



本科生实验报告

实验课程:操作系统原理

实验名称:lab8 从内核态到用户态

专业名称:计算机科学与技术(人工智能与大数据方向)

学生姓名:刘卓逸

学生学号:21307303

实验地点:东校园实验中心大楼D503

实验成绩:

报告时间:2023年6月15日

1.实验要求

- DDL: 2023年06月15号 23:59
- 提交的内容: 请将**3个assignment**的代码和实验报告放到压缩包中, 命名为“**lab8-姓名-学号**”, 并交到课程邮箱 `os_sysu_lab@163.com`

实验报告的pdf提交至 `http://inbox.weiyun.com/CmccX5vB`
- 材料的代码放置在 **src** 目录下。

1. 实验不限语言, C/C++/Rust都可以。
2. 实验不限平台, Windows、Linux和MacOS等都可以。
3. 实验不限CPU, ARM/Intel/Risc-V都可以。

2.实验过程

Assignment 1 系统调用

实验内容

编写一个系统调用, 然后在进程中调用之, 根据结果回答以下问题。

- 展现系统调用执行结果的正确性, 结果截图并说说你的实现思路。
- 分析执行系统调用后的栈的变化情况。
- 说明TSS在系统调用执行过程中的作用。

实验原理

asm_system_call

先保存现场, 后将系统调用参数传入寄存器, 执行 `int 0x80` 中断, 最后恢复现场。

由于系统调用时CPU会从TSS中加载高特权级的栈地址到esp寄存中, 所以原本的栈和系统调用时的栈不是同一个(指针不同), 所以需要将参数传入寄存器

SystemService类

管理系统调用的类。

定义如下

```
#include "os_constant.h"
class SystemService
{
public:
    SystemService();
    void initialize();
    // 设置系统调用, index=系统调用号, function=处理第index个系统调用函数的地址
    bool setSystemCall(int index, int function);
};

void SystemService::initialize()
{
    memset((char *)system_call_table, 0, sizeof(int) * MAX_SYSTEM_CALL);
    // 代码段的选择子默认是DPL=0的平坦模式代码段选择子,
    // 但中断描述符的DPL=3, 否则用户态程序无法使用该中断描述符
    interruptManager.setInterruptDescriptor(0x80, (uint32)asm_system_call_handler,
    3);
}
```

asm_system_call_handler

0x80中断处理函数:

保护现场, 从tss加载ds,es,fs,gs寄存器,

开启中断并调用系统处理函数, 调用完关中断,

将参数弹出栈, 并把系统处理函数的返回值从eax存到ASM_TEMP中

最后还原现场, 完成系统调用

实验过程

1.展现系统调用结果的正确性

实验代码

在 `assignment1/include/syscall.h`中添加系统调用的函数声明:

```
// 第1个系统调用
void syscall_move_cursor(int x,int y);
```

```
// 第2个系统调用
void syscall_print(const char *str);
```

在 `assignment1/src/kernel/setup.cpp` 中添加系统调用的实现

```
void syscall_move_cursor(int x,int y)
{
    stdio.moveCursor(x,y);
}
void syscall_print(const char *str)
{
    stdio.print(str);
}
```

在 `assignment1/src/kernel/setup.cpp/setup_kernel()` 中调用 `system_call`

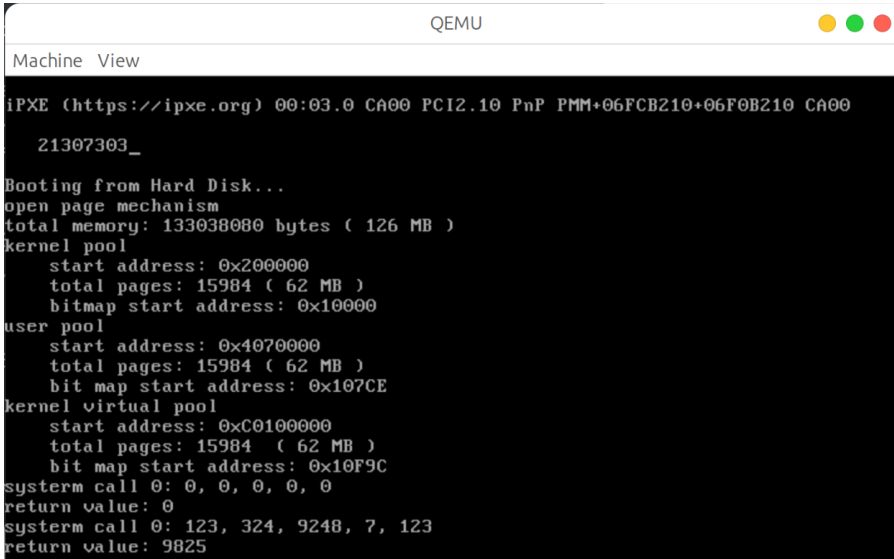
```
ret = asm_system_call(0);
printf("return value: %d\n", ret);

ret = asm_system_call(0, 123, 324, 9248, 7, 123);
printf("return value: %d\n", ret);

asm_system_call(1,3,3);

asm_system_call(2,(int)((const char *)"21307303"));
```

实验结果



```
Machine View
IPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+06FCB210+06F0B210 CA00
21307303_
Booting from Hard Disk...
open page mechanism
total memory: 133038080 bytes ( 126 MB )
kernel pool
  start address: 0x200000
  total pages: 15984 ( 62 MB )
  bitmap start address: 0x10000
user pool
  start address: 0x4070000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x107CE
kernel virtual pool
  start address: 0xC0100000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x10F9C
system call 0: 0, 0, 0, 0, 0
return value: 0
system call 0: 123, 324, 9248, 7, 123
return value: 9825
```

可见光标移动到了(3,3)并输出了我的学号，说明系统调用成功

2.分析系统调用后栈的变化情况

准备进入system_call 0x7be4

```
终端
+
../src/kernel/setup.cpp
66 // 设置2号系统调用
67 systemService.setSystemCall(2, (int)syscall_print);
68
69 int ret;
70
B+> 71 ret = asm_system_call(0);
72 printf("return value: %d\n", ret);
73
74 ret = asm_system_call(0, 123, 324, 9248, 7, 123);
75 printf("return value: %d\n", ret);
76
77 asm_system_call(1,3,3);
78

remote Thread 1.1 In: setup kernel L71 PC: 0x20f6c
Breakpoint 1 at 0x20f6c: file ../src/kernel/setup.cpp, line 71.
(gdb) c
Continuing.

Breakpoint 1, setup_kernel () at ../src/kernel/setup.cpp:71
(gdb) info register esp
esp          0x7be4          0x7be4
```

进入system_call 0x7bc0

```
终端
+
../src/utls/asm_utils.asm
66 pop ds
67 mov eax, [ASM_TEMP]
68
69 iret
70 asm_system_call:
> 71 push ebp
72 mov ebp, esp
73
74 push ebx
75 push ecx
76 push edx
77 push esi
78 push edi

remote Thread 1.1 In: asm_system_call L71 PC: 0x2222c
(gdb) info register esp
esp          0x7be4          0x7be4
(gdb) si
(gdb) s
asm_system_call () at ../src/utls/asm_utils.asm:71
(gdb) info register esp
esp          0x7bc0          0x7bc0
```

将ebp给esp后 0x7bbc

```
终端
+
../src/utls/asm_utils.asm
66 pop ds
67 mov eax, [ASM_TEMP]
68
69 iret
70 asm_system_call:
71 push ebp
72 mov ebp, esp
> 73
74 push ebx
75 push ecx
76 push edx
77 push esi
78 push edi

remote Thread 1.1 In: asm_system_call L74 PC: 0x2222f
asm_system_call () at ../src/utls/asm_utils.asm:71
(gdb) info register esp
esp          0x7bc0          0x7bc0
(gdb) si
(gdb) si
(gdb) info register esp
esp          0x7bbc          0x7bbc
```

刚进入asm_system_call_handler 0x7b8c

```
终端
+
../src/utils/asm_utils.asm
24         dd 0
25
26     ASM_TEMP dd 0
27     ; int asm_system_call_handler();
28     asm_system_call_handler:
> 29         push ds
30         push es
31         push fs
32         push gs
33         pushad
34
35         push eax
36

remote Thread 1.1 In: asm_system_call_handler L29 PC: 0x221f0
Continuing.

Breakpoint 2, asm_system_call () at ../src/utils/asm_utils.asm:91
(gdb) si
asm_system_call_handler () at ../src/utils/asm_utils.asm:29
(gdb) info register esp
esp             0x7b8c             0x7b8c
```

保存完现场后 0x7b48

```
终端
+
../src/utils/asm_utils.asm
46         pop eax
47
48         ; 参数压栈
49         push edi
50         push esi
51         push edx
52         push ecx
53         push ebx
54
8+> 55         sti
56         call dword[system_call_table + eax * 4]
57         cli
58

remote Thread 1.1 In: asm_system_call_handler L55 PC: 0x2220e
Breakpoint 4 at 0x2220e: file ../src/utils/asm_utils.asm, line 55.
(gdb) c
Continuing.

Breakpoint 4, asm_system_call_handler () at ../src/utils/asm_utils.asm:55
(gdb) info register esp
esp             0x7b48             0x7b48
```

开中断，并进入内核态 0x7b44

```
终端
+
../src/kernel/setup.cpp
17     MemoryManager memoryManager;
18     // 系统调用
19     SystemService systemService;
20
21     int syscall_0(int first, int second, int third, int forth, int fifth)
> 22     {
23         printf("system call 0: %d, %d, %d, %d, %d\n",
24             first, second, third, forth, fifth);
25         return first + second + third + forth + fifth;
26     }
27     void syscall_move_cursor(int x,int y)
28     {
29         stdio moveCursor(x,y);
30     }

remote Thread 1.1 In: syscall_0 L22 PC: 0x20e10
Breakpoint 5 at 0x2221a: file ../src/utils/asm_utils.asm, line 61.
(gdb) si
(gdb) si
syscall_0 (first=0, second=0, third=0, forth=0, fifth=0)
at ../src/kernel/setup.cpp:22
(gdb) info register esp
esp             0x7b44             0x7b44
```

即将退出asm_system_call_handler 0x7b8c

```
终端
+
../src/utils/asm_utils.asm
58
59     add esp, 5 * 4
60
B+ 61     mov [ASM_TEMP], eax
62     popad
63     pop gs
64     pop fs
65     pop es
66     pop ds
B+> 67     mov eax, [ASM_TEMP]
68
69     iret
70 asm_system_call:

remote Thread 1.1 In: asm_system_call handler      L67  PC: 0x22226
Breakpoint 6 at 0x22226: file ../src/utils/asm_utils.asm, line 67.
(gdb) c
Continuing.

Breakpoint 6, asm_system_call_handler () at ../src/utils/asm_utils.asm:67
(gdb) info register esp
esp             0x7b8c             0x7b8c
```

回到asm_system_call 0x7b98

```
终端
+
../src/utils/asm_utils.asm
88     mov esi, [ebp + 6 * 4]
89     mov edi, [ebp + 7 * 4]
90
B+ 91     int 0x80
92
> 93     pop gs
94     pop fs
95     pop es
96     pop ds
97     pop edi
98     pop esi
99     pop edx
100    pop ecx

remote Thread 1.1 In: asm_system_call      L93  PC: 0x2224e
(gdb) info register esp
esp             0x7b8c             0x7b8c
(gdb) si
(gdb) si
asm_system_call () at ../src/utils/asm_utils.asm:93
(gdb) info register esp
esp             0x7b98             0x7b98
```

即将退出asm_system_call 0x7bc0

```
终端
+
../src/utils/asm_utils.asm
99     pop edx
100    pop ecx
101    pop ebx
B+ 102    pop ebp
103
> 104    ret
105
106 ; void asm_init_page_reg(int *directory);
107 asm_init_page_reg:
108     push ebp
109     mov ebp, esp
110
111     push eax

remote Thread 1.1 In: asm_system_call      L104 PC: 0x2225a
(gdb) c
Continuing.

Breakpoint 2, asm_system_call () at ../src/utils/asm_utils.asm:102
(gdb) ni
(gdb) info register esp
esp             0x7bc0             0x7bc0
```

回到setup.cpp 0x7bc4

```
终端
+ .../src/kernel/setup.cpp
66 // 设置2号系统调用
67 systemService.setSystemCall(2, (int)syscall_print);
68
69 int ret;
70
71 ret = asm_system_call(0);
72 printf("return value: %d\n", ret);
73
74 ret = asm_system_call(0, 123, 324, 9248, 7, 123);
75 printf("return value: %d\n", ret);
76
77 asm_system_call(1,3,3);
78

remote Thread 1.1 In: setup_kernel L71 PC: 0x20f80
(gdb) ni
(gdb) info register esp
esp 0x7bc0 0x7bc0
(gdb) ni
0x00020f80 in setup_kernel () at ../src/kernel/setup.cpp:71
(gdb) info register esp
esp 0x7bc4 0x7bc4
```

完全经过系统调用语句 0x7be4

```
终端
+ .../src/kernel/setup.cpp
66 // 设置2号系统调用
67 systemService.setSystemCall(2, (int)syscall_print);
68
69 int ret;
70
71 ret = asm_system_call(0);
72 printf("return value: %d\n", ret);
73
74 ret = asm_system_call(0, 123, 324, 9248, 7, 123);
75 printf("return value: %d\n", ret);
76
77 asm_system_call(1,3,3);
78

remote Thread 1.1 In: setup_kernel L72 PC: 0x20f86
(gdb) ni
0x00020f80 in setup_kernel () at ../src/kernel/setup.cpp:71
(gdb) info register esp
esp 0x7bc4 0x7bc4
(gdb) n
(gdb) info register esp
esp 0x7be4 0x7be4
```

进入一层函数都会压一个返回地址，过完函数后返回值与返回地址弹出栈。

在asm__system_call_handler中会有一次更换esp的动作，其余时候都是正常压栈弹栈

3，说明TSS在系统调用中的作用

TSS(任务状态段)是在操作系统进程管理的过程中，进程切换时的任务现场信息。

在系统调用时，低特权级转移到高特权级时，CPU要从TTS中把高特权级的栈地址加载到esp寄存器中

Assignment 2 Fork的奥秘

实验内容

实现fork函数，并回答以下问题。

- 请根据代码逻辑和执行结果来分析fork实现的基本思路。
- 从子进程第一次被调度执行时开始，逐步跟踪子进程的执行流程一直到子进程从fork返回，根据gdb来分析子进程的跳转地址、数据寄存器和段寄存器的变化。同时，比较上述过程和父进程执行完 ProgramManager::fork后的返回过程的异同。
- 请根据代码逻辑和gdb来解释fork是如何保证子进程的 fork返回值是0，而父进程的 fork返回值是子进程的pid。

实验原理

实验代码

实验结果

Assignment 3 哼哈二将 wait & exit

实验内容

实现wait函数和exit函数，并回答以下问题。

- 请结合代码逻辑和具体的实例来分析exit的执行过程。
- 请分析进程退出后能够隐式地调用exit和此时的exit返回值是0的原因。
- 请结合代码逻辑和具体的实例来分析wait的执行过程。
- 如果一个父进程先于子进程退出，那么子进程在退出之前会被称为孤儿进程。子进程在退出后，从状态被标记为 DEAD开始到被回收，子进程会被称为僵尸进程。请分析src/6代码实例中，实现回收僵尸进程的有效方法。

实验原理

实验代码

实验结果

3.总结
