Adding System call in xv6

Mahima Kothari · Follow
5 min read · Apr 16, 2020

what is xv6 ?

> *xv6 operating system is used for pedagogical purposes in MIT's Operating Systems Engineering course as well as Georgia Tech's Design of Operating Systems Course and in many other institutes. xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix Version 6 (v6). xv6 loosely follows the structure and style of v6, but is implemented for a modern x86-based multiprocessor using ANSI C.*

what is System call ?

(you can skip this if you already know about system calls)

We all know that, Operating system support two modes of execution

```
1) Kernel mode
2) User mode
```

Privileged instructions (eg. IN, OUT instruction) can be executed only in kernel mode. Where as, Unprivileged instruction or Normal instruction can be executed in user mode.

user processes are present in os in lower address space with user privileges. But the kernel code is present in higher address space with kernel privileges. Hence, Due to this privilege levels user process can't access kernel code directly.

But, user program want to access some hardware , some resources which is managed by operating system.
Eg. consider, that program has printf() . This printf() want to print the text on screen. For this we have to access the hardware and display unit of our system. And we can't allow the user process to access the hardware and resources directly, because , user processes are untrusted, they can be malicious process. Hence, this hardware and resources are managed by a trust worthy guy.. Operating System.

For this reason, there are system calls. This are supported by OS and any user program can call them to access hardware and other resources.

There are already many system calls in operating system which executes different types of task.

How to add system call in xv6 ?

There are already some system calls in xv6 operating system.Here, we see how can we add our own system call in xv6.
For adding system call in xv6,

```
we need to modify following files :
   1) syscall.c
   2) syscall.h
   3) sysproc.c
   4) usys.S
   5) user.h
```

We will add system call to print "Hello world";

1. syscall.c file

This file contain array of function pointers for all system call functions.
Add the below entry in this array .

```
[SYS_hello]  sys_hello,
```

```
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[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_hello]   sys_hello,
};
```

One more change is to be done in this file.
We are not going to write implementation of our system call in syscall.c file.
We will write it in different file. Hence, we will add the function prototype in this file.

```
extern int sys_hello(void);
```

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(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_hello(void);
```

2. syscall.h

There is array of function pointers in file syscall.c To index in this array , we will define number of system call in syscall.h file. This number is used for indexing in that array of function pointers.

```
#define SYS_hello 22
```

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#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_hello  22
~
```

3. sysproc.c

We will implement our system call in this file.

```
int
sys_hello(void) {
   cprintf(“Hello world\n”);
   return 12;
}
```

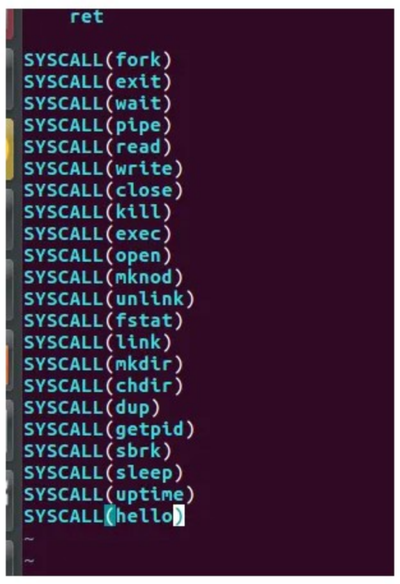
```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"


int
sys_hello(void)
{
  cprintf("Hello World\n");
  return 12;
}

int
sys_fork(void)
{
  return fork();
}
```

This is basic implementation of system call. We are returning just dummy value for testing purpose. For doing something more, like passing arguments to sysetem call, refer other system calls written already in sysproc.c. Eg. for passing int as an argument to system all, refer “kill system call “.

4. usys.S

For user program to able to call this system call, interface need to be added. This interface is added in usys.S file. (extension for assembly files .S)

```
SYSCALL(hello)
```



5. user.h

Now, we need to add function prototype which user program will call. This is added in user.h file.

```
int hello(void);
```

This function is mapped to system call from the array of system call defined in file syscall.c with the index 22 which is defined in syscall.h.

Now, we have successfully added system call in xv6. We need to write small user program to call this system call.

User Program named hello.c :

```
#include “types.h”
#include “stat.h”
#include “user.h”

int
main(void) {
    printf(1, “return val of system call is %d\n”, hello());
    printf(1, “Congrats !! You have successfully added new system
call in xv6 OS :) \n”);
    exit();
 }
```

One last step for running this user program, we need to modify the Makefile. Add the filename of user program without file extension in UPROGS section of Makefile.
In Makefile,

```
In UPROGS=\
 ….
 …
 …
 _hello\
```

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

fs.img: mkfs README $(UPROGS)
        ./mkfs fs.img README $(UPROGS)
```

Run xv6 & test your newly added system call.
How to run xv6 & this user program ? (For those who don't know how to run xv6 OS.)
For running xv6, open the terminal in the folder where you have xv6 code.
Run the following command in terminal

```
make qemu-nox
```

```
$ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

Now, Run "ls" command to see list of user programs (Here, you should see program "hello", if you have successfully done above changes in xv6 source code).

```
ls
```

```
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 n
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 13436
echo           2 4 12508
forktest       2 5 8232
grep           2 6 15260
init           2 7 13100
kill           2 8 12552
ln             2 9 12460
ls             2 10 14676
mkdir          2 11 12572
rm             2 12 12556
sh             2 13 23196
stressfs       2 14 13228
usertests      2 15 56112
wc             2 16 14088
number         2 17 12744
zombie         2 18 12284
hello          2 19 12484
console        3 20 0
$
```

Run the command "hello" to run our user program and see the output of system call hello , that we have added.

```
hello
```

```
console        3 20 0
$ hello
Hello World
return val of system call is 12
Congrats !! You have successfully added new system call in xv6 OS :)
$
```

Congrats, You have done little hack with xv6 operating system. Keep exploring xv6. Feel Free to command here if you have any doubt.

Xv6 System Call Operating Systems

55 1

M

Written by Mahima Kothari

3 Followers

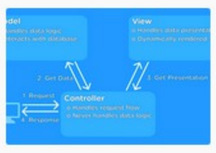
Computer Engineering

Follow

More from Mahima Kothari

Mahima Kothari

What is MVC Architecture ?



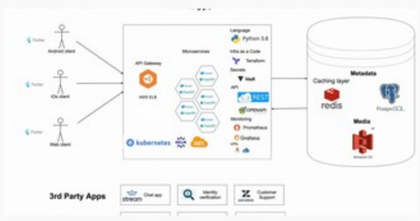
MVC stands for Model View Controller. MVC is an an architectural pattern that separates an application into three main logical components...

3 min read · Apr 23, 2020

1

See all from Mahima Kothari

Recommended from Medium



Dmitry Kruglov in Better Programming

The Architecture of a Modern Startup

Hype wave, pragmatic evidence vs the need to move fast

16 min read · Nov 7, 2022

5.4K 48

Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my...

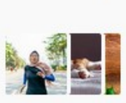
· 4 min read · Feb 16, 2022

61K 914

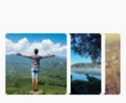
Lists

Staff Picks
424 stories · 254 saves

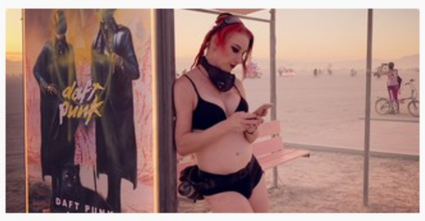
Stories to Help You Level-Up at Work
19 stories · 197 saves



Self-Improvement 101
20 stories · 509 saves



Productivity 101
20 stories · 485 saves



The Bold Italic in The Bold Italic

Why I'm breaking up with Burning Man

Adriana Roberts is a well-known burner who has gone for 30 years and needs to quit the...

7 min read · Aug 23

10.5K 168

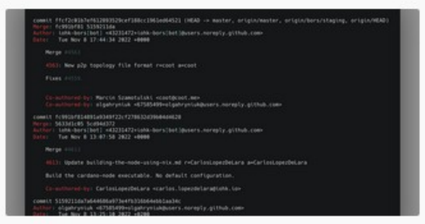
Love Shar... in ByteByteGo System Design Allian...

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However,...

· 9 min read · Apr 20

7.7K 53



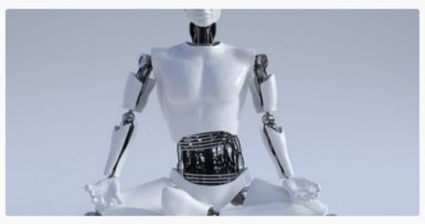
Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

· 4 min read · Nov 15, 2022

9.1K 100



The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of...

Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

31K 568

See more recommendations