1. 什么是进程同步，为什么要引入进程同步?

进程同步是指多个并发进程或者线程对共享资源的访问顺序，防止访问的时候发生资源冲突。

引入进程同步的原因是在多个进程访问同一个资源的时候，避免对某一个资源的某个数据的混乱修改和使资源数据错误或者不一致的情况。

1. 同步机制应该遵循的原则是什么？为什么要遵循这些原则（可以通过举例进行解释）

互斥性，可见性，有序性 ，不饥饿性。

互斥性：同一时间只能有一个进程/线程访问共享资源。

可见性：一个线程对共享资源的修改对于其他线程必须是可见的。

有序性：线程的执行必须按照一定的顺序来进行，不能出现混乱的情况。

不饥饿性：所有线程都应该有公平的机会访问共享资源，而不是被其他线程永久地阻塞。

1. 什么是信号量，为什么引入信号量机制？（李老师的视频讲解非常详细）

信号量是管理并发资源解决冲突引入的机制，有互斥和同步机制。通过信号量的擦欧总实现对某个资源的互斥访问确保资源的安全，避免了进程之间的冲突和进程，保证了资源和数据的安全性。

引入信号量机制是为了解决进程同步和互斥的问题，在多进程或者多线程的情况下同时访问某个资源可能造成不可预估的错误。因此信号量机制的引入可以解决不同线程对于同一个资源的访问安全性并保证数据的正确性。

1. 给出记录型信号量wait()和signal（）操作的实现代码，可以用任何语言表达，书上是类Pascal，可以用C，python，Java等，进程的等待队列可以用tasK\_struct结构表示

typedef struct {

int value; // 记录可用资源的数目

process\_queue\_t \*waiting\_queue; // 记录等待该资源的进程队列

} semaphore\_t;

1. 用wait()，signal（）或者sem\_wait()和sem\_post（）实现司机和售票员的同步

sem\_t 是信号量类型，是 POSIX 线程库提供的一种线程同步机制，用于实现进程内的线程同步。它包含一个计数器和一个等待队列，可以实现多个线程之间的同步与互斥访问。在 sem\_t 中，计数器表示资源的数量，等待队列表示等待使用该资源的线程队列。sem\_t 提供了两个主要的操作：sem\_wait() 和 sem\_post()，用于等待资源和释放资源。在多线程编程中，通常使用信号量来保证线程同步和资源共享的正确性。

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t driver\_sem; // 司机信号量

sem\_t ticket\_sem; // 售票员信号量

void \*driver(void \*arg) {

printf("driver arrive at the station.\n");

sem\_post(&driver\_sem); // 司机到达火车站，发信号量唤醒售票员

sem\_wait(&ticket\_sem); // 等待售票员卖完票

printf("driver start the train and leave the station.\n");

pthread\_exit(NULL);

}

void \*ticket\_seller(void \*arg) {

printf("ticket seller is selling ticket.\n");

sem\_wait(&driver\_sem); // 等待司机到达

printf("ticket seller sold the ticket and wake up the driver.\n");

sem\_post(&ticket\_sem); // 售票员卖完票，发信号量唤醒司机

pthread\_exit(NULL);

}

int main(int argc, char \*\*argv) {

pthread\_t driver\_thread, ticket\_thread;

sem\_init(&driver\_sem, 0, 0);

sem\_init(&ticket\_sem, 0, 0);

pthread\_create(&driver\_thread, NULL, driver, NULL);

pthread\_create(&ticket\_thread, NULL, ticket\_seller, NULL);

pthread\_join(driver\_thread, NULL);

pthread\_join(ticket\_thread, NULL);

sem\_destroy(&driver\_sem);

sem\_destroy(&ticket\_sem);

return 0;

}

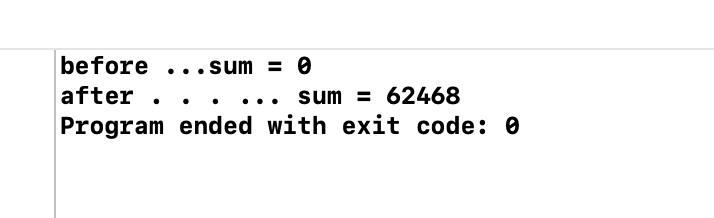
使用了两个信号量**driver\_sem**和**ticket\_sem**，分别表示司机和售票员之间的同步。在**driver()**中，首先等待售票员卖完票（即等待**ticket\_sem**），然后开车离开火车站。在**ticket\_seller()**中，首先等待司机到达火车站（即等待**driver\_sem**），然后卖票并唤醒司机（即发出**ticket\_sem**信号量）。

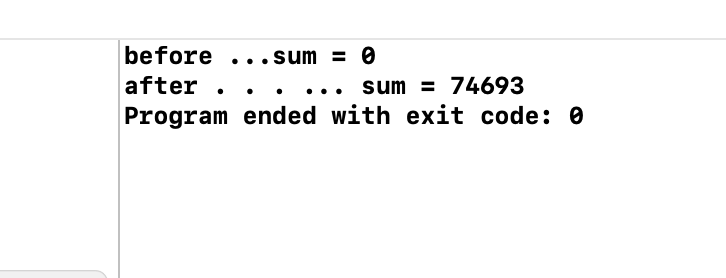
使用信号量可以确保售票员和司机之间的同步，避免了售票员在司机到达前就开始卖票或者司机在售票员卖完票前就出发的情况发生。

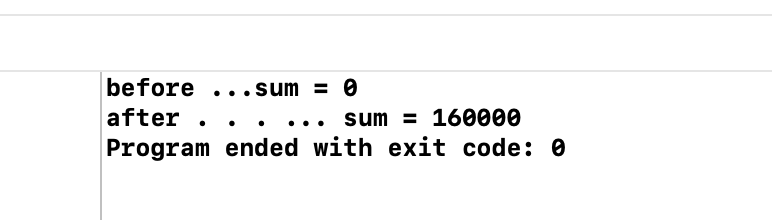
1. 调试实验楼上的实验12，编写小节1程序并运行（提供代码和运行结果截图），阅读小节2-5内容，分析小节1程序运行结果原因。

发现在thread函数里面sum+=的前后加锁和解锁，就能保证程序的资源不产生冲突而平稳运行，得到正确的答案。

sum += 1; 不是原子操作，它包含了读取sum变量的值、加1和将结果写回到sum变量中三个操作，这些操作并不是一气呵成的，而是可能会被操作系统中断，从而导致多个线程同时访问和修改sum变量的值，进而引起数据竞争和计算错误。







1. 在作业评论区提交小组讨论图。