操作系统实验一实验报告
银行柜员服务问题

张子扬 无 06 2020010790

一、 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。

二、 实验平台

本实验在 Python3.10.0 上运行。

三、 实验原理

实验采用信号量（semaphore）的 P、V 操作，实现不同线程间的同步和资源访问的互斥。其中，用于实现互斥的二元信号量采用互斥量（mutex）。

Python 提供的线程库 threading 中包含了对信号量和互斥量的封装，实验中直接使用它们进行同步与互斥操作。

四、 算法设计思路

实验采用同步算法的顾客（customer）线程和柜员（teller）线程的伪代码如下：

```
queue waiting_customers;
mutex waiting_customers_mtx;
semaphore waiting_customers_sem = 0;

class customer
{
    idx;
    semaphore sem = 0;
};

def customer_action(customer)
{
    enter_bank();
    P(waiting_customers_mtx);
    waiting_customers.put(customer);
```

```
    V(waiting_customers_sem);
    V(waiting_customers_mtx);

    P(customer.sem);
    leave_bank();
}

def teller_action(teller)
{
    while True {
        P(waiting_customers_sem);
        P(waiting_customers_mtx);
        cust = waiting_customers.get();
        V(waiting_customers_mtx);
        serve(cust);
        V(cust.sem);
    }
}
```

五、 算法实现

```
class Customer:
    def __init__(self, id, arrival_time, service_time):
        self.id = id
        self.arrival_time = arrival_time
        self.service_time = service_time
        self.start_service_time = 0
        self.leave_time = 0
        self.served_by = None
        self.sem = threading.Semaphore(0)

def customer_action(customer, waiting_customers, waiting_customers_mtx,
waiting_customers_sem):
    time.sleep(customer.arrival_time) # 进入银行
    waiting_customers_mtx.acquire() # P 操作
    waiting_customers.put(customer)
    waiting_customers_mtx.release() # V 操作
    waiting_customers_sem.release()
    customer.sem.acquire() # 等待服务完成
    # 离开银行，不需要操作

def teller_action(teller_id, init_time, waiting_customers,
waiting_customers_mtx, waiting_customers_sem):
    while True:
        waiting_customers_sem.acquire() # 等待顾客到达
```

```
        waiting_customers_mtx.acquire()
        customer = waiting_customers.get()
        waiting_customers_mtx.release()
        customer.start_service_time = round(time.time() - init_time)
        customer.served_by = teller_id
        time.sleep(customer.service_time) # 服务顾客
        customer.leave_time = customer.start_service_time +
customer.service_time
        customer.sem.release() # 完成服务
        print("顾客{}进入银行时间为{}，开始服务时间为{}，离开银行时间为{}，服务
柜员号为{}".format(customer.id, customer.arrival_time,
customer.start_service_time, customer.leave_time, customer.served_by))
```

六、 运行测例

运行的测例默认采用大作业的要求测试数据：

1 1 10

2 5 2

3 6 3

设置银行柜员有 2 个，输出结果为：

```
顾客2进入银行时间为5，开始服务时间为5，离开银行时间为7，服务柜员号为2
顾客3进入银行时间为6，开始服务时间为7，离开银行时间为10，服务柜员号为2
顾客1进入银行时间为1，开始服务时间为1，离开银行时间为11，服务柜员号为1
```

设置银行柜员有 3 个，输出结果为：

```
顾客2进入银行时间为5，开始服务时间为5，离开银行时间为7，服务柜员号为2
顾客3进入银行时间为6，开始服务时间为6，离开银行时间为9，服务柜员号为3
顾客1进入银行时间为1，开始服务时间为1，离开银行时间为11，服务柜员号为1
```

七、 思考题

1. 柜员人数和顾客人数对结果分别有什么影响？

可以首先想象一个理想情况，每分钟有 n 个顾客到达银行，每个顾客需要被服务 m 分钟，那么此时需要 n*m 个银行柜员则恰好不会有顾客等待，且恰好每个柜员都始终在工作。在这个理想情况的基础上，我们设想以下两种情况：

如果顾客平均到达速率 n 和平均需要服务时间 m 不变，柜员人数增加，那么顾客的平均等待时间 t 会减少，因为有更多的柜员可以为顾客服务。但是，如果柜员人数过多，顾客的平均等待时间 t 降至 0 之后就不会再下降，此时会有某些柜员空闲，从而浪费资源。

如果顾客平均到达速率 n 和平均需要服务时间 m 增多，而柜员人数不变，那么顾客的平均等待时间 t 会增加，因为有更多的顾客在等待服务。随着顾客平均到达速率 n 和平均需要服务时间 m 增多，顾客的平均等待时间 t 会一直增长，与 n*m 近似成线性关系。

因此，在设计银行柜员服务系统时，需要根据实际情况合理安排柜员人数和顾客人数，以保证银行的运营效率和顾客的满意度。

2. 实现互斥的方法有哪些?各自有什么特点?效率如何?

（1） 禁用中断：适用于单核处理器，在进入临界区之前关闭硬件中断，防止调度。禁用中断效率高，但不安全且不适用于多核处理器。

（2） 忙等待：一些算法例如锁变量、严格轮换法、Petersen 算法、硬件指令法，都使用了忙等待的算法。通过不断检查变量的值来实现同步，避免了用户态和内核态切换的开销。然而，忙等待占用 CPU 资源且可能存在优先级反转问题。

（3） 信号量：使用 P、V 原语实现同步和互斥，阻塞时不占用 CPU。效率较高，但部分实现会引起用户态和内核态转换的开销。

（4） 互斥量：信号量的简化形式，只有二元信号量和同步的 P、V 操作。相对于信号量，互斥量使用简单且性能更好。

（5） 管程：编程语言提供的特性，通过互斥保证一个线程在管程函数内执行。便利但有些语言不支持和部分实现可能无法在中途阻塞。