Name: Sumedh Kate

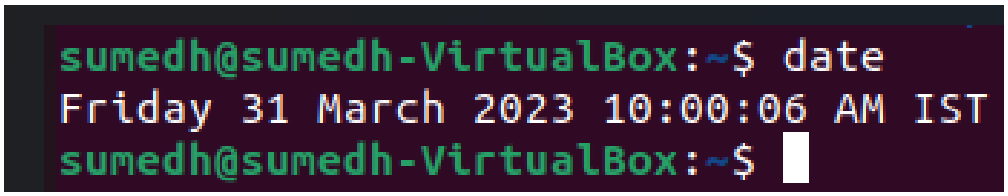
SY_CS-B

Roll No: 52

OS_Lab1: Linux Commands

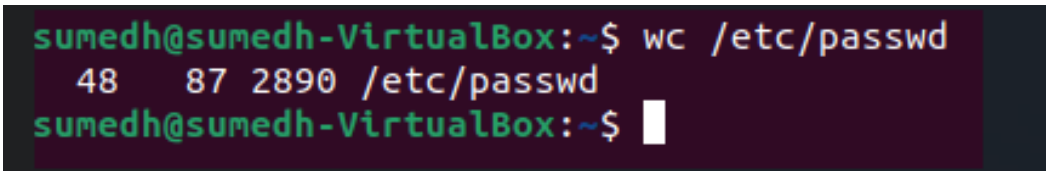
Objectives: This lab assignment aims to use linux commands to perform various operations.

1. Display the system's date.



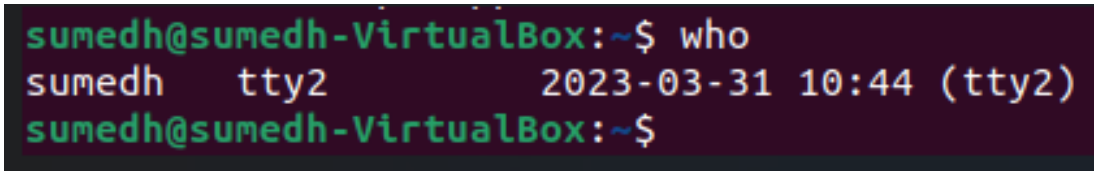
The date command can be used to print current date in linux.

2. Count the number of lines in the /etc/passwd file.



wc command is used to find out number of lines, word count, byte and characters count.

3. Find out who else is on the system.



who command prints the users logged in in the system.

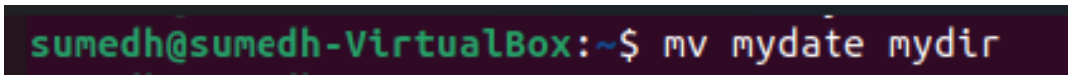
4. Direct the output of the man pages for the date command to a file named mydate.

man command displays the user manual on terminal.

5. Create a subdirectory called mydir

mkdir is command to create a new directory

6. Move the file mydate into the new subdirectory.



To move files from a specific location to destination mv command is used.

7. Go to the subdirectory mydir and copy the file mydate to a new file called ourdate

```
sumedh@sumedh-VirtualBox:~$ touch ourdate.txt
sumedh@sumedh-VirtualBox:~$ cd mydir
sumedh@sumedh-VirtualBox:~/mydir$ cp mydate ourdate
```

touch commmand is used to create a new file, cd is used to change the directory and cp is used to copy data from one file to another file

8. List the contents of mydir.

```
sumedh@sumedh-VirtualBox:~$ cd mydir
sumedh@sumedh-VirtualBox:~/mydir$ ls
mydate  ourdate
sumedh@sumedh-VirtualBox:~/mydir$
```

ls is used to list all files present in current working directory

9. Do a long listing on the file ourdate and note the permissions.

```
sumedh@sumedh-VirtualBox:~$ cd mydir
sumedh@sumedh-VirtualBox:~/mydir$ ls -la
total 24
drwxrwxr-x  2 sumedh sumedh 4096 Mar 31 10:51 .
drwxr-x--- 18 sumedh sumedh 4096 Mar 31 10:51 ..
-rw-rw-r--  1 sumedh sumedh 6509 Mar 31 10:49 mydate
-rw-rw-r--  1 sumedh sumedh 6509 Mar 31 10:51 ourdate
sumedh@sumedh-VirtualBox:~/mydir$
```

ls -la display detailed information of files.

10. Display the name of the current directory starting from the root.

```
sumedh@sumedh-VirtualBox:~/mydir$ pwd
/home/sumedh/mydir
sumedh@sumedh-VirtualBox:~/mydir$
```

pwd refers to present working directory, it prints path to the directory in which we are currently working.

11. List all the files in your HOME directory.

```
sumedh@sumedh-VirtualBox:~$ ls
bubblesort.sh  Desktop                 factorial.sh  mystates       Public
capital        Documents               fibonacci.sh  ourdate.txt    reverse.sh
capital1       Downloads               Music         palindrome.sh  snap
capital2       evenodd.sh              mycapitals    Pictures       Templates
capital3       factorial_recursion.sh  mydate        prime.sh       Videos
sumedh@sumedh-VirtualBox:~$
```

ls command is used in home directory to list all the files.

12. Display the first 5 lines of mydate

```
sumedh@sumedh-VirtualBox:~$ cd mydir
sumedh@sumedh-VirtualBox:~/mydir$ head -5 mydate
DATE(1)                         User Commands                         DATE(1)

NAME
       date - print or set the system date and time

sumedh@sumedh-VirtualBox:~/mydir$
```

head command is used to print the desired number of lines from the top of the file.

13. Display the last 8 lines of mydate.

```
sumedh@sumedh-VirtualBox:~/mydir$ tail -8 mydate
       This  is  free  software:  you are free to change and redistribute it.
       There is NO WARRANTY, to the extent permitted by law.

SEE ALSO
       Full documentation <https://www.gnu.org/software/coreutils/date>
       or available locally via: info '(coreutils) date invocation'

GNU coreutils 8.32              February 2022                        DATE(1)
sumedh@sumedh-VirtualBox:~/mydir$
```

tail command is used to print the desired number of lines from the bottom of the file.

14. Remove the directory mydir.

```
sumedh@sumedh-VirtualBox:~$ rmdir mydir
```

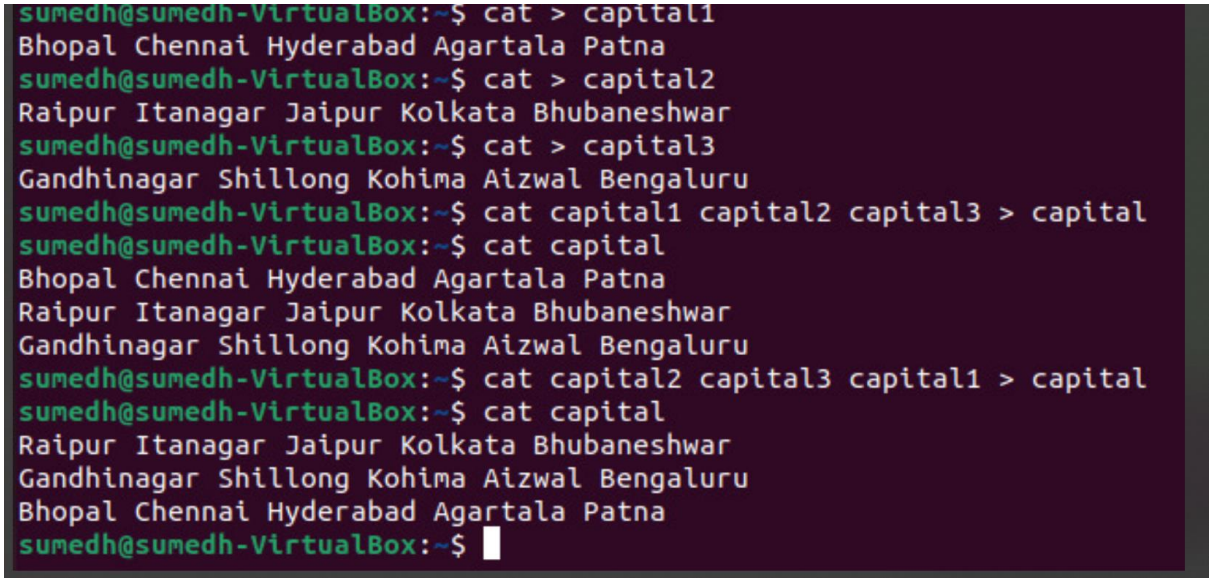
Command rmdir is used to remove the directory or delete the directory.

15. Redirect the output of the long listing of files to a file named *list*.

```
sumedh@sumedh-VirtualBox:~$ ls > list
sumedh@sumedh-VirtualBox:~$ cat > list
cases.sh
factorial.sh
sumedh@sumedh-VirtualBox:~$
```

To redirect the contents from one file to another “>” symbol is used.

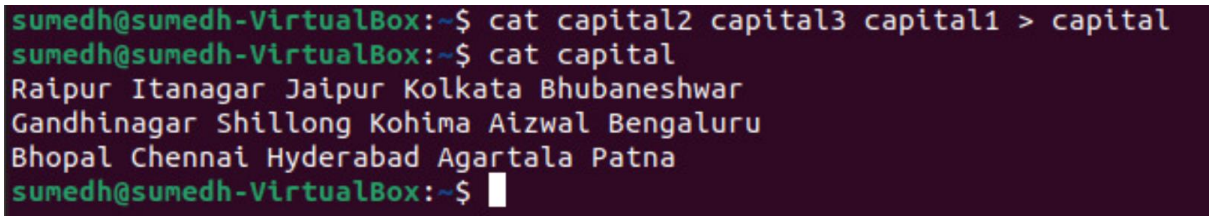
16. Select any 5 capitals of states in India and enter them in a file named *capitals1*. Choose 5 more capitals and enter them in a file named *capitals2*. Choose 5 more capitals and enter them in a file named *capitals3*. Concatenate all 3 files and redirect the output to a file named *capitals*.



cat command can be used to store the data in file by using syntax cat>filename

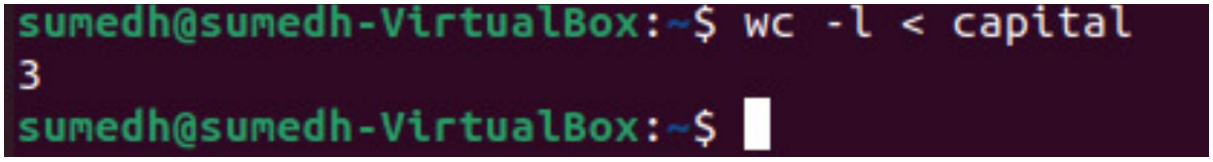
cat can be used to print the contents of the file by using syntax cat filename.

17. Concatenate the file *capitals2* at the end of file *capitals*



cat command can be used to concatenate the contents of multiple file.

18. Redirect the file *capitals* as an input to the command "wc –l".



wc -l tells us how big the selected document is

19. Give read and write permissions to all users for the file *capitals*.

```
sumedh@sumedh-VirtualBox:~$ chmod u+rw capital
```

chmod refers to change mode this command can be used to provide access of the files to user, owner for read, write, execute. Here u+rw means user is given an access for reading and writing.

20. Give read permissions only to the owner of the file *capitals*. Open the file, make some changes and try to save it. What happens ?

```
sumedh@sumedh-VirtualBox:~$ chmod o+r capital
```

Here owner is given access only to read the file

21. Create an alias to concatenate the 3 files *capitals1*, *capitals2*, *capitals3* and redirect the output to a file named *capitals*. Activate the alias and make it run.

```
sumedh@sumedh-VirtualBox:~$ alias c='cat capital3 capital2 capital1 > caital'
```

Alias is used to replace one string with another string.

22. What are the environment variables PATH, HOME and TERM set to on your terminal?

```
sumedh@sumedh-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/lo
cal/games:/snap/bin:/snap/bin
sumedh@sumedh-VirtualBox:~$ echo $HOME
/home/sumedh
sumedh@sumedh-VirtualBox:~$ echo $TERM
xterm-256color
sumedh@sumedh-VirtualBox:~$
```

touchcat

echo command prints the output on screen.

23. Find out the number of times the string “the” appears in the file *mydate*.

```
sumedh@sumedh-VirtualBox:~$ grep -c the mydate
4
```

grep command is used to find occurrence of specific word, letter in the given file.

Here grep -c given the number of times word “the” has occurred in file mydate

24. Find out the line numbers on which the string “date” exists in *mydate*.

```
sumedh@sumedh-VirtualBox:~$ grep -n 'the' mydate
1:the is the thing for which i have to the shit
2:this is the assignment for linux commands
3:the assignment is very important
4:thus the study is required as it carries the marks for the exam
```

Here grep -n print the lines in which word “date has occurred”.

25. Print all lines of *mydate* except those that have the letter “i” in them.

```
sumedh@sumedh-VirtualBox:~$ grep -v 'i' mydate
sumedh@sumedh-VirtualBox:~$ grep -v 'z' mydate
the is the thing for which i have to the shit
this is the assignment for linux commands
the assignment is very important
thus the study is required as it carries the marks for the exam
```

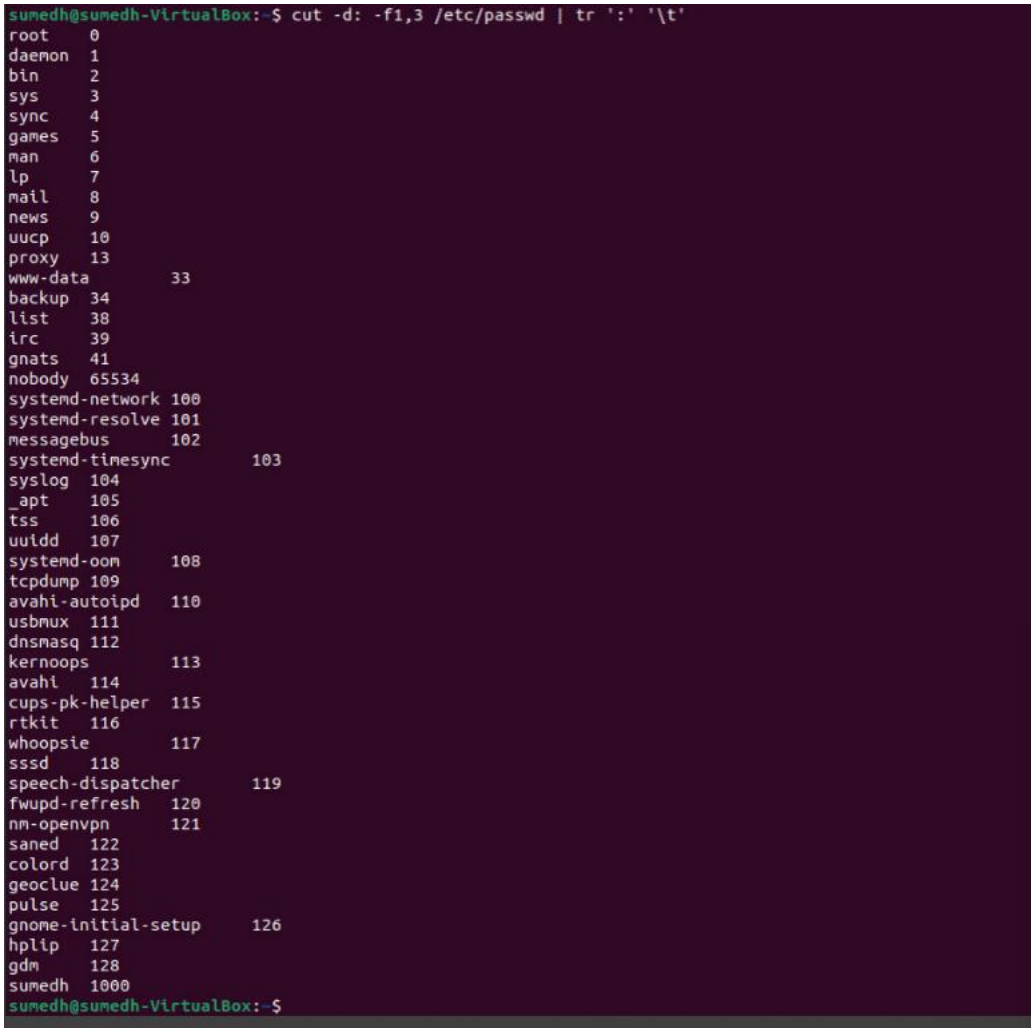
Here grep -v command print all the lines except the lines having letter “i” in them.

26. List 5 states in north east India in a file *mystates*. List their corresponding capitals in a file *mycapitals*. Use the *paste* command to join the 2 files.

```
sumedh@sumedh-VirtualBox:~$ cat >  mystates
Assam
Chhatisgarh
Madhya Pradesh
Tamil Nadu
Manipur
sumedh@sumedh-VirtualBox:~$ cat > mycapitals
Dispur
Raipur
Bhopal
Chennai
Imphal
sumedh@sumedh-VirtualBox:~$ paste mystates mycapitals
Assam   Dispur
Chhatisgarh     Raipur
Madhya Pradesh  Bhopal
Tamil Nadu      Chennai
Manipur Imphal
sumedh@sumedh-VirtualBox:~$
```

paste command can be used to paste the contents of one file into another.

27. Use the *cut* command to print the 1st and 3rd columns of the /etc/passwd file for all students in this class.



cut command is used to remove the unwanted columns, rows, data from the file.

28. Count the number of people logged in and also trap the users in a file using the *tee* command.

Wc -w prints the bytes.

Conclusion:

Utilising the command line is faster than using a GUI. Because there is less overhead, command-line programmes start faster than graphical ones. This is one of the reasons why, when Linux initially appeared on Computers, distributions defaulted to the console environment.

Name: Sumedh Kate

SY_CS-B

Roll No: 52

OS_Lab2: Shell Scripting

Objective: Objective of this lab assignment is to implement shell scripting to perform various operations.

A shell script is a text file containing a command sequence for a UNIX-based operating system. A shell script is so named because it condenses a series of commands that would otherwise have to be put into the keyboard one at a time into a single script. The shell is the command-line interface (CLI) and interpreter for the collection of commands used to communicate with the operating system. A shell script is typically written for command sequences that a user needs to use regularly to save time. The shell script, like other programmes, can include arguments, comments, and subcommands that the shell must follow. Users start the shell script's command sequence by simply typing file name on command-line

1. Find whether the entered number is even or odd

```
 GNU nano 6.2                                  evenodd.sh
echo  "Enter a number"
read n
if [ n%2==0 ]
then
        echo "$n is even"
else
        echo "$n is odd"
fi
```

```
sumedh@sumedh-VirtualBox:~$ nano evenodd.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x evenodd.sh
sumedh@sumedh-VirtualBox:~$ ./evenodd.sh
Enter a number
6
6 is even
```

Here for testing whether the number is even or odd if else loop using shell scripting is implemented.

Nano command is used to open the file and write in it.

The user need to be given an access to execute any operation hence chmod u+x is used

"./" is used before filename to print the output.

2. Reverse the number

```
  GNU nano 6.2                                  reverse.sh
echo "Enter a number"
read n
num=0
while [ $n -gt 0 ]
do
num=$(expr $num \* 10)
k=$(expr $n % 10)
num=$(expr $num + $k)
n=$(expr $n / 10)
done
echo "REverse of the number is $num"
```

```
sumedh@sumedh-VirtualBox:~$ nano reverse.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x reverse.sh
sumedh@sumedh-VirtualBox:~$ ./reverse.sh
Enter a number
678
./reverse.sh: line 4: while[ 678 -gt 0 ]: command not found
./reverse.sh: line 5: syntax error near unexpected token `do'
./reverse.sh: line 5: `do'
sumedh@sumedh-VirtualBox:~$ nano reverse.sh
sumedh@sumedh-VirtualBox:~$ ./reverse.sh
Enter a number
678
REverse of the number is 876
```

In this question we take a number input, a while loop is used in shell scripting to reverse the given number.

3. Check whether the number is prime or not

```
  GNU nano 6.2                                  prime.sh
#!/bin/bash
echo "Enter a number"
read n
for((i=2; i<=$n/2; i++))
do
  ans=$(( n%i ))
  if [ $ans -eq 0 ]
  then
    echo "$n is not a prime number."
    exit 0
  fi
done
echo "$n is a prime number."
```

```
sumedh@sumedh-VirtualBox:~$ nano prime.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x prime.sh
sumedh@sumedh-VirtualBox:~$ ./prime.sh
Enter a number
13
13 is a prime number.
sumedh@sumedh-VirtualBox:~$ ./prime.sh
Enter a number
15
15 is not a prime number.
```

Here for loop is implemented in shell scripting to find if the entered number is prime or not.

4. Check for palindrome number

```
  GNU nano 6.2                                                palindrome.sh *
echo "Enter a number"
read n
function pal
{
number=$n
reverse=0
while [ $n -gt 0 ]
do
a=`expr $n % 10 `
n=`expr $n / 10 `
reverse=`expr $reverse \* 10 + $a`
done
echo $reverse
if [ $number -eq $reverse ]
then
        echo "Number is palindrome"
else
        echo "Number is not palindrome"
fi
}
r=`pal $n`
echo "$r"
```

```
sumedh@sumedh-VirtualBox:~$ nano prime.sh
sumedh@sumedh-VirtualBox:~$ nano palindrome.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x palindrome.sh
sumedh@sumedh-VirtualBox:~$ ./palindrome.sh
Enter a number
12021
12021
Number is palindrome
sumedh@sumedh-VirtualBox:~$ 1234
1234: command not found
sumedh@sumedh-VirtualBox:~$ ./palindrome.sh
Enter a number
1234
4321
Number is not palindrome
```

While loop and if-else conditions are implemented to check if the entered number is palindrome or not

5. Factorial using recursion

```
 GNU nano 6.2                                          factorial_recursion.sh
#!/bin/bash


factorial()
{
        product=$1


        if((product <= 2))
        then
            echo "$product"
        else
            f=$((product -1))
f=$(factorial $f)
f=$((f*product))
echo $f
fi
}

echo "Enter a number"
read n

if((n == 0))
then
        echo 1
else
factorial $n
fi
```

```
sumedh@sumedh-VirtualBox:~$ nano factorial_recursion.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x factorial_recursion.sh
sumedh@sumedh-VirtualBox:~$ ./factorial_recursion.sh
Enter a number
6
720
```

Here a recursive function for factorial is created and called to print factorial of a given number.

6. Factorial without recursion

```
 GNU nano 6.2                                                  factorial.sh
echo "Enter a number"
read n
fact=1
for((i=2; i<=n; i++))
{
  fact=$((fact * i))
}
echo $fact
```

```
sumedh@sumedh-VirtualBox:~$ nano factorial.sh
sumedh@sumedh-VirtualBox:~$ ./factorial.sh
Enter a number
5
120
sumedh@sumedh-VirtualBox:~$
```

Here we use iterative method to find factorial of a given number.

7. Bubble Sort

```
 GNU nano 6.2                                        bubblesort.sh
arr=(10 8 20 100 12)

echo "Array in original order"
echo ${arr[*]}

# Performing Bubble sort
for ((i = 0; i<5; i++))
do

    for((j = 0; j<5-i-1; j++))
    do

        if [ ${arr[j]} -gt ${arr[$((j+1))]} ]
        then
            # swap
            temp=${arr[j]}
            arr[$j]=${arr[$((j+1))]}
            arr[$((j+1))]=$temp
        fi
    done
done

echo "Array in sorted order :"
echo ${arr[*]}
```

```
sumedh@sumedh-VirtualBox:~$ nano bubblesort.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x bubblesort.sh
sumedh@sumedh-VirtualBox:~$ ./bubblesort.sh
Array in original order
10 8 20 100 12
Array in sorted order :
8 10 12 20 100
```

An unsorted array is inserted and Bubble sort, sorting algorithm is used to sort the array.

8. Fibonacci Series

```
 GNU nano 6.2                                                fibonacci.sh
echo "Enter the value of n"
read n
a=0
b=1
count=2
echo "Fibonacci series:"
echo $a
echo $b
while [ $count -le $n ]
do
fib=`expr $a + $b`
a=$b
b=$fib
echo $fib
count=`expr $count + 1`
done
```

```
sumedh@sumedh-VirtualBox:~$ nano fibonacci.sh
sumedh@sumedh-VirtualBox:~$ chmod u+x fibonacii.sh
chmod: cannot access 'fibonacii.sh': No such file or directory
sumedh@sumedh-VirtualBox:~$ chmod u+x fibonacci.sh
sumedh@sumedh-VirtualBox:~$ ./fibonacci.sh
Enter the value of n
6
Fibonacci series:
0
1
1
2
3
5
8
```

Here we print the Fibonacci series using shell scripting.

Name: Sumedh Kate

SY_CS-B

Roll No: 52

OS_Lab3: Reader-Writer, Producer-Consumer, Dinning Philosopher Problems.

Objectives: This lab assignment aims to solve the reader-writer, producer-consumer, dinning philosopher problems using semaphore and mutex.

Semaphore and Mutex:

A semaphore is a variable used in the operating system to manage access to a shared resource, whereas a mutex is just a lock obtained before entering a crucial area and released. A semaphore is preferable for several instances of the same resource, but a mutex is preferable for a single shared resource.

1. **Reader-Writer:**
   The readers-writers dilemma is related to an item that is shared by numerous processes, such as a file. Some of these processes are readers, meaning they just want to read data from the object, while others are writers, meaning they want to write data into the object. The readers-writers problem is used to maintain synchronisation so that the object data is not corrupted. There is no difficulty, for example, if two readers access the object at the same time. However, there may be issues if two writers or a reader and writer access the item at the same time.
   To remedy this scenario, a writer should be granted exclusive access to an item, which means that no reader or writer should be able to access it while the writer is doing so. Several readers, however, can access the object at the same time.

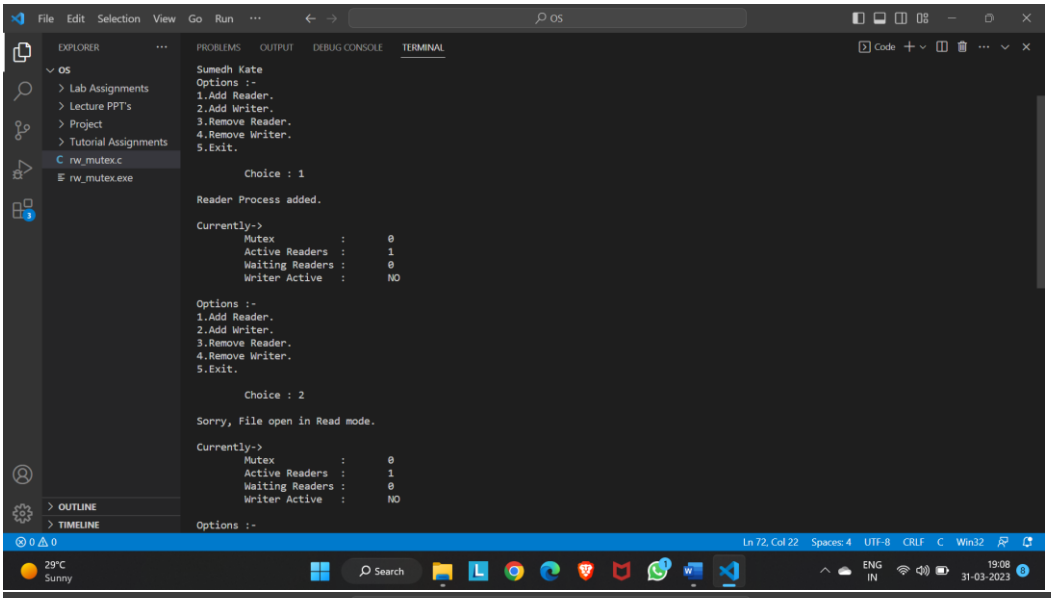
a. Using Mutex

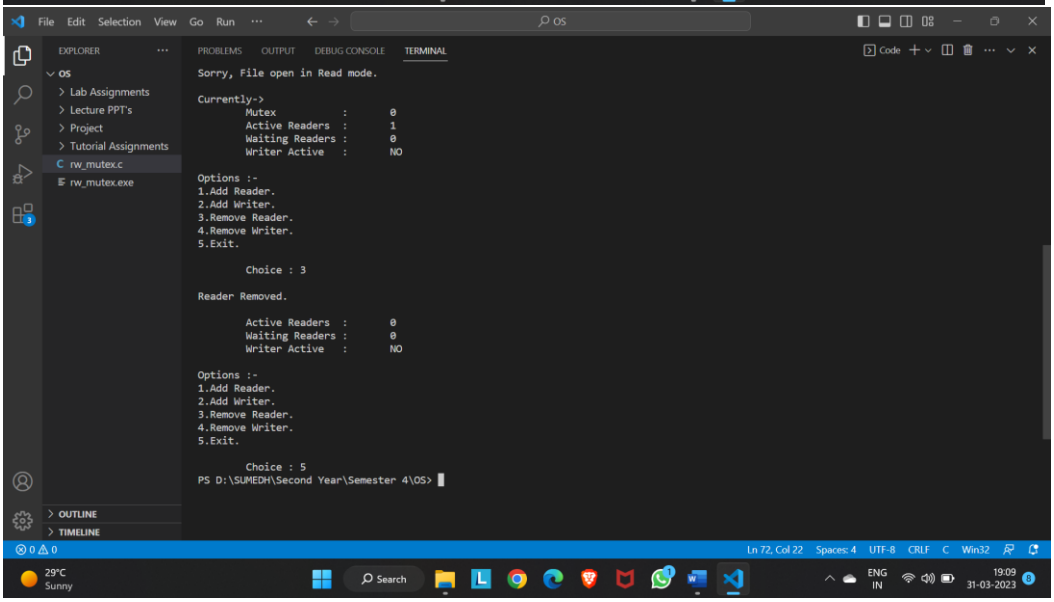
```
#include<stdio.h>
#include<stdbool.h>
struct semaphore
{
 int mutex;
 int rcount;
 int rwait;
 bool wrt;
};
void addR(struct semaphore *s)
{
 if (s->mutex == 0 && s->rcount == 0)
 {
 printf("\nSorry, File open in Write mode.\nNew Reader added to queue.\n");
 s->rwait++;
```

```
 }
 else
 {
 printf("\nReader Process added.\n");
 s->rcount++;
 s->mutex--;
 }
 return ;
}
void addW(struct semaphore *s)
{
 if(s->mutex==1)
 {
 s->mutex--;
 s->wrt=1;
 printf("\nWriter Process added.\n");
 }
 else if(s->wrt)
 printf("\nSorry, Writer already operational.\n");
 else
 printf("\nSorry, File open in Read mode.\n");
 return ;
}
void remR(struct semaphore *s)
{
 if(s->rcount == 0)
 printf("\nNo readers to remove.\n");
 else
 {
 printf("\nReader Removed.\n");
 s->rcount--;
 s->mutex++;
 }
 return ;
}
void remW(struct semaphore *s)
{
 if(s->wrt==0)
 printf("\nNo Writer to Remove");
 else
 {
 printf("\nWriter Removed\n");
 s->mutex++;
 s->wrt=0;
```

```
 if(s->rwait!=0)
 {
 s->mutex-=s->rwait;
 s->rcount=s->rwait;
 s->rwait=0;
 printf("%d waiting Readers Added.",s->rcount);
 }
 }
}
int main()
{
 struct semaphore S1={1,0,0};
printf("Sumedh Kate\n");
 while(1)
 {
 printf("Options :-\n1.Add Reader.\n2.Add Writer.\n3.Remove Reader.\n4.Remove
Writer.\n5.Exit.\n\n\tChoice : ");
 int ch;
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:
 addR(&S1); break;
 case 2:
 addW(&S1); break;
 case 3:
 remR(&S1); break;
 case 4:
 remW(&S1); break;
 case 5:
 getch();
 return 0;
 default:
 printf("\nInvalid Entry!");
 continue;
 }
 printf("\nCurrently->\n\tMutex\t\t:\t%d\n\tActive Readers\t:\t%d\n\tWaiting
Readers\t:\t%d\n\tWriter Active\t:\t%s\n\n",S1.mutex, S1.rcount, S1.rwait,
(S1.mutex==0 && S1.rcount==0)? "YES" : "NO");
 }
}
```

Output:





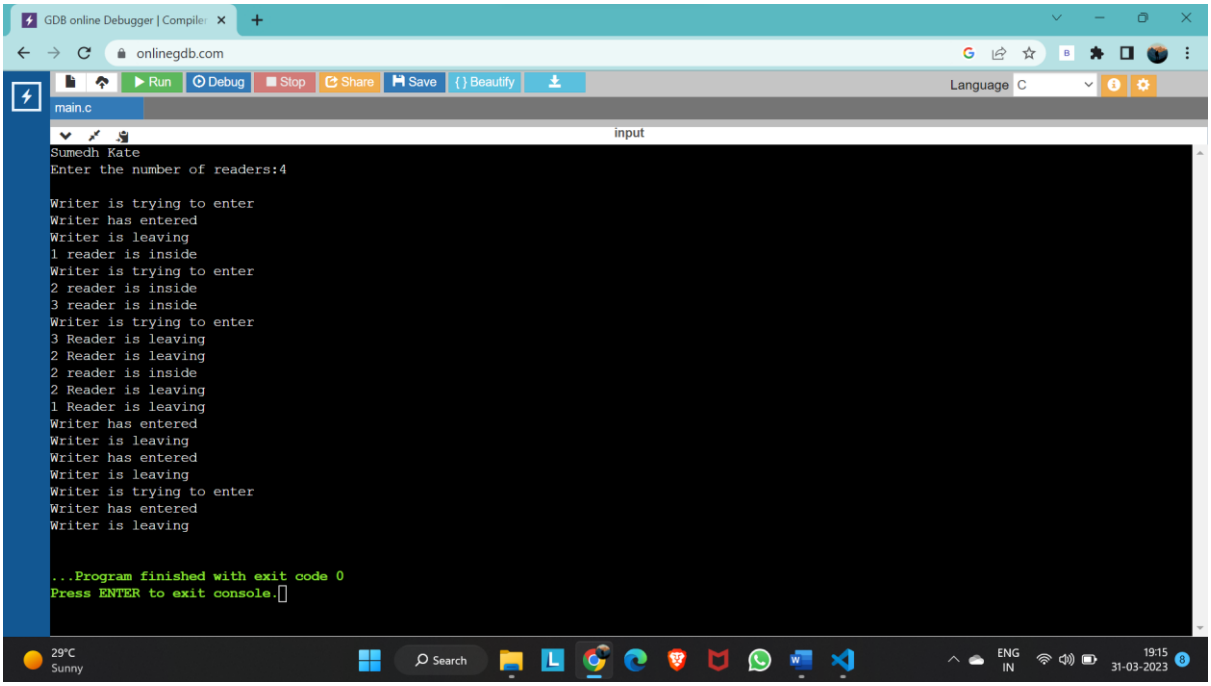
b. Using Semaphore

```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount = 0;

void *reader(void* param)
{
```

```
    sem_wait(&x);
    readercount++;
    if(readercount==1)
        sem_wait(&y);
    sem_post(&x);
    printf("%d reader is inside\n",readercount);
    usleep(3);
    sem_wait(&x);
    readercount--;
    if(readercount==0)
    {
        sem_post(&y);
    }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}

void *writer(void* param)
{
    printf("Writer is trying to enter\n");
    sem_wait(&y);
    printf("Writer has entered\n");
    sem_post(&y);
    printf("Writer is leaving\n");
    return NULL;
}

int main()
{
    int n2,i;
printf(“Sumedh Kate\n”);
    printf("Enter the number of readers:");
    scanf("%d",&n2);
    printf("\n");
    int n1[n2];
    sem_init(&x,0,1);
    sem_init(&y,0,1);
    for(i=0;i<n2;i++)
    {
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);
    }
    for(i=0;i<n2;i++)
```

```
    {
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }

}
```

Output:



2. Producer-Consumer

We have a fixed-size buffer. A producer can create an item and deposit it in the buffer. A consumer can select and consume stuff. We must ensure that when a producer places an item in the buffer, the consumer does not consume any item at the same moment.

a. Using Mutex

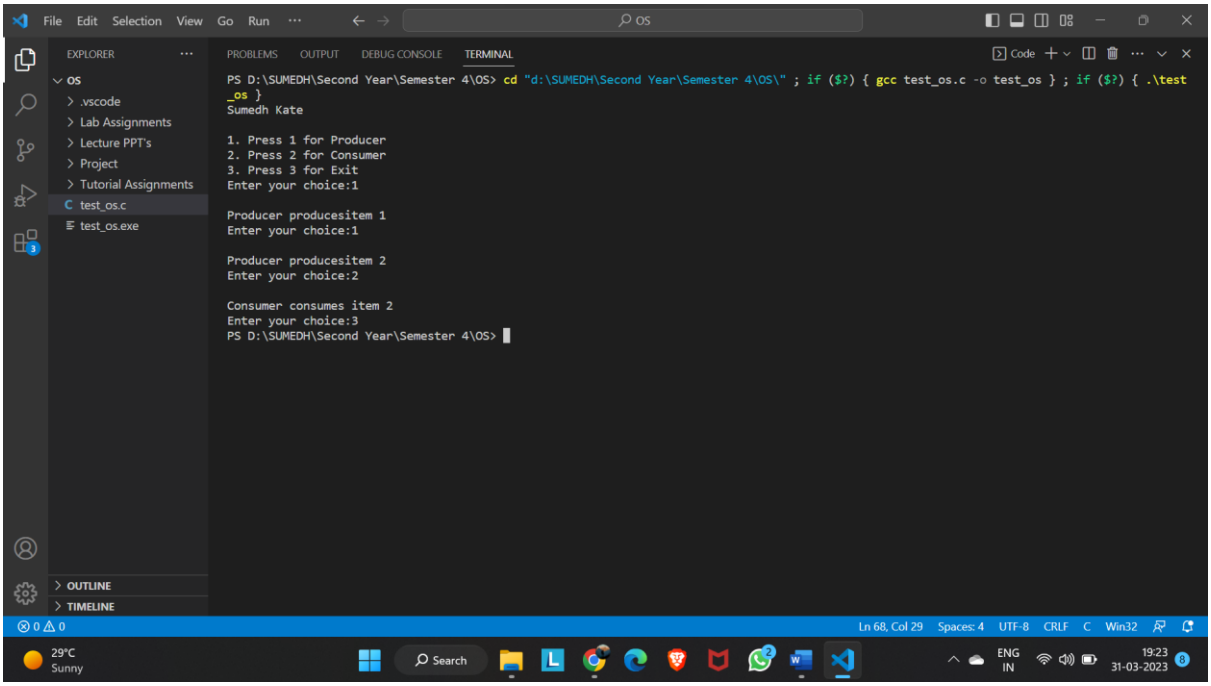
```
// C program for the above approach

#include <stdio.h>
#include <stdlib.h>

// Initialize a mutex to 1
int mutex = 1;
```

```
// Number of full slots as 0
int full = 0;

// Number of empty slots as size
// of buffer
int empty = 10, x = 0;

// Function to produce an item and
// add it to the buffer
void producer()
{
        // Decrease mutex value by 1
        --mutex;

        // Increase the number of full
        // slots by 1
        ++full;

        // Decrease the number of empty
        // slots by 1
        --empty;

        // Item produced
        x++;
        printf("\nProducer produces"
          "item %d",
          x);

        // Increase mutex value by 1
        ++mutex;
}

// Function to consume an item and
// remove it from buffer
void consumer()
{
        // Decrease mutex value by 1
        --mutex;

        // Decrease the number of full
        // slots by 1
        --full;

        // Increase the number of empty
```

```
                // slots by 1
                ++empty;
                printf("\nConsumer consumes "
                  "item %d",
                  x);
                x--;

                // Increase mutex value by 1
                ++mutex;
}

// Driver Code
int main()
{
                int n, i;
                printf(“Sumedh Kate\n”);
                printf("\n1. Press 1 for Producer"
                  "\n2. Press 2 for Consumer"
                  "\n3. Press 3 for Exit");

// Using '#pragma omp parallel for'
// can give wrong value due to
// synchronization issues.

// 'critical' specifies that code is
// executed by only one thread at a
// time i.e., only one thread enters
// the critical section at a given time
#pragma omp critical

                for (i = 1; i > 0; i++) {

                  printf("\nEnter your choice:");
                  scanf("%d", &n);

                  // Switch Cases
                  switch (n) {
                  case 1:

                                // If mutex is 1 and empty
                                // is non-zero, then it is
                                // possible to produce
                                if ((mutex == 1)
                                          && (empty != 0)) {
```

```
                producer();
            }

            // Otherwise, print buffer
            // is full
            else {
                printf("Buffer is full!");
            }
            break;

        case 2:

            // If mutex is 1 and full
            // is non-zero, then it is
            // possible to consume
            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }

            // Otherwise, print Buffer
            // is empty
            else {
                printf("Buffer is empty!");
            }
            break;

        // Exit Condition
        case 3:
            exit(0);
            break;
        }
    }
}
```

Output:



b. Using Semaphore

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

/*
use the pthread flag with gcc to compile this code
~$ gcc -pthread producer_consumer.c -o producer_consumer
*/


pthread_t *producers;
pthread_t *consumers;
```

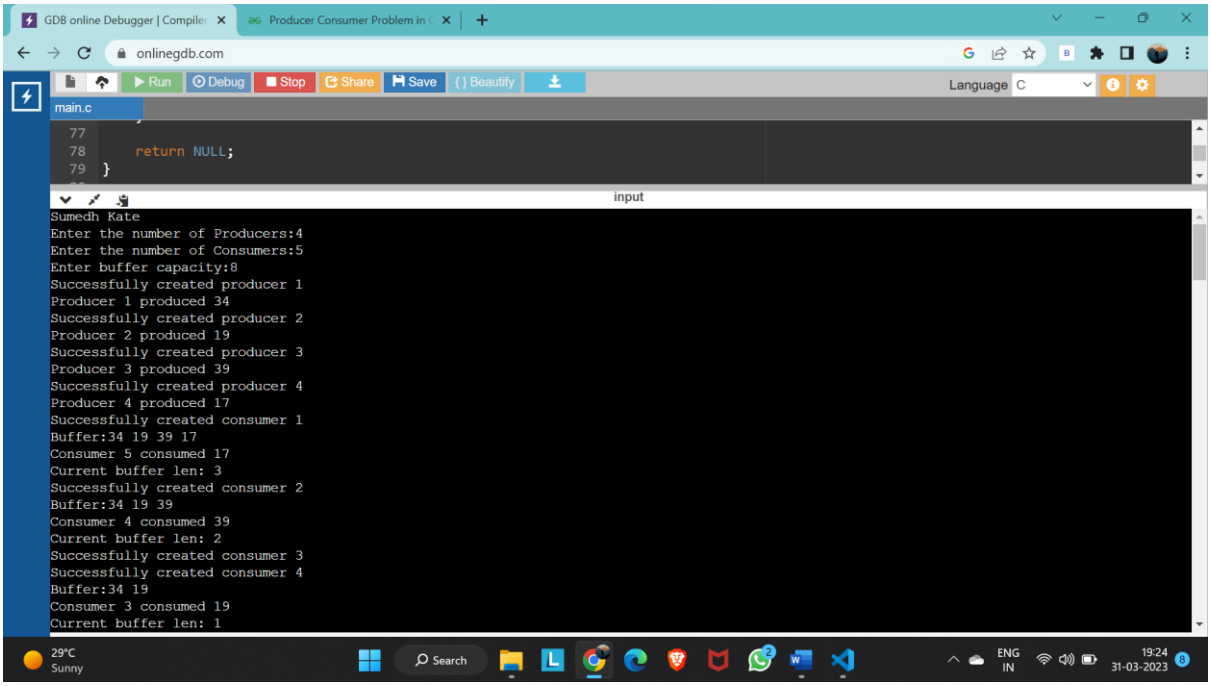
```
sem_t buf_mutex,empty_count,fill_count;

int *buf,buf_pos=-1,prod_count,con_count,buf_len;

int produce(pthread_t self){
        int i = 0;
        int p = 1 + rand()%40;
        while(!pthread_equal(*(producers+i),self) && i < prod_count){
                i++;
        }
        printf("Producer %d produced %d \n",i+1,p);
        return p;
}

void consume(int p,pthread_t self){
        int i = 0;
        while(!pthread_equal(*(consumers+i),self) && i < con_count){
                i++;
        }

        printf("Buffer:");
        for(i=0;i<=buf_pos;++i)
                printf("%d ",*(buf+i));
        printf("\nConsumer %d consumed %d \nCurrent buffer len:
%d\n",i+1,p,buf_pos);

}
```

```
void* producer(void *args){

        while(1){
                int p = produce(pthread_self());
                sem_wait(&empty_count);
                sem_wait(&buf_mutex);
                ++buf_pos;                      // critical section
                *(buf + buf_pos) = p;
                sem_post(&buf_mutex);
                sem_post(&fill_count);
                sleep(1 + rand()%3);
        }

        return NULL;
}


void* consumer(void *args){
        int c;
        while(1){
                sem_wait(&fill_count);
                sem_wait(&buf_mutex);
                c = *(buf+buf_pos);
                consume(c,pthread_self());
                --buf_pos;
                sem_post(&buf_mutex);
                sem_post(&empty_count);
```

```
            sleep(1+rand()%5);
        }

        return NULL;
}

int main(void){

        int i,err;
        printf(“Sumedh Kate\n”);
        srand(time(NULL));

        sem_init(&buf_mutex,0,1);
        sem_init(&fill_count,0,0);

        printf("Enter the number of Producers:");
        scanf("%d",&prod_count);
        producers = (pthread_t*) malloc(prod_count*sizeof(pthread_t));

        printf("Enter the number of Consumers:");
        scanf("%d",&con_count);
        consumers = (pthread_t*) malloc(con_count*sizeof(pthread_t));

        printf("Enter buffer capacity:");
        scanf("%d",&buf_len);
        buf = (int*) malloc(buf_len*sizeof(int));

        sem_init(&empty_count,0,buf_len);
```

```
        for(i=0;i<prod_count;i++){
                err = pthread_create(producers+i,NULL,&producer,NULL);
                if(err != 0){
                        printf("Error creating producer %d: %s\n",i+1,strerror(err));
                }else{
                        printf("Successfully created producer %d\n",i+1);
                }
        }

        for(i=0;i<con_count;i++){
                err = pthread_create(consumers+i,NULL,&consumer,NULL);
                if(err != 0){
                        printf("Error creating consumer %d: %s\n",i+1,strerror(err));
                }else{
                        printf("Successfully created consumer %d\n",i+1);
                }
        }

        for(i=0;i<prod_count;i++){
                pthread_join(*(producers+i),NULL);
        }
        for(i=0;i<con_count;i++){
                pthread_join(*(consumers+i),NULL);
        }


        return 0;
}
```

Output:



3. Dinning Philosopher

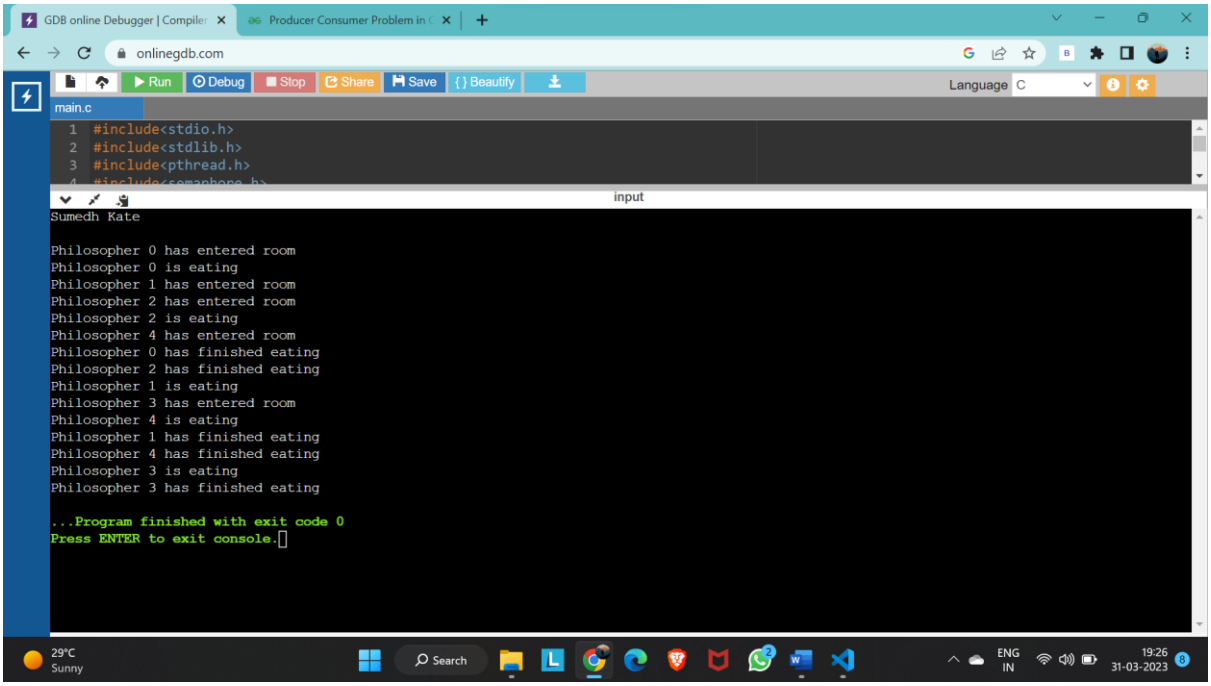
According to the dining philosophers conundrum, five philosophers share a circular table and eat and think alternately. Each philosopher receives a dish of rice and five chopsticks. To eat, a philosopher requires both their right and left chopstick. A hungry philosopher can only eat if both chopsticks are available. Otherwise, a philosopher will put down their chopstick and resume their thought.

a. Using Semaphore

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
sem_t room;
sem_t chopstick[5];
void * philosopher(void *);
void eat(int);
void eat(int phil)
{
printf("\nPhilosopher %d is eating",phil);
```

```
}
int main()
{
   printf("Sumedh Kate\n");
int i,a[5];
pthread_t tid[5];
sem_init(&room,0,4);
for(i=0;i<5;i++)
sem_init(&chopstick[i],0,1);
for(i=0;i<5;i++){
a[i]=i;
pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
}
for(i=0;i<5;i++)
pthread_join(tid[i],NULL);
}
void * philosopher(void * num)
{
int phil=*(int *)num;
sem_wait(&room);
printf("\nPhilosopher %d has entered room",phil);
sem_wait(&chopstick[phil]);
sem_wait(&chopstick[(phil+1)%5]);
eat(phil);
sleep(2);
printf("\nPhilosopher %d has finished eating",phil);
sem_post(&chopstick[(phil+1)%5]);
sem_post(&chopstick[phil]);
sem_post(&room);
}
```

Output:



Conclusion:

The above-mentioned solution for reader-writer, producer consumer, dinning philosopher problem. For an operating system to work efficiently and give maximum possible throughput we need to tackle these problems. The above mentioned C programs are the solution using which all three problems can be solved and semaphore and mutex are implemented for the same.

Name: Sumedh Kate

SY_CS-B

Roll No: 52

OS_Lab4: Scheduling Algorithms

Objectives: This lab assignment aims to write a c function for various scheduling algorithms, algorithms like FCFS, SJF, Priority and round Robin are used for scheduling the processes in operating system.

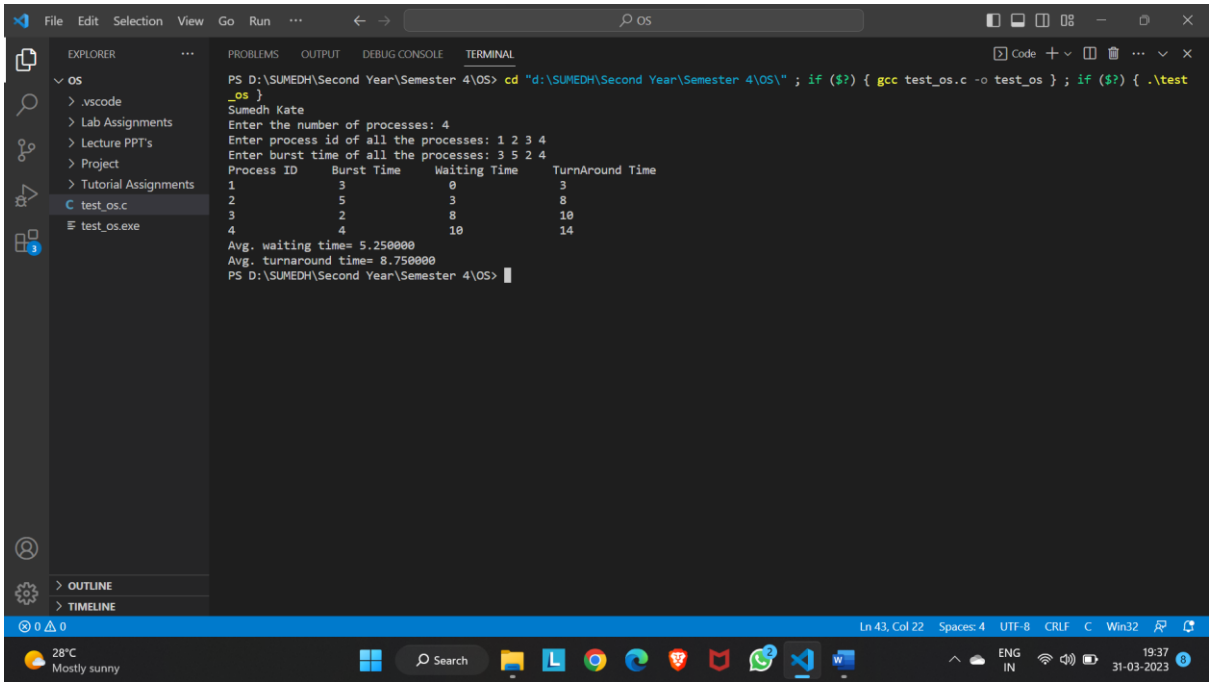
1. **First Come First Serve (FCFS)**

The simplest CPU scheduling method, which arranges processes based on their arrival times. According to the first come, first served scheduling strategy, the process that demands the CPU first is assigned the CPU first. The FIFO queue is used to implement it. When a process enters the ready queue, its PCB is connected to the queue's tail. When the CPU becomes available, it is assigned to the process at the front of the queue. The currently active process is then removed from the queue. FCFS is a scheduling algorithm that uses non-preemptive scheduling.

```
#include <stdio.h>

int main()

{

   printf("Sumedh Kate\n");

   int pid[15];

   int bt[15];

   int n;

   printf("Enter the number of processes: ");

   scanf("%d",&n);


   printf("Enter process id of all the processes: ");

   for(int i=0;i<n;i++)

   {

      scanf("%d",&pid[i]);

   }


   printf("Enter burst time of all the processes: ");
```

```
for(int i=0;i<n;i++)
{
   scanf("%d",&bt[i]);
}

int i, wt[n];
wt[0]=0;

//for calculating waiting time of each process
for(i=1; i<n; i++)
{
   wt[i]= bt[i-1]+ wt[i-1];
}

printf("Process ID     Burst Time     Waiting Time     TurnAround Time\n");
float twt=0.0;
float tat= 0.0;
for(i=0; i<n; i++)
{
   printf("%d\t\t", pid[i]);
   printf("%d\t\t", bt[i]);
   printf("%d\t\t", wt[i]);

   //calculating and printing turnaround time of each process
   printf("%d\t\t", bt[i]+wt[i]);
   printf("\n");

   //for calculating total waiting time
   twt += wt[i];
```

```
    //for calculating total turnaround time
    tat += (wt[i]+bt[i]);
  }
  float att,awt;

  //for calculating average waiting time
  awt = twt/n;

  //for calculating average turnaround time
  att = tat/n;
  printf("Avg. waiting time= %f\n",awt);
  printf("Avg. turnaround time= %f",att);
}
```

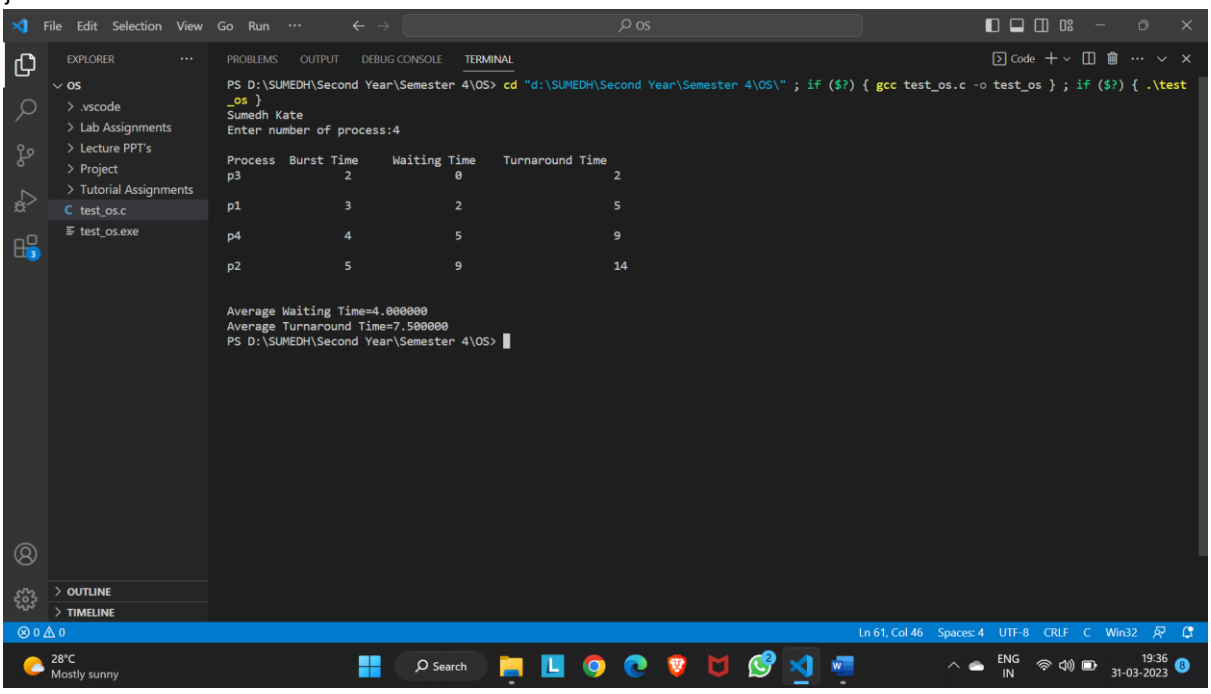


2. Shortest Job first (SJF)

SJF (or shortest job next) is a scheduling policy that chooses the waiting process with the shortest execution time to execute next. SJN, or Shortest Job Next (SJN), can be either preemptive or non-preemptive.

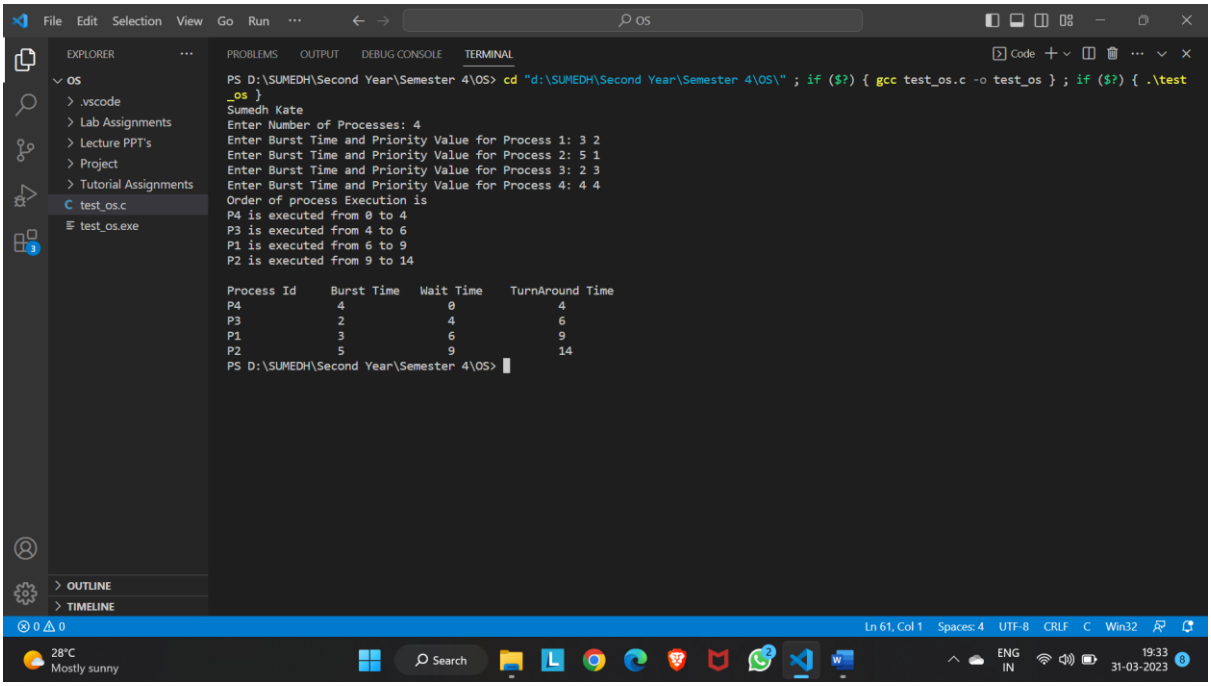
```
#include<stdio.h>
int main()
{ printf("Sumedh Kate\n");
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }

   //sorting of burst times
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
            pos=j;
      }

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;

   //finding the waiting time of all the processes
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
          //individual WT by adding BT of all previous completed processes
         wt[i]+=bt[j];
```

```
    //total waiting time
    total+=wt[i];
  }

  //average waiting time
  avg_wt=(float)total/n;

  printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
  for(i=0;i<n;i++)
  {
    //turnaround time of individual processes
    tat[i]=bt[i]+wt[i];

    //total turnaround time
    totalT+=tat[i];
    printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
  }

  //average turnaround time
  avg_tat=(float)totalT/n;
  printf("\n\nAverage Waiting Time=%f",avg_wt);
  printf("\nAverage Turnaround Time=%f",avg_tat);
}
```



3. Priority

Priority scheduling is a non-preemptive technique that is one of the most commonly used in batch systems. Each process is assigned a first arrival time (the process with the lowest arrival time is allotted first); if two processes have the same arrival time, the priority are compared (highest process first). In addition, if two processes have the same priority, compare the process numbers (less process number first). This method is repeated until all processes are completed.



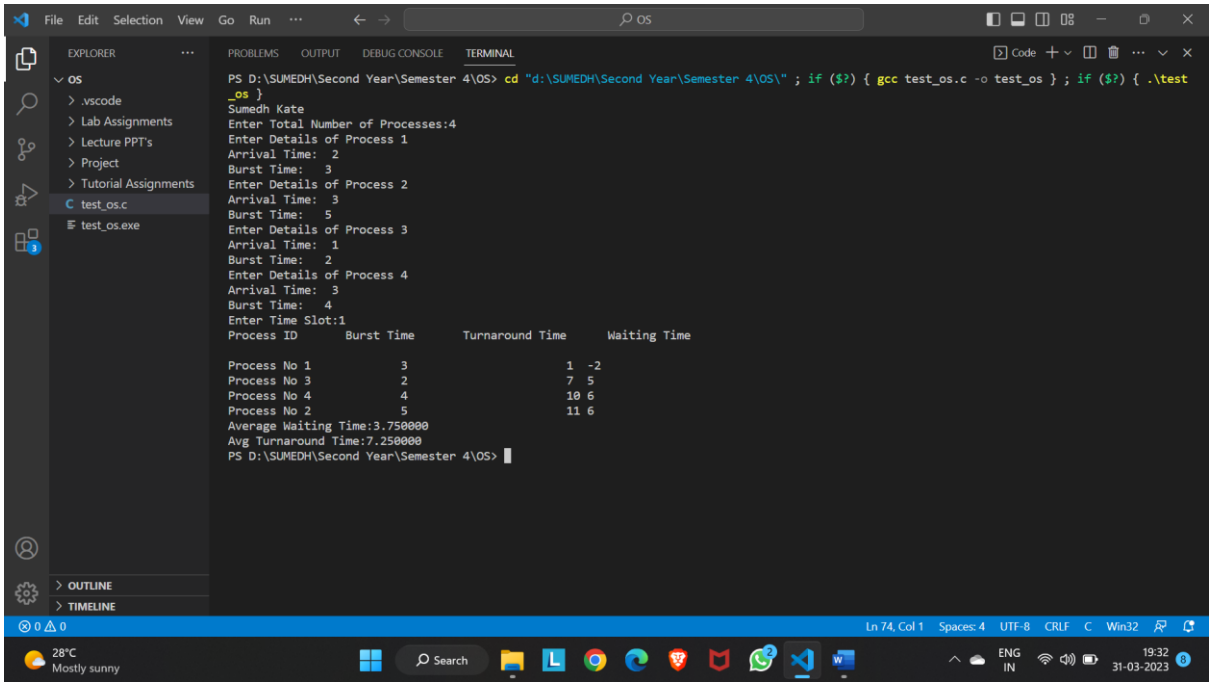
4. Round-Robin

First, the processes that are eligible to enter the ready queue do so. After joining the Ready Queue, the first process is executed for a Time Quantum slice of time. When the process is finished, it gets removed from the ready queue. Even now, the process takes some time to finish its execution before being placed to the Ready Queue. The Ready Queue does not contain processes that are already in the Ready Queue. The Ready Queue is built such that it does not contain non-unique processes. Maintaining the same processes promotes process redundancy. When the process execution is over, the Ready Queue does not hold the completed process.

```
#include<stdio.h>
int main()
{
  //Input no of processed
  int  n;
  printf("Sumedh Kate\n");
  printf("Enter Total Number of Processes:");
  scanf("%d", &n);
  int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
  int x = n;
```

```
//Input details of processes
for(int i = 0; i < n; i++)
{
   printf("Enter Details of Process %d \n", i + 1);
   printf("Arrival Time:  ");
   scanf("%d", &arr_time[i]);
   printf("Burst Time:   ");
   scanf("%d", &burst_time[i]);
   temp_burst_time[i] = burst_time[i];
}

//Input time slot
int time_slot;
printf("Enter Time Slot:");
scanf("%d", &time_slot);

//Total indicates total time
//counter indicates which process is executed
int total = 0,  counter = 0,i;
printf("Process ID      Burst Time      Turnaround Time      Waiting Time\n");
for(total=0, i = 0; x!=0; )
{
   // define the conditions
   if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
   {
      total = total + temp_burst_time[i];
      temp_burst_time[i] = 0;
      counter=1;
   }
   else if(temp_burst_time[i] > 0)
   {
      temp_burst_time[i] = temp_burst_time[i] - time_slot;
      total  += time_slot;
   }
   if(temp_burst_time[i]==0 && counter==1)
   {
      x--; //decrement the process no.
      printf("\nProcess No %d  \t\t %d\t\t\t %d\t\t\t %d\t\t", i+1, burst_time[i],
            total-arr_time[i], total-arr_time[i]-burst_time[i]);
      wait_time = wait_time+total-arr_time[i]-burst_time[i];
      ta_time += total -arr_time[i];
      counter =0;
   }
   if(i==n-1)
   {
      i=0;
   }
```

```
        else if(arr_time[i+1]<=total)
        {
           i++;
        }
        else
        {
           i=0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAvg Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```



Conclusion:

Here the observation is for same burst time for all processes the average waiting time and average turnaround time given by round robin algorithm is comparatively less than any other algorithm.

	Average Waiting Time	Average Turnaround Time
FCFS	5.25	8.75
SJF	4.0	7.5
Priority	4.7	8.2
Round robin	3.75	7.25

A CPU scheduling algorithm determines which processes will use the CPU for execution and which processes will be held or removed from execution. The primary goal of CPU scheduling algorithms in

operating systems is to ensure that the CPU is never idle, which means that the OS has at least one process ready for execution among the available processes in the ready queue.

Name: Sumedh Kate

SY_CS-B

Roll No: 52

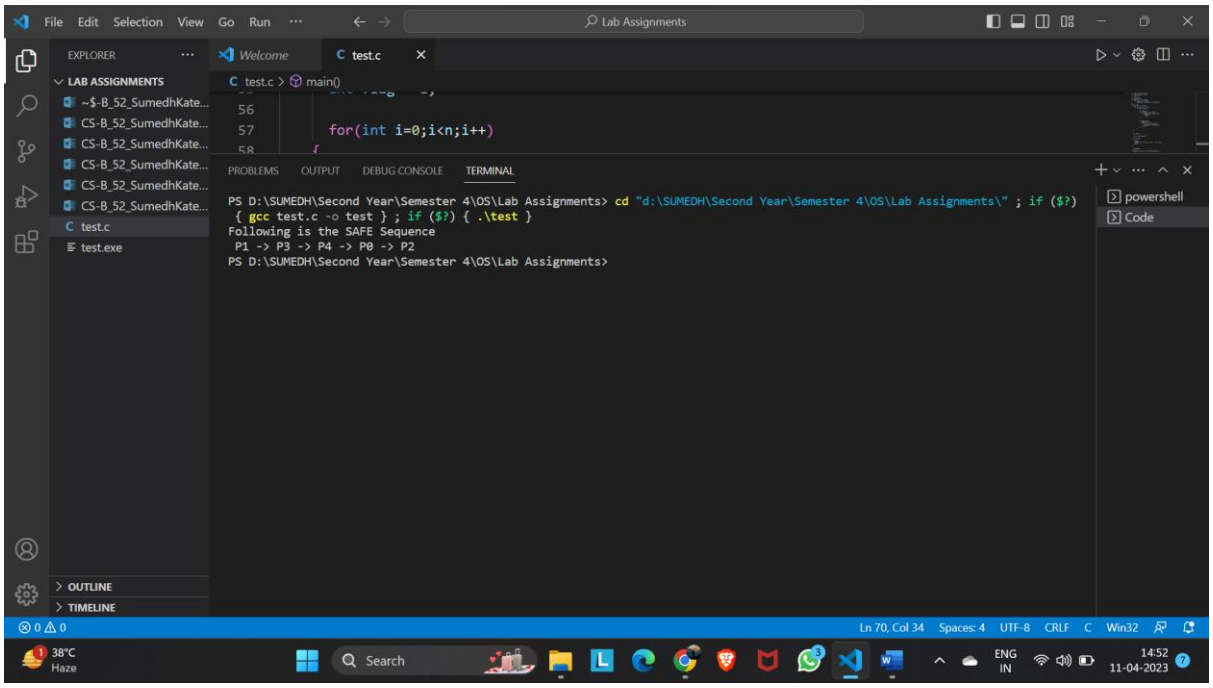
OS_Lab5: Banker's Algorithm for Deadlock Avoidance.

```c
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4

    int max[5][3] = { { 7, 5, 3 }, // P0    // MAX Matrix
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    int avail[3] = { 3, 3, 2 }; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
```

```
            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                     break;
                }
            }

            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}

  int flag = 1;

  for(int i=0;i<n;i++)
{
  if(f[i]==0)
  {
    flag=0;
     printf("The following system is not safe");
    break;
  }
}

  if(flag==1)
{
  printf("Following is the SAFE Sequence\n");
  for (i = 0; i < n - 1; i++)
    printf(" P%d ->", ans[i]);
  printf(" P%d", ans[n - 1]);
}


return (0);
```

```
    // This code is contributed by Deep Baldha (CandyZack)
}
```

Output:



Name: Sumedh Kate

SY_CS-B

Roll No: 52

OS_Lab6: Page Replacement Algorithm.

1. **First-In-First-Out (FIFO):**
   FIFO (First-In-First-Out) is a page replacement algorithm used by the operating system to manage the virtual memory of a computer system. The basic idea behind this algorithm is to keep track of the pages that were brought into memory first and remove them first when required. The FIFO algorithm maintains a queue of pages in the order they were brought into the main memory. Whenever a page fault occurs, the operating system removes the page that came first into the memory, i.e., the one at the front of the queue. It then replaces this page with the new page that needs to be loaded into the memory.

Algorithm for FIFO Page Replacement:

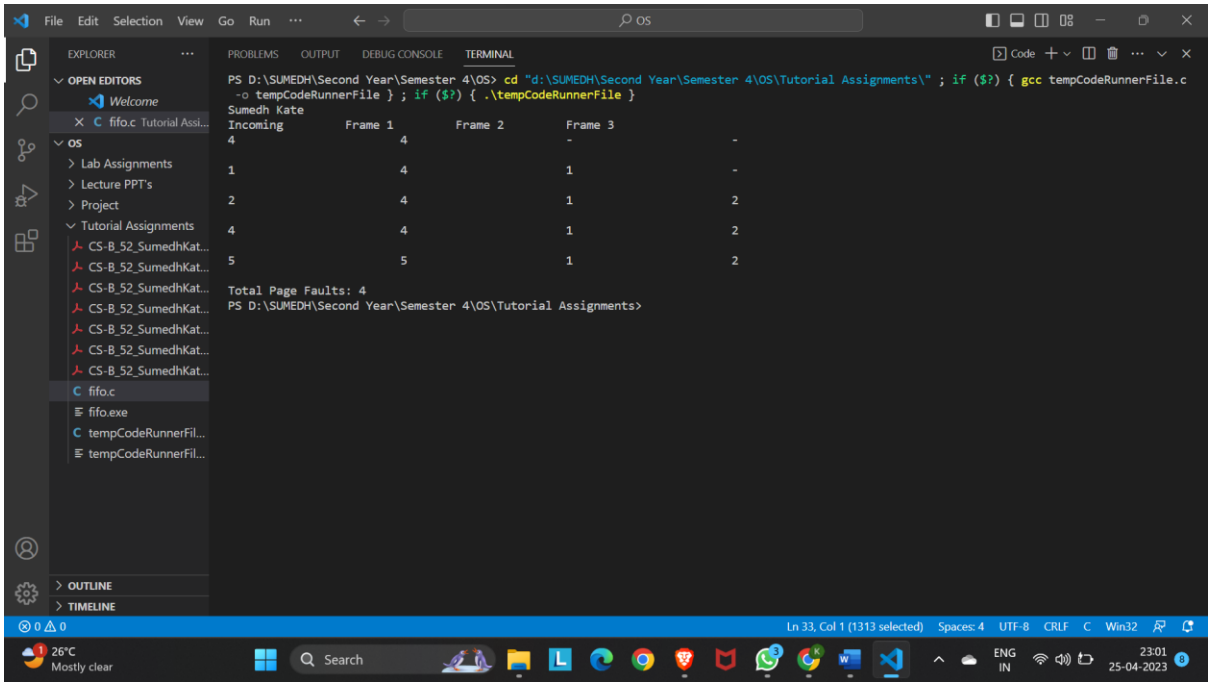
1. Start to traverse the pages.
2. If the memory holds fewer pages, then the capacity else goes to step 5.
3. Push pages in the queue one at a time until the queue reaches its maximum capacity or all page requests are fulfilled.
4. If the current page is present in the memory, do nothing.
5. Else, pop the topmost page from the queue as it was inserted first.
6. Replace the topmost page with the current page from the string.
7. Increment the page faults.
8. Stop

Code:

```
#include<stdio.h>
int main()
{     printf("Sumedh Kate \n");
      int incomingStream[] = {4, 1, 2, 4, 5};
      int pageFaults = 0;
      int frames = 3;
      int m, n, s, pages;
      pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

    printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");     int temp[frames];
    for(m = 0; m < frames; m++)
    {         temp[m] = -1;
    }
    for(m = 0; m < pages; m++)
    {
```

```
      s = 0;
       for(n = 0; n < frames; n++)
      {
         if(incomingStream[m] == temp[n])
         {              s++;
            pageFaults--;
         }       }       pageFaults++;
             if((pageFaults <= frames) && (s == 0))
      {           temp[m] = incomingStream[m];
      }        else if(s == 0)
      {           temp[(pageFaults - 1) % frames] = incomingStream[m];
      }             printf("\n");       printf("%d\t\t\t",incomingStream[m]);       for(n = 0; n <
frames; n++)
      {           if(temp[n] != -1)              printf(" %d\t\t\t",
temp[n]);          else              printf(" - \t\t\t");
      }
      printf("\n");
   }     printf("\nTotal Page Faults: %d\n", pageFaults);     return 0;
}
```

Output:



The main advantage of the FIFO algorithm is its simplicity. It requires minimal bookkeeping, as it only needs to keep track of the order in which the pages were brought into the

memory. It is also easy to implement and can work efficiently on hardware with limited resources.

However, the main disadvantage of the FIFO algorithm is that it suffers from the "Belady's anomaly." This means that increasing the number of page frames allocated to a process may result in an increase in the number of page faults. This is because the algorithm may replace a page that is still required by the process, leading to unnecessary page faults.

2. Optimal:

The optimal page replacement algorithm is an algorithm used in operating systems to manage the allocation of memory to different processes. The goal of the algorithm is to minimize the number of page faults that occur, where a page fault occurs when a process attempts to access a page that is not currently in memory.

The optimal page replacement algorithm works by predicting which page will be needed furthest in the future and evicting that page from memory. This algorithm requires knowledge of the future memory access pattern of a process, which is not always possible in practice. However, it serves as a theoretical benchmark against which other page replacement algorithms can be compared.

Algorithm for Optimal Page Replacement:

1. Maintain a list of pages that are currently in memory.
2. For each memory access, determine if the page is already in memory.
3. If the page is already in memory, no action is required.
4. If the page is not in memory, determine which page in memory will not be used for the longest period of time in the future.
5. Replace the page that will not be used for the longest period of time in the future with the new page.
6. Continue this process for each memory access until the process completes.

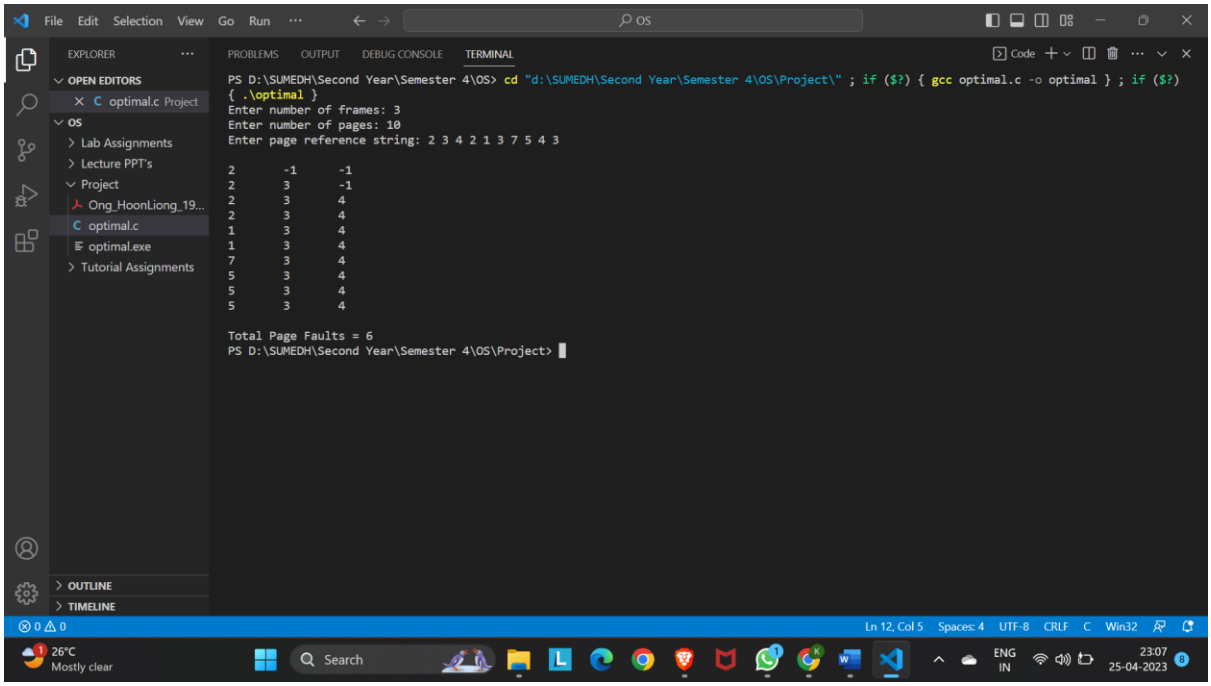
Code:

```
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k,
pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page reference string: ");
```

```
for(i = 0; i < no_of_pages; ++i){
   scanf("%d", &pages[i]);
}

for(i = 0; i < no_of_frames; ++i){
   frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i){
   flag1 = flag2 = 0;

   for(j = 0; j < no_of_frames; ++j){
      if(frames[j] == pages[i]){
           flag1 = flag2 = 1;
           break;
        }
   }

   if(flag1 == 0){
      for(j = 0; j < no_of_frames; ++j){
         if(frames[j] == -1){
            faults++;
            frames[j] = pages[i];
            flag2 = 1;
            break;
         }
      }
   }

   if(flag2 == 0){
    flag3 =0;

      for(j = 0; j < no_of_frames; ++j){
       temp[j] = -1;

       for(k = i + 1; k < no_of_pages; ++k){
       if(frames[j] == pages[k]){
       temp[j] = k;
       break;
       }
       }
      }
```

```
          for(j = 0; j < no_of_frames; ++j){
           if(temp[j] == -1){
           pos = j;
           flag3 = 1;
           break;
           }
          }

          if(flag3 ==0){
           max = temp[0];
           pos = 0;

           for(j = 1; j < no_of_frames; ++j){
           if(temp[j] > max){
           max = temp[j];
           pos = j;
           }
           }
          }
frames[pos] = pages[i];
faults++;
      }

      printf("\n");

      for(j = 0; j < no_of_frames; ++j){
         printf("%d\t", frames[j]);
      }
   }

   printf("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```

Output:



3. LRU:

LRU (Least Recently Used) is a page replacement algorithm used by the operating system to manage the virtual memory of a computer system. The basic idea behind this algorithm is to keep track of the pages that have not been used recently and remove them first when required. The LRU algorithm maintains a queue of pages in the order of their usage. Whenever a page fault occurs, the operating system checks if there is a free page frame available in the main memory. If there is, the new page is loaded into the free page frame. If there is no free page frame available, the LRU algorithm selects the page that has not been used for the longest time and replaces it with the new page.

Algorithm for LRU Page Replacement:

1. Maintain a list of all the pages currently in memory.
2. When a new page is requested, check if it is already in memory.
3. If the page is already in memory, mark it as accessed.
4. If the page is not in memory, find the page that has not been accessed for the longest period of time (i.e., the least recently used page) and replace it with the new page.
5. Repeat steps 2-4 for each new page request.

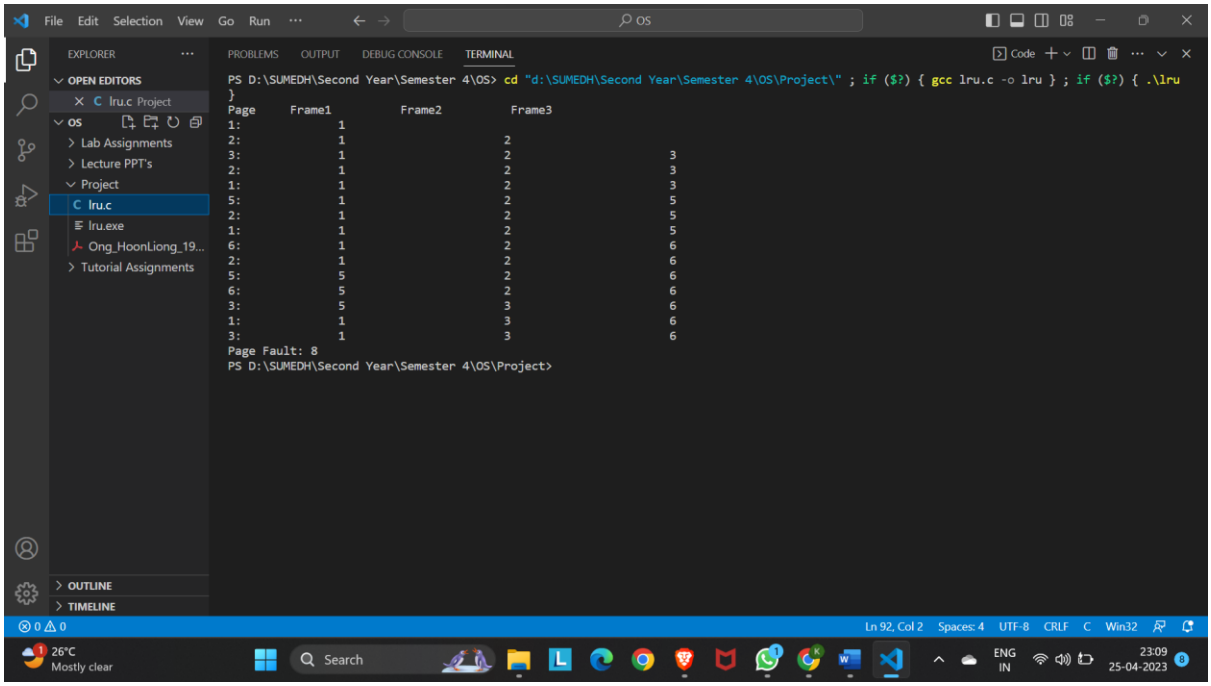
Code:

```
#include<stdio.h>
#include<limits.h>

int checkHit(int incomingPage, int queue[], int occupied){

  for(int i = 0; i < occupied; i++){
    if(incomingPage == queue[i])
      return 1;
```

```
    }

    return 0;
}

void printFrame(int queue[], int occupied)
{
    for(int i = 0; i < occupied; i++)
        printf("%d\t\t\t",queue[i]);
}

int main()
{

//    int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1};
//    int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};
    int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

    int n = sizeof(incomingStream)/sizeof(incomingStream[0]);
    int frames = 3;
    int queue[n];
    int distance[n];
    int occupied = 0;
    int pagefault = 0;

    printf("Page\t Frame1 \t Frame2 \t Frame3\n");

    for(int i = 0;i < n; i++)
    {
        printf("%d:  \t\t",incomingStream[i]);
        // what if currently in frame 7
        // next item that appears also 7
        // didnt write condition for HIT

        if(checkHit(incomingStream[i], queue, occupied)){
            printFrame(queue, occupied);
        }

        // filling when frame(s) is/are empty
        else if(occupied < frames){
            queue[occupied] = incomingStream[i];
            pagefault++;
```

```
            occupied++;

            printFrame(queue, occupied);
        }
        else{

            int max = INT_MIN;
            int index;
            // get LRU distance for each item in frame
            for (int j = 0; j < frames; j++)
            {
                distance[j] = 0;
                // traverse in reverse direction to find
                // at what distance  frame item occurred last
                for(int k = i - 1; k >= 0; k--)
                {
                    ++distance[j];

                    if(queue[j] == incomingStream[k])
                        break;
                }

                // find frame item with max distance for LRU
                // also notes the index of frame item in queue
                // which appears furthest(max distance)
                if(distance[j] > max){
                    max = distance[j];
                    index = j;
                }
            }
            queue[index] = incomingStream[i];
            printFrame(queue, occupied);
            pagefault++;
        }

        printf("\n");
    }

    printf("Page Fault: %d",pagefault);

    return 0;
}
```

Output:


File Edit Selection View Go Run
OS
EXPLORER
OPEN EDITORS
lru.c Project
OS
Lab Assignments
Lecture PPT's
Project
lru.c
lru.exe
Ong_HoonLiong_19...
Tutorial Assignments
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
PS D:\SUMEDH\Second Year\Semester 4\OS> cd "d:\SUMEDH\Second Year\Semester 4\OS\Project\" ; if ($?) { gcc lru.c -o lru } ; if ($?) { .\lru
}
Page Frame1 Frame2 Frame3
1: 1
2: 1 2
3: 1 2 3
2: 1 2 3
1: 1 2 3
5: 1 2 5
2: 1 2 5
1: 1 2 5
6: 1 2 6
2: 1 2 6
5: 5 2 6
6: 5 2 6
3: 5 3 6
1: 1 3 6
3: 1 3 6
Page Fault: 8
PS D:\SUMEDH\Second Year\Semester 4\OS\Project>
Ln 92, Col 2 Spaces: 4 UTF-8 CRLF C Win32
26°C
Mostly clear
Search
ENG
IN
23:09
25-04-2023