**CSE331 TERM PROJECT**

**by**

**Burak Ok**

**Emirhan Önder**

**Eren Yağız Gürbüz**

**CSE 331 Operating Systems Design**

**Term Project Report**

**Yeditepe University**

**Faculty of Engineering**

**Department of Computer Engineering**

**Spring 2023**

**TABLE OF CONTENTS**

**1. INTRODUCTION**10

**2. DESIGN and IMPLEMENTATION**11

**2.1. LINUX DEFAULT SCHEDULER(SCHED\_OTHER)**13

**2.2. FAIR SHARE SCHEDULER**14

**2.2.1. FAIR SHARE SCHEDULER PROCESS SELECTION**15

**3. TESTS and RESULTS**16

**3.1. DEFAULT SCHEDULING ALGORITHM**19

**3.2. FAIR SHARE ALGORITHM**44

**3.3. GRAPHS**70

**4. CONCLUSION**76

**REFERENCES**78

**LIST OF FIGURES**

**Figure 2.1** **Task Struct** 11

**Figure 2.2 Default scheduler** 12

**Figure 2.1.1 Default scheduler example**13

**Figure 2.2.1 Fair-share scheduler example**14

**Figure 2.2.2 mysystemcall.c**15

**Figure 2.2.1.1 Two-dimensional array**15

**Figure 2.2.1.2 Fill the array** 16

**Figure 2.2.1.3 Assign counter values of each process**16

**Figure 2.2.1.4 Full code**17

**Figure 3.1.1.1.1 Average CPU Usage of u1p1**19

**Figure 3.1.1.1.2 Average CPU Usage of u1p2**19

**Figure 3.1.1.1.3 Average CPU Usage of u1p3**19

**Figure 3.1.1.1.4 Average CPU Usage of u2p1**19

**Figure 3.1.1.1.5 Average CPU Usage of u2p2**19

**Figure 3.1.1.1.6 Average CPU Usage of u3p1**19

**Figure 3.1.1.2.1 Average CPU Usage of u1p1**20

**Figure 3.1.1.2.2 Average CPU Usage of u1p2**20

**Figure 3.1.1.2.3 Average CPU Usage of u1p3**20

**Figure 3.1.1.2.4 Average CPU Usage of u2p1**20

**Figure 3.1.1.2.5 Average CPU Usage of u2p2**20

**Figure 3.1.1.2.6 Average CPU Usage of u3p1**20

**Figure 3.1.1.2.7 Average CPU Usage of u4p1**20

**Figure 3.1.1.2.8 Average CPU Usage of u4p2**21

**Figure 3.1.1.2.9 Average CPU Usage of u4p3**21

**Figure 3.1.1.2.10 Average CPU Usage of u4p4**21

**Figure 3.1.1.3.1 Average CPU Usage of u1p1**21

**Figure 3.1.1.3.2 Average CPU Usage of u1p2**21

**Figure 3.1.1.3.3 Average CPU Usage of u2p1**21

**Figure 3.1.1.3.4 Average CPU Usage of u2p2**21

**Figure 3.1.1.3.5 Average CPU Usage of u2p3**22

**Figure 3.1.1.4.1 Average CPU Usage of u1p1**22

**Figure 3.1.1.4.2 Average CPU Usage of u2p1**22

**Figure 3.1.1.4.3 Average CPU Usage of u2p2**22

**Figure 3.1.1.4.4 Average CPU Usage of u3p1**22

**Figure 3.1.1.5.1 Average CPU Usage of u1p1**23

**Figure 3.1.1.5.2 Average CPU Usage of u2p1**23

**Figure 3.1.1.5.3 Average CPU Usage of u2p2**23

**Figure 3.1.1.5.4 Average CPU Usage of u3p1**23

**Figure 3.1.1.5.5 Average CPU Usage of u4p1**23

**Figure 3.1.1.5.6 Average CPU Usage of u4p2**23

**Figure 3.1.1.5.7 Average CPU Usage of u4p3**24

**Figure 3.1.1.5.8 Average CPU Usage of u4p4**24

**Figure 3.1.1.5.9 Average CPU Usage of u5p1**24

**Figure 3.1.1.5.10 Average CPU Usage of u5p2**24

**Figure 3.1.1.5.11 Average CPU Usage of u5p3**24

**Figure 3.1.2.1.1 Mean Square Error of u1p1**25

**Figure 3.1.2.1.2 Mean Square Error of u1p2**25

**Figure 3.1.2.1.3 Mean Square Error of u1p3**26

**Figure 3.1.2.1.4 Mean Square Error of u2p1**26

**Figure 3.1.2.1.5 Mean Square Error of u2p2**27

**Figure 3.1.2.1.6 Mean Square Error of u3p1**27

**Figure 3.1.2.2.1 Mean Square Error of u1p1**28

**Figure 3.1.2.2.2 Mean Square Error of u1p2**28

**Figure 3.1.2.2.3 Mean Square Error of u1p3**29

**Figure 3.1.2.2.4 Mean Square Error of u2p1**29

**Figure 3.1.2.2.5 Mean Square Error of u2p2**30

**Figure 3.1.2.2.6 Mean Square Error of u3p1**30

**Figure 3.1.2.2.7 Mean Square Error of u4p1**31

**Figure 3.1.2.2.8 Mean Square Error of u4p2**31

**Figure 3.1.2.2.9 Mean Square Error of u4p3**32

**Figure 3.1.2.2.10 Mean Square Error of u4p4**32

**Figure 3.1.2.3.1 Mean Square Error of u1p1**33

**Figure 3.1.2.3.2 Mean Square Error of u1p2**33

**Figure 3.1.2.3.3 Mean Square Error of u2p1**34

**Figure 3.1.2.3.4 Mean Square Error of u2p2**34

**Figure 3.1.2.3.5 Mean Square Error of u2p3**35

**Figure 3.1.2.4.1 Mean Square Error of u1p1**36

**Figure 3.1.2.4.2 Mean Square Error of u2p1**36

**Figure 3.1.2.4.3 Mean Square Error of u2p2**37

**Figure 3.1.2.4.4 Mean Square Error of u3p1**37

**Figure 3.1.2.5.1 Mean Square Error of u1p1**38

**Figure 3.1.2.5.2 Mean Square Error of u2p1**38

**Figure 3.1.2.5.3 Mean Square Error of u2p2**39

**Figure 3.1.2.5.4 Mean Square Error of u3p1**39

**Figure 3.1.2.5.5 Mean Square Error of u4p1**40

**Figure 3.1.2.5.6 Mean Square Error of u4p2**40

**Figure 3.1.2.5.7 Mean Square Error of u4p3**41

**Figure 3.1.2.5.8 Mean Square Error of u4p4**41

**Figure 3.1.2.5.9 Mean Square Error of u5p1**42

**Figure 3.1.2.5.10 Mean Square Error of u5p2**42

**Figure 3.1.2.5.11 Mean Square Error of u5p3**43

**Figure 3.2.1.1.1 Average CPU Usage of u1p1**44

**Figure 3.2.1.1.2 Average CPU Usage of u1p2**44

**Figure 3.2.1.1.3 Average CPU Usage of u1p3**44

**Figure 3.2.1.1.4 Average CPU Usage of u2p1**44

**Figure 3.2.1.1.5 Average CPU Usage of u2p2**44

**Figure 3.2.1.1.6 Average CPU Usage of u3p1**44

**Figure 3.2.1.2.1 Average CPU Usage of u1p1**45

**Figure 3.2.1.2.2 Average CPU Usage of u1p2**45

**Figure 3.2.1.2.3 Average CPU Usage of u1p3**45

**Figure 3.2.1.2.4 Average CPU Usage of u2p1**45

**Figure 3.2.1.2.5 Average CPU Usage of u2p2**45

**Figure 3.2.1.2.6 Average CPU Usage of u3p1**45

**Figure 3.2.1.2.7 Average CPU Usage of u4p1**46

**Figure 3.2.1.2.8 Average CPU Usage of u4p2**46

**Figure 3.2.1.2.9 Average CPU Usage of u4p3**46

**Figure 3.2.1.2.10 Average CPU Usage of u4p4**46

**Figure 3.2.1.3.1 Average CPU Usage of u1p1**47

**Figure 3.2.1.3.2 Average CPU Usage of u1p2**47

**Figure 3.2.1.3.3 Average CPU Usage of u2p1**47

**Figure 3.2.1.3.4 Average CPU Usage of u2p2**47

**Figure 3.2.1.3.5 Average CPU Usage of u2p3**47

**Figure 3.2.1.4.1 Average CPU Usage of u1p1**48

**Figure 3.2.1.4.2 Average CPU Usage of u2p1**48

**Figure 3.2.1.4.3 Average CPU Usage of u2p2**48

**Figure 3.2.1.4.4 Average CPU Usage of u3p1**48

**Figure 3.2.1.5.1 Average CPU Usage of u1p1**49

**Figure 3.2.1.5.2 Average CPU Usage of u2p1**49

**Figure 3.2.1.5.3 Average CPU Usage of u2p2**49

**Figure 3.2.1.5.4 Average CPU Usage of u3p1**49

**Figure 3.2.1.5.5 Average CPU Usage of u4p1**49

**Figure 3.2.1.5.6 Average CPU Usage of u4p2**49

**Figure 3.2.1.5.7 Average CPU Usage of u4p3**50

**Figure 3.2.1.5.8 Average CPU Usage of u4p4**50

**Figure 3.2.1.5.9 Average CPU Usage of u5p1**50

**Figure 3.2.1.5.10 Average CPU Usage of u5p2**50

**Figure 3.2.1.5.11 Average CPU Usage of u5p3**50

**Figure 3.2.2.1.1 Mean Square Error of u1p1**51

**Figure 3.2.2.1.2 Mean Square Error of u1p2**51

**Figure 3.2.2.1.3 Mean Square Error of u1p3**52

**Figure 3.2.2.1.4 Mean Square Error of u2p1**52

**Figure 3.2.2.1.5 Mean Square Error of u2p2**53

**Figure 3.2.2.1.6 Mean Square Error of u3p1**53

**Figure 3.2.2.2.1 Mean Square Error of u1p1**54

**Figure 3.2.2.2.2 Mean Square Error of u1p2**54

**Figure 3.2.2.2.3 Mean Square Error of u1p3**55

**Figure 3.2.2.2.4 Mean Square Error of u2p1**55

**Figure 3.2.2.2.5 Mean Square Error of u2p2**56

**Figure 3.2.2.2.6 Mean Square Error of u3p1**56

**Figure 3.2.2.2.7 Mean Square Error of u4p1**57

**Figure 3.2.2.2.8 Mean Square Error of u4p2**57

**Figure 3.2.2.2.9 Mean Square Error of u4p3**58

**Figure 3.2.2.2.10 Mean Square Error of u4p4**58

**Figure 3.2.2.3.1 Mean Square Error of u1p1**59

**Figure 3.2.2.3.2 Mean Square Error of u1p2**59

**Figure 3.2.2.3.3 Mean Square Error of u2p1**60

**Figure 3.2.2.3.4 Mean Square Error of u2p2**60

**Figure 3.2.2.3.5 Mean Square Error of u2p3**61

**Figure 3.2.2.4.1 Mean Square Error of u1p1**62

**Figure 3.2.2.4.2 Mean Square Error of u2p1**62

**Figure 3.2.2.4.3 Mean Square Error of u2p2**63

**Figure 3.2.2.4.4 Mean Square Error of u3p1**63

**Figure 3.2.2.5.1 Mean Square Error of u1p1**64

**Figure 3.2.2.5.2 Mean Square Error of u2p1**64

**Figure 3.2.2.5.3 Mean Square Error of u2p2**65

**Figure 3.2.2.5.4 Mean Square Error of u3p1**65

**Figure 3.2.2.5.5 Mean Square Error of u4p1**66

**Figure 3.2.2.5.6 Mean Square Error of u4p2**66

**Figure 3.2.2.5.7 Mean Square Error of u4p3**67

**Figure 3.2.2.5.8 Mean Square Error of u4p4**67

**Figure 3.2.2.5.9 Mean Square Error of u5p1**68

**Figure 3.2.2.5.10 Mean Square Error of u5p2**68

**Figure 3.2.2.5.11 Mean Square Error of u5p3**69

**Figure 3.3.1.1.1 Graph of TestCase-1 (Average CPU Usage)** 70

**Figure 3.3.1.2.1 Graph of TestCase-2 (Average CPU Usage)** 71

**Figure 3.3.1.3.1 Graph of TestCase-3 (Average CPU Usage)** 71

**Figure 3.3.1.4.1 Graph of TestCase-4 (Average CPU Usage)** 72

**Figure 3.3.1.5.1 Graph of TestCase-5 (Average CPU Usage)** 72

**Figure 3.3.2.1.1 Graph of TestCase-1 (Mean Square Error)** 73

**Figure 3.3.2.2.1 Graph of TestCase-2 (Mean Square Error)** 73

**Figure 3.3.2.3.1 Graph of TestCase-3 (Mean Square Error)** 74

**Figure 3.3.2.4.1 Graph of TestCase-4 (Mean Square Error)** 74

**Figure 3.3.2.5.1 Graph of TestCase-5 (Mean Square Error)** 75

# INTRODUCTION

In a traditional Linux environment, the CPU scheduling policies distribute the computational resources among running processes. However, these policies don't consider the ownership of these processes, potentially leading to a situation where a user running a large number of processes can monopolize the system's computational power. This situation may not be an issue in a single-user system, but it can significantly impact fairness in a multi-user environment, where equitable distribution of resources is crucial.

Our project aimed to resolve this issue by developing a fair-share scheduling policy. The primary objective of this policy is to ensure that each user gets an equitable share of the CPU, irrespective of the number of processes they are running. The fair-share scheduler accomplishes this by grouping processes by their owning user and then distributing CPU resources among these groups. As a result, each user gets a fair share of the CPU, and within each user's group, the CPU time is further divided among their respective processes.

The implementation of the fair-share scheduling policy required a deep understanding of the Linux kernel, specifically the kernel's scheduling policies and their implementation. It also necessitated a careful and meticulous design process to ensure minimal disruption to the existing kernel functionalities, and to maintain system stability and performance while improving fairness.

By implementing the fair-share scheduling policy, we strived to make the Linux operating environment more equitable for all users. This project was not only about enhancing the performance of the system but also about ensuring fairness and maximizing the efficient utilization of computational resources. In the following sections, we will delve deeper into the design, implementation, and testing of our fair-share scheduler.

# DESIGN and IMPLEMENTATION

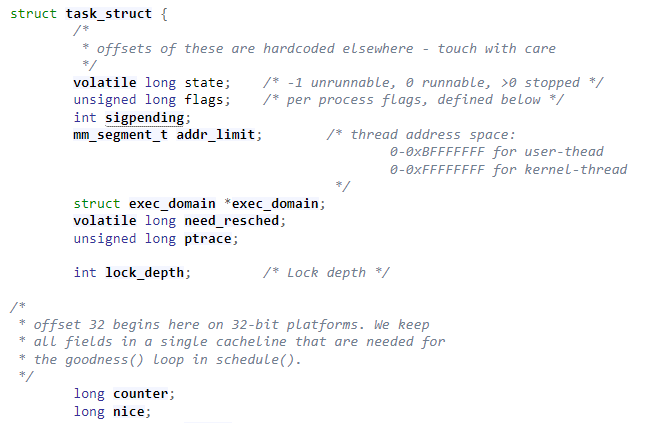
Linux uses two different process scheduling algorithms. These are time-sharing and real-time algorithms. Time-sharing algorithm is fair and preemptive scheduling between multiple processes. This algorithm, which is the Linux default scheduler(SCHED\_OTHER), simultaneously divides the time in 10ms intervals. On the other hand in Real-time algorithms(SCHED\_FIFO,SCHED\_RR),absolute priorities are more important than fairness. It is used to allow the processor to allocate more time to processes that need to run in real time.

In Linux default scheduler, each process has a value called nice. Nice values ​​determine the dynamic priority of the process it belongs to. The nice values ​​of newly created processes are set to 0 by default. If the nice value of a process is negative, this process has a high priority. If it is a positive value, the priority of the process is low. The priority of the process is calculated with 20-nice.If a process has I/O burst ,system lowers nice value of a process to provide higher priority but if a process only has CPU burst , nice values is not changed.

The time slices allocated with the priority of the process is directly proportional. The system keeps the time slices allocated to the processes with the counter variable and each process has a counter variable. The counter variable specifies how many time slices the process is allocated. When the calculated counters are finished, the counter values ​​of the processes are recalculated, this event is called epoch.

**Task Structure**

Task structure is the structure that holds the information of a process. Values ​​such as nice, counter, state, etc. are kept in the task structure. The system obtains the information of the processes through task structure and makes the process selection according to this information.



**Figure 2.1** Task struct

**Process Selection**

Linux makes the process selection with the schedule() function. The system enters the schedule() function every 10ms to find the process with the highest counter .Inside the Schedule() function, it navigates the processes waiting in the ready state with the for loop and checks whether the processes are scheduled or not. At the same time, the goodness value is calculated according to the priority of each process. The goodness value is used to find the best candidate process to run. If the calculated goodness value is -1000, the process must never be selected. If the goodness value is 0, the process has exhausted .If the goodness value is between 0 and 1000, the process can run and its value is calculated with (20-nice)+counter. On the other hand if goodness value is >=1000 this process is a real time process and higher priority to run. The system repeats this algorithm every 10ms time slices(In figure 2.2).



**Figure 2.2** Default scheduler

## 2.1 LINUX DEFAULT SCHEDULER(SCHED\_OTHER)

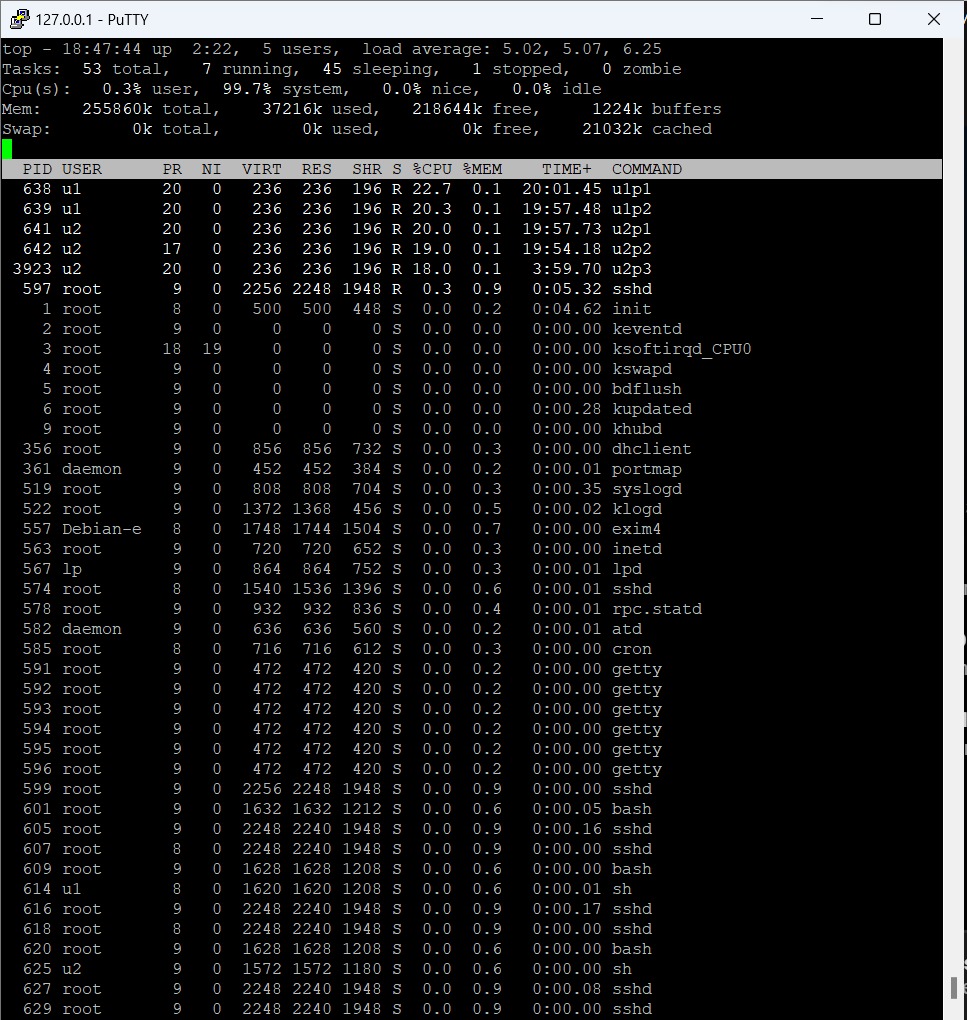
Linux default schedular is a process based algorithm. In this algorithm, regardless of the number of users, the CPU is directly proportional to the number of processes. For example, in a system has 3 users and first user has 1 process , second user has 2 process and third user has 1 process. In this case CPU is distributed equally (~25%) to each process(In figure 2.1).



**Figure 2.1.1** Default scheduler example

## 2.2 FAIR SHARE SCHEDULER

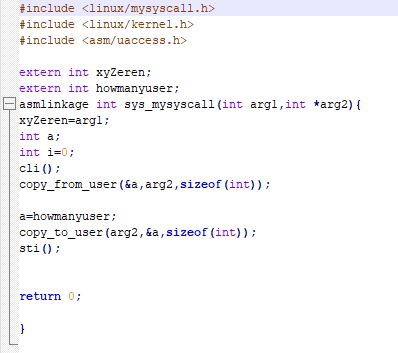
Fair share schedule is a user based algorithm. The fair share schedule provides a user-based fairness, unlike the default schedule.

For example, if the system has 2 different users and first user has 2 processes and second user has 3 processes. Fair share schedule ensures that the CPU is distributed according to the number of users. (~ 50% for each user) Moreover, it redistributes the CPU according to the number of processes the user has. Therefore each process of the first user gets ~25% and each process of the second user gets ~16.6%.(In figure 2.2 )

**Figure 2.2.1** Fair share scheduler example

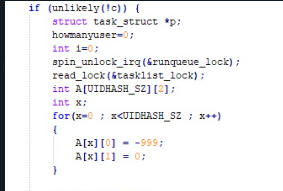
We use a system call to switch from the default schedular to fair share scheduler. In this

system call decides which scheduler the system will switch to by looking at the value of the variable in it.

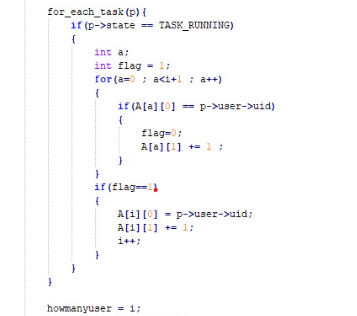


**Figure 2.2.2** mysystemcall.c

## FAIR SHARE SCHEDULER PROCESS SELECTION

In order to create a user-based fairness in the fair share schedule, we first created a two-dimensional array. In this array, we kept the user IDs of the running processes that matched this ID (In figure 2.2.1.1).

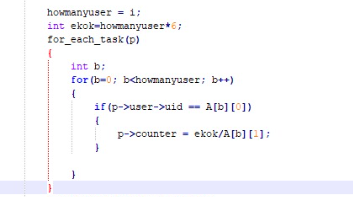
**Figure 2.2.1.1** Two-dimensional array

To fill the array, we first visited all the processes in the running state with the help of the for\_each\_task. Then, we assigned the user IDs of the processes to the first column of the array and the number of running processes owned by the user id hold in the first column is written in the second column. While doing these, we also counted the number of users in the system (In figure 2.2.1.2).

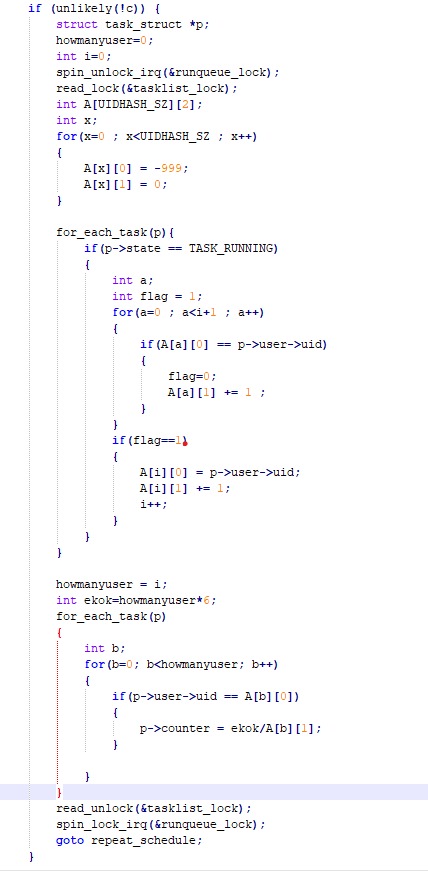
**Figure 2.2.1.2** Fill the array

After these processes are finished, we calculated the ekok variable as the number of users times 6 so that each process gets a divisible counter value.(In figure 2.2.1.3).

Finally, with the help of the for\_each\_task, we navigated the processes in our two-dimensional array that matched the user IDs and assigned the counters of each process by dividing them by the number of running processes that matched user ID have (In figure 2.2.1.3).



**Figure 2.2.1.3** Assign counter values of each process



**Figure 2.2.1.4** Full code

# TESTS and RESULTS

The reason we run performance tests is to measure how fast and efficiently the code written by us works and to detect potential bottlenecks or performance issues. In this way, we will be able to make the necessary adjustments to better optimize the code and improve the user experience. Also, by performing performance tests, we can evaluate the use of system resources and the scalability of the code and understand how ready we are to meet future growth and demands.

Since performance tests are important for the success of the application and user satisfaction, these tests should be carried out regularly and their results should be taken into account. During testing, we can observe how the code performs under different scenarios and user loads, thus making it suitable for a wider range of users.

Also, through performance tests, we can evaluate how the code works in different hardware and software environments, thus ensuring a wider range of application compatibility. This will help the app reach more users and provide a better user experience.

As a result, we ensure that the code we write is constantly developed and optimized by performing performance tests. This helps make the application faster, more reliable, and more scalable, thus providing a better foundation to meet and exceed users' expectations. Therefore, we should not ignore the importance of performance tests and consider them as an integral part of the software development process.

In the tests we made below, we tried to test whether the CPU is fair by increasing the number of processes and the number of users. We repeated the tests we made a certain number of times and tested whether we were staying within the limits we determined with the random distribution, whether we wrote a really fair code, and analyzed these results statistically.

In general, we observed that when we do more repetitions in terms of number, we reach a value closer to the expected result. We will present these results to you with pictures in the next section.

**3.1.** **DEFAULT SCHEDULING ALGORITHM**

**3.1.1. AVERAGE CPU USAGE**

**3.1.1.1.** **TestCase-1:**

In test case-1 , we have 3 users.

User1 -> 3 processes

User2 -> 2 processes

User3 -> 1 process



**Figure 3.1.1.1.1** Average CPU Usage of u1p1



**Figure 3.1.1.1.2** Average CPU Usage of u1p2



**Figure 3.1.1.1.3** Average CPU Usage of u1p3



**Figure 3.1.1.1.4** Average CPU Usage of u2p1



**Figure 3.1.1.1.5** Average CPU Usage of u2p2



**Figure 3.1.1.1.6** Average CPU Usage of u3p1

**3.1.1.2.** **TestCase-2:**

In test case-2 , we have 4 users.

User1 -> 3 processes

User2 -> 2 processes

User3 -> 1 process

User4 -> 4 processes



**Figure 3.1.1.2.1** Average CPU Usage of u1p1



**Figure 3.1.1.2.2** Average CPU Usage of u1p2



**Figure 3.1.1.2.3** Average CPU Usage of u1p3

**Figure 3.1.1.2.4** Average CPU Usage of u2p1



**Figure 3.1.1.2.5** Average CPU Usage of u2p2



**Figure 3.1.1.2.6** Average CPU Usage of u3p1



**Figure 3.1.1.2.7** Average CPU Usage of u4p1



**Figure 3.1.1.2.8** Average CPU Usage of u4p2

**Figure 3.1.1.2.9** Average CPU Usage of u4p3



**Figure 3.1.1.2.10** Average CPU Usage of u4p4

**3.1.1.3.** **TestCase-3:**

In test case-3 , we have 2 users.

User1 -> 2 processes

User2 -> 3 processes



**Figure 3.1.1.3.1** Average CPU Usage of u1p1

**Figure 3.1.1.3.2** Average CPU Usage of u1p2

**Figure 3.1.1.3.3** Average CPU Usage of u2p1

**Figure 3.1.1.3.4** Average CPU Usage of u2p2



**Figure 3.1.1.3.5** Average CPU Usage of u2p3

**3.1.1.4.** **TestCase-4:**

In test case-4 , we have 3 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process



**Figure 3.1.1.4.1** Average CPU Usage of u1p1



**Figure 3.1.1.4.2** Average CPU Usage of u2p1



**Figure 3.1.1.4.3** Average CPU Usage of u2p2



**Figure 3.1.1.4.4** Average CPU Usage of u3p1

**3.1.1.5.** **TestCase-5:**

In test case-5 , we have 5 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process

User4 -> 4 processes

User5 -> 3 processes



**Figure 3.1.1.5.1** Average CPU Usage of u1p1



**Figure 3.1.1.5.2** Average CPU Usage of u2p1



**Figure 3.1.1.5.3** Average CPU Usage of u2p2



**Figure 3.1.1.5.4** Average CPU Usage of u3p1



**Figure 3.1.1.5.5** Average CPU Usage of u4p1



**Figure 3.1.1.5.6** Average CPU Usage of u4p2



**Figure 3.1.1.5.7** Average CPU Usage of u4p3



**Figure 3.1.1.5.8** Average CPU Usage of u4p4



**Figure 3.1.1.5.9** Average CPU Usage of u5p1



**Figure 3.1.1.5.10** Average CPU Usage of u5p2



**Figure 3.1.1.5.11** Average CPU Usage of u5p3

**3.1.2. MEAN SQUARE ERROR**

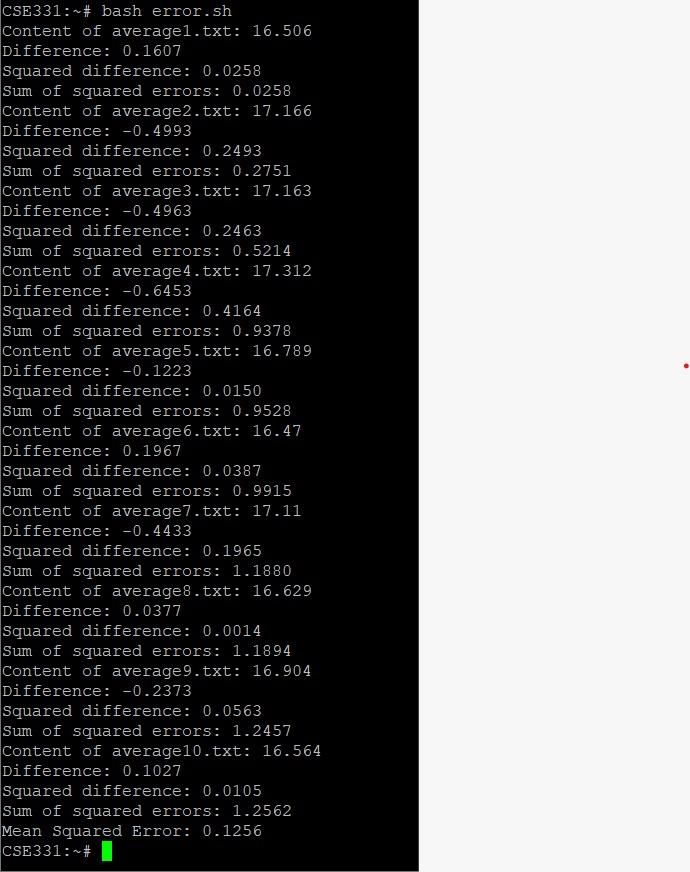
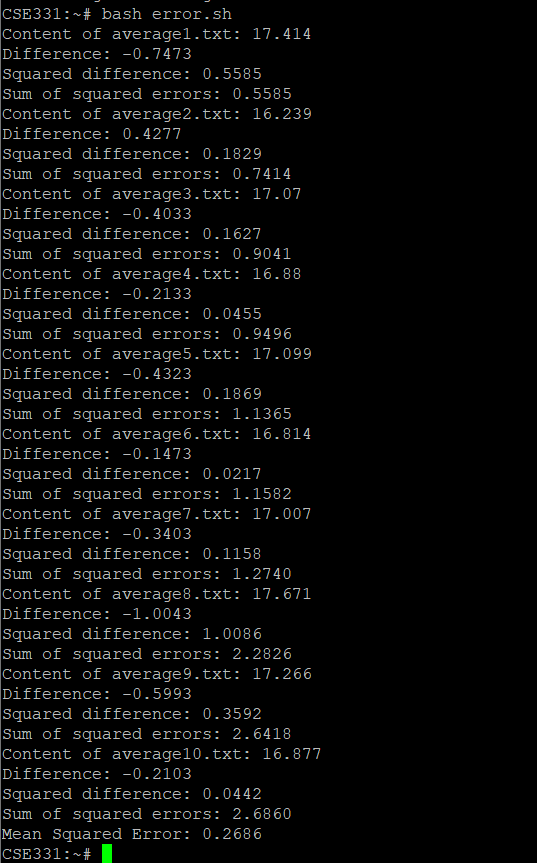
**3.1.2.1. TestCase-1:**

In test case-1 , we have 3 users.

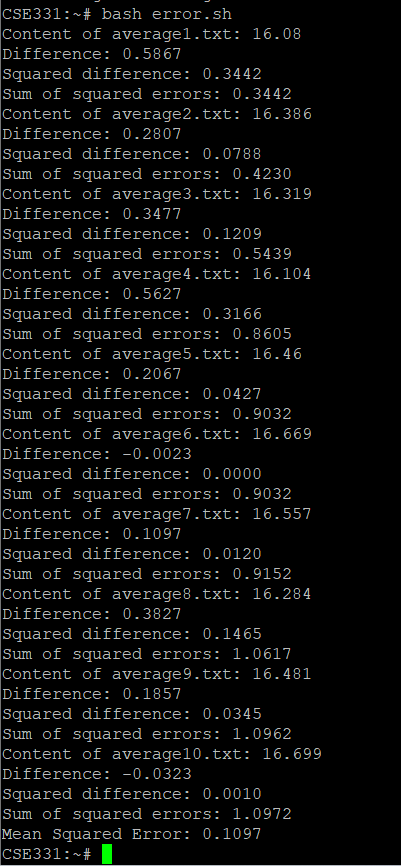
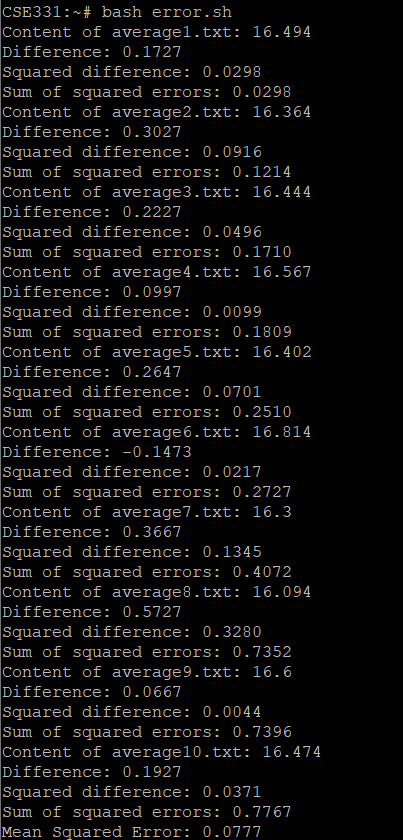
User1 -> 3 processes

User2 -> 2 processes

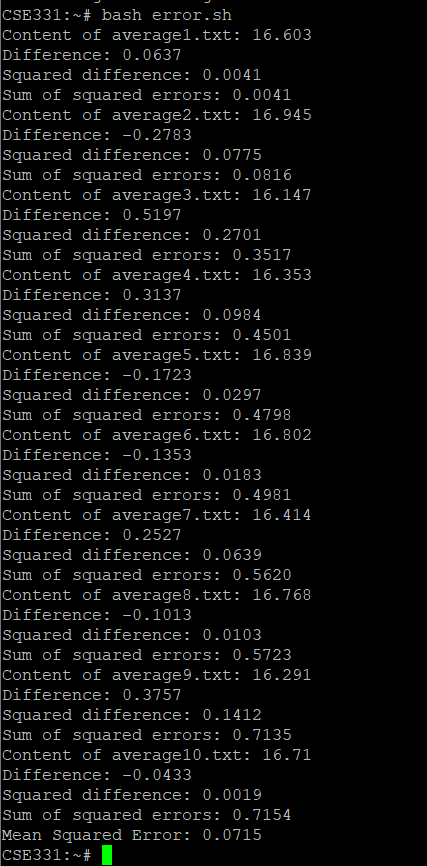
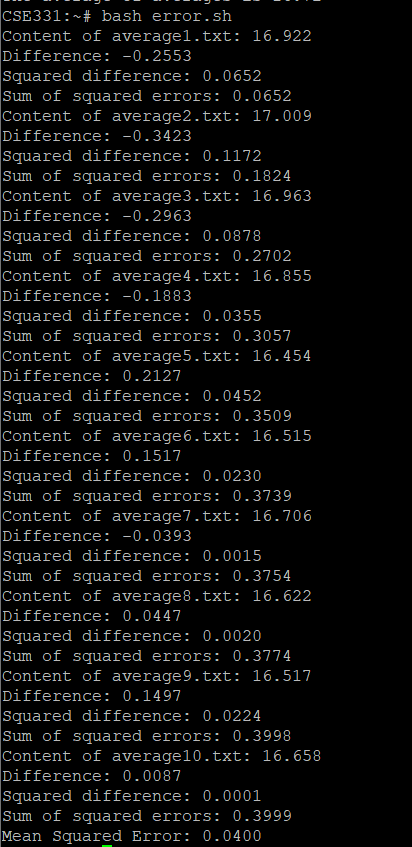
User3 -> 1 process



**Figure 3.1.2.1.1** Mean Square Error of u1p1 **Figure 3.1.2.1.2** Mean Square Error of u1p2



**Figure 3.1.2.1.3** Mean Square Error of u1p3 **Figure 3.1.2.1.4** Mean Square Error of u2p1

**Figure 3.1.2.1.5** Mean Square Error of u2p2 **Figure 3.1.2.1.6** Mean Square Error of u3p1

**3.1.2.2. TestCase-2:**

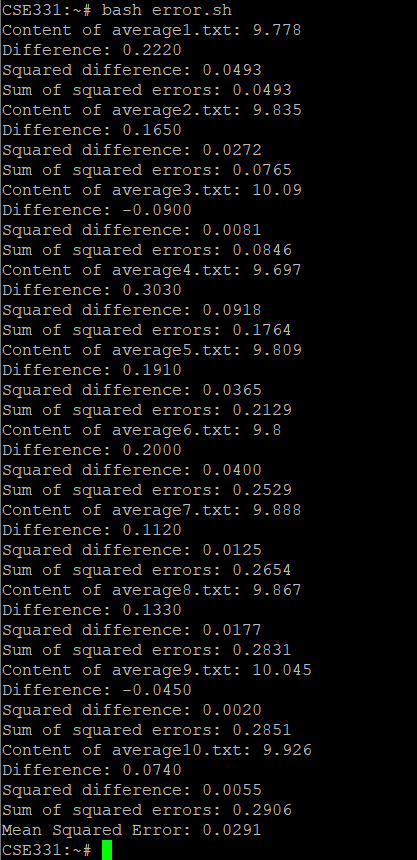
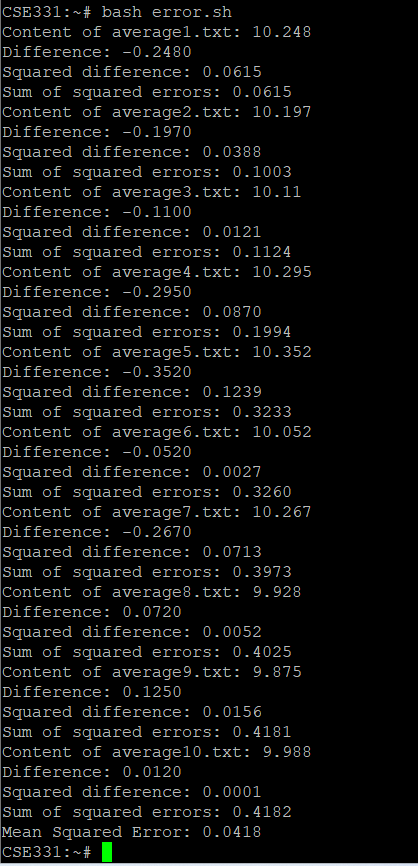
In test case-1 , we have 4 users.

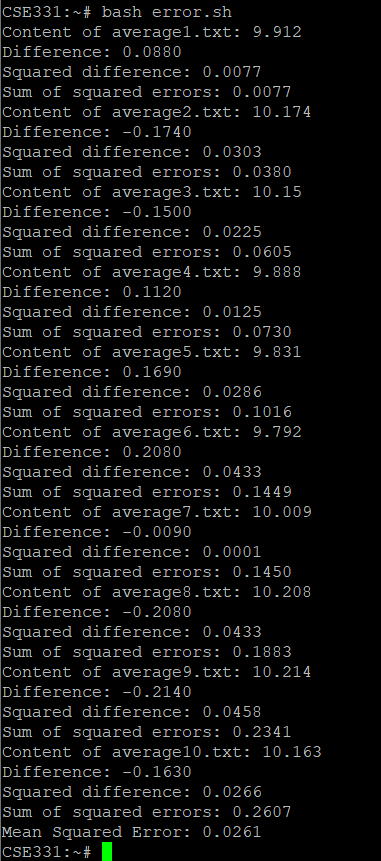
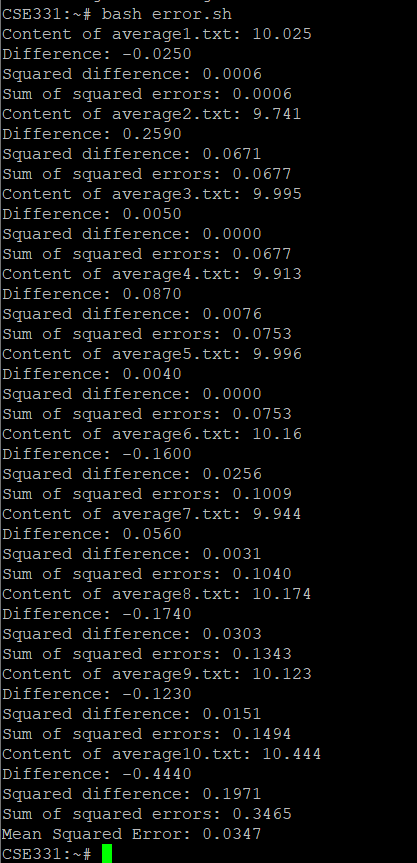
User1 -> 3 processes

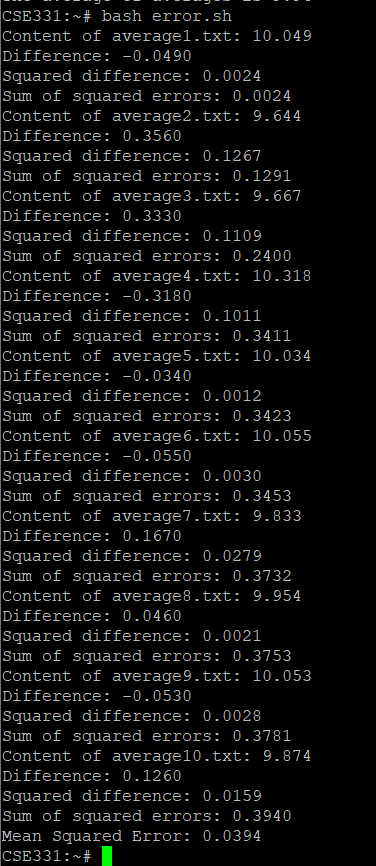
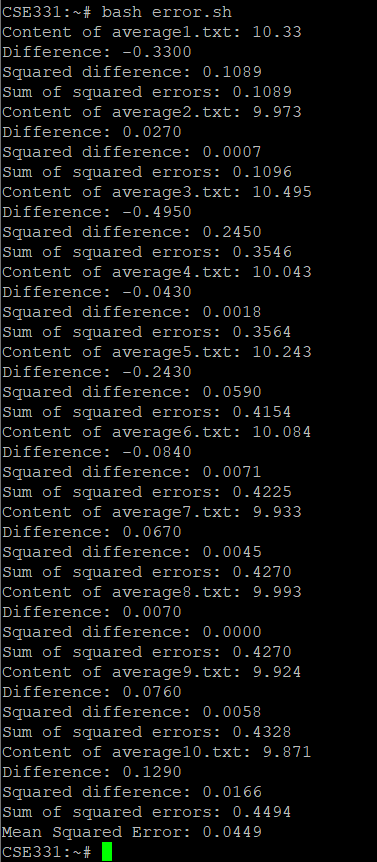
User2 -> 2 processes

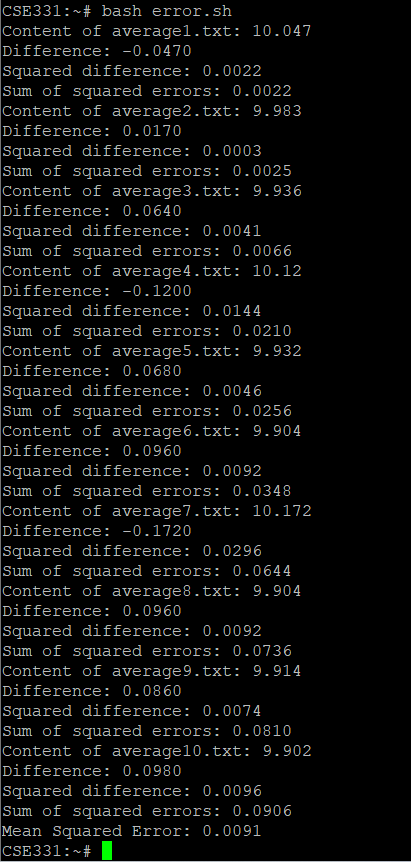
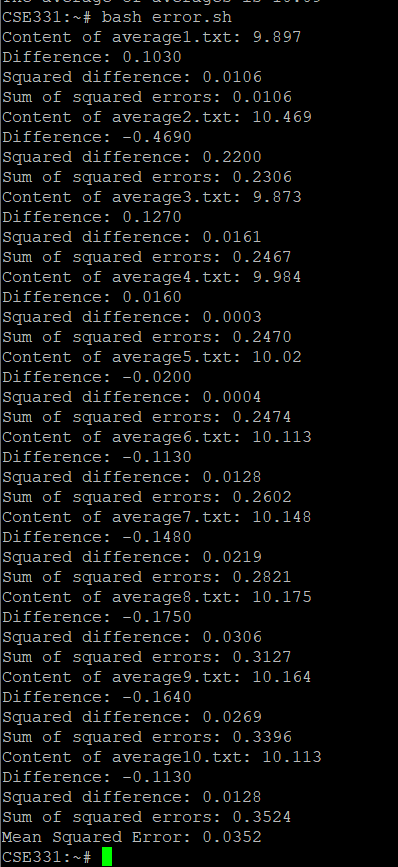
User3 -> 1 process

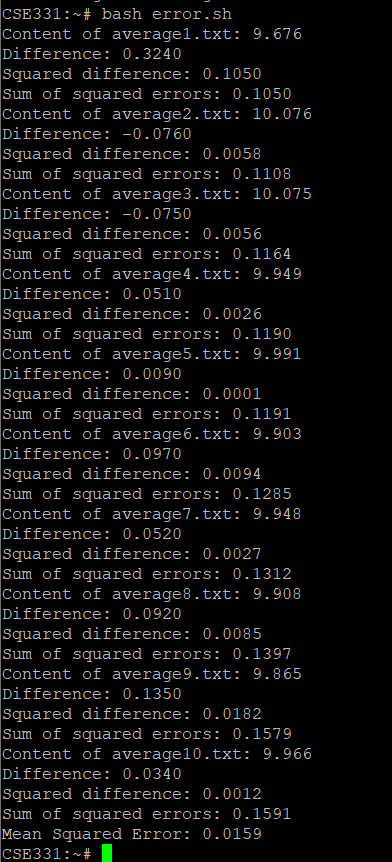
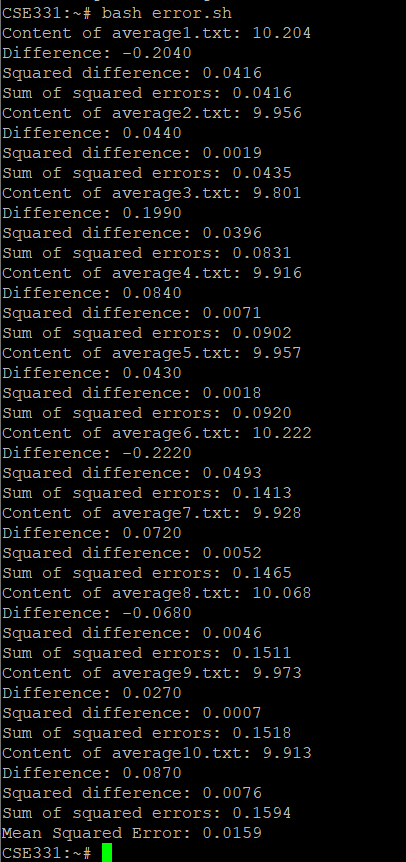
User4 -> 4 processes

 **Figure 3.1.2.2.1** Mean Square Error of u1p1 **Figure 3.1.2.2.2** Mean Square Error of u1p2

**Figure 3.1.2.2.3** Mean Square Error of u1p3 **Figure 3.1.2.2.4** Mean Square Error of u2p1

**Figure 3.1.2.2.5** Mean Square Error of u2p2 **Figure 3.1.2.2.6** Mean Square Error of u3p1

**Figure 3.1.2.2.7** Mean Square Error of u4p1 **Figure 3.1.2.2.8** Mean Square Error of u4p2

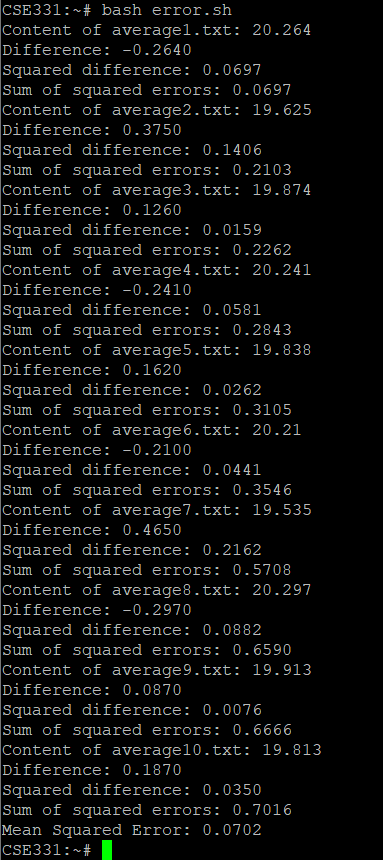
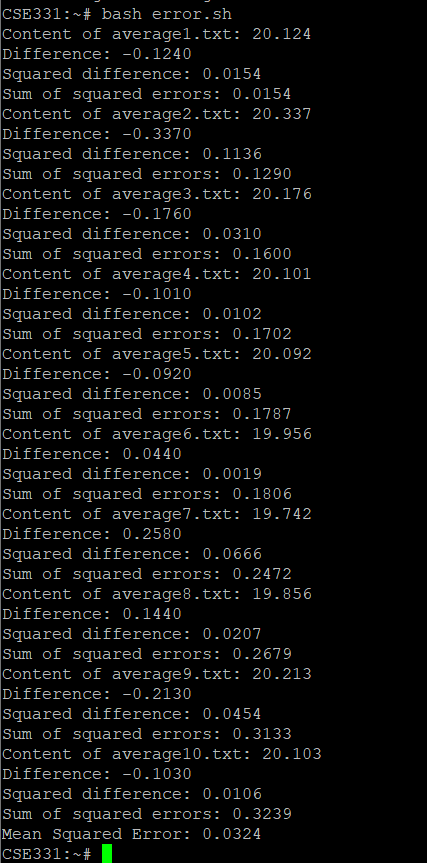
 **Figure 3.1.2.2.9** Mean Square Error of u4p3 **Figure 3.1.2.2.10** Mean Square Error of u4p4

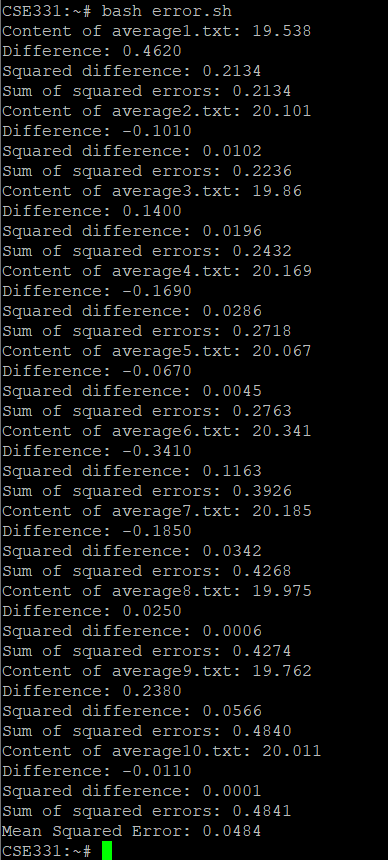
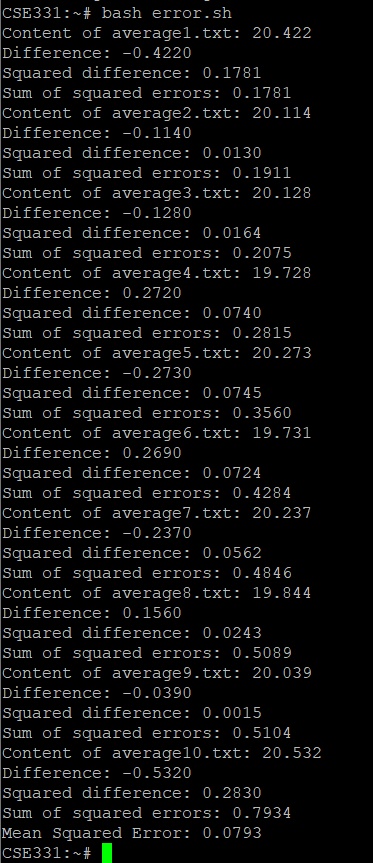
**3.1.2.3. TestCase-3:**

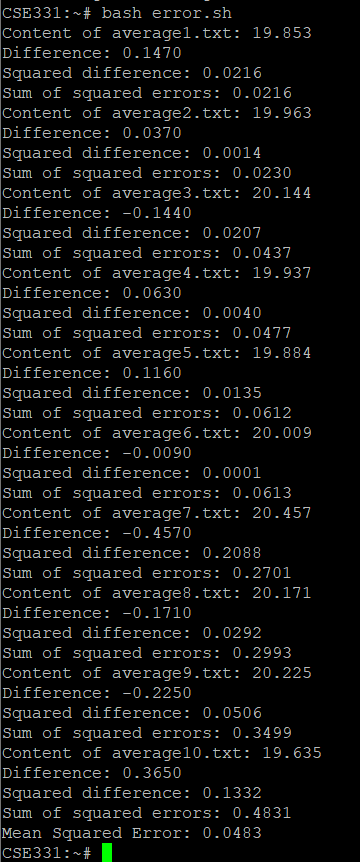
In test case-1 , we have 2 users.

User1 -> 2 processes

User2 -> 3 processes

**Figure 3.1.2.3.1** Mean Square Error of u1p1 **Figure 3.1.2.3.1** Mean Square Error of u1p2

**Figure 3.1.2.3.3** Mean Square Error of u2p1 **Figure 3.1.2.3.4** Mean Square Error of u2p2 



**Figure 3.1.2.3.5** Mean Square Error of u2p3

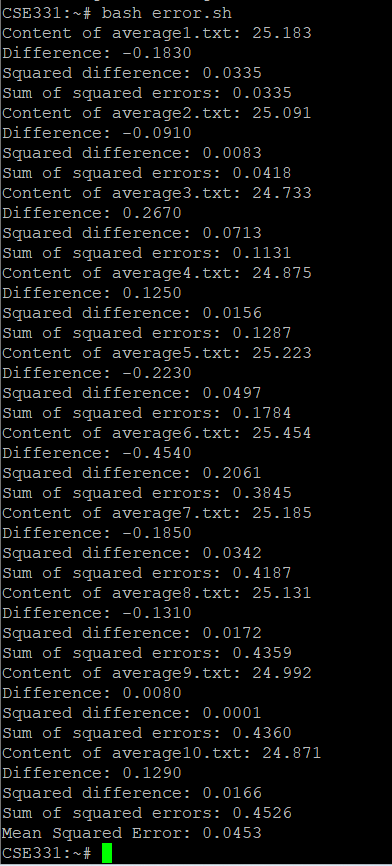
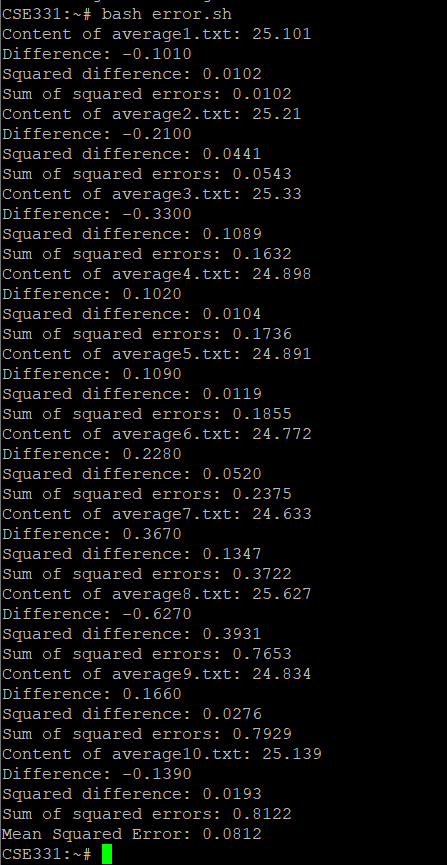
**3.1.2.4. TestCase-4:**

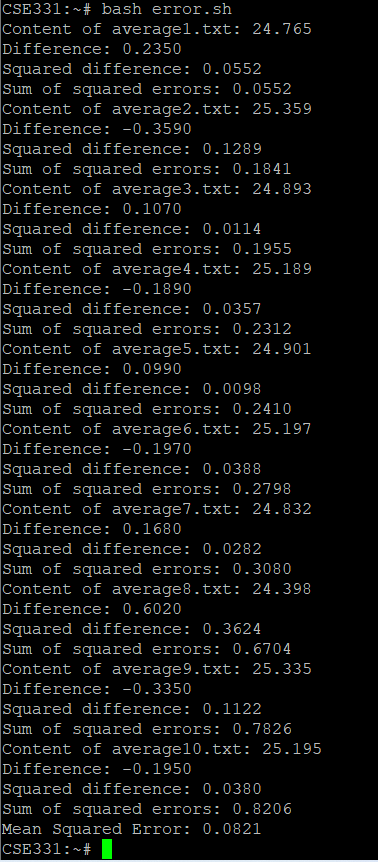
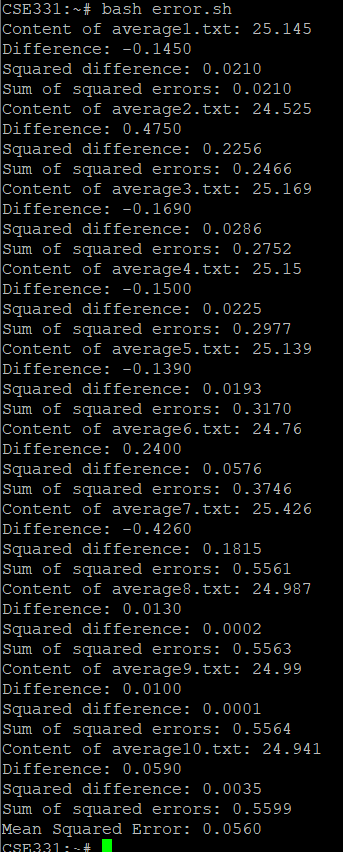
In test case-1 , we have 3 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process

**Figure 3.1.2.4.1** Mean Square Error of u1p1 **Figure 3.1.2.4.2** Mean Square Error of u2p1



**Figure 3.1.2.4.3** Mean Square Error of u2p2 **Figure 3.1.2.4.4** Mean Square Error of u3p1

**3.1.2.5. TestCase-5:**

In test case-1 , we have 5 users.

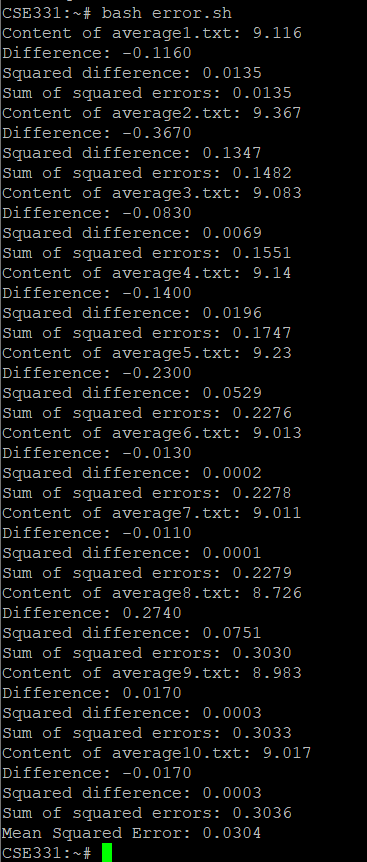
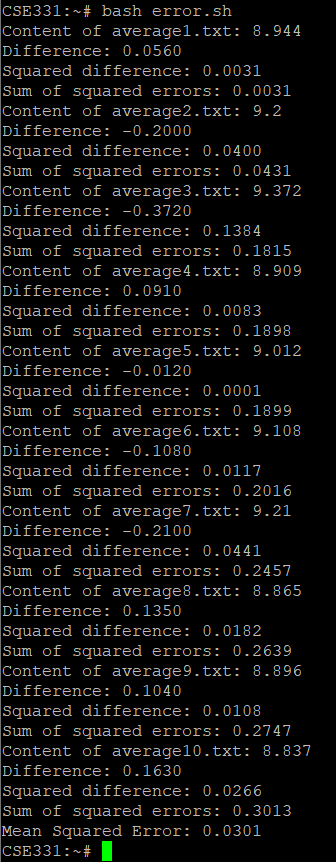
User1 -> 1 process

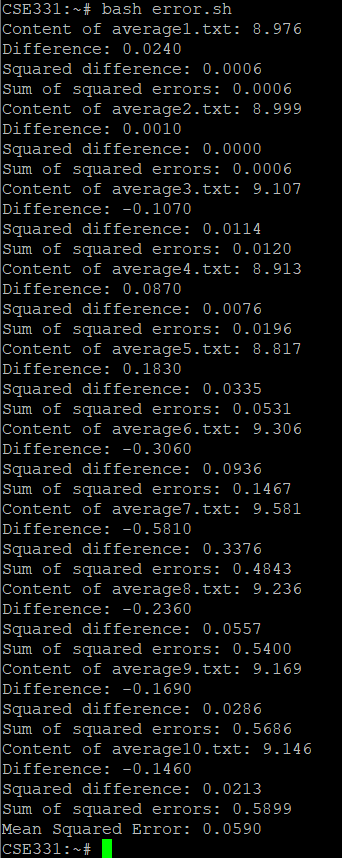
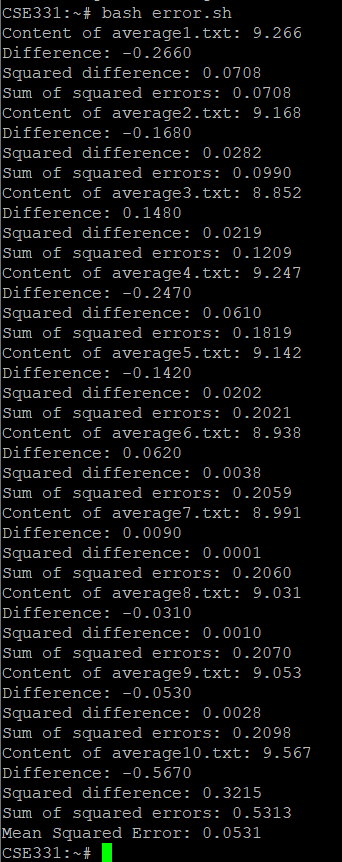
User2 -> 2 processes

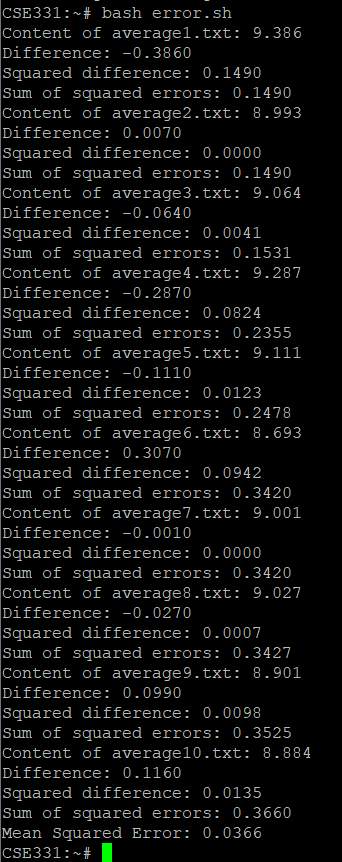
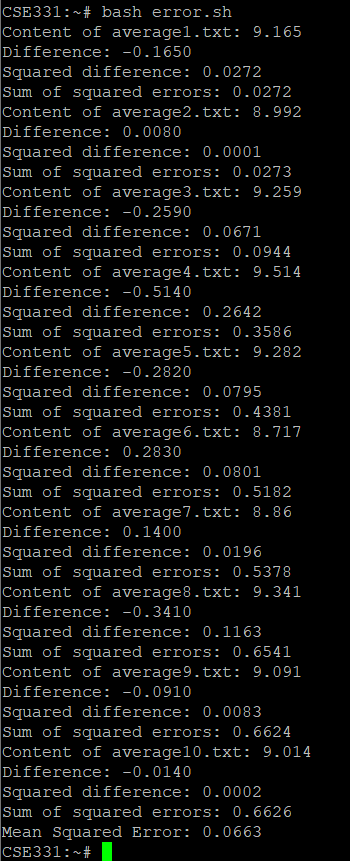
User3 -> 1 process

User4 -> 4 process

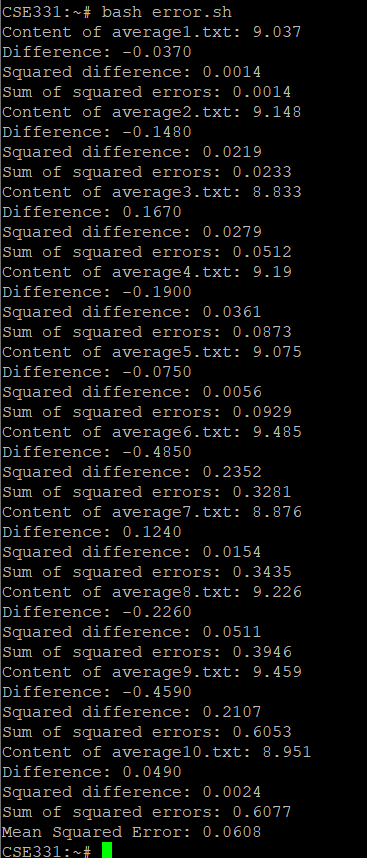
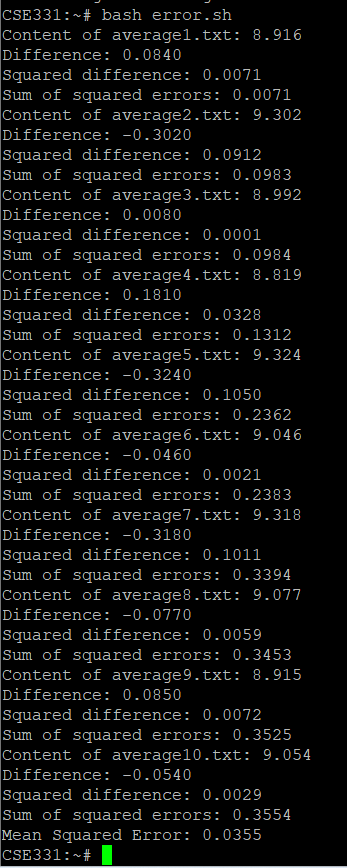
User5 -> 3 process

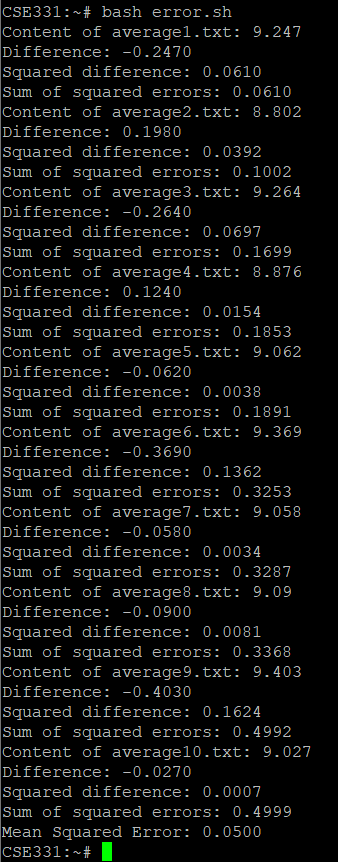
 **Figure 3.1.2.5.1** Mean Square Error of u1p1 **Figure 3.1.2.5.2** Mean Square Error of u2p1 

**Figure 3.1.2.5.3** Mean Square Error of u2p2 **Figure 3.1.2.5.4** Mean Square Error of u3p1

 **Figure 3.1.2.5.5** Mean Square Error of u4p1 **Figure 3.1.2.5.6** Mean Square Error of u4p2

 **Figure 3.1.2.5.7** Mean Square Error of u4p3 **Figure 3.1.2.5.8** Mean Square Error of u4p4

**Figure 3.1.2.5.9** Mean Square Error of u5p1 **Figure 3.1.2.5.10** Mean Square Error of u5p2



**Figure 3.1.2.5.11** Mean Square Error of u5p3

**3.2.** **FAIR-SHARE ALGORITHM**

**3.2.1. AVERAGE CPU USAGE**

**3.2.1.1. TestCase-1:**

In test case-1 , we have 3 users.

User1 -> 3 processes

User2 -> 2 processes

User3 -> 1 process



**Figure 3.2.1.1.1** Average CPU Usage of u1p1



**Figure 3.2.1.1.2** Average CPU Usage of u1p2



**Figure 3.2.1.1.3** Average CPU Usage of u1p3



**Figure 3.2.1.1.4** Average CPU Usage of u2p1



**Figure 3.2.1.1.5** Average CPU Usage of u2p2



**Figure 3.2.1.1.6** Average CPU Usage of u3p1

**3.2.1.2.** **TestCase-2:**

In test case-2 , we have 4 users.

User1 -> 3 processes

User2 -> 2 processes

User3 -> 1 process

User4 -> 4 processes



**Figure 3.2.1.2.1** Average CPU Usage of u1p1



**Figure 3.2.1.2.2** Average CPU Usage of u1p2



**Figure 3.2.1.2.3** Average CPU Usage of u1p3



**Figure 3.2.1.2.4** Average CPU Usage of u2p1



**Figure 3.2.1.2.5** Average CPU Usage of u2p2



**Figure 3.2.1.2.6** Average CPU Usage of u3p1



**Figure 3.2.1.2.7** Average CPU Usage of u4p1



**Figure 3.2.1.2.8** Average CPU Usage of u4p2



**Figure 3.2.1.2.9** Average CPU Usage of u4p3



**Figure 3.2.1.2.10** Average CPU Usage of u4p4

**3.2.1.3.** **TestCase-3:**

In test case-3 , we have 2 users.

User1 -> 2 processes

User2 -> 3 processes



**Figure 3.2.1.3.1** Average CPU Usage of u1p1



**Figure 3.2.1.3.2** Average CPU Usage of u1p2



**Figure 3.2.1.3.3** Average CPU Usage of u2p1



**Figure 3.2.1.3.4** Average CPU Usage of u2p2



**Figure 3.2.1.3.5** Average CPU Usage of u2p3

**3.2.1.4. TestCase-4:**

In test case-1 , we have 3 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process



**Figure 3.2.1.4.1** Average CPU Usage of u1p1



**Figure 3.2.1.4.2** Average CPU Usage of u2p1



**Figure 3.2.1.4.3** Average CPU Usage of u2p2



**Figure 3.2.1.4.4** Average CPU Usage of u3p1

**3.2.1.5. TestCase-5:**

In test case-1 , we have 5 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process

User4 -> 4 process

User5 -> 3 process



**Figure 3.2.1.5.1** Average CPU Usage of u1p1



**Figure 3.2.1.5.2** Average CPU Usage of u2p1



**Figure 3.2.1.5.3** Average CPU Usage of u2p2



**Figure 3.2.1.5.4** Average CPU Usage of u3p1



**Figure 3.2.1.5.5** Average CPU Usage of u4p1



**Figure 3.2.1.5.6** Average CPU Usage of u4p2



**Figure 3.2.1.5.7** Average CPU Usage of u4p3



**Figure 3.2.1.5.8** Average CPU Usage of u4p4



**Figure 3.2.1.5.9** Average CPU Usage of u5p1



**Figure 3.2.1.5.10** Average CPU Usage of u5p2



**Figure 3.2.1.5.11** Average CPU Usage of u5p3

**3.2.2. MEAN SQUARE ERROR**

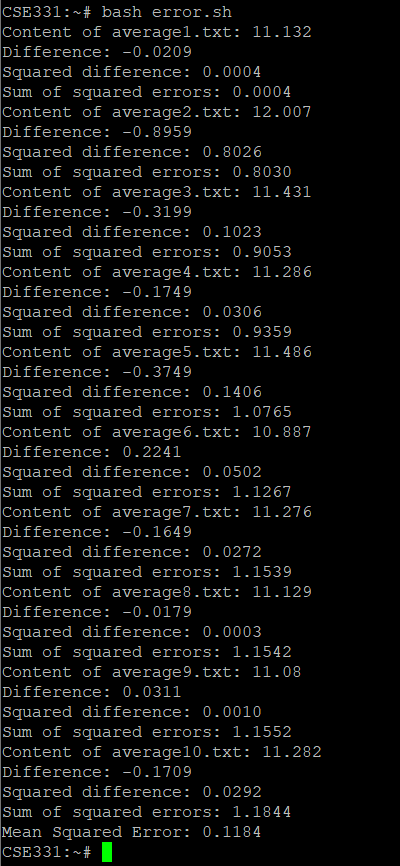
**3.2.2.1. TestCase-1:**

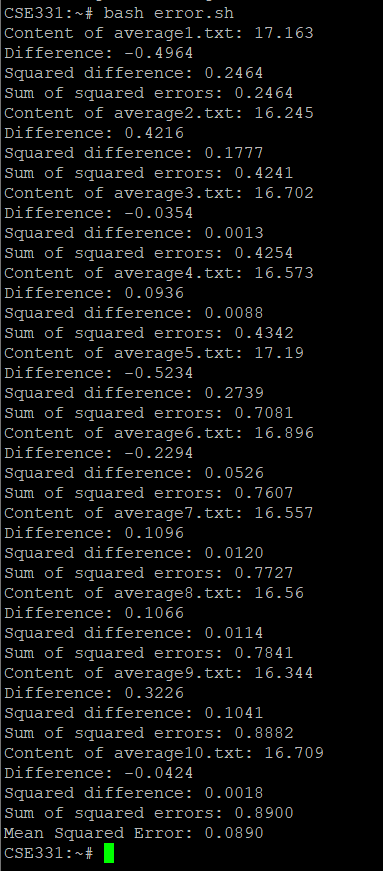
In test case-1 , we have 3 users.

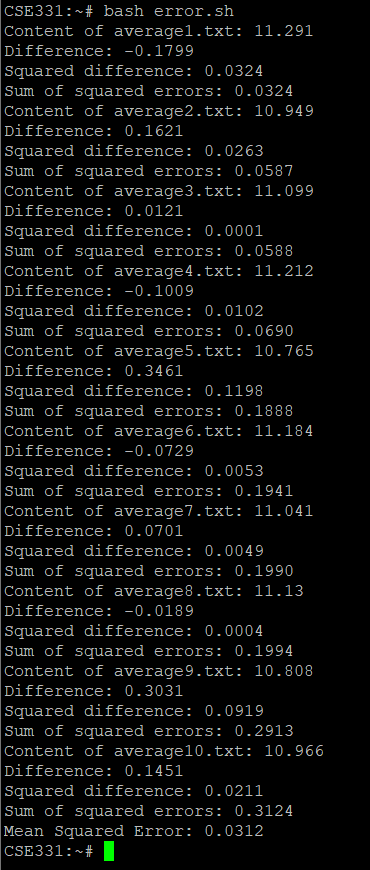
User1 -> 3 processes

User2 -> 2 processes

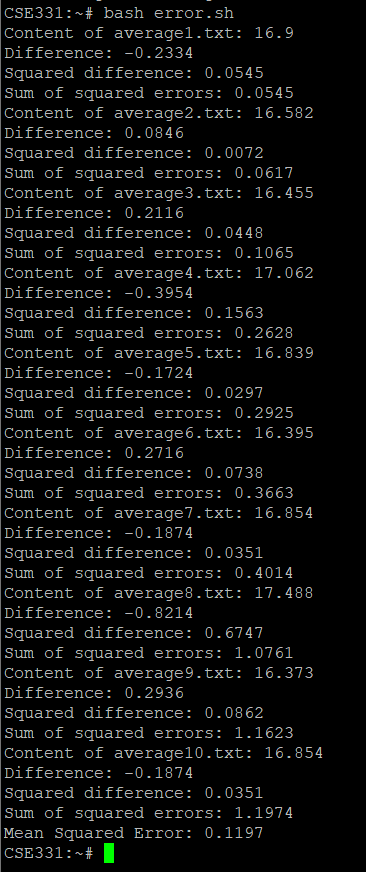
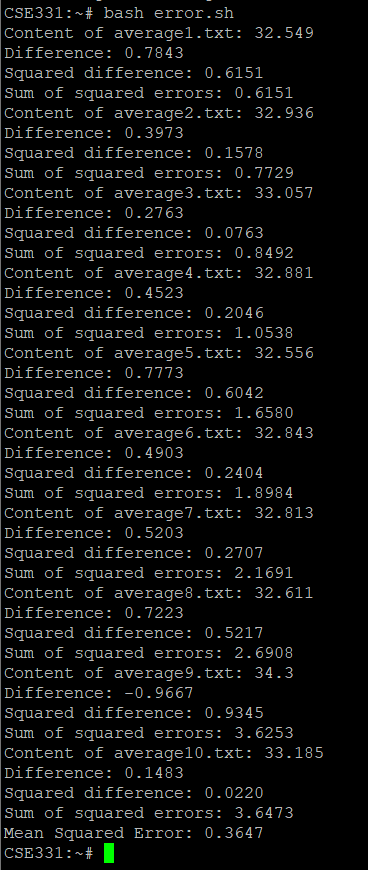
User3 -> 1 process

 **Figure 3.2.2.1.1** Mean Square Error of u1p1 **Figure 3.2.2.1.2** Mean Square Error of u1p2





**Figure 3.2.2.1.3** Mean Square Error of u1p3 **Figure 3.2.2.1.4** Mean Square Error of u2p1

 **Figure 3.2.2.1.5** Mean Square Error of u2p2 **Figure 3.2.2.1.6** Mean Square Error of u3p1

**3.2.2.2. TestCase-2:**

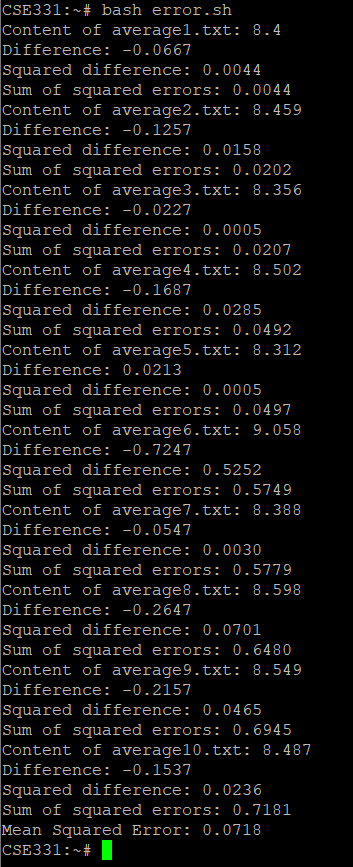
In test case-1 , we have 4 users.

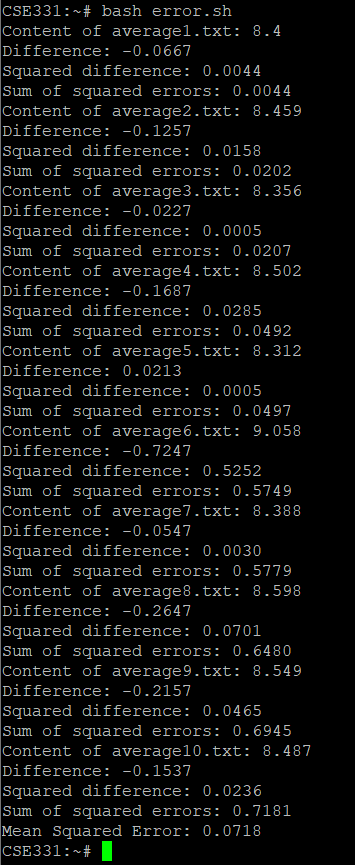
User1 -> 3 processes

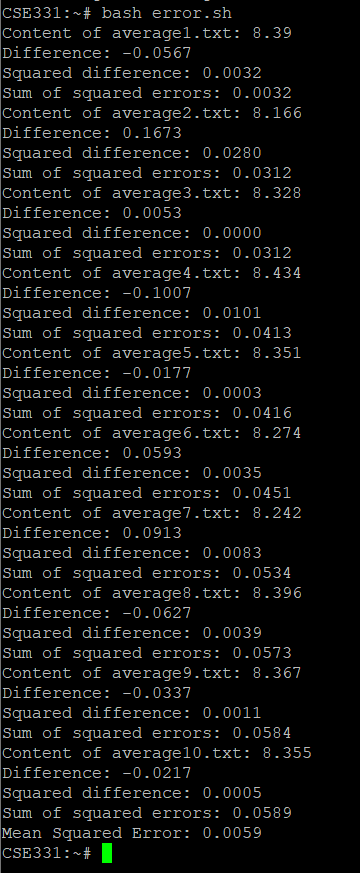
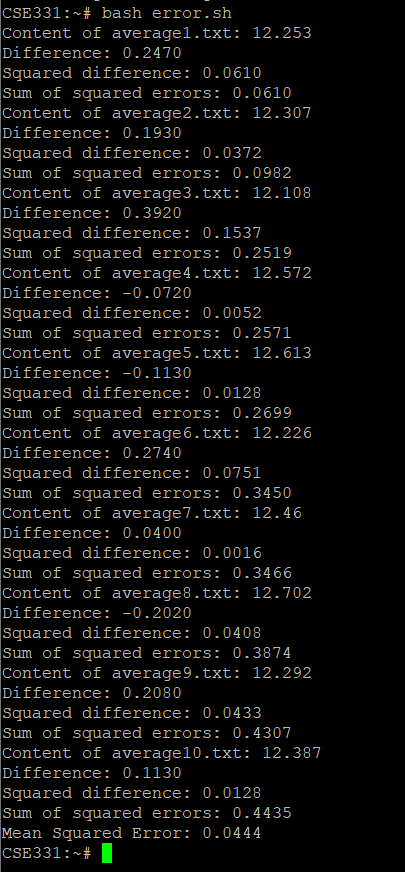
User2 -> 2 processes

User3 -> 1 process

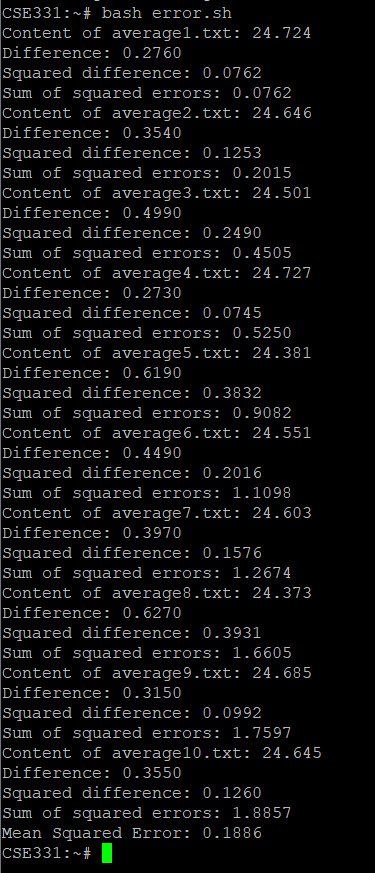
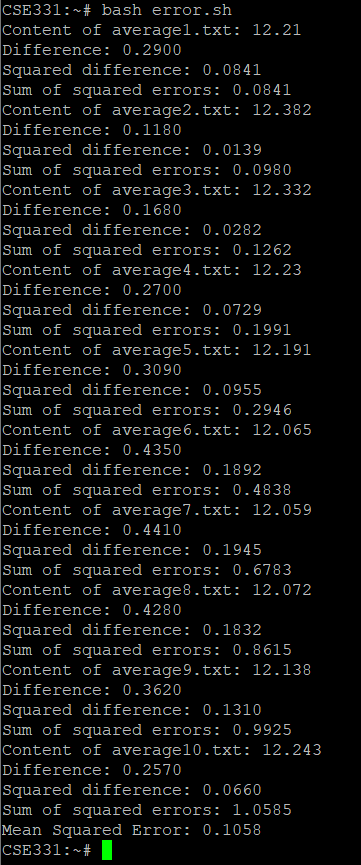
User4 -> 4 processes

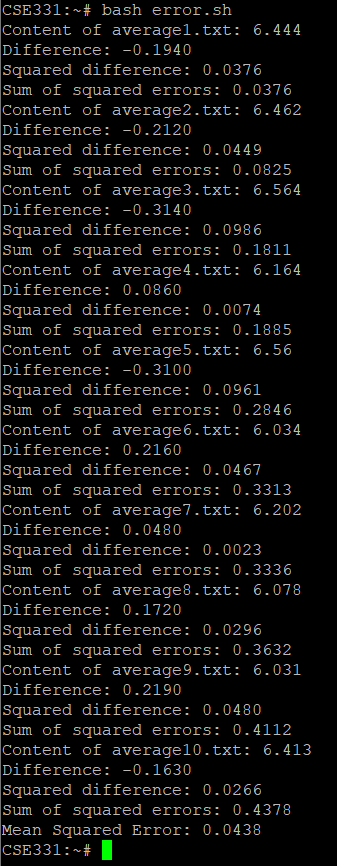
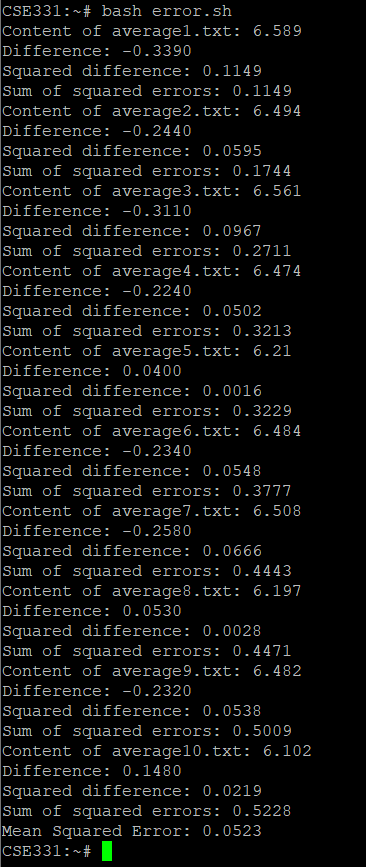


 **Figure 3.2.2.2.1** Mean Square Error of u1p1 **Figure 3.2.2.2.2** Mean Square Error of u1p2

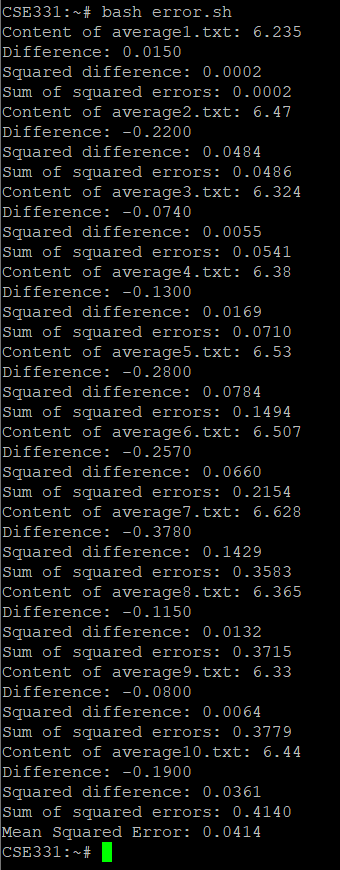
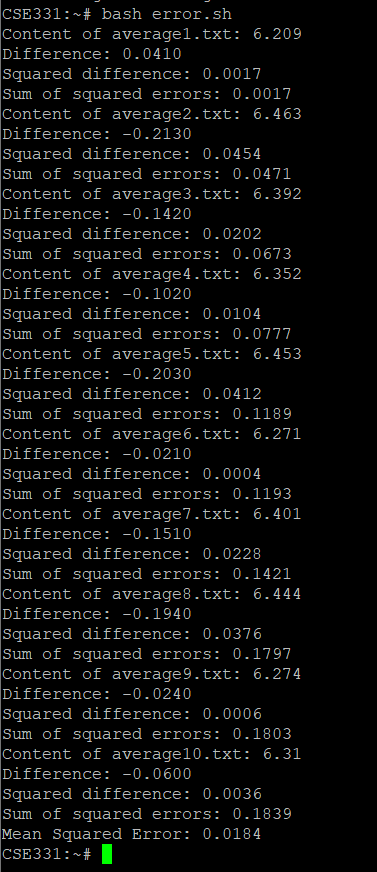
 

**Figure 3.2.2.2.3** Mean Square Error of u1p3 **Figure 3.2.2.2.1** Mean Square Error of u2p1

**Figure 3.2.2.2.5** Mean Square Error of u2p2 **Figure 3.2.2.2.6** Mean Square Error of u3p1

**Figure 3.2.2.2.7** Mean Square Error of u4p1 **Figure 3.2.2.2.8** Mean Square Error of u4p2

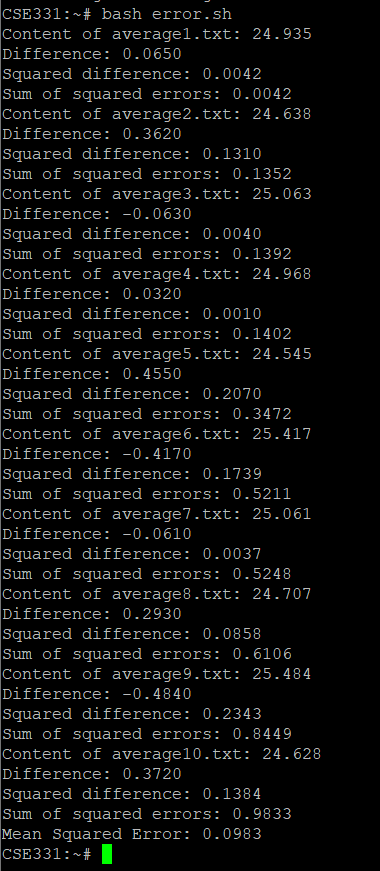
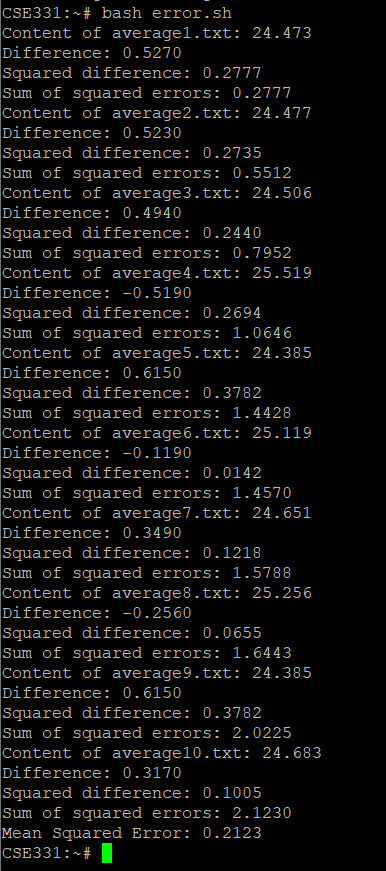
 **Figure 3.2.2.2.9** Mean Square Error of u4p3 **Figure 3.2.2.2.10** Mean Square Error of u4p4

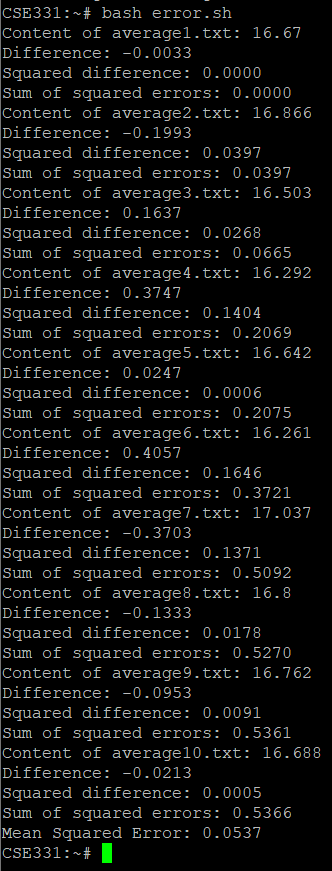
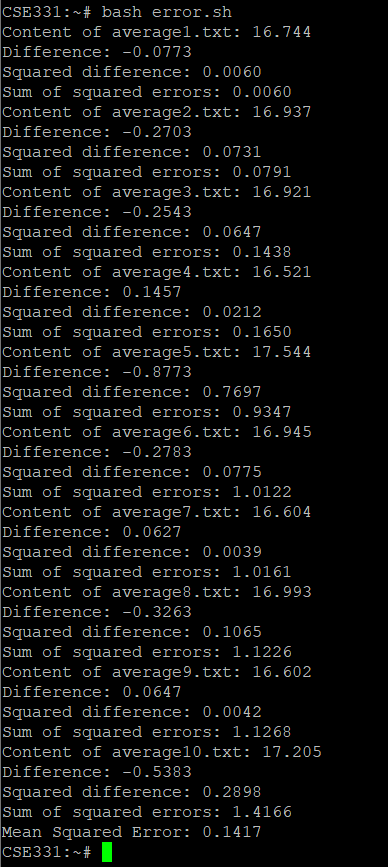
**3.2.2.3. TestCase-3:**

In test case-1 , we have 2 users.

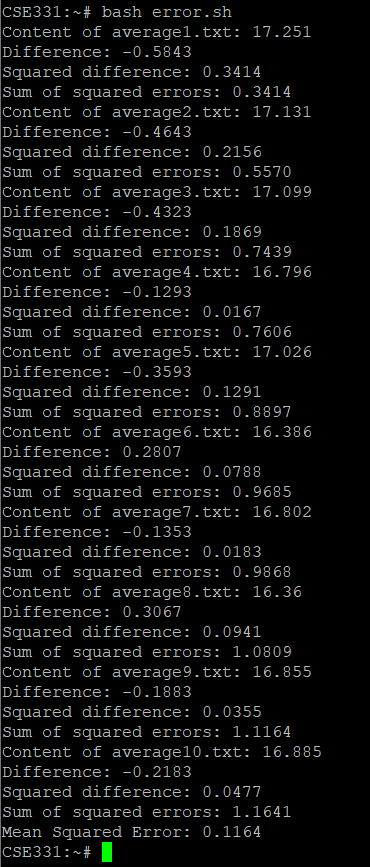
User1 -> 2 processes

User2 -> 3 processes

 **Figure 3.2.2.3.1** Mean Square Error of u1p1 **Figure 3.2.2.3.2** Mean Square Error of u1p2

**Figure 3.2.2.3.3** Mean Square Error of u2p1 **Figure 3.2.2.3.4** Mean Square Error of u2p2



**Figure 3.2.2.3.5** Mean Square Error of u2p3

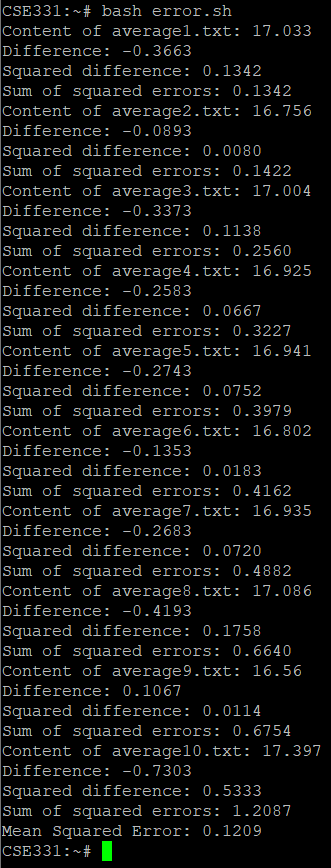
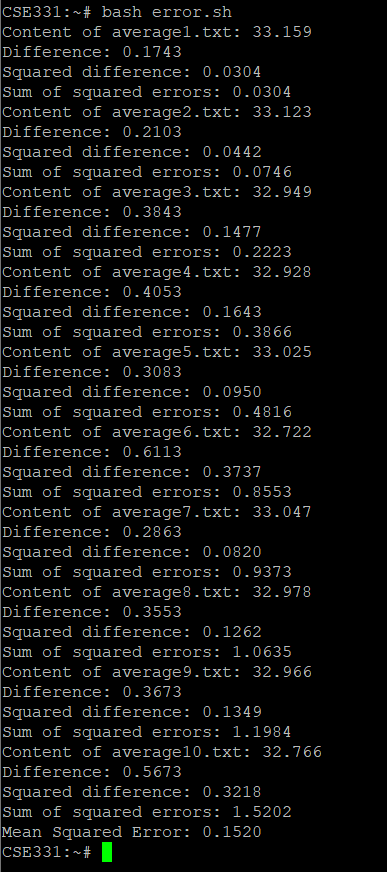
**3.2.2.4. TestCase-4:**

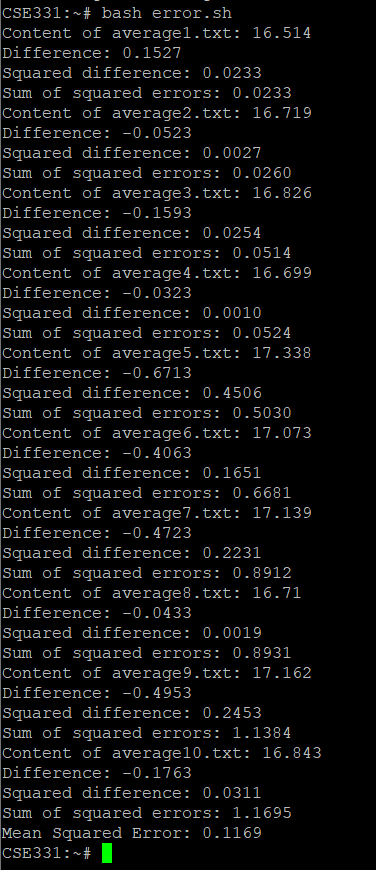
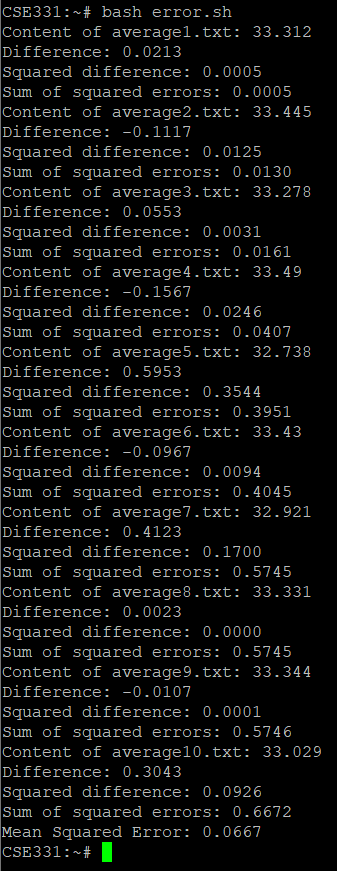
In test case-1 , we have 3 users.

User1 -> 1 process

User2 -> 2 processes

User3 -> 1 process

**Figure 3.2.2.4.1** Mean Square Error of u1p1 **Figure 3.2.2.4.2** Mean Square Error of u2p1

 **Figure 3.2.2.4.3** Mean Square Error of u2p2 **Figure 3.2.2.4.4** Mean Square Error of u3p1

**3.2.2.5. TestCase-5:**

In test case-1 , we have 5 users.

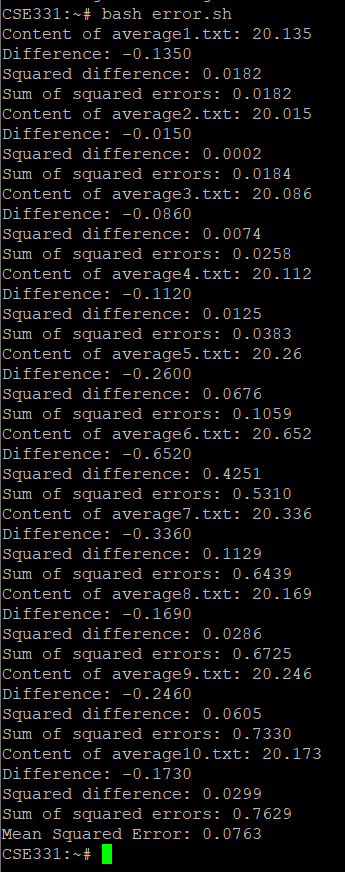
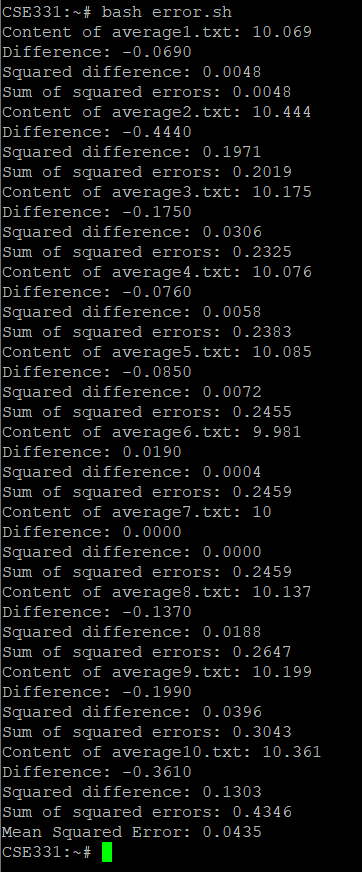
User1 -> 1 process

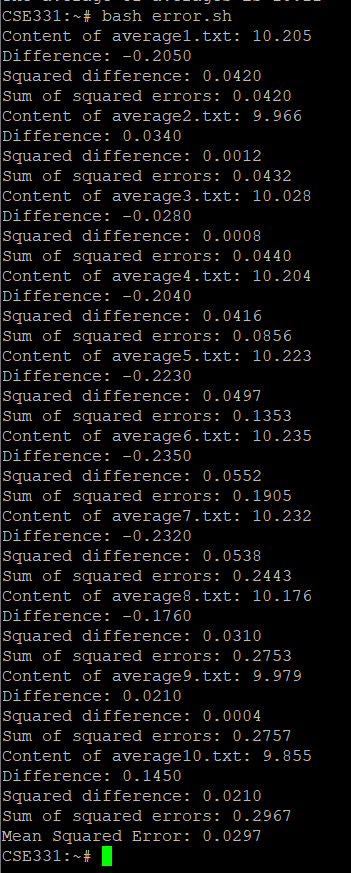
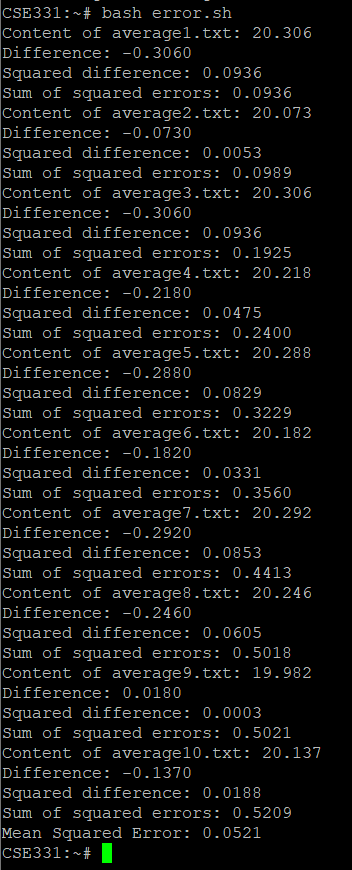
User2 -> 2 processes

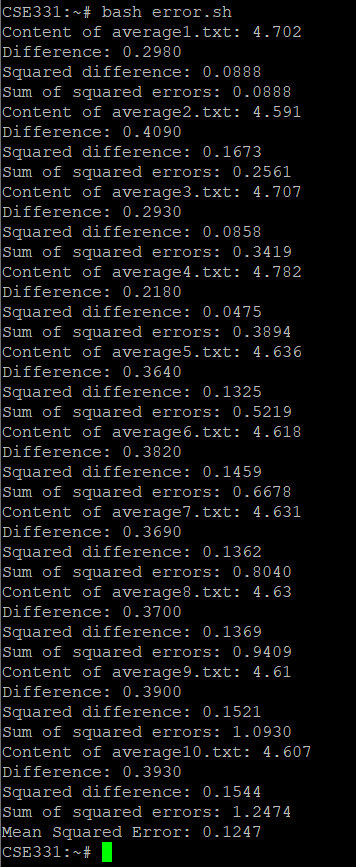
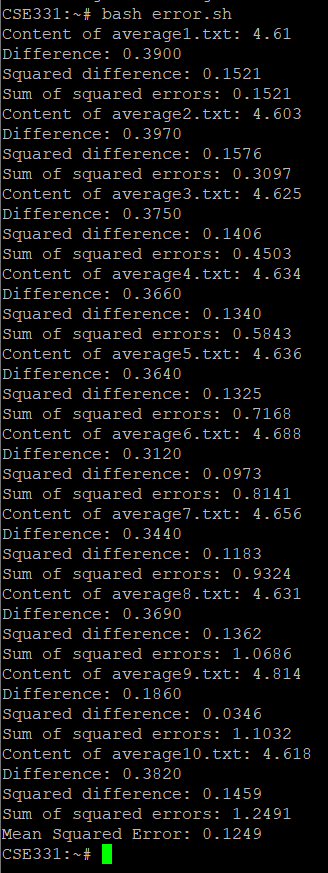
User3 -> 1 process

User4 -> 4 processes

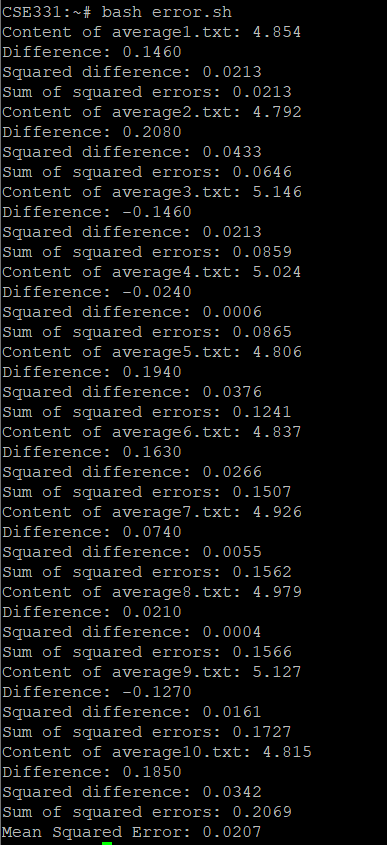
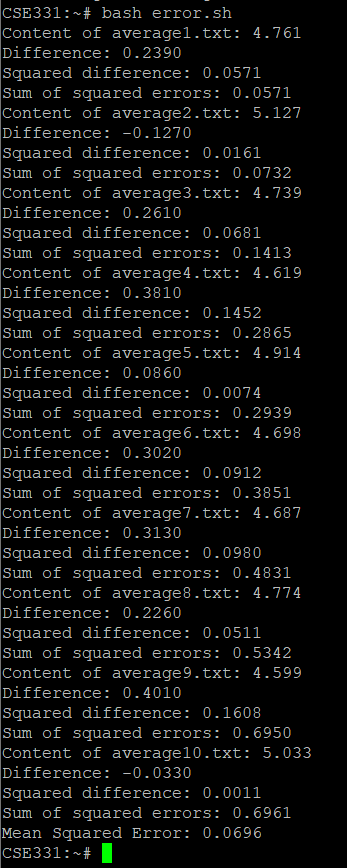
User5 -> 3 processes

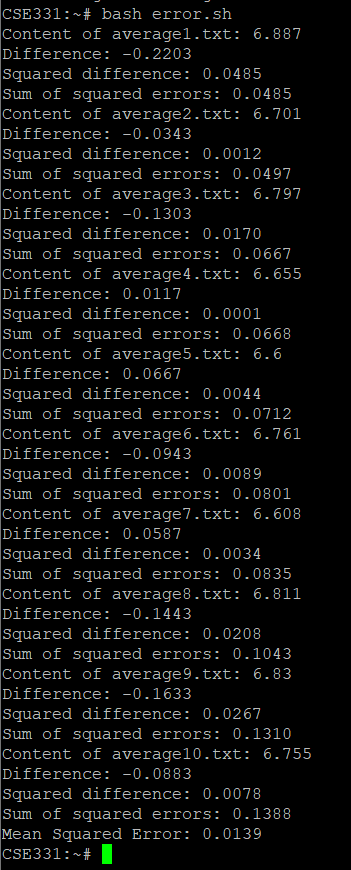
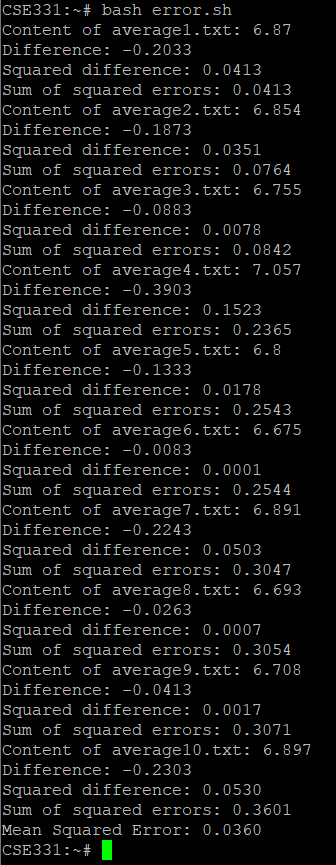
 **Figure 3.2.2.5.1** Mean Square Error of u1p1  **Figure 3.2.2.5.2** Mean Square Error of u2p1

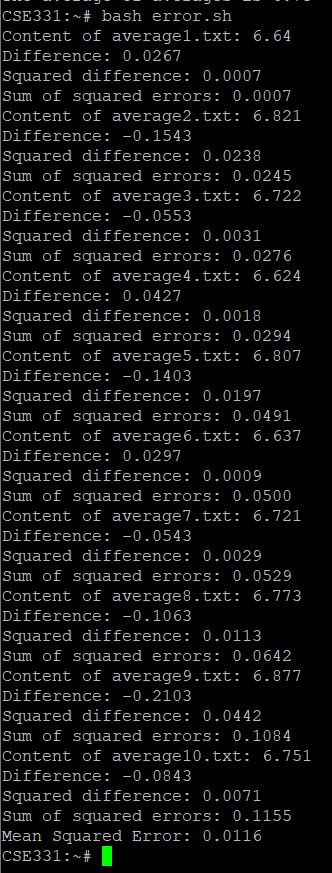
 **Figure 3.2.2.5.3** Mean Square Error of u2p2  **Figure 3.2.2.5.4** Mean Square Error of u3p1

**Figure 3.2.2.5.5** Mean Square Error of u4p1 **Figure 3.2.2.5.6** Mean Square Error of u4p2

 **Figure 3.2.2.5.7** Mean Square Error of u4p3 **Figure 3.2.2.5.8** Mean Square Error of u4p4

 **Figure 3.2.2.5.9** Mean Square Error of u5p1  **Figure 3.2.2.5.10** Mean Square Error of u5p2

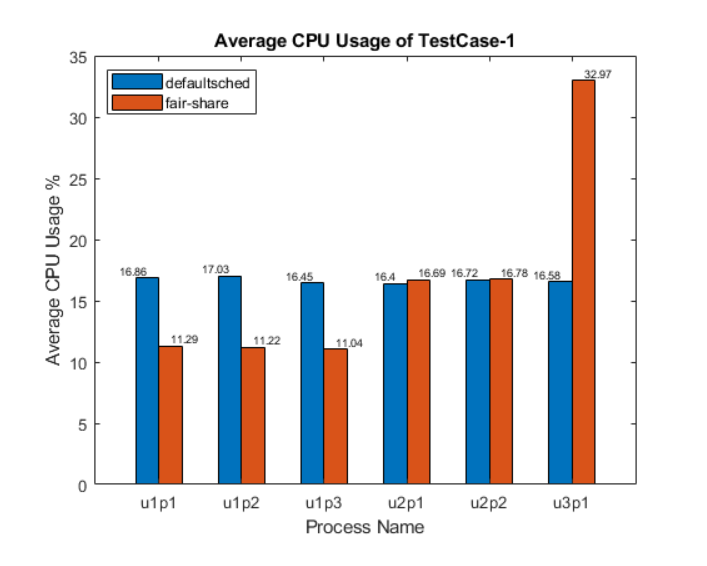


**Figure 3.2.2.5.11** Mean Square Error of u5p3

**3.3 GRAPHS**

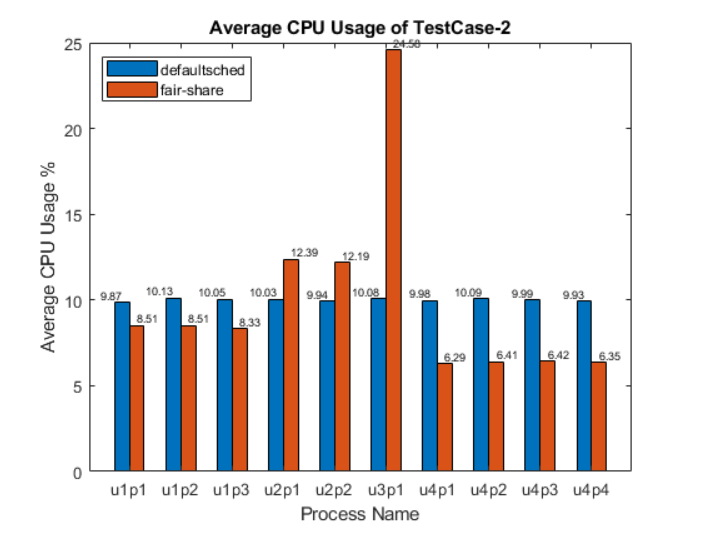
**3.3.1. Average CPU Usage**

**3.3.1.1. Testcase-1:**

****

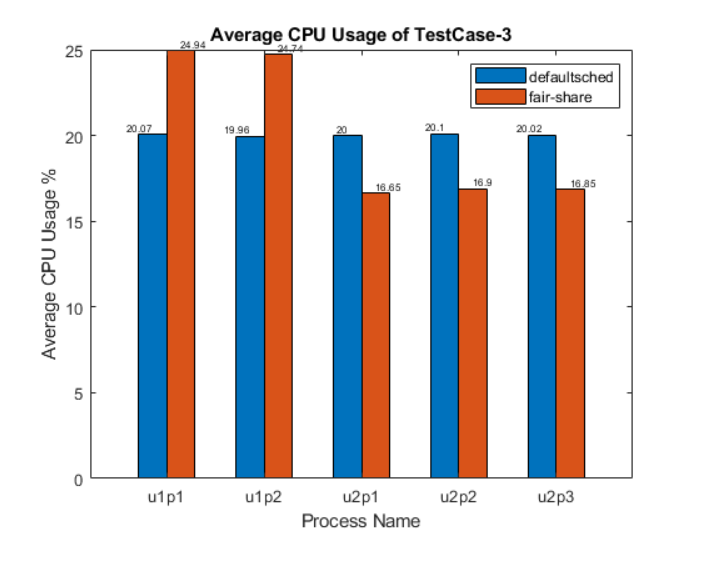
**Figure 3.3.1.1.1.** Graph of TestCase-1(Average CPU Usage)

**3.3.1.2. Testcase-2:**

****

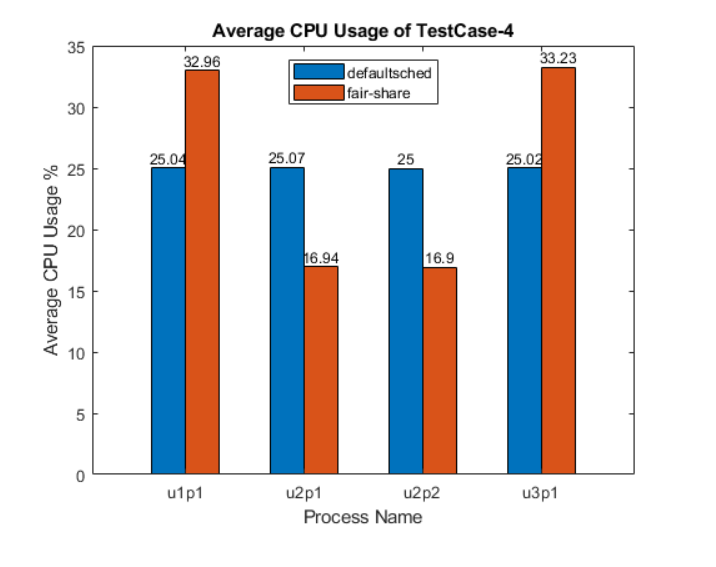
**Figure 3.3.1.2.1** Graph of TestCase-2(Average CPU Usage)

**3.3.1.3. Testcase-3:**

****

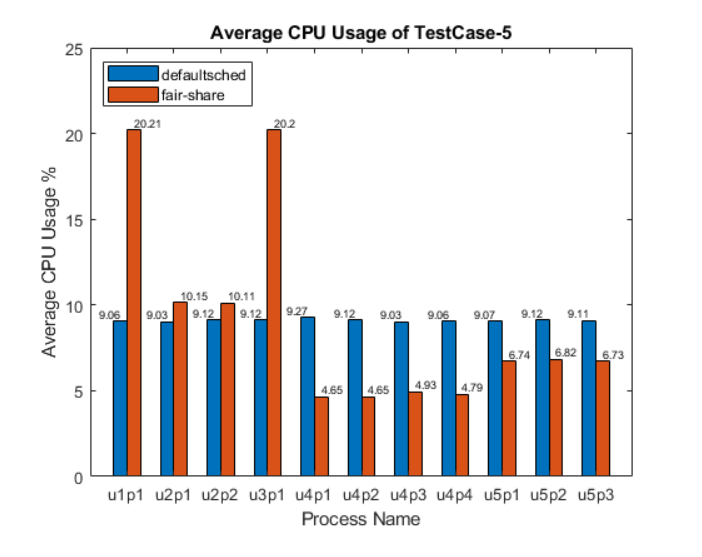
**Figure 3.3.1.3.1.** Graph of TestCase-3(Average CPU Usage)

**3.3.1.4. Testcase-4:**

****

**Figure 3.3.1.4.1.** Graph of TestCase-4(Average CPU Usage)

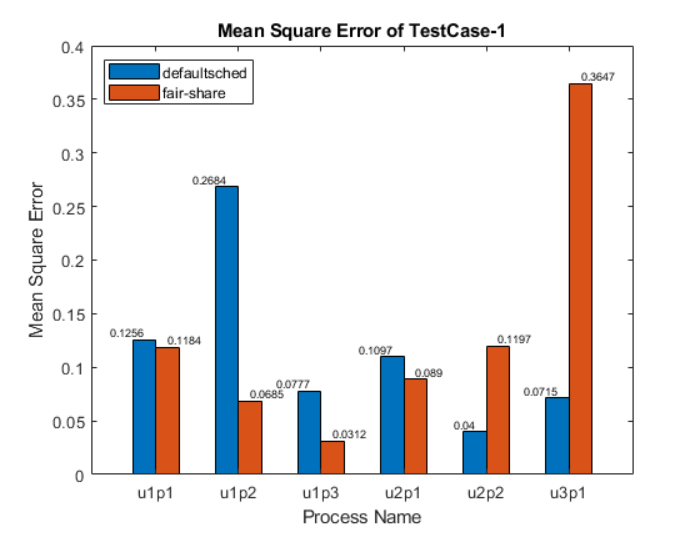
**3.3.1.5. Testcase-5:**

****

**Figure 3.3.1.5.1.** Graph of TestCase-5(Average CPU Usage)

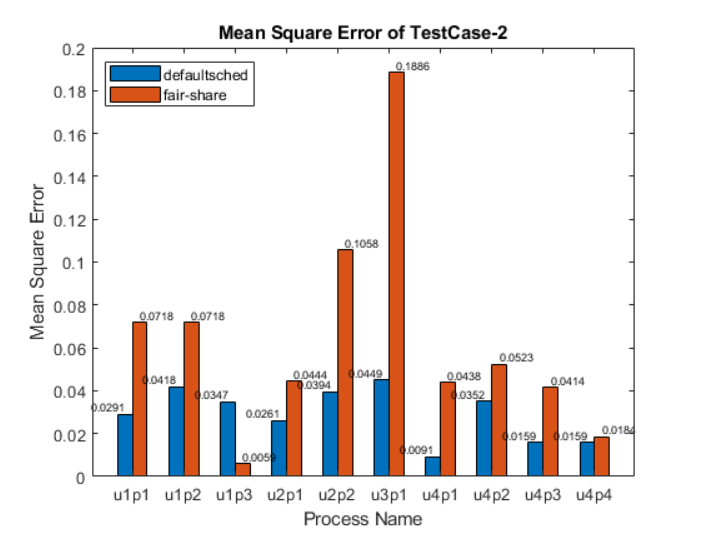
**3.3.2. Mean Square Error**

**3.3.2.1 TestCase-1:**

****

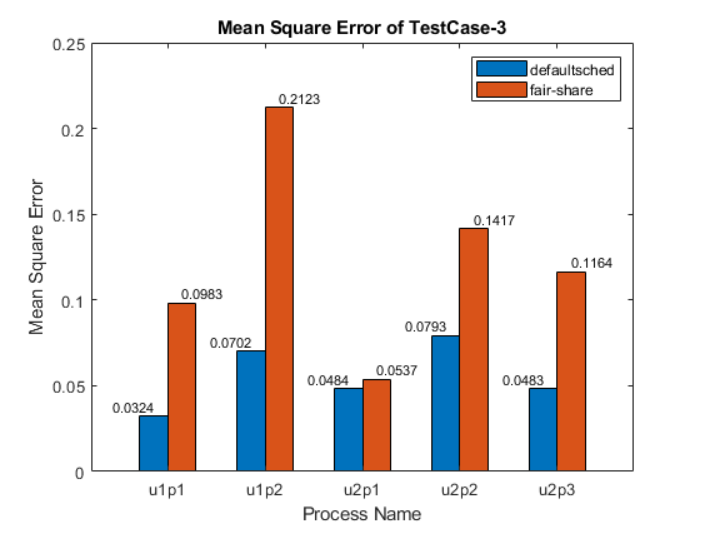
**Figure 3.3.2.1.1.** Graph of TestCase-1(Mean Square Error)

**3.3.2.2 TestCase-2:**

****

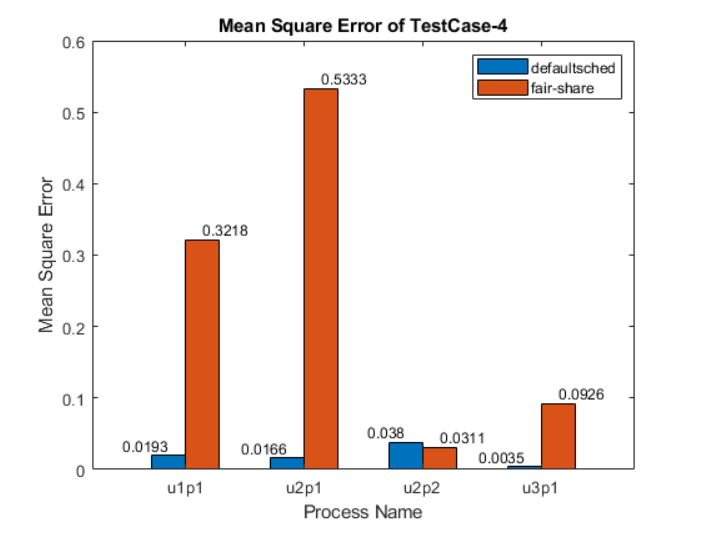
**Figure 3.3.2.2.1.** Graph of TestCase-2(Mean Square Error)

**3.3.2.3 TestCase-3:**

****

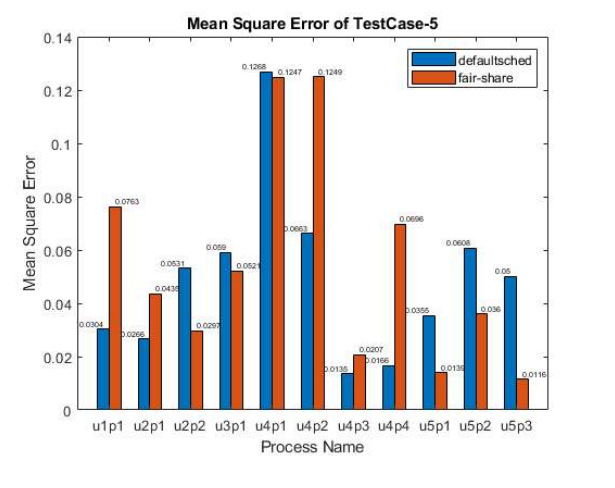
**Figure 3.3.2.3.1.** Graph of TestCase-3(Mean Square Error)

**3.3.2.4 TestCase-4:**

****

**Figure 3.3.2.4.1.** Graph of TestCase-4(Mean Square Error)

**3.3.2.5 TestCase-5:**

****

**Figure 3.3.2.5.1.** Graph of TestCase-5(Mean Square Error)

# CONCLUSION

The Linux operating system has been widely adopted due to its flexibility, open-source nature, and robust performance. However, managing system resources fairly among multiple users has been a challenge. In this report, we present the implementation of a user-based fair process divider scheduler, which aims to improve resource allocation and enhance the overall user experience in Linux systems.

The primary objective of this project is to develop and implement a user-based fair process divider scheduler that ensures equitable distribution of system resources among users, while maintaining optimal system performance and security.

* Analyze the existing Linux scheduler and identify areas for improvement.
* Design a user-based fair process divider scheduler that allocates resources fairly among users.
* Implement the scheduler in a Linux test environment.
* Evaluate the performance and resource allocation of the new scheduler compared to the existing one.
* Document the implementation process and results.

After implementing the user-based fair process divider scheduler, we observed the following improvements:

* Fair distribution of system resources: The new scheduler ensures that each user receives an equitable share of system resources, preventing any single user from monopolizing them.
* Improved system performance: By allocating resources fairly, the scheduler prevents resource contention and optimizes overall system performance.
* Enhanced user experience: Users can now manage their own scheduling priorities and tasks, allowing for a more personalized experience.
* Better resource management: System administrators can more effectively monitor and control resource usage, ensuring optimal system performance and security.

The implementation of a user-based fair process divider scheduler in Linux systems has successfully addressed the challenge of equitable resource allocation among multiple users. This new scheduler not only improves system performance and security but also enhances the user experience by allowing users to manage their own scheduling priorities and tasks.We believe that the widespread adoption of user-based fair process divider schedulers in Linux systems will significantly improve operating system performance and user experience for both individual users and enterprise customers. By ensuring fair distribution of system resources, this scheduler paves the way for a more powerful and flexible operating system that caters to the diverse needs of its users.

# REFERENCES

<https://elixir.bootlin.com/linux/2.4.27/source>

CSE 331 Operating System Design Course Materials (COPY TO/FROM USER.pdf)

O’Gorman, J. (2003). The linux process manager: The internals of scheduling, interrupts, and signals. Wiley. (p. 177-181)

Bovet, D. P., & Cesati, M. (2006). *Understanding the linux kernel: From I/O ports to process management ; covers version 2.6*. O’Reilly. (p. 258-270)