上 海 交 通 大 学 试 卷（ A 卷）

（ 2018 至 2019 学年 第 2 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: CPU Scheduling

1. [1] [2]

   [3] [4]

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

   [5] [6] [7] [8]

   [9] [10] [11] [12]

3.

Problem 2: Replacement Policy

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

   [5] [6] [7] [8]

2. [9] [10] [11] [12]

   [13] [14] [15] [16]

   [17] [18]

3.

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

Problem 3: Address Translation

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

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

   [5]   [6]   [7]   [8]

3.

Problem 4: Concurrency

1

2. [1]   [2]   [3]

    [4]   [5]   [6]

    [7]   [8]   [9]

Problem 5: Locks

1. 1)

    2)

    3)

2. 1)

    2)

3)

4)

5)

6)

Problem 1: Scheduling (20 points)

1. Please fill the following table with policy or mechanism. (1' * 4 = 4')

	Policy or Mechanism?
First in, first out	Policy
Implemented FIFO by using a queue	Mechanism
First come, first serve	[1]
How long a time-slide should be	[2]
How to make schedule decisions between different threads	[3]
Use MMU for virtual memory management	[4]

2. Assume we have the following two jobs in the workload and no I/O issues are involved. Please fill in the following two tables with the execution of CPU when we decide to use two different schedule policies respectively. (**NOTE**: Time 0 means the task running during [0ms,1ms]) (1' * 12 = 12')

Job	Arrival Time	Run time
A	0ms	5ms
B	3ms	5ms

Assume we decide to use **FIFO** scheduling policy

Time	0	1	2	3	4	5	6	7	8	9
CPU	A	A	A	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Assume we decide to use **MLFQ** scheduling policy with **2 priority queues**, the highest one has time-slice of **1ms**, the lowest one has time-slice of **2ms**. We use **RR** in each queue and priority boost **isn't supported**.

Time	0	1	2	3	4	5	6	7	8	9
CPU	A	A	A	B	B	[8]	[9]	[10]	[11]	[12]

3. What do you think are the **advantages** of MLFQ compared to FIFO. (**Note**: You need to give **two** advantages) (4')

Problem 2: Replacement Policy (21 points)

Suppose we have a primary which has **3 physical blocks**, please complete the following questions.

1. Suppose we are using **FIFO** replacement policy, please complete the following table. (1' * 8 = 8') (**Note**: no need to consider the order of primary device contents)

Time	0	1	2	3	4	5	6	7	8
Reference String	1	2	3	4	2	5	2	1	5
Primary Device Contents	1	1 2	1 2 3	2 3 4	2 3 4	[1]	[3]	[5]	[7]
Hit (Y or N)	N	N	N	N	Y	[2]	[4]	[6]	[8]

2. Suppose we are using **LRU** replacement policy, please complete the following table. (1' * 8 = 8') (**Note**: no need to consider the order of primary device contents)

Time	0	1	2	3	4	5	6	7	8
Reference String	1	2	3	4	2	5	2	1	5
Primary Device Contents	1	1 2	1 2 3	[9]	3 4 2	[11]	[13]	[15]	[17]
Hit (Y or N)	N	N	N	[10]	Y	[12]	[14]	[16]	[18]

3. If you are a god, and you know the "**Optimal Replacement Policy**". What is the hit rate if we use the Optimal Replacement Policy? (3')

Problem 3: Address Translation (21 points)

Assume we have a machine with the following specifications:
- ◇ The memory is byte-addressable
- ◇ 48-bit physical address space
- ◇ 42-bit virtual address space
- ◇ Each page is 64KB
- ◇ The size of one page table equals to the size of page
- ◇ length of each PTE is 8B
- ◇ 512 entries, 4-way associative TLB
- ◇ LRU replacement policy in TLB
- ◇ Each L1 cache line is 64B
- ◇ 1KB, 4-way associative L1 cache

1. Please fill the following table. (1' * 5 = 5')

The VPO bits	**[1]**
The number of PTE in one page table	**[2]**
The number of VPN bits for each level	**[3]**
The TLB tag bits	**[4]**
The number of page table level	**[5]**

2. Given the following page table contents and cache/TLB state, finish the following address translation. Please fill the blanks in hexadecimal notation. If the value is unknown or meaningless, enter "--" for them. (1.5' * 8 = 12')

   NOTE: Accesses are independent, which means they won't affect the TLB and cache state in the next access. You **don't** need to consider cache accesses of page tables.

VPN	PPN	Valid
0A	2	1
12	c	0
3C	9	1

Part of L1 page table

VPN	PPN	Valid
10	1f	1
80	2c	0

PT @0x20000

VPN	PPN	Valid
03	36	1
1A	02	0

PT @0x90000

Set	Valid	Tag	PPN	Valid	Tag	PPN
0	1	9f7	aa80	1	bf0	01a5
	0	281	8bcf	1	d38	201f
1	0	ada	7790	1	147	00af
	1	f20	6012	1	65f	2019
2	1	ada	3960	0	80f	4f88
	0	ada	790a	0	3bf	8a12
3	0	311	0127	1	f00	0036
	0	2d8	1213	0	db2	1989

Part of TLB state

Set	Valid	Tag	bytes	Valid	Tag	bytes
0	0	0015	...	1	063d	...
	1	3aaa	...	1	366d	...
1	0	366d	...	0	19bb	...
	1	340f	...	1	2ca8	...
2	1	2ca5	...	1	2caa	...
	0	0002	...	1	30b5	...
3	1	01de	...	0	28a0	...
	1	3379	...	1	1ae8	...

Part of cache state

Parameter	Value
Virtual Address	0x14080aa80
TLB Hit? (Y/N)	**[1]**
Page Fault? (Y/N)	**[2]**
Physical Address	**[3]**
Cache Hit? (Y/N)	**[4]**

Parameter	Value
Virtual Address	0x780036d03
TLB Hit? (Y/N)	**[5]**
Page Fault? (Y/N)	**[6]**
Physical Address	**[7]**
Cache Hit? (Y/N)	**[8]**

3. If at this time OS schedules to another process which also accesses virtual address **0x780036d03,** will it access the same physical page as the previous process accessed in problem 2? Why? (4')

Problem 4: Concurrency (17 points)

1. Deadlock is an important problem in concurrent programs. Consider the below execution flow. Whether it will cause deadlock or not? (2') Please draw a **progress graph** and explain the reason base on the graph. (6')

Initially: A=1, B=1, C=1		
Thread	Thread 1	Thread 2
Step1	P(A)	P(C)
Step2	P(B)	V(C)
Step3	P(C)	P(B)
Step4	V(A)	P(A)
Step5	V(B)	V(B)
Step6	V(C)	V(A)

2. Please fill in the blanks with initial values for the three semaphores and add `P()` and `V()` semaphore operations such that the process is guaranteed to terminate. (**NOTE**: You can only fill in **one P(x) or V(x)** operation in [4]~[9]) (9')

   HINT: Using **a** and **b** as iterators for each thread to control loop times while using **c** as lock to protect the modification on variable **x**.

```
/* Initialize x */
   int x = 1;

/* Initialize semaphores */
   sem_t a, b, c;
   sem_init(&a, 0, _[1]_);
   sem_init(&b, 0, _[2]_);
   sem_init(&c, 0, _[3]_);
```

```
void thread1()                      void thread2()
{                                   {
    while (x != 18) {                   while (x != 18) {
        ___[4]___;                          ___[7]___;
        ___[5]___;                          ___[8]___;
        x = x * 2;                          x = x * 3;
        ___[6]___;                          ___[9]___;
    }                                   }
    exit(0);                            exit(0);
}                                   }
```

Problem 5: Lock (21 points)

1. Sam modifies the ticket lock by adding one line "`IDLE(...);`" in the while-loop.

   ✧ `IDLE(n)` will consume $C*n$ CPU cycles where C is a user-defined **constant**.

The original ticket lock	A modified ticket lock
```void lock(lock_t *lock) { int myturn = FetchAndAdd(&lock->ticket); while (lock->turn != myturn) ; }```	```void lock(lock_t *lock) { int myturn = FetchAndAdd(&lock->ticket); while (lock->turn != myturn) IDLE(myturn - lock->turn); }```

   1) What's the **advantage** after adding the "`IDLE(...);`" line? (2')

   2) What's the **disadvantage** after adding the "`IDLE(...);`" line? (2')
      HINT: Consider what if an inappropriate constant C is chosen.

   3) For setting **IDLE** time, why to use "`myturn-lock->turn`" rather than a fixed value? (2')

2. Barrier is commonly used to synchronize the execution of a given number of threads. For example, suppose a barrier is initialized to synchronize **2 threads**. When the first thread calls `barrier_wait`, it will wait until the second thread calls `barrier_wait`. After two threads come, both of them will return from the `barrier_wait`.

```
1 typedef struct __barrier_t {
2    int count;
3    int sense;
4    int total;
5 } barrier_t;

6 void barrier_init(barrier_t *b, int total) {
7    b->count = 0;
8    b->sense = 0;
9    b->total = total;
10 }

11 void barrier_wait(barrier_t *b) {
12    int local_sense = !(b->sense);
13    if (FetchAndAdd(&b->count) == (b->total-1)) {
14       b->count = 0;
15       b->sense = local_sense;
16    }
17    else
18       while (local_sense != b->sense);
19 }
```

Please try to understand the code and answer the questions below.

1) Suppose Sam wants to use a barrier to synchronize **5 threads**. Please describe what will happen if he initializes the barrier with `total=6`. (2')

2) Suppose Sam wants to use a barrier to synchronize **5 threads**. Please describe what will happen if he initializes the barrier with `total=4`. (2')

3) What will happen if **Line 15** is removed? (2')

4) Please describe why **Line 14** is necessary. (2')

5) Is it still correct if switching **Line 14** and **15**? Please explain your answer. (3')

6) You are required to use this barrier to synchronize the process and its **forked child**. In this case, however, the memory spaces of these two processes are isolated. The modification of `b->sense` in one process does not propagate to the other process. Thus, how do you make it work using the virtual memory mapping mechanism we learned in class? (4') **NOTE**: you **CAN NOT** modify the barrier implementation.