

同济大学计算机系

操作系统实验报告



学 号 2152809

姓 名 曾崇然

专 业 计算机科学与技术

授课老师 方钰老师

一. 添加新的系统调用接口

(1) 修改系统调用处理子程序入口表

interrupt	{ 2, &Sys_Ssig },	/* 48 = sig */
DiskInterrupt.cpp	{ 1, &Sys_Getppid },	/* 49 = getppid */
Exception.cpp	{ 0, &Sys_Nosys },	/* 50 = nosys */
KeyboardInterrupt.cpp	{ 0, &Sys_Nosys },	/* 51 = nosys */
SystemCall.cpp	{ 0, &Sys_Nosys },	/* 52 = nosys */
TimeInterrupt.cpp	{ 0, &Sys_Nosys },	/* 53 = nosys */
Makefile	{ 0, &Sys_Nosys },	/* 54 = nosys */
		/* 55 = nosys */

{0, &Sys_Nosys}项表示该入口未被使用, 将其替换为所要添加的{1, &Sys_Getppid}, 1 表示参数, &Sys_Getppid 表示系统调用处理子程序的入口地址。此处为所要新增的系统调用添加了入口。

(2) 添加系统调用处理子程序的声明和定义

在 SystemCall.h 中添加声明:

```
Regs.h
Simple.h
SwapperManager.h
SystemCall.h
TaskStateSegment.h

/* 49 = getppid count = 1 */
static int Sys_Getppid();

/* 50 ~ 63 = nosys count = 0 */
```

在 SystemCall.cpp 中添加定义:

```
Regs.h
Simple.h
SwapperManager.h
SystemCall.h
TaskStateSegment.h
Text.h
TimeInterrupt.h
TTY.h
User.h
Utility.h
Video.h
interrupt
DiskInterrupt.cpp
Exception.cpp
KeyboardInterrupt.cpp
SystemCall.cpp
TimeInterrupt.cpp
Makefile
kernel

/* 49 = getppid count = 1 */
int SystemCall::Sys_Getppid()
{
    ProcessManager& procMgr = Kernel::Instance().GetProcessManager();
    User& u = Kernel::Instance().GetUser();

    int i;
    int curpid = (int)u.u_arg[0];

    u.u_arg[User::EAX] = -1;

    for (i = 0; i < ProcessManager::NPROC; i++)
    {
        if (procMgr.process[i].p_pid == curpid)
        {
            u.u_arg[User::EAX] = procMgr.process[i].p_pid;
        }
    }

    return 0;
}
```

二. 为新的系统调用添加新的库函数

(1) 添加库函数的声明

```
stdarg.h
stdio.h
stdlib.h
string.h
sys.h
time.h

int getppid();
int getpid();
unsigned int getgid();
```

(2) 添加库函数的定义

```
printf.c
sprintf.c
stdio.c
stdlib.c
string.c
sys.c
time.c
valist.h
Lib_V6++.a
Makefile

int getppid(int pid)
{
    int res;
    asm volatile ("int $0x80":"=a"(res):"a"(49),"b"(pid));
    if (res >= 0)
        return res;
    return -1;
}
```

(3) 进行编译

```
> proc
> program
> shell
> test
> tty
  Link.ld
  Makefile
  Makefile.inc
> targets
> tools
```

```
CDI Build Console [oos]
已复制 1 个文件。
copy ..\targets\objs\kernel.bin ..\tools\MakeImage\bin\Debug\kernel.bin
已复制 1 个文件。
copy ..\targets\img\c.img ..\tools\MakeImage\bin\Debug\c.img
已复制 1 个文件。
cd ..\tools\MakeImage\bin\Debug && build.exe c.img boot.bin kernel.bin
programs
copy ..\tools\MakeImage\bin\Debug\c.img "..\targets\UNIXV6++"\c.img
已复制 1 个文件。

**** Build Finished ****
```

编译成功，系统调用处理子程序和对应的库函数添加成功

三. 添加测试程序进行测试

(1) 添加代码并编译生成程序

添加代码：

```
> machine
> mm
> pe
> proc
  > objs
    > cat.c
    > cat1.c
    > copyfile.c
    > cp.c
    > date.c
    > echo.c
    > forks.c
    > GetOptAndPath.h
    > getppid.c
    > ls.c
```

```
SystemCall.h sys.h sys.c MemoryDescriptor.h getppid.c Process.cpp
#include <stdio.h>
#include <sys.h>

int main1()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid(pid);

    printf("This is Process %d# speaking...\n", pid);
    printf("My parent process ID is: %d\n", ppid);

    return 0;
}
```

修改配置文件：

```
newsig.c
performance.c
rm.c
showStack.c
shutdown.c
```

```
$(TARGET)\getppid.exe : getppid.c
$(CC) $(CFLAGS) -I"$(INCLUDE)" -I"$(LIB_INCLUDE)" $< -e _main1 $(V6++LIB) -o $@
copy $(TARGET)\getppid.exe $(MAKEIMAGEPATH)\$(BIN)\getppid.exe

clean:
del $(TARGET)\*
```

编译生成：

```
systest.c
stack.c
test.c
trace.c
cmd.exe - [x86/le]
cmd.exe.lnk
Makefile
```

```
已复制 1 个文件。
cd ..\tools\MakeImage\bin\Debug && build.exe c.img boot.bin kernel.bi
programs
copy ..\tools\MakeImage\bin\Debug\c.img "..\targets\UNIXV6++"\c.img
已复制 1 个文件。

**** Build Finished ****
```

(2) 运行和调试

添加断点

```
proc
program
shell
test
tty
```

```
int i;
int curpid = (int)u.u_arg[0];

u.u_ar0[User::EAX] = -1;

for (i = 0; i < ProcessManager::NPROC; i++)
```

执行到断点处，通过 u_ar0 可知系统调用号 49 通过压栈操作由 EAX 带入核心栈

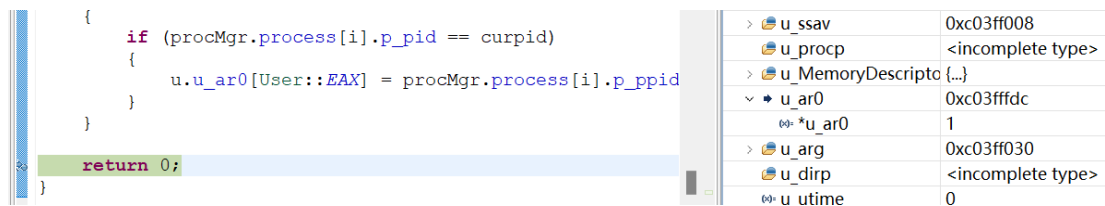
```
int SystemCall::Sys_Getppid()
{
    ProcessManager& procMgr = Kernel::Instance().GetProces
    User& u = Kernel::Instance().GetUser();

    int i;
    int curpid = (int)u.u_arg[0];

    u.u_ar0[User::EAX] = -1;
```

u_ssav	0xc03ff008
u_procp	<incomplete type>
u_MemoryDescripto	{...}
* u_ar0	0xc03fffdc
*u_ar0	49
u_arg	0xc03ff030
u_dirp	<incomplete type>
u_utime	0

执行到 Sys_Getppid 最后一步，u_ar0 指向的地址的内容变为 1



查看核心栈（停止在 int curpid = (int)u.u_arg[0]时）：

0xc011ae94	C03FFFB0	C03FFFE0	00000000	00000000	00000023
0xc0208000	C03FFFC0	00000023	00000002	00000001	00407030
0xc0200000	C03FFFD0	000E0000	0000FFAC	C03FFFE8	00000031
0xc0201000	C03FFFE0	00000016	C03FFFE0	007FFFB8	0040141C
0xc0202000	C03FFFF0	0000001B	00000216	007FFFAC	00000023
0xc0203000	C0400000	000E0000	0000FFAC	C03FFFE8	00000031
0xc03fffdc	C0400010	000E0000	0000FFAC	C03FFFE8	00000031

运行结果：

```
[/]#cd bin
[/bin]#getppid.exe
This is Process 2# speaking...
My parent process ID is: 1
[/bin]#_
```

四. 确定黄色标注的地址

(1) 获取核心栈内容

0xc03fffdc	Address	0 - 3	4 - 7	8 - B	C - F
0x007fffb0	C03FFFD0	000E0000	0000FFAC	C03FFFE8	00000031
0x007fffd8	C03FFFE0	00000016	C03FFFE0	007FFFB8	0040141C
	C03FFFF0	0000001B	00000216	007FFFAC	00000023

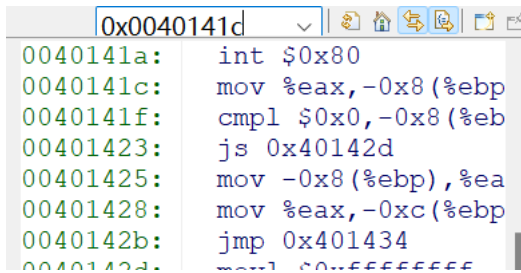
0xc03fffdc	EAX	0x00000031
0xc03fffe0	入口局变	0x00000016
0xc03fffe4	入口局变	0xc03fffec
0xc03fffe8	OLD EBP	0x007fffb8
0xc03fffec	EIP	0x0040141C
0xc03ffff0	CS	0x0000001B
0xc03ffff4	EFLAGS	0x00000216
0xc03ffff8	ESP	0x007fffac
0xc03ffffc	SS	0x00000023

(2) 观察用户栈

根据 ebp 找到用户栈

0xc03fffdc	Address	0 - 3	4 - 7	8 - B	C - F
0x007fffb8	007FFF80	00000000	00000000	00000000	00000000
	007FFF90	00000000	00000000	00000000	00000000
	007FFFA0	00000000	00000000	00000000	00000000
	007FFFB0	00000002	00407DA4	007FFFD8	00401019

(3) 找到对应指令



```
0x0040141d
0040141a: int $0x80
0040141c: mov %eax, -0x8(%ebp)
0040141f: cmpl $0x0, -0x8(%ebp)
00401423: js 0x40142d
00401425: mov -0x8(%ebp), %eax
00401428: mov %eax, -0xc(%ebp)
0040142b: jmp 0x401434
0040142d: mov $0xffffffff, %eax
```

(4) 确定黄色块的内容

0xC03FFFD8	EBP	0xC03FFFE8	指向当前栈帧基地址
0xC03FFFD8	EAX	0x00000031	
0xC03FFFE0	入口局变	0x00000016	
0xC03FFFE4	入口局变	0xC03FFFE8	
0xC03FFFE8	OLD EBP	0x007FFFB8	调用程序的基址
0xC03FFFE8	EIP	0x0040141C	指向返回后下一条指令
0xC03FFFF0	CS	0x0000001B	
0xC03FFFF4	EFLAGS	0x00000216	
0xC03FFFF8	ESP	0x007FFFAC	调用程序用户栈栈顶指针
0xC03FFFFC	SS	0x00000023	