

# Linux Kernel Compilation & Working of malloc() in Linux Operating Systems

## **Group Members**

## **Content & Index**

1. Problem.....	4
2. Description.....	4
3. Implementation & Testing.....	9
4. Conclusion.....	39
5. Sources.....	40
6. Appendix.....	41



## 1. Problem

The aim of this project is to understand the compilation process of the Linux kernel and to investigate how malloc () or the dynamic memory allocation works in Linux operating system and the role of kernel in dynamic memory allocation.

## 2. Introduction

This report describes the step by step process of Linux kernel compilation which involves downloading complete Linux kernel source code, setting the configurations, compiling the kernel source. Also, this report describes about our experience in understanding of working of malloc() in Linux operating system by tracing malloc() function and allocation of virtual memory to the user space processes.

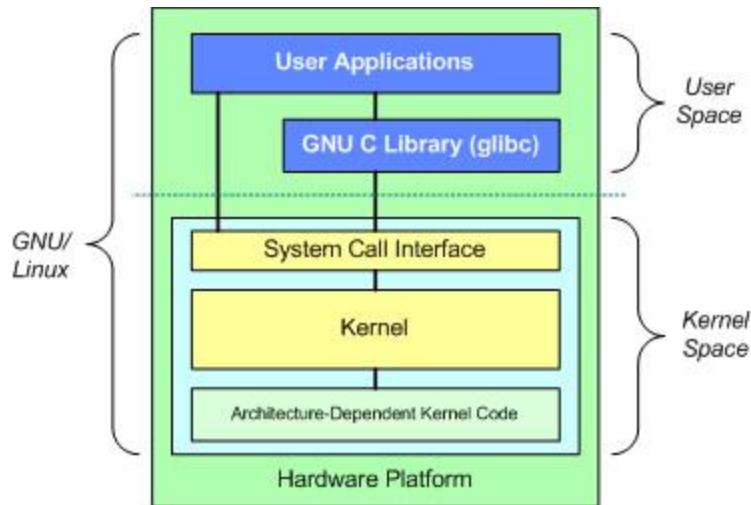
## 2. Description/Background

Linux is a free and open source operating system. It can be easily used by the developers to understand its internals as against to proprietary operating systems like windows. The core of the operating system is called kernel, which our team decided to explore in order to understand its components and functionality.

Kernel is a program which is the main component of the operating system. Kernel interacts with underlying hardware of the computer on one end and user commands on the other end. In Linux, the architecture is divided into user space and kernel space. In user space the user applications are executed and kernel space is where Linux kernel exists. The kernel and user applications occupy different protected address spaces. Each user-space process occupies its own virtual address space and the kernel occupies the single address space.

Also, there is one more important component in the Linux operating system architecture, which is GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and kernel.

In this project, our main aim was to understand the kernel space , the user space applications and their interaction through system calls. The below diagram gives the high level view of the Linux architecture.



To understand the kernel space, we need to download and install the separate kernel source code on our Linux system. The advantage of installing a separate Linux kernel is that we can easily understand the various components of the Linux kernel. Not only we can understand but we can remove, modify the existing components and we can add our own customized components. These components of the Linux kernels are called as loadable kernel modules (lkm), which are pieces of code written in C language. These modules can be loaded and unloaded into the kernel upon demand dynamically. The reason that not all modules are loaded into kernel and are loaded dynamically on demand is because, it takes up large amount of memory if all the modules are loaded. An example of the kernel module is device driver which allows the kernel to access hardware connected to the computer.

The compiled module in the linux operating system can be found at `/lib/modules/kernel-version/` directory. The `modprobe` command can be used to add or remove a module from the Linux kernel. The modprobe command looks in the module directory `/lib/modules/$(uname -r)` for all the modules and other files, except for the optional `/etc/modprobe.conf` configuration file and `/etc/modprobe.d` directory.

## Malloc() in Linux Operating System

### Background

`Malloc()` is a c library function, defined in `malloc.c` file in `glibc`, is memory allocator that can be used to allocate memory dynamically. Its prototype is as defined in `<stdlib.h>`

`Void *(size_t size);`

`Malloc()` takes an argument specifying the size of the memory block that the user wishes to allocate and returns a pointer to a block of memory of that size. If this function fails to allocate the memory because of no memory available, it will return 0.

To deallocate the allocated memory, the free() function of the glibc in linux is used.

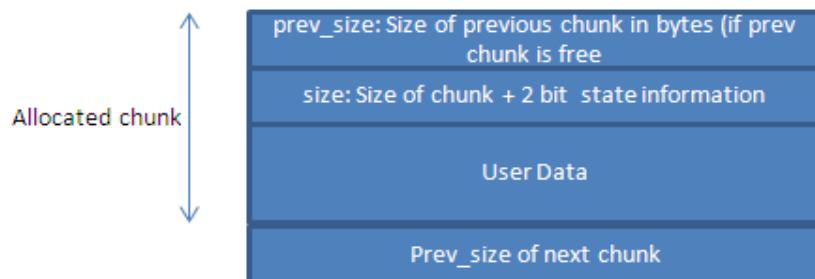
```
Void free(void*ptr);
```

### **How does memory allocator allocate the memory ?**

Different operating systems can implement dynamic allocation in their own which they think makes it efficient. In Linux, the memory allocator or malloc() uses the concept of segregated- “free-lists” or bins. Each free-lists or bins contains blocks for a particular range of sizes.

The Linux allocator keeps 128 bins of different sizes. The first half of the bins keep exactly one size in them. These are for smaller sizes, 16 bytes through 512 bytes. The other half of the bins keep the rest of the sizes between 576 bytes and  $2^{31}$  bytes. Unlike the first half, these bins don't need exact sizes. The sizes found in the given bin can be between the previous bin size and the current size of the bin. All bins that contain different size blocks are sorted from smallest to largest, making the algorithm always return the best-fit for any given size.

When a block is freed, it is combined with its free neighbors, and the resulting block is placed in the appropriate bin. The allocator keeps track of the location of a block's neighbors by making use of a “size tag.” The size tag is four bytes of memory that is allocated along with the block requested by the application. It resides right before the beginning of the user block of memory. This size is also repeated at the end of the block so that it is easy to determine the size of the block right behind any given block as well as the block ahead of any given block.



The allocator uses these size tags to keep track of exactly how much memory is allocated where. That is why we don't specify a size when the memory needs to be free memory — the allocator simply looks back 4 bytes from the given address to determine the size of the block. The size tag at the beginning of the block also keeps a bit reserved to indicate whether the block is free or in use.

The use of these two size tags adds an additional eight bytes to the size of any allocated block. The allocator must also keep the pointers for the list of free blocks for each bin.

In the Linux memory allocator, each bin is a doubly-linked list of free blocks in the given size range. The allocator stores the previous and next pointers for each block in the actual user space of the block that was allocated. By storing two pointers in the freed blocks, the allocator imposes a minimum size of 8 bytes on any allocated user block. Therefore, no matter how few bytes a program allocates, a block of at least 16 bytes will be allocated.

### **Where is memory being allocated in Linux Operating System ?**

Every Linux process has its own dedicated memory address space. The kernel maps these “virtual” addresses in the process address space to actual physical memory addresses as necessary. The data stored at address 0xDEADBEEF in one process is not necessarily the same as the data stored at the same address in another process. When this data is tried to access, the kernel translates the virtual address into an appropriate physical address in order to retrieve the data from memory.

### **What is Heap ?**

In the processes virtual address space, there is an area called as heap from where malloc() allocates memory dynamically to the user processes. The top of the heap is called the break point which specifies the end of the heap area. But if more memory is required by the process, the user application can ask the kernel to increase the heap size by moving up the break point.

### **How does user space applications tell the kernel to increase the heap size or process memory ?**

The user space application or malloc tells the kernel to increase the heap size through brk() system call. When brk() system call is called, the kernel moves the break point of the heap up. There is also sbrk() function which is glibc function that makes increase in the size of heap easy by simply passing the number of bytes the application wishes to increase the size of the heap , as an argument.

brk() system call returns 0 if the new allocation is successful and -1 if not. sbrk() function returns a pointer to the new break point and -1 if the break point could not be moved.

There is an `__curbrk` is a global variable that keeps track of the current position of the break point . The sbrk()function simply adds the increment passed into it to `__curbrk` and passes this value to the brk()system call to actually allocate the memory.

### **What happens when brk() system call is called from userspace ?**

When brk() is called, it goes to the kernel at mm/mmap.c where it is defined. The Linux kernel performs a few checks and then allocates the new memory for the process. The

kernel first aligns the old and the new break point to be on page boundaries. After aligning the addresses to fall on a boundary that is divisible by 4096 bytes, the system call checks to see if the amount of memory is decreased. If so, it immediately decreases the amount of heap space for that process with a call to *do\_munmap()* and returns the new address pointer.

### **What does kernel do when if the application process wants an increase in memory ?**

If the application wants to increase the amount of heap space allocated to it, the kernel performs three checks to make sure the allocation can succeed. The first check verifies that the specified limit of memory for this process will not be exceeded if more memory is allocated. *setrlimit()* function sets the limit of the size of the memory in Linux for the processes.

After verifying that a process is not requiring more memory than its limit allows, the kernel then verifies that increasing the memory will not interfere with any of the other parts of memory for that process. For Example, it should not interfere with stack or mmaped files or text area of the virtual address space.

Finally, as a last check, the kernel checks to see if there is enough memory to accommodate the new process. Once kernel verifies it calls the *do\_brk()* function to increase the memory for the process.

The two basic functions used by the *sys\_brk()* system call to allocate and deallocate memory are *do\_brk()* and *do\_munmap()*. In fact, the *do\_brk()* function is really just a simplified version of the *do\_munmap()* function. These two functions are similar to the *libc* functions *mmap()* and *munmap()*.

In the implementation section, we will try to illustrate to see the files containing above functions and system calls to verify the compilation and the dynamic memory allocation process through *malloc()*.

# 1. Implementation

## Compilation of Linux Kernel

### Platform used

The Linux operating system which we used for our experiment is Ubuntu 12.10 virtual machine. We allocated RAM of 6 GB and Hard disk size of 80 GB's in order to carry out our experiments uninterrupted.

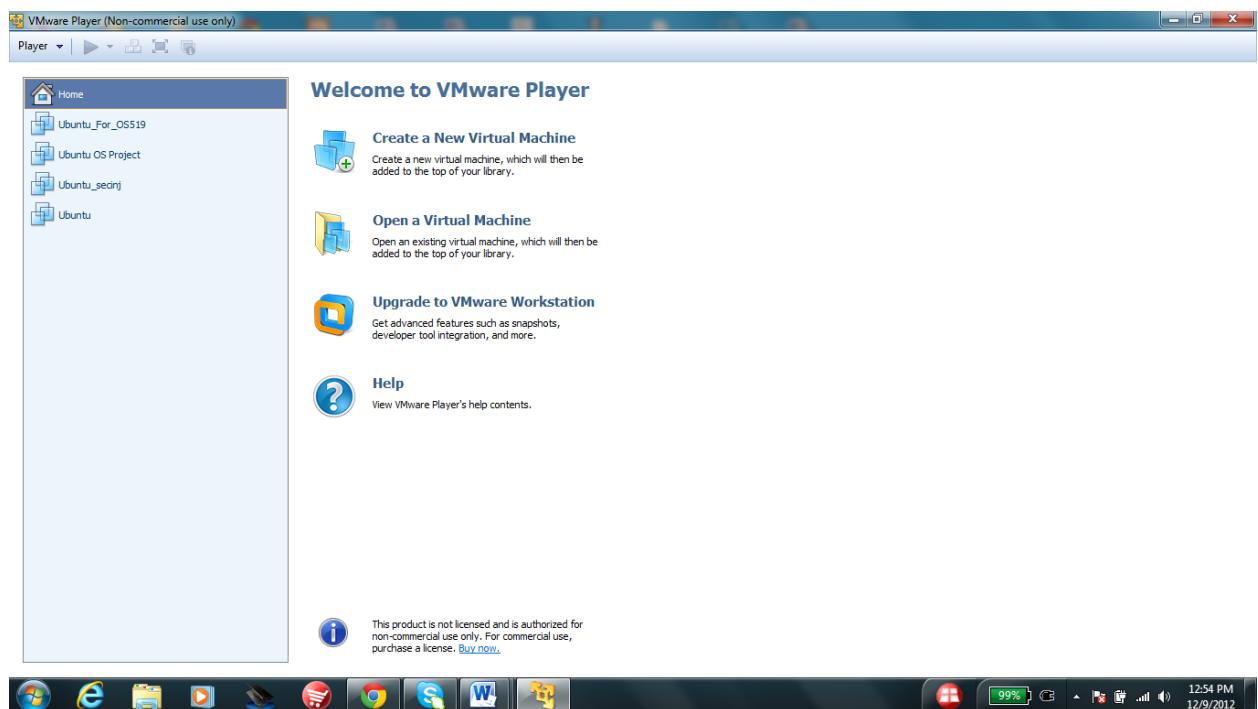
### Why we used Virtual Machine

The reason for using virtual machine in our experiment was to make testing of kernel easy. There are chances while playing with kernel source code, something can go wrong and operating systems needs to be installed again. With the use of virtual machine, we can have the backup of the Linux operating system and can be used anytime if something goes wrong with the kernel which we experienced while performing the compilation of Linux kernel.

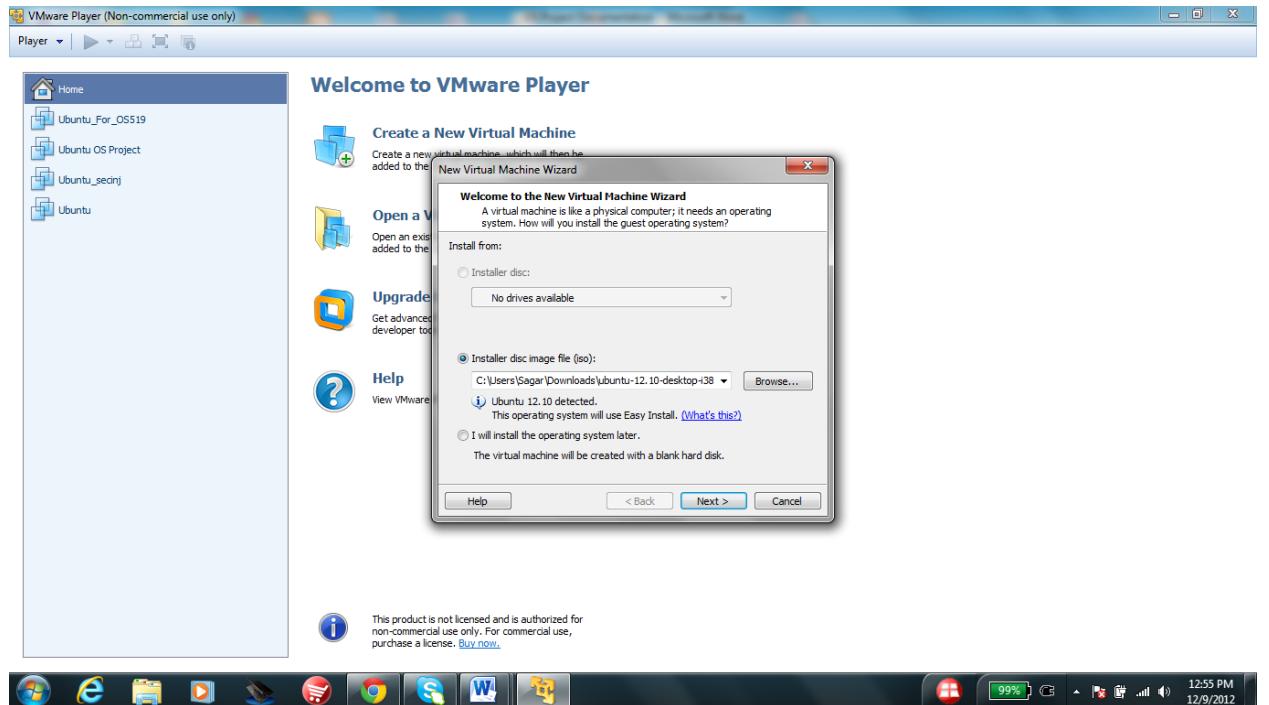
### Proof of Implementation

Installation of virtual machine

1.

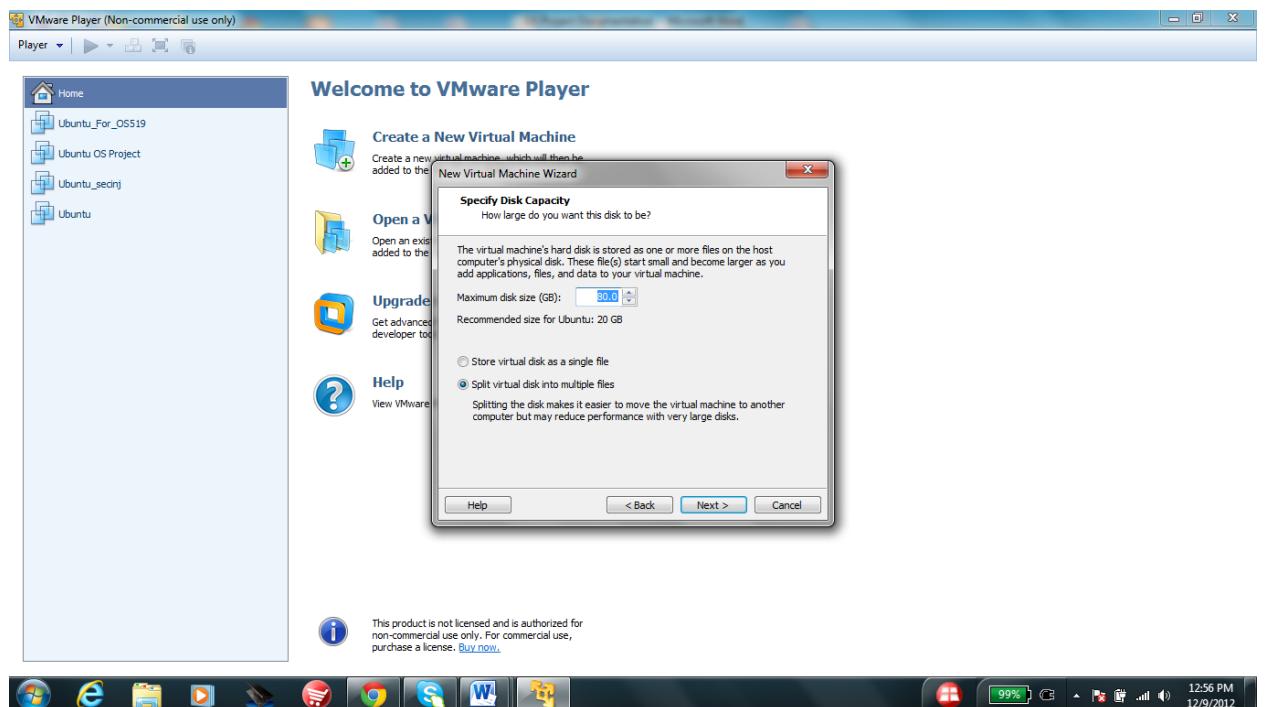


2.



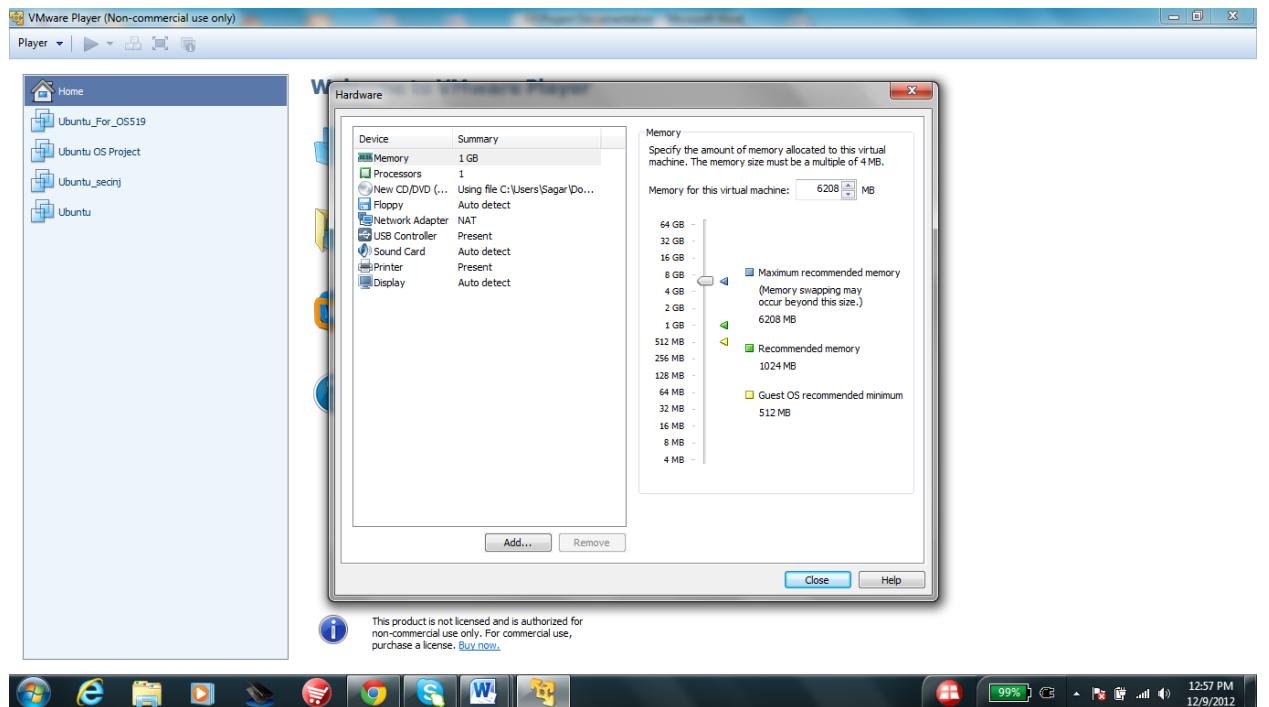
3.

### Allocating Hard disk space



4.

## Allocating 6 GB of RAM



5.

Installation process starts

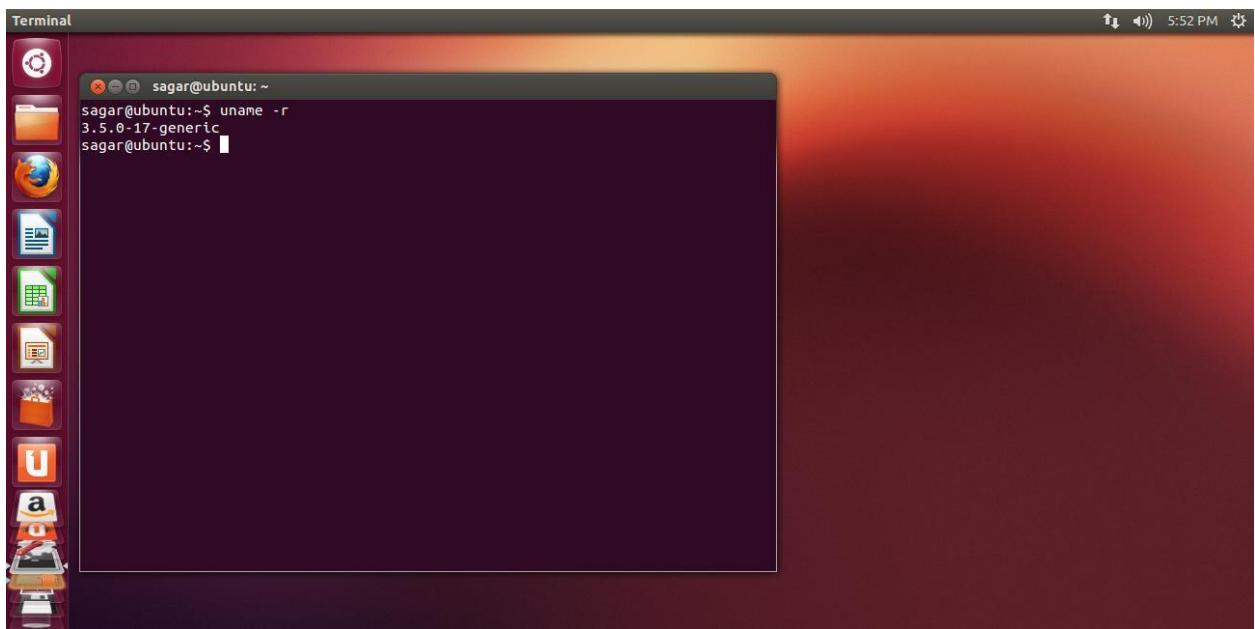


6.



7.

The Ubuntu 12.10 is installed now on our virtual Machine . We used uname –r command to check our original kernel version which is 3.5.0-17-generic



8.

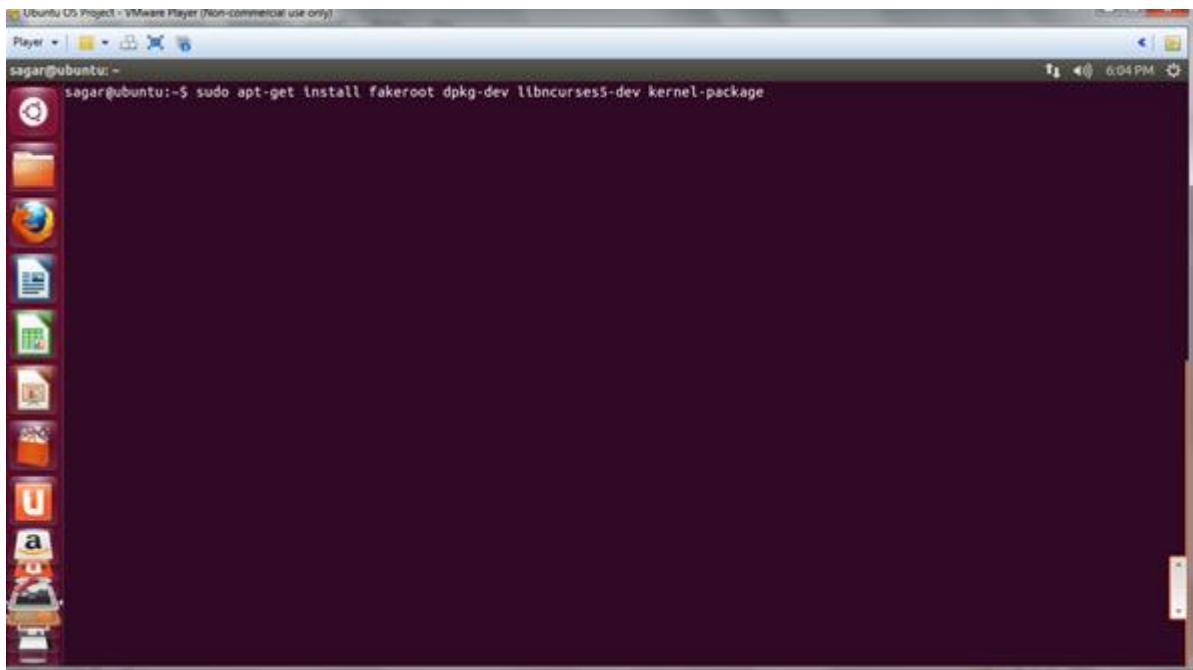
Now we need to install our custom kernel, inorder to carry our experimentations . We will first install the required packages before we actually start installing our custom Linux kernel.

Debian manages the kernel in the form of a package.To compile a Linux kernel the debian way,we need to include libncurses5-dev package, fakeroot package.

Fakeroot package will enable creation of the Debian package without using administrator's rights.

**Linux command used:**

```
sudo apt-get install fakeroot dpkg-dev libncurses5-dev kernel-package
```



### Output screen shots of the above command

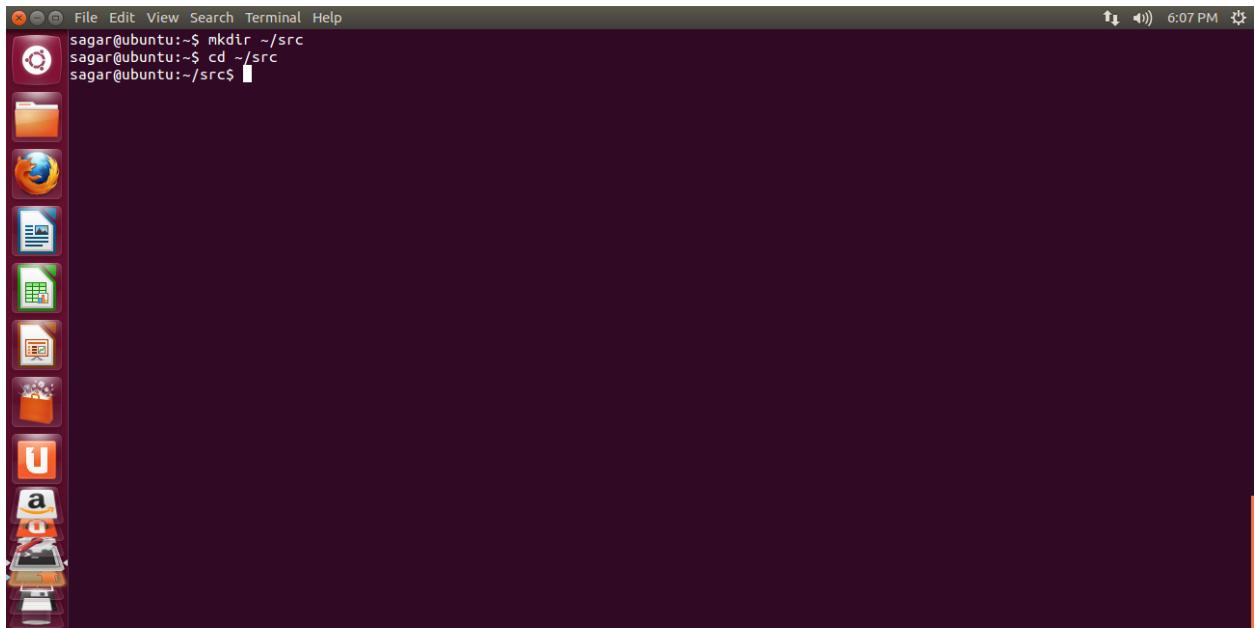
```
sagar@ubuntu:~$ sudo apt-get install fakeroot dpkg-dev libncurses5-dev kernel-package
[sudo] password for sagar:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  build-essential g++-4.7 libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libmail-sendmail-perl
  libstdc++6-4.7-dev libsys-hostname-long-perl libtinfo-dev po-debconf
Suggested packages:
  debian-keyring g++-multilib gcc-4.7-doc libstdc++6-4.7-dbg linux-source kernel-source libncurses-dev docbook-utils
  xlnto grub grub2 jfsutils mcelog oprofile reiserfsprogs squashfs-tools xfsprogs quota btrfs-tools ncurses-doc libstdc++6-4.7-doc
  libmail-box-perl
The following NEW packages will be installed:
  build-essential dpkg-dev fakeroot g++-4.7 kernel-package libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libmail-sendmail-perl libncurses5-dev libstdc++6-4.7-dev libsys-hostname-long-perl libtinfo-dev po-debconf
0 upgraded, 15 newly installed, 0 to remove and 158 not upgraded.
Need to get 11.2 MB of archives.
After this operation, 32.1 MB of additional disk space will be used.
Do you want to continue [Y/n]? ■
```

```
Unpacking libalgorithm-diff-perl (from .../libalgorithm-diff-perl_1.19.02-2_all.deb) ...
Selecting previously unselected package libalgorithm-diff-xs-perl.
Unpacking libalgorithm-diff-xs-perl (from .../libalgorithm-diff-xs-perl_0.04-2build3_i386.deb) ...
Selecting previously unselected package libalgorithm-merge-perl.
Unpacking libalgorithm-merge-perl (from .../libalgorithm-merge-perl_0.08-2_all.deb) ...
Selecting previously unselected package libsys-hostname-long-perl.
Unpacking libsys-hostname-long-perl (from .../libsys-hostname-long-perl_1.4-2_all.deb) ...
Selecting previously unselected package libmail-sendmail-perl.
Unpacking libmail-sendmail-perl (from .../libmail-sendmail-perl_0.79.16-1_all.deb) ...
Selecting previously unselected package libtinfo-dev:i386.
Unpacking libtinfo-dev:i386 (from .../libtinfo-dev_5.9-10ubuntu1_i386.deb) ...
Selecting previously unselected package libncurses5-dev.
Unpacking libncurses5-dev (from .../libncurses5-dev_5.9-10ubuntu1_i386.deb) ...
Processing triggers for man-db ...
Processing triggers for doc-base ...
Processing 33 changed doc-base files, 1 added doc-base file...
Setting up dpkg-dev (1.16.7ubuntu8) ...
Setting up fakeroot (1.18.4-2) ...
update-alternatives: using /usr/bin/fakeroot-sysv to provide /usr/bin/fakeroot (fakeroot) in auto mode
Setting up po-debconf (3.0.16+nmu2ubuntu1) ...
Setting up libalgorithm-diff-perl (1.19.02-2) ...
Setting up libalgorithm-diff-xs-perl (0.04-2build3) ...
Setting up libalgorithm-merge-perl (0.08-2) ...
Setting up libsys-hostname-long-perl (1.4-2) ...
Setting up libmail-sendmail-perl (0.79.16-1) ...
Setting up libtinfo-dev:i386 (5.9-10ubuntu1) ...
Setting up libncurses5-dev (5.9-10ubuntu1) ...
Setting up libstdc++6-4.7-dev (4.7.2-2ubuntu1) ...
Setting up g++-4.7 (4.7.2-2ubuntu1) ...
Setting up g++ (4:4.7.2-3ubuntu2) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (11.1ubuntu3) ...
Setting up kernel-package (3.2.0-3+nmu3) ...
sagar@ubuntu:~$
```

Now we need to create the source directory where we will store of kernel source code.

**Linux Command:**

```
mkdir ~/src  
cd ~/src
```

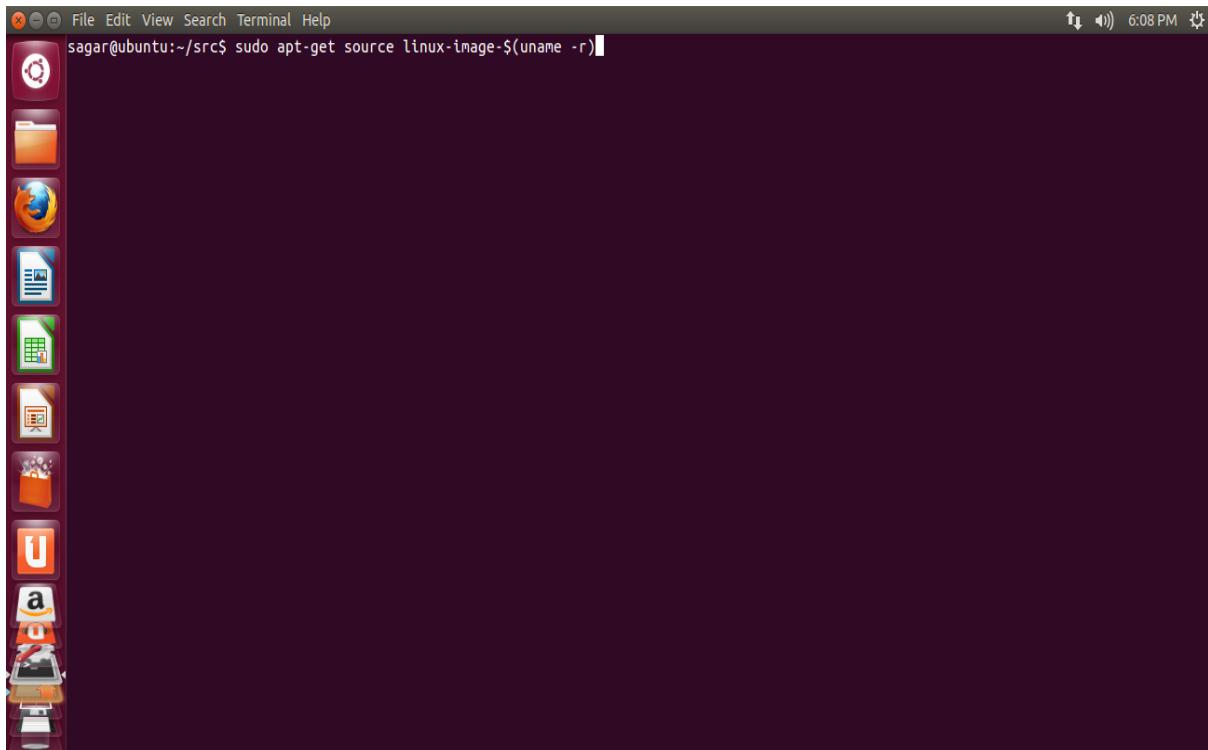


10.

Now we need to download the kernel source code , of the same version which our Ubuntu 12.10 is using currently.

**Linux Command used**

```
sudo apt-get source linux-image-$(uname -r)
```



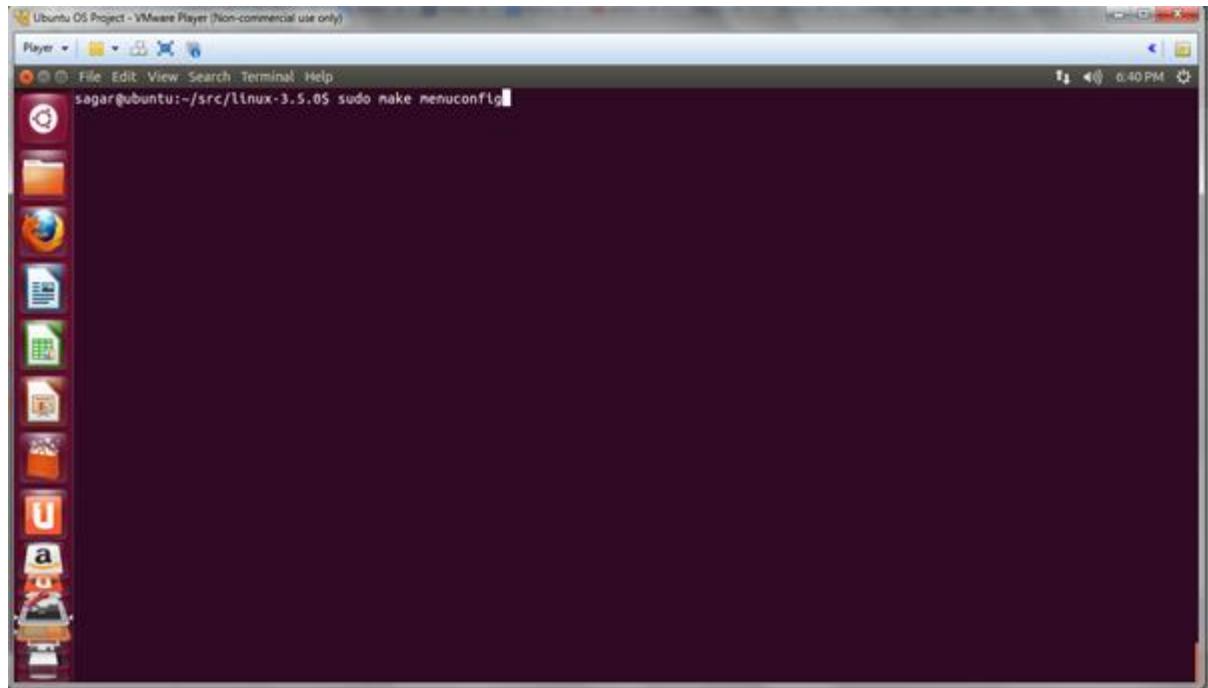
### Command Output Screen Shot:

```
Ubuntu OS Project - VMware Player (Non-commercial use only)
File Edit View Search Terminal Help
Player | 0:12 PM
File Edit View Search Terminal Help
linux-3.5.0/ubuntu/aufs/sblkinfo.c
linux-3.5.0/ubuntu/aufs/spl.h
linux-3.5.0/ubuntu/aufs/super.c
linux-3.5.0/ubuntu/aufs/super.h
linux-3.5.0/ubuntu/aufs/sysaufs.c
linux-3.5.0/ubuntu/aufs/sysaufs.h
linux-3.5.0/ubuntu/aufs/sysfs.c
linux-3.5.0/ubuntu/aufs/sysrq.c
linux-3.5.0/ubuntu/aufs/vdirc.c
linux-3.5.0/ubuntu/aufs/vfsub.c
linux-3.5.0/ubuntu/aufs/vfsub.h
linux-3.5.0/ubuntu/aufs/wbr_policy.c
linux-3.5.0/ubuntu/aufs/whout.c
linux-3.5.0/ubuntu/aufs/whout.h
linux-3.5.0/ubuntu/aufs/wkq.c
linux-3.5.0/ubuntu/aufs/wkq.h
linux-3.5.0/ubuntu/aufs/xino.c
linux-3.5.0/ubuntu/dm-raid4-5/BOM
linux-3.5.0/ubuntu/dm-raid4-5/Kconfig
linux-3.5.0/ubuntu/dm-raid4-5/Makefile
linux-3.5.0/ubuntu/dm-raid4-5/dm-memcache.c
linux-3.5.0/ubuntu/dm-raid4-5/dm-memcache.h
linux-3.5.0/ubuntu/dm-raid4-5/dm-message.c
linux-3.5.0/ubuntu/dm-raid4-5/dm-message.h
linux-3.5.0/ubuntu/dm-raid4-5/dm-raid4-5.c
linux-3.5.0/ubuntu/dm-raid4-5/dm-raid4-5.h
linux-3.5.0/ubuntu/dm-raid4-5/dm-raid4-5.h
linux-3.5.0/ubuntu/dm-raid4-5/dm-region-hash.c
linux-3.5.0/ubuntu/dm-raid4-5/dm-region-hash.h
linux-3.5.0/ubuntu/include/kbuild
linux-3.5.0/ubuntu/include/README
linux-3.5.0/ubuntu/include/linux/kbuild
linux-3.5.0/ubuntu/include/linux/aufs_type.h
sagar@ubuntu:~/src$
```

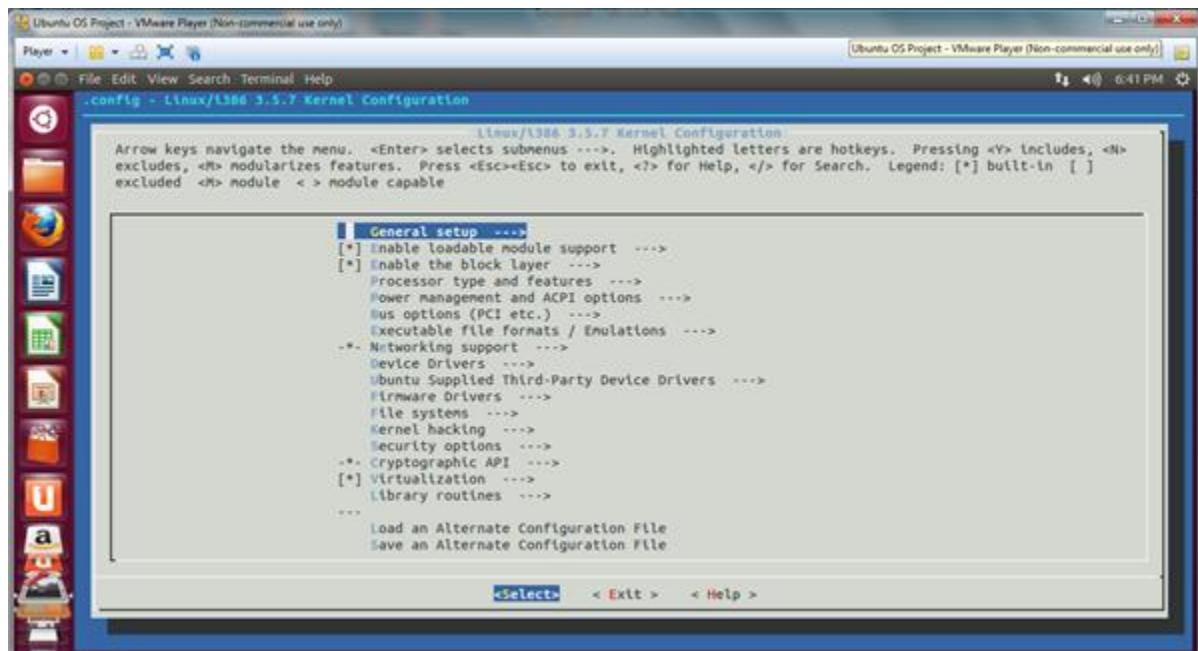
11. Now the next step is to show you, how to do the kernel configurations before the kernel compilation. In this menu you can specify which modules or specific area you want to compile. We will stick with general configurations.

### **Linux Command Used:**

sudo make menuconfig



### **Output Screen Shots**



We will move to Exit in the above menu screen and Exit

12. To speed up the compilation process, we can shoot a command.

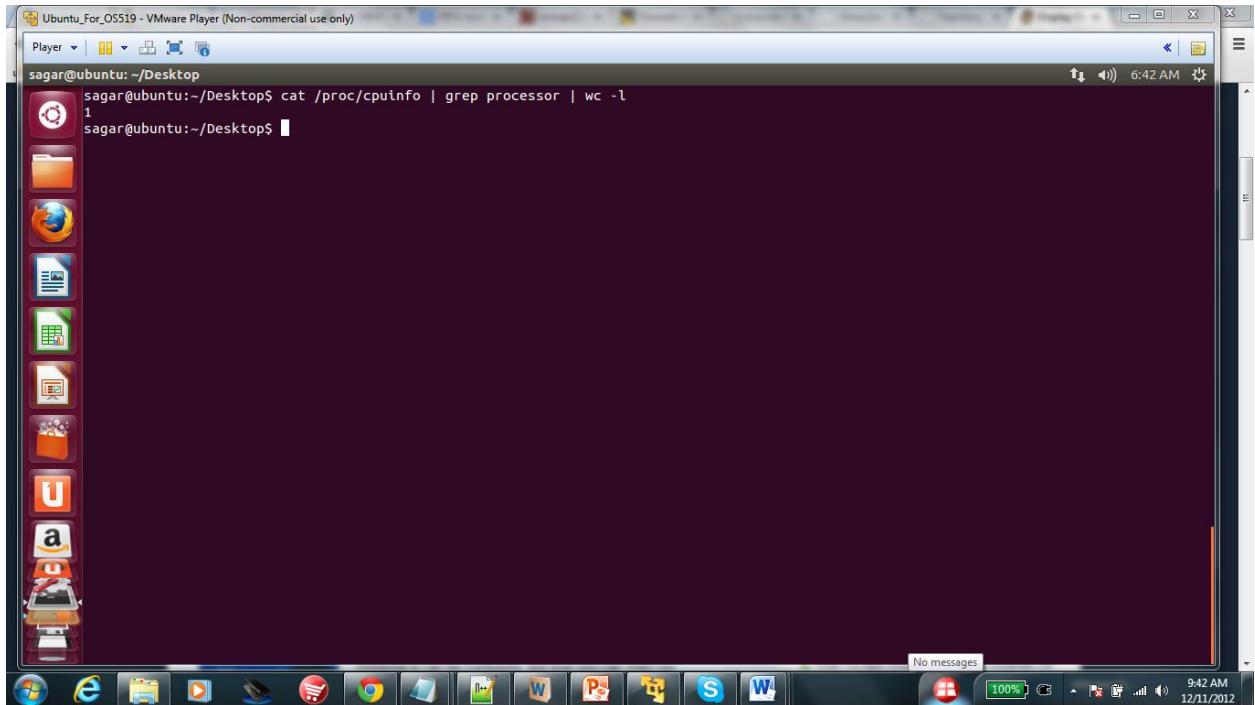
### **Linux Command used**

```
export CONCURRENCY_LEVEL=2
```

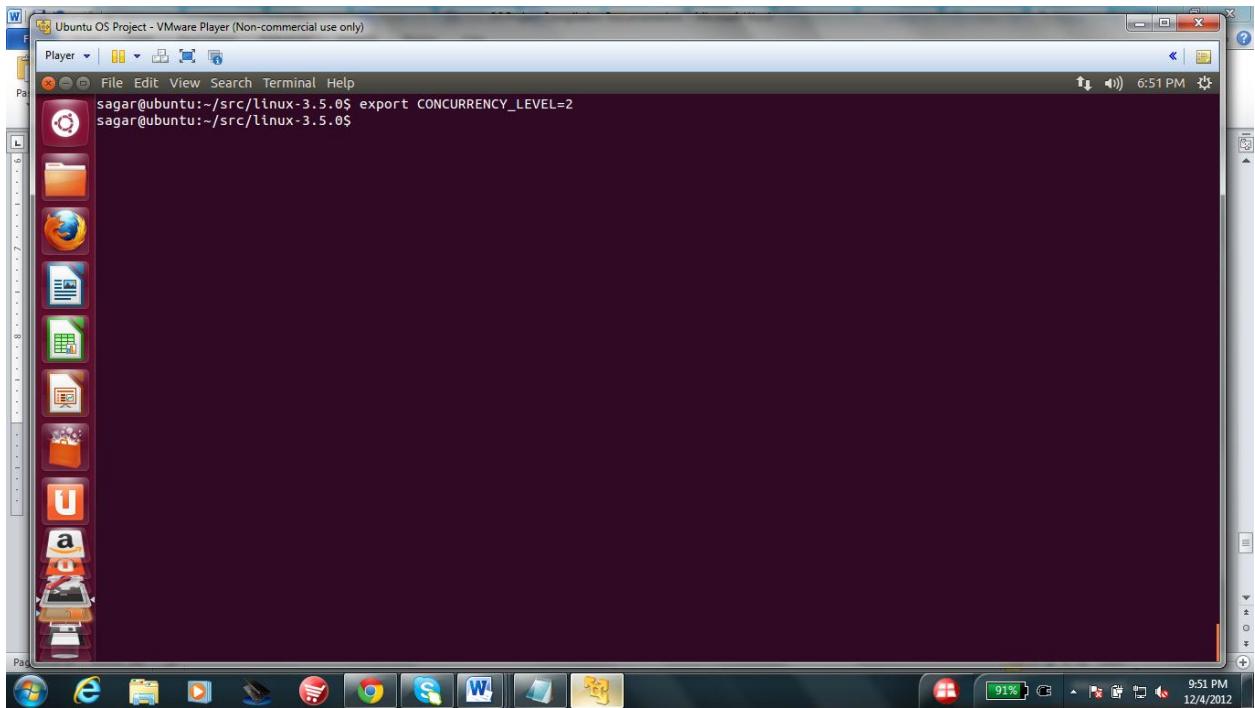
The above command is used depending upon the number of the CPU's your computer is using. The general formula to come up with the number like shown above is – number of CPU's + 1. In our case, we have only one cpu, so we set it to 2.

To verify the number of processors in your computer use the following command

```
cat /proc/cpuinfo | grep processor | wc -l
```



Speed up Output Screen shot



13.

The next step is to clean the temporary files generated if earlier some kernel compilation was done.

### **Linux command used:**

```
sudo make-kpkg clean
```

```
sagar@ubuntu:~/src/linux-3.5.0
[sudo] password for sagar:
exec make-kpkg_version=12.036+nmu3 -f /usr/share/kernel-package/ruleset/minimal.mk clean
===== making target minimal_clean [new prereqs: ]=====
This is kernel package version 12.036+nmu3.
test ! -f .config || cp -pf .config config.precious
test ! -e stamp-building || rm -f stamp-building
test ! -f Makefile || \
    make ARCH=i386 distclean
make[1]: Entering directory `/home/sagar/src/linux-3.5.0'
CLEAN scripts/basic
CLEAN scripts/kconfig
CLEAN include/config include/generated
CLEAN .config
make[1]: Leaving directory `/home/sagar/src/linux-3.5.0'
test ! -f config.precious || mv -f config.precious .config
rm -f modules/modversions.h modules/ksyms.ver scripts/cramfs/cramfsck scripts/cramfs/mkcramfs
sagar@ubuntu:~/src/linux-3.5.0$
```

14.

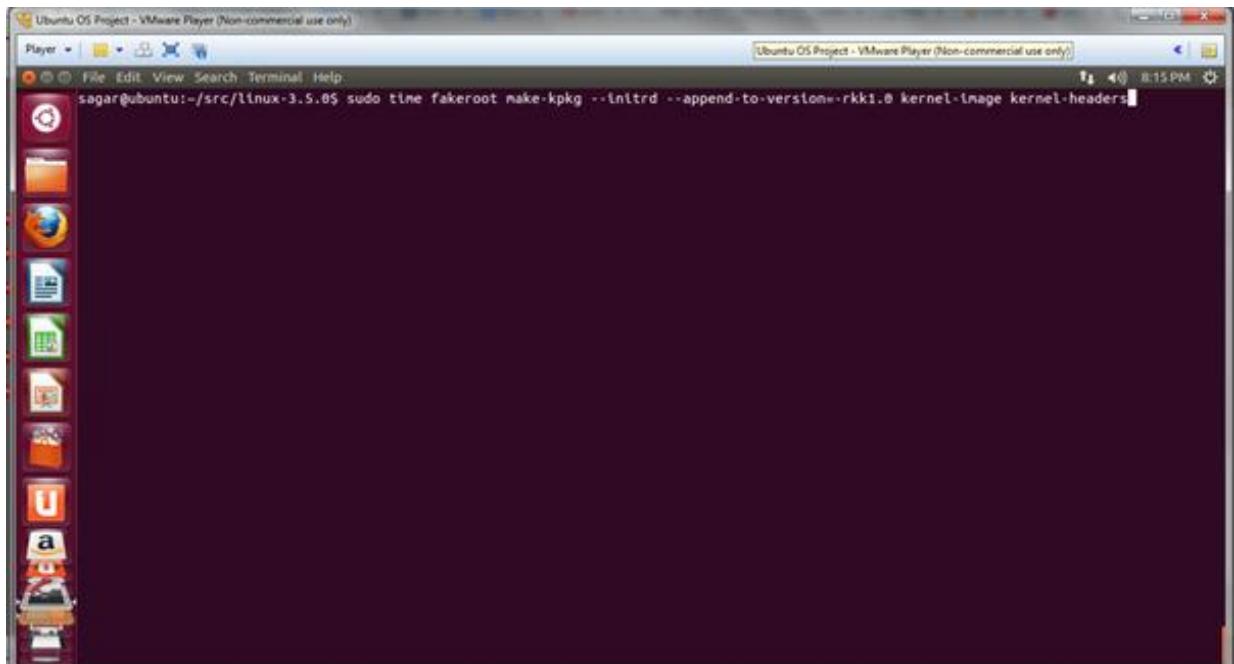
The next step is to compile the kernel. We use make command to compile the kernel. make command runs the Makefile in the src directory which has already configurations described in it.

#### **Linux command used in our case is:**

```
sudo time fakeroot make-kpkg --initrd --append-to-version=-rkk1.0 kernel-image kernel-headers
```

This command will generate the compiled image and header files appended version number with rkk1.0 our custom name to distinguish it from our previous kernel version.

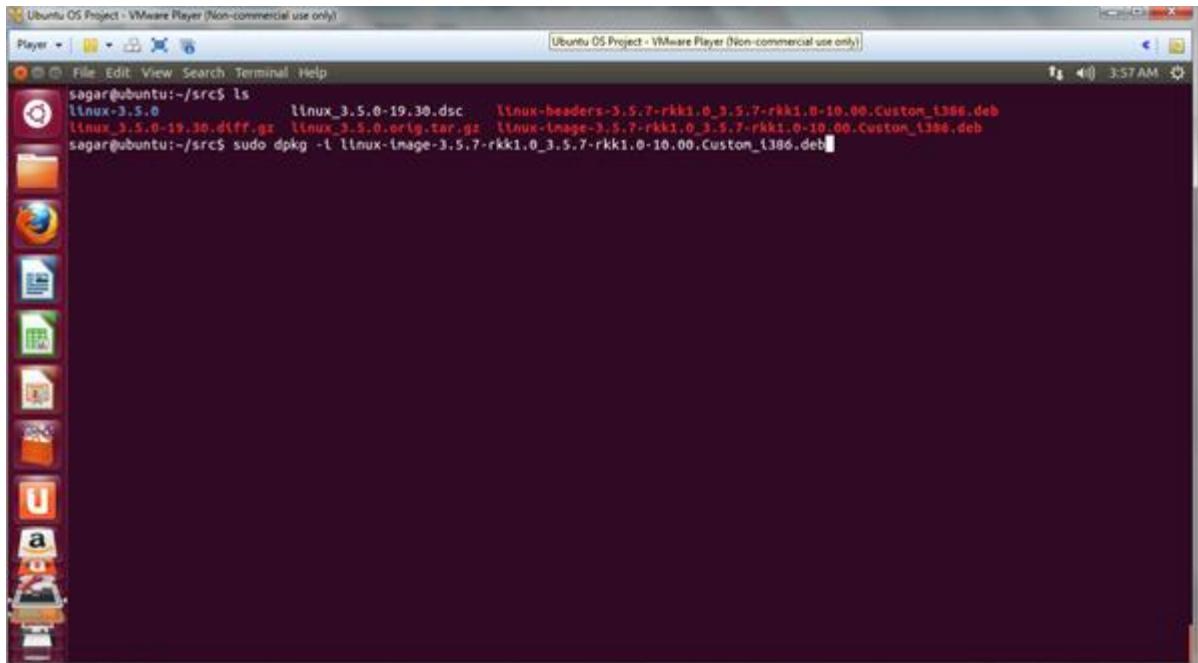
This is the most time consuming phase in the compilation process of the Linux kernel. For us it took almost 10 – 11 hours to compile the entire kernel. Later when we increased the RAM to 4 GB , the compilation time consumption reduced to 6 hours and again we increased the RAM of virtual Machine to 6 GB. The compilation time reduced to 4 GB's



## **Output screen of this command**

```
Ubuntu OS Project - VMware Player (Non-commercial use only)
Player File Edit View Search Terminal Help
  • e 's=/linux/g' -e 's=/R/g' \
  • e 's=/KPV/12.036+nmu3/g' \
  • e 's=/X/vmlinuz/g' \
  • e 's=/I/YES/g' -e 's,=0,boot,g' \
  • e 's/P/linux-headers-3.5.7-rkk1.0/g' \
  • e 's@=A@l386@g' \
  • e 'se=B@x86@g' \
    ./debian/pkg/headers/postinst >      /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0/DEBIAN/postinst
chmod 755 /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0/DEBIAN/postinst
cp -pf debian/control debian/control.dist
k='find /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0 -type f | { while read t; do
  if file -b $t | egrep -q "ELF.*executable.*dynamically linked"; then \
    j=$t $t";
    fi;
    done; echo $j; }'; test -z "$k" || dpkg-shlibdeps $k;
  echo "Elf Files: $k" >      /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0/usr/share/doc/linux-headers-3.5.7-rkk1.0/elffiles;
  test -n "$k" || perl -pe 's/$(\$libs:Depends)\|,?//g' debian/control
test ! -e debian/control- || rm -f debian/control-
dpkg-gencontrol -isp -DArchitecture=i386 -plinux-headers-3.5.7-rkk1.0 \
  -P/home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0/
create_md5sums_fn () { cd $1 ; find . -type f ! -regex '/DEBIAN/*' ! -regex '/var/*' -printf '%P\n' | xargs -r md5sum > DEBIAN/md5sums ; if [ -z "DEBIAN/md5sums" ] ; then rm -f DEBIAN/md5sums ; fi ; }; create_md5sums_fn /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0
chown -R root:root /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0
chmod -R ogxr /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0
dpkg --build /home/sagar/src/linux-3.5.0/debian/linux-headers-3.5.7-rkk1.0 ..
dpkg-deb: building package 'linux-headers-3.5.7-rkk1.0' in '../linux-headers-3.5.7-rkk1.0_3.5.7-rkk1.0-10.00.Custom_i386.deb'.
cp -pf debian/control.dist debian/control
make[2]: Leaving directory '/home/sagar/src/linux-3.5.0'
make[1]: Leaving directory '/home/sagar/src/linux-3.5.0'
1249.76user 5643.30system 7:36:47elapsed 25NCPU (0avgtext+0avgdata 458356maxresident)k
2846560inputs+23643504outputs (1122major+183696327minor)pagefaults 0swaps
sagar@ubuntu:~/src/linux-3.5.0$
```

If you see your src directory, you will see that the two compiled files got created , one is image file and another one is header file. The below screen shot mentions that.



16.

Now we have got the compiled files, now we need to install the kernel. We will install both the .deb files step by step

#### **Linux command used:**

```
cd ~/src
sudo dpkg -i linux-image-3.0.36-rkk1.0_3.0.36-rkk1.0-10.00.Custom_i386.deb
sudo dpkg -i linux-headers-3.0.36-rkk1.0_3.0.36-rkk1.0-10.00.Custom_i386.deb
```

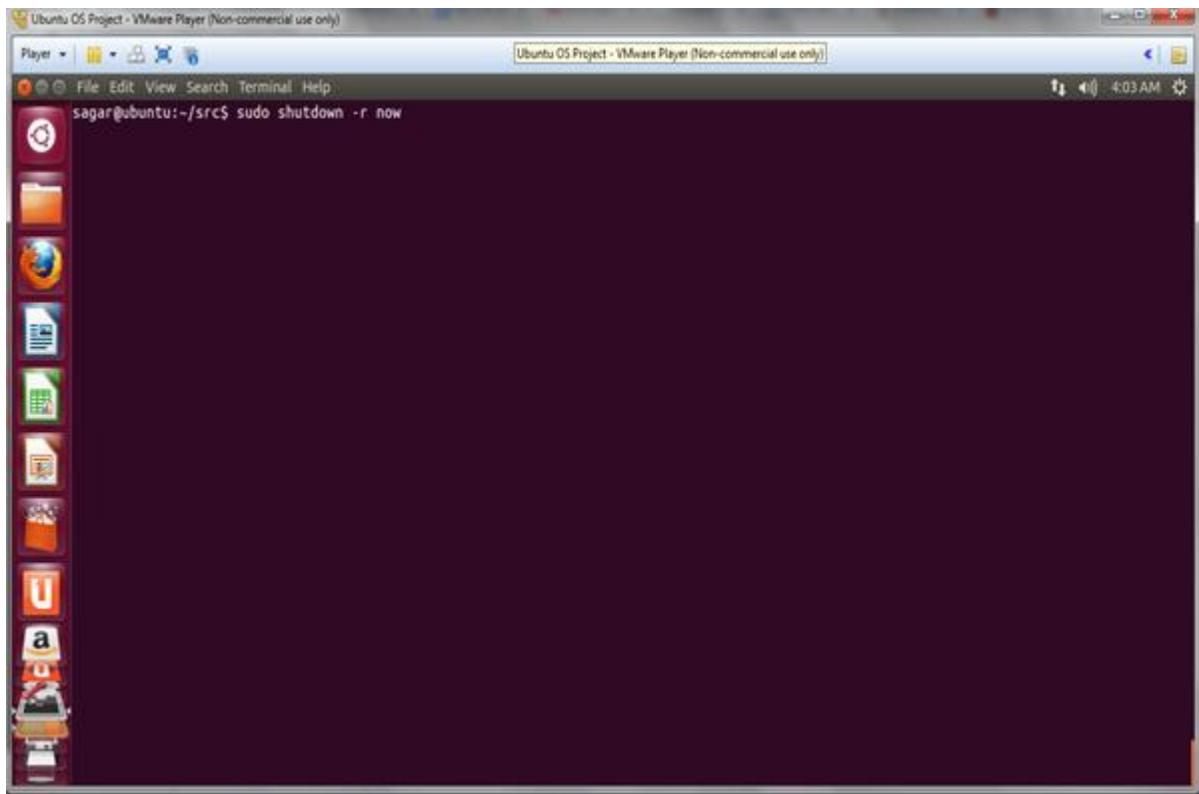
-i option tells to install the files set up in options.

17.

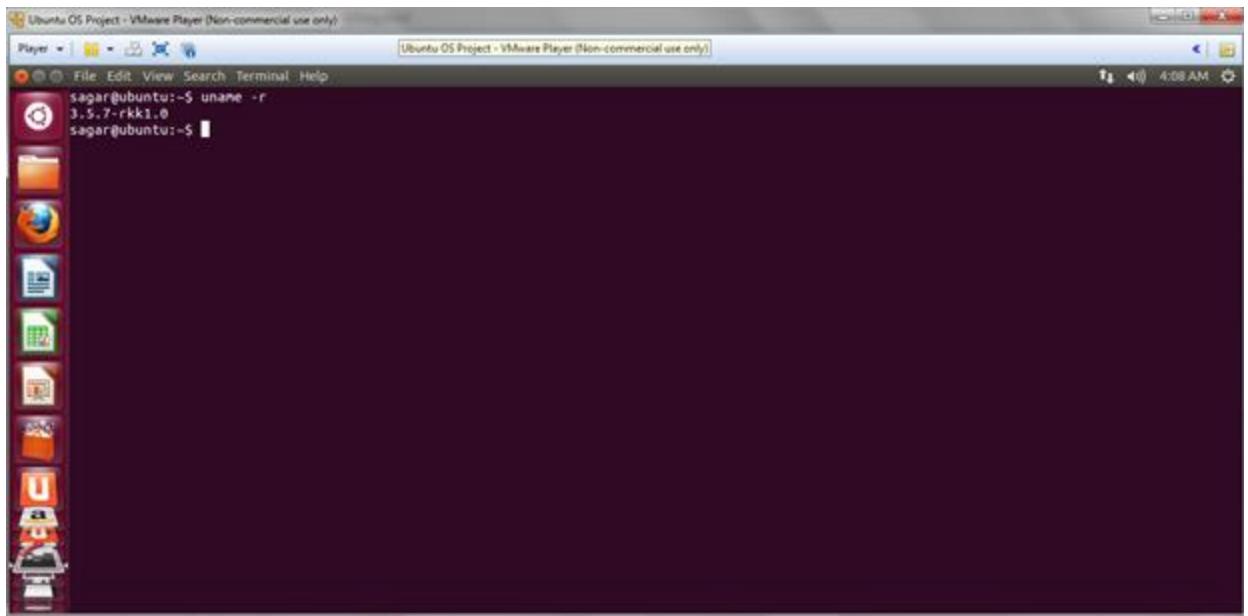
This is last step in Compilation and Installation process, which is to shutdown and restart the Ubuntu after the installation is complete.

#### **Linux command used:**

```
sudo shutdown -r now
```



Let's check or verify whether the new kernel has got installed. We will use the same uname –r command to see the version of our new kernel



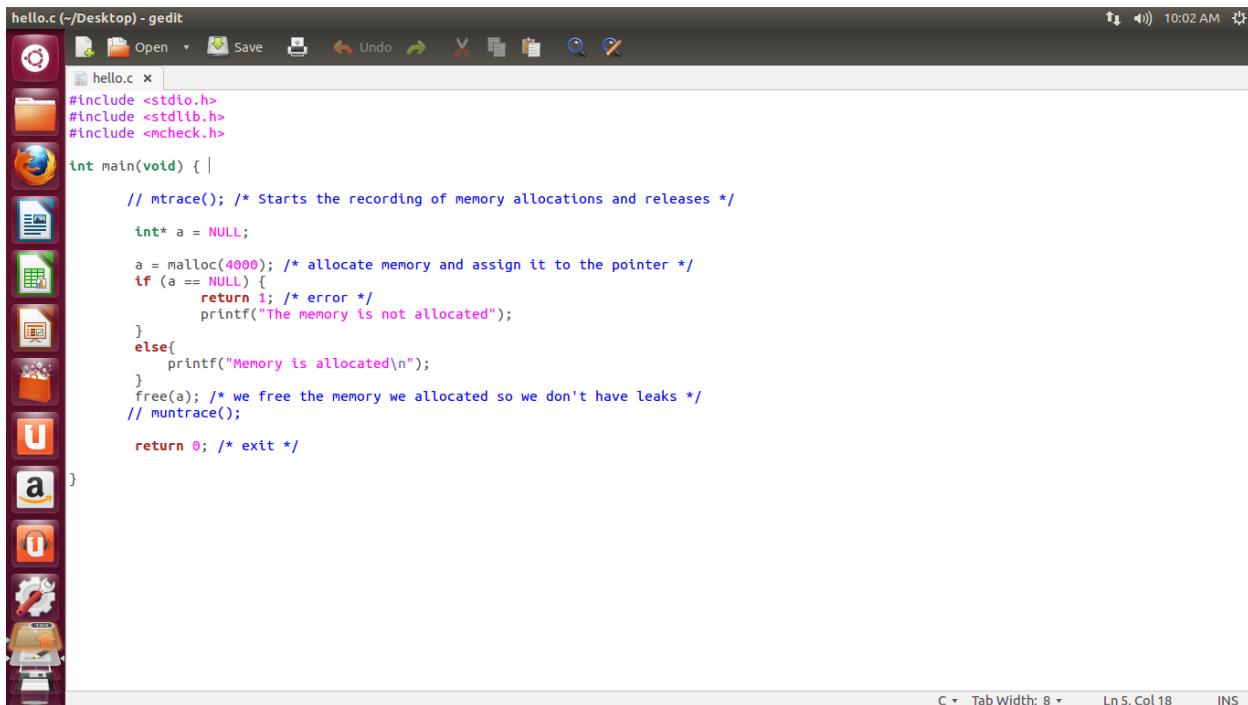
The new kernel has got installed in our system. The kernel version name is 3.5.7-rkk1.0

The Kernel compilation and installation process is complete.

## Understanding the malloc() implementation in Linux operating system

malloc() is userspace dynamic memory allocation function for the processes. In linux kernel, it has its own version of malloc() called as kmalloc(). For this project we were only interested in knowing the working of malloc(), which is the userspace function.

To understand the entire process, we created a small user application program hello.c in C language which uses malloc() function.



```

hello.c (~/Desktop) - gedit
  Open Save Undo Redo Find Replace Cut Copy Paste Select All Find in Files
hello.c x
#include <stdio.h>
#include <stdlib.h>
#include <mcheck.h>

int main(void) {
    // mtrace(); /* Starts the recording of memory allocations and releases */
    int* a = NULL;

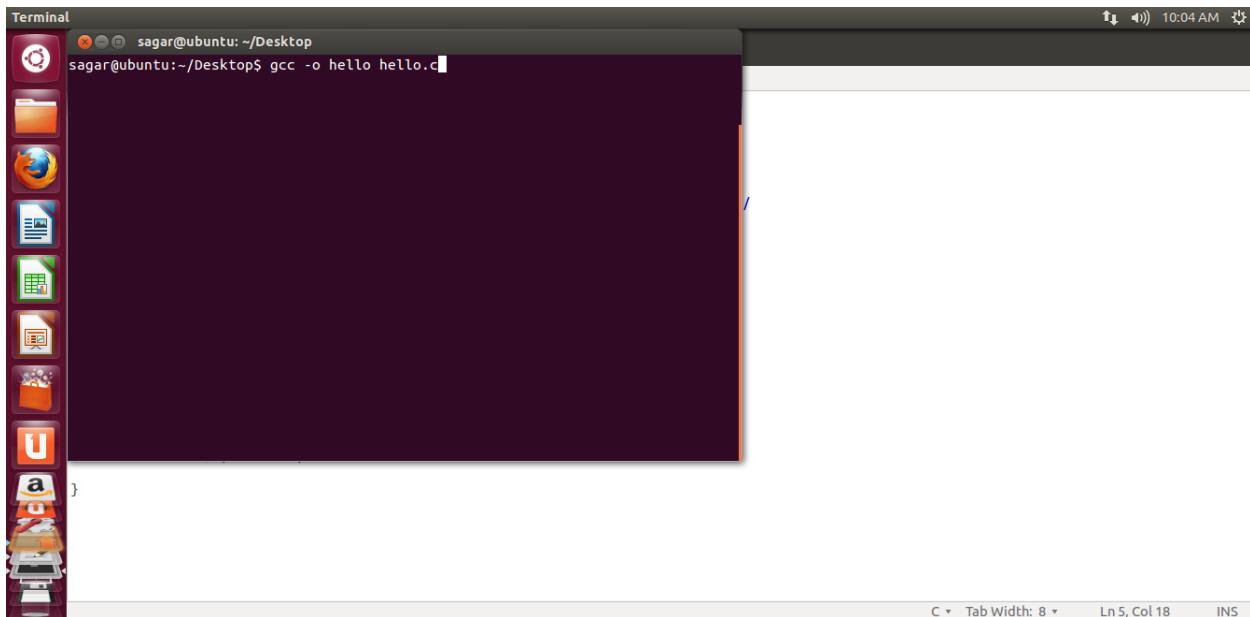
    a = malloc(4000); /* allocate memory and assign it to the pointer */
    if (a == NULL) {
        return 1; /* error */
        printf("The memory is not allocated");
    }
    else{
        printf("Memory is allocated\n");
    }
    free(a); /* we free the memory we allocated so we don't have leaks */
    // muntrace();

    return 0; /* exit */
}
  
```

The above program wants to assign the memory of 4000 bytes using malloc() function. What ever returned from the malloc() function is a starting address of the memory allocated and stored in a pointer a.

We first need to compile this program. The command used to compile this program is

```
gcc -o hello hello.c
```

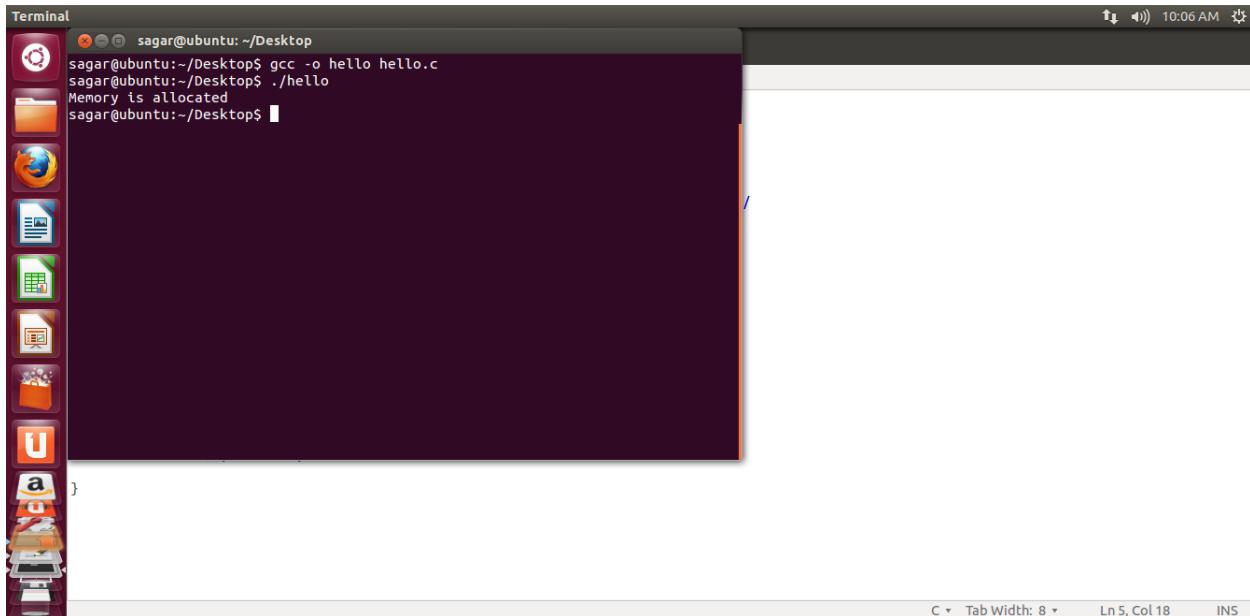


A screenshot of a terminal window titled "Terminal". The window shows a command-line session. The user is at the prompt "sagar@ubuntu:~/Desktop\$". They type "gcc -o hello hello.c" and press Enter. The terminal then displays the output of the program, which is "Memory is allocated". The terminal has a dark purple background and a light gray border. The status bar at the bottom right shows "C Tab Width: 8 Ln 5, Col 18 INS". To the left of the terminal is a vertical dock containing icons for various applications like a file manager, browser, and terminal.

Now let's run this program

Command used is :

`./hello`



A screenshot of a terminal window titled "Terminal". The window shows a command-line session. The user is at the prompt "sagar@ubuntu:~/Desktop\$". They type "gcc -o hello hello.c" and press Enter. Then they type "./hello" and press Enter. The terminal displays the output of the program, which is "Memory is allocated". The terminal has a dark purple background and a light gray border. The status bar at the bottom right shows "C Tab Width: 8 Ln 5, Col 18 INS". To the left of the terminal is a vertical dock containing icons for various applications like a file manager, browser, and terminal.

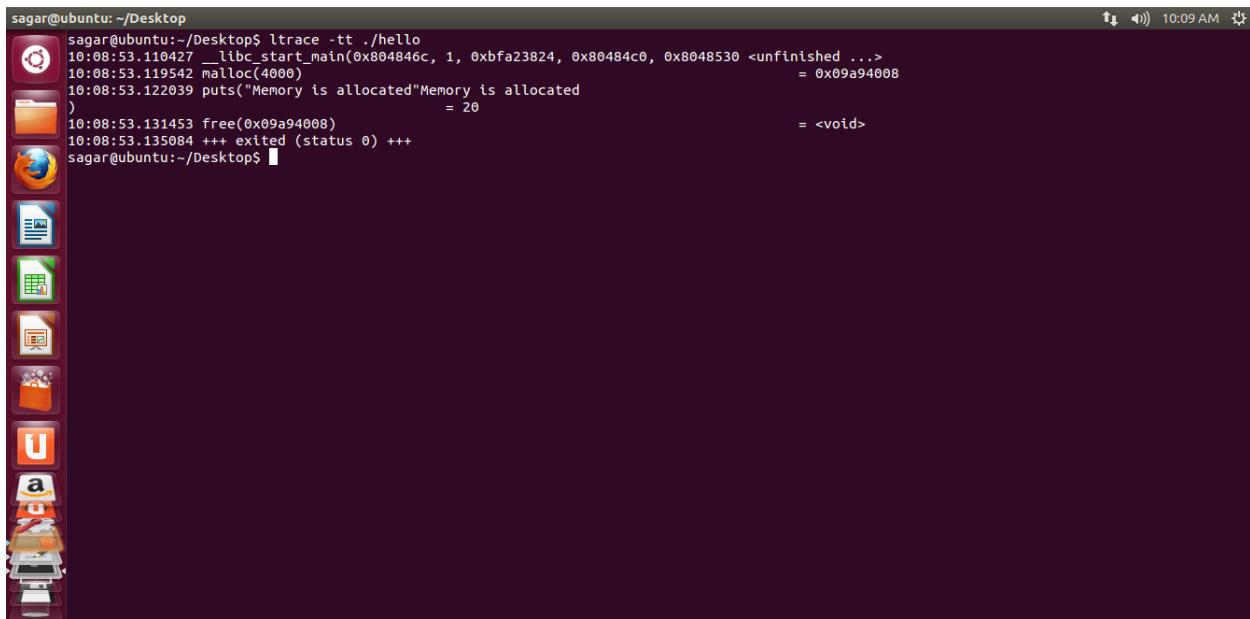
Based on the output the memory of 4000 bytes have been allocated.

We will now try to trace this program using either ltrace command or strace command to see which library functions or system calls are involved while running this hello.c application program.

ltrace command in linux traces all the library functions used by the file specified. When given an option of -S in ltrace command, it traces all the library functions and System calls used while running the hello.c

### Linux command

ltrace ./hello



```
sagar@ubuntu:~/Desktop$ ltrace -tt ./hello
10:08:53.110427 __libc_start_main(0x804846c, 1, 0xbfa23824, 0x80484c0, 0x8048530 <unfinished ...>
10:08:53.119542 malloc(4000) = 0x09a94008
10:08:53.122039 puts("Memory is allocated"Memory is allocated
) = 20
10:08:53.131453 free(0x09a94008) = <void>
10:08:53.135084 +++ exited (status 0) +++
sagar@ubuntu:~/Desktop$
```

Seeing the output of the above command, we can see that execution accesses the `__libc_start_main()` shared library function. Then it executes `malloc()` function. From here we got the clue that, `malloc()` is defined in glibc shared libraries .

Now we will try to see which system calls are being used while running the hello.c program

**Command used:** ltrace -S -t ./hello

```

Ubuntu_For_OSS19 - VMware Player (Non-commercial use only)
Player [ ] < | > 8:07 AM
sagar@ubuntu: ~/Desktop$ 
08:07:02.904556 SYS_brk(NULL) = 0x08796000
08:07:02.906640 SYS_access("/etc/ld.so.nohwcap", 00) = -2
08:07:02.908844 SYS_mmap2(0, 8192, 3, 34, -1) = 0xb7787000
08:07:02.910053 SYS_access("/etc/ld.so.preload", 04) = -2
08:07:02.912215 SYS_open("/etc/ld.so.cache", 524288, 00) = 3
08:07:02.914425 SYS_fstatat64(3, 0xbfb23ab0, 0xb77ab000, 0xb77ab8bc, 3) = 0
08:07:02.915781 SYS_mmap2(0, 66157, 1, 2, 3) = 0xb7776000
08:07:02.916454 SYS_close(3) = 0
08:07:02.917373 SYS_access("/etc/ld.so.nohwcap", 00) = -2
08:07:02.918652 SYS_open("/lib/i386-linux-gnu/libc.so.6", 524288, 0160045) = 3
08:07:02.920262 SYS_read(3, "\177ELF\001\001\001", 512) = 512
08:07:02.921274 SYS_fstatat64(3, 0xbfb23b10, 0xb77ab000, 0x804824c, 0xb77abb20) = 0
08:07:02.922271 SYS_mmap2(0, 0x1a9adc, 5, 2050, 3) = 0xb75cc000
08:07:02.923076 SYS_mprotect(0xb776f000, 4096, 0) = 0
08:07:02.923725 SYS_mmap2(0xb7770000, 12288, 3, 2066, 3) = 0xb7770000
08:07:02.924517 SYS_mmap2(0xb7773000, 10972, 3, 50, -1) = 0xb7773000
08:07:02.925392 SYS_close(3) = 0
08:07:02.926011 SYS_mmap2(0, 4096, 3, 34, -1) = 0xb75cb000
08:07:02.927128 SYS_set_thread_area(0xbfb23fe0, 0xb77ab000, 0xb75cb900, 1, 0) = 0
08:07:02.928270 SYS_mprotect(0xb7770000, 8192, 1) = 0
08:07:02.929534 SYS_mprotect(0x08049000, 4096, 1) = 0
08:07:02.930288 SYS_mprotect(0xb77aa000, 4096, 1) = 0
08:07:02.936605 SYS_munmap(0xb776f000, 66157) = 0
08:07:02.936998 __libc_start_main(0x0804846c, 1, 0xbfb24214, 0x80484c0, 0x8048530 <unfinished ...>
08:07:02.937930 malloc(4000 <unfinished ...>
08:07:02.938499 SYS_brk(NULL) = 0x08796000
08:07:02.938791 SYS_brk(0x087b7000) = 0x087b7000
08:07:02.939155 <... malloc resumed: > = 0x08796008
08:07:02.939636 puts("Memory is allocated" <unfinished ...>
08:07:02.941047 SYS_fstatat64(1, 0xbfb24000, 0xb7772000, 0xb7772a20, 8192) = 0
08:07:02.941785 SYS_mmap2(0, 4096, 3, 34, -1) = 0xb7786000
08:07:02.942594 SYS_write(1, "Memory is allocated\n", 20Memory is allocated
) = 20
08:07:02.949003 <... puts resumed: > = 20
08:07:02.949324 free(0x08796008) = <void>

```

The hello.c file uses the system calls as mentioned above in the screen. Now we will go in source to see what these system calls are and what is their purpose.

### sys\_brk : Location: mm/mmap.c

sys\_brk() system call is gives the starting address of the new chunk or block of memory allocated to the process requested by the application from the user space.

Below is the screen shot of the source code

```

Ubuntu_For_OSS19 - VMware Player (Non-commercial use only)
Player [ ] < | > 8:21 AM
sagar@ubuntu: ~/src/linux-3.5.0/mm$ 
static unsigned long do_brk(unsigned long addr, unsigned long len);
SYSCALL_DEFINE1(brk, unsigned long, brk)
{
    unsigned long rlim, retval;
    unsigned long newbrk, oldbrk;
    struct mm_struct *mm = current->mm;
    unsigned long min_brk;
    printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
    down_write(&mm->mmap_sem);

#if defined CONFIG_COMPAT_BRK
    /*
     * CONFIG_COMPAT_BRK can still be overridden by setting
     * randomize_v_a_space to 2, which will still cause mm->start_brk
     * to be arbitrarily shifted
     */
    if (current->brk_randomized)
        min_brk = mm->start_brk;
    else
        min_brk = mm->end_data;
#endif
    printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
    min_brk = mm->start_brk;
#endif
    if (brk < min_brk)
        goto out;

    /*
     * Check against rlimit here. If this check is done later after the test
     * of oldbrk with newbrk then it can escape the test and let the data
     * segment grow beyond its set limit in case where the limit is
     * not page aligned -Ram Gupta

```

```

static unsigned long do_brk(unsigned long addr, unsigned long len);

SYSCALL_DEFINE1(brk, unsigned long, brk)
{
    unsigned long rlim, retval;
    unsigned long newbrk, oldbrk;
    struct mm_struct *mm = current->mm;
    unsigned long min_brk;
    printk(KERN_INFO "I am currently under sys call definition of sys brk\n");
    down_write(&mm->mmap_sem);

#endif CONFIG_COMPAT_BRK
/*
 * CONFIG_COMPAT_BRK can still be overridden by setting
 * randomize_va_space to 2, which will still cause mm->start_brk
 * to be arbitrarily shifted
 */
if (current->brk_randomized)
    min_brk = mm->start_brk;
else
    min_brk = mm->end_data;
#else
printk(KERN_INFO "I am currently under sys call definition of sys brk\n");
min_brk = mm->start_brk;
#endif
if (brk < min_brk)
    goto out;

/*
 * Check against rlimit here. If this check is done later after the test
 * of oldbrk with newbrk then it can escape the test and let the data
 * segment grow beyond its set limit the in case where the limit is
 * not page aligned -Ram Gupta
 */
rlim = rlimit(RLIMIT_DATA);
if (rlim < RLIM_INFINITY && (brk - mm->start_brk) +
    (mm->end_data - mm->start_data) > rlim)
    goto out;

newbrk = PAGE_ALIGN(brk);
oldbrk = PAGE_ALIGN(mm->brk);
if (oldbrk == newbrk)
    goto set_brk;

/* Always allow shrinking brk. */
if (brk <= mm->brk) {
    if (!do_munmap(mm, newbrk, oldbrk-newbrk))
        goto set_brk;
    goto out;
}

/* Check against existing mmap mappings. */
if (find_vma_intersection(mm, oldbrk, newbrk+PAGE_SIZE))
    goto out;

/* Ok, looks good - let it rip. */
if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
    goto out;
set_brk:
    mm->brk = brk;
out:
    retval = mm->brk;
    up_write(&mm->mmap_sem);
    printk(KERN_INFO "We are still in sys_brk system call definition under out section\n");
    printk(KERN_INFO "We are still in sys_brk system call definition under out section\n");
    return retval;
}

```

When the brk() is called, the system call checks to see if the amount of memory is decreased. If so, it immediately decreases the amount of heap space for that process with a call to *do\_munmap()* and returns the new address pointer.

If the application wants to increase the amount of heap space allocated to it, the kernel performs three checks to make sure the allocation can succeed. The first check verifies that the specified limit of memory for this process will not be exceeded if more memory is allocated. *setrlimit()* function sets the limit of the size of the memory in Linux for the processes.

After verifying that a process is not requiring more memory than its limit allows, the kernel then verifies that increasing the memory will not interfere with any of the other parts of memory for that process. For Example, it should not interfere with stack or mmaped files or text area of the virtual address space.

Finally, as a last check, the kernel checks to see if there is enough memory to accommodate the new process. Once kernel verifies it calls the *do\_brk()* function to increase the memory for the process.

### Editing sys brk() system to add printk statement

To verify if sys\_brk is accessed by the shared library functions, we added the printk statement in all the system calls we got from ltrace -S -tt ./hello command

The screen shots are as follows. 1. sys\_access in mm/mmap.c file

```

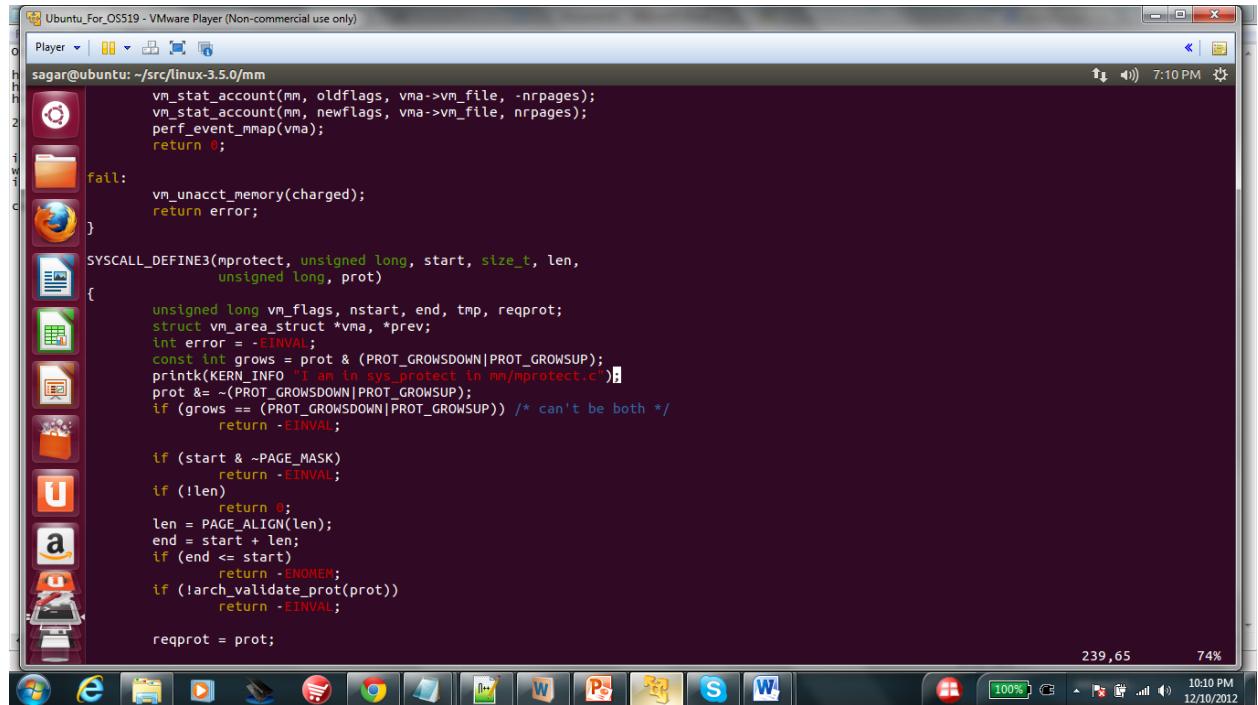
Ubuntu_For_OS519 - VMware Player (Non-commercial use only)
Player | 10:40 AM
sagar@ubuntu: ~/src/linux-3.5.0/mm
static unsigned long do_brk(unsigned long addr, unsigned long len);
SYSCALL_DEFINE1(brk, unsigned long, brk)
{
    unsigned long rlim, rval;
    unsigned long newbrk, oldbrk;
    struct mm_struct *mm = current->mm;
    unsigned long min_brk;
    printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
    down_write(&mm->mmap_sem);

#endif CONFIG_COMPAT_BRK
/*
 * CONFIG_COMPAT_BRK can still be overridden by setting
 * randomize_va_space to 2 which will still cause mm->start_brk
 * to be arbitrarily shifted
 */
if (current->brk.randomized)
    min_brk = mm->start_brk;
else
    min_brk = mm->end_data;
#else
printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
min_brk = mm->start_brk;
#endif
if (brk < min_brk)
    goto out;

/*
 * Check against rlimit here. If this check is done later after the test
 * of oldbrk with newbrk then it can escape the test and let the data
 * segment grow beyond its set limit in the case where the limit is
 * not page aligned -Ram Gupta
276,19-26 9%

```

## sys\_mprotect at mm/mprotect.c



```

sagar@ubuntu:~/src/linux-3.5.0/mm
    vm_stat_account(mm, oldflags, vma->vm_file, -nrpages);
    vm_stat_account(mm, newflags, vma->vm_file, nrpages);
    perf_event_mmap(vma);
    return 0;
}

fail:
    vm_unacct_memory(charged);
    return error;
}

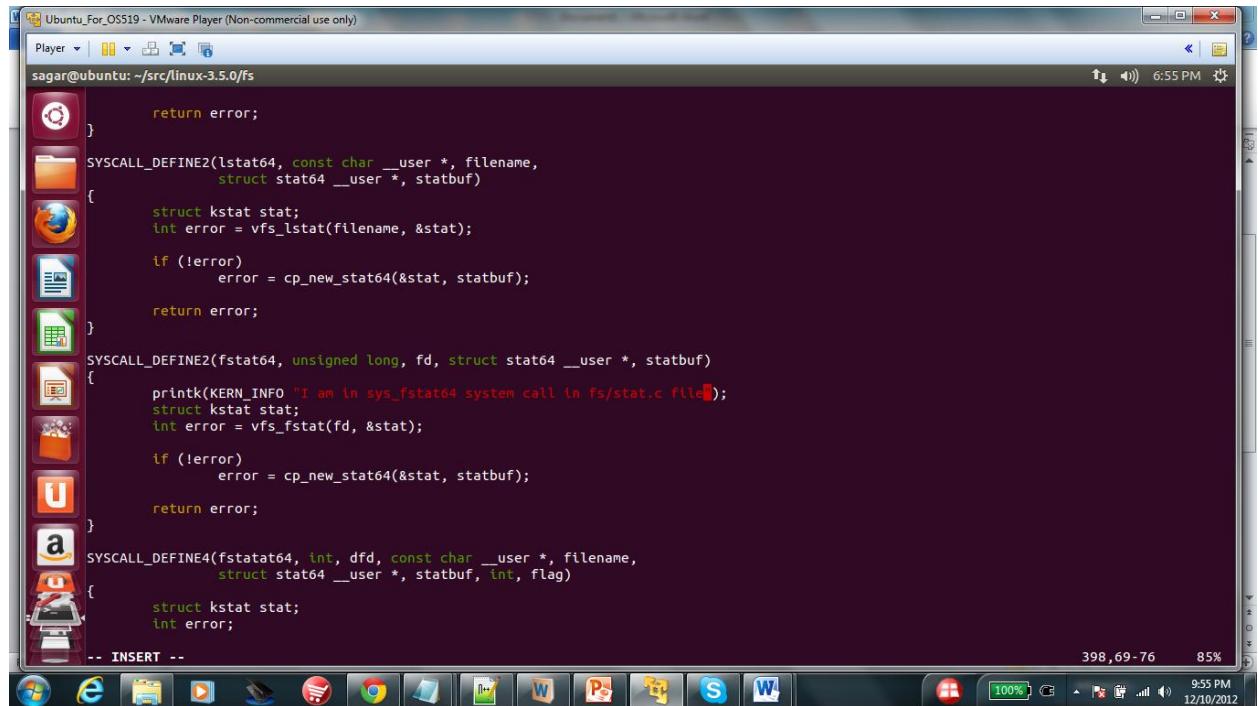
SYSCALL_DEFINE3(mprotect, unsigned long, start, size_t, len,
               unsigned long, prot)
{
    unsigned long vm_flags, nstart, end, tmp, reqprot;
    struct vm_area_struct *vma, *prev;
    int error = -EINVAL;
    const int grows = prot & (PROT_GROWSDOWN|PROT_GROWSUP);
    printk(KERN_INFO "I am in sys_protect in mm/mprotect.c");
    prot &= ~(PROT_GROWSDOWN|PROT_GROWSUP);
    if (grows == (PROT_GROWSDOWN|PROT_GROWSUP)) /* can't be both */
        return -EINVAL;

    if (start & ~PAGE_MASK)
        return -EINVAL;
    if (!len)
        return 0;
    len = PAGE_ALIGN(len);
    end = start + len;
    if (end <= start)
        return -ENOMEM;
    if (!arch_validate_prot(prot))
        return -EINVAL;

    reqprot = prot;
}

```

## sys\_fstat64 at location fs/stat.c



```

sagar@ubuntu:~/src/linux-3.5.0/fs
    return error;
}

SYSCALL_DEFINE2(lstat64, const char __user *, filename,
               struct stat64 __user *, statbuf)
{
    struct kstat stat;
    int error = vfs_lstat(filename, &stat);

    if (!error)
        error = cp_new_stat64(&stat, statbuf);

    return error;
}

SYSCALL_DEFINE2(fstat64, unsigned long, fd, struct stat64 __user *, statbuf)
{
    printk(KERN_INFO "I am in sys_fstat64 system call in fs/stat.c file");
    struct kstat stat;
    int error = vfs_fstat(fd, &stat);

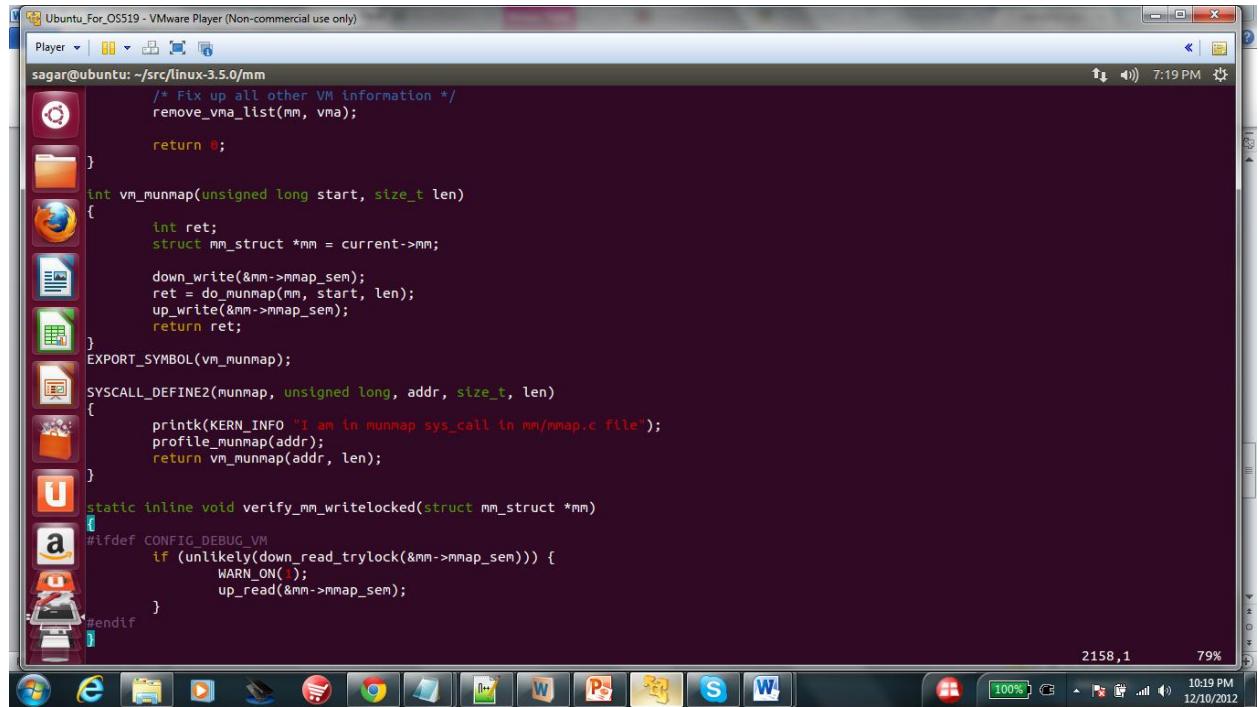
    if (!error)
        error = cp_new_stat64(&stat, statbuf);

    return error;
}

SYSCALL_DEFINE4(fstatat64, int, dfd, const char __user *, filename,
               struct stat64 __user *, statbuf, int, flag)
{
    struct kstat stat;
    int error;

```

### sys\_munmap at mm/mmap.c



```

sagar@ubuntu:~/src/linux-3.5.0/mm
    /* Fix up all other VM information */
    remove_vma_list(mm, vma);

    return 0;
}

int vm_munmap(unsigned long start, size_t len)
{
    int ret;
    struct mm_struct *mm = current->mm;

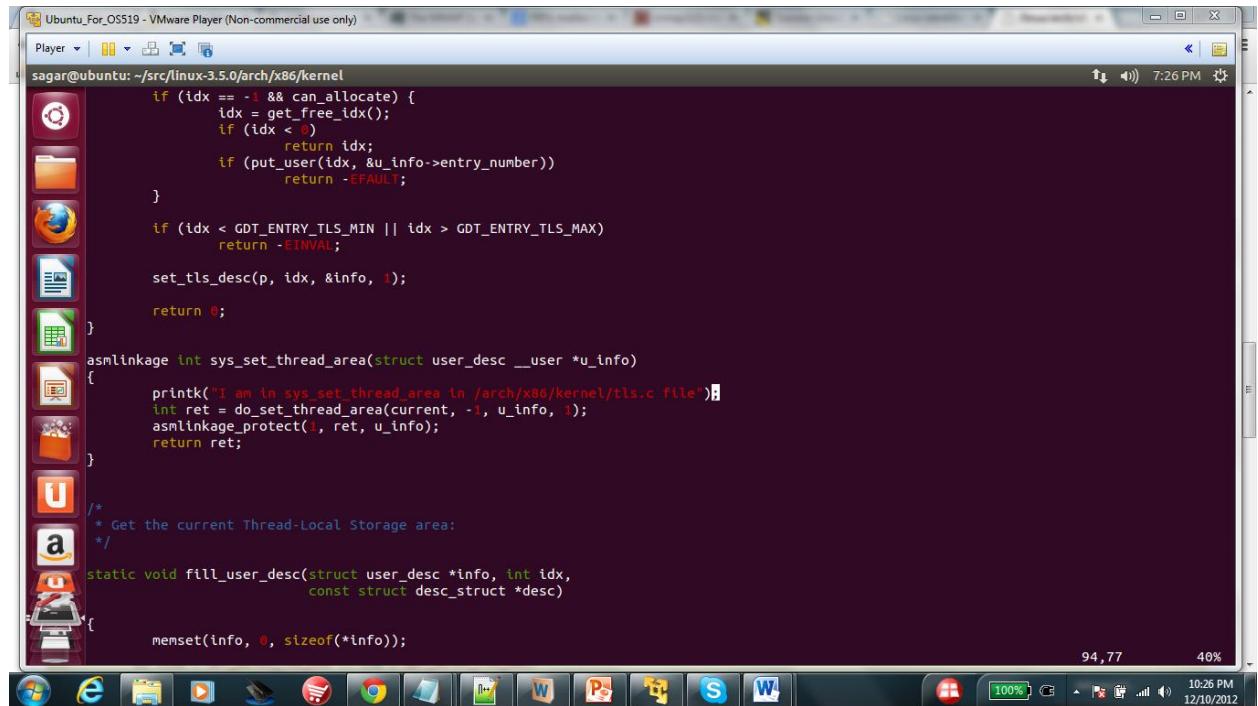
    down_write(&mm->mmap_sem);
    ret = do_munmap(mm, start, len);
    up_write(&mm->mmap_sem);
    return ret;
}
EXPORT_SYMBOL(vm_munmap);

SYSCALL_DEFINE2(munmap, unsigned long, addr, size_t, len)
{
    printk(KERN_INFO "I am in munmap sys_call in mm/mmap.c file");
    profile_munmap(addr);
    return vm_munmap(addr, len);
}

static inline void verify_mm_writelocked(struct mm_struct *mm)
#endif CONFIG_DEBUG_VM
{
    if (unlikely(down_read_trylock(&mm->mmap_sem)))
        WARN_ON();
    up_read(&mm->mmap_sem);
}
#endif

```

### sys\_set\_thread\_area at location /arch/x86/kernel



```

sagar@ubuntu:~/src/linux-3.5.0/arch/x86/kernel
    if (idx == -1 && can_allocate) {
        idx = get_free_idx();
        if (idx < 0)
            return idx;
        if (put_user(idx, &u_info->entry_number))
            return -EFAULT;
    }

    if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX)
        return -EINVAL;

    set_tls_desc(p, idx, &u_info, 1);
    return 0;
}

asmlinkage int sys_set_thread_area(struct user_desc __user *u_info)
{
    printk("I am in sys_set_thread_area in /arch/x86/kernel/tls.c file");
    int ret = do_set_thread_area(current, -1, u_info, 1);
    asmlinkage_protect(1, ret, u_info);
    return ret;
}

/*
 * Get the current Thread-Local Storage area:
 */
static void fill_user_desc(struct user_desc *info, int idx,
                           const struct desc_struct *desc)
{
    memset(info, 0, sizeof(*info));
}

```

Since the above system call uses function do\_set\_thread\_area, the below screen shot includes that function too.

```

sagar@ubuntu: ~/src/linux-3.5.0/arch/x86/kernel
int do_set_thread_area(struct task_struct *p, int idx,
                      struct user_desc __user *u_info,
                      int can_allocate)
{
    printk(KERN_INFO "I am in so_set_thread_area in /arch/x86/kernel/tls.c");
    struct user_desc info;

    if (copy_from_user(&info, u_info, sizeof(info)))
        return -EFAULT;

    if (idx == -1)
        idx = info.entry_number;

    /*
     * index -1 means the kernel should try to find and
     * allocate an empty descriptor:
     */
    if (idx == -1 && can_allocate) {
        idx = get_free_idx();
        if (idx < 0)
            return idx;
        if (put_user(idx, &u_info->entry_number))
            return -EFAULT;
    }

    if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX)
        return -EINVAL;

    set_tls_desc(p, idx, &info, 1);
    return 0;
}

```

After inserting printk statements as shown above, we compiled our kernel again and checked /var/messages/log as shown below.

```

sagar@ubuntu: ~/src/linux-3.5.0/mm
Dec 11 05:37:59 ubuntu syslogd 1.5.0#6ubuntu1: restart.
Dec 11 05:37:59 ubuntu kernel: [ 1306.502980] I am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503092] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503143] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503241] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503329] I am in sys_protect in mm/mpprotect.cI am in sys_protect in mm/mpprotect.c
Dec 11 05:37:59 ubuntu kernel: [ 1306.503371] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503419] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503512] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503578] I am in sys_protect in mm/mpprotect.cI am in sys_protect in mm/mpprotect.c
Dec 11 05:37:59 ubuntu kernel: [ 1306.503614] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.503699] I am in munmap sys_call in mm/mmap.c file<6>[ 1306.510930] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.510932] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.510933] We are still in sys_brk system call definition under out section
Dec 11 05:37:59 ubuntu kernel: [ 1306.511004] I am in sys_access system probing done for malloc
Dec 11 05:37:59 ubuntu kernel: [ 1306.511053] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.511106] I am in sys_access system probing done for mallocI am in sys_fstat64 system call in fs/stat.c file
Dec 11 05:37:59 ubuntu kernel: [ 1306.511169] I am in sys_protect in mm/mpprotect.cI am in sys_set_thread_area in /arch/x86/kernel/tls.c
Dec 11 05:37:59 ubuntu kernel: [ 1306.511269] I am in so_set_thread_area in /arch/x86/kernel/tls.cI am in sys_protect in mm/mpprotect.c
Dec 11 05:37:59 ubuntu kernel: [ 1306.511476] I am in sys_protect in mm/mpprotect.cI am in sys_protect in mm/mpprotect.c
Dec 11 05:37:59 ubuntu kernel: [ 1306.511500] I am in munmap sys_call in mm/mmap.c file<6>[ 1306.511705] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.511706] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.511767] We are still in sys_brk system call definition under out section
Dec 11 05:37:59 ubuntu kernel: [ 1306.511769] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.511716] I am currently under sys_call definition of sys_brk
Dec 11 05:37:59 ubuntu kernel: [ 1306.511712] We are still in sys_brk system call definition under out section
"/var/log/messages" [readonly] 22440L, 2922615C

```

```

sagar@ubuntu:~/src/linux-3.5.0/mm
Dec 11 10:59:49 ubuntu kernel: [18781.596426] I am in sys_fstat64 system call in fs/stat.c fileI am in sys_fstat64 system call in fs/stat.c f
ile
Dec 11 10:59:49 ubuntu kernel: [18781.596738] I am in munmap sys_call in mm/mmap.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.596975] I am in sys_fstat64 system call in fs/stat.c fileI am in sys_fstat64 system call in fs/stat.c f
ile
Dec 11 10:59:49 ubuntu kernel: [18781.597198] I am in sys_fstat64 system call in fs/stat.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.597530] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.612011] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.612013] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.612018] We are still in sys_brk system call definition under out section
Dec 11 10:59:49 ubuntu kernel: [18781.631267] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.631274] We are still in sys_brk system call definition under out section
Dec 11 10:59:49 ubuntu kernel: [18781.659034] I am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.659105] I am in munmap sys_call in mm/mmap.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.659493] I am in sys_fstat64 system call in fs/stat.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.659664] I am in sys_fstat64 system call in fs/stat.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.659925] I am in sys_fstat64 system call in fs/stat.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.660288] I am in sys_fstat64 system call in fs/stat.c fileI am in munmap sys_call in mm/mmap.c file
Dec 11 10:59:49 ubuntu kernel: [18781.660723] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660724] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660728] We are still in sys_brk system call definition under out section
Dec 11 10:59:49 ubuntu kernel: [18781.660753] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660755] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660761] We are still in sys_brk system call definition under out section
Dec 11 10:59:49 ubuntu kernel: [18781.660763] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660764] I am currently under sys_call definition of sys_brk
Dec 11 10:59:49 ubuntu kernel: [18781.660767] We are still in sys_brk system call definition under out section
Dec 11 10:59:49 ubuntu kernel: [18781.660778] I am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.660823] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.660915] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.661119] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.661263] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
Dec 11 10:59:49 ubuntu kernel: [18781.661464] I am in munmap sys_call in mm/mmap.c fileI am in sys_fstat64 system call in fs/stat.c file
22440,1 Bot

```

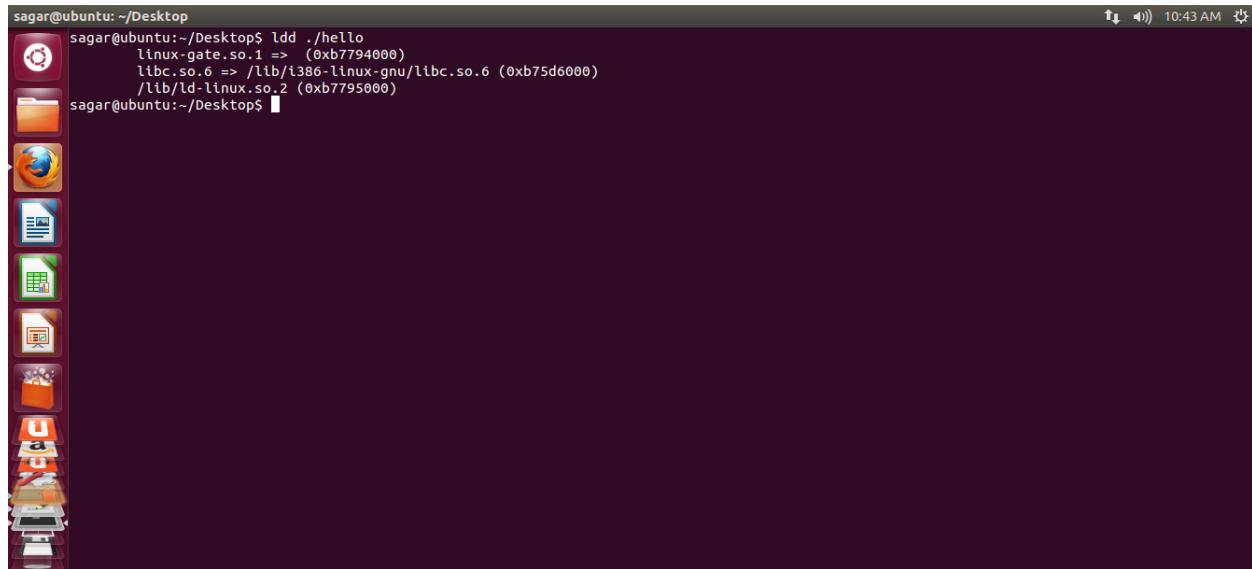
Analyzing the above output , we could see printk statements which we added from sys\_brk system call, sys\_access ,sys\_fstat64,sys\_munmap, sys\_protect, sys\_set\_thread\_area .

Despite of not running the hello.c program we were still getting so many printk statements, we concluded that not only malloc by many other processes need memory dynamically there these system calls are accessed again and again and hence many printk statement.

To filter out which system calls are accessed by malloc(), we decided to download the glibc source code of the same version our kernel is using. We planned to install the glibc library and put a print statement in malloc () function in malloc.c file

Below are the screen shots of downloading and compilation of glibc but we failed to compile it because it needed some patch files, which we could not locate.

To see which version of glibc file our ./hello program is using we did the following



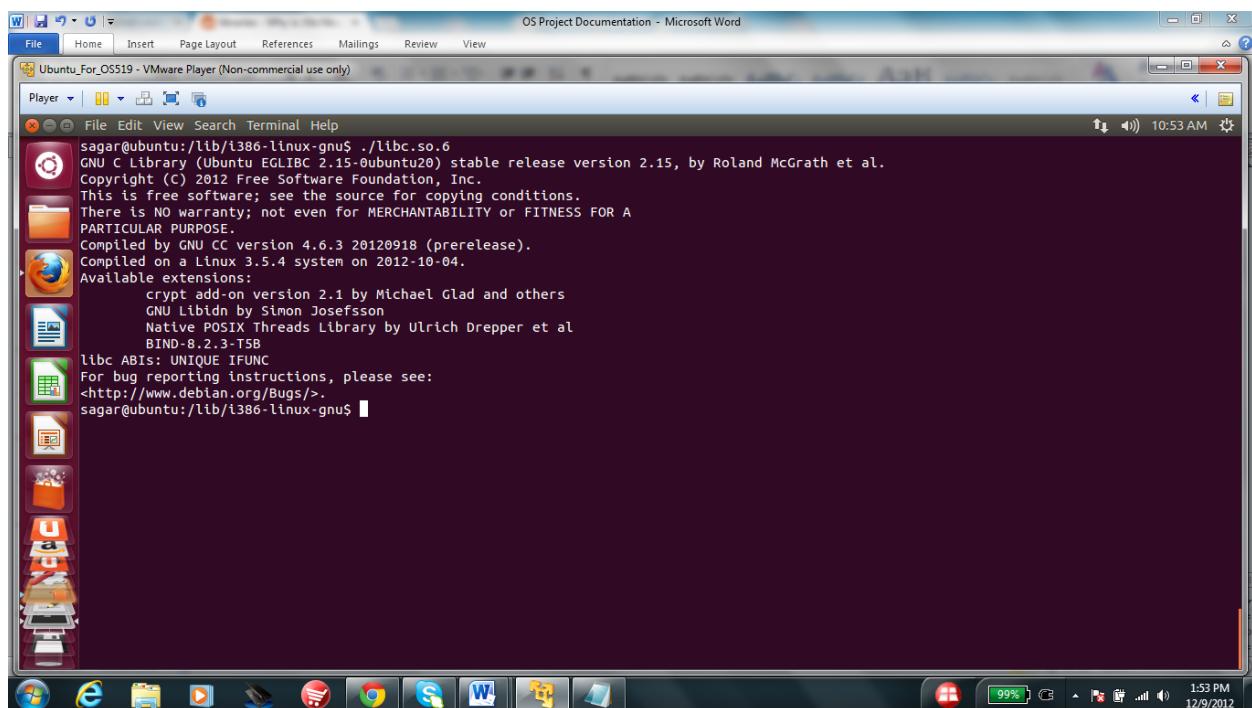
```
sagar@ubuntu:~/Desktop
sagar@ubuntu:~/Desktop$ ldd ./hello
    linux-gate.so.1 => (0xb7794000)
    libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb75d6000)
    /lib/ld-linux.so.2 (0xb7795000)
sagar@ubuntu:~/Desktop$
```

The screenshot shows a standard Ubuntu desktop environment with a dark purple background. On the left, there is a vertical dock containing icons for various applications like the Dash, Home, Insert, Page Layout, References, Mailings, Review, View, and a terminal window. The terminal window is open and displays the command 'ldd ./hello' followed by its output. The status bar at the top right shows the date and time as 10:43 AM. The bottom of the screen features the Unity interface with its characteristic dock of icons.

From the second line of output we can see that libc.so.6 from glibc shared libraries is loaded.

Now to see which version of glibc kernel is using , we need to run libc.so.6 file using the following command

./libc.so.6 at /lib/i386-linux-gnu/ directory

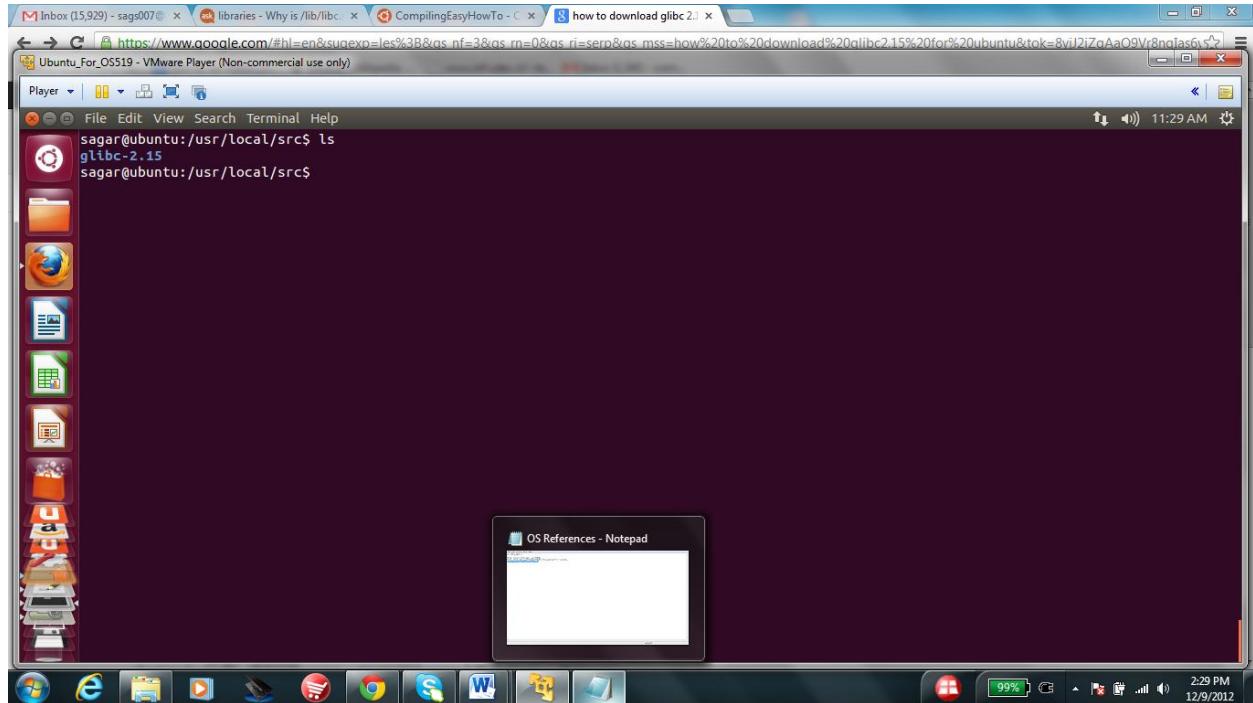


```
sagar@ubuntu:/lib/i386-linux-gnu$ ./libc.so.6
GNU C Library (Ubuntu EGLIBC 2.15-0ubuntu20) stable release version 2.15, by Roland McGrath et al.
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 4.6.3 20120918 (prerelease).
Compiled on a Linux 3.5.4 system on 2012-10-04.
Available extensions:
      crypt add-on version 2.1 by Michael Glad and others
      GNU Libidn by Simon Josefsson
      Native POSIX Threads Library by Ulrich Drepper et al
      BIND-8.2.3-TSB
libc ABIs: UNIQUE IFUNC
For bug reporting instructions, please see:
<http://www.debian.org/Bugs/>
sagar@ubuntu:/lib/i386-linux-gnu$
```

The screenshot shows a Windows desktop environment with a blue title bar for 'OS Project Documentation - Microsoft Word'. Below it, a terminal window is open in a VMware Player session titled 'Ubuntu\_For\_O5519 - VMware Player (Non-commercial use only)'. The terminal window displays the command './libc.so.6' followed by its detailed output. The status bar at the top right shows the date and time as 10:53 AM. The bottom of the screen features the Windows taskbar with various application icons.

The glibc version which is current used by the kernel is 2.15

Let's download, the glibc version 2.15 source code from the following source  
<http://ftp.gnu.org/gnu/glibc/> and store it at /usr/local/src in our linux file system.

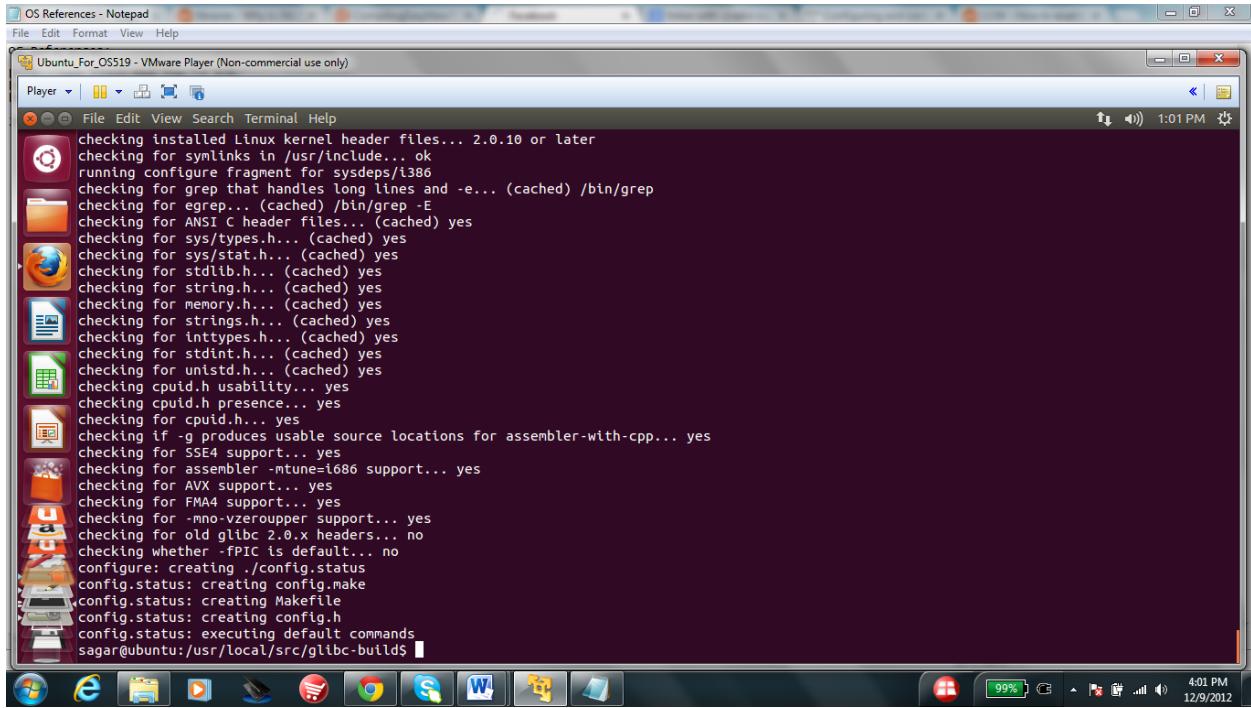


Below screen shot shows the files in glibc library

```
sagar@ubuntu: /usr/local/src/glibc-2.15
sagar@ubuntu: /usr/local/src$ ls
glibc-2.15
sagar@ubuntu: /usr/local/src$ cd glibc-2.15
sagar@ubuntu: /usr/local/src/glibc-2.15$ ls
abilist      ChangeLog.13  config     FAQ       libidn     NEWS      rt        termios
abi-tags     ChangeLog.14  configure.in  FAQ.in    libio      nis       Rules     test-skeleton.c
aclocal.m4    ChangeLog.15  conform     gmon     LICENSES  NOTES    scripts   time
aout        ChangeLog.16  CONFORMANCE  gnulib     locale    nptl     setjmp   timezone
argp        ChangeLog.17  COPYING     grp      localizedata  nptl_db  shadow   version.h
assert      ChangeLog.2   COPYING.LIB  gshadow   login     nscd     shlib-versions Versions.def
bits         ChangeLog.3   cppflags-iterator.mk  hurd     Makeconfig o-iterator.mk  signal   wcsnbs
BUGS         ChangeLog.4   crypt      iconv     Makefile   po       socket   wctype
CANCEL-FCT-WAIVE ChangeLog.5   csu      include   Makerules PROJECTS stdio-common
CANCEL-FILE-WAIVE ChangeLog.6   ctype     install   malloc    pwd      streams
catgets     ChangeLog.7   debug      intl     manual    README   string
ChangeLog   ChangeLog.8   dirent     ifuncn   INSTALL   math    README.libm sunrpc
ChangeLog.1 ChangeLog.9   elf      intl     manual    README   string
ChangeLog.10 conf      extra-lib.mk  to      misc     resolv   sysdeps
ChangeLog.11 config.h.in extra-modules.mk  libc-abis  NAMESPACe resource  sysvipc
ChangeLog.12 config.make.in extra-modules.mk
sagar@ubuntu: /usr/local/src/glibc-2.15$
```

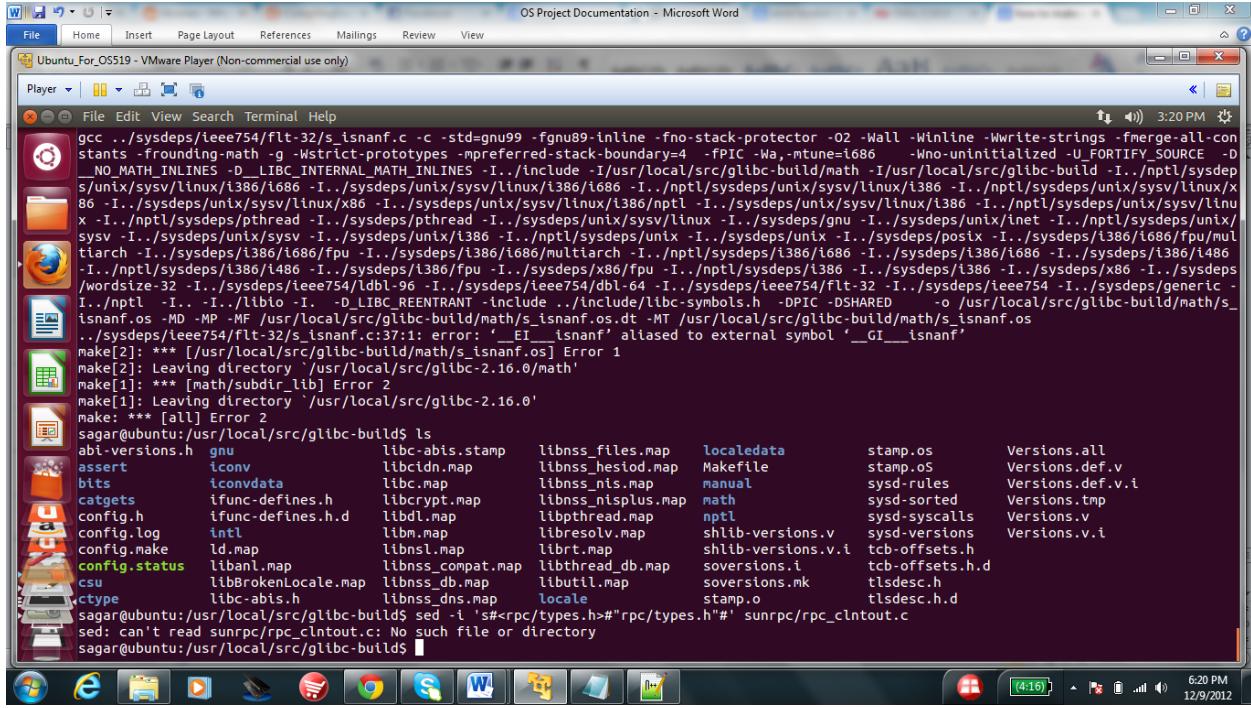
Now we compiled the source code

```
sagar@ubuntu: /usr/local/src/glibc-2.15$ make
sagar@ubuntu: /usr/local/src/glibc-2.15$
```



```
OS References - Notepad
File Edit Format View Help
Ubuntu_Forum - VMware Player (Non-commercial use only)
Player File Edit View Search Terminal Help
checking installed Linux kernel header files... 2.0.10 or later
checking for symlinks in /usr/include... ok
running configure fragment for sysdeps/i386
checking for grep that handles long lines and -e... (cached) /bin/grep
checking for egrep... (cached) /bin/grep -E
checking for ANSI C header files... (cached) yes
checking for sys/types.h... (cached) yes
checking for sys/stat.h... (cached) yes
checking for stdlib.h... (cached) yes
checking for string.h... (cached) yes
checking for memory.h... (cached) yes
checking for strings.h... (cached) yes
checking for inttypes.h... (cached) yes
checking for stdint.h... (cached) yes
checking for unistd.h... (cached) yes
checking cpuid.h usability... yes
checking cpuid.h presence... yes
checking for cpuid.h... yes
checking if -g produces usable source locations for assembler-with-cpp... yes
checking for SSE4 support... yes
checking for assembler -mtune=i686 support... yes
checking for AVX support... yes
checking for FMA4 support... yes
checking for -mno-vzeroupper support... yes
checking for old glibc 2.0.x headers... no
checking whether -fPIC is default... no
configure: creating ./config.status
config.status: creating config.make
config.status: creating Makefile
config.status: creating config.h
config.status: executing default commands
sagar@ubuntu:/usr/local/src/glibc-build$
```

But we finally got the errors while compiling



```
W File Home Insert Page Layout References Mailings Review View
Ubuntu_Forum - VMware Player (Non-commercial use only)
Player File Edit View Search Terminal Help
gc .../sysdeps/ieee754/flt-32/s_isnanf.c -c -std=gnu99 -fno-stack-protector -O2 -Wall -Winline -Wwrite-strings -fmerge-all-constants -frounding-math -g -Wstrict-prototypes -mpreferred-stack-boundary=4 -fPIC -Wa,-mtune=i686 -Wno-uninitialized -U_FORTIFY_SOURCE -D __NO_MATH_INLINES -D_LIBC_INTERNAL_MATH_INLINES -I./include -I/usr/local/src/glibc-build/math -I/usr/local/src/glibc-build -I../sysdeps/unix/sysv/linux/i386/i686 -I..../sysdeps/unix/sysv/linux/i386/i686 -I..../sysdeps/unix/sysv/linux/i386 -I..../sysdeps/unix/sysv/linux/x86 -I..../sysdeps/unix/sysv/linux/x86 -I..../sysdeps/unix/sysv/linux/i386/nptl -I..../sysdeps/unix/sysv/linux/i386 -I..../nptl/sysdeps/pthread -I..../sysdeps/pthread -I..../sysdeps/unix/sysv/linux -I..../sysdeps/unix/sysv/linux -I..../sysdeps/unix/sysv -I..../sysdeps/unix/sysv -I..../sysdeps/unix/i386 -I..../nptl/sysdeps/unix -I..../sysdeps/unix -I..../sysdeps/postx -I..../sysdeps/untx -I..../sysdeps/tstx -I..../sysdeps/tstx -I..../sysdeps/fpu/multarch -I..../sysdeps/ieee754/i386/i686/multarch -I..../nptl/sysdeps/ieee754/i386/i686 -I..../sysdeps/ieee754/i386/i686 -I..../sysdeps/ieee754/i386/i486 -I..../nptl/sysdeps/ieee754/i386/i486 -I..../sysdeps/ieee754/fpu -I..../nptl/sysdeps/i386 -I..../sysdeps/ieee754/i386 -I..../sysdeps/ieee754/x86 -I..../sysdeps/wordsizes-32 -I..../sysdeps/ieee754/dbl-96 -I..../sysdeps/ieee754/dbl-64 -I..../sysdeps/ieee754/flt-32 -I..../sysdeps/ieee754 -I..../sysdeps/generic -I..../nptl -I..../libio -I..../LIBC_REENTRANT -Iinclude -Iinclude/libc-symbols.h -DPIC -DSHARED -o /usr/local/src/glibc-build/math/s_isnanf.os -MD -MP -MF /usr/local/src/glibc-build/math/s_isnanf.os.dt -MT /usr/local/src/glibc-build/math/s_isnanf.os -MD -MF /usr/local/src/glibc-build/math/s_isnanf.os -MT /usr/local/src/glibc-build/math/s_isnanf.os Error 1
make[2]: *** [/usr/local/src/glibc-build/math/s_isnanf.os] Error 1
make[2]: Leaving directory '/usr/local/src/glibc-2.16.0/math'
make[1]: *** [math subdir lib] Error 2
make[1]: Leaving directory '/usr/local/src/glibc-2.16.0'
make: *** [all] Error 2
sagar@ubuntu:/usr/local/src/glibc-build$ ls
abi-versions.h  gnu                libc-abis.stamp    libnss_files.map    localedata          stamp.os         Versions.all
assert           iconv               libcldns.map       libnss_hesiod.map  Makefile           stamp.os         Versions.def.v
bits              iconvdata            libc.map          libnss_nis.map     manual            sysd-rules      Versions.def.v.i
catgets          ifunc-defines.h    libcrypt.map      libnss_nisplus.map  nptl              sysd-sorted     Versions.tmrp
config.h          ifunc-defines.h.d  libndl.map       libpthread.read.map  sysd-syscalls    Versions.v
config.log        intl                libnm.map        libresolv.map    shlib-versions.v   sysd-versions  Versions.v.i
config.make       ld.map               libnsl.map       librt.map         shlib-versions.v.i tcb-offsets.h
config.status     libban.map          libnss_compat.map libthread_db.map    sysd-versions.mk tcb-offsets.h.d
csu              libBrokenLocale.map libnss_dns.map    libutil.map       soversions.mk    tlsdesc.h
ctype            libc-abis.h        libnss_db.map    locale          stamp.o        tlsdesc.h.d
sagar@ubuntu:/usr/local/src/glibc-build$ sed -i '#sunrpc/rpc_clntout.c:# sunrpc/rpc_clntout.c' sunrpc/rpc_clntout.c
sed: can't read sunrpc/rpc_clntout.c: No such file or directory
sagar@ubuntu:/usr/local/src/glibc-build$
```

Finally we were left to analyze the malloc.c source code in malloc directory in glibc library.

After studying carefully we analyzed that, malloc function allocates the dynamic memory from heap and it only need help from kernel when, processes current memory is over and it needs

more memory. When process needs more memory . malloc() function calls either sys\_brk or mmap2 system calls to increase the size of the heap.

The source code for sys\_brk() and mmap() is at Appendix at the end of the document.

## Conclusion:

Based on our research, we could conclude that malloc() function allocates the dynamic memory from heap and it only needs help from kernel when, processes current memory is over and it needs more memory. When process needs more memory . malloc() function calls either sys\_brk or mmap2 system calls to increase or decrease the size of the heap.

## Things Learned

1. Our team learned the process of compiling the kernel and during the process we faced many challenges. For Example, Compilation time for getting over 10 hours. We increased the RAM size to 6 GB instead of 2 GB earlier. The compilation time reduced to 3-4 hours. We compiled and installed the Kernel 3 – 4 times and every time we learned something new.
2. Use of Virtual Machine was always helpful. First of all it acted as a backup. We use to save the state of virtual machine every time we start working on the project. Because there were always chances of getting wrong while compiling and installing kernel.
3. We learnt the file structure of the Linux operating system, which gave us confidence to understand Linux related things
4. We learnt how to modify already installed kernel files. For Example we edited various system calls by putting printk statements , to analyse the flow of the program.
5. We learned how malloc() function flow work.

## Sources:

<http://www.linux-mag.com/id/806/>  
<http://www.linux-mag.com/id/827/>  
<http://lxr.free-electrons.com/>  
<http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux>  
<http://www.ibm.com/developerworks/linux/library/l-linuxboot>  
[http://en.wikipedia.org/wiki/Linux\\_startup\\_process](http://en.wikipedia.org/wiki/Linux_startup_process)  
<http://en.wikipedia.org/wiki/Dpkg>  
[http://en.wikipedia.org/wiki/Advanced\\_Packaging\\_Tool](http://en.wikipedia.org/wiki/Advanced_Packaging_Tool)  
<http://www.gratisoft.us/sudo/intro.html>  
[http://en.wikipedia.org/wiki/Make\\_%28software%29](http://en.wikipedia.org/wiki/Make_%28software%29)  
[http://en.wikipedia.org/wiki/Topological\\_sorting](http://en.wikipedia.org/wiki/Topological_sorting)  
<http://www.wlug.org.nz/MakefileHowto>  
[http://linuxdevcenter.com/pub/a/linux/2002/01/31/make\\_intro.html](http://linuxdevcenter.com/pub/a/linux/2002/01/31/make_intro.html)  
<http://www.gnu.org/software/make/manual/make.html>  
<http://www.dedoimedo.com/computers/grub.html>

# Appendix

```
// malloc() This code is glibc/malloc/malloc.c file This part only shows malloc()
//function otherwise this file contains 6000 lines of code

static void*
_int_malloc(mstate av, size_t bytes)
{
    INTERNAL_SIZE_T nb;           /* normalized request size */
    unsigned int     idx;          /* associated bin index */
    mbinptr         bin;          /* associated bin */

    mchunkptr       victim;        /* inspected/selected chunk */
    INTERNAL_SIZE_T size;         /* its size */
    int             victim_index; /* its bin index */

    mchunkptr       remainder;     /* remainder from a split */
    unsigned long   remainder_size; /* its size */

    unsigned int     block;        /* bit map traverser */
    unsigned int     bit;          /* bit map traverser */
    unsigned int     map;          /* current word of binmap */

    mchunkptr       fwd;           /* misc temp for linking */
    mchunkptr       bck;           /* misc temp for linking */

    const char *errstr = NULL;

/*
 * Convert request size to internal form by adding SIZE_SZ bytes
 * overhead plus possibly more to obtain necessary alignment and/or
 * to obtain a size of at least MINSIZE, the smallest allocatable
 * size. Also, checked_request2size traps (returning 0) request sizes
 * that are so large that they wrap around zero when padded and
 * aligned.
 */
checked_request2size(bytes, nb);

/*
 * If the size qualifies as a fastbin, first check corresponding bin.
 * This code is safe to execute even if av is not yet initialized, so we
 * can try it without checking, which saves some time on this fast path.
 */
if ((unsigned long)(nb) <= (unsigned long)(get_max_fast ())) {
    idx = fastbin_index(nb);
    mfastbinptr* fb = &fastbin (av, idx);
    mchunkptr pp = *fb;
    do
    {
        victim = pp;
        if (victim == NULL)
            break;
    }
    while ((pp = catomic_compare_and_exchange_val_acq (fb, victim->fd, victim))
           != victim);
    if (victim != 0) {
        if (__builtin_expect (fastbin_index (chunksize (victim)) != idx, 0))
        {
            errstr = "malloc(): memory corruption (fast)";
        }
    }
}
```

```

    errout:
        malloc_printerr (check_action, errstr, chunk2mem (victim));
        return NULL;
    }
    check_remalloced_chunk(av, victim, nb);
    void *p = chunk2mem(victim);
    if (_builtin_expect (perturb_byte, 0))
        alloc_perturb (p, bytes);
    return p;
}
}

/*
If a small request, check regular bin. Since these "smallbins"
hold one size each, no searching within bins is necessary.
(For a large request, we need to wait until unsorted chunks are
processed to find best fit. But for small ones, fits are exact
anyway, so we can check now, which is faster.)
*/
if (in_smallbin_range(nb)) {
    idx = smallbin_index(nb);
    bin = bin_at(av, idx);

    if ( (victim = last(bin)) != bin) {
        if (victim == 0) /* initialization check */
            malloc_consolidate(av);
        else {
            bck = victim->bk;
            if (_builtin_expect (bck->fd != victim, 0))
            {
                errstr = "malloc(): smallbin double linked list corrupted";
                goto errout;
            }
            set_inuse_bit_at_offset(victim, nb);
            bin->bk = bck;
            bck->fd = bin;

            if (av != &main_arena)
                victim->size |= NON_MAIN_arena;
            check_malloced_chunk(av, victim, nb);
            void *p = chunk2mem(victim);
            if (_builtin_expect (perturb_byte, 0))
                alloc_perturb (p, bytes);
            return p;
        }
    }
}

/*
If this is a large request, consolidate fastbins before continuing.
While it might look excessive to kill all fastbins before
even seeing if there is space available, this avoids
fragmentation problems normally associated with fastbins.
Also, in practice, programs tend to have runs of either small or
large requests, but less often mixtures, so consolidation is not
invoked all that often in most programs. And the programs that
it is called frequently in otherwise tend to fragment.
*/
else {
    idx = largebin_index(nb);
    if (have_fastchunks(av))

```

```

    malloc_consolidate(av);
}

/*
Process recently freed or remaindered chunks, taking one only if
it is exact fit, or, if this a small request, the chunk is remainder from
the most recent non-exact fit. Place other traversed chunks in
bins. Note that this step is the only place in any routine where
chunks are placed in bins.

The outer loop here is needed because we might not realize until
near the end of malloc that we should have consolidated, so must
do so and retry. This happens at most once, and only when we would
otherwise need to expand memory to service a "small" request.
*/
for(;;) {

    int iters = 0;
    while ( (victim = unsorted_chunks(av)->bk) != unsorted_chunks(av)) {
        bck = victim->bk;
        if (_builtin_expect (victim->size <= 2 * SIZE_SZ, 0)
            || _builtin_expect (victim->size > av->system_mem, 0))
            malloc_perr (check_action, "malloc(): memory corruption",
                         chunk2mem (victim));
        size = chunksizesize(victim);

        /*
        If a small request, try to use last remainder if it is the
        only chunk in unsorted bin. This helps promote locality for
        runs of consecutive small requests. This is the only
        exception to best-fit, and applies only when there is
        no exact fit for a small chunk.
        */
        if (in_smallbin_range(nb) &&
            bck == unsorted_chunks(av) &&
            victim == av->last_remainder &&
            (unsigned long)(size) > (unsigned long)(nb + MINSIZE)) {

            /* split and reattach remainder */
            remainder_size = size - nb;
            remainder = chunk_at_offset(victim, nb);
            unsorted_chunks(av)->bk = unsorted_chunks(av)->fd = remainder;
            av->last_remainder = remainder;
            remainder->bk = remainder->fd = unsorted_chunks(av);
            if (!in_smallbin_range(remainder_size))
            {
                remainder->fd_nextsize = NULL;
                remainder->bk_nextsize = NULL;
            }

            set_head(victim, nb | PREV_INUSE |
                     (av != &main_arena ? NON_MAIN_ARENA : 0));
            set_head(remainder, remainder_size | PREV_INUSE);
            set_foot(remainder, remainder_size);

            check_malloced_chunk(av, victim, nb);
            void *p = chunk2mem(victim);
            if (_builtin_expect (perturb_byte, 0))
                alloc_perturb (p, bytes);
            return p;
        }
    }
}

```

```

/* remove from unsorted list */
unsorted_chunks(av)->bk = bck;
bck->fd = unsorted_chunks(av);

/* Take now instead of binning if exact fit */

if (size == nb) {
    set_inuse_bit_at_offset(victim, size);
    if (av != &main_arena)
        victim->size |= NON_MAIN_arena;
    check_malloced_chunk(av, victim, nb);
    void *p = chunk2mem(victim);
    if (_builtin_expect (perturb_byte, 0))
        alloc_perturb (p, bytes);
    return p;
}

/* place chunk in bin */

if (in_smallbin_range(size)) {
    victim_index = smallbin_index(size);
    bck = bin_at(av, victim_index);
    fwd = bck->fd;
}
else {
    victim_index = largebin_index(size);
    bck = bin_at(av, victim_index);
    fwd = bck->fd;
}

/* maintain large bins in sorted order */
if (fwd != bck) {
    /* Or with inuse bit to speed comparisons */
    size |= PREV_INUSE;
    /* if smaller than smallest, bypass loop below */
    assert((bck->bk->size & NON_MAIN_arena) == 0);
    if ((unsigned long)(size) < (unsigned long)(bck->bk->size)) {
        fwd = bck;
        bck = bck->bk;

        victim->fd_nextsize = fwd->fd;
        victim->bk_nextsize = fwd->fd->bk_nextsize;
        fwd->fd->bk_nextsize = victim->bk_nextsize->fd_nextsize = victim;
    }
    else {
        assert((fwd->size & NON_MAIN_arena) == 0);
        while ((unsigned long) size < fwd->size)
        {
            fwd = fwd->fd_nextsize;
            assert((fwd->size & NON_MAIN_arena) == 0);
        }

        if ((unsigned long) size == (unsigned long) fwd->size)
            /* Always insert in the second position. */
            fwd = fwd->fd;
        else
        {
            victim->fd_nextsize = fwd;
            victim->bk_nextsize = fwd->bk_nextsize;
            fwd->bk_nextsize = victim;
            victim->bk_nextsize->fd_nextsize = victim;
        }
        bck = fwd->bk;
    }
}

```

```

    }
} else
    victim->fd_nexsize = victim->bk_nexsize = victim;
}

mark_bin(av, victim_index);
victim->bk = bck;
victim->fd = fwd;
fwd->bk = victim;
bck->fd = victim;

#define MAX_ITERS 10000
if (++iters >= MAX_ITERS)
    break;
}

/*
   If a large request, scan through the chunks of current bin in
   sorted order to find smallest that fits.  Use the skip list for this.
*/

if (!in_smallbin_range(nb)) {
    bin = bin_at(av, idx);

    /* skip scan if empty or largest chunk is too small */
    if ((victim = first(bin)) != bin &&
        (unsigned long)(victim->size) >= (unsigned long)(nb)) {

        victim = victim->bk_nexsize;
        while (((unsigned long)(size = chunksize(victim)) <
                (unsigned long)(nb)))
            victim = victim->bk_nexsize;

        /* Avoid removing the first entry for a size so that the skip
           list does not have to be rerouted.  */
        if (victim != last(bin) && victim->size == victim->fd->size)
            victim = victim->fd;

        remainder_size = size - nb;
        unlink(victim, bck, fwd);

        /* Exhaust */
        if (remainder_size < MINSIZE) {
            set_inuse_bit_at_offset(victim, size);
            if (av != &main_arena)
                victim->size |= NON_MAIN_arena;
        }
        /* Split */
    } else {
        remainder = chunk_at_offset(victim, nb);
        /* We cannot assume the unsorted list is empty and therefore
           have to perform a complete insert here.  */
        bck = unsorted_chunks(av);
        fwd = bck->fd;
        if (_builtin_expect (fwd->bk != bck, 0))
        {
            errstr = "malloc(): corrupted unsorted chunks";
            goto errout;
        }
        remainder->bk = bck;
        remainder->fd = fwd;
        bck->fd = remainder;
        fwd->bk = remainder;
    }
}

```

```

if (!in_smallbin_range(remainder_size))
{
    remainder->fd_nextsize = NULL;
    remainder->bk_nextsize = NULL;
}
set_head(victim, nb | PREV_INUSE |
          (av != &main_arena ? NON_MAIN_ARENA : 0));
set_head(remainder, remainder_size | PREV_INUSE);
set_foot(remainder, remainder_size);
}
check_malloced_chunk(av, victim, nb);
void *p = chunk2mem(victim);
if (_builtin_expect (perturb_byte, 0))
    alloc_perturb (p, bytes);
return p;
}

/*
Search for a chunk by scanning bins, starting with next largest
bin. This search is strictly by best-fit; i.e., the smallest
(with ties going to approximately the least recently used) chunk
that fits is selected.

The bitmap avoids needing to check that most blocks are nonempty.
The particular case of skipping all bins during warm-up phases
when no chunks have been returned yet is faster than it might look.
*/
++idx;
bin = bin_at(av, idx);
block = idx2block(idx);
map = av->binmap[block];
bit = idx2bit(idx);

for (;;) {
    /* Skip rest of block if there are no more set bits in this block. */
    if (bit > map || bit == 0) {
        do {
            if (++block >= BINMAPSIZE) /* out of bins */
                goto use_top;
            map = av->binmap[block];
        } while (map == 0);
        bin = bin_at(av, (block << BINMAPSHIFT));
        bit = 1;
    }

    /* Advance to bin with set bit. There must be one. */
    while ((bit & map) == 0) {
        bin = next_bin(bin);
        bit <= 1;
        assert(bit != 0);
    }

    /* Inspect the bin. It is likely to be non-empty */
    victim = last(bin);

    /* If a false alarm (empty bin), clear the bit. */
    if (victim == bin) {
        av->binmap[block] = map &= ~bit; /* Write through */
        bin = next_bin(bin);
        bit <= 1;
    }
}

```

```

    }

else {
    size = chunksizes(victim);

    /* We know the first chunk in this bin is big enough to use. */
    assert((unsigned long)(size) >= (unsigned long)(nb));

    remainder_size = size - nb;

    /* unlink */
    unlink(victim, bck, fwd);

    /* Exhaust */
    if (remainder_size < MINSIZE) {
        set_inuse_bit_at_offset(victim, size);
        if (av != &main_arena)
            victim->size |= NON_MAIN_arena;
    }

    /* Split */
    else {
        remainder = chunk_at_offset(victim, nb);

        /* We cannot assume the unsorted list is empty and therefore
           have to perform a complete insert here. */
        bck = unsorted_chunks(av);
        fwd = bck->fd;
        if (_builtin_expect (fwd->bk != bck, 0))
        {
            errstr = "malloc(): corrupted unsorted chunks 2";
            goto errout;
        }
        remainder->bk = bck;
        remainder->fd = fwd;
        bck->fd = remainder;
        fwd->bk = remainder;

        /* advertise as last remainder */
        if (in_smallbin_range(nb))
            av->last_remainder = remainder;
        if (!in_smallbin_range(remainder_size))
        {
            remainder->fd_nextsize = NULL;
            remainder->bk_nextsize = NULL;
        }
        set_head(victim, nb | PREV_INUSE |
                  (av != &main_arena ? NON_MAIN_arena : 0));
        set_head(remainder, remainder_size | PREV_INUSE);
        set_foot(remainder, remainder_size);
    }

    check_malloced_chunk(av, victim, nb);
    void *p = chunk2mem(victim);
    if (_builtin_expect (perturb_byte, 0))
        alloc_perturb (p, bytes);
    return p;
}

use_top:
/*
 If large enough, split off the chunk bordering the end of memory
 (held in av->top). Note that this is in accord with the best-fit

```

```
search rule. In effect, av->top is treated as larger (and thus
less well fitting) than any other available chunk since it can
be extended to be as large as necessary (up to system
limitations).
```

We require that av->top always exists (i.e., has size >= MINSIZE) after initialization, so if it would otherwise be exhausted by current request, it is replenished. (The main reason for ensuring it exists is that we may need MINSIZE space to put in fenceposts in sysmalloc.)

```
*/
```

```
victim = av->top;
size = chunkszie(victim);

if ((unsigned long)(size) >= (unsigned long)(nb + MINSIZE)) {
    remainder_size = size - nb;
    remainder = chunk_at_offset(victim, nb);
    av->top = remainder;
    set_head(victim, nb | PREV_INUSE |
              (av != &main_arena ? NON_MAIN_ARENA : 0));
    set_head(remainder, remainder_size | PREV_INUSE);

    check_malloced_chunk(av, victim, nb);
    void *p = chunk2mem(victim);
    if (_builtin_expect (perturb_byte, 0))
        alloc_perturb (p, bytes);
    return p;
}

/* When we are using atomic ops to free fast chunks we can get
   here for all block sizes. */
else if (have_fastchunks(av)) {
    malloc_consolidate(av);
    /* restore original bin index */
    if (in_smallbin_range(nb))
        idx = smallbin_index(nb);
    else
        idx = largebin_index(nb);
}

/*
   Otherwise, relay to handle system-dependent cases
*/
else {
    void *p = sysmalloc(nb, av);
    if (p != NULL && _builtin_expect (perturb_byte, 0))
        alloc_perturb (p, bytes);
    return p;
}
}
```

### Sys\_brk() system call , located at mm/mmap.c file

```
static unsigned long do_brk(unsigned long addr, unsigned long len);

SYSCALL_DEFINE1(brk, unsigned long, brk)
{
    unsigned long rlim, retval;
    unsigned long newbrk, oldbrk;
    struct mm_struct *mm = current->mm;
    unsigned long min brk;
    printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
    down write(&mm->mmap sem);

#endif CONFIG_COMPAT_BRK
/*
 * CONFIG_COMPAT_BRK can still be overridden by setting
 * randomize_va_space to 2, which will still cause mm->start_brk
 * to be arbitrarily shifted
 */
    if (current->brk_randomized)
        min brk = mm->start_brk;
    else
        min brk = mm->end_data;
#else
    printk(KERN_INFO "I am currently under sys_call definition of sys_brk\n");
    min brk = mm->start_brk;
#endif
    if (brk < min brk)
        goto out;

/*
 * Check against rlimit here. If this check is done later after the test
 * of oldbrk with newbrk then it can escape the test and let the data
```

```

    * segment grow beyond its set limit the in case where the limit is
    * not page aligned -Ram Gupta
    */
rlim = rlimit(RLIMIT_DATA);
if (rlim < RLIM_INFINITY && (brk - mm->start_brk) +
    (mm->end_data - mm->start_data) > rlim)
    goto out;

newbrk = PAGE_ALIGN(brk);
oldbrk = PAGE_ALIGN(mm->brk);
if (oldbrk == newbrk)
    goto set_brk;

/* Always allow shrinking brk. */
if (brk <= mm->brk) {
    if (!do_munmap(mm, newbrk, oldbrk-newbrk))
        goto set_brk;
    goto out;
}

/* Check against existing mmap mappings. */
if (find_vma_intersection(mm, oldbrk, newbrk+PAGE_SIZE))
    goto out;

/* Ok, looks good - let it rip. */
if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
    goto out;
set brk:
    mm->brk = brk;
out:
    retval = mm->brk;
    up_write(&mm->mmap_sem);
    printk(KERN_INFO "We are still in sys_brk system call definition under out section\n");
    printk(KERN_INFO "We are still in sys brk system call definition under out section\n");
    return retval;
}

```

**sys\_mprotect system call defined in mm/mprotect.c file**

```

SYSCALL_DEFINE3(mprotect, unsigned long, start, size_t, len,
               unsigned long, prot)
{
    unsigned long vm_flags, nstart, end, tmp, reqprot;
    struct vm_area_struct *vma, *prev;
    int error = -EINVAL;
    const int grows = prot & (PROT_GROWSDOWN|PROT_GROWSUP);
    prot &= ~(PROT_GROWSDOWN|PROT_GROWSUP);
    if (grows == (PROT_GROWSDOWN|PROT_GROWSUP)) /* can't be both */
        return -EINVAL;

    if (start & ~PAGE_MASK)
        return -EINVAL;
    if (!len)
        return 0;
    len = PAGE_ALIGN(len);
    end = start + len;
    if (end <= start)
        return -ENOMEM;
    if (!arch_validate_prot(prot))
        return -EINVAL;

    reqprot = prot;
    /*
     * Does the application expect PROT_READ to imply PROT_EXEC:
     */

```

```

/*
 * Does the application expect PROT_READ to imply PROT_EXEC:
 */
if ((prot & PROT_READ) && (current->personality & READ_IMPLIES_EXEC))
    prot |= PROT_EXEC;

vm_flags = calc_vm_prot_bits(prot);

down_write(&current->mm->mmap_sem);

vma = find_vma(current->mm, start);
error = -ENOMEM;
if (!vma)
    goto out;
prev = vma->vm_prev;
if (unlikely(grows & PROT_GROWSDOWN)) {
    if (vma->vm_start >= end)
        goto out;
    start = vma->vm_start;
    error = -EINVAL;
    if (!(vma->vm_flags & VM_GROWSDOWN))
        goto out;
}
else {
    if (vma->vm_start > start)
        goto out;
    if (unlikely(grows & PROT_GROWSUP)) {
        end = vma->vm_end;
        error = -EINVAL;
        if (!(vma->vm_flags & VM_GROWSUP))
            goto out;
    }
}
if (start > vma->vm_start)
    prev = vma;

for (nstart = start ; ; ) {
    unsigned long newflags;

    /* Here we know that vma->vm_start <= nstart < vma->vm_end. */

    newflags = vm_flags | (vma->vm_flags & ~(VM_READ | VM_WRITE |
VM_EXEC));
    /* newflags >> 4 shift VM_MAY% in place of VM_% */
    if ((newflags & ~(newflags >> 4)) & (VM_READ | VM_WRITE | VM_EXEC)) {
        error = -EACCES;
        goto out;
    }

    error = security_file_mprotect(vma, reqprot, prot);
    if (error)
        goto out;

    tmp = vma->vm_end;

```

```
    if (tmp > end)
        tmp = end;
    error = mprotect_fixup(vma, &prev, nstart, tmp, newflags);
    if (error)
        goto out;
    nstart = tmp;

    if (nstart < prev->vm_end)
        nstart = prev->vm_end;
    if (nstart >= end)
        goto out;

    vma = prev->vm_next;
if (!vma || vma->vm_start != nstart) {
    error = -ENOMEM;
    goto out;
}
out:
    up_write(&current->mm->mmap_sem);
    return error;
}
```

## sys\_mmap2

Location of file /arch/sh/kernel/sys\_sh.c

```
asmlinkage long sys_mmap2(unsigned long addr, unsigned long len,
    unsigned long prot, unsigned long flags,
    unsigned long fd, unsigned long pgoff)
{
    printk(KERN_INFO "I am currently in sys_mmap2 system call in
/arch/sh/kernel/sys_sh.c file");
    /*
     * The shift for mmap2 is constant, regardless of PAGE_SIZE
     * setting.
     */
    if (pgoff & ((1 << (PAGE_SHIFT - 12)) - 1))
        return -EINVAL;

    pgoff >= PAGE_SHIFT - 12;

    return sys_mmap_pgoff(addr, len, prot, flags, fd, pgoff);
}
```

## do\_munmap() mm/mmap.c

```
int do_munmap(struct mm_struct *mm, unsigned long start, size_t len)
{
    unsigned long end;
    struct vm_area_struct *vma, *prev, *last;

    if ((start & ~PAGE_MASK) || start > TASK_SIZE || len > TASK_SIZE-start)
        return -EINVAL;

    if ((len = PAGE_ALIGN(len)) == 0)
        return -EINVAL;

    /* Find the first overlapping VMA */
    vma = find_vma(mm, start);
    if (!vma)
        return 0;
    prev = vma->vm_prev;
    /* we have start < vma->vm_end */

    /* if it doesn't overlap, we have nothing.. */
    end = start + len;
    if (vma->vm_start >= end)
        return 0;

    /*
     * If we need to split any vma, do it now to save pain later.
     *
     * Note: mremap's move_vma VM_ACCOUNT handling assumes a partially
     * unmapped vm_area_struct will remain in use: so lower split_vma
     * places tmp vma above, and higher split_vma places tmp vma below.
     */
    if (start > vma->vm_start) {
        int error;
```

```

if (start > vma->vm_start) {
    int error;

    /*
     * Make sure that map_count on return from munmap() will
     * not exceed its limit; but let map_count go just above
     * its limit temporarily, to help free resources as expected.
     */
    if (end < vma->vm_end && mm->map_count >= sysctl_max_map_count)
        return -ENOMEM;

    error = __split_vma(mm, vma, start, 0);
    if (error)
        return error;
    prev = vma;
}

/* Does it split the last one? */
last = find_vma(mm, end);
if (last && end > last->vm_start) {
    int error = __split_vma(mm, last, end, 1);
    if (error)
        return error;
}
vma = prev? prev->vm_next: mm->mmap;

/*
 * unlock any mlock()ed ranges before detaching vmas
 */
if (mm->locked_vm) {
    struct vm_area_struct *tmp = vma;
    while (tmp && tmp->vm_start < end) {
        if (tmp->vm_flags & VM_LOCKED) {
            mm->locked_vm -= vma_pages(tmp);
munlock_vma_pages_all(tmp);
        }
        tmp = tmp->vm_next;
    }
}

/*
 * Remove the vma's, and unmap the actual pages
 */
detach_vmas_to_be_unmapped(mm, vma, prev, end);
unmap_region(mm, vma, prev, start, end);

/* Fix up all other VM information */
remove_vma_list(mm, vma);

return 0;
}

int vm_munmap(unsigned long start, size_t len)
{
    int ret;
}

```

```

        struct mm_struct *mm = current->mm;

        down_write(&mm->mmap_sem);
        ret = do_munmap(mm, start, len);
        up_write(&mm->mmap_sem);
        return ret;
    }

EXPORT_SYMBOL(vm_munmap);
/* Does it split the last one? */
last = find_vma(mm, end);
if (last && end > last->vm_start) {
    int error = __split_vma(mm, last, end, 1);
    if (error)
        return error;
}
vma = prev? prev->vm_next: mm->mmap;

/*
 * unlock any mlock()ed ranges before detaching vmas
 */
if (mm->locked_vm) {
    struct vm_area_struct *tmp = vma;
    while (tmp && tmp->vm_start < end) {
        if (tmp->vm_flags & VM_LOCKED) {
            mm->locked_vm -= vma_pages(tmp);
munlock_vma_pages_all(tmp);
        }
        tmp = tmp->vm_next;
    }
}

/*
 * Remove the vma's, and unmap the actual pages
 */
detach_vmas_to_be_unmapped(mm, vma, prev, end);
unmap_region(mm, vma, prev, start, end);

/* Fix up all other VM information */
remove_vma_list(mm, vma);

```

```

    return 0;
}

int vm_munmap(unsigned long start, size_t len)
{
    int ret;
    struct mm_struct *mm = current->mm;

    down_write(&mm->mmap_sem);
    ret = do_munmap(mm, start, len);
    up_write(&mm->mmap_sem);
    return ret;
}
```

```
EXPORT_SYMBOL(vm_munmap);
```