上 海 交 通 大 学 试 卷（期 中）

（2018 至 2019 学年　第 1 学期）

班级号_____ 学号_____ 姓名_____

课程名称_____计算机系统工程_____ 成绩_____

Problem 1: File System (24')

1. According to the inode-based FS you have learnt in class, could you please describe the process of looking up the data blocks of the file '*/cse/courses/mid-exam.pdf*' (4') (e.g., which disk blocks, inodes, etc.)

   inode 0->data block of /->inode of /cse/->data blocks of / (maybe indirect blocks of /)->inode of /cse/courses/->data blocks->inode of /cse/courses/mid-exam.pdf->data blocks

2. What are the benefits of using a file system? Why not just access data directly on a disk, just as accessing memory? (4')

   Multiple users may share the disk, FS can help to provide security and isolation. Disk is a persistent storage, FS is needed to ensure consistency and fault tolerance.

3. Bob is implementing a prototype of a file system according to what he has learnt in class, and he finds out that the file system has poor performance on the 'Rododo' benchmark. The benchmark is full of operations with the following pattern:

   1. OPEN source_file
   2. READ the first block data of source_file
   3. WRITE the data to destination_file

   Each source/destination file is in a 256-level deep directory (e.g., "/1/2/3/.../256/a.txt"), and every directory at each level has 256 different directories/files.

   The source file list:

   /verylonglonglonglonglevel1dir1/verylonglonglonglonglevel2dir1/…/verylonglonglonglonglevel256dir1/sourcefile1

   …

   /verylonglonglonglonglevel1dir256/verylonglonglonglonglevel2dir256/…/verylonglonglonglonglevel256dir256/sourcefile256

   The destination file list:

```
/verylonglonglonglonglevel1dir1/verylonglonglonglonglevel2dir1/…/verylonglon
glonglonglevel256dir1/destinationfile1
…
/verylonglonglonglonglevel1dir256/verylonglonglonglonglevel2dir256/…/verylo
nglonglonglonglevel256dir256/destinationfile256
```

A. What are the possible reasons that the file system has a poor performance on this benchmark? (4')

Every time the benchmark tries to open a source file, it needs to read inode and data blocks of directories in each level. This may move disk head back and forward, and the seek time is very long.

B. For each reason you listed, can you help Bob to optimize his file system to achieve better performance on this benchmark? (4')

Make inode and data block co-located/embedded so the disk head doesn't need to move again and again.

4. Alice uses hash function to accelerate file lookup in her FS.

```
inode_index_t hash(string filename)
```

The hash function accepts the entire file path string as input, and the return value is the index to the corresponding inode.

But she came across a problem when she performed **MOVE** operation on a directory to another. Sometimes, the **MOVE** operation is very slow.

(a) Why is **MOVE** operation very slow (in what cases and explain)? (4')

All subdirectories/files need to be rehashed recursively, and its inodes need to be moved.

(b) Please design another solution which doesn't suffer from the MOVE problem. (4')

Instead of use hash as inode number, just remember recently used path in a in-memory cache based on tree structures, and directly modify the cache when need to move entries.

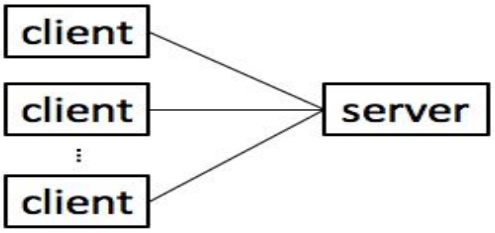
Problem 2: Performance (22')

Bob develops his key-value store service using Client/Server model. The server hosts a table that contains `<key, value>` pairs. For the GET() operation, the client sends requests containing a key to the server, and server responds with the value of the key. Each request/response packet has 100 bytes header and 100 bytes of data (key or value).

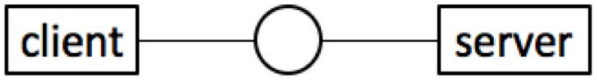
Suppose the link in the network has a capacity of 20 Mbit/s and a one-way latency of 10ms and all the links considered in this problem are the same. The maximum packet size in the network is 1000 bytes. The lookup time in the server is 10us in average.

1. Please compute the latency of the system. (4')

2. Suppose there is one server thread handling requests and multiple clients taking full advantage of the system. Each client and the server is connected by a link. Please compute the throughput (requests/second) of the system. (4')



3. As the service scales, Bob decides to add a switch between the client and the server as shown below. Only consider one client thread and one server thread for simplicity.



Bob has come up with three possible ways of optimization.

A. **Parallelism**: Add an additional network link, with 20Mbit/s capacity and 10ms latency between the client and the switch. Now clients can send requests over both links.
B. **Caching**: Cache server responses at the switch. If the switch has a cached `<key, value>` pair, it will return the value immediately when request contains key. Suppose the cache lookup time can be neglected.
C. **Batching**: Have the client batch the requests before sending to the server. The request could include multiple keys and assume the server can process multi-key packets.

a. Please analyze the effects of each optimization on latency. Which (if any) will most significantly improve latency? Please give your reasons. (6')
b. Please analyze the effects of each optimization on throughput. Which (if any) will most significantly improve throughput? Please give your reasons. (6')

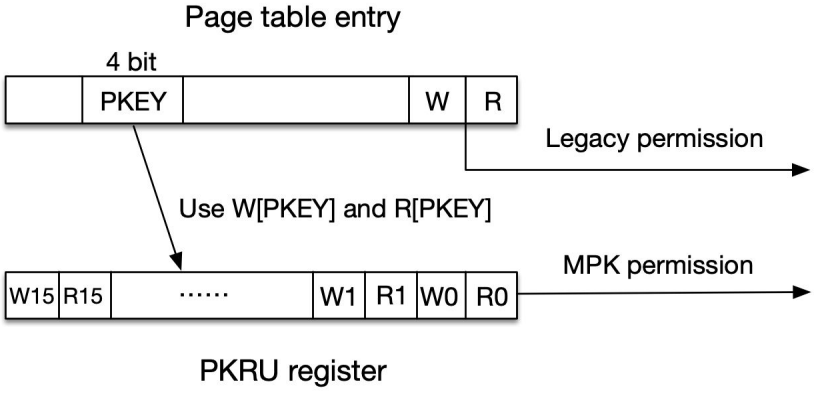
4. The server uses FCFS (first come first serve) policy for CPU scheduling. Gradually Bob finds the CPU idling. Please give possible reasons for the phenomenon and solutions. (4')

Problem 3: Memory (26')

1. In virtualization, when using *shadow paging* mechanism, the page tables in the guest VMs are set as read-only. Why? (4')

2. If an application wants to set a page as read-only (for example, the page contains some important data and the application needs to protect them from being mistakenly overwritten), how to do that? What will happen if a read-only page is written? Please describe the process in details. (*hint: the behavior of application, kernel and CPU.*) (4')

A new hardware feature, **MPK** (Memory Protection Keys), is an extension to existing page-based memory permissions. In MPK, each page table entry uses 4 bits (previously unused) as a "key" (thus there could be as many as 16 keys). Each page can be associated with one key. There is also a new 32-bit processor register, **PKRU**, with two bits for each key value (referred as W[PKEY] and R[PKEY], for write and read permission). As shown in the following figure:



Now, when accessing a virtual address, there are *two* permission checks. First, the legacy write/read permission is checked by referencing W bit and R bit in page table entry (PTE). In addition, CPU must check R[PKEY] bits and W[PKEY] bits in the **PKRU** register to get MPK introduced read/write permission for the virtual address. A successful access should pass both legacy permission check and MPK permission check. The PKRU register can be set in userspace.

3. For the following configuration of PTEs and PKRU, what is the read/write permission for each virtual address and thread? Fill in the following table. (6')

Virtual address / PTE

0x5000		0000		1	1
0x6000		0001		0	1
0x7000		0001		0	0
0x8000		0010		1	0

PKRU

Thread 0	·····	0	1	0	1	1	1
Thread 1	·····	1	1	0	0	0	1

Address	Thread 0		Thread 1	
	Write	Read	Write	Read
0x5000	Y	Y	N	Y
0x6000				
0x7000				
0x8000				

4. Legacy permission already performs read/write check, can you give some benefits of the additional MPK permission? (4')

The interfaces for MPK in Linux are listed below (slightly modified for simplicity):

- **`pkey_alloc()`:** Alloc a PKEY.
- **`pkey_set(pkey, perm)`:** Change the read/write permission of **`pkey`** in PKRU register (ie, W[**`pkey`**] and R[**`pkey`**]) to **`perm`**. **`perm`** can be R, W, R|W or 0.
- **`pkey_mprotect(addr, size, pkey)`:** Change the PKEY value to **`pkey`** in PTEs which translate address **`addr`** within size **`size`**.
- **`pkey_free(pkey)`:** Free a PKEY.

Suppose the default pkey for all PTEs is 0. The default permission for pkey 0 is R|W. You never get pkey 0 from **`pkey_alloc()`**.

5. In current design, after calling **`pkey_free(pkey)`**, the PKEY fields of PTEs whose PKEY fields are **`pkey`** will not be reset to default value 0. Why? What is the pros and cons for such design? (4')

6. The follow program tries to get input from the user. The length of input must be less than 4096 bytes. Can you use MPK's interface to prevent the **`scanf`** from overflow? (4')

```
__attribute__((aligned(4096))) char buf[4096];

int main() {
    // your code here:

    scanf("%s", buf);
    // your code here:

}
```

Problem 4: Thread & Synchronization (26')

1. Given the following three versions of `send()`:

```
send1(bb, m):
    while True:
        if bb.in - bb.out < N:
            acquire(bb.send_lock)
            bb.buf[bb.in mod N] <- m
            bb.in <- bb.in + 1
            release(bb.send_lock)
            return
```

```
send2(bb, m):
    while True:
        acquire(bb.send_lock)
        if bb.in - bb.out < N:
            bb.buf[bb.in mod N] <- m
            bb.in <- bb.in + 1
            release(bb.send_lock)
            return
        release(bb.send_lock)
```

```
1   send3(bb, m):
2       while True:
3           if bb.in - bb.out < N:
4               acquire(bb.send_lock)
5               if bb.in - bb.out >= N:
6                   release(bb.send_lock)
7                   continue
8               bb.buf[bb.in mod N] <- m
9               bb.in <- bb.in + 1
10              release(bb.send_lock)
11              return
```

a. Is **send3** correct? Why? (4')

b. Compare **send2** and **send3**, in which cases will **send2** be faster than **send3**? Why? (4')

2. Given the following code of **yield_wait()**:

```
1    yield_wait(): // called by wait(), set state to WAITING
2        id = cpus[CPU].thread
3        cpus[CPU].thread = null
4        threads[id].sp = SP
5        SP = cpus[CPU].stack
6        do:
7            id = (id + 1) mod N
8        while threads[id].state != RUNNABLE
9        SP = threads[id].sp
10       threads[id].state = RUNNING
11       cpus[CPU].thread = id
```

a) The implementation is buggy. It may hang in some corner cases. Please show a condition that the code will hang and explain why. (4')

b) Please give at least two solutions to solve the problem. (3' * 2)

3. Assume we don't have **wait()** and **yield_wait()**. The code of **yield()** is:

```
1   yield():
2       acquire(t_lock)
3       id = cpus[CPU].thread
4       threads[id].state = RUNNABLE
5       threads[id].sp = SP
6       SP = cpus[CPU].stack
7       do:
8           id = (id + 1) mod N
9       while threads[id].state != RUNNABLE
10      threads[id].state = RUNNING
11      SP = threads[id].sp
12      cpus[CPU].thread = id
13      release(t_lock)
```

a) Which variable may have race condition if we remove the **t_lock**? Please describe in which cases the race condition will occur. (4')

b) Please remove the **t_lock** and use atomic operation **compare_and_swap()** to protect the variable. (4')

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号									
得分									
批阅人(流水阅卷教师签名处)									