Problem 1: Concurrency (20 points)

1. main_A: (4')
   i4get here
   c4get here
   main_B: (4')
   i4get here
   c4get here
   or
   i5get here
   c4get here
   or
   i4get here
   c3get here .....

2. (4')

```
#include "caspp.h"
Sem_t mutex;
int n = 5;
int main_B()
{
    Sem_init(&mutex, 0, 1);
    ...
}
```

In do_it1():	In do_it2():

P(&mutex);
int a = n;
n--;
printf("%c%d",c, n);
int b = n;
V(&mutex);
	P(&mutex);
	n++;
	V(&mutex);

3. (4')

```
#include "caspp.h"
Sem_t mutex1, mutex2;
int n = 5;
int main_B()
{
    Sem_init(&mutex1, 0, 1);
    Sem_init(&mutex2, 0, 0);
    ...
}
```

In do_it1():	In do_it2():
P(&mutex1); n--; V(&mutex2);	P(&mutex2); n++; V(&mutex1);

4. (4') When thread1 hold a but require b, thread3 hold b but require a.

Problem 2: Address Translation (28 points)

1 1) (2') 28

2) (2') 2^(28-10) = 2^18

2 (14' + 10')

1) [1] F [2] 7 [3] 1 [4] Y

[5] N [6] 0D [7] 36C1

2) [1] 1 [2] B [3] 9D [4] Y

[5] 96

Problem 3: Virtual Memory (30points)

1 1) (2') 2^20 + 2^10

2) (2') MAP_SHARED | MAP_ANON

2 (2' * 8 = 16')

[1] 0xb77e0000 – 0xb77fffff [2] 0xb7800000 – 0xb781ffff

[3] 0x2dd [4] 0x3e0 [5] 0x2de [6] 0x000

[7] 0Xef400b74 [8] 0xef400b78

3 1) (4') 33

Array pointed by p1 occupies 32 pages. Access to the first byte of each page should raise a page fault exception. Besides, index of L1 PTE of p2 is 0x2de, which has not been filled at label A. Therefore the page for L2 page table whose index in L1 page table is 0x2de should also be paged into physical memory and updated. Total number of page fault exceptions is 32 + 1 = 33.

(解释的时候要提到 p1 指向的数组一共 32 个页，以及对 p2 调用 mmap 的初始化修改页表的过程中会导致存放二级页表的页发生缺页中断)

2) (4') 32

Because of copy-on-write mechanism, there's no need to allocate new pages for read operations to p1. Write operation to shared array object pointed by p2 will cause page fault exception. Because p2 is pointed to a shared array object, 32 pages need to be paged into physical memory. Total number of page fault exceptions is 32.

(解释的时候得分点在于提到对私有对象 p1 的读操作由于 copy-on-write 机制而不会导致缺页，并且 p2 是共享的，一次缺页处理之后两个进程中对应的页都缓存在内存中了)

3) (2') The code section of the program, which starts from 0x08048000

Problem 4: Networking (22points)

1.

[main] (6′)

```
...
listenfd =  Open_listenfd(8080);
Sem_init(&mutex, 0, 1);
while (1) {
    argp=Malloc(sizeof(int));
    *argp=Accept(listenfd, &clientaddr, &clientlen);
    pthread_create(&tid, NULL, process_request, argp);
}
...
```

[Process_request] (8′)

```
...
clientfd = *(int *)argp;
Rio_readinitb(&rio, clientfd);
Rio_readlineb(&rio, buf, 100);
sscanf(buf + 5, "%d", &objectid);
if (objectid >= 0 && objectid <= 2) {
    P(&mutex);
    votes[objectid]++;
    sprintf(buf, "OK %d %d %d\n", votes[0], votes[1], votes[2]);
    V(&mutex);
} else {
    sprintf(buf, "ERR\n");
}
Rio_writen(clientfd, buf, strlen(buf));
close(clientfd);
...
```

2. (4′) 0 1 0

3. (4′)

Remember clients' address for each object and check every time if a client votes for the same object again.