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：2019年05月20日上午09:50~11:25

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19个小题，共7页。
	5. 考试完毕, 必须将试题纸和答卷本一起交回。

一、(20分)判断题

1. [ ] 操作系统通过vfork创建子进程时，需要拷贝父进程的所有页表内容作为子进程的页表内容。
2. [ ] 通过mmap系统调用，可以实现通过直接内存访问来访问文件，避免了read/write系统调用的特权级切换开销。
3. [ ] 当前ucore lab8中实现的磁盘读写操作采用CPU轮询机制来实现。
4. [ ] 信号量与管程都是高层同步互斥机制，无法基于信号量机制来实现管程。
5. [ ] 基于mesa机制的管程在执行条件变量的signal操作过程中，不会睡眠。
6. [ ] 基于hoare机制的管程在执行条件变量的signal操作过程中，不会睡眠。
7. [ ] 基于spinlock的互斥机制实现中，可以确保处于忙等线程集合将按线程忙等的等待时间顺序依次进入临界区。
8. [ ] 访问频率置换算法(Frequency-based Replacement)综合采用了LRU和LFU算法用于磁盘缓存置换。
9. [ ] 发生进程复制时，新进程的内核堆栈可以先于进程地址空间复制操作之前进行创建。
10. [ ] 实模式x86-32 CPU在冷启动后执行的第一条指令的地址由CS段基址加EIP确定，即FFFF_FFF0H = FFFF_0000H + 0000_FFF0H。

二、(15分)填空题

11. 请在下面有数字标号的空格中简要写出文件系统实现中文件分配的三种方式及其主要特点（需指明是否有外碎片问题、随机或顺序读取性能的好坏，可靠性差/好），并解释理由。

连续分配的特点：文件顺序读取性能（---1---），随机读取性能（---2---），（---3---）外碎片。链式分配的特点：（---4---）外碎片，随机读取性能（---5---），可靠性（---6---）。索引分配的特点：（---7---）外碎片，随机访问性能（---8---），可靠性（---9---）。请用小于90个字给出对上面三类分配方式的填写内容的解释理由（---10---）

12. 某系统中共有M种类型资源，每种资源的个数为R_1，R_2，R_3，...，R_M，共有N个进程竞争使用这些资源，每个进程对这些资源的需求量为K_1，K_2，K_3，...，K_M，每个进程会依次获取这些资源，如下面的伪代码所示：

```
//获取资源
for i = 1 to M:        // 遍历每种类型
   for k = 1 to Ki:    // 每次获取Ri的一个资源，获取Ki次
     P(Ri)

// 使用资源
...
// 释放资源
for i = 1 to M:
   for k = 1 to Ki:
      V(Ri)
 ...
```

当M=1，N=2时，在下面几种设定中，按照上面的获取方式，这些进程会发生死锁吗?

R_1=11，K_1=7 （---11---）

R_1=11，K_1=6 （---12---）

当M=1时，R_1，N，K_1需要满足（---13--- ）不等式关系，才能使得这些进程不发生死锁。

13. 设初始情况下，一个一般文件file1的当前引用计数值为1，如果先建立file1的符号链接（软链接）文件file2，再建立file1的硬链接文件file3，然后再建立file3的硬链接文件file4。此时，file3和file4的引用计数值分别是 （---14---）和（---15--- ）。

三、(65分)问答题

14. (12分)

设在单CPU的计算机系统中，信号量的原理性定义如下

```
class Semaphore {
    int sem;//信号量值
    WaitQueue q;//等待队列
    Semaphore(): sem(1) {} //信号量初始值
}
```

若P、V操作的原理性实现有A和B两种形式，请分别指出这两种实现形式是否存在潜在的问题？若存在，请说明存在的问题并给出具体的例子来对应问题。若不存在，请给出解释理由。

A实现形式:

```
Semaphore::P(){
    if (sem <= 0) {
        t = get_current_thread();//当前正在运行的线程t
        q.enqueue(t);//让t线程入等待队列
        block(t);//让t线程等待
    }
    sem--;
```

```
}

Semaphore::V() {
    sem++;
    if (!q.empty()) {
        t = q.dequeue();//让t线程出等待队列
        wakeup(t);//让t线程出等待队列
    }
}
```

B实现形式：

```
Semaphore::P(){
    while (sem == 0) {
        t = get_current_thread();
        q.enqueue(t);
        block(t);
    }
    sem--;
}

Semaphore::V() {
    sem++;
    if (!q.empty()) {
        t = q.dequeue();
        wakeup(t);
    }
}
```

15. (12分)

读者优先的读者-写者问题：有读者和写者两组并发进程，共享一个数据文件。当两个或以上的读进程同时访问该数据文件时，不会产生副作用；但如某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误。因此要求各进程按如下规则访问数据文件。

1. ）允许多个读者可以同时对数据文件进行读操作；
2. ）只允许一个写者对数据文件进行写操作；
3. ）写者的写操作和读者的读操作是不能同时进行的。

请基于下面代码框架回答如下问题：

1. ）用信号量机制实现读者写者问题的正确且高效的同步与互斥；要求用类C语言的伪代码实现，并给出必要的简明代码注释。
2. ）请按数据文件访问规则的要求，给出测试用例；要求至少给出5种可能情况的测试用例，并在每个测试用例中注明至少一种不同的情况。

读者-写者问题的实现代码框架：

```
#include "stdio.h"
#include "stdlib.h"
```

```c
#include "pthread.h"
#include "unistd.h"
#include "string.h"

#define BUFFER_SIZE 2
#define SLEEP_SPAN 5
#define WORK_SPAN 4
#define N 1

// Initialization: YOUR WORK --(1)--

struct arg_struct{
    int id;
    int start;
    int work;
    char indent[10];
};

void * reader(void * argv){
    struct arg_struct arg = *(struct arg_struct*)argv;
    int id;
    id = arg.id;
    char* indent = arg.indent;

    sleep(arg.start);
    printf("%ssReader\n", indent);

    //Start Read:YOUR WORK --(2)--

    //Read
    printf("%ssRead\n", indent);
    sleep(arg.work);
    printf("%seRead\n", indent);

    //End Read: YOUR WORK --(3)--

    return NULL;
}

void * writer(void * argv){
    struct arg_struct arg = *(struct arg_struct*)argv;
    int id;
    id = arg.id;
    char* indent = arg.indent;

    sleep(arg.start);
    printf("%ssWriter\n", indent);

    //Start Write: YOUR WORK --(4)--

    //Write
    printf("%ssWrite\n", indent);
    sleep(arg.work);
```

```
    printf("%seWrite\n", indent);

    //End Write: YOUR WORK --(5)--

    return NULL;
}

int main(int argc, char** argv) {
    srand((unsigned)time(NULL));

    // Initialization: YOUR WORK --(6)--

    pthread_t p_reader[2 * N], p_writer[2 * N];

#define WRITER 0
#define READER 1

    /* For managed creation of 2 * N threads */

    int st_time = 0;
    int inst[2 * N][10] = {
        //{READER, 0, rand()%WORK_SPAN +1},
        //{READER, 1, rand()%WORK_SPAN +1}
        // Testcase: YOUR WORK --(7)--

    };

    /* Print the first line */
    int tmp_r = 0, tmp_w = 0;
    for (int i = 0; i < 2 * N; i++){
        if (inst[i][0] == READER){
            printf("R%d\t", tmp_r++);
        } else if (inst[i][0] == WRITER){
            printf("W%d\t", tmp_w++);
        }
    }
    printf("\n");

    /* Create Readers and Writers according to $inst*/
    int rc;
    int r_count = 0;
    int w_count = 0;
    char indent[10];
    struct arg_struct args[2 * N];

    strcpy(indent,  "");
    for (int i = 0; i < 2 * N; i++){
        args[i].id = inst[i][0];
        args[i].start = inst[i][1];
        args[i].work = inst[i][2];
        strcpy(args[i].indent, indent);
        if (inst[i][0] == READER){
            rc = pthread_create(p_reader + r_count, NULL, reader, &args[i]);
```

```
            if (rc) printf("ERROR\n");
            r_count++;
        } else if (inst[i][0] == WRITER){
            rc = pthread_create(p_writer + w_count, NULL, writer, &args[i]);
            if (rc) printf("ERROR\n");
            w_count++;
        }
        strcat(indent, "\t");
    }

    /* wait until every thread finishes*/
    for (int i = 0; i < r_count; i++){
        pthread_join(p_reader[i], NULL);
    }
    for (int i = 0; i < w_count; i++){
        pthread_join(p_writer[i], NULL);
    }

    return 0;
}
```

测试用例的输出示例：

```
MacBookPro-3:reader-writer xyongcn$ gcc -pthread reader-writer.c
MacBookPro-3:reader-writer xyongcn$ ./a.out
R0  R1  R2  W0  R3  W1  R4  R5  W2  R6
sReader
sRead
    sReader
    sRead
    eRead
eRead
MacBookPro-3:reader-writer xyongcn$
```

16. (12分)

ucore lab 6中需要完成 stride 调度算法。为此需要对stride调度算法有清楚准确的理解。

1. ）假设某个调度器需要调度三个进程 A B C，它们的优先级分别是 1, 2, 4。 在初始时刻，它们的 stride 值均为 0。 假设他们总不会被挂起，意即： **所有调度的发生都是时间片用完导致的抢占。** 假设不考虑精度和溢出，即你可以认为 stride 值用整数表示。 假设不会有新的进程加入，并且 A B C 也不会退出。

   请证明（比如用归纳法）：在理论上，对于任意非负整数 n，当 $7n$ 个时间片过去后，A B C三个进程分别执行了 n, $2n$ 和 $4n$ 个时间片。

2. ）第一问的结论说明了 stride调度算法具有什么样的特征？除精度和溢出外，列出一个可能对stride调度算法特征产生影响/改变的因素。

3. ）假设用 w 位 **无符号整数** 表示 stride 值和进程的优先级。 假设系统中有 N 个进程，进程 i 的优先级是 p_i（也是 w 位无符号整数）。 若要使用 stride 调度算法调度这 N 个进程，并且保证第一问中 stride调度算法的特征近似成立。 **请问**：他们的 p_i 需要满足什么条件？请列出两个不等式来表示条件，并给出必要的解释描述。

17. (10分)

文件系统中函数`void rename(char* from, char* to)`的作用是，将`from`路径名的文件改名为`to`路径名。请回答如下问题：

1. ） 利用`link`和`unlink`系统调用可以在用户库中实现函数`rename`的功能。请给出伪代码，并简要注释。其中`link(char* from, char* to)`实现了硬链接，即创建路径名为`to`的目录项指向路径名为`from`的文件，`unlink(char\* name)`删除名为`name`的目录项，该目录项如果不存在则unlink为空操作。
2. ）对于上面这种实现方式，如果应用程序在调用`rename("a", "b")`用户态函数的过程中出现了某种死循环的应用程序bug，请列出文件系统此时可能的状态。（假设"a","b"已经存在于当前目录下，且指向不同的inode；unlink/link系统调用要么完全成功，要么完全失败。）

18. (11分)

某系统中有8台打印机，它们由N个进程竞争使用，每个进程可能需要3台打印机。请问N取值为多少时，系统没有死锁危险，请解释理由。现在系统中再增加两台打印机，达到了10台打印机，且只有三个进程P_1，P_2，P_3竞争使用。P_1，P_2，P_3总共分别需要8台，7台和4台。若P_1，P_2，P_3已申请到4台，2台和2台。试问：按银行家算法能安全分配吗？如能安全分配，请说明分配过程。 如不能安全分配，请说明理由。

19. (8分)

在一个采用虚拟页式存储的超微计算机系统中，内存为256字节。页面大小为16字节，采用二级页表结构，每个页表项占1字节，页表大小4字节。页表项组成如下：高4位为二级页表或访问页面的物理页号，低4位分别是存在位(Exist)、访问位(Read)、修改位(Write)和保留位(Reserved)。假定当前系统中正有两个并发执行的进程，进程A的第一级页表起始地址为0xB0。整个物理内存的存储内容如下。

请描述进程A访问逻辑地址0x20、0x40和0x80的地址转换计算过程。要求写出计算过程，并给出对应的一级页表项、二级页表项和访存单元的物理地址和对应的存储内容。

Complete Physical (Linear) View

0x00	FC	00	00	00	0x40	00	00	00	00	0x80	CA	FE	BA	BE	0xc0	DE	AD	BE	EF
0x04	00	00	00	00	0x44	00	00	00	00	0x84	00	00	00	00	0xc4	00	00	00	00
0x08	00	00	00	00	0x48	00	00	00	00	0x88	00	00	00	00	0xc8	00	00	00	00
0x0c	00	00	00	00	0x4c	00	00	00	00	0x8c	00	00	00	00	0xcc	00	00	00	00
0x10	BC	00	00	00	0x50	00	00	00	00	0x90	AA	AA	AA	AA	0xd0	BB	BB	BB	BB
0x14	00	00	00	00	0x54	00	00	00	00	0x94	AA	AA	AA	AA	0xd4	BB	BB	BB	BB
0x18	00	00	00	00	0x58	00	00	00	00	0x98	AA	AA	AA	AA	0xd8	BB	BB	BB	BB
0x1c	00	00	00	00	0x5c	00	00	00	00	0x9c	AA	AA	AA	AA	0xdc	BB	BB	BB	BB
0x20	00	00	00	00	0x60	00	00	00	00	0xa0	00	00	00	00	0xe0	00	00	00	00
0x24	00	00	00	00	0x64	00	00	00	00	0xa4	00	00	00	00	0xe4	00	00	00	00
0x28	00	00	00	00	0x68	00	00	00	00	0xa8	00	00	00	00	0xe8	00	00	00	00
0x2c	00	00	00	00	0x6c	00	00	00	00	0xac	00	00	00	00	0xec	00	00	00	00
0x30	00	00	00	00	0x70	00	00	00	00	0xb0	8C	9A	A9	00	0xf0	CC	DA	E9	00
0x34	00	00	00	00	0x74	00	00	00	00	0xb4	00	00	00	00	0xf4	00	00	00	00
0x38	00	00	00	00	0x78	00	00	00	00	0xb8	00	00	00	00	0xf8	00	00	00	00
0x3c	00	00	00	00	0x7c	00	00	00	00	0xbc	00	00	00	00	0xfc	00	00	00	00