Lab3 Based on xv6

Part 1. Prepare xv6.

在虚拟机中打开终端

安装 QEMU，在终端中输入

```
sudo apt-get install qemu-system
qemu-system-i386 -version
```

```
y4n@y4n-virtual-machine:~/桌面$ qemu-system-i386 --version
QEMU emulator version 6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.22)
Copyright (c) 2003-2021 Fabrice Bellard and the QEMU Project developers
```

设置 xv6，使用 git clone 命令从 GitHub 克隆 xv6 的公共仓库到本地目录，在终端中输入

```
cd ~
git clone https://github.com/mit-pdos/xv6-public.git xv6
```

进入 xv6 文件夹，使用 make 命令来编译 xv6，在终端中输入

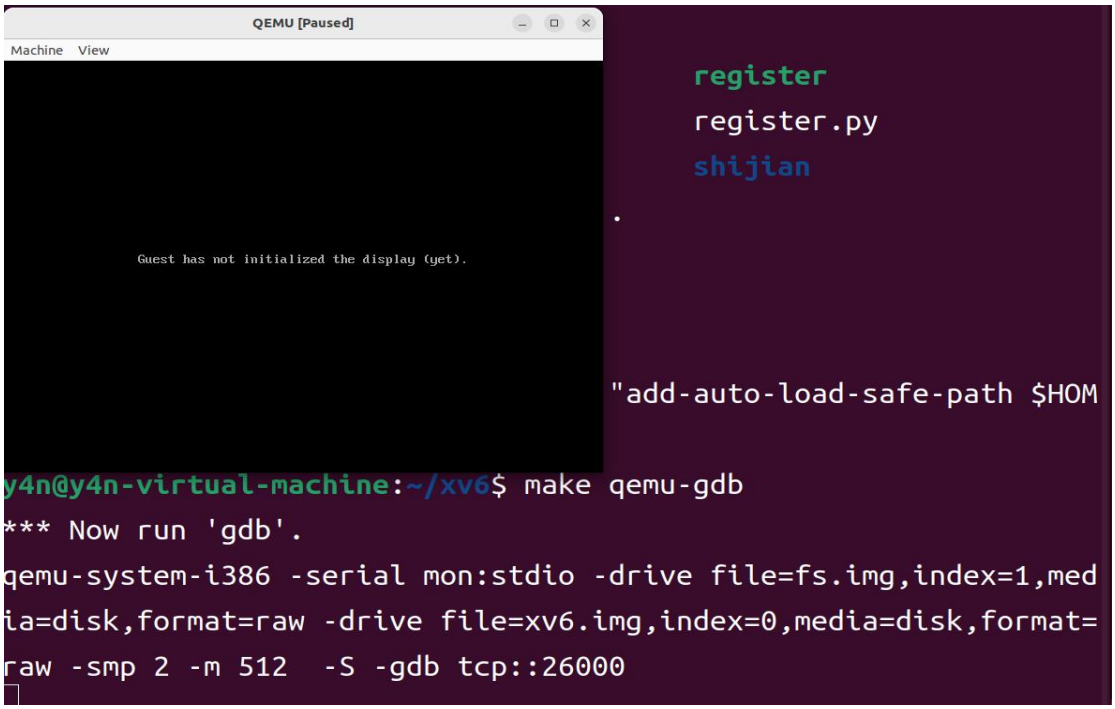
```
cd xv6
make
```

```
y4n@y4n-virtual-machine:~/xv6$ make
make: “xv6.img”已是最新。
```

允许加载本地 gdbinit（只做一次），使用 echo 命令将 add-auto-load-safe-path $HOME/xv6/.gdbinit 添加到~/.gdbinit 文件中，这样 GDB 就会自动加载 xv6 的调试配置，使用 make qemu-gdb 命令启动 QEMU，并附加 GDB 调试器，在终端中输入

```
echo "add-auto-load-safe-path $HOME/xv6/.gdbinit" > ~/.gdbinit
make qemu-gdb
```

启动 QEMU 如图



开第二个终端，在运行 make 的同一目录下运行 gdb，在终端中输入

```
gdb -q -iex "set auto-load safe-path /home/y4n/xv6"
```

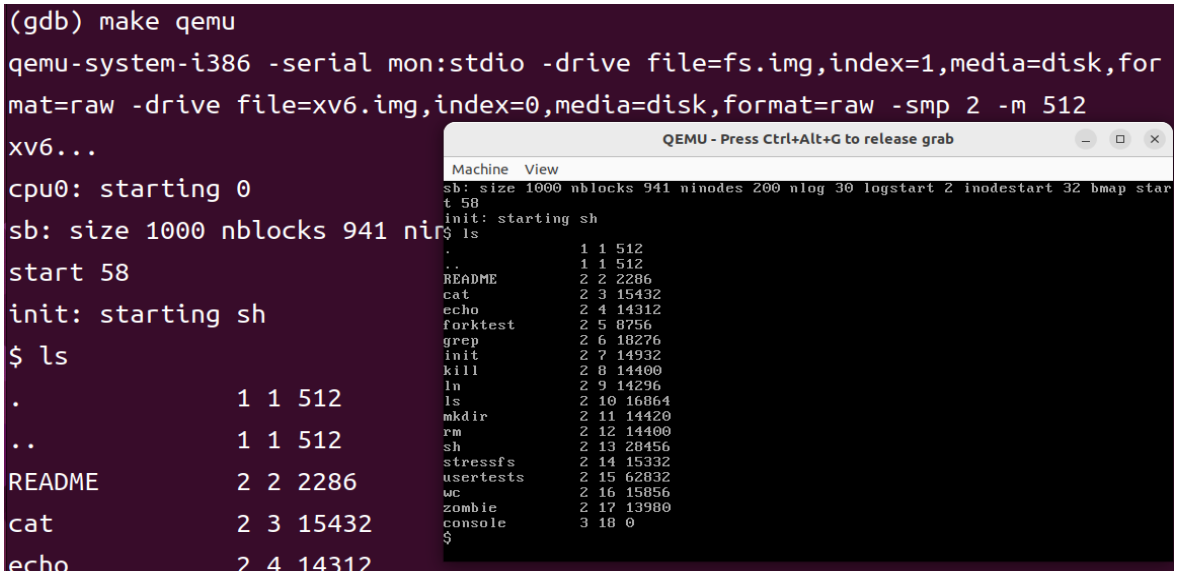
启动 GDB，并设置自动加载安全路径

在 gdb 中设置断点并继续执行

```
br * 0x0010000c
continue
```

查看文件系统

```
make qemu
ls
```



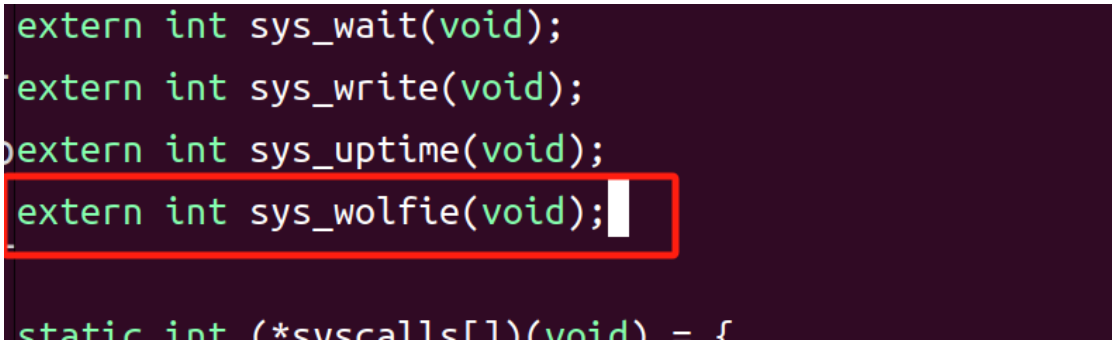
配置完成

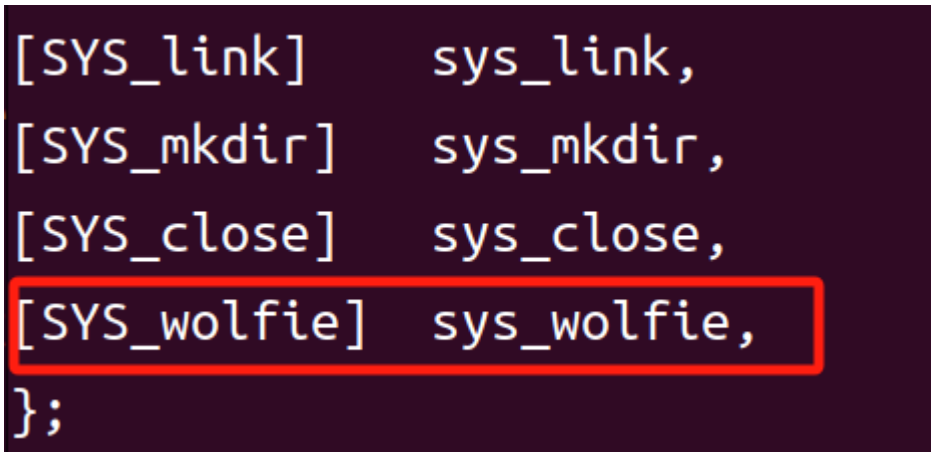
Part 2. Adding a system call

需要修改以下文件，本实验统一使用 vim 修改

1.编辑 syscall.c 文件，添加新的系统调用函数 sys_wolfie 的声明

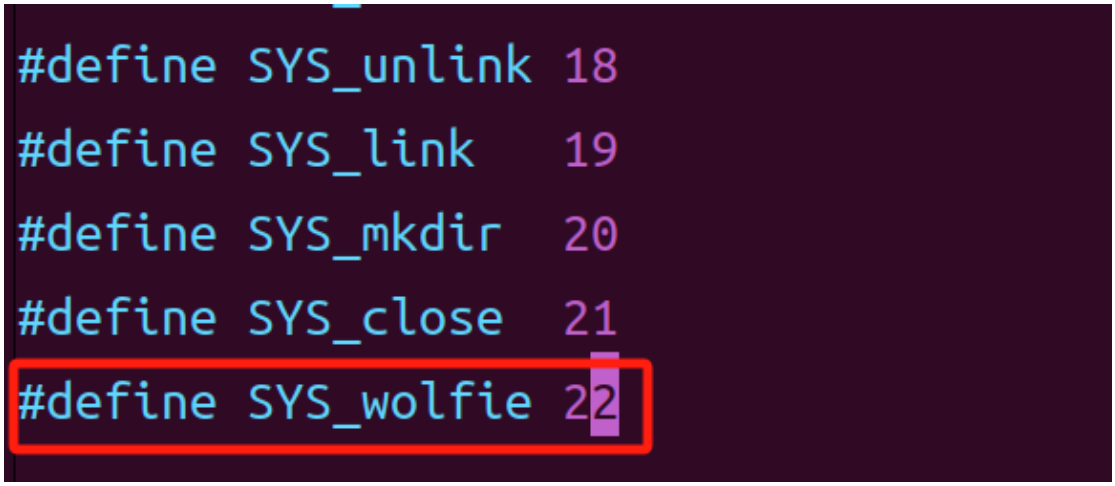
```
extern int sys_wolfie(void);
[SYS_wolfie] sys_wolfie,
```





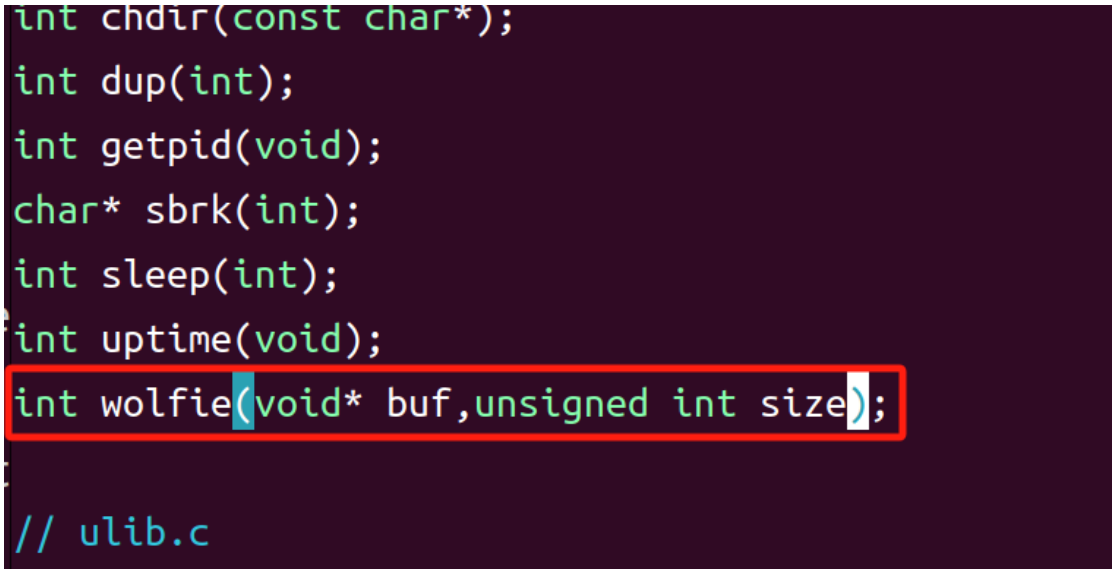
2.编辑 syscall.h 文件，定义新的系统调用编号 SYS_wolfie

```
#define SYS_wolfie 22
```



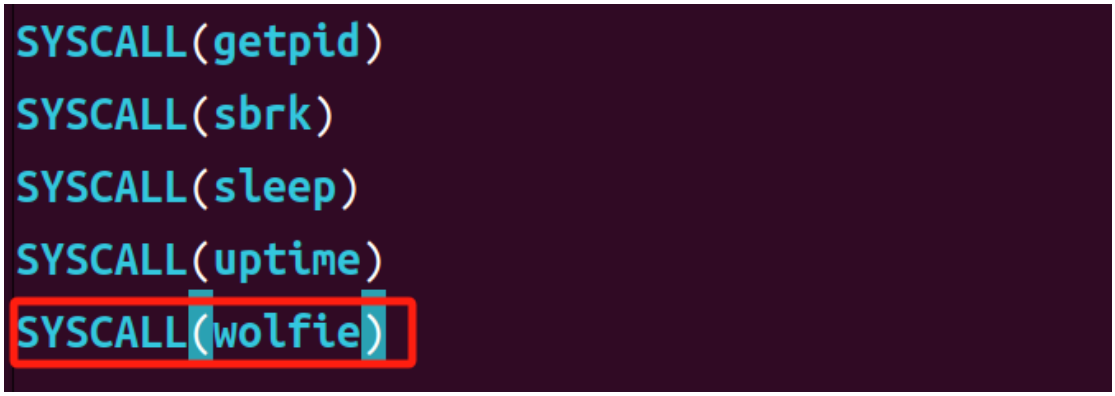
3.编辑 user.h 文件，添加用户级程序调用 wolfie 函数的声明

```
int wolfie(void* buf, unsigned int size);
```



4.编辑 usys.S 文件，添加新的系统调用的汇编代码

```
SYSCALL(wolfie)
```



5.在 sysproc.c 文件中实现 sys_wolfie 函数，它将用户空间的参数传递给内核空间的 wolfie 函数

```
int
sys_wolfie(void)
{
char* buf;
int size;
if( argint(1, &size) < 0){
return -1;
}
if( argptr(0, &buf, size) < 0){
return -1;
}
return wolfie(buf, size);
}
```

```
//new syscall
int
sys_wolfie(void)
{
  char* buf;
  int size;
  if( argint(1, &size) < 0){
    return -1;
  }
  if( argptr(0, &buf, size) < 0){
    return -1;
  }
  return wolfie(buf, size);
}
```

6. 在 proc.c 文件中实现 wolfie 函数，该函数检查传入的缓冲区大小，并在满足条件时打印 ASCII 艺术并复制到用户空间的缓冲区

```
int
wolfie(char *buf, unsigned int size)
{
  if(size < 200)
  {
    return -1;
  }
```

```
  char temp[300] = "***************************\n*      name:y4n12345
     *\n*        ASCII Art          *\n*     (y4n@y4n12345.cn)     *\n**
*************************\n";
  cprintf(temp);
  strncpy( (char*)buf, temp, size);
  return 200;
}
```

```
int
wolfie(char *buf, unsigned int size)
{
  if(size < 200)
  {
    return -1;
  }
  char temp[300] = "***************************\n*      name:y4n12345
 *\n*        ASCII Art          *\n*     (y4n@y4n12345.cn)     *\n************
****************\n";
  cprintf(temp);
  strncpy( (char*)buf, temp, size);
  return 200;
}
                                                    548,0-1       底端
```

7.编辑 Makefile 文件，在 UPROGS=\中添加代码_wolfie\以添加新的用户程序_wolfie，这样在编译时会包含这个程序

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _wolfie\
```

8.编辑 defs.h 文件，在 defs.h 文件中添加 wolfie 函数的声明，在 proc.c 中添加代码

```
int wolfie(char* buf, unsigned int size);
```

```
//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             wolfie(char* buf,unsigned int size);
```

9.编写一个用户级应用程序 wolfie.c，它调用新的系统调用 wolfie，并将结果打印到控制台，具体代码如下

```
#include "types.h"
#include "stat.h"
#include "user.h"

void show_ans(int ans)
{
  if(ans == -1)
  {
    printf(0, "the buffer is not enough\n");
  }
  else
  {
    printf(0, "the string length is %d\n", ans);
  }
  printf(0, "\n");
```

```
  return;
}

int main(int argc, char* argv[])
{
  char buf_one[1000] = {0};
  printf(0, "create a buffer which size is 1000\n");
  int ans_one = wolfie(buf_one, 1000);
  show_ans(ans_one);
  printf(0, "%s", buf_one);

  char buf_two[10] = {0};
  printf(0, "create a buffer which size is 10\n");
  int ans_two = wolfie(buf_two, 10);
  show_ans(ans_two);

  return 0;
}
```

```
#include "types.h"
#include "stat.h"
#include "user.h"

void show_ans(int ans)
{
  if(ans == -1)
  {
    printf(0, "the buffer is not enough\n");
  }
  else
  {
    printf(0, "the string length is %d\n", ans);
  }
  printf(0, "\n");
  return;
}
```

```c
int main(int argc, char* argv[])
{
  char buf_one[1000] = {0};
  printf(0, "create a buffer which size is 1000\n");
  int ans_one = wolfie(buf_one, 1000);
  show_ans(ans_one);
  printf(0, "%s", buf_one);

  char buf_two[10] = {0};
  printf(0, "create a buffer which size is 10\n");
  int ans_two = wolfie(buf_two, 10);
  show_ans(ans_two);

  return 0;
}
```

10.在终端中输入 `make qemu` 编译并运行 xv6，然后执行 wolfie 函数，可以看到第一次创建了一个 1000 个字符大小的数组，大于 200，调用系统函数后返回了长度，第二次创建了一个 10 个字符大小的数组，小于 200，调用系统函数后返回 not enough 的提示。