

ADDING SYSTEM CALL, IMPLEMENTATION OF
PRIORITY SCHEDULER AND
COPY ON WRITE FORK IN XV6

REVIEW REPORT

Submitted by

Ankith Lagupudi (19BCE0478)

Simbothula Varun Kumar (19BCI0050)

Sighakolli Susmitha (19BCB0056)

Prepared For

OPERATING SYSTEMS (CSE2005)

PROJECT COMPONENT

School of Computer Science and Engineering



AIM:

The aim of the project is to add a system calls, to implement priority scheduler, copy on write fork on Operating system. Xv6 operating system, obtained from the github library, would be used, and we aim to add simple system calls into the existing code using an emulator (QEMU).

Abstract:

System calls are interfaces between the user and the services of the Operating system. They are required to execute processes that require memory spaces/ file accessing/ device connection etc. However, developers themselves don't have direct access to system calls. System calls are used for process control, file manipulation, device management, communication and information management. In this project, we use a type 2 hypervisor (QEMU) to run a pre-existing Operating system Xv6 . We then add the code we develop for executing system calls to the operating system.

OPERATING SYSTEM USED:

Xv6 is a simple Operating system developed by MIT for its own Operating Systems course. It is developed from the sixth edition of Unix and is coded in the C language. It is an Open Source Software and is freely available for us to develop upon.

EMULATOR USED: QEMU

QEMU:

QEMU stands for quick emulator. QEMU is a generic open source machine emulator and virtualizer. It is able to emulate other operating systems on another operating systems. The performance of QEMU is far better than virtual box. It emulates the machine's processor through dynamic binary translation (where sequences of instructions are translated from a source instruction to the target instruction set).

Literature Review

1. PB Hansen - 1973 - dl.acm.org Operating system Principles

This introductory book was really helpful as it helped us get a grasp of the operating systems concept and explained the function of operating system. It elaborates on the principles it is built on, the main one being enforcing behavioural rules on users to enable sharing of computer systems. Safe methods that help in making large programs super efficient are explored and help us get a grasp on how different operating systems operate on the basis of the same fundamental principles. It explains sequential and concurrent processes and also helps us get a solid foundation on the various theory concepts that we learnt - resource sharing and management, scheduling algorithms etc.

2. M Barabanov - 1997 - yodaiken.com- [A linux-based real-time operating system](#)

This book explores the idea of a real time linux. With respect to our project it helped us understand interrupt controllers and emulation as the OS functions on interrupt control emulation and user defined schedulers. It helped us understand better the concepts of Scheduling and interprocess communication that are critical for the functioning of any OS.

3. R Cox, [MF Kaashoek](#), [R Morris](#)- xv6: a simple, Unix-like teaching operating system

This was one of the most important and useful resource for our project. It explains the working of xv6 and concepts of operating system and how they are employed in xv6. Interfaces, organization, page tables, traps and device drivers, locking, file systems all concepts are completely elaborated and code snippets from xv6 are used to explain them. This helped us in inserting and executing the system call concept.

4. M. Nakajima and S. Oikawa, "Effective I/O Processing with Exception-Less System Calls for Low-Latency Devices," 2015 Third International Symposium on Computing and Networking (CANDAR), Sapporo, 2015, pp. 604-606, doi: 10.1109/CANDAR.2015.91.

Latency is the turnaround time of execution of a request. To make our system calls more

effective, we referred to this document. However, while it was very insightful and helped us to learn about the future of input output systems, the execution process remained advanced for our execution.

5. S. Oikawa, "Delegating the kernel functions to an application program in UV6," 2012 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2012), Hong Kong, 2012, pp. 406-409, doi: 10.1109/ICSPCC.2012.6335626. UV6 improves upon xv6 and this paper clearly shows how assigning kernel functions to application programs helps in significantly increasing processor utilization and efficiency of the operating system increases multifold. It helped us to get new ideas to implement interrupts and system calls. The ideas also allowed us to think further about the future prospects of proxy kernels.

6. Copy on write file system consistency and block usage-[David HitzMichael MalcolmJames LauByron Rakitzis](#)

This paper significantly helped us realize the successful implementation of copy on write . It explains the importance of file system snapshots that help in preserving disk space while giving perfect information about the inode file without cloning it.

7. Systems and methods for adaptive copy on write- [Darren P. SchackEric M. LemarNeal T. Fachan](#)

This paper helped us dive deep into the concept of copy on write. It introduced to us the idea of poin-in-time-copy thats another way of recording file system state and taking the snapshot. It shows how various physically distributed systems can efficiently use copy on write function.

INSTALLATION PROCEDURE:

UPDATING: Before installing anything, we have to make sure that the ubuntu operating system is up to date. The Updated operating system makes our work easier and keeps our PC secured.**COMMAND:** sudo apt-get update. This

command is used to update ubuntu operating system

SCREENSHOT:

```
susmitha@susmitha-VirtualBox: ~  
susmitha@susmitha-VirtualBox:~$ sudo apt-get update  
[sudo] password for susmitha:  
Hit:1 http://in.archive.ubuntu.com/ubuntu focal InRelease  
Get:2 http://in.archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]  
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]  
Get:4 http://in.archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]  
Get:5 http://in.archive.ubuntu.com/ubuntu focal/main DEP-11 48x48 Icons [98.4 kB]  
Get:6 http://in.archive.ubuntu.com/ubuntu focal/main DEP-11 64x64 Icons [163 kB]  
Get:7 http://in.archive.ubuntu.com/ubuntu focal/main DEP-11 64x64@2 Icons [15.8 kB]  
Get:8 http://in.archive.ubuntu.com/ubuntu focal/universe DEP-11 48x48 Icons [3,816 kB]  
Get:9 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [174 kB]  
Get:10 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [68.6 kB]  
Get:11 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [62.8 kB]  
Get:12 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [24.3 kB]  
Get:13 http://security.ubuntu.com/ubuntu focal-security/main DEP-11 48x48 Icons [11.8 kB]  
Get:14 http://security.ubuntu.com/ubuntu focal-security/main DEP-11 64x64 Icons [16.5 kB]  
Get:15 http://security.ubuntu.com/ubuntu focal-security/main DEP-11 64x64@2 Icons [29 B]  
Get:16 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [4,508 B]  
Get:17 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [31.7 kB]  
Get:18 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [8,312 B]  
Get:19 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [56.0 kB]  
Get:20 http://security.ubuntu.com/ubuntu focal-security/universe i386 Packages [26.9 kB]  
Get:21 http://security.ubuntu.com/ubuntu focal-security/universe Translation-en [29.4 kB]  
Get:22 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [52.3 kB]  
Get:23 http://security.ubuntu.com/ubuntu focal-security/universe DEP-11 48x48 Icons [15.4 kB]  
Get:24 http://security.ubuntu.com/ubuntu focal-security/universe DEP-11 64x64 Icons [22.5 kB]  
Get:25 http://security.ubuntu.com/ubuntu focal-security/universe DEP-11 64x64@2 Icons [29 B]  
Get:26 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [2,388 B]  
Get:27 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [1,172 B]  
Get:28 http://in.archive.ubuntu.com/ubuntu focal/universe DEP-11 64x64 Icons [7,794 kB]  
Get:29 http://in.archive.ubuntu.com/ubuntu focal/universe DEP-11 64x64@2 Icons [44.3 kB]  
Get:30 http://in.archive.ubuntu.com/ubuntu focal/multiverse DEP-11 48x48 Icons [23.1 kB]  
Get:31 http://in.archive.ubuntu.com/ubuntu focal/multiverse DEP-11 64x64 Icons [192 kB]
```

```
susmitha@susmitha-VirtualBox: ~  
Get:31 http://in.archive.ubuntu.com/ubuntu focal/multiverse DEP-11 64x64 Icons [192 kB]  
Get:32 http://in.archive.ubuntu.com/ubuntu focal/multiverse DEP-11 64x64@2 Icons [244 B]  
Get:33 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [343 kB]  
Get:34 http://in.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [203 kB]  
Get:35 http://in.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [130 kB]  
Get:36 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [204 kB]  
Get:37 http://in.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 48x48 Icons [48.9 kB]  
Get:38 http://in.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 64x64 Icons [74.3 kB]  
Get:39 http://in.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 64x64@2 Icons [29 B]  
Get:40 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [8,884 B]  
Get:41 http://in.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [31.9 kB]  
Get:42 http://in.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [8,340 B]  
Get:43 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [163 kB]  
Get:44 http://in.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [88.7 kB]  
Get:45 http://in.archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [82.8 kB]  
Get:46 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [184 kB]  
Get:47 http://in.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 48x48 Icons [93.4 kB]  
Get:48 http://in.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 64x64 Icons [161 kB]  
Get:49 http://in.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 64x64@2 Icons [29 B]  
Get:50 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [5,404 B]  
Get:51 http://in.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [11.6 kB]  
Get:52 http://in.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [2,468 B]  
Get:53 http://in.archive.ubuntu.com/ubuntu focal-updates/multiverse DEP-11 48x48 Icons [29 B]  
Get:54 http://in.archive.ubuntu.com/ubuntu focal-updates/multiverse DEP-11 64x64 Icons [2,638 B]  
Get:55 http://in.archive.ubuntu.com/ubuntu focal-updates/multiverse DEP-11 64x64@2 Icons [29 B]  
Get:56 http://in.archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [3,092 B]  
Get:57 http://in.archive.ubuntu.com/ubuntu focal-backports/universe i386 Packages [2,264 B]  
Get:58 http://in.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [1,976 B]  
Get:59 http://in.archive.ubuntu.com/ubuntu focal-backports/universe DEP-11 48x48 Icons [2,809 B]  
Get:60 http://in.archive.ubuntu.com/ubuntu focal-backports/universe DEP-11 64x64 Icons [3,943 B]  
Get:61 http://in.archive.ubuntu.com/ubuntu focal-backports/universe DEP-11 64x64@2 Icons [29 B]  
Get:62 http://in.archive.ubuntu.com/ubuntu focal-backports/universe amd64 c-n-f Metadata [224 B]
```

QEMU INSTALLATION:

COMMAND:

Sudo apt-get install qemu

SCREENSHOT:

```
susmitha@susmitha-VirtualBox: ~$ sudo apt-get install qemu
[sudo] password for susmitha:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm9 python3-click python3-colorama
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  qemu
0 upgraded, 1 newly installed, 0 to remove and 146 not upgraded.
Need to get 15.9 kB of archives.
After this operation, 122 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 qemu amd64 1:4.2-3ubuntu6.4 [15.9 kB]
Fetched 15.9 kB in 1s (28.6 kB/s)
Selecting previously unselected package qemu.
(Reading database ... 185793 files and directories currently installed.)
Preparing to unpack .../qemu_1%3a4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu (1:4.2-3ubuntu6.4) ...
Setting up qemu (1:4.2-3ubuntu6.4) ...
susmitha@susmitha-VirtualBox: ~$
```

COMMAND:

sudo apt get-install qemu-kvm

```
susmitha@susmitha-VirtualBox: ~$ sudo apt-get install qemu-kvm
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm9 python3-click python3-colorama
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  cpu-checker ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 libfdt1 libibverbs1 libiscsi7 libipmem1
  librados2 librdma1 librdmacm1 libslirp0 libspice-server1 libusbredirparser1 libvirglrenderer1 msr-tools ovmf qemu-block-extra
  qemu-system-common qemu-system-data qemu-system-gui qemu-system-x86 qemu-utils seabios sharutils
Suggested packages:
  samba vde2 debootstrap sharutils-doc bsd-mailx | mailx
The following NEW packages will be installed:
  cpu-checker ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 libfdt1 libibverbs1 libiscsi7 libipmem1
  librados2 librdma1 librdmacm1 libslirp0 libspice-server1 libusbredirparser1 libvirglrenderer1 msr-tools ovmf qemu-block-extra qemu-kvm
  qemu-system-common qemu-system-data qemu-system-gui qemu-system-x86 qemu-utils seabios sharutils
0 upgraded, 28 newly installed, 0 to remove and 146 not upgraded.
Need to get 19.7 MB of archives.
After this operation, 82.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 msr-tools amd64 1.3-3 [10.0 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu focal/main amd64 cpu-checker amd64 0.7-1.1 [6,936 B]
Get:3 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libibverbs1 amd64 28.0-1ubuntu1 [53.6 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu focal/main amd64 ibverbs-providers amd64 28.0-1ubuntu1 [232 kB]
Get:5 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 ipxe-qemu all 1.0.0+git-20190109.133f4c4-0ubuntu3.2 [930 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu focal/main amd64 ipxe-qemu-256k-compat-efi-roms all 1.0.0+git-20150424.a25a16d-0ubuntu4 [552 kB]
Get:7 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libaio1 amd64 0.3.112-5 [7,184 B]
Get:8 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libcacard0 amd64 1:2.6.1-1 [32.4 kB]
Get:9 http://in.archive.ubuntu.com/ubuntu focal/main amd64 librdmacm1 amd64 28.0-1ubuntu1 [64.9 kB]
Get:10 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libiscsi7 amd64 1.18.0-2 [63.9 kB]
Get:11 http://in.archive.ubuntu.com/ubuntu focal/main amd64 libipmem1 amd64 1.8-1ubuntu1 [63.8 kB]
Get:12 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 librados2 amd64 15.2.3-0ubuntu0.20.04.1 [3,239 kB]
```

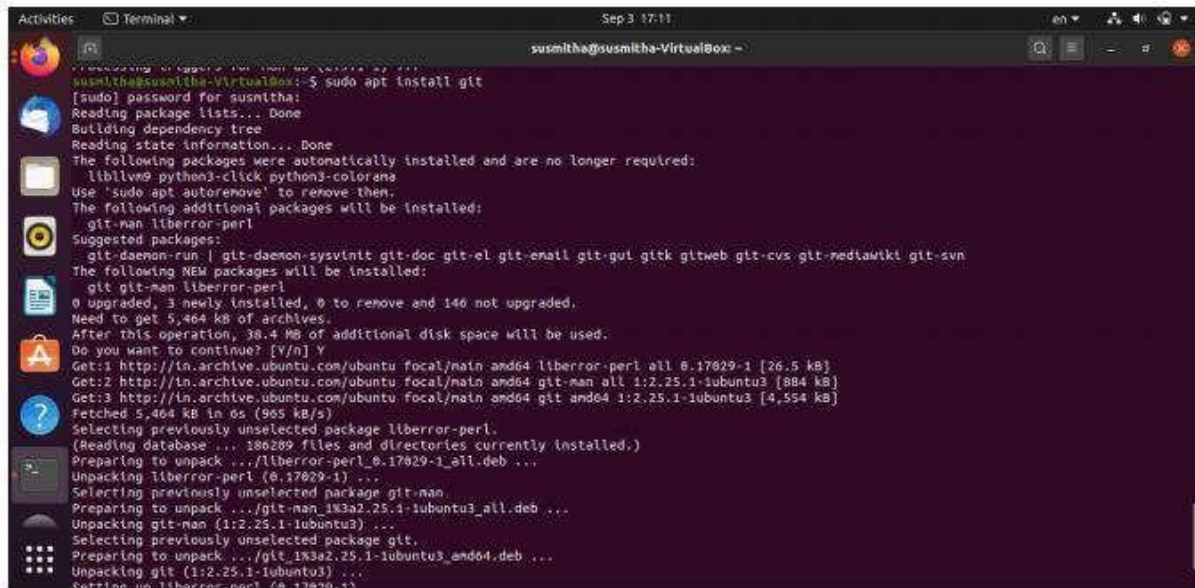
```
susmitha@susmitha-VirtualBox: ~
Unpacking libcups-server:amd64 (0.8.0-1ubuntu2) ...
Selecting previously unselected package libusbredirparser:amd64.
Preparing to unpack .../15-libusbredirparser_0.8.0-1_amd64.deb ...
Unpacking libusbredirparser:amd64 (0.8.0-1) ...
Selecting previously unselected package libvirglrenderer:amd64.
Preparing to unpack .../16-libvirglrenderer_0.8.2-1ubuntu1_amd64.deb ...
Unpacking libvirglrenderer:amd64 (0.8.2-1ubuntu1) ...
Selecting previously unselected package qemu-block-extra:amd64.
Preparing to unpack .../17-qemu-block-extra_1:4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu-block-extra:amd64 (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package libfdt1:amd64.
Preparing to unpack .../18-libfdt1_1.5.1-1_amd64.deb ...
Unpacking libfdt1:amd64 (1.5.1-1) ...
Selecting previously unselected package qemu-system-common.
Preparing to unpack .../19-qemu-system-common_1:4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu-system-common (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package qemu-system-data.
Preparing to unpack .../20-qemu-system-data_1:4.2-3ubuntu6.4_all.deb ...
Unpacking qemu-system-data (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package seabios.
Preparing to unpack .../21-seabios_1.13.0-1ubuntu1_all.deb ...
Unpacking seabios (1.13.0-1ubuntu1) ...
Selecting previously unselected package qemu-system-x86.
Preparing to unpack .../22-qemu-system-x86_1:4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu-system-x86 (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package qemu-kvm.
Preparing to unpack .../23-qemu-kvm_1:4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu-kvm (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package qemu-system-gui:amd64.
Preparing to unpack .../24-qemu-system-gui_1:4.2-3ubuntu6.4_amd64.deb ...
Unpacking qemu-system-gui:amd64 (1:4.2-3ubuntu6.4) ...
Selecting previously unselected package qemu-utils.
Preparing to unpack .../25-qemu-utils_1:4.2-3ubuntu6.4_amd64.deb ...
```

```
susmitha@susmitha-VirtualBox: ~
Setting up libfdt1:amd64 (1.5.1-1) ...
Setting up libusbredirparser:amd64 (0.8.0-1) ...
Setting up libccard0:amd64 (1:2.6.1-1) ...
Setting up ovmf (0.20191122-bd85bf54-2ubuntu3) ...
Setting up libvirglrenderer:amd64 (0.8.2-1ubuntu1) ...
Setting up qemu-system-data (1:4.2-3ubuntu6.4) ...
Setting up seabios (1.13.0-1ubuntu1) ...
Setting up libslirp0:amd64 (4.1.0-2ubuntu2.1) ...
Setting up cpu-checker (0.7-1-1) ...
Setting up qemu (1:8.0+git-20190109.133f4c4-0ubuntu3.2) ...
Setting up qemu-guest-agent (1:8.0+git-20190109.133f4c4-0ubuntu3.2) ...
Setting up qemu-guest-agent (1:8.0+git-20190109.133f4c4-0ubuntu3.2) ...
Setting up qemu-guest-agent (1:8.0+git-20190109.133f4c4-0ubuntu3.2) ...
Setting up libalot:amd64 (0.3.12-5) ...
Setting up libpxenl:amd64 (1.8-1ubuntu1) ...
Setting up librdmacm1:amd64 (28.0-1ubuntu1) ...
Setting up librados2 (15.2.3-0ubuntu0.20.04.1) ...
Setting up librd1 (15.2.3-0ubuntu0.20.04.1) ...
Setting up libiscsi7:amd64 (1.18.0-2) ...
Setting up qemu-block-extra:amd64 (1:4.2-3ubuntu6.4) ...
Setting up qemu-system-common (1:4.2-3ubuntu6.4) ...
Created symlink /etc/systemd/system/multi-user.target.wants/qemu-kvm.service → /lib/systemd/system/qemu-kvm.service.
Setting up qemu-system-x86 (1:4.2-3ubuntu6.4) ...
Setting up qemu-utils (1:4.2-3ubuntu6.4) ...
Setting up qemu-kvm (1:4.2-3ubuntu6.4) ...
Processing triggers for install-info (6.7.0-2) ...
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for gnome-menus (3.30.0-1ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
Processing triggers for man-db (2.9.1-1) ...
susmitha@susmitha-VirtualBox: ~$
```

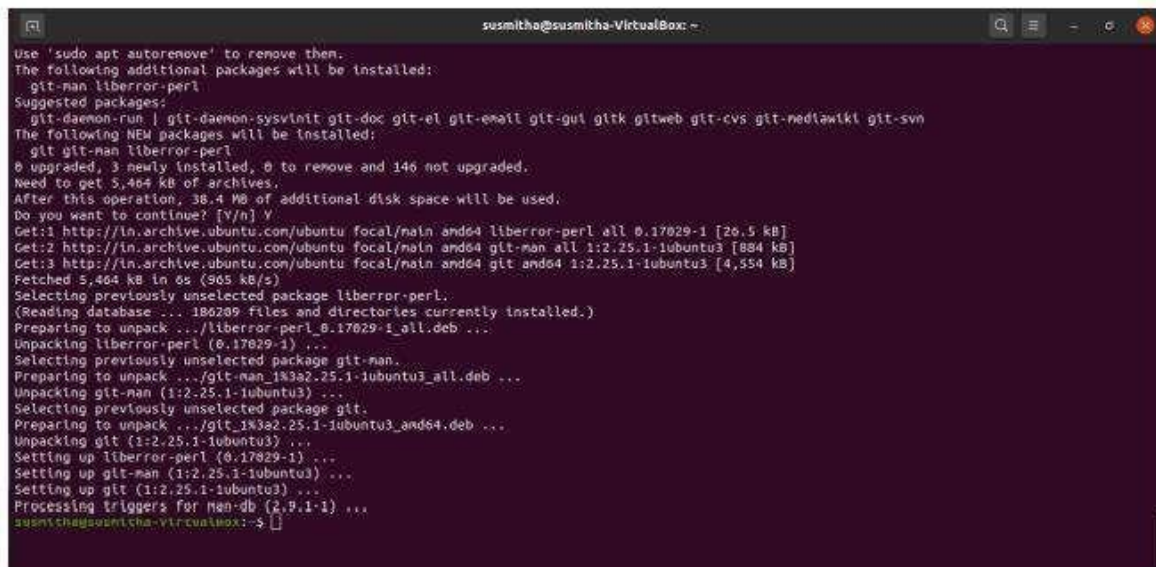
INSTALLING GIT REPOSITORY:

Git repository is installed to clone XV6 from github.

COMMAND: sudo apt install git



```
Activities Terminal Sep 3 17:11
susmitha@susmitha-VirtualBox: ~
susmitha@susmitha-VirtualBox:~$ sudo apt install git
[sudo] password for susmitha:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm9 python3-click python3-colorama
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitch gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 146 not upgraded.
Need to get 5,464 kB of archives.
After this operation, 38.4 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 liberror-perl all 0.17029-1 [26.5 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu focal/main amd64 git-man all 1:2.25.1-1ubuntu3 [884 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu focal/main amd64 git amd64 1:2.25.1-1ubuntu3 [4,554 kB]
Fetched 5,464 kB in 0s (965 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 186289 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1:2.25.1-1ubuntu3_all.deb ...
Unpacking git-man (1:2.25.1-1ubuntu3) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1:2.25.1-1ubuntu3_amd64.deb ...
Unpacking git (1:2.25.1-1ubuntu3) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.25.1-1ubuntu3) ...
Setting up git (1:2.25.1-1ubuntu3) ...
Processing triggers for man-db (2.9.1-1) ...
susmitha@susmitha-VirtualBox:~$
```



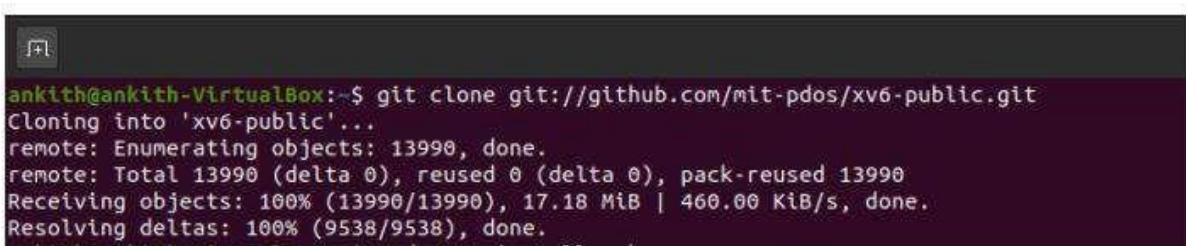
```
susmitha@susmitha-VirtualBox: ~
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitch gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 146 not upgraded.
Need to get 5,464 kB of archives.
After this operation, 38.4 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 liberror-perl all 0.17029-1 [26.5 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu focal/main amd64 git-man all 1:2.25.1-1ubuntu3 [884 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu focal/main amd64 git amd64 1:2.25.1-1ubuntu3 [4,554 kB]
Fetched 5,464 kB in 0s (965 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 186289 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1:2.25.1-1ubuntu3_all.deb ...
Unpacking git-man (1:2.25.1-1ubuntu3) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1:2.25.1-1ubuntu3_amd64.deb ...
Unpacking git (1:2.25.1-1ubuntu3) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.25.1-1ubuntu3) ...
Setting up git (1:2.25.1-1ubuntu3) ...
Processing triggers for man-db (2.9.1-1) ...
susmitha@susmitha-VirtualBox:~$
```


CLONING XV6 FROM GITHUB:

Using git and cloning XV6 OS from [git://github.com/mit-pdos/xv6-public.git](https://github.com/mit-pdos/xv6-public.git)

COMMAND:

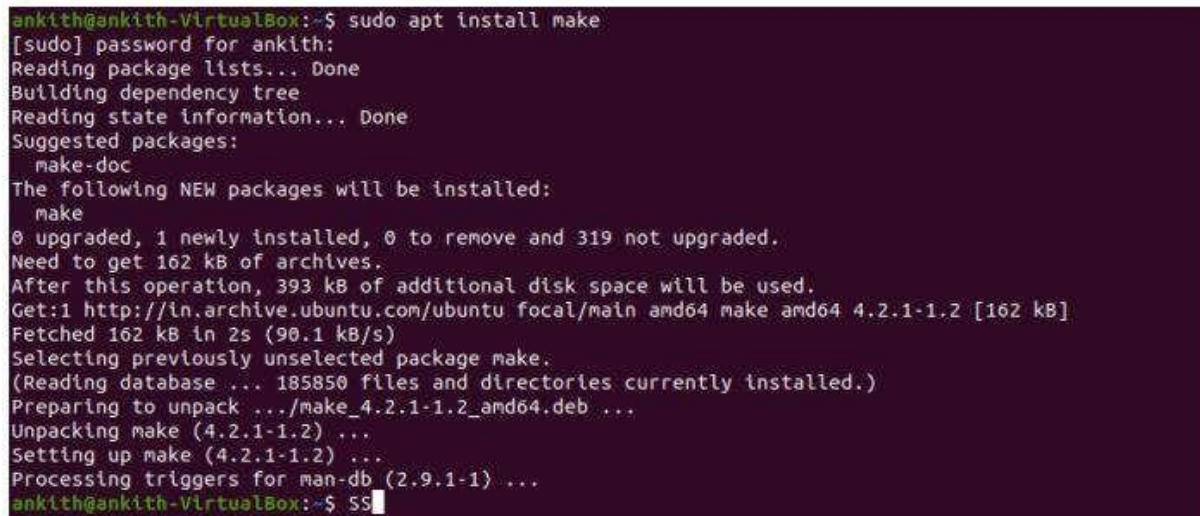
`git clone git://github.com/mit-pdos/xv6-public.git`

A terminal window with a dark background and light green text. The prompt is 'ankith@ankith-VirtualBox:~\$'. The command entered is 'git clone git://github.com/mit-pdos/xv6-public.git'. The output shows the cloning process: 'Cloning into 'xv6-public'...', 'remote: Enumerating objects: 13990, done.', 'remote: Total 13990 (delta 0), reused 0 (delta 0), pack-reused 13990', 'Receiving objects: 100% (13990/13990), 17.18 MiB | 460.00 KiB/s, done.', and 'Resolving deltas: 100% (9538/9538), done.'

```
ankith@ankith-VirtualBox:~$ git clone git://github.com/mit-pdos/xv6-public.git
Cloning into 'xv6-public'...
remote: Enumerating objects: 13990, done.
remote: Total 13990 (delta 0), reused 0 (delta 0), pack-reused 13990
Receiving objects: 100% (13990/13990), 17.18 MiB | 460.00 KiB/s, done.
Resolving deltas: 100% (9538/9538), done.
```

INSTALLING MAKE REPOSITORY:

COMMAND: Sudo apt install make

A terminal window with a dark background and light green text. The prompt is 'ankith@ankith-VirtualBox:~\$'. The command entered is 'sudo apt install make'. The output shows the installation process: '[sudo] password for ankith:', 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', 'Suggested packages: make-doc', 'The following NEW packages will be installed: make', 'make', '0 upgraded, 1 newly installed, 0 to remove and 319 not upgraded.', 'Need to get 162 kB of archives.', 'After this operation, 393 kB of additional disk space will be used.', 'Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 make amd64 4.2.1-1.2 [162 kB]', 'Fetched 162 kB in 2s (90.1 kB/s)', 'Selecting previously unselected package make.', '(Reading database ... 185850 files and directories currently installed.)', 'Preparing to unpack .../make_4.2.1-1.2_amd64.deb ...', 'Unpacking make (4.2.1-1.2) ...', 'Setting up make (4.2.1-1.2) ...', 'Processing triggers for man-db (2.9.1-1) ...', and the final prompt 'ankith@ankith-VirtualBox:~\$' with a cursor.

```
ankith@ankith-VirtualBox:~$ sudo apt install make
[sudo] password for ankith:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  make-doc
The following NEW packages will be installed:
  make
0 upgraded, 1 newly installed, 0 to remove and 319 not upgraded.
Need to get 162 kB of archives.
After this operation, 393 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 make amd64 4.2.1-1.2 [162 kB]
Fetched 162 kB in 2s (90.1 kB/s)
Selecting previously unselected package make.
(Reading database ... 185850 files and directories currently installed.)
Preparing to unpack .../make_4.2.1-1.2_amd64.deb ...
Unpacking make (4.2.1-1.2) ...
Setting up make (4.2.1-1.2) ...
Processing triggers for man-db (2.9.1-1) ...
ankith@ankith-VirtualBox:~$
```

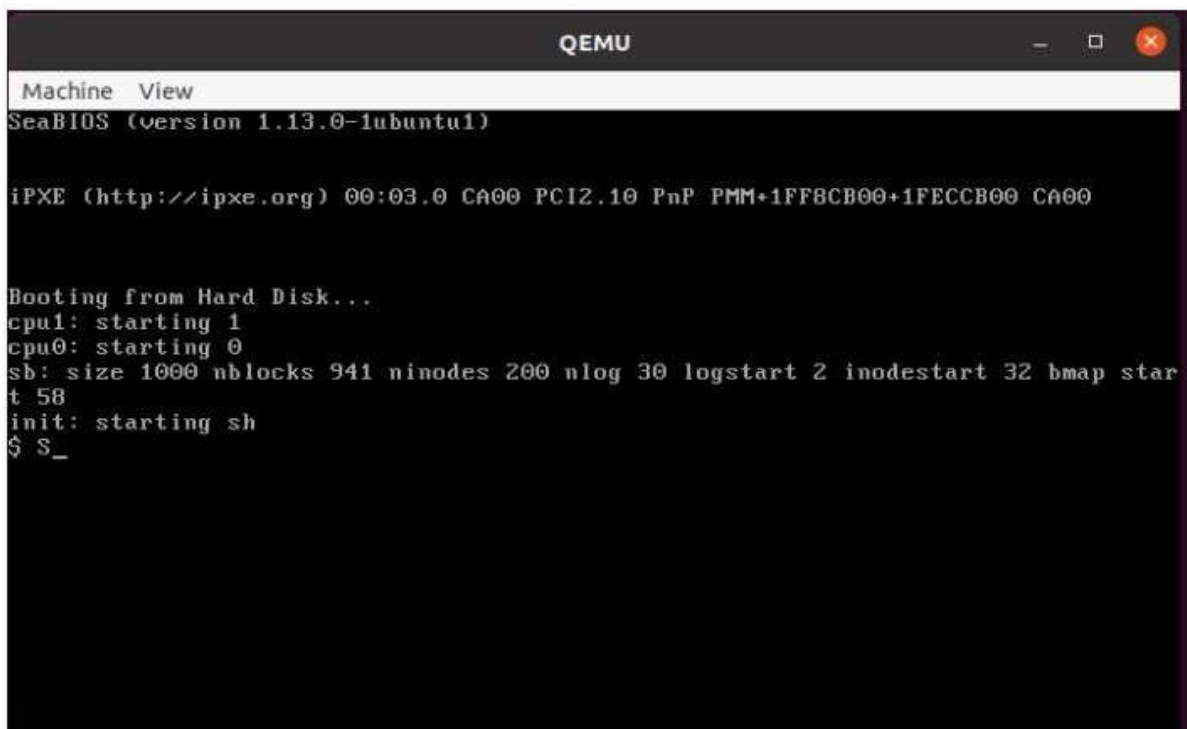
RUNNING XV6:

COMMAND:

cd xv6-public

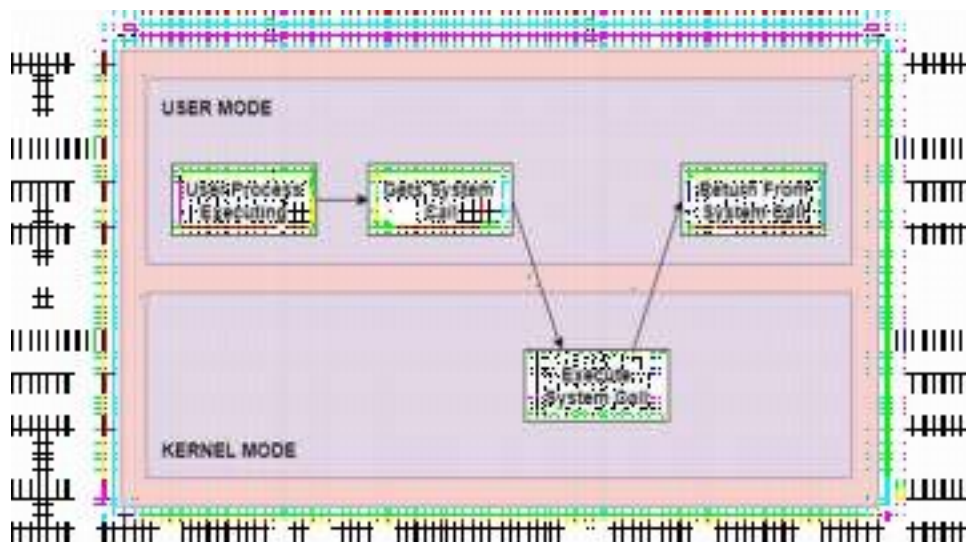
make qemu

```
ankith@ankith-VirtualBox:~$ cd xv6-public
ankith@ankith-VirtualBox:~/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -snp 2 -n 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
```



System call:

A system call is a way for programs to interact with the operating system. System calls provide an interface to the services made available by an operating system. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers. System calls are usually made when a process in user mode requires access to a resource or needs service from the kernel. Then it requests the kernel to provide the resource via a system call. A figure representing the execution of the system call is given as follows.



As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

Adding Simple User Program To Xv6:

First of all, We created a C program as shown in below image. We saved it inside the source code directory of xv6 operating system with the name myprogram.c

CODE:

```
#include "types.h"

#include "stat.h"

#include "user.h"

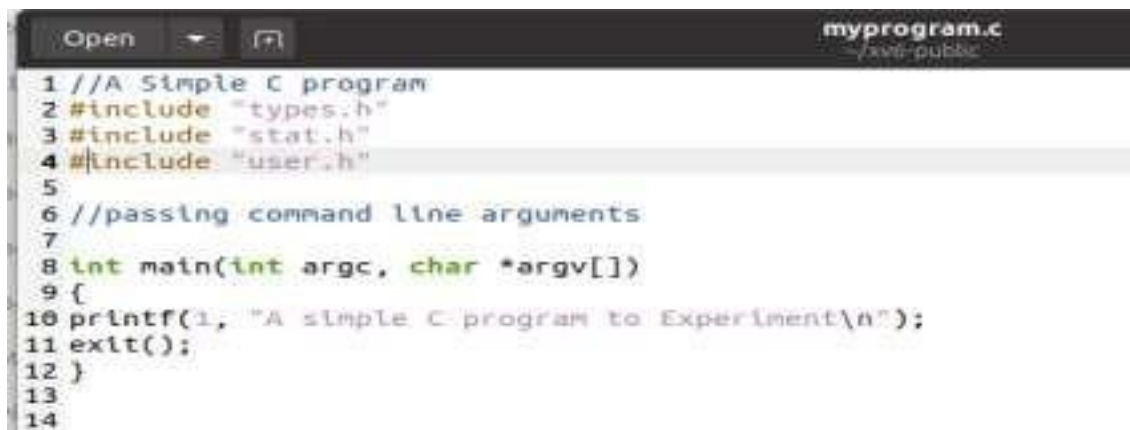
Int main(int argc, char *argv[ ])

{

    Printf("a simple c program to experiment\n");

    Exit();

}
```

A screenshot of a code editor window titled 'myprogram.c' with a subtitle '~xv6-public'. The editor shows a C program with line numbers 1 through 14 on the left margin. The code is as follows:

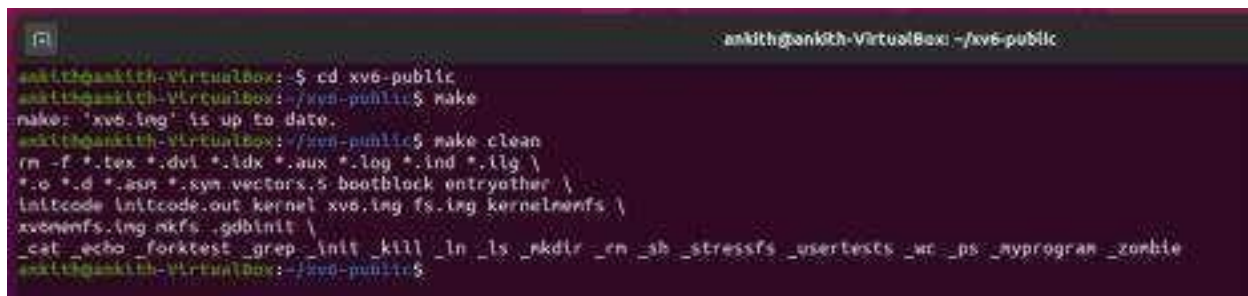
```
1 //A Simple C program
2 #include "types.h"
3 #include "stat.h"
4 #include "user.h"
5
6 //passing command line arguments
7
8 int main(int argc, char *argv[])
9 {
10 printf(1, "A simple C program to Experiment\n");
11 exit();
12 }
13
14
```


Makefile.c:

The Makefile needs to be edited to make our program available for the xv6 source code for compilation. The following sections of the Makefile needs to be edited to add our program myprogram.c



```
159 MKFS = mkfs.fs
160 gcc -Werror -Wall -o mkfs mkfs.c
161
162 # Prevent deletion of intermediate files, e.g. cat.o,
163 # that disk image changes after first build are persi
164 # details:
165 # http://www.gnu.org/software/make/manual/html_node/C
166 PRECIOUS = %.o
167
168 UPROGS = \
169     _cat \
170     _echo \
171     _forktest \
172     _grep \
173     _init \
174     _kill \
175     _ln \
176     _ls \
177     _mkdir \
178     _rm \
179     _sh \
180     _stressfs \
181     _usertests \
182     _wc \
183     _ps \
184     _myprogram \
185     _zombie \
```



```
ankith@ankith-VirtualBox: ~/xv6-public
ankith@ankith-VirtualBox: $ cd xv6-public
ankith@ankith-VirtualBox: ~/xv6-public$ make
make: 'xv6.img' is up to date.
ankith@ankith-VirtualBox: ~/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.$ bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _ps _myprogram _zombie
ankith@ankith-VirtualBox: ~/xv6-public$
```

- Now, start xv6 system on QEMU and when it booted up, run ls command to check whether our program is available for the user.
- Here myprogram is available in the list and by giving the name we can see the output of the Program In image below.

```
Machine View
-
1 1 512
1 1 512
README 2 2 2286
cat 2 3 16264
echo 2 4 15120
forktest 2 5 9432
grep 2 6 18484
init 2 7 15704
kill 2 8 15148
ln 2 9 15004
ls 2 10 17632
mkdir 2 11 15248
rm 2 12 15224
sh 2 13 27860
stressfs 2 14 16140
usertests 2 15 67244
wc 2 16 17000
ps 2 17 14844
myprogram 2 18 14900
zombie 2 19 14816
console 3 20 0
$ myprogram
A simple C program to Experiment
$
```

Adding New System Calls To xv6:

A system call is simply a kernel function that a user application can use to access or utilize system resources. Functions **fork()**, and **exec()** are well-known examples of system calls in UNIX and xv6. Here, we will use a simple example to walk you through the steps of adding a new system call to xv6. We name the system call **cps()**, which prints out the current running and sleeping processes.

An application signals the kernel it needs a service by issuing a software interrupt, a signal generated to notify the processor that it needs to stop its current task, and response to the signal request. Before switching to handling the new task, the processor has to save the current state, so that it can resume the execution in this context after the request has been handled. The following is a code that calls a system call in xv6 (found in *initcode.S*)

```

9 .globl start
10 start:
11     pushl $argv
12     pushl $init
13     pushl $0 // where caller pc would be
14     movl $SYS_exec, %eax
15     int $T_SYSCALL

```

Basically, it pushes the argument of the call to the stack, and puts the system call number, which is `$SYS_exec` in the example, into `%eax`. All the system call numbers are specified and saved in a table and the system calls of xv6 can be found in the file *syscall.h*. Next, the code `int $T_SYSCALL` generates a software interrupt, indexing the interrupt descriptor table to obtain the appropriate interrupt handler. The function **trap()** (in *trap.c*) is the specific code that finds the appropriate interrupt handler. It checks whether the trap number in the generated *trapframe* (a structure representing the processor's state at the time the trap happened) is equal to `T_SYSCALL`. If it is, it calls **syscall()**, the software interrupt handler that's available in *syscall.c*.

```

36 void
37 trap(struct trapframe *tf)
38 {
39     if(tf->trapno == T_SYSCALL){
40         if(myproc()->killed)
41             exit();
42         myproc()->tf = tf;
43         syscall();
44         if(myproc()->killed)
45             exit();
46         return;
47     }
48 }

```

The function **syscall()** is the final function that checks out `%eax` to obtain the system call's number, which is used to index the table with the system call pointers, and to execute the code corresponding to that system call:

```

136 void
137 syscall(void)
138 {
139     int num;
140     struct proc *curproc = myproc();
141
142     num = curproc->tf->eax;
143     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
144         curproc->tf->eax = syscalls[num]();
145     } else {
146         cprintf("%d %s: unknown sys call %d\n",
147             curproc->pid, curproc->name, num);
148         curproc->tf->eax = -1;
149     }
150 }

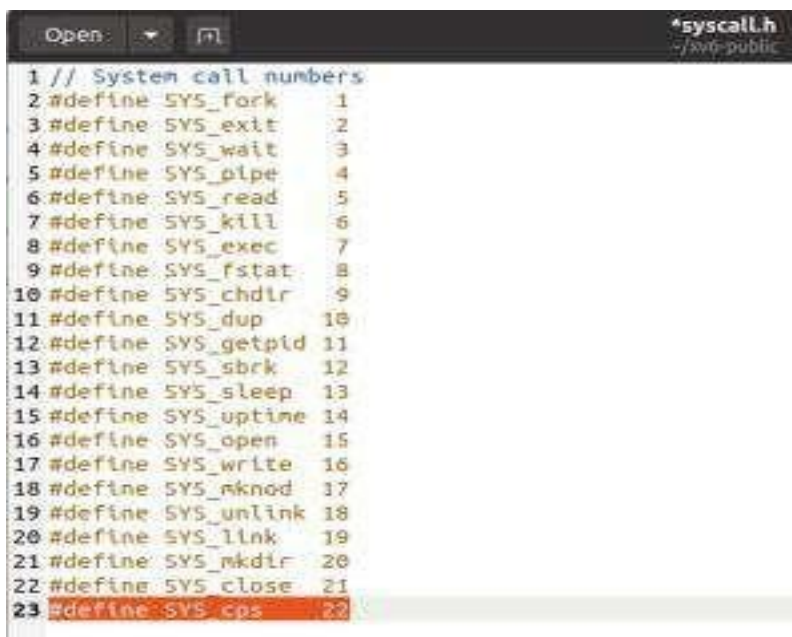
```

The following are the changes to be done to add our system call cps () to xv6:

1) Add name to syscall.h:

This defines the position of the system call vector that connects to the implementation.

CODE: #define sys_cps 22



```

Open: [v] *syscall.h
~/xv6-public

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define sys_cps 22

```


2) Add function prototype to *defs.h* :

This adds a forward declaration for the new system call. We add this function in proc.c

CODE: int cps(void);

```
105 // proc.c
106 int      cpuid(void);
107 void     exit(void);
108 int      fork(void);
109 int      growproc(int);
110 int      kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void     pinit(void);
114 void     procdump(void);
115 void     scheduler(void) __attribute__((noreturn));
116 void     sched(void);
117 void     setproc(struct proc*);
118 void     sleep(void*, struct spinlock*);
119 void     userinit(void);
120 int      wait(void);
121 void     wakeup(void*);
122 void     yield(void);
123 int      cps ( void );
```

3) Add function prototype to *user.h* :

It defines the function that can be called through the shell. We

add this function prototype in syscalls.

CODE: int cps(void);

```
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int cps ( void );
```

4) Add function call to sysproc.c :

We add the real implementation of our method here. We add a function sys_cps in the file sysproc.c which calls the function cps().

CODE:

```
Int sys_cps(void)
```

```
{
```

```
    Return cps();
```

```
}
```

```
80 // return how many clock tick interrupts have occurred
81 // since start.
82 int
83 sys_uptime(void)
84 {
85     uint xticks;
86
87     acquire(&tickslock);
88     xticks = ticks;
89     release(&tickslock);
90     return xticks;
91 }
92
93 int
94 sys_cps ( void )
95 {
96     return cps ();
97 }
```

5) Add call to *usys.S*:

It uses the macro to define connect the call of user to the system call function.

CODE: SYSCALL(cps)

```
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(cps)
```

6) Add call to *syscall.c*:

It defines the function that connects the kernel and the shell and by using the position defined in syscall.h it adds the function to the system call.

CODE: extern int sys_cps(void);

```
85 extern int sys_chdir(void);
86 extern int sys_close(void);
87 extern int sys_dup(void);
88 extern int sys_exec(void);
89 extern int sys_exit(void);
90 extern int sys_fork(void);
91 extern int sys_fstat(void);
92 extern int sys_getpid(void);
93 extern int sys_kill(void);
94 extern int sys_link(void);
95 extern int sys_mkdir(void);
96 extern int sys_mknod(void);
97 extern int sys_open(void);
98 extern int sys_pipe(void);
99 extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_cps(void);
```

CODE: [SYS_cps] sys_cps,

```
113 static int (*syscalls[])(void) = {
114 [SYS_fork]      sys_fork,
115 [SYS_exit]      sys_exit,
116 [SYS_wait]      sys_wait,
117 [SYS_pipe]      sys_pipe,
118 [SYS_read]      sys_read,
119 [SYS_kill]      sys_kill,
120 [SYS_exec]      sys_exec,
121 [SYS_fstat]     sys_fstat,
122 [SYS_chdir]     sys_chdir,
123 [SYS_dup]       sys_dup,
124 [SYS_getpid]    sys_getpid,
125 [SYS_sbrk]      sys_sbrk,
126 [SYS_sleep]     sys_sleep,
127 [SYS_uptime]    sys_uptime,
128 [SYS_open]      sys_open,
129 [SYS_write]     sys_write,
130 [SYS_mknod]     sys_mknod,
131 [SYS_unlink]    sys_unlink,
132 [SYS_link]      sys_link,
133 [SYS_mkdir]     sys_mkdir,
134 [SYS_close]     sys_close,
135 [SYS_cps]       sys_cps,
```

7) Add code to *proc.c*:

We add this code to *proc.c* as written below.

It interrupts on the processor. It acquires a lock. It runs through the process table and checks whether the process is SLEEPING or RUNNING or RUNNABLE and then prints the same pid and status of the process. It releases the lock. It returns the syscall number which is 22.

CODE:

Int cps()

{

struct proc *p; // Enable interrupts on this

processor. sti();

// Loop over process table looking for process with pid.

acquire(&ptable.lock); // acquiring lock before use of critical

section


```
cprintf("name \t pid \t state \n");
```

```
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) // checking process table
```

```
{
```

```
    if(p->state == SLEEPING)
```

```
        cprintf("%s \t %d \t SLEEPING \n", p->name, p->pid); // printing pid,pname,state
```

```
    else if(p->state == RUNNING)
```

```
        cprintf("%s \t %d \t RUNNING \n", p->name, p->pid);
```

```
}
```

```
release(&ptable.lock); // releasing acquired
```

```
lock return 22;
```

```
}
```

```
525     state = "???" ;
526     cprintf("%d %s %s", p->pid, state, p->name);
527     if(p->state == SLEEPING){
528         getcallerpcs((uint*)p->context->ebp+2, pc);
529         for(i=0; i<10 && pc[i] != 0; i++)
530             cprintf(" %p", pc[i]);
531     }
532     cprintf("\n");
533 }
534 }
535
536
537 //current process status
538 int
539 cps()
540 {
541     struct proc *p;
542
543     // Enable interrupts on this processor.
544     sti();
545
546     // Loop over process table looking for process with pid.
547     acquire(&ptable.lock);
548     cprintf("name \t pid \t state \n");
549     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
550         if ( p->state == SLEEPING )
551             cprintf("%s \t %d \t SLEEPING \n ", p->name, p->pid );
552         else if ( p->state == RUNNING )
553             cprintf("%s \t %d \t RUNNING \n ", p->name, p->pid );
554     }
555
556     release(&ptable.lock);
557
558     return 22;
559 }
```

8) Create testing file *ps.c* with code shown

below: CODE:

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

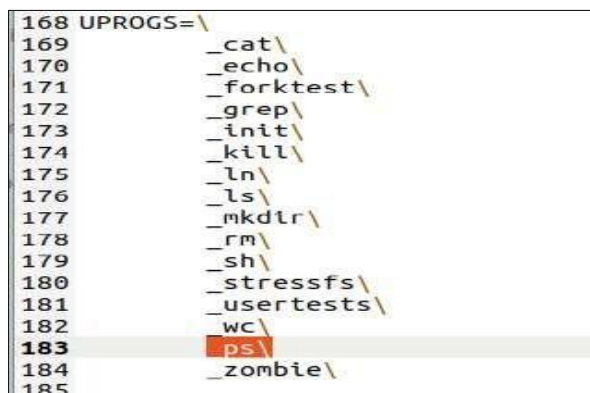
int main(int argc, char *argv[])
{
    Cps();                // calling cps
    exit();
}
```

A screenshot of a code editor window titled 'ps.c' with the path '~ /xv6-public'. The editor shows the following code:

```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     cps();
10
11     exit();
12 }
```

9) Modify Makefile :

Then we make which compiles all changes we made inside the xv6 directories and subdirectories.

A screenshot of a Makefile showing the 'UPROGS' variable. The list of programs includes: cat, echo, forktest, grep, init, kill, ln, ls, mkdir, rm, sh, stressfs, user tests, wc, ps, and zombie. The 'ps' entry is highlighted in red.

```
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _user tests\
182     _wc\
183     _ps\
184     _zombie\
185
```

Output:

Now we will compile the whole code and execute the OS after the above changes are made. Our new syscall is now visible in the list:

```
Machine View
t 58
init: starting sh
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat       2 3 16264
echo      2 4 15120
forktest  2 5 9432
grep      2 6 18484
init      2 7 15704
kill      2 8 15148
ln        2 9 15004
ls        2 10 17632
mkdir     2 11 15248
rm        2 12 15224
sh        2 13 27860
stressfs  2 14 16140
usertests 2 15 67244
wc        2 16 17000
ps        2 17 14844
zombie    2 18 14816
console   3 19 0
$ _
```

```
QEMU
Machine View
README    2 2 2286
cat       2 3 16264
echo      2 4 15120
forktest  2 5 9432
grep      2 6 18484
init      2 7 15704
kill      2 8 15148
ln        2 9 15004
ls        2 10 17632
mkdir     2 11 15248
rm        2 12 15224
sh        2 13 27860
stressfs  2 14 16140
usertests 2 15 67244
wc        2 16 17000
ps        2 17 14844
zombie    2 18 14816
console   3 19 0
$ ps
name  o pid  o state
init  o 1   o SLEEPING
sh    o 2   o SLEEPING
ps    o 4   o RUNNING
$
```

IMPLEMENTATION OF PRIORITY SCHEDULING:

An overview of Priority scheduling:

Priority scheduling is one of the most common scheduling algorithms in batch systems. Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority. Each process is assigned a priority. Process with the highest priority is to be executed first and so on.

Processes with the same priority are executed on first come first served basis.

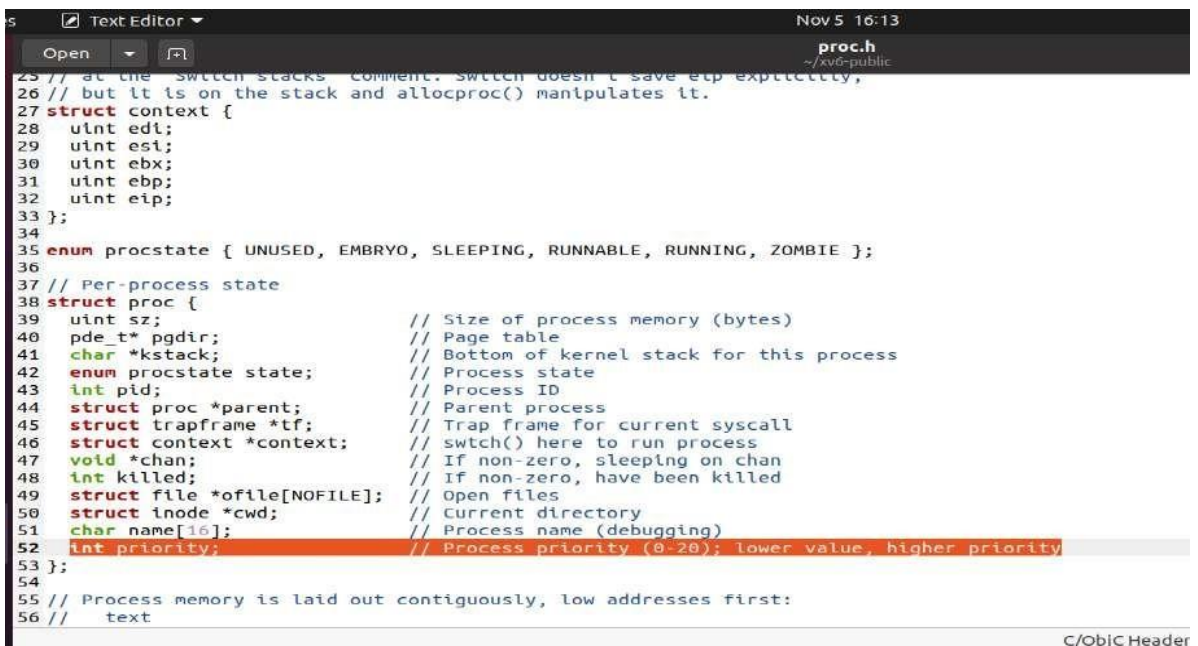
Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Default scheduling algorithm in XV6 operating system is round robin scheduling algorithm. It is not effective. It has more waiting time but priority scheduling algorithm reduces average waiting time.

1. Add priority to struct proc in proc.h:

Struct proc in proc.h is typically like PCB(process control block).It consists information about all the processes in the system. We add attribute priority in the struct proc which represents the priority of the process.

CODE: `int priority; // process priority`



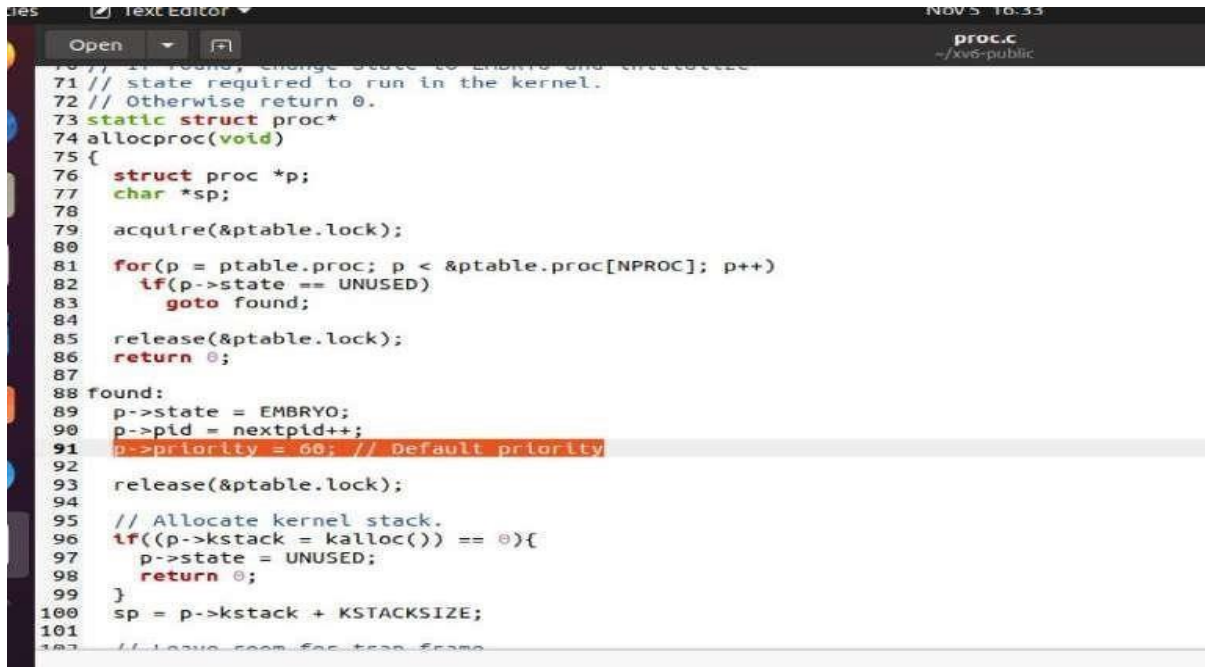
```
25 // at the switch stacks comment. Switch doesn't save eip explicitly,
26 // but it is on the stack and allocproc() manipulates it.
27 struct context {
28     uint edi;
29     uint esi;
30     uint ebx;
31     uint ebp;
32     uint eip;
33 };
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39     uint sz; // Size of process memory (bytes)
40     pde_t* pgdir; // Page table
41     char *kstack; // Bottom of kernel stack for this process
42     enum procstate state; // Process state
43     int pid; // Process ID
44     struct proc *parent; // Parent process
45     struct trapframe *tf; // Trap frame for current syscall
46     struct context *context; // swtch() here to run process
47     void *chan; // If non-zero, sleeping on chan
48     int killed; // If non-zero, have been killed
49     struct file *ofile[NOFILE]; // Open files
50     struct inode *cwd; // Current directory
51     char name[16]; // Process name (debugging)
52     int priority; // Process priority (0-20); lower value, higher priority
53 };
54
55 // Process memory is laid out contiguously, low addresses first:
56 // text
```


2. Assign a default priority in proc.h:

allocproc is a function that allocates resources to new process. It scans the entire process table and if it finds an unused entry then it will assign pid and resources to process. Here we set default priority for all the process to 60 in allocproc function.

CODE:

p->priority=60; //default priority set to 60



```
70 // 21 found, change state to EMBRYO and allocate
71 // state required to run in the kernel.
72 // Otherwise return 0.
73 static struct proc*
74 allocproc(void)
75 {
76     struct proc *p;
77     char *sp;
78     acquire(&ptable.lock);
79
80     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
81         if(p->state == UNUSED)
82             goto found;
83     release(&ptable.lock);
84     return 0;
85 found:
86     p->state = EMBRYO;
87     p->pid = nextpid++;
88     p->priority = 60; // Default priority
89     release(&ptable.lock);
90
91     // Allocate kernel stack.
92     if((p->kstack = kalloc()) == 0){
93         p->state = UNUSED;
94         return 0;
95     }
96     sp = p->kstack + KSTACKSIZE;
97
98     // Leave room for trap frame
99     // ...
```

3. Adding code to print priority of process in cps in proc.c:

We have added code in cps function in proc.c to print priority of process along with process id, current process state and process name.

```

561 }
562 }
563
564 int
565 cps()
566 {
567     struct proc *p;
568
569     // Enable interrupts on this processor.
570     sti();
571
572     // Loop over process table looking for process with pid.
573     acquire(&ptable.lock);
574     cprintf("name \t pid \t state \t \t priority \n");
575     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
576     {
577         if(p->state == SLEEPING)
578             cprintf("%s \t %d \t SLEEPING \t %d\n", p->name, p->pid, p->priority);
579         else if(p->state == RUNNING)
580             cprintf("%s \t %d \t RUNNING \t %d\n", p->name, p->pid, p->priority);
581         else if(p->state == RUNNABLE)
582             cprintf("%s \t %d \t RUNNABLE \t %d\n", p->name, p->pid, p->priority);
583     }
584     release(&ptable.lock);
585
586     return 22;
587 }
588 }
589
590 // Change priority
591 int
592 ChangePriority(int pid, int priority)

```

4. Creation of dummy program foo.c:

We create a dummy program named as foo.c and this dummy program will create a child and do sum dummy computations or calculations to waste CPU time. The main in this program take 2 arguments. The first argument is number of child processes that has to be created. We have used fork system call in this program to create child process.

CODE:

```

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

int main(int argc, char *argv[])
{
    int k, n, id;
    double x=0, z;

    if(argc < 2)
        n = 1;    // default
    value else
        n = atoi(argv[1]);    // from user
    input if(n<0 || n>100)
        n = 2;

```

```

x = 0;
id = 0;
for(k=0; k<n; k++)
{
    id = fork();
    if(id < 0)
        printf(1, "%d failed in fork!\n", getpid());
    else if(id > 0)
    { // Parent
        printf(1, "Parent %d creating child %d\n", getpid(), id);
        wait();
    }
    else
    { // Child
        printf(1, "Child %d created\n", getpid());
        for(z=0; z<8000000.0; z+=0.001)
            x = x + 3.14*69.69; // Useless calculations to consume CPU
        time break;
    }
}
}

```

The screenshot shows a code editor window with a dark title bar and a light orange background. The code is written in C and implements the logic from the pseudocode above. It includes variable declarations for `k`, `n`, `id`, `x`, and `z`. It handles command-line arguments to set `n`, with a default of 1. The main loop uses `fork()` to create child processes. The parent process prints a message and calls `wait()`. The child process prints a message and enters a loop that performs useless calculations to consume CPU time, with a `break;` statement at the end of the loop. The editor's status bar at the bottom right shows the filename `foo.c` and the path `~/xv6-public`.

```

5
6 int main(int argc, char *argv[])
7 {
8     int k, n, id;
9     double x=0, z;
10
11     if(argc < 2)
12         n = 1; // default value
13     else
14         n = atoi(argv[1]); // from user input
15     if(n<0 || n>100)
16         n = 2;
17
18     x = 0;
19     id = 0;
20     for(k=0; k<n; k++)
21     {
22         id = fork();
23         if(id < 0)
24             printf(1, "%d failed in fork!\n", getpid());
25         else if(id > 0)
26         { // Parent
27             printf(1, "Parent %d creating child %d\n", getpid(), id);
28             wait();
29         }
30         else
31         { // child
32             printf(1, "Child %d created\n", getpid());
33             for(z=0; z<8000000.0; z+=0.001)
34                 x = x + 3.14*69.69; // Useless calculations to consume CPU time
35             break;
36         }
37     }
38 }

```

5. Addition of new function chpr(change priority) to proc.c:

We add a new function named as chpr in proc.c file. This function takes two arguments. First argument is process id, and second argument is the priority, This function changes the priority of given process id.

CODE:

// Change priority

int

ChangePriority(int pid, int priority)

{

struct proc *p;

acquire(&ptable.lock); // acquire lock to access critical

section for(p=ptable.proc; p<&ptable.proc[NPROC]; p++)

{

if(p->pid == pid) // checking process table

{

p->priority = priority; //changing priority of

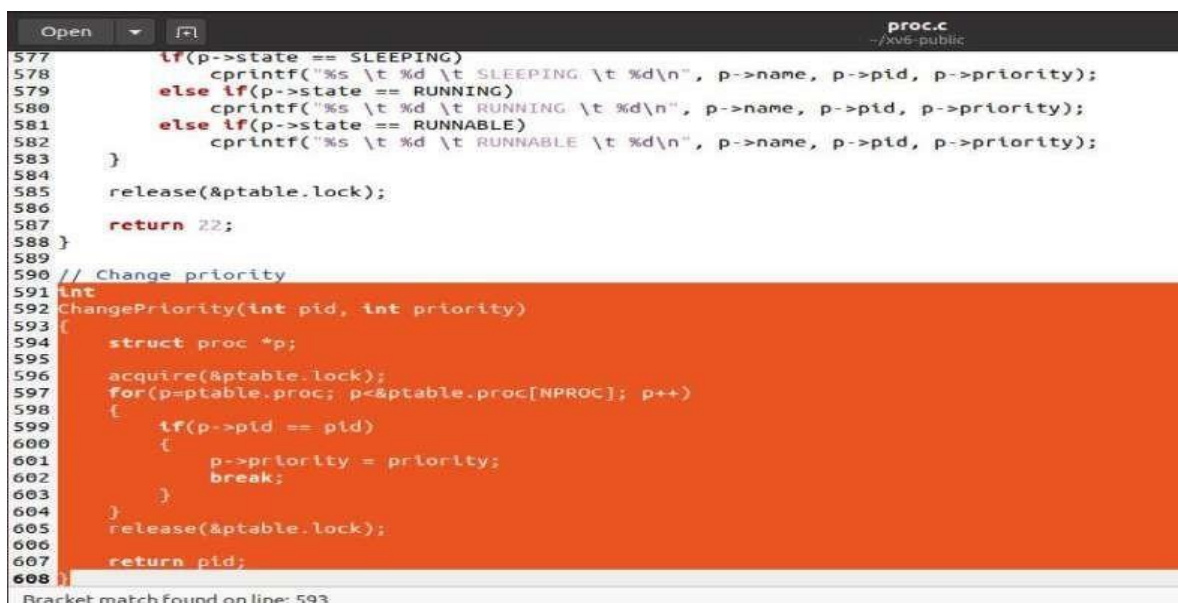
process break;

}

}

release(&ptable.lock);

return pid; }



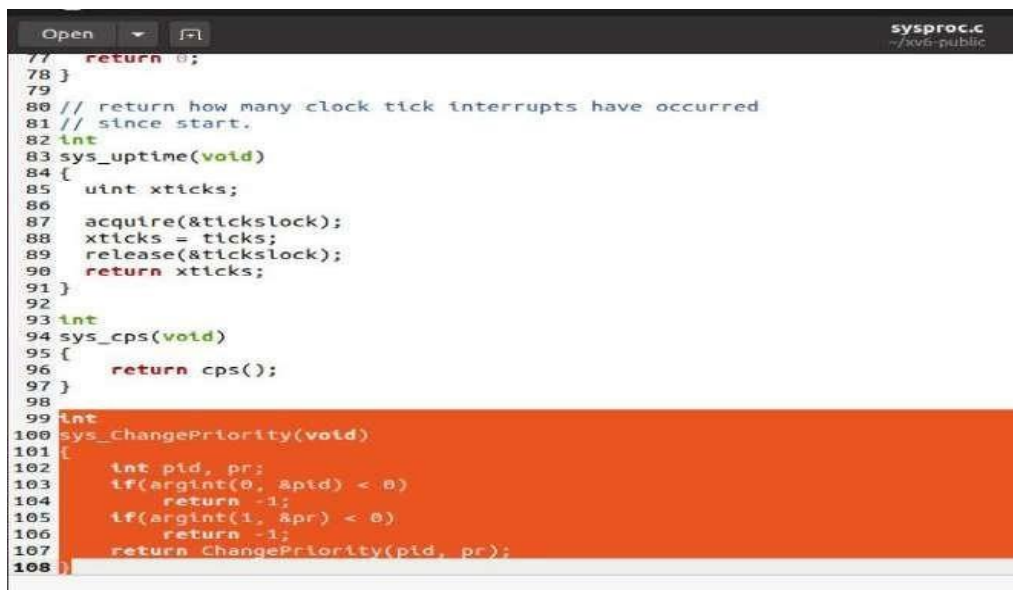
```
577 if(p->state == SLEEPING)
578     cprintf("%s \t %d \t SLEEPING \t %d\n", p->name, p->pid, p->priority);
579 else if(p->state == RUNNING)
580     cprintf("%s \t %d \t RUNNING \t %d\n", p->name, p->pid, p->priority);
581 else if(p->state == RUNNABLE)
582     cprintf("%s \t %d \t RUNNABLE \t %d\n", p->name, p->pid, p->priority);
583 }
584 release(&ptable.lock);
585 return 22;
586 }
587 }
588 }
589 // Change priority
590 int
591 ChangePriority(int pid, int priority)
592 {
593     struct proc *p;
594     acquire(&ptable.lock);
595     for(p=ptable.proc; p<&ptable.proc[NPROC]; p++)
596     {
597         if(p->pid == pid)
598         {
599             p->priority = priority;
600             break;
601         }
602     }
603     release(&ptable.lock);
604     return pid;
605 }
```

Bracket match found on line: 593

6. Adding system call(chpr):

As we have added system call cps, we have to follow same steps to add system call chpr in xv6 operating system.

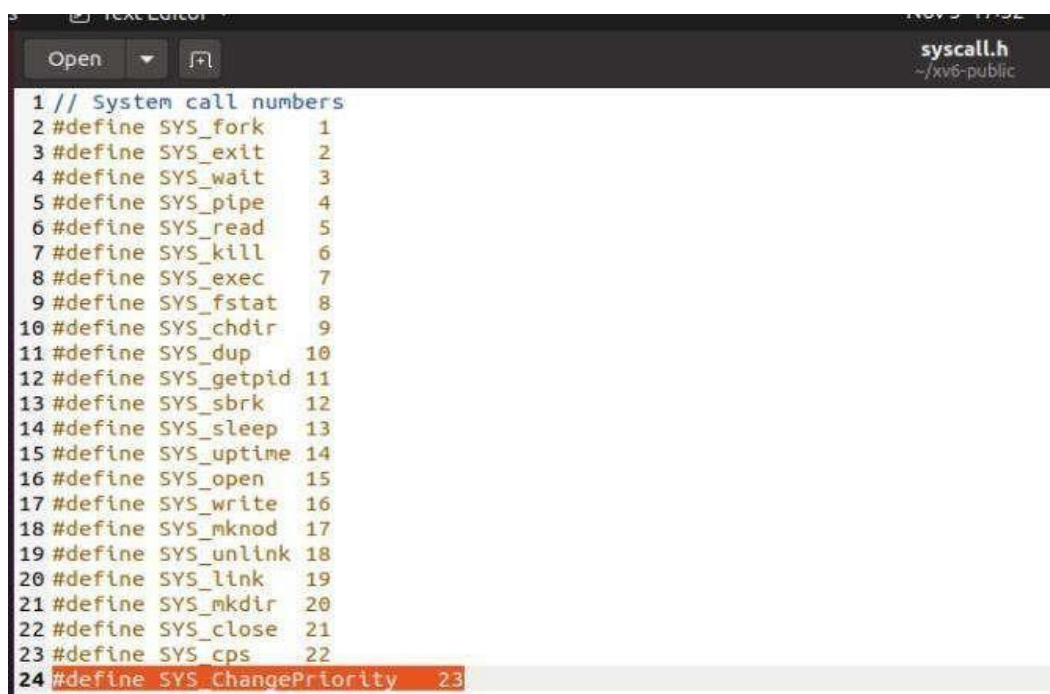
- Add sys_chpr to sysproc.c:



```
sysproc.c
~/xv6-public

77 return 0;
78 }
79
80 // return how many clock tick interrupts have occurred
81 // since start.
82 int
83 sys_uptime(void)
84 {
85     uint xticks;
86     acquire(&tickslock);
87     xticks = ticks;
88     release(&tickslock);
89     return xticks;
90 }
91
92
93 int
94 sys_cps(void)
95 {
96     return cps();
97 }
98
99 int
100 sys_ChangePriority(void)
101 {
102     int pid, pr;
103     if(argint(0, &pid) < 0)
104         return -1;
105     if(argint(1, &pr) < 0)
106         return -1;
107     return ChangePriority(pid, pr);
108 }
```

- Adding to syscall.h:



```
Text Editor
syscall.h
~/xv6-public

1 // System call numbers
2 #define SYS_fork    1
3 #define SYS_exit    2
4 #define SYS_wait    3
5 #define SYS_pipe    4
6 #define SYS_read    5
7 #define SYS_kill    6
8 #define SYS_exec    7
9 #define SYS_fstat    8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid  11
13 #define SYS_sbrk   12
14 #define SYS_sleep  13
15 #define SYS_uptime 14
16 #define SYS_open   15
17 #define SYS_write  16
18 #define SYS_mknod  17
19 #define SYS_unlink 18
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
23 #define SYS_cps    22
24 #define SYS_ChangePriority 23
```


- Modify defs.h:

```

Open  defs.h
~/xv6-public

95 void      picenable(int);
96 void      picinit(void);
97
98 // pipe.c
99 int        pipealloc(struct file**, struct file**);
100 void       pipeclose(struct pipe*, int);
101 int        piperead(struct pipe*, char*, int);
102 int        pipewrite(struct pipe*, char*, int);
103
104 //PAGEBREAK: 16
105 // proc.c
106 int        cpuid(void);
107 void       exit(void);
108 int        fork(void);
109 int        growproc(int);
110 int        kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void       pinit(void);
114 void       procdump(void);
115 void       scheduler(void) __attribute__((noreturn));
116 void       sched(void);
117 void       setproc(struct proc*);
118 void       sleep(void*, struct spinlock*);
119 void       userinit(void);
120 int        wait(void);
121 void       wakeup(void*);
122 void       yield(void);
123 int        cps(void);
124 int        ChangePriority(int pid, int priority);
125
126 // swtch.S

```

- Modify user.h:

```

Open  user.h
~/xv6-public

1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(char*, short, short);
16 int unlink(char*);
17 int fstat(int fd, struct stat*);
18 int link(char*, char*);
19 int mkdir(char*);
20 int chdir(char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int cps(void);
27 int ChangePriority(int pid, int priority);
28
29 // ulib.c
30 int stat(const char*, struct stat*);
31 char* strcpy(char*, const char*);
32 void *memmove(void*, const void*, int);

```

- **Modify usys.s:**

```

Open  ▾  [?]  usys.S
~/xv6-public
4 #define SYSCALL(name) \
5   .globl name; \
6   name: \
7     movl $SYS_ ## name, %eax; \
8     int $T_SYSCALL; \
9     ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(cps)
33 SYSCALL(ChangePriority)
34
35

```

Modify syscall.c:

```

Open  ▾  [?]  syscall.c
~/xv6-public
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_cps(void);
107 extern int sys_ChangePriority(void);
108
109 static int (*syscalls[])(void) = {
110 [SYS_fork]      sys_fork,
111 [SYS_exit]      sys_exit,
112 [SYS_wait]      sys_wait,
113 [SYS_pipe]      sys_pipe,
114 [SYS_read]      sys_read,
115 [SYS_kill]      sys_kill,
116 [SYS_exec]      sys_exec,
117 [SYS_fstat]     sys_fstat,
118 [SYS_chdir]     sys_chdir,
119 [SYS_dup]       sys_dup,
120 [SYS_getpid]    sys_getpid,
121 [SYS_sbrk]      sys_sbrk,
122 [SYS_sleep]     sys_sleep,
123 [SYS_uptime]    sys_uptime,
124 [SYS_open]      sys_open,
125 [SYS_write]     sys_write,
126 [SYS_mknod]     sys_mknod,
127 [SYS_unlink]    sys_unlink,
128 [SYS_link]      sys_link,
129 [SYS_mkdir]     sys_mkdir,
130 [SYS_close]     sys_close,
131 [SYS_cps]       sys_cps,
132 [SYS_ChangePriority] sys_ChangePriority,
133 };
134

```

```
Open [f+] syscall.c ~/xv6-public
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_cps(void);
107 extern int sys_ChangePriority(void);
108
109 static int (*syscalls[])(void) = {
110 [SYS_fork]      sys_fork,
111 [SYS_exit]      sys_exit,
112 [SYS_wait]      sys_wait,
113 [SYS_pipe]      sys_pipe,
114 [SYS_read]      sys_read,
115 [SYS_kill]      sys_kill,
116 [SYS_exec]      sys_exec,
117 [SYS_fstat]     sys_fstat,
118 [SYS_chdir]     sys_chdir,
119 [SYS_dup]       sys_dup,
120 [SYS_getpid]    sys_getpid,
121 [SYS_sbrk]      sys_sbrk,
122 [SYS_sleep]     sys_sleep,
123 [SYS_uptime]    sys_uptime,
124 [SYS_open]      sys_open,
125 [SYS_write]     sys_write,
126 [SYS_mknod]     sys_mknod,
127 [SYS_unlink]    sys_unlink,
128 [SYS_link]      sys_link,
129 [SYS_mkdir]     sys_mkdir,
130 [SYS_close]     sys_close,
131 [SYS_cps]       sys_cps,
132 [SYS_ChangePriority] sys_ChangePriority,
133 };
134
135 void
```

6. Adding nice.c program:

This user program will call system call chpr(change priority) to change priority of the process.

CODE:

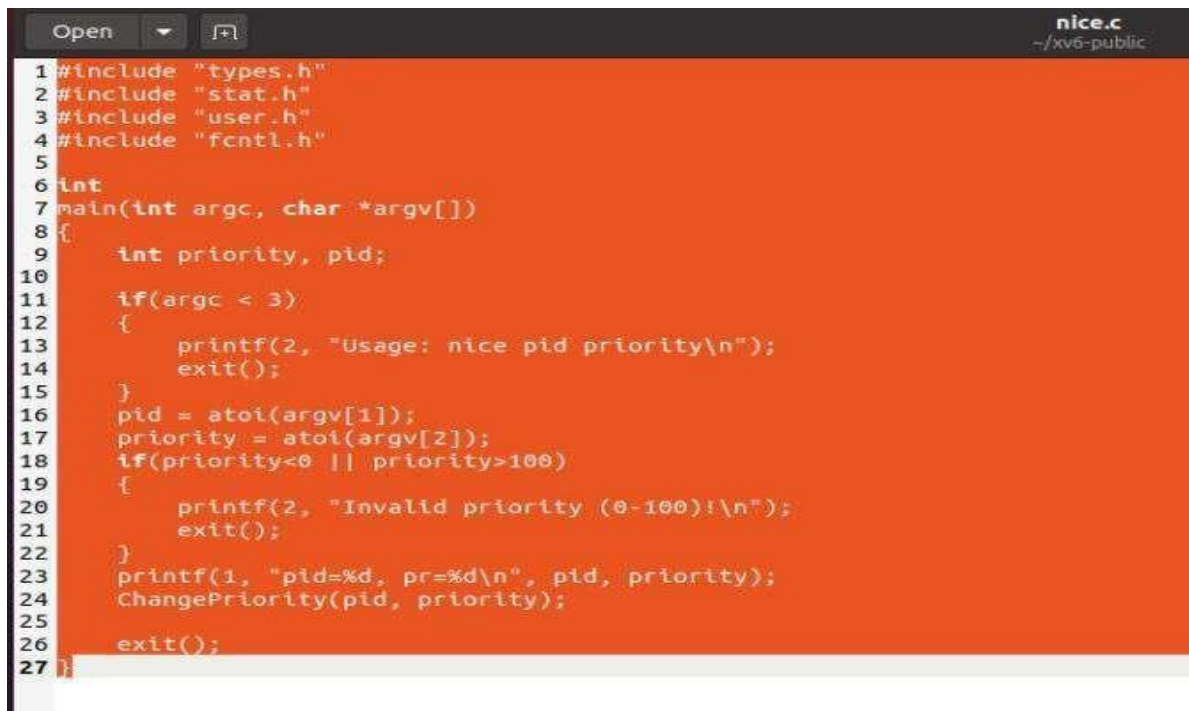
```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char *argv[])
{
    int priority, pid;

    if(argc < 3)
    {
        printf(2, "Usage: nice pid priority\n");
        exit();
    }
    pid = atoi(argv[1]);
```

```
priority = atoi(argv[2]);
if(priority<0 || priority>100)
{
    printf(2, "Invalid priority (0-100)!\n");
    exit();
}
printf(1, "pid=%d, pr=%d\n", pid, priority);
ChangePriority(pid, priority);

exit();
}
```



The screenshot shows a code editor window with a dark theme. The title bar on the right says "nice.c" and the path is "~/xv6-public". The code is written in C and is as follows:

```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     int priority, pid;
10
11     if(argc < 3)
12     {
13         printf(2, "Usage: nice pid priority\n");
14         exit();
15     }
16     pid = atoi(argv[1]);
17     priority = atoi(argv[2]);
18     if(priority<0 || priority>100)
19     {
20         printf(2, "Invalid priority (0-100)!\n");
21         exit();
22     }
23     printf(1, "pid=%d, pr=%d\n", pid, priority);
24     ChangePriority(pid, priority);
25
26     exit();
27 }
```

OUTPUT:

```
es  terminal Nov 5 18:07
susmitha@susmitha-VirtualBox: ~/xv6-public

$ ps
name      pid      state      priority
init       1      SLEEPING      60
sh         2      SLEEPING      60
ps         4      RUNNING      60
$ foo 2 &
$ Parent 7 creating child 8
Child 8 created
$ ps
name      pid      state      priority
init       1      SLEEPING      60
sh         2      SLEEPING      60
foo        8      RUNNING      60
ps         9      RUNNING      60
foo        7      SLEEPING      60
$ ps
name      pid      state      priority
init       1      SLEEPING      60
sh         2      SLEEPING      60
foo        8      RUNNING      60
ps        10      RUNNING      60
foo        7      SLEEPING      60
$ nice 8 10
pid=8, pr=10
$ ps
name      pid      state      priority
init       1      SLEEPING      60
sh         2      SLEEPING      60
foo        8      RUNNING      10
ps        12      RUNNING      60
foo        7      SLEEPING      60
```

```
QEMU

$ ps
name  pid  state  priority
init  1    SLEEPING  60
sh    2    SLEEPING  60
foo   8    RUNNING  60
ps    9    RUNNING  60
foo   7    SLEEPING  60
$ ps
name  pid  state  priority
init  1    SLEEPING  60
sh    2    SLEEPING  60
foo   8    RUNNING  60
ps    10   RUNNING  60
foo   7    SLEEPING  60
$ nice 8 10
pid=8, pr=10
$ ps
name  pid  state  priority
init  1    SLEEPING  60
sh    2    SLEEPING  60
foo   8    RUNNING  10
ps    12   RUNNING  60
foo   7    SLEEPING  60
$
```


Copy on write fork:

The fork() system call in xv6 copies all of the parent process's user-space memory into the child. If the parent is large, copying can take a long time. The work is often largely wasted; for example, a fork() followed by exec() in the child will cause the child to discard the copied memory, probably without ever using most of it.

The goal of copy-on-write (COW) fork() is to defer allocating and copying physical memory pages for the child until the copies are actually needed, if ever. Copy-On-Write avoids this expense by being lazy. Rather than copy all the memory at once it pretends it was copied and only actually copies when the parent and child need to hold different values at the same address.

COW fork() creates just a pagetable for the child, with PTEs for user memory pointing to the parent's physical pages. COW fork() marks all the user PTEs in both parent and child as not writable. When either process tries to write one of these COW pages, the CPU will force a page fault. The kernel page-fault handler detects this case, allocates a page of physical memory for the faulting process, copies the original page into the new page, and modifies the relevant PTE in the faulting process to refer to the new page, this time with the PTE marked writeable. When the page fault handler returns, the user process will be able to write its copy of the page.

Implementation of Copy-on-write System Call:

Adding getNumFreePages system call:

Add the variable numFreePages to implement the given system call in kalloc.c :

In kalloc.c add the entry in struct kmem:

```
int numFreePages;
```

The system call getNumFreePages() should return the total number of free pages in the system. This system call will help you see when pages are consumed, and can help you debug your CoW implementation. You must add code to maintain and track freepages in kalloc.c, and access this information when this system call is invoked.

```

1 // Physical memory allocator
2 // memory for user processes
3 // and pipe buffers. Allocat
4
5 #include "types.h"
6 #include "defs.h"
7 #include "param.h"
8 #include "memlayout.h"
9 #include "mmu.h"
10 #include "spinlock.h"
11
12 void freerange(void *vstart,
13 extern char end[]; // first i
14 file
15 struct run {
16     struct run *next;
17 };
18
19 struct {
20     struct spinlock lock;
21     int use_lock;
22     struct run *freelist;
23     int numFreePages;
24 } kmem;
25

```

Adding given system calls: add following code in defs.h int

getNumFreePages(void);

The file defs.h acts as the header file for several parts of the kernel code.

```

defs.h
~/Downloads/xyb

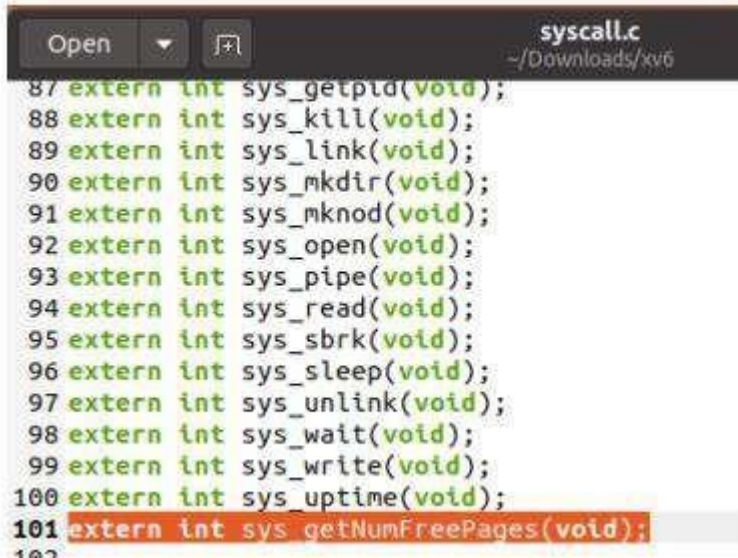
52 int      write(struct inode*, char*, u
53
54 // ide.c
55 void      ideinit(void);
56 void      ideintr(void);
57 void      iderw(struct buf*);
58
59 // ioapic.c
60 void      ioapicenable(int irq, int cpu)
61 extern uchar ioapicid;
62 void      ioapicinit(void);
63
64 // kalloc.c
65 char*      kalloc(void);
66 void      kfree(char*);
67 void      kinit1(void*, void*);
68 void      kinit2(void*, void*);
69 int      getNumFreePages(void);
70

```

Add following code in sysproc.c:

```
57
58 int
59 sys_sleep(void)
60 {
61     int n;
62     uint ticks0;
63
64     if(argint(0, &n) < 0)
65         return -1;
66     acquire(&tickslock);
67     ticks0 = ticks;
68     while(ticks - ticks0 < n){
69         if(proc->killed){
70             release(&tickslock);
71             return -1;
72         }
73         sleep(&ticks, &tickslock);
74     }
75     release(&tickslock);
76     return 0;
77 }
78
79 // return how many clock tick interrupts have
80 // since start.
81 int
82 sys_uptime(void)
83 {
84     uint xticks;
85
86     acquire(&tickslock);
87     xticks = ticks;
88     release(&tickslock);
89     return xticks;
90 }
91
92 int
93 sys_getNumFreePages(void)
94 {
95     return getNumFreePages();
96 }
```

Add the following declaration along with other system calls in syscall.c:



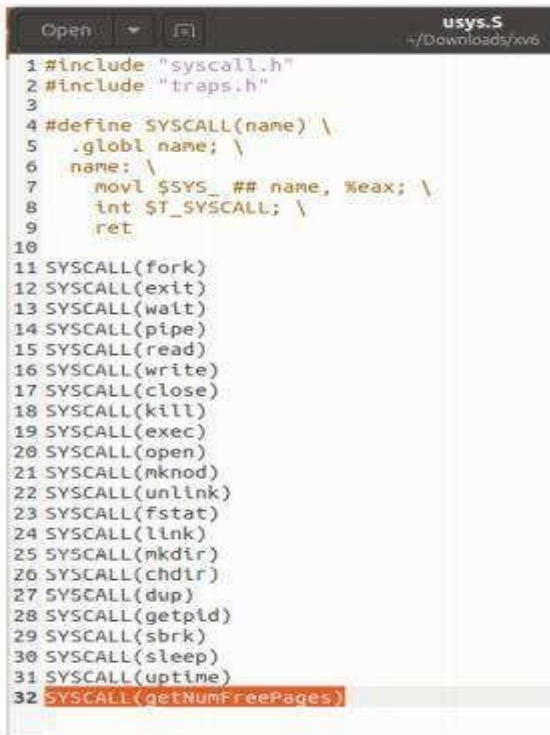
```
107 extern int sys_getpid(void);
108 extern int sys_kill(void);
109 extern int sys_link(void);
110 extern int sys_mkdir(void);
111 extern int sys_mknod(void);
112 extern int sys_open(void);
113 extern int sys_pipe(void);
114 extern int sys_read(void);
115 extern int sys_sbrk(void);
116 extern int sys_sleep(void);
117 extern int sys_unlink(void);
118 extern int sys_wait(void);
119 extern int sys_write(void);
120 extern int sys_uptime(void);
121 extern int sys_getNumFreePages(void);
```

Add following fields in the same file like other system calls in syscall.c:



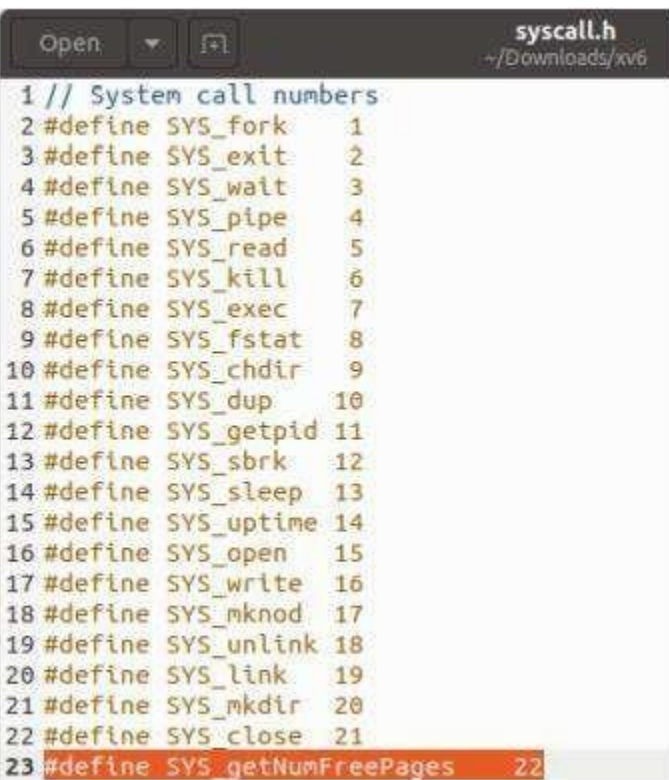
```
122
123 static int (*syscalls[])(void) = {
124 [SYS_fork]      sys_fork,
125 [SYS_exit]      sys_exit,
126 [SYS_wait]      sys_wait,
127 [SYS_pipe]      sys_pipe,
128 [SYS_read]      sys_read,
129 [SYS_kill]      sys_kill,
130 [SYS_exec]      sys_exec,
131 [SYS_fstat]     sys_fstat,
132 [SYS_chdir]     sys_chdir,
133 [SYS_dup]       sys_dup,
134 [SYS_getpid]    sys_getpid,
135 [SYS_sbrk]      sys_sbrk,
136 [SYS_sleep]     sys_sleep,
137 [SYS_uptime]    sys_uptime,
138 [SYS_open]      sys_open,
139 [SYS_write]     sys_write,
140 [SYS_mknod]     sys_mknod,
141 [SYS_unlink]    sys_unlink,
142 [SYS_link]      sys_link,
143 [SYS_mkdir]     sys_mkdir,
144 [SYS_close]     sys_close,
145 [SYS_getNumFreePages] sys_getNumFreePages,
146 };
147
```

Add the following lines in usys.S:



```
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(getNumFreePages)
```

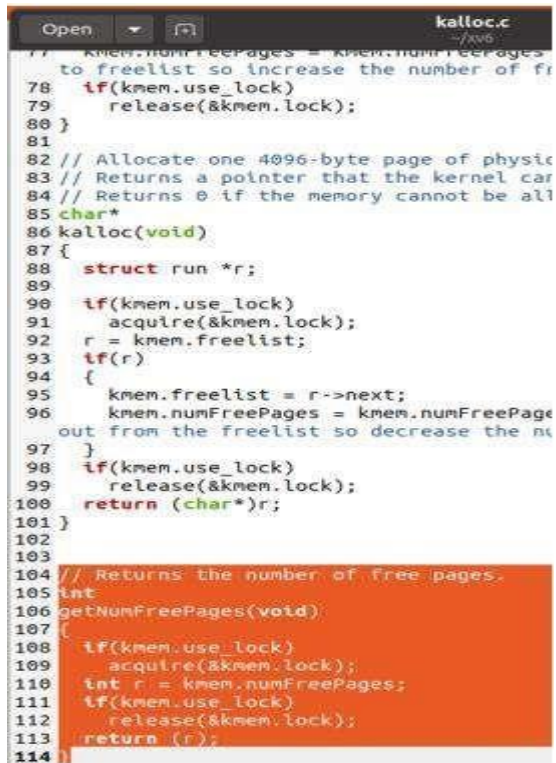
Add following lines in syscall.h:



```
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_getNumFreePages 22
```

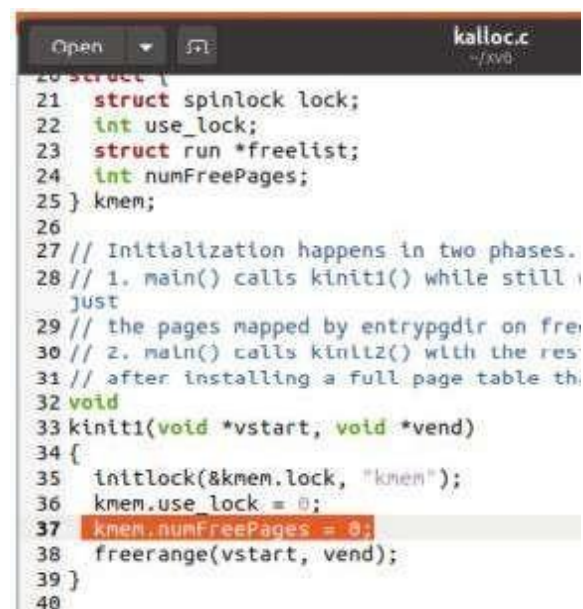

Add the function body of `getNumFreePages` in `kalloc.c`:

The files `vm.c` and `kalloc.c` contain most of the logic for memory management in the xv6 kernel



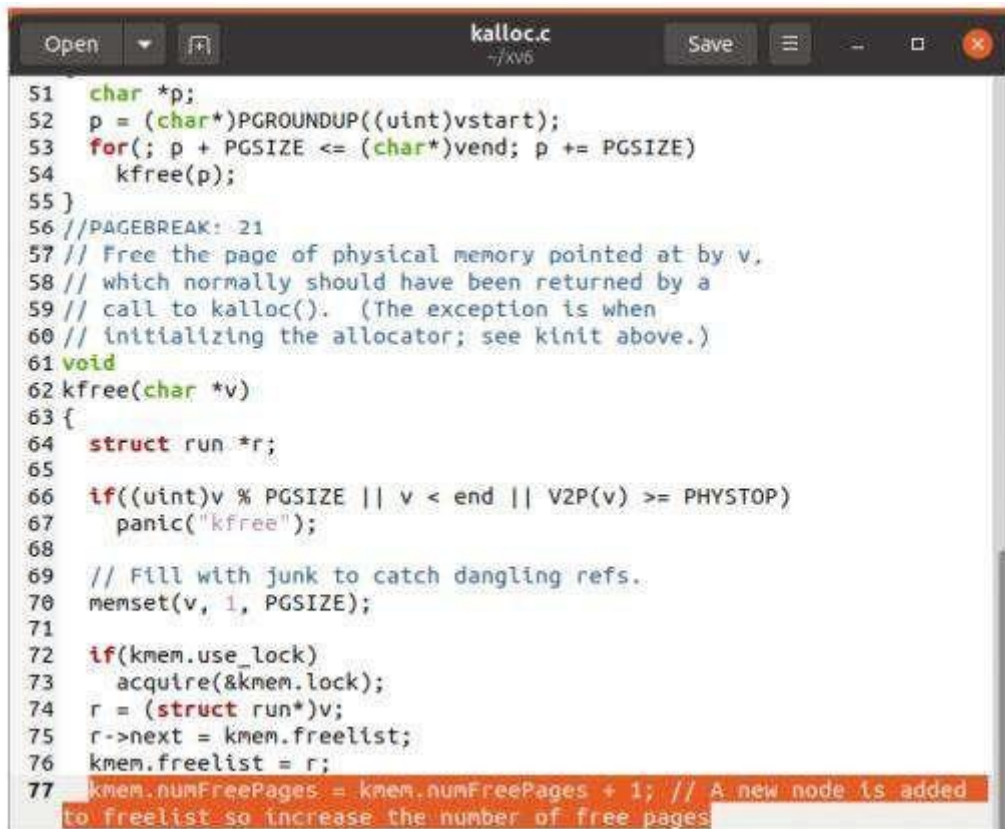
```
Open  kalloc.c
// kmem.numFreePages = kmem.numFreePages
// to freelist so increase the number of fr
78 if(kmem.use_lock)
79   release(&kmem.lock);
80 }
81
82 // Allocate one 4096-byte page of physic
83 // Returns a pointer that the kernel can
84 // Returns 0 if the memory cannot be all
85 char*
86 kalloc(void)
87 {
88   struct run *r;
89
90   if(kmem.use_lock)
91     acquire(&kmem.lock);
92   r = kmem.freelist;
93   if(r)
94   {
95     kmem.freelist = r->next;
96     kmem.numFreePages = kmem.numFreePage
97     out from the freelist so decrease the n
98   }
99   if(kmem.use_lock)
100     release(&kmem.lock);
101   return (char*)r;
102 }
103
104 // Returns the number of free pages.
105 int
106 getNumFreePages(void)
107 {
108   if(kmem.use_lock)
109     acquire(&kmem.lock);
110   int r = kmem.numFreePages;
111   if(kmem.use_lock)
112     release(&kmem.lock);
113   return (r);
114 }
```

Initialize `numFreePages` in `kinit1`:



```
Open  kalloc.c
20 struct {
21   struct spinlock lock;
22   int use_lock;
23   struct run *freelist;
24   int numFreePages;
25 } kmem;
26
27 // Initialization happens in two phases.
28 // 1. main() calls kinit1() while still
29 // just
30 // the pages mapped by entrypgdir on fre
31 // 2. main() calls kinit2() with the res
32 // after installing a full page table th
33 void
34 kinit1(void *vstart, void *vend)
35 {
36   initlock(&kmem.lock, "kmem");
37   kmem.use_lock = 0;
38   kmem.numFreePages = 0;
39   freerange(vstart, vend);
40 }
```


In kfree:



```
51 char *p;
52 p = (char*)PGROUNDUP((uint)vstart);
53 for(; p + PGSIZE <= (char*)vend; p += PGSIZE)
54     kfree(p);
55 }
56 //PAGEBREAK: 21
57 // Free the page of physical memory pointed at by v,
58 // which normally should have been returned by a
59 // call to kalloc(). (The exception is when
60 // initializing the allocator; see kinit above.)
61 void
62 kfree(char *v)
63 {
64     struct run *r;
65
66     if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
67         panic("kfree");
68
69     // Fill with junk to catch dangling refs.
70     memset(v, 1, PGSIZE);
71
72     if(kmem.use_lock)
73         acquire(&kmem.lock);
74     r = (struct run*)v;
75     r->next = kmem.freelist;
76     kmem.freelist = r;
77     kmem.numFreePages = kmem.numFreePages + 1; // A new node is added
to freelist so increase the number of free pages
```

In kalloc:



```
68 // Fill with junk to catch dangling refs.
69 memset(v, 1, PGSIZE);
70
71 if(kmem.use_lock)
72     acquire(&kmem.lock);
73 r = (struct run*)v;
74 r->next = kmem.freelist;
75 kmem.freelist = r;
76 kmem.numFreePages = kmem.numFreePages + 1; // A new node is added
to freelist so increase the number of free pages
77 if(kmem.use_lock)
78     release(&kmem.lock);
79 }
80
81 // Allocate one 4096 byte page of physical memory.
82 // Returns a pointer that the kernel can use.
83 // Returns 0 if the memory cannot be allocated.
84 char*
85 kalloc(void)
86 {
87     struct run *r;
88
89     if(kmem.use_lock)
90         acquire(&kmem.lock);
91     r = kmem.freelist;
92     if(r)
93         kmem.freelist = r->next;
94     kmem.numFreePages = kmem.numFreePages - 1; // A node is popped
out from the freelist so decrease the number of free pages.
95 }
96
97
```

Reinstalling of page table:

Whenever the flags are changed in copyvm function the page table must be reinstalled using:

```
lcr3(v2p(pgdir)); // reinstall the page table
```

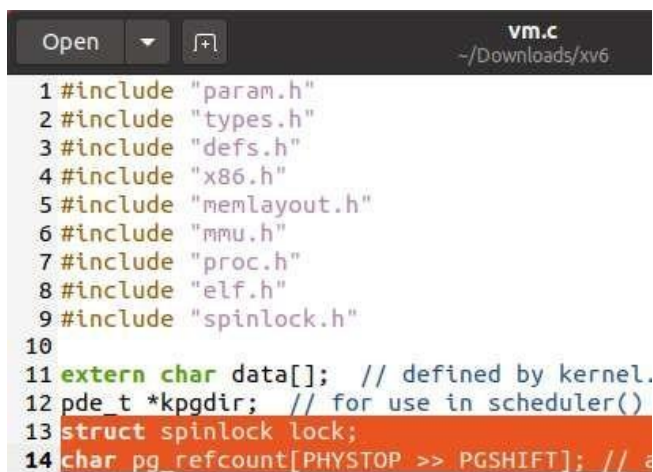
Keeping track of the reference count of pages:

In vm.c add the declaration of array and lock:

Begin with changes to kalloc.c. To correctly implement CoW fork, you must track reference counts of memory pages. A reference count of a page should indicate the number of processes that map the page into their virtual address space. The reference count of a page is set to one when a freepage is allocated for use by some process.

```
struct spinlock lock;
```

```
char pg_refcount[PHYSTOP >> PGSHIFT]; // array to store refcount
```

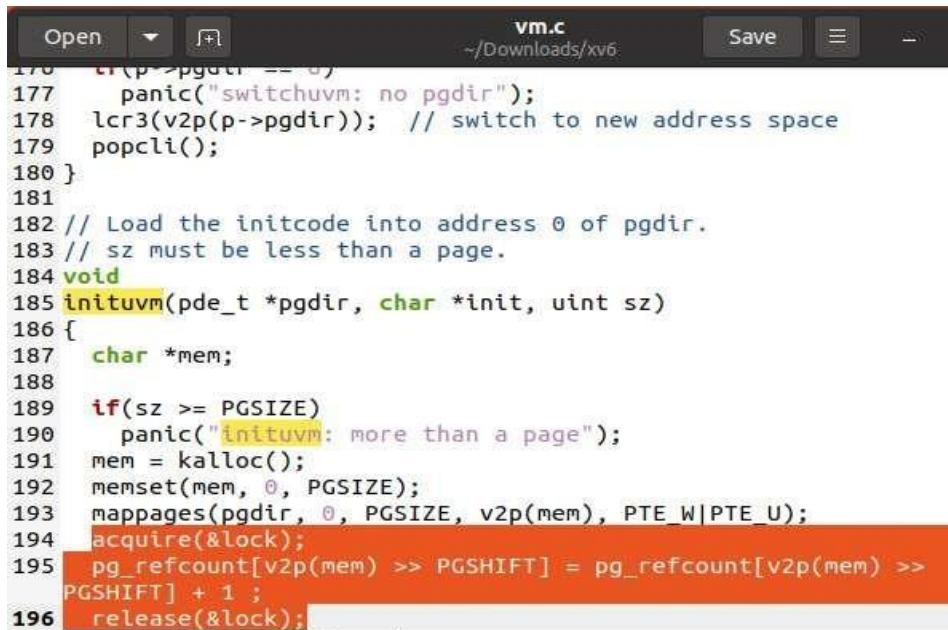


```
1 #include "param.h"
2 #include "types.h"
3 #include "defs.h"
4 #include "x86.h"
5 #include "memlayout.h"
6 #include "mmu.h"
7 #include "proc.h"
8 #include "elf.h"
9 #include "spinlock.h"
10
11 extern char data[]; // defined by kernel.
12 pde_t *kpgdir; // for use in scheduler()
13 struct spinlock lock;
14 char pg_refcount[PHYSTOP >> PGSHIFT]; // e
```

In initvm function:

When a freepage is allocated for use by some process. Whenever an additional process points to an already existing page (e.g., when parent forks a child and both share the same memory page), the reference count must be incremented.

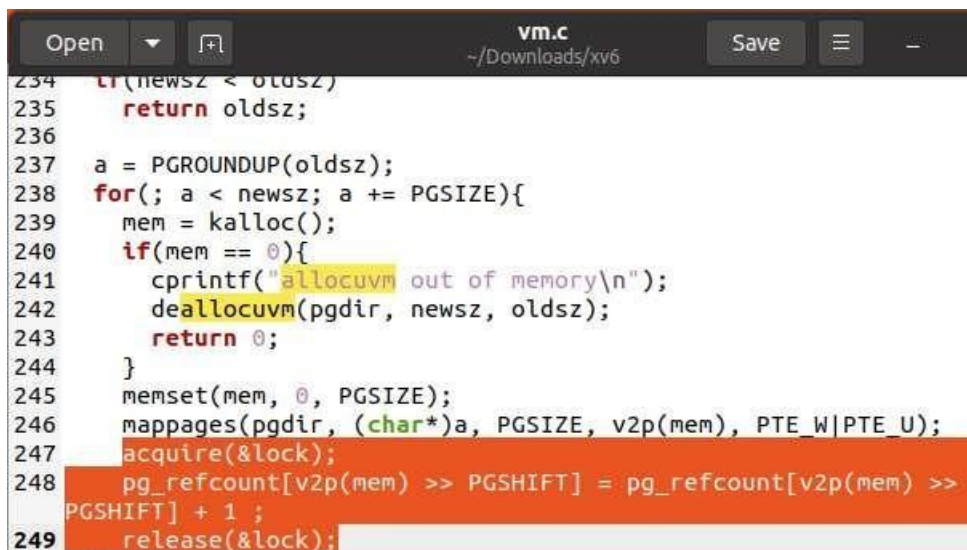
```
acquire(&lock);
pg_refcount[v2p(mem) >> PGSHIFT] = pg_refcount[v2p(mem) >> PGSHIFT] + 1 ;
release(&lock);
```



```
176 if (p->pgdir == 0)
177     panic("switchvm: no pgdir");
178 lcr3(v2p(p->pgdir)); // switch to new address space
179 popcli();
180 }
181
182 // Load the initcode into address 0 of pgdir.
183 // sz must be less than a page.
184 void
185 initvm(pde_t *pgdir, char *init, uint sz)
186 {
187     char *mem;
188
189     if (sz >= PGSIZE)
190         panic("initvm: more than a page");
191     mem = kalloc();
192     memset(mem, 0, PGSIZE);
193     mappages(pgdir, 0, PGSIZE, v2p(mem), PTE_W|PTE_U);
194     acquire(&lock);
195     pg_refcount[v2p(mem) >> PGSHIFT] = pg_refcount[v2p(mem) >>
196     PGSHIFT] + 1 ;
197     release(&lock);
```

In allocvm function:

```
acquire(&lock);
pg_refcount[v2p(mem) >> PGSHIFT] = pg_refcount[v2p(mem) >> PGSHIFT] + 1 ;
release(&lock);
```

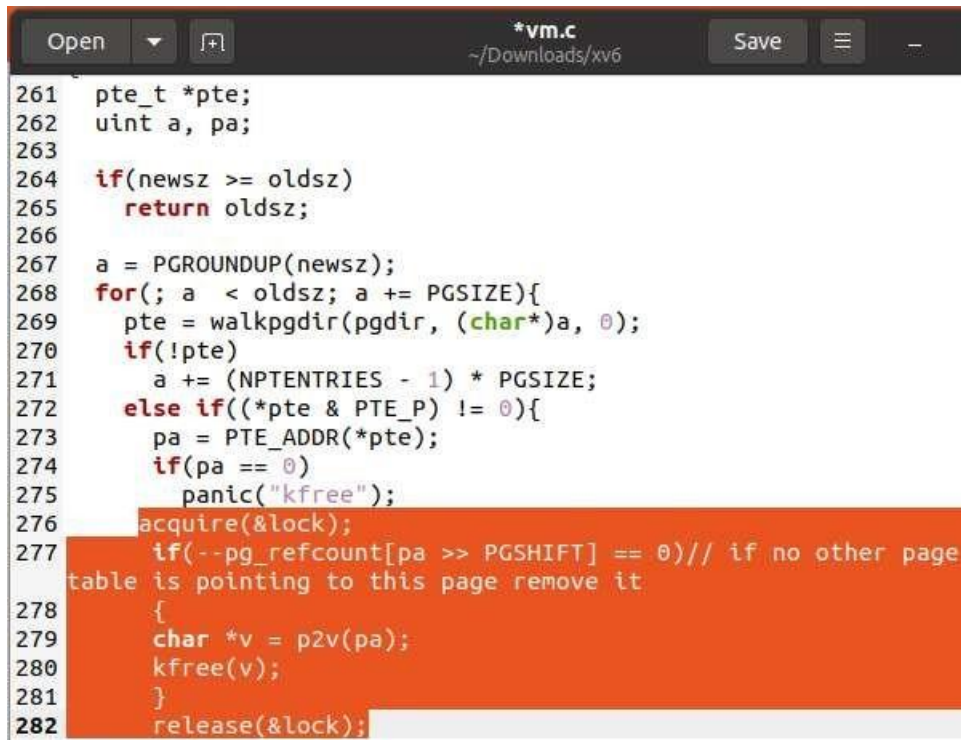


```
234 if (newsz < oldsz)
235     return oldsz;
236
237 a = PGROUNDUP(oldsz);
238 for (; a < newsz; a += PGSIZE){
239     mem = kalloc();
240     if (mem == 0){
241         cprintf("allocvm out of memory\n");
242         deallocvm(pgdir, newsz, oldsz);
243         return 0;
244     }
245     memset(mem, 0, PGSIZE);
246     mappages(pgdir, (char*)a, PGSIZE, v2p(mem), PTE_W|PTE_U);
247     acquire(&lock);
248     pg_refcount[v2p(mem) >> PGSHIFT] = pg_refcount[v2p(mem) >>
249     PGSHIFT] + 1 ;
250     release(&lock);
```

In deallocvmm free the page only when no other page table is pointing it:

The reference count must be decremented when a process no longer points to the page from its page table. A page can be freed up and returned to the freelist only when there are no active references to it, i.e., when its reference count is zero.

```
acquire(&lock);
if(--pg_refcount[pa >> PGSHIFT] == 0)// if no other page table
is pointing to this page remove it
{
char *v = p2v(pa);
kfree(v);
}
release(&lock);
```



```
*vm.c
~/Downloads/xv6

261 pte_t *pte;
262 uint a, pa;
263
264 if(newsz >= oldsz)
265     return oldsz;
266
267 a = PGROUNDUP(newsz);
268 for(; a < oldsz; a += PGSIZE){
269     pte = walkpgdir(pgdir, (char*)a, 0);
270     if(!pte)
271         a += (NPENTRIES - 1) * PGSIZE;
272     else if((*pte & PTE_P) != 0){
273         pa = PTE_ADDR(*pte);
274         if(pa == 0)
275             panic("kfree");
276         acquire(&lock);
277         if(--pg_refcount[pa >> PGSHIFT] == 0)// if no other page
table is pointing to this page remove it
278         {
279             char *v = p2v(pa);
280             kfree(v);
281         }
282         release(&lock);
```

In copyvmm function when a process is forked the refcount of that permanent address should be incremented:

```
acquire(&lock);
pg_refcount[pa >> PGSHIFT] = pg_refcount[pa >> PGSHIFT] + 1; //
increase reference count of that permanent page.
release(&lock);
```

```
Open  [icon]  *vm.c  Save  [icon]  [icon]  [icon]  [icon]
~/Downloads/xv6

336     if(!(*pte & PTE_P))
337         panic("copyuvm: page not present");
338     *pte &= ~PTE_W;
339     pa = PTE_ADDR(*pte);
340     flags = PTE_FLAGS(*pte);
341     /*
342     if((mem = kalloc()) == 0)
343         goto bad;
344     memmove(mem, (char*)p2v(pa), PGSIZE);
345     if(mappages(d, (void*)i, PGSIZE, v2p(mem), flags) < 0)
346         /*
347         // No need of page allocation
348         if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0) // map the
child's page table to same permanent addresses
349             goto bad;
350         acquire(&lock);
351         pg_refcount[pa >> PGSHIFT] = pg_refcount[pa >> PGSHIFT] + 1; //
increase reference count of that permanent page.
352         release(&lock);
```

Change of copyuvm function:

Make the pagetable unwritable and then assign the same permanent addresses to the new page table

```
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
    pde_t *d;
    pte_t *pte;
    uint pa, i, flags;
    //char *mem; //No need to allocate new memory
    if((d = setupkvm()) == 0)
        return 0;
    for(i = 0; i < sz; i += PGSIZE){
        if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
            panic("copyuvm: pte should exist");
        if(!(*pte & PTE_P))
            panic("copyuvm: page not present");
        *pte &= ~PTE_W; // make this page table unwritable
        pa = PTE_ADDR(*pte);
        flags = PTE_FLAGS(*pte);
```



```

// No need of page allocation
if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0) // map the
child's page table to same permanent addresses
goto bad;
acquire(&lock);
pg_refcount[pa >> PGSHIFT] = pg_refcount[pa >> PGSHIFT] + 1; //
increase reference count of that permanent page.
release(&lock);
}
lcr3(v2p(pgdir)); // reinstall the page table
return d;
bad:
freevm(d);
lcr3(v2p(pgdir)); // reinstall the page table
return 0;
}

```

Adding trap handler to handle pagefaults:

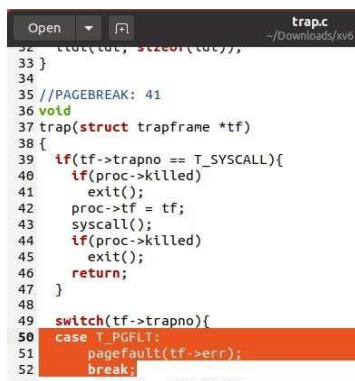
In trap.c:

Once you have changed the fork implementation as described above, both parent and child will execute over the same read-only memory image. Now, when the parent or child processes attempt to write to a page marked read-only, a page fault occurs. The trap handling code in xv6 does not currently handle the T_PGFLT exception (that is defined already, but not caught). You must write a trap handler to handle page faults in trap.c. You can simply print an error message initially, but eventually this trap handling code must call the function that makes a copy of user memory.

```

case T_PGFLT:
pagefault(tf->err);
break;

```



```

trap.c
~/Downloads/xv6
32  tdt(tdt, sizeof(tdt));
33 }
34
35 //PAGEBREAK: 41
36 void
37 trap(struct trapframe *tf)
38 {
39     if(tf->trapno == T_SYSCALL){
40         if(proc->killed)
41             exit();
42         proc->tf = tf;
43         syscall();
44         if(proc->killed)
45             exit();
46         return;
47     }
48
49     switch(tf->trapno){
50     case T_PGFLT:
51         pagefault(tf->err);
52         break;
53     }
54 }

```




In vm.c:

```
void pagefault(uint err_code)
{
    cprintf("Pagefault occured");
    return;
}
```

Adding trap handling function to make copy of user memory:

In vm.c:

The bulk of your changes will be in this new function you will write to handle page faults. When a page fault occurs, the CR2 register holds the faulting virtual address, which you can get using the xv6 function call rcr2(). You must now look at this virtual address and decide what must be done about it. If this address is in an illegal range of virtual addresses that are not mapped in the page table of the process, you must print an error message and kill the process. Otherwise, if this trap was generated due to the CoW pages that were marked as read-only, you must proceed to make copies of the pages as needed.

Note that between the parent and the child, the first one that tries to write to a page should get a new memory page allocated to it. This new page's content must be copied from the contents of the original page pointed to by the virtual address. Even after this copy is made, note that the page is still marked as read only in the page table of the second process, and it will soon trap as well when it attempts to write to the read-only page. When the second process traps, no new pages need to be allocated; it suffices to remove the read-only restriction on the trapping page, since the first process already has its copy. Your page fault handling code should distinguish between these two cases using the reference count variable, and handle them suitably. Make sure you modify the reference counts correctly, and remember to flush the TLB whenever you change page table entries.

```

void pagefault(uint err_code)
{
    uint va = rcr2();
    uint pa;
    pte_t *pte;
    char *mem;
    if(va >= KERNBASE)
    {
        cprintf("pid %d %s: Illegal memory access on CPU %d due
to virtual address 0x%x is mapped to kernel code. So killing
the process\n", proc->pid, proc->name, cpu->id, va);
        proc->killed = 1;
        return;
    }
    if((pte = walkpgdir(proc->pgdir, (void*)va, 0))==0)
    {
        cprintf("pid %d %s: Illegal memory access on CPU %d due
to virtual address 0x%x is mapped to NULL pte. So killing the
process\n", proc->pid, proc->name, cpu->id, va);
        proc->killed = 1;
        return;
    }
    if(!(*pte & PTE_P))
    {
        cprintf("pid %d %s: Illegal memory access on CPU %d due
to virtual address 0x%x is mapped to pte which is not
present.
So killing the process\n", proc->pid, proc->name, cpu->id,
va);
        proc->killed = 1;
        return;
    }
    if(!(*pte & PTE_U))
    {
        cprintf("pid %d %s: Illegal memory access on CPU %d due
to virtual address 0x%x is mapped to pte which is not
accessible to user. So killing the process\n", proc->pid,
proc->name, cpu->id, va);
        proc->killed = 1;
    }
}

```

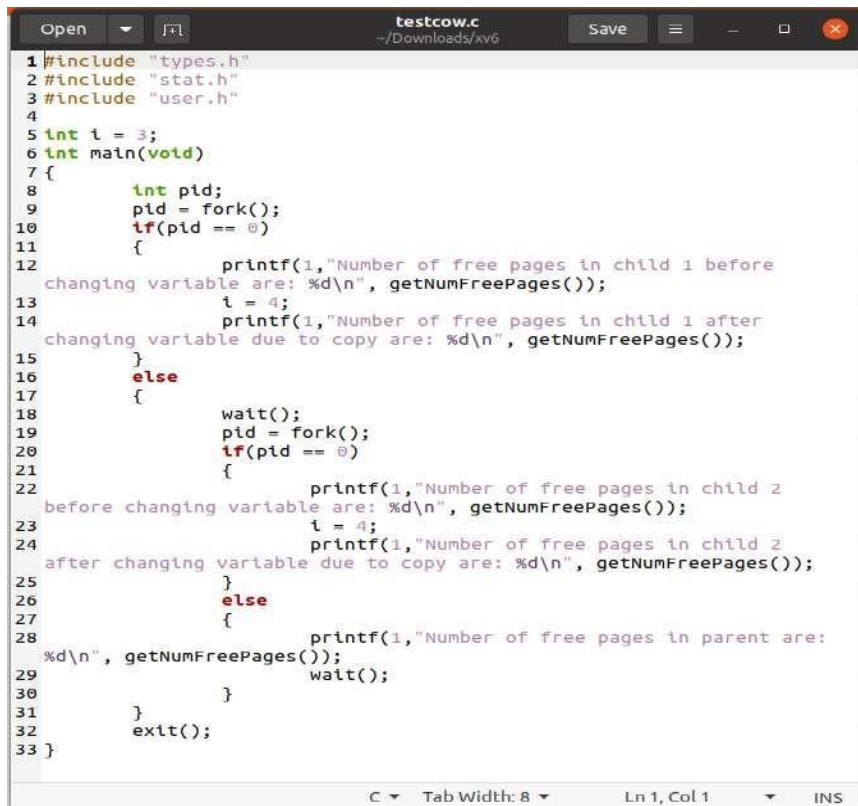
```
return;
}
if(*pte & PTE_W)
{
panic("Unknown page fault due to a writable pte");
}
else
{
pa = PTE_ADDR(*pte);
acquire(&lock);
if(pg_refcount[pa >> PGSHIFT] == 1)
{
release(&lock);
*pte |= PTE_W;
}
else
{
if(pg_refcount[pa >> PGSHIFT] > 1)
{
release(&lock);
if((mem = kalloc()) == 0)
{
cprintf("pid %d %s: Pagefault due to out of
memory", proc->pid, proc->name);
proc->killed = 1;
return;
}
memmove(mem, (char*)p2v(pa), PGSIZE);
acquire(&lock);
```

```

pg_refcount[pa >> PGSHIFT] = pg_refcount[pa >> PGSHIFT]
- 1;
pg_refcount[v2p(mem) >> PGSHIFT] = pg_refcount[v2p(mem) >>
PGSHIFT] + 1;
release(&lock);
*pte = v2p(mem) | PTE_P | PTE_W | PTE_U;
}
else
{
release(&lock);
panic("Pagefault due to wrong ref count");
}
}
lcr3(v2p(proc->pgdir));
}
}

```

Test case testcow:



```

testcow.c
~/Downloads/xv6

1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 int t = 3;
6 int main(void)
7 {
8     int pid;
9     pid = fork();
10    if(pid == 0)
11    {
12        printf(1, "Number of free pages in child 1 before
changing variable are: %d\n", getNumFreePages());
13        t = 4;
14        printf(1, "Number of free pages in child 1 after
changing variable due to copy are: %d\n", getNumFreePages());
15    }
16    else
17    {
18        wait();
19        pid = fork();
20        if(pid == 0)
21        {
22            printf(1, "Number of free pages in child 2
before changing variable are: %d\n", getNumFreePages());
23            t = 4;
24            printf(1, "Number of free pages in child 2
after changing variable due to copy are: %d\n", getNumFreePages());
25        }
26        else
27        {
28            printf(1, "Number of free pages in parent are:
%d\n", getNumFreePages());
29            wait();
30        }
31    }
32    exit();
33 }

```

```
ankith@ankith-VirtualBox: ~/xv6
ankith@ankith-VirtualBox:~$ cd xv6
ankith@ankith-VirtualBox:~/xv6$ make qemu
qemu-system-x86_64 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

Booting from Hard Disk...
—
```


Result:

Output of above test case is :

\$ testcow

Number of free pages in child 1 before changing variable are:

56710 Number of free pages in child 1 after changing variable due
to copy are: 56709

Number of free pages in parent are: 56710

Number of free pages in child 2 before changing variable are:

56710 Number of free pages in child 2 after changing variable due
to copy are: 56709

References :

- <https://stackoverflow.com/questions/21653195/xv6-add-a-system-call-that-counts-system-calls>
- <https://www.youtube.com/watch?v=21SVYiKhcwM> 2.
<https://medium.com/@silvamatteus/adding-new-system-calls-to-xv6-217b7daefbe1>
- <https://medium.com/@viduniwickramarachchi/add-a-new-system-call-in-xv6-5486c2437573>.
- <https://stackoverflow.com/questions/8021774/how-do-i-add-a-system-call-utility-in-xv6>
- <https://stackoverflow.com/questions/21653195/xv6-add-a-system-call-that-countssystem-calls>
- <https://arjunkrishnababu96.gitlab.io/post/xv6-system-call/>
- https://github.com/sayak119/xv6_scheduler
- https://www.cse.iitb.ac.in/~mythili/teaching/cs347_autumn2016/index.html
- https://www.reddit.com/r/osdev/comments/32dtz0/copy_on_write_fork_are_the_page_tables_hierarchy/

- <https://medium.com/@viduniwickramarachchi/add-a-new-system-call-in-xv6-5486c243757>