%d reading: %s\n", msg_arg.mid, buff_ptr);
    fflush(stdout);
    sleep(rate);
    // 发读完成消息
    msg_arg.mtype = FINISHED;
    msgsnd(quest_id, &msg_arg, sizeof(msg_arg), quest_flg);
}
return EXIT_SUCCESS;
}

```

5. writer.c

```

#include "ipc.h"
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 main(int argc, char* argv[])
{
    int i, j = 0;
    int rate;
    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)
        rate = atoi(argv[1]);
    else
        rate = 3;
    // 附加一个要读内容的共享内存 buff_key = 101;
    buff_num = STRSIZ + 1;
    shm_flg = IPC_CREAT | 0644;
    buff_ptr = (char*)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 = 201;
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 = 202;
respond_id = set_msq(respond_key, respond_flg);
// 循环请求写
msg_arg.mid = getpid();
while (1)
{
    // 发写请求消息
    msg_arg.mtype = WRITERQUEST;
    msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
    printf("%d writer quest\n", msg_arg.mid);
    fflush(stdout);
    // 等待允许写消息
    msgrcv(respond_id, &msg_arg, sizeof(msg_arg), msg_arg.mid, 0);
    // 写入 STRSIZ 个相同的字符
    for (i = 0; i < STRSIZ; i++)
        buff_ptr[i] = 'A' + j;
    j = (j + 1) % STRSIZ; // 按 STRSIZ 循环变换字符
    printf("%d writing: %s\n", msg_arg.mid, buff_ptr);
    fflush(stdout);
    sleep(rate);
    // 发写完成消息
    msg_arg.mtype = FINISHED;
    msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
}
return EXIT_SUCCESS;
}

```

6. Makefile

```

BUILD_DIR = build
hdrs = ipc.h
c_src = control.c ipc.c
c_obj = control.o ipc.o
r_src = reader.c ipc.c
r_obj = reader.o ipc.o
w_src = writer.c ipc.c
w_obj = writer.o ipc.o
opts    = -g -c
all:    control reader writer
control: $(c_obj)
        gcc $(c_obj) -o control

```

```

control.o: $(c_src) $(hdrs)
    gcc $(opts) $(c_src)
reader: $(r_obj)
    gcc $(r_obj) -o reader
reader.o: $(r_src) $(hdrs)
    gcc $(opts) $(r_src)
writer: $(w_obj)
    gcc $(w_obj) -o writer
writer.o: $(w_src) $(hdrs)
    gcc $(opts) $(w_src)
clean:
    rm control reader writer *.o

```

为了便于我们测试程序，我们还有以下两个脚本：

7. test.sh

```

#!/bin/bash
mkdir 实验日志 2>/dev/null >> /dev/null
echo -e "启用 make clean 命令\n"
make clean 2>/dev/null
echo -e "编译程序\n"
make 2>/dev/null

touch control_log.txt
touch writer_log.txt
touch reader1_log.txt
touch reader2_log.txt
current_time=$(date "+%Y-%m-%d-%H-%M-%S")
mkdir $current_time
mkdir 标准运行测试
echo -e "开始进行标准运行测试\n"
./control >>./control_log.txt&
echo -e "已启动 control 进程\n"
ps | grep control
./reader 10 >>./reader1_log.txt&
./reader 1 >>./reader2_log.txt&
echo -e "已启动 reader1 进程和 reader2 进程\n"
ps | grep reader
echo -e "等待 5 秒后启动 writer 进程\n"
sleep 5
./writer 8 >>./writer_log.txt&
echo -e "controler , writer , reader1 , reader2 process start!\n"
ps | grep controler
ps | grep writer
ps | grep reader
echo -e "\n"

```

```
echo -e "等待 10 秒后杀掉 writer 进程\n"
sleep 10
./endtest.sh
echo -e "标准运行测试结束\n"
mv control_log.txt 标准运行测试
mv writer_log.txt 标准运行测试
mv reader1_log.txt 标准运行测试
mv reader2_log.txt 标准运行测试
mv 标准运行测试 $current_time
echo -e "实验日志已保存至 $current_time 目录下\n"
echo -e "-----分割线-----\n"
echo -e "开始进行随机读写测试\n"
random_test_time=$((RANDOM%10+10))
echo -e "随机读写测试将持续 $random_test_time 秒(每秒随机创建读写进程)\n"
touch control_log.txt
./control >> ./control_log.txt&
echo -e "已启动 control 进程\n"
ps | grep control
touch writer_log.txt
touch reader_log.txt
for i in $(seq 1 $random_test_time)
do
    sleep 1
    random_num=$((RANDOM%2+1))
    if [ $random_num -eq 1 ]
    then
        echo -e "随机创建 writer 进程\n"
        random_writer_time=$((RANDOM%8+1))
        echo -e "启动进程 ./writer
$random_writer_time >> ./writer_log.txt&\n"
        ./writer $random_writer_time >> ./writer_log.txt&
    else
        echo -e "随机创建 reader 进程\n"
        random_reader_time=$((RANDOM%8+1))
        echo -e "启动进程 ./reader
$random_reader_time >> ./reader_log.txt&\n"
        ./reader $random_reader_time >> ./reader_log.txt&
    fi
done
echo -e "writer , reader process start!\n"
ps | grep writer
ps | grep reader
echo -e "\n"
random_test_time=$((RANDOM%10+5))
```

```
echo -e "等待 $random_test_time 秒后结束测试\n"
sleep $random_test_time
./endtest.sh
echo -e "随机读写测试结束\n"
mkdir 随机读写测试
mv control_log.txt 随机读写测试
mv writer_log.txt 随机读写测试
mv reader_log.txt 随机读写测试
mv 随机读写测试 $current_time
echo -e "实验日志已保存至 $current_time 目录下\n"
echo -e "-----分割线-----\n"
echo -e "开始进行进程饥饿测试\n"
touch control_log.txt
./control >>./control_log.txt&
echo -e "已启动 control 进程\n"
ps | grep control
touch writer_log.txt
./writer 8 >>./writer_log.txt&
echo -e "已启动 writer 进程\n"
ps | grep writer
touch reader1_log.txt
./reader 10 >>./reader1_log.txt&
touch reader2_log.txt
./reader 1 >>./reader2_log.txt&
echo -e "已启动 reader1 进程和 reader2 进程\n"
ps | grep reader
echo -e "等待 10 秒后杀掉 writer 进程\n"
sleep 10
./endtest.sh
echo -e "进程饥饿测试结束\n"
mkdir 进程饥饿测试
mv control_log.txt 进程饥饿测试
mv writer_log.txt 进程饥饿测试
mv reader1_log.txt 进程饥饿测试
mv reader2_log.txt 进程饥饿测试
mv 进程饥饿测试 $current_time
echo -e "实验日志已保存至 $current_time 目录下\n"
echo -e "-----分割线-----\n"
echo -e "实验结束\n"
mv $current_time 实验日志
echo -e "实验日志已保存至 实验日志 目录下\n"
```

8. endtest.sh

```
#!/bin/bash
```

```
echo -e "these process will be killed\n"
```

```

ps | grep control
ps | grep writer
ps | grep reader
pkill control
pkill writer
pkill reader
echo -e "all process killed\n"

```

然后我们执行 test.sh 可以得到以下输出(仅展示部分用于示意):

1. 标准实验, 即最初测试功能的实验:

The screenshot displays four terminal windows, each showing a log file. The windows are titled 'control_log.txt', 'reader1_log.txt', 'reader2_log.txt', and 'writer_log.txt'. The 'control_log.txt' window shows a sequence of events including 'Wait quest', 'quest read', 'reader finished', and 'quest read' for various process IDs (e.g., 24499, 24500). The 'reader1_log.txt' window shows 'reader quest' and 'reading' events for process 24499. The 'reader2_log.txt' window shows 'reader quest' and 'reading' events for process 24500. The 'writer_log.txt' window shows 'writer quest' and 'reading' events for process 24553. Each log entry is preceded by a line number.

2. 随机实验, 即随机创建读者写者的实验:

```
control_log.txt × reader_log.txt × writer_log.txt ×
Homework > 操作系统作业 > 实验 > 实验5 > 示例实验 > 实验5 > 示例实验 > 实验日志 > 2025-04-18-16-35-24 > 随机 > 2025-04-18-16-35-24 > 随机读写测试 > writer_log.txt

1 Wait quest
2 24617 quest read
3 24617 reader finished
4 24617 quest read
5 24630 quest read
6 24630 reader finished
7 24630 quest read
8 24630 reader finished
9 24630 quest read
10 24638 quest read
11 24617 reader finished
12 24617 quest read
13 24630 reader finished
14 24630 quest read
15 24630 reader finished
16 24630 quest read
17 24630 reader finished
18 24630 quest read
19 24638 reader finished
20 24638 quest read
21 24617 reader finished
22 24617 quest read
23 24630 reader finished
24 24630 quest read
25 24660 quest read
26 24630 reader finished
27 24630 quest read
28 24660 reader finished
29 24660 quest read
30 24630 reader finished
31 24630 quest read
32 24638 reader finished
33 24638 quest read
34 24660 reader finished
35 24660 quest read
36 24617 reader finished
37 24617 quest read
38 24630 reader finished
39 24630 quest read

1 24617 reader quest
2 24617 reading: AAAAAAAAA
3 24617 reader quest
4 24617 reading: AAAAAAAAA
5 24630 reader quest
6 24630 reading: AAAAAAAAA
7 24630 reader quest
8 24630 reading: AAAAAAAAA
9 24630 reader quest
10 24630 reading: AAAAAAAAA
11 24638 reader quest
12 24638 reading: AAAAAAAAA
13 24617 reader quest
14 24617 reading: AAAAAAAAA
15 24630 reader quest
16 24630 reading: AAAAAAAAA
17 24630 reader quest
18 24630 reading: AAAAAAAAA
19 24630 reader quest
20 24630 reading: AAAAAAAAA
21 24638 reader quest
22 24638 reading: AAAAAAAAA
23 24617 reader quest
24 24617 reading: AAAAAAAAA
25 24630 reader quest
26 24630 reading: AAAAAAAAA
27 24660 reader quest
28 24660 reading: AAAAAAAAA
29 24630 reader quest
30 24630 reading: AAAAAAAAA
31 24660 reader quest
32 24660 reading: AAAAAAAAA
33 24630 reader quest
34 24630 reading: AAAAAAAAA
35 24638 reader quest
36 24638 reading: AAAAAAAAA
37 24660 reader quest
38 24660 reading: AAAAAAAAA
39 24617 reader quest

1 24621 writer quest
2 24625 writer quest
3 24634 writer quest
4 24642 writer quest
5 24646 writer quest
6 24654 writer quest
7 24668 writer quest
8 24674 writer quest
9 24682 writer quest
10 24686 writer quest
11 24690 writer quest
12
```

3. 饥饿实验:


```
control_log.txt × ... reader1_log.txt ... reader2_log.txt ... writer_log.txt ...
6-35-24 > 进程饥饿测试 > control_log.txt 5-35-24 > 进程饥饿测试 > reader1_log.txt 5-35-24 > 进程饥饿测试 > reader2_log.txt 16-35-24 > 进程饥饿测试 > writer_log.txt

1 Wait quest
2 24745 quest read
3 24747 quest read
4 24747 reader finished
5 24747 quest read
6 24747 reader finished
7 24747 quest read
8 24747 reader finished
9 24747 quest read
10 24747 reader finished
11 24747 quest read
12 24747 reader finished
13 24747 quest read
14 24747 reader finished
15 24747 quest read
16 24747 reader finished
17 24747 quest read
18 24747 reader finished
19 24747 quest read
20 24747 reader finished
21 24747 quest read
22 24745 reader finished
23 24745 quest read
24 24747 reader finished
25 24747 quest read
26

1 24745 reader quest
2 24745 reading:
  AAAAAAAAA
3 24745 reader quest
4 24745 reading:
  AAAAAAAAA
5

1 24747 reader quest
2 24747 reading:
  AAAAAAAAA
3 24747 reader quest
4 24747 reading:
  AAAAAAAAA
5 24747 reader quest
6 24747 reading:
  AAAAAAAAA
7 24747 reader quest
8 24747 reading:
  AAAAAAAAA
9 24747 reader quest
10 24747 reading:
  AAAAAAAAA
11 24747 reader quest
12 24747 reading:
  AAAAAAAAA
13 24747 reader quest
14 24747 reading:
  AAAAAAAAA
15 24747 reader quest
16 24747 reading:
  AAAAAAAAA
17 24747 reader quest
18 24747 reading:
  AAAAAAAAA
19 24747 reader quest
20 24747 reading:
  AAAAAAAAA
21 24747 reader quest
22 24747 reading:
  AAAAAAAAA
23

1 24741 writer quest
2
```

然后就是独立实验部分，这一部分考虑到纯 C 语言编写会有亿点麻烦，所以我们采取 C++ 来编写，其中我们封装了 sem、shm、user_sems、service_sems 这几个类，将对信号量和共享内存的操作分离开，并通过同步 key 值以及 ftok 函数的方式对齐信号量和共享内存号。首先我们将整个架构分为了理发店、理发师、用户，其中用户负责申请服务，理发师负责提供服务，理发店负责接受客户、转移客户，维护客户排队队列，管理收入，维护打烊计时器。

我们共有以下代码：

1. common.h

```
#include <cstdlib>
#include <iostream>
#include <ostream>
#include <queue>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <time.h>
```

```

#include <unistd.h>
#include <vector>
#define SHM_SIZE sizeof(double)
#define SHM_MODE 0666 | IPC_CREAT
#define SEM_MODE 0666 | IPC_CREAT
#define SEM_VALUE 1
#define PATH "./common.h"
#define RECEIVER_TIME_INTERVAL 1 // 接受新顾客的耗时
#define BARBER_BARBE_TIME_INTERVAL 1 // 理发师理发的耗时
#define CASHIER_CASH_TIME_INTERVAL 1 // 收银的耗时
#define MOVER_MOVE_TIME_INTERVAL 1 // 移动顾客的耗时
#define MANAGER_TIME_INTERVAL 1 // 管理顾客付钱的耗时
#define WAIT_ROOM_SIZE 13 // 等待区大小
#define SOFA_SIZE 4 // 座位大小
#define BARBERSHOP_SIZE 20 // 理发店总容量
#define REST_CHECK_TIME_INTERVAL 1 // 休息检查的耗时
#define SHOP_RUNNING_TIME 12 // 店铺运行时间
static inline void log_prefix()
{
    time_t current_time;
    struct tm* time_info;
    time(&current_time);
    char time_str[100];
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", time_info);
    printf("[ %s ] Pid : %d, ", time_str, getpid());
    fflush(stdout);
}
static std::ostream& log()
{
    log_prefix();
    return std::cout;
}
static inline void error_prefix()
{
    time_t current_time;
    struct tm* time_info;
    time(&current_time);
    char time_str[100];
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", time_info);
    fprintf(stderr, "[ %s ] Pid : %d, ", time_str, getpid());
    fflush(stdout);
}

```



```

}
static int key_cnt = 0;
class sem
{
private:
    key_t key;
    int sem_id;
public:
    sem()
    {
        key = ftok(PATH, key_cnt);
        if (key == -1)
        {
            error_prefix();
            perror("ftok");
            throw std::runtime_error("syscall failed");
        }
        key_cnt++;
        sem_id = semget(key, SEM_VALUE, SEM_MODE);
    }
    sem(int value)
    {
        key = ftok(PATH, key_cnt);
        if (key == -1)
        {
            error_prefix();
            perror("ftok");
            throw std::runtime_error("syscall failed");
        }
        key_cnt++;
        sem_id = semget(key, SEM_VALUE, SEM_MODE);
        if (semctl(sem_id, 0, SETVAL, value) == -1)
        {
            error_prefix();
            perror("semctl");
            throw std::runtime_error("syscall failed");
        }
    }
    ~sem()
    {
        if (semctl(sem_id, 0, IPC_STAT, 0) != 1)
        {
            return;
        }
    }
}

```

```

        if (semctl(sem_id, 0, IPC_RMID, 0) == -1)
        {
            error_prefix();
            perror("semctl");
        }
    }

    void down()
    {
        struct sembuf sem[1];
        sem[0].sem_num = 0;
        sem[0].sem_op = -1;
        sem[0].sem_flg = SEM_UNDO;
        if (semop(sem_id, sem, 1) == -1)
        {
            error_prefix();
            perror("semop");
            throw std::runtime_error("syscall failed");
        }
    }

    void up()
    {
        struct sembuf sem[1];
        sem[0].sem_num = 0;
        sem[0].sem_op = 1;
        sem[0].sem_flg = SEM_UNDO;
        if (semop(sem_id, sem, 1) == -1)
        {
            error_prefix();
            perror("semop");
            throw std::runtime_error("syscall failed");
        }
    }
};

class shm
{
private:
    key_t key;
    int shm_id;
    void* shm_ptr;
    int shm_size;
public:
    shm()
    {
        key = ftok(PATH, key_cnt);
    }

```

```

if (key == -1)
{
    error_prefix();
    perror("ftok");
    throw std::runtime_error("syscall failed");
}
key_cnt++;
shm_id = shmget(key, SHM_SIZE, SHM_MODE);
if (shm_id == -1)
{
    error_prefix();
    perror("shmget");
    throw std::runtime_error("syscall failed");
}
shm_ptr = shmat(shm_id, NULL, 0);
if (shm_ptr == (void*)-1)
{
    error_prefix();
    perror("shmat");
    throw std::runtime_error("syscall failed");
}
shm_size = SHM_SIZE;
}
shm(int size)
{
    key = ftok(PATH, key_cnt);
    if (key == -1)
    {
        error_prefix();
        perror("ftok");
        throw std::runtime_error("syscall failed");
    }
    key_cnt++;
    shm_id = shmget(key, size, SHM_MODE);
    if (shm_id == -1)
    {
        error_prefix();
        perror("shmget");
        throw std::runtime_error("syscall failed");
    }
    shm_ptr = shmat(shm_id, NULL, 0);
    if (shm_ptr == (void*)-1)
    {
        error_prefix();

```

```

        perror("shmat");
        throw std::runtime_error("syscall failed");
    }
    shm_size = size;
}
~shm()
{
    if (shmdt(shm_ptr) == -1)
    {
        error_prefix();
        perror("shmdt");
    }
    if (shmctl(shm_id, IPC_STAT, 0) != 1)
    {
        return;
    }
    if (shmctl(shm_id, IPC_RMID, 0) == -1)
    {
        error_prefix();
        perror("shmctl");
    }
}
template <typename T> void write(const T data)
{
    if (sizeof(T) > shm_size)
    {
        error_prefix();
        fprintf(stderr, "shm size is not enough to write data");
        throw std::runtime_error("syscall failed");
    }
    memcpy(shm_ptr, (void*)&data, sizeof(T));
}
template <typename T> void read(T& data)
{
    if (sizeof(T) > shm_size)
    {
        error_prefix();
        fprintf(stderr, "shm size is not enough to read data");
        throw std::runtime_error("syscall failed");
    }
    memcpy((void*)&data, shm_ptr, sizeof(T));
}
};
static shm barber_shop_capacity(sizeof(int));

```

```

static shm for_sys_key_cnt(sizeof(int));
static shm money_trade_buffer(sizeof(double));
static shm sum_money(sizeof(double));
static sem new_customer_in(0);
static sem new_customer_come(0);
static sem is_someone_is_paying(1);
static sem is_someone_is_receiving_money(1);
static sem the_man_who_is_paying_is_done(0);
static sem is_someone_is_entring(1);
static sem needed_services_cnt(0);
static sem needed_pay_money_cnt(0);
static sem barber_finish_service(0);
static sem barber_finish_receive_money(0);
class user_sems
{
private:
    sem move_to_sofa;
    sem get_service;
    sem give_money;
public:
    void wait_for_move_to_sofa() { move_to_sofa.down(); }
    void wait_for_get_service() { get_service.down(); }
    void wait_for_give_money() { give_money.down(); }
};
static int service_cnt = 0;
class service_sems
{
private:
    sem user_move_to_sofa;
    sem support_service;
    sem receive_money;
    int id;
public:
    void let_user_move_to_sofa()
    {
        user_move_to_sofa.up();
        needed_services_cnt.up();
    }
    void support_service_for_user()
    {
        support_service.up();
        needed_pay_money_cnt.up();
    }
    void let_user_pay_money() { receive_money.up(); }
};

```

```

service_sems()
{
    int customer_num = 0;
    barber_shop_capacity.read(customer_num);
    customer_num++;
    barber_shop_capacity.write(customer_num);
    id = service_cnt++;
}
int get_id() { return id; }
};
static shm is_closing(sizeof(int));

```

2. barbershop.cpp

```

#include "common.h"
#include <chrono>
#include <cmath>
#include <thread>
#include <unistd.h>

static std::queue<service_sems*> wait_room_queue;
static std::queue<service_sems*> sofa_queue;
static std::queue<service_sems*> cashier_queue;
static std::queue<service_sems*> barber_queue;
bool is_ending = false;
void customer_receiver()
{
    log() << "customer receiver start running..." << std::endl;
    while (true)
    {
        sleep(RECEIVER_TIME_INTERVAL);
        new_customer_come.down();
        new_customer_in.up();
        for_sys_key_cnt.write(key_cnt);
        log() << "key_cnt: " << key_cnt << std::endl;
        auto service_obj = new service_sems;
        wait_room_queue.push(service_obj);
        int current_customer_cnt = 0;
        barber_shop_capacity.read(current_customer_cnt);
        current_customer_cnt++;
        barber_shop_capacity.write(current_customer_cnt);
        log() << "customer received and moved to wait room"
            << " id: " << service_obj->get_id() << std::endl;
        log() << "current customer count: " << current_customer_cnt <<
std::endl;
    }
}

```

```

}
void customer_mover()
{
    log() << "customer mover start running..." << std::endl;
    while (true)
    {
        sleep(MOVER_MOVE_TIME_INTERVAL);
        if (sofa_queue.size() < SOFA_SIZE)
        {
            if (!wait_room_queue.empty())
            {
                sofa_queue.push(wait_room_queue.front());
                wait_room_queue.front()->let_user_move_to_sofa();
                log() << "customer moved to sofa"
                    << " id: " << wait_room_queue.front()->get_id() << std::endl;
                wait_room_queue.pop();
            }
        }
    }
}

void cashier_mover()
{
    log() << "cashier mover start running..." << std::endl;
    while (true)
    {
        sleep(CASHIER_CASH_TIME_INTERVAL);
        if (!sofa_queue.empty())
        {
            barber_finish_service.down();
            cashier_queue.push(sofa_queue.front());
            cashier_queue.front()->support_service_for_user();
            log() << "customer finished service and moved to cashier"
                << " id: " << sofa_queue.front()->get_id() << std::endl;
            sofa_queue.pop();
        }
    }
}

void cashier_receiver()
{
    log() << "cashier receiver start running..." << std::endl;
    while (true)
    {
        if (!cashier_queue.empty())
        {

```

```

        sleep(RECEIVER_TIME_INTERVAL);
        barber_finish_receive_money.down();
        cashier_queue.front()->let_user_pay_money();
        log() << "customer paid money"
            << " id: " << cashier_queue.front()->get_id() << std::endl;
        cashier_queue.pop();
        the_man_who_is_paying_is_done.down();
        double received_money = 0;
        double current_money = 0;
        money_trade_buffer.read(received_money);
        sum_money.read(current_money);
        current_money += received_money;
        sum_money.write(current_money);
        log() << "received money: " << received_money << std::endl;
        log() << "current money: " << current_money << std::endl;
        barber_finish_receive_money.up();
    }
}

void shop_running_timer()
{
    log() << "shop running timer start running..." << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(SHOP_RUNNING_TIME));
    is_ending = true;
    is_closing.write(1);
}

void run()
{
    barber_shop_capacity.write(0);
    int status = 666;
    is_closing.write(status);
    log() << "barbershop start running..." << std::endl;
    std::thread customer_receiver_thread(customer_receiver);
    std::thread customer_mover_thread(customer_mover);
    std::thread cashier_mover_thread(cashier_mover);
    std::thread cashier_receiver_thread(cashier_receiver);
    std::thread shop_running_timer_thread(shop_running_timer);
    customer_mover_thread.detach();
    customer_receiver_thread.detach();
    cashier_mover_thread.detach();
    cashier_receiver_thread.detach();
    shop_running_timer_thread.detach();
    while (true)
    {

```



```

        if (is_ending)
        {
            log() << "barbershop is closing..." << std::endl;
            break;
        }
    }
}

int main()
{
    try
    {
        run();
    }
    catch (std::exception& e)
    {
        log() << e.what() << std::endl;
    }
    return 0;
}

```

3. barber.cpp

```

#include "common.h"
#include <exception>
#include <iostream>
#include <thread>
#include <unistd.h>
bool is_working = false;

void provide_service()
{
    while (true)
    {
        if (is_working)
        {
            continue;
        }
        needed_services_cnt.down();
        log() << "barber is providing a service..." << std::endl;
        is_working = true;
        sleep(BARBER_BARBE_TIME_INTERVAL);
        barber_finish_service.up();
        is_working = false;
    }
}

void cashier_service()

```

```

{
    while (true)
    {
        if (is_working)
        {
            continue;
        }
        needed_pay_money_cnt.down();
        is_someone_is_receiving_money.down();
        log() << "barber is taking money..." << std::endl;
        is_working = true;
        sleep(CASHIER_CASH_TIME_INTERVAL);
        barber_finish_receive_money.up();
        is_someone_is_receiving_money.up();
        is_working = false;
    }
}

void have_a_break()
{
    static bool is_resting = false;
    while (true)
    {
        sleep(REST_CHECK_TIME_INTERVAL);
        if (!is_working && !is_resting)
        {
            is_resting = true;
            log() << "barber taking a break..." << std::endl;
        }
        if (is_resting && is_working)
        {
            is_resting = false;
            log() << "barber stopped resting..." << std::endl;
        }
    }
}

bool is_closed = false;
void check_is_barber_closed()
{
    while (true)
    {
        sleep(1);
        int status = 1;
        is_closing.read(status);
        if (status != 666)
    }
}

```

```

        {
            is_closed = true;
        }
    }
}

void run()
{
    log() << "barber came to work..." << std::endl;
    int status = 0;
    is_closing.read(status);
    if (status != 666)
    {
        log() << "barbershop is closed, barber is leaving..." << std::endl;
        return;
    }
    std::thread provide_service_thread(provide_service);
    std::thread cashier_service_thread(cashier_service);
    std::thread have_a_break_thread(have_a_break);
    std::thread check_is_barber_closed_thread(check_is_barber_closed);
    provide_service_thread.detach();
    cashier_service_thread.detach();
    have_a_break_thread.detach();
    check_is_barber_closed_thread.detach();
    while (!is_closed)
    {
        sleep(1);
    }
    log() << "barbershop is closed, barber is leaving..." << std::endl;
}

int main()
{
    try
    {
        run();
    }
    catch (std::exception& e)
    {
        log() << e.what() << std::endl;
    }
}

```

4. user.cpp

```

#include "common.h"
#include <cstdlib>
#include <ctime>

```

```

#include <thread>
double money_for_haircut = 10.0;
bool is_closed = false;
bool end_consumption = false;
void check_is_barber_closed()
{
    while (true)
    {
        sleep(1);
        int status = 1;
        is_closing.read(status);
        if (status == 1)
        {
            is_closed = true;
            break;
        }
    }
}
void consume()
{
    log() << "spawned" << std::endl;
    is_someone_is_entring.down();
    int customer_count;
    barber_shop_capacity.read(customer_count);
    log() << "customer_count: " << customer_count << std::endl;
    int status = 0;
    is_closing.read(status);
    if (status != 666)
    {
        log() << "barber shop is closed, cannot enter" << std::endl;
        return;
    }

    if (customer_count >= BARBERSHOP_SIZE)
    {
        log() << "too many customers, this customer leaves" << std::endl;
        return;
    }
    else
    {
        log() << "capacity available, customer enters" << std::endl;
        new_customer_come.up();
        new_customer_in.down();
        for_sys_key_cnt.read(key_cnt);
    }
}

```

```

log() << "key_cnt: " << key_cnt << std::endl;
user_sems local_user;
is_someone_is_entring.up();
local_user.wait_for_move_to_sofa();
log() << "customer is seated" << std::endl;
log() << "customer is ready to cut hair" << std::endl;
local_user.wait_for_get_service();
log() << "customer finished getting hair cut" << std::endl;
log() << "customer is waiting for payment" << std::endl;
local_user.wair_for_give_money();
is_someone_is_paying.down();
money_trade_buffer.write(money_for_haircut);
is_someone_is_paying.up();
the_man_who_is_paying_is_done.up();
log() << "customer paid: " << money_for_haircut << std::endl;
log() << "customer is leaving" << std::endl;
int current_customer_count;
barber_shop_capacity.read(current_customer_count);
current_customer_count--;
barber_shop_capacity.write(current_customer_count);
end_consumption = true;
}
}
void run()
{
    std::thread check_is_barber_closed_thread(check_is_barber_closed);
    check_is_barber_closed_thread.detach();
    std::thread consume_thread(consume);
    consume_thread.detach();
    while (true)
    {
        if (end_consumption)
        {
            break;
        }
        if (is_closed)
        {
            log() << "barber shop is closed, have to end consumption" << std::endl;
            break;
        }
    }
}
int main()
{

```

```

try
{
    std::srand(std::time(0));
    double random_number = (std::rand() / (RAND_MAX + 1.0)) * 20.0;
    money_for_haircut = random_number;
    run();
}
catch (std::exception& e)
{
    log() << e.what() << std::endl;
}
return 0;
}

```

5. Makefile

```

hdrs = common.h
user_srcs = user.cpp
user_objs = user.o
barbershop_srcs = barbershop.cpp
barbershop_objs = barbershop.o
barber_srcs = barber.cpp
barber_objs = barber.o
all: user barbershop barber
$(user_objs): $(user_srcs) $(hdrs)
    g++ -c $(user_srcs) -o $(user_objs)
user: $(user_objs)
    g++ -o user $(user_objs)
$(barbershop_objs): $(barbershop_srcs) $(hdrs)
    g++ -c $(barbershop_srcs) -o $(barbershop_objs)
barbershop: $(barbershop_objs)
    g++ -o barbershop $(barbershop_objs)
$(barber_objs): $(barber_srcs) $(hdrs)
    g++ -c $(barber_srcs) -o $(barber_objs)
barber: $(barber_objs)
    g++ -o barber $(barber_objs)
clean:
    rm user *.o barbershop barber

```

为了便于我们进行测试，我们同样编写了 test.sh 和 endtest.sh 脚本，另外额外加了一个 easytest.sh 脚本用于测试功能。

6. easytest.sh

```

#!/bin/bash

mkdir 实验日志 2>/dev/null >> /dev/null
echo -e "Testing start running...\n\n"
make clean > /dev/null 2>/dev/null

```

```

echo -e "创建测试环境...\n\n"
echo -e "需要先清空 ipc 文件, 否则可能导致测试失败。是否继续?(y/n)"
read choice
if [ "$choice" == "y" ]; then
    ipcrm -a
else
    echo -e "测试环境创建失败, 请检查系统环境。\n\n"
    exit 1
fi
make all
touch barbershop.log
touch barber.log
touch user.log
echo -e "启动测试...\n\n"
current_time=$(date "+%Y-%m-%d-%H-%M-%S")
mkdir $current_time
echo -e "拉起理发店进程...\n"
./barbershop >> barbershop.log 2>> barbershop.log &
echo -e "拉起理发师进程...\n"
./barber >> barber.log 2>> barber.log &
echo -e "以下为测试中固定使用的进程:\n"
ps | grep barber
ps | grep barbershop
echo -e "开始随机唤起顾客进程...\n"
for i in $(seq 1 15); do
    echo -e "拉起顾客进程 $i...\n"
    ./user $i >> user.log 2>> user.log &
    sleep 1
done
mv *.log $current_time/
mv $current_time 实验日志
echo -e "测试结束, 请查看日志文件。\n\n"

```

7. endtest.sh

```

#!/bin/bash

echo -e "The following processes will be killed:\n"
ps | grep user
ps | grep barber
ps | grep barbershop
echo -e "\n"
pkill user
pkill barber
pkill barbershop
echo -e "\nAll done."

```

```
ipcrm -a
```

```
8. test.sh
```

```
#!/bin/bash
```

```
mkdir 实验日志 2>/dev/null >> /dev/null
```

```
echo -e "Testing start running...\n\n"
```

```
make clean > /dev/null 2>/dev/null
```

```
echo -e "创建测试环境...\n\n"
```

```
echo -e "需要先清空 ipc 文件, 否则可能导致测试失败。是否继续?(y/n)"
```

```
read choice
```

```
if [ "$choice" == "y" ]; then
```

```
    ipcrm -a
```

```
else
```

```
    echo -e "测试环境创建失败, 请检查系统环境.\n\n"
```

```
    exit 1
```

```
fi
```

```
make all
```

```
touch barbershop.log
```

```
touch barber1.log
```

```
touch barber2.log
```

```
touch barber3.log
```

```
echo -e "启动测试...\n\n"
```

```
current_time=$(date "+%Y-%m-%d-%H-%M-%S")
```

```
mkdir $current_time
```

```
echo -e "拉起理发店进程...\n"
```

```
./barbershop >> barbershop.log 2>> barbershop.log &
```

```
echo -e "拉起理发师进程...\n"
```

```
./barber >> barber1.log 2>> barber1.log &
```

```
./barber >> barber2.log 2>> barber2.log &
```

```
./barber >> barber3.log 2>> barber3.log &
```

```
echo -e "以下为测试中固定使用的进程:\n"
```

```
ps | grep barber
```

```
ps | grep barbershop
```

```
echo -e "开始随机唤起顾客进程...\n"
```

```
echo -e "注意测试仅持续 20 秒, 且顾客进程会在每 1 秒随机唤起随机数量(0-6)。若需要持续测试, 请自行修改脚本。"
```

```
user_cnt=0 # 总数, 用于修改日志文件名
```

```
for i in {1..20}
```

```
do
```

```
    random_num=$((RANDOM%7))
```

```
    for j in $(seq 1 $random_num)
```

```
    do
```

```
        user_cnt=$((user_cnt+1))
```

```
        echo -e "唤起顾客进程 $user_cnt...\n"
```



```
./user >> $current_time/user$user_cnt.log 2>>
$current_time/user$user_cnt.log &
done
sleep 1
done
sleep 2
echo -e "测试结束, 请查看日志文件.\n\n"
./endtest.sh
mv *.log $current_time/ 2>/dev/null >> /dev/null
mv $current_time 实验日志/ 2>/dev/null >> /dev/null#!/bin/bash

mkdir 实验日志 2>/dev/null >> /dev/null
echo -e "Testing start running...\n\n"
make clean > /dev/null 2>/dev/null
echo -e "创建测试环境...\n\n"
echo -e "需要先清空 ipc 文件, 否则可能导致测试失败。是否继续?(y/n)"
read choice
if [ "$choice" == "y" ]; then
    ipcrm -a
else
    echo -e "测试环境创建失败, 请检查系统环境.\n\n"
    exit 1
fi
make all
touch barbershop.log
touch barber1.log
touch barber2.log
touch barber3.log
echo -e "启动测试...\n\n"
current_time=$(date "+%Y-%m-%d-%H-%M-%S")
mkdir $current_time
echo -e "拉起理发店进程...\n"
./barbershop >> barbershop.log 2>> barbershop.log &
echo -e "拉起理发师进程...\n"
./barber >> barber1.log 2>> barber1.log &
./barber >> barber2.log 2>> barber2.log &
./barber >> barber3.log 2>> barber3.log &
echo -e "以下为测试中固定使用的进程:\n"
ps | grep barber
ps | grep barbershop
echo -e "开始随机唤起顾客进程...\n"
echo -e "注意测试仅持续 20 秒, 且顾客进程会在每 1 秒随机唤起随机数量(0-6)。若需
要持续测试, 请自行修改脚本。"
user_cnt=0 # 总数, 用于修改日志文件名
```

```

for i in {1..20}
do
    random_num=$((RANDOM%7))
    for j in $(seq 1 $random_num)
    do
        user_cnt=$((user_cnt+1))
        echo -e "唤起顾客进程 $user_cnt...\n"
        ./user >> $current_time/user$user_cnt.log 2>>
$current_time/user$user_cnt.log &
    done
    sleep 1
done
sleep 2
echo -e "测试结束, 请查看日志文件.\n\n"
./endtest.sh
mv *.log $current_time/ 2>/dev/null >> /dev/null
mv $current_time 实验日志/ 2>/dev/null >> /dev/null
#!/bin/bash

mkdir 实验日志 2>/dev/null >> /dev/null
echo -e "Testing start runnning...\n\n"
make clean > /dev/null 2>/dev/null
echo -e "创建测试环境...\n\n"
echo -e "需要先清空 ipc 文件, 否则可能导致测试失败。是否继续?(y/n)"
read choice
if [ "$choice" == "y" ]; then
    ipcrm -a
else
    echo -e "测试环境创建失败, 请检查系统环境.\n\n"
    exit 1
fi
make all
touch barbershop.log
touch barber1.log
touch barber2.log
touch barber3.log
echo -e "启动测试...\n\n"
current_time=$(date "+%Y-%m-%d-%H-%M-%S")
mkdir $current_time
echo -e "拉起理发店进程...\n"
./barbershop >> barbershop.log 2>> barbershop.log &
echo -e "拉起理发师进程...\n"
./barber >> barber1.log 2>> barber1.log &
./barber >> barber2.log 2>> barber2.log &

```

```

./barber >> barber3.log 2>> barber3.log &
echo -e "以下为测试中固定使用的进程:\n"
ps | grep barber
ps | grep barbershop
echo -e "开始随机唤起顾客进程...\n"
echo -e "注意测试仅持续 20 秒, 且顾客进程会在每 1 秒随机唤起随机数量(0-6)。若需
要持续测试, 请自行修改脚本。"
user_cnt=0 # 总数, 用于修改日志文件名
for i in {1..20}
do
    random_num=$((RANDOM%7))
    for j in $(seq 1 $random_num)
    do
        user_cnt=$((user_cnt+1))
        echo -e "唤起顾客进程 $user_cnt...\n"
        ./user >> $current_time/user$user_cnt.log 2>>
$current_time/user$user_cnt.log &
    done
    sleep 1
done
sleep 2
echo -e "测试结束, 请查看日志文件.\n\n"
./endtest.sh
mv *.log $current_time/ 2>/dev/null >> /dev/null
mv $current_time 实验日志/ 2>/dev/null >> /dev/null

```

我们现在就就可以直接执行这个测试脚本来获取实验日志了, 以下为独立实验的日志, 为了便于展示我们字号设置的较小:

首先是 barbershop.log

```

[ 2025-04-19 01:17:44 ] Pid : 194571, barbershop start running...
[ 2025-04-19 01:17:44 ] Pid : 194571, customer receiver start running...
[ 2025-04-19 01:17:44 ] Pid : 194571, customer mover start running...
[ 2025-04-19 01:17:44 ] Pid : 194571, cashier mover start running...
[ 2025-04-19 01:17:44 ] Pid : 194571, cashier receiver start running...
[ 2025-04-19 01:17:44 ] Pid : 194571, shop running timer start running...
[ 2025-04-19 01:17:45 ] Pid : 194571, key_cnt: 15
[ 2025-04-19 01:17:45 ] Pid : 194571, customer received and moved to wait room id: 0
[ 2025-04-19 01:17:45 ] Pid : 194571, current customer count: 2
[ 2025-04-19 01:17:46 ] Pid : 194571, customer moved to sofa id: 0
[ 2025-04-19 01:17:46 ] Pid : 194571, key_cnt: 18
[ 2025-04-19 01:17:46 ] Pid : 194571, customer received and moved to wait room id: 1
[ 2025-04-19 01:17:46 ] Pid : 194571, current customer count: 4
[ 2025-04-19 01:17:47 ] Pid : 194571, customer moved to sofa id: 1
[ 2025-04-19 01:17:47 ] Pid : 194571, key_cnt: 21
[ 2025-04-19 01:17:47 ] Pid : 194571, customer received and moved to wait room id: 2

```

```
[ 2025-04-19 01:17:47 ] Pid : 194571, current customer count: 6
[ 2025-04-19 01:17:47 ] Pid : 194571, customer finished service and moved to cashier id: 0
[ 2025-04-19 01:17:48 ] Pid : 194571, customer moved to sofa id: 2
[ 2025-04-19 01:17:48 ] Pid : 194571, customer paid money id: 0
[ 2025-04-19 01:17:48 ] Pid : 194571, key_cnt: 24
[ 2025-04-19 01:17:48 ] Pid : 194571, customer received and moved to wait room id: 3
[ 2025-04-19 01:17:48 ] Pid : 194571, current customer count: 8
[ 2025-04-19 01:17:48 ] Pid : 194571, customer finished service and moved to cashier id: 1
[ 2025-04-19 01:17:48 ] Pid : 194571, received money: 11.9914
[ 2025-04-19 01:17:48 ] Pid : 194571, current money: 11.9914
[ 2025-04-19 01:17:49 ] Pid : 194571, customer moved to sofa id: 3
[ 2025-04-19 01:17:49 ] Pid : 194571, key_cnt: 27
[ 2025-04-19 01:17:49 ] Pid : 194571, customer received and moved to wait room id: 4
[ 2025-04-19 01:17:49 ] Pid : 194571, current customer count: 9
[ 2025-04-19 01:17:49 ] Pid : 194571, customer finished service and moved to cashier id: 2
[ 2025-04-19 01:17:49 ] Pid : 194571, customer paid money id: 1
[ 2025-04-19 01:17:49 ] Pid : 194571, received money: 11.9914
[ 2025-04-19 01:17:49 ] Pid : 194571, current money: 23.9827
[ 2025-04-19 01:17:50 ] Pid : 194571, customer moved to sofa id: 4
[ 2025-04-19 01:17:50 ] Pid : 194571, key_cnt: 30
[ 2025-04-19 01:17:50 ] Pid : 194571, customer received and moved to wait room id: 5
[ 2025-04-19 01:17:50 ] Pid : 194571, current customer count: 10
[ 2025-04-19 01:17:50 ] Pid : 194571, customer finished service and moved to cashier id: 3
[ 2025-04-19 01:17:50 ] Pid : 194571, customer paid money id: 2
[ 2025-04-19 01:17:50 ] Pid : 194571, received money: 11.9914
[ 2025-04-19 01:17:50 ] Pid : 194571, current money: 35.9741
[ 2025-04-19 01:17:51 ] Pid : 194571, customer moved to sofa id: 5
[ 2025-04-19 01:17:51 ] Pid : 194571, key_cnt: 33
[ 2025-04-19 01:17:51 ] Pid : 194571, customer received and moved to wait room id: 6
[ 2025-04-19 01:17:51 ] Pid : 194571, current customer count: 11
[ 2025-04-19 01:17:51 ] Pid : 194571, customer finished service and moved to cashier id: 4
[ 2025-04-19 01:17:51 ] Pid : 194571, customer paid money id: 3
[ 2025-04-19 01:17:51 ] Pid : 194571, received money: 9.10601
[ 2025-04-19 01:17:51 ] Pid : 194571, current money: 45.0801
[ 2025-04-19 01:17:52 ] Pid : 194571, customer moved to sofa id: 6
[ 2025-04-19 01:17:52 ] Pid : 194571, key_cnt: 36
[ 2025-04-19 01:17:52 ] Pid : 194571, customer received and moved to wait room id: 7
[ 2025-04-19 01:17:52 ] Pid : 194571, current customer count: 12
[ 2025-04-19 01:17:52 ] Pid : 194571, customer finished service and moved to cashier id: 5
[ 2025-04-19 01:17:52 ] Pid : 194571, customer paid money id: 4
[ 2025-04-19 01:17:52 ] Pid : 194571, received money: 9.10601
[ 2025-04-19 01:17:52 ] Pid : 194571, current money: 54.1861
[ 2025-04-19 01:17:53 ] Pid : 194571, customer moved to sofa id: 7
[ 2025-04-19 01:17:53 ] Pid : 194571, key_cnt: 39
```

```
[ 2025-04-19 01:17:53 ] Pid : 194571, customer received and moved to wait room id: 8
[ 2025-04-19 01:17:53 ] Pid : 194571, current customer count: 13
[ 2025-04-19 01:17:53 ] Pid : 194571, customer finished service and moved to cashier id: 6
[ 2025-04-19 01:17:53 ] Pid : 194571, customer paid money id: 5
[ 2025-04-19 01:17:53 ] Pid : 194571, received money: 9.10601
[ 2025-04-19 01:17:53 ] Pid : 194571, current money: 63.2921
[ 2025-04-19 01:17:54 ] Pid : 194571, customer moved to sofa id: 8
[ 2025-04-19 01:17:54 ] Pid : 194571, key_cnt: 42
[ 2025-04-19 01:17:54 ] Pid : 194571, customer received and moved to wait room id: 9
[ 2025-04-19 01:17:54 ] Pid : 194571, current customer count: 14
[ 2025-04-19 01:17:54 ] Pid : 194571, customer finished service and moved to cashier id: 7
[ 2025-04-19 01:17:54 ] Pid : 194571, customer paid money id: 6
[ 2025-04-19 01:17:54 ] Pid : 194571, received money: 16.4142
[ 2025-04-19 01:17:54 ] Pid : 194571, current money: 79.7063
[ 2025-04-19 01:17:55 ] Pid : 194571, customer moved to sofa id: 9
[ 2025-04-19 01:17:55 ] Pid : 194571, key_cnt: 45
[ 2025-04-19 01:17:55 ] Pid : 194571, customer received and moved to wait room id: 10
[ 2025-04-19 01:17:55 ] Pid : 194571, current customer count: 15
[ 2025-04-19 01:17:55 ] Pid : 194571, customer finished service and moved to cashier id: 8
[ 2025-04-19 01:17:55 ] Pid : 194571, customer paid money id: 7
[ 2025-04-19 01:17:55 ] Pid : 194571, received money: 16.4142
[ 2025-04-19 01:17:55 ] Pid : 194571, current money: 96.1205
[ 2025-04-19 01:17:56 ] Pid : 194571, barbershop is closing...
```

然后是 barber1.log

```
[ 2025-04-19 01:17:44 ] Pid : 194572, barber came to work...
[ 2025-04-19 01:17:45 ] Pid : 194572, barber taking a break...
[ 2025-04-19 01:17:46 ] Pid : 194572, barber is providing a service...
[ 2025-04-19 01:17:47 ] Pid : 194572, barber stopped resting...
[ 2025-04-19 01:17:47 ] Pid : 194572, barber is taking money...
[ 2025-04-19 01:17:49 ] Pid : 194572, barber taking a break...
[ 2025-04-19 01:17:49 ] Pid : 194572, barber is providing a service...
[ 2025-04-19 01:17:50 ] Pid : 194572, barber stopped resting...
[ 2025-04-19 01:17:50 ] Pid : 194572, barber is taking money...
[ 2025-04-19 01:17:52 ] Pid : 194572, barber is providing a service...
[ 2025-04-19 01:17:53 ] Pid : 194572, barber taking a break...
[ 2025-04-19 01:17:53 ] Pid : 194572, barber is taking money...
[ 2025-04-19 01:17:54 ] Pid : 194572, barber stopped resting...
[ 2025-04-19 01:17:55 ] Pid : 194572, barber is providing a service...
[ 2025-04-19 01:17:56 ] Pid : 194572, barbershop is closed, barber is leaving...
```

然后是 barber2.log

```
[ 2025-04-19 01:17:44 ] Pid : 194573, barber came to work...
[ 2025-04-19 01:17:45 ] Pid : 194573, barber taking a break...
[ 2025-04-19 01:17:47 ] Pid : 194573, barber is providing a service...
[ 2025-04-19 01:17:48 ] Pid : 194573, barber stopped resting...
```

```
[ 2025-04-19 01:17:48 ] Pid : 194573, barber is taking money...
[ 2025-04-19 01:17:50 ] Pid : 194573, barber is providing a service...
[ 2025-04-19 01:17:50 ] Pid : 194573, barber taking a break...
[ 2025-04-19 01:17:50 ] Pid : 194573, barber stopped resting...
[ 2025-04-19 01:17:51 ] Pid : 194573, barber taking a break...
[ 2025-04-19 01:17:51 ] Pid : 194573, barber is taking money...
[ 2025-04-19 01:17:52 ] Pid : 194573, barber stopped resting...
[ 2025-04-19 01:17:53 ] Pid : 194573, barber is providing a service...
[ 2025-04-19 01:17:54 ] Pid : 194573, barber taking a break...
[ 2025-04-19 01:17:54 ] Pid : 194573, barber is taking money...
[ 2025-04-19 01:17:55 ] Pid : 194573, barber stopped resting...
[ 2025-04-19 01:17:56 ] Pid : 194573, barbershop is closed, barber is leaving...
```

然后是 barber3.log

```
[ 2025-04-19 01:17:44 ] Pid : 194574, barber came to work...
[ 2025-04-19 01:17:45 ] Pid : 194574, barber taking a break...
[ 2025-04-19 01:17:48 ] Pid : 194574, barber is providing a service...
[ 2025-04-19 01:17:49 ] Pid : 194574, barber stopped resting...
[ 2025-04-19 01:17:49 ] Pid : 194574, barber is taking money...
[ 2025-04-19 01:17:51 ] Pid : 194574, barber taking a break...
[ 2025-04-19 01:17:51 ] Pid : 194574, barber is providing a service...
[ 2025-04-19 01:17:52 ] Pid : 194574, barber stopped resting...
[ 2025-04-19 01:17:52 ] Pid : 194574, barber is taking money...
[ 2025-04-19 01:17:54 ] Pid : 194574, barber taking a break...
[ 2025-04-19 01:17:54 ] Pid : 194574, barber is providing a service...
[ 2025-04-19 01:17:55 ] Pid : 194574, barber stopped resting...
[ 2025-04-19 01:17:55 ] Pid : 194574, barber is taking money...
[ 2025-04-19 01:17:56 ] Pid : 194574, barbershop is closed, barber is leaving...
```

用户的日志因为是分别独立记录的，因此我们以本次测试为例，共有 69 个 user%d.txt，我们为了展示方便，不再使用 vscode 渲染的方式，而是使用 cat 命令来展示：

```
$ cat *.log
```

下方为筛选出的 user 的部分，顺序受 cat 访问顺序影响

```
[ 2025-04-19 01:17:47 ] Pid : 194641, spawned
[ 2025-04-19 01:17:47 ] Pid : 194641, customer_count: 6
[ 2025-04-19 01:17:47 ] Pid : 194641, capacity available, customer enters
[ 2025-04-19 01:17:54 ] Pid : 194641, key_cnt: 42
[ 2025-04-19 01:17:55 ] Pid : 194641, customer is seated
[ 2025-04-19 01:17:55 ] Pid : 194641, customer is ready to cut hair
[ 2025-04-19 01:17:56 ] Pid : 194641, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:47 ] Pid : 194642, spawned
[ 2025-04-19 01:17:47 ] Pid : 194642, customer_count: 6
[ 2025-04-19 01:17:47 ] Pid : 194642, capacity available, customer enters
[ 2025-04-19 01:17:55 ] Pid : 194642, key_cnt: 45
[ 2025-04-19 01:17:56 ] Pid : 194642, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:47 ] Pid : 194643, spawned
```

```
[ 2025-04-19 01:17:48 ] Pid : 194643, customer_count: 8
[ 2025-04-19 01:17:48 ] Pid : 194643, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194643, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:47 ] Pid : 194644, spawned
[ 2025-04-19 01:17:47 ] Pid : 194644, customer_count: 6
[ 2025-04-19 01:17:47 ] Pid : 194644, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194644, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:47 ] Pid : 194645, spawned
[ 2025-04-19 01:17:48 ] Pid : 194645, customer_count: 7
[ 2025-04-19 01:17:48 ] Pid : 194645, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194645, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:48 ] Pid : 194664, spawned
[ 2025-04-19 01:17:49 ] Pid : 194664, customer_count: 8
[ 2025-04-19 01:17:49 ] Pid : 194664, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194664, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:50 ] Pid : 194679, spawned
[ 2025-04-19 01:17:50 ] Pid : 194679, customer_count: 9
[ 2025-04-19 01:17:50 ] Pid : 194679, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194679, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:50 ] Pid : 194680, spawned
[ 2025-04-19 01:17:50 ] Pid : 194680, customer_count: 9
[ 2025-04-19 01:17:50 ] Pid : 194680, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194680, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:50 ] Pid : 194681, spawned
[ 2025-04-19 01:17:50 ] Pid : 194681, customer_count: 9
[ 2025-04-19 01:17:50 ] Pid : 194681, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194681, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:50 ] Pid : 194682, spawned
[ 2025-04-19 01:17:51 ] Pid : 194682, customer_count: 11
[ 2025-04-19 01:17:51 ] Pid : 194682, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194682, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:44 ] Pid : 194599, spawned
[ 2025-04-19 01:17:45 ] Pid : 194599, customer_count: 2
[ 2025-04-19 01:17:45 ] Pid : 194599, capacity available, customer enters
[ 2025-04-19 01:17:47 ] Pid : 194599, key_cnt: 21
[ 2025-04-19 01:17:48 ] Pid : 194599, customer is seated
[ 2025-04-19 01:17:48 ] Pid : 194599, customer is ready to cut hair
[ 2025-04-19 01:17:50 ] Pid : 194599, customer finished getting hair cut
[ 2025-04-19 01:17:50 ] Pid : 194599, customer is waiting for payment
[ 2025-04-19 01:17:50 ] Pid : 194599, customer paid: 11.9914
[ 2025-04-19 01:17:50 ] Pid : 194599, customer is leaving
[ 2025-04-19 01:17:51 ] Pid : 194697, spawned
[ 2025-04-19 01:17:51 ] Pid : 194697, customer_count: 10
[ 2025-04-19 01:17:51 ] Pid : 194697, capacity available, customer enters
```

```
[ 2025-04-19 01:17:56 ] Pid : 194697, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:51 ] Pid : 194698, spawned
[ 2025-04-19 01:17:51 ] Pid : 194698, customer_count: 10
[ 2025-04-19 01:17:51 ] Pid : 194698, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194698, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:51 ] Pid : 194699, spawned
[ 2025-04-19 01:17:51 ] Pid : 194699, customer_count: 10
[ 2025-04-19 01:17:51 ] Pid : 194699, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194699, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:51 ] Pid : 194700, spawned
[ 2025-04-19 01:17:52 ] Pid : 194700, customer_count: 12
[ 2025-04-19 01:17:52 ] Pid : 194700, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194700, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:53 ] Pid : 194722, spawned
[ 2025-04-19 01:17:54 ] Pid : 194722, customer_count: 14
[ 2025-04-19 01:17:54 ] Pid : 194722, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194722, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:53 ] Pid : 194723, spawned
[ 2025-04-19 01:17:53 ] Pid : 194723, customer_count: 12
[ 2025-04-19 01:17:53 ] Pid : 194723, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194723, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:54 ] Pid : 194734, spawned
[ 2025-04-19 01:17:54 ] Pid : 194734, customer_count: 13
[ 2025-04-19 01:17:54 ] Pid : 194734, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194734, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:54 ] Pid : 194735, spawned
[ 2025-04-19 01:17:54 ] Pid : 194735, customer_count: 13
[ 2025-04-19 01:17:54 ] Pid : 194735, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194735, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:54 ] Pid : 194736, spawned
[ 2025-04-19 01:17:54 ] Pid : 194736, customer_count: 13
[ 2025-04-19 01:17:54 ] Pid : 194736, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194736, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:54 ] Pid : 194737, spawned
[ 2025-04-19 01:17:55 ] Pid : 194737, customer_count: 13
[ 2025-04-19 01:17:55 ] Pid : 194737, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194737, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:44 ] Pid : 194600, spawned
[ 2025-04-19 01:17:45 ] Pid : 194600, customer_count: 2
[ 2025-04-19 01:17:45 ] Pid : 194600, capacity available, customer enters
[ 2025-04-19 01:17:46 ] Pid : 194600, key_cnt: 18
[ 2025-04-19 01:17:47 ] Pid : 194600, customer is seated
[ 2025-04-19 01:17:47 ] Pid : 194600, customer is ready to cut hair
[ 2025-04-19 01:17:48 ] Pid : 194600, customer finished getting hair cut
```



```
[ 2025-04-19 01:17:48 ] Pid : 194600, customer is waiting for payment
[ 2025-04-19 01:17:49 ] Pid : 194600, customer paid: 11.9914
[ 2025-04-19 01:17:49 ] Pid : 194600, customer is leaving
[ 2025-04-19 01:17:54 ] Pid : 194738, spawned
[ 2025-04-19 01:17:55 ] Pid : 194738, customer_count: 14
[ 2025-04-19 01:17:55 ] Pid : 194738, capacity available, customer enters
[ 2025-04-19 01:17:56 ] Pid : 194738, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:55 ] Pid : 194755, spawned
[ 2025-04-19 01:17:56 ] Pid : 194755, customer_count: 14
[ 2025-04-19 01:17:56 ] Pid : 194755, barber shop is closed, cannot enter
[ 2025-04-19 01:17:56 ] Pid : 194755, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:56 ] Pid : 194764, spawned
[ 2025-04-19 01:17:57 ] Pid : 194764, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:56 ] Pid : 194765, spawned
[ 2025-04-19 01:17:56 ] Pid : 194765, customer_count: 14
[ 2025-04-19 01:17:56 ] Pid : 194765, barber shop is closed, cannot enter
[ 2025-04-19 01:17:57 ] Pid : 194765, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:56 ] Pid : 194766, spawned
[ 2025-04-19 01:17:56 ] Pid : 194766, customer_count: 14
[ 2025-04-19 01:17:56 ] Pid : 194766, barber shop is closed, cannot enter
[ 2025-04-19 01:17:57 ] Pid : 194766, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:56 ] Pid : 194767, spawned
[ 2025-04-19 01:17:57 ] Pid : 194767, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:56 ] Pid : 194768, spawned
[ 2025-04-19 01:17:57 ] Pid : 194768, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:57 ] Pid : 194786, spawned
[ 2025-04-19 01:17:57 ] Pid : 194786, customer_count: 14
[ 2025-04-19 01:17:57 ] Pid : 194786, barber shop is closed, cannot enter
[ 2025-04-19 01:17:58 ] Pid : 194786, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:57 ] Pid : 194787, spawned
[ 2025-04-19 01:17:57 ] Pid : 194787, customer_count: 14
[ 2025-04-19 01:17:57 ] Pid : 194787, barber shop is closed, cannot enter
[ 2025-04-19 01:17:58 ] Pid : 194787, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:57 ] Pid : 194788, spawned
[ 2025-04-19 01:17:57 ] Pid : 194788, customer_count: 14
[ 2025-04-19 01:17:57 ] Pid : 194788, barber shop is closed, cannot enter
[ 2025-04-19 01:17:58 ] Pid : 194788, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:45 ] Pid : 194613, spawned
[ 2025-04-19 01:17:45 ] Pid : 194613, customer_count: 2
[ 2025-04-19 01:17:45 ] Pid : 194613, capacity available, customer enters
[ 2025-04-19 01:17:48 ] Pid : 194613, key_cnt: 24
[ 2025-04-19 01:17:49 ] Pid : 194613, customer is seated
[ 2025-04-19 01:17:49 ] Pid : 194613, customer is ready to cut hair
[ 2025-04-19 01:17:51 ] Pid : 194613, customer finished getting hair cut
```

```
[ 2025-04-19 01:17:51 ] Pid : 194613, customer is waiting for payment
[ 2025-04-19 01:17:51 ] Pid : 194613, customer paid: 9.10601
[ 2025-04-19 01:17:51 ] Pid : 194613, customer is leaving
[ 2025-04-19 01:17:57 ] Pid : 194789, spawned
[ 2025-04-19 01:17:58 ] Pid : 194789, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:57 ] Pid : 194790, spawned
[ 2025-04-19 01:17:57 ] Pid : 194790, customer_count: 14
[ 2025-04-19 01:17:57 ] Pid : 194790, barber shop is closed, cannot enter
[ 2025-04-19 01:17:58 ] Pid : 194790, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:57 ] Pid : 194791, spawned
[ 2025-04-19 01:17:58 ] Pid : 194791, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:58 ] Pid : 194810, spawned
[ 2025-04-19 01:17:58 ] Pid : 194810, customer_count: 14
[ 2025-04-19 01:17:58 ] Pid : 194810, barber shop is closed, cannot enter
[ 2025-04-19 01:17:59 ] Pid : 194810, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:58 ] Pid : 194811, spawned
[ 2025-04-19 01:17:58 ] Pid : 194811, customer_count: 14
[ 2025-04-19 01:17:58 ] Pid : 194811, barber shop is closed, cannot enter
[ 2025-04-19 01:17:59 ] Pid : 194811, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:58 ] Pid : 194812, spawned
[ 2025-04-19 01:17:58 ] Pid : 194812, customer_count: 14
[ 2025-04-19 01:17:58 ] Pid : 194812, barber shop is closed, cannot enter
[ 2025-04-19 01:17:59 ] Pid : 194812, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:58 ] Pid : 194813, spawned
[ 2025-04-19 01:17:58 ] Pid : 194813, customer_count: 14
[ 2025-04-19 01:17:58 ] Pid : 194813, barber shop is closed, cannot enter
[ 2025-04-19 01:17:59 ] Pid : 194813, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:58 ] Pid : 194814, spawned
[ 2025-04-19 01:17:59 ] Pid : 194814, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:59 ] Pid : 194831, spawned
[ 2025-04-19 01:17:59 ] Pid : 194831, customer_count: 14
[ 2025-04-19 01:17:59 ] Pid : 194831, barber shop is closed, cannot enter
[ 2025-04-19 01:18:00 ] Pid : 194831, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:59 ] Pid : 194832, spawned
[ 2025-04-19 01:18:00 ] Pid : 194832, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:45 ] Pid : 194614, spawned
[ 2025-04-19 01:17:46 ] Pid : 194614, customer_count: 4
[ 2025-04-19 01:17:46 ] Pid : 194614, capacity available, customer enters
[ 2025-04-19 01:17:50 ] Pid : 194614, key_cnt: 30
[ 2025-04-19 01:17:51 ] Pid : 194614, customer is seated
[ 2025-04-19 01:17:51 ] Pid : 194614, customer is ready to cut hair
[ 2025-04-19 01:17:53 ] Pid : 194614, customer finished getting hair cut
[ 2025-04-19 01:17:53 ] Pid : 194614, customer is waiting for payment
[ 2025-04-19 01:17:53 ] Pid : 194614, customer paid: 9.10601
```

```
[ 2025-04-19 01:17:53 ] Pid : 194614, customer is leaving
[ 2025-04-19 01:17:59 ] Pid : 194833, spawned
[ 2025-04-19 01:18:00 ] Pid : 194833, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:59 ] Pid : 194834, spawned
[ 2025-04-19 01:18:00 ] Pid : 194834, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:59 ] Pid : 194835, spawned
[ 2025-04-19 01:18:00 ] Pid : 194835, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194852, spawned
[ 2025-04-19 01:18:00 ] Pid : 194852, customer_count: 14
[ 2025-04-19 01:18:00 ] Pid : 194852, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194852, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194853, spawned
[ 2025-04-19 01:18:00 ] Pid : 194853, customer_count: 14
[ 2025-04-19 01:18:00 ] Pid : 194853, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194853, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194854, spawned
[ 2025-04-19 01:18:00 ] Pid : 194854, customer_count: 14
[ 2025-04-19 01:18:00 ] Pid : 194854, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194854, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194855, spawned
[ 2025-04-19 01:18:00 ] Pid : 194855, customer_count: 14
[ 2025-04-19 01:18:00 ] Pid : 194855, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194855, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194856, spawned
[ 2025-04-19 01:18:01 ] Pid : 194856, customer_count: 14
[ 2025-04-19 01:18:01 ] Pid : 194856, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194856, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:00 ] Pid : 194857, spawned
[ 2025-04-19 01:18:01 ] Pid : 194857, customer_count: 14
[ 2025-04-19 01:18:01 ] Pid : 194857, barber shop is closed, cannot enter
[ 2025-04-19 01:18:01 ] Pid : 194857, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:01 ] Pid : 194879, spawned
[ 2025-04-19 01:18:01 ] Pid : 194879, customer_count: 14
[ 2025-04-19 01:18:01 ] Pid : 194879, barber shop is closed, cannot enter
[ 2025-04-19 01:18:02 ] Pid : 194879, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:45 ] Pid : 194615, spawned
[ 2025-04-19 01:17:46 ] Pid : 194615, customer_count: 4
[ 2025-04-19 01:17:46 ] Pid : 194615, capacity available, customer enters
[ 2025-04-19 01:17:49 ] Pid : 194615, key_cnt: 27
[ 2025-04-19 01:17:50 ] Pid : 194615, customer is seated
[ 2025-04-19 01:17:50 ] Pid : 194615, customer is ready to cut hair
[ 2025-04-19 01:17:52 ] Pid : 194615, customer finished getting hair cut
[ 2025-04-19 01:17:52 ] Pid : 194615, customer is waiting for payment
[ 2025-04-19 01:17:52 ] Pid : 194615, customer paid: 9.10601
```

```
[ 2025-04-19 01:17:52 ] Pid : 194615, customer is leaving
[ 2025-04-19 01:18:01 ] Pid : 194880, spawned
[ 2025-04-19 01:18:02 ] Pid : 194880, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:01 ] Pid : 194881, spawned
[ 2025-04-19 01:18:02 ] Pid : 194881, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194894, spawned
[ 2025-04-19 01:18:02 ] Pid : 194894, customer_count: 14
[ 2025-04-19 01:18:02 ] Pid : 194894, barber shop is closed, cannot enter
[ 2025-04-19 01:18:03 ] Pid : 194894, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194895, spawned
[ 2025-04-19 01:18:02 ] Pid : 194895, customer_count: 14
[ 2025-04-19 01:18:02 ] Pid : 194895, barber shop is closed, cannot enter
[ 2025-04-19 01:18:03 ] Pid : 194895, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194896, spawned
[ 2025-04-19 01:18:03 ] Pid : 194896, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194897, spawned
[ 2025-04-19 01:18:03 ] Pid : 194897, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194898, spawned
[ 2025-04-19 01:18:03 ] Pid : 194898, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:02 ] Pid : 194899, spawned
[ 2025-04-19 01:18:03 ] Pid : 194899, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:03 ] Pid : 194918, spawned
[ 2025-04-19 01:18:03 ] Pid : 194918, customer_count: 14
[ 2025-04-19 01:18:03 ] Pid : 194918, barber shop is closed, cannot enter
[ 2025-04-19 01:18:04 ] Pid : 194918, barber shop is closed, have to end consumption
[ 2025-04-19 01:18:03 ] Pid : 194919, spawned
[ 2025-04-19 01:18:04 ] Pid : 194919, barber shop is closed, have to end consumption
[ 2025-04-19 01:17:46 ] Pid : 194628, spawned
[ 2025-04-19 01:17:46 ] Pid : 194628, customer_count: 4
[ 2025-04-19 01:17:46 ] Pid : 194628, capacity available, customer enters
[ 2025-04-19 01:17:51 ] Pid : 194628, key_cnt: 33
[ 2025-04-19 01:17:52 ] Pid : 194628, customer is seated
[ 2025-04-19 01:17:52 ] Pid : 194628, customer is ready to cut hair
[ 2025-04-19 01:17:54 ] Pid : 194628, customer finished getting hair cut
[ 2025-04-19 01:17:54 ] Pid : 194628, customer is waiting for payment
[ 2025-04-19 01:17:54 ] Pid : 194628, customer paid: 16.4142
[ 2025-04-19 01:17:54 ] Pid : 194628, customer is leaving
[ 2025-04-19 01:17:46 ] Pid : 194629, spawned
[ 2025-04-19 01:17:47 ] Pid : 194629, customer_count: 6
[ 2025-04-19 01:17:47 ] Pid : 194629, capacity available, customer enters
[ 2025-04-19 01:17:52 ] Pid : 194629, key_cnt: 36
[ 2025-04-19 01:17:53 ] Pid : 194629, customer is seated
[ 2025-04-19 01:17:53 ] Pid : 194629, customer is ready to cut hair
[ 2025-04-19 01:17:55 ] Pid : 194629, customer finished getting hair cut
```

```
[ 2025-04-19 01:17:55 ] Pid : 194629, customer is waiting for payment
[ 2025-04-19 01:17:55 ] Pid : 194629, customer paid: 16.4142
[ 2025-04-19 01:17:55 ] Pid : 194629, customer is leaving
[ 2025-04-19 01:17:47 ] Pid : 194640, spawned
[ 2025-04-19 01:17:47 ] Pid : 194640, customer_count: 6
[ 2025-04-19 01:17:47 ] Pid : 194640, capacity available, customer enters
[ 2025-04-19 01:17:53 ] Pid : 194640, key_cnt: 39
[ 2025-04-19 01:17:54 ] Pid : 194640, customer is seated
[ 2025-04-19 01:17:54 ] Pid : 194640, customer is ready to cut hair
[ 2025-04-19 01:17:56 ] Pid : 194640, barber shop is closed, have to end consumption
```

结论分析：

问题 1：总结和分析示例实验和独立实验中观察到的调试和运行信息，说明您对与解决非对称性互斥操作的算法有哪些新的理解和认识？

回答 1：在示例实验中，通过实现读者写者问题，观察到在不同的请求处理策略下，读者和写者进程的互斥与同步机制。读者优先策略允许多个读者同时访问共享资源，而写者需要等待所有读者释放资源后才能进行写操作。此策略在读操作频繁且写操作较少的场景下能有效提高资源利用率，但也可能导致写者饥饿现象。

在独立实验中，将信号量封装成类后，通过同步 key 值和 ftok 函数对齐信号量和共享内存号，进一步理解了信号量在进程间通信中的作用。同时，将整个架构划分为理发店、理发师和用户，通过消息传递实现进程间通信，体会到了消息传递在解决复杂并发场景下的优势。

问题 2：为什么会出现进程饥饿现象？

回答 2：进程饥饿现象通常由于资源分配策略不合理或进程调度算法缺陷导致。在示例实验中，若一直有读者请求到达，写者进程可能长时间无法获得对共享内存的独占访问权，从而被饥饿。这是因为读者优先策略下，写者必须等待所有读者完成读操作后才能执行，若读者不断有新请求，写者可能一直等待。

问题 3：本实验的饥饿现象是怎样表现的？

回答 3：在示例实验的读者写者模型中，饥饿现象表现为写者进程长时间无法执行写操作。当控制台输出持续显示读者进程的读操作完成信息，而写者进程的写操作信息很少或没有时，可推断写者可能被饥饿。在实验日志中，若写者进程的“write finished”输出远少于读者的“reader finished”输出，且写者请求与完成时间间隔过长，也说明饥饿现象存在。而在理发店独立实验里，饥饿现象的主要表现是理发师睡觉和用户进不去理发店。

问题 4：怎样解决并发进程间发生的饥饿现象？

回答 4：公平调度算法：采用公平的进程调度算法，如时间片轮转法，确保每个进程都能在一定时间内获得 CPU 执行权，避免某些进程一直被饥饿。

资源分配策略调整：合理调整资源分配策略，避免某些进程长期独占资源。如在读者写者问题中，可采用写者优先策略或读写公平策略，防止写者饥饿。

增加资源访问超时机制：为进程访问共享资源设置超时时间，若进程在超时时间内无法获得资源，则强制释放部分资源或调整优先级，让其他等待进程得以执行。

问题 5：您对于并发进程间使用消息传递解决进程通信问题有哪些新的理解和认识？

回答 5：通过实验，发现消息传递是一种直观且有效的进程间通信方式。它允许进程通过发送和接收消息来传递信息和同步状态，无需共享内存，降低了进程间耦合度，提高了系统的模块化和可维护性。但消息传递的效率可能低于共享内存，因为涉及系统调用和消息队列操作等开销。在设计时需权衡通信效率和系统复杂度。

结论：

本次实验深入理解了操作系统中进程互斥、同步和通信机制。通过读者写者问题的示例实验，掌握了信号量和共享内存的使用方法，学会了运用不同互斥算法解决实际问题，如读者优先策略在多读者少写者场景下的高效性，但也认识到其可能导致写者饥饿的局限性。独立实验中，将信号量和共享内存封装为类，并通过消息传递实现复杂场景下的进程间协作，提升了对 IPC 机制的灵活运用能力。同时，通过观察实验现象和分析日志，加深了对进程饥饿问题的理解，认识到合理设计资源分配和调度策略的重要性。实验过程中，代码调试和日志分析是解决问题的关键手段，培养了严谨的编程习惯和问题排查能力。总体而言，本次实验巩固了操作系统核心概念，增强了实践能力，为后续开发和研究打下坚实基础。