Lab Assignment 7

April 4, 2025

1 Assignment 7

Name: Manobal Singh Bagady
SID: 21104129
Semester: 8th
Branch: Electrical Engineering
Due Date: 04/04/2025

1.1 Banker's Algorithm for Deadlock Avoidance

Concept Overview:

- **Resources and Processes:**
  The system is assumed to have a fixed number of resources and several processes. Each process has a maximum resource requirement (maximum matrix) and a current allocation (allocation matrix).
- **Need Calculation:**
  For each process, the remaining resource requirement (need) is computed as:
  need = maximum − allocation
- **Safety Check:**
  The algorithm checks if the system is in a safe state by trying to find an ordering of processes (a safe sequence) where each process's remaining need can be satisfied by the available resources. When a process completes, it releases its resources, which are added back to the pool of available resources.

Python Implementation:

[8]:
```python
def bankers_algorithm(available, max_matrix, allocation):
    """
    Implements the Banker's algorithm to check if the system is in a safe state.

    Parameters:
      available: A list of available resources.
      max_matrix: A 2D list where each row represents the maximum resources␣
  ↪each process might request.
      allocation: A 2D list where each row represents the resources currently␣
  ↪allocated to each process.
```

```
    Returns:
      (is_safe, safe_sequence) where:
        is_safe is a Boolean indicating if the system is safe,
        safe_sequence is the order in which processes can finish (if safe).
    """
    n_processes = len(max_matrix)
    n_resources = len(available)

    # Calculate the Need matrix
    need = [[max_matrix[i][j] - allocation[i][j] for j in range(n_resources)]
            for i in range(n_processes)]

    safe_seq = []
    finish = [False] * n_processes
    work = available.copy()  # Copy of available resources to simulate resource␣
↪allocation

    while len(safe_seq) < n_processes:
        allocated = False
        for i in range(n_processes):
            if not finish[i]:
                # Check if the process's need can be satisfied by the current␣
↪work (available resources)
                if all(need[i][j] <= work[j] for j in range(n_resources)):
                    # Simulate allocation: process i finishes and releases its␣
↪resources
                    for j in range(n_resources):
                        work[j] += allocation[i][j]
                    safe_seq.append(i)
                    finish[i] = True
                    allocated = True
        if not allocated:
            # No process could be allocated resources, so the system is unsafe.
            break

    is_safe = (len(safe_seq) == n_processes)
    return is_safe, safe_seq

# Example usage:
if __name__ == "__main__":
    # Example resource state for three processes and three resource types:
    available = [0, 1, 1]
    max_matrix = [
        [7, 5, 3],
        [3, 2, 2],
        [9, 0, 2]
```

```python
    ]
    allocation = [
        [3, 0, 2],
        [1, 1, 1],
        [2, 0, 1]
    ]

    print("Example 1:")
    safe, sequence = bankers_algorithm(available, max_matrix, allocation)
    if safe:
        print("System is in a safe state.")
        print("Safe sequence:", sequence)
    else:
        print("System is NOT in a safe state. No safe sequence exists.")

    # Example with a different state:
    available = [2, 3, 2]
    max_matrix = [
        [6, 4, 3],
        [3, 2, 2],
        [4, 2, 2]
    ]
    allocation = [
        [2, 1, 1],
        [1, 1, 1],
        [1, 0, 1]
    ]

    print("\nExample 2:")
    safe, sequence = bankers_algorithm(available, max_matrix, allocation)
    if safe:
        print("System is in a safe state.")
        print("Safe sequence:", sequence)
    else:
        print("System is NOT in a safe state. No safe sequence exists.")
```

```
Example 1:
System is NOT in a safe state. No safe sequence exists.

Example 2:
System is in a safe state.
Safe sequence: [1, 2, 0]
```

1.2 Deadlock Detection Algorithm

Concept Overview:

- **Purpose:**
  Unlike the Banker's algorithm (which avoids deadlock), the deadlock detection algorithm checks whether a deadlock has occurred given the current state of resource allocation.

- **Steps:**

  1. **Compute the Need:**
     Similar to Banker's, calculate need = maximum − allocation.

  2. **Mark Processes with Zero Allocation:**
     Processes that are not holding any resources are marked as finished.

  3. **Simulate Process Completion:**
     Iteratively, for each uncompleted process, if its need can be met by the current available resources, mark it as finished and add its allocated resources back to the pool.

  4. **Identify Deadlocked Processes:**
     Any process that cannot complete by the end of the simulation is considered deadlocked.

Python Implementation:

[10]:
```
def deadlock_detection(available, allocation, max_matrix):
    """
    Implements a deadlock detection algorithm.

    Parameters:
      available: A list of available resources.
      allocation: A 2D list where each row represents the resources currently␣
↪allocated to each process.
      max_matrix: A 2D list where each row represents the maximum resources␣
↪each process may request.

    Returns:
      A list of process indices that are deadlocked.
    """
    n_processes = len(max_matrix)
    n_resources = len(available)

    # Calculate the Need matrix
    need = [[max_matrix[i][j] - allocation[i][j] for j in range(n_resources)]
            for i in range(n_processes)]

    finish = [False] * n_processes
    work = available.copy()

    # Mark processes that have no allocated resources as finished (they cannot␣
↪be deadlocked)
    for i in range(n_processes):
        if all(allocation[i][j] == 0 for j in range(n_resources)):
            finish[i] = True
```

```python
    while True:
        found = False
        for i in range(n_processes):
            if not finish[i]:
                # Check if the process's need can be met by the current␣
 ↪available resources
                if all(need[i][j] <= work[j] for j in range(n_resources)):
                    for j in range(n_resources):
                        work[j] += allocation[i][j]
                    finish[i] = True
                    found = True
        if not found:
            break

    # Processes that are not finished are deadlocked
    deadlocked = [i for i, done in enumerate(finish) if not done]
    return deadlocked

# Example usage:
if __name__ == "__main__":
    # Example resource state for three processes and three resource types:
    available = [1, 5, 2]
    max_matrix = [
        [3, 3, 2],
        [1, 2, 2],
        [1, 3, 3]
    ]
    allocation = [
        [2, 0, 1],
        [0, 1, 0],
        [1, 1, 1]
    ]

    print("Example 1:")
    deadlocked_processes = deadlock_detection(available, allocation, max_matrix)
    if deadlocked_processes:
        print("Deadlock detected in processes:", deadlocked_processes)
    else:
        print("No deadlock detected.")

    # Example with a different state:
    available = [0, 1, 1]
    max_matrix = [
        [7, 5, 3],
        [3, 2, 2],
        [9, 0, 2]
```

```python
    ]
    allocation = [
        [3, 0, 2],
        [1, 1, 1],
        [2, 0, 1]
    ]

    print("\nExample 2:")
    deadlocked_processes = deadlock_detection(available, allocation, max_matrix)
    if deadlocked_processes:
        print("Deadlock detected in processes:", deadlocked_processes)
    else:
        print("No deadlock detected.")
```

```
Example 1:
No deadlock detected.

Example 2:
Deadlock detected in processes: [0, 1, 2]
```