Project 1: Fantasy Ptrace

概述

ptrace是process和trace的简写，直译为进程跟踪。它提供了一种使父进程得以监视和控制其子进程的方式，它还能够改变子进程中的寄存器和内核映像，因而可以实现系统调用的跟踪和断点调试。

Deadline

3.28日下午18:00

使用方法

64位 Ubuntu

妥。可以直接运行

64位 MAC OSX

不妥。解决方案:

1. 虚拟机

- Parallels 软件 葡萄下载： https://pt.sjtu.edu.cn/details.php?id=140821
- 安装好后在软件中安装一个Ubantu 16.04，点击就装

2. 悄悄使用本可爱的助教实验室的服务器

使用ssh链接到服务器，密码是 `system2018`

[注意]

禁止查看和编辑服务器上别人的代码

跑完后记得清理掉自己的代码哦 `rm -r ~/system/Yourname`

密码不要外传

不要运行project1以外的程序

```
ssh jinning@anl.sjtu.edu.cn
mkdir ~/system/Yourname
```

然后`exit`回到本机
传送你的文件到服务器上的文件夹

```
scp -r YourFilePath jinning@anl.sjtu.edu.cn:~/system/Yourname
```

然后连接到服务器上开始玩耍

```
ssh jinning@anl.sjtu.edu.cn
cd ~/system/Yourname
```

64位 Windows

不妥。解决方案：

1. 虚拟机

- 自己找找看安装包哦 paralells 或 VMWare

2. 悄悄使用本可爱的助教实验室的服务器

- 想法子用一些windows的软件ssh连接上去玩耍，和 上面的64位 MAC OSX 2. 步骤一样。
  例如可以使用putty:
  https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

  ```
  putty -ssh jinning@anl.sjtu.edu.cn
  mkdir ~/system/Yourname
  ```

  以及

  ```
  pscp -r YourFilePath jinning@anl.sjtu.edu.cn:~/system/Yourname
  ```

32位机器

不妥。请想办法使用本可爱的助教实验室的服务器，或者把代码改写成32位机子能运行的代码。

Task 1 输出反转

准备知识

进程

程序是一个静态的概念,它只是一些预先编译好的指令和数据集合的一个文件;而进程是一个动态的概念,它是程序运行时的一个过程。

创建进程

```
#include <unistd.h>
pid_t fork(void);

// 返回值:子进程返回0,父进程返回子进程ID,出错返回-1
```

fork函数被调用一次，却能够返回两次，它可能有三种不同的返回值：

- 在父进程中，fork返回新创建子进程的进程ID;
- 在子进程中，fork返回0;
- 如果出现错误，fork返回-1;

在fork函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。可以通过fork返回的值来判断当前进程是子进程还是父进程。

系统调用

在现代的操作系统里,程序运行的时候，本身是没有权利访问多少系统资源的。由于系统有限的资源有可能被多个不同的应用程序同时访问，因此，如果不加以保护，那么各个应用程序难免产生冲突。所以现代操作系统都将可能产生冲突的系统资源给保护起来，阻止应用程序直接访问。这些资源包括文件、网络、IO、各种设备等。
每个操作系统都会提供一套接口，来封装对系统资源的调用，这套接口就是系统调用。

操作系统把进程空间分为了用户空间和内核空间，系统调用是运行在内核空间的，而应用程序基本都是运行在用户空间的。应用程序想要访问系统资源，就必须通过系统调用。用户空间的应用程序要想调用内核空间的系统调用，就需要从用户空间切换到内核空间，这一般是通过中断来实现的。什么是中断呢？中断是一个硬件或者软件发出的请求，要求CPU暂停当前的工作转手去处理更加重要的事情。中断一般具有两个属性，中断号和中断处理程序。在内核中，有一个叫做中断向量表的数组来存放中断号和中断处理程序。当中断到来的时候，CPU会根据中断号找到对应的中断处理程序，并调用它。中断处理程序执行完成后，CPU会继续执行之前的代码。

通常意义上，中断有两种类型，一种称为硬件中断，这种中断来自于硬件的异常或其他事件的发生；另一种称为软件中断，软件中断通常是一条指令(i386下是int),带有一个参数记录中断号。linux系统使用`int 0x80`来触发所有的系统调用，和中断一样，系统调用带有一个系统调用号，这个系统调用就像身份标识一样来表明是哪一个系统调用。32位下，这个系统调用号会放在eax寄存器中。64位下，这个系统调用号会放在rax寄存器中。

系统调用号表：http://blog.csdn.net/sinat_26227857/article/details/44244433

32位下如果系统调用有一个参数，那么参数通过ebx寄存器传入，x86下linux支持的系统调用参数至多有6个，分别使用6个寄存器来传递，它们分别是ebx、ecx、edx、esi、edi和ebp。64位下略有不同。

例如对于`write`调用,`ebx(64位为rdi)`存放的是fd;`ecx(64位为rsi)`存放的是字符串数据的地址;`edx(64位为rdx)`存放的是字符串长度。

触发系统调用后，CPU首先需要切换堆栈，当程序的当前栈从用户态切换到内核态后，会找到系统调用号对应的调用函数，它们都是以"sys_"开头的，当执行完调用函数后，返回值会存放在eax(rax)寄存器返回到用户态。

信号

信号是在软件层次上对中断机制的一种模拟，它是一种进程间异步通信的机制。

一个进程要发信号给另一个进程，可以使用这些函数：kill()、raise()、 sigqueue()、alarm()、setitimer()以及abort()。它其实是通过系统调用把信号先发给内核。当另一个进程从内核态回用户态的时候，它会先去找一下有没有发给自己的信号，如果有，就处理掉。

进程可以通过三种方式来响应一个信号：

- 忽略信号，即对信号不做任何处理;
- 捕捉信号。通过signal()定义信号处理函数，当信号发生时，执行相应的处理函数;
- 执行缺省操作，Linux对每种信号都规定了默认操作。

ptrace函数

函数原型如下：

```
#include <sys/ptrace.h>
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *
data);
```

ptrace有四个参数:

- enum __ptrace_request request：指示了ptrace要执行的命令。
- pid_t pid: 指示ptrace要跟踪的进程。
- void *addr: 指示要监控的内存地址。
- void *data: 存放读取出的或者要写入的数据。
  request参数决定了ptrace的具体功能：

1.PTRACE_TRACEME

`ptrace(PTRACE_TRACEME,0 ,0 ,0)`

描述：子进程使用，使得本进程被其父进程所跟踪。

2.PTRACE_PEEKTEXT, PTRACE_PEEKDATA

`ptrace(PTRACE_PEEKDATA, pid, addr, data)`

描述：从内存地址中读取一个字，数据地址由函数返回,pid表示被跟踪的子进程，内存地址由addr给出，data参数被忽略。

3.PTRACE_POKETEXT, PTRACE_POKEDATA

`ptrace(PTRACE_POKEDATA, pid, addr, data)`

描述：往内存地址中写入一个字。pid表示被跟踪的子进程，内存地址由addr给出，data为所要被写入的数据的地址。

4.PTRACE_PEEKUSER

`ptrace(PTRACE_PEEKUSER, pid, addr, data)`

描述：从USER区域中读取一个字节，pid表示被跟踪的子进程，addr表示读取数据在USER区域的偏移量，返回值为函数返回值，data参数被忽略。

5.PTRACE_POKEUSER

`ptrace(PTRACE_POKEUSER, pid, addr, data)`

描述：往USER区域中写入一个字节，pid表示被跟踪的子进程，USER区域地址由addr给出，data为需写入的数据。

6.PTRACE_CONT

`ptrace(PTRACE_CONT, pid, 0, signal)`

描述：继续执行。pid表示被跟踪的子进程，signal为0则忽略引起调试进程中止的信号，若不为0则继续处理信号signal。

7.PTRACE_SYSCALL

`ptrace(PTRACE_SYSCALL, pid, 0, signal)`

描述：继续执行。pid表示被跟踪的子进程，signal为0则忽略引起调试进程中止的信号，若不为0则继续处理信号signal。与PTRACE_CONT不同的是进行系统调用跟踪。在被跟踪进程继续运行直到调用系统调用开始或结束时，被跟踪进程被中止，并通知父进程。

8.PTRACE_KILL

`ptrace(PTRACE_KILL,pid)`

描述：杀掉子进程，使它退出。pid表示被跟踪的子进程。

9.PTRACE_SINGLESTEP

`ptrace(PTRACE_SINGLESTEP, pid, 0, signal)`

描述：设置单步执行标志，单步执行一条指令。pid表示被跟踪的子进程。signal为0则忽略引起调试进程中止的信号，若不为0则继续处理信号signal。当被跟踪进程单步执行完一个指令后，被跟踪进程被中止，并通知父进程。

10.PTRACE_ATTACH

`ptrace(PTRACE_ATTACH, pid)`

描述：跟踪指定pid 进程。pid表示被跟踪进程。被跟踪进程将成为当前进程的子进程，并进入中止状态。

11.PTRACE_DETACH
`ptrace(PTRACE_DETACH, pid)`
描述：结束跟踪。 pid表示被跟踪的子进程。结束跟踪后被跟踪进程将继续执行。

12.PTRACE_GETREGS
`ptrace(PTRACE_GETREGS, pid, 0, data)`
描述：读取寄存器值，pid表示被跟踪的子进程，data为用户变量地址用于返回读到的数据。

13.PTRACE_SETREGS
`ptrace(PTRACE_SETREGS, pid, 0, data)`
描述：设置寄存器值，pid表示被跟踪的子进程，data为用户数据地址。

> 更多详细的信息可以参考linux下man手册:
> http://linux.die.net/man/2/ptrace

Task 1

`fantasy.c`是一个简单的打印字符串的程序:

```
#include "stdio.h"

int main()
{
    printf("\nOh, Fantasy!\n");
    return 0;
}
```

它将打印出 `\nOh, Fantasy!\n` 。

现在你要做的是按照文件中的提示补全 `reverse.c` 中的代码（有 `TODO` 的地方），使得 `reverse.c` 在通过ptrace追踪 `fantasy.c` 时，一旦检测到 `fantasy.c` 调用了 `SYS_write` 系统调用的时候，通过修改系统调用的参数把write要输出的字符串反转，即 `\n!ysatnaF ,hO\n` ，再进行系统调用。当然，你也可以不按照 `reverse.c` 的模版来写，自由发挥是资瓷的。

补全好以后运行的步骤是先编译 `fantasy.c` ：

```
gcc fantasy.c -o fantasy
```

然后在同一目录下编译运行 `reverse.c` ：

```
gcc reverse.c -o reverse
./reverse
```

Task 2 单步调试

准备知识

GDB调试工具

GDB是GNU开源组织发布的一个强大的UNIX下的程序调试工具。一般来说，GDB主要帮忙你完成下面四个方面的功能：

- 启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。
- 可让被调试的程序在你所指定的调置的断点处停住。
- 当程序被停住时，可以检查此时你的程序中所发生的事。
- 动态的改变你程序的执行环境。

Task 2

可爱的助教当然不会让你手写GDB啦。你只需要写一个简单的调试工具，可以跟踪程序执行，监控程序每一条指令的运行，并输出当前指令、指令的值、`EAX` `EBX` `ECX` `EDX` 寄存器的状态。原理是通过PTRACE_PEEKUSER 和 PTRACE_GETREGS来访问寄存器，用PTRACE_PEEKTEXT来访问内存，通过PTRACE_SINGLESTEP来实现单步调试。

`interesting.s` 是要被调试的汇编程序，他的功能是先打印`Oh, \n`，然后打印`interesting!\n`。你可以打开它来看看有哪些代码。

`step.c`是不完整的单步调试工具的代码。你要做的是根据`step.c`中的要求将其补充完整，然后进行测试。`step.c`的一个输入参数`argv`是要被调试的程序路径。当然，你也可以不按照`step.c`的模版来写，自由发挥是资瓷的。

补充完整以后，首先编译一下`interesting.s`：

```
as interesting.s -o interesting.o
ld -o interesting interesting.o
```

然后编译一下`step.c`：

```
gcc step.c -o step
```

然后使用`step`来单步调试`interesting`：

```
./step interesting
```

这时应该会输出这样的结果:

```
[1] RIP = 0x00000000004000b0, Instruction = 0x00000000000004b8
EAX: 0x00000000 EBX: 0x00000000
ECX: 0x00000000 EDX: 0x00000000
Press any key to continue...
......
```

然后就一直按回车就行啦，可以观察到这个过程中四个寄存器的变化，看看和 `interesting.s` 中的是不是对应了呢:

```
...
movl $4, %eax
movl $1, %ebx
movl $s1, %ecx
movl $5, %edx
int $0x80
...
```

然后可以运行:

```
objdump -d interesting
```

来检查 `RIP` 和 `Instruction` 输出的是不是正确。

提交

把改好的 `project1` 整个文件夹压缩成 `学号-姓名-project1.zip` 发送到 `137645534@qq.com`，邮件名 `学号-姓名-project1`.

加油!

有任何困难或不明白的地方，请联系我。

黎先生
Tel: 17621782336
QQ: 137645534
email: 137645534@qq.com / lijinning@sjtu.edu.cn