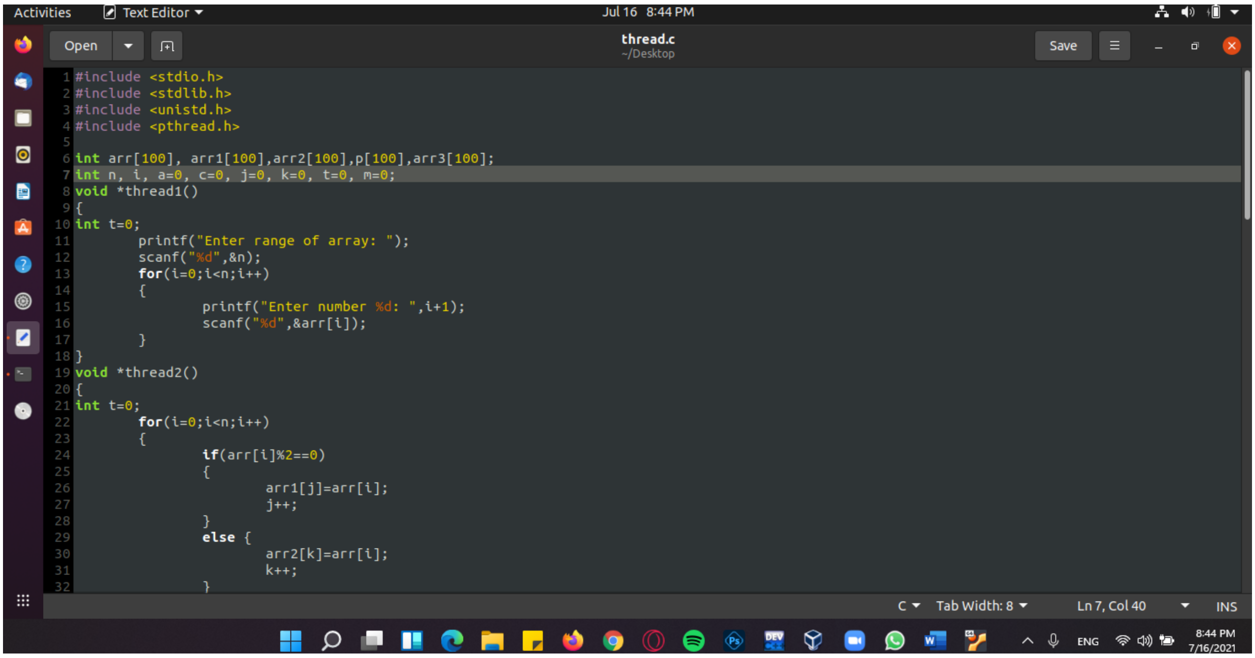
OPERATING SYSTEMS ASSIGNMENT – 02

Name: Kamisha Salim
S.ID: 11070

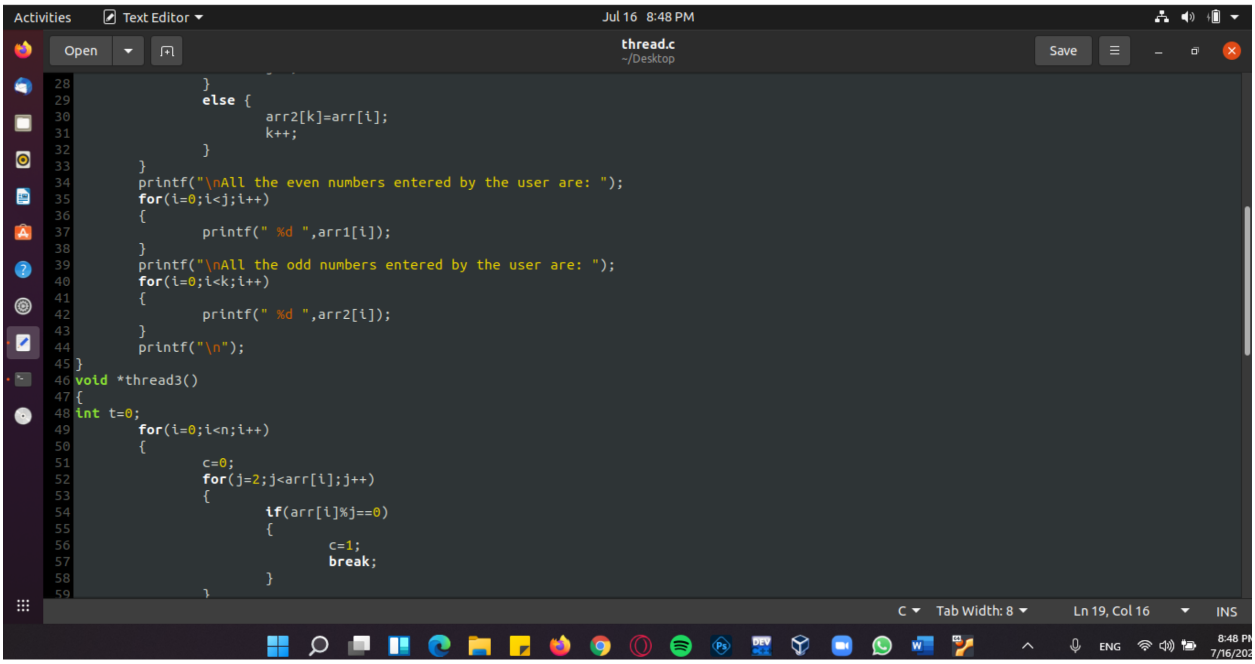
QUESTION – 1

Provide two programming examples in which multithreading provides better performance than a single-threaded solution.

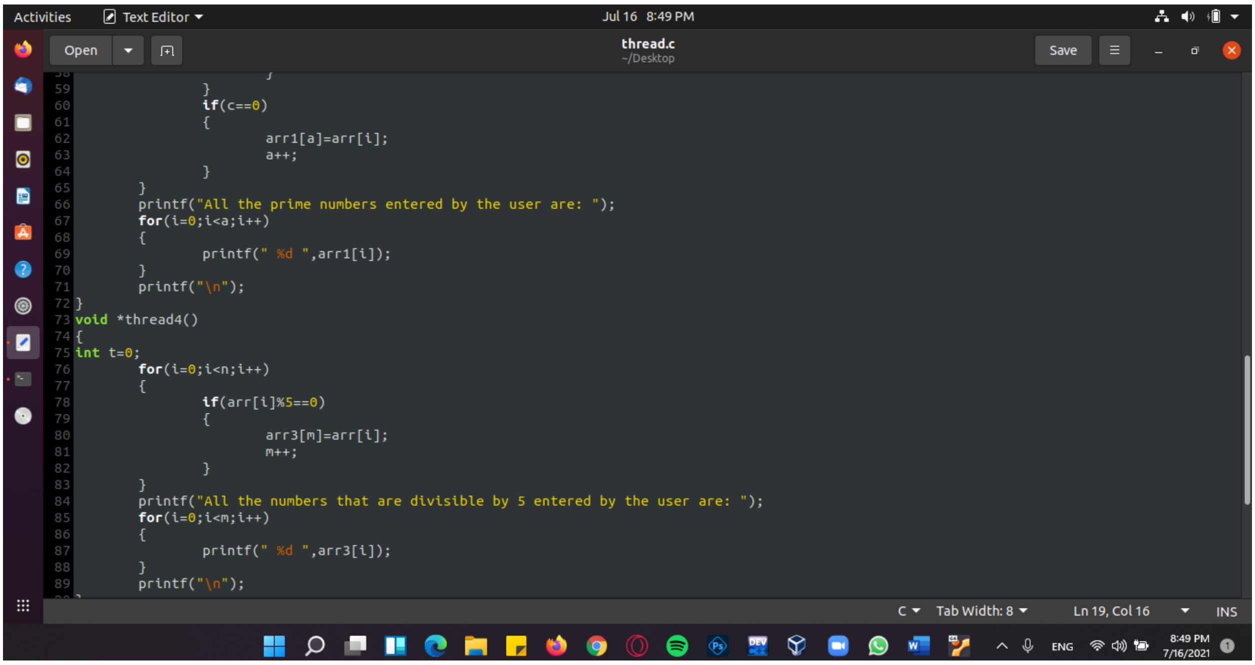
Example – 1: Here we have divided our program into 4 threads. 1st thread asking the user to input numbers in an array and the 2nd, 3rd and 4th threads are different programs that will be implemented on the user given array. If we were to block any of the last 3 threads except the 1st thread, i.e.: thread no. 2, 3 or 4, the program will automatically jump to the next available thread without any interruptions. For e.g.: if we blocked thread no. 2, the program will jump to thread no. 3 and will display the output without running thread no. 2. This is the advantage of multithreading because if we used single-threading, the program would have executed the entire process in a single sequence and we would not be able to block any processes according to our need in comparison to multithreading.



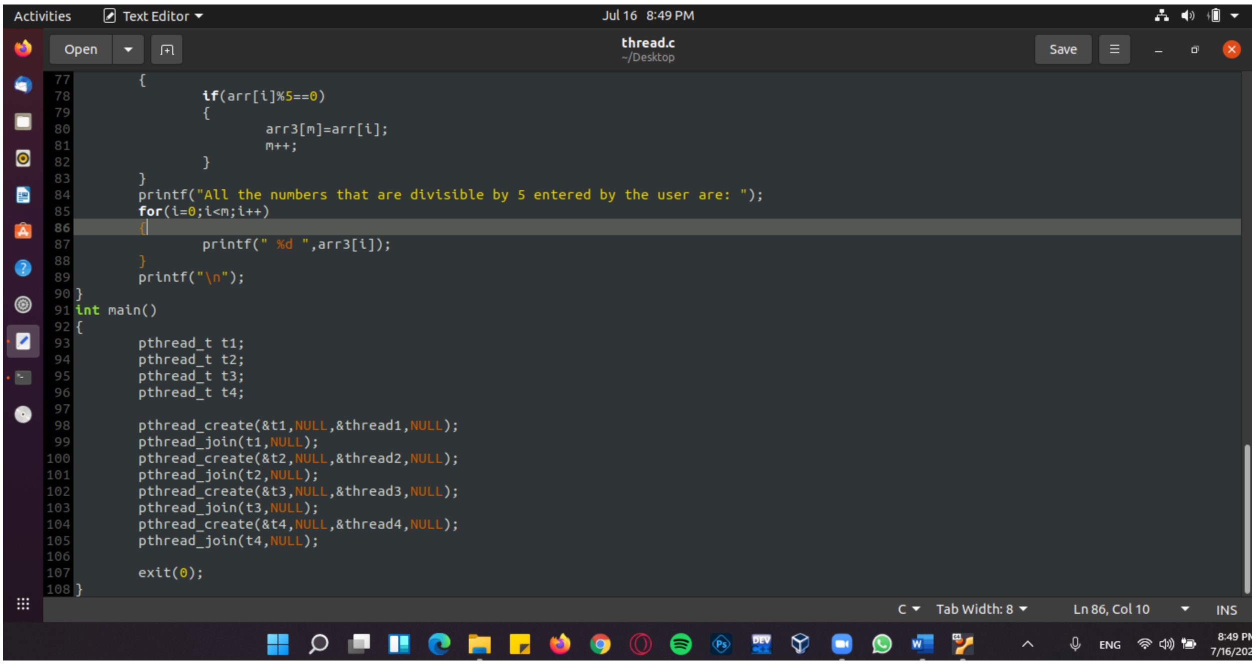
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

int arr[100], arr1[100],arr2[100],p[100],arr3[100];
int n, i, a=0, c=0, j=0, k=0, t=0, m=0;
void *thread1()
{
int t=0;
        printf("Enter range of array: ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter number %d: ",i+1);
                scanf("%d",&arr[i]);
        }
}
void *thread2()
{
int t=0;
        for(i=0;i<n;i++)
        {
                if(arr[i]%2==0)
                {
                        arr1[j]=arr[i];
                        j++;
                }
                else {
                        arr2[k]=arr[i];
                        k++;
                }
```



```
                }
                else {
                        arr2[k]=arr[i];
                        k++;
                }
        }
        printf("\nAll the even numbers entered by the user are: ");
        for(i=0;i<j;i++)
        {
                printf(" %d ",arr1[i]);
        }
        printf("\nAll the odd numbers entered by the user are: ");
        for(i=0;i<k;i++)
        {
                printf(" %d ",arr2[i]);
        }
        printf("\n");
}
void *thread3()
{
int t=0;
        for(i=0;i<n;i++)
        {
                c=0;
                for(j=2;j<arr[i];j++)
                {
                        if(arr[i]%j==0)
                        {
                                c=1;
                                break;
                        }
```

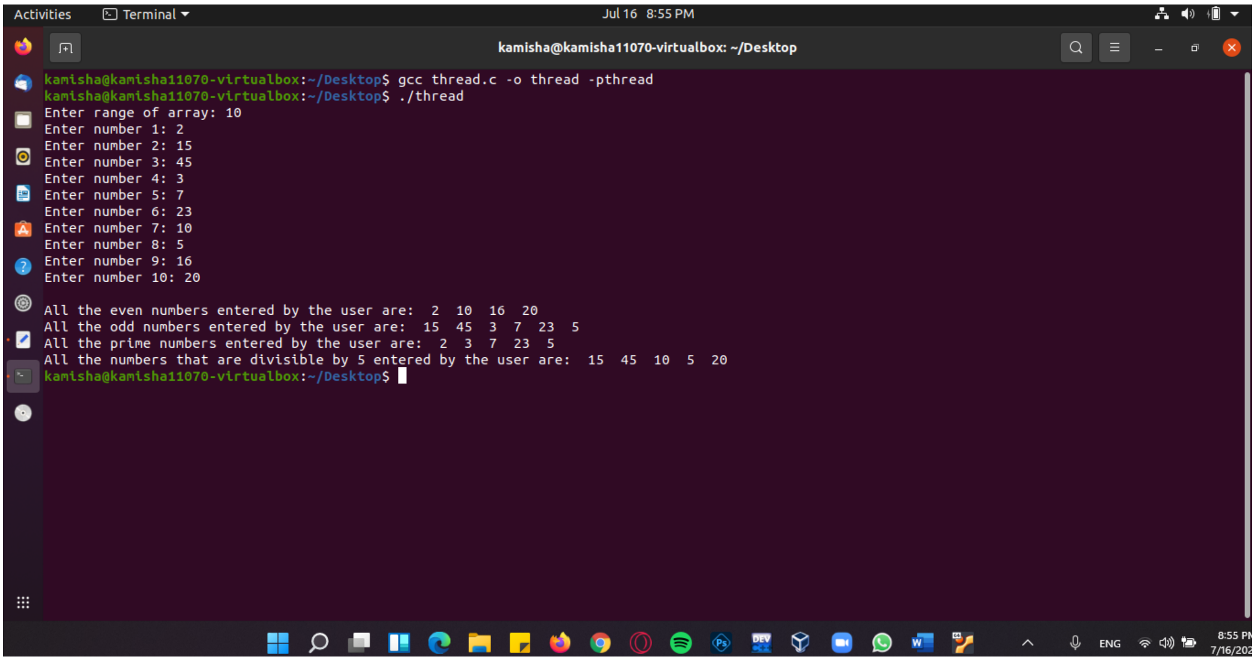


```
                }
                if(c==0)
                {
                        arr1[a]=arr[i];
                        a++;
                }
        }
        printf("All the prime numbers entered by the user are: ");
        for(i=0;i<a;i++)
        {
                printf(" %d ",arr1[i]);
        }
        printf("\n");
}
void *thread4()
{
int t=0;
        for(i=0;i<n;i++)
        {
                if(arr[i]%5==0)
                {
                        arr3[m]=arr[i];
                        m++;
                }
        }
        printf("All the numbers that are divisible by 5 entered by the user are: ");
        for(i=0;i<m;i++)
        {
                printf(" %d ",arr3[i]);
        }
        printf("\n");
```



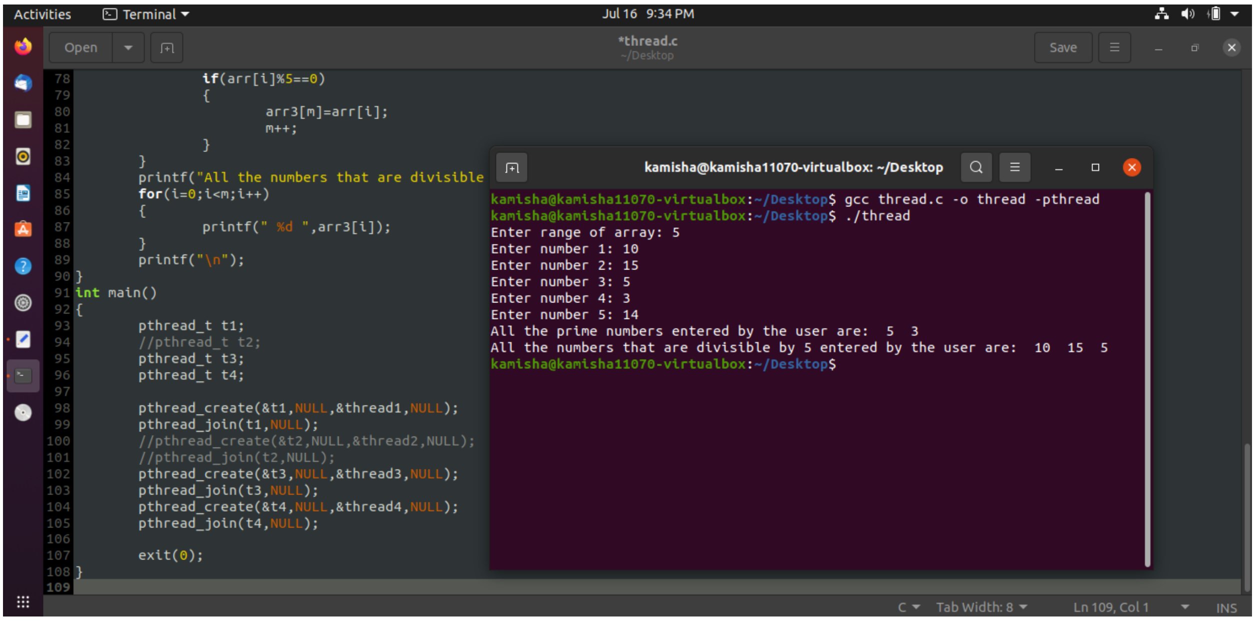
```
        {
                if(arr[i]%5==0)
                {
                        arr3[m]=arr[i];
                        m++;
                }
        }
        printf("All the numbers that are divisible by 5 entered by the user are: ");
        for(i=0;i<m;i++)
        {
                printf(" %d ",arr3[i]);
        }
        printf("\n");
}
int main()
{
        pthread_t t1;
        pthread_t t2;
        pthread_t t3;
        pthread_t t4;

        pthread_create(&t1,NULL,&thread1,NULL);
        pthread_join(t1,NULL);
        pthread_create(&t2,NULL,&thread2,NULL);
        pthread_join(t2,NULL);
        pthread_create(&t3,NULL,&thread3,NULL);
        pthread_join(t3,NULL);
        pthread_create(&t4,NULL,&thread4,NULL);
        pthread_join(t4,NULL);

        exit(0);
}
```

OUTPUT:



```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc thread.c -o thread -pthread
kamisha@kamisha11070-virtualbox:~/Desktop$ ./thread
Enter range of array: 10
Enter number 1: 2
Enter number 2: 15
Enter number 3: 45
Enter number 4: 3
Enter number 5: 7
Enter number 6: 23
Enter number 7: 10
Enter number 8: 5
Enter number 9: 16
Enter number 10: 20

All the even numbers entered by the user are:  2  10  16  20
All the odd numbers entered by the user are:  15  45  3  7  23  5
All the prime numbers entered by the user are:  2  3  7  23  5
All the numbers that are divisible by 5 entered by the user are:  15  45  10  5  20
kamisha@kamisha11070-virtualbox:~/Desktop$
```

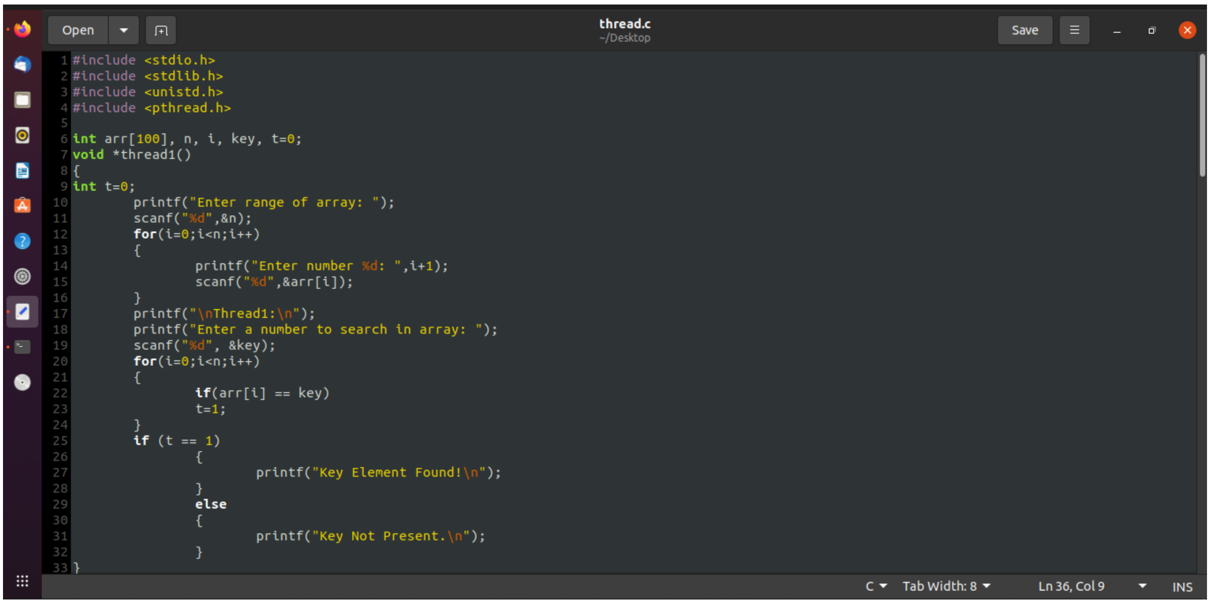
Output if we blocked thread no. 2:



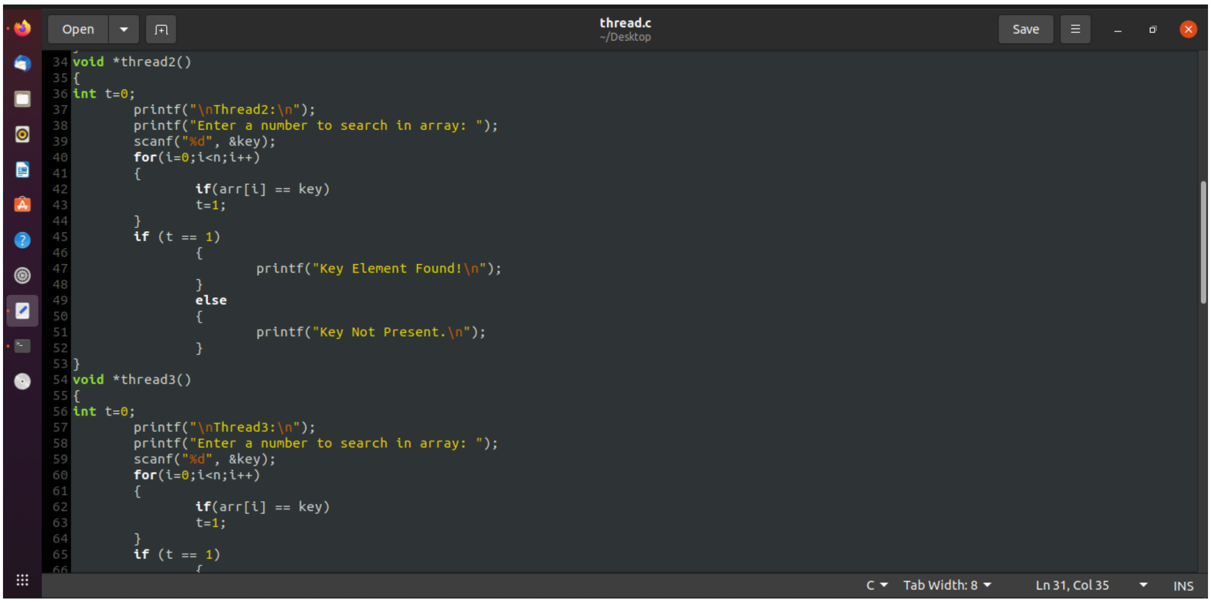
```c
                if(arr[i]%5==0)
                {
                        arr3[m]=arr[i];
                        m++;
                }
        }
        printf("All the numbers that are divisible
        for(i=0;i<m;i++)
        {
                printf(" %d ",arr3[i]);
        }
        printf("\n");
}
int main()
{
        pthread_t t1;
        //pthread_t t2;
        pthread_t t3;
        pthread_t t4;

        pthread_create(&t1,NULL,&thread1,NULL);
        pthread_join(t1,NULL);
        //pthread_create(&t2,NULL,&thread2,NULL);
        //pthread_join(t2,NULL);
        pthread_create(&t3,NULL,&thread3,NULL);
        pthread_join(t3,NULL);
        pthread_create(&t4,NULL,&thread4,NULL);
        pthread_join(t4,NULL);

        exit(0);
}
```

```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc thread.c -o thread -pthread
kamisha@kamisha11070-virtualbox:~/Desktop$ ./thread
Enter range of array: 5
Enter number 1: 10
Enter number 2: 15
Enter number 3: 5
Enter number 4: 3
Enter number 5: 14
All the prime numbers entered by the user are:  5  3
All the numbers that are divisible by 5 entered by the user are:  10  15  5
kamisha@kamisha11070-virtualbox:~/Desktop$
```

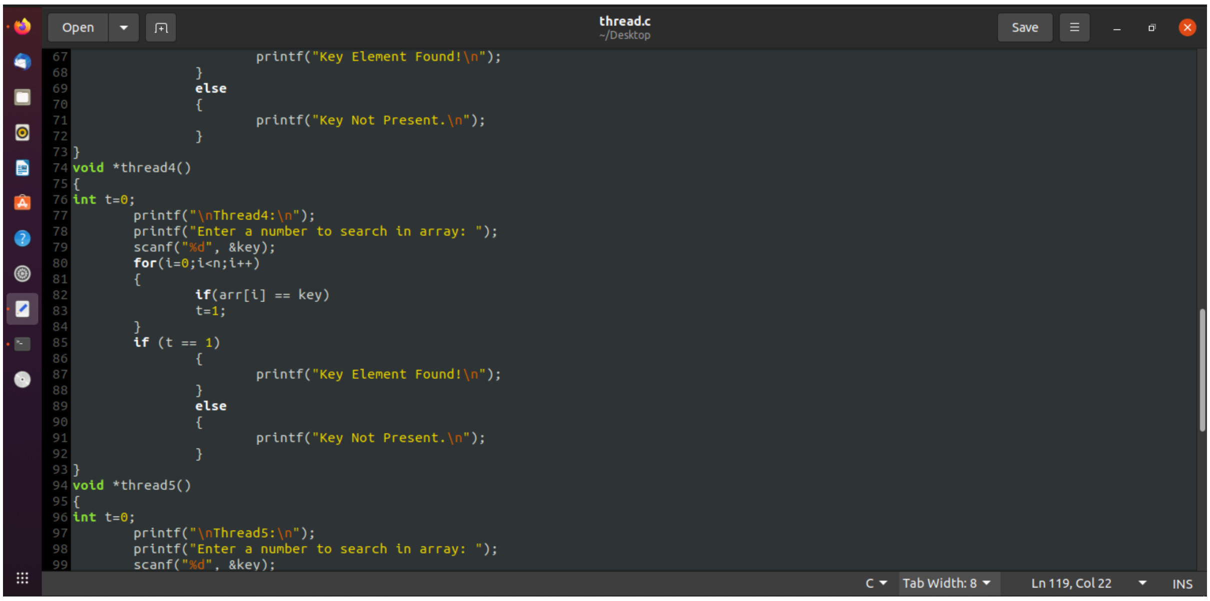
Example – 2: This example is similar to the first example. Here we are searching in the array five times and likewise the first example, if we blocked a thread here, then we will be searching through the array one time less, i.e.: 4 times w.r.t. to our example and can vary. But if we used single-threading here, we would've not been able to block any processes unless deleted by the user itself.



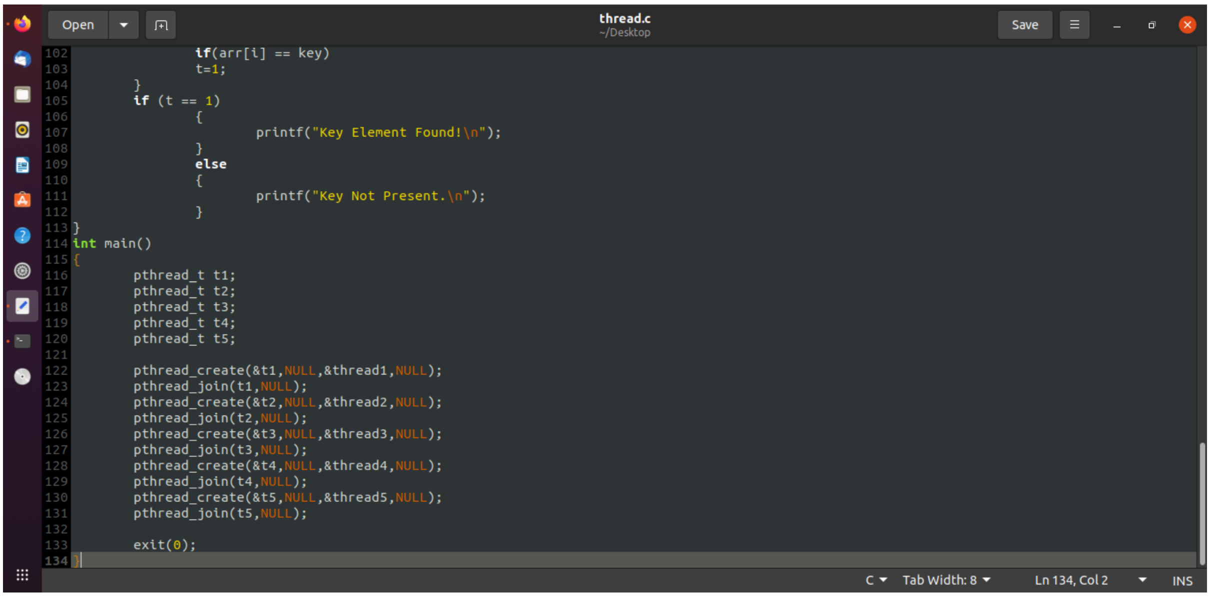
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

int arr[100], n, i, key, t=0;
void *thread1()
{
int t=0;
        printf("Enter range of array: ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter number %d: ",i+1);
                scanf("%d",&arr[i]);
        }
        printf("\nThread1:\n");
        printf("Enter a number to search in array: ");
        scanf("%d", &key);
        for(i=0;i<n;i++)
        {
                if(arr[i] == key)
                t=1;
        }
        if (t == 1)
                {
                        printf("Key Element Found!\n");
                }
                else
                {
                        printf("Key Not Present.\n");
                }
}
```



```c
void *thread2()
{
int t=0;
        printf("\nThread2:\n");
        printf("Enter a number to search in array: ");
        scanf("%d", &key);
        for(i=0;i<n;i++)
        {
                if(arr[i] == key)
                t=1;
        }
        if (t == 1)
                {
                        printf("Key Element Found!\n");
                }
                else
                {
                        printf("Key Not Present.\n");
                }
}
void *thread3()
{
int t=0;
        printf("\nThread3:\n");
        printf("Enter a number to search in array: ");
        scanf("%d", &key);
        for(i=0;i<n;i++)
        {
                if(arr[i] == key)
                t=1;
        }
        if (t == 1)
                {
```

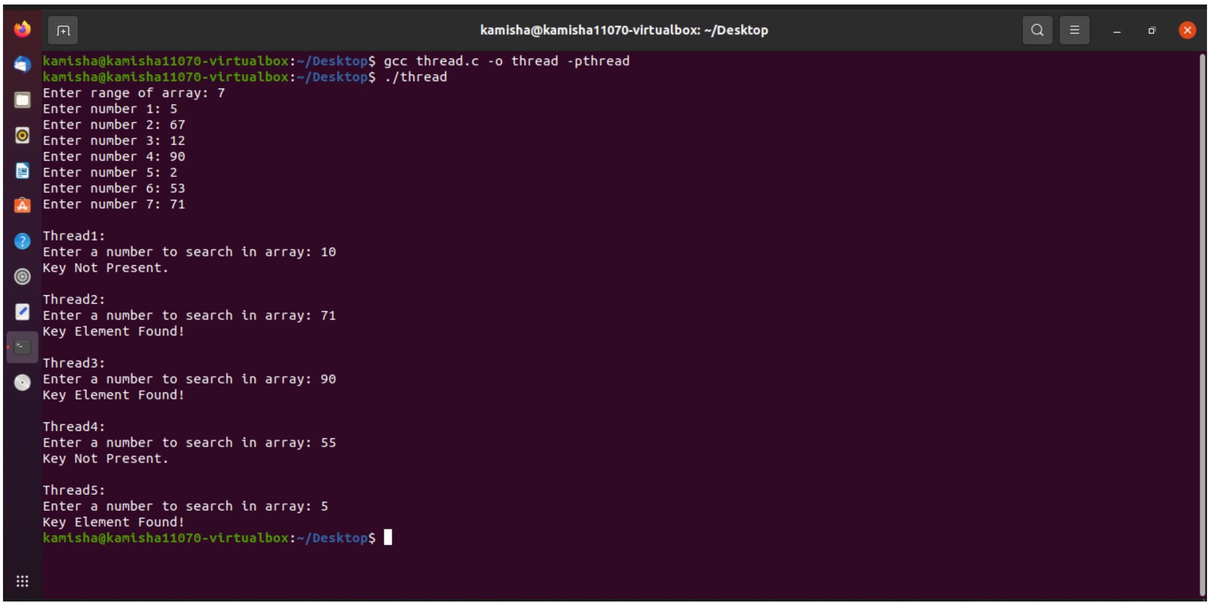


```c
                        printf("Key Element Found!\n");
                }
                else
                {
                        printf("Key Not Present.\n");
                }
}
void *thread4()
{
int t=0;
        printf("\nThread4:\n");
        printf("Enter a number to search in array: ");
        scanf("%d", &key);
        for(i=0;i<n;i++)
        {
                if(arr[i] == key)
                t=1;
        }
        if (t == 1)
                {
                        printf("Key Element Found!\n");
                }
                else
                {
                        printf("Key Not Present.\n");
                }
}
void *thread5()
{
int t=0;
        printf("\nThread5:\n");
        printf("Enter a number to search in array: ");
        scanf("%d", &key);
```



```c
                if(arr[i] == key)
                t=1;
        }
        if (t == 1)
                {
                        printf("Key Element Found!\n");
                }
                else
                {
                        printf("Key Not Present.\n");
                }
}
int main()
{
        pthread_t t1;
        pthread_t t2;
        pthread_t t3;
        pthread_t t4;
        pthread_t t5;

        pthread_create(&t1,NULL,&thread1,NULL);
        pthread_join(t1,NULL);
        pthread_create(&t2,NULL,&thread2,NULL);
        pthread_join(t2,NULL);
        pthread_create(&t3,NULL,&thread3,NULL);
        pthread_join(t3,NULL);
        pthread_create(&t4,NULL,&thread4,NULL);
        pthread_join(t4,NULL);
        pthread_create(&t5,NULL,&thread5,NULL);
        pthread_join(t5,NULL);

        exit(0);
}
```

OUTPUT:

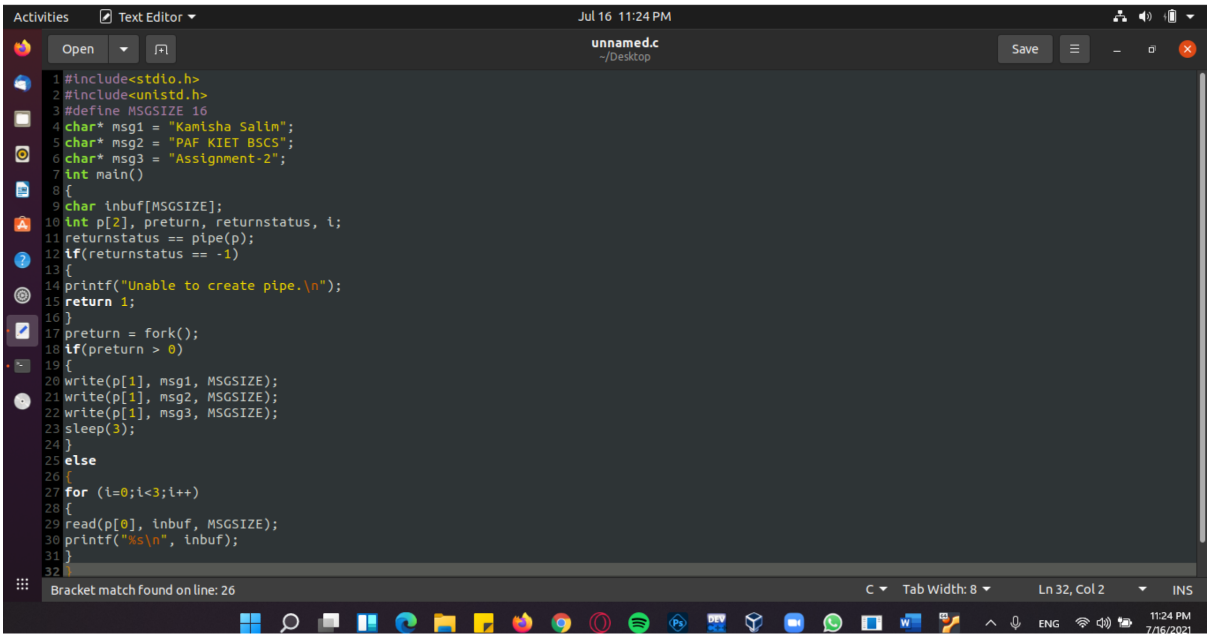


```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc thread.c -o thread -pthread
kamisha@kamisha11070-virtualbox:~/Desktop$ ./thread
Enter range of array: 7
Enter number 1: 5
Enter number 2: 67
Enter number 3: 12
Enter number 4: 90
Enter number 5: 2
Enter number 6: 53
Enter number 7: 71

Thread1:
Enter a number to search in array: 10
Key Not Present.

Thread2:
Enter a number to search in array: 71
Key Element Found!

Thread3:
Enter a number to search in array: 90
Key Element Found!

Thread4:
Enter a number to search in array: 55
Key Not Present.

Thread5:
Enter a number to search in array: 5
Key Element Found!
kamisha@kamisha11070-virtualbox:~/Desktop$
```

QUESTION – 2

Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

Example – 1: Unnamed pipes are only used for communication between a child and its parent process. It can either be a one-way or a two-way communication. So, for such communications, ordinary pipes or unnamed pipes are more suitable than named pipes.

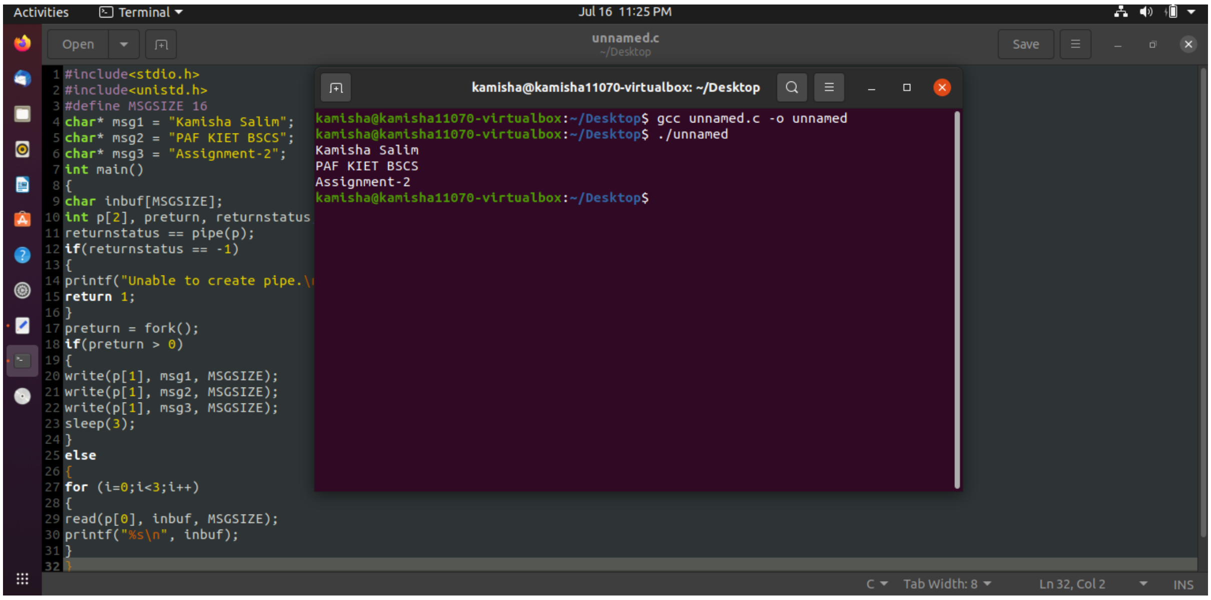


```c
#include<stdio.h>
#include<unistd.h>
#define MSGSIZE 16
char* msg1 = "Kamisha Salim";
char* msg2 = "PAF KIET BSCS";
char* msg3 = "Assignment-2";
int main()
{
char inbuf[MSGSIZE];
int p[2], preturn, returnstatus, i;
returnstatus == pipe(p);
if(returnstatus == -1)
{
printf("Unable to create pipe.\n");
return 1;
}
preturn = fork();
if(preturn > 0)
{
write(p[1], msg1, MSGSIZE);
write(p[1], msg2, MSGSIZE);
write(p[1], msg3, MSGSIZE);
sleep(3);
}
else
{
for (i=0;i<3;i++)
{
read(p[0], inbuf, MSGSIZE);
printf("%s\n", inbuf);
}
}
```

Bracket match found on line: 26

OUTPUT:

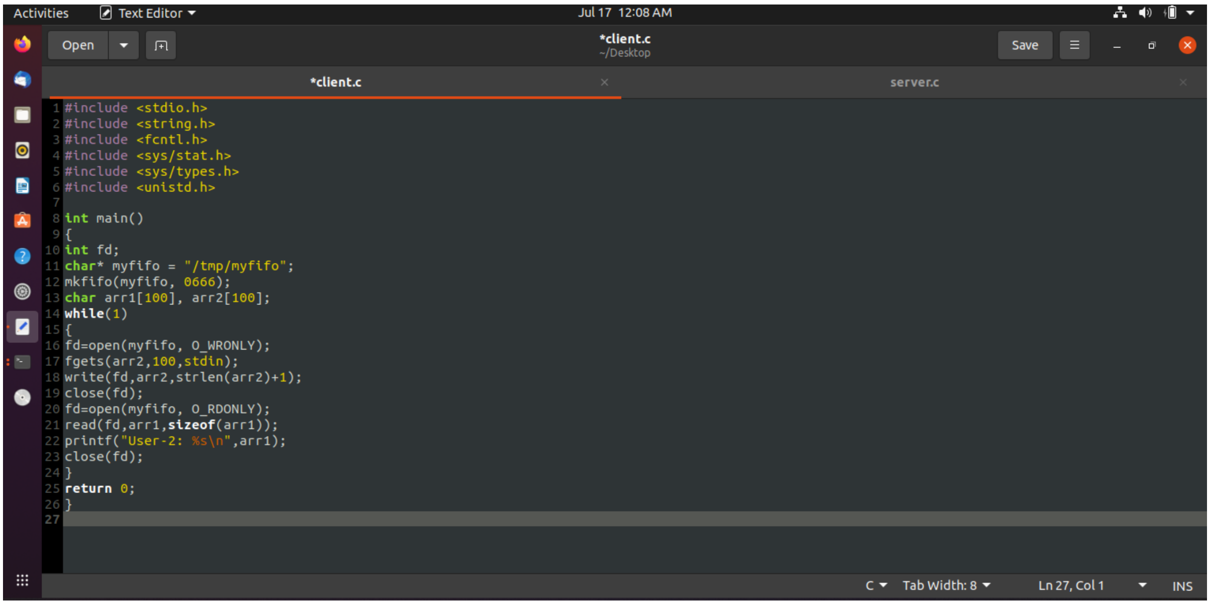
One-way communication between a child and its parent process.



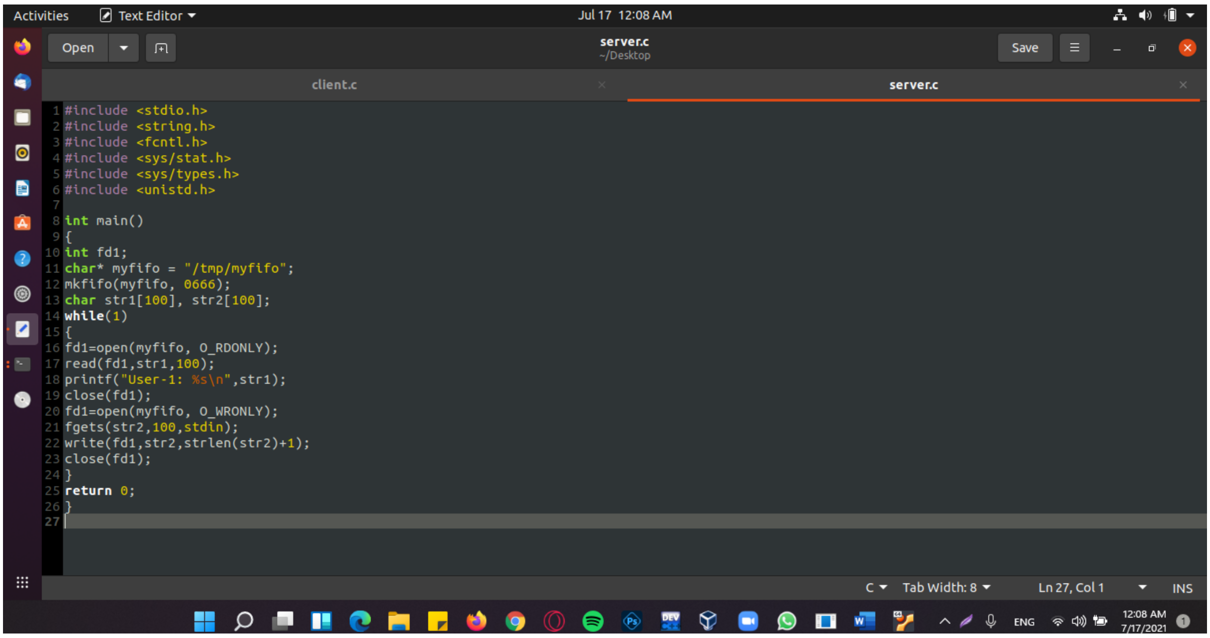
```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc unnamed.c -o unnamed
kamisha@kamisha11070-virtualbox:~/Desktop$ ./unnamed
Kamisha Salim
PAF KIET BSCS
Assignment-2
kamisha@kamisha11070-virtualbox:~/Desktop$
```

```c
#include<stdio.h>
#include<unistd.h>
#define MSGSIZE 16
char* msg1 = "Kamisha Salim";
char* msg2 = "PAF KIET BSCS";
char* msg3 = "Assignment-2";
int main()
{
char inbuf[MSGSIZE];
int p[2], preturn, returnstatus
returnstatus == pipe(p);
if(returnstatus == -1)
{
printf("Unable to create pipe.\
return 1;
}
preturn = fork();
if(preturn > 0)
{
write(p[1], msg1, MSGSIZE);
write(p[1], msg2, MSGSIZE);
write(p[1], msg3, MSGSIZE);
sleep(3);
}
else
{
for (i=0;i<3;i++)
{
read(p[0], inbuf, MSGSIZE);
printf("%s\n", inbuf);
}
}
```

Example – 2: Named pipes are meant for communication between two or more unrelated or unnamed processes and can also have bi-directional communication. So, for such communications, named pipes are more suitable than ordinary pipes.



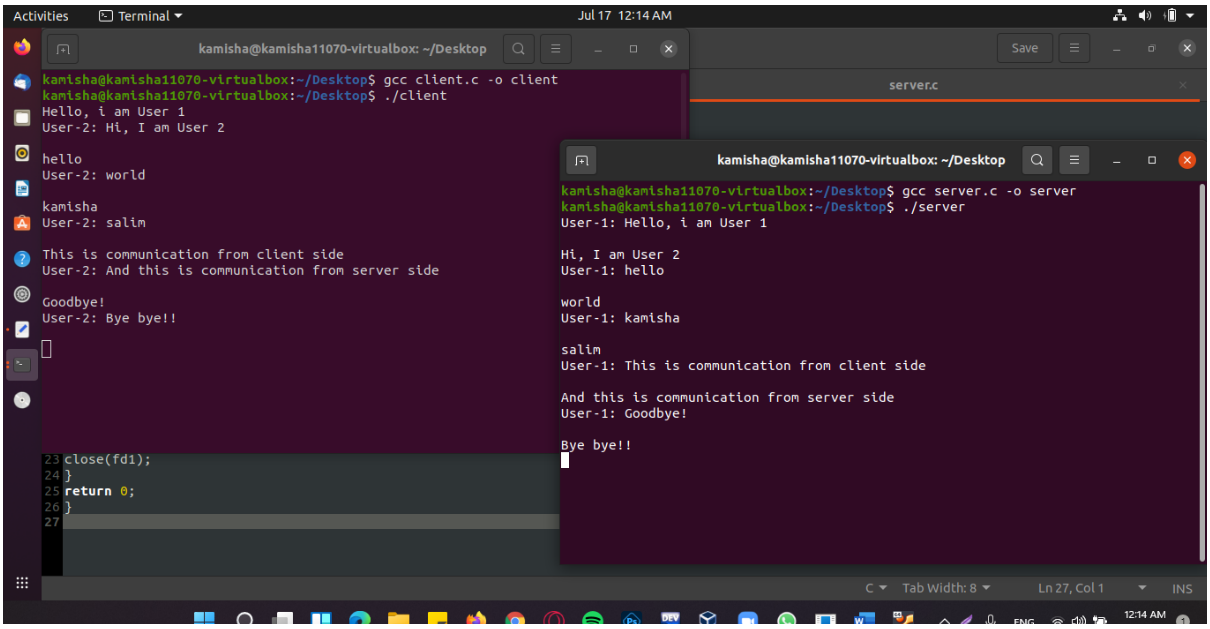
```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
int fd;
char* myfifo = "/tmp/myfifo";
mkfifo(myfifo, 0666);
char arr1[100], arr2[100];
while(1)
{
fd=open(myfifo, O_WRONLY);
fgets(arr2,100,stdin);
write(fd,arr2,strlen(arr2)+1);
close(fd);
fd=open(myfifo, O_RDONLY);
read(fd,arr1,sizeof(arr1));
printf("User-2: %s\n",arr1);
close(fd);
}
return 0;
}
```



```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
int fd1;
char* myfifo = "/tmp/myfifo";
mkfifo(myfifo, 0666);
char str1[100], str2[100];
while(1)
{
fd1=open(myfifo, O_RDONLY);
read(fd1,str1,100);
printf("User-1: %s\n",str1);
close(fd1);
fd1=open(myfifo, O_WRONLY);
fgets(str2,100,stdin);
write(fd1,str2,strlen(str2)+1);
close(fd1);
}
return 0;
}
```

OUTPUT:

A chat app like communication (two-way communication) is established through named pipes.



Activities
Terminal
Jul 17 12:14 AM
kamisha@kamisha11070-virtualbox: ~/Desktop
Save
server.c
kamisha@kamisha11070-virtualbox: ~/Desktop
C
Tab Width: 8
Ln 27, Col 1
INS
12:14 AM

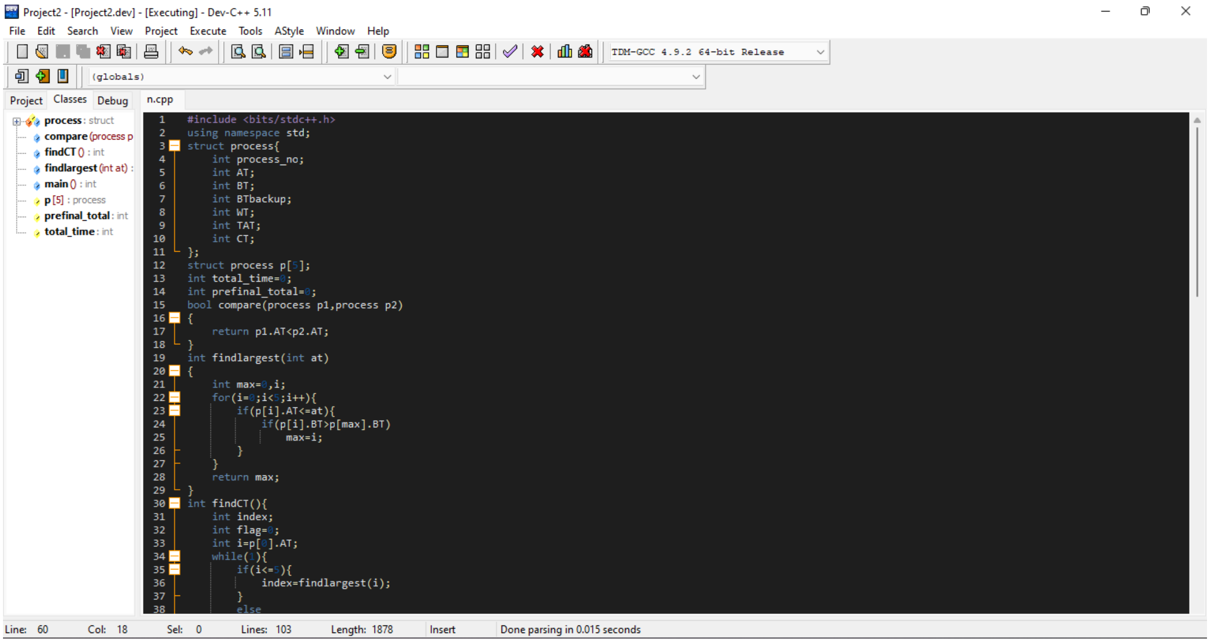
```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc client.c -o client
kamisha@kamisha11070-virtualbox:~/Desktop$ ./client
Hello, i am User 1
User-2: Hi, I am User 2

hello
User-2: world

kamisha
User-2: salim

This is communication from client side
User-2: And this is communication from server side

Goodbye!
User-2: Bye bye!!
```

```
close(fd1);
}
return 0;
}
```

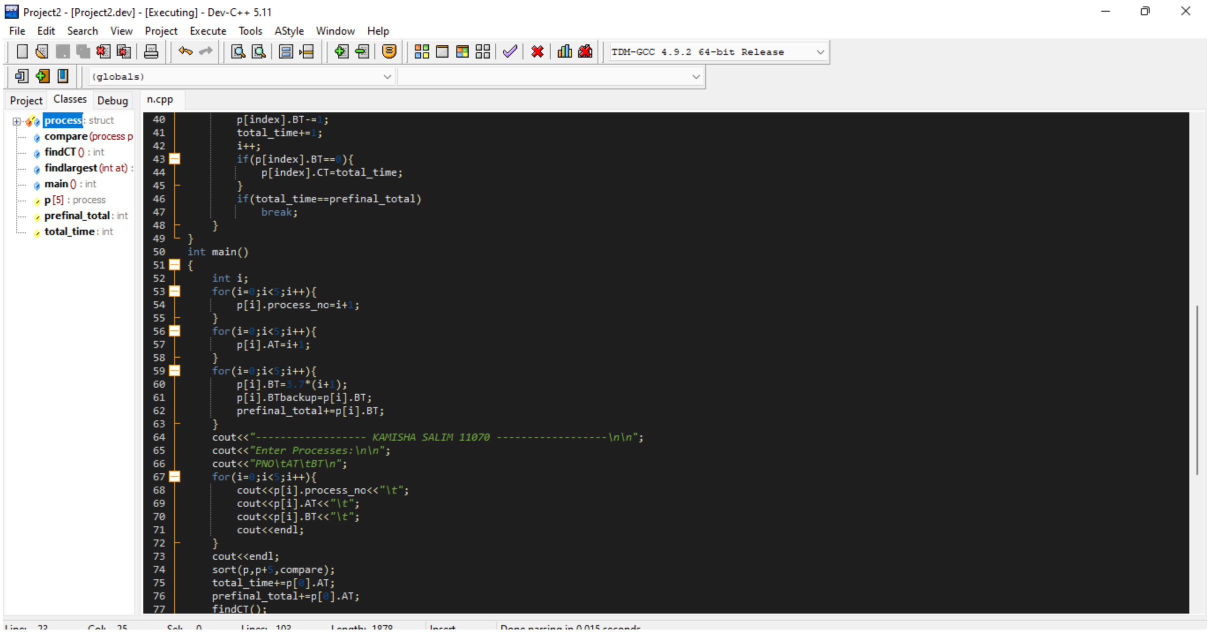
```
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc server.c -o server
kamisha@kamisha11070-virtualbox:~/Desktop$ ./server
User-1: Hello, i am User 1

Hi, I am User 2
User-1: hello

world
User-1: kamisha

salim
User-1: This is communication from client side

And this is communication from server side
User-1: Goodbye!

Bye bye!!
```

QUESTION – 3

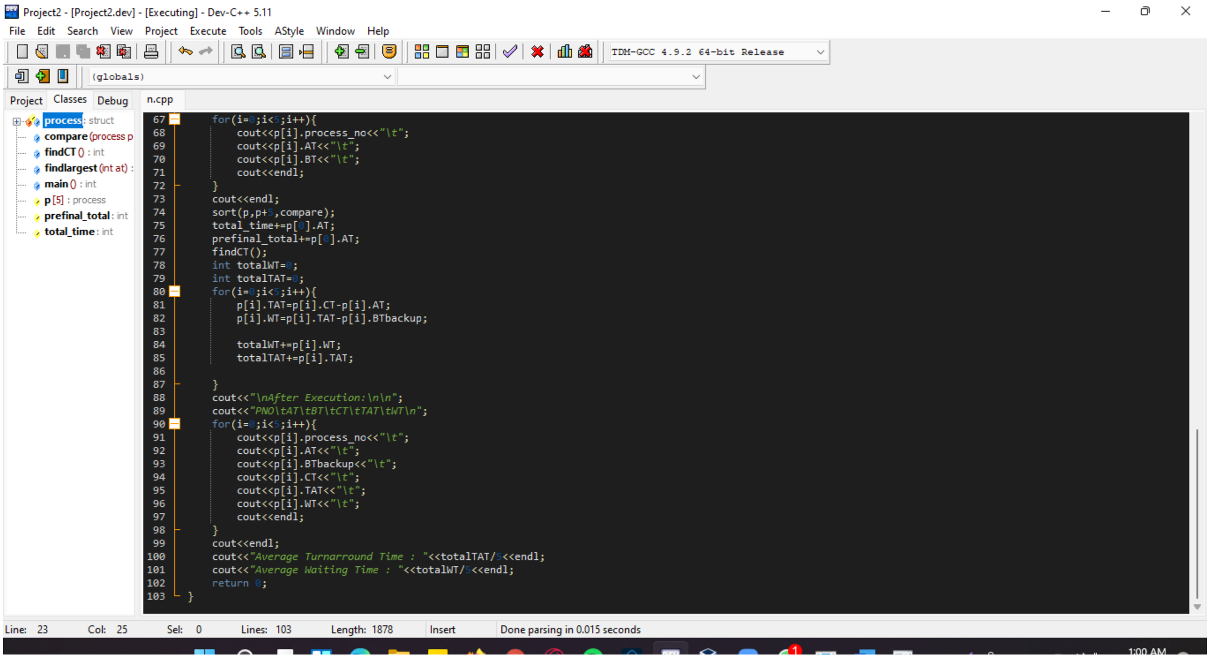
Implement The Longest Remaining Time First (LRTF) scheduling.


Project2 - [Project2.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
n.cpp
process : struct
compare (process p
findCT () : int
findlargest (int at) :
main () : int
p [5] : process
prefinal_total : int
total_time : int
Line: 60 Col: 18 Sel: 0 Lines: 103 Length: 1878 Insert Done parsing in 0.015 seconds

```cpp
#include <bits/stdc++.h>
using namespace std;
struct process{
    int process_no;
    int AT;
    int BT;
    int BTbackup;
    int WT;
    int TAT;
    int CT;
};
struct process p[5];
int total_time=0;
int prefinal_total=0;
bool compare(process p1,process p2)
{
    return p1.AT<p2.AT;
}
int findlargest(int at)
{
    int max=0,i;
    for(i=0;i<5;i++){
        if(p[i].AT<=at){
            if(p[i].BT>p[max].BT)
                max=i;
        }
    }
    return max;
}
int findCT(){
    int index;
    int flag=0;
    int i=p[0].AT;
    while(1){
        if(i<=5){
            index=findlargest(i);
        }
        else
```

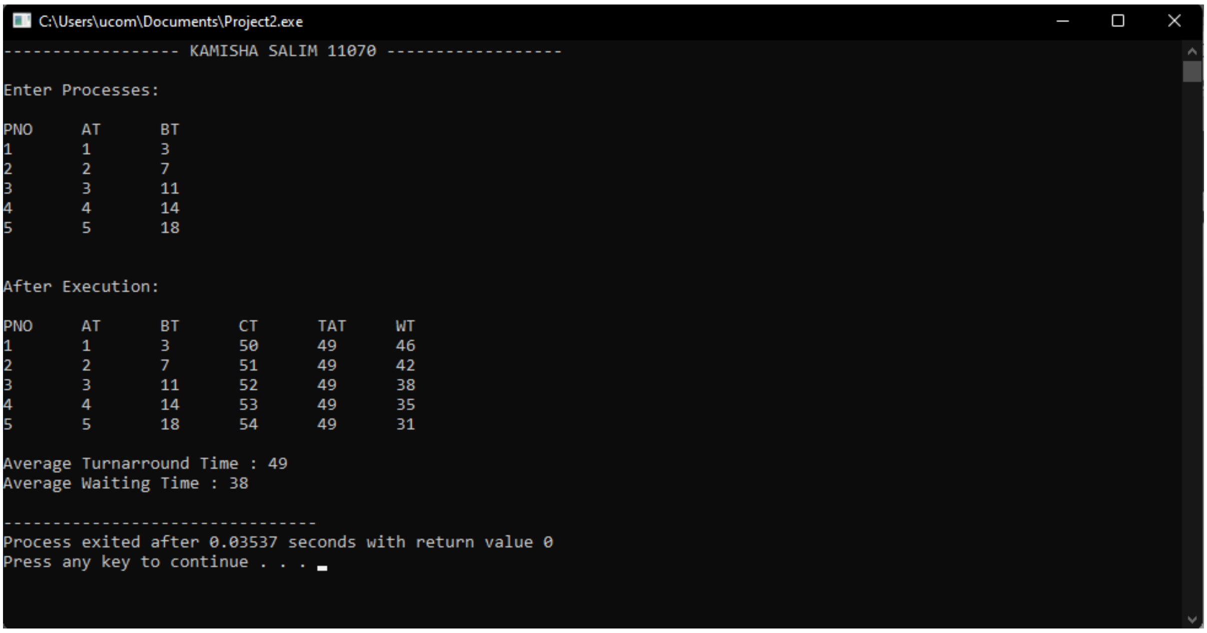


```cpp
            p[index].BT-=1;
            total_time+=1;
            i++;
            if(p[index].BT==0){
                p[index].CT=total_time;
            }
            if(total_time==prefinal_total)
                break;
    }
}
int main()
{
    int i;
    for(i=0;i<5;i++){
        p[i].process_no=i+1;
    }
    for(i=0;i<5;i++){
        p[i].AT=i+1;
    }
    for(i=0;i<5;i++){
        p[i].BT=3.7*(i+1);
        p[i].BTbackup=p[i].BT;
        prefinal_total+=p[i].BT;
    }
    cout<<"----------------- KAMISHA SALIM 11070 -----------------\n\n";
    cout<<"Enter Processes:\n\n";
    cout<<"PNO\tAT\tBT\n";
    for(i=0;i<5;i++){
        cout<<p[i].process_no<<"\t";
        cout<<p[i].AT<<"\t";
        cout<<p[i].BT<<"\t";
        cout<<endl;
    }
    cout<<endl;
    sort(p,p+5,compare);
    total_time+=p[0].AT;
    prefinal_total+=p[0].AT;
    findCT();
```



```cpp
    for(i=0;i<5;i++){
        cout<<p[i].process_no<<"\t";
        cout<<p[i].AT<<"\t";
        cout<<p[i].BT<<"\t";
        cout<<endl;
    }
    cout<<endl;
    sort(p,p+5,compare);
    total_time+=p[0].AT;
    prefinal_total+=p[0].AT;
    findCT();
    int totalWT=0;
    int totalTAT=0;
    for(i=0;i<5;i++){
        p[i].TAT=p[i].CT-p[i].AT;
        p[i].WT=p[i].TAT-p[i].BTbackup;

        totalWT+=p[i].WT;
        totalTAT+=p[i].TAT;

    }
    cout<<"\nAfter Execution:\n\n";
    cout<<"PNO\tAT\tBT\tCT\tTAT\tWT\n";
    for(i=0;i<5;i++){
        cout<<p[i].process_no<<"\t";
        cout<<p[i].AT<<"\t";
        cout<<p[i].BTbackup<<"\t";
        cout<<p[i].CT<<"\t";
        cout<<p[i].TAT<<"\t";
        cout<<p[i].WT<<"\t";
        cout<<endl;
    }
    cout<<endl;
    cout<<"Average Turnarround Time : "<<totalTAT/5<<endl;
    cout<<"Average Waiting Time : "<<totalWT/5<<endl;
    return 0;
}
```

OUTPUT:



```
----------------- KAMISHA SALIM 11070 -----------------

Enter Processes:

PNO     AT      BT
1       1       3
2       2       7
3       3       11
4       4       14
5       5       18


After Execution:

PNO     AT      BT      CT      TAT     WT
1       1       3       50      49      46
2       2       7       51      49      42
3       3       11      52      49      38
4       4       14      53      49      35
5       5       18      54      49      31

Average Turnarround Time : 49
Average Waiting Time : 38

--------------------------------
Process exited after 0.03537 seconds with return value 0
Press any key to continue . . .
```