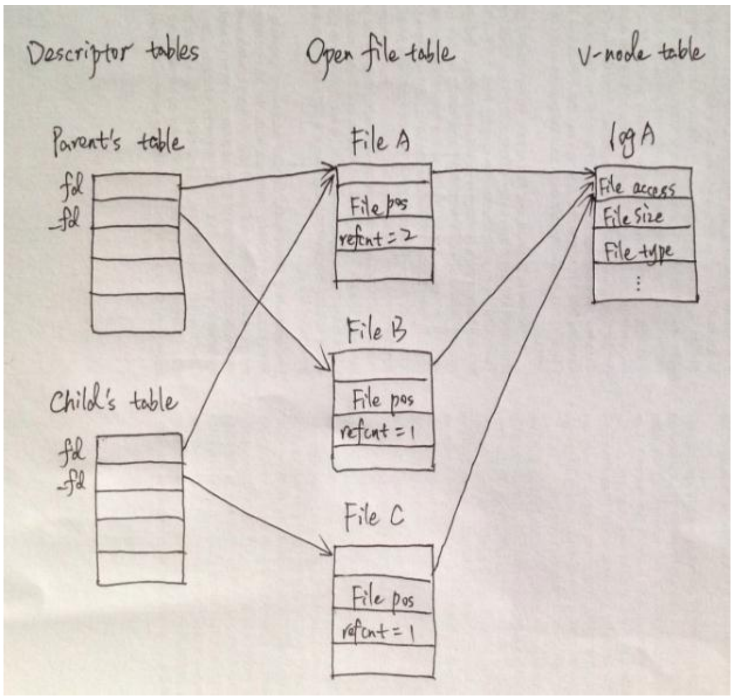
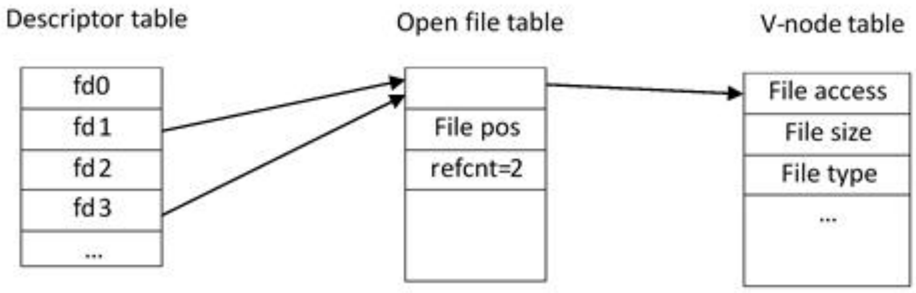
## Problem 1: Process/Thread and I/O (26 points)

1. (6') [1] No [2] No [3]No [4] No [5] Yes [6] Yes

2. (6') Prog.1



Prog.2



3. (4')

```
Child: c2 = 1
Child: c = 1
Parent: c2 = 1
Parent: c = o
```

4. (4') [1] Yes [2] Not sure [3] No [4] No

5. (6') Three possible context

```
Thread_1\nThread_2\nEnding...\n
Thread_2\nThread_1\nEnding...\n
Thread_2\nEnding...\n
```

**Problem 2: Signal & Concurrency (20 points)**

1. (5’) 0\n1\n2\nhandler\n.
   Because the SIGINT is blocked and OS can't queue the signal.
   The second and third signal will be discarded.

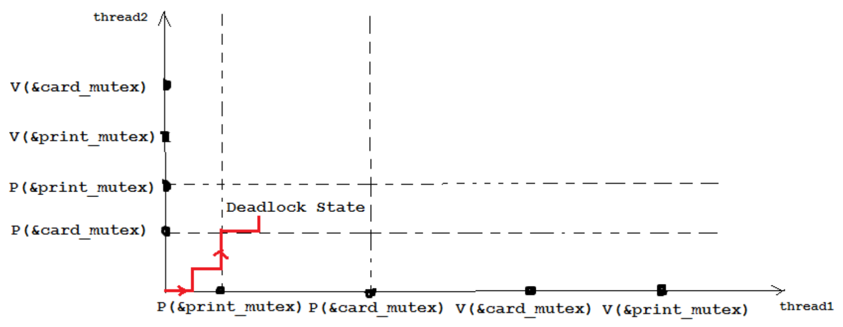
2. (5’)

   [1] In the normal case, interrupt from keyboard occurs before P(&mutex) or after V(&mutex). The "doSomething()" is executed, require the mutex and release the mutex after time-comsuming work. So the interrupt also can be executed normally.

   [2]When interrupt from keyboard occurs during the Time-consuming work in doSomething() of the main function, then the function can't release mutex. So the interrupt handler can't get the mutex resource and be blocked all the time. As a result, deadlock occurs.

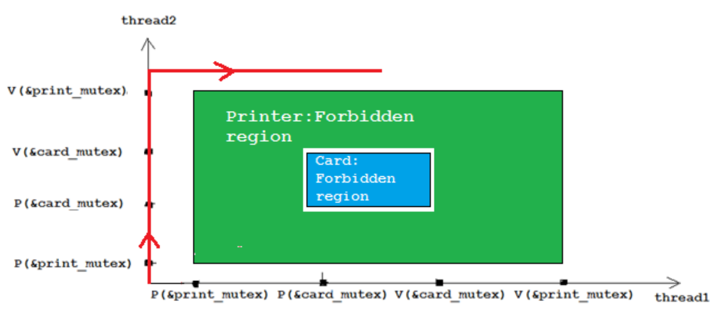
3. (10’)

   1) Assume that Printer and Card Reader have been assigned to thread1, thread2. When thread1 and thread2 hold their resource and request Card Reader and Printer, then deadlock appeared.



   2) Thread1 and thread2 holds both card_mutex and print_mutex simultaneously and locks them in the same order.

```
void *thread2(void *var){
  P(&print_mutex)
  P(&card_mutex);
  if(checkCardReader()){
    printCard();
  }
  V(&card_mutex);
  V(&print_mutex);
}
```



**Problem 3: Address Translation (22 points)**

1. (4’) [1] 5 [2] 2 [3] 1024 [4] 7

2. (18’)

|  |  |  |  |  |
|---|---|---|---|---|
| [1] 0x00F | [2] 0x3 | [3] 0x03 | [4] N | [5] N |
| [6] 0x39 | [7] 0x725 | [8] N | [9] -- |  |
| [1] 0x005 | [2] 0x1 | [3] 0x01 | [4] Y | [5] N |
| [6] 0x00 | [7] 0x01F | [8] Y | [9] 0x08 |  |

**Problem 4: Virtual Memory (32points)**

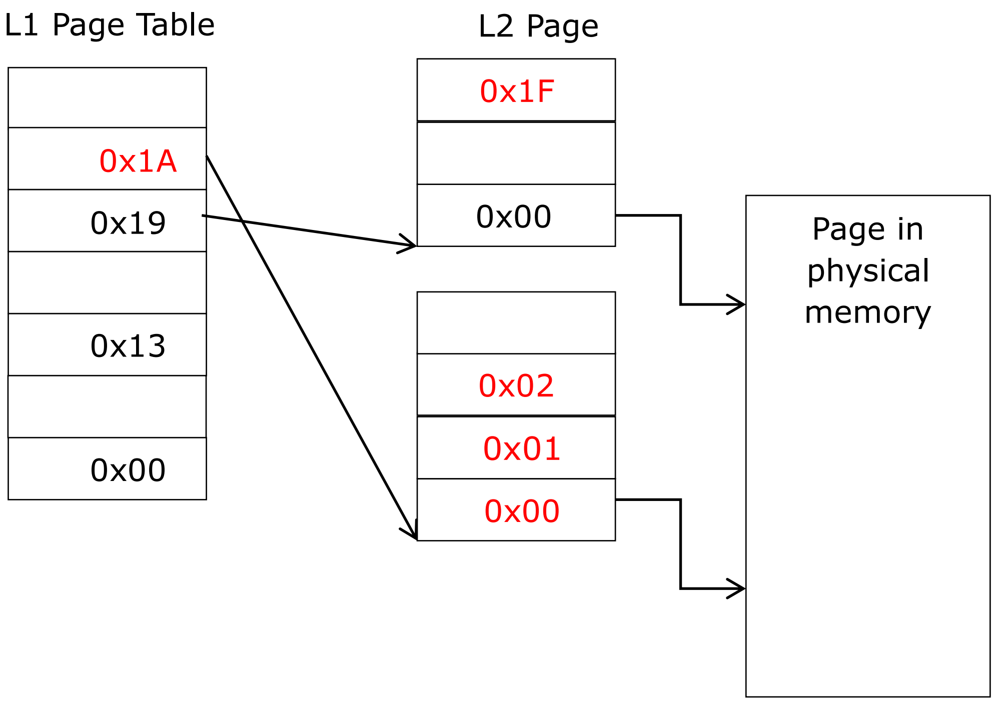
1. (2)  [1] 64 bytes

2. (2’)  Process will exit because of segmentation fault, access the violated address trigger the protection exception.

3. (6’)

|  |  |  |  |
|---|---|---|---|
| [1] | 0xCFC0 – 0xD0BF | [2] 0x19 | [3] 0x1F |
| [4] | 0xB700 – 0xB7FF | [5] 0x16 | [6] 0x1C |

4. (8’)
arr1 0xCFC0 – 0xD0C0, thus L1 PTE index from 0x19 – 0x1A
0x19: L2 PTE index is 0x1F
0x1A: L2 PTE index is 0x00 to 0x02



每个标红项为 2 分，错一个扣 2 分 （5 个标红项）

2) 4

Page size is 64 bytes

As to arr1, Array pointed by arr1 occupies 4 pages. Access to the first byte of each page should raise a page fault exception, that is 4 page faults.

3) 4 + 4

Because of copy-on-write mechanism, there’s no need to allocate new pages for read operations to arr1. Write operation to shared array object pointed by arr2 will cause page fault exception in each process (4 times in each process)

Answer 4 and point out in single process, and there are 2 processes (4’)

5. (6’)

4-level page table , address format is [3:3:3:3:4]
Set k = page table level, n = each level bit number, m = offset bit number
kn + m = 16
2^n * 2 = 2^m = 16, k = 1,2,3,….
Solve above equations.
Get the answer.