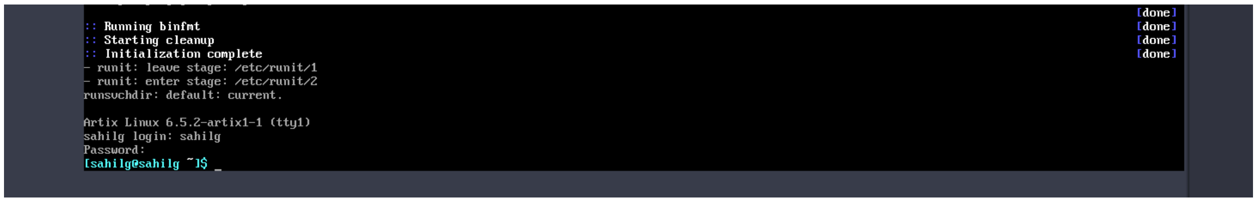
OS-Assignment-2(write-up)

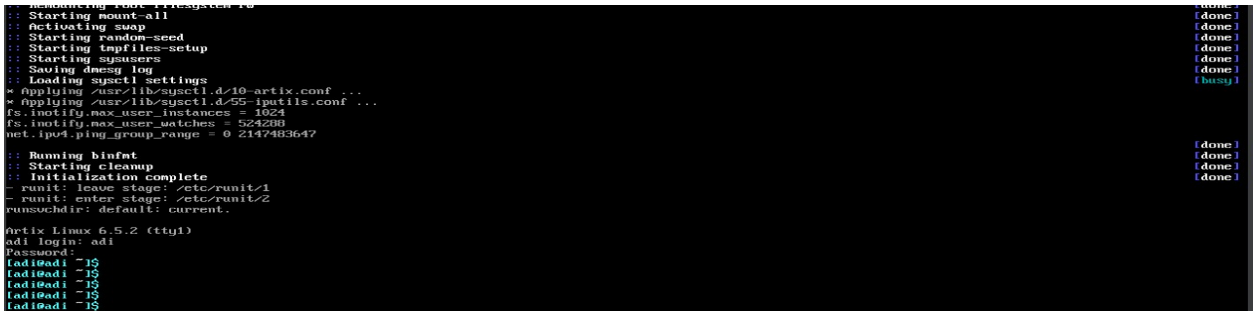
Ans-1)

For the first question, we have made installed artix on oracle virtualbox and have suitable partitions in it. I have made a new user in it named sahilg. On the device of other partner the user is named as adi. The following are the screenshots of the artix login.



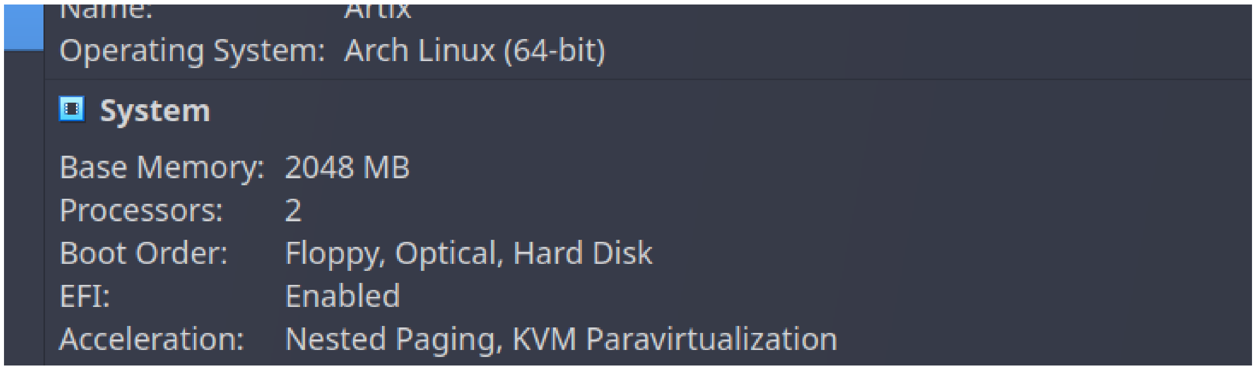
```
                                                                      [done]
:: Running binfmt                                                     [done]
:: Starting cleanup                                                   [done]
:: Initialization complete                                            [done]
- runit: leave stage: /etc/runit/1
- runit: enter stage: /etc/runit/2
runsvchdir: default: current.

Artix Linux 6.5.2-artix1-1 (tty1)
sahilg login: sahilg
Password:
[sahilg@sahilg ~]$ _
```

And



```
:: Starting mount-all                                                 [done]
:: Activating swap                                                    [done]
:: Starting random-seed                                               [done]
:: Starting tmpfiles-setup                                            [done]
:: Starting sysusers                                                  [done]
:: Saving dmesg log                                                   [done]
:: Loading sysctl settings                                            [busy]
* Applying /usr/lib/sysctl.d/10-artix.conf ...
* Applying /usr/lib/sysctl.d/55-iputils.conf ...
fs.inotify.max_user_instances = 1024
fs.inotify.max_user_watches = 524288
net.ipv4.ping_group_range = 0 2147483647
                                                                      [done]
:: Running binfmt                                                     [done]
:: Starting cleanup                                                   [done]
:: Initialization complete                                            [done]
- runit: leave stage: /etc/runit/1
- runit: enter stage: /etc/runit/2
runsvchdir: default: current.

Artix Linux 6.5.2 (tty1)
adi login: adi
Password:
[adi@adi ~]$
[adi@adi ~]$
[adi@adi ~]$
[adi@adi ~]$
[adi@adi ~]$
```

also all the arch repositories are enabled and necessary libraries like binutils, elfutils, gcc, gdb, make, automake,autoconf, yasm and vim are installed . I have given 20 GB of hard drive space and have used approx 4GB RAM and 2 virtual CPUs and efi enabled.

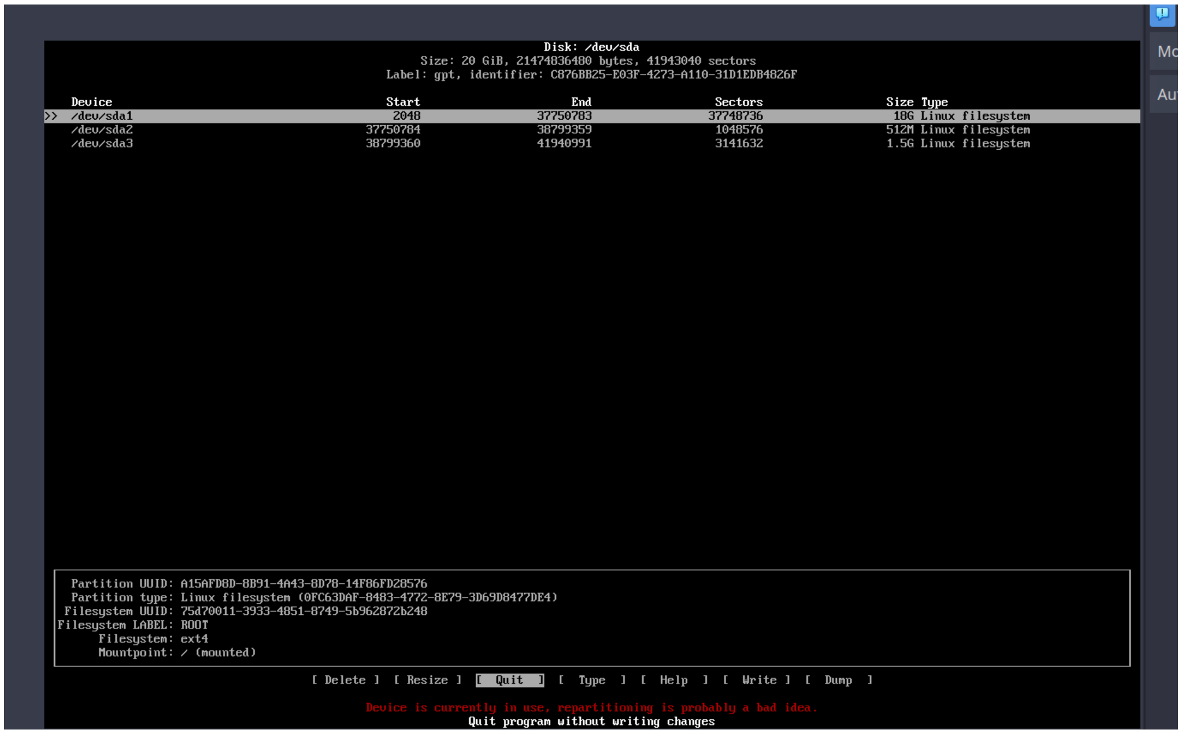


Name: Artix
Operating System: Arch Linux (64-bit)

System

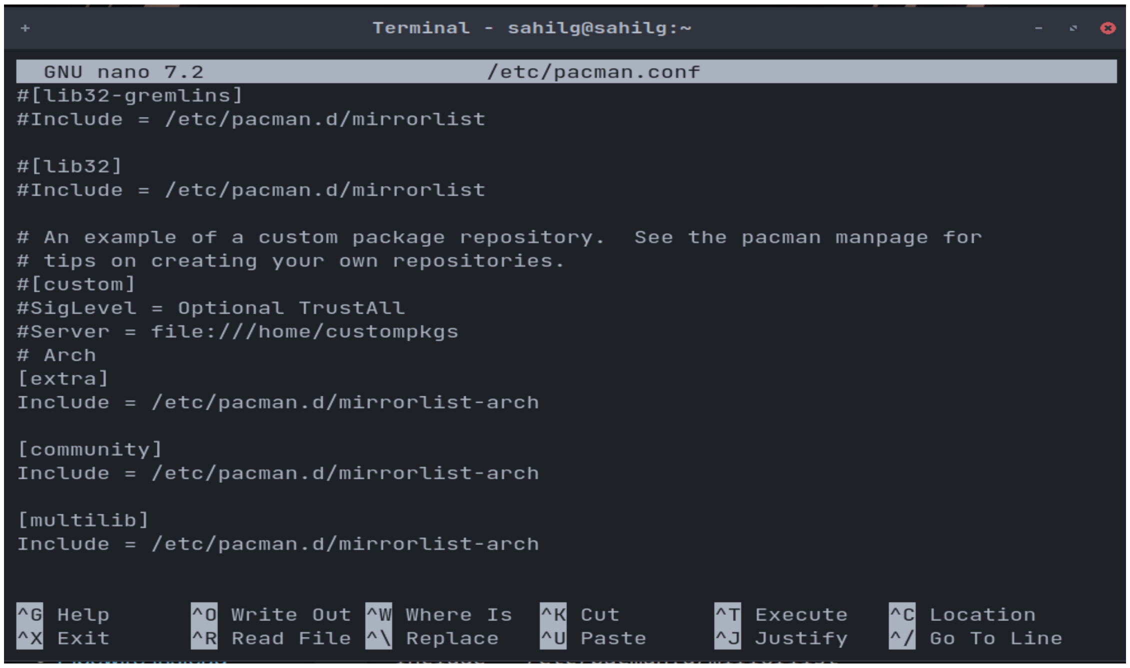
Base Memory: 2048 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
EFI: Enabled
Acceleration: Nested Paging, KVM Paravirtualization

The partition table created is as follows :



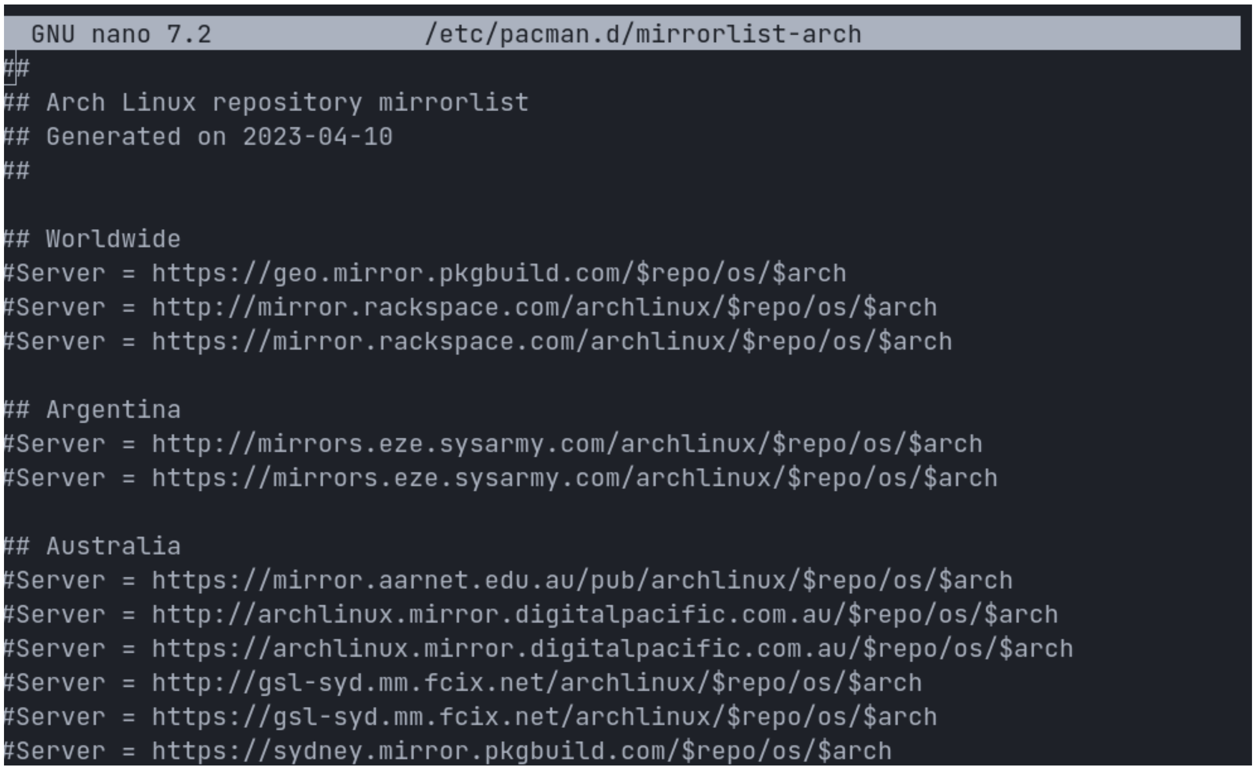
```
                                    Disk: /dev/sda
                   Size: 20 GiB, 21474836480 bytes, 41943040 sectors
               Label: gpt, identifier: C876BB25-E03F-4273-A110-31D1EDB4826F

    Device                Start          End        Sectors      Size Type
>>  /dev/sda1              2048     37750783       37748736       18G Linux filesystem
    /dev/sda2          37750784     38799359        1048576      512M Linux filesystem
    /dev/sda3          38799360     41940991        3141632      1.5G Linux filesystem

  Partition UUID: A15AFD8D-8B91-4A43-8D78-14F86FD28576
  Partition type: Linux filesystem (0FC63DAF-8483-4772-8E79-3D69D8477DE4)
 Filesystem UUID: 75d70011-3933-4851-8749-5b962872b248
Filesystem LABEL: ROOT
      Filesystem: ext4
      Mountpoint: / (mounted)

        [ Delete ]  [ Resize ]  [  Quit  ]  [  Type  ]  [  Help  ]  [  Write ]  [  Dump  ]

               Device is currently in use, repartitioning is probably a bad idea.
                           Quit program without writing changes
```

In this we can see the partitioning of the sda 1 , sda 2 and sda3 .
Enabling Arch repositories:



```
Terminal - sahilg@sahilg:~

  GNU nano 7.2                          /etc/pacman.conf
#[lib32-gremlins]
#Include = /etc/pacman.d/mirrorlist

#[lib32]
#Include = /etc/pacman.d/mirrorlist

# An example of a custom package repository.  See the pacman manpage for
# tips on creating your own repositories.
#[custom]
#SigLevel = Optional TrustAll
#Server = file:///home/custompkgs
# Arch
[extra]
Include = /etc/pacman.d/mirrorlist-arch

[community]
Include = /etc/pacman.d/mirrorlist-arch

[multilib]
Include = /etc/pacman.d/mirrorlist-arch

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```

And we added the file /etc/pacman.d/mirrorlist-arch having server of all countries including India
.

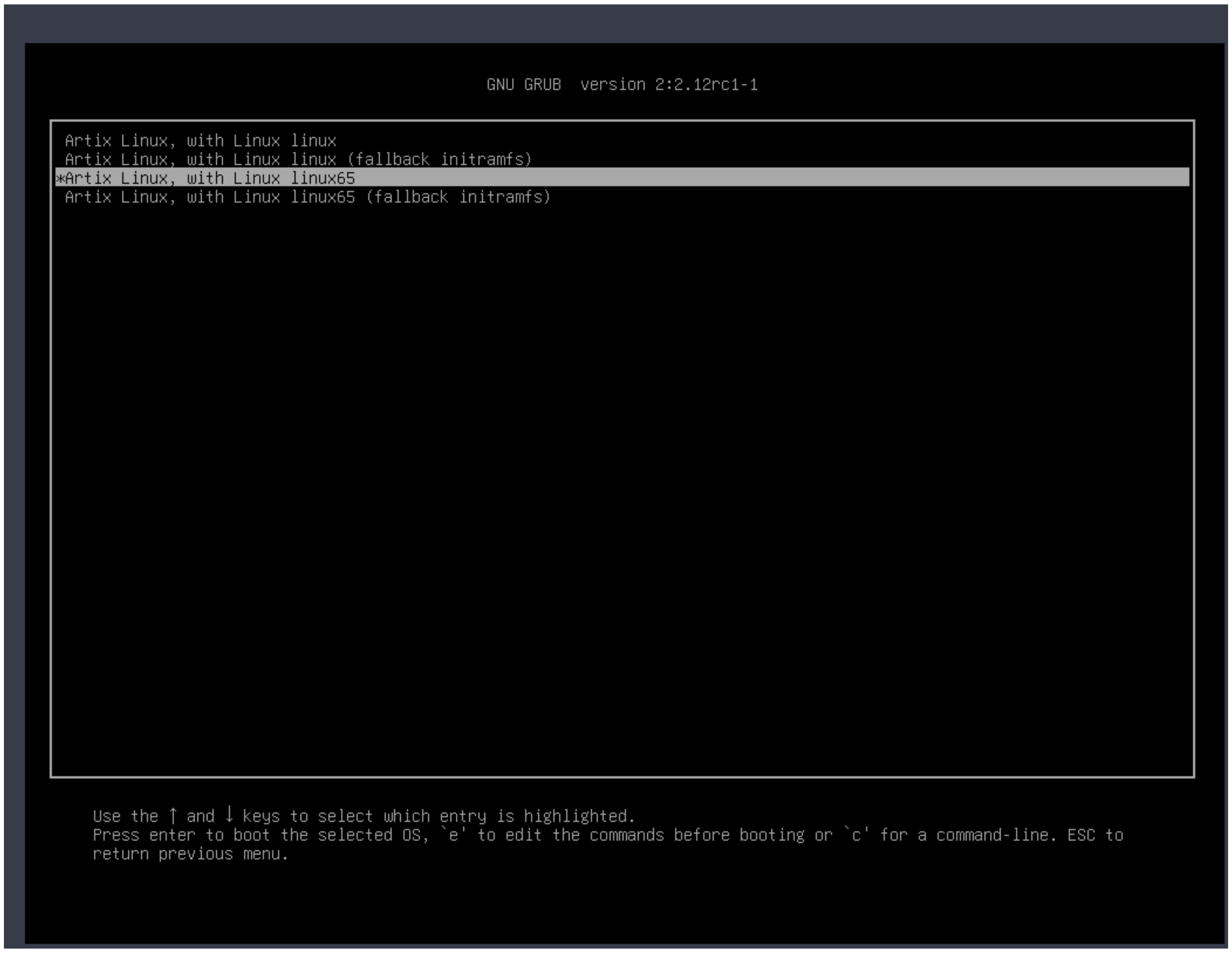


```
  GNU nano 7.2              /etc/pacman.d/mirrorlist-arch
##
## Arch Linux repository mirrorlist
## Generated on 2023-04-10
##

## Worldwide
#Server = https://geo.mirror.pkgbuild.com/$repo/os/$arch
#Server = http://mirror.rackspace.com/archlinux/$repo/os/$arch
#Server = https://mirror.rackspace.com/archlinux/$repo/os/$arch

## Argentina
#Server = http://mirrors.eze.sysarmy.com/archlinux/$repo/os/$arch
#Server = https://mirrors.eze.sysarmy.com/archlinux/$repo/os/$arch

## Australia
#Server = https://mirror.aarnet.edu.au/pub/archlinux/$repo/os/$arch
#Server = http://archlinux.mirror.digitalpacific.com.au/$repo/os/$arch
#Server = https://archlinux.mirror.digitalpacific.com.au/$repo/os/$arch
#Server = http://gsl-syd.mm.fcix.net/archlinux/$repo/os/$arch
#Server = https://gsl-syd.mm.fcix.net/archlinux/$repo/os/$arch
#Server = https://sydney.mirror.pkgbuild.com/$repo/os/$arch
```

Now we needed to compile the kernel , that is any stable kernel from https://www.kernel.org/
We have compiled linux 6.5.2 stable version.

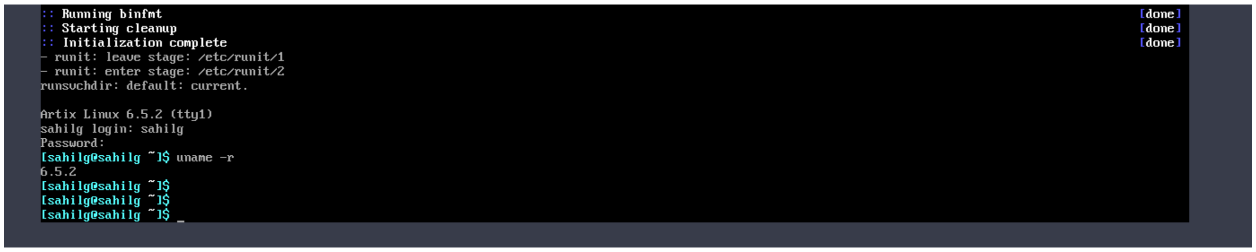
We have followed the step of the arch wiki
https://wiki.archlinux.org/title/Kernel/Traditional_compilation

After compiling the kernel with the following procedure and setting grub for linux65, we get the following result on rebooting the system



```
                        GNU GRUB  version 2:2.12rc1-1

 Artix Linux, with Linux linux
 Artix Linux, with Linux linux (fallback initramfs)
*Artix Linux, with Linux linux65
 Artix Linux, with Linux linux65 (fallback initramfs)

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands before booting or `c' for a command-line. ESC to
   return previous menu.
```

Going in the Arch Linux , with Linux linux65 , we run the stock kernel which is now running (compiled) .
We can check the kernel version using uname -r .



```
:: Running binfmt                                                              [done]
:: Starting cleanup                                                            [done]
:: Initialization complete                                                     [done]
- runit: leave stage: /etc/runit/1
- runit: enter stage: /etc/runit/2
runsvchdir: default: current.

Artix Linux 6.5.2 (tty1)
sahilg login: sahilg
Password:
[sahilg@sahilg ~]$ uname -r
6.5.2
[sahilg@sahilg ~]$
[sahilg@sahilg ~]$
[sahilg@sahilg ~]$ _
```

So , artix is installed and we compiled the stable kernel in it.

Ans-2)

For this question we needed to make three processes using fork such that we do the scheduling inside them and they run as per that scheduling policy and execute a file which counts the numbers from 1 to 2^32 using execl family system calls.
Then we need to calculate the time to see how the cpu is deciding which scheduling policy to run first. To see the ordering of the scheduling priorities , we can use this method .
So , first we make the count.c file which counts from 1 to 2^32 . This program can be called in the main scheduler.c using execvp(part of execl family).
The count.c file is

```
#include <stdio.h>
#include <math.h>
int main(){
   long int n = 0 ;
   for (long i = 0 ; i< pow(2, 32); i++){
       n++;
       //printf("%ld\n",n);
   }
   //printf("%ld\n",n);
}
```

This increasing the value of n till 2^32 so this would help in seeing which scheduling policy runs the c file in least time and which takes most time based on the priority.

So , in the scheduler.c file we first include all the necessary libraries that will be used while doing the schedule processing in the child processes.

```
#include <bits/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sched.h>
#include <unistd.h>
#include <sys/types.h>
#include <time.h>
```

I used the print_timestamp function earlier just to debug the code how it was behaving as per the scheduling policies and priorities of the policies.

```
void print_timestamp(const char* label) {
    struct timespec timestamp;
    clock_gettime(CLOCK_REALTIME, &timestamp);
    printf("%s: %ld.%09ld\n", label, timestamp.tv_sec, timestamp.tv_nsec);
}
```

After this we go in the int main() to use fork and make the 3 child and schedule them using the sched_schduler function which takes the input (arguments) as pid policy and param.
Also we use the start , end and time_for_scheduling array to store the time we get for each process and then finally print the time each process took .
We call the clock_gettime just before the fork so that we get the starting time of each process .
The code for the same goes like:

```
int priority_other;
int priority_rr ;
int priority_fifo;
struct timespec start[3], end[3], time_for_scheduling[3];

int main(int argc, char *argv[]) {
    priority_other = 0;
    priority_rr = 50;
    priority_fifo = 50;
    int status = 0;
 //  pid_t ret;
    pid_t rc[3];
    for (int i = 0; i < 3; i++) {
        clock_gettime(CLOCK_REALTIME, &start[i]);
        rc[i] = fork();
        //ret = rc[i];
        if (rc[i] < 0) {
            perror("fork failed");
            exit(1);
        }
        else if (rc[i] == 0) {
            if (i == 0) {
                struct sched_param parameter_A;
                parameter_A.sched_priority = priority_other;
                nice(0);
                sched_setscheduler(rc[i], SCHED_OTHER, &parameter_A);
                perror("scheduling1");
              //  printf("child 1 started\n");
                //ret = rc[i];
            //    clock_gettime(CLOCK_REALTIME, &start[i]);
                execvp("./count", NULL);
                perror("execl failed");

            }
            else if (i == 1) {
                struct sched_param parameter_B;
                parameter_B.sched_priority = priority_rr;
                sched_setscheduler(rc[i], SCHED_RR, &parameter_B);
                perror("scheduling2");
               // printf("child 2 started\n");
                //ret = rc[i];
            //    clock_gettime(CLOCK_REALTIME, &start[i]);
                execvp("./count", NULL);
                perror("execl failed");
            }
            else if (i == 2) {
                struct sched_param parameter_C;
                parameter_C.sched_priority = priority_fifo;
                sched_setscheduler(rc[i], SCHED_FIFO, &parameter_C);
                perror("scheduling3");
               // printf("child 3 started\n");
                //ret = rc[i];
            //    clock_gettime(CLOCK_REALTIME, &start[i]);
                execvp("./count", NULL);
                perror("execl failed");
            }
          //  struct timespec child_start;
        //    clock_gettime(CLOCK_REALTIME, &child_start);
            //printf("Child number:%d - ", i);
            //print_timestamp("Child started");
            exit(1);
        }
    }
```

Here we have set the priority of sched_other as 0 , which is 0 as per the linux fair scheduling policy and we set the priority of sched_rr as 50 and

sched_fifo as 50 as instructed. (we can set priorities of sched_rr and sched_fifo from 1 to 99 but it is said to take it as 50 on slack channel). After that we make the parent wait for the process the cpu decided to as per the scheduling policy by comaparing the pid we get from `waitpid(-1, &status, 0);` and the one we stored in the array `pid_t rc[3];` .
So, after comparing the pid we store the end time of the process in that array at that index.

```
   for (int i = 0; i<3;i++) {
       pid_t wc = waitpid(-1, &status, 0);
       //printf("%d\n",wc);
       for(int k = 0 ; k<3; k++){
           if(wc == rc[k]){
               clock_gettime(CLOCK_REALTIME, &end[k]);
           }
       }
  //     clock_gettime(CLOCK_REALTIME, &end[i]);
   }
```

Now we have the arrays `start[3], end[3]` in which we have the starting time and the end time of the 3 processes. So we can now calculate time it took to run the 3 processes by making the struct duration in a for loop which gives the time using duration.tv_sec - duration.tv_nsec and if duration.tv_nsec<0 we decrease duration.sec with 1 and add 1000000000 to it to get the correct time for each process .
The code for the same goes like :

```
for (int i = 0;i<3;i++) {
       struct timespec duration = {
           end[i].tv_sec - start[i].tv_sec,
           end[i].tv_nsec - start[i].tv_nsec
       };
       if(duration.tv_nsec < 0) {
           duration.tv_sec -= 1;
           duration.tv_nsec += 1000000000;
       }
       if(i == 0) {
           printf("SCHED_OTHER took time: %ld.%09ld\n", duration.tv_sec,
duration.tv_nsec);

       }
       else if (i == 1) {
           printf("SCHED_RR took time: %ld.%09ld\n", duration.tv_sec,
duration.tv_nsec);
       }
       else if (i == 2) {
           printf("SCHED_FIFO took time: %ld.%09ld\n", duration.tv_sec,
duration.tv_nsec);
       }
       time_for_scheduling[i] = duration;
   }
```

At the end we write our findings in the file time.txt in which we write the time of other first then the time of rr and then the time of fifo . We open the file using FILE *ptr and then use fopen to overwrite in the file time.txt and thus opening it with the mode "w".
Then in the for loop we write the fie using the time_for_scheduing array. So, we write it using fprintf function in which the first argument is is ptr .second argument is "%ld.%09ld" , third as time_for_scheduling[i].tv_sec and last argument as time_for_scheduling.tv_nsec . The code for the same goes like:

```
    FILE *ptr;
    ptr = fopen("time.txt", "w");
    if (ptr == NULL) {
        perror("fopen failed");
        exit(1);
    }
  // fprintf(ptr, "%d ",priority_other);

  // fprintf(ptr, "%d ",priority_rr);

  // fprintf(ptr, "%d ",priority_fifo);
    for (int i = 0; i < 3; i++) {
        fprintf(ptr, "%ld.%09ld ", time_for_scheduling[i].tv_sec,
time_for_scheduling[i].tv_nsec);
    }
```

At last , we call close this file and then call the python file in which we have made our histogram for the time for each scheduling policy .

```
    fprintf(ptr, "\n");
    fclose(ptr);
    char *args[] = {"python3", "scheduler.py", NULL};
    execvp("python3", args);
    perror("execvp failed");
```

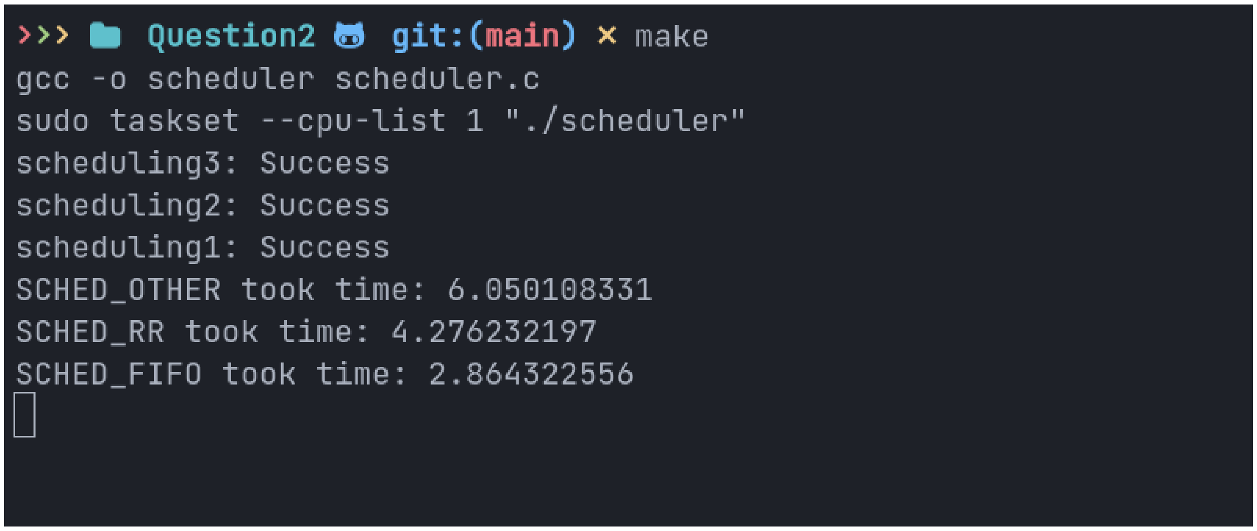
```
   exit(1);
   return 0;
}
```

Now , to make the histogram using python we can read the time.txt file and use matplotlib and numpy to get .barh function which creates the histogram for x axis as time and y axis as scheduling policy .
We set the colors of the scheduling policies as blue for Other , green for RR and red for Fifo .

The python code goes like this:

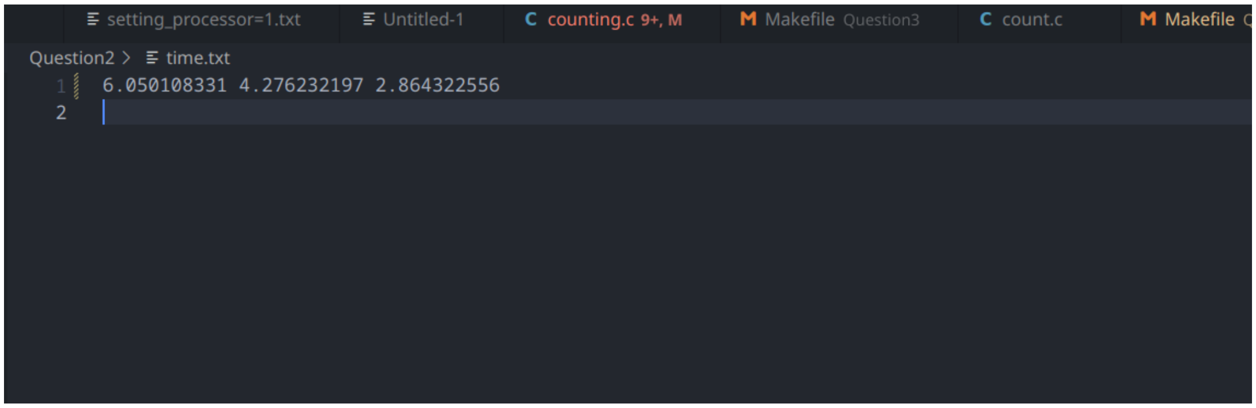
```
import matplotlib.pyplot as plt
import numpy as np
with open("time.txt", "r") as f:
   for x in f :
       y=x.split()
data=[]
for i in y:
   data.append(float(i))
labels = ['Other', 'RR', 'Fifo']
colors = ['blue', 'green', 'red']
y = np.arange(0, len(data))
bar_width = 1
bars = plt.barh(y, data, height=bar_width, color=colors, alpha=0.7,
tick_label=labels)
plt.xlabel('Time Taken')
plt.ylabel('Scheduling Policy')
plt.title('Histogram for scheduling policies')
x_ticks = np.arange(0, max(data) + 0.5, 0.5)
plt.xticks(x_ticks)
legend_labels = [plt.Line2D([0], [0], color=color, lw=4, label=label) for
color, label in zip(colors, labels)]
plt.legend(handles=legend_labels)
plt.show()
```

So when we run the make we get the following as the result:



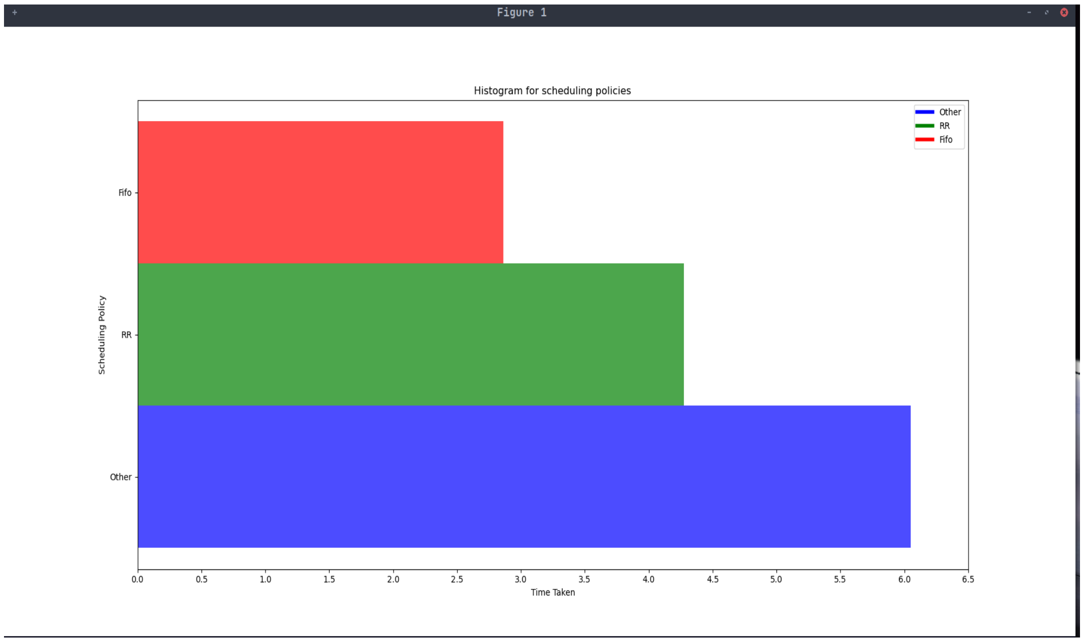
```
>>> Question2 git:(main) ✗ make
gcc -o scheduler scheduler.c
sudo taskset --cpu-list 1 "./scheduler"
scheduling3: Success
scheduling2: Success
scheduling1: Success
SCHED_OTHER took time: 6.050108331
SCHED_RR took time: 4.276232197
SCHED_FIFO took time: 2.864322556
```

The time.txt file contains

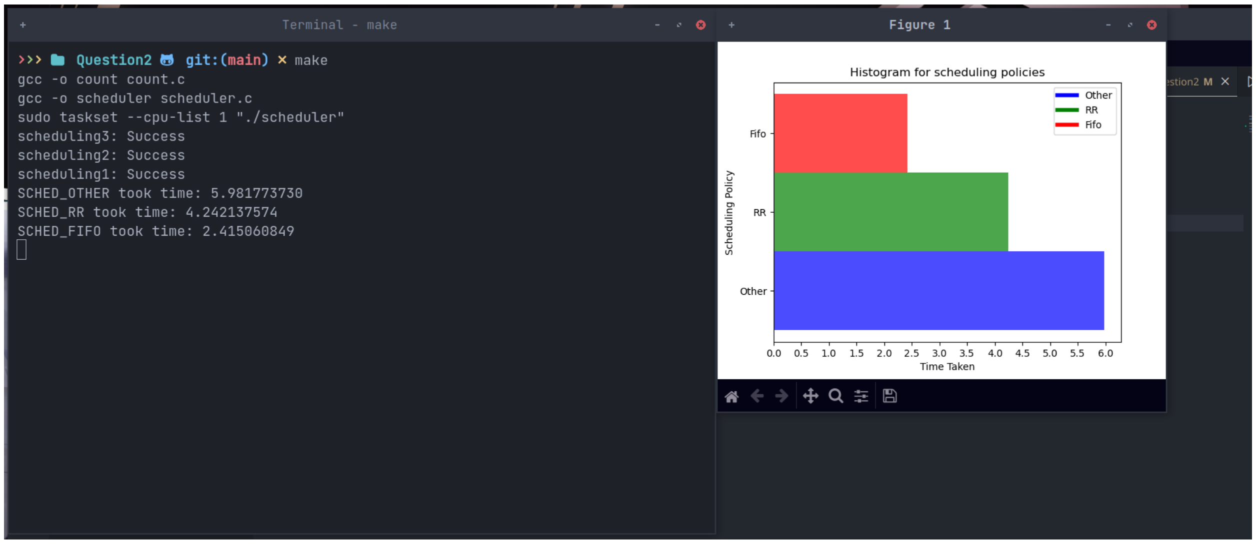


```
6.050108331 4.276232197 2.864322556
```

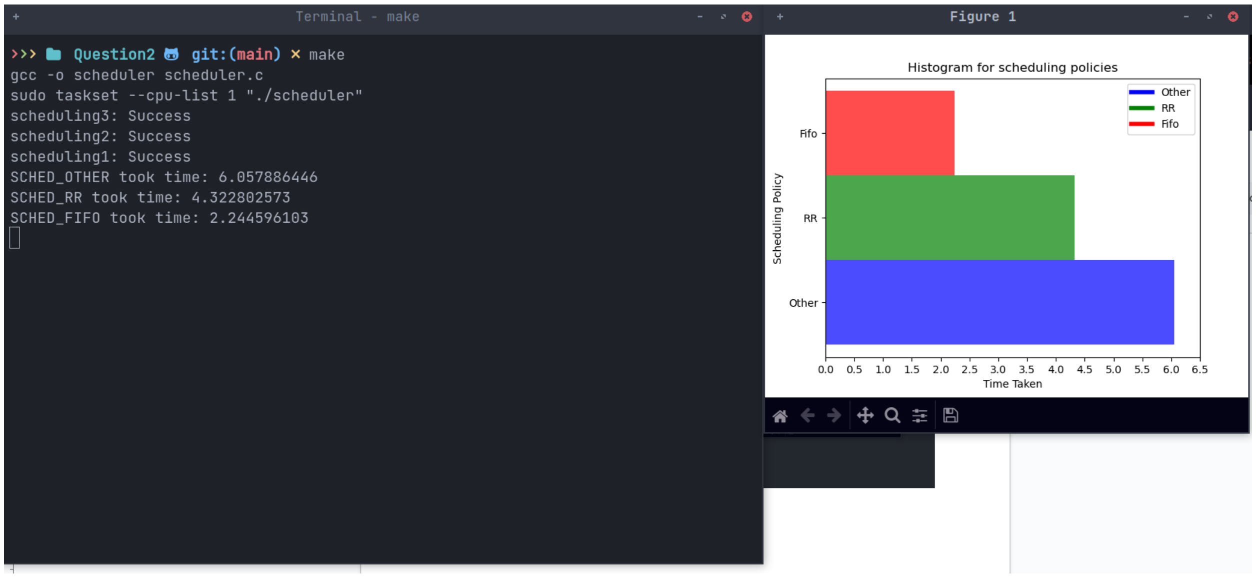
The times of other , rr and fifo respectively .
And we get the histogram as follows :



Doing it multiple times fetches us the same result everytime , that is fifo takes the least time, then rr and then other .
For multiple times the results are :



```
>>> Question2 git:(main) ✗ make
gcc -o count count.c
gcc -o scheduler scheduler.c
sudo taskset --cpu-list 1 "./scheduler"
scheduling3: Success
scheduling2: Success
scheduling1: Success
SCHED_OTHER took time: 5.981773730
SCHED_RR took time: 4.242137574
SCHED_FIFO took time: 2.415060849
```



```
>>> Question2 git:(main) ✗ make
gcc -o scheduler scheduler.c
sudo taskset --cpu-list 1 "./scheduler"
scheduling3: Success
scheduling2: Success
scheduling1: Success
SCHED_OTHER took time: 6.057886446
SCHED_RR took time: 4.322802573
SCHED_FIFO took time: 2.244596103
```

Hence we get the same order every time we run the code that is fifo takes least time , then rr and then at last other.

At last the makefile for this goes like :

```
all:count scheduler

count:count.c
    gcc -o count count.c
```

```
scheduler:scheduler.c
   gcc -o scheduler scheduler.c
   sudo taskset --cpu-list 1 "./scheduler"

clean:
   rm -f count
   rm -f scheduler
```

We have done all the error handling like

```
       if (rc[i] < 0) {
           perror("fork failed");
           exit(1);
       }
```

```
               struct sched_param parameter_A;
               parameter_A.sched_priority = priority_other;
               nice(0);
               sched_setscheduler(rc[i], SCHED_OTHER, &parameter_A);
               perror("scheduling1");
             //  printf("child 1 started\n");
               //ret = rc[i];
           //    clock_gettime(CLOCK_REALTIME, &start[i]);
               execvp("./count", NULL);
               perror("execl failed");
```

```
   ptr = fopen("time.txt", "w");
   if (ptr == NULL) {
       perror("fopen failed");
       exit(1);
   }
```

etc …

So, this bench marks the scheduling policies in which cpu decided to give more cpu time to fifo then rr and then other .

<u>Conclusion :</u>

So, as the priority of fifo and rr is 50 which is greater than 0 , they take lesser time than other and among fifo and rr , fifo is taking lesser time because it is favored by the cpu when their is same priority of both rr and fifo .

Ans-3

So, in this question we need to make our own module such that it acts as system call to read the entries of the process task_struct and print the number of currently running processes.
So, to proceed we first need to use the necessary header files .
The following are the necessary header files used:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>
#include <linux/syscalls.h>
#include <linux/list.h>
#include <linux/signal.h>
```

Then we set up the MODULE_LICENSE(“GPL”) and MODULE_AUTHOR(“Aditya and Sahil”) and MODULE_DESCRIPTION(“module counting number of running processes”).
This would mark the important details about the module .

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Aditya and Sahil");
MODULE_DESCRIPTION("module counting number of running processes.");
```

Then we make the function which counts the number of running processes as follows :
We name the function as counting_processes which returns a long type which take struct task_struct and count the number of of task using for_each_process(task) which iterates through all the processes and then checks the condition from if(task_is_running) and we have made a count that adds when this condition is fulfilled . At the end we return the count .
The function’s code part is :

```
asmlinkage long counting_process(void) {
   struct task_struct *task;
   long count = 0;
   for_each_process(task) {
      // printk(KERN_INFO "%d",task->__state);
       // task->__state == 0
       if (task_is_running(task)){
           //printk(task->comm);
           count++;
      // printk("HI\n");
       }
   }
   return count;
}
```

Then we make the two functions static int __init display_init(void) and static void __exit display _cleanup(void) which helps in loading and unloading the module when sudo insmod try.ko(it loads) is called and when sudo rmmod try.ko(it unloads) is called .

So, in the first function static int __init display_init(void) :
It takes an long k and then call the function counting_processes to assign it the desired value .
Then finally we print the value of k which is the number of running processes .
The function of __init display_init(void) is :

```
static int __init display_init(void) {
   long k;
   printk(KERN_INFO "Module loaded ... \n");
   k=counting_process();
   printk(KERN_INFO "The number of running processes are: %ld\n",k);
   return 0;
}
```

Then similarly for __exit display_cleanup , we simply print the statement that the Module is unloaded .
The function of __exit display_cleanup is :

```
static void __exit display_cleanup(void) {
   printk(KERN_INFO "Module Unloaded ... \n");
   // Unregister the custom system call
   // sys_call_table[__NR_count_processes] = NULL;
}
```

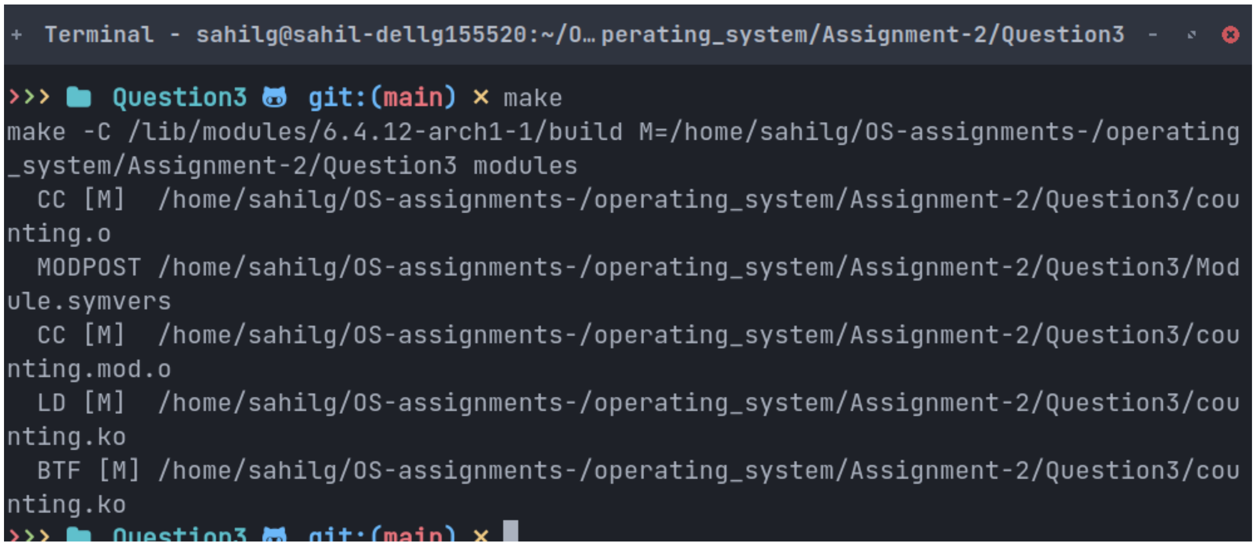
At the end we call module_init in which we give argument display_init as display_init is acting as the loader and then we call module_exit in which we give the argument display_cleanup as it acts as unloader .

NOTE- We have taken the format from the tut -3 .
Now in the Makefile of this question we use `CONFIG_MODULE_SIG = n` which means that the modules can be loaded without requiring a valid digital signature .
Now we add the .o file of our c file in obj-m using += operator .
Then in the all :
      We give `make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules`
      And in clean we give `make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean`
Hence Makfile goes like :

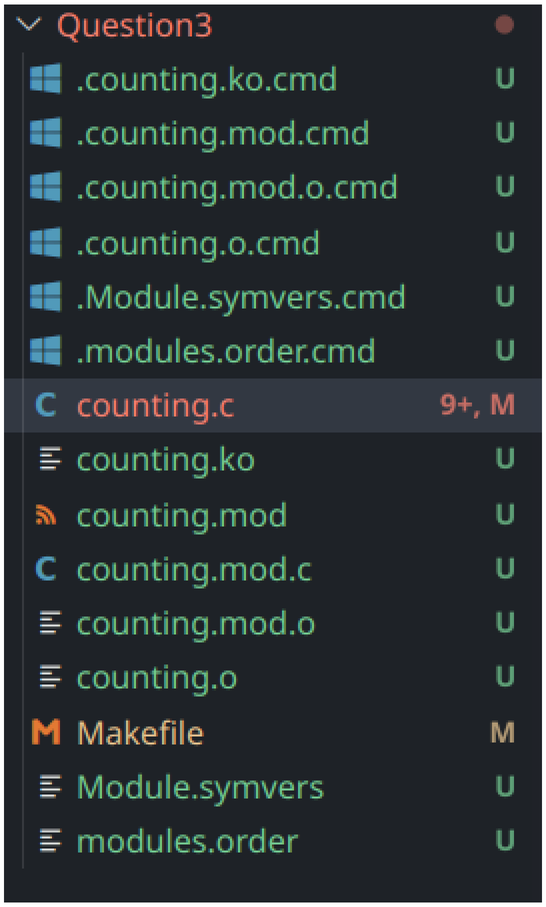
```
CONFIG_MODULE_SIG = n
obj-m+=counting.o
```

```
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Now , doing make we get all the files :

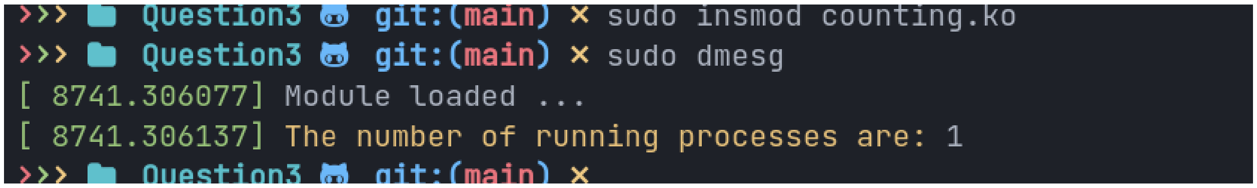


```
+ Terminal - sahilg@sahil-dellg155520:~/O… perating_system/Assignment-2/Question3
>>> Question3 git:(main) ✗ make
make -C /lib/modules/6.4.12-arch1-1/build M=/home/sahilg/OS-assignments-/operating
_system/Assignment-2/Question3 modules
  CC [M]  /home/sahilg/OS-assignments-/operating_system/Assignment-2/Question3/cou
nting.o
  MODPOST /home/sahilg/OS-assignments-/operating_system/Assignment-2/Question3/Mod
ule.symvers
  CC [M]  /home/sahilg/OS-assignments-/operating_system/Assignment-2/Question3/cou
nting.mod.o
  LD [M]  /home/sahilg/OS-assignments-/operating_system/Assignment-2/Question3/cou
nting.ko
  BTF [M] /home/sahilg/OS-assignments-/operating_system/Assignment-2/Question3/cou
nting.ko
>>> Question3 git:(main) ✗
```



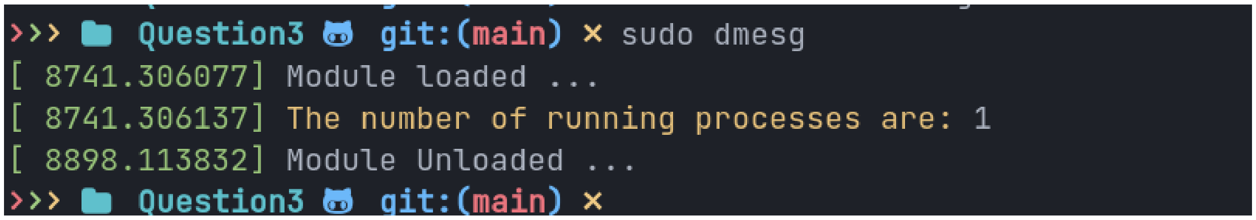
```
Question3
  .counting.ko.cmd        U
  .counting.mod.cmd       U
  .counting.mod.o.cmd     U
  .counting.o.cmd         U
  .Module.symvers.cmd     U
  .modules.order.cmd      U
  counting.c          9+, M
  counting.ko             U
  counting.mod            U
  counting.mod.c          U
  counting.mod.o          U
  counting.o              U
  Makefile                M
  Module.symvers          U
  modules.order           U
```

Now loading the module and getting the message from the loaded module uing dmesg we get,



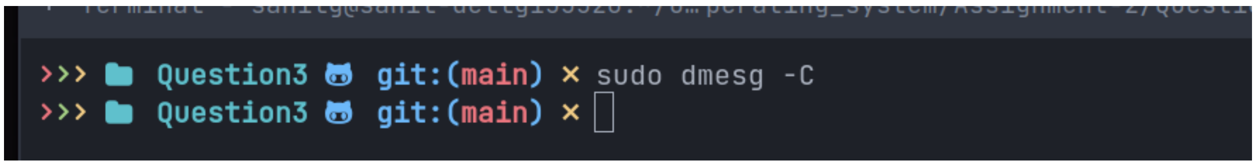
```
>>> Question3 git:(main) ✕ sudo insmod counting.ko
>>> Question3 git:(main) ✕ sudo dmesg
[ 8741.306077] Module loaded ...
[ 8741.306137] The number of running processes are: 1
>>> Question3 git:(main) ✕
```

Now unloading the module and printing the message we get ,



```
>>> Question3 git:(main) ✕ sudo dmesg
[ 8741.306077] Module loaded ...
[ 8741.306137] The number of running processes are: 1
[ 8898.113832] Module Unloaded ...
>>> Question3 git:(main) ✕
```

Now we can clear the message by sudo dmseg -C



```
>>> Question3 git:(main) ✕ sudo dmesg -C
>>> Question3 git:(main) ✕
```

So, this only when module is loaded .
Finally we can clean the files using make clean .

END OF DOCUMENT