Adding a system call

陈建腾, 1120213582, 单人完成
jianteng.chen@bit.edu.cn

1. 实验要求

Your task is to add a system call to xv6. It will help to start by reading `syscall.c` (the kernel side of the system call table), `user.h` (the user-level header for the system calls), and `usys.S` (the user-level system call definitions). You may add additional files to xv6 to implement this call. For more information about how xv6 implements system calls, you should read Chapter 3 of the xv6 book.

1. Exercise 1. Create a system call `int sys_wolfie(void *buf, uint size)` , which copies an ASCII art image to a user-supplied buffer, provided that the buffer is large enough. You are welcome to use an ASCII art generator, or draw your own by hand.

   If the buffer is too small, or not valid, return a negative value. If the call succeeds, return the number of bytes copied.

   You may find it helpful to review how other system calls are implemented and compiled into the kernel, such as read.
2. Exercise 2.

   You will also write a user-level application, called wolfietest.c that gets the image from the kernel, and prints it to the console.

   You will have to modify the Makefile to add your user application in the compile task.

2. 实验环境

1. Linux Environment

- OS: Ubuntu 22.04.2 LTS on Windows 10 x86_64
- Kernel: 5.15.90.1-microsoft-standard-WSL2
- Shell: zsh 5.8.1
- CPU: AMD Ryzen 7 5800H with Radeon Graphics (8) @ 3.194GHz
- GPU: af4e:00:00.0 Microsoft Corporation Device 008e
- Memory: 781MiB / 1906MiB

2. xv6 Environment

通过以下代码在Linux环境上安装xv6：

```
sudo apt install qemu-system-x86
cd ~
git clone https://github.com/mit-pdos/xv6-public.git xv6
cd xv6
make
echo "add-auto-load-safe-path $HOME/xv6/.gdbinit" > ~/.gdbinit
```

3. Start the xv6

分裂终端，在一个终端内输入以下命令：

```
make-nox-gdb
```

在另一终端内打开gdb即可。

3. 实验过程

1. 创建系统调用

在内核中合适的位置实现系统调用，即修改 `sysproc.c` ，加入系统调用函数。

```
int
sys_wolfie(void)
{
  void *buf;
  uint size;
  if (argptr(0, (void*)&buf, sizeof(buf)) < 0)
    return -1;
  if (argptr(1, (void*)&size, sizeof(size)) < 0)
    return -1;
  char image[] =
"   _    ____   ____ ___ ___   ____    _    _   _ _   _ _____ ____\n\
   / \\  / ___| / ___|_ _|_ _| | __ )  / \\  | \\ | | \\ | | ____|  _ \\\n\
  / _ \\ \\___ \\| |    | | | |  |  _ \\ / _ \\ |  \\| |  \\| |  _| | |_) |\n\
 / ___ \\ ___) | |___ | | | |  | |_) / ___ \\| |\\  | |\\  | |___|  _ < \n\
/_/   \\_\\____/ \\____|___|___| |____/_/   \\_\\_| \\_|_| \\_|_____|_| \\_\\\n";
  if (sizeof(image) > size) return -1;
  strncpy((char*)buf, image, sizeof(image));
  return sizeof(image);
}
```

2. 在`syscall.h`中加入新系统调用序号。

```
#define SYS_wolfie 22
```

3. 在 `syscall.c` 中声明新的内核调用函数，并且在 syscalls 映射表中，加入之前定义的调用函数指针的映射。

```
extern int sys_wolfie(void);

static int (*syscalls[])(void) = {
...
[SYS_wolfie]  sys_wolfie,
};
```

4. 修改 `usys.S` ，里面为system call的用户态跳板函数的编译结果

```
SYSCALL(wolfie)
```

5. 修改 `user.h` ，在用户态的头文件加入定义，使得用户态程序可以找到这个跳板入口函数。

```
int wolfie(void*, uint);
```

6. 创建用户态级别程序 `wolfietest.c`，并修改 `Makefile`

```
// wolfietest.c
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(void){
    char buf[500];
    printf(1, "sys call wolfie, return: %d\n", wolfie((void*)buf, 500));
    printf(1, "%s", buf);
    exit();
}
```

```
UPROGS=\
  ...
  _wolfietest\

EXTRA=\
  mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
  ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c wolfietest.c\
  printf.c umalloc.c\
  README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
  .gdbinit.tmpl gdbutil\
```

4. 运行结果

1. 缓冲区未溢出

2. 缓冲区溢出