上 海 交 通 大 学 试 卷（ A 卷）

（ 2014 至 2015 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: Address Translation (21points)

1. [1] [2] [3]
2. [1] [2] [3] [4]
   [5] [6] [7]
   [1] [2] [3] [4]
   [5] [6] [7]
3.

Problem 2: I/O (16points)

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

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)									

Problem 3: Concurrency (15points)

1. [1] [2] [3]

   [4] [5] [6]

   [7] [8] [9]

2.

Problem 4: Deadlock (5 points)

Problem 5: Networking (18points)

1

2

3

4

5

Problem 6: Virtual Address (25points)

1. 1)

   2)

3)

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

2)

3)

3.

Problem 1: Address Translation (21 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).
- ✧ The system is configured with **128 MB** of virtual memory.
- ✧ The system use **one-level** page table.
- ✧ VPN is **14 bits** wide.
- ✧ Physical addresses are **20 bits** wide.
- ✧ The TLB is **4-way** set associative with **16** total entries.

1. Warm-up Questions (1′ * 3 = 3′)

The page size	**__[1]__**
The amount of PTE	**__[2]__**
The number of bits for TLBT	**__[3]__**

The contents of the TLB and first 16 entries of the page table 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	0EE	71	1	CFF	00	1
	B00	68	1	360	48	1
1	AF0	2D	0	010	00	1
	7E0	01	1	450	3F	0
2	0CF	7F	1	001	50	0
	F00	07	1	090	11	0
3	E00	42	1	BE1	15	1
	013	39	1	0F2	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	66	1	0E	6F	1
07	27	1	0F	39	1

Page Table: Only the first 16 PTEs are shown

2. Please translate virtual address to physical address and get the data from cache if hit (otherwise entering '--') (1' * 14 = 14')

Parameter	Value
Virtual Address	0x067D234
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]__

Parameter	Value
Virtual Address	0x000CABC
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]__

3. Most processors do address translation before accessing the cache since the cache uses **physical** address. Please explain why not directly use virtual address instead of physical address to access the cache and delay the address translation after checking the cache? (4')

Problem 2: I/O (16 points)

```
#include "csapp.h"

int cnt;

void *foo(void *id) {
    int fd1, fdw;
    char c;

    cnt ++;
    fd1 = (*(int *)id);
    fdw = open("ics.txt",
           O_WRONLY|O_APPEND,0);

    read(fd1, &c, 1);
    write(fdw, &c, 1);
    close(fdw);
    return;
}
```

```
int main (void) {
    int fd1, fd2;
    pthread_t pid1;
    cnt = 0;
    fd1 = open("ics.txt",O_RDONLY,0);
    fd2 = open("ics.txt",O_RDONLY,0);
    pthread_create(&pid1, NULL,
                   foo,(void*)&fd1);
    pthread_join(pid1,NULL);
    if(fork()==0) {
        dup2(fd2, fd1);
        foo((void*)&fd1);
        exit(0);
    }
    wait(NULL);
    foo((void*)&fd1);
    exit(0);
}
```

NOTE: the initial content of file **`ics.txt`** is "**`ics2014`**".

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

Prog.1	Referenced by peer thread?	Referenced by child process?
fd1.m	[1]	[2]
fd2.m	[3]	[4]
cnt	[5]	[6]

2) Please write down the content in the file **`ics.txt`** at three various moments? (2'*3)

   [1] before the **1st** time to call **`close(fdw)`**

   [2] before the **2nd** time to call **`close(fdw)`**

   [3] before the **3rd** time to call **`close(fdw)`**

3) Please draw the kernel data structures of the file **`ics.txt`** at moment "**[2]**" like the Figure 10.12 and 10.13 in text book. (4')

   NOTE: You only need to draw contents of file table and v-node table concerning **`ics.txt`**, but you need to draw entirely descriptor table and mark the **`fd`** number.

   You need to mark each file table entry according to the sequence of opening. (For example, mark the first entry as "A", the second as "B")

Problem 3: Concurrency (15 points)

```
#include "csapp.h"

void *do0(void *p){
    P(&_[4]_);
    putchar('0');
    V(&_[5]_);
}
void *do1(void *p){
    P(&_[6]_);
    putchar('1');
    V(&_[7]_);
}
void *do2(void *p){
    P(&_[8]_);
    putchar('2');
    V(&_[9]_);
}
```

```
sem_t a, b, c;
void *(*dos[3])(void *) = {do0, do1, do2};

int main(void) {
    pthread_t tid[6];
    int i;
    sem_init(&a,0,_[1]__);
    sem_init(&b,0,_[2]__);
    sem_init(&c,0,_[3]__);
    for (i=0; i<6; ++i)
        pthread_create(&tid[i], NULL,
                       dos[i%3], NULL);
    for (i=0; i<6; ++i) {
        pthread_join(tid[i], NULL);
        // if (i % 3 == 2) putchar('\n'); ①
    }
    putchar('\n');
    return 0;
}
```

1. Please fill in the blanks to make the above program print "012012\n". (9')
2. Suppose we want to make the program print "012\n012\n\n", so you **uncomment** the instructions at ①. However, the program doesn't always print the string you want. Please explain **why** and give **2 possible** unexpected output strings (besides "012\n012\n\n") (2'+4') **NOTE**: use '\n' rather than a real new line in your answer.

Problem 4: Deadlock (5 points)

Please consider the following executing flow. Is it possible to cause deadlock? If yes, give an execution order that deadlocks, otherwise explain why. (2'+3')

Initially: a=1, b=1, c=1	
Thread 1	**Thread 2**
P(a)	P(c)
P(b)	P(b)
V(a)	V(c)
P(c)	P(a)
V(b)	V(b)
V(c)	V(a)

Problem 5: Networking (18 points)

Your task is to design a simple scoring system to give each TA a score. This system includes one server and several clients. The server process serves at **59.78.3.103:6666**

Firstly, a client needs to send **login** message including "**LOGIN**" command and a **student ID** to server. If the **student ID** is valid, server will send "**LOGIN OK**" back to client, otherwise client will receive "**LOGIN ERR**". If client receives "**LOGIN ERR**", it will print "**LOGIN ERR**" on screen and call **exit** function. If it receives "**LOGIN OK**", it will then send **grade** message which include "**SCORE**" command and four scores (e.g., **SCORE 1 2 3 4**) to server. Client will receive "**GRADE OK**" if the **grade** message is valid, otherwise it will receive "**GRADE ERR**". No matter which message the client receives, it will print such message on the screen and call exit function. (**NOTE**: The server never send invalid message to client, and there is no network problem.)

Examples:

1].Client => Server: LOGIN 5130379000 (This ID is invalid)
Server => Client: LOGIN ERR
Then the Client prints: LOGIN ERR

2].Client => Server: LOGIN 5130379001
Server => Client: LOGIN OK
Client => Server: SCORE 100 100 100 100 100 (Must provide 4 scores)
Server => Client: GRADE ERR
Then the Client prints: GRADE ERR

3].Client => Server: LOGIN 5130379001
Server => Client: LOGIN OK
Client => Server: SCORE 100 100 100 100
Server => Client: GRADE OK
Then the Client prints: GRADE OK

Please complete the client side code (18') (NOTE: you can use **sprintf** to construct the message.)

```
int main(void) {
    /* don't need consider overflow. */
    int serverfd, student_id, score1, score2, score3, score4;

    /* the buffers used for communicate with server. */
    char recv_buf[1024], send_buf[1024];

    /* get the scores and student_id from user. */
    getStudentId(&student_id);
    getScore(&score1,&score2,&score3,&score4);

    /* build connection with the server */
    __[1]__ /* write your code here */


    /* initialize the login message and send it to server. */
    __[2]__ /* write your code here */


    /* handle the login response (you may need to print some message
    according to the protocol). */
    __[3]__ /* write your code here */


    /* initialize the grade message */
    __[4]__ /* write your code here */


    /* handle the grade response (you may need to print some message
    according to the protocol). */
    __[5]__ /* write your code here */


    return 0;
}
```

Problem 6: Virtual Memory (25 points)

Suppose the program runs on the Pentium/Linux Memory System discussed in section 9.7 of the CSAPP. Please answer the following questions. **Note** copy on write (COW) is supported

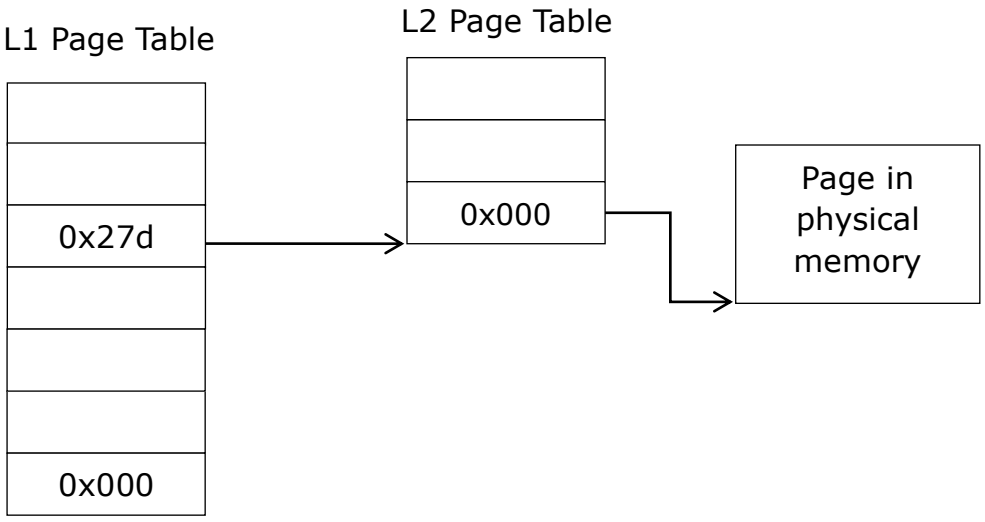
```
#include <unistd.h>
#include <sys/mman.h>
#include <stdio.h>

#define PAGE_SIZE (1<<12)
int main (void) {
   int *sarr = NULL;
   int *parr = NULL;
   volatile int idx;   /* idx will be stored in the stack */
A:
   parr = mmap(0, 64 * PAGE_SIZE, PROT_READ|PROT_WRITE,
               __[1]__, 0, 0);
   sarr = mmap(0, 64 * PAGE_SIZE, PROT_READ|PROT_WRITE,
               MAP_SHARED|MAP_ANON, 0, 0);

   for (idx = 0; idx < 64*PAGE_SIZE/sizeof(int); idx++)
      sarr[idx] = 2012;
B:
   if(fork() == 0) {
      for (idx = 0; idx < 64*PAGE_SIZE/sizeof(int); idx++)
         parr[idx] = 2013;

      for (idx = 0; idx < 64*PAGE_SIZE/sizeof(int); idx++)
         sarr[idx] = 2014;
C:
      return 0;
   } else {
      waitpid(-1, NULL, 0);
      printf("%d\n", sarr[0]);
   }
   return 0;
}
```

1. Warming-up questions (10’)
   1) `parr` should point to a **private** array object which is initialized by **zero-filled** pages. Please fill in the blank in the program properly.(2’)
   2) After `fork()`, the first COW will occur in __[2]__ (2’)
      A. Kernel code and data
      B. User stack

C. Memory mapped region for shared libraries
D. **`.data`** section
E. None of the above
Please also write down your reasons. (**NOTE**: Assume that there is NO any COW during **`fork`**) (2')

3) What is the output for the **`printf`**? Please also write down your reasons (2'+2')

2. Assume the address of **`parr`** is **`0x9f7ce000`** and that of **`sarr`** is **`0x9f80e000`** before label **`B`**. The following figure shows the page table 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. Please answer the following questions (**NOTE** please ignore the page fault on the **stack** for the following questions) (12')



1) Please fill in the following blanks. (4')
The size of **`parr`** is _[1]__ bytes
The address range of array pointed by **`parr`** is between **`0x9f7ce000`** and __[2]__
The address range of array pointed by **`sarr`** is between **`0x9f80e000`** and __[3]__
The index of L1 PTE for **`sarr`** is __[4]__

2) Please draw a graph like above to show the page table of the child process when the program reach label **C**. Note that you can use a **black** block to represent the consecutive PDE/PTEs. (For example, you can draw (**`0x1`**) (black block) (**`0xA`**) to represent the consecutive entries from **`0x1`** to **`0xA`**) (4')

3) How many page fault exceptions are raised by code between label **A** and **C**? (2')
Please also write down your explanation.(2')

3. Based on this program, do you know for which purpose a shared object will be used in multi-process programs? (3')