

四川大学计算机学院、软件学院

实验报告

学号： 2022141460180 姓名： 封欢欢 专业： 计算机科学与技术 班级： 行政4班 第 14-15 周

课程名称	操作系统课程设计	实验课时	4 小时
实验项目	奖励实验	实验时间	第 14 周到第 15 周
实验目的	运用操作系统的相关知识(作业调度、互斥访问等)解决一个现实使用场景的问题；		
实验环境	HUAWEI 电脑 Clion2024.1.1		
实验内容（算法、程序、步骤和方法）	<p>实验内容：</p> <p>打车软件分配订单的调度方法</p> <p>在打车软件中，我们需要根据乘客和司机的位置，选择最佳的匹配算法，提高车辆周转效率，减少客户的等待时间。</p> <p>因此在本模型中，我们做出一些简化：</p> <p>对于司机来说，他们不想在接客途中走太远（这段距离是赚不到钱的），但是并不关心乘客到哪里。对于乘客来说，路上花费的时间是一定的，他们能做的就是通过最小化等车时间来尽快到达目的地。并且我们认为，两点间花费的时间与距离正相关。在城市中，道路的水平或者竖直的，两点的距离为曼哈顿距离，即$AB = abs(x_a - x_b) + abs(y_a - y_b)$</p> <p>从某种意义上说，司机和乘客的需求是相同的，也就是找到离自己最近的匹配。本着为顾客服务的思想，我们的程序将以顾客的视角寻找最近的车辆，因此司机不一定会得到最近的乘客。</p> <p>于是在本次实验中，我们需要运用操作系统中的作业调度和进程同步相关的思想：</p> <ul style="list-style-type: none">• 作业调度，当乘客发出请求后，我们会根据先来先服务（FCFS），按顺序为乘客分配；而对于每一个乘客，我们采用类似于高响应比的算法的最近匹配算法。• 进程同步：因为每个司机在同一时刻只能为一个乘客服务，因此司机作为临界资源，必须保证互斥访问，同时，当没有司机空闲时，乘客只能等待。 <p>程序：</p>		

```

#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <fstream>
#include <queue>
#include <vector>
#include <cmath>
#include <utility>

using namespace std;

#define sleep(n) Sleep(n*1000)

struct ThreadInfo {
    int tid;          // 线程 ID
    char role;        // 扮演角色 P or C
    double delay;     // 线程延迟
    double persist;   // 线程读写操作持续时间
    int x, y;         // 乘客或司机的位置
};

HANDLE mutex;
HANDLE driverAvailable;
HANDLE passengerWaiting;

queue<ThreadInfo*> passengerQueue;
priority_queue<pair<int, ThreadInfo*>, vector<pair<int, ThreadInfo*>>, greater<pair<int, ThreadInfo*>>>
driverQueue;

int calculateDistance(ThreadInfo* passenger, ThreadInfo* driver) {
    return abs(passenger->x - driver->x) + abs(passenger->y - driver->y);
}

void DriversThread(LPVOID lpParam) {
    ThreadInfo* info = (ThreadInfo*)lpParam;
    sleep(info->delay); // 模拟延时时间
    printf("Driver thread %d is ready to take a ride request.\n", info->tid);

    WaitForSingleObject(mutex, INFINITE);
    driverQueue.push({info->tid, info});
    ReleaseSemaphore(driverAvailable, 1, NULL);
    ReleaseMutex(mutex);
}

```

```

// 等待乘客匹配
WaitForSingleObject(passengerWaiting, INFINITE);

WaitForSingleObject(mutex, INFINITE);
// 匹配乘客
if (!passengerQueue.empty()) {
    ThreadInfo* passenger = passengerQueue.front();
    passengerQueue.pop();
    ReleaseMutex(mutex);

    printf("Driver thread %d is picking up
passenger %d.\n", info->tid, passenger->tid);
    sleep(info->persist); // 模拟持续时间
    printf("Driver thread %d finished the ride.\n",
info->tid);
} else {
    ReleaseMutex(mutex);
}
}

void PassengersThread(LPVOID lpParam) {
    ThreadInfo* info = (ThreadInfo*)lpParam;
    sleep(info->delay); // 模拟延时时间
    printf("Passenger thread %d requests a ride.\n",
info->tid);

    WaitForSingleObject(mutex, INFINITE);
    passengerQueue.push(info);
    ReleaseSemaphore(passengerWaiting, 1, NULL);
    ReleaseMutex(mutex);

    // 等待司机匹配
    WaitForSingleObject(driverAvailable, INFINITE);

    WaitForSingleObject(mutex, INFINITE);
    // 匹配司机
    if (!driverQueue.empty()) {
        auto driverPair = driverQueue.top();
        driverQueue.pop();
        ReleaseMutex(mutex);

        printf("Passenger thread %d is being served by
driver %d.\n", info->tid, driverPair.second->tid);
        sleep(info->persist); // 模拟持续时间
        printf("Passenger thread %d finished the ride.\n",
info->tid);
    }
}

```

```

    } else {
        ReleaseMutex(mutex);
    }
}

int main() {
    DWORD n_thread = 0;        // 线程数目
    DWORD thread_ID;           // 线程 ID
    // 线程对象数组
    HANDLE h_thread[20];
    ThreadInfo thread_info[20];

    // 创建信号量和互斥量
    driverAvailable = CreateSemaphore(NULL, 0, 10, NULL);
    passengerWaiting = CreateSemaphore(NULL, 0, 10, NULL);
    mutex = CreateMutex(NULL, FALSE, NULL);


    // 读取输入文件
    cout << "Drivers and Passengers simulation:" << endl;
    cout << endl;
    ifstream inFile;
    inFile.open("../data2.txt");
    if (!inFile) {
        printf("Error opening file!\n");
        return -1;
    }

    string s;
    getline(inFile,s);
    while (inFile >> thread_info[n_thread].tid) {
        inFile >> thread_info[n_thread].role;
        inFile >> thread_info[n_thread].delay;
        inFile >> thread_info[n_thread].persist;
        inFile >> thread_info[n_thread].x;
        inFile >> thread_info[n_thread].y;
        n_thread++;
    }
    inFile.close();

    // 创建线程
    for (int i = 0; i < n_thread; i++) {
        if (thread_info[i].role == 'P' || thread_info[i].role
== 'p') {
            h_thread[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE) (PassengersThread),
&thread_info[i], 0, &thread_ID);
        } else if (thread_info[i].role == 'D' ||

```

	<pre>thread_info[i].role == 'd') { h_thread[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) (DriversThread), &thread_info[i], 0, &thread_ID); } // 等待所有线程结束 WaitForMultipleObjects(n_thread, h_thread, TRUE, INFINITE); cout << endl; printf("All drivers and passengers have finished their operations!\n"); _getch(); return 0; }</pre>																																																	
数据记录 和计算	<p>其中 data2.txt 内容如下：</p> <table><tr><th>1</th><th>id</th><th>role</th><th>delay</th><th>persist</th><th>x</th><th>y</th></tr><tr><td>2</td><td>1</td><td>P</td><td>3</td><td>5</td><td>0</td><td>0</td></tr><tr><td>3</td><td>1</td><td>D</td><td>2</td><td>2</td><td>1</td><td>7</td></tr><tr><td>4</td><td>2</td><td>P</td><td>5</td><td>1</td><td>7</td><td>0</td></tr><tr><td>5</td><td>2</td><td>D</td><td>6</td><td>2</td><td>8</td><td>8</td></tr><tr><td>6</td><td>3</td><td>D</td><td>1</td><td>4</td><td>3</td><td>7</td></tr><tr><td>7</td><td>3</td><td>P</td><td>2</td><td>9</td><td>2</td><td>0</td></tr></table>	1	id	role	delay	persist	x	y	2	1	P	3	5	0	0	3	1	D	2	2	1	7	4	2	P	5	1	7	0	5	2	D	6	2	8	8	6	3	D	1	4	3	7	7	3	P	2	9	2	0
1	id	role	delay	persist	x	y																																												
2	1	P	3	5	0	0																																												
3	1	D	2	2	1	7																																												
4	2	P	5	1	7	0																																												
5	2	D	6	2	8	8																																												
6	3	D	1	4	3	7																																												
7	3	P	2	9	2	0																																												

<p>结 论 (结 果)</p>	<div data-bbox="502 197 1428 974"><pre>E:\ClionProject\cmake-build-debug\ClionProject.exe Drivers and Passengers simulation: Driver thread 3 is ready to take a ride request. Driver thread 1 is ready to take a ride request. Passenger thread 3 requests a ride. Passenger thread 3 is being served by driver 1. Driver thread 3 is picking up passenger 3. Passenger thread 1 requests a ride. Passenger thread 1 is being served by driver 3. Driver thread 1 is picking up passenger 1. Passenger thread 2 requests a ride. Driver thread 1 finished the ride. Driver thread 2 is ready to take a ride request. Driver thread 2 is picking up passenger 2. Passenger thread 2 is being served by driver 2. Driver thread 3 finished the ride. Passenger thread 2 finished the ride. Driver thread 2 finished the ride. Passenger thread 1 finished the ride. Passenger thread 3 finished the ride. All drivers and passengers have finished their operations!</pre></div> <p>发现输出结果和理论结果一直，实验成功。</p>
<p>小 结</p>	<p>通过此次实验，加深了我对作业调度、互斥访问的理解，并且发现操作系统的相关知识的思想在现实使用场景中的应用也屡见不鲜；</p>
<p>指导老师评议</p>	<div data-bbox="544 1487 670 1520">成绩评定：</div> <div data-bbox="1023 1487 1204 1520">指导教师签名：</div>