

Experiment 08- To write a program to implement the solution of banker's algorithm

Learning Objective: Students should be able to understand banker's algorithms by using different coding languages, such as C/C++, Java, and Python.

Tools: Online compiler

Theory: The Banker's Algorithm is a deadlock avoidance algorithm that ensures a system remains in a safe state by pre-allocating resources based on the maximum demand of each process. It works as follows:

1. **Need Matrix Calculation:** The need for each process is determined using the formula: $\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$
2. **Safety Check:** The system checks if a safe sequence of process execution exists, ensuring that no deadlock occurs.
3. **Resource Allocation:** If a process's request can be satisfied with available resources, they are allocated temporarily to check for safety.
4. **Safe Sequence Determination:** The algorithm identifies a sequence of process execution such that all processes complete execution without deadlocks.

The Banker's Algorithm is widely used in operating systems to manage multiple processes requiring shared resources. It ensures that the system remains in a safe state by granting resource requests only if doing so does not lead to a deadlock. It is particularly useful in environments where resource allocation must be managed efficiently, such as cloud computing, real-time systems, and database management. By simulating different possible execution sequences, the algorithm guarantees that all processes can eventually complete without causing system instability. This method is crucial in modern computing for maintaining system reliability and optimal performance.

Code: `def calculate_need(need, maxm, allot, P, R):`

`for i in range(P):`

`for j in range(R):`

`need[i][j] = maxm[i][j] - allot[i][j]`

`def is_safe(processes, avail, maxm, allot, P, R):`

`need = [[0] * R for _ in range(P)]`

`calculate_need(need, maxm, allot, P, R)`

```
finish = [False] * P

safe_seq = [0] * P

work = avail[:]

count = 0

while count < P:

    found = False

    for p in range(P):

        if not finish[p]:

            if all(need[p][j] <= work[j] for j in range(R)):

                for k in range(R):

                    work[k] += allot[p][k]

                safe_seq[count] = p

                count += 1

                finish[p] = True

                found = True

            if not found:

                print("System is not in a safe state")

                return False

    print("System is in a safe state.")

    print("Safe sequence is:", *safe_seq)

    return True

if __name__ == "__main__":

    P = 5
```

R = 3

processes = [0, 1, 2, 3, 4]

avail = [3, 3, 2]

maxm = [[7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]]




allot = [[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]]

is_safe(processes, avail, maxm, allot, P, R)

Output Screenshot:

```
31      R = 3
32      processes = [0, 1, 2, 3, 4]
33      avail = [3, 3, 2]
34      maxm = [[7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]]
35      allot = [[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]]
36      is_safe(processes, avail, maxm, allot, P, R)
```

Ln: 36, Col: 49

 Run  Share  Command Line Arguments

```
System is in a safe state.
Safe sequence is: 1 3 4 0 2

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Learning Outcomes: The student should have the ability to:

LO2.1 Outline various compilers for different language

LO2.2 Understood the Banker's Algorithm

LO2.3 Choose an appropriate compiler to solve the Bankers algorithm

Course Outcomes: Upon completion of the course, students will be able to learn about operating systems and security concepts.

Conclusion: The Banker's Algorithm is an effective deadlock avoidance technique used in resource allocation. It ensures that a system does not enter an unsafe state by allocating resources

only if a safe sequence exists. The successful implementation of the algorithm guarantees that all processes complete execution without leading to deadlocks.

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance/ Learning Attitude [20%]	
Marks Obtained				