上 海 交 通 大 学 试 卷（A 卷）

（ 2019 至 2020 学年 第 2 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称_____操作系统_____成绩 _____

一、 内存(10’)

1. 下面哪些场景中需要 TLBflush：
   a) 程序触发了系统调用下陷到 linux 内核
   b) Chcore 中程序通过 ipc 调用文件系统提供的服务
   c) 外核中程序调用 libOS 提供的功能接口
   如果需要 TLBflush，那么总共需要几次？请分别列出。（提示：考虑 call 和 return）(3’)

   答案：a) 两次 b) 四次 如果考虑 ASID 等情况说明之后也算正确

2. 小明运行了大矩阵计算程序，但发现其性能并不理想，小明通过 perf 对程序运行时的状态进行了分析，下面是得到的分析结果，请问你觉得性能不理想的原因是什么？应该如何改进？(3’)

```
Performance counter stats for './matrix':

13,596          cache-misses
52,749,489      dTLB-loads
9,819,564       dTLB-load-misses
215             iTLB-loads
7,790           iTLB-load-misses
33.068447       task-clock (msec)         #    0.996 CPUs utilized
0               context-switches          #    0.000 K/sec
145,495,064     cycles                    #    4.400 GHz
110,889,441     instructions              #    0.76  insn per cycle
12,163,513      branches                  #  367.828 M/sec
8,234           branch-misses             #    0.07% of all branches

0.036513072 seconds time elapsed
```

   答案：TLB miss 过多，使用大页

3. 右下图是矩阵乘法的代码，已知 a 地址为 0x10f00，b 地址为 0x20000，ans 地址为 0x30000，CPU 计算一条乘法指令需要 3 个 cycle，加法需要 1 个 cycle （不考虑分支跳转指令与内存读取指令的开销）。工作集 W(2^{14}，2^{8})表示在时钟周期（2^{14}-2^{8}，2^{14}）内存页的集合，为了跟踪工作集，内核采用下表的结构记录内存页的状态，其中循环开始前工作集为空，工作集的大小为 6 页，时钟周期（0，2^{8}）对应的 Tick 为 0，请问工作集 W(2^{14}，2^{8})中的页为哪些，并在下表中填写对应的内核记录的页的状态。（如果内存页没有被访问可以不用

填写）(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)									

Addr	Age	Access bit
10000	0	1
30000	0	1
20000	0	1
21000	0	1
22000	0	1
23000	0	1

```
void mul(int ans[][], int a[][], int b[][]){
    for(int i=0; i<64; i++){
        for(int j=0; j<64; j++){
            for(int k=0; k<64; k++){
                ans[i][j]+=a[i][k]*b[k][j];
            }}}}
```

二、 进程(18’)

小明希望实现一个有键值存储功能的网络服务器，负责处理用户发起的数据存取请求。下面罗列了他的设计实现：

- 所有的数据（键值对）**存储在内存中；**
- 当服务器收到一个请求后，它的主进程会调用 **fork()** 创建新的进程处理用户的请求，在请求处理完后，对应的进程被回收；
- 服务器采用 **Round Robin 策略调度**任务。

1. 在运行服务器后，小明发现了一个 BUG：虽然用户对数据的写操作能够成功返回，但是后续的读操作无法读到之前写操作的更新。并且他发现这个 BUG 是由于 **fork()**所导致的，请问为何 **fork()**会引起这个 BUG？(4’)
   Fork 会生成新的进程造成内存隔离（2’）
   多个请求在不同进程无法观察到彼此的修改（2’）

2. 为了修改上述 BUG，小明修改了服务器设计。当一个请求到来时，会使用 pthread_create 接口创建一个新的**内核态线程**处理，并在请求完成后回收该线程。接着小明开始对服务器测试，他发现：
   1） 大量 CPU 时间被用在了内核中，而不是在用户态逻辑中；
   2） 随着用户请求数量的增加，花费在内核中的时间进一步上升。
   请问造成上述现象的原因可能是哪些？(4’)可以如何完善当前的服务器设计以解决上述现象？(2’)
   （请结合**内核态线程**、**调度**、**服务器设计**等角度分析，回答**至少两点**）
   使用内核态线程，创建、销毁开销大；采用 round robin 调度，时间片过小，调度开销大；

服务器为每个请求都创建新的线程（4’，回答两点即可，其他合理答案也给分）
采用线程池；增大 round robin 时间片；（2’）

3. 为了进一步减少内核态线程的开销，小明决定更改用户态线程和内核态线程的映射关系，从**一对一，多对一，多对多**线程模型中选取一个合理的模型应用在服务器上，请帮助小明作出选择，并说明原因。(4’)
多对多（2’）
多个用户态线程可以在与 CPU 数量相同的内核态线程中调度执行，保证并发性。
每个内核态线程管理自己的用户态线程，降低调度开销（2’）

4. 小明将服务器部署在有两个 NUMA 节点的服务器上并测试。他发现使用的 NUMA 节点从一个扩展到两个后，对应的请求处理**时延增加**了，请问这可能是由于什么原因导致的？(2’)
可以如何避免这一问题？(2’)
由于访问相同数据的请求在不同 NUMA 上被处理（2’）
对数据划分，访问相同数据的请求只能在某一个指定 NUMA 完成（2’）

三、 调度 (10’)

小明拥有一台自己的服务器，并且设置服务器上的调度策略为多优先级队列。

1. 实现多优先级队列时，小明设计了两个优先级：高优先级和低优先级。对于两类任务：命令行交互式应用与科学计算应用，小明应该如何设置两者的优先级（2’）？
命令行交互式应用：与用户交互，高优先级；
科学计算应用：后台应用，低优先级

2. 进一步实现多优先级队列时，小明添加了一个优先级（即目前共有三个优先级）。为了测试，小明创建了 A、B、C 三个线程，代码如下。三个线程执行在同一个 CPU 上，优先级为 A>B>C。然而，在实际执行中，小明却发现在一些情况下，线程 B 执行优先级“仿佛高于”线程 A，即线程 B 相比于线程 A 占用了更多的 CPU 资源（注：不考虑拿锁放锁占用的 CPU 资源）。试分析其中的原因，并在操作系统层面提出一种解决办法(4’)。

// 线程 A	// 线程 B	// 线程 C	// 全局变量

while(true) {
  lock(A);
  lock(B);
  a += b;
  unlock(B);
  unlock(A);
  sleep(1);
}
	while(true) {
	  lock(B);
	  b += 1;
	  unlock(B);
	  sleep(1);
	}
		while(true) {
		  lock(A);
		  a += 1;
		  unlock(A);
		  sleep(1);
		}
			int a, b;
			lock A, B;

原因：若线程 C 先拿到锁 A，则线程 A 需要等待线程 C 放锁才可执行；于此同时，线程 B 由于优先级较高，会获得比线程 C 更高的优先级，占用更多的 CPU 资源。锁的调度同优先级的调度不匹配，造成了优先级反转的问题（2’）

解决方法：优先级继承（2’）

3. 小明决定共享自己的服务器给其他同学使用，为了更好地分配服务器资源，小明为调度器添加了权重的支持，使用 Lottery Scheduling 的方式进行有权重的调度。其核心代码大致如下：

```
node_t *current = head;
// 获取随机数，total_tickets 为所有 ticket 的和
int winner = random() % total_tickets;
while (current) {
  counter = counter + current->tickets;
  if (counter > winner)
    break; // 找到了合适的线程
  current = current->next;
}
// 调度 current 线程
```

然而，在具体使用中，小明发现，随着线程数的增加，上下文切换的开销变得越来越大。请分析原因，并提出一种可能的解决方法(4’)。

原因：每次调度中选择合适的线程需以 O(n)的复杂度遍历链表，线程较多是开销过大（2’）

解决方法：可以采用一些带索引的数据结构（如红黑树）替换链表（或使用 Stride Scheduling）（2’）

四、 进程间通信 (12’)

1. 下面是课堂中提到过的四种进程间通信方式：
   - 使用阻塞的消息传递进行进程间的直接通讯；
   - 使用非阻塞的消息传递进行进程间的直接通讯；
   - 使用“信箱”的方式进行进程间进行间接通讯；
   - 通过轮询共享内存的方式进行进程间的通讯。

   针对下面四个应用场景，请为每个场景从上述进程间通信方式中挑选出最为合适的一种，并简述理由（注：每种进程间通信方式仅可被使用一次）。（2’ * 4）

a) 电商网站中的反向代理进程希望通过进程间通信的方式将收到的用户请求转发给一系列服务进程，使得某服务进程空闲后即可处理该请求。

b) 电商网站中的服务进程希望通过进程间通信的方式从锁服务（Lock Service）进程中获取一把锁，从而进入临界区（Critical Section）执行商品购买逻辑。

c) 电商网站中的服务进程希望通过进程间通信的方式，将包含用户请求执行结果的网络包通过用户态网络驱动服务进程，以尽可能低的时延发送出去。

d) 电商网站中的服务进程希望通过进程间通信的方式将一条用户购买记录发送给后台推荐分析进程。

各 2 分，其他合理答案也可酌情给分

a) 使用信箱，因为是一对多通信，并且不指定接受者

b) 使用阻塞的消息传递，因为进入临界区需要确认拿锁成功

c) 使用轮询，因为要求低时延，需要在存在 IPC 请求时尽快的发现并进行处理

d) 使用非阻塞的消息传递，因为推荐分析进程对业务核心逻辑无影响，可以异步执行

2. 小明希望能够在 ChCore 中实现了一套类似于课程中学习的管道的进程间通信机制，部分内核代码如下。在测试时，小明发现经常会发生消息发送者和接受者均进入等待状态的 BUG。请结合代码，分析发生该 BUG 的原因，并简述应如何修复。（4’）

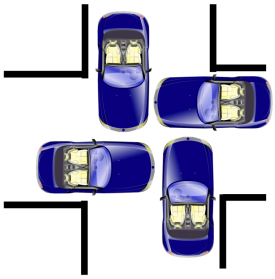
```
// 发送端代码
void send_chan(msg_channel *c, char *buf, int size) {
  lock(c->lock);
  for (int i=0; i<size; i++) {
    if (c->write_pos==c->read_pos+4096) {
      unlock(c->lock);
      // 唤醒所有等待在信道 c 上的读者
      wakeup_reader(c);
      // 令当前线程等待在信道 c 上（即在被其他线程唤醒之
      // 前，当前线程不会被操作系统调度到）
      set_thread_wait(c);
      // 切换至其他线程执行
      sched();
      lock(c->lock);
    }
    c->data[c->write_pos%4096] = buf[i];
    c->write_pos++;
  }
  // 写入完成，唤醒可能被阻塞的读者
  wakeup_reader(c);
  unlock(c->lock);
}
```

```
struct msg_channel {
  struct lock *lock;
  char data[4096];
  int read_pos;
  int write_pos;
}
// 接收端代码
int recv_chan(msg_channel *c, char *buf, int max_size) {
  lock(c->lock);
  // 令当前线程等待在信道 c 上，直到其中存在可读数据
   // 注：该该函数内部包含判断有无数据、放锁、进入等待状态、
   // 重新拿锁的循环，不存在 bug
  wait_until_has_data(c, c->lock);
  for (int i=0; i<max_size; i++) {
    if (c->read_pos==c->write_pos) break;
    buf[i]=c->data[c->read_pos%4096];
    c->read_pos++;
  }
  // 读取完成，唤醒可能被阻塞的写者
  wakeup_writer(c);
  unlock(c->lock);
  return i;
}
```

原因：sender 在唤醒 reader 并进入 wait 状态之前就放锁了。若 Sender 放锁并唤醒 Reader 后且进入 wait 状态之前，Reader 完成了所有操作并结束了 wakeup_write 的调用，则后续 Sender 进入 Wait 状态后不存在新的 Receiver 对其进行唤醒；而 Receiver 由于没有数据可读，也会进入 Wait 状态。（2’）
修复：将 Sender 端的 unlock 移动到 set_thread_wait 后面。（2’）

五、 同步 (10’)

1. 假设现在有锁 A、B、C 和 D 以及线程 1，2，3 和 4。请描述一个与“十字路口困境”相似的，同时涉及 4 把锁死锁的例子(2’)请提出一个检测或预防死锁的方案(2’)。



线程 1：Lock A; Unlock B;线程 2：Lock B; Unlock C;
线程 3：Lock C; Unlock D;线程 4：Lock D; Unlock A;（2’）

所有线程按顺序拿锁、银行家算法等（2'，其它可行方案也给分）

2. 排号锁（Ticket Lock）是能够保证拿锁公平性的自旋锁，每个申请锁的核（core）本地维护一个自己持有的序号 myTicket，所有核共享个全局变量 owner。当某个核的 myTicket 与 owner 相等时，视为这个核持有了锁。当持有锁的核希望放锁时，它将 owner 原子地加一，视为将锁释放。

1） 如果多个核同时竞争同一个排号锁，会产生严重的缓存行竞争。请问是排号锁的什么设计导致了严重的缓存行竞争？(2')

针对 owner 变量的读取会产生严重的冲突（2'）

2） 小明希望对排号锁因为缓存行竞争而带来的性能影响有一个更加直观的认识，他首先作了如下假设：

- 缓存行只能由**最后写了该缓存行的核**迁移给另一个核，并且**同一时间只能迁移给一个核，不能广播**
- 不考虑**硬件中断、调度**的影响，不考虑**临界区的执行时间**，仅考虑缓存行迁移的开销
- 假设当前有 N 个核在同时竞争同一把排号锁，并且它们释放锁后不会再一次申请

请具体描述 N 个核都申请到排号锁的耗时的最坏情况。(4')

每当锁传递给下一个应该申请到该锁的核时，该核都是最后一个读取 owner 所在缓存行的。所以最坏情况下总共会有 N+(N-1)+…+1 次缓存行迁移。（4'，意思相近即可）

六、 文件系统 (20')

1. 日志（Journaling）是一种保证崩溃一致性的方式。Ext4 的日志系统 JBD2 有三种日志模式可供选择——日志模式（journal），顺序模式（ordered）以及写回模式（writeback）：

a）日志模式：所有数据和元数据都要被写入日志中。

b） 顺序模式：只有元数据需要被写入日志，并且保证在元数据在日志中被标记提交之前，数据已经被写到磁盘中。

c）写回模式：只有元数据需要被写入日志，没有其他顺序保证。

3） 如果我们需要保证从 U 盘中将文件拷贝至磁盘这一操作不会在崩溃后导致看到文件但是看不到文件内容的情况，同时要保证性能，应该选择哪种日志模式？为什么？（4'）

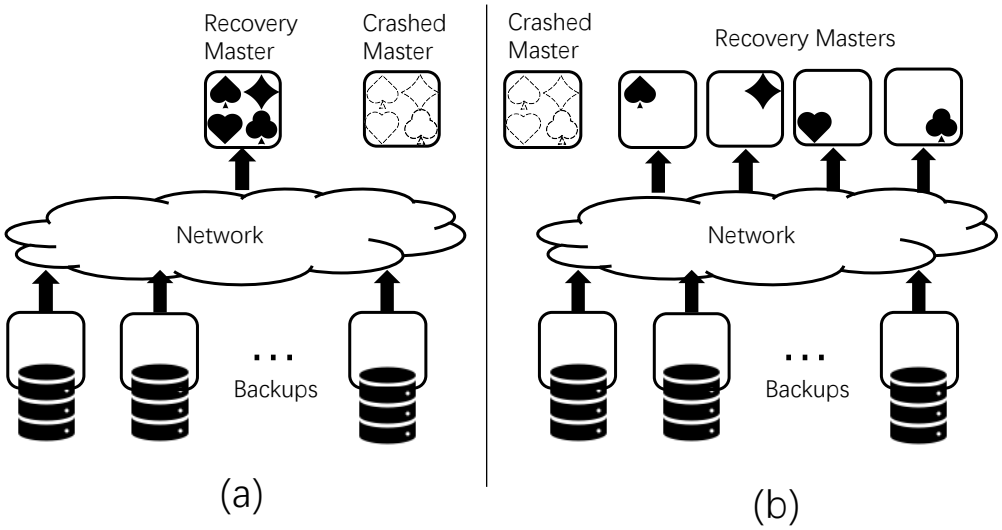
b)（2'）。元数据在数据之后写，保证有元数据一定能看到数据。只写少量的元数据。（2'）

4） 新型的瓦式磁盘（SMR）是通过重叠物理磁道以提升磁盘密度而设计的。对于 SMR 而言，记录日志会增加一个 log-structured 的日志缓存区域，这是针对 SMR 的什么性能特性设计的？为什么？（4'）

随机写慢(2')，顺序写快(2')（其他只要提到由结构带来的性能问题均给分）

5） 日志模式虽然可以同时保证数据和元数据的安全性。写时复制（copy-on-write）技术也可以保证数据和元数据。然而，这两种技术在不同场景下的性能表现不同。就修改的**数据量大小**而言，请分析这两种技术分别适合什么情况，并说明理由。（4’）
数据量大：写时复制，日志模式需要写两遍数据（2’）
数据量小：日志模式，写时复制需要拷贝完整的数据块（2’）
（言之有理，均酌情给分）

2. 除了在单机上进行容错恢复的设计，也可以进行分布式的容错恢复。RAMCloud是一个基于内存的存储系统，通过日志结构（log-structured）的方式提供快速恢复：RAMCloud无需在内存中存放多个备份，而是将持久化日志发送到远端机器，远端机器再将这些日志异步地刷到磁盘中。这些机器会有备用电池保证至少内存中的数据可以被刷到磁盘中。一些硬件参数如下：
每台机器装有一个带宽为100MB/s（1GB=1000MB）的磁盘，并且使用10Gbps（10Gbps = 1.25GBps）的网卡连接。并且假设日志段可以从多个磁盘上同时被读到。



1) 一个简单的镜像备份如图（a）所示，其中每个 master 在每个备份机（backup）上都存放所有日志段的完整备份。假设有 3 个备份机，请问并行地读总共 80GB 的数据需要多少**磁盘 IO 时间**？如果这个时间需要在 1 秒以内，至少要多个备份机？（2’+2’）
Read 80GB from disk: 80 * 1000 / **100** / 3 = 266.67 sec（2’）
# of disks = 80 * 1000 / 1/ 100 = 800（2’）（步骤正确，最后计算错误也给分）
2) 为了进一步提高性能，RAMCloud 使用多个 recovery master，如图（b）所示。在崩溃 master 上的数据将会被切分多个等大的数据分区，每个分区可以在不同的 recovery master 上恢复。假设恢复可以**完全并行**，有 4 个 recovery master 和 100 个备份机，需要多久时间（包括磁盘 IO 和网络 IO）可以恢复 80GB 的数据？（4’）
Read 80GB from disk: 80 * 1000 / 100 / 100 = 8 sec（1’）
Network time = 80 / 1.25 / 4 = 16 sec（1’）
Total time = max{disk, network} = 16 sec（2’）（给出公式就给分）

七、 虚拟化(20')

1. （a）在 Azure 等云中，云厂商为了提供更高性能的云服务，会使用 Xen 或者类 Xen 的虚拟机（hyper-V）；（b）助教在发布 lab 的时候使用 VMware workstation 虚拟机。请简要分析这两类的虚拟化方案有什么不同，以及（a）和（b）中采用不同虚拟化方案的原因？(3')

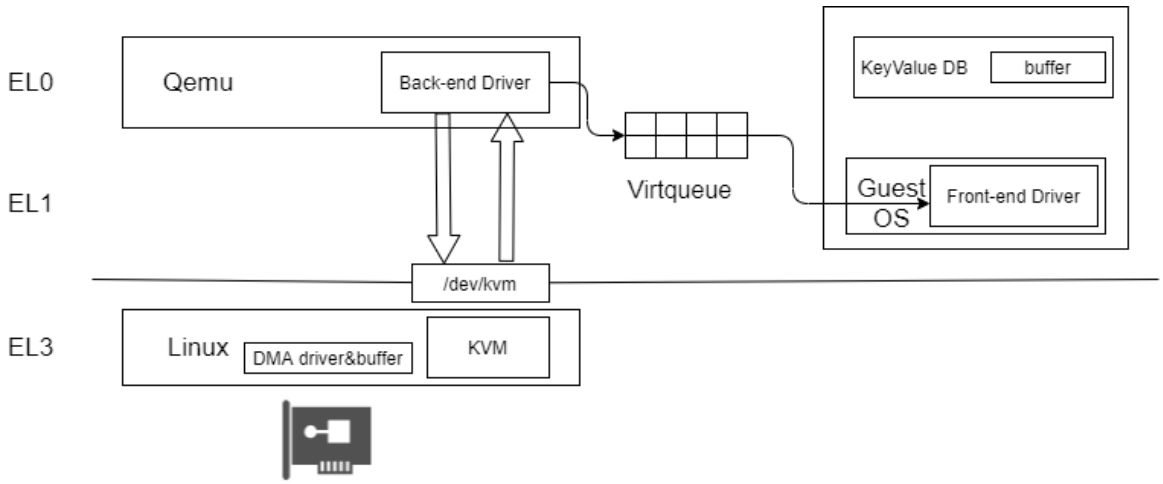
   答案：Type-1 虚拟化和 type-2 虚拟化，前者拥有更好的性能；后者已于实现与安装，能够服用主机系统的大部分功能

2. 为了加速虚拟机中两阶段页表的翻译，(a)在 TLB 中分别记录 GVA->GPA，以及 GPA->HPA 的条目；（b）在 TLB 中直接记录 GVA->HPA 的条目。请问这两种方式各有什么优劣，请做简要的分析。(3')

   答案：(a) 能够 guest pt 和 ept 使用不同的粒度的页

   (b) TLB hit 时候拥有较好的性能，TLB miss 的时候需要访问 24 次内存

3. 小明想要测试在 qemu 虚拟机中运行 keyValue 数据库的性能，他写了一个简单的测试脚本，读取数据库中所有的内容，但是测试发现数据库的吞吐量下降了几个数量级。请用你上课所学的知识，帮助小明优化虚拟机中 KeyValue 数据库的性能。（14'）
   a) 小明通过查找资料发现，在启动 qemu 的时候加上 –enable-kvm(使用内核 kvm 模块) -device virtio-serial-pci(使用 virtio 驱动)参数之后，虚拟机中 keyValue 数据库的吞吐量得到了大幅度的提升，请回答其中的原因并分析。(3')
      答案：qemu 使用二进制翻译的技术，而 kvm 使用了 CPU 虚拟化的技术；virtio 采用了半虚拟化技术加速 IO。
   b) 为了测试结果尽可能严谨，小明运行了测试脚本多次。他发现，第一次运行的时间多于之后的运行时间。经过分析小明发现第一次运行时候会产生更多虚拟机退出事件。请解释其原因。(3')
      答案：第一次运行的时候会遇到较多 ept page fault，需要退出到 vmm 中进行处理。
   c) 小明解决了（a)、(b）中遇到的问题，对虚拟机中运行 KeyValue 数据库重新做了测试，虽然优化后数据库的性能得到了大幅度的提升，但是相较于数据库原始的性能，还有几倍的开销，并且和读取数据大小呈正相关。下图展示当前虚拟机和 KVM 的架构，运行在虚拟机中 keyValue 数据库想要读磁盘中数据，至少要经过几次拷贝，并给出分析？在不更改硬件情况下你能想到优化的方式吗？请结合下图作简要分析。(4')



答案：需要拷贝三次，DMA->DMA buffer；DMA buffer->Virtqueue；Virtqueue->KV buffer。可以将 Back end driver 放在 EL3 中或者将 KV 可以直接读取 front end driver 中的 Virtqueue.

d) 小明更换了支持 SR-IOV 磁盘和支持 ITS（Interrupt Translation Service）的 CPU，请问这些新的硬件技术如何提升虚拟机中 KeyValue 数据库的性能？（和（c）中使用的方式作比较）。(4’)

答案：SR-IOV 可是实现硬件直通，ITS 不需要打断虚拟机执行，能够直接将中断分派给 VM