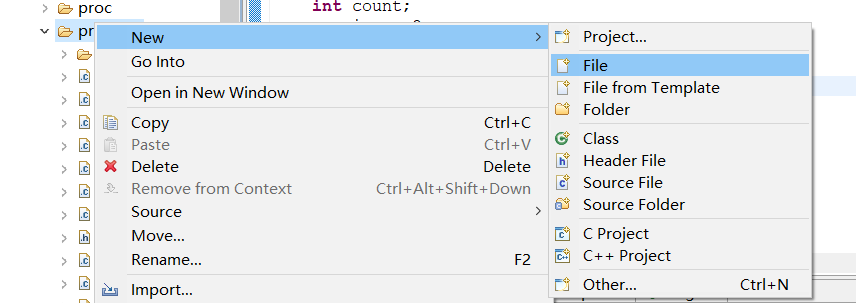
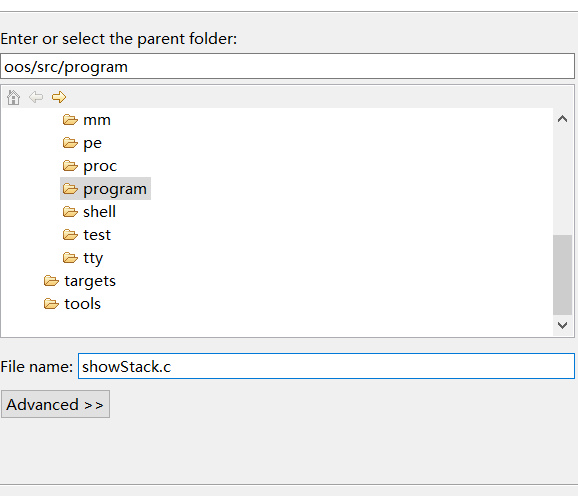
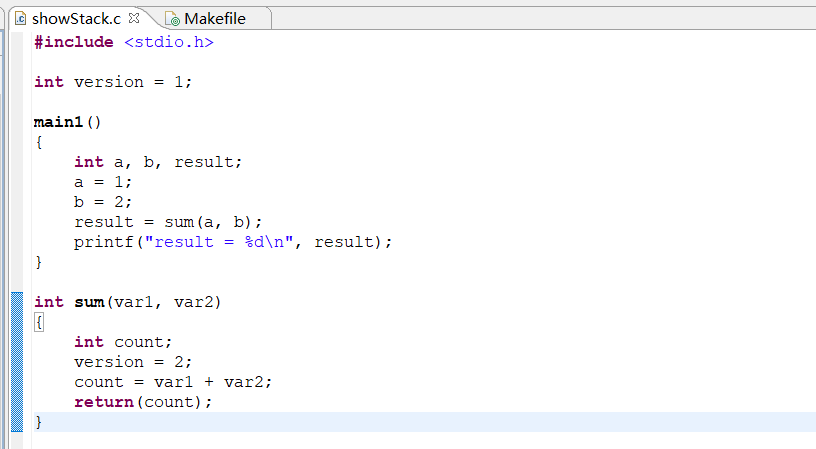
**P03：UNIX V6++完整的进程图像**

**2152118 史君宝**

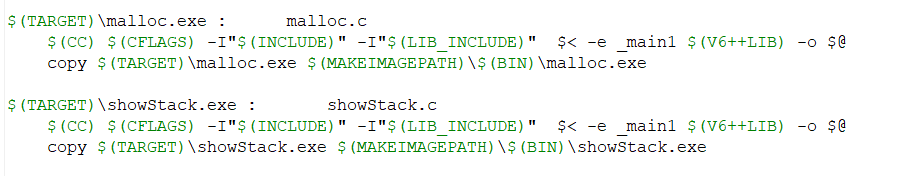
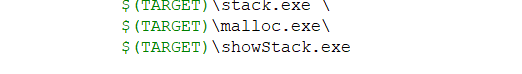
1. **完成实验4.1-4.2，回答问题**
2. 在 program 文件加入一个新的 c 语言文件



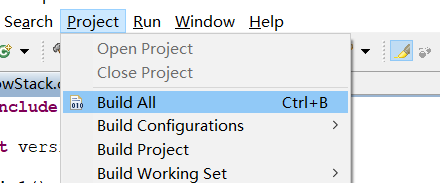


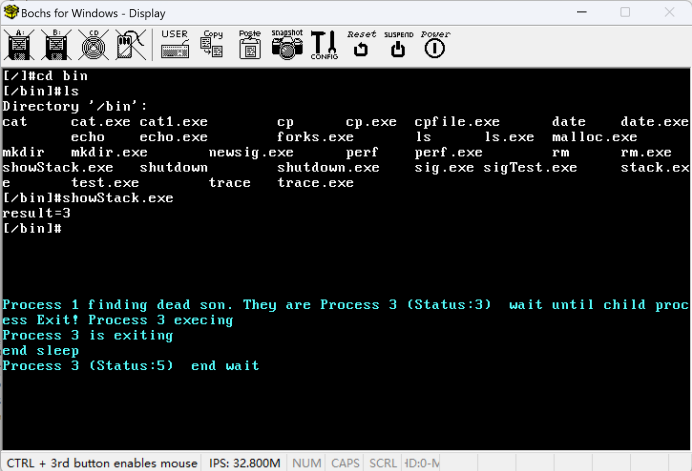


1. 修改编译需要使用的 Makefile 文件



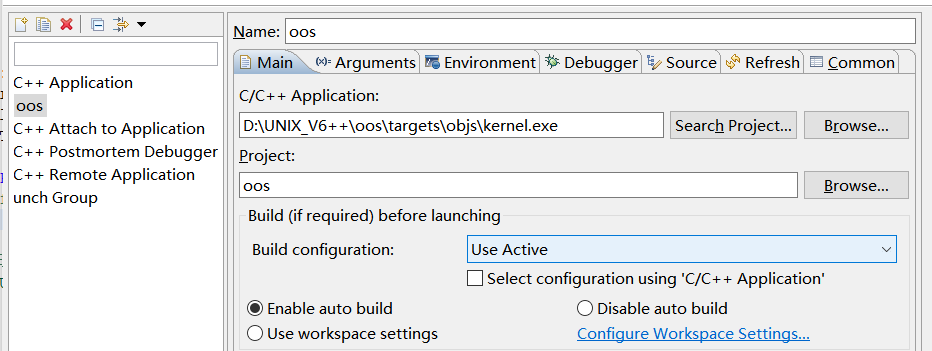
1. 重新编译运行 UNIX V6++代码

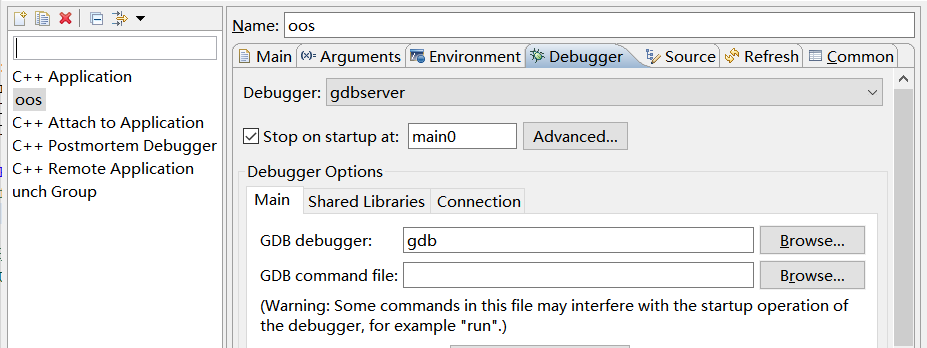




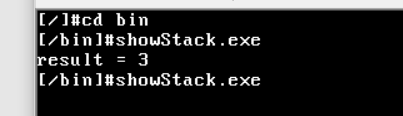
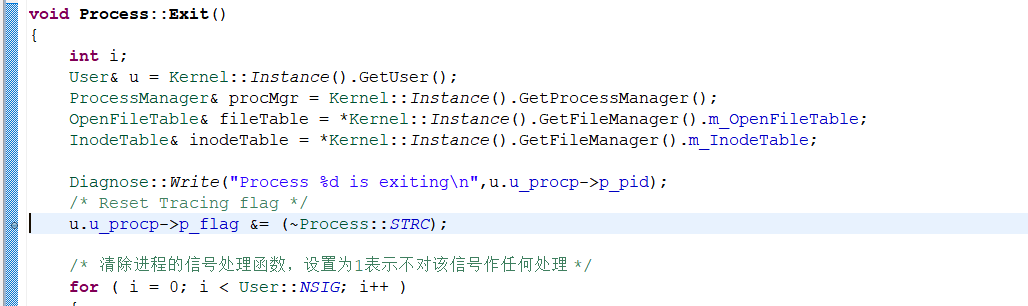
1. 设置调试对象和调试起点

在实验二中我们需要具体观察showStack.c的具体的执行过程，所以调试对象设置的为showStack.exe。而在本实验中我们要观察进程的页表，应将调试对象修改为Kernel.exe。



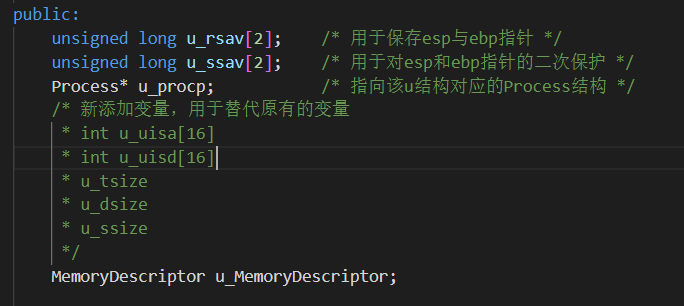


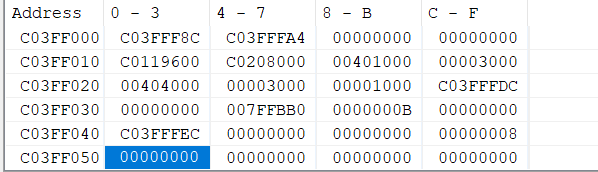
1. 设置断点，并开始调试



1. 观察User的文件，查看信息：

Use.h：

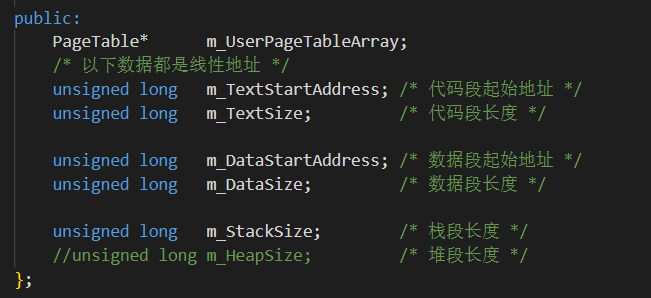




其中0XC03FF000开始的两个四字节是u\_rsav，之后的两个四字节是u\_ssav，0XC0119600是u\_procp。之后便是u\_MemoryDescriptor

我们查找具体的代码，可以知道：

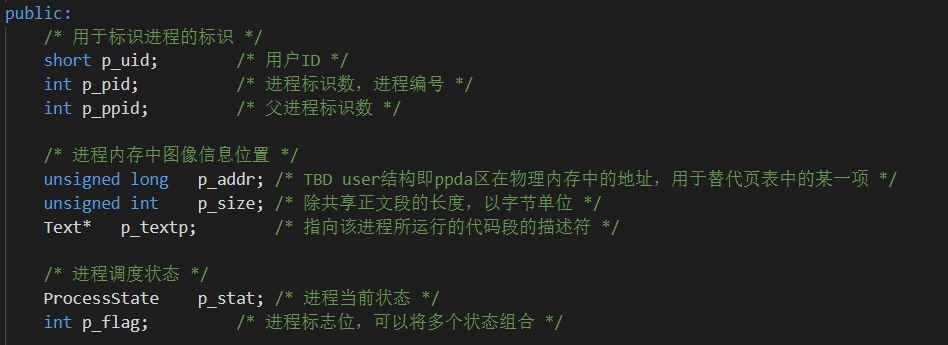
MemoryDescriptor.h：



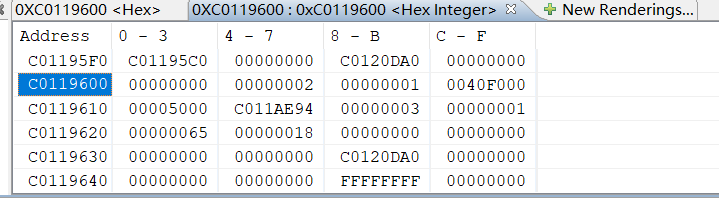
|  |  |  |
| --- | --- | --- |
| u\_rsav | 用于保存esp与eb指针 | 0XC03FFF8C 0XC03FFFA4 |
| u\_ssav | 用于对esp和ebp指针的二次保护 |  |
| u\_procp | 指向process结构 | 0XC0119600 |
| u\_MemoryDescriptor |
| m\_UserPageTableArray | 相对映射段首地址 | 0XC0208000 |
| m\_TextStartAddress | 代码段首地址 | 0x00401000=4M+4K |
| m\_TextSize | 代码段长度 | 0x00003000=12K |
| m\_DataStartAddress | 数据段首地址 | 0x00404000=4M+16K |
| m\_DataSize | 数据段长度 | 0x00003000=12K |
| m\_StackSize | 栈段长度 | 0x00001000=4K |

（7）进程的proc结构，获得信息

Process.h：



对应的Memory信息为：

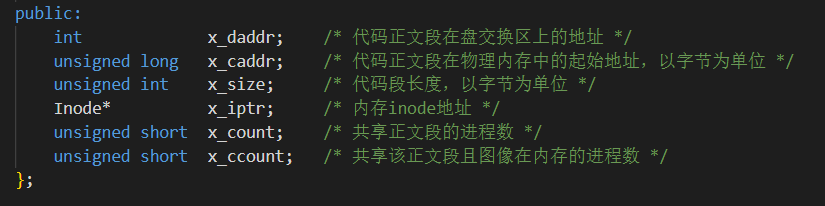


可以知道00000000是用户ID，00000002是进程编号，00000001是父进程标识数，0X0040F000是PPDA区在物理内存之中的地址，0X00005000是p\_size，而0XC011AE94是指向Text表中的Text对象的指针。

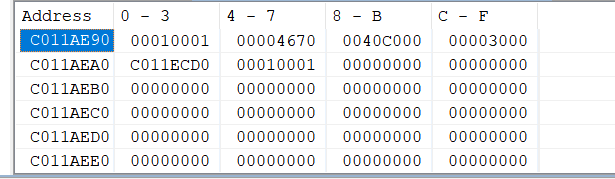
因此我们可以对Process结构的内容画出下面的图：



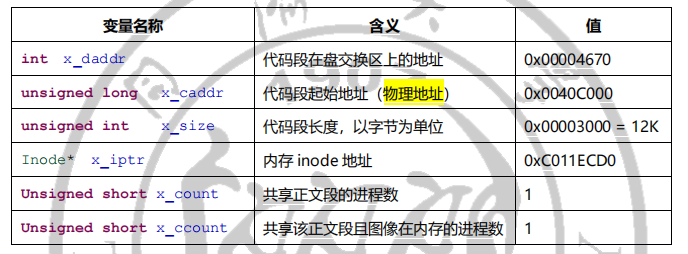
(8)获得Text结构，观察信息：



之后我们根据前面获得的Text表中的Text对象逻辑地址0XC011AE94打开Memory对应位置：

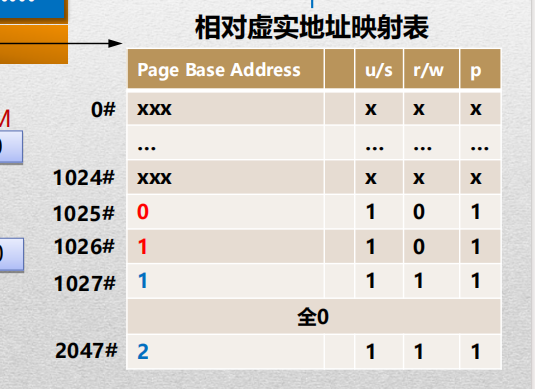


因此可以得到：

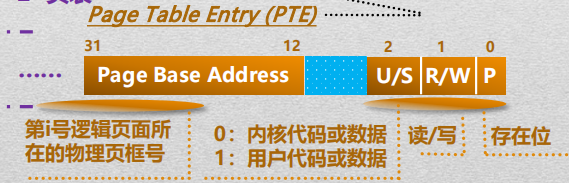


1. 绘制相对虚实地址映射表和物理页表。

我们查询课本的知识，绘制相对虚实地址映射表：



其中可以知道的是：

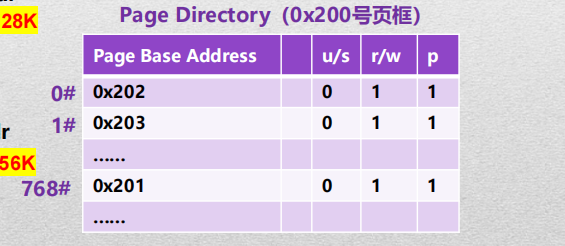


因此可以绘制：

|  |  |  |  |
| --- | --- | --- | --- |
| 页号 | 地址 | 页框号 | 低12位 |
| 0# | 0XC0208000-0XC0208003 | / | / |
| …… | …… | …… | …… |
| 1024# | 0XC0209000-0XC0209003 | / | / |
| 1025# | 0XC0209004-0XC0209007 | 0(0X00000) | 0X005 |
| 1026# | 0XC0209008-0XC020900B | 1(0X00001) | 0X005 |
| 1027# | 0XC020900C-0XC020900F | 2(0X00002) | 0X005 |
| 1028# | 0XC0209010-0XC0209013 | 1(0X00001) | 0X007 |
| 1029# | 0XC0209014-0XC0209017 | 2(0X00002) | 0X007 |
| 1030# | 0XC0209018-0XC020901B | 3(0X00003) | 0X007 |
| 1031# | 0XC020901C-0XC020901F | 0(0X00000) | 0X004 |
| …… | …… | …… | …… |
| 2047# | 0XC0209FFC-0XC0209FFF | 4(0X00004) | 0X007 |

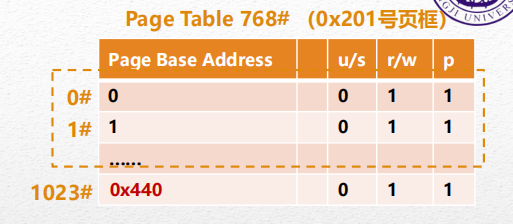
物理页表为：

目录页表0X200#



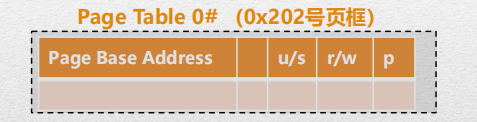
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 页号 | PBA | U/S | R/W | P |
| 0# | 0X202 | 0 | 1 | 1 |
| 1# | 0X203 | 0 | 1 | 1 |
| …… | …… | …… | …… | …… |
| 768# | 0X201 | 0 | 1 | 1 |
| …… | …… | …… | …… | …… |

目录页表0X201#



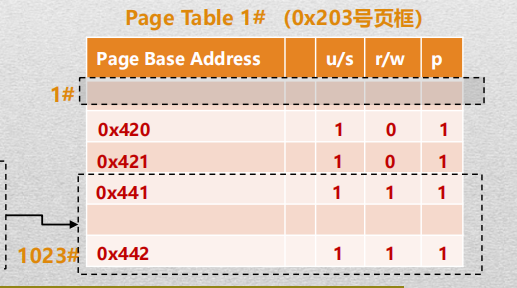
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 页号 | PBA | U/S | R/W | P |
| 0# | 0X000 | 0 | 1 | 1 |
| 1# | 0X001 | 0 | 1 | 1 |
| …… | …… | …… | …… | …… |
| 1023# | 0X40F(p\_addr>>12) | 0 | 1 | 1 |

目录页表0X202#



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 页号 | PBA | U/S | R/W | P |
| / | / | / | / | / |

目录页表0X203#



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 页号 | PBA | U/S | R/W | P |
| 0# | / | / | / | / |
| 1# | 0X40C(x\_caddr>>12) | 1 | 0 | 1 |
| 2# | 0X40D(x\_caddr>>12+1) | 1 | 0 | 1 |
| 3# | 0X40E(x\_caddr>>12+2) | 1 | 0 | 1 |
| 4# | 0X410(p\_addr>>12+1) | 1 | 1 | 1 |
| 5# | 0X411(p\_addr>>12+2) | 1 | 1 | 1 |
| 6# | 0X412(p\_addr>>12+3) | 1 | 1 | 1 |
| …… | …… | …… | …… | …… |
| 1023# | 0X413(p\_addr>>12+4) | 1 | 1 | 1 |

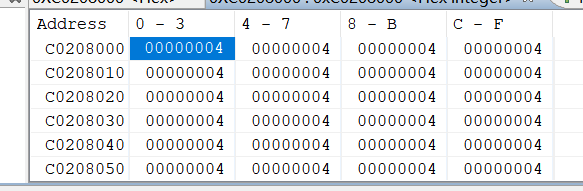
（10）总结：获得进程代码段和PPDA去起始地址的方法。

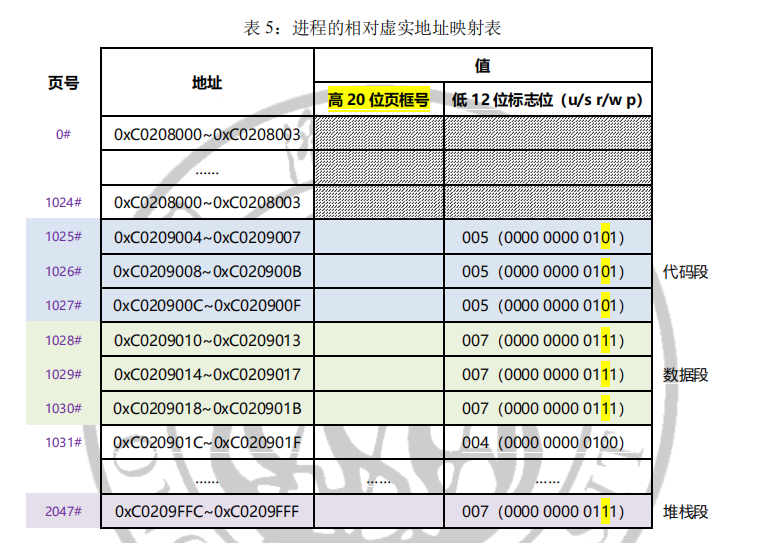
由于进程User结构的逻辑地址固定，恒为0xC03FF000，这也是PPDA区的逻辑地址。我们打开其对应位置Memory，可知proc结构在逻辑空间中的地址，且u\_MemoryDescriptor中的m\_TextStartAddress就是代码段的逻辑地址。我们打开proc的Memory可以从p\_addr中知道PPDA区的物理地址，并从p\_textp可以知道text结构的逻辑地址，打开对应Memory从p\_caddr就可以知道代码段的物理地址了。



1. **完成实验 4.3，获取完整的进程相对虚实地址映射表，补齐表 5。**

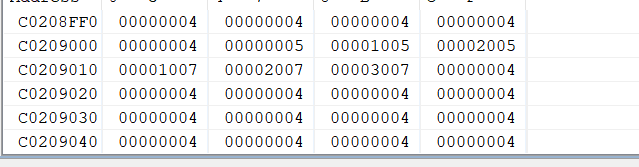
1. 找到对应Memory，补齐表五：





由上可知，在上述表中的0#到1024#的位置都是0X0000 0004

之后我们找到1025# 的位置：



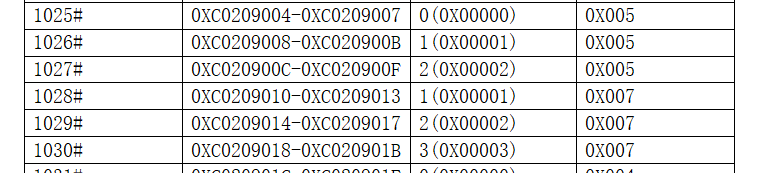
可以看到1024#的位置是0X00000004，之后我们需要补齐1025#-1027#，它们分别是0X00000005，0X00001005，0X00002005；对应的物理页框就是0X00000,0X00001,0X00002：

与所绘制的表是相同的

在1028#-1030#，它们分别是0X00001007，0X00002007，0X00003007；对应的物理页框就是0X00001,0X00002,0X00003：

与所绘制的表是相同的

如图：



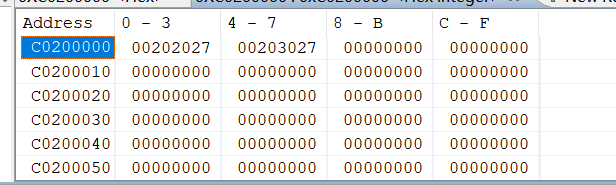
所以完整的相对虚实映射表：

|  |  |  |  |
| --- | --- | --- | --- |
| 页号 | 地址 | 页框号 | 低12位 |
| 0# | 0XC0208000-0XC0208003 | / | / |
| …… | …… | …… | …… |
| 1024# | 0XC0209000-0XC0209003 | / | / |
| 1025# | 0XC0209004-0XC0209007 | 0(0X00000) | 0X005 |
| 1026# | 0XC0209008-0XC020900B | 1(0X00001) | 0X005 |
| 1027# | 0XC020900C-0XC020900F | 2(0X00002) | 0X005 |
| 1028# | 0XC0209010-0XC0209013 | 1(0X00001) | 0X007 |
| 1029# | 0XC0209014-0XC0209017 | 2(0X00002) | 0X007 |
| 1030# | 0XC0209018-0XC020901B | 3(0X00003) | 0X007 |
| 1031# | 0XC020901C-0XC020901F | 0(0X00000) | 0X004 |
| …… | …… | …… | …… |
| 2047# | 0XC0209FFC-0XC0209FFF | 4(0X00004) | 0X007 |

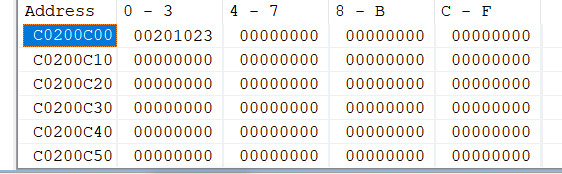
1. **回答四张页表的逻辑地址，并绘制。**

在物理地址上，四张页表占据2M——2M+16k的空间，在逻辑地址上需要平移3G，应该在3G+2M之后的四个页表。

我们在Memory中找到这些页表，可以看到：



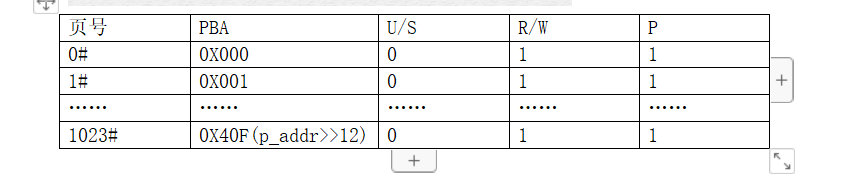
可以知道用户页表0#的页框号是0X202，1#的页框号是0X203



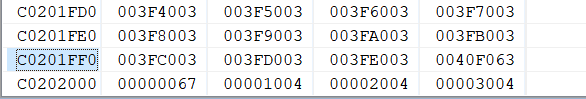
内核页表768#的页框号是0X201。

之后我们依次验证三个页表

0X201：

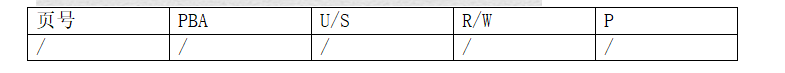


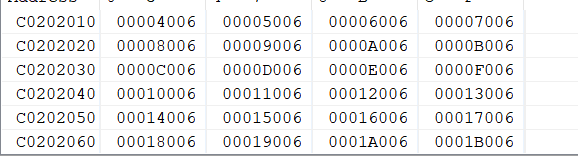
Memory图像为：



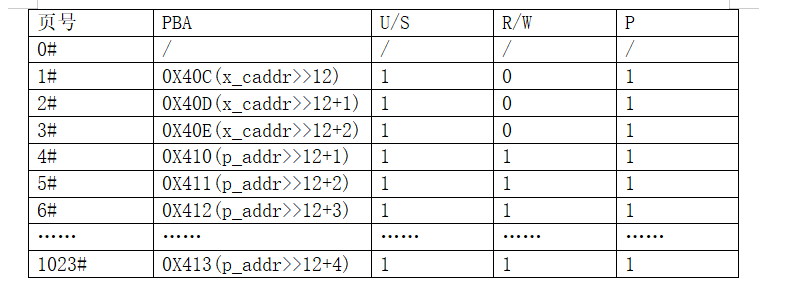
可以看到0X201的最后一项是0X40F。

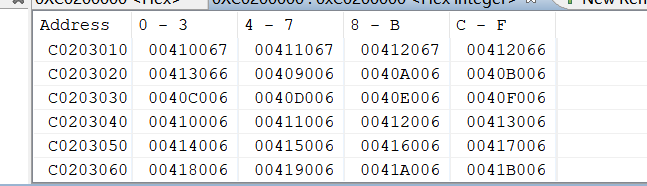
0X202：





0X203：





也是相同的。

1. **内存图像：**



|  |  |
| --- | --- |
| 0-1M | 保留区 |
| 1M-1.5M | 内核 |
| 1.5M-2M | 内核堆对象 |
| 2M-2M+4K | 0X200#物理页面 |
| 2M+4K-2M+8K | 0X201#物理页面 |
| 2M+8K-2M+12K | 0X202#物理页面 |
| 2M+12K-2M+16K | 0X203#物理页面 |
| …… | 其他页表 |
| 4M+48K-4M+60K | 代码段 |
| 4M+60K-4M+64K | PPDA区 |
| 4M+64K-4M+76K | 数据段 |
| 4M+76K-4M+80K | 堆栈段 |
| …… | …… |