实验三：UNIX V6++完整的进程图象

1. 实验目的

结合课程所学知识，通过编写一个简单的 C++代码，并在 UNIX V6++中编译和运行调试。通过查找关键地址单元的值，绘制出整个进程的图象，进而加深对 UNIX 进程图象的理解，特别是对逻辑地址空间与物理地址空间的理解。

2. 实验设备及工具

已配置好 UNIX V6++运行和调试环境的 PC 机一台。

3. 预备知识

（1）UNIX V6++完整进程图象的构成。

（2）关于进程逻辑地址和物理地址的基本概念。

（3）UNIX V6++的运行和调试方法。

4. 实验内容

4.1. 实验准备

开始本实验之前，做好如下的环境准备和知识准备是必须的。

（1）可以直接采用实验二中可以运行的可执行程序 showStack.exe，也可以再重新编写一个简单的小程序，并按照实验二中的方法让它可以在 UNIX V6++中运行起来。

（2）因为本实验要观察进程的页表，属于内核代码部分，因此需将调试目标设置为 Kernel.exe。设置方法在实验二中已经详细介绍，这里不再赘述。

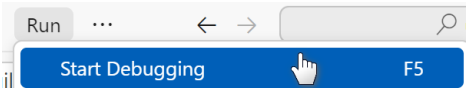
（3）我们在 vscode 环境中调试程序时，看到的所有地址，都是逻辑地址，这一点需要读者理解并牢记。所以，在进行本实验之前，回看一下实验二，理解其中所有的地址值都是程序的逻辑地址将是很有帮助的。

4.2. 找到进程完整的图象

（1）调试运行你的可执行程序

因为需要查看进程图象，所以必须在进程执行过程中让它停下来，建议在 Process::Exit()函数中设置断点，如图 1 所示，此时程序代码已执行完毕，程序马上要结束。

以调试模式启动 UNIX V6++。当 UNIX V6++启动成功，等待调试指令时，点击工具条上的，开始调试。



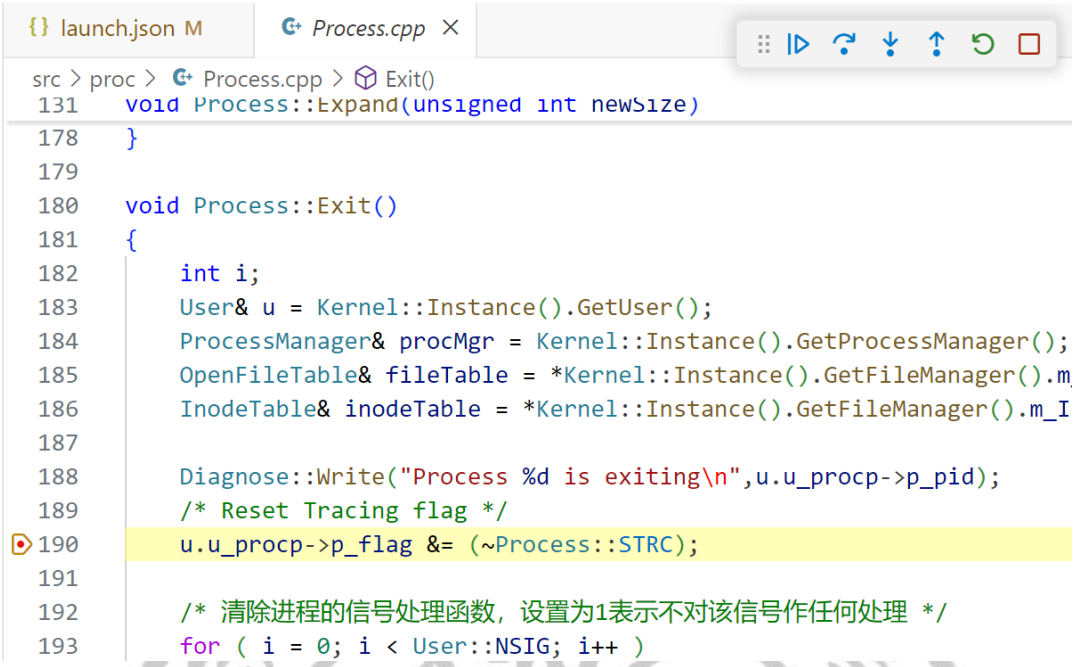


图 1：设置合适的断点

以实验二中的 showStack 程序为例，在键入“cd bin↙”和“showStack.exe↙”后，程序执行完输出语句，停在 Process::Exit()函数中的断点处，如图 2 所示。

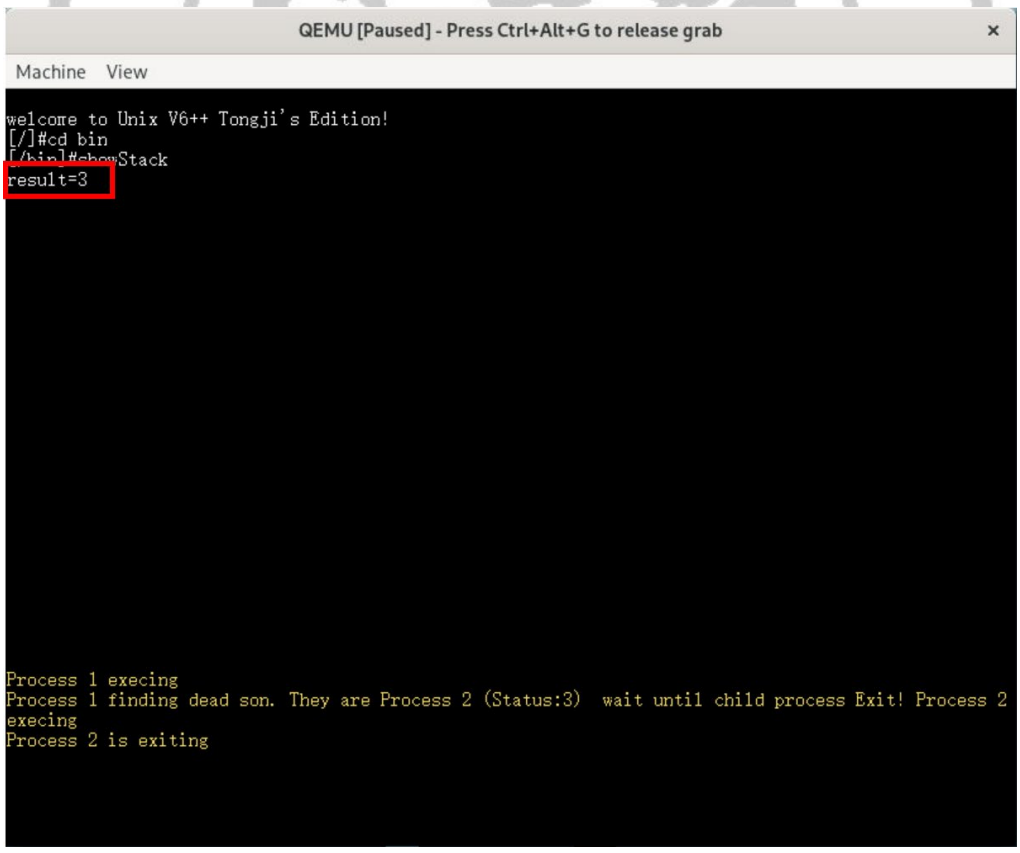


图 2：UNIX V6++停在断点时的状态

(2) 获取进程的 User 结构

我们知道，UNIX V6++的进程 User 结构逻辑地址是固定的，始终位于 3G ~ 3G+4M 部分的最后一页，即：0xC03FF000。这也是为什么 Kernel 中提供的 GetUser 函数通过返回逻辑地址 0xC03FF000 来达到找到当前进程 User 结构的目的（见代码 1）。

```
public:
    static const unsigned long USER_ADDRESS = 0x400000 - 0x1000 + 0xc0000000;/* 0xC03FF000 */

User& Kernel::GetUser()
{
    return *(User*)USER_ADDRESS;
}
```

代码 1

因此，此时利用该逻辑地址，我们可以找到进程的 User 结构。在调试命令窗口中，通过如下命令：

```
-exec x /100xw 0xC03FF000
```

可以看到从该地址开始的全部内容。对应 User 结构的定义，通过与变量窗口中的显示内容的比对（如图 3 所示），就可以确定其全部数据成员的值。如表 1 所示。表 1 中我们只给出了一部分与进程图象相关的地址信息的值。其他部分的值有兴趣的读者可尝试自行整理。

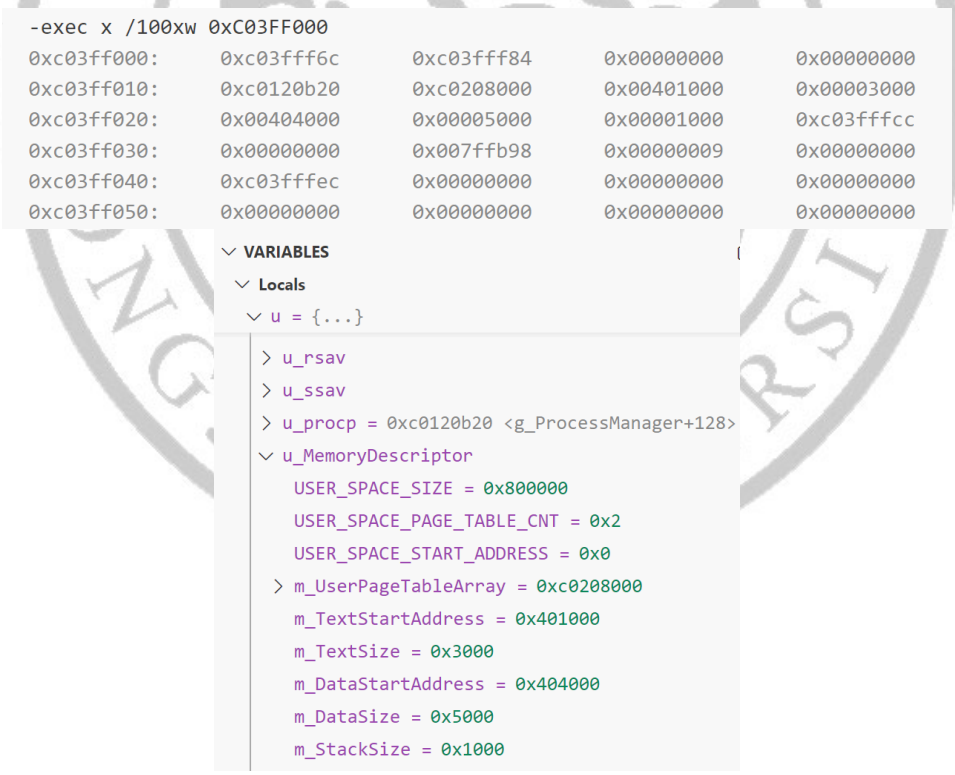


图 3：查看 User 结构的内容

表 1：User 结构各数据成员的值

变量名称	含义	值
Process* u_procp	Proc结构的逻辑地址	0xc0120b20

MemoryDescriptor u_MemoryDescriptor（定义如下，此处均为逻辑地址）			
PageTable*	m_UserPageTableArray	相对映射表首地址	0xC0208000
unsigned long	m_TextStartAddress	代码段起始地址	0x00401000=4M+4K
unsigned long	m_TextSize	代码段长度	0x00003000=12K
unsigned long	m_DataStartAddress	数据段起始地址	0x00404000=4M+16K
unsigned long	m_DataSize	数据段长度	0x00005000=20K
unsigned long	m_StackSize	栈段长度	0x00001000=4K

（3）获取进程的 Proc 结构

表 1 中可以获得的一个重要信息是 u_procp 的值，即进程 proc 结构的逻辑地址：0xc0120b20，使用相同的办法，通过查看该地址起始的内存单元，与变量显示窗口中的内容比对（如图 4 所示），我们可以整理出表 2 中进程 proc 结构中所有数据成员的值。从中我们可以看到进程的 ID 号，父进程的 ID 号，进程的调度状态，进程图象的特征标志等。

这里需要特别强调的是，p_addr 中的值是物理地址，所以这里的 0x00411000 是 User 结构的物理地址，而不是我们前面提到的 User 结构的逻辑地址。

```
-exec x /100xw 0xc0120b20
0xc0120b20 <g_ProcessManager+128>:      0x00000000      0x00000002      0x00000001      0x00411000
0xc0120b30 <g_ProcessManager+144>:      0x00007000      0xc01223b4      0x00000003      0x00000001
0xc0120b40 <g_ProcessManager+160>:      0x00000065      0x0000001d      0x00000000      0x00000000
0xc0120b50 <g_ProcessManager+176>:      0x00000000      0x00000000      0xc0125a80      0x00000000
```

```
u_procp = 0xc0120b20 <g_ProcessManager+128>
  p_uid = 0x0
  p_pid = 0x2
  p_ppid = 0x1
  p_addr = 0x411000
  p_size = 0x7000
> p_textp = 0xc01223b4 <g_ProcessManager+6420>
  p_stat = Process::SRUN
  p_flag = 0x1
  p_pri = 0x65
  p_cpu = 0x1d
  p_nice = 0x0
  p_time = 0x0
  p_wchan = 0x0
  p_sig = 0x0
> p_ttyp = 0xc0125a80 <g_TTy>
  p_sigmap = 0x0
```

图 4：查看 Proc 结构的内容

表 2：Proc 结构各数据成员的值

变量名称	含义	值
short p_uid	用户 ID	0
int p_pid	进程标识数	2
int p_ppid	父进程标识数	1

unsigned long p_addr	user 结构即 ppda 区的物理地址	0x00411000
unsigned int p_size	除共享正文段的长度，以字节单位	0x00007000=28K
Text* p_textp	指向代码段 Text 结构的逻辑地址	0xc01223b4
ProcessState p_stat	进程调度状态	3=SRUN
int p_flag	进程标志位	1=SLOAD
int p_pri	进程优先数	65
int p_cpu	cpu 值，用于计算 p_pri	1d
int p_nice	进程优先数微调参数	0
int p_time	进程在盘上(内存内)驻留时间	0
unsigned long p_wchan	进程睡眠原因	0

(4) 获取进程代码段的 Text 结构

从表 2 中，可以得到的另一个重要的地址信息是 p_textp 指针的值，注意，这里也是该进程代码段 Text 结构的逻辑地址 0xc01223b4。由此，通过查看该地址起始的内存单元，与变量显示窗口中的内容比对（如图 5 所示），我们可以获取到该 Text 结构的值如表 3 所示。这里同样需要强调的是，x_caddr 中是代码段在内存的物理地址，所以这里是 0x0040e000 是代码段的物理地址，而不是我们前面 User 结构中的 u_Memory- Descriptor. m_TextStartAddress 中记录的 0x00401000。此外，x_count 与 x_ccount 由于是 short 类型，所以分别只占用了 2 个字节。

```
-exec x /100xw 0xc01223b4
0xc01223b4 <g_ProcessManager+6420>:     0x00004738      0x0040e000      0x00003000      0xc011ad9c
0xc01223c4 <g_ProcessManager+6436>:     0x00010001      0x00000000      0x00000000      0x00000000
```

```
v p_textp = 0xc01223b4 <g_ProcessManager+6420>
    x_daddr = 0x4738
    x_caddr = 0x40e000
    x_size = 0x3000
  > x_iptr = 0xc011ad9c <g_InodeTable+380>
    x_count = 0x1
    x_ccount = 0x1
```

图 5：查看 Text 结构的内容

表 3：Text 结构各数据成员的值

变量名称	含义	值
int x_daddr	代码段在盘交换区上的地址	0x00004738
unsigned long x_caddr	代码段起始地址（物理地址）	0x0040e000
unsigned int x_size	代码段长度，以字节为单位	0x00003000 = 12K
Inode* x_iptr	内存 inode 地址	0xC011ad9c
Unsigned short x_count	共享正文段的进程数	1

Unsigned short x_ccount	共享该正文段且图像在内存的进程数	1

至此，我们找到了进程图象的两大部分：进程图象的可交换部分和代码段在逻辑地址空 间和物理地址空间的分布，见表 4。由此，可帮助读者更深刻体会通过 GetUser 函数可以获取现运行进程完整图像的过程。由表 4，读者可根据课程所学知识，绘制出进程的相对 虚实地址映射表和物理页表，并通过后续实验，验证是否正确。

表 4：进程图象完整信息

名称	逻辑地址	物理地址	大小
代码段	0x00401000	0x0040e000	12K
可交换部分	0xC03FF000	0x00411000	28K
PPDA 区	0xC03FF000	0x00411000	4K
数据段	0x00404000	0x00412000	20K
堆栈段		0x00417000	4K

4.3. 找到进程的相对虚实地址映射表

通过表 1 中的数据我们知道，进程的相对虚实地址映射表位于 0xC0208000 起始的两个 4K 的页中，注意，这里 0xC0208000 也是逻辑地址。通过查看该地址起始的内存单元，我们将进程相对虚实地址映射表绘制在表 5 中，其中高 20 位页框号（除阴影部分外）请读者补全，并与你在实验 4.2 中绘制的相对虚实地址映射表相比对，看看是否正确。

表 5：进程的相对虚实地址映射表

页号	地址	值		
		高 20 位页框号	低 12 位标志位（u/s r/w p）	
0#	0xC0208000~0xC0208003			
	……			
1024#	0xC0208000~0xC0208003			
1025#	0xC0209004~0xC0209007		FFD（1111 1111 1101）	
1026#	0xC0209008~0xC020900B		FFD（1111 1111 1101）	代码段
1027#	0xC020900C~0xC020900F		FFD（1111 1111 1101）	
1028#	0xC0209010~0xC0209013		FFF（1111 1111 1111）	
1029#	0xC0209014~0xC0209017		FFF（1111 1111 1111）	
1030#	0xC0209018~0xC020901B		FFF（1111 1111 1111）	数据段
1031#	0xC020901C~0xC020901F		FFF（1111 1111 1111）	
1032#	0xC0209020~0xC0209023		FFF（1111 1111 1111）	
1033#	0xC020901C~0xC020901F		FFC（1111 1111 1100）	
	……	……	……	
2047#	0xC0209FFC~0xC0209FFF		FFF（0000 0000 0111）	堆栈段

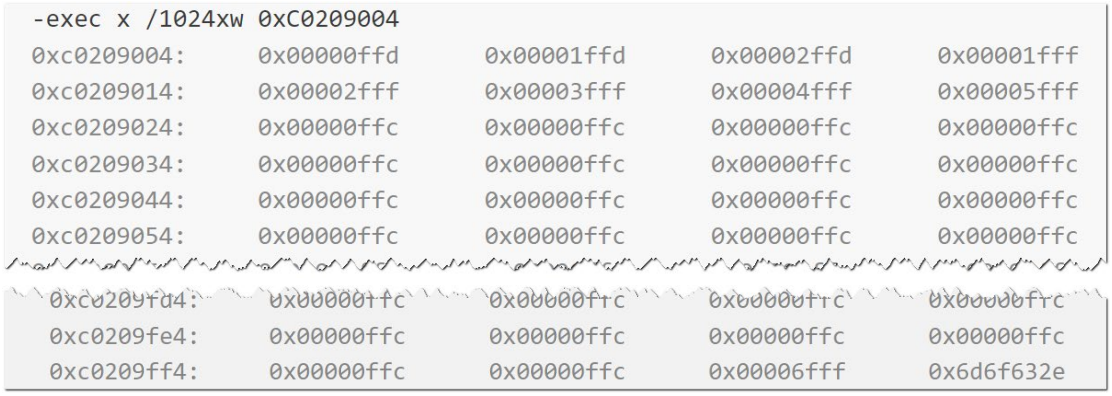


图 6：查看相对虚实地址映射表所在内存

4.4. 找到进程的物理页表

现运行进程的物理页表包括页目录，一张核心页表和两张物理页表。请读者首先思考，这四张页表的逻辑地址是什么，再利用查看该该逻辑地址的内存单元的值，尝试观察四张页表的全部内容，并与你在实验 4.2 中绘制的物理页表相比对，看看是否正确。

5. 实验报告要求

本次实验报告需完成以下内容：

（1）（1 分）完成实验 4.1~4.2，依照实验指导的步骤，依次获取现运行进程 User 结构、Proc 结构和 Text 结构的内容，截图或绘制表格说明，并总结在 UNIX V6++中获取进程的代码段和可交换部分起始位置的逻辑地址和物理地址的方法。

（2）（1 分）根据实验 4.1~4.2 的实验结果，绘制进程的相对虚实地址映射表和 4 张物理页表。

（3）（1 分）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，并验证（2）中给出的相对虚实地址映射表是否正确。

（4）（1 分）首先回答现运行进程的四张页表的逻辑地址是什么，再根据这个逻辑地址获取完整的进程物理页表，仿照表 5，自行绘制表格说明，并验证（2）中给出的 4 张页表是否正确。