节选自清华计算机考研912真题（2016-2020年）操作系统部分，按照清华大学陈俞向勇老师的操作系统课程PPT对知识点的分类形式对这部分考题做了简单的分类，并配有本人相应解答，仅供参考，如有疑问，可以在文章下方留言。

**物理内存，虚拟存储**

**Belady现象**

**判断**：FIFO算法存在belady现象 (2020年)

正确 ✔

FIFO算法有Belady现象，并且时钟算法也有Belady现象。

但是LRU和OPT算法没有Belady现象。

产生belady的算法：FIFO，CLOCK，改进CLOCK，不恢复计数的LFU。

不产生belady算法：OPT，LRU，恢复计数的LFU。

**证明**：(6分) LRU、BEST、CLOCK、FIFO 页面置换算法是否能产生 belady 异常，若可以举出例子，不可以给出证明 (2018年) ！重要

答：

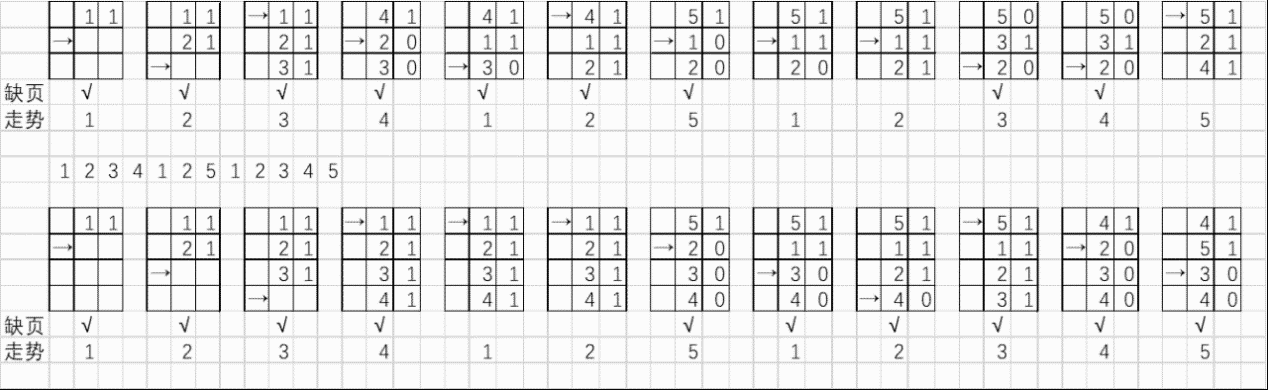
* 产生belady异常的页面置换算法：FIFO，CLOCK，不恢复计数的LFU
* 不产生belady异常的页面置换算法：BEST，LRU，恢复计数的LFU

**FIFO**反例：序列：123412512345

物理页面数为4时缺页10次，而物理页面数为3时缺页9次，页面增加，缺页次数反而增多，故属于belady异常。

**CLOCK**反例：因如果所有页的访问位都为1时，clock算法将退化为FIFO，可以同样构造序列：123412512345。

物理页面数为4时缺页10次，而物理页面数为3时缺页9次，页面增加，缺页次数反而增多，故属于belady异常。



**不恢复计数的LFU**反例：序列：0 0 1 1 1 2 2 0 0 2 2 3 1 3 1 3 1 3 1 3 1 3 1 ... 最后的序列是3号页和1号页的循环

当物理页帧数为2时，之后的3和1循环段不会出现任何缺页当物理页帧数为3时，之后的3和1循环段总是缺页

构造实例的方法是：

对于指定的置换算法，在把所有已分配的页面用上后，访问一个新页，就会出现缺页；然后就可以构造一个序列，每次都访问刚被置换的页面。这个序列就应该是belady现象的实例。

**OPT**证明：<https://piazza.com/class/i5j09fnsl7k5x0?cid=1296>

* 不存在，n个物理页面，包含1个最近访问的页面，未来一段时间t1将访问的不同页面的集合S1构成了剩下n-1个页（t1为访问第n个不同页面的时间 ）。若有n+1个物理页面，访问第n+1个不同页面的时间t2显然大于t1，剩下的n个页的集合S2也应包含S1。

**LRU**证明：<https://piazza.com/class/i5j09fnsl7k5x0?cid=205>

* 不存在，对于LRU算法，如果内存页的集合为最近引用的页，那么对于帧的增加，这n页仍然是最近引用的页，所以也仍然在内存中，所以对于帧数为n的内存页集合是对于帧数为n+1的内存页集合的子集 。

**恢复计数的LFU**证明：<https://piazza.com/class/i5j09fnsl7k5x0?cid=1297>

对不产生belady的**统一的证明思路**：

**假设n个物理页，记内存中的页面集合为S，若物理页增加1，则内存中的页面集合为S‘，只需证明。**（[From Here](https://piazza.com/class/i5j09fnsl7k5x0?cid=1014)）

* 对于OPT，S是最近的n个之后要访问的页面，S’是最近的n+1个之后要访问的页面，集合S’包含集合S
* 对于LRU，S是最近的n个之前已访问的页面，S’是最近的n+1个之前已访问的页面，集合S’包含集合S

所以S中命中的页，在S‘中也一定命中。因而缺页率必然不会增加，因此不会产生belady异常。

**多选题**：以下会发生 belady 异常的是（） (2017年)

a）FIFO 算法

b）LRU 算法

c）CLOCK 算法

d）LFU 算法

e) 改进 CLOCK 算法

选 A C D E

产生belady的算法：FIFO，CLOCK，改进CLOCK，不恢复计数的LFU。

不产生belady算法：OPT，LRU，恢复计数的LFU。

LFU是最近最不常用页面置换算法(Least Frequently Used),也就是淘汰一定时期内被访问次数最少的页。这里需要注意，对于**恢复计数**，当某页被换出后，其访问次数会被记录下来为n，当此页被再次访问，并被换入时，此页的访问次数为n+1。在这种定义下的LFU是没有Belady异常的。

若LFU没有特指，则认为LFU不恢复计数，因为PPT/讲课中使用的不恢复计数的LFU。

**内存分配算法**

多选题：以下算法会产生很多不必要的小碎片的分区 (2017年)

a）最佳匹配算法

b）首次适应算法

c）最坏匹配算法

选A



多选题：能够有效避免产生小碎片的算法是（） (2017年)

a）最佳匹配算法

b）首次适应算法

c）最坏匹配算法

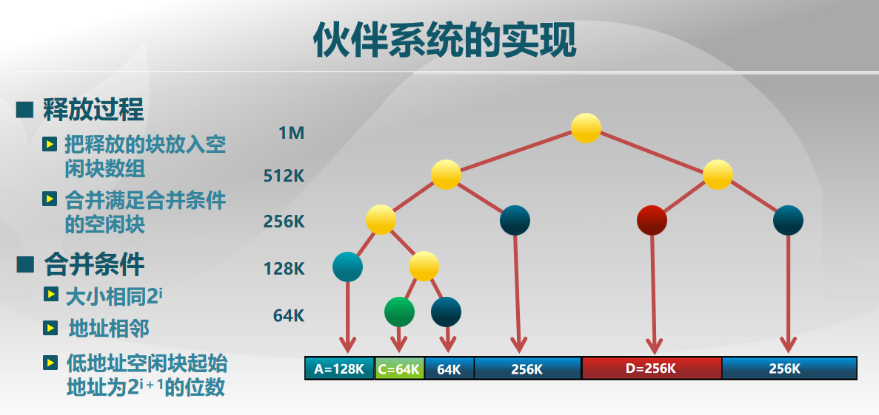
选 C

**判断：**Buddy 算法中，释放一个空间后可以根据起始长度和大小与相邻空闲空间合并 (2018年)

正确 ✔

必须满足以下3个条件才可以合并

1. 这两个空闲空间大小相同都为2^i
2. 地址相邻
3. 低地址空闲块起始地址为2^(i+1)位数



**中断**

**判断：**中断向量表中存放着中断门的优先级。 (2019年)

错误 ❌

不完全正确，有可能是题目不全。

实模式下一般叫中断向量表

保护模式下一般叫中断描述符表（也可简称为保护模式下的中断向量表）

而对于中断向量表，实模式下没有特权级的说法，实模式的中断向量表的每一项只有段基址+偏移地址。

保护模式下的中断向量表又称为中断描述符表，每一项中保存有DPL（即优先级）。

多选题：以下由硬件完成的是( ) (2017年)

A.获取中断源

B.形成中断入口地址

C.Eax 寄存器保存

D.

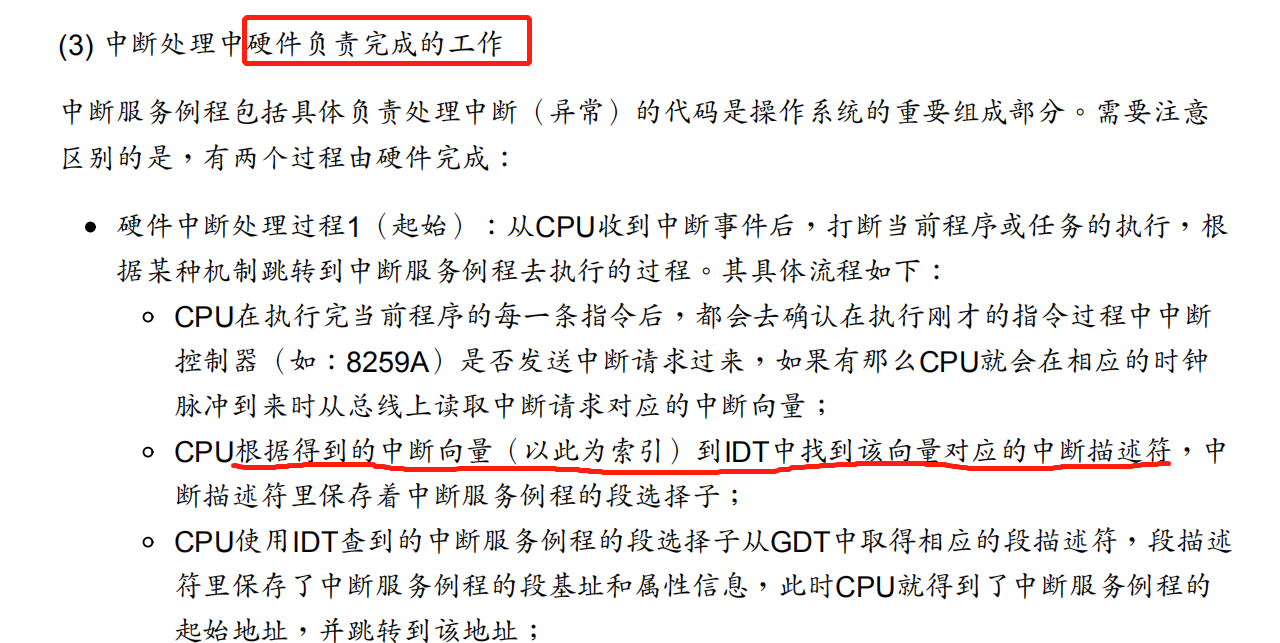
选A，B

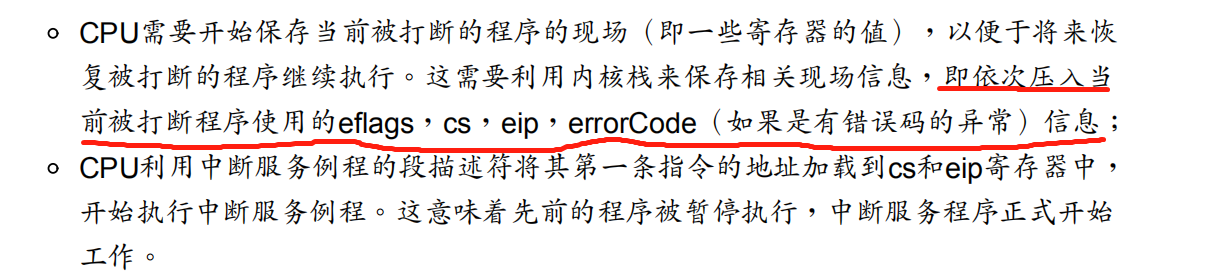
EFLAGES、CS、EIP、ErrorCode由硬件负责保存，通用寄存器由OS负责保存（switch\_to）。

获取中断源是指CPU根据中断向量查询到对应的中断描述符，从而确认造成中断的原因。

x86是根据IDTR加上相应的偏移去取中断描述符表里面的项，然后找到对应中断描述符项里面的地址，然后跳过去，之后才是中断处理程序。

参考： *实验指导书-P125*





**ucore相关**

**判断：**x86开启了二级页表，则⼀级页表可以不在内存中。 (2019年)

错误 ❌

一级页表要放在内存中

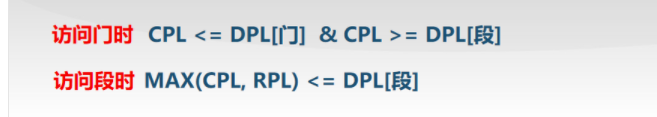
**判断：**ucore时钟中断为10ms，故不能完成 10ms以下的定时。 (2019年)

正确 ✔

定时需要靠时钟中断，且是最小的单位了，是不可被屏蔽的中断

**填空：**CPL<=DPL[门]和CPL<=DPL[段]， 表示请求时可以和门特权级相同， 表示请求时应低于段的特权级。 (2019年)

CPL<=DPL[门] 表示请求时可以和门特权级相同，CPL<=DPL[段] 表示请求时应低于段的特权级。



x86-32CPU的硬件组成，CR3寄存器用于存储页目录表起始 。 (2019年)

物理地址

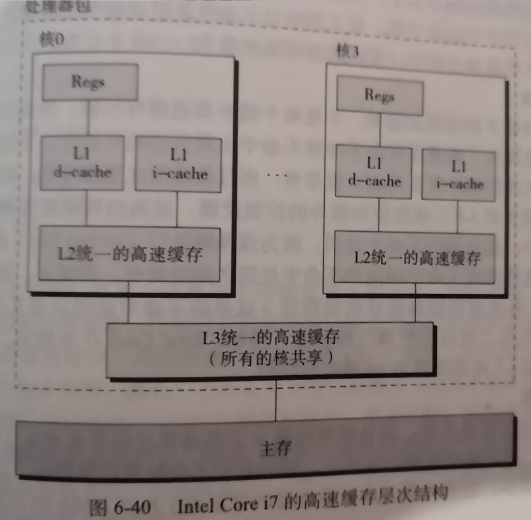
**CR3属于MMU**的页表管理部分，由于实现地址转换必须要有物理地址定位页目录。所以CR3是物理地址。CR3寄存器中保存的是页目录的起始物理地址，CPU只在地址转换中使用它的内容。

忘了就记得选项 (2016年)

C 有些数据 cache 并不能存放

D 多核处理器共享一个高速缓存

D选项错，高速缓存分级，多核处理器的高速缓存有共享部分也有互斥部分。



**简答题**

（6分）UCORE在以下代码中找到页表切换，堆栈切换，switch\_to函数获得两个函数参数的代码部分并注释 (2019年)



答：

**页表切换**：

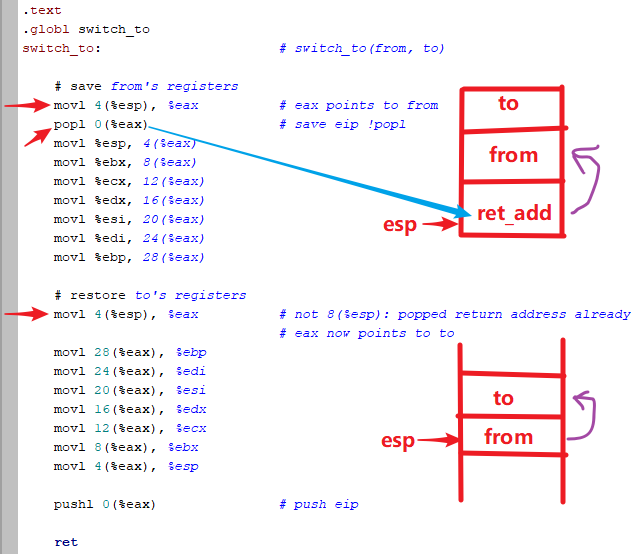
* lcr3(next->cr3);：设置CR3为next内核线程的页目录起始地址。

**堆栈切换**：

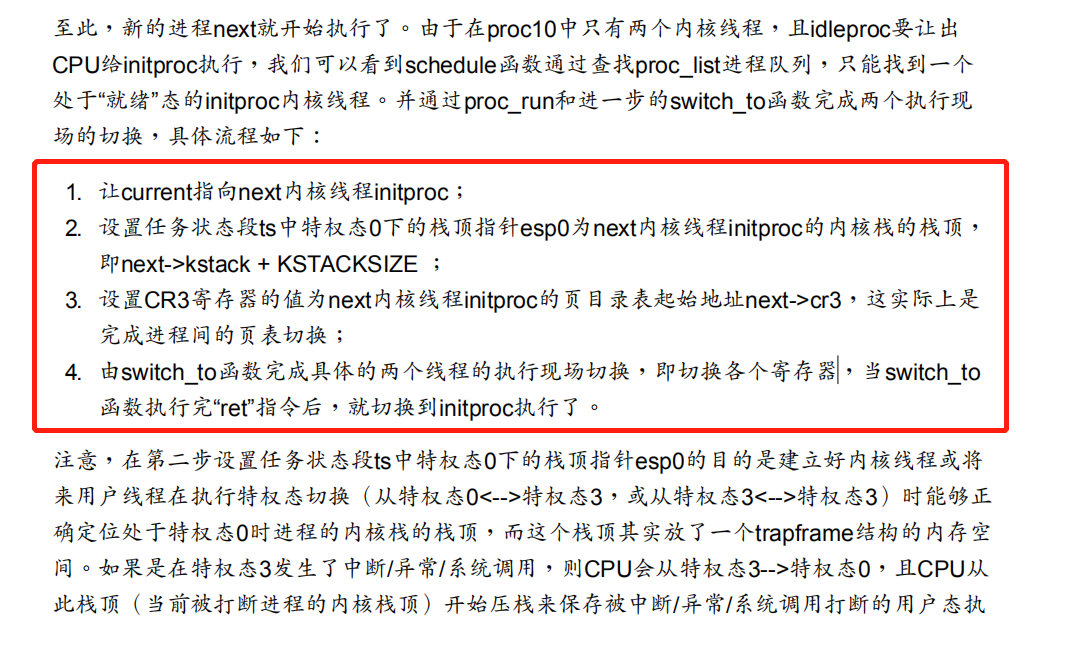
* load\_esp0(next->kstack + KSTACKSIZE);：将esp0内核栈顶指针指向next内核线程的内核栈栈顶。

**switch\_to**：（汇编代码在kernel/process/switch.S的switch\_to中）

* 函数原型：void switch\_to(struct context \*from, struct context \*to);即获得from和to两个指针的代码。
  + 调用函数时候，switch\_to(from, to)翻译成了pushl to, pushl from, call switch\_to
  + 第1个movl 4(%esp), %eax：将栈中from的context指针赋给%eax
  + 第2个movl 4(%esp), %eax：将栈中to的context指针赋给%eax（此时已经popl了一次栈顶）

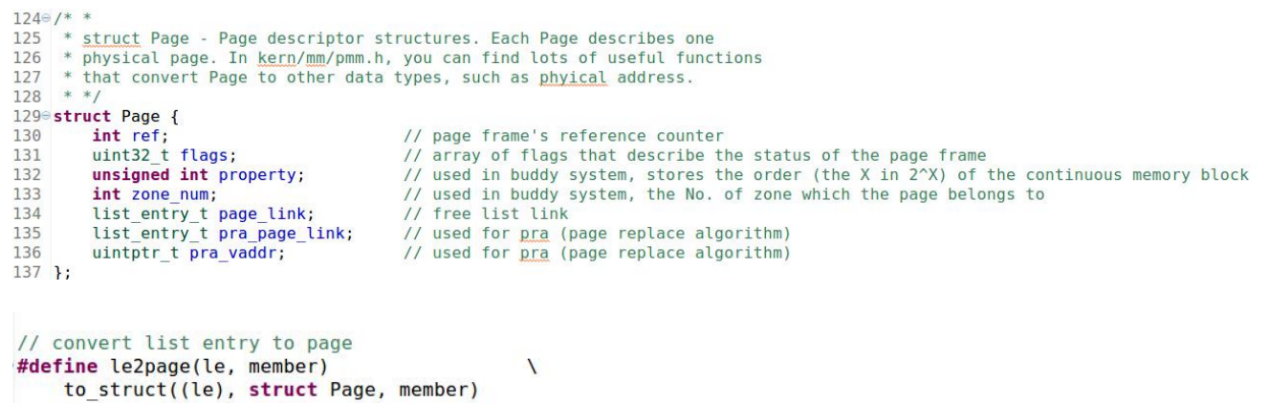


该部分的详细内容位于实验书的P227（lab4-内核线程-调度并执行内核线程initproc）。



（4分）内存分析。给了段内存的 dump，以及⼀些页表项的信息，分析 A、B 进程的逻辑地址转换 (2019年)

（6分）ucore (2018年)





le2page(\*page,page\_link)语句都需要展开那些宏定义？说明这个语句的含义。(还有一段 ucore 代码是 buddy 算法的页面分配函数，好像跟这道题关系不大，就不贴了主要是没找到。)

答：le2page(\*page,page\_link)语句 需要展开 to\_struct宏，offsetof宏

le2page的作用是根据链表指针寻找到包含此链表指针的Page数据类型的头指针（即其宿主）。

le2page：list entry to page，是一个寻找Page类型数据结构的头指针的宏。调用to\_struct宏。

* #define le2page(le, member) to\_struct((le), struct Page, member)

to\_struct：根据链表节点地址和成员在类型中的偏移量确定宿主数据结构头指针。调用offsetof宏。

* #define to\_struct(ptr, type, member) ((type \*)((char \*)(ptr) - offsetof(type, member)))

offsetof：计算成员在类型中的偏移量。

* #define offsetof(type, member) ((size\_t)(&((type \*)0)->member))

Intel X86-32 CPU 使用分页管理，每页 4KB，逻辑地址格式如图所示，这种 CPU最多支持 4GB 内存。为了使系统能够使用 64GB 内存，使用物理内存扩展技术，使物理地址长度变为 64 位，页面大小仍为 4KB。试设计逻辑地址格式，使其可以在支持物理内存扩展技术的 X86-32 CPU 上运行。(2018年)



解答：

支持4GB 内存时：

4KB -> 12位，

页面数：4GB/4KB = 1M地址空间 -> 20位

逻辑地址格式为：20位的页号 + 12位的页内偏移

页表项格式位：12个控制位 + 20位物理页

支持64GB 内存时：

页面数：64GB/4KB = 16M地址空间 -> 24位

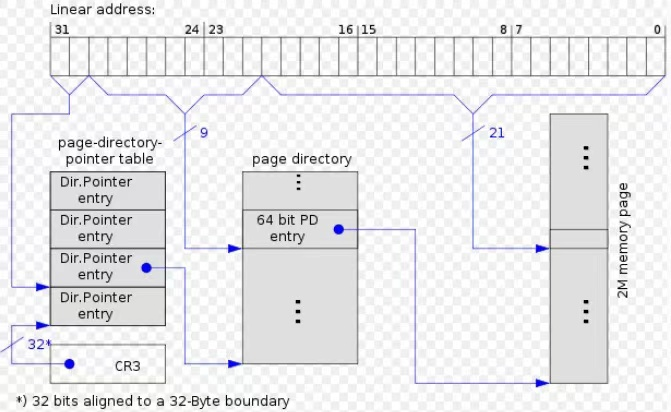
加上12个控制位，得到页表项大小36位。为满足对齐要求，向上扩展取为64位=8B。

则每个页可含 4KB/8B = 512个页表项。

此时，页面offset = 12位，二级页表offset = 9 位，一级页表offset = 9位

此时需要额外设计offset=2位的页目录指针表。

故逻辑地址新格式为32位=2 + 9 + 9 + 12。



更多可参考：<https://blog.csdn.net/trochiluses/article/details/12853027>

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 9 | 9 | 12 |
| 页目录指针表 | 一级页表offset | 二级页表offset | 页面offset |

一道关于 ucore 的题目 (2017年)

1）很多代码 balabala，求 intr

2）flag，turn balabala 填一行代码

一台计算机虚拟空间 8KB，物理空间 4KB，二级页表，页表项 2B,页目录项 1B，页表大小32B，求进程页面大小有多少 b (2017年)

每个页表有 32B/2B = 16个页表项      -> 4位

每个页表有 32B/1B = 32个页目录项     -> 5位

进程地址空间8KB = 2^13B                   -> 13位

则一个页面所占位数：13- 5-4 = 4 位

进程页面大小 ：2^4 ×8b = 128b

**进程和线程**

**判断：**只有⼀个main函数的程序不能有多个线程 (2019年)

错误 ❌

进程是指一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程。线程是进程的一部分，描述指令流执行状态。

所有的进程都是有≥1个的线程 。

**判断：**如果用户强制使用任务管理器 kill 一个进程，那么即使它处于就绪状态/阻塞状态，操作系统也要把它变成运行状态 (2018年)

正确 ✔

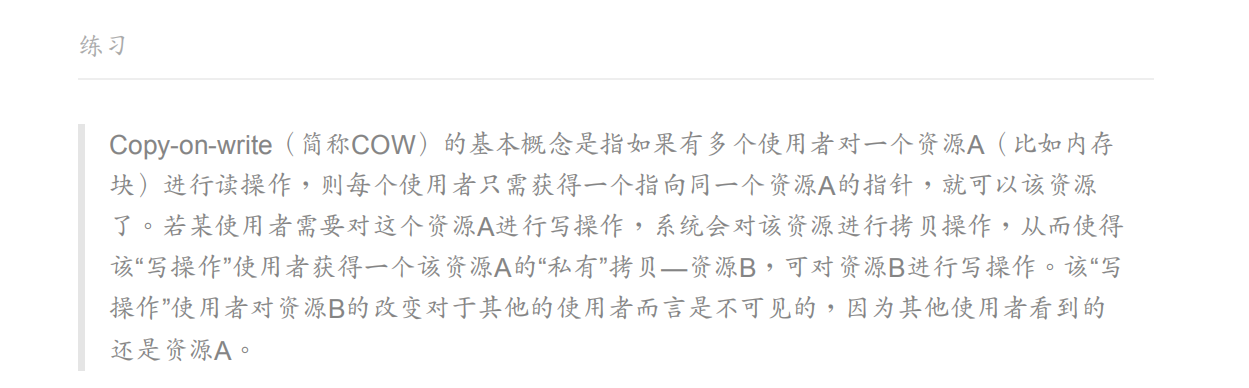
按照进程状态图，所有的进程都必须从运行态进入退出态。kill只是向进程发送一个信号，终止工作由被终止进程自己完成。

**判断：**操作系统采用 copy on write 机制时，fork()函数会复制进程的页目录表 (2018年)

错误 ❌。copy on write机制下不会复制进程的页目录表，而是共享。

copy on write机制下，它们共享页表，只有当对某页发生写操作时，才引发Page Fault单独复制并分配该页给操作进程。

【课程视频14.6】



**判断：**8 位 cpu 不能进行线程切换 (2016年)

错误 ❌。

线程能够切换跟cpu计算机位数没有关系啊。

线程切换是多道程序设计计算机的基本功能。

**填空：**父进程先退出，还未退出的子进程被称为 。子进程退出时，其父进程还没wait，此时子进程被称为 。 (2019年)

答：孤儿进程、僵尸进程

**填空：**父进程退出后，没结束的子进程变成 （ ）。 (2018年)

答：孤儿进程。

exec（）系统调用会改变以下哪些参数（） (2017年)

A.进程 ID

B.父进程 ID

C.文件打开指针

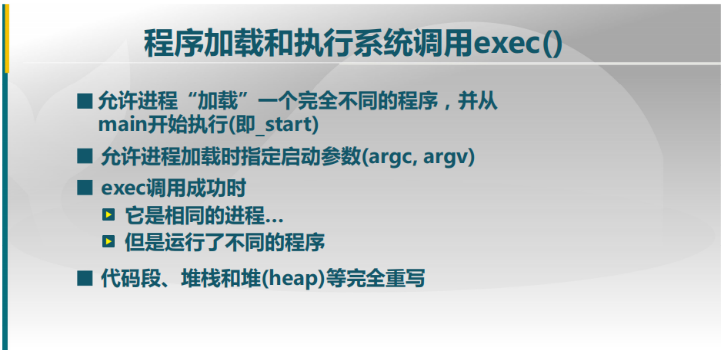
D.？

选C。

exec()系统调用允许进程加载一个完全不同的程序。

exec()调用后进程ID（以及PID）不变，但代码、堆栈、堆、数据都完全被重写。（**相同的进程，不同的程序**）

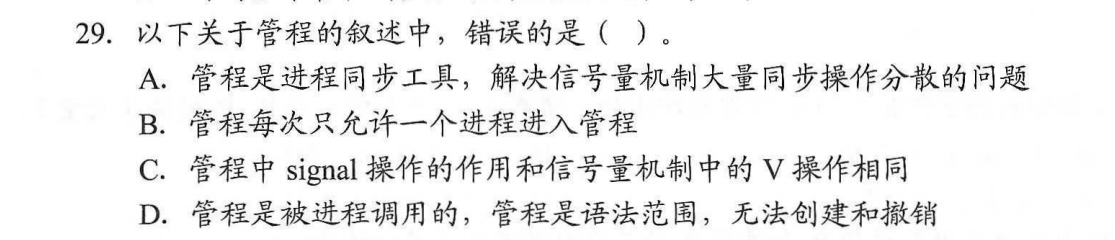


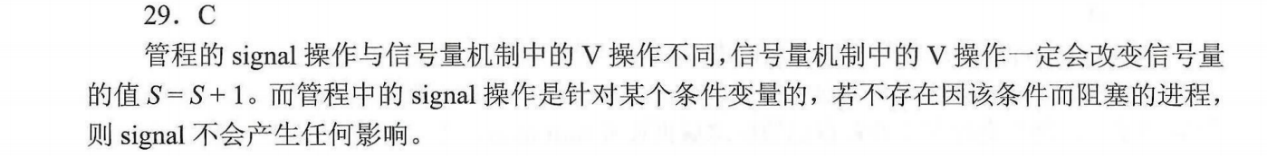


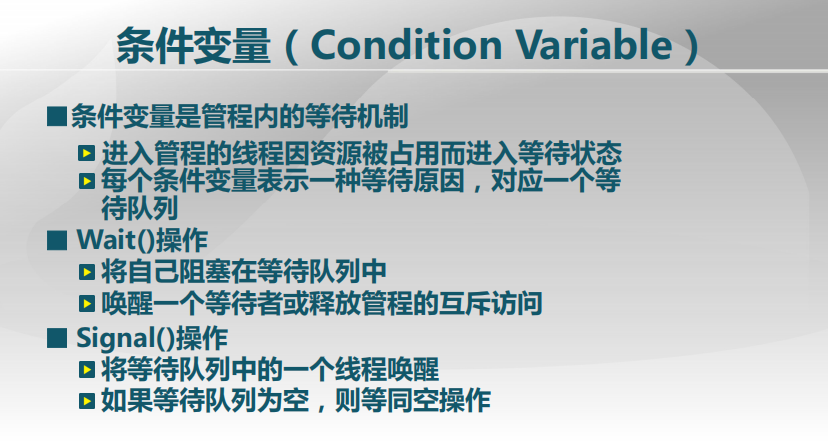
关于线程和管程错误的是（） (2017年)

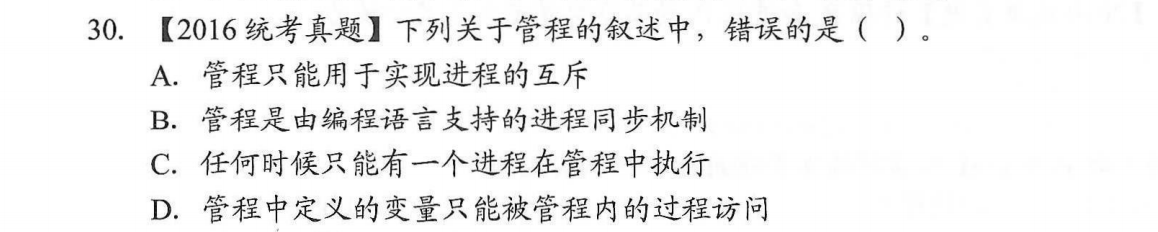
本题题目不全

《王道操作系统》上的有关管程的习题









选A，A表述错误，管程既可以用来实现进程的互斥，还可以实现同步。

多重选择题 (2016年)

程序和进程的区别（   ）

A 程序啥啥啥

B 进程啥啥啥

C 进程是动态啥啥啥

D 程序是动态啥啥啥

**处理机调度**

**死锁**

**判断：**死锁的充分必要条件是互斥，持有并等待 (2020年)

错误 ❌

死锁的必要条件有4个

1. 互斥
2. 非抢占
3. 持有并等待
4. 循环等待

但即使有这4个了，也不能叫充分必要条件，只能叫必要条件，因为系统中即使同时存在这4个条件，也不一定就会产生死锁。

另外缺一都不叫必要条件了。

**判断：**死锁必要条件：互斥访问，占有并等待，非剥夺，循环等待 (2016年)

正确 ✔

**判断：**不安全状态就是死锁状态 (2019年)

错误 ❌。

死锁状态一定是不安全状态，但是不安全状态不一定是死锁状态。



**管程**

**判断：**管程就是一个黑箱子，程序员往里面扔函数，同一时间只有一个函数在执行 (2018年)

正确 ✔

可以这么类比，因为管程每次只允许一个进程进入管程。

**判断：**管程和信号量在功能上等价 (2018年)

正确 ✔

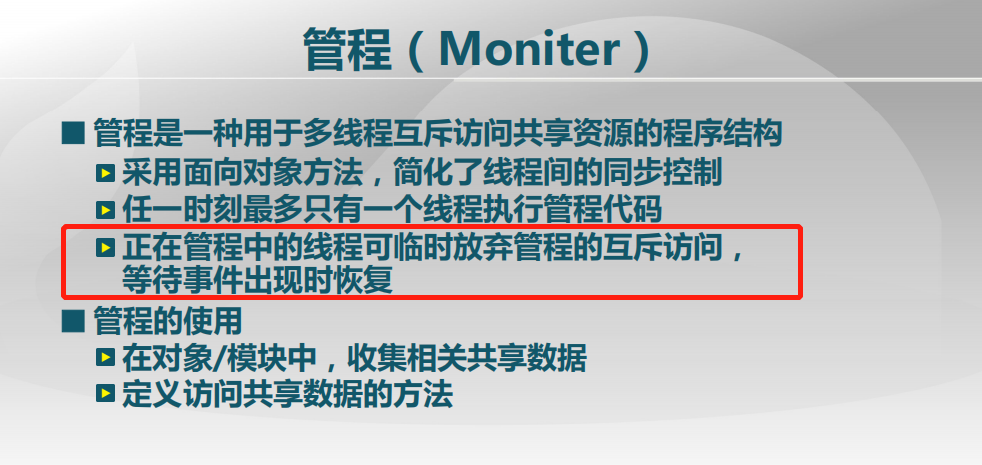
管程和信号量的功能都是实现同步互斥。

**判断：**管程将资源抽象成条件变量，通过变量值的增减来控制进程的访问。 (2018年)

正确 ✔

支持暂时放弃互斥资源访问权，等待信号 (2018年)

答：管程



**同步互斥、信号量**

**判断：**信号量机制可以解决程序死循环的问题 (2020年)

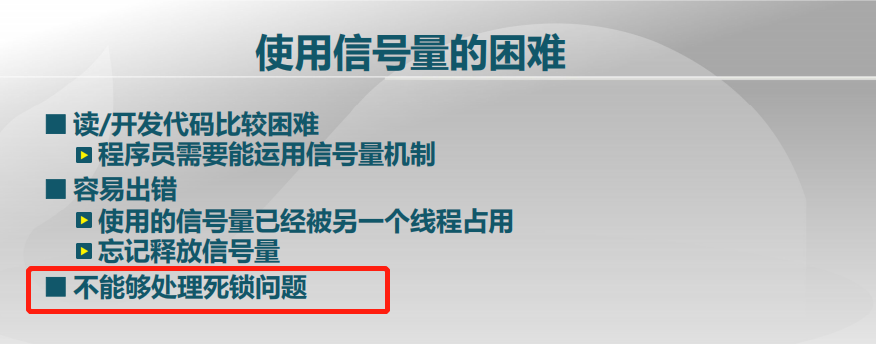
错误 ❌。

这里的死循环即死锁的意思。

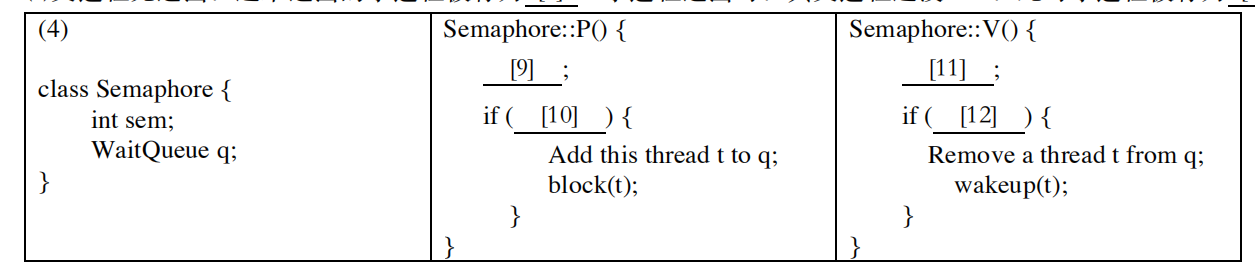
信号量机制是用来解决程序间同步互斥的问题的，不能解决死锁的问题。

要解决死锁，主要采取以下3个措施

* 死锁预防 - 破坏死锁的4个必要条件
* 死锁避免 - 分配资源其检测其是否会可能导致产生死锁 - 如银行家算法
* 死锁检测 - 定期调用死锁检测算法来搜索图中是否存在死锁



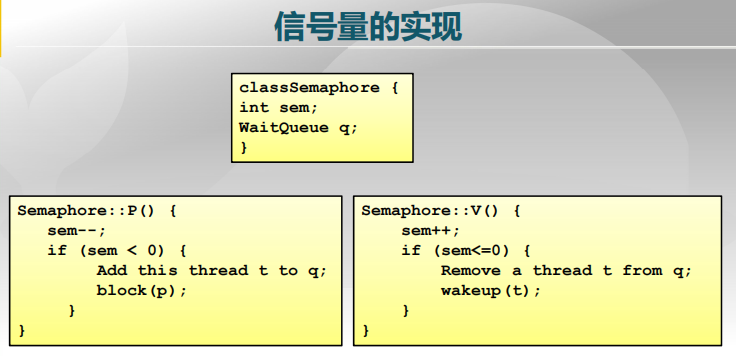
(2019年)



解答：信号量的P操作和V操作

[9] sem--                 [10] sem < 0

[11] sem++          [12] sem <= 0



**哲学家就餐问题** (2018年)

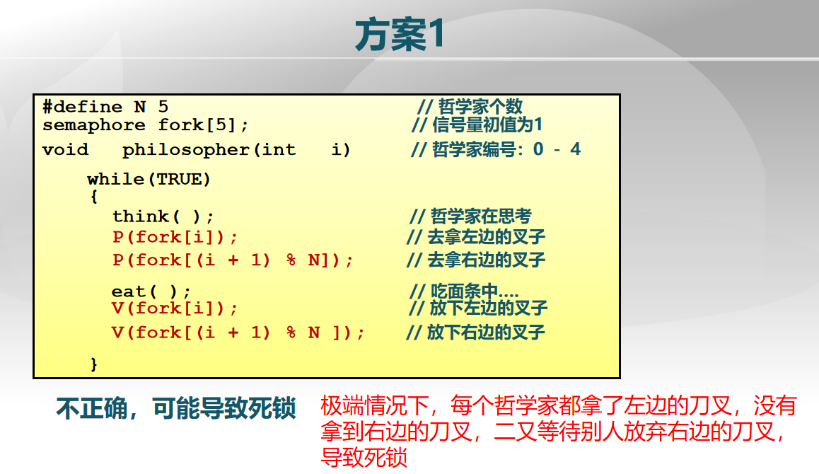


1）上述算法会不会死锁，如果会请举例

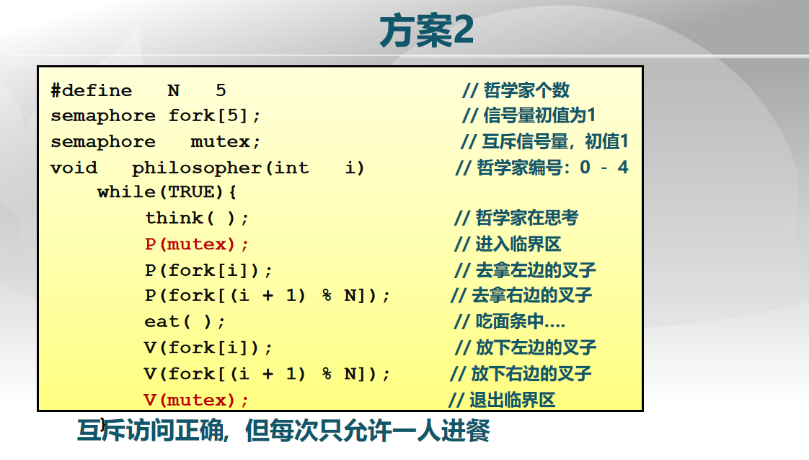
2）算法是否允许两个哲学家同时进餐，若可以请举例

两版回忆，一个贴的方案1，一个贴的方案2。考虑到第二问，应该更可能是方案二，这里都分析一下。

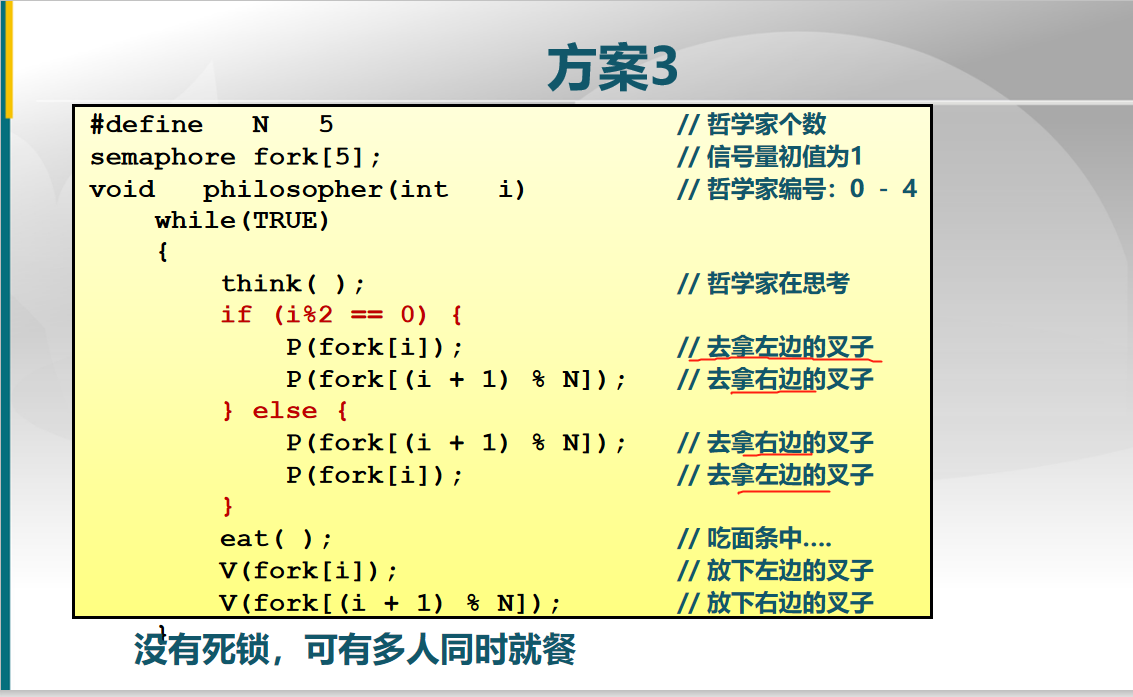
方案1：



方案2：只允许一人进餐，不允许两个哲学家同时进餐。



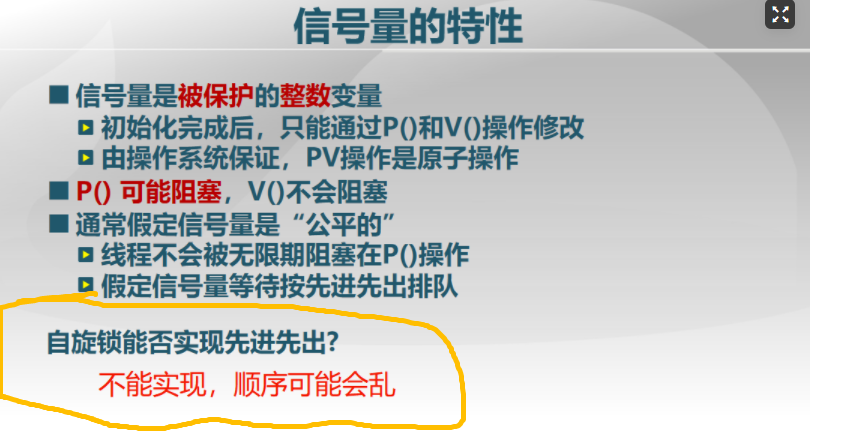
方案3：



**判断：**使用自旋锁不能保证进程按先来后到的顺序使用 cpu 资源 (2018年)

正确 ✔

**自旋锁具有随机性**。自旋锁需要占用CPU随时随地去查，有可能临界区的使用者退出时它刚改完，下一个进入者是谁去查，那它就能进去，如果说运气不好（正好是这个资源变成有效）老是被人抢先查过，就没法按照等待的顺序进行。



**调度算法**

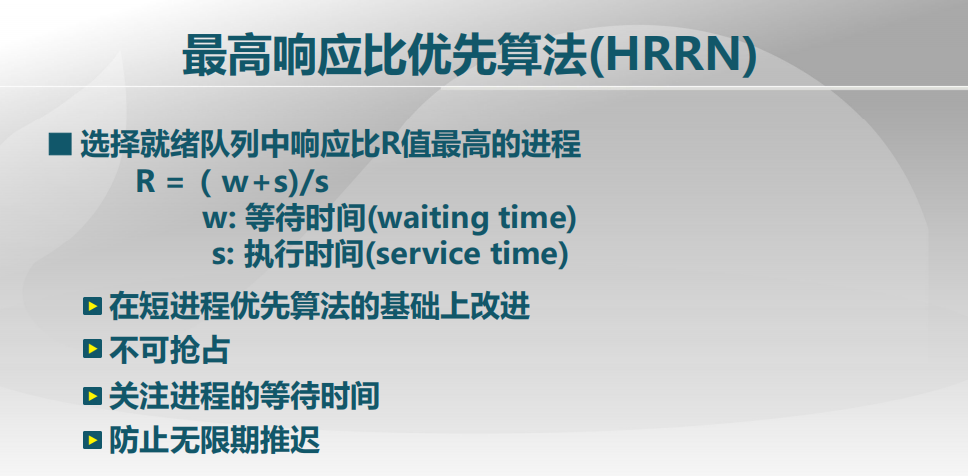
**判断：**SJF调度算法可能出现饥饿现象 (2020年)

正确 ✔

SJF调度算法即是短作业优先调度算法。优先处理短的，会导致长作业饥饿。

高响应比调度算法的分子是 ，分母是 。 (2018年)

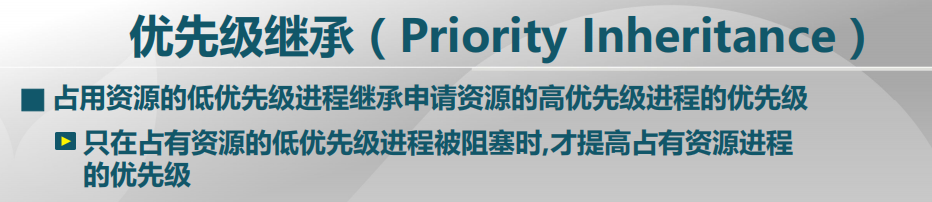
答：等待时间+执行时间 ； 执行时间



优先级反置指的是 抢占了 的资源， 时低优先级进程能动态改变优先级 (2018年)

答：低优先级进程；高优先级进程；优先级继承





**填空：**stride调度算法中，进程分配时间和其优先级成反比，优先级越低进程上CPU运行的时间就会更长，用八位无符号数来表示进程A的运行时间有可能会溢出，但是当步长s \_\_\_[1] \_\_\_127时，只要做小小的技巧就仍可以正确判断时间，然后又引入进程B，a为A的stride，b为B的stride，当sign(a-b) \_\_[2]\_\_\_ 时说明A正在运行，此时a+s \_\_\_[3]\_\_\_ 256，当sign(a+s-b)\_\_\_ [4]\_\_\_\_\_ 0时，轮到B运行。(2019年)

解答：

当步长s \_\_\_<=\_\_\_127时，只要做小小的技巧就仍可以正确判断时间

最大步进S<=有符号最大值=127。

由于要保证sign(a+s-b)的结果正确，其实就是保证a+s-b作为有符号数不会溢出，即a+s-b<=127。又由于sign(a+s-b)只可能出现在A被调度之后，即意味着a<=b，故a+s-b<=s+0=s<=127。

sign(a-b) \_\_<= 0 \_\_\_ 时说明A正在运⾏

sign(a-b)<=0 等价于 a-b<0 等价于 a<b 等价于 A优先

此时a+s \_\_\_\_<\_\_ 256

A正在运⾏时，A的stride还没有溢出。

当sign(a+s-b)\_\_\_\_\_>\_\_\_ 0时，轮到B运⾏。

A经过执行后的理论stride=a+s大于理论b的stride=b，则转为调度B。

* 注意a+s-b的结果整体在计算过程中当作有符号数。比如，sign(254+2-255)=sign(1)>0。

**系统调用**

**简答题**

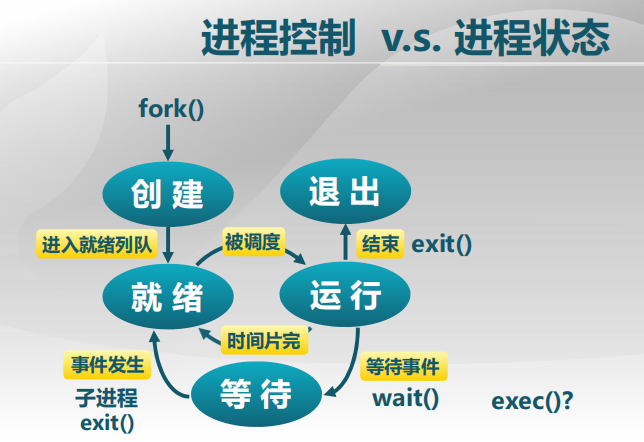
1. 给出系统调用的四个分类，例如xx类完成xx的创建、撤销和退出 （2020）

答：fork()进程创建

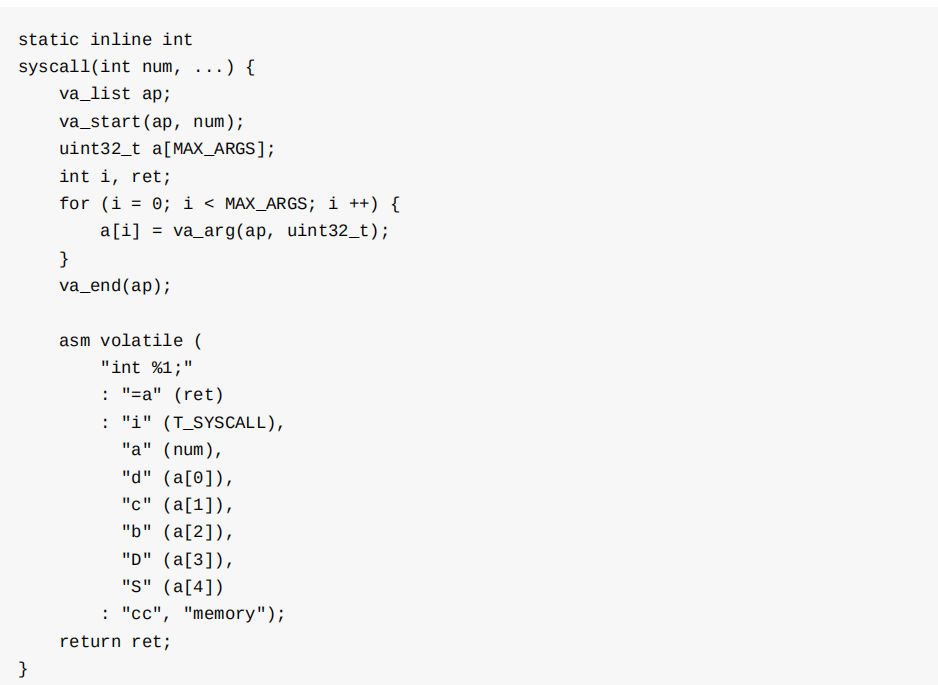
exec()进程执行

wait()用于父进程等待子进程的结束

exit()进程的有序终止



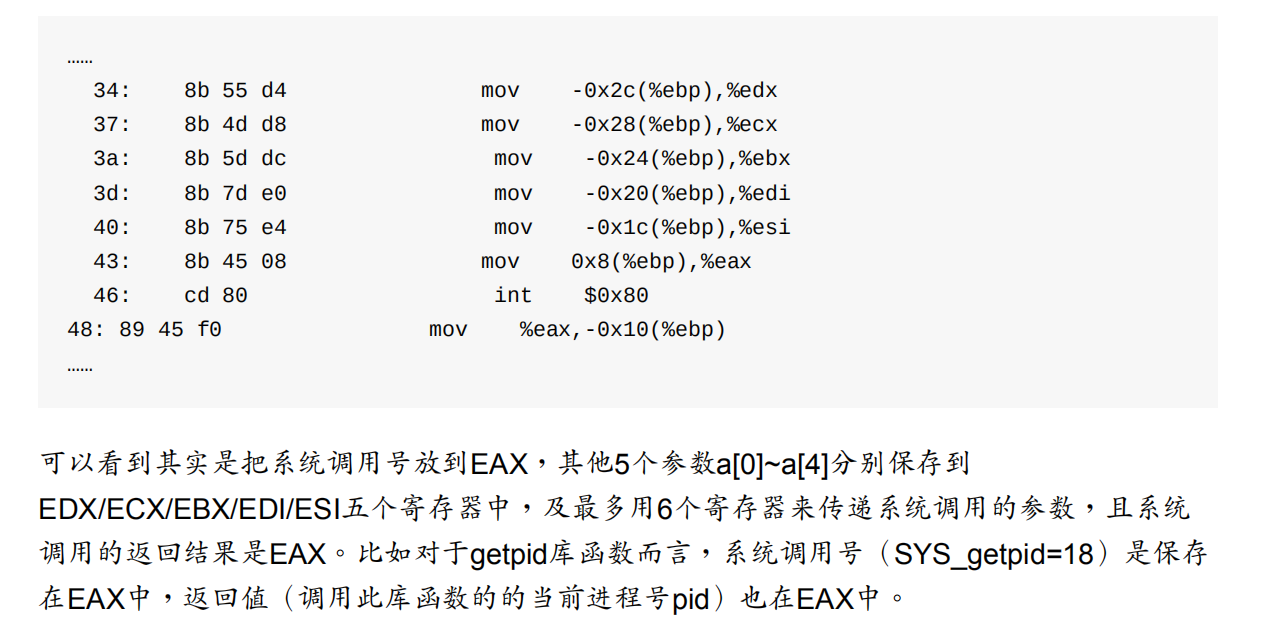
2. 给出getpid系统调用的执行过程



以上系统调用允许的最大参数个数为 ， 其中num是 ，edx和ecx存储的分别是第几个参数，系统调用的返回值存放在\_\_\_\_\_。

答：6个，num是系统调用号，eax存储的是第一个参数，edx存储的是第2个参数，ecx存储的是第3个参数，系统调用返回值存放在eax。

来自 【ucore实验指导书-Lab5-用户进程管理-系统调用实现 P255】



GCC扩展内联汇编的基本格式：



**文件系统**

**判断：**删除一个文件，该文件的所在的当前目录将改变。 (2020年)

错误 ❌

删除一个文件，文件所在的当前目录还在。

**判断：**PCB中的当前工作目录可以加速文件的查找。 (2020年)

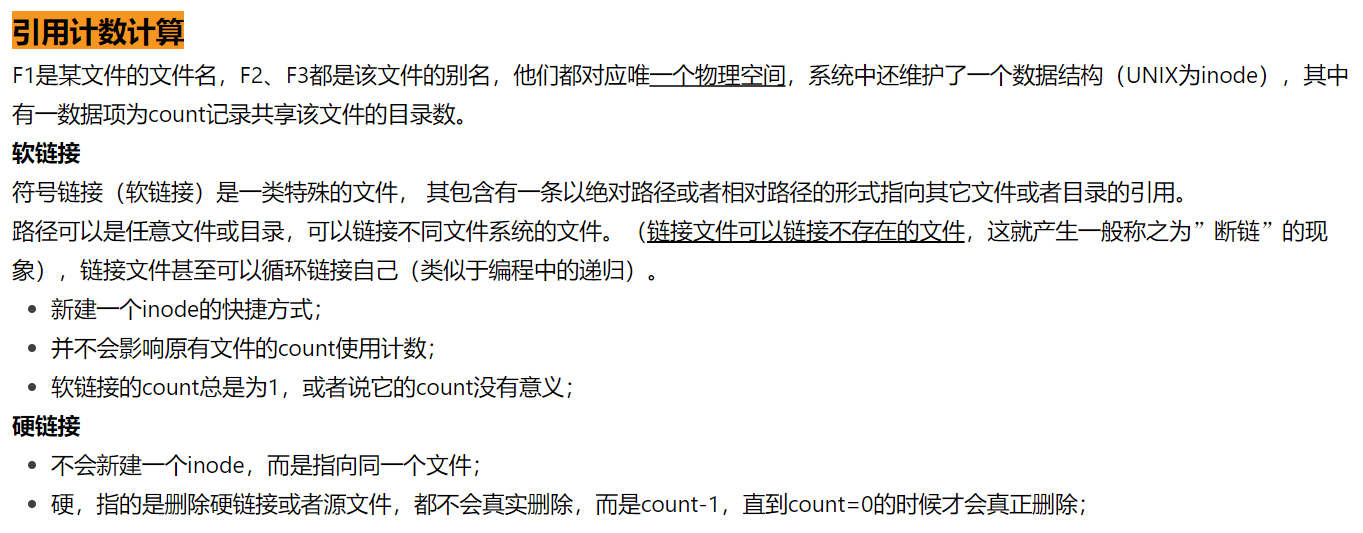
正确 ✔

从当前目录开始查找比从根目录查找快多了。

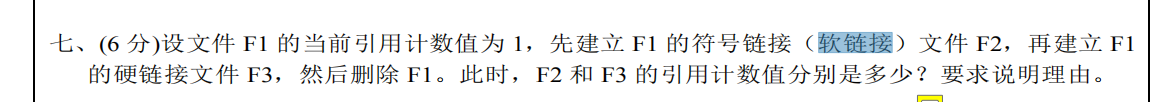
设文件F1的当前引用计数值为1，先建立文件F1的符号链接（软链接）文件F2，在建立文件F1的硬链接F3，然后删除文件F1。此时，文件F2和文件F3的引用计数分别是（ ）。 (2020年)

答：F2的引用计数为1，F3的引用计数也为1

在本题中，当建立F2时，F1和F2的引用计数值都为1。当再建立F3时，F1和F3的引用计数值就都变成了2。当后来删除F1时，F3的引用计数值为2-1=1。F2的引用计数值仍然保持不变。



本题出现在 《清华 操作系统-20120618-期末》



A是计数为1的文件，创建A的硬链接B，再创建A的软连接C，在创建B的硬链D，B计数 [14] , D计数 [15] 。 (2019年)

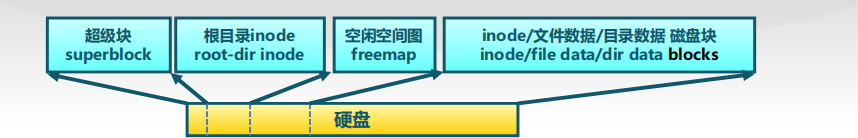
答：B计数为3，D计数为3

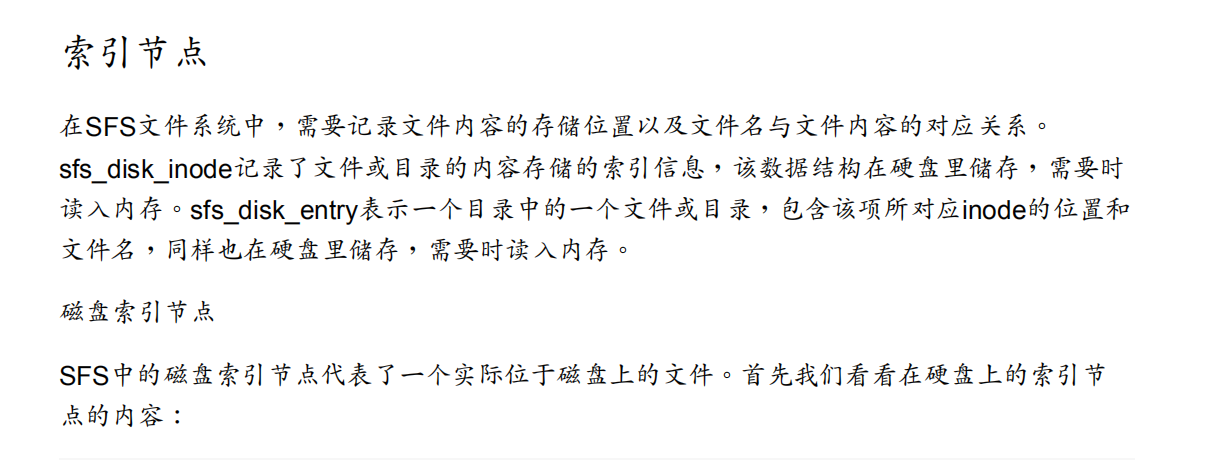
A的计数为3，B计数为3，C计数为1，D计数为3

UNIX索引结构存放的位置是（ ） (2020年)

A. 超级块                 B. 索引节点                         C.                         D.

应该选B。





以下哪种磁盘阵列存取速度快（ ） (2017年)

A.RAID0

B.RAID1

C.RAID4

D.RAID5

选A

组成原理2019年的真题

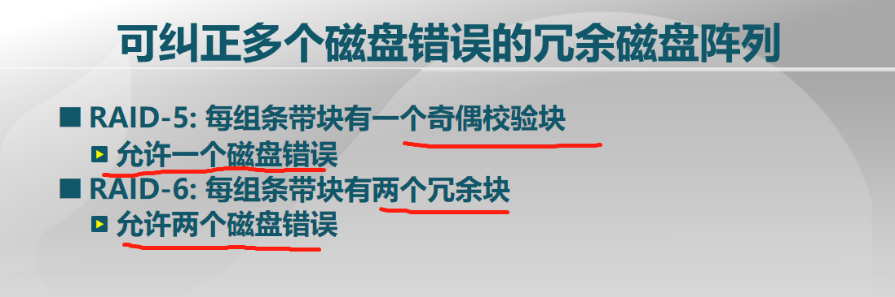
下列哪⼀项没有容错能⼒ A. RAID0     B. RAID1     C. RAID5     D. RAID6

选A ，RAID0只是增加了访问带宽，没有容错能力，而RAID1有一个镜像磁盘，RAID5，RAID6都是带校验的。

组成原理2018年的真题

RAID6坏两个磁盘也可以工作

答：正确 ✔。RAID6每组条带块有两个冗余块，允许两个磁盘错误。



**简答题**

一个文件系统采用索引结点方式存储文件，一个索引结点包括两个直接文件指针，一个一级间接文件指针表（糟糕，忘记是索引表还是指针表了）一个存储块为 8KB，一个指针 4B，问理论上这个文件系统能存放的最大文件是多大？用 TB+GB+MB+KB+B 表示 (2016年)

答：能存放的最大文件大小 = 能寻址到的最大的存储空间 = 数据块数量 × 数据块大小。

索引节点含：2个直接块，1个一级指针。

* 2个文件直接指针 -> 2个直接块：2 × 8KB = 16KB
* 1个一级间接文件指针 ->：8KB / 4B = 2K个直接块
  + 2K个直接块：2K × 8KB = 16MB
* 总计：16MB+16KB

**I/O子系统**

**判断：**延迟写操作可以减少对磁盘的访问次数。 (2020年)

正确 ✔

**判断：**最短寻道时间算法在SSD存储设备中无效。 (2020年)

正确 ✔

最短寻道时间算法是用在磁盘上的。

SSD存储设备俗称固态硬盘。

固态硬盘是用固态电子[存储芯片](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E8%8A%AF%E7%89%87/5596657)阵列制成的硬盘。固态硬盘不用[磁头](https://baike.baidu.com/item/%E7%A3%81%E5%A4%B4)，寻道时间几乎为0