OS MINI PROJECT

SOURCE : XV6 OPERATING SYSTEM

TOPIC : Creation of a new system call.

INTRODUCTION TO XV6 OPERATING SYSTEM :

DESCRIPTION:

Developed by	:	Massachusetts Institute of Technology
Default user Interface	:	Command-line Interface
Kernel type	:	Monolithic Kernel
OS family	:	Unix-like
Source model	:	Open-source Software
Written in	:	C, Assembly Language

ABOUT XV6:

xv6 is a modern reimplementation of Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems.

It is used for pedagogical purposes in MIT's Operating Systems Engineering (6.828) course as well as Georgia Tech's (CS 3210) Design of Operating Systems Course, IIIT Hyderabad, IIIT Delhi and as well as many other institutions.

Xv6 is designed to be small, compact, easy to understand and modify, and similar in structure to Linux and Unix systems.

1. Operating System Interfaces:

The operating system, xv6, provides the basic interfaces introduced by Ken Thompson and Dennis Ritchie’s Unix operating system, as well as mimicking Unix’s internal design.

2. Shell type:

Its’ shell is a simple implementation of the essence of the Unix Bourne shell.

3. Process and Memory:

An xv6 process consists of user-space memory (instructions, data, and stack) and per-process state private to the kernel.

Xv6 can time-share processes: it transparently switches the available CPUs among the set of processes waiting to execute. When a process is not executing, xv6 saves its CPU registers, restoring them when it next runs the process.

4. File System:

The xv6 file system provides data files, which are uninterpreted byte arrays, and directories, which contain named references to data files and other directories. The directories form a tree, starting at a special directory called the root.

PROBLEM STATEMENT:

This project describes how to create and add a new system call to xv6 and how to execute it in the terminal.

INNOVATION:

I add a system call to get the address space size of the currently running user program. Let the name of our system call be getmysize() and call this system call from within a user program.

GENERAL PROCEDURE:

To add a system call that can be called in xv6's shell, we should do something with these five files.

sysproc.c : Add the real implementation of the method.

syscall.h : Define the position of the system call vector that connect to our implementation.

user.h : Define the function that can be called through the shell.

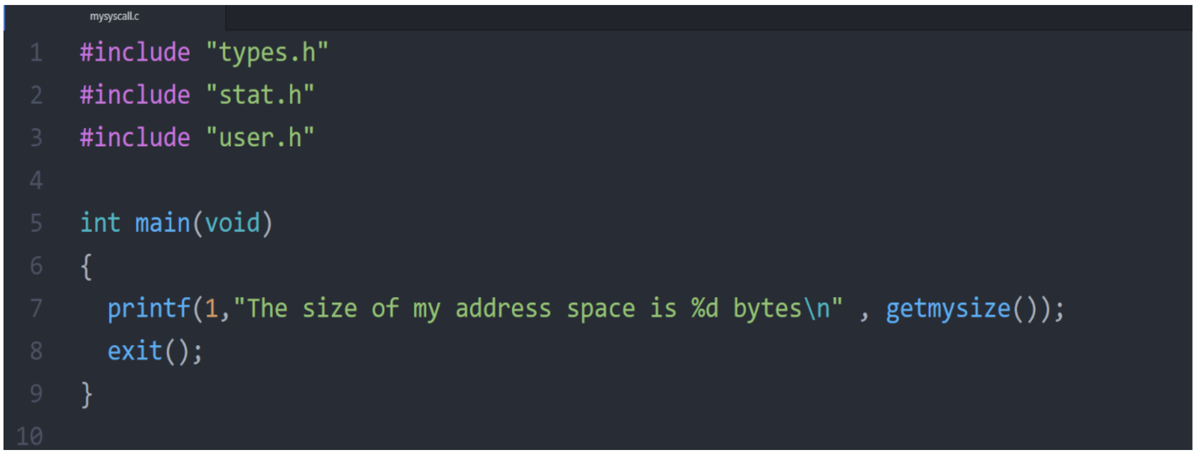
syscall.c : Externally define the function that connect the shell and the kernel, use the position defined in **syscall.h** to add the function to the system call vector.

usys.S : Use the macro 'define' , connects the call of user to the system call function.

STEPS TO BE FOLLOWED TO CREATE A NEW SYSTEM CALL:

1)Creation of user program

Let the program name be mysyscall.c



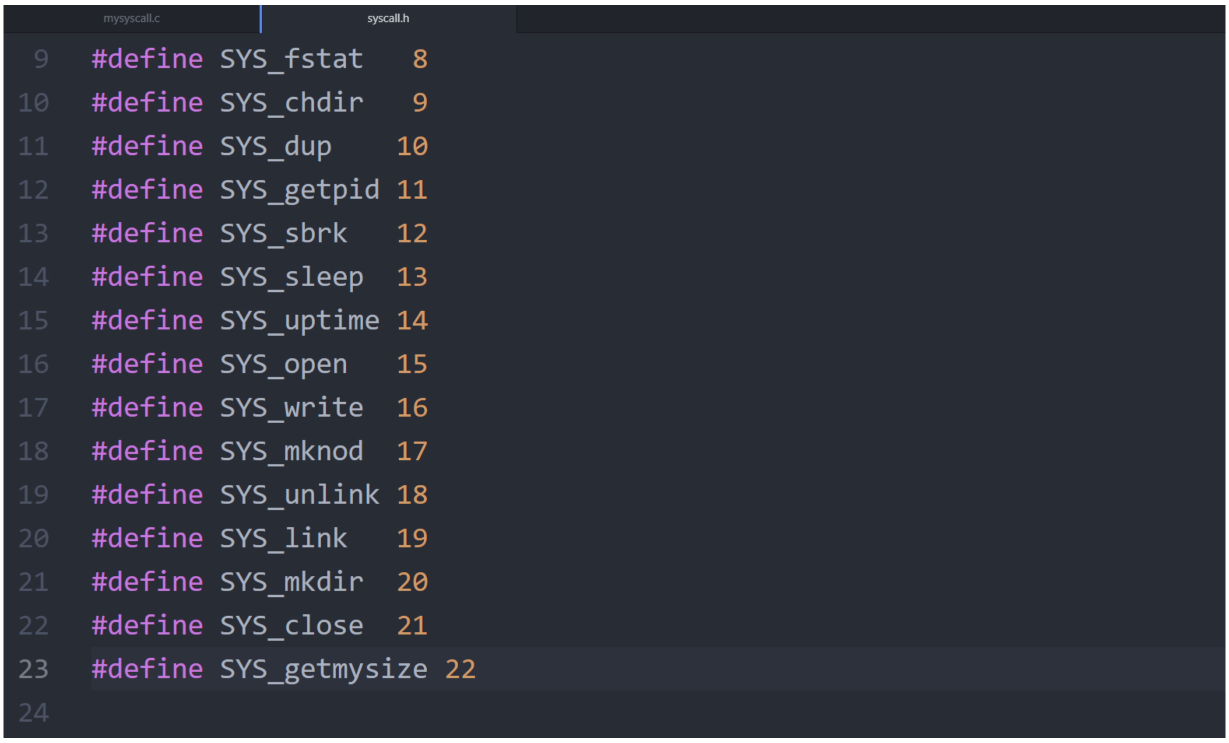
```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void)
{
  printf(1,"The size of my address space is %d bytes\n" , getmysize());
  exit();
}

```

2)Addition of the following line at the end of the file **syscall.h.**

#define SYS_getmysize 23



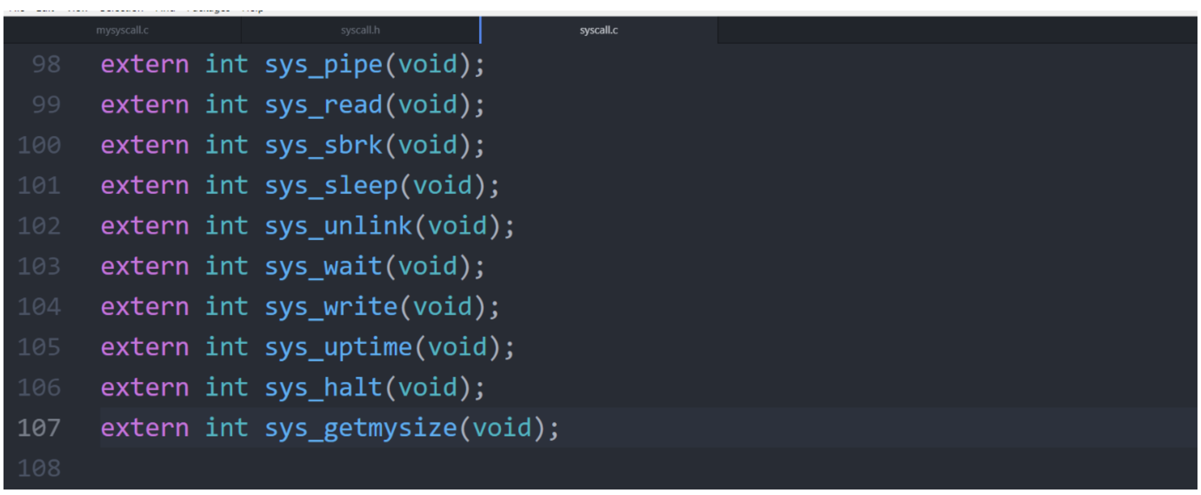
```c
#define SYS_fstat    8
#define SYS_chdir    9
#define SYS_dup     10
#define SYS_getpid  11
#define SYS_sbrk    12
#define SYS_sleep   13
#define SYS_uptime  14
#define SYS_open    15
#define SYS_write   16
#define SYS_mknod   17
#define SYS_unlink  18
#define SYS_link    19
#define SYS_mkdir   20
#define SYS_close   21
#define SYS_getmysize 22

```

Note that the 22 here might change depending on the number given before the line you are going to add in the file.

3)Now add the following lines to the syscall.c file.

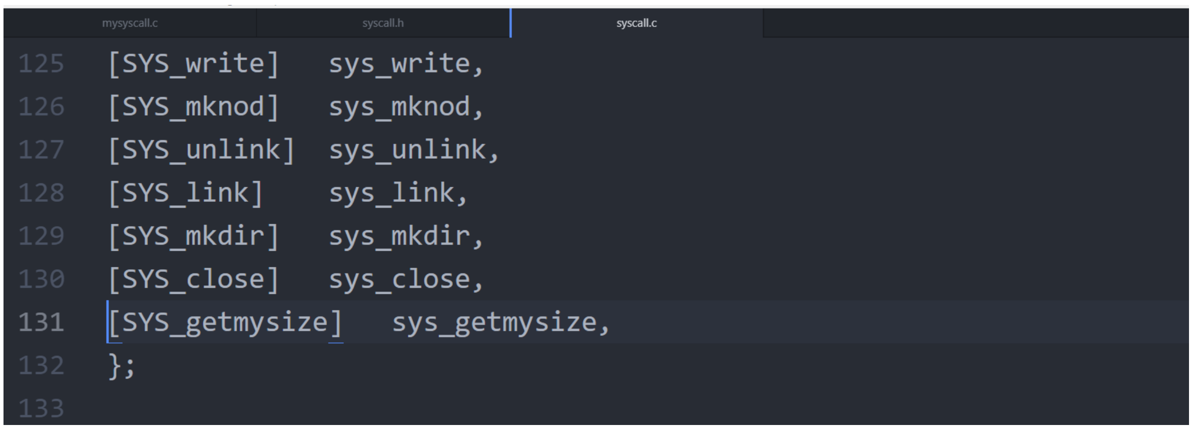
extern int sys_getmysize(void);



```
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_halt(void);
extern int sys_getmysize(void);
```

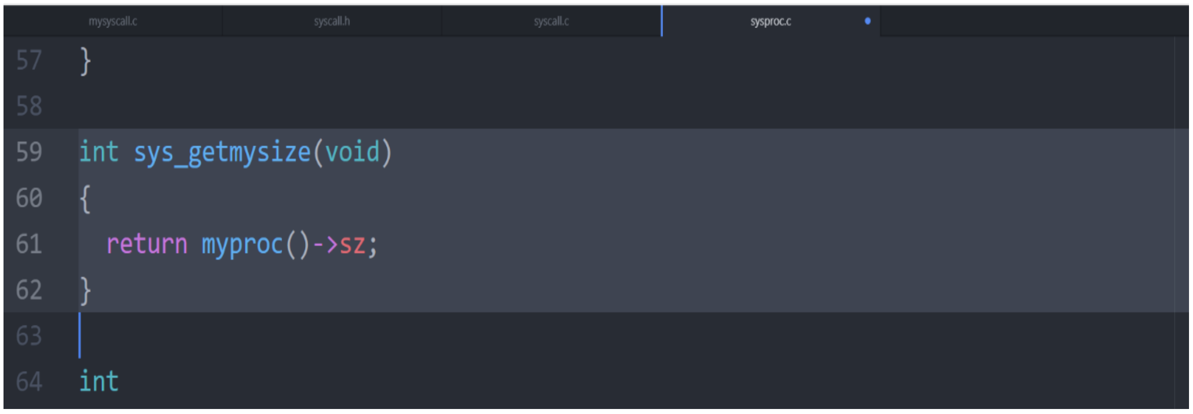
and in the array of syscalls, append the following line

[SYS_getmysize] sys_getmysize,



```
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_getmysize]   sys_getmysize,
};
```

4)In the file sysproc.c, that is where the implementation of our system call goes if it is a system call related to process handling or memory management, put the implementation of our system call as follows.



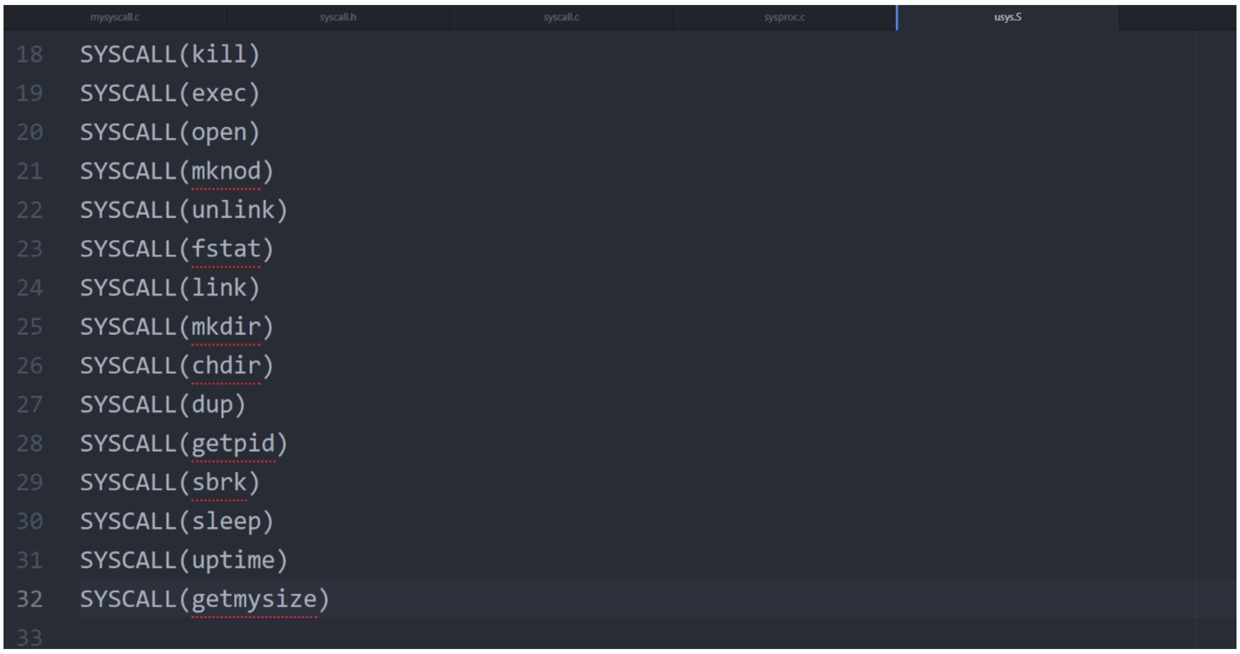
```
}

int sys_getmysize(void)
{
  return myproc()->sz;
}

int
```

5)Now in the usys.S file, add the following line,

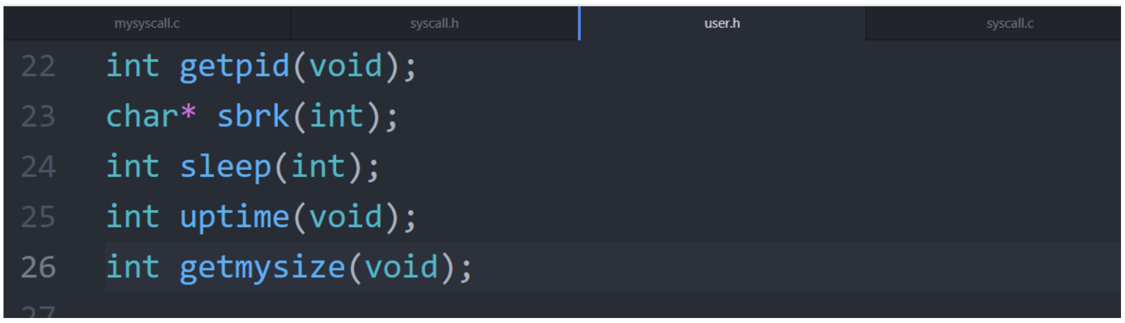
SYSCALL(getmysize)



```
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getmysize)
```

6)Then in the user.h file, add the following.

int getmysize(void);

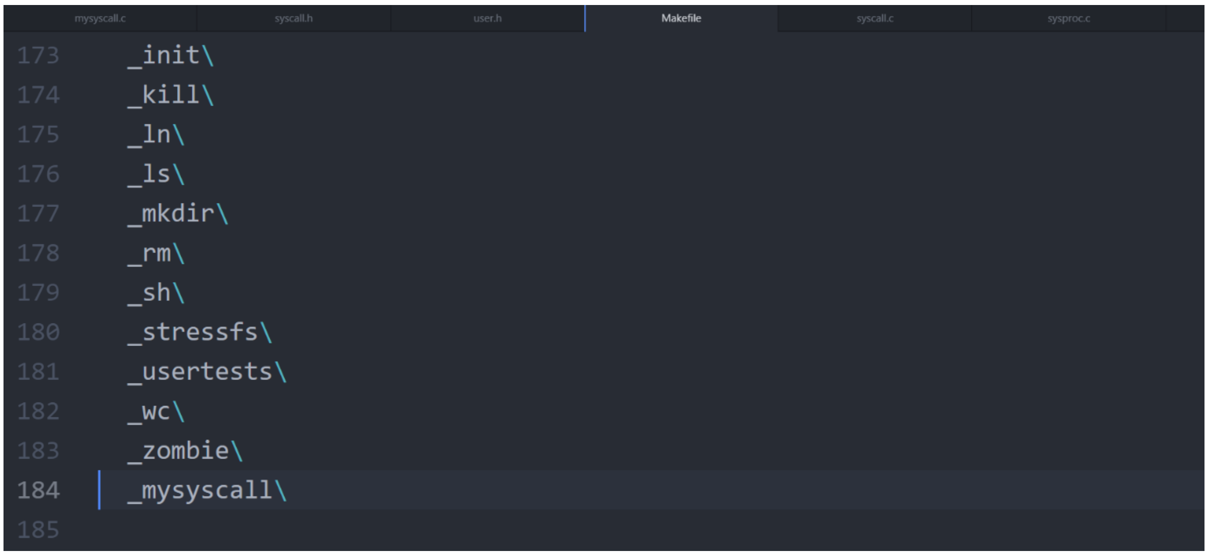


```
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getmysize(void);
```

7)Now add the user program(mysyscall.c) to xv6

Add the following lines in Makefile

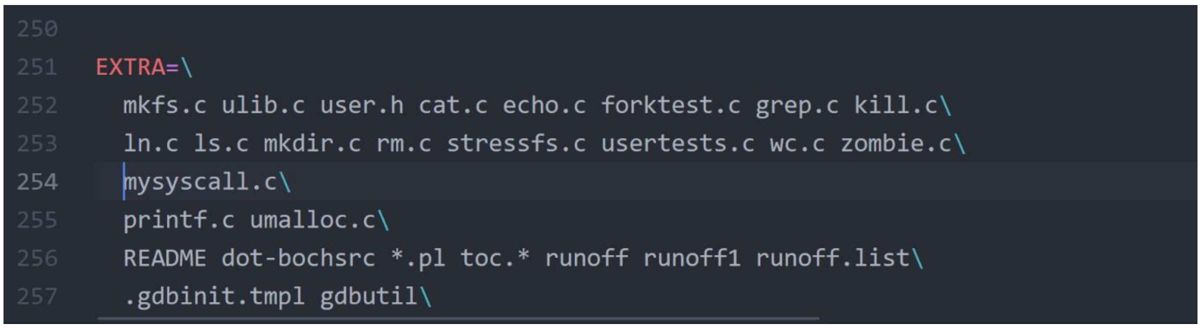
_mysyscall



```
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_mysyscall\

```

mysyscall.c



```

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\
	mysyscall.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\
```

Ok. Now the user program is ready to execute

<u>EXECUTION:</u>

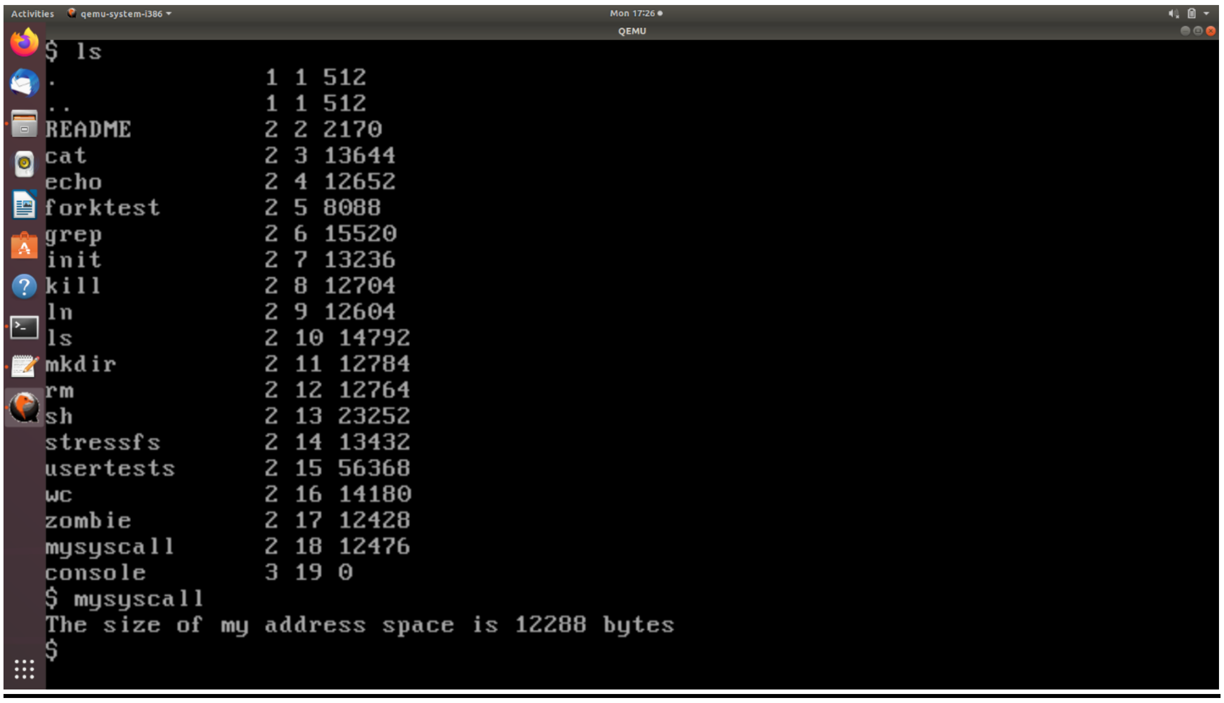
Type the following commands in the ubuntu terminal:

1) cd xv6
2) make clean
3) make (compile the whole folder xv6)
4) make qemu

Now the QEMU terminal appears,Type the user program name (mysyscall) which calls the system call, the size of the address space of this same program will be returned in **QEMU** terminal.

The output will be as the following:

OUTPUT:



```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 13644
echo           2 4 12652
forktest       2 5 8088
grep           2 6 15520
init           2 7 13236
kill           2 8 12704
ln             2 9 12604
ls             2 10 14792
mkdir          2 11 12784
rm             2 12 12764
sh             2 13 23252
stressfs       2 14 13432
usertests      2 15 56368
wc             2 16 14180
zombie         2 17 12428
mysyscall      2 18 12476
console        3 19 0
$ mysyscall
The size of my address space is 12288 bytes
$
```