

Practical no: 9

Problem Statement: : Write a Java program to implement Banker's Algorithm.

Name: Aditi Dinesh Mulay

Class: T.E. Computer

Subject: SPOS

Div: A

Roll no: 02

PRN No. 71918146B

Spos

Assignment C-2

Aditi Dinesh Mway
T.E. Comp Div: A
Roll no: 02.

Aim: Bankers algorithm for deadlock detection & avoidance.

Theory:

Banker's algorithm is a resource allocation & deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an 's-state' check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following Data Structures used to implement Banker's Algorithm.

Let 'n' be the number of processes in the system & 'm' be the number of resources type.

Available:

- 1) It is a 1-d array of size 'm' indicating the no of available resources of each type.
- 2) Available [j] = k means there are 'k' instances of resource type Rj.

Max:

- 1) It is a 2-d array of size $n \times m$ that defines the maximum demand of each process in a system.
- 2) $\text{Max}[i, j] = k$ means process P_i may request atmost 'k' instances of resource type Rj.

Allocation:

- 1) It is 2-d array of size $n \times m$ that defines the no. of resources of each type currently allocated to each process.
- 2) Allocation $[i, j] = k$ means process P_i is currently allocated k instance of resource type R_j .

Need:

- 1) It is 2-d array of size $n \times m$ that indicates the remaining resource need of each process.
- 2) Need $[i, j] = k$ means process P_i currently need k instances of resource type R_j for its execution.
- 3) Need $[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

Banker's algorithm consist of Safety algorithm & Resource request algorithm:

Safety Algorithm:

- 1) Let work and Finish be vectors of length m & n respectively.
Initialize: work = Available
Finish[i] = false; for $i = 1, 2, 3, 4 \dots n$.
- 2) Find an i such that both
 - a) Finish[i] = false
 - b) Need_i \leq Work_iif no such i exists goto step 4.
- 3) Work = Work + Allocation[i]
Finish[i] = true
goto step(2)

4) if $\text{Finish}[i] \leftarrow \text{true}$ for all i
then system is in safe state.

Ex: Considering a system with 5 processes P_0 through P_4 & 3 resources type A, B, C. Resource type A has 10 instances, B has 5 & C has 7 instances.
At time t_0 .

Process	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			

1) What will be the content of Need matrix?
 $\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$
 \therefore Need of Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

2) Is the system in safe state? If yes, then what is safe sequence?
 \rightarrow Applying a safety algorithm on system,

Step 1 of Safety Algo

$m=3, n=5$
Work = Available
Work =

3	3	2
---	---	---

Finish =

false	false	false	false	false
-------	-------	-------	-------	-------

For $i=0$
Need₀ = 7, 4, 3
Finish [0] is false and Need₀ > Work
So P_0 must wait

For $i=1$
Need₁ = 1, 2, 2
Finish [1] is false and Need₁ < Work
So P_1 must be kept in safe sequence

Step 3
Work = Work + Allocation₁
Work =

5	3	2
---	---	---

Finish =

false	true	false	false	false
-------	------	-------	-------	-------

For $i=2$
Need₂ = 6, 0, 0
Finish [2] is false and Need₂ > Work
So P_2 must wait

For $i=3$
Need₃ = 0, 1, 1
Finish [3] is false and Need₃ < Work
So P_3 must be kept in safe sequence

Step 3
Work = Work + Allocation₃
Work =

7	4	3
---	---	---

Finish =

false	true	false	true	false
-------	------	-------	------	-------

For $i=4$
Need₄ = 4, 3, 1
Finish [4] is false and Need₄ < Work
So P_4 must be kept in safe sequence

Step 3
Work = Work + Allocation₄
Work =

7	4	5
---	---	---

Finish =

false	true	false	true	true
-------	------	-------	------	------

Step 4
Finish [i] = true for $0 \leq i \leq n$
Hence the system is in Safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2

Step 2
Need₃ = 0, 1, 1
Finish [3] is false and Need₃ < Work
So P_3 must be kept in safe sequence

Step 3
Work = Work + Allocation₃
Work =

7	4	3
---	---	---

Finish =

false	true	false	true	false
-------	------	-------	------	-------

Step 2
Need₂ = 6, 0, 0
Finish [2] is false and Need₂ < Work
So P_2 must be kept in safe sequence

Step 3
Work = Work + Allocation₂
Work =

10	5	7
----	---	---

Finish =

true	true	true	true	true
------	------	------	------	------

Step 4
Finish [i] = true for $0 \leq i \leq n$
Hence the system is in Safe state

Step 2
Need₃ = 0, 1, 1
Finish [3] is false and Need₃ < Work
So P_3 must be kept in safe sequence

Step 3
Work = Work + Allocation₃
Work =

7	4	3
---	---	---

Finish =

false	true	false	true	false
-------	------	-------	------	-------

Step 2
Need₂ = 6, 0, 0
Finish [2] is false and Need₂ < Work
So P_2 must be kept in safe sequence

Step 3
Work = Work + Allocation₂
Work =

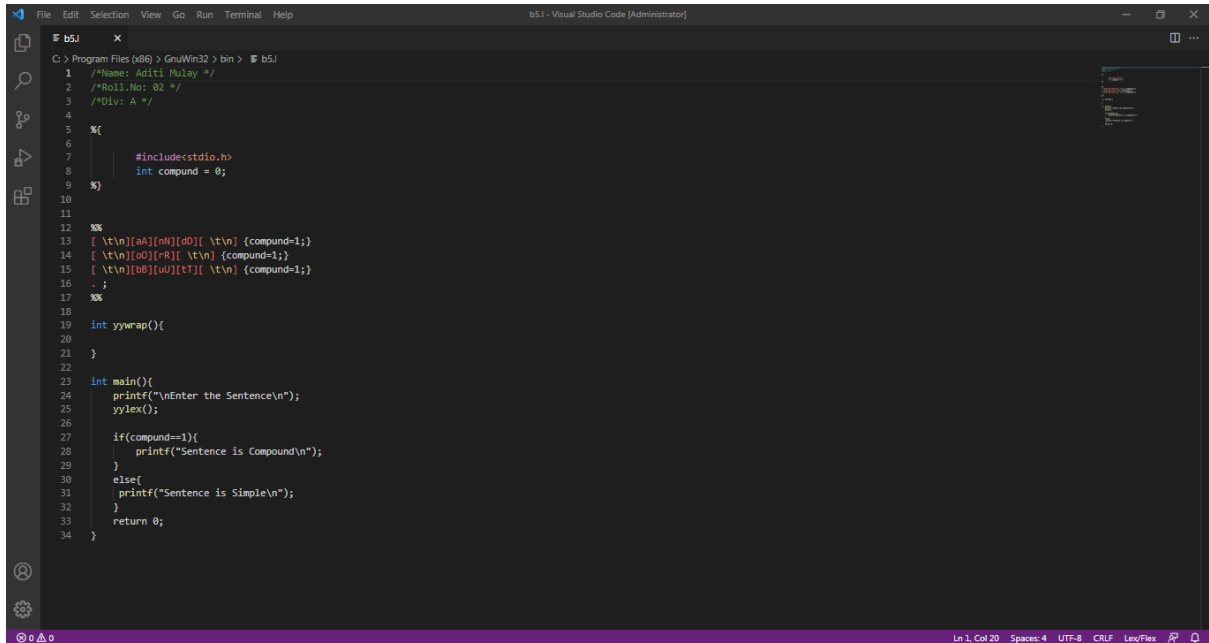
10	5	7
----	---	---

Finish =

true	true	true	true	true
------	------	------	------	------

Step 4
Finish [i] = true for $0 \leq i \leq n$
Hence the system is in Safe state

Program:



The image shows a Visual Studio Code editor window with a C program. The program is designed to classify a sentence as either 'Compound' or 'Simple' based on the presence of specific punctuation marks. The code includes comments for author and roll number, and uses a character array to store the input sentence. It checks for commas, semicolons, and exclamation marks to determine if the sentence is compound. If none of these are found, it classifies the sentence as simple. The program uses the `yywrap()` function for line wrapping and the `system()` function to clear the screen.

```
1  /*Name: Aditi Mulya */
2  /*Roll.No: 02 */
3  /*Div: A */
4
5  %
6
7  #include<stdio.h>
8  int compound = 0;
9  %
10
11  %
12
13  [ \t\n][aA][mM][dD][ \t\n] {compound=1;}
14  [ \t\n][oO][rR][ \t\n] {compound=1;}
15  [ \t\n][bB][uU][tT][ \t\n] {compound=1;}
16  - ;
17  %
18
19  int yywrap(){
20  }
21
22
23  int main(){
24      printf("\nEnter the Sentence\n");
25      yylex();
26
27      if(compound==1){
28          printf("Sentence is Compound\n");
29      }
30      else{
31          printf("Sentence is Simple\n");
32      }
33      return 0;
34  }
```