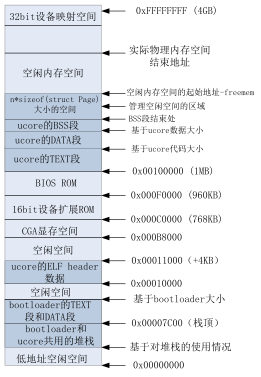
# 操作系统实验二

## 思考的问题及答案

###### 1.如何在建立页表的过程中维护全局段描述符表（GDT）和页表的关系，确保ucore能够在各个时间段上都能正常寻址？

对于GDT进行简单功能描述（通过GDT的index得到段描述符）。在建立页表的时候，此时还是链接时由bootloader生成的GDT.在建立页表完成后，需要更新全局GDT.这才真正开启了分页机制。

###### 2.对于哪些物理内存空间需要建立页映射关系？



可用的物理空间。由此图的内存分布可以看出，我们可用物理空间起始地址是ucore的bss段结束地址上。

###### 3.具体的页映射关系是什么？

拿到一个虚拟地址后，拿出pde index +基地址，得到pte的物理地址。然后该物理地址偏移个pte的index后，可以得到该虚拟页面的物理地址。

当看到这里的时候，我们可以清楚的知道存放在等于pte下索引为index的位置下的值在加上offset的到最终的对应物理地址。

###### 4.页目录表的起始地址设置在哪里？

VPT=0xFAC00000

pde\_t \* const vpd = (pde\_t \*)PGADDR(PDX(VPT), PDX(VPT), 0);

vpd就是页目录表的起始虚拟地址=0xFAFEB000。

###### 5.页表的起始地址设置在哪里，需要多大空间？

页表的理论连续虚拟地址空间0xFAC00000~0xFB000000,大小为4MB。因为这个连

续地址空间的大小为4MB,可有1M个PTE。由于ucore的KERNTOP为3G+896MB,所以最大内核虚地址KERNTOP的页表项虚地址为:vpt+0xF8000000/0x1000\*4=0xFAC00000+0xF8000\*4=0xFAFE0000。

###### 6.如何设置页目录表项的内容？

通过页目录表项，我们可以得到该项页表的物理地址及其访问权限、状态等。

boot\_pgdir[PDX(la)] = PADDR (页表物理地址) | PTE\_P | PTE\_W

###### 7.如何设置页表项的内容？

通过设置页表和对应的页表项，可建立虚拟内存地址和物理内存地址的对应关系

前面练习的get\_pte函数是设置页表项环节中的一个重要步骤。此函数找到一个虚地址

对应的二级页表项的内核虚地址，如果此二级页表项不存在，则分配一个包含此项

的二级页表。

###### 8.出现页访问异常时，硬件执行的工作：

当启动分页机制以后，如果一条指令或数据的虚拟地址所对应的物理页不在内存中或者访问的类型有误（比如写一个只读页或用户态程序访问内核态的数据等），就会发生页错误异常。

而产生页面异常的原因主要有:

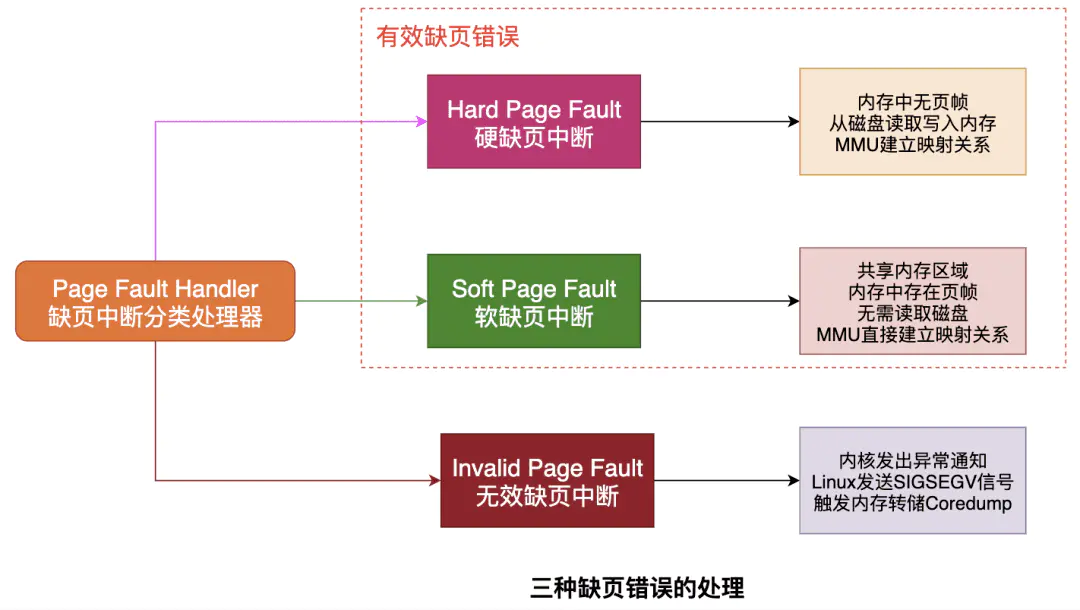
①目标页面不存在（页表项全为0，即该线性地址与物理地址尚未建立映射或者已经撤销）；

②相应的物理页面不在内存中（页表项非空，但Present标志位=0，比如将页表交换到磁盘）；

③访问权限不符合（比如企图写只读页面）。

当出现上面情况之一,那么就会产生页面page fault(#PF)异常。产生异常的虚拟地址（线性地址）存储在CR2中，并且将是page fault的错误类型保存在error code中。引发异常后将外存的数据换到内存中，进行上下文切换，执行中断服务例程，退出中断，返回到中断前的状态。

Linux中对于page fault有详细的分类：



这里说一下控制寄存器CR0-4的作用

CR0的0位是PE位，如果为1则启动保护模式，其余位也有自己的作用

CR1是未定义控制寄存器，留着以后用

CR2是页故障线性地址寄存器，保存最后一次出现页故障的全32位线性地址

CR3是页目录基址寄存器，保存PDT的物理地址

CR4在Pentium系列处理器中才实现，它处理的事务包括诸如何时启用虚拟8086模式等