CS307 Assignment-2 Report

Instructor: Dr. Aditya Nigam

Submitted By-

1. Ashutosh Sharma - B18010
2. Anuj Goel - B18161
3. Om Pandey - B18182

Q1. Custom Linux Kernel

OS Used: Lubuntu 20.04 Virtual Machine on Virtual Box

I tried to build and install the 5.10.22 version of the kernel. The earlier size and the version of the default kernel are shown in the picture. The default kernel provided in the 20.04 version of Lubuntu is 5.4.

Initial version of Kernel in Lubuntu 20.04:

```
5.4.0-56-generic
```

I used the menuconfig command to select modules to be compiled in the custom kernel. After installing the kernel version:

Removed modules to decrease the size of Kernel:

1. **Bluetooth Module**
2. **Wireless Module**
3. **USB support**
4. **Extra file systems**
5. **Support for Amd architecture**
6. **Android drivers**
7. **Gamesports**
8. **Touch Screen**
9. **Debug Statements**
10. **Remote control support etc.**

The size of the original kernel is around 207 MB.
Reduced size of the kernel around 70 MB.

Writing modules for a kernel:

Libraries used:

1. linux/module.h
2. linux/kernel.h
3. linux/init.h

A Simple Module:
When this module is entered into the kernel it only prints `Loading test module.` in the kernel log and `Unloading test module.` in the kernel log when it exits.

Code for simple module.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_LICENSE("No License");
MODULE_AUTHOR("Ashutosh Sharma");
MODULE_DESCRIPTION("Crash Test");
MODULE_VERSION("0.1");
static int __init start(void)
{
    printk(KERN_INFO "Loading test module.\n");
    return 0;
}

static void __exit end(void)
{
```

```
    printk(KERN_INFO "Unloading test module.\n");
}
module_init(start);
module_exit(end);
```

Kernel log for simple module.

```
Mar 11 09:41:15 osboxes kernel: [ 1777.261644] Loading test module.
Mar 11 09:41:23 osboxes kernel: [ 1785.010335] Unloading test module.
```

Crash tests on the kernel:

Test 1: Divide by 0.

This test was unsuccessful and the kernel worked fine. As a proof we can see the kernel log in the photo below the program code.

Code for dividing by 0.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_LICENSE("No License");
MODULE_AUTHOR("Ashutosh Sharma");
MODULE_DESCRIPTION("Crash Test");
MODULE_VERSION("0.1");
static int __init start(void)
{
    printk(KERN_INFO "Loading test module.\n");
      int x=0,y=1;
      y/=x;
      return 0;
}

static void __exit end(void)
{
    printk(KERN_INFO "Unloading test module.\n");
}

module_init(start);
module_exit(end);
```

Kernel log when divided by 0 module is entered into the kernel.

```
Mar 11 10:01:07 osboxes kernel: [ 2973.962268] Loading test module.
Mar 11 10:01:58 osboxes kernel: [ 3025.766203] Unloading test module.
```

Test 2: Returning 1 from in_it mod function:

In the second test I tried to return 1 instead of 0 from the in_it mod function of the module. This

test was successful and we can see the kernel log below.

Code for returning 1 from in_it mod function from the module.

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_LICENSE("No License");
MODULE_AUTHOR("Ashutosh Sharma");
MODULE_DESCRIPTION("Crash Test");
MODULE_VERSION("0.1");
static int __init start(void)
{
    printk(KERN_INFO "Loading test module.\n");
    return 1;
}

static void __exit end(void)
{
    printk(KERN_INFO "Unloading test module.\n");
}

module_init(start);
module_exit(end);
```

Kernel log when module with return 1 from in_it mod enters the kernel.

```
Mar 11 10:05:01 osboxes kernel: [ 3209.061392]  entry_SYSCALL_64_after_hwfram
e+0x44/0xa9
Mar 11 10:05:01 osboxes kernel: [ 3209.061395] RIP: 0033:0x7fc578d3f89d
Mar 11 10:05:01 osboxes kernel: [ 3209.061399] Code: 00 c3 66 2e 0f 1f 84 00
00 00 00 00 90 f3 0f 1e fa 48 89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4d 89
 c8 4c 8b 4c 24 08 0f 05 <48> 3d 01 f0 ff ff 73 01 c3 48 8b 0d c3 f5 0c 00 f7
 d8 64 89 01 48
Mar 11 10:05:01 osboxes kernel: [ 3209.061401] RSP: 002b:00007fffa1582548 EFL
AGS: 00000246 ORIG_RAX: 0000000000000139
Mar 11 10:05:01 osboxes kernel: [ 3209.061404] RAX: ffffffffffffffda RBX: 000
05625d42a4790 RCX: 00007fc578d3f89d
Mar 11 10:05:01 osboxes kernel: [ 3209.061406] RDX: 0000000000000000 RSI: 000
05625d3338358 RDI: 0000000000000003
Mar 11 10:05:01 osboxes kernel: [ 3209.061407] RBP: 0000000000000000 R08: 000
0000000000000 R09: 00007fc578e13260
Mar 11 10:05:01 osboxes kernel: [ 3209.061409] R10: 0000000000000003 R11: 000
0000000000246 R12: 00005625d3338358
Mar 11 10:05:01 osboxes kernel: [ 3209.061410] R13: 0000000000000000 R14: 000
05625d42a4750 R15: 0000000000000000
Mar 11 10:05:40 osboxes kernel: [ 3248.567239] Unloading test module.
```

Test 3: Dereferencing a NULL pointer.

In this test I tried to dereference a null pointer and the test was successful. As soon as I tried to enter this module in the kernel **"Killed"** appeared on the terminal.

Code for module which dereference a null pointer.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_LICENSE("No License");
MODULE_AUTHOR("Ashutosh Sharma");
MODULE_DESCRIPTION("Crash Test");
MODULE_VERSION("0.1");
static int __init start(void)
{
    printk(KERN_INFO "Loading test module.\n");
   *(int *)0=0;
    return 0;
}

static void __exit end(void)
{
    printk(KERN_INFO "Unloading test module.\n");
}

module_init(start);
module_exit(end);
```

Kernel log when a module which dereference a null pointer enters the kernel.

```
osboxes@osboxes:~/Desktop/Bad_Modules/Crash_Test_3$ tail /var/log/kern.log
Mar 11 10:07:50 osboxes kernel: [ 3378.641021] Code: Unable to access opcode
bytes at RIP 0xffffffffc0b29fed.
Mar 11 10:07:50 osboxes kernel: [ 3378.641023] RSP: 0018:ffffb600c563fc58 EFL
AGS: 00010246
Mar 11 10:07:50 osboxes kernel: [ 3378.641026] RAX: 0000000000000000 RBX: 000
0000000000000 RCX: 0000000000000000
Mar 11 10:07:50 osboxes kernel: [ 3378.641028] RDX: 0000000000000000 RSI: fff
fa03f3a618a40 RDI: ffffa03f3a618a40
Mar 11 10:07:50 osboxes kernel: [ 3378.641030] RBP: ffffb600c563fc58 R08: fff
fa03f3a618a40 R09: ffffb600c563fa30
Mar 11 10:07:50 osboxes kernel: [ 3378.641032] R10: 0000000000000001 R11: 000
0000000000001 R12: ffffffffc0b2a000
Mar 11 10:07:50 osboxes kernel: [ 3378.641035] R13: ffffa03e8d959ec0 R14: fff
fa03e8f7c1780 R15: ffffb600c563fe70
Mar 11 10:07:50 osboxes kernel: [ 3378.641038] FS:  00007fead4133540(0000) GS
:ffffa03f3a600000(0000) knlGS:0000000000000000
Mar 11 10:07:50 osboxes kernel: [ 3378.641040] CS:  0010 DS: 0000 ES: 0000 CR
0: 0000000080050033
Mar 11 10:07:50 osboxes kernel: [ 3378.641042] CR2: ffffffffc0b29fed CR3: 000
0000000586a002 CR4: 00000000000706f0
```

```
osboxes@osboxes:~/Desktop/Bad_Modules/Crash_Test_3$ sudo insmod code.ko
[sudo] password for osboxes:
Killed
osboxes@osboxes:~/Desktop/Bad_Modules/Crash_Test_3$ sudo rmmod code
rmmod: ERROR: Module code is in use
osboxes@osboxes:~/Desktop/Bad_Modules/Crash_Test_3$
```

Q2. Round Robin Scheduling

Commands to run the program:

make

>>./q2

<No. Of processes>

<array of arrival time>

<array of burst time>

<Time quantum>

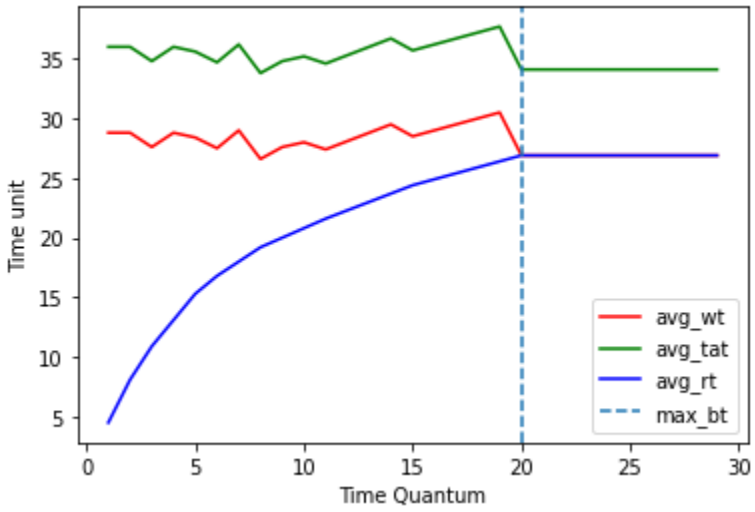
Sample input shown below:

```
[(base) anujgoel@Anujs-MacBook-Air Sp-assign2 % make
 make: Nothing to be done for `all'.
[(base) anujgoel@Anujs-MacBook-Air Sp-assign2 % ./q2
 5
 0 5 1 6 8
 8 2 7 3 5
 3
```

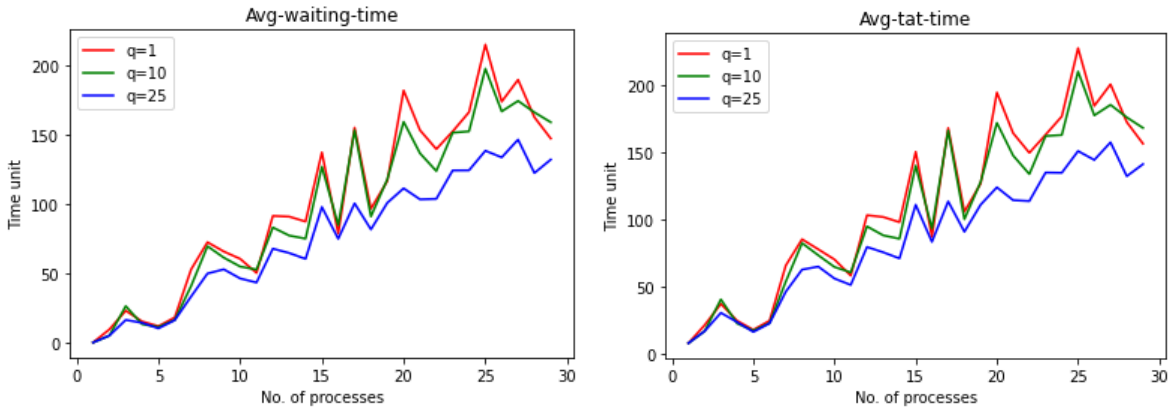
Sample output shown below for the above input:

```
 Process No          Arv Time          Burst Time          TAT          Waiting Time          Start Time          Comp Time          Resp Time
       1             0                 8                   22                 14                    0                  22                 0
       3             1                 7                   22                 15                    3                  23                 2
       2             5                 2                   6                  4                     9                  11                 4
       4             6                 3                   8                  5                     11                 14                 5
       5             8                 5                   17                 12                    17                 25                 9
 Avg-wt= 10
 Avg-tat= 15
 Avg-rt= 4
 Throughput= 5
```

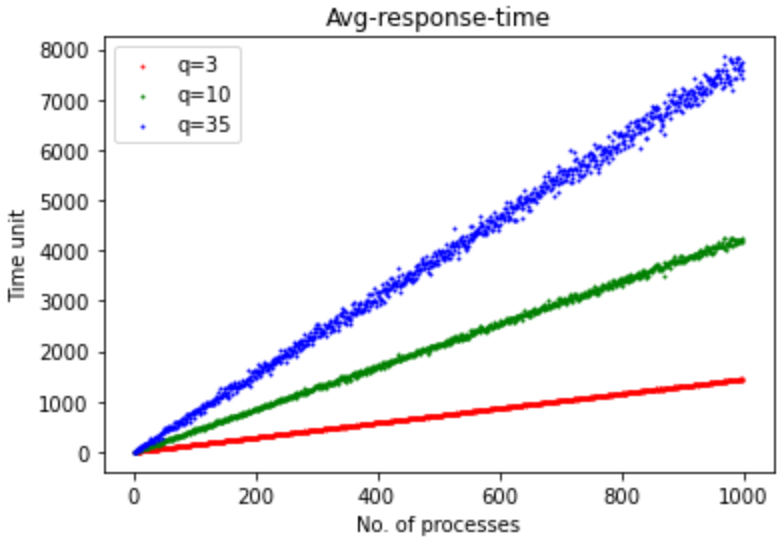
Analysis:



We can see from the above graph that for the same no. of processes if vary the time quantum then avg waiting time and avg turnaround time show similar trend. Avg response time increases because with larger time quantum processes in the queue have to wait longer as CPU spends longer time on individual processes. We can observe after a certain time quantum all becomes constant because the scheduling becomes FCFS in that case. This time quantum corresponds to the the largest burst time among all the processes.



From the above figures we infer that for the same time quantum if we increase the no. of processes then avg wait time, avg turnaround time increases as processes have to wait longer as there are many processes before them in the queue.



With increase in no. of processes avg response time increases as processes in the queue have to wait longer to get turn as there are many processes before it in the queue.

Q3. Merge Sort with multithreading

Commands to run the program:

>> make

>> ./myprogram <number_of_threads> <array_size>

Here, no. of threads and array size are command line arguments.

Sample output is shown below

Sample Output:

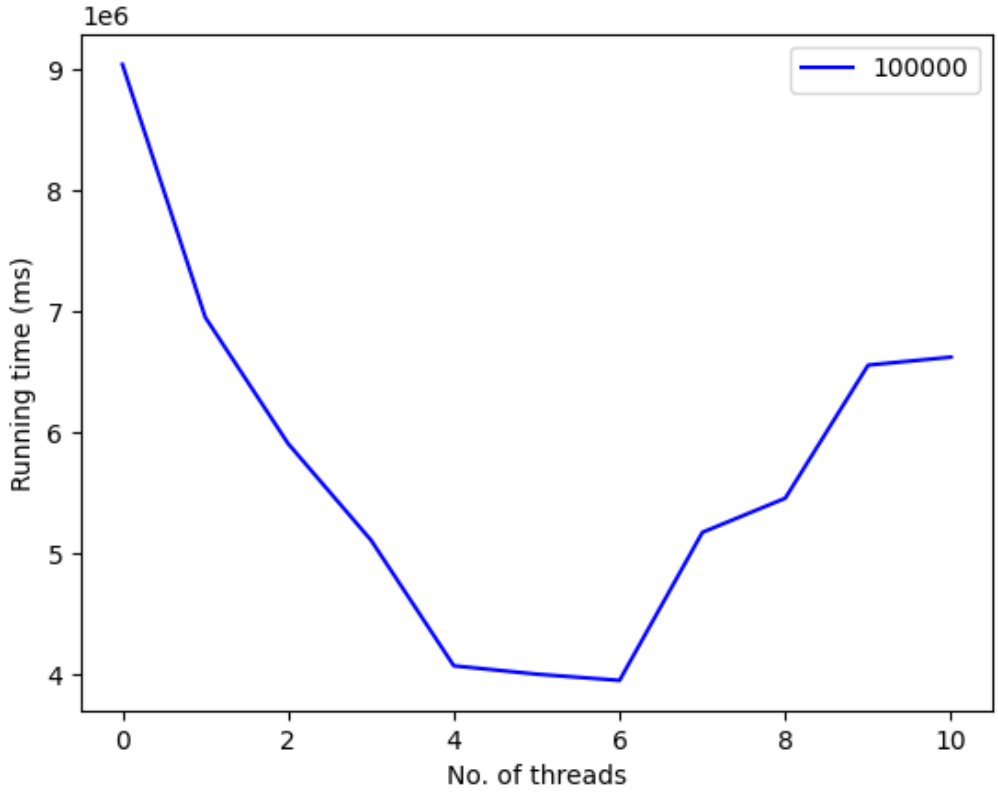
```
Initial Array: 39   10   17   35   1   15   22   28   25   1   30   7   38   3
Final Array: 1   1   3   7   7   10   15   17   22   25   28   30   35   38   3
Time taken by program: 2073 microseconds
```

The output format complies with the format given in the question

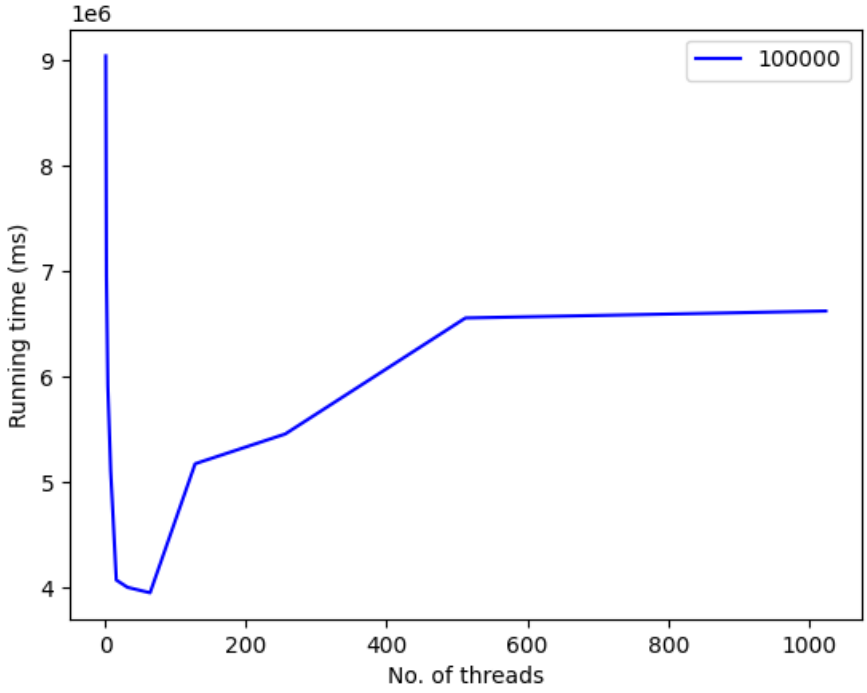
Results:

Time was recorded after varying both, number of threads and size of array. The graphs of array size = 100000 is plotted separately as it overshadowed other plots.

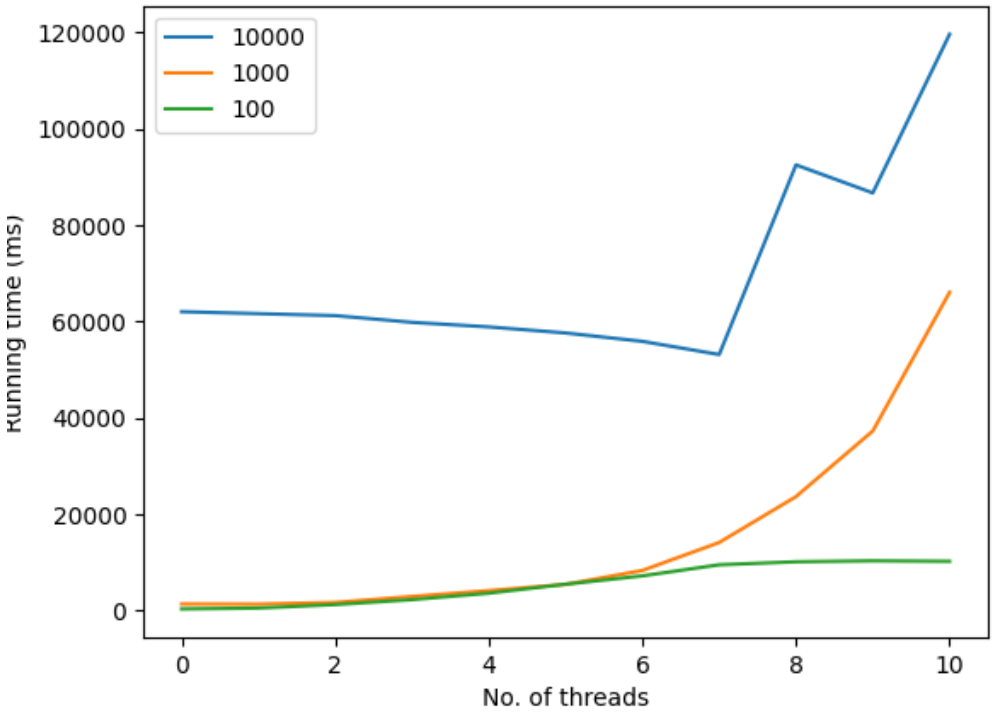
Array Sizes were taken as 100,1000,10000 and 100000. Plots on both, log scale and normal scale were plotted. Labels show the size of array


1e6
100000
Running time (ms)
No. of threads
0
2
4
6
8
10
4
5
6
7
8
9

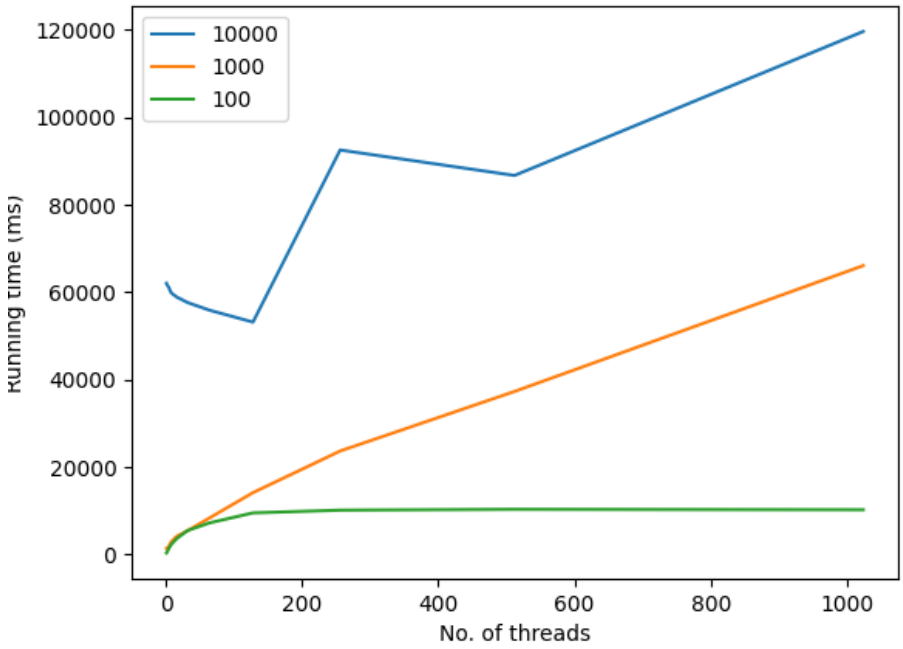
Log Scale


1e6
100000
Running time (ms)
No. of threads
0
200
400
600
800
1000
4
5
6
7
8
9

Normal Scale



Log Scale



Normal Scale

Inferences:
From the above plots, it can be inferred that for arrays whose size is small, parallelization using threads is not very cost effective. It increases the time of execution because of the overhead of thread creation, which is not compensated enough by the parallelization for the small arrays. However, for the large arrays, parallelization reduces time of execution significantly. Although as we increase the number of threads, the overhead of creation of new threads becomes more and more and we end up with a U-shaped curve as shown above.