上 海 交 通 大 学 试 卷（ B 卷）

（ 2020 至 2021 学年 第 2 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称_____操作系统_____成绩 _____

一、 中断与系统调用(28')

1. 现代 CPU 至少有两个特权级，分别用来运行内核和用户程序。为什么要这样设计？(4’)

2. 用户程序执行的过程会触发同步异常或被异步中断打断，此时 CPU 特权级会切换到内核态。在 ChCore 中，用户程序的上下文被保存在内核栈中。这里的内核栈是每个线程独立的还是所有线程共享的？为什么？(4’)

3. Linux 把中断处理过程分为了上半部和下半部。
   a) 这样设计有什么优点？(4’)
   b) 考虑一次中断发生后，以下操作哪些应该放在上半部，哪些应该放在下半部？请简要解释原因。(4’)
      1) 读设备寄存器检查设备状态
      2) 调用一些可能导致睡眠的函数
      3) 处理设备通过 DMA 发送的数据
      4) 解除中断屏蔽

4. 考虑一个处理网络请求的场景：（左为用户态程序，右为简化的内核态的网卡驱动）

```
// usermode
…
while (working) {
    read(sockfd, request, len);
    handle(request);
}
…
```

```
// kernel
void do_read() {
…
    if (packet_buffer is empty) {
        wait_event(queue);
    }
    raw_packet = get(packet_buffer);
    packet = protocol_stack(raw_packet);
    copy_to_user(userbuf, packet, len);
…
}
void handle_irq_netdev() {
    put(packet_buffer, dma_buffer);
    wake_up(queue);
}
```

面对高速网卡，上述代码难以达到最高性能，此时的性能瓶颈在什么地方？应该如何改进？(4’)

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号	一	二	三	四
得分				
批阅人(流水阅卷教师签名处)				

5. 请分别简要描述读取磁盘时，read()系统调用在**宏内核**和**微内核**的执行流程。(4’)

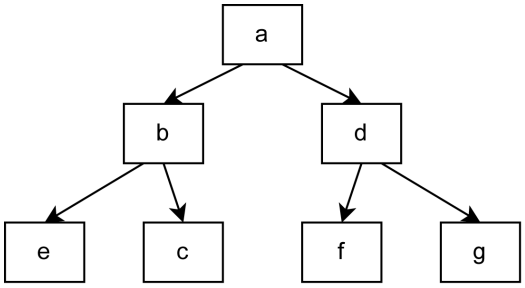
6. Exception 会带来哪些额外的开销？试设计一个 Exception-Less 的系统调用方案，并与传统方案作比较（**提示：利用多核**）。(4’)

二、 文件系统(28')

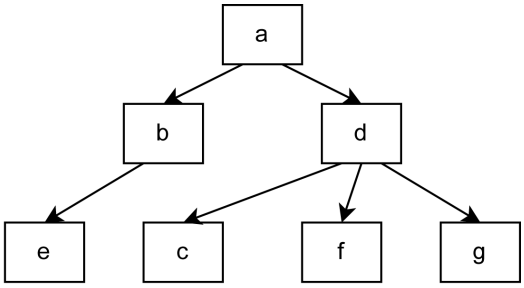
1. 软链接和硬链接是在文件系统中创建引用的方式。请分别说明是否可以通过软链接或硬链接的方式在本机目录下（例如“/home/student”）创建一个到外部存储（例如 U 盘）内文件的引用。(4’)

2. Ordered Mode 和 Journaling Mode 分别是 Ext4 文件系统的两个运行模式。
   a) 请简述这两种模式的流程。(4’)
   b) 请从性能和崩溃一致性两个方面说明两种模式的区别。(4’)

3. Copy-on-write 是一种在 NVM 文件系统上常用的保证操作原子性的方法，请描述 Copy-on-write 原子性地完成 rename /a/b/c /a/d/c 操作的过程，将下图目录树(1)转化为(2)。(4’)



(1) Original File Tree



(2) Final File Tree

4. 请阅读如下代码，并回答问题：

```
int main() {
    int fd;
    char *addr;
    char buf[256];

    memset(buf, 0, 256);

    fd = open("./b.txt", O_RDWR);
    addr = mmap(NULL, 10, PROT_WRITE, MAP_SHARED, fd, 0);
    printf("Before Assign:\t%s\n", (char *)addr);
    addr[5] = 'Y';
    printf("After Assign:\t%s\n", (char *)addr);
    read(fd, buf, 10);
    printf("Read From fd:\t%s\n", (char *)buf);

    close(fd);
}
```

输出：

```
Before Assign:  1234567890
After Assign:   12345Y7890
Read From fd:   12345Y7890
```

a) 小雪在 ubuntu 20.04 中用 c 语言编写了如上代码并获得了如上的输出，她发现使用 mmap 将文件映射成内存后修改，再立刻使用传统的 read 接口读取 fd 中的数据，仍然可以获得修改后的最新数据。而她马上想到只是修改虚拟内存中的数据并不会触发系统调用写磁盘文件，你能帮她解释为什么 read 函数仍然可以读到最新的数据吗？(4’)

b) 小雪随后在 ubuntu 20.04 中，尝试读取一个非常大的文件，她发现用 read 系统调用把文件读入内存需要很长的时间，但换一种方法，调用 mmap 函数时函数立刻就返回并且随后马上可以在内存正常读写数据了，造成这种现象的原因是什么？(4’)

c) 小雪尝试在 ChCore 微内核中实现 mmap 映射文件的功能，她查阅资料发现，宏内核中文件系统作为内核模块在内核态工作，而微内核中文件系统作为一个用户程序运行在系统中。显然，宏内核的 mmap 设计在微内核中并不适用，请结合微内核设计的特点，列出一条微内核中实现 mmap 会遇到的问题并解释原因。(4’)

三、 同步(24')

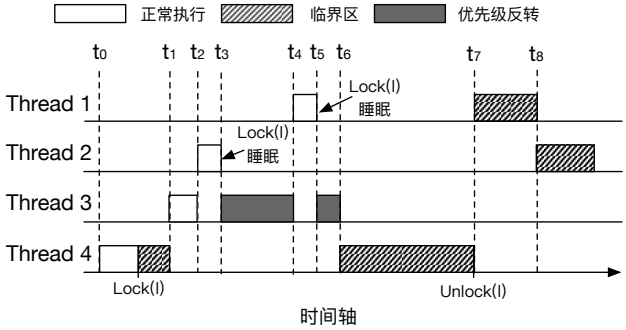
1. 互斥锁是为了方便并行编程而抽象出来的一种同步原语。Ticket Spinlock、MCS Lock 是常见的互斥锁实现。请简要说明他们各自的**特点**和**适用场景**。(4’)

2. 并行和并发程序中通常需要进行线程间的同步，请举出**两个**需要同步的例子。(4’)

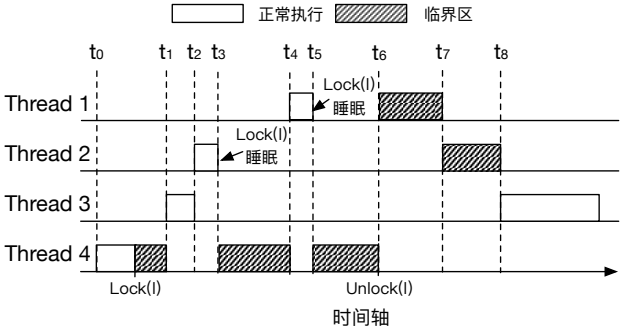
3. 下面的代码是 ticket spinlock 的简单实现，我们发现 lock 结构中的变量都注明了 volatile 关键字，volatile 关键字会告诉编译器不要对相关的变量进行优化。当去掉 volatile 关键字之后，我们发现，有时候本应该轮到某个线程拿到锁，即 lock->owner 已经等于该线程拿到的 ticket，但这个线程仍然一直停留在在第 8 行的循环，无法跳出，导致整个程序无法继续进行。**请从编译优化的角度，根据代码具体分析该现象出现的原因，并总结出 volatile 关键字的使用场景**。(4’)

```
1   struct lock {
2     volatile int owner;
3     volatile int next;
4   };
5
6   void lock(struct lock *lock) {
7     int my_ticket = atomic_FAA(&lock->next, 1);
8     while(lock->owner != my_ticket) {}
9     barrier();
10  }
11
12  void unlock() {
13    barrier();
14    lock->owner++;
15  }
```

4. 优先级反转是同步可能会导致的一类问题，它是由于同步导致线程执行顺序违反线预设优先级的问题。假设一个 CPU 核心上有 Thread1、Thread2、Thread3 和 Thread4 四个线程，其优先级分别为 1、2、3、4（数字越小，优先级越大），其中 Thread1、Thread2、Thread4 竞争同一把锁，Thread3 不申请锁。下图 a 展现了出现优先级反转的一种场景，图 b 展现了使用<u>优先级继承协议</u>来避免优先级反转问题的例子。请填写完成下面的表格，判断在使用优先级继承协议后，**图 b** 中 **Thread 4** 的在下表各时刻的优先级。(4’)



图a：优先级反转



图b：优先级继承协议

时刻	Thread4 优先级
t0	
t1	
t2	
t5	

5. 不同架构的处理器使用不同的内存模型，例如 ARM 架构使用弱一致性模型（弱一致性模型不保证任何对不同的地址的读写操作顺序），Intel x86 架构使用 TSO（Total Store Ordering）模型（TSO 模型保证对不同地址的读-读、读-写、写-写顺序，不保证写-读顺序）。下面的代码是并发哈希表的一种实现，其仅仅用锁保护了 put 操作，我们发现上述实现在某些处理器上总能正确运行，在另一些处理器上，lookup 操作有时会返回错误的值或者找不到已经 put 的值。

   a) 在**不考虑编译乱序**的情况下，请从**内存一致性的角度**出发，分析两种处理器架构下出现不同现象的原因。(4’)

   b) 请分别用添加锁（lock()和 unlock()）和添加内存屏障（barrier()）的方法修改下述代码，使得修改后的代码能在不同架构的处理器上都能够正确运行，**并使得不同的读者可以同时查询同一个 bucket 中的元素**。(4’)

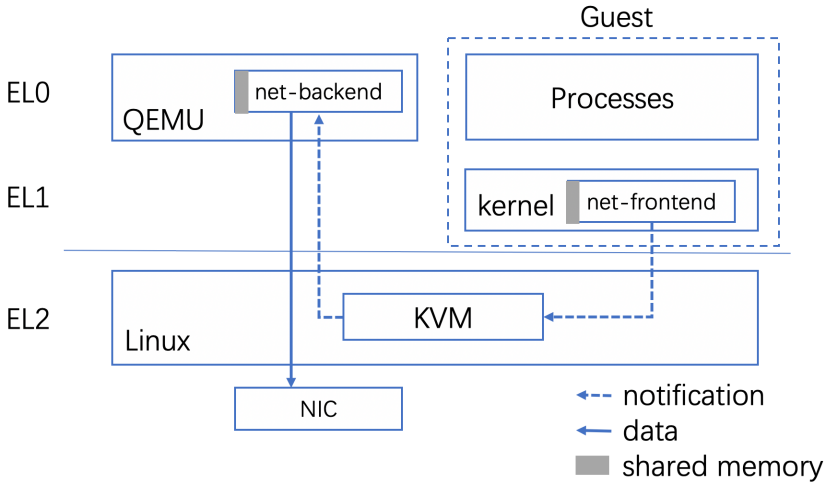
```
struct entry {
  int key;
  int value;
  struct entry *next;
};

struct entry *table[N_BUCKET];
struct lock *bucket_locks[N_BUCKET];

void insert(int key, int value, struct entry **p) {
  struct entry *e = malloc(sizeof(struct entry));
  e->key = key;
  e->value = value;
  e->next = *p;
  *p = e;
}

void put(int key, int value) {
  lock(&bucket_locks[key % N_BUCKET]);

  struct entry **p = &table[key % N_BUCKET];
  insert(key, value, p);

  unlock(&bucket_locks[key % N_BUCKET]);
}

struct entry* lookup(int key) {
  struct entry *e = table[key % N_BUCKET];
  for (; e != NULL; e = e->next) {
    if(e->key == key) break;
  }
  return e;
}
```

四、 虚拟化(20')

1. 虚拟机监控器分为 Type-1 和 Type-2 两种，它们的设计各有什么优点？你能各举出两种类型虚拟机监控器的一个例子吗？(4’)

2. DMA: 是一种快速传送数据的机制。DMA 传输将数据从一个地址空间复制到另外一个地址空间。CPU 负责初始化这个传输动作，而传输动作本身是由 DMA 控制器来实行和完成。最初的 DMA 地址是物理地址，但这会带来安全问题，因此出现了 IOMMU 硬件。
   a) 使用 IOMMU 之后，设备看到的还是物理地址吗？设备访问内存的过程是怎样的？（4’）
   b) 在使用 IOMMU 之前，设备访问主机内存会有什么安全问题？IOMMU 是如何解决它的？除了能够解决安全问题，IOMMU 提供的抽象还有什么别的好处？(4’)

3. IO 虚拟化



   a) 上图展示了 guest 发送网络包时，data 和 notification 的传输路径(guest 和 qemu 通过 share memory 共享数据)，这种设计发送网络包的延迟很高，你认为影响它性能的原因有哪些？**（至少提出两个）**(4’)
   b) 为了解决 3.a 的性能问题，有人提出可以将 net-backend 放在 host kernel 中(backend 和 frontend 仍通过 share memory 共享数据)，它从哪两个方面解决了 3.a 的性能问题？(4’)