同济大学

TONGJI UNIVERSITY

操作系统实验报告

学　　院	电子与信息工程学院
专　　业	计算机科学与技术专业
学生姓名	吕博文
学　　号	2151769
指导教师	方钰
日　　期	2023 年 11 月 11 日

目 录

1 UnixV6++获取进程 User 结构，Proc 结构，Text 结构

1.1 获取进程 User 结构

首先，我们按照 UnixV6++调试内核的步骤将调试的主程序设置为 Kernel.exe，在 Ecilpse 中启动调试。

因为 UnixV6++中 User 结构的逻辑地址固定为：: 0xC03FF000, 所以我们在调试过程中可以首先在 Mermory 列表中该地址附近的值，通过查看 User 类中的变量定义声明可以发现，User 结构中按顺序定义了 u_rsav[2],u_ssav[2],u_procp, 以及结构体类型 u_MemoryDescriptor 等等，通过观察 Memmory 中的每个字节的内容，我们可以得到变量和值的一一对应。

```
Process* u_procp   =  0xC0119600          //Process结构的逻辑地址
MemoryDescriptor u_MemoryDescriptor
PageTable* m_UserPageTableArray = 0xC0208000     //相对地址映射表首地址
unsigned long m_TextStartAddress = 0x00401000;//代码段起始地址
unsigned long m_TextSize = 0x00003000    //代码段长度（12K）
unsigned long m_DataStartAddress = 0x0040400     //数据段起始地址
unsigned long m_DataSize = 0x00003000        //数据段长度(12K)
unsigned long m_StackSize = 0x00001000;      //堆栈长度(4K)
```

0xc03ff000 : 0xC03FF000 <Hex Integer> New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C03FF000	C03FFF8C	C03FFFA4	00000000	00000000
C03FF010	C0119600	C0208000	00401000	00003000
C03FF020	00404000	00003000	00001000	C03FFFDC
C03FF030	00000000	007FFBB0	0000000B	00000000
C03FF040	C03FFFEC	00000000	00000000	00000009
C03FF050	00000000	00000000	00000000	00000000
C03FF060	00000000	00000000	00000000	00000000
C03FF070	00000000	00000000	00000000	00000000

1.2 获取进程 Proc 结构

我们通过获取当前进程的 User 结构中的内容，就获得了一个重要信息 u_procp, 即当前进程 proc 结构的逻辑地址，我们通过在 Memroy 中查看该地址附近的值来查看当前进程 Proc 表中的相关信息。

通过 User 结构，我们得到 u_procp = 0xC0119600, 所以我们得到该进程 proc 块的变量值如下：

```
short p_uid = 0      //用户ID
int p_pid = 2        //进程标识数
int p_ppid = 1       //父进程标识数
unsigned long p_addr = 0x0040F000    //user结构（PPDA区）的物理地址
unsigned ing p_size = 0x00005000     //除共享正文段的长度(20K)
Text* p_textp    = 0xC011AE94     //指向代码段的Text结构的逻辑地址
```

```
ProcessState p_state = 3(SRUN)  //进程调度状态
int p_flag = 1(SLOAD)   //进程标志位
int p_pri = 65  //进程优先数
int p_cpu = 19  //cpu值，用于计算p_pri
int p_nice = 0  //进程优先数微调参数
int p_time = 0  //进程在盘上（内存上）驻留时间
unsigned long p_wchan = 0   //进程睡眠时间
```

0xc0119600 : 0xC0119600 <Hex Integer> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C0119600	00000000	00000002	00000001	0040F000
C0119610	00005000	C011AE94	00000003	00000001
C0119620	00000065	00000019	00000000	00000000
C0119630	00000000	00000000	C0120DA0	00000000
C0119640	00000000	00000000	FFFFFFFF	00000000
C0119650	00000000	00000000	00000000	00000000
C0119660	00000000	00000000	00000000	00000000
C0119670	00000000	00000000	00000000	00000000

1.3 获取进程 Text 结构

我们从当前进程中的 proc 表中得到了指向该进程代码段 text 结构的逻辑地址 p_textp = 0xC011AE94, 所以我们同样可以 Memory 窗口来观察 text 结构的内容，如下所示：

```
int x_daddr = 0x00004670    //代码段在盘交换区上的地址
unsigned long x_caddr = 0x0040C000  //代码段起始地址（物理地址）
unsigned int x_size = 0x0000300     //代码段长度(12K)
Inode* x_iptr   = 0xC011ECD0    //内存inode地址
unsigned short x_count = 1  //共享正文段的进程数
unsigned shrot x_ccount = 1 //共享正文段且图像在内存中的进程数
```

0xc011ae94 : 0xC011AE94 <Hex Integer> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C011AE90	00010001	00004670	0040C000	00003000
C011AEA0	C011ECD0	00010001	00000000	00000000
C011AEB0	00000000	00000000	00000000	00000000
C011AEC0	00000000	00000000	00000000	00000000
C011AED0	00000000	00000000	00000000	00000000
C011AEE0	00000000	00000000	00000000	00000000
C011AEF0	00000000	00000000	00000000	00000000
C011AF00	00000000	00000000	00000000	00000000

1.4 总结

通过上述的实验流程，我们可以得到在 UnixV6++系统中，进程可交换部分 (User 结构首地址), 和代码段的逻辑地址对于每个进程来说都是固定的，可交换部分的逻辑地址为 (3G+4M - 4k = 0xC03FF000), 代码段的逻辑地址为 (4M+4K = 0x00401000); 之后我们通过可交换部分的逻辑地址可以得到指向当前进程的 proc 表的指针 u_procp, 之后我们可以通过当前进程的 proc 表中的内容德奥可交换部分的物理地址:p_addr, 同时通过 p_textp 得到该进程代码段的 text 结构的逻辑地址，并最终通过 text 结构的 x_caddr 得到代码段在内存中物理地址。

2 获取完整的进程相对虚实地址映射表

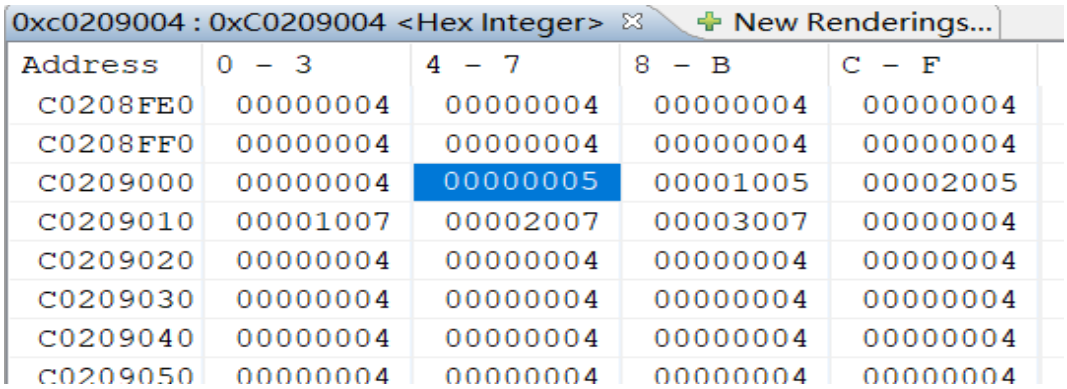
2.1 进程相对虚实地址映射表

我们在上一节的实验中，得到了当前运行进程的虚实地址映射表的逻辑地址 0xC0208000, 所以我们通过观察 Memory 中的值来完善当前进程的相对虚实地址映射表，我们指导，虚实地址映射表大小为 8K，我们需要关注的核心部分是从页号 1025 开始的表项，从该页号开始，以此是一些代码段的页号，数据段的页号，最后一个页表项 2047 是堆栈段的页号 (具体有多少个页号由代码段和数据段的长度有关)

根据我们当前正在调试的进程可知，代码段为 12K，数据段为 12K，所以应该从 1025 号页表项开始连续三项为代码段的页框号和标志位，之后三项为数据段的页框号和标志位。我们观察 Memory 如下：

0xc0209004 : 0xC0209004 <Hex Integer> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C0208FE0	00000004	00000004	00000004	00000004
C0208FF0	00000004	00000004	00000004	00000004
C0209000	00000004	00000005	00001005	00002005
C0209010	00001007	00002007	00003007	00000004
C0209020	00000004	00000004	00000004	00000004
C0209030	00000004	00000004	00000004	00000004
C0209040	00000004	00000004	00000004	00000004
C0209050	00000004	00000004	00000004	00000004



0xc0209004 : 0xC0209004 <Hex Integer> New Renderings...

Address	0 – 3	4 – 7	8 – B	C – F
C0208FE0	00000004	00000004	00000004	00000004
C0208FF0	00000004	00000004	00000004	00000004
C0209000	00000004	00000005	00001005	00002005
C0209010	00001007	00002007	00003007	00000004
C0209020	00000004	00000004	00000004	00000004
C0209030	00000004	00000004	00000004	00000004
C0209040	00000004	00000004	00000004	00000004
C0209050	00000004	00000004	00000004	00000004

可知，三项连续的代码段对应的页框号为 0,1,2，三项连续的数据段对应的页框号为 1,2,3，而最后一个表项对应页框号为 4，为堆栈段的页框号。

所以，根据分析，我们对相对虚实地址映射表补全如下：

地址	高 20 位页框号	低 12 位标志位
0xC020800~0xC0208003	/	/
……	……	……
0xC0208000~0xC0208003	/	/
0xC0209004~0xC0209007	0	005
0xC0209008~0xC020900B	1	005
0xC020900C~0xC020900F	2	005
0xC0209010~0xC0209013	1	007
0xC0209014~0xC0209017	2	007
0xC0209018~0xC020901B	3	007
……	……	……
0xC0209FFC~0xC0209FFF		007

装 订 线

3 现运行进程四张页表查找及分析

3.1 物理页表查找分析

我们通过对于 UnxiV6++进程图像的分析可以得知，页表区位于逻辑地址下的 3G+2M 开始的部分，以 4k 为单位，向下分别为页目录，核心页表（768 项），用户页表 0（0 项），用户页表 1（1 项），我们在 Memory 中可以查看四张页表的内存情况如下：

（1）页目录

Address	0 - 3	4 - 7	8 - B	C - F
C0200000	00202027	00203027	00000000	00000000
C0200010	00000000	00000000	00000000	00000000
C0200020	00000000	00000000	00000000	00000000
C0200030	00000000	00000000	00000000	00000000
C0200040	00000000	00000000	00000000	00000000
C0200050	00000000	00000000	00000000	00000000
C0200060	00000000	00000000	00000000	00000000

0xC0200000 : 0xC0200000 <Hex Integer> New Renderings...

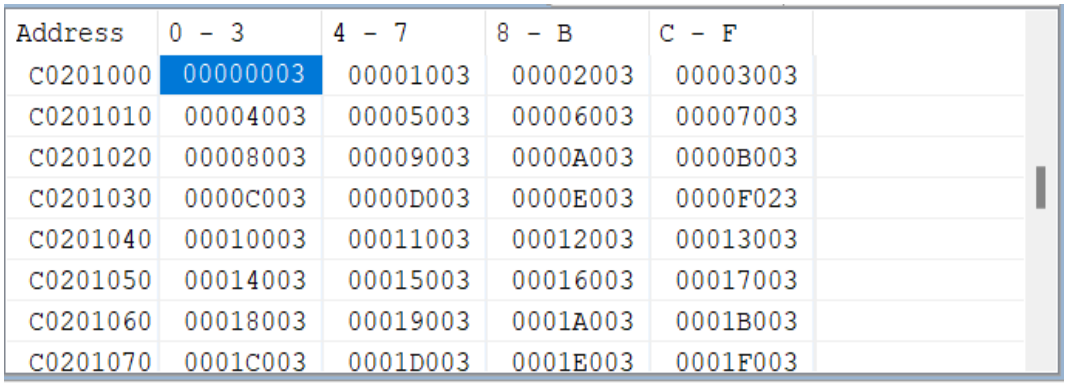
Address	0 - 3	4 - 7	8 - B	C - F
C0200C00	00201023	00000000	00000000	00000000
C0200C10	00000000	00000000	00000000	00000000
C0200C20	00000000	00000000	00000000	00000000
C0200C30	00000000	00000000	00000000	00000000
C0200C40	00000000	00000000	00000000	00000000
C0200C50	00000000	00000000	00000000	00000000
C0200C60	00000000	00000000	00000000	00000000
C0200C70	00000000	00000000	00000000	00000000

地址	高 20 位页框号	低 12 位标志位
0xC0200000~0xC0200003	0x00202	027
0xC0200004~0xC0200007	0x00202	027
……	……	……
0xC0200C00~0xC0200C03	0x00201	023
……	……	……

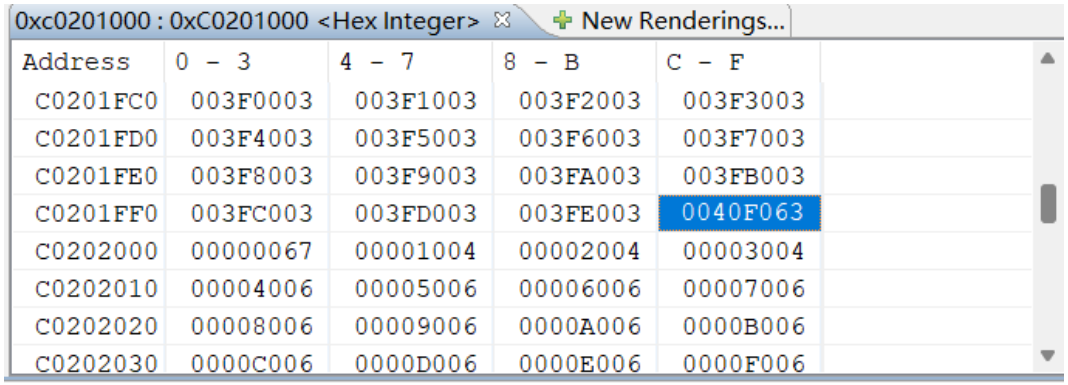
我们通过查看页目录中的内容可知，页目录中对应位置表示二级页表的地址，但通过具体值分

析可知，这个地址仍需要加上 CR3 寄存器中的基地址才能有效表示二级页表的实际物理地址，如图所示，该进程中页目录有效项只有 0，1,768 项，分别指向用户页表 0，用户页表 1，核心页表。

（2）核心页表



Address	0 - 3	4 - 7	8 - B	C - F
C0201000	00000003	00001003	00002003	00003003
C0201010	00004003	00005003	00006003	00007003
C0201020	00008003	00009003	0000A003	0000B003
C0201030	0000C003	0000D003	0000E003	0000F023
C0201040	00010003	00011003	00012003	00013003
C0201050	00014003	00015003	00016003	00017003
C0201060	00018003	00019003	0001A003	0001B003
C0201070	0001C003	0001D003	0001E003	0001F003



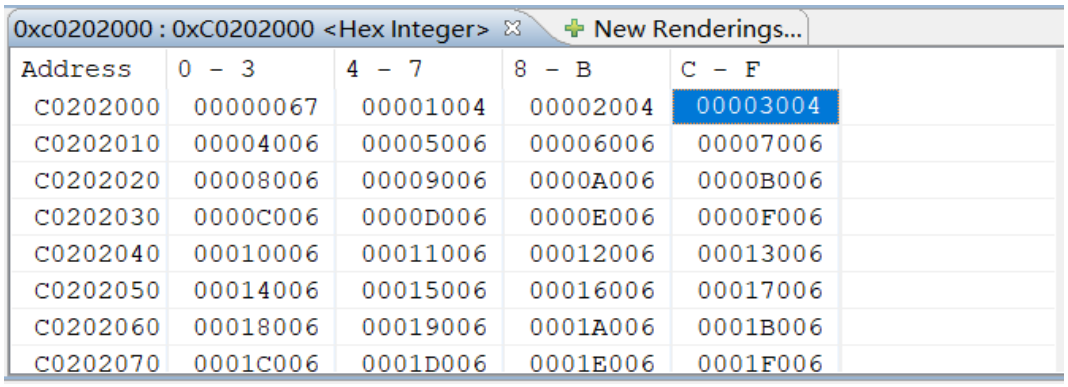
0xc0201000 : 0xC0201000 <Hex Integer> New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C0201FC0	003F0003	003F1003	003F2003	003F3003
C0201FD0	003F4003	003F5003	003F6003	003F7003
C0201FE0	003F8003	003F9003	003FA003	003FB003
C0201FF0	003FC003	003FD003	003FE003	0040F063
C0202000	00000067	00001004	00002004	00003004
C0202010	00004006	00005006	00006006	00007006
C0202020	00008006	00009006	0000A006	0000B006
C0202030	0000C006	0000D006	0000E006	0000F006

地址	高 20 位页框号	低 12 位标志位
0xC0201000~0xC0201003	0x00000	003
0xC0201004~0xC0201007	0x00001	003
.....	.....	.....
0xC020100C~0xC020100F	0x0040F0	063

核心页表中前 1023 项都是内核地址的一一对应，最后一项为 PPDA 区的物理地址。

（3）用户页表 0



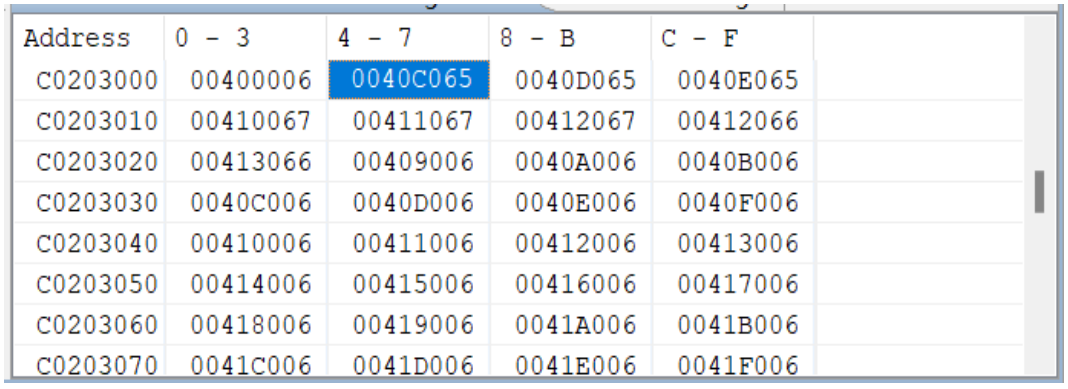
0xc0202000 : 0xC0202000 <Hex Integer> New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
C0202000	00000067	00001004	00002004	00003004
C0202010	00004006	00005006	00006006	00007006
C0202020	00008006	00009006	0000A006	0000B006
C0202030	0000C006	0000D006	0000E006	0000F006
C0202040	00010006	00011006	00012006	00013006
C0202050	00014006	00015006	00016006	00017006
C0202060	00018006	00019006	0001A006	0001B006
C0202070	0001C006	0001D006	0001E006	0001F006

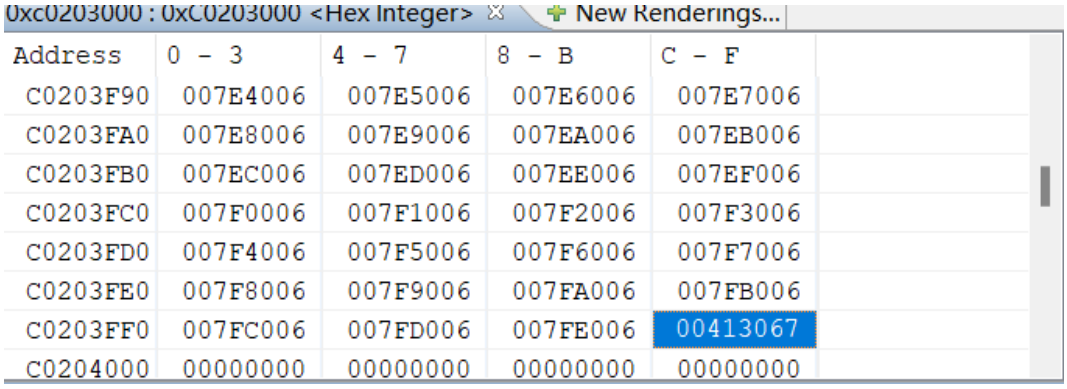
地址	高 20 位页框号	低 12 位标志位
0xC0202000~0xC0202003	/	/
0xC0202004~0xC0202007	/	/
.....	.....	.....
0xC020200C~0xC020200F	/	/

用户页表 0 对应物理地址的 0 到 4M 是编译器的预留空间。

（4）用户页表 1



Address	0 - 3	4 - 7	8 - B	C - F
C0203000	00400006	0040C065	0040D065	0040E065
C0203010	00410067	00411067	00412067	00412066
C0203020	00413066	00409006	0040A006	0040B006
C0203030	0040C006	0040D006	0040E006	0040F006
C0203040	00410006	00411006	00412006	00413006
C0203050	00414006	00415006	00416006	00417006
C0203060	00418006	00419006	0041A006	0041B006
C0203070	0041C006	0041D006	0041E006	0041F006



0xc0203000 : 0xC0203000 <Hex Integer> New Renderings...

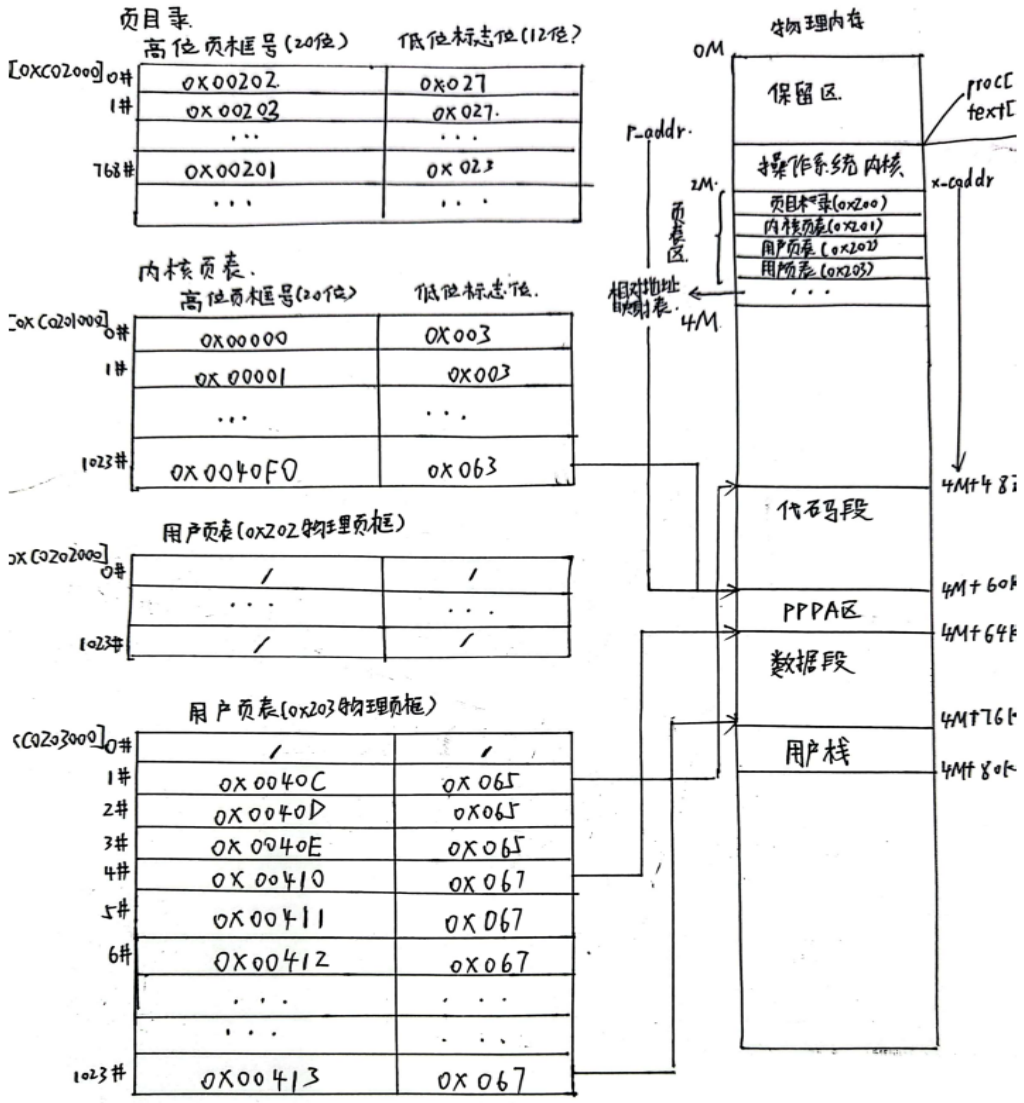
Address	0 - 3	4 - 7	8 - B	C - F
C0203F90	007E4006	007E5006	007E6006	007E7006
C0203FA0	007E8006	007E9006	007EA006	007EB006
C0203FB0	007EC006	007ED006	007EE006	007EF006
C0203FC0	007F0006	007F1006	007F2006	007F3006
C0203FD0	007F4006	007F5006	007F6006	007F7006
C0203FE0	007F8006	007F9006	007FA006	007FB006
C0203FF0	007FC006	007FD006	007FE006	00413067
C0204000	00000000	00000000	00000000	00000000

地址	高 20 位页框号	低 12 位标志位
0xC0203000~0xC0203003	/	/
0xC0203004~0xC0203007	0x0040C	065
0xC0203008~0xC020300B	0x0040D	065
0xC020300C~0xC020300F	0x0040E	065
0xC0203010~0xC0203013	0x00410	067
0xC0203014~0xC0203017	0x00411	067
0xC0203018~0xC020302B	0x00412	067
……	……	……
0xC0203FFC~0xC0203FFF	0x00413	067

用户页表 1 从第二项开始，连续三项为代码段的信息，高位为代码段的地址，地位为标志位，之后连续三项为数据单的信息，高位为地址，地位为标志位，页表最后一项对应用户栈的地址。

4 进程图像的绘制

经过以上实验过程，可以绘制进程图像如下：


页目录
高位页框号(20位)
低位标志位(12位)
[0XC02000]
0#
1#
768#
1023#
0X00202
0X027
0X00203
0X027
0X00201
0X023
内核页表
高位页框号(20位)
低位标志位
[0XC0201000]
0X00000
0X003
0X00001
0X003
0X0040F0
0X063
用户页表(0X202物理页框)
[0XC0202000]
用户页表(0X203物理页框)
[0XC0203000]
0X00040C
0X065
0X00040D
0X065
0X00040E
0X065
0X00410
0X067
0X00411
0X067
0X00412
0X067
0X00413
0X067
物理内存
0M
保留区
r_addr
2M
页目录区
4M
相对地址偏移表
x_caddr
操作系统内核
页目录(0X200)
内核页表(0X201)
用户页表(0X202)
用户页表(0X203)
代码段
PPPA区
数据段
用户栈