清华大学本科生考试试题专用纸

考试课程：**操作系统（A 卷）**　　时间：2016 年 05 月 25 日下午 3:20~5:20

系别：_____ 班级：_____ 学号：_____ 姓名：

答卷注意事项：
1. 答题前，请先在试题纸和答卷本上写明 A 卷或 B 卷、系别、班级、学号和姓名。
2. 在答卷本上答题时，要写明题号，不必抄题。
3. 答题时，要书写清楚和整洁。
4. 请注意回答所有试题。本试卷有 29 个题目，共 5 页。
5. 考试完毕，必须将试题纸和答卷本一起交回。

一、判断题（20 分）

1. [ ]在进程控制块数据结构中，必须为进程建立内核栈结构，确保进程可以得到操作系统的可靠服务和管理等支持。
2. [ ]在进程切换过程中，进程上下文信息的保存与恢复过程必须在内核态完成。
3. [ ]对于父进程而言，fork()的返回值只能是子进程的 pid 号。
4. [ ]对于分属不同进程的线程 A 和线程 B 之间进行切换，必须要切换页表。
5. [ ]在用户空间中实现的线程模型可以有效的避开操作系统调度带来的时间开销。
6. [ ]对于应用程序而言，编译器生成的程序地址是虚拟地址，由操作系统建立段/页表完成虚实地址转换。
7. [ ]对于采用段页式的 x86 而言，CPU 访问一个虚拟地址时，如 TLB 访问缺失，则需先通过页表，再通过段表才能找到对应的物理地址。
8. [ ]通过动态链接库和操作系统的页表设置，可以让多个不同的应用程序运行时共用一个库函数（如 printf 等）的代码实现。
9. [ ] 当设置好 GDT（全局描述符表）的内容；然后 CPU 执行“lgdt”指令加载 GDT；接着立刻执行“incl 0x80”指令时，CPU 将查找 GDT 并完成虚拟地址 0x80 到线性地址的转换。
10. [ ]在 32 位计算机系统中，因为 4GB 内存普遍存在，导致虚拟内存管理已经不再有存在的必要。
11. [ ]对于实时系统中的优先级反转（反置）问题，可通过优先级继承算法或优先级天花板算法来解决。
12. [ ]信号量机制可实现基于条件变量的管程机制，反之亦然。
13. [ ]在多 CPU 系统中，仅通过 cpu 中断使能和屏蔽指令，就可实现对临界区代码的互斥保护。
14. [ ]在银行家算法中，不安全状态不一定会造成死锁。
15. [ ]操作系统中的虚拟文件系统屏蔽了底层具体文件系统的差异性，给上层应用提供了统一的访问接口。
16. [ ]在 Linux 中，存在不需要把数据保存到磁盘上的文件系统，比如"/proc"文件系统，其作用是给应用程序提供一种内核信息的访问通道。
17. [ ]在当前的计算机系统中，存在计算能力比 CPU 还快的外设。
18. [ ] DMA 机制允许外设不经过 CPU 进行数据传输。

19. [ ]循环扫描算法(C-SCAN)对硬盘访问带来的好处在 U 盘上不存在。

20. [ ]访问频率置换算法(Frequency-based Replacement)的基本思路是，在短周期中使用 LFU 算法，而在长周期中使用 LRU 算法。

二、填空题（20 分）

21. 在基于 x86-32 的 ucore 操作系统中，一般函数调用的参数通过(__21.1__)传递，系统调用的参数通过(__21.2__)传递，将系统调用号存放在(__21.3__)，通过(__21.4__)指令进入内核态。此时还应该保存执行现场，需要在 trapframe 里保存(__21.5__)、(__21.6__)、(__21.7__)等信息（填三项即可）。

22. (__22.1__)是一种将不同文件名链接至同一个文件的机制。它可以使同一文件具有多个不同的名字，而文件系统只存在一个文件内容的副本。(__22.2__)和原文件共享一个相同的 inode 号（文件在文件系统上的唯一标识）。若原文件删除了，则(__22.3__)不能访问它指向的原文件，而(__22.4__)则是可以的。(__22.5__)可以跨越磁盘分区，但(__22.6__)不具备这个特性。

23. RAID 是一种机制，即把多块独立的硬盘按某种方式组合，形成硬盘阵列，从而提供比单块硬盘更快的访问性能或更可靠的数据存储能力。组成磁盘阵列的不同方式称为 RAID 级别，其中，(__23.1__)级别没有数据冗余存储功能，而(__23.2__)的数据可靠性在所有的 RAID 级别中是最高的。RAID 5 是一种存储性能、数据安全和存储成本兼顾的磁盘阵列组成方式。它至少需要(__23.3__)块硬盘。当 RAID5 的一个磁盘数据发生损坏后，可利用剩下的数据和相应的(__23.4__)信息去恢复被损坏的数据。

24. 信号提供了异步处理事件的一种方式。例如，用户在终端按下“Ctrl-C”键，会产生可使当前进程终止的 SIGINT 信号。每一个信号对应一个(__24.1__)数，定义在头文件<signal.h>中。信号处理行为可有三种方式可供选择：(__24.2__)、(__24.3__)、(__24.4__)。

三、问答题

25. （10 分）下面是采用银行家算法的操作系统在某一时刻的资源分配状态。

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

请回答下列问题：

1)请写出当前时刻的 Need 矩阵的内容是什么？

2)当前时刻，系统是否处于安全状态？

3)接下来，如果进程 P1 发出一个请求（0，4，2，0）。这个请求能否立刻被满足？

26. （10 分）通过软件方式可正确实现互斥机制。

1）下列二线程互斥机制的伪码实现是否有错？请给出原因分析，如果有错请给出反例。

```
INITIALIZATION:

   shared int turn;
   ...
   turn = i ;

ENTRY PROTOCOL (for Thread i ):
   /* wait until it's our turn */
   while (turn != i ) {
   }

EXIT PROTOCOL (for Thread i ):
   /* pass the turn on */
   turn = j ;
```

2）下列 N 线程互斥机制的伪码实现是否有误？请给出原因分析，如果有错请给出反例。

```
INITIALIZATION:

   typedef char boolean;
   ...
   shared int num[n];
   ...
   for (j=0; j < n; j++) {
      num[j] = 0;
   }
   ...
ENTRY PROTOCOL (for Thread i):
   /* choose a number */
   num[i] = max(num[0], ..., num[n-1]) + 1;

   /* for all other Threads */
   for (j=0; j < n; j++) {

      /* wait if the Thread has a number and comes ahead of us */
      if ((num[j] > 0) &&
        ((num[j] < num[i]) ||
        (num[j] == num[i]) && (j < i))) {
          while (num[j] > 0) {}
```

```
      }
   }

EXIT PROTOCOL (for Thread i):
   /* clear our number */
   num[i] = 0;
```

27. （15 分）下面是采用信号量实现的管程机制的伪码。

```
 0 IMPLEMENTATION:
 1   monitor mt {
 2     -----variable in monitor-----------
 3     semaphore mutex;                          // the mutex lock for going into the routines in monitor,
should be initialized to 1
 4     semaphore next;                           // the next  is used to down the signaling proc, some
proc should wake up the sleeped cv.signaling proc. should be initialized to 0
 5     int next_count;                           // the number of of sleeped signaling proc, should be
initialized to 0
 6     condvar {int count, sempahore sem}  cv[N];   // the condvars in monitor, count initial value 0,
sem initial value 0
 7     other shared variables in mt;               // shared variables should protected by mutex lock
 8     --------condvar wait---------------
 9     cond_wait (cv) {
 10        cv.count ++;
 11       if(mt.next_count>0)
 12         V(mt.next)                          // first perform the EXIT PROTOCOL
 13       else
 14         V(mt.mutex);
 15       P(cv.sem);                            // now wait on the condition waiting queue (cv.sem)
 16       cv.count --;
 17    }
 18    --------condvar signal--------------
 19     cond_signal(cv) {
 20        if(cv.count>0) {                      // do nothing unless a process is waiting on condition
waiting queue (cv.sem)
 21          mt.next_count ++;
 22          V(cv.sem);                          //  release the  waiting process which on  condition
waiting queue (cv.sem)
 23          P(mt.next);                         // wait on the "next" waiting queue for cv.signaling proc
 24          mt.next_count--;
 25       }
```

```
 26     }
 27    --------routines in monitor-------------
 28    routineA_in_mt () {
 29       P(mt.mutex);                          // ENTRY PROTOCOL (at the beginning of each monitor
routines), wait for exclusive access to the monitor
 30       ...
 31       real body of routineA                              // in here, may access shared variables, call
cond_wait OR cond_signal
 32       ...
 33       if(next_count>0)                       // EXIT PROTOCOL (at the end of each monitor function)
 34          V(mt.next);                        // if there are processes(sleeped cv.signaling proc) in
the "next" queue, release one
 35       else
 36          V(mt.mutex);                       // otherwise, release the monitor
 37    }
```

1）请说明管程的特征。上述管程实现是哪种类型的管程？

2)在上述伪码中，如果有 3 个线程 a,b,c 需要访问管程，并会使用管程中的 2 个条件变量 cv[0],cv[1]。

2.1）请问 cv[i]->count 含义是什么？cv[i]->count 是否可能<0, 是否可能>1？请举例或说明原因。

2.1）请问 mt->next_count 含义是什么？mt->next_count 是否可能<0, 是否可能>1？请举例或说明原因。

28. （10 分）请描述 stride 调度算法的思路？stride 算法的特征是什么？stride 调度算法是如何避免 stride 溢出问题的？

29.（15 分）1）试以图示描述 ucore 操作系统中的 SFS 文件系统的文件组织方式。

2）下面是 SFS 的磁盘索引节点数据结构定义。

```
struct sfs_disk_inode {
   uint32_t size;
   uint16_t type;
   uint16_t nlinks;
   uint32_t blocks;
   uint32_t direct[SFS_NDIRECT];
   uint32_t indirect;
};
```

假定 ucore 里 SFS_NDIRECT 的取值是 16，而磁盘上数据块大小为 1KB。请计算这时 ucore 支持的最大文件大小。请给出计算过程。（这样可给步骤分）