

同济大学计算机系

操作系统课程实验报告



学 号 2251557

姓 名 代文波

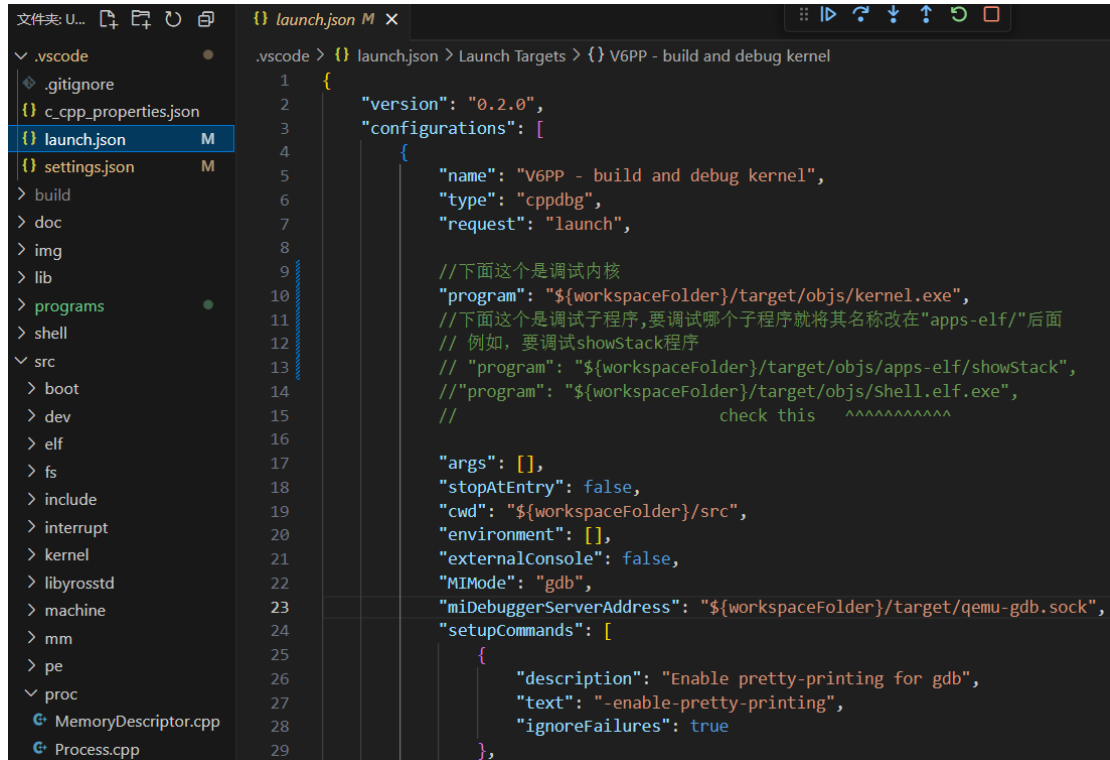
专 业 计算机科学与技术

授课老师 方钰

一、实验 4.1~4.2 基础内容

1.1 准备工作

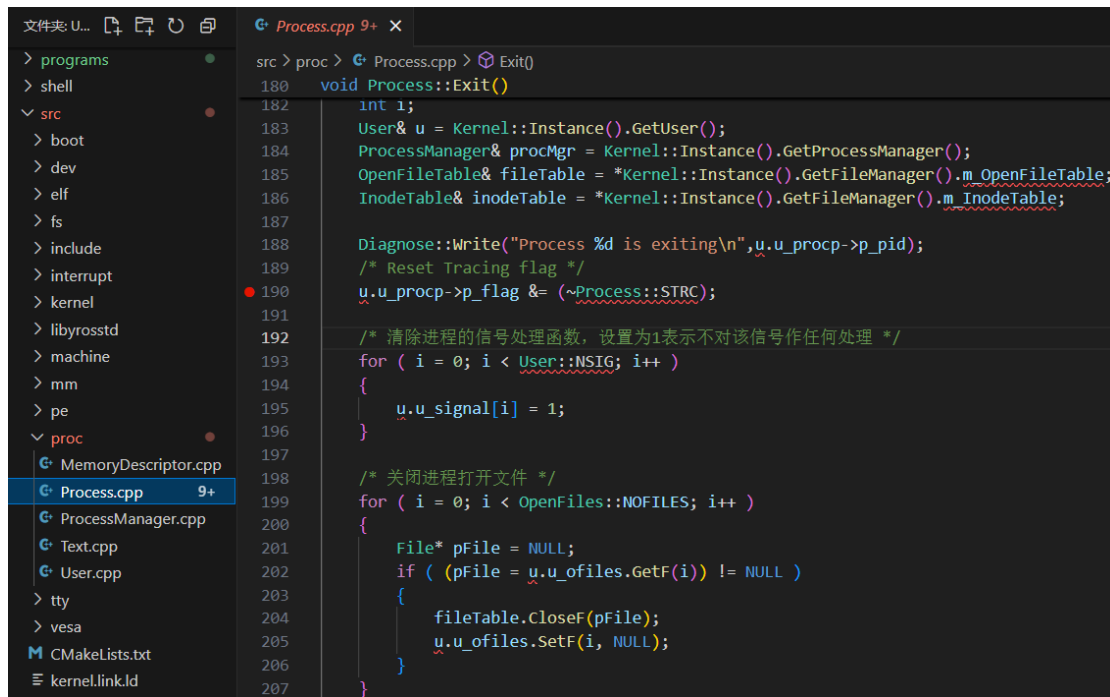
(1) 设置调试对象为 Kernel.exe



The screenshot shows the VS Code interface with the `launch.json` file open. The file is located in the `.vscode` folder. The configuration is for a V6PP build and debug kernel. The `program` is set to `${workspaceFolder}/target/objs/kernel.exe`. The `args` are empty. The `stopAtEntry` is false. The `cwd` is `${workspaceFolder}/src`. The `environment` is empty. The `externalConsole` is false. The `MIMode` is `"gdb"`. The `miDebuggerServerAddress` is `${workspaceFolder}/target/qemu-gdb.sock`. The `setupCommands` are defined to enable pretty-printing for gdb.

```
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "V6PP - build and debug kernel",
6       "type": "cppdbg",
7       "request": "launch",
8
9       //下面这个是调试内核
10      "program": "${workspaceFolder}/target/objs/kernel.exe",
11      //下面这个是调试子程序,要调试哪个子程序就将其名称改在"apps-elf/"后面
12      // 例如, 要调试showStack程序
13      "program": "${workspaceFolder}/target/objs/apps-elf/showStack",
14      // "program": "${workspaceFolder}/target/objs/shell.elf.exe",
15      // check this ^^^^^^^^^^^
16
17      "args": [],
18      "stopAtEntry": false,
19      "cwd": "${workspaceFolder}/src",
20      "environment": [],
21      "externalConsole": false,
22      "MIMode": "gdb",
23      "miDebuggerServerAddress": "${workspaceFolder}/target/qemu-gdb.sock",
24      "setupCommands": [
25        {
26          "description": "Enable pretty-printing for gdb",
27          "text": "-enable-pretty-printing",
28          "ignoreFailures": true
29        }
30      ]
31    }
32  ]
33 }
```

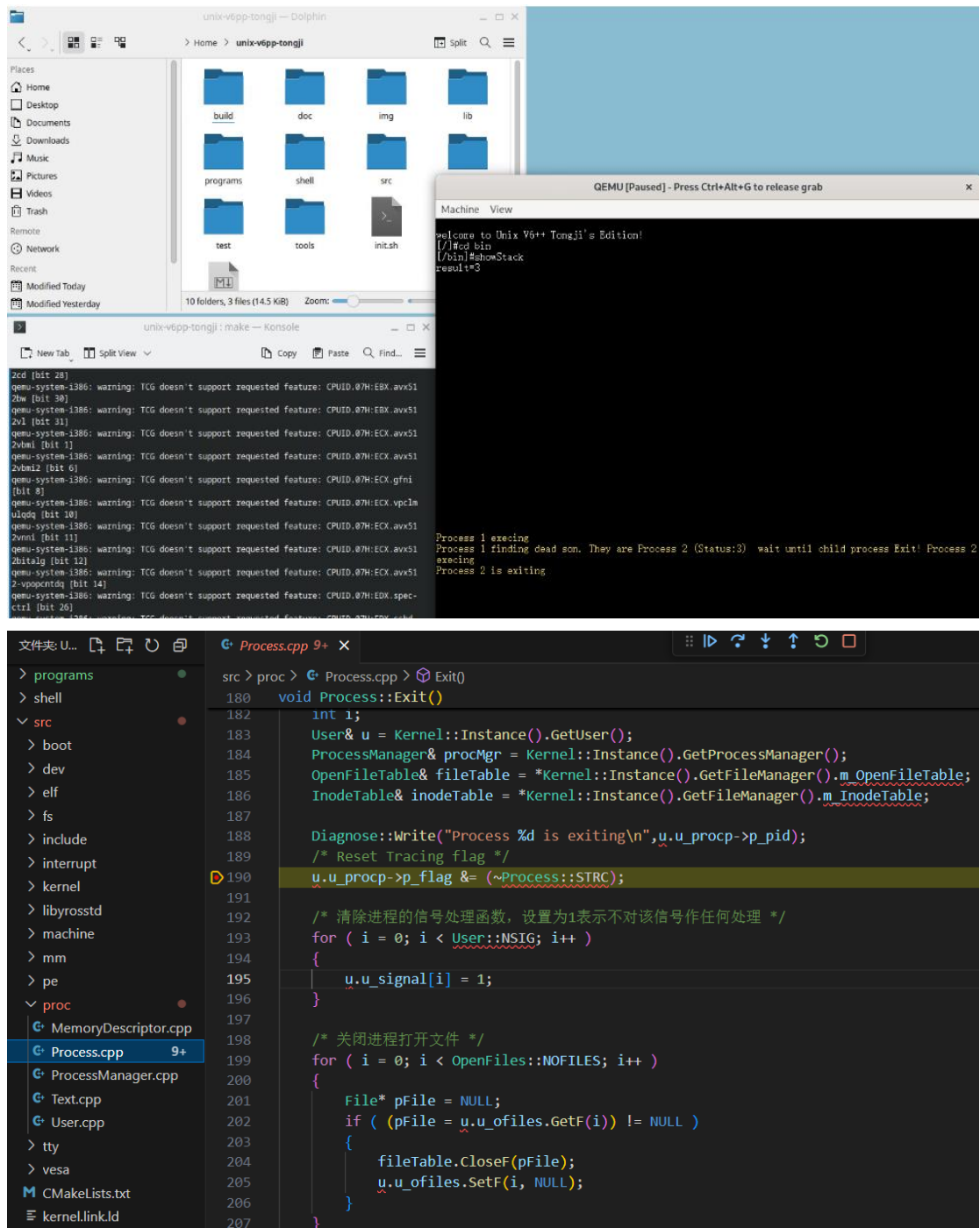
(2) 设置断点位置



The screenshot shows the VS Code interface with the `Process.cpp` file open. The file is located in the `src` folder. The code is for the `Process::Exit()` function. A breakpoint is set at line 190, which is `u.u_procp->p_flag &= (~Process::STRC);`. The code includes comments in Chinese and C++ code for handling process exit.

```
180 void Process::Exit()
181 {
182   int i;
183   User& u = Kernel::Instance().GetUser();
184   ProcessManager& procMgr = Kernel::Instance().GetProcessManager();
185   OpenFileTable& fileTable = *Kernel::Instance().GetFileManager().m_OpenFileTable;
186   InodeTable& inodeTable = *Kernel::Instance().GetFileManager().m_InodeTable;
187
188   Diagnose::Write("Process %d is exiting\n", u.u_procp->p_pid);
189   /* Reset Tracing flag */
190   u.u_procp->p_flag &= (~Process::STRC);
191
192   /* 清除进程的信号处理函数, 设置为1表示不对该信号作任何处理 */
193   for ( i = 0; i < User::NSIG; i++ )
194   {
195     u.u_signal[i] = 1;
196   }
197
198   /* 关闭进程打开文件 */
199   for ( i = 0; i < OpenFiles::NOFILES; i++ )
200   {
201     File* pFile = NULL;
202     if ( (pFile = u.u_ofiles.GetF(i)) != NULL )
203     {
204       fileTable.CloseF(pFile);
205       u.u_ofiles.SetF(i, NULL);
206     }
207   }
208 }
```

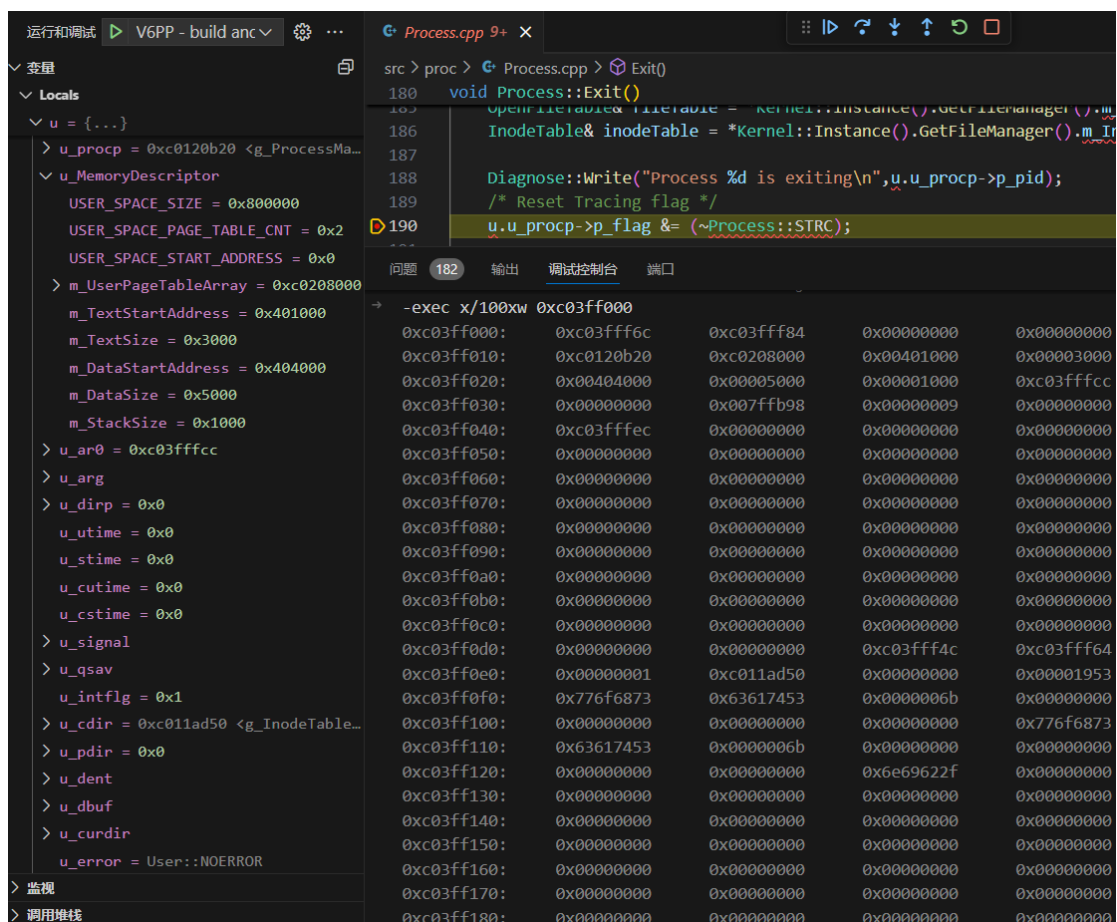
(3) 调试停在断点位置处



下面将逐步获取 User 结构、Proc 结构、Text 结构的内容。

1.2 User 结构

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 (3G+4M-4K)。我们利用该逻辑地址，通过 -exec x/100xw 0xC03FF000 命令可以找到进程的 User 结构的具体内容。

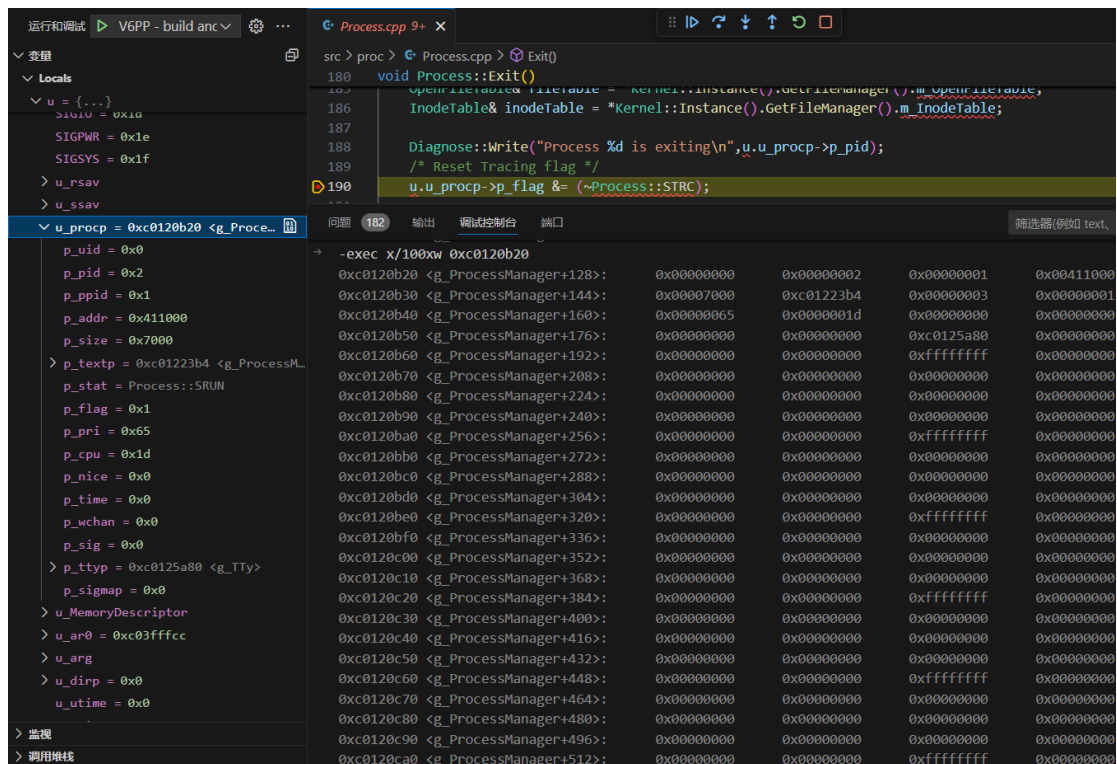


根据信息绘制表格如下：

User 结构		
变量名称	含义	值
Process* u_procp	Proc 结构的逻辑地址	0xC0120B20
MemoryDescriptor u_MemoryDescriptor (具体内容如下，此处均为逻辑地址)		
PageTable* m_UserPageTableArray	相对虚实映射表首地址	0xC0208000
unsigned long m_TextStartAddress	代码段起始地址	0x401000=4M+4K
unsigned long m_TextSize	代码段长度	0x3000=12K
unsigned long m_DataStartAddress	数据段起始地址	0x404000=4M+16K
unsigned long m_DataSize	数据段长度	0x5000=20K
unsigned long m_StackSize	栈段长度	0x1000=4K

1.3 Proc 结构

通过观察 User 结构可以获得的一个重要信息是 u_procp 的值，即进程 proc 结构的逻辑地址为 0xC0120B20。我们利用该逻辑地址，通过 `-exec x /100xw 0xC0120B20` 命令可以找到进程的 Proc 结构。

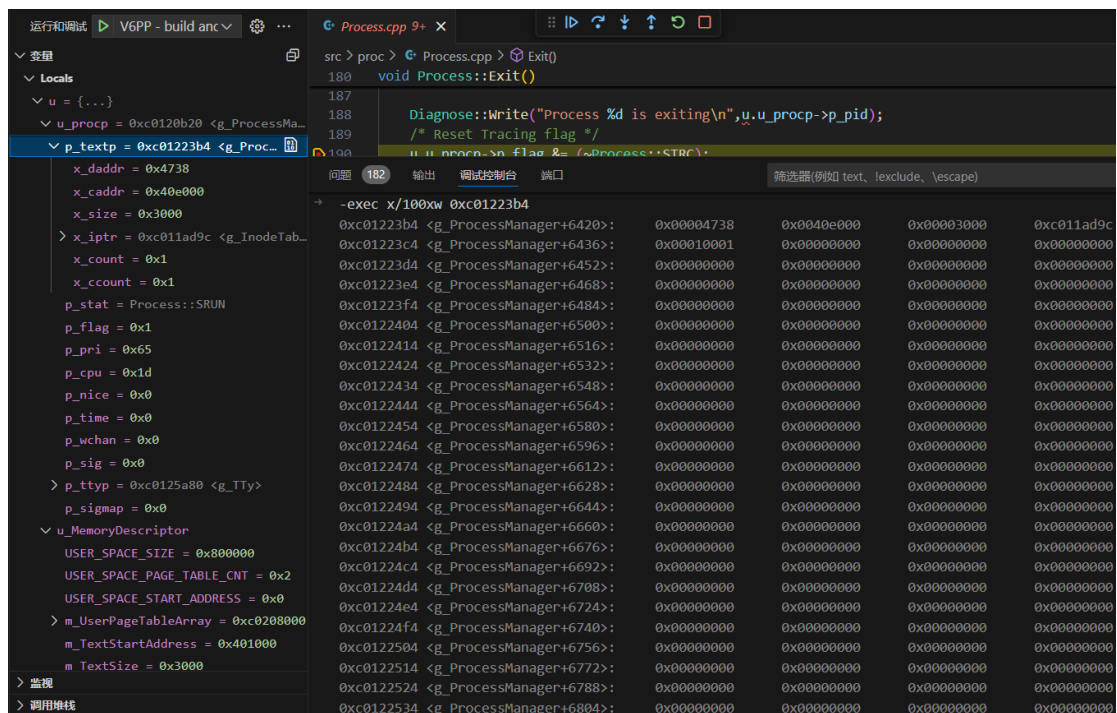


根据信息绘制表格如下：

Proc 结构		
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 区物理地址	0x411000
unsigned int p_size	除共享正文段的长度，以字节为单位	0x7000=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
int p_pri	进程优先数	0x65
int p_cpu	cpu 值，用于计算 p_pri	0x1d
int p_nice	进程优先数微调参数	0x0
int p_time	进程在盘上（内存里）驻留时间	0x0
unsigned long p_wchan	进程睡眠原因	0x0

1.4 Text 结构

通过观察 Proc 结构可以获得的一个重要信息是 p_textp 的值，即进程代码段逻辑地址 Text 结构的逻辑地址为 0xc01223B4。我们利用该逻辑地址，通过 -exec x /100xw 0xc01223B4 命令可以找到进程代码段的 Text 结构。



根据信息绘制表格如下：

Text 结构		
变量名称	含义	值
int x_daddr	代码段在盘交换区的地址	0x4738
Unsigned long x_caddr	代码段的起始地址（物理地址）	0x40E000
Unsigned int x_size	代码段长度	0x3000
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	共享正文段且图像在内存的进程数	1

1.5 总结

经过上面的实验，我们找到了进程图像的两大部分——进程图像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。

由此，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。

进程图像的完整信息：

进程图像的完整信息			
名称	逻辑地址	物理地址	大小
代码段	0x401000=4M+4K	0x40E000	12k
可交换部分	0xC03FF000=3G+4M-4K	0x411000	28k
ppda 区	0xC03FF000=3G+4M-4K	0x411000	4k
数据段	0x404000=4M+16K	0x412000	20k
堆栈段		0x417000	4k

经验总结：

(1) 可交换部分

逻辑地址：

固定值，始终位于 3G~3G+4M 的最后一页，即 0xc03FF000=3G+4M-4K

物理地址：

通过固定的 User 结构的逻辑地址查看到 u_procp (进程 Proc 结构的逻辑地址)，然后找到 Proc 结构的 p_addr (ppda 区的物理地址)。

(2) 代码段

逻辑地址：

通过固定的 User 结构找到 u_MemoryDescriptor.m_TextStartAddress (代码段的逻辑地址)

物理地址：

通过固定的 User 结构的逻辑地址 (0xc03FF000) 查看到 u_procp (进程 Proc 结构的逻辑地址)，然后找到 Proc 结构的 p_textp (进程 Text 结构的逻辑地址)，之后找到 Text 结构的 x_caddr (代码段在内存的物理地址) 或者 x_daddr (代码段在盘交换区的物理地址)。

二、实验 4.1、4.2 进阶

2.1 进程的相对虚实地址映射表

页号	地址	值				
		Page Base Address	中间标志位	u/s	r/w	p
0#	0xC0208000~0xC0208003	x	x	x	x	x
...
1024#	0xC0208000~0xC0208003	x	...	x	x	x
1025#	0xC0209004~0xC0209007	0x0	...	1	0	1
1026#	0xC0209008~0xC020900B	0x1	...	1	0	1
1027#	0xC020900C~0xC020900F	0x2	...	1	0	1
1028#	0xC0209010~0xC0209013	0x1	...	1	1	1
1029#	0xC0209014~0xC0209017	0x2	...	1	1	1
1030#	0xC0209018~0xC020901B	0x3	...	1	1	1
1031#	0xC020901C~0xC020901F	0x4	...	1	1	1
1032#	0xC0209020~0xC0209023	0x5	...	1	1	1
1033#	0xC0209024~0xC0209027	x	...	x	x	x
...
2047#	0xC0209FFC~0xC0209FFF	0x6	...	1	1	1

2.2 四张物理页表

2.2.1 目录页 (0x200#物理页框)

页号	地址	值				
		Page Base Address	中间标志位	u/s	r/w	p
#0	0xC0200000~0xC0200003	0x202	...	1	1	1
#1	0xC0200004~0xC0200007	0x203	...	1	1	1

...
#768	0xC0200C00~0xC0200C03	0x201	...	0	1	1
...

2.2.2 内核页表 (0x201#物理页框)

页号	地址	值				
		Page Base Address	中间标志位	u/s	r/w	p
#0	0xC0201000~0xC0201003	0x0	...	0	1	1
#1	0xC0201004~0xC0201007	0x1	...	0	1	1
...
#1023	0xC0201FFC~0xC0201FFF	0x411	...	0	1	1

2.2.3 用户页表 (0x202、0x203)

(1) 用户页表 1 (0x202)

页号	地址	值				
		Page Base Address	中间标志位	u/s	r/w	p
...

(2) 用户页表 2 (0x203)

页号	地址	值				
		Page Base Address	中间标志位	u/s	r/w	p
#0	0xC0203000~0xC0203003
#1	0xC0203004~0xC0203007	0x40E	...	1	0	1
#2	0xC0203008~0xC020300B	0x40F	...	1	0	1
#3	0xC020300C~0xC020300F	0x410	...	1	0	1
#4	0xC0203010~0xC0203013	0x412	...	1	1	1
#5	0xC0203014~0xC0203017	0x413	...	1	1	1
#6	0xC0203018~0xC020301B	0x414	...	1	1	1
#7	0xC020301C~0xC020301F	0x415	...	1	1	1
#8	0xC0203020~0xC0203023	0x416	...	1	1	1
#9	0xC0203024~0xC0203027	x	...	x	x	x
...
#1023	0xC0203FFC~0xC0203FFF	0x417	...	1	1	1

三、实验 4.3

通过上面的实验，我们知道进程的相对虚实地址映射表位于 0xC0208000（逻辑地址）起始的两个 4K 的页面中。由于第一个页面和第二个页面的第一项都留给编译器了，所以我们先直接从第二个页表的第二项开始看，即通过查看 0xC0209004 起始的内存单元，来了解相对虚实地址映射表中进程的代码段内容和数据段内容；然后查看第二个页表的最后一项，即通过查看 0xC0209FFC 起始的内存单元，来了解相对虚实地址映射表

中进程堆栈段内容。

```
src > proc > Process.cpp > Exit()
180 void Process::Exit()
181 {
182     Diagnose::Write("Process %d is exiting\n", u.u_procp->p_pid);
183     /* Reset Tracing flag */
184     u.u_procp->p_flag &= (~Process::STRC);
185     /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */
186     for (i = 0; i < HUP; i++)
187         u.u_procp->p_sig[i] = 0;
188 }
```

问题 182 输出 调试控制台 端口

-exec x/100xw 0xC0209004

0xc0209004:	0x00000ffd	0x00001ffd	0x00002ffd	0x00001fff
0xc0209014:	0x00002fff	0x00003fff	0x00004fff	0x00005fff
0xc0209024:	0x00000ffc	0x00000ffc	0x00000ffc	0x00000ffc
0xc0209034:	0x00000ffc	0x00000ffc	0x00000ffc	0x00000ffc
0xc0209044:	0x00000ffc	0x00000ffc	0x00000ffc	0x00000ffc
0xc0209054:	0x00000ffc	0x00000ffc	0x00000ffc	0x00000ffc

```
src > proc > Process.cpp > Exit()
180 void Process::Exit()
181 {
182     Diagnose::Write("Process %d is exiting\n", u.u_procp->p_pid);
183     /* Reset Tracing flag */
184     u.u_procp->p_flag &= (~Process::STRC);
185     /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */
186     for (i = 0; i < HUP; i++)
187         u.u_procp->p_sig[i] = 0;
188 }
```

问题 182 输出 调试控制台 端口

-exec x/100xw 0xC0209FFC

0xc0209ffc:	0x00006fff	0x6d6f632e	0x746e656d	0x00000150
0xc020a00c:	0xffc00000	0x00000200	0x00000600	0x00000000
0xc020a01c:	0x00000000	0x00000000	0x40000000	0x7865742e
0xc020a02c:	0x00000074	0x00002a41	0x00001000	0x00002c00
0xc020a03c:	0x00000800	0x00000000	0x00000000	0x00000000
0xc020a04c:	0x60000020	0x7461642e	0x00000061	0x000002f8
0xc020a05c:	0x00004000	0x00000400	0x00003400	0x00000000

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	x	x
...
1024#	0xC0208000~0xC0208003	x	x
1025#	0xC0209004~0xC0209007	0x0	0xFFD (1111 1111 1101)
1026#	0xC0209008~0xC020900B	0x1	0xFFD (1111 1111 1101)
1027#	0xC020900C~0xC020900F	0x2	0xFFD (1111 1111 1101)
1028#	0xC0209010~0xC0209013	0x1	0xFFF (1111 1111 1111)
1029#	0xC0209014~0xC0209017	0x2	0xFFF (1111 1111 1111)
1030#	0xC0209018~0xC020901B	0x3	0xFFF (1111 1111 1111)
1031#	0xC020901C~0xC020901F	0x4	0xFFF (1111 1111 1111)
1032#	0xC0209020~0xC0209023	0x5	0xFFF (1111 1111 1111)
1033#	0xC0209024~0xC0209027	0x0	0xFFC (1111 1111 1100)
...
2047#	0xC0209FFC~0xC0209FFF	0x6	0xFFF (1111 1111 1111)

经验证，步骤二中给出的相对虚实地址映射表正确。

四、实验 4.4

这四张表的逻辑地址：

这四张页表按照：目录页、内核页表、用户页表 1、用户页表 2 的顺序排列在逻辑内存的 3G+2M~3G+2M-4K 的前四个页框内。这四张页表开始位置的逻辑地址依次为：0xC0200000（3G+2M）、0xC0201000（3G+2M+4K）、0xC0202000（3G+2M+8K）、0xC0203000（3G+2M+12K）。

(1) 目录页

变量

Locals

i = 0xc01223b4

> u = {...}

> procMgr = {...}

> fileTable = {...}

> inodeTable = {...}

> swapperMgr = {...}

> bufMgr = {...}

> blkno = 0x120aa0

> pBuf = 0xc03ff000

> current = 0xc011a5f0 <g_OpenFileTa...

> userPageMgr = {...}

> this = 0xc0120b20 <g_ProcessManage...

> Registers

src > proc > Process.cpp > Exit()

180 void Process::Exit()

187

188 Diagnose::Write("Process %d is exiting\n",u.u_procp->p_pid);

189 /* Reset Tracing flag */

190 u.u_procp->p_flag &= (~Process::STRC);

191

192 /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */

193 for (i = 0; i < HOOK_MSTC; i++)

问题 182 输出 调试控制台 端口

→ -exec x/100xw 0xC0200000

0xc0200000: 0x00202027 0x00203027 0x00000000 0x00000000

0xc0200010: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200020: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200030: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200040: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200050: 0x00000000 0x00000000 0x00000000 0x00000000

变量

Locals

i = 0xc01223b4

> u = {...}

> procMgr = {...}

> fileTable = {...}

> inodeTable = {...}

> swapperMgr = {...}

> bufMgr = {...}

> blkno = 0x120aa0

> pBuf = 0xc03ff000

> current = 0xc011a5f0 <g_OpenFileTa...

> userPageMgr = {...}

> this = 0xc0120b20 <g_ProcessManage...

> Registers

src > proc > Process.cpp > Exit()

180 void Process::Exit()

187

188 Diagnose::Write("Process %d is exiting\n",u.u_procp->p_pid);

189 /* Reset Tracing flag */

190 u.u_procp->p_flag &= (~Process::STRC);

191

192 /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */

193 for (i = 0; i < HOOK_MSTC; i++)

问题 182 输出 调试控制台 端口

→ -exec x/100xw 0xC0200C00

0xc0200c00: 0x00201023 0x00000000 0x00000000 0x00000000

0xc0200c10: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200c20: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200c30: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200c40: 0x00000000 0x00000000 0x00000000 0x00000000

0xc0200c50: 0x00000000 0x00000000 0x00000000 0x00000000

页号	地址	值	
		高 20 位页编号	低 12 位标志位
#0	0xC0200000~0xC0200003	0x202	0x027 (0000 0010 0111)
#1	0xC0200004~0xC0200007	0x203	0x027 (0000 0010 0111)
...
#768	0xC0200C00~0xC0200C03	0x201	0x023 (0000 0010 0011)
...

(2) 内核页表

变量

Locals

i = 0xc01223b4
> u = {...}
> procMgr = {...}
> fileTable = {...}
> inodeTable = {...}
> swapperMgr = {...}
> bufMgr = {...}
blkno = 0x120aa0
> pBuf = 0xc03ff000
> current = 0xc011a5f0 <g_OpenFileTa...
> userPageMgr = {...}

src > proc > Process.cpp > Exit()
180 void Process::Exit()
188 Diagnose::Write("Process %d is exiting\n",u.u_procp->p_pid);
189 /* Reset Tracing flag */
190 u.u_procp->p_flag &= (~Process::STRC);
191
192 /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */
for (i = 0; i < User::NSTRC; i++)

问题 182 输出 调试控制台 端口
→ -exec x/100xw 0xC0201000
0xc0201000: 0x00000003 0x00001003 0x00002003 0x00003003
0xc0201010: 0x00004003 0x00005003 0x00006003 0x00007023
0xc0201020: 0x00008003 0x00009003 0x0000a003 0x0000b003
0xc0201030: 0x0000c003 0x0000d003 0x0000e003 0x0000f003

运行和调试 V6PP - build anc

变量

Locals

i = 0xc01223b4
> u = {...}
> procMgr = {...}
> fileTable = {...}
> inodeTable = {...}
> swapperMgr = {...}
> bufMgr = {...}
blkno = 0x120aa0
> pBuf = 0xc03ff000
> current = 0xc011a5f0 <g_OpenFileTa...
> userPageMgr = {...}
> this = 0xc0120b20 <g_ProcessManage...
> Registers

src > proc > Process.cpp > Exit()
180 void Process::Exit()
188 Diagnose::Write("Process %d is exiting\n",u.u_procp->p_pid);
189 /* Reset Tracing flag */
190 u.u_procp->p_flag &= (~Process::STRC);
191
192 /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */
for (i = 0; i < User::NSTRC; i++)

问题 182 输出 调试控制台 端口
→ -exec x/100xw 0xC0201FF8
0xc0201ff8: 0x003fe003 0x00411063 0x00000067 0x00404004
0xc0202008: 0x00404004 0x00404004 0x00404006 0x00404006
0xc0202018: 0x00404026 0x00404006 0x00404006 0x00404006
0xc0202028: 0x00404006 0x00404006 0x00404006 0x00404006
0xc0202038: 0x00404006 0x00404006 0x00404006 0x00404006
0xc0202048: 0x00404006 0x00404006 0x00404006 0x00404006

页号	地址	值	
		高 20 位页编号	低 12 位标志位
#0	0xC0201000~0xC0200003	0x0	0x003 (0000 0000 0011)
#1	0xC0201004~0xC0200007	0x1	0x003 (0000 0000 0011)
...
#1022	0xC0201FF8~0xC0200FFB	0x3FE (=1022)	0x003 (0000 0000 0011)
#1023	0xC0201FFC~0xC0201FFF	0x411	0x063 (0000 0110 0011)

(3) 用户页表 1

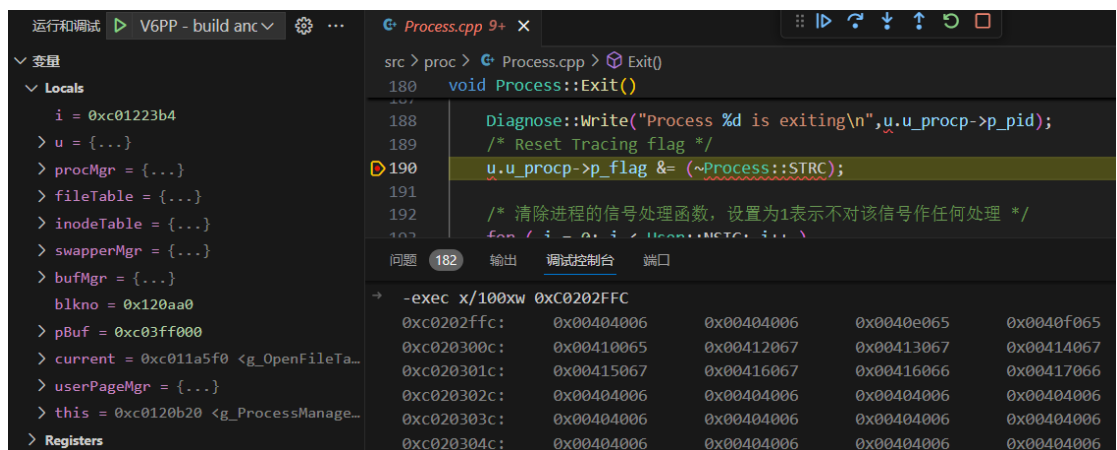
变量

Locals

i = 0xc01223b4
> u = {...}
> procMgr = {...}
> fileTable = {...}
> inodeTable = {...}
> swapperMgr = {...}
> bufMgr = {...}
blkno = 0x120aa0
> pBuf = 0xc03ff000
> current = 0xc011a5f0 <g_OpenFileTa...
> userPageMgr = {...}
> this = 0xc0120b20 <g_ProcessManage...
> Registers

src > proc > Process.cpp > Exit()
180 void Process::Exit()
188 Diagnose::Write("Process %d is exiting\n",u.u_procp->p_pid);
189 /* Reset Tracing flag */
190 u.u_procp->p_flag &= (~Process::STRC);
191
192 /* 清除进程的信号处理函数，设置为1表示不对该信号作任何处理 */
for (i = 0; i < User::NSTRC; i++)

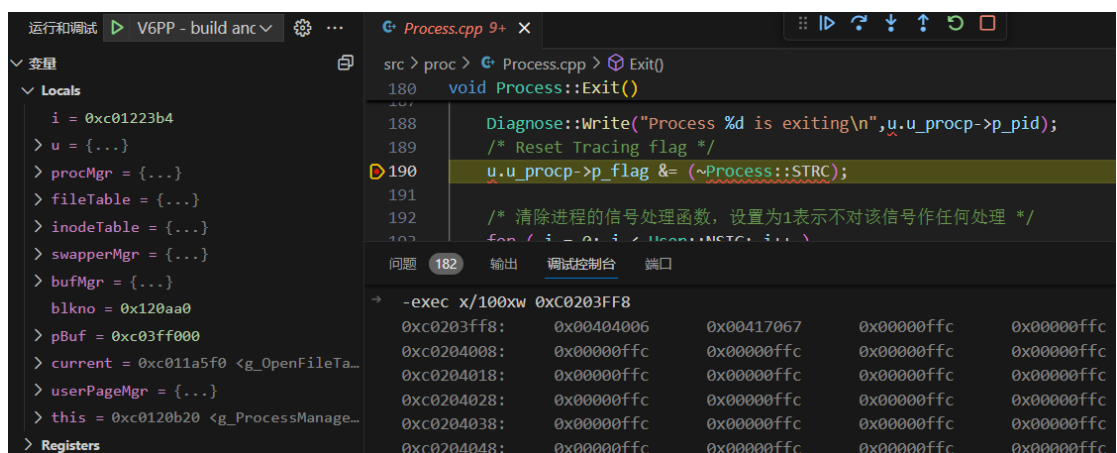
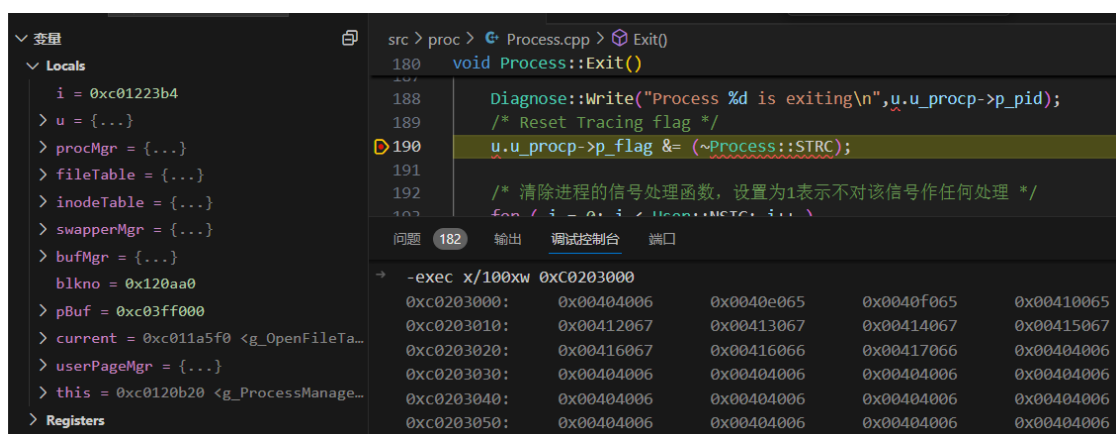
问题 182 输出 调试控制台 端口
→ -exec x/100xw 0xC0202000
0xc0202000: 0x00000067 0x00404004 0x00404004 0x00404004
0xc0202010: 0x00404006 0x00404006 0x00404026 0x00404006
0xc0202020: 0x00404006 0x00404006 0x00404006 0x00404006
0xc0202030: 0x00404006 0x00404006 0x00404006 0x00404006
0xc0202040: 0x00404006 0x00404006 0x00404006 0x00404006
0xc0202050: 0x00404006 0x00404006 0x00404006 0x00404006



页号	地址	值	
		高 20 位页编号	低 12 位标志位
#0	0xC0202000~0xC0202003	0x0	0x067 (0000 0110 0111)
...
#1023	0xC0202FFC~0xC0202FFF	0x404	0x006 (0000 0000 0110)

由于这一部分是留给编译器使用的，所以这一部分页表的具体内容这里不做仔细研究了。

(4) 用户页表 2



页号	地址	值	
		高 20 位页编号	低 12 位标志位
#0	0xC0203000~0xC0203003	0x404	0x006 (0000 0000 0110)

#1	0xC0203004~0xC0203007	0x40E	0x065 (0000 0000 0101)
#2	0xC0203008~0xC020300B	0x40F	0x065 (0000 0000 0101)
#3	0xC020300C~0xC020300F	0x410	0x065 (0000 0000 0101)
#4	0xC0203010~0xC0203013	0x412	0x067 (0000 0000 0111)
#5	0xC0203014~0xC0203017	0x413	0x067 (0000 0000 0111)
#6	0xC0203018~0xC020301B	0x414	0x067 (0000 0000 0111)
#7	0xC020301C~0xC020301F	0x415	0x067 (0000 0000 0111)
#8	0xC0203020~0xC0203023	0x416	0x067 (0000 0000 0111)
#9	0xC0203024~0xC0203027	0x416	0x066 (0000 0000 0110)
...
#1022	0xC0203FF8~0xC0203FFB	0x404	0x006 (0000 0000 0110)
#1023	0xC0203FFC~0xC0203FFF	0x417	0x067 (0000 0110 0111)

经检验，步骤二中给出的四张物理页表均正确。