## 实验报告：多线程下的理发师问题

1. 概述

本实验通过C语言实现了多线程环境下的理发师问题，涉及到一个理发店，包括一个理发师、一张理发椅以及N（本实验中设定为5）把等待理发的顾客椅。当没有顾客时，理发师会在理发椅上睡觉。当顾客到来时，他们会唤醒理发师进行理发，理发的平均时间为3秒。

1. 工具和依赖

GCC（GNU Compiler Collection）：确保已在系统上安装。

sudo apt-get install build-essential

编译

使用GCC和Make工具编译程序。

gcc -o barber\_shop barber\_shop.c -lpthread

执行

运行编译后的程序。

./barber\_shop

1. 实现细节

* POSIX线程和信号量

pthread\_create()

使用pthread\_create()函数为理发师和顾客创建线程。该函数需要传递线程标识符、线程属性、线程执行的函数以及传递给函数的参数。在这里，通过该函数创建了理发师和顾客的线程。

sem\_init()

sem\_init()函数用于初始化用于同步的信号量。在本实验中，使用了三个信号量：barber、customers和mutexlock。barber用于通知理发师是否有顾客到来，customers用于通知顾客是否可以理发，mutexlock是一个互斥锁，用于保护共享资源。

sem\_wait() 和 sem\_post()

sem\_wait()和sem\_post()是使用信号量进行同步的关键函数。sem\_wait()会阻塞当前线程，直到信号量的值大于0。sem\_post()则增加信号量的值，唤醒等待的线程。这两个函数保证了对关键资源的互斥访问和同步操作。

* 理发师进程

理发师的睡眠和工作

理发师进程通过一个无限循环实现，一开始它通过sem\_wait(&customers)等待顾客的到来。如果没有顾客，理发师会一直处于休眠状态。一旦有顾客到来，它会减少等待顾客的数量，然后通过sem\_post(&barber)通知有顾客需要理发，最后进入剪发的工作状态。

* 互斥锁保护

在进入和离开临界区（修改共享变量）时，使用了互斥锁mutexlock。这确保了在修改等待顾客数量时的互斥操作，防止多个线程同时修改导致竞态条件。

* 顾客进程

顾客到来和等待

顾客进程也通过一个无限循环实现，模拟了顾客的随机到来。当顾客到来时，它首先使用互斥锁mutexlock进入临界区，判断是否有空椅子。如果有空椅子，顾客会将自己加入等待队列，增加等待顾客的数量，并通过sem\_post(&customers)通知理发师有顾客等待。

顾客离开和剪发

如果没有空椅子，顾客会释放互斥锁，表示没有等待的位置，然后离开理发店。当理发师通知有空位时，顾客会通过sem\_wait(&barber)等待理发师剪发，然后执行剪发操作。

* 互斥锁（Mutex Lock）

互斥锁的作用

互斥锁mutexlock用于保护共享资源，如等待顾客数量的变量waiting。在进入和离开临界区时，须先获取互斥锁，以确保同时只有一个线程可以访问临界区，避免竞态条件。

* 随机函数

rand() 函数的使用

rand()函数用于生成随机数，模拟了顾客到达的随机时间。通过调用srand(time(0))初始化随机数生成器，确保每次运行程序时得到不同的随机数序列。

* 输出

行为序列的输出

程序通过标准输出输出每位顾客和理发师的行为序列。输出包括了进入和离开临界区、理发师剪发以及顾客行为等内容。这确保了对程序执行过程的清晰记录。

1. 心得体会

本程序通过C语言实现了多线程环境下的理发师问题。通过使用多线程和信号量，模拟了理发师问题的场景。对于理发店的共享资源，如等待顾客数量，通过互斥锁的使用得以保护，加深了对多线程中的同步临界资源的理解，以及熟悉了同步方式信号量、互斥锁的使用。

源代码

// 包含必要的库

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#include <unistd.h>

#include <iostream>

#include <queue>

using namespace std;

// 定义常量和变量

#define CHAIRS 5

int waiting = 0;

sem\_t barber, customers, mutexlock;

queue<int> wait\_queue;

// 函数声明：理发师和顾客进程

void\* barber\_process(void\*);

void\* customer\_process(void\*);

int main() {

// 信号量初始化

sem\_init(&barber, 0 ,0);

sem\_init(&customers, 0, 0);

sem\_init(&mutexlock, 0, 1);

// 线程句柄

pthread\_t p\_barber;

pthread\_t p\_customers[20];

// 创建理发师线程

pthread\_create(&p\_barber, nullptr, barber\_process, nullptr);

// 创建顾客线程，并设置随机到达时间

for (int i = 0; i < 20; i++) {

pthread\_create(&p\_customers[i], nullptr, customer\_process, &i);

srand(time(0));

sleep(rand() % 2 + 1);

}

// 加入顾客线程

for (int i = 0; i < 20; i++) {

pthread\_join(p\_customers[i], nullptr);

}

// 等待一段时间以完成剩余的理发

sleep(5);

return 0;

}

// 实现理发师进程

void\* barber\_process(void\*) {

while (true) {

sem\_wait(&customers); // 如果没有顾客则休眠

sem\_wait(&mutexlock); // 进入临界区

waiting = waiting - 1; // 减少等待顾客的数量

int id = wait\_queue.front();

wait\_queue.pop();

sem\_post(&barber); // 理发师准备好剪发

sem\_post(&mutexlock); // 退出临界区

cout << "理发师为顾客 " << id << " 剪发" << endl;

sleep(3); // 模拟剪发时间

}

}

// 实现顾客进程

void\* customer\_process(void\* p) {

int id = \*(int\*)p;

sem\_wait(&mutexlock); // 进入临界区

if (waiting < CHAIRS) {

wait\_queue.push(id);

waiting = waiting + 1;

sem\_post(&customers); // 如果需要，唤醒理发师

sem\_post(&mutexlock); // 退出临界区

sem\_wait(&barber); // 如果理发师忙碌，顾客休眠

cout << "顾客 " << id << " 正在接受理发师的服务" << endl;

cout << "等待理发的顾客数： " << waiting << " 位" << endl << endl;

} else {

sem\_post(&mutexlock); // 退出临界区

cout << "门外的顾客 " << id << " 看见没有空椅子就离开了" << endl << endl;

}

pthread\_exit(0);

}