OS_Lab10

姓名	李懋
学号	2213189

1 实验题目

- 睡眠理发师问题
- 问题描述：
  - 理发店有1名理发师、1把理发椅和N（=3）把供等待理发的顾客暂坐的椅子。
  - 如果没有顾客，理发师则在理发椅上睡觉。
  - 当一个顾客来了，他叫醒理发师为其理发，理发的平均时间是5秒。
  - 如果理发师正在理发时有顾客来了，如果此时有空着的暂坐椅子，就坐下来等着，否则就离开理发师。
  - 累计随机来了30位顾客，最后理发师和顾客都结束。

2 实验目的

- 编写一个多线程的C程序，实现“睡眠理发师”问题
- 加深对多线程编程，线程同步，线程互斥等知识点的理解
- 具体的知识点
  1. **多线程编程**：使用POSIX线程（pthread）库创建和管理线程。
  2. **线程同步**：使用信号量（semaphore）和互斥锁（mutex）实现线程间的同步和互斥。
  3. **条件变量**：协调线程之间的等待和唤醒操作。
  4. **随机数生成**：模拟顾客到达和理发时间的随机性。
  5. **线程安全的输出**：确保多线程环境下输出的日志信息不乱序。

3 实验原理

1. 多线程：
   1. 使用 pthread_create() 创建线程，分别表示理发师和顾客
   2. 线程之间通过共享资源（等待椅，理发椅）进行交互
2. 线程同步：
   1. 使用信号量来控制对共享资源的访问
   2. 信号量用于表示等待椅的数量和理发师是否空闲
   3. 互斥锁用于保护临界区，确保同一时间只有一个线程修改共享数据
3. 条件变量
   1. 用于理发师和顾客之间协调，如理发师等待顾客到来，顾客等待理发师空闲
4. 随机数生成

1. 使用 srand() rand() 函数生成随机数，模拟顾客到达时间和理发时间

4 实验具体步骤

4.1 实现思路

1. **定义全局变量**：
   - `pthread_mutex_t salon_mutex`：用于保护共享资源的互斥锁。
   - `pthread_cond_t customer_cond, barber_cond`：用于理发师和顾客之间的条件变量。
   - `int waiting_customers`：记录当前等待的顾客数量。
   - `int processed_customers`：记录已处理的顾客数量。
2. **打印时间函数**：
   - `print_time`：用于按时间顺序输出日志信息，确保线程安全的输出。
3. **理发师线程函数**：
   - `barber`：理发师在没有顾客时进入睡眠状态，直到有顾客到来。理发师为顾客理发，理发时间随机。
4. **顾客线程函数**：
   - `customer`：顾客到达时，如果理发师正在理发且等待椅已满，则顾客离开；否则，顾客坐下等待。顾客理发时间随机。
5. **主函数**：
   - 初始化互斥锁和条件变量。
   - 创建理发师线程和顾客线程。
   - 等待所有顾客线程结束。
   - 销毁互斥锁和条件变量。

4.2 代码

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
#include <stdarg.h>  // 添加头文件

#define MAX_CHAIRS 3
#define TOTAL_CUSTOMERS 30

pthread_mutex_t salon_mutex;  // 互斥锁，保护共享资源
pthread_cond_t customer_cond, barber_cond;  // 条件变量，协调理发师和顾客
int waiting_customers = 0;  // 当前等待的顾客数量
int processed_customers = 0;  // 已处理的顾客数量

// 打印时间函数，确保线程安全的输出
void print_time(const char *message, ...) {
    struct timespec ts;
    clock_gettime(CLOCK_REALTIME, &ts);
```

```
    printf("%ld.%06ld ", (long)ts.tv_sec, (long)ts.tv_nsec/1000);
    va_list args;
    va_start(args, message);  // 使用 va_start
    vprintf(message, args);
    va_end(args);  // 使用 va_end
}

// 理发师线程函数
void *barber(void *arg) {
    while(processed_customers < TOTAL_CUSTOMERS) {
        pthread_mutex_lock(&salon_mutex);  // 加锁，进入临界区
        print_time("理发师准备进入临界区\n");
        while(waiting_customers == 0 && processed_customers < TOTAL_CUSTOMERS) {
            print_time("理发师进入睡眠\n");
            pthread_cond_wait(&customer_cond, &salon_mutex);  // 等待顾客到来
        }
        print_time("理发师进入临界区\n");
        if(processed_customers >= TOTAL_CUSTOMERS) {
            pthread_mutex_unlock(&salon_mutex);  // 解锁，离开临界区
            break;
        }
        waiting_customers--;  // 减少等待顾客数量
        processed_customers++;  // 增加已处理顾客数量
        print_time("理发师开始理发，顾客等待数: %d\n", waiting_customers);
        pthread_mutex_unlock(&salon_mutex);  // 解锁，离开临界区
        // 理发时间
        sleep(rand() % 5 + 1);
        pthread_mutex_lock(&salon_mutex);  // 加锁，进入临界区
        print_time("理发师结束理发\n");
        pthread_mutex_unlock(&salon_mutex);  // 解锁，离开临界区
        pthread_cond_signal(&barber_cond);  // 唤醒等待的顾客
    }
    return NULL;
}

// 顾客线程函数
void *customer(void *arg) {
    int customer_id = *(int*)arg;
    free(arg);
    // 顾客到达
    sleep(rand() % 5); // 随机到达时间
    pthread_mutex_lock(&salon_mutex);  // 加锁，进入临界区
    print_time("顾客%d 准备进入临界区\n", customer_id);
    print_time("顾客%d 进入临界区\n", customer_id);
    if(waiting_customers >= MAX_CHAIRS) {
        print_time("顾客%d 离开，没有空椅子\n", customer_id);
        pthread_mutex_unlock(&salon_mutex);  // 解锁，离开临界区
    } else {
        print_time("顾客%d 进店，等待理发，顾客等待数: %d\n", customer_id,
waiting_customers);
        waiting_customers++;  // 增加等待顾客数量
        pthread_cond_signal(&customer_cond);  // 唤醒理发师
        pthread_cond_wait(&barber_cond, &salon_mutex);  // 等待理发师
        pthread_mutex_unlock(&salon_mutex);  // 解锁，离开临界区
        // 理发时间
        sleep(rand() % 5 + 1);
```

```
        print_time("顾客%d 理发完毕，离开\n", customer_id);
    }
    return NULL;  // 添加返回值
}

// 主函数
int main() {
    srand(time(NULL));  // 初始化随机数种子
    pthread_mutex_init(&salon_mutex, NULL);  // 初始化互斥锁
    pthread_cond_init(&customer_cond, NULL);  // 初始化顾客条件变量
    pthread_cond_init(&barber_cond, NULL);  // 初始化理发师条件变量

    pthread_t barber_thread;
    pthread_create(&barber_thread, NULL, barber, NULL);  // 创建理发师线程

    pthread_t customers[TOTAL_CUSTOMERS];
    for(int i = 0; i < TOTAL_CUSTOMERS; i++) {
        int *id = (int *)malloc(sizeof(int));  // 显式转换为 int*
        *id = i+1;
        pthread_create(&customers[i], NULL, customer, id);  // 创建顾客线程
    }

    for(int i = 0; i < TOTAL_CUSTOMERS; i++) {
        pthread_join(customers[i], NULL);  // 等待顾客线程结束
    }
    pthread_cancel(barber_thread);  // 取消理发师线程
    pthread_join(barber_thread, NULL);  // 等待理发师线程结束

    pthread_mutex_destroy(&salon_mutex);  // 销毁互斥锁
    pthread_cond_destroy(&customer_cond);  // 销毁顾客条件变量
    pthread_cond_destroy(&barber_cond);  // 销毁理发师条件变量

    return 0;
}
```

4.3 实验结果

```
leemao2213189@leemao2213189-VMware-Virtual-Platform:~/Documents/OS_Lab/Lab10$ gcc test.cpp -o test
leemao2213189@leemao2213189-VMware-Virtual-Platform:~/Documents/OS_Lab/Lab10$ ./test
1732869867.366366 理发师准备进入临界区
1732869867.366620 理发师进入睡眠
1732869867.369051 顾客15 准备进入临界区
1732869867.369086 顾客15 进入临界区
1732869867.369097 顾客15 进店， 等待理发， 顾客等待数: 0
1732869867.369122 理发师进入临界区
1732869867.369134 理发师开始理发， 顾客等待数: 0
1732869867.370112 顾客17 准备进入临界区
1732869867.370187 顾客17 进入临界区
1732869867.370199 顾客17 进店， 等待理发， 顾客等待数: 0
1732869867.370650 顾客19 准备进入临界区
1732869867.370720 顾客19 进入临界区
1732869867.370732 顾客19 进店， 等待理发， 顾客等待数: 1
1732869867.371031 顾客22 准备进入临界区
1732869867.371048 顾客22 进入临界区
1732869867.371057 顾客22 进店， 等待理发， 顾客等待数: 2
1732869867.371117 顾客26 准备进入临界区
1732869867.371137 顾客26 进入临界区
1732869867.371140 顾客26 离开， 没有空椅子
1732869867.371209 顾客28 准备进入临界区
1732869867.371223 顾客28 进入临界区
1732869867.371233 顾客28 离开， 没有空椅子
1732869868.369527 理发师结束理发
1732869868.369559 理发师准备进入临界区
1732869868.369561 理发师进入临界区
1732869868.369562 理发师开始理发， 顾客等待数: 2
1732869868.369590 顾客10 准备进入临界区
1732869868.369603 顾客10 进入临界区
1732869868.369611 顾客10 进店， 等待理发， 顾客等待数: 2
1732869868.369636 顾客8 准备进入临界区
1732869868.369869 顾客8 进入临界区
1732869868.370069 顾客8 离开， 没有空椅子
1732869868.370138 顾客16 准备进入临界区
1732869868.370218 顾客16 进入临界区
1732869868.370227 顾客16 离开， 没有空椅子
1732869868.370551 顾客11 准备进入临界区
1732869868.370809 顾客11 进入临界区
1732869868.370819 顾客11 离开， 没有空椅子
1732869868.371233 顾客25 准备进入临界区
1732869868.371306 顾客25 进入临界区
1732869868.371316 顾客25 离开， 没有空椅子
1732869869.367931 顾客3 准备进入临界区
1732869869.368642 顾客3 进入临界区
1732869869.368670 顾客3 离开， 没有空椅子
1732869869.369364 顾客5 准备进入临界区
```

```
1732869870.369476 顾客6 离开， 没有空椅子
1732869870.369537 顾客14 准备进入临界区
1732869870.369564 顾客14 进入临界区
1732869870.369570 顾客14 离开， 没有空椅子
1732869870.369722 顾客13 准备进入临界区
1732869870.369747 顾客13 进入临界区
1732869870.369755 顾客13 离开， 没有空椅子
1732869870.371028 顾客21 准备进入临界区
1732869870.371070 顾客21 进入临界区
1732869870.371075 顾客21 离开， 没有空椅子
1732869870.371382 顾客20 准备进入临界区
1732869870.371507 顾客20 进入临界区
1732869870.371518 顾客20 离开， 没有空椅子
1732869870.371702 顾客30 准备进入临界区
1732869870.371748 顾客30 进入临界区
1732869870.371750 顾客30 离开， 没有空椅子
1732869870.371871 顾客29 准备进入临界区
1732869870.371892 顾客29 进入临界区
1732869870.371901 顾客29 离开， 没有空椅子
1732869871.367846 顾客2 准备进入临界区
1732869871.368054 顾客2 进入临界区
1732869871.368069 顾客2 离开， 没有空椅子
1732869871.378304 顾客9 准备进入临界区
1732869871.378474 顾客9 进入临界区
1732869871.378495 顾客9 离开， 没有空椅子
1732869871.378855 1732869871.378862 顾客15 理发完毕， 离开
顾客7 准备进入临界区
1732869871.379509 顾客7 进入临界区
1732869871.379515 顾客7 离开， 没有空椅子
1732869873.371544 理发师结束理发
1732869873.371692 理发师准备进入临界区
1732869873.371710 理发师进入临界区
1732869873.371721 理发师开始理发， 顾客等待数: 2
1732869875.372135 顾客17 理发完毕， 离开
1732869878.373764 理发师结束理发
1732869878.373786 理发师准备进入临界区
1732869878.373787 理发师进入临界区
1732869878.373788 理发师开始理发， 顾客等待数: 1
1732869882.381725 顾客19 理发完毕， 离开
1732869883.377651 理发师结束理发
1732869883.377789 理发师准备进入临界区
1732869883.377795 理发师进入临界区
1732869883.377798 理发师开始理发， 顾客等待数: 0
1732869886.377961 顾客22 理发完毕， 离开
1732869886.378208 理发师结束理发
1732869886.378279 理发师准备进入临界区
1732869886.378285 理发师进入睡眠
1732869888.382394 顾客10 理发完毕， 离开
leemao2213189@leemao2213189-VMware-Virtual-Platform:~/Documents/OS_Lab/Lab10$
```

5 总结

1. 通过本次实验，我成功实现了“睡眠理发师问题”的多线程模拟。
2. 程序中使用了互斥锁和条件变量来确保线程间的正确同步和互斥。顾客和理发师的行为按时间顺序输出，确保了行为的可见性和可追溯性。
3. 实验过程中，我学习了如何使用POSIX线程进行多线程编程，并掌握了线程同步的基本方法。
4. 通过调试和测试，我确保了程序的正确性和 robustness，达到了实验的目的。
5. 具体来说，实验中遇到的主要问题是如何确保线程安全的输出和正确管理线程的同步，通过查阅资料和调试代码，我成功解决了这些问题。实验过程中，我深刻理解了线程同步的重要性，避免了竞态条件和忙等待的发生。
6. 此外，通过模拟随机事件，我学会了如何在程序中引入随机性，使程序行为更加接近现实世界。
7. 总之，本次实验不仅巩固了我对多线程编程的理解，还提高了我解决实际问题的能力。