ABSTRACT

This is the Project Report which Deals with the Linux Based Environment. In this, we are going to Study about Adding System Call to the Processor Which More People are Not Familiar about. The System Call which is specified in the Kernel with the Different links to it. System Call controls many types of Communication between many System files, Compilers and I/O Devices. In this Project I Specified the Step by Step Process to add the System Call to the Linux Kernel.

In now a days the scope of this Open source Linux OS is Taking a Turn in Competing with Generic OS like Apple Macintosh and Microsoft Windows etc. It is necessary to learn and Implement the Ideas what we have in these Open Sources. The reason behind that is, it is Left for Developer to Innovate the Things.

Chapter 1 Introduction

1.1 OPERATING SYSTEMS

What is an Operating System?

Not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of tasks to perform, very straightforward input to expect (a numbered keypad and a few pre-set buttons) and simple, never-changing hardware to control. For a computer like this, an operating system would be unnecessary baggage, driving up the development and manufacturing costs significantly and adding complexity where none is required. Instead, the computer in a microwave oven simply runs a single hard-wired program all the time.

For other devices, an operating system creates the ability to:

I. Serve a variety of purposes
II. Interact with users in more complicated ways
III. Keep up with needs that change over time

All desktop computers have operating systems. The most common are the Windows family of operating systems developed by Microsoft, the Macintosh operating systems developed by Apple and the UNIX family of operating systems (which have been developed by a whole history of individuals, corporations and collaborators). There are hundreds of other operating systems available for special-purpose applications, including specializations for mainframes, robotics, and manufacturing, real-time control systems and so on.

In any device that has an operating system, there's usually a way to make changes to how the device works. This is far from a happy accident; one of the reasons operating systems are made out of portable code rather than permanent physical circuits is so that they can be changed or modified without having to scrap the whole device.

For a desktop computer user, this means you can add a new security update, system patch, new application or even an entirely new operating system rather than junk your computer and start again with a new one when you need to make a change. As long as you understand how an operating system works and how to get at it, in many cases you can change some of the ways it behaves. The same thing goes for your phone, too.[1]

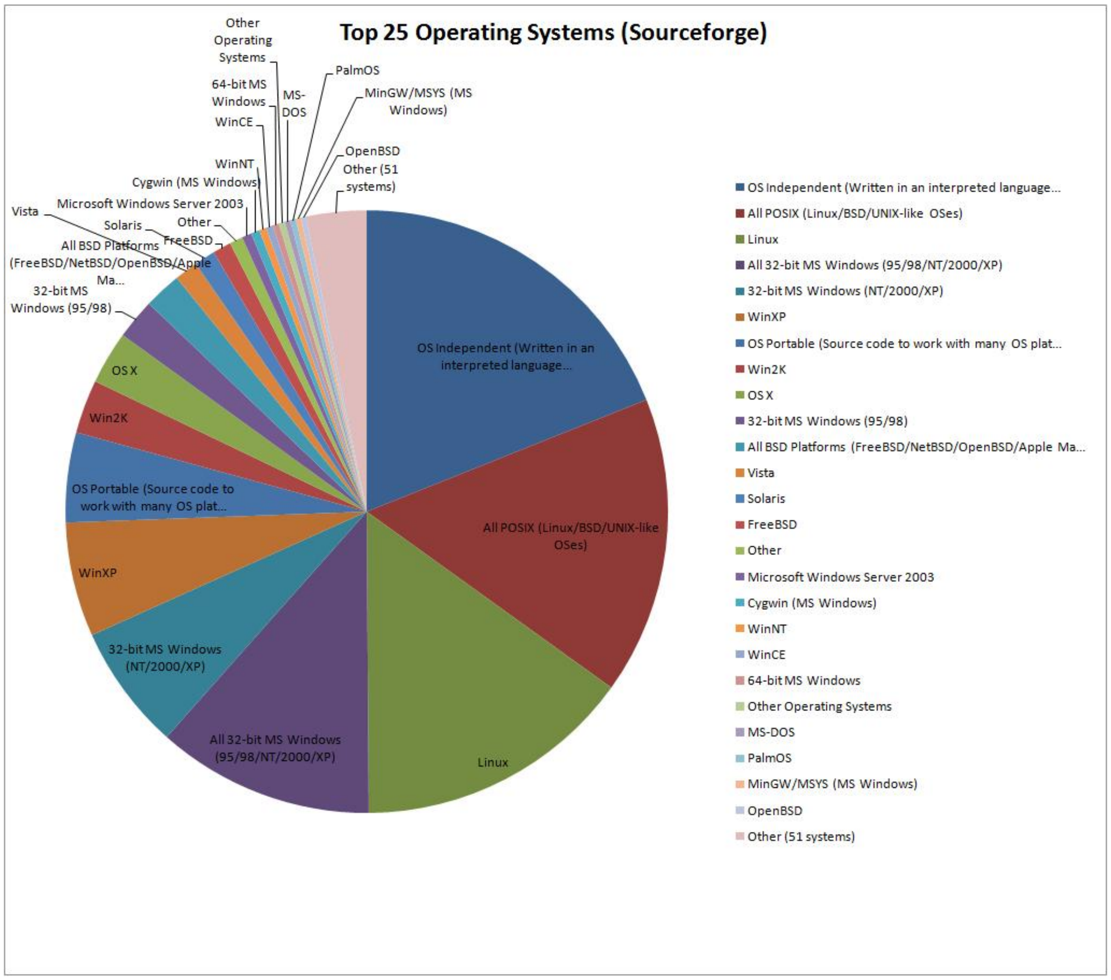


Figure 1.1.1

Real-time

A real-time operating system is a multitasking operating system that aims at executing real-time applications. Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behavior. The main objective of real-time operating systems is their quick and predictable response to events. They have an event-driven or time-sharing design and often aspects of both. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

Multi-user

A multi-user operating system allows multiple users to access a computer system at the same time. Time-sharing systems and Internet servers can be classified as multi-user systems as they enable multiple-user access to a computer through the sharing of time. Single-user operating systems have only one user but may allow multiple programs to run at the same time.

Multi-tasking vs. single-tasking

A multi-tasking operating system allows more than one program to be running at the same time, from the point of view of human time scales. A single-tasking system has only one running program. Multi-tasking can be of two types: pre-emptive and co-operative. In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking, as does AmigaOS. Cooperative multitasking is achieved by relying on each process to give time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking. 32-bit versions of both Windows NT and Win9x, used pre-emptive multi-tasking. Mac OS prior to OS X used to support cooperative multitasking.

Distributed

A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

Templated

In an o/s, distributed and cloud computing context, templating refers to creating a single virtual machine image as a guest operating system, then saving it as a tool for multiple running virtual machines (Gagne, 2012, p. 716). The technique is used both in virtualization and cloud computing management, and is common in large server warehouses.

Embedded

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by

Design. Windows CE and Minix 3 are some examples of embedded operating systems. [2] Priv Rings of OS is Shown below in figure 1.1.2

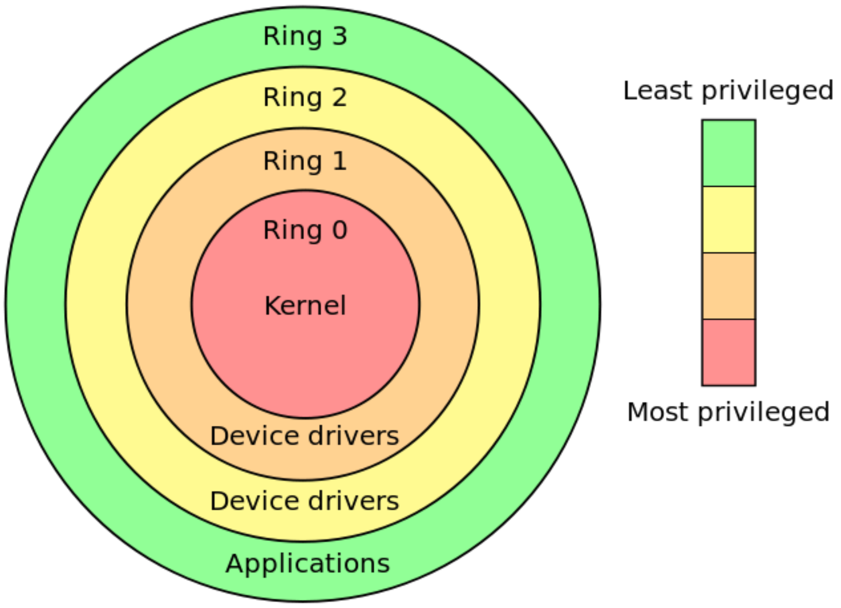


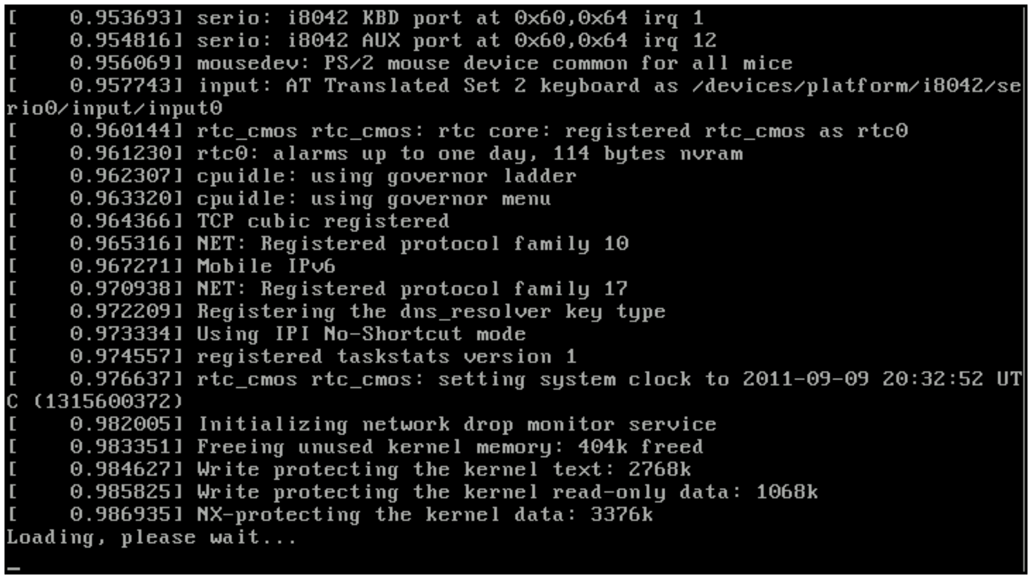
Figure 1.1.2 Priv Rings

1.2 KERNEL

The Linux kernel is a Unix-like operating system kernel used by a variety of operating systems based on it, which are usually in the form of Linux distributions. The Linux kernel is a prominent example of free and open source software.

The Linux kernel is released under the GNU General Public License version 2 (GPLv2) (plus some firmware images with various non-free licenses, and is developed by contributors worldwide. Day-to-day development discussions take place on the Linux kernel mailing list.

The Linux kernel was initially conceived and created in 1991 by Finnish computer science student Linus Torvalds. Linux rapidly accumulated developers and users who adapted code from other free software projects for use with the new operating system. The Linux kernel has received contributions from thousands of programmers. [3] Figure 1.2.1 shows the booting of linux Kernel 3.0.0



```
[    0.953693] serio: i8042 KBD port at 0x60,0x64 irq 1
[    0.954816] serio: i8042 AUX port at 0x60,0x64 irq 12
[    0.956069] mousedev: PS/2 mouse device common for all mice
[    0.957743] input: AT Translated Set 2 keyboard as /devices/platform/i8042/se
rio0/input/input0
[    0.960144] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as rtc0
[    0.961230] rtc0: alarms up to one day, 114 bytes nvram
[    0.962307] cpuidle: using governor ladder
[    0.963320] cpuidle: using governor menu
[    0.964366] TCP cubic registered
[    0.965316] NET: Registered protocol family 10
[    0.967271] Mobile IPv6
[    0.970938] NET: Registered protocol family 17
[    0.972209] Registering the dns_resolver key type
[    0.973334] Using IPI No-Shortcut mode
[    0.974557] registered taskstats version 1
[    0.976637] rtc_cmos rtc_cmos: setting system clock to 2011-09-09 20:32:52 UT
C (1315600372)
[    0.982005] Initializing network drop monitor service
[    0.983351] Freeing unused kernel memory: 404k freed
[    0.984627] Write protecting the kernel text: 2768k
[    0.985825] Write protecting the kernel read-only data: 1068k
[    0.986935] NX-protecting the kernel data: 3376k
Loading, please wait...
_
```

Figure 1.2.1 Linux kernel 3.0.0 booting

1.3 SYSTEM CALLS

In computing, a system call is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). System calls provide an essential interface between a process and the operating system.

The design of the microprocessor architecture on practically all modern systems (except some embedded systems) involves a security model (such as the rings model) which specifies multiple privilege levels under which software may be executed; for instance, a program is usually limited to its own address space so that it cannot access or modify other running programs or the operating system itself, and a program is usually prevented from directly manipulating hardware devices (e.g. the frame buffer or network devices).

However, many normal applications obviously need access to these components, so system calls are made available by the operating system to provide well-defined, safe implementations for such operations. The operating system executes at the highest level of privilege, and allows applications to request services via system calls, which are often executed via interrupts; an interrupt automatically puts the CPU into some required privilege level, and then passes control to the kernel, which determines whether the calling program should be granted the requested service. If the service is granted, the kernel executes a specific set of

Instructions over which the calling program has no direct control, returns the privilege level to that of the calling program, and then returns control to the calling program. [3] Below Diagram Narrates the System Call Interface (SCI) which is a Linux Kernel's API to rest of the OS.

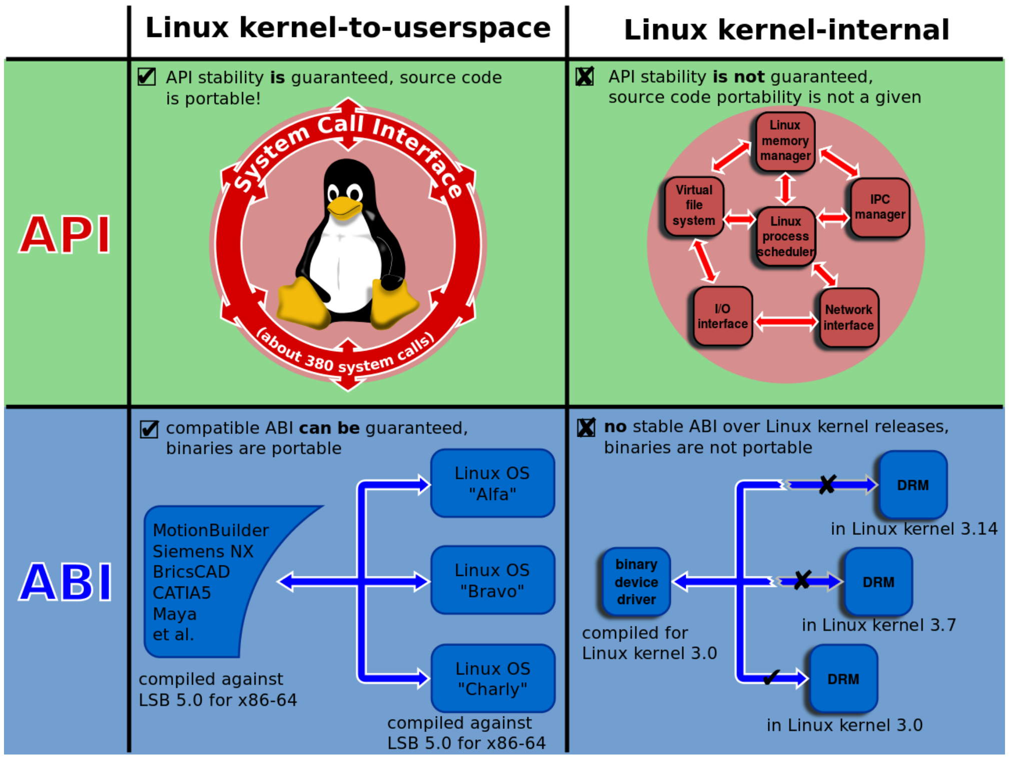

Linux kernel-to-userspace
Linux kernel-internal
API
API stability is guaranteed, source code is portable!
System Call Interface
(about 380 system calls)
API stability is not guaranteed, source code portability is not a given
Linux memory manager
Virtual file system
IPC manager
Linux process scheduler
I/O interface
Network interface
ABI
compatible ABI can be guaranteed, binaries are portable
MotionBuilder
Siemens NX
BricsCAD
CATIA5
Maya
et al.
compiled against LSB 5.0 for x86-64
Linux OS "Alfa"
Linux OS "Bravo"
Linux OS "Charly"
compiled against LSB 5.0 for x86-64
no stable ABI over Linux kernel releases, binaries are not portable
binary device driver
compiled for Linux kernel 3.0
DRM
in Linux kernel 3.14
DRM
in Linux kernel 3.7
DRM
in Linux kernel 3.0

Figure 1.3.1 The System Call Interface (SCI) is a Linux Kernel's API to the rest of OS

CHAPTER 2 ARCHITECTURE USED

In this Section I am Going to Specify the Specification of My Computer Which I Have Used. Basically, it is a Desktop Having Quad core **Intel 2nd Gen i3** Processor **3.10Ghz**. I Installed Linux Named **Ubuntu 14.04** Which Helped Me To Get Through this Project. The Kernel Source Code Which I Have Used is **Linux 3.3.8** Stable Version.

Before **14.04** this is Also Tried in **Fedora 16**, **Ubuntu 12.04 LTS** and **Ubuntu 13.04 LTS** along With **Linux Mint** & **Linux Mint 2**. The below Snapshot Shows the Kernel Architecture of Linux.

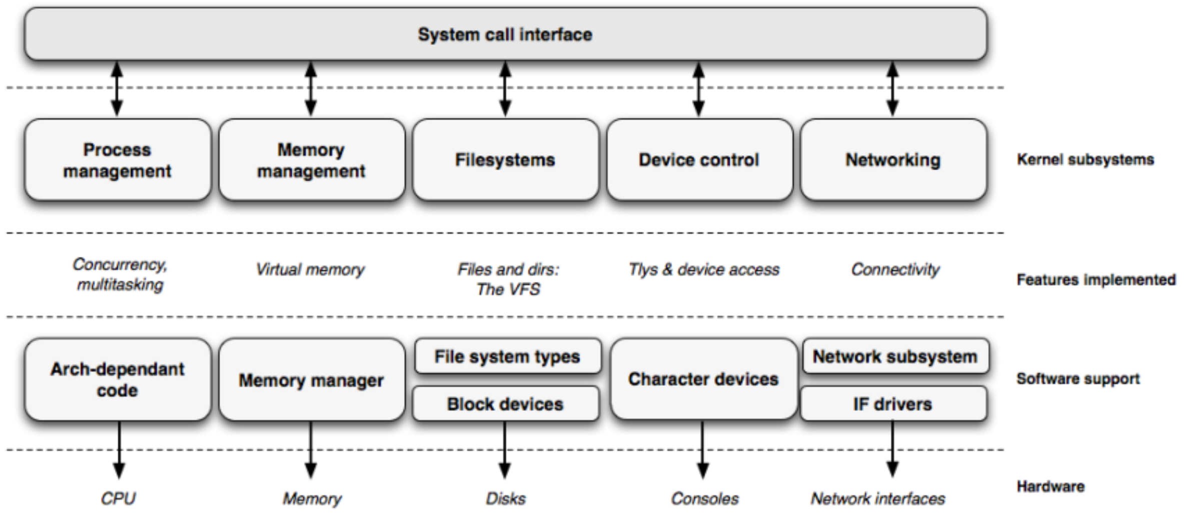


Figure 2.1

CHAPTER 3 IMPLEMENTATION

Step 1: Downloaded the Kernel from kernel.org (Stable Version Downloaded) [4]

Saved the file and extracted it to */home/vharithsa/kernel* (Extracted File was around 540MBs)

Step 2: Open Terminal

In Terminal Gone to Super user Mode by Using Command su (in Ubuntu sudo). Afterwards Changed the Directory to The Location Where My Extracted File was Present

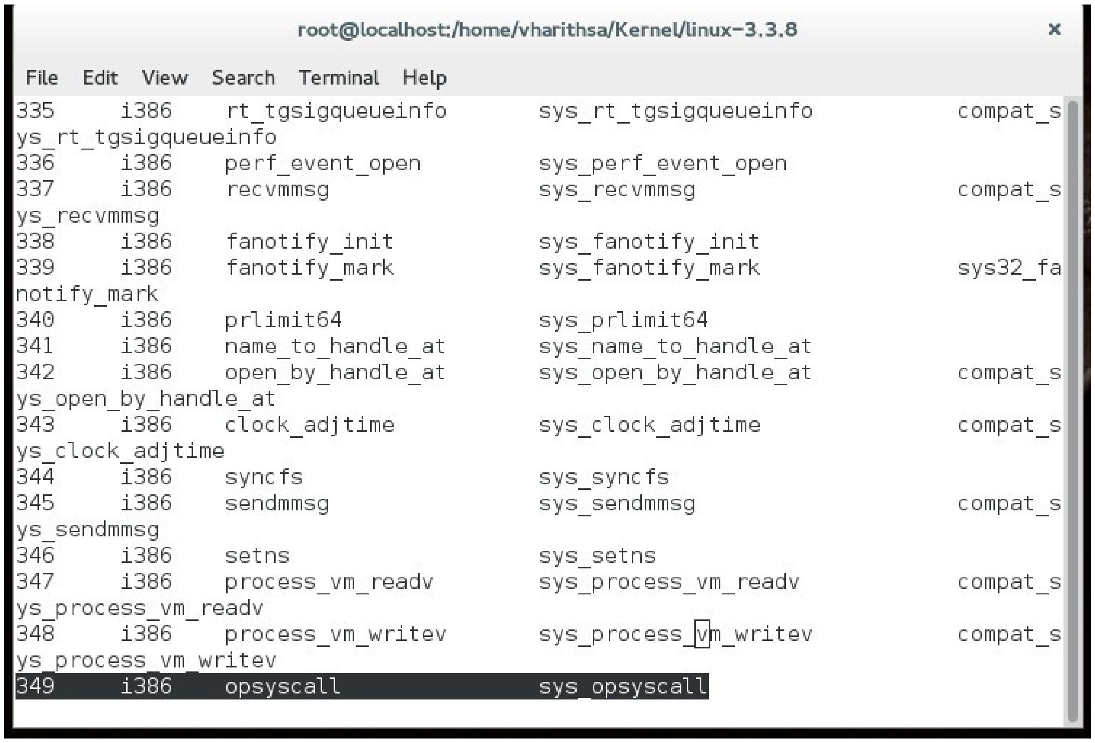
/home/vharithsa/Kernel/linux-3.3.8

Step 3: Appending the System Call to System Call Table

The Table is Present in /arch/x86/syscalls folder through that I got Access to syscall_32.tbl file

#vi /arch/x86/syscalls/syscall_32.tbl

The Line Appended is, *349 i386 opsyscall sys_opsyscall* (349 is the Number of My System Call) as shown in the Snapshot 3.1


root@localhost:/home/vharithsa/Kernel/linux-3.3.8
File Edit View Search Terminal Help

```
335     i386    rt_tgsigqueueinfo       sys_rt_tgsigqueueinfo           compat_s
ys_rt_tgsigqueueinfo
336     i386    perf_event_open         sys_perf_event_open
337     i386    recvmmsg                sys_recvmmsg                    compat_s
ys_recvmmsg
338     i386    fanotify_init           sys_fanotify_init
339     i386    fanotify_mark           sys_fanotify_mark               sys32_fa
notify_mark
340     i386    prlimit64               sys_prlimit64
341     i386    name_to_handle_at       sys_name_to_handle_at
342     i386    open_by_handle_at       sys_open_by_handle_at           compat_s
ys_open_by_handle_at
343     i386    clock_adjtime           sys_clock_adjtime               compat_s
ys_clock_adjtime
344     i386    syncfs                  sys_syncfs
345     i386    sendmmsg                sys_sendmmsg                    compat_s
ys_sendmmsg
346     i386    setns                   sys_setns
347     i386    process_vm_readv        sys_process_vm_readv            compat_s
ys_process_vm_readv
348     i386    process_vm_writev       sys_process_vm_writev           compat_s
ys_process_vm_writev
349     i386    opsyscall               sys_opsyscall
```

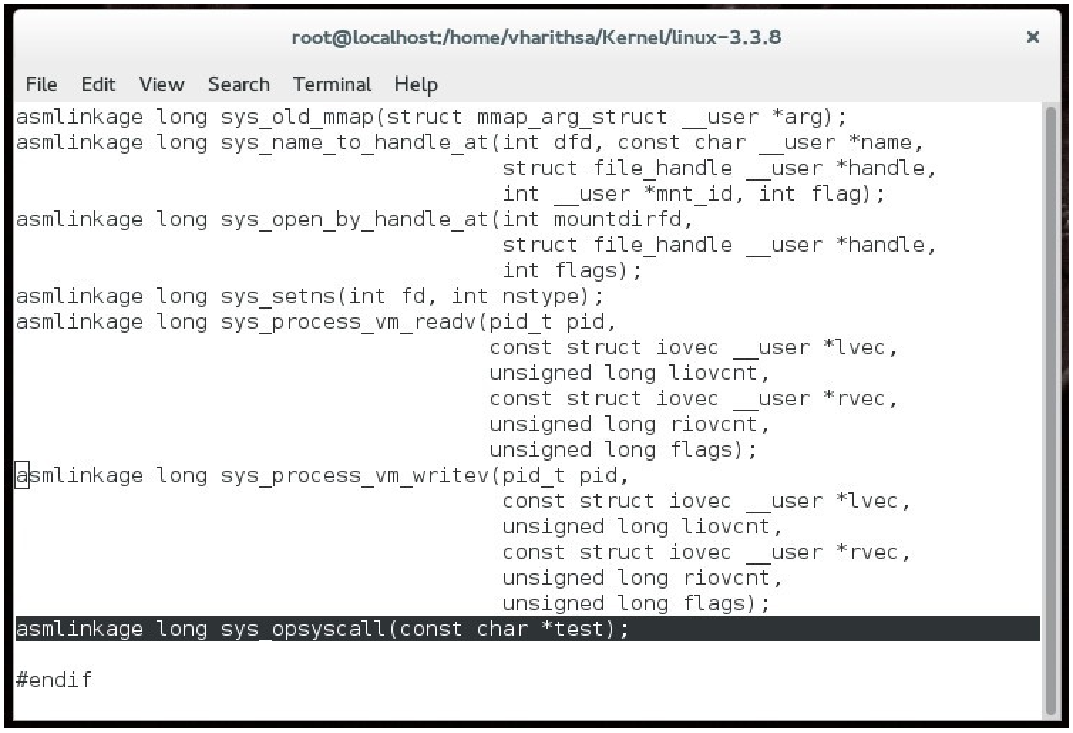
Snapshot 2.1

Step 4: Linking System Call to Header File

#vi include/linux/syscalls.h (Syscalls.h is a Header file)

Linking Line I Appended to the Header File

*asmlinkage long sys_opsyscall(const char *test);* (Snapshot 3.2 Shows the Inserted Line below)


root@localhost:/home/vharithsa/Kernel/linux-3.3.8
File Edit View Search Terminal Help

```
asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
asmlinkage long sys_name_to_handle_at(int dfd, const char __user *name,
                                      struct file_handle __user *handle,
                                      int __user *mnt_id, int flag);
asmlinkage long sys_open_by_handle_at(int mountdirfd,
                                      struct file_handle __user *handle,
                                      int flags);
asmlinkage long sys_setns(int fd, int nstype);
asmlinkage long sys_process_vm_readv(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);
asmlinkage long sys_process_vm_writev(pid_t pid,
                                      const struct iovec __user *lvec,
                                      unsigned long liovcnt,
                                      const struct iovec __user *rvec,
                                      unsigned long riovcnt,
                                      unsigned long flags);
asmlinkage long sys_opsyscall(const char *test);

#endif
```

Snapshot 3.2

Step 5: Create opsyscall Directory

Now I Created a Directory Under linux-3.3.8 (Root Folder of Source Kernel)

#mkdir opsyscall

Step 6: Creating .C file in opsyscall

#vi opsyscall/syscalls.c

Now in The Editor This Program Has Been Inserted:

```
// Printk and KERN_*macros
#include<linux/kernel.h
#include<linux/syscalls.h>

//The System Call
Int main ()
{
Printk (KERN_ALERT "Called with String" %s\n", test);
//o for Success
Return 0;
}
```

Step 7:

Linking Object file to The above Program

#vi opsyscall/Makefile

Type these Lines into the File

```
# include the object in the kernel core
        obj-y  ;= syscalls.o
```

Step 8: Makefile

In the File which is in the Root Directory of the Source Kernel, We need to find

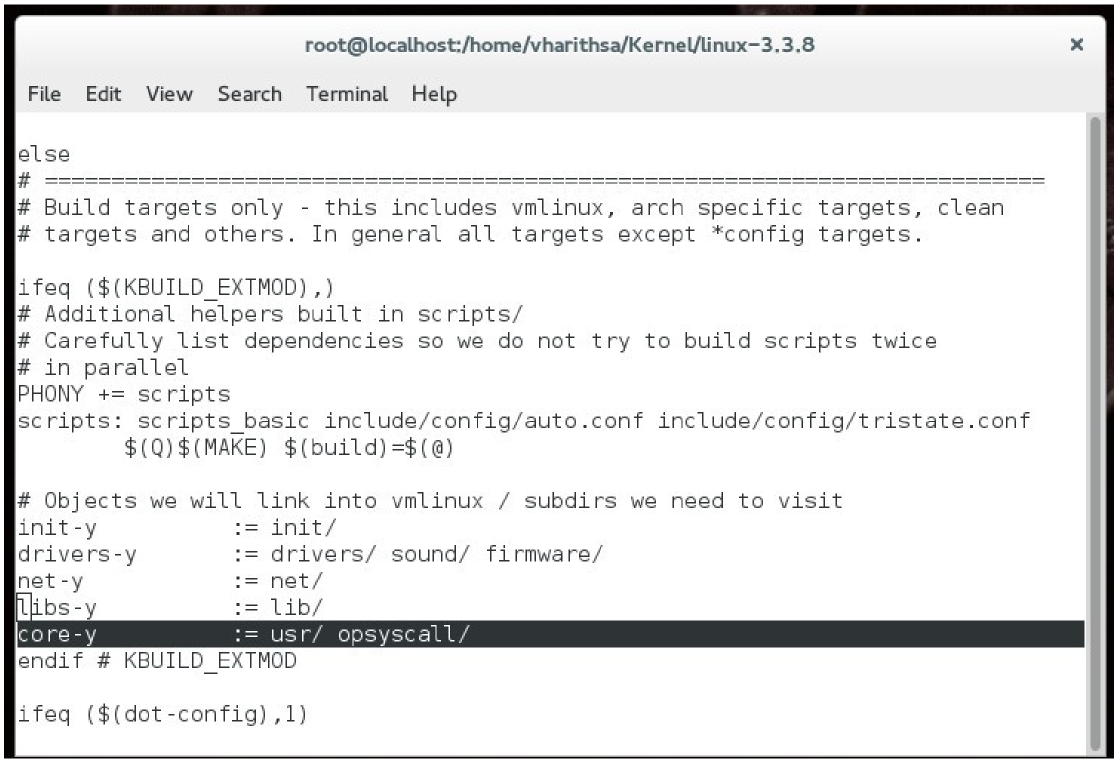
core-y := usr line and Have to replace this By *core-y := usr / opsyscall/*

Snapshot 3.3 shows how I replaced the line.

Step 9: oldconfig

This is The Old method of creating a .config file in The Kernel which is Necessary to compile it. You Have to View Everything which is asked by the Kernel. U must select the Configuration By its given Choice.

#make oldconfig


root@localhost:/home/vharithsa/Kernel/linux-3.3.8
File Edit View Search Terminal Help

```
else
# ===========================================================================
# Build targets only - this includes vmlinux, arch specific targets, clean
# targets and others. In general all targets except *config targets.

ifeq ($(KBUILD_EXTMOD),)
# Additional helpers built in scripts/
# Carefully list dependencies so we do not try to build scripts twice
# in parallel
PHONY += scripts
scripts: scripts_basic include/config/auto.conf include/config/tristate.conf
        $(Q)$(MAKE) $(build)=$(@)

# Objects we will link into vmlinux / subdirs we need to visit
init-y          := init/
drivers-y       := drivers/ sound/ firmware/
net-y           := net/
libs-y          := lib/
core-y          := usr/ opsyscall/
endif # KBUILD_EXTMOD

ifeq ($(dot-config),1)
```

Snapshot 3.3

Step 10: Compile the Kernel

Compile the Kernel Using make Command. Compiling Taken 50 Minutes and After that Install the Kernel.

#make && make install

Step 11: Testing

Open Terminal after Rebooting the System and Create a .c file to Test Ur System Call Status.

#vi userspace.c

Enter this Program in the File:

```
#include<unistd.h>

//unistandard header
file


int main()
{

syscall(349, “vharithsa
is fun”);

return 0;
}
```

Syslog

Before viewing syslog we need to test the Program to so I did execute these Commands [4]

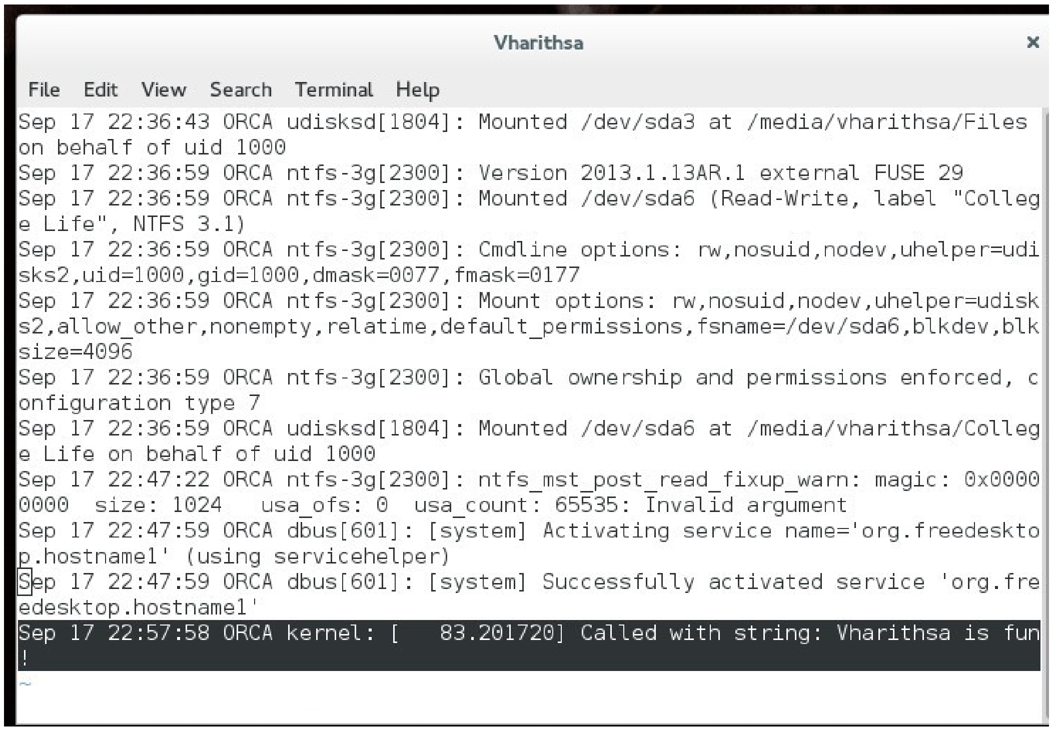
#gcc –o test userspace.c

#./test

Now check the syslog file

# vi var/log/syslog

Snapshot 3.4 is taken while the **syslog** file has opened and it shows the output at the last line along with our tested program.



```
Sep 17 22:36:43 ORCA udisksd[1804]: Mounted /dev/sda3 at /media/vharithsa/Files
on behalf of uid 1000
Sep 17 22:36:59 ORCA ntfs-3g[2300]: Version 2013.1.13AR.1 external FUSE 29
Sep 17 22:36:59 ORCA ntfs-3g[2300]: Mounted /dev/sda6 (Read-Write, label "Colleg
e Life", NTFS 3.1)
Sep 17 22:36:59 ORCA ntfs-3g[2300]: Cmdline options: rw,nosuid,nodev,uhelper=udi
sks2,uid=1000,gid=1000,dmask=0077,fmask=0177
Sep 17 22:36:59 ORCA ntfs-3g[2300]: Mount options: rw,nosuid,nodev,uhelper=udisk
s2,allow_other,nonempty,relatime,default_permissions,fsname=/dev/sda6,blkdev,blk
size=4096
Sep 17 22:36:59 ORCA ntfs-3g[2300]: Global ownership and permissions enforced, c
onfiguration type 7
Sep 17 22:36:59 ORCA udisksd[1804]: Mounted /dev/sda6 at /media/vharithsa/Colleg
e Life on behalf of uid 1000
Sep 17 22:47:22 ORCA ntfs-3g[2300]: ntfs_mst_post_read_fixup_warn: magic: 0x0000
0000  size: 1024   usa_ofs: 0  usa_count: 65535: Invalid argument
Sep 17 22:47:59 ORCA dbus[601]: [system] Activating service name='org.freedeskto
p.hostname1' (using servicehelper)
Sep 17 22:47:59 ORCA dbus[601]: [system] Successfully activated service 'org.fre
edesktop.hostname1'
Sep 17 22:57:58 ORCA kernel: [   83.201720] Called with string: Vharithsa is fun
!
~
```

Snapshot 3.4

CONCLUSION

This Project has been given lot of things to learn. Even though it is a small project in the sense, it has become certain huge for me until I got some concepts about this. It's just a beginning. Now I learnt how to add my own System Call to the Kernel but In future I can go for more complex. Something Big like developing my own OS. Before doing that my Next Project will be building my own Kernel.

System Call is something different which I never heard of. By doing this project I learnt what the Kernel made of and how we can link different programs. Creation of object file, Working and Compilation of Programs etc. these are one of the very few things I studied. Hopefully for the betterment of Knowledge I will learn every second of my life.

Thanks to the Faculty Vinay TP for assigning this project and I am very Kind to Mr. Basavesh D Who also helped me in this project. Finally I want to thank my Computer Which Held my Hand throughout the day.

Implicitly the credit mainly goes to all Faculty of our ISE Department for providing the facility.

Further reading

i) Auslander, Marc A.; Larkin, David C.; Scherr, Allan L. (1981). **The evolution of the MVS Operating System. IBM J. Research & Development.**

ii) Deitel, Harvey M.; Deitel, Paul; Choffnes, David**. Operating Systems. Pearson/ Prentice Hall. ISBN 978-0-13-092641-8.**

iii) Bic, Lubomur F.; Shaw, Alan C. (2003). Operating Systems. **Pearson: Prentice Hall.**

iv) Silberschatz, Avi; Galvin, Peter; Gagne, Greg (2008). **Operating Systems Concepts. John Wiley & Sons. ISBN 0-470-12872-0.**

v) O'Brien, J.A., & Marakas, G.M.(2011). Management Information Systems. **10e. McGraw-Hill Irwin**

vi) Leva, Alberto; Maggio, Martina; Papadopoulos, Alessandro Vittorio; Terraneo, Federico (2013). **Control-based Operating System Design. IET. ISBN 978-1-84919-609-3. [5][3]**

REFERENCES

1. How Stuff Works??
2. Stallings (2005). Operating Systems, Internals and Design Principles. Pearson: Prentice Hall. p. 6.
3. Wikipedia.org
4. Benjamin Buzbee (YouTube Channel)
5. Google.com