实验 1：进程间同步/互斥问题

1. 实验目的：

1. 通过对进程间通信同步/互斥问题的编程实现，加深理解信号量和 P、V 操作的原理；
2. 对 Windows 或 Linux 涉及的几种互斥、同步机制有更进一步的了解；
3. 熟悉 Windows 或 Linux 中定义的与互斥、同步有关的函数。

2. 实验报告

1.设计思路与程序结构：

实验环境：windows 平台下，由 Python 编程语言和 C++语言完成。

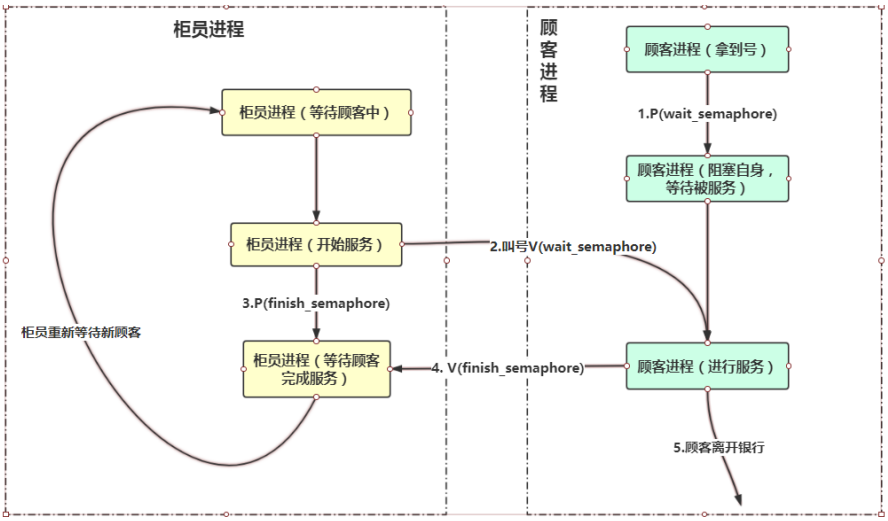
Python 程序用于搭建程序原型，C++程序调用 Windows 的多线程 API 完成整个任务。

本实验完成**银行柜员服务问题**，Python 程序的实现，主要基于 multiprocessing 进程库，利用 Lock, Semaphore, Queue 等类实现进程间通信。而 C++中调用 Windows 的多线程 API，并利用全局变量、锁对象、临界区对象完成线程间通信。

（a）Python 程序设计思路：

程序中定义两个二元信号量（互斥量，程序中为 Lock 对象），实现同时只能有一个顾客拿号，一个柜员叫号。

同时定义了一“顾客多进程队列”，每个拿到号的顾客，立刻加入顾客队列。加入队列后，顾客设置两个**初值为 0** 的信号量（wait_semaphore, finish_semaphore），并尝试对 wait_semaphore 信号量进行 P 操作，以实现自身的阻塞。，等待被叫到号时（离开队列），柜员对同一信号量进行 V 操作，唤醒阻塞的顾客。然后柜员对 finish_semaphore 信号量进行 P 操作，直到顾客完成相应的服务后，顾客对同一信号量进行 V 操作，使得柜员被唤醒。**这样利用一个进程先阻塞自身，只能在另一个进程中去除阻塞，就实现了同时进行服务的柜员和顾客（两个进程）的同步**。图示如下：

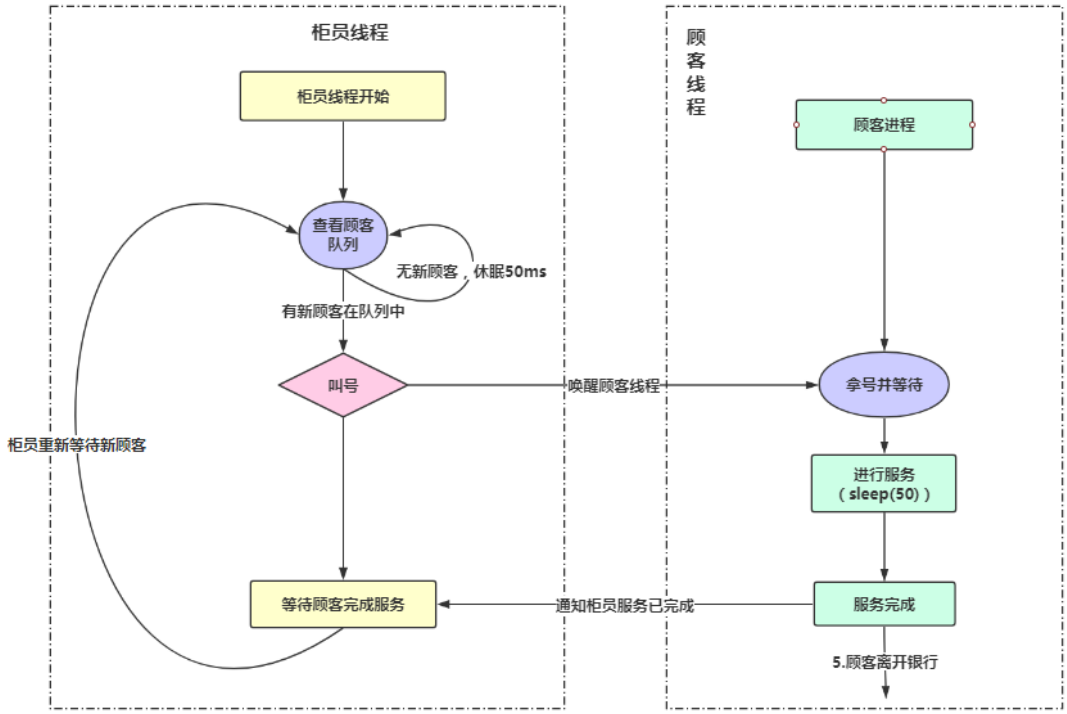


由上述流程知，柜员和顾客的同步，基于顾客的服务只进行一个步骤（即柜员先进行一个步骤，在柜员完成前，顾客等待（阻塞）。柜员完成后，顾客再进行一个步骤，在顾客完成前，柜员阻塞）。若柜员和顾客需要在多个步骤中相互完成步骤且另一方等待，才能完成一次服务，可以利用上述相同方法和流程，设置循环，来完成每个步骤和相互等待的过程。

程序中，柜员和顾客分别为 Counter 和 Client 函数。主进程中先启动指定数量的进程运行 Counter 函数。之后在根据每个顾客指定的进入时间、服务时长启动进程运行 Client 函数。具体代码请见 **bank.py**，测试文件为 **bank_test.txt**

(b)C++程序设计思路：

流程图如下：



C++程序中同样利用 CreateMutex 函数，定义两个互斥量，实现同时只能有一个顾客拿号，一个柜员叫号。

与 Python 程序不同，C++程序中利用全局队列变量保存进入银行的顾客信息。同时，使用 InitializeCriticalSectionAndSpinCount 函数创建临界区，每次柜员线程需要修改顾客队列（叫号）时，首先需要进入临界区，从而防止多个柜员线程同时修改顾客队列。新的顾客进入队列时，顾客线程也需要首先进入临界区。

此外，C++程序使用 **事件对象**（CreateEvent 函数） 来对柜员和顾客线程进程同步。顾客进入等待队列后，等待被叫号事件的发生（发生阻塞），代码如下：

```
//顾客阻塞，直到等待被叫到号
WaitForSingleObject(wait_event, INFINITE);
```

顾客队列中，每个元素包含该顾客的信息 struct 类，等待被叫号的事件句柄，与通知柜

员线程服务已完成的事件句柄。顾客队列元素 struct 类定义如下：

```
struct client_queue_elem
{
    client_info cinfo;                  //用于该柜员和顾客交流的管道
    HANDLE wait_event;                  //用于触发顾客被叫到号事件
    HANDLE finish_event;                //用于触发顾客结束服务事件
};
```

而每当有一个柜员线程空闲时，该柜员线程会尝试查看顾客队列中是否有新的顾客。如果没有，则这个柜员线程挂起 50ms 后再次尝试 ,如果有，则进行叫号 SetEvent(wait_event)。通过顾客的被叫号事件句柄，通知该顾客线程解除阻塞。然后柜员等待顾客完成服务的事件发生 (WaitForSingleObject(finish_event, INFINITE);)

顾客线程解除阻塞后，利用 Sleep(sever_time)来模拟进行一段时间的服务。然后 SetEvent(finish_event);通知柜员服务已完成。该顾客线程结束，对应柜员线程重新进入空闲状态。**因此利用事件对象来实现顾客和柜员之间相互等待，从而实现了同时进行服务的柜员和顾客线程进程的同步。**

2.程序运行情况：

(a) Python 程序：

已知测试的输入为：

1 1 10
2 5 2
3 6 3

利用 time.clock()函数获取进程的实际运行时间。

当柜员数为 5 时（顾客不需要等待其他顾客完成），输出为：

client 2: 5.0 5.006328 7.006685333333333 5
client 3: 6.0 6.006887555555555 9.006991555555555 4
client 1: 1.0 1.0065004444444445 11.006634222222223 1

输出分别为：顾客号，进入时间，开始时间，离开时间，柜员号。

由于顾客不需要等待其他顾客完成，因此可以看出，进入时间与开始时间相同（排除程序中部分语句的运行时间后）。离开时间为服务所需时间+进入时间。

当柜员数 1 时（每个顾客都需要之前的顾客完成服务），输出为：

client 1: 1.0 1.008164 11.008337333333333 1
client 2: 5.000000444444445 10.976570666666667 12.977187111111112 1
client 3: 6.0 12.986993333333334 15.987306222222223 1

可以看出，每个顾客都需要之前的顾客完成服务才能开始服务。

(b) C++程序：

已知测试的输入为：

1 1 10
2 5 2

3 6 3

当柜员数为 5 时（顾客不需要等待其他顾客完成），输出为：

顾客 2: 5 5 7 0

顾客 3: 6 6 9 2

顾客 1: 1 1 11 1

输出分别为：顾客号，进入时间，开始时间，离开时间，柜员号。

由于顾客不需要等待其他顾客完成，因此可以看出，进入时间与开始时间相同（排除程序中部分语句的运行时间后）。离开时间为服务所需时间+进入时间。

当柜员数 1 时（每个顾客都需要之前的顾客完成服务），输出为：

顾客 1: 1 1 11 0

顾客 2: 5 11 13 0

顾客 3: 6 13 16 0

可以看出，每个顾客都需要之前的顾客完成服务才能开始服务。

3.思考题解答

1. 柜员人数和顾客人数对结果分别有什么影响？

答：

定性分析：

当柜员数目足够多时，每个顾客不需要等待。在顾客进入银行后，只需要经过所需的服务时间即可离开银行。

当某个时刻有新顾客进入银行，但无空闲柜员时，则该顾客需要柜员空闲且等待之前所有等待队列中的顾客完成服务。

过程模拟：

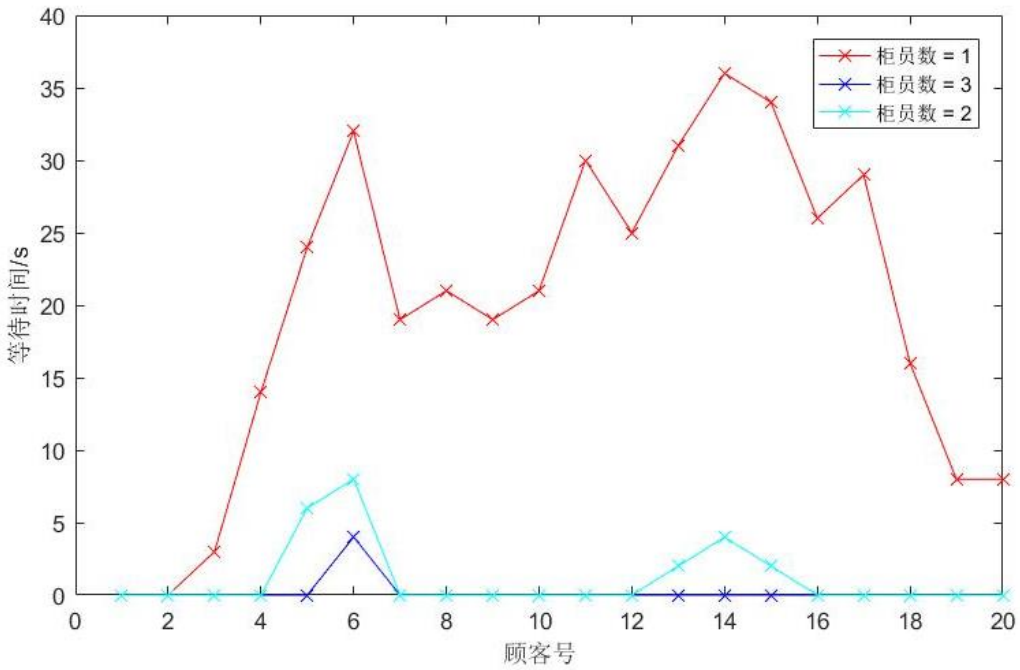
使用 matlab 生成模拟 **poisson 过程的**顾客流，每个顾客停留时间服从指数分布。其中 poisson 过程$\lambda = 5$，指数分布$\mu = 5$。所有时间已进行整数化处理。固定顾客总数为 20 个，研究顾客的平均等待时间与柜员数的关系。

生成代码如下（具体请见 **Poisson.m** 文件）：

```
NUM_CLIENT = 10;
% 利用指数分布累加模拟泊松分布
r = exprnd(3, [1, NUM_CLIENT]);
r = ceil(r);
enter_time = cumsum(r);
```

```
% 服务时间
serve_time = ceil(exprnd(5, [1, NUM_CLIENT]));
% 写入文件
f = fopen('./client_stream.txt', 'w');
for i = 1:numel(enter_time)
    fprintf(f, '%d %d %d\n', i, enter_time(i), serve_time(i));
end
fclose(f);
```

结果如下：



可以看到，当增加柜员数量时，可以显著地减少顾客的平均等待时间。

2. 实现互斥的方法有哪些?各自有什么特点?效率如何?

答：

（1）互斥量（mutex）:只有两种状态，解锁与加锁。当一互斥量已经加锁时，其

他想加锁的进程将被阻塞。

（2）信号量：代表资源的剩余数量，可取任意整数。当信号量小于等于零时，想进行 down 操作的进程将被阻塞。

（3）基于 TSL 或 XCHG 的锁变量：利用特定机器指令，实现对共享变量的原子操作。

（4）严格轮换法：通过循环给每个进程轮流运行机会。

效率：基于 TSL 或 XCHG 的锁变量 > 互斥量 > 信号量 > 严格轮换法

4.遇到的问题与体会

（a）

当进行多线程/多进程编程时，十分重要一点的就是线程/进程间的互斥与同步。

刚开始写好程序时，由于未注意到柜员进程和顾客进程间的同步，使得顾客进程还未结束，柜员进程就重新进入了等待新顾客的状态，导致结果错误。因此，为了使多线程/进程间协调一致，行为可控，必须设计精心的互斥与同步。

(b)

观察顾客流模拟的结果可以发现：当增加柜员数量时，可以显著地减少顾客的平均等待时间。因此，在设计类似的“等待过程”的实际应用中，可以适当增加处理方的数量来减少任务完成的等待时间。例如在网络的路由器中，通过多个处理器来处理收到的分组，来减少每个分组的排队时延，同时减少等待队列的长度，从而减少了缓存队列中分组数量过多，新的分组被丢弃的现象。