里面有一题ucore填空的没有，可参考https://blog.csdn.net/marshazheng/article/details/84639441

剩下的应该是官方答案，题目可能有些许不同。祝好

一、对错题 用V表示对，用X表示错

1. [x]在多CPU场景下，多个进程通过自旋锁(spinlock)争抢进入临界区执行，第一个成功进入临界区的进程是第一个执行自旋锁争抢的进程。
2. [v] 运行在内核态的内核线程共享操作系统内核态中的一个页表。
3. [v] 操作系统创建用户进程时需要为此用户进程创建一个内核栈用于执行系统调用服务等。
4. [v] 通用操作系统的调度算法的主要目标是低延迟，高吞吐量，公平，负载均衡。
5. [v] 单处理器场景下，短剩余时间优先调度算法(SRT)可达到具有最小平均周转时间的效果。
6. [x]单处理器场景下，无法通过打开和关闭中断的机制来保证内核中临界区代码的互斥性。
7. [v] 信号量可用于解决需要互斥和同步需求的问题。
8. [v] 属于管程范围的函数/子程序相互之间具有互斥性。
9. [v] 操作系统处于安全状态，一定没有死锁；操作系统处于不安全状态，可能出现死锁。
10. [v] 80386取指地址是base+eip，base是隐藏寄存器，初始化为0xffff0000，eip初始化为0xfff0，故执行的第一条指令是0xfffffff0。
11. [v] 在x86-32 CPU下，操作系统可以实现让用户态程序直接接收并处理硬件中断。
12. [v]由于符号链接（软链接）实际上是一个特殊的文件，它的内容就是其所指向的文件或目录的路径，所以符号链接可以指向一个不存在的文件或目录。
13. [x]文件系统中，文件访问控制信息存储的合理位置是文件分配表。
14. [x]在操作系统中一旦出现死锁, 所有进程都不能运行。
15. [x]在ucore for x86-32中，子进程通过sys_exit()执行进程退出时，ucore kernel会先释放子进程自身内核堆栈和进程控制块等，再唤醒父进程（或initproc），最后执行iret返回。

RAID(9分)

现有一个RAID磁盘阵列，包含6个磁盘，每个磁盘大小都是2TB，最大写入速度 200 MB/s，最大读取速度 250 MB/s 的硬盘。用它们分别组成RAID级分别为0、1和5。请在理想情况下回答下列问题。

1. 用它们组成的 RAID0 阵列的总可用空间为 (`__1__`)，最大写入速度为 (`__2__`)，最大读取速度为 (`__3__`);

2. 用它们组成的 RAID1 阵列的总可用空间为 (`__4__`)，最大写入速度为 (`__5__`)，最大读取速度为 (`__6__`);

3. 用它们组成的 RAID5 阵列的总可用空间为 (`__7__`)，最大写入速度为 (`__8__`)，最大读取速度为 (`__9__`)。

参考答案

1. 用它们组成的 RAID0 阵列的总可用空间为 (12 TB)，最大写入速度为 (1.2TB/s)，最大读取速度为 (1.5TB/s);

2. 用它们组成的 RAID1 阵列的总可用空间为 (6TB)，最大写入速度为 (600 MB/s)，最大读取速度为 (1.5TB/s);

3. 用它们组成的 RAID5 阵列的总可用空间为 (10 TB)，最大写入速度为 (1TB/s)，最大读取速度为 (1.25TB/s)。

页表（5分）

假定在X86-32平台上的ucore的虚拟存储系统中，采用4KB页大小和二级页表结构。请补全功能为通过虚拟地址找到对应的页表项的`get_pte()`函数。

可能需要用到的函数有：

`page2pa()` 获取物理页对应的物理地址；

`page2ppn()` 获取物理页对应的物理页号；

`pa2page()` 获取物理页号对应的物理页数据结构指针；

`page2kva()` 获取物理页对应内核虚拟地址；

`kva2page()` 从内核虚拟地址获取物理页数据结构指针；

可能用到的宏有：

`memset(p,v,n)` 对指定地址`p`开始的长度为`n`的内存区域进行赋值`v`

`PDX(la)` 虚拟地址`la`对应的页目录项序号；

`KADDR(pa)` 物理地址`pa`对应的内核虚拟地址；

`PADDR(kva)` 虚拟地址`kva`对应的物理地址；

`PTE_P`：存在标志位

`PTE_W`：可修改标志位

`PTE_U`：用户可访问标志位

```c
pte_t *get_pte(pde_t *pgdir, uintptr_t la, bool create){

    pde_t *pdep = 1._________;
    if(!*pdep & 2. _______){
        struct Page *page;
        if(!create || (page = alloc()) == NULL){
            return NULL;
        }
        set_page_ref(page,1);
        uintptr_t pa = 3. ________;
        memset(4._________,0, PGSIZE);
        *pdep = 5._________;
    }
    return &((pte_t *)KADDR(PDE_ADDR(*pdep)))[PTX(la)];

}
```

答案

1. &pgdir[PDX(la)] 或者 pgdir + PDX(la) ,任意能够计算出类似地址的

2. PTE_P

3. page2pa(page)

4. KADDR(pa)

5. pa | PTE_U | PTE_W | PTE_P;

文件系统（15分）

假定某UFS文件系统采用多级索引分配的方法，普通文件的索引节点(inode)中包括一个占8字节的文件长度字段和15个占4字节的数据块指针（即块号）。其中，前12个是直接索引块（direct block）的指针，第13个是1级间接索引块（indirect block）指针，第14个是2级间接索引块（doubly indirect block）指针，第15个是3级间接索引块（triply indirect block）指针。间接索引块中连续存放占4字节的数据块指针。数据块以及间接块的大小为4KB。请回答下列问题。

1. 计算该文件系统理论上能够支持的最大文件长度。

2. 假设读取磁盘上一个块需要1ms，块缓存机制只缓存文件的索引节点，并且所有需要的索引节点都已加载到缓存；不缓存数据块以及间接索引块；读取文件操作不会发生写入操作，访问内存的时间忽略不计。计算从头到尾读取一个8MB（即2048块）文件需要的时间。

3. 假设读取磁盘上一个块需要1ms，块缓存机制缓存文件的索引节点、数据块和间接索引块，并且所有需要的索引节点都已加载到缓存；开始时缓存没有加载任何数据块或者间接块；读取文件操作不会发生写入操作，访问内存的时间忽略不计。计算从头到尾读取一个8MB（即2048块）文件需要的时间。

参考答案

（1）请计算该文件系统理论上能够支持的最大文件大小。但是，事实上ext2并不支持这么大的文件，请想一想这是为什么（不必作答）。

直接数据块数：12

1级间接的数据块数：$(\frac{4K}{4})^1 = 1K$

2级间接的数据块数：$(\frac{4K}{4})^2 = 1M$

3级间接的数据块数：$(\frac{4K}{4})^3 = 1G$

所以，理论最大文件大小为$(12 + 1K + 1M + 1G) \times 4KiB = 48KiB + 4MiB + 4GiB + 4TiB \approx 4TiB$

（2）假设读取磁盘上一个块需要1ms，块缓存机制只对文件的inode起作用且需要的inode已加载到缓存，对数据块以及间接块没有缓存，那么请问从头到尾读取一个10MiB = 2560块的文件需要多长时间？假设读取文件操作不会发生写入操作，访问内存的时间忽略不计。

读取直接块访问磁盘次数：1×12

此时，还剩2548块。

读取1级间接的数据块访问磁盘次数：$2 \times 1024 = 2048$

此时，还剩1524块。

读取2级间接的数据块访问磁盘次数：$3 \times 1524 = 4572$

总计次数为$12 + 2048 + 4572 = 6632$，所以时间为$6.632s$。

（3）如果对数据块以及间接块有缓存呢？假设一开始缓存没有加载任何数据块或者间接块。

读取直接块访问磁盘次数：1×12

此时，还剩2548块。

读取1级间接的数据块访问磁盘次数：$1 + 1 \times 1024 = 1025$

此时，还剩1524块。

由于$1024 \lt 1524 \le 2048$，

读取2级间接的数据块访问磁盘次数：$1 + 2 + 1 \times 1524 = 1527$

总计次数为$12 + 1025 + 1527 =2564$，所以时间为$2.564s$。

答$2.563s$是错的。

同步算法（5分）

无锁 (Lock-Free) 数据结构在工程实践中有十分重要的应用，因而常用处理器的指令集都提供了相应的指令来帮助我们实现无锁数据结构，如下为 x86 平台的 **cmpxchg** 指令的伪代码。

```c
int cmpxchg(void* addr, uint32_t oldval, uint32_t newval) {
    if (*addr != oldval) {
        return 0;
    }
    *addr = newval;
    return 1;
}
```

考虑使用该指令实现一个无锁 LIFO 队列

```c
struct node {
    struct node* next;
    /* Other data fields */
};

struct node* head;

void push(struct node* node) {
    do {
        node->next = head;
    } while (!cmpxchg(&head, (uint32_t)node->next, (uint32_t)node));
}

struct node* pop() {
    while (1) {
        struct node* node = head;
        if (node == NULL) {
            return NULL;
        }
        struct node* next = node->next; // (*)
        if (cmpxchg(&head, (uint32_t)node, (uint32_t)next)) {
            return node;
        }
    }
}
```

上述实现存在 bug，假设现在有 CPU1 和 CPU2 同时操作该 LIFO 队列，请给出一个操作序列，使得最终该队列处于一个不一致的状态，即已经弹出的元素仍然在 LIFO 队列中或者未弹出的元素不在 LIFO 队列中。以下给出了 LIFO 队列的初始状态和第一个操作，请补全能够触发 bug 的操作序列

```
0. LIFO 队列初始状态: `head -> node A -> node B -> node C -> NULL`

1. CPU1 开始调用 `pop`，运行至函数体内 `(*)` 处，此时有 `node = A` 和 `next = B`

2. ......
```

Answer

2. CPU2 调用 `pop`，结果状态: `head -> node B -> node C -> NULL`

3. CPU2 调用 `pop`，结果状态: `head -> node C -> NULL`

4. CPU2 调用 `push(A)`，结果状态: `head -> node A -> node C -> NULL`

5. CPU1 继续运行，`cmpxchg` 发现 `head == A`，于是设置 `head = B`，此时已经弹出的 `node B` 出现在了 LIFO 栈中

管程（20分）

（**20**分）理发店理有m位理发师、m把理发椅和n把供等候理发的顾客坐的椅子。理发师按如下规则理发。

1. 理发师为一位顾客理完发后，查看是否有顾客等待，如有则唤醒一位为其理发；如果没有顾客，理发师便在理发椅上睡觉。

2. 一个新顾客到来时，首先查看理发师在干什么，如果理发师在理发椅上睡觉，他必须叫醒理发师，然后理发师给顾客理发；如果理发师正在理发，则新顾客会在有空椅子可坐时坐下来等待，否则就会离开。

请回答如下问题：

1. 用管程机制实现理发师问题的正确且高效的同步与互斥活动；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. 请按理发规则的要求，给出测试用例；要求至少给出5种可能情况的测试用例。

参考答案

https\://piazza.com/class/i5j09fnsl7k5x0?cid=391

http\://os.cs.tsinghua.edu.cn/oscourse/OS2013/lab7/sync04

Stride调度算法（10分）

ucore lab6中使用Stride调度算法，取BIGSTRIDE=100，假定各个进程的stride初始化为0。请回答下列问题。

1. 如果不考虑进程stride的值的溢出，那么对于任意两个进程A、B的stride值SA和SB，应当恒有abs(SA-SB) ≤ (1)_____，为什么?

2. 考虑到`abs(SA-SB) `这一性质，假设stride值存在溢出，可将stride值的更新变为：

`stride = (stride + BIGSTRIDE/priority) mod n`

那么，只要n > (2)_____，那么stride算法就可以正常运行，为什么?

答案

（1）：100

原因：stride算法要求调度当前stride值最小的进程，假设某个时刻首次出现了A、B两个进程存在SB>SA+BIGSTRIDE的情况，那么意味着在上一轮调度中，必然在SB>SA（但SB≤SA+BIGSTRIDE）的情况下调度了B而不是A，这就与stride算法矛盾。

（2）：200

原因：若真实值有SB>SA，但SB溢出了SA没溢出，那么需要它们的差值SA-SB>BIGSTRIDE用以判断出SB已经溢出，实际上真实的stride值应当是SB>SA。

即需要满足：

a. SB-SA ≤ BIGSTRIDE

b. SA-(SB-n) > BIGSTRIDE

若需要(b)式恒成立，需要n>2*BIGSTRIDE