2018 年春季学期《操作系统原理与设计》期末试题

一、填空题 10 * 1% = 10%

(1)操作系统的4大特性：**并发性，共享性，虚拟性，异步性**。

(2)进程调用信号量的P操作失败让该进程从(**运行**)状态转变到(**阻塞**)状态，并加入该信号量的(**等待队列**)

(3)队列逻辑地址0x34567的项是页表中的第( )项;若物理页帧号为8，求物理地址( );访问该地 址需访存( )次(条件: 无TLB,页大小为4kb

二、概念解释题 4 * 5% = 20%

(1)RAID0

RAID0位具有块分条但没有冗余（如镜像或校验）的磁盘阵列

(2)饿死现象

饿死又称为“无限等待”，当某一进程所需要的资源正在被其它进程所占用，当资源被释放时，由于该进程优先级较低，或者因为调度算法（如LIFO）的原因，因此资源又被其它优先级更高的进程占用，如果这些高优先级的进程周期性地占用资源，就可能使得该低优先级进程“饿死”，永远在等待资源，但又永远得不到。

(3)局部性原理

局部性原理包括2个，分别为时间局部性和空间局部性，时间局部性是指如果程序中的某条指令（也就是内存中的某一数据项）被访问，那么不久之后该数据项可能被再次访问。空间局部性是指，一旦访问了某个存储单元，那么其附近的存储单元也可能在不久之后被访问。局部性模型指出，随着进程执行，它从一个局部移向另一个局部。

(4)硬链接

A hard link is a directory entry pointing to an existing file. Conceptually speaking, this creates a file with two pathnames. They point to the same inode (no new file content) .

(5)页抖动

如果进程没有需要支持活动使用页面的帧数，那么它将产生缺页错误并置换某个页 面，但是，由于它的所有页面都在使用中，所以必须立即置换需要再次使用的页面。 这样很快又会产生缺页错误，这种高度的页面调度活动称为“抖动thrashing”，表示一 个进程的调页时间多于它的执行时间。

书中P271页对抖动的产生原因是这样阐述的:CPU调度程序再CPU利用率降低时会增 加多道程度。新进程试图从其他运行进程中获取帧来启动，而其他运行进程也需要这 些帧，从而导致了更多的缺页错误和更长的调页设备队列，因此CPU利用率下降，而 CPU调度程序将再次试图增加多道程度，(有点类似于死锁)，这样就出现了抖动， 系统吞吐量骤降，进程都在忙于调页。

三、简答题 4 * 5% = 20%

(1)给一段程序，fork后父子进程分别修改全局变量，问输出结果

(2)RR调度需要哪些调度相关信息，系统如何维护这些信息

轮转时间（quantum），各个任务的到达时间，CPU请求时间，CPU剩余时间

(3)系统调用与库函数调用的区别

一：系统调用和库函数调用的区别：

1：系统调用是最底层的应用，是面向硬件的。而库函数的调用是面向开发的，相当于应用程序的API(即预先定义好的函数)接口；

2：各个操作系统的系统调用是不同的，因此系统调用一般是没有跨操作系统的可移植性，而库函数的移植性良好(c库在Windows和Linux环境下都可以操作)；

3：库函数属于过程调用，调用开销小；系统调用需要在用户空间和内核上下文环境切换，开销较大；

4：库函数在罪中执行时需要调用系统调用，但封装了系统调用的细节；系统调用调用的是系统内核的服务。

5: 系统调用在内核态执行，它的运行时间属于“**系统时间**”，函数调用在用户态执行，它的运行时间属于“**用户时间**”。

(4)Linux文件权限755、644的含义

755——文件所有者读写执行全部权限，文件用户组读执行权限，其他用户，读权限

644——文件所有者读写权限，文件用户组读权限，其他用户读权限

四、解答题 10+20+20=50%

(1)给了TLB命中率，缺页率，page fault的两种处理时间等等信息，计算平均有效访存时间

(2)比较SSTF, SCAN, C-SCAN的优点、不足，(具体好像是两两对比，记不太清了) 给出后 两者存在的相同的不足之处的解决方案

SSTF——最短寻道时间调度，该调度算法选择距离磁头最近的位置，相对fcfs方法，它的磁头移动距离较短，但它可能会导致无限等待。

SCAN——从一端扫描到一端，再回头扫描到另一端，优点是可以公平的响应每一个位置的请求。但是当扫描到某一端时，另一端需要等很久才能响应到。

C-SCAN——循环扫描就是为了解决SCAN的问题，它能够提供更均匀的等待时间，但是即使一些没有请求的地方也会花时间去扫描。

(3)详细解释 ppt 上哲学家用餐问题前两种解决方案的不足、默写 final-solution

第一种方案是将每个筷子信号量，作为如果能拿起第二个筷子就吃，否则就等待，这种方案可能会出现死锁。

第二种方法是，如果发现第二个筷子被占用，就放下第一个筷子并等待，直到两个筷子都可以用，这种方案可能会导致每个哲学家都在拿起放下筷子，但没有人吃过。

```
#define N 5     //五个哲学家
#define LEFT  ((i-1) % N) //左边一个
#define RIGHT  ((i+1) % N) //右边一个
int state[N];   //每个哲学家有eating hungry thinking三种状态
semaphore mutex = 1; //保护stateN
semaphore s[N]; //记录第i个哲学家是否可以吃（筷子状态），目的是叫醒正在sleep的哲学家

void take(int i) {
  if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=   EATING) {
    state[i] = EATING;
    up(&s[i]);  //Wake up the one who can eat!
  }
}

//Section entry
  void take(int i) {
      down(&mutex);
      state[i] = HUNGRY;
      test(i);
      up(&mutex);
      down(&s[i]);
}
//Section exit
  void put(int i) {
      down(&mutex);
      state[i] = THINKING;
      test(LEFT);
      test(RIGHT);
      up(&mutex);
}

void philosopher(int i) {
```

```
  think();
  take(i);
  eat();
  put(i);
}
```