西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab | Report

Report Subject: OS Experiment - Lab 6

Student ID	**:**	2019380141
Student Name	**:**	ABID ALI
Experiment Name	**:**	CPU Scheduling and deadlock

Objective:

- You will build a simulator of a simple operating system to learn more about operating systems in general and process schedulers in specific.
- You will write a banker algorithm program in a high-level language, and use the banker algorithm to simulate the allocation of resources and security checks.

Equipment:

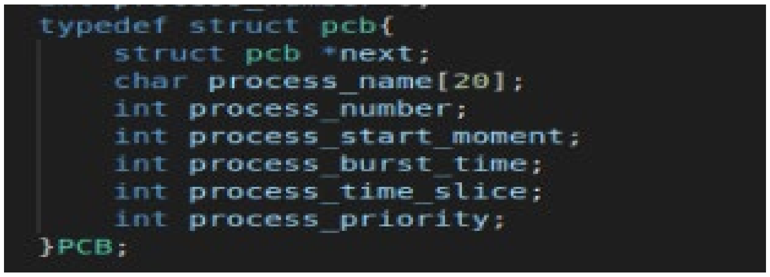
VMware with Ubuntu Linux

Methodology:

2.1 Experiment 1: CPU Scheduling Simulator

- First-Come/First-Served (FCFS): The process ready queue is arranged in descending order of arrival time. According to processes' arrival time, the processes can obtain their CPU burst with none-preemption.
- Shortest Job First (SJF): according to processes' arrival time and CPU burst, the shortest CPU burst will get their CPU burst.
- Priority scheduling: The process ready queue is arranged in descending order of priority number and arrival time, and the first process of the chain is put in first.
- Round Robin (RR): The process ready queue is arranged in descending order of arrival time. The program that implements processor scheduling by time slice rotation.

```
typedef struct pcb{

        struct pcb *next;              //The next process control block pointer
        char process_name[20];              //Process name

        int process_number;                 //Process number

        int process_start_moment;               //Process start time

        int process_burst_time;               //Required running time

        int process_time_slice;               //Time slice

        int process_priority;                //Priority number

}PCB;
```



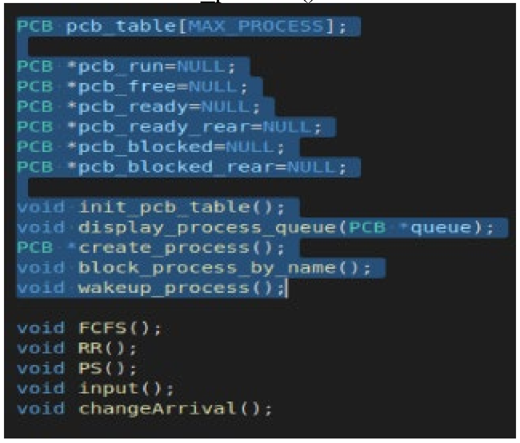
```
typedef struct pcb{
    struct pcb *next;
    char process_name[20];
    int process_number;
    int process_start_moment;
    int process_burst_time;
    int process_time_slice;
    int process_priority;
}PCB;
```

- Initialize the process control block:

  void init_pcb_table()

- Display process queue:
  void display_process_queue(PCB *queue)

- Create process:
  PCB *create_process()



```c
PCB pcb_table[MAX_PROCESS];

PCB *pcb_run=NULL;
PCB *pcb_free=NULL;
PCB *pcb_ready=NULL;
PCB *pcb_ready_rear=NULL;
PCB *pcb_blocked=NULL;
PCB *pcb_blocked_rear=NULL;

void init_pcb_table();
void display_process_queue(PCB *queue);
PCB *create_process();
void block_process_by_name();
void wakeup_process();

void FCFS();
void RR();
void PS();
void input();
void changeArrival();
```

We can see in the picture that,I followed the instruction of teacher.

- First come first serve process scheduling algorithm:

```c
void FCFS()
{

    int n,bt[5],wt[5],tat[5],i,j;
    float avwt=0,avtat=0;
    printw("Enter total number of processes(maximum 5):");
    scanw("%d",&n);
    printw("Arrival Time is considered 0\n");
    printw("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printw("P[%d]:",i);
        scanw("%d",&bt[i]);
    }

    wt[0]=0;    //waiting time for first process is 0

    //calculating waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }

    printw("|    Process  |    Burst  Time  |  Waiting  Time | Turnaround Time |\n");

    //calculating turnaround time
    for(i=0;i<n;i++)
    {
       tat[i]=bt[i]+wt[i];
       avwt+=wt[i];
       avtat+=tat[i];
       printw("\tP[%d]\t\t%d\t\t%d\t\t%d\n",i,bt[i],wt[i],tat[i]);
    }

    avwt/=i;
    avtat/=i;
    printw("Average Waiting Time:%f\n",avwt);
    printw("Average Turnaround Time:%f\n",avtat);
         refresh();
}
```

- ## Priority scheduling:

```
void PS()
{
         int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printw("Enter Total Number of Process:");
    scanw("%d",&n);

    printw("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
       printw("\nP[%d]\n",i+1);
       printw("Burst Time:");
       scanw("%d",&bt[i]);
       printw("Priority:");
       scanw("%d",&pr[i]);
       p[i]=i+1;           //contains process number
    }

    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
       pos=i;
       for(j=i+1;j<n;j++)
       {
          if(pr[j]<pr[pos])
             pos=j;
       }

       temp=pr[i];
       pr[i]=pr[pos];
       pr[pos]=temp;
```

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

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

        wt[0]=0; //waiting time for first process is zero

        //calculate waiting time
        for(i=1;i<n;i++)
        {
            wt[i]=0;
            for(j=0;j<i;j++)
                wt[i]+=bt[j];

            total+=wt[i];
        }

        avg_wt=total/n;      //average waiting time
        total=0;

        printw("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
        for(i=0;i<n;i++)
        {
            tat[i]=bt[i]+wt[i];     //calculate turnaround time
            total+=tat[i];
            printw("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
        }

        avg_tat=total/n;     //average turnaround time
        printw("\n\nAverage Waiting Time=%d",avg_wt);
        printw("\nAverage Turnaround Time=%d\n",avg_tat);
}
```

- ## Round Robin (RR) scheduling algorithm:

```
void RR()
{
 // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printw(" Total number of process in the system: ");
   scanw("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printw("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printw(" Arrival time is: \t");  // Accept arrival time
scanw("%d", &at[i]);
printw(" \nBurst time is: \t"); // Accept the Burst time
scanw("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printw("Enter the Time Quantum for the process: \t");
scanw("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printw("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
   {
      y--; //decrement the process no.
      printw("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
      wt = wt+sum-at[i]-bt[i];
      tat = tat+sum-at[i];
      count =0;
   }
   if(i==NOP-1)
   {
      i=0;
   }
   else if(at[i+1]<=sum)
   {
      i++;
   }
   else
   {
      i=0;
   }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printw("\n Average Turn Around Time: \t%f", avg_wt);
printw("\n Average Waiting Time: \t%f\n", avg_tat);
```

```
        refresh();
    }
```

Requirements:

1) You can start to implement those scheduling algorithms with non-preemption..

FCFS Scheduling, Priority scheduling these are two non-preemption scheduling.

Note: **Priority scheduling** can be non-preemption scheduling and preemption scheduling.

2) Detailed analysis of the scheduling algorithm, based on a careful analysis, completely understand the role of the main data structures and processes described and shown a flowchart of the main module of the main data structure is given.

FCFS Scheduling:

- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always **non-preemptive** in nature.

Round-Robin Scheduling:

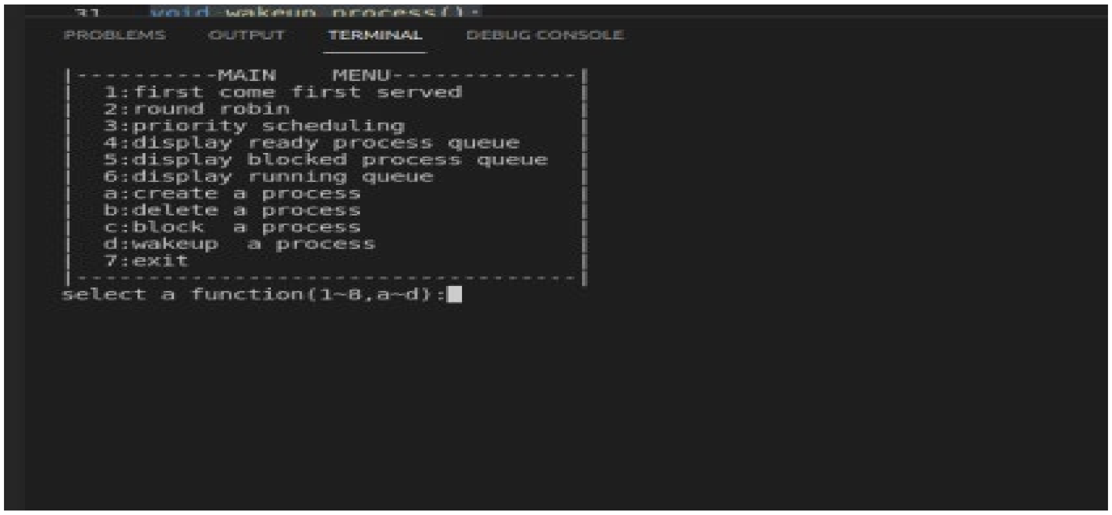
- Round robin is a **pre-emptive algorithm**
- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.

Priority scheduling:

- In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.
- The difference between **preemptive priority scheduling** and **non preemptive priority scheduling** is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

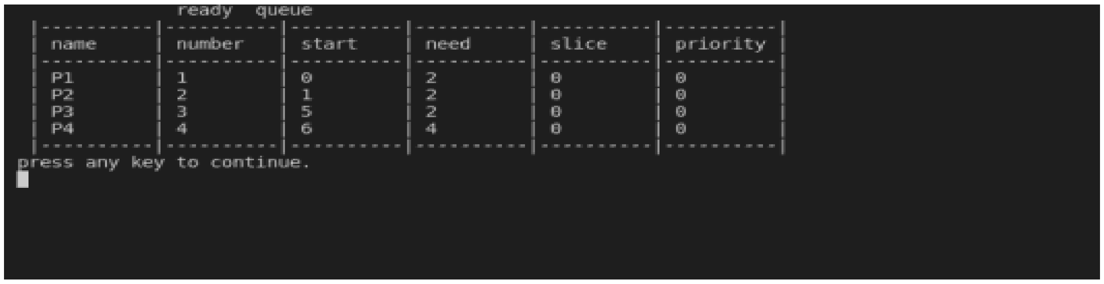
Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done

3) Follow the prompts, write the function complete, the program became a run.



```
void wakeup_process();
```

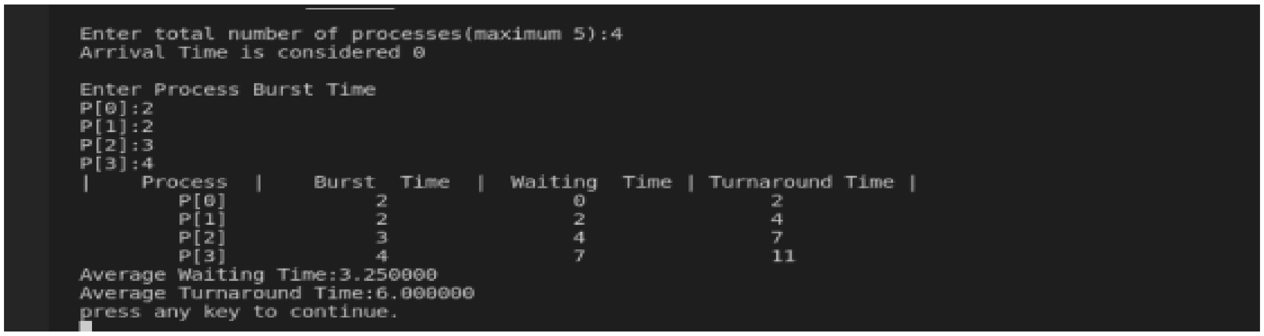
```
|----------MAIN     MENU-------------|
|   1:first come first served        |
|   2:round robin                    |
|   3:priority scheduling            |
|   4:display ready process queue    |
|   5:display blocked process queue  |
|   6:display running queue          |
|   a:create a process               |
|   b:delete a process               |
|   c:block  a process               |
|   d:wakeup  a process              |
|   7:exit                           |
|------------------------------------|
select a function(1~8,a~d):
```



```
               ready  queue
|----------|----------|----------|----------|----------|----------|
| name     | number   | start    | need     | slice    | priority |
|----------|----------|----------|----------|----------|----------|
| P1       | 1        | 0        | 2        | 0        | 0        |
| P2       | 2        | 1        | 2        | 0        | 0        |
| P3       | 3        | 5        | 2        | 0        | 0        |
| P4       | 4        | 6        | 4        | 0        | 0        |
|----------|----------|----------|----------|----------|----------|
press any key to continue.
```

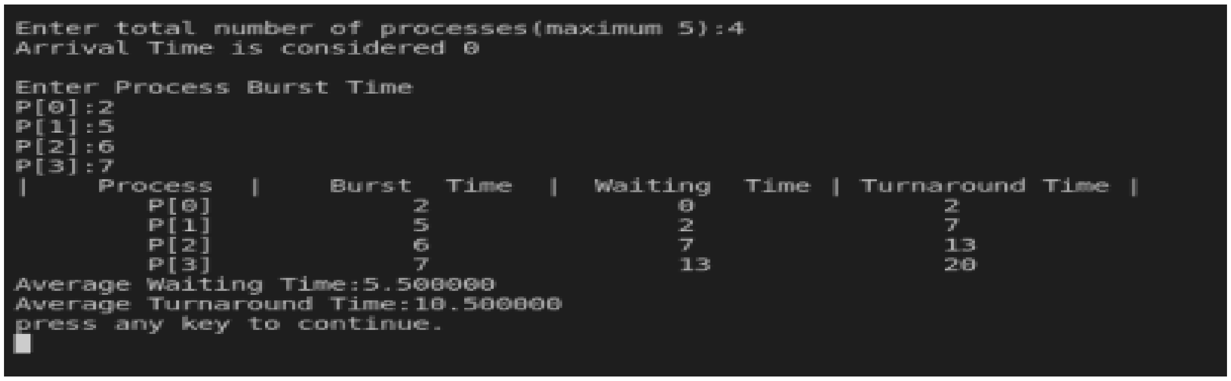
4) Repeatedly run the program, the program execution result of the observation, verify the correctness of the analysis and the results of a given final operation performed by the calculated result and the weighted turnaround time turnaround time.

First come first serve process scheduling



```
Enter total number of processes(maximum 5):4
Arrival Time is considered 0

Enter Process Burst Time
P[0]:2
P[1]:2
P[2]:3
P[3]:4
|    Process  |    Burst  Time  |  Waiting  Time | Turnaround Time |
        P[0]            2                0               2
        P[1]            2                2               4
        P[2]            3                4               7
        P[3]            4                7               11
Average Waiting Time:3.250000
Average Turnaround Time:6.000000
press any key to continue.
```

This is a an exercise,I did from online and the answers are correct.



```
Enter total number of processes(maximum 5):4
Arrival Time is considered 0

Enter Process Burst Time
P[0]:2
P[1]:5
P[2]:6
P[3]:7
|    Process   |     Burst  Time   |  Waiting  Time | Turnaround Time |
        P[0]            2                0                2
        P[1]            5                2                7
        P[2]            6                7                13
        P[3]            7                13               20
Average Waiting Time:5.500000
Average Turnaround Time:10.500000
press any key to continue.
```

Same result like the teacher.

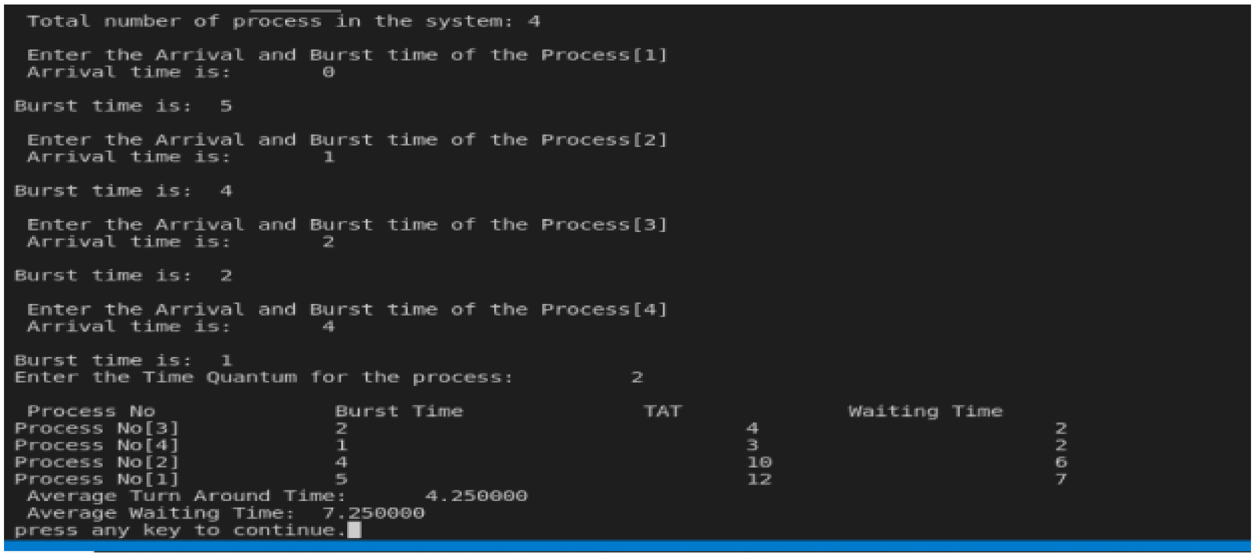
Output:

```
PROCESS      BURST TIME   WAITING TIME        TURNAROUND TIME
------------------------------------------------------------------------------------------------
   P0              2              0                      2
   P1              5              2                      7
   P2              6              7                      13
   P3              7              13                     20
```

Average Waiting Time = 5.500000

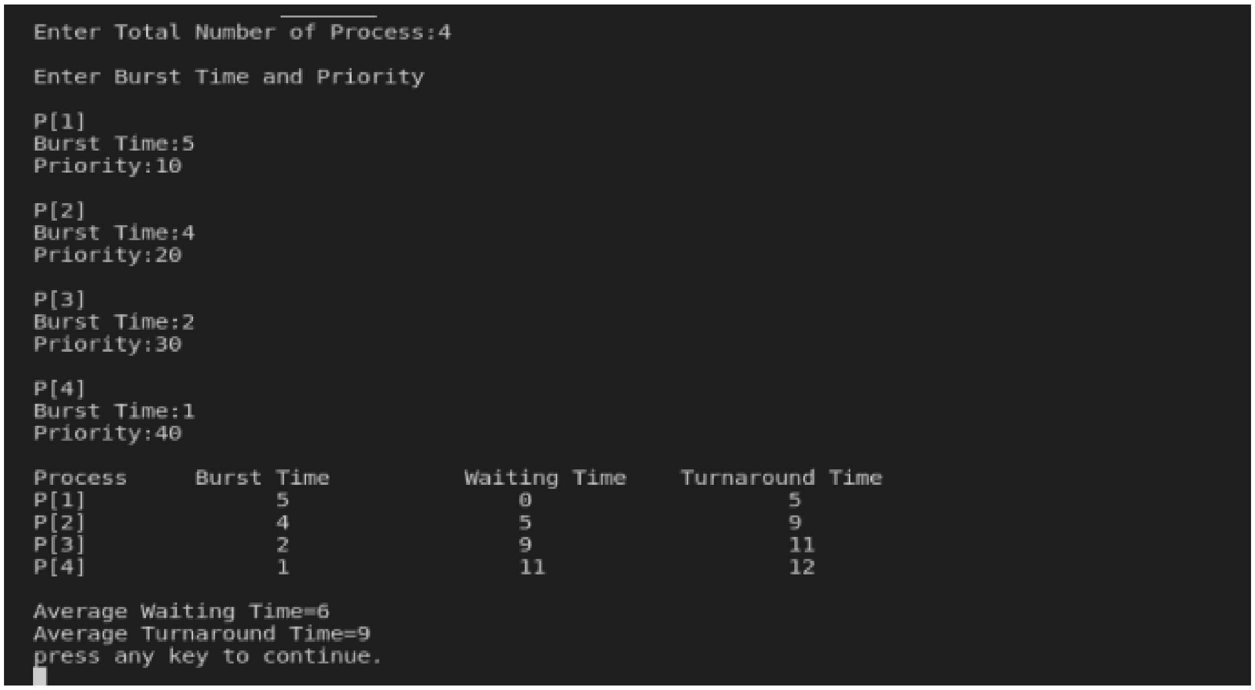
Average Turnaround Time = 10.500000

Round-Robin Scheduling:



```
 Total number of process in the system: 4

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0

Burst time is:  5

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       1

Burst time is:  4

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       2

Burst time is:  2

 Enter the Arrival and Burst time of the Process[4]
 Arrival time is:       4

Burst time is:  1
Enter the Time Quantum for the process:         2

 Process No              Burst Time              TAT             Waiting Time
Process No[3]            2                               4                       2
Process No[4]            1                               3                       2
Process No[2]            4                               10                      6
Process No[1]            5                               12                      7
 Average Turn Around Time:      4.250000
 Average Waiting Time:  7.250000
press any key to continue.
```

Priority scheduling:



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:5
Priority:10

P[2]
Burst Time:4
Priority:20

P[3]
Burst Time:2
Priority:30

P[4]
Burst Time:1
Priority:40

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              5                 0                    5
P[2]              4                 5                    9
P[3]              2                 9                    11
P[4]              1                 11                   12

Average Waiting Time=6
Average Turnaround Time=9
press any key to continue.
```

We can see that, Average Waiting Time and Average Turnaround Time are almost very close to each other.

There is no universal "best" scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms.

First Come First Serve

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

From the program we can see that, the program had the fastest time of completitition.

Characteristics of FCFS method:

- It offers non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis

Round-Robin Scheduling

Round robin is the simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each program gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

We can see the implement of this algorithm and notice that, time was less used for the following exercise which is pre-emptive algorithm schedule.

Characteristics of Round-Robin Scheduling

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

Priority Based Scheduling

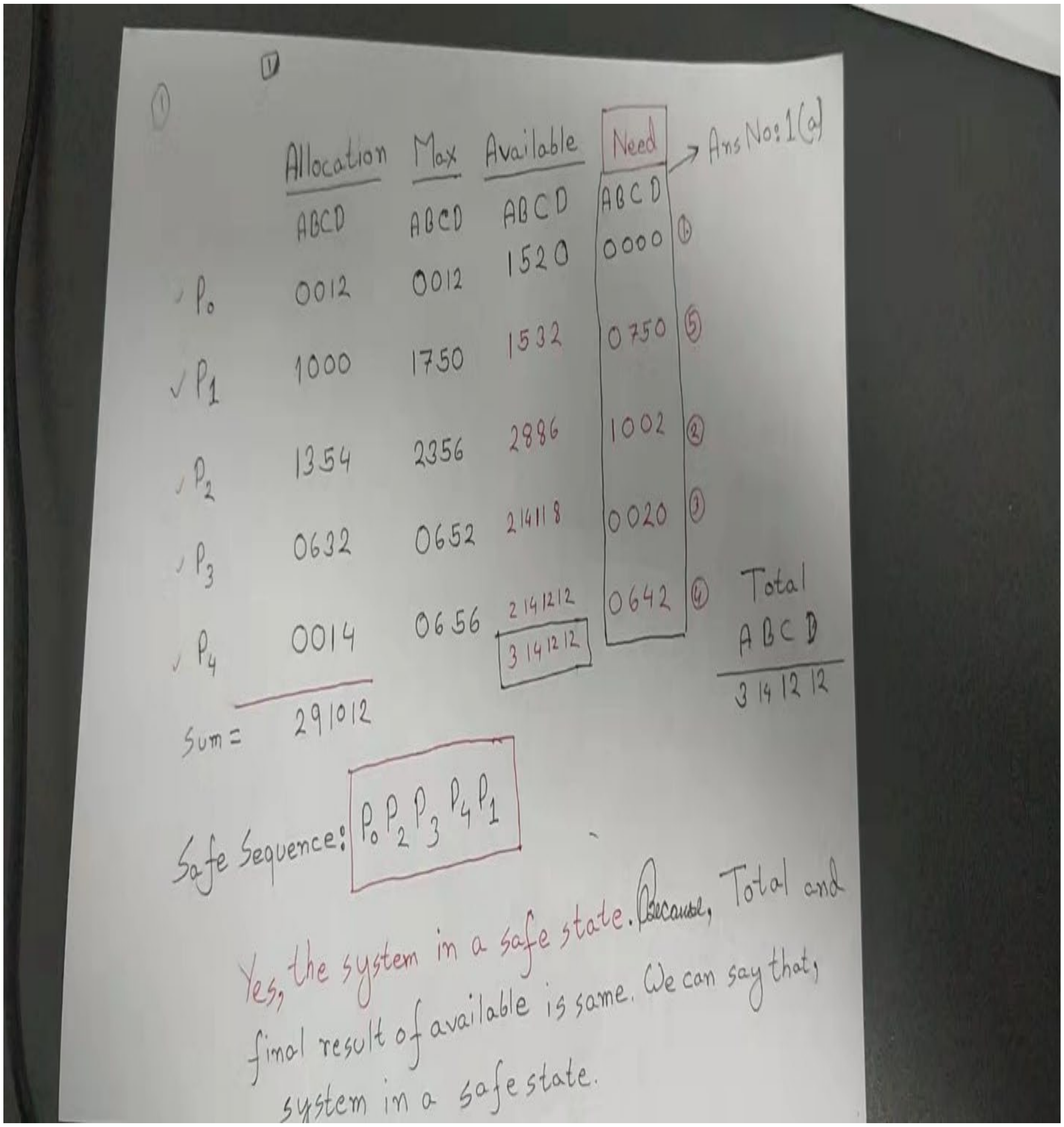
Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

We can see in the program that,for that program that took the most time among all the algorithm we used.

I came to the conclusion,by extensive running the code with different values.

2.2 Experiment 2: Safety Algorithm Implementation



1 1

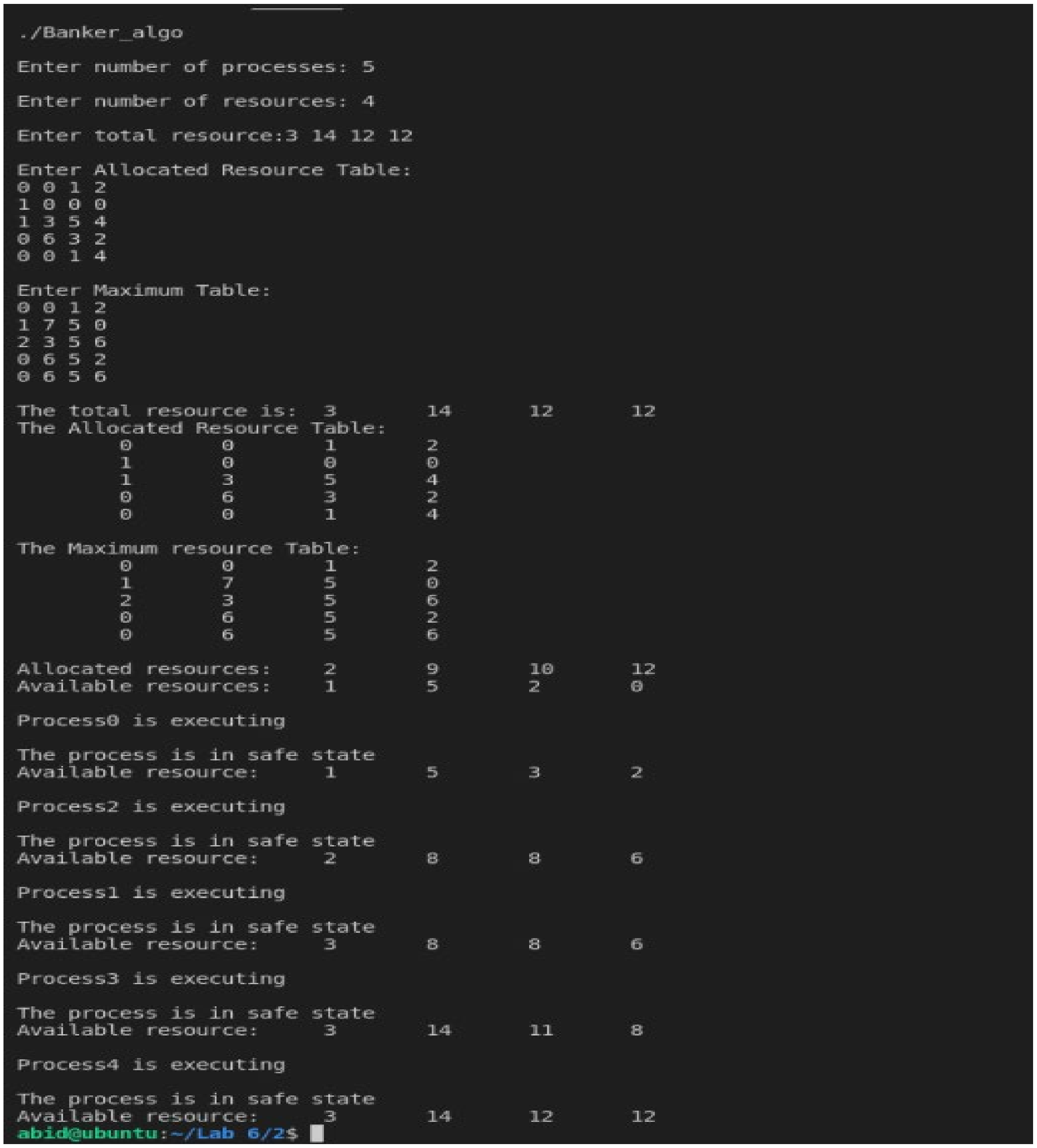
	Allocation	Max	Available	Need → Ans No: 1(a)	
	ABCD	ABCD	ABCD	ABCD	
			1520	0000	1
✓ P_0	0012	0012			
			1532	0750	5
✓ P_1	1000	1750			
			2886	1002	2
✓ P_2	1354	2356			
			2 14 11 8	0020	3
✓ P_3	0632	0652			
			2 14 12 12	0642	4
✓ P_4	0014	0656	$\boxed{3\ 14\ 12\ 12}$		
Sum =	2 9 10 12				

Total
ABCD
3 14 12 12

Safe Sequence: $\boxed{P_0 P_2 P_3 P_4 P_1}$

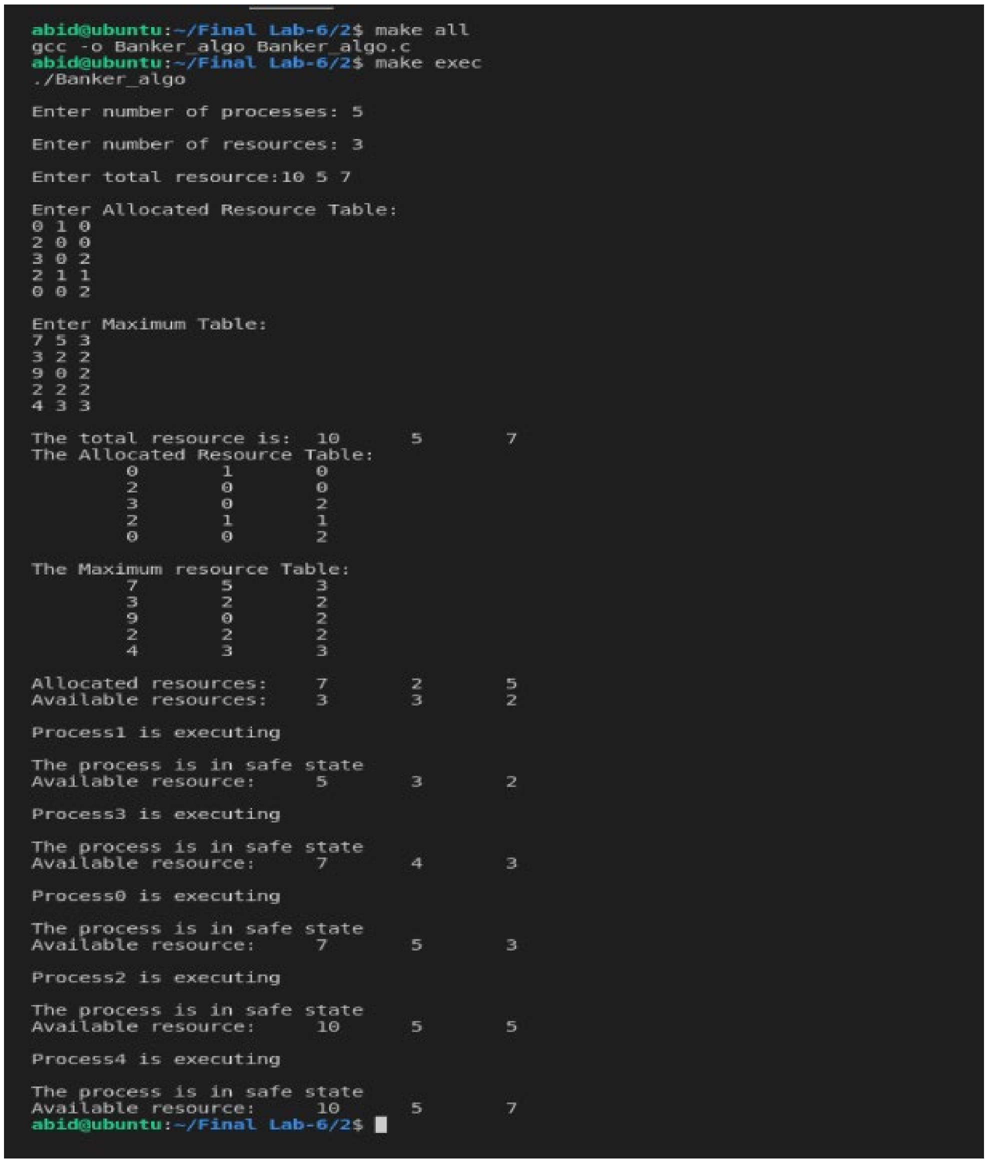
Yes, the system in a safe state. Because, Total and final result of available is same. We can say that, system in a safe state.

This is the homework answer and I tried to run this homework .The program gave the exact result and gave a different safe sequence.



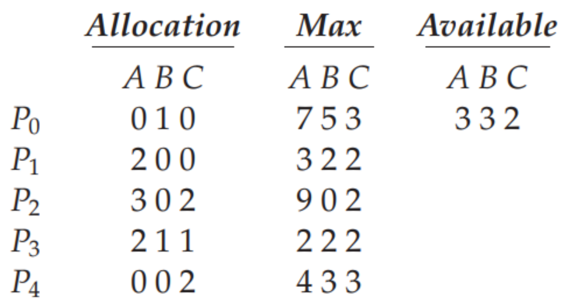
```
./Banker_algo

Enter number of processes: 5

Enter number of resources: 4

Enter total resource:3 14 12 12

Enter Allocated Resource Table:
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4

Enter Maximum Table:
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 6 5 6

The total resource is:  3       14      12      12
The Allocated Resource Table:
        0       0       1       2
        1       0       0       0
        1       3       5       4
        0       6       3       2
        0       0       1       4

The Maximum resource Table:
        0       0       1       2
        1       7       5       0
        2       3       5       6
        0       6       5       2
        0       6       5       6

Allocated resources:    2       9       10      12
Available resources:    1       5       2       0

Process0 is executing

The process is in safe state
Available resource:     1       5       3       2

Process2 is executing

The process is in safe state
Available resource:     2       8       8       6

Process1 is executing

The process is in safe state
Available resource:     3       8       8       6

Process3 is executing

The process is in safe state
Available resource:     3       14      11      8

Process4 is executing

The process is in safe state
Available resource:     3       14      12      12
abid@ubuntu:~/Lab 6/2$
```

Safe Sequence :P0,P2,P1,P3,P4



```
abid@ubuntu:~/Final Lab-6/2$ make all
gcc -o Banker_algo Banker_algo.c
abid@ubuntu:~/Final Lab-6/2$ make exec
./Banker_algo

Enter number of processes: 5

Enter number of resources: 3

Enter total resource:10 5 7

Enter Allocated Resource Table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Maximum Table:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

The total resource is:  10      5       7
The Allocated Resource Table:
        0       1       0
        2       0       0
        3       0       2
        2       1       1
        0       0       2

The Maximum resource Table:
        7       5       3
        3       2       2
        9       0       2
        2       2       2
        4       3       3

Allocated resources:    7       2       5
Available resources:    3       3       2

Process1 is executing

The process is in safe state
Available resource:     5       3       2

Process3 is executing

The process is in safe state
Available resource:     7       4       3

Process0 is executing

The process is in safe state
Available resource:     7       5       3

Process2 is executing

The process is in safe state
Available resource:     10      5       5

Process4 is executing

The process is in safe state
Available resource:     10      5       7
abid@ubuntu:~/Final Lab-6/2$ 
```

Safe Sequence :P1,P3,P0,P2,P4



	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

2.3 Experiment 3: Bankers Algorithm (optional)

The Banker

The banker will consider requests from n customers for m resources types. The banker will keep track of the resources using the following data structures:

```
/* these may be any values >= 0 */
   #define NUMBER OF CUSTOMERS 5
   #define NUMBER OF RESOURCES 3
/* the available amount of each resource */
      int available[NUMBER OF RESOURCES];
/*the maximum demand of each customer */
      int maximum[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];
/* the amount currently allocated to each customer */
      int allocation[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];
/* the remaining need of each customer */
      int need[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];
```

The Customers

Create n customer threads that request and release resources from the bank. The customers will continually loop, requesting and then releasing random numbers of resources. The customers' requests for resources will be bounded by their respective values in the need array. The banker will grant a request if it satisfies the safety algorithm outlined in previous experiment. If a request does not leave the system in a safe state, the banker will deny it. Function prototypes for requesting and releasing resources are as follows:

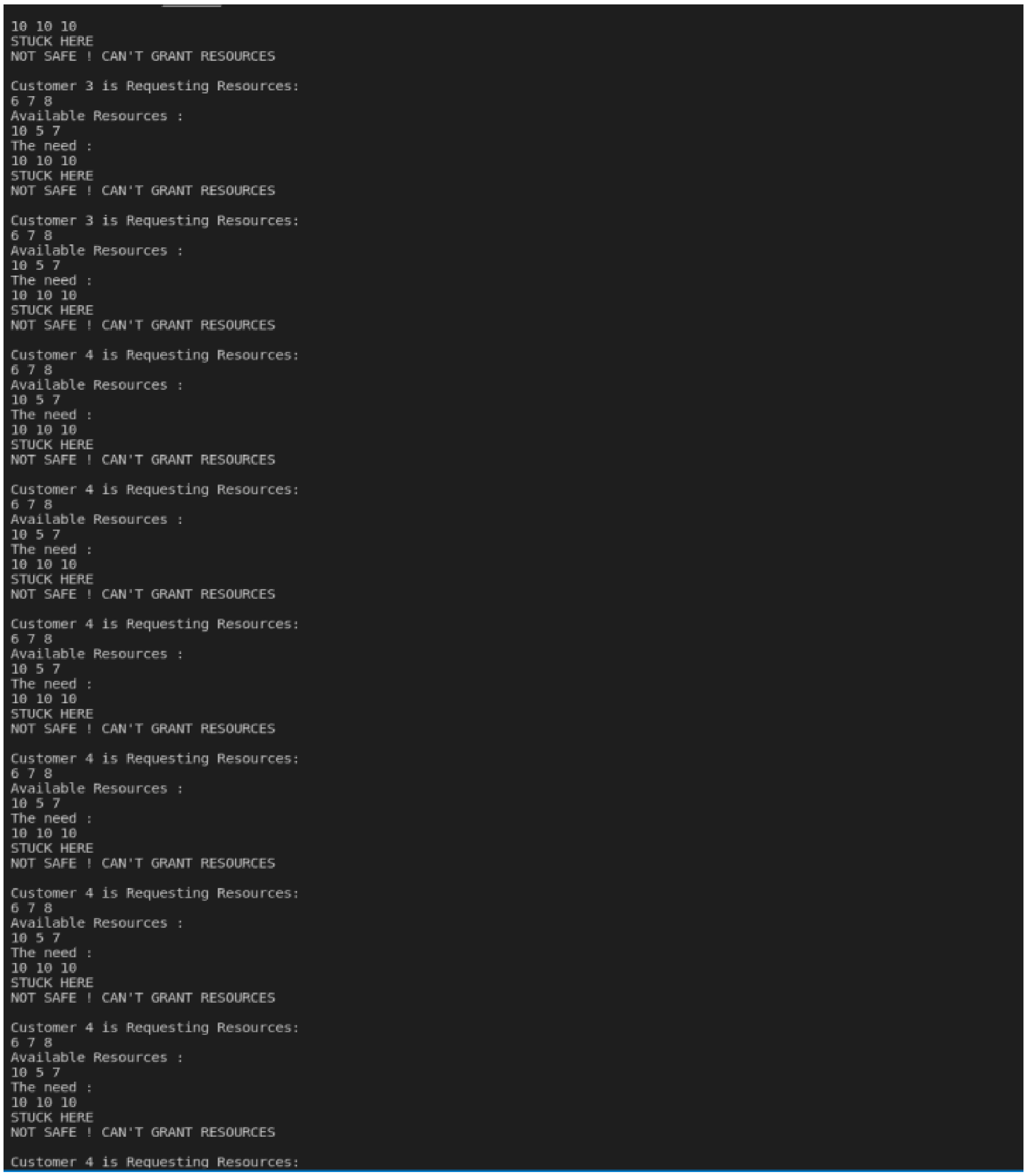
```
int request resources(int customer num, int request[]);

int release resources(int customer num, int release[]);
```

These two functions should return 0 if successful (the request has been granted) and –1 if unsuccessful. Multiple threads (customers) will concurrently. access shared data through these two functions. Therefore, access must be controlled through mutex locks to prevent race conditions. The use of Pthreads mutex locks are described in the project entitled “Producer–Consumer Problem”.

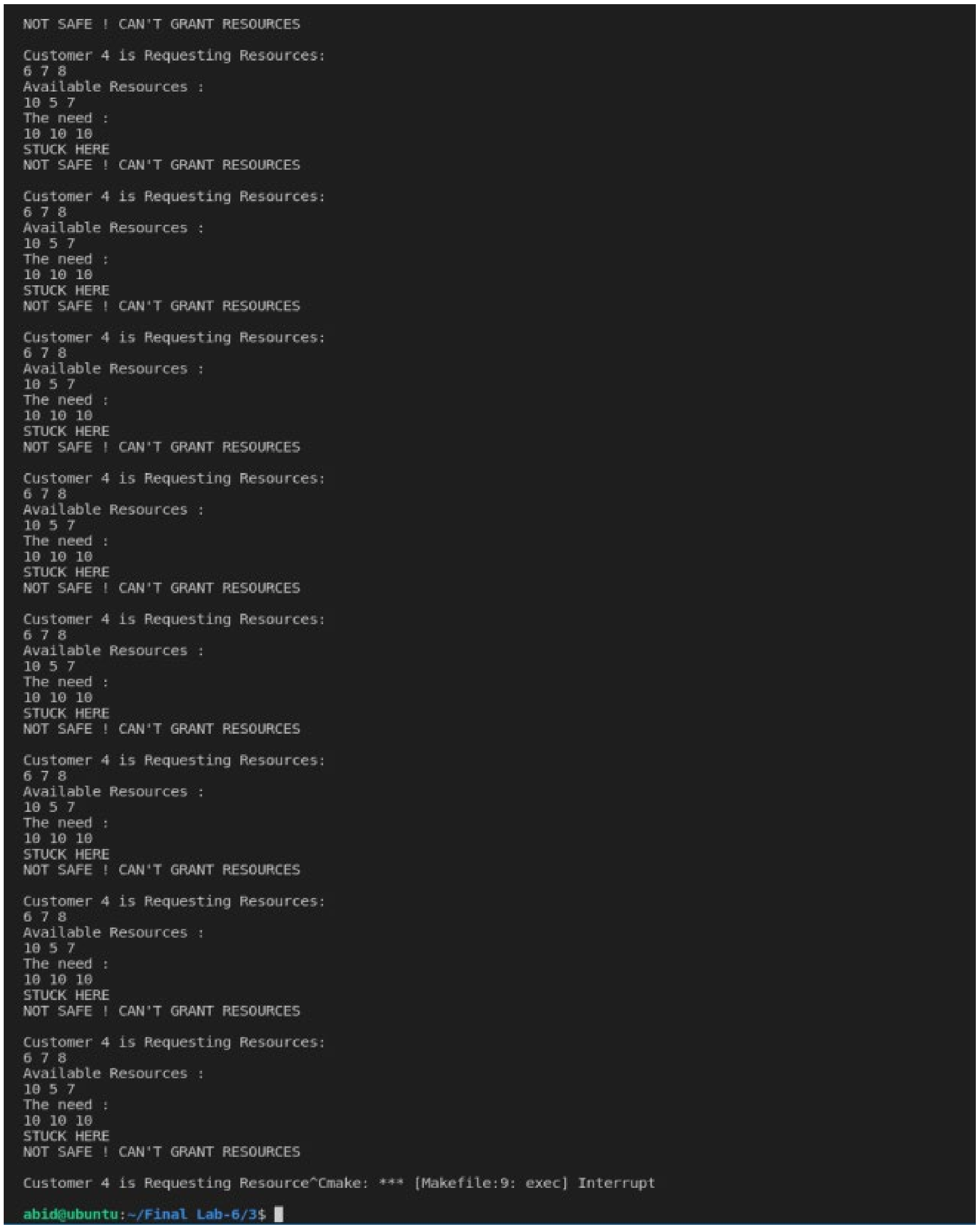
Implementation

You should invoke your program by passing the number of resources of each type on the command line. For example, if there were three resource types, with ten instances of the first type, five of the second type, and seven of the third type, you would invoke your program follows:

```
./banker 10 5 7
```



```
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 3 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 3 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
```



```
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resources:
6 7 8
Available Resources :
10 5 7
The need :
10 10 10
STUCK HERE
NOT SAFE ! CAN'T GRANT RESOURCES

Customer 4 is Requesting Resource^Cmake: *** [Makefile:9: exec] Interrupt

abid@ubuntu:~/Final Lab-6/3$
```

Solution:

To solve those problems I looked for information in internet. In order to understand some questions and procedure I also asked the teacher to help me understand them. And provided instructions helped to solve some of my errors during the experiment.

Problems:

The problem that I faced during was how to use different keywords kind of algorithms and implanting those algorithm and how to make a simulator.I was having problem with synchronization because the output was coming randomly and hard to control those also those linked list values.

Conclusion:

At the beginning, I was unfamiliar with those algorithm and how to make simulation. Gradually, reading lot of article and reading teachers ppt then I solved those problem one by one. In this experiment ,small helps and suggestions from the teacher was very helpful that saved my time.I enjoyed the practical and learned lot of interesting things.

Attachments:

1) ABID ALI_2019380141_OS(Lab 6).docx
2) ABID ALI_2019380141_OS(Lab 6).pdf
3)Code_ABID ALI_2019380141(Lab-6).zip

Reference:

1) https://www.guru99.com/cpu-scheduling-algorithms.html
2) https://www.javatpoint.com/os-preemptive-priority-scheduling
3) https://www.tutorialspoint.com/preemptive-and-non-preemptive-scheduling#:~:text=Non%2Dpreemptive%20Scheduling%20is%20a,processor%20switches%20to%20another%20process.
4) https://www.gatevidyalay.com/first-come-first-serve-cpu-scheduling/