REGISTRATION NUMBER – 16BIT0453
KRISHNA KUMAR MAHTO

EXPERIMENT-7

CODE:

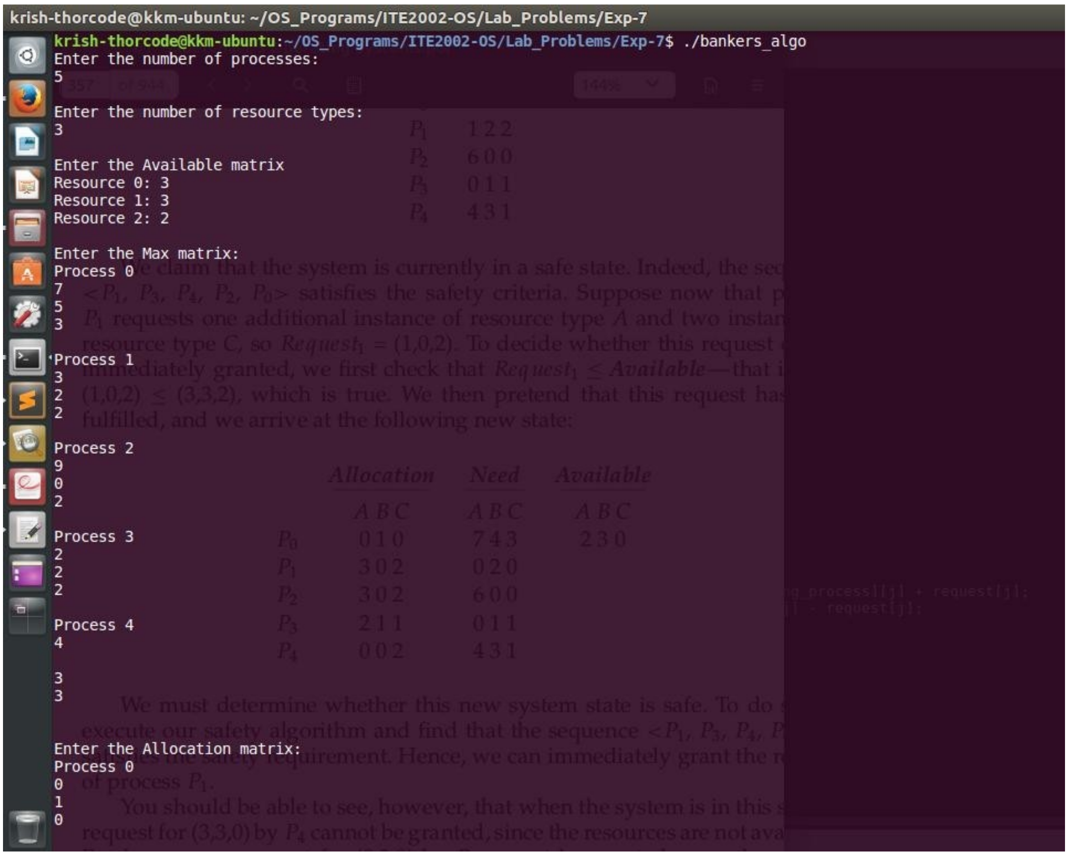
```
#include<stdio.h>
#include<stdlib.h>

#define MAX 5
#define false 0
#define true 1

//n = number of processes
//m = number of resource types
//compare = matrix comparator

int available[MAX]; //holds available number of instances for each resource i
int max[MAX][MAX]; //n*m matrix holds max demand for each process
int allocation[MAX][MAX]; //n*m matrix holds number of resources of each type allocated to each
process
int need[MAX][MAX]; //n*m matrix remaining resource need for each process
int n=0,m=0;
int request[MAX];
int safe_sequence[MAX];

void present_snapshot()
{
	int i,j;
	char ch;

	puts("\tAllocation Matrix\n");
	for(ch='A',i=0;i<m;i++,ch++)
		printf("\t%c",ch);
	puts("\n");

	for(i=0;i<n;i++)
	{
		printf("P%d",i);
		for(j=0;j<m;j++)
			printf("\t%d",allocation[i][j]);
		puts("");
	}
	puts("");

	puts("\tMax Matrix\n");
	for(ch='A',i=0;i<m;i++,ch++)
		printf("\t%c",ch);
	puts("");

	for(i=0;i<n;i++)
	{
		printf("P%d",i);
		for(j=0;j<m;j++)
			printf("\t%d",max[i][j]);
		puts("");
	}
	puts("");

	puts("\tAvailable Matrix\n");
	for(ch='A',i=0;i<m;i++,ch++)
		printf("\t%c",ch);
	puts("");

	for(j=0;j<m;j++)
		printf("\t%d",available[j]);
	puts("");


}

int compare(int need[],int work[])
{
	int j;
```

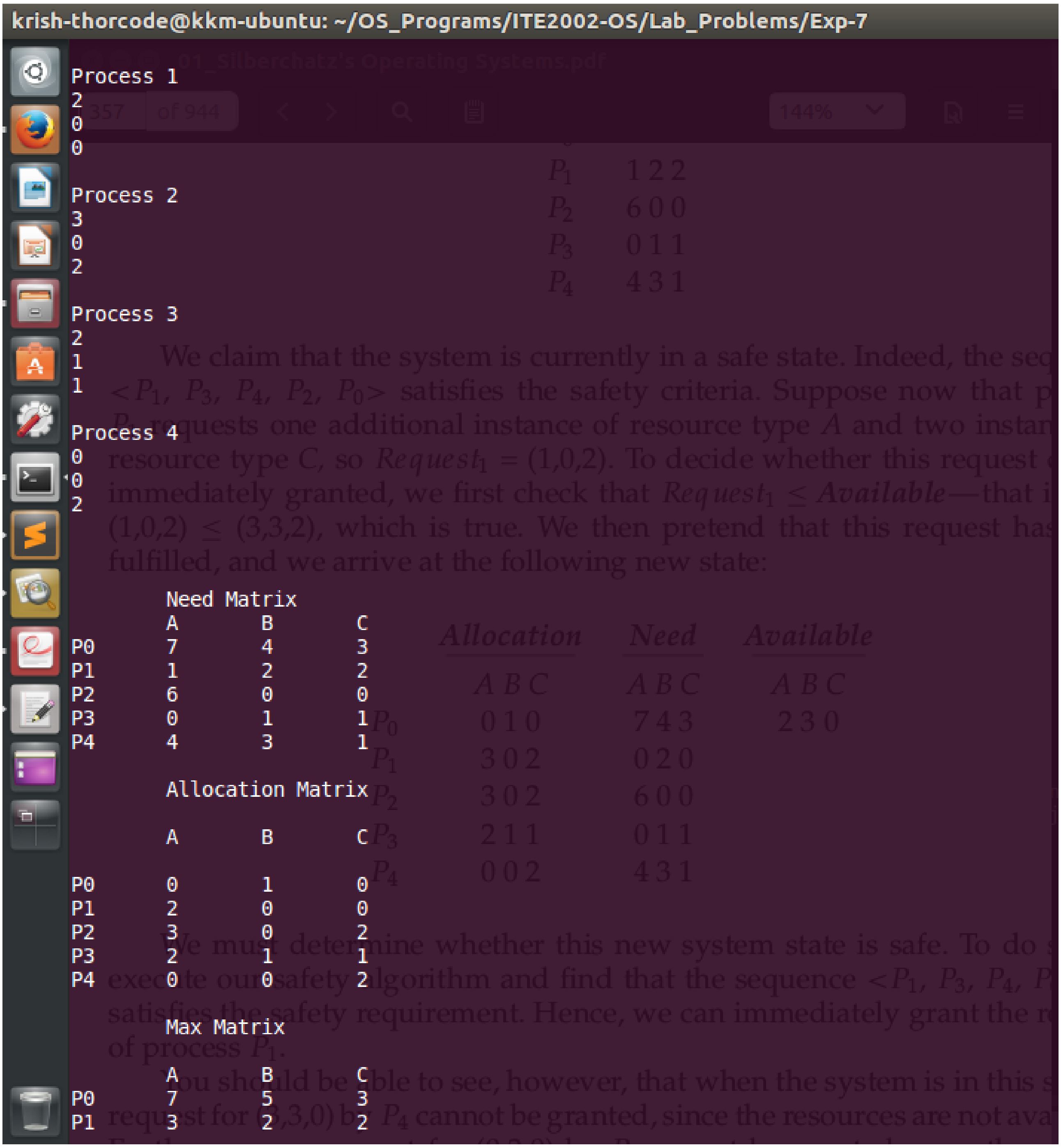
```
	for(j=0;j<m;j++)
	{
		if(need[j]<=work[j])
			continue;
		else
			return 0;
	}
	if(j==m-1)
		return 1;
}

int safety_algorithm()
{
	int finish[n],ss=0, *work, i, j, flag = false, safe = true, count,cycle;

	work = (int*)malloc(sizeof(int)*MAX);

	work = available;
	for(i=0;i<n;i++)
		finish[i]=false;
	for(count=0,cycle = 0;count<n && cycle < 2*n;cycle++) //if the loop runs twice through the
processes and still finish[i] = false for some process(es), then no safe sequeunce found
	{
		for(i=0;i<n;i++)
		{
			if((finish[i]==false) && compare(need[i],work))
			{
				count++;
				flag = true;
				for(j=0;j<m;j++)
					work[j] += allocation[i][j];
				finish[i] = true;
				safe_sequence[ss++] = i;
				continue;
			}

			else if(i<n-1)
				continue;
		}
	}


	for(i=0;i<n;i++)
		if(finish[i] == false)
		{
			safe = false;
			break;
		}

	if(ss == n-1)
	{
		puts("Safe sequence: ");
		printf("< ");
		for(i=0;i<n;i++)
		{
			printf("%d, ",safe_sequence[i]);
		}
		printf(">");
		puts("");
	}

	return safe;

}



int main(int argc,char *argv[])
{

	int i,j,requesting_process,safe; //i for iterating through n processes and j for iterating
through m processes
	char ch;

	puts("Enter the number of processes: ");
	scanf("%d",&n);
	puts("");
```

```
puts("Enter the number of resource types: ");
scanf("%d",&m);
puts("");

puts("Enter the Available matrix");
for(j=0;j<m;j++)
{
	printf("Resource %d: ",j);
	scanf("%d",&available[j]);
}
puts("");

puts("Enter the Max matrix: ");
for(i=0;i<n;i++)
{
	printf("Process %d\n",i);
	for(j=0;j<m;j++)
		scanf("%d",&max[i][j]);
	puts("");
}
puts("");

puts("Enter the Allocation matrix:");
for(i=0;i<n;i++)
{
	printf("Process %d\n",i);
	for(j=0;j<m;j++)
		scanf("%d",&allocation[i][j]);
	puts("");
}
puts("");

for(i=0;i<n;i++)
	for(j=0;j<m;j++)
		need[i][j]=max[i][j]-allocation[i][j];
puts("");

/* NEED MATRIX PRINT*/
puts("\tNeed Matrix");
for(ch='A',i=0;i<m;i++,ch++)
	printf("\t%c",ch);

puts("");
for(i=0;i<n;i++)
{
	printf("P%d",i);
	for(j=0;j<m;j++)
		printf("\t%d",need[i][j]);
	puts("");
}
puts("");


present_snapshot();

safe = safety_algorithm();

if(safe == false)
{
	puts("System is in unsafe state!");
	puts("Exiting...");
	exit(1);
}

else
{
	puts("Enter the process requesting: ");
	scanf("%d",&requesting_process);

	puts("Enter the Request matrix: ");
	for(j=0;j<m;j++)
		scanf("%d",&request[j]);

	if(compare(request,need[requesting_process]))
	{
		if(compare(request,available))
		{
			for(j=0;j<m;j++)
```

```
				{
					available[j] = available[j]-request[j];
					allocation[requesting_process][j] = 
allocation[requesting_process][j] + request[j];
					need[requesting_process][j] = need[requesting_process][j] - 
request[j];
				}
				safe = safety_algorithm();

				if(safe)
				{
					puts("Resources allocated successfully.");
					exit(0);
				}
				else
				{
					//restore to previous allocation state
					available[j] = available[j] + request[j];
					allocation[requesting_process][j] = 
allocation[requesting_process][j] - request[j];
					need[requesting_process][j] = need[requesting_process][j] + 
request[j];
				}

			}
			else
			{
				printf("Process_%d must wait\n",requesting_process);
				exit(1);
			}
		}
		else
		{
			puts("Maximum request limit exceeded");
			exit(1);
		}
	}
	return 0;
}
```

OUTPUT:


krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7
krish-thorcode@kkm-ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7$ ./bankers_algo
Enter the number of processes:
5
Enter the number of resource types:
3
Enter the Available matrix
Resource 0: 3
Resource 1: 3
Resource 2: 2
Enter the Max matrix:
Process 0
7
5
3
Process 1
3
2
2
Process 2
9
0
2
Process 3
2
2
2
Process 4
4
3
3
Enter the Allocation matrix:
Process 0
0
1
0

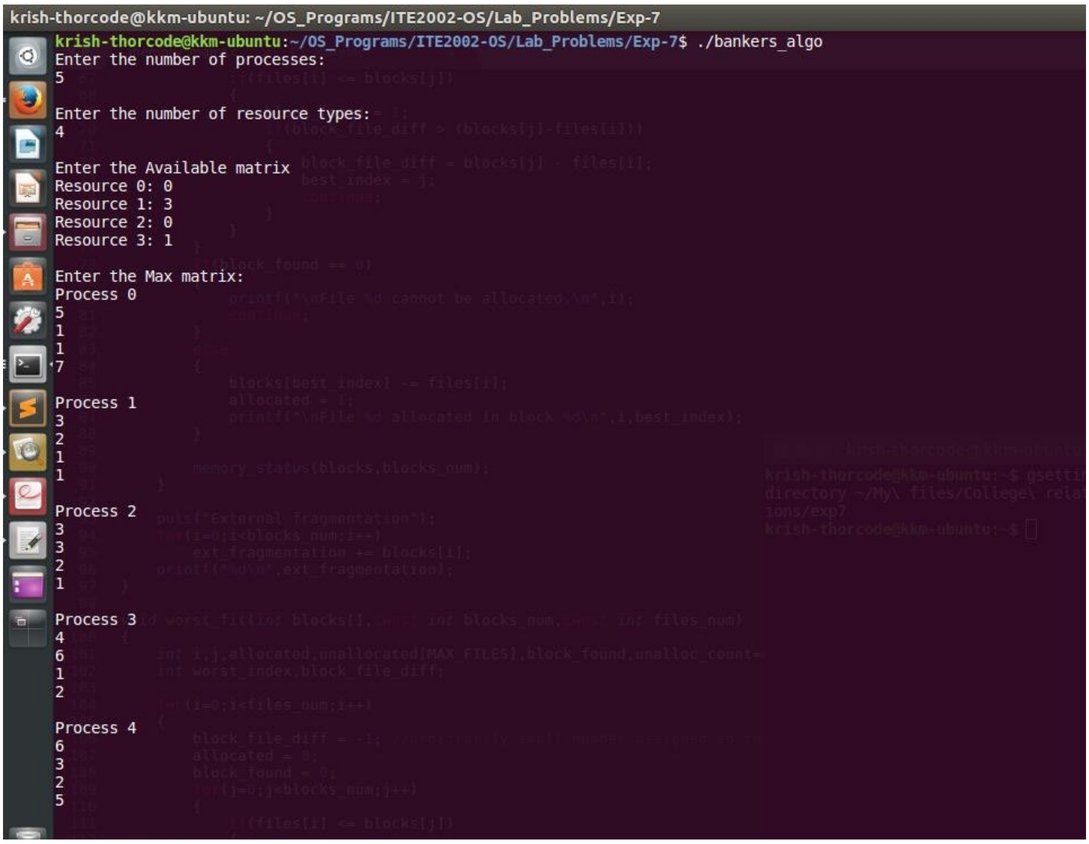

krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7
Process 1
2
0
0
Process 2
3
0
2
Process 3
2
1
1
Process 4
0
0
2
Need Matrix
A B C
P0 7 4 3
P1 1 2 2
P2 6 0 0
P3 0 1 1
P4 4 3 1
Allocation Matrix
A B C
P0 0 1 0
P1 2 0 0
P2 3 0 2
P3 2 1 1
P4 0 0 2
Max Matrix
A B C
P0 7 5 3
P1 3 2 2

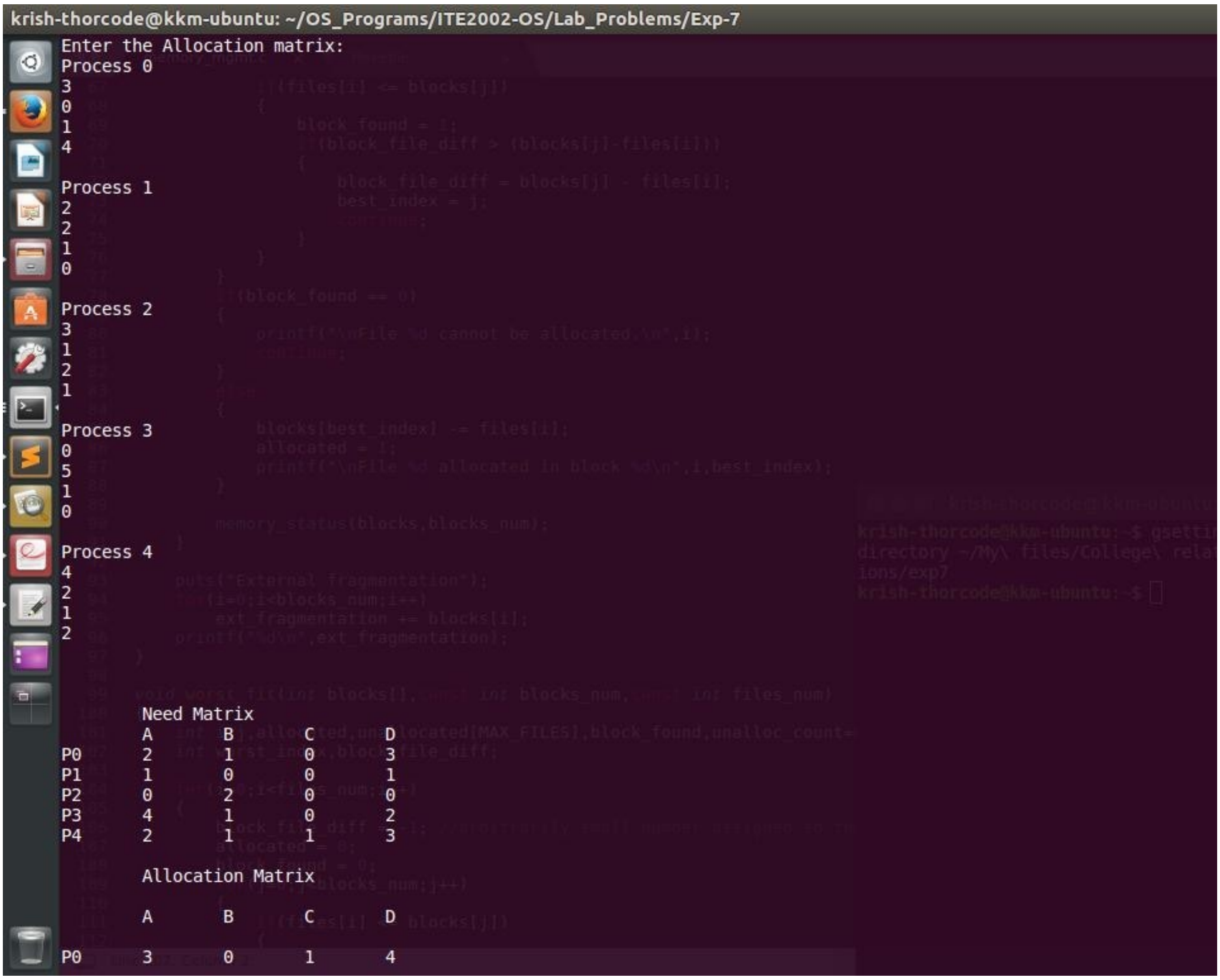
krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7

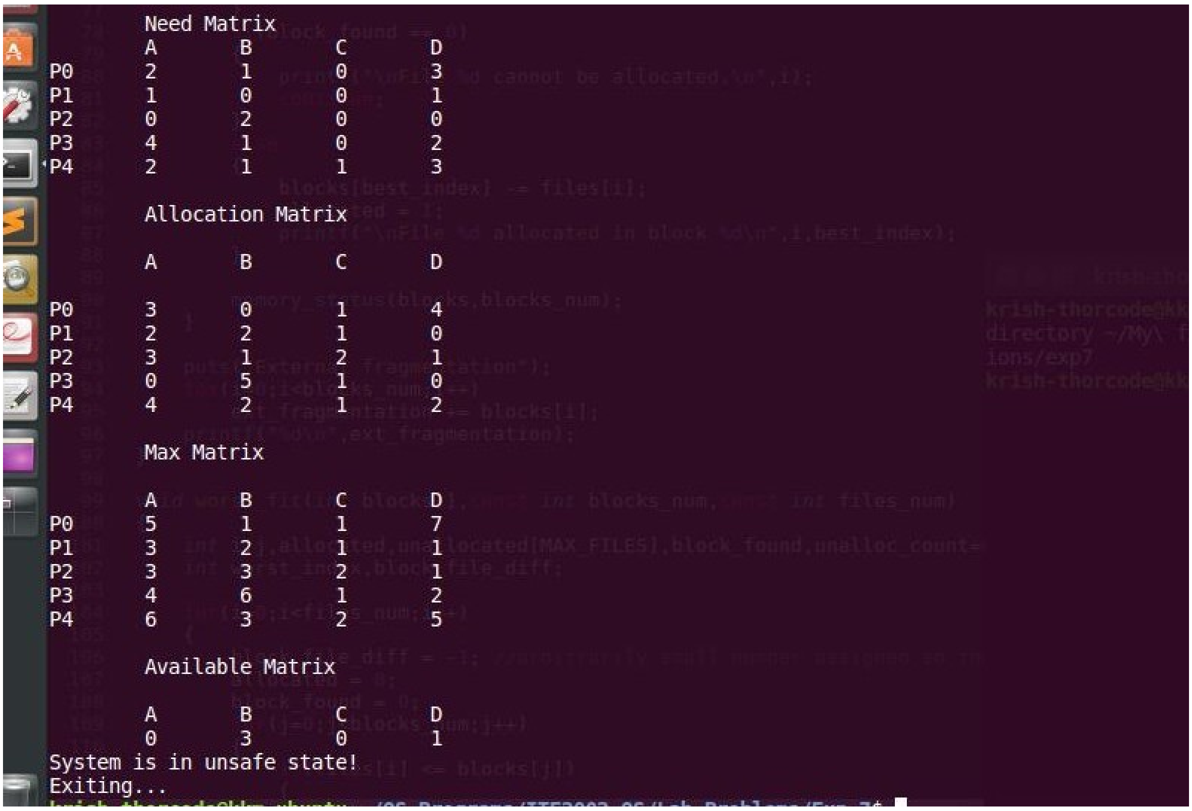
```
        A       B       C
P0      7       5       3
P1      3       2       2
P2      9       0       2
P3      2       2       2
P4      4       3       3

        Available Matrix

        A       B       C
        3       3       2
Safe sequence:
< 1, 3, 4, 0, 2, >
Enter the process requesting:
1
Enter the Request matrix:
1
0
2
        Allocation Matrix

        A       B       C

P0      0       1       0
P1      3       0       2
P2      3       0       2
P3      2       1       1
P4      0       0       2

        Max Matrix

        A       B       C
P0      7       5       3
P1      3       2       2
P2      9       0       2
P3      2       2       2
P4      4       3       3

        Available Matrix

        A       B       C
        9       5       5
Safe sequence:
< 0, 1, 2, 3, 4, >
Resources allocated successfully.
krish-thorcode@kkm-ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7$
```

b.) For the given snapshot:

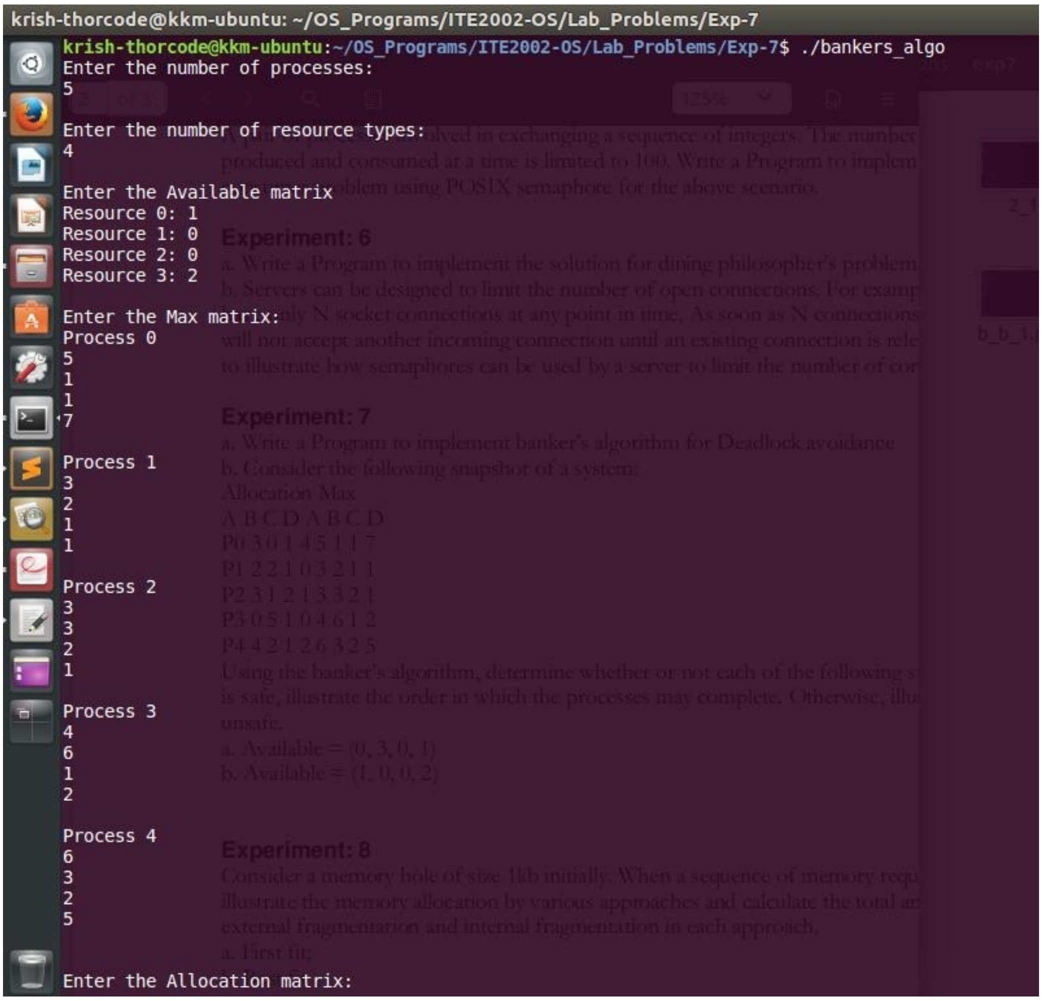
i)

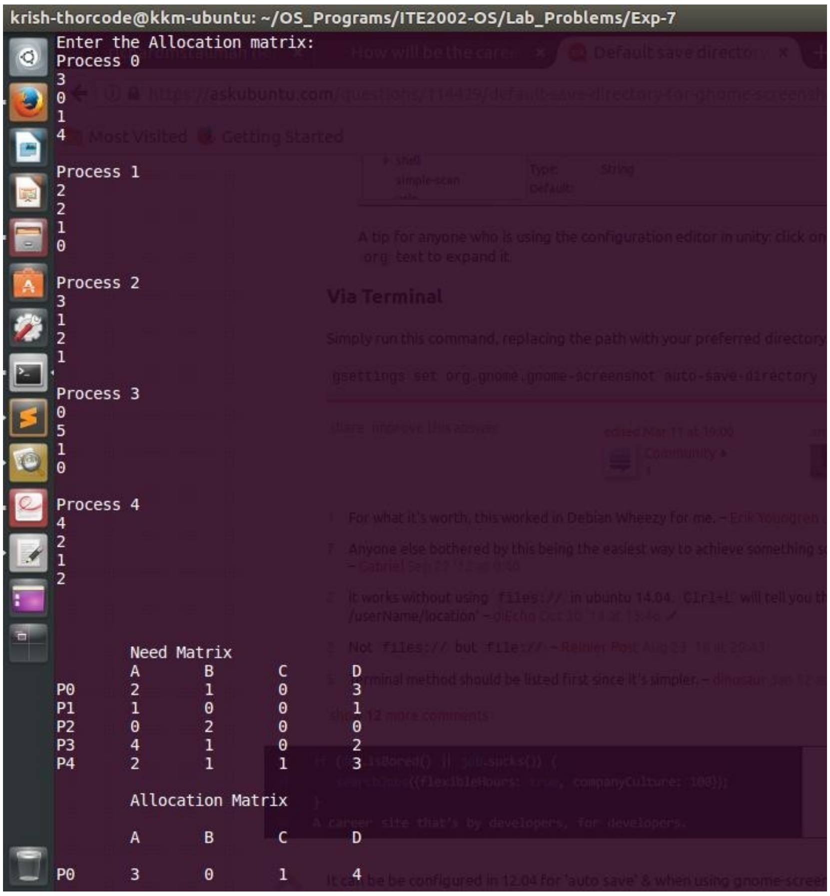




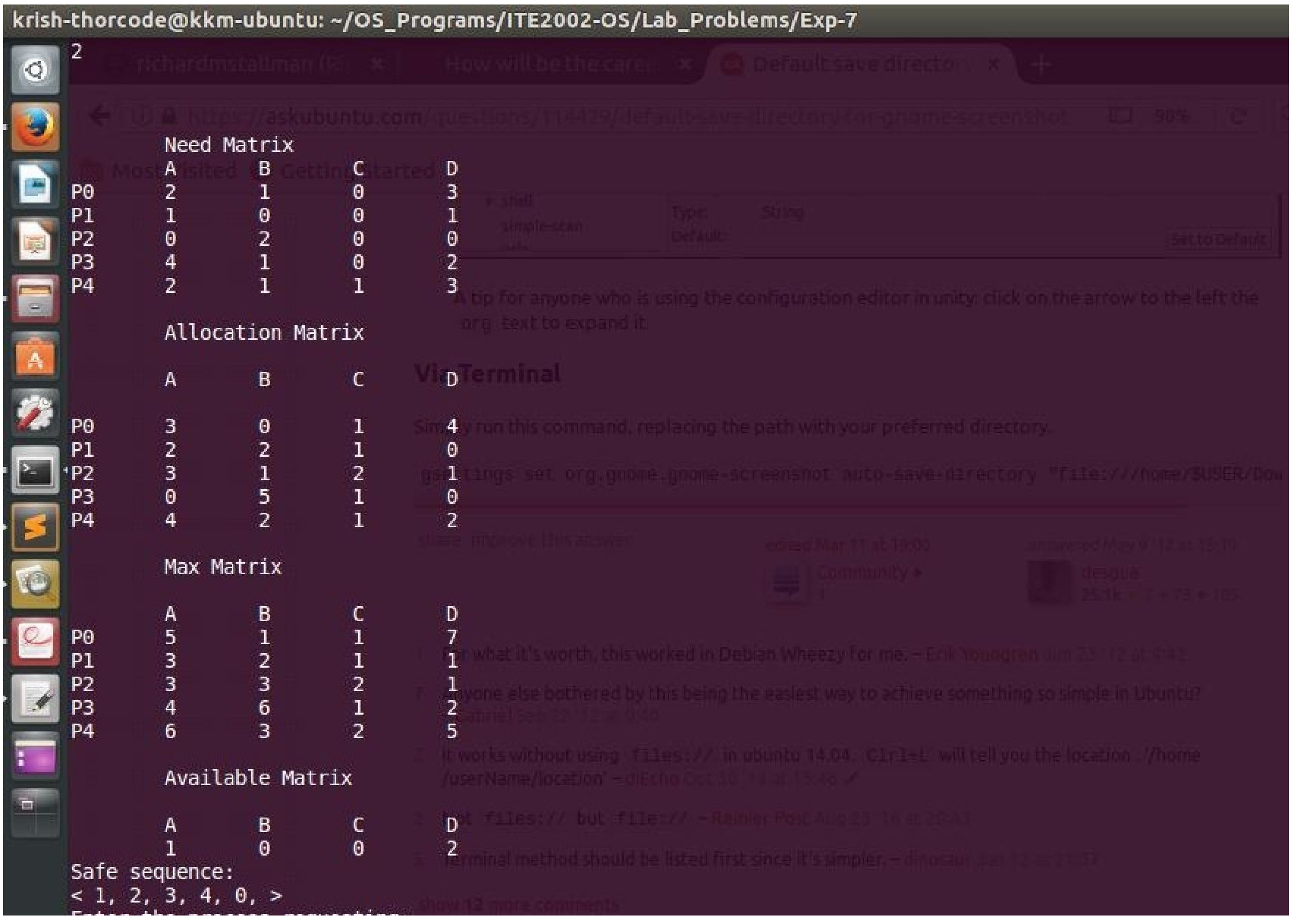

Need Matrix
A B C D
P0 2 1 0 3
P1 1 0 0 1
P2 0 2 0 0
P3 4 1 0 2
P4 2 1 1 3
Allocation Matrix
A B C D
P0 3 0 1 4
P1 2 2 1 0
P2 3 1 2 1
P3 0 5 1 0
P4 4 2 1 2
Max Matrix
A B C D
P0 5 1 1 7
P1 3 2 1 1
P2 3 3 2 1
P3 4 6 1 2
P4 6 3 2 5
Available Matrix
A B C D
0 3 0 1
System is in unsafe state!
Exiting...

ii.)


krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7
krish-thorcode@kkm-ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7$ ./bankers_algo
Enter the number of processes:
5
Enter the number of resource types:
4
Enter the Available matrix
Resource 0: 1
Resource 1: 0
Resource 2: 0
Resource 3: 2
Enter the Max matrix:
Process 0
5
1
1
7
Process 1
3
2
1
1
Process 2
3
3
2
1
Process 3
4
6
1
2
Process 4
6
3
2
5
Enter the Allocation matrix:



```
krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7
Enter the Allocation matrix:
Process 0
3
0
1
4

Process 1
2
2
1
0

Process 2
3
1
2
1

Process 3
0
5
1
0

Process 4
4
2
1
2


        Need Matrix
        A       B       C       D
P0      2       1       0       3
P1      1       0       0       1
P2      0       2       0       0
P3      4       1       0       2
P4      2       1       1       3

        Allocation Matrix

        A       B       C       D

P0      3       0       1       4
```



```
krish-thorcode@kkm-ubuntu: ~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-7
2


        Need Matrix
        A       B       C       D
P0      2       1       0       3
P1      1       0       0       1
P2      0       2       0       0
P3      4       1       0       2
P4      2       1       1       3

        Allocation Matrix

        A       B       C       D

P0      3       0       1       4
P1      2       2       1       0
P2      3       1       2       1
P3      0       5       1       0
P4      4       2       1       2

        Max Matrix

        A       B       C       D
P0      5       1       1       7
P1      3       2       1       1
P2      3       3       2       1
P3      4       6       1       2
P4      6       3       2       5

        Available Matrix

        A       B       C       D
        1       0       0       2
Safe sequence:
< 1, 2, 3, 4, 0, >
Enter the process requesting
```