

OPERATING SYSTEMS ASSIGNMENT – 02

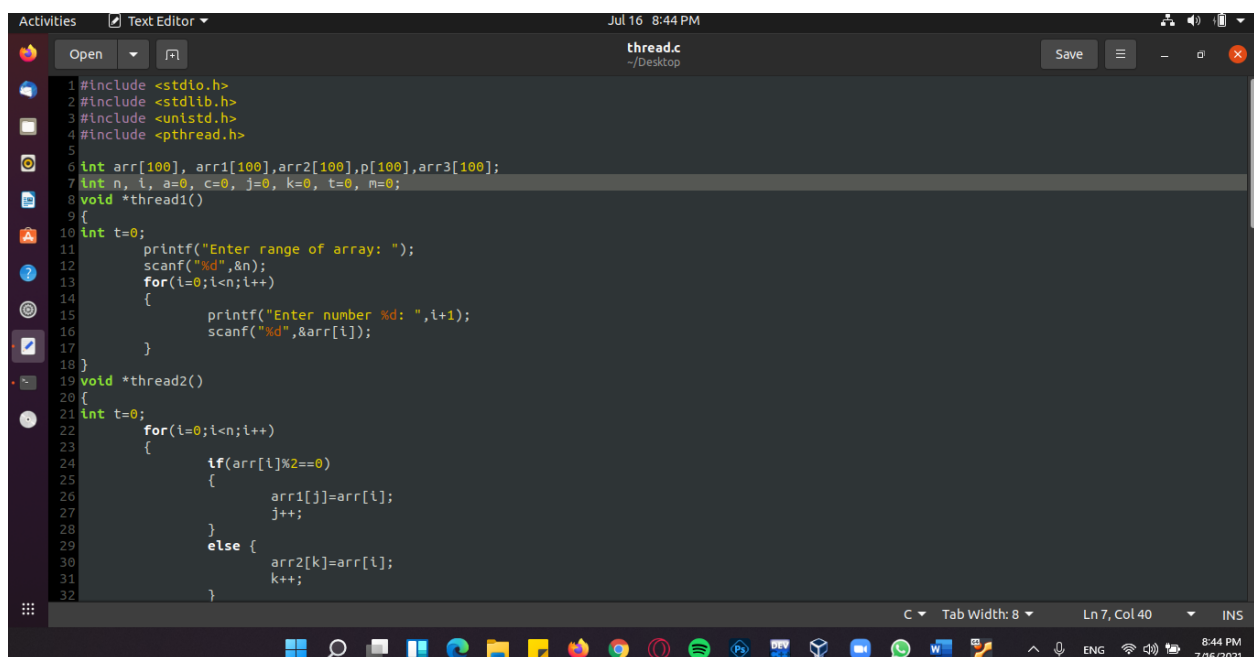
Name: Kamisha Salim

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.

A screenshot of a Linux desktop environment showing a text editor window titled 'thread.c' at the path '~/Desktop'. The editor contains C++ code for a multithreaded program. The code includes headers for stdio, stdlib, unistd, and pthread. It declares an array 'arr' of size 100 and pointers to it. The main function 'main' initializes variables and calls 'pthread_create' to create three threads: 'thread1', 'thread2', and 'thread3'. 'thread1' is responsible for taking user input for the array. 'thread2' and 'thread3' process the array elements based on a condition (if the element is even or odd). The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 int arr[100], arr1[100], arr2[100], p[100], arr3[100];
7 int n, i, a=0, c=0, j=0, k=0, t=0, m=0;
8 void *thread1()
9 {
10  int t=0;
11  printf("Enter range of array: ");
12  scanf("%d",&n);
13  for(i=0;i<n;i++)
14  {
15      printf("Enter number %d: ",i+1);
16      scanf("%d",&arr[i]);
17  }
18 }
19 void *thread2()
20 {
21  int t=0;
22  for(i=0;i<n;i++)
23  {
24      if(arr[i]%2==0)
25      {
26          arr1[j]=arr[i];
27          j++;
28      }
29      else {
30          arr2[k]=arr[i];
31          k++;
32      }
33  }
```

The bottom of the image shows the Ubuntu desktop with various application icons in the dock and the system status bar at the bottom right indicating the time as 8:44 PM on 7/16/2021.

```
Activities Text Editor Jul 16 8:48 PM
thread.c ~/Desktop Save

28 }
29 else {
30     arr2[k]=arr[i];
31     k++;
32 }
33 }
34 printf("\nAll the even numbers entered by the user are: ");
35 for(i=0;i<j;i++)
36 {
37     printf(" %d ",arr1[i]);
38 }
39 printf("\nAll the odd numbers entered by the user are: ");
40 for(i=0;i<k;i++)
41 {
42     printf(" %d ",arr2[i]);
43 }
44 printf("\n");
45 }
46 void *thread3()
47 {
48     int t=0;
49     for(i=0;i<n;i++)
50     {
51         c=0;
52         for(j=2;j<arr[i];j++)
53         {
54             if(arr[i]%j==0)
55             {
56                 c=1;
57                 break;
58             }
59         }

```

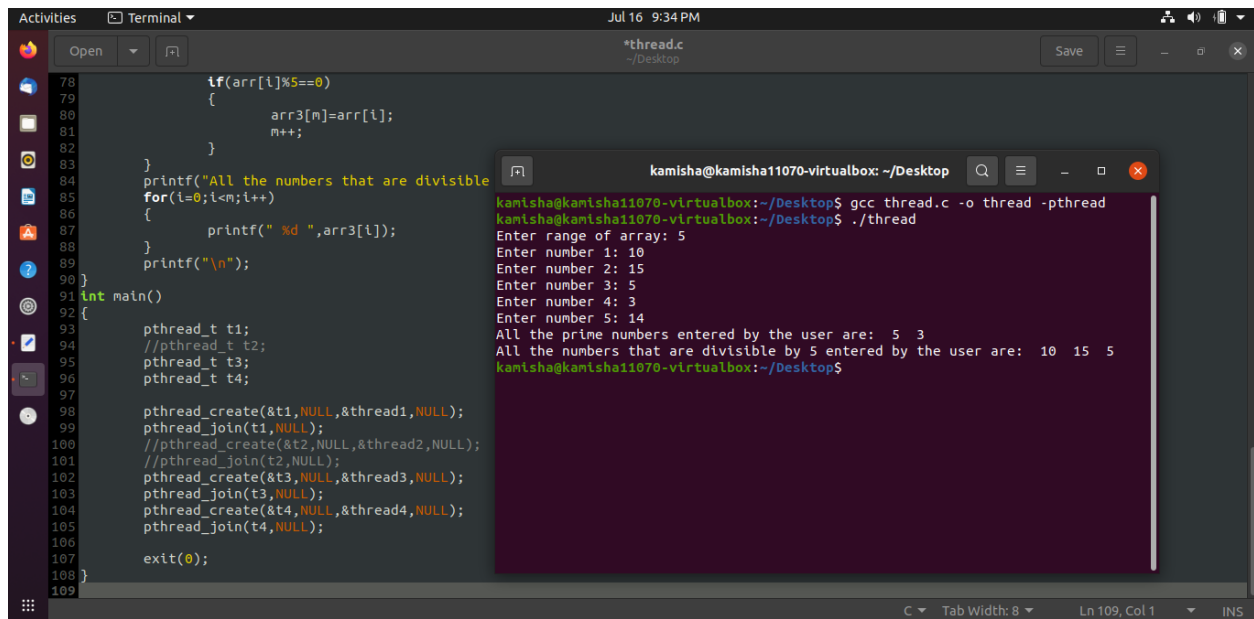
```
Activities Text Editor Jul 16 8:49 PM
thread.c ~/Desktop Save

59 }
60 if(c==0)
61 {
62     arr1[a]=arr[i];
63     a++;
64 }
65 }
66 printf("All the prime numbers entered by the user are: ");
67 for(i=0;i<a;i++)
68 {
69     printf(" %d ",arr1[i]);
70 }
71 printf("\n");
72 }
73 void *thread4()
74 {
75     int t=0;
76     for(i=0;i<n;i++)
77     {
78         if(arr[i]%5==0)
79         {
80             arr3[m]=arr[i];
81             m++;
82         }
83     }
84     printf("All the numbers that are divisible by 5 entered by the user are: ");
85     for(i=0;i<m;i++)
86     {
87         printf(" %d ",arr3[i]);
88     }
89     printf("\n");

```

OUTPUT:

Output if we blocked thread no. 2:



```
78     if(arr[i]%5==0)
79     {
80         arr3[m]=arr[i];
81         m++;
82     }
83 }
84 printf("All the numbers that are divisible
85 for(i=0;i<m;i++)
86 {
87     printf(" %d ",arr3[i]);
88 }
89 printf("\n");
90 }
91 int main()
92 {
93     pthread_t t1;
94     //pthread_t t2;
95     pthread_t t3;
96     pthread_t t4;
97
98     pthread_create(&t1,NULL,&thread1,NULL);
99     pthread_join(t1,NULL);
100    //pthread_create(&t2,NULL,&thread2,NULL);
101    //pthread_join(t2,NULL);
102    pthread_create(&t3,NULL,&thread3,NULL);
103    pthread_join(t3,NULL);
104    pthread_create(&t4,NULL,&thread4,NULL);
105    pthread_join(t4,NULL);
106
107    exit(0);
108 }
109
```

```
kamisha@kamisha11070-virtualbox: ~/Desktop
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.



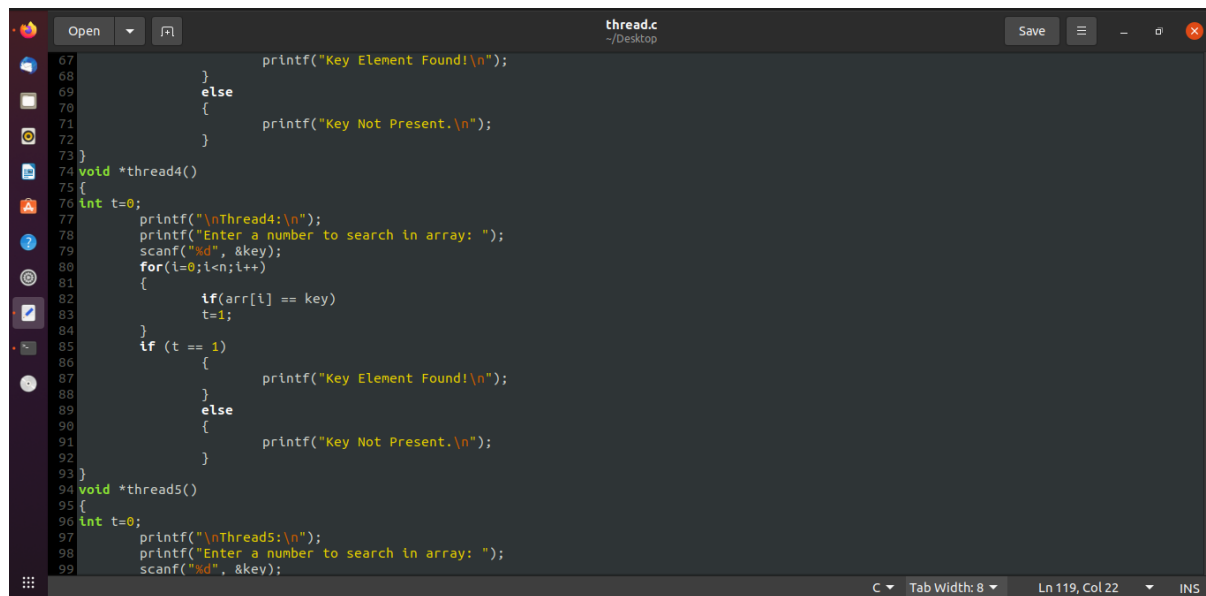
```
1#include <stdio.h>
2#include <stdlib.h>
3#include <unistd.h>
4#include <pthread.h>
5
6int arr[100], n, i, key, t=0;
7void *thread1()
8{
9    int t=0;
10    printf("Enter range of array: ");
11    scanf("%d",&n);
12    for(i=0;i<n;i++)
13    {
14        printf("Enter number %d: ",i+1);
15        scanf("%d",&arr[i]);
16    }
17    printf("\nThread1:\n");
18    printf("Enter a number to search in array: ");
19    scanf("%d", &key);
20    for(i=0;i<n;i++)
21    {
22        if(arr[i] == key)
23            t=1;
24    }
25    if (t == 1)
26    {
27        printf("Key Element Found!\n");
28    }
29    else
30    {
31        printf("Key Not Present.\n");
32    }
33 }
```



The screenshot shows a code editor window titled "thread.c" with the file path "~/Desktop". The editor contains two functions, `thread2()` and `thread3()`, both implemented as `void` pointers. Each function starts with a `printf` statement to announce its execution, followed by a `scanf` statement to read a number from the user. A `for` loop then iterates through an array `arr` from index 0 to `n-1`. Inside the loop, an `if` statement checks if the current element `arr[i]` is equal to the user-provided `key`. If a match is found, a flag `t` is set to 1. After the loop, another `if` statement checks the value of `t`. If `t` is 1, it prints "Key Element Found!\n"; otherwise, it prints "Key Not Present.\n".

```
34 void *thread2()
35 {
36     int t=0;
37     printf("\nThread2:\n");
38     printf("Enter a number to search in array: ");
39     scanf("%d", &key);
40     for(i=0;i<n;i++)
41     {
42         if(arr[i] == key)
43             t=1;
44     }
45     if (t == 1)
46     {
47         printf("Key Element Found!\n");
48     }
49     else
50     {
51         printf("Key Not Present.\n");
52     }
53 }
54 void *thread3()
55 {
56     int t=0;
57     printf("\nThread3:\n");
58     printf("Enter a number to search in array: ");
59     scanf("%d", &key);
60     for(i=0;i<n;i++)
61     {
62         if(arr[i] == key)
63             t=1;
64     }
65     if (t == 1)
66     {
67         printf("Key Element Found!\n");
68     }
69     else
70     {
71         printf("Key Not Present.\n");
72     }
73 }
74 void *thread4()
75 {
76     int t=0;
77     printf("\nThread4:\n");
78     printf("Enter a number to search in array: ");
79     scanf("%d", &key);
80     for(i=0;i<n;i++)
81     {
82         if(arr[i] == key)
83             t=1;
84     }
85     if (t == 1)
86     {
87         printf("Key Element Found!\n");
88     }
89     else
90     {
91         printf("Key Not Present.\n");
92     }
93 }
94 void *thread5()
95 {
96     int t=0;
97     printf("\nThread5:\n");
98     printf("Enter a number to search in array: ");
99     scanf("%d", &key);
```

The status bar at the bottom indicates "C", "Tab Width: 8", "Ln 31, Col 35", and "INS".



The screenshot shows the same code editor window, now displaying the implementation of `thread4()` and `thread5()`. The structure of these functions is identical to `thread2()` and `thread3()`, respectively. Each function prints its name, reads a search key, iterates through the array, and reports whether the key was found. The code continues from line 99.

```
99     scanf("%d", &key);
```

The status bar at the bottom indicates "C", "Tab Width: 8", "Ln 119, Col 22", and "INS".

```
thread.c
~/Desktop

102     if(arr[i] == key)
103     {
104         t=1;
105     }
106     if (t == 1)
107     {
108         printf("Key Element Found!\n");
109     }
110     else
111     {
112         printf("Key Not Present.\n");
113     }
114 }
115 int main()
116 {
117     pthread_t t1;
118     pthread_t t2;
119     pthread_t t3;
120     pthread_t t4;
121     pthread_t t5;
122     pthread_create(&t1,NULL,&thread1,NULL);
123     pthread_join(t1,NULL);
124     pthread_create(&t2,NULL,&thread2,NULL);
125     pthread_join(t2,NULL);
126     pthread_create(&t3,NULL,&thread3,NULL);
127     pthread_join(t3,NULL);
128     pthread_create(&t4,NULL,&thread4,NULL);
129     pthread_join(t4,NULL);
130     pthread_create(&t5,NULL,&thread5,NULL);
131     pthread_join(t5,NULL);
132     exit(0);
133 }
134
```

OUTPUT:

```
kamisha@kamisha11070-virtualbox: ~/Desktop

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!

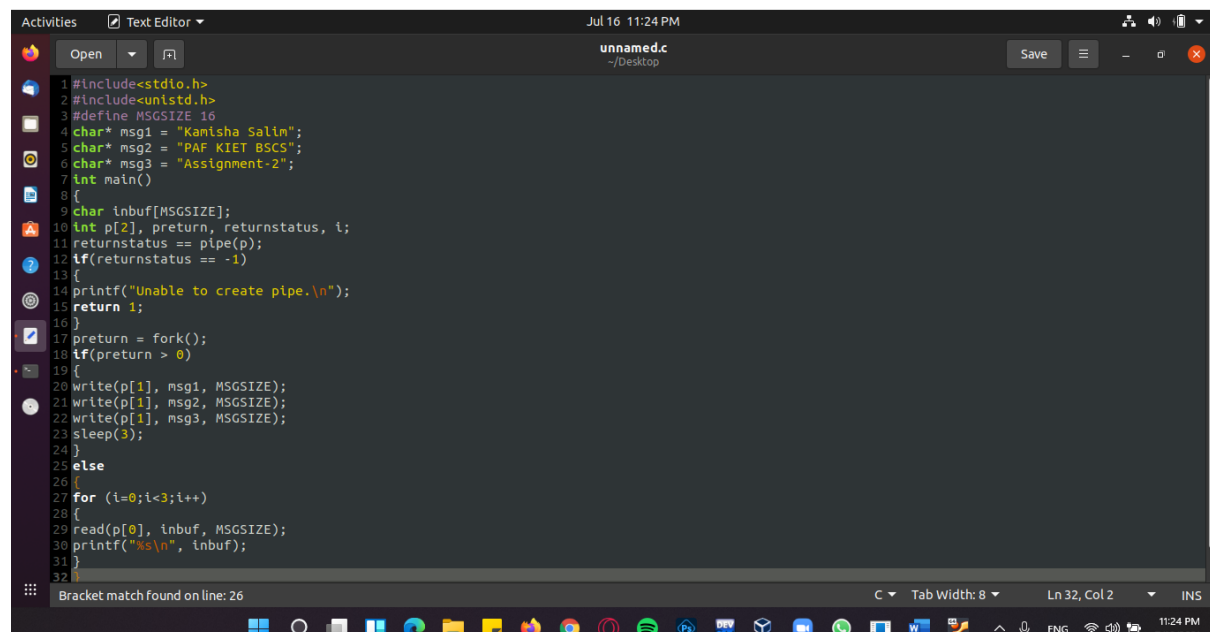
Thread4:
Enter a number to search in array: 55
Key Not Present.

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.

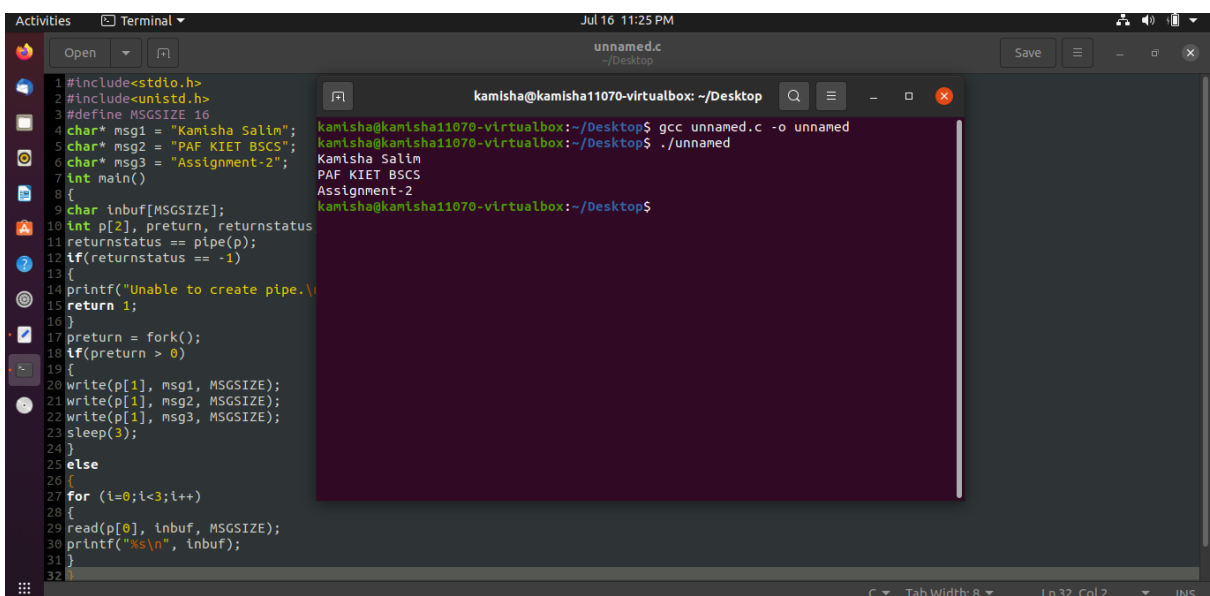
Example – 1: Unnamed pipes are only used for communication between a child and its parent process. It can either be a one-way or a two-way communication. So, for such communications, ordinary pipes or unnamed pipes are more suitable than named pipes.



```
1#include<stdio.h>
2#include<unistd.h>
3#define MSGSIZE 16
4char* msg1 = "Kamisha Salim";
5char* msg2 = "PAF KIET BSCS";
6char* msg3 = "Assignment-2";
7int main()
8{
9    char inbuf[MSGSIZE];
10    int p[2], preturn, returnstatus, i;
11    returnstatus = pipe(p);
12    if(returnstatus == -1)
13    {
14        printf("Unable to create pipe.\n");
15        return 1;
16    }
17    preturn = fork();
18    if(preturn > 0)
19    {
20        write(p[1], msg1, MSGSIZE);
21        write(p[1], msg2, MSGSIZE);
22        write(p[1], msg3, MSGSIZE);
23        sleep(3);
24    }
25    else
26    {
27        for (i=0;i<3;i++)
28        {
29            read(p[0], inbuf, MSGSIZE);
30            printf("%s\n", inbuf);
31        }
32    }
```

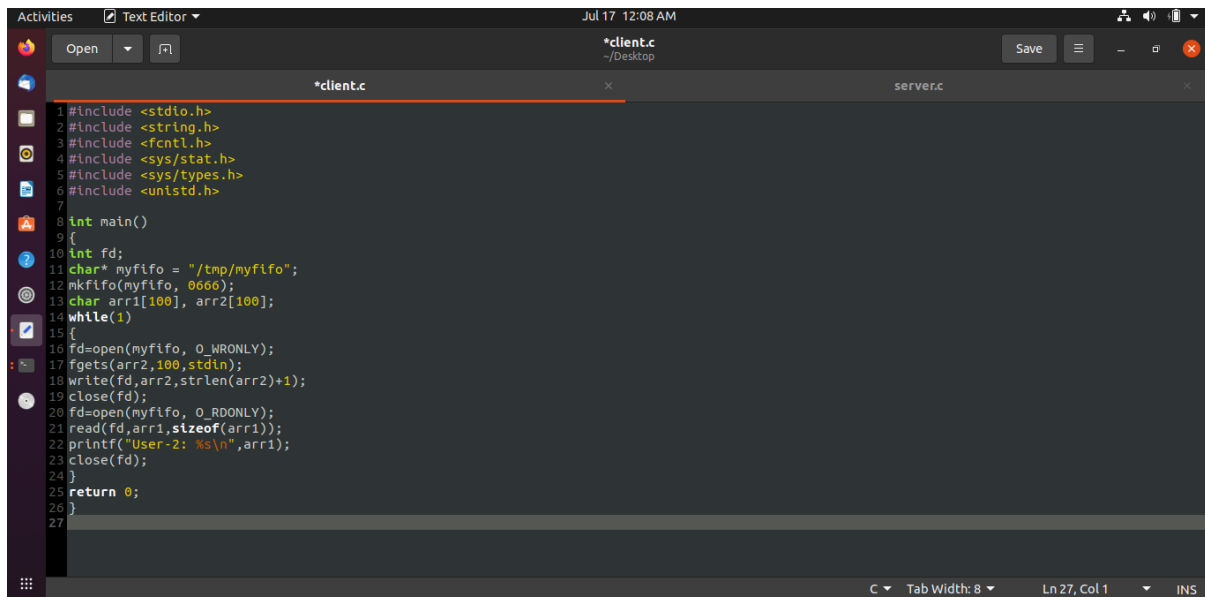
OUTPUT:

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



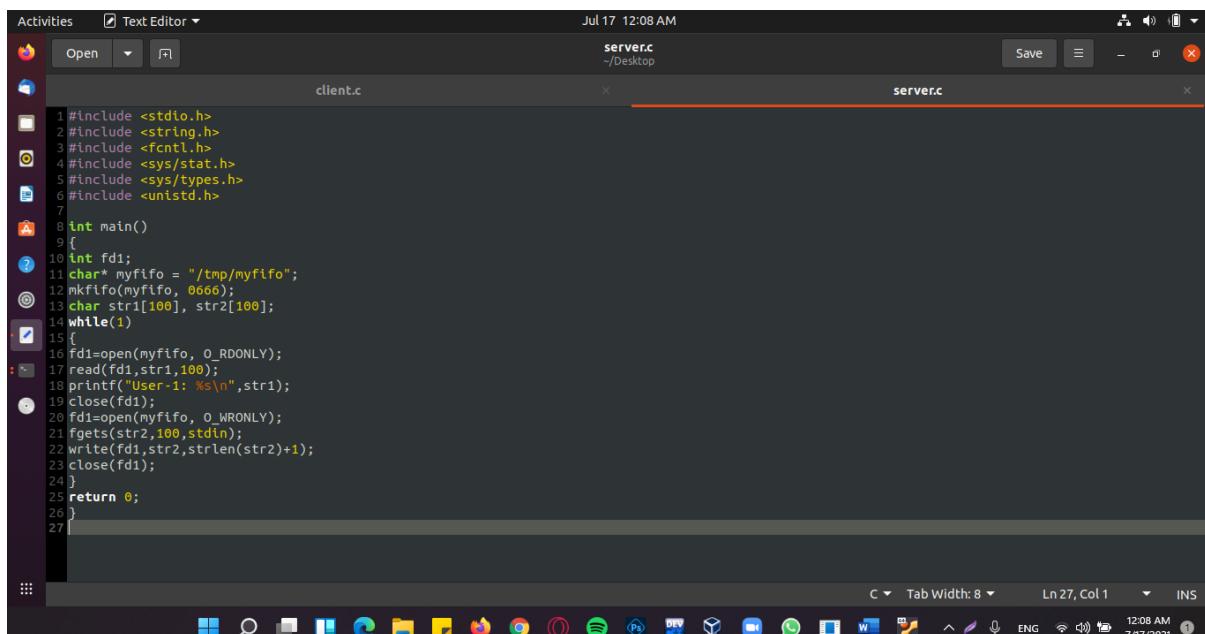
```
kamisha@kamisha11070-virtualbox: ~/Desktop
kamisha@kamisha11070-virtualbox:~/Desktop$ gcc unnamed.c -o unnamed
kamisha@kamisha11070-virtualbox:~/Desktop$ ./unnamed
Kamisha Salim
PAF KIET BSCS
Assignment-2
kamisha@kamisha11070-virtualbox:~/Desktop$
```


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.



The screenshot shows a text editor window titled "Text Editor" with a file named "client.c" open. The code is as follows:

```
1#include <stdio.h>
2#include <string.h>
3#include <fcntl.h>
4#include <sys/stat.h>
5#include <sys/types.h>
6#include <unistd.h>
7
8int main()
9{
10    int fd;
11    char* myfifo = "/tmp/myfifo";
12    mkfifo(myfifo, 0666);
13    char arr1[100], arr2[100];
14    while(1)
15    {
16        fd=open(myfifo, O_WRONLY);
17        fgets(arr2,100,stdin);
18        write(fd,arr2,strlen(arr2)+1);
19        close(fd);
20        fd=open(myfifo, O_RDONLY);
21        read(fd,arr1,sizeof(arr1));
22        printf("User-2: %s\n",arr1);
23        close(fd);
24    }
25    return 0;
26}
```

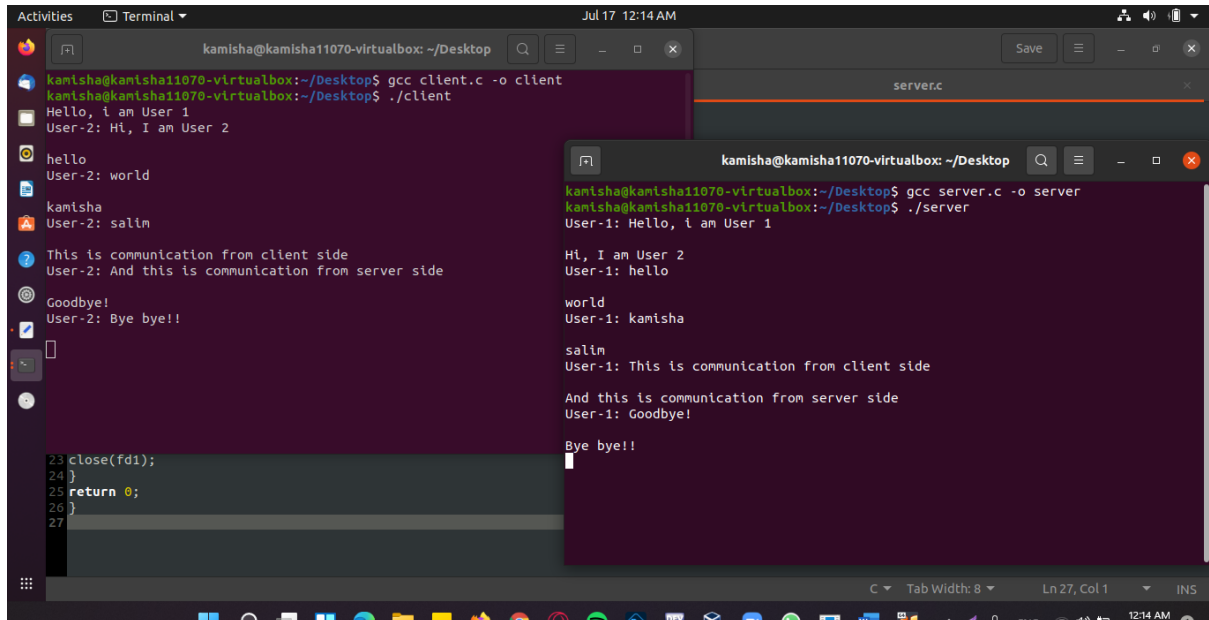


The screenshot shows a text editor window titled "Text Editor" with a file named "server.c" open. The code is as follows:

```
1#include <stdio.h>
2#include <string.h>
3#include <fcntl.h>
4#include <sys/stat.h>
5#include <sys/types.h>
6#include <unistd.h>
7
8int main()
9{
10    int fd1;
11    char* myfifo = "/tmp/myfifo";
12    mkfifo(myfifo, 0666);
13    char str1[100], str2[100];
14    while(1)
15    {
16        fd1=open(myfifo, O_RDONLY);
17        read(fd1,str1,100);
18        printf("User-1: %s\n",str1);
19        close(fd1);
20        fd1=open(myfifo, O_WRONLY);
21        fgets(str2,100,stdin);
22        write(fd1,str2,strlen(str2)+1);
23        close(fd1);
24    }
25    return 0;
26}
```

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
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: salin

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

Goodbye!
User-2: Bye bye!!

23 close(fd1);
24 }
25 return 0;
26 }
27

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

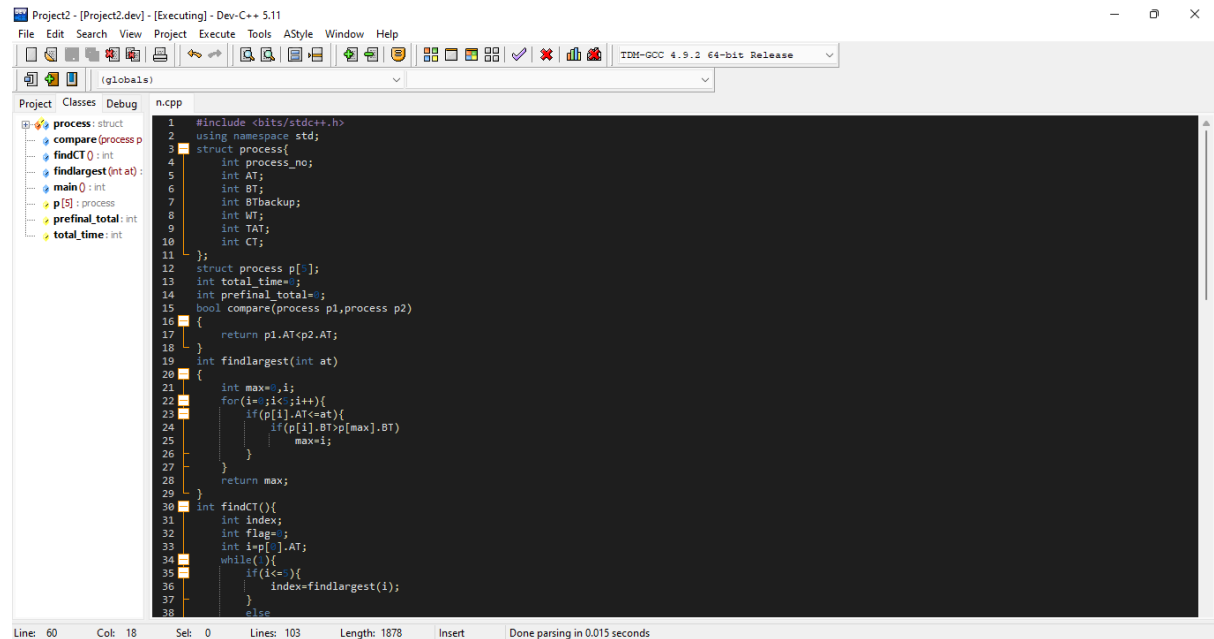
salin
User-1: This is communication from client side

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

Bye bye!!
```

QUESTION – 3

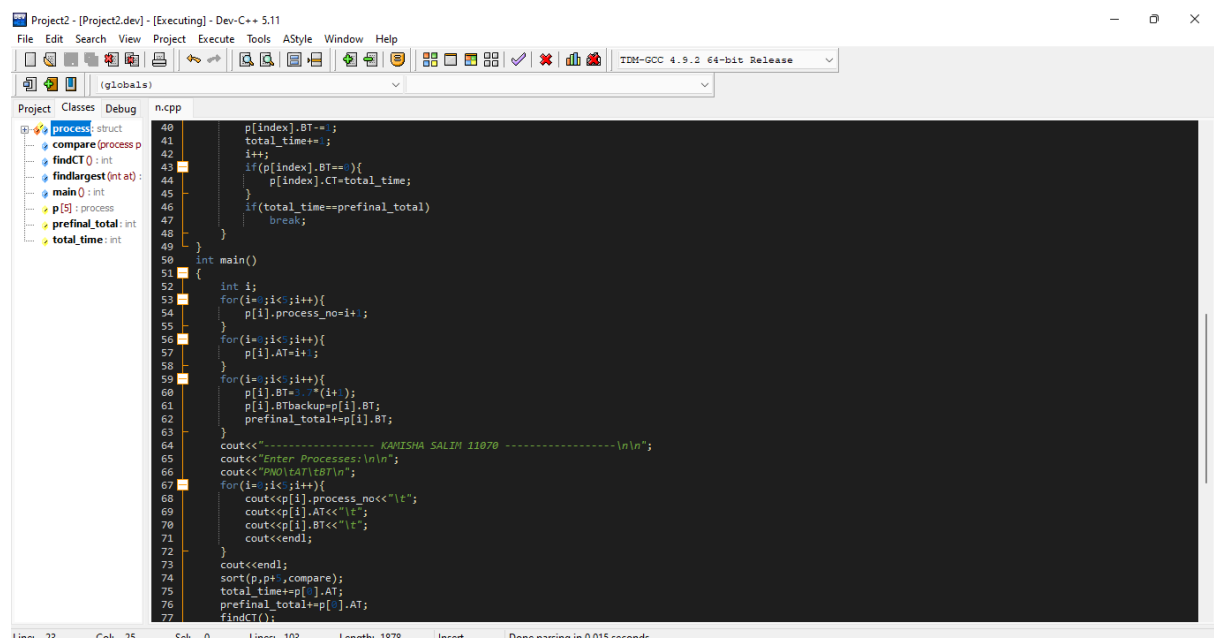
Implement The Longest Remaining Time First (LRTF) scheduling.



The screenshot shows a C++ IDE with the following code in `n.cpp`:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct process{
4     int process_no;
5     int AT;
6     int BT;
7     int BTbackup;
8     int WT;
9     int TAT;
10    int CT;
11 };
12 struct process p[10];
13 int total_time=0;
14 int prefinal_total=0;
15 bool compare(process p1,process p2)
16 {
17     return p1.AT<p2.AT;
18 }
19 int findlargest(int at)
20 {
21     int max=-1;
22     for(i=0;i<10;i++){
23         if(p[i].AT<=at){
24             if(p[i].BT>p[max].BT)
25                 max=i;
26         }
27     }
28     return max;
29 }
30 int findCT(){
31     int index;
32     int flag=0;
33     int i=p[0].AT;
34     while(flag){
35         if(i<=){
36             index=findlargest(i);
37         }
38         else
```

The status bar at the bottom indicates: Line: 60, Col: 18, Sek: 0, Lines: 103, Length: 1878, Insert, Done parsing in 0.015 seconds.



The screenshot shows the continuation of the C++ code in `n.cpp`:

```
40 p[index].BT-=1;
41 total_time+=1;
42 i++;
43 if(p[index].BT==0){
44     p[index].CT=total_time;
45 }
46 if(total_time==prefinal_total)
47     break;
48 }
49 }
50 int main()
51 {
52     int i;
53     for(i=0;i<10;i++){
54         p[i].process_no=i+1;
55     }
56     for(i=0;i<10;i++){
57         p[i].AT=i+1;
58     }
59     for(i=0;i<10;i++){
60         p[i].BT=1.7*(i+1);
61         p[i].BTbackup=p[i].BT;
62         prefinal_total+=p[i].BT;
63     }
64     cout<<"----- KAVISHA SALIM 11070 ----- \n\n";
65     cout<<"Enter Processes:\n\n";
66     cout<<"PNO\tAT\tBT\n";
67     for(i=0;i<10;i++){
68         cout<<p[i].process_no<<"\t";
69         cout<<p[i].AT<<"\t";
70         cout<<p[i].BT<<"\t";
71         cout<<endl;
72     }
73     cout<<endl;
74     sort(p,p+10,compare);
75     total_time=p[0].AT;
76     prefinal_total+=p[0].AT;
77     findCT();
```

The status bar at the bottom indicates: Line: 72, Col: 36, Sek: 0, Lines: 103, Length: 1070, Insert, Done parsing in 0.015 seconds.

```

Project2 - [Project2.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(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

67 for(i=0;i<5;i++){
68     cout<<p[i].process_no<<"\t";
69     cout<<p[i].AT<<"\t";
70     cout<<p[i].BT<<"\t";
71     cout<<endl;
72 }
73 cout<<endl;
74 sort(p,p+5,compare);
75 total_time+=p[0].AT;
76 prefinal_total+=p[0].AT;
77 findCT();
78 int totalWT=0;
79 int totalTAT=0;
80 for(i=0;i<5;i++){
81     p[i].TAT+=p[i].CT-p[i].AT;
82     p[i].WT=p[i].TAT-p[i].BTbackup;
83     totalWT+=p[i].WT;
84     totalTAT+=p[i].TAT;
85 }
86
87 cout<<"\nAfter Execution:\n\n";
88 cout<<"PNO\tAT\tBT\tCT\tTAT\tWT\n";
89
90 for(i=0;i<5;i++){
91     cout<<p[i].process_no<<"\t";
92     cout<<p[i].AT<<"\t";
93     cout<<p[i].BTbackup<<"\t";
94     cout<<p[i].CT<<"\t";
95     cout<<p[i].TAT<<"\t";
96     cout<<p[i].WT<<"\t";
97     cout<<endl;
98 }
99 cout<<endl;
100 cout<<"Average Turnaround Time : "<<totalTAT/<<endl;
101 cout<<"Average Waiting Time : "<<totalWT/<<endl;
102 return 0;
103 }

```

Line: 23 Col: 25 Sel: 0 Lines: 103 Length: 1878 Insert Done parsing in 0.015 seconds

OUTPUT:

```

C:\Users\ucom\Documents\Project2.exe
----- 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 Turnaround Time : 49
Average Waiting Time : 38

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

```