Process Synchronization

1. Multiple choice

   a. A mutex lock ____.

   A) is exactly like a counting semaphore

   B) is essentially a boolean variable

   C) is not guaranteed to be atomic

   D) can be used to eliminate busy waiting

   Ans: B

   b. When using semaphores, a process invokes the wait() operation before accessing its critical section, followed by the signal() operation upon completion of its critical section. Consider reversing the order of these two operations—first calling signal(), then calling wait(). What would be a possible outcome of this?

   A) Starvation is possible.

   B) Mutual exclusion is still assured.

   C) Several processes could be active in their critical sections at the same time.

   D) Deadlock is possible.

   Ans: C

2. Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: deposit(amount) and withdraw(amount). These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function, and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

Answer:

Assume that the balance in the account is $250.00 and that the husband calls withdraw($50) and the wife calls deposit($100). Obviously, the correct value should be $300.00. Since these two transactions will be serialized, the local value of the balance for the husband becomes $200.00, but before he can commit the transaction, the deposit(100) operation takes place and updates the shared value of the balance to $300.00. We then switch back to the husband, and the value of the shared balance is set to $200.00—obviously an incorrect value.

3. Servers can be designed to limit the number of open connections. For example, a server may wish to have only *N* socket connections at any point in time. As soon as *N* connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

Answer:

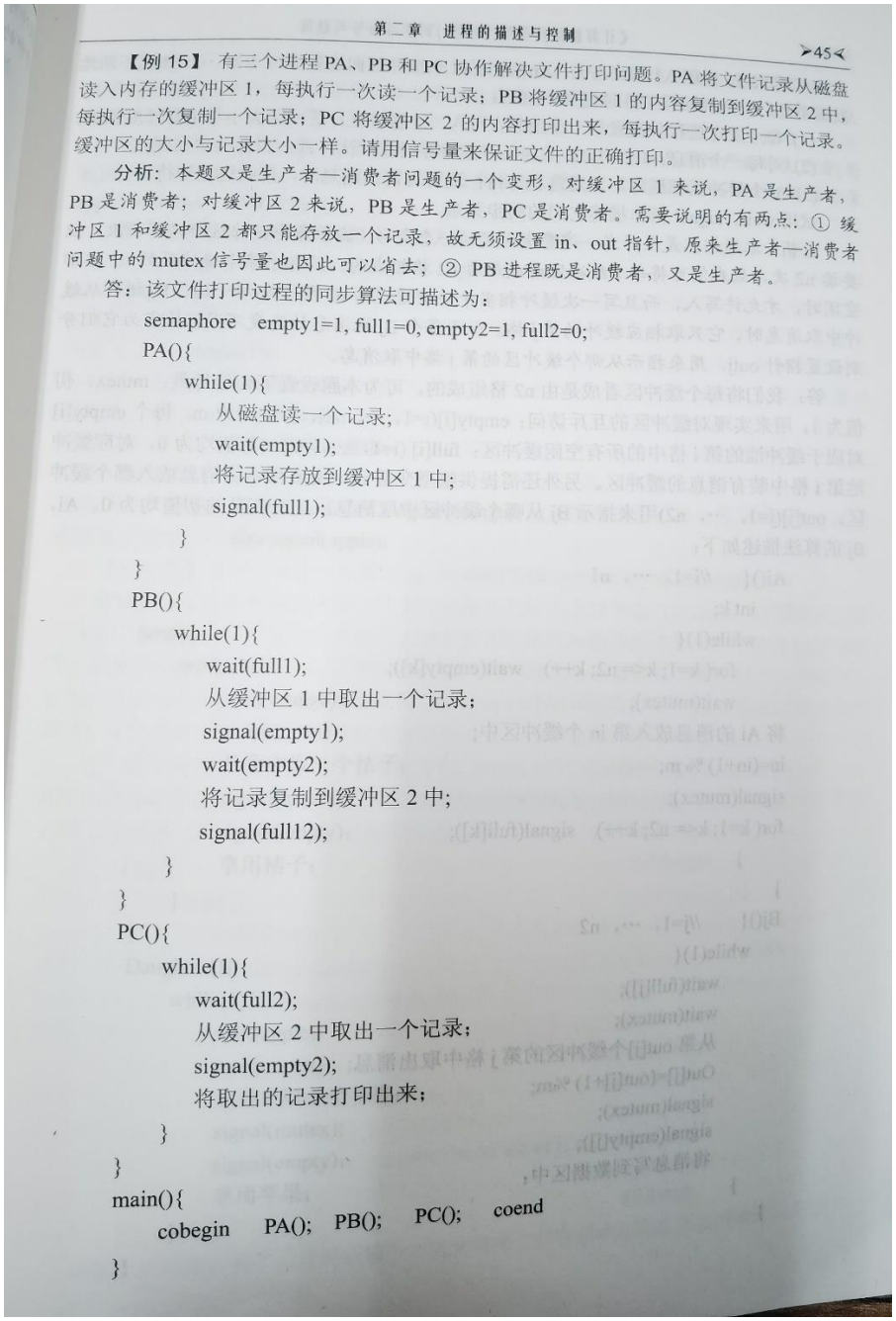
A semaphore is initialized to the number of allowable open socket connections. When a connection is accepted, the acquire() method is called; when a connection is released, the release() method is called. If the system reaches the number of allowable socket

connections, subsequent calls to acquire() will block until an existing connection is terminated and the release method is invoked.

4. Explain the difference between the first readers–writers problem and the second readers–-writers problem.

Ans: The first readers–writers problem requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database; whereas the second readers–writers problem requires that once a writer is ready, that writer performs its write as soon as possible.

5. Three processes P1, P2, P3 cooperate to print files. P1 reads one record from disk to buffer1, P2 copy the record from buffer1 to buffer2, and P3 print the record in buffer2. If the sizes of buffer and record are same, please write the pseudocode to print files correctly using semaphores.



第二章　进程的描述与控制　　45

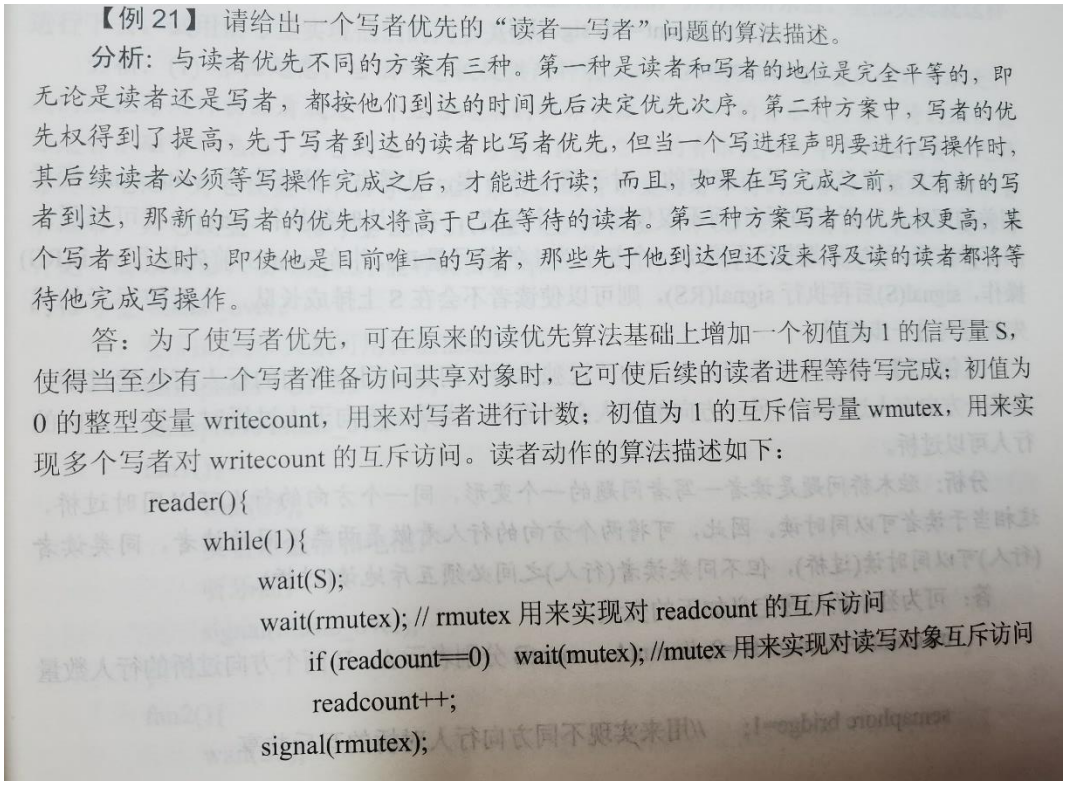
【例 15】 有三个进程 PA、PB 和 PC 协作解决文件打印问题。PA 将文件记录从磁盘读入内存的缓冲区 1，每执行一次读一个记录；PB 将缓冲区 1 的内容复制到缓冲区 2 中，每执行一次复制一个记录；PC 将缓冲区 2 的内容打印出来，每执行一次打印一个记录。缓冲区的大小与记录大小一样。请用信号量来保证文件的正确打印。

分析：本题又是生产者—消费者问题的一个变形，对缓冲区 1 来说，PA 是生产者，PB 是消费者；对缓冲区 2 来说，PB 是生产者，PC 是消费者。需要说明的有两点：① 缓冲区 1 和缓冲区 2 都只能存放一个记录，故无须设置 in、out 指针，原来生产者—消费者问题中的 mutex 信号量也因此可以省去；② PB 进程既是消费者，又是生产者。

答：该文件打印过程的同步算法可描述为：

```
semaphore  empty1=1, full1=0, empty2=1, full2=0;
PA(){
    while(1){
        从磁盘读一个记录;
        wait(empty1);
        将记录存放到缓冲区 1 中;
        signal(full1);
    }
}
PB(){
    while(1){
        wait(full1);
        从缓冲区 1 中取出一个记录;
        signal(empty1);
        wait(empty2);
        将记录复制到缓冲区 2 中;
        signal(full12);
    }
}
PC(){
    while(1){
        wait(full2);
        从缓冲区 2 中取出一个记录;
        signal(empty2);
        将取出的记录打印出来;
    }
}
main(){
    cobegin   PA();  PB();  PC();  coend
}
```

6. Please write pseudocode for solving Reader-Writer problem with writer-preference.



【例 21】 请给出一个写者优先的“读者—写者”问题的算法描述。

分析：与读者优先不同的方案有三种。第一种是读者和写者的地位是完全平等的，即无论是读者还是写者，都按他们到达的时间先后决定优先次序。第二种方案中，写者的优先权得到了提高，先于写者到达的读者比写者优先，但当一个写进程声明要进行写操作时，其后续读者必须等写操作完成之后，才能进行读；而且，如果在写完成之前，又有新的写者到达，那新的写者的优先权将高于已在等待的读者。第三种方案写者的优先权更高，某个写者到达时，即使他是目前唯一的写者，那些先于他到达但还没来得及读的读者都将等待他完成写操作。

答：为了使写者优先，可在原来的读优先算法基础上增加一个初值为 1 的信号量 S，使得当至少有一个写者准备访问共享对象时，它可使后续的读者进程等待写完成；初值为 0 的整型变量 writecount，用来对写者进行计数；初值为 1 的互斥信号量 wmutex，用来实现多个写者对 writecount 的互斥访问。读者动作的算法描述如下：

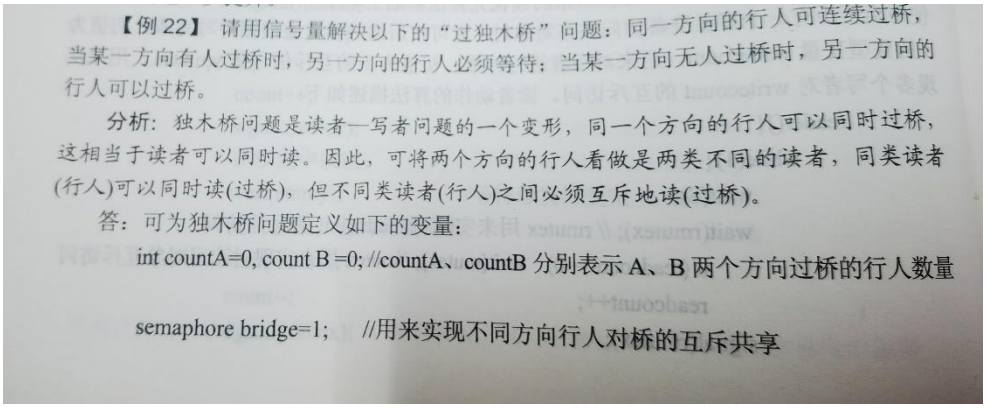
```
reader(){
    while(1){
        wait(S);
        wait(rmutex); // rmutex 用来实现对 readcount 的互斥访问
            if (readcount==0)  wait(mutex); //mutex 用来实现对读写对象互斥访问
            readcount++;
        signal(rmutex);
        signal(S);
        perform read operation;
        wait(rmutex);
            readcount--;
            if (readcount==0) signal(mutex);
        signal(rmutex);
    }
}
```

写者动作的算法描述如下：

```
writer(){
    while(1){
        wait(wmutex);   // wmutex 用来实现对 writecount 的互斥访问
            if( writecount==0) wait(S);
            writecount++;
        signal(wmutex);
        wait(mutex);
        perform write operation;
        signal(mutex);
        wait(wmutex);
            writecount--;
            if( writecount==0) signal(S);
        signal(wmutex);
    }
}
```

上述算法是按第二种方案写的，对于第一种方案，只需去掉算法中跟 writecount 变量相关的部分，让所有的写者(而不仅仅是第一个写者)，在到达时都执行 wait(S)就可以了。第三种方案，可对读者进程再增加一个初值为 1 的信号量 RS，并在 wait(S)前先执行 wait(RS)操作，signal(S)后再执行 signal(RS)，则可以使读者不会在 S 上排成长队，从而使写者的优先权得到进一步提升。

7. Consider a shared single-lane bridge, allow people go through the bridge in the same direction continuously, and the people in the other direction have to wait. While nobody goes through the bridge, the people in the other direction can be allowed to cross the bridge. Please write pseudocode to solve the problem correctly using semaphores.



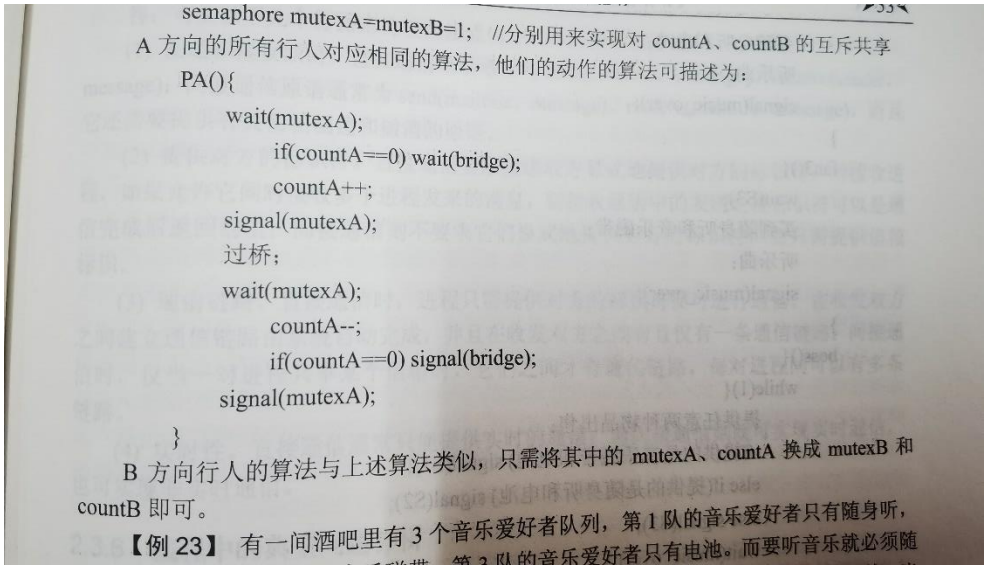
【例 22】 请用信号量解决以下的“过独木桥”问题：同一方向的行人可连续过桥，当某一方向有人过桥时，另一方向的行人必须等待；当某一方向无人过桥时，另一方向的行人可以过桥。

分析：独木桥问题是读者—写者问题的一个变形，同一个方向的行人可以同时过桥，这相当于读者可以同时读。因此，可将两个方向的行人看做是两类不同的读者，同类读者(行人)可以同时读(过桥)，但不同类读者(行人)之间必须互斥地读(过桥)。

答：可为独木桥问题定义如下的变量：

```
int countA=0, count B =0; //countA、countB 分别表示 A、B 两个方向过桥的行人数量

semaphore bridge=1;    //用来实现不同方向行人对桥的互斥共享
```



```
semaphore mutexA=mutexB=1;  //分别用来实现对 countA、countB 的互斥共享
```

A 方向的所有行人对应相同的算法，他们的动作的算法可描述为：

```
PA(){
    wait(mutexA);
        if(countA==0) wait(bridge);
        countA++;
    signal(mutexA);
    过桥;
    wait(mutexA);
        countA--;
        if(countA==0) signal(bridge);
    signal(mutexA);
}
```

B 方向行人的算法与上述算法类似，只需将其中的 mutexA、countA 换成 mutexB 和 countB 即可。

【例 23】 有一间酒吧里有 3 个音乐爱好者队列，第 1 队的音乐爱好者只有随身听，

Deadlocks

1. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.

Answer:

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, new resources are bought and added to the system. If deadlock is

controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

a. Increase **Available** (new resources added)

b. Decrease **Available** (resource permanently removed from system)

c. Increase **Max** for one process (the process needs or wants more resources than allowed).

d. Decrease **Max** for one process (the process decides it does not need that many resources)

e. Increase the number of processes

f. Decrease the number of processes

Answer:

a. Increase **Available** (new resources added)—This could safely be changed without any problems.

b. Decrease **Available** (resource permanently removed from system)—This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

c. Increase **Max** for one process (the process needs more resources than allowed, it may want more)—This could have an effect on the system and introduce the possibility of deadlock.

d. Decrease **Max** for one process (the process decides it does not need that many resources)—This could safely be changed without any problems.

e. Increase the number of processes—This could be allowed assuming that resources were allocated to the new process(es) such that the system does not enter an unsafe state.

f. Decrease the number of processes—This could safely be changed without any problems.

2. Consider the following snapshot of a system:

	Allocation				*Max*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need*?

b. Is the system in a safe state?

c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:

a. The values of *Need* for processes P_0 through P_4, respectively, are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).
b. The system is in a safe state. With *Available* equal to (1, 5, 2, 0), either process P_0 or P_3 could run. Once process P_3 runs, it releases its resources, which allows all other existing processes to run.
c. The request can be granted immediately. The value of *Available* is then (1, 1, 0, 0). One ordering of processes that can finish is P_0, P_2, P_3, P_1, and P_4.

3. Consider the following snapshot of a system:

	Allocation				*Max*			
	A	B	C	D	A	B	C	D
P_0	3	0	1	4	5	1	1	7
P_1	2	2	1	0	3	2	1	1
P_2	3	1	2	1	3	3	2	1
P_3	0	5	1	0	4	6	1	2
P_4	4	2	1	2	6	3	2	5

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.
a. *Available* = (0, 3, 0, 1)
b. *Available* = (1, 0, 0, 2)

Answer:

a. Not safe. Processes P_2, P_1, and P_3 are able to finish, but no remaining processes can finish.
b. Safe. Processes P_1, P_2, and P_3 are able to finish. Following this, processes P_0 and P_4 are also able to finish.

4. Consider the following snapshot of a system:

	Allocation				*Max*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	0	1	4	2	1	2	3	3	2	1
P_1	3	1	2	1	5	2	5	2				

P_2	2	1	0	3	2	3	1	6				
P_3	1	3	1	2	1	4	2	4				
P_4	1	4	3	2	3	6	6	5				

Answer the following questions using the banker's algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.

b. If a request from process P_1 arrives for (1, 1, 0, 0), can the request be granted immediately?

c. If a request from process P_4 arrives for (0, 0, 2, 0), can the request be granted immediately?

Main Memory

1. Explain the difference between internal and external fragmentation.

Answer:

a. Internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

b. External fragmentation is unused space between allocated regions of memory. Typically external fragmentation results in memory regions that are too small to satisfy a memory request, but if we were to combine all the regions of external fragmentation, we would have enough memory to satisfy a memory request.

2. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Answer:

a. First-fit:

b. 212K is put in 500K partition

c. 417K is put in 600K partition

d. 112K is put in 288K partition (new partition 288K = 500K − 212K)

e. 426K must wait

f. Best-fit:

g. 212K is put in 300K partition

h. 417K is put in 500K partition

i. 112K is put in 200K partition

j. 426K is put in 600K partition

k. Worst-fit:

l. 212K is put in 600K partition

m. 417K is put in 500K partition

n. 112K is put in 388K partition

o. 426K must wait

In this example, best-fit turns out to be the best.

3. Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):
   a. 2375
   b. 19366
   c. 30000
   d. 256
   e. 16385

Answer:

a. page = 1; offset = 391

b. page = 18; offset = 934

c. page = 29; offset = 304

d. page = 0; offset = 256

e. page = 1; offset = 1

4. Consider a paging system with the page table stored in memory.
   a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
   b. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes zero time, if the entry is there.)

Answer:

a. 400 nanoseconds: 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.

5. Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1352	96

What are the physical addresses for the following logical addresses?

a. 0,430

b. 1,10

c. 2,500

d. 3,400

e. 4,112

Answer:

a. 219 + 430 = 649

b. 2300 + 10 = 2310

c. illegal reference, trap to operating system

d. 1327 + 400 = 1727

e. illegal reference, trap to operating system