上 海 交 通 大 学 试 卷（ A 卷）

（ 2013 至 2014 学年 第 1 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称 计算机系统基础（2） 成绩 _____

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

1. [1] [2] [3] [4] [5] [6]

2. Prog.1

Prog.2

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号	1	2	3	4	5				
得分									
批阅人(流水阅卷教师签名处)									

3.

4. [1] [2] [3] [4]

5.

Problem 2: Signal & Concurrency (20points)

1

2

3

Problem 3: Address Translation (22points)

1 [1] [2] [3] [4]

2 [1] [2] [3] [4] [5]

[6] [7] [8] [9]

[1] [2] [3] [4] [5]

[6] [7] [8] [9]

Problem 4: Virtual Address (32points)

1. [1]

2.

3 [1] [2] [3]

[4] [5] [6]

4. 1)

2)

3)

5.

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

Prog.1

```
#include "csapp.h"
int main(void){
  int fd, fd2;
  char c, c2;
  fd = open("logA", O_RDONLY, 0);

  if(fork() == 0){
    fd2 = open("logA", O_RDONLY, 0);
    read(fd2, &c2, 1);
    printf("Child: c2 = %c\n", c2);
    read(fd, &c, 1);
    printf("Child: c = %c\n", c);
    exit(0);
  }
  wait(NULL);
  fd2 = open("logA", O_RDONLY, 0);
  read(fd2, &c2, 1);
  printf("Parent: c2 = %c\n", c2);
  read(fd, &c, 1);
  printf("Parent: c = %c\n", c);
  exit(0);
}
```

Prog.2

```
#include "csapp.h"
void *thread_func_1(void *vargp){
  int fd = *((int*)vargp);
  printf("Thread_1\n");
}

void *thread_func_2(void *vargp){
  int fd = *((int*)vargp);
  dup2(fd, 1);
  printf("Thread_2\n");
}

int main(void){
  int fd;
  pthread_t pid1,pid2;
  fd  =  open("logB",
         O_WRONLY|O_APPEND, 0);

  printf("Creating...\n");
  pthread_create(&pid1, NULL,
       thread_func_1, (void*)&fd);
  pthread_create(&pid2, NULL,
       thread_func_2, (void*)&fd);
  pthread_join(pid1,NULL);
  pthread_join(pid2,NULL);
  printf("Ending...\n");
  exit(0);
}
```

NOTE: the initial content of file **logA** is "**logA.**" and file **logB** is empty.

1) Please fill the following table with "Yes" or "No" like practice problem 12.6 (6')

Prog.1	Shared by child process?	Referenced by child process?
fd.m	[1]	[2]
c.m	[3]	[4]
Prog.2	Shared by threads?	Referenced by threads?
fd.m	[5]	[6]

2) Please draw the kernel data structures of open files for Prog.1 and Prog.2 respectively like the Figure 10.11~10.13. You should identify the "refcnt" on each open file table. (6')

3) Please list the outputs on screen for Prog.1. (4')

4) Please decide whether the following sentences are printed on screen by Prog.2. You need answer with "Yes", "No", or "Not sure". (1′ * 4 = 4′)

```
[1] "Creating...\n"
[2] "Thread_1\n"
[3] "Thread_2\n"
[4] "Ending... \n"
```

5) Please list all possible contents of file "logB**"** after running Prog.2 one by one. (6′)

Problem 2: Signal and Concurrency (20 points)

1. Read the following code and list the output and your explanation. (2' + 3')

```
int main(void) {
  signal(SIGINT, handler);
  sigemptyset(&sigs);
  sigaddset(&sigs, SIGINT);
  sigprocmask(SIG_BLOCK, &sigs, 0);
  for (int i=0; i<3; i++){
    printf("%d\n", i);
    kill(getpid(), SIGINT);
  }
  sigprocmask(SIG_UNBLOCK, &sigs, 0);
  return 0;
}
```

```
sigset_t sigs

void handler(int sig) {
   printf("handler\n");
}
```

2. The following code can work well **mostly**. But it will occur deadlock when encountered interrupt from keyboard occasionally. Please analyze the cause of deadlock and provide an example. (5')

```
sem_t mutex;

void doSomething(void){
  P(&mutex);
  ...; // Time-consuming work
  V(&mutex);
}
```

```
void handler(int sig){
  doSomething();
}

int main(void){
  sem_init(&mutex,0,1);
  signal(SIGINT, handler);
  doSomething();
}
```

3. When sharing a Printer and a Card Reader between multiple threads, they use two semaphores to protect them. Suppose both semaphores are initiated to 1. Mostly, the following code is running well. But abnormal situation occurs occasionally.

```
void *thread1(void *var){
  P(&print_mutex);
  if(checkPrinter()){
    P(&card_mutex)
    readCard();
    V(&card_mutex);
  }
  V(&print_mutex);
}
```

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

1) Please analyze the abnormal situation by using Progress graph like Figure 12.42. (5′)

2) Please modify above code to avoid abnormal situation, and draw new progress graph. (3′ + 2′)

Problem 3: Address Translation (22 points)

This problem concerns the way virtual addresses are translated into physical addresses. Below are the specifications of the system on which the translation occurs:

- ✧ The main memory is byte addressable.
- ✧ The memory accesses are to **1-byte** words (not 4-byte words).
- ✧ Virtual address are **15 bits** wide.
- ✧ Physical addresses are **12 bits** wide.
- ✧ Both single level page table and the page size are **32 bytes**.
- ✧ The TLB is **4-way** set associative with **16** total entries.
- ✧ The **L1** d-cache is physically addressed and **2-way** set associative, with a **2-byte** line size and **16** total sets.

1. Warm-up Questions (1′ * 4 = 4′)

The number of bits for VPO	**__[1]__**
The number of bits for TLBI	**__[2]__**
The amount of PTE	**__[3]__**
The number of bits for CT	**__[4]__**

The contents of the TLB, first 16 entries of the page table and cache are given below. All numbers are in hexadecimal. According to the illustration and answer the questions. Please fill in the blanks.

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	0E	71	1	CF	00	1
	B0	68	1	36	48	1
1	AF	2D	0	01	00	1
	7E	01	1	45	3F	0
2	00	7F	0	24	50	1
	F0	07	1	09	11	0
3	E0	42	1	BE	15	1
	01	39	1	0F	0D	1

TLB: 4 sets, 16 entries, 4-way set associative

VPN	PPN	Valid	VPN	PPN	Valid
00	7F	1	08	11	0
01	2B	0	09	3F	0
02	0B	1	0A	00	1
03	00	0	0B	78	1
04	24	1	0C	01	0
05	2F	1	0D	1C	1
06	45	1	0E	6F	1
07	27	1	0F	39	1

Page Table: Only the first 16 PTEs are shown

Index	Tag	Valid	Blk0	Blk1	Tag	Valid	Blk0	Blk1
0	21	0	09	EE	5B	0	12	61
1	60	1	5F	4D	30	1	3D	2B
2	3A	1	F0	45	7B	1	21	07
3	09	0	86	37	41	0	65	51
4	18	0	-	-	76	0	4A	4B
5	45	1	EF	E0	1B	1	3E	25
6	22	0	11	C2	1A	0	DF	CD
7	2D	1	53	33	39	1	2F	4A
8	71	1	88	64	01	1	34	3A
9	1C	1	71	F0	2B	1	70	18
A	40	0	49	4F	00	0	-	-
B	46	1	8F	00	16	1	D4	05
C	2B	0	-	-	51	0	09	47
D	14	0	1F	CD	61	0	23	00
E	65	1	24	57	05	1	7A	09
F	3F	1	01	48	00	1	56	08

Cache: 16 sets, 1-byte blocks, 2-way set associative

2. Please translate virtual address to physical address and get the data from cache if hit (otherwise entering '--') (1' * 18 = 18')

Parameter	Value
Virtual Address	0x01E5
VPN	0x__[1]__
TLB Index:	0x__[2]__
TLB Tag:	0x__[3]__
TLB Hit? (Y/N)	__[4]__
Page Fault? (Y/N)	__[5]__
PPN	0x__[6]__
Physical Address	0x__[7]__
Cache hit? (Y/N)	__[8]__
Cache byte returned	__[9]__

Parameter	Value
Virtual Address	0x00BF
VPN	0x__[1]__
TLB Index:	0x__[2]__
TLB Tag:	0x__[3]__
TLB Hit? (Y/N)	__[4]__
Page Fault? (Y/N)	__[5]__
PPN	0x__[6]__
Physical Address	0x__[7]__
Cache hit? (Y/N)	__[8]__
Cache byte returned	__[9]__

Problem 4: Virtual Memory (32 points)

This problem is based on a **16-bit** byte addressed system. Suppose its virtual address is also **16-bit** wide and has **two-level** page table, each PTE takes **2 byte**s. [15:11] of virtual address is index of L1 page table entry (PTE), and [10:6] is index of L2 page table entry, remaining bits represent offset within a page.

```
#include <unistd.h>
#include <sys/mman.h>
#include <stdio.h>

#define SIZE  (4 * 64)
int main (void) {
   int *arr1 = NULL;
   int *arr2 = NULL;
   int idx;
A:
   arr1 = mmap(0, SIZE, PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANON, 0, 0);
   arr2 = mmap(0, SIZE, PROT_READ | PROT_WRITE,
               MAP_SHARED | MAP_ANON, 0, 0);

   for (idx = 0; idx < SIZE/sizeof(int); idx++)
      arr1[idx] = 1;
B:
   fork();
C:
   for (idx = 0; idx < SIZE/sizeof(int); idx++)
      arr2[idx] = arr1[idx];
D:
   return 0;
}
```

Note: copy on write (COW) is supported, and **`sizeof(int) == 2`**

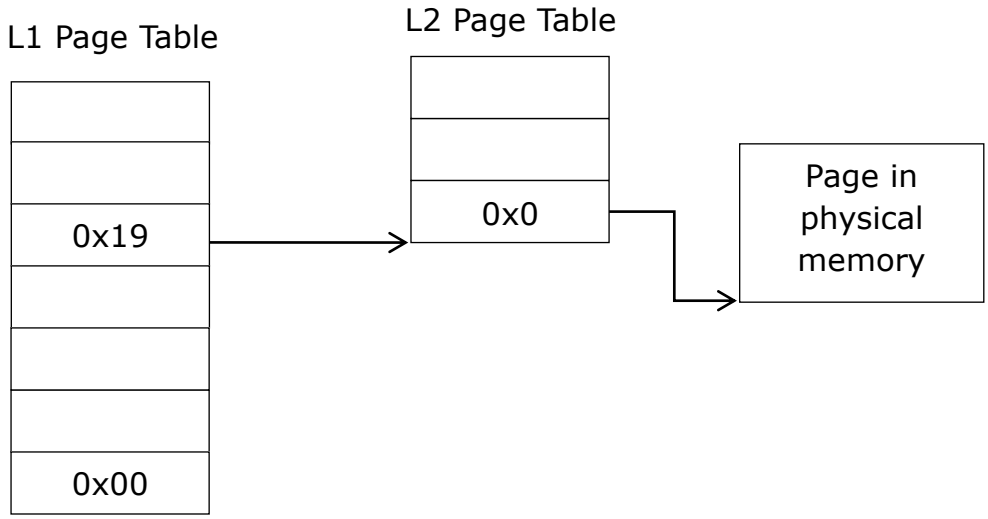
1. Page Size = __[1]__ bytes. (2')
2. If the following code is inserted before label **A**, please explain what will happen? (2')

   ```
   *arr1 = 1;
   ```

3. After calling **`mmap()`**, **`arr1`** is **`0xcfc0`** and **`arr2`** is **`0xb700`**. Please fill in the following blanks in Hex (1'*6 = 6')

   Address range of the area pointed by **arr1**: __[1]__
   Index of L1 PTE of **arr1**: __[2]__ Index of L2 PTE of **arr1**: __[3]__
   Address range of the area pointed by **arr2**: __[4]___
   Index of L1 PTE of **arr2**: __[5]__ Index of L2 PTE of **arr2**: __[6]__

4. Assume the page table is like below when the program arrives at label A. The number within block is the index of page table. The white block without number means one or more empty page table entries.



   1) Please draw a graph like above to indicate the page table when the program reach label B. You can use a black block to represent multiple filled PDE/PTEs (keep it tidy). (8')
   2) How many page fault exceptions are raised by codes between **label A** and **B**. Please briefly explain your answer. (2' + 2' = 4')
   3) How many page fault exceptions are raised by codes between **label C** and **D**. Please briefly explain your answer. (2' + 2' = 4')

5. Suppose we change above system's page size into **16 bytes**, each PTE still takes **2 bytes**. How many levels of page tables would be required to map a 16-bit address if **every page table exactly fits into one single page**? Show your design and explain it in detail. (6')