Operating Systems Lab

Fall 2024-25(L59+60)

Student Name:- Parth Suri

Registration No:- 22BDS0116

Class No. :- VL2024250102445

Faculty Name:- Rahul Srivastava

Deadlock Avoidance using Bankers’ Algorithm

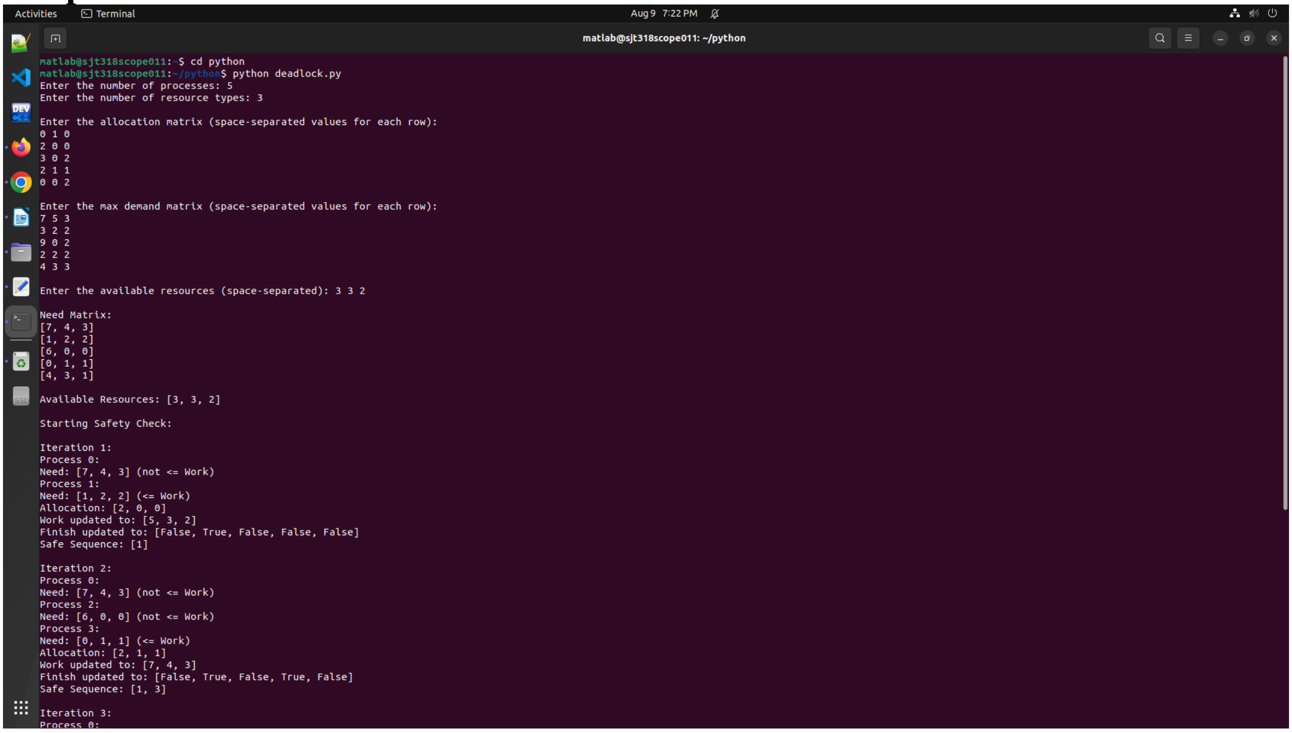
The Banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for the safety of resource allocation by simulating resource allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities before deciding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

Code:-

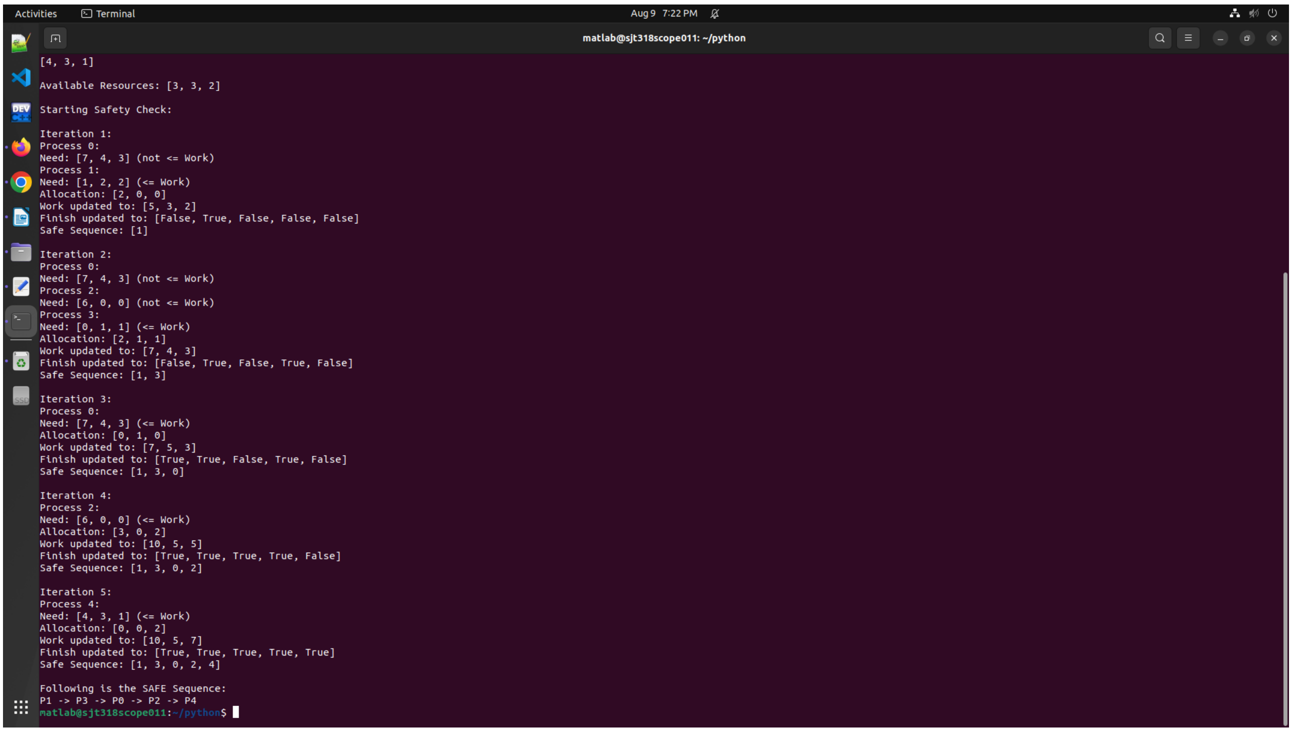
```
import threading

def calculate_need(max_demand, alloc):
    """
    Calculate the Need matrix.
    Need[i][j] is the remaining number of resources of type j needed by process i.
    """
    need = []
    for i in range(len(max_demand)):
        # Need is computed as Max - Allocation
        need.append([max_demand[i][j] - alloc[i][j] for j in 
range(len(max_demand[i]))])
    return need

def is_safe(available, max_demand, alloc, need):
    """
    Check if the system is in a safe state.
    Uses the Banker's Algorithm to determine if there is a safe sequence.
    """
    n = len(alloc)  # Number of processes
    m = len(available)  # Number of resource types
    work = available[:]  # Work is initially the available resources
    finish = [False] * n  # Finish array to track finished processes
    safe_sequence = []  # List to store the safe sequence

    print("\nStarting Safety Check:")

    iteration = 1
    while len(safe_sequence) < n:
        allocated = False
        print(f"\nIteration {iteration}:")
        for i in range(n):
            if not finish[i]:
                print(f"Process {i}:")
                print(f"Need: {need[i]}", end=" ")
                if all(need[i][j] <= work[j] for j in range(m)):
                    # If the need of the process can be satisfied with current work
                    print("(<= Work)")
                    print(f"Allocation: {alloc[i]}")
                    # Update work and finish status
                    for j in range(m):
                        work[j] += alloc[i][j]
                    finish[i] = True
                    safe_sequence.append(i)
                    allocated = True
                    print(f"Work updated to: {work}")
                    print(f"Finish updated to: {finish}")
                    print(f"Safe Sequence: {safe_sequence}")
                    break
                else:
                    print("(not <= Work)")
        if not allocated:
            # If no process could be allocated in this iteration
            print("\nNo process can be allocated at this iteration.")
            return False, []
        iteration += 1
    return True, safe_sequence

if __name__ == "__main__":
    # Input number of processes and resources
    n = int(input("Enter the number of processes: "))
    m = int(input("Enter the number of resource types: "))

    # Input Allocation matrix
    alloc = []
    print("\nEnter the allocation matrix (space-separated values for each row):")
    for i in range(n):
        alloc.append(list(map(int, input().split())))

    # Input Max demand matrix
    max_demand = []
    print("\nEnter the max demand matrix (space-separated values for each row):")
    for i in range(n):
        max_demand.append(list(map(int, input().split())))

    # Input Available resources
    available = list(map(int, input("\nEnter the available resources (space-separated): ").split()))

    # Calculate Need matrix
```

```
need = calculate_need(max_demand, alloc)

# Display Need matrix
print("\nNeed Matrix:")
for row in need:
    print(row)

# Display Available resources
print("\nAvailable Resources:", available)

# Display the safe sequence
safe, safe_sequence = is_safe(available, max_demand, alloc, need)
if safe:
    print("\nFollowing is the SAFE Sequence:")
    for i in safe_sequence:
        print(f"P{i}", end=" -> " if i != safe_sequence[-1] else "\n")
else:
    print("No SAFE Sequence found.")
```

Output:-



```
matlab@sjt318scope011:~$ cd python
matlab@sjt318scope011:~/python$ python deadlock.py
Enter the number of processes: 5
Enter the number of resource types: 3

Enter the allocation matrix (space-separated values for each row):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter the max demand matrix (space-separated values for each row):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter the available resources (space-separated): 3 3 2

Need Matrix:
[7, 4, 3]
[1, 2, 2]
[6, 0, 0]
[0, 1, 1]
[4, 3, 1]

Available Resources: [3, 3, 2]

Starting Safety Check:

Iteration 1:
Process 0:
Need: [7, 4, 3] (not <= Work)
Process 1:
Need: [1, 2, 2] (<= Work)
Allocation: [2, 0, 0]
Work updated to: [5, 3, 2]
Finish updated to: [False, True, False, False, False]
Safe Sequence: [1]

Iteration 2:
Process 0:
Need: [7, 4, 3] (not <= Work)
Process 2:
Need: [6, 0, 0] (not <= Work)
Process 3:
Need: [0, 1, 1] (<= Work)
Allocation: [2, 1, 1]
Work updated to: [7, 4, 3]
Finish updated to: [False, True, False, True, False]
Safe Sequence: [1, 3]

Iteration 3:
Process 0:
```



```
[4, 3, 1]

Available Resources: [3, 3, 2]

Starting Safety Check:

Iteration 1:
Process 0:
Need: [7, 4, 3] (not <= Work)
Process 1:
Need: [1, 2, 2] (<= Work)
Allocation: [2, 0, 0]
Work updated to: [5, 3, 2]
Finish updated to: [False, True, False, False, False]
Safe Sequence: [1]

Iteration 2:
Process 0:
Need: [7, 4, 3] (not <= Work)
Process 2:
Need: [6, 0, 0] (not <= Work)
Process 3:
Need: [0, 1, 1] (<= Work)
Allocation: [2, 1, 1]
Work updated to: [7, 4, 3]
Finish updated to: [False, True, False, True, False]
Safe Sequence: [1, 3]

Iteration 3:
Process 0:
Need: [7, 4, 3] (<= Work)
Allocation: [0, 1, 0]
Work updated to: [7, 5, 3]
Finish updated to: [True, True, False, True, False]
Safe Sequence: [1, 3, 0]

Iteration 4:
Process 2:
Need: [6, 0, 0] (<= Work)
Allocation: [3, 0, 2]
Work updated to: [10, 5, 5]
Finish updated to: [True, True, True, True, False]
Safe Sequence: [1, 3, 0, 2]

Iteration 5:
Process 4:
Need: [4, 3, 1] (<= Work)
Allocation: [0, 0, 2]
Work updated to: [10, 5, 7]
Finish updated to: [True, True, True, True, True]
Safe Sequence: [1, 3, 0, 2, 4]

Following is the SAFE Sequence:
P1 -> P3 -> P0 -> P2 -> P4
matlab@sjt318scope011:~/python$
```