# 4 Project4的设计及实现

## 4.1 项目设计目的

了解虚拟存储器管理设计原理，掌握请求分页虚拟存储管理的具体实现技术。

## 项目设计要求

1. 段式先将逻辑地址映射成线性地址；
2. 页式将线性地址映射成物理地址；
3. 请求分页机制的实现；
4. 创建页目录PGD和页表PT数据结构；
5. 系统全局页链表g\_pageList,s\_freeList；
6. 初始化页面文件数据结构；
7. 初始化页面文件数据结构；
8. 为页面文件分配与释放磁盘块；
9. 读写页数据函数的实现；
10. 内核缓冲区与用户缓冲区之间的数据交换；
11. 用户级进程在分页系统中的创建，执行与销毁。

## 4.3 项目设计原理

为了实现分页存储系统的地址转换机制，系统增如了个新的寄存器CR3作

为指向当前页目录的指针。这样,从线性地址到物现地址的映射过程为:  
（1）从CR3取得页目录的基地址；  
（2）以线性地址中的页目录位段为下标，在目录中取得相应页表的基地址；

（3）以线性地址中的页表位段为下标，在所得到的页表中取得相应的页面

描述项；  
（4）将页面描述项中给出的页面基地址与线性地址中的页内偏移位段相加得

到物理地址。

上述映射过程可用图只管地表示，具体如下图所示。

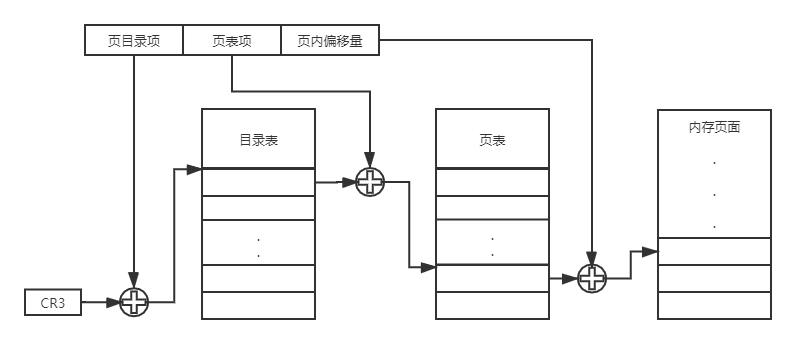


图 4.1 线性地址与物理地址的映射

在GeekOS中，所有内核级进程共享一个页表，而每个用户级进程都有各自的页表。当系统需要运行某个进程时，就把该进程对应的页表调入内存，并使之驻留在内存中，这样就可以运行该用户级线程。此外，用户模式进程的页表也包含访问内核模式内存的入口。GeekOS的内存布局如下图所示。

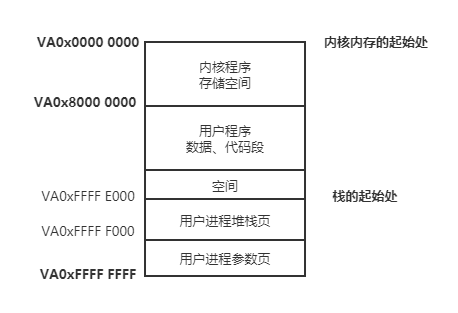


图 4.2 用户进程的线性地址空间

通过编写一个初始化页表和允许在处理器中使用分页模式的函数来为内核级进程创建一个页目录和页表入口，这个函数就是<project4\src\geekos\paging.c>中的Init\_ VM函数。在<paging.c>的Init \_VM的Hints (提示)中，用户可以看到此函数的功能主要有以下三个：

(1)建立内核页目录表和页表；  
(2)调用Enable\_Paging函数使分页机制有效；  
(3)加入一个缺页中断处理程序，并注册其中断号为14。

用户设计的缺页中断处理程序应该能够认识到此页在page file中是存在的，并将其从磁盘读进内存。而当用户将一页从磁盘调入内存时，需要释放这一页占用的空间。

缺页中断处理程序应做的工作如下表所示：

表 4.1缺页处理表

|  |  |  |
| --- | --- | --- |
| 缺页情况 | 标识 | 相应处理 |
| 堆栈生长到新页 | 超出原来分配一页的限制 | 分配一个新页，进程继续 |
| 此页保存在磁盘上 | 数据标识这一页在pagefile中存在 | 从pagefile读入需要的页，进程继续 |
| 因为无限地址缺页 | 非法地址访问 | 终止用户进程 |

## 4.4 项目具体实现

1.在<src/geekos/paging.c>文件中编写代码完成以下函数：

（1） Init\_VM()(defined in )函数将建立一个初始的内存页目录和页表，并且

安装一个页面出错处理程序。

（2） Init\_Paging()函数(定义在src/geekos/paging.c)初始化操作页面调度文件所需的所有数据结构。就如前面说到的，Get\_Paging\_Device()函数指定分页调度文件定位在哪一个设备和占用磁盘块的地址范围。

（3） Find\_Space\_On\_Paging\_File()函数应该在分页调度文件里面找到一个空闲的足够大的页空间。它将返回这个大块的索引，或者当没有合适的空间就返回-1。

（4）Free\_Space\_On\_Paging\_File()函数将释放由Find\_Space\_On\_Paging\_File()函数在分页调度文件里所分配的的磁盘块。

（5）Write\_To\_Paging\_File()函数将把存储在内存的一页数据写出到分页调度文件里。

（6）Read\_From\_Paging\_File()函数将读取分页调度文件里的一页数据到内存空间。

2.在<src/geekos/uservm.c>文件中编写代码完成以下函数：

1. Destroy\_User\_Context()释放进程所占用的所有内存和其它资源。

（2） Load\_User\_Program()装载可执行文件到内存里，创建一个就绪的用户地址空间，功能类似于分段系统的实现。

（3） Copy\_From\_User()从一个用户缓冲区复制数据到一个内核缓冲区。

1. Copy\_To\_User()从一个内核缓冲区复制数据到一个用户缓冲区。

（5）Switch\_To\_Address\_Space()利用它装载相应页目录和LDT（局部描述符表）来切换到一个用户地址空间。

操作系统将需要在磁盘设备上创建一个page file文件暂时保存从内存中替换出去的页，实现一个类LRU（最近最少使用页面）算法在内存中选取一个替换页把它写到磁盘的page file文件中，然后根据表4.1 缺页处理表进行缺页中断处理。

在“/src/geekos/mem.c”文件中，已经定义了一个函数Alloc\_Pageable\_Page实现交换一页到磁盘的操作，具体执行步骤如下：

1.调用mem.c文件中已经实现的Find\_Page\_To\_Page\_Out函数来确定要替换的页（这个函数依赖于页数据结构中的clock域）；

2.调用paging.c文件中已经实现的Find\_Space\_On\_Paging\_File函数在page file中找到空闲的存储空间；

3.调用paging.c文件中已经实现的Write\_To\_Paging\_File函数把被替换的页写到page file文件中；

4.修改页表的相应表项，清除页存在的标志，标识为此页在内存为不存在；

5.修改页表项的页基地址为包含这一页的第一个磁盘块号；

6.修改页表项的kernelInfo位标识为KINFO\_PAGE\_ON\_DISK状态（标识这一页是在磁盘上存在，而不是没有效）；

7.调用lowlevel.asm文件中已经实现的Flush\_TLB来刷新TLB。

## 4.5 调试运行结果

在Ubuntu9.04的终端中编译GeekOS项目0，步骤如下：

1.进入进入geekos-0.3.0/src/project4/build目录；

2. 执行 make depend；

3.执行 make，成功之后在build 目录下生成软盘镜像文件fd.img文件和diskc.img文件；

4.在build目录下启动执行bochs，成功后，运行结果如下：

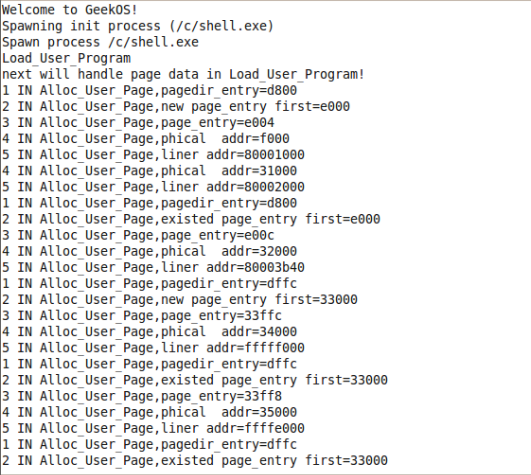


图 4.3 Project4运行截图

由图4.3 Project4运行截图所示，当前系统有进程号为1~5的进程正在运行，进程1打印当前页目录表的入口起始地址，进程2打印当前页表当前起始地址，进程3打印当前所在页表位置，进程4打印物理地址起始地址，进程5打印线性地址，再运行一次进程4打印当前线性地址对应物理地址。

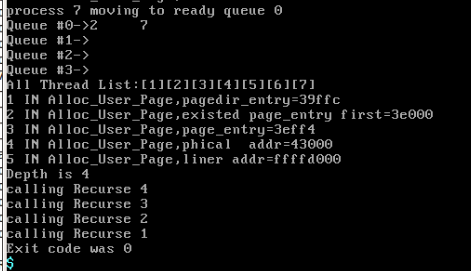


图4.4 递归迭代下的分页存储运行截图

输入命令rec 4后，可以得到如图4.4迭代递归下的 Project4运行截图所示结果。由结果可以看出，当前进程队列下共有7个进程，进程1显示出当前页目录表入口为39ffc，进程2显示已存在页表的入口地址为3e000，进程3显示该线性地址空间下对应查找到的页表下标地址为3eff4，进程4显示对应物理地址为43000，进程5显示当前线性地址为ffffd000；最后输出此次迭代递归的搜索深度为4，在这个过程中一共递归调用过4次进程，分别是调用递归进程4、进程3、进程2，以及进程1，最后结束递归调用，程序运行结束。

## 总结

由于上学期就安装过VMware，而Ubuntu9.04安装包里面也已经有GeekOS-0.3.0，可以直接在主文件夹右键解压即可。但是我们需要构建实验编译和运行环境。编译好的项目需要在bochs下运行，但由于以前没有接触过，所以还是会出现不能正常启动的情况，后来查资料要解决这个问题主要还是要取修改bochs的配置文件。

项目4要做的工作量感觉比较大。首先建立分页调度一定要了解请求分页虚拟存储管理的原理，以及分页系统的地址转换机制，然后才能更好地去掌握请求分页虚拟存储管理地具体实现技术。