Banker's Algorithm to avoid Deadlocks.

1. Problem Statement

To simulate the banker's algorithm for avoiding deadlock in an operating system as demonstrated in Fig. 6.9 of the course text. The input is provided in the form of a formatted input file containing the number of resources and processes. The file also contains the resource vector, the claims matrix and the allocation vector.

The importance of the project is to learn the concepts of deadlock avoidance in an OS when the maximum number of a resource required by a process is known and the independence of the processes in question is guaranteed. Deadlock is avoided by checking if the current system state meets all the criteria for all the processes to be completed in a specific sequence that can avoid a deadlock, if not the system is in an unsafe state and some other workaround must be employed.

2. Approach

Language used: C++11

Compiler used: g++

Editor used: micro

Debugging tools used: gdb.

First task was to read in the input file and calculate the current request matrix of all processes as well as the vector of resource availability. And also create a struct which can hold the current system state info as given in Fig. 6.9a of the course textbook.

Next task was to implement the resource allocation algorithm as given in Fig 6.9b of the textbook.

Last came the implementation of the banker's algorithm, which is the crux of this program. The pseudo-code for the algorithm as state in Fig 6.9c was implemented as a method that can be called by the code in the resource allocation algorithm. Methods were also defined to reduce code repetition and make the program more modular and easier to follow.

3. Solution

No major bugs were detected while implementing the algorithms mentioned earlier.

The solution is merely an adaptation of the algorithm provided in Fig 6.9 with blanks filled in where the algorithm abstracted a step in terms natural language.

To build the code g++ is invoked as follows.

```
$ g++ -o prog3 main.cpp
```

To run the code the following command is executed, with the name of the input file to be used supplied as a command line argument. Optionally, output from the programmed can be redirected to a .txt file instead of output to the terminal window and the optional args. are mentioned in the curly braces.

```
$ ./prog3 input3.txt {> output1.txt}
```

Three sets of input files were used for the program verification.

1. **input1.txt** – this follows the input used for the example illustrated in the textbook (Fig. 6.7), the various intermediary states attained are printed out and finally the sequence of process execution for deadlock avoidance is printed out to the screen.

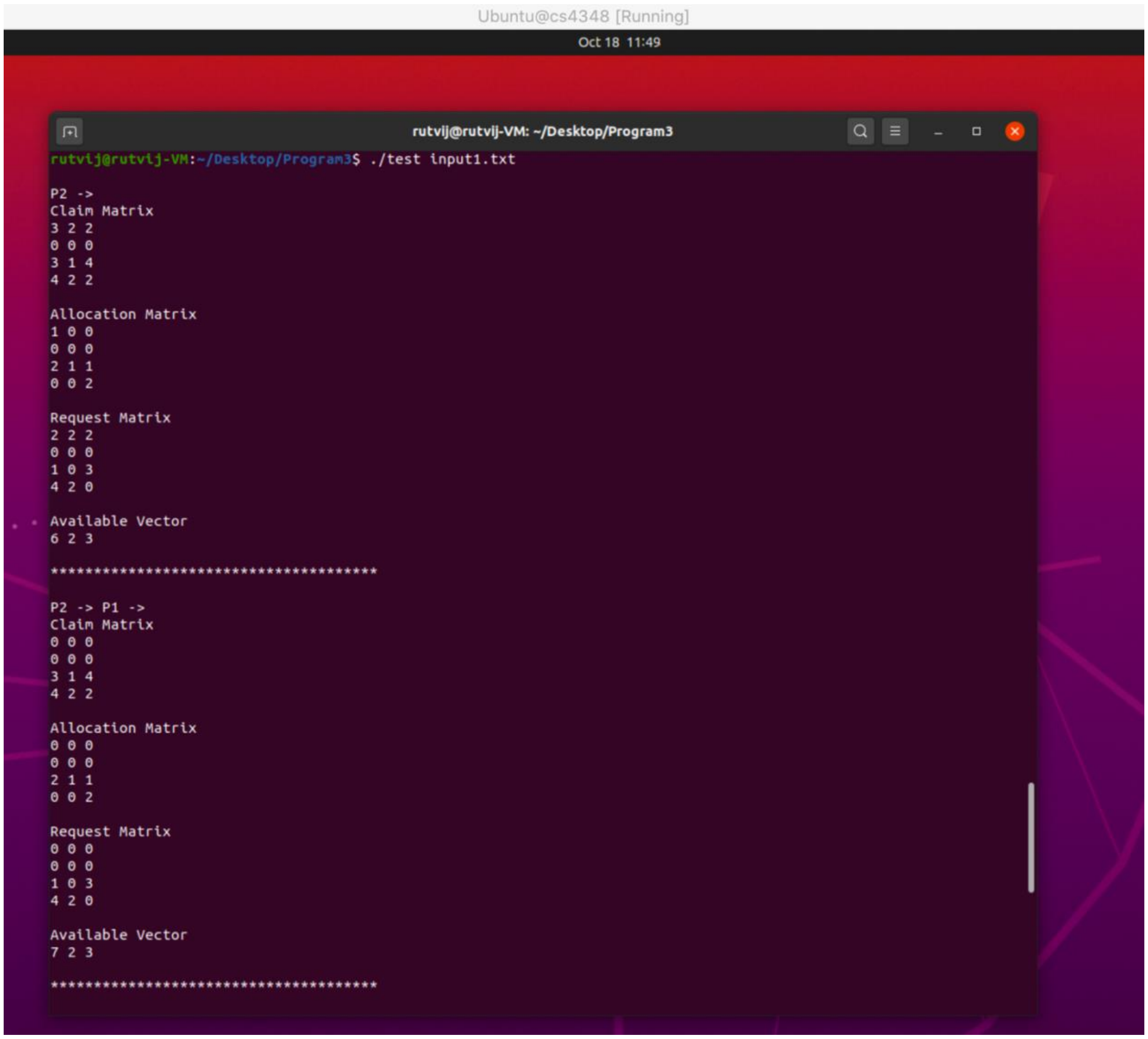
   The output generated by this input is in the figures (Figure 1 and 2) below and also in the text file "output1.txt."

2. i**nput2.txt** – this follows the input in Problem 6.5 of the after-chapter questions, the various intermediary states attained are printed out and finally the sequence of process execution for deadlock avoidance is printed out to the screen.

   The output generated by this input is in the figures (Figure 3, 4 and 5) below and also in the text file "output2.txt."

3. **input3.txt** – was the input provided with the problem statement and for which we had to check whether the system is in a safe state or not. The output shows that the system wasn't in a safe state.

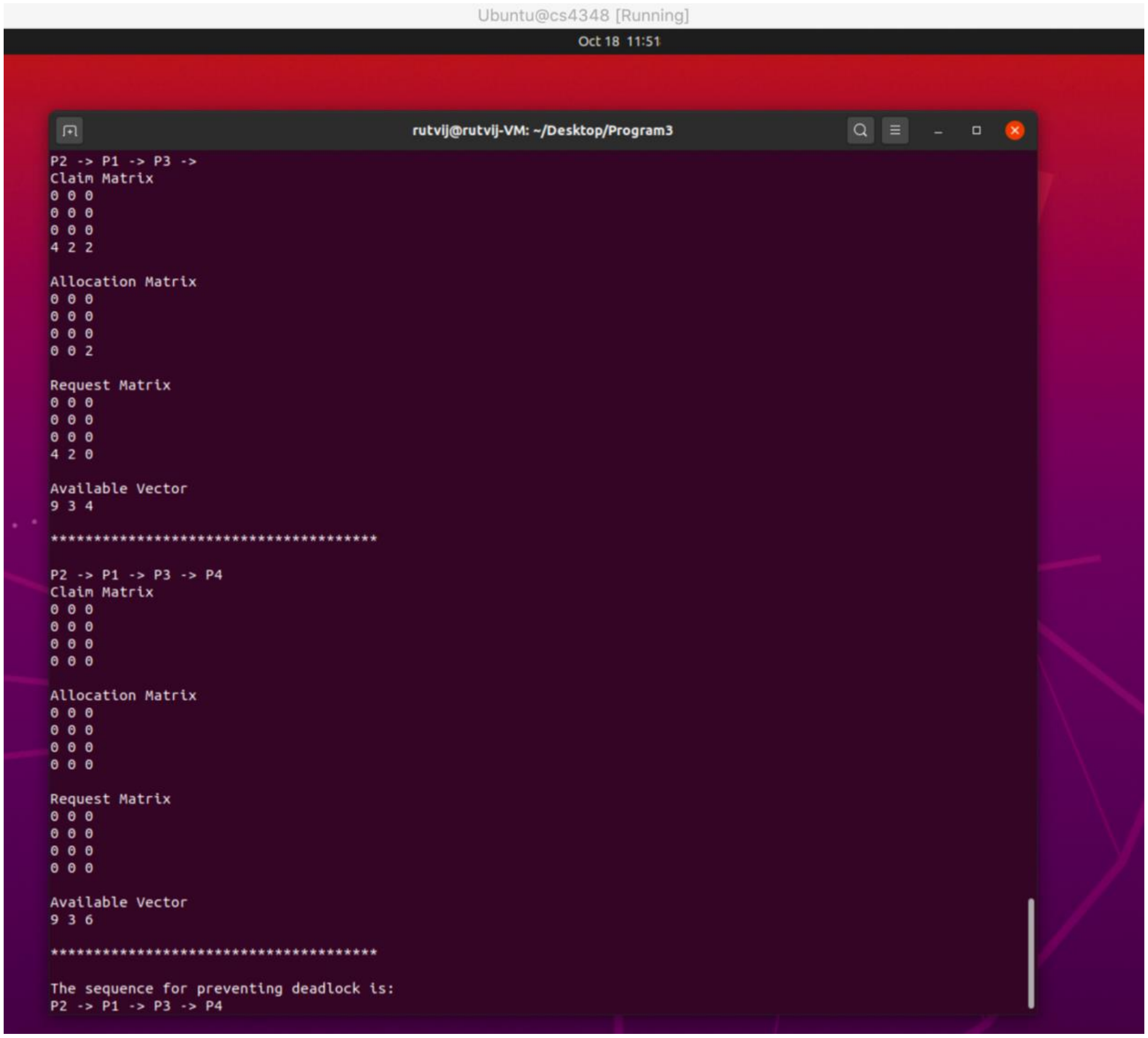
   The output generated by this input is in the figures (Figure 6) below and also in the text file "output3.txt."



```
rutvij@rutvij-VM:~/Desktop/Program3$ ./test input1.txt

P2 ->
Claim Matrix
3 2 2
0 0 0
3 1 4
4 2 2

Allocation Matrix
1 0 0
0 0 0
2 1 1
0 0 2

Request Matrix
2 2 2
0 0 0
1 0 3
4 2 0

Available Vector
6 2 3

**************************************

P2 -> P1 ->
Claim Matrix
0 0 0
0 0 0
3 1 4
4 2 2

Allocation Matrix
0 0 0
0 0 0
2 1 1
0 0 2

Request Matrix
0 0 0
0 0 0
1 0 3
4 2 0

Available Vector
7 2 3

**************************************
```

Figure 1

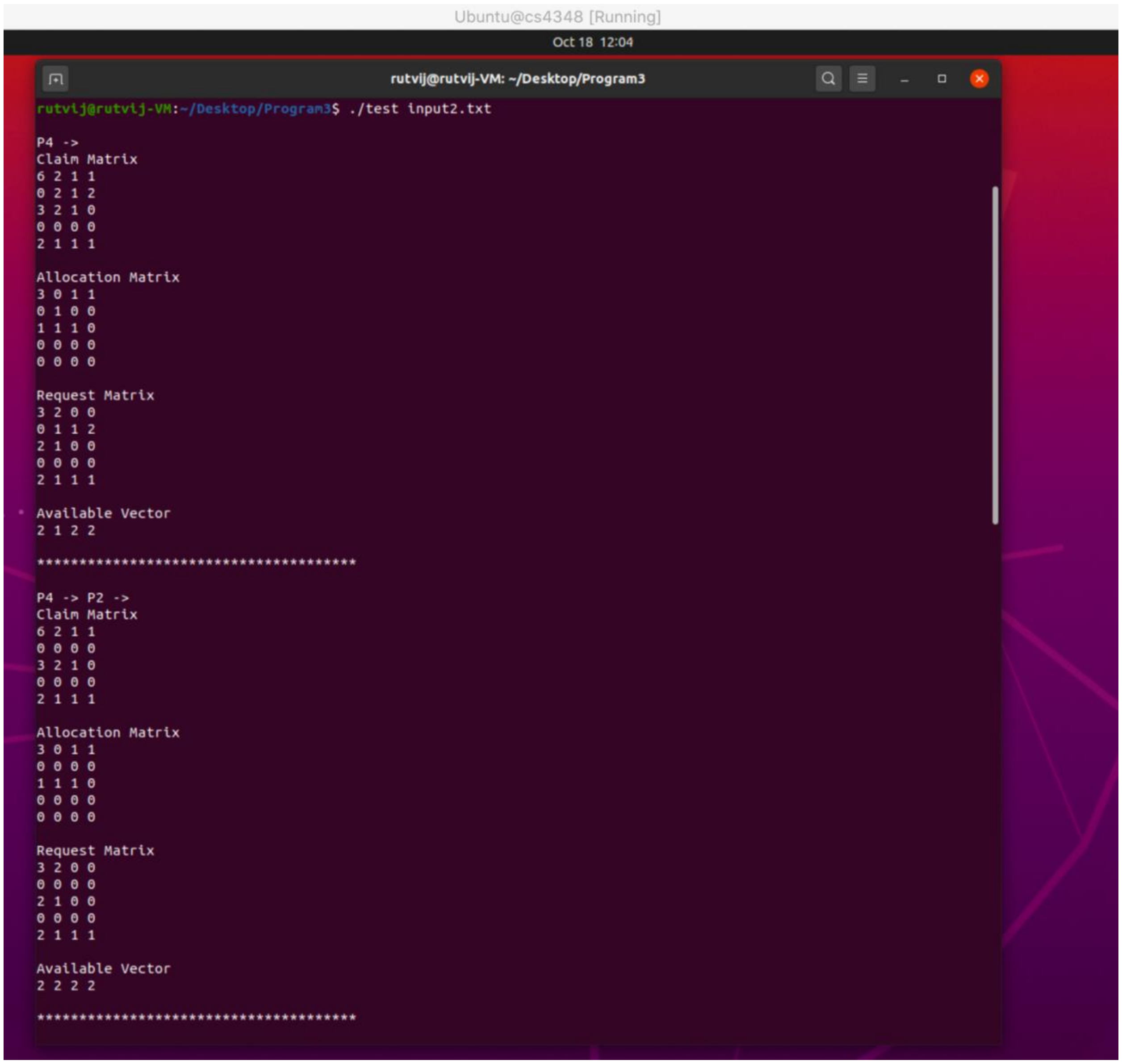
The state matrices as the program simulates input from input1.txt



```
P2 -> P1 -> P3 ->
Claim Matrix
0 0 0
0 0 0
0 0 0
4 2 2

Allocation Matrix
0 0 0
0 0 0
0 0 0
0 0 2

Request Matrix
0 0 0
0 0 0
0 0 0
4 2 0

Available Vector
9 3 4

**************************************

P2 -> P1 -> P3 -> P4
Claim Matrix
0 0 0
0 0 0
0 0 0
0 0 0

Allocation Matrix
0 0 0
0 0 0
0 0 0
0 0 0

Request Matrix
0 0 0
0 0 0
0 0 0
0 0 0

Available Vector
9 3 6

**************************************

The sequence for preventing deadlock is:
P2 -> P1 -> P3 -> P4
```

Figure 2

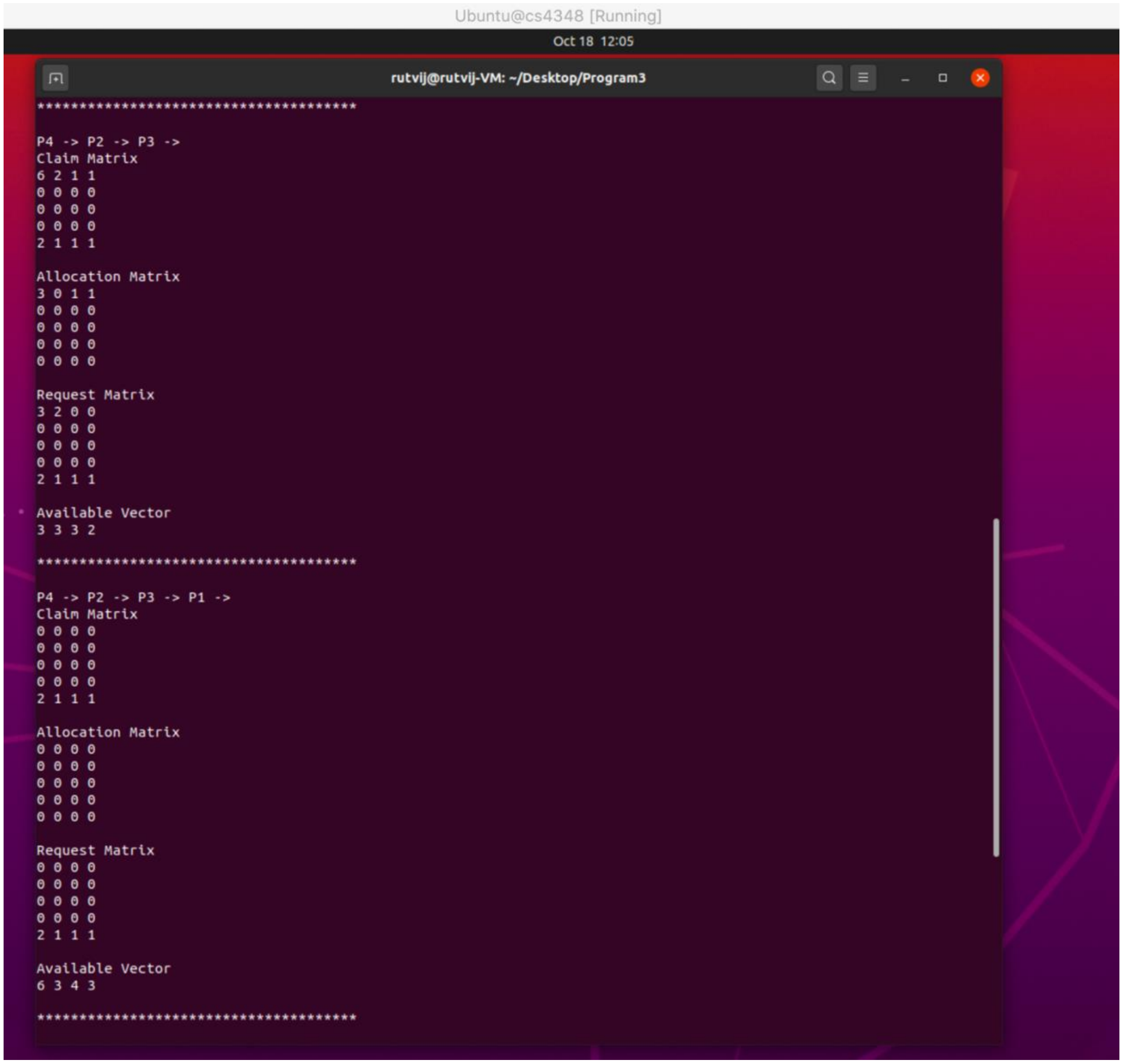
The state matrices as the program simulates input from input1.txt. Last line gives the sequence of process execution to avoid deadlock.



```
rutvij@rutvij-VM:~/Desktop/Program3$ ./test input2.txt

P4 ->
Claim Matrix
6 2 1 1
0 2 1 2
3 2 1 0
0 0 0 0
2 1 1 1

Allocation Matrix
3 0 1 1
0 1 0 0
1 1 1 0
0 0 0 0
0 0 0 0

Request Matrix
3 2 0 0
0 1 1 2
2 1 0 0
0 0 0 0
2 1 1 1

Available Vector
2 1 2 2

**************************************

P4 -> P2 ->
Claim Matrix
6 2 1 1
0 0 0 0
3 2 1 0
0 0 0 0
2 1 1 1

Allocation Matrix
3 0 1 1
0 0 0 0
1 1 1 0
0 0 0 0
0 0 0 0

Request Matrix
3 2 0 0
0 0 0 0
2 1 0 0
0 0 0 0
2 1 1 1

Available Vector
2 2 2 2

**************************************
```

Figure 3

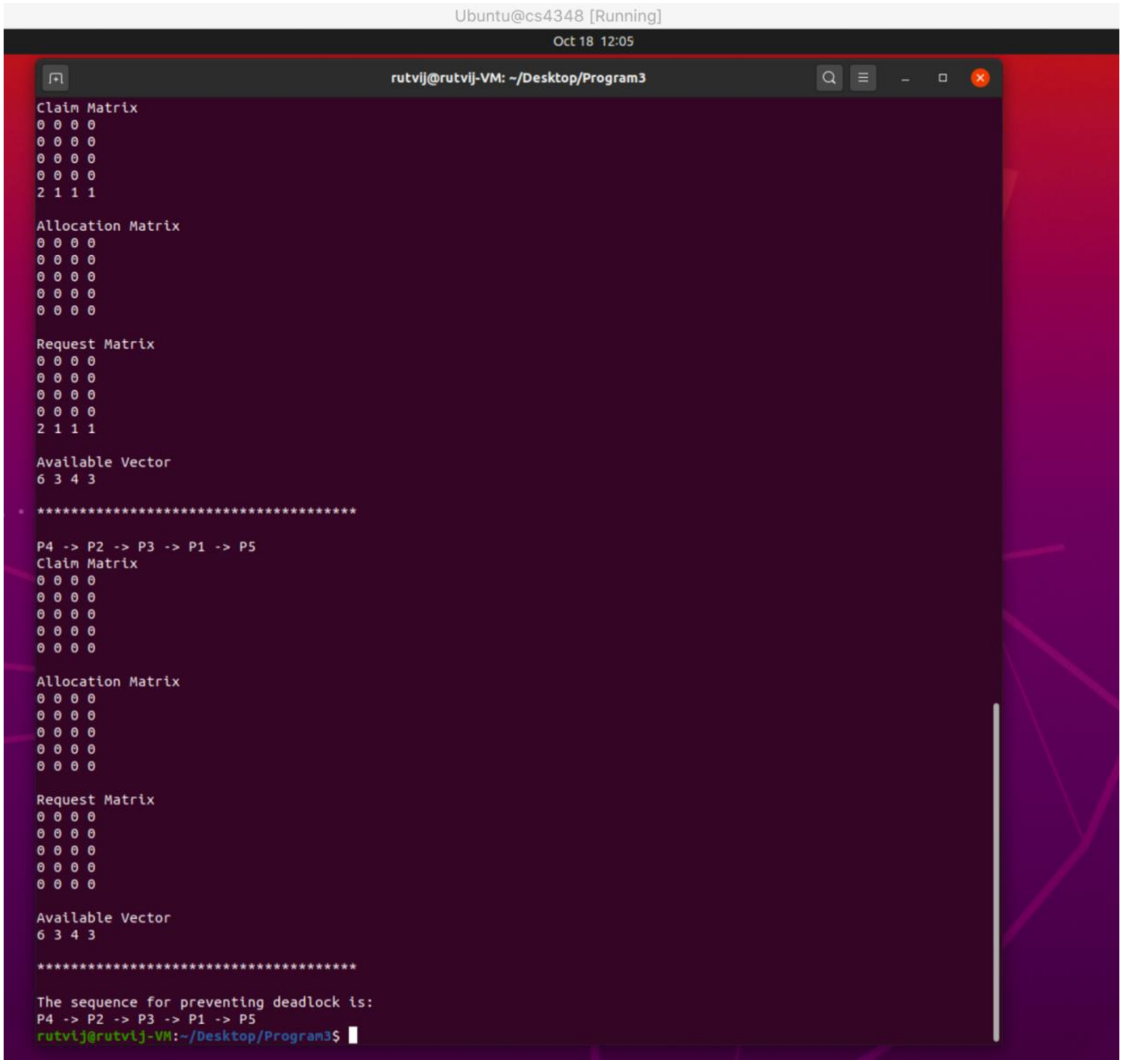
The state matrices as the program simulates input from input2.txt



```
**************************************

P4 -> P2 -> P3 ->
Claim Matrix
6 2 1 1
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Allocation Matrix
3 0 1 1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Request Matrix
3 2 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Available Vector
3 3 3 2

**************************************

P4 -> P2 -> P3 -> P1 ->
Claim Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Allocation Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Request Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Available Vector
6 3 4 3

**************************************
```

Figure 4

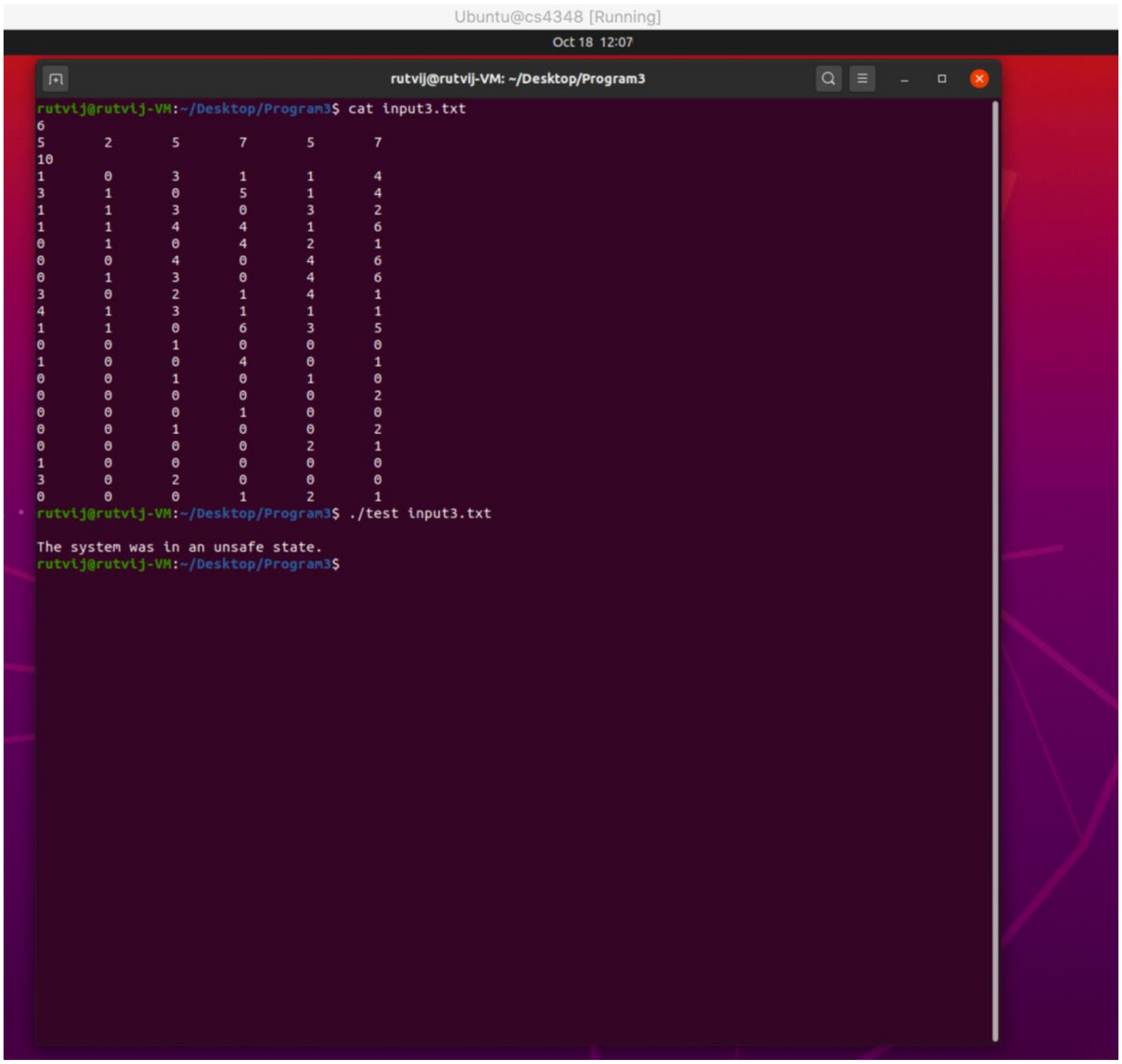
The state matrices as the program simulates input from input2.txt



```
Claim Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Allocation Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Request Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 1 1 1

Available Vector
6 3 4 3

**************************************

P4 -> P2 -> P3 -> P1 -> P5
Claim Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Allocation Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Request Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Available Vector
6 3 4 3

**************************************

The sequence for preventing deadlock is:
P4 -> P2 -> P3 -> P1 -> P5
rutvij@rutvij-VM:~/Desktop/Program3$
```

Figure 5

The state matrices as the program simulates input from input2.txt and the last line gives a sequence of process execution which would help avoid deadlock.



```
rutvij@rutvij-VM:~/Desktop/Program3$ cat input3.txt
6
5       2       5       7       5       7
10
1       0       3       1       1       4
3       1       0       5       1       4
1       1       3       0       3       2
1       1       4       4       1       6
0       1       0       4       2       1
0       0       4       0       4       6
0       1       3       0       4       6
3       0       2       1       4       1
4       1       3       1       1       1
1       1       0       6       3       5
0       0       1       0       0       0
1       0       0       4       0       1
0       0       1       0       1       0
0       0       0       0       0       2
0       0       0       1       0       0
0       0       1       0       0       2
0       0       0       0       2       1
1       0       0       0       0       0
3       0       2       0       0       0
0       0       0       1       2       1
rutvij@rutvij-VM:~/Desktop/Program3$ ./test input3.txt

The system was in an unsafe state.
rutvij@rutvij-VM:~/Desktop/Program3$
```

Figure 6

The contents of input3.txt which can be verified to be the same as the contents of “Pgm3 Input.txt” which is the file supplied by the professor. It also shows the execution of the program on the contents of the file and that the algorithm finds that the system is in an unsafe state to begin with.