TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Lab Report Of Distributed System On Banker's Algorithm

Submitted By:

Raj Kumar Dhakal

THA076BCT033

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

August, 2023

TITLE: IMPLEMENTATION OF BANKER'S ALGORITHM FOR AVOIDING DEDLOCK

THEORY

BANKER’S ALGORITHM

The Banker's Algorithm is an algorithm used in operating systems for resource allocation and deadlock avoidance. It was created by Edsger W. Dijkstra in 1965. The main objective of this algorithm is to efficiently allocate resources to multiple processes in a manner that guarantees the system remains in a secure state and prevents deadlock.

Here are some key concepts associated with the Banker's Algorithm:

1. **Resources:** These are essential elements in a computer system that processes require to complete their tasks. Examples of resources include CPU time, memory, files, and I/O devices.
2. **Available Resources:** This refers to the number of resources currently accessible and ready to be assigned to processes.
3. **Maximum Need:** Each process has a maximum requirement for resources during its execution, indicating the maximum amount of each resource type it may need.
4. **Allocated Resources**: These are the resources that have already been assigned to each individual process.
5. **Need**: The "Need" of a process represents the difference between its maximum requirement and the resources currently allocated to it.

DEADLOCK

Deadlock is a critical issue that can occur in a computer system when multiple processes are unable to proceed because each process is waiting for a resource that is being held by another process in the system. This creates a circular dependency where no process can release the resources it holds, leading to a standstill or complete system freeze.

To address the deadlock problem, the Banker's Algorithm was developed. It is a resource allocation and deadlock avoidance algorithm used in operating systems. The primary purpose of the Banker's Algorithm is to manage the allocation of resources to multiple processes in a way that ensures the system remains in a safe state and avoids deadlock.

STEPS TO IMPLEMENT BANKER’S ALGORITHM

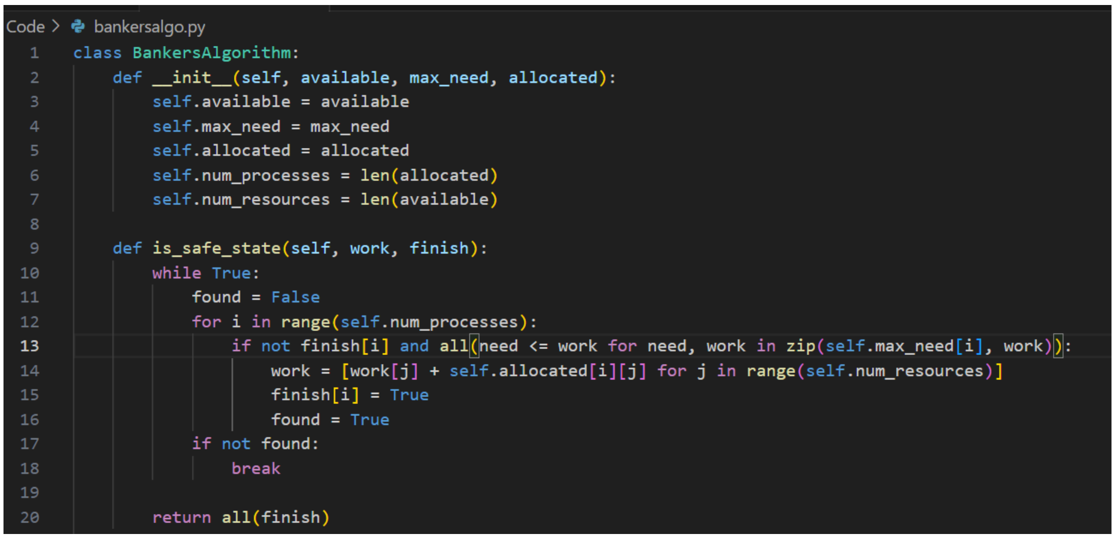
The Banker's Algorithm ensures that resource allocations are managed in a way that avoids deadlock and maintains a safe state in the system, allowing processes to execute efficiently without getting stuck due to resource contention.

1. **Initialization**
   Gather information about the available resources and the maximum need of each process. This includes the total number of instances of each resource type, the current available resources, and the maximum resource need for each process.
2. **Request Phase**
   When a process requests additional resources, check if the request can be granted without causing the system to enter an unsafe state (i.e., causing deadlock). Compare the requested resources with the available resources and the maximum need of the process. If the request is feasible, temporarily allocate the resources to the process.
3. **Safety Check**
   After the resource allocation in the Request Phase, perform a safety check to determine if the system remains in a safe state. Simulate the resource allocation to all processes (including the process that made the request) until all processes can complete their execution without deadlock.
4. **Execution or Block**
   If the safety check in the Safety Check Phase indicates that the system remains in a safe state, the requested resources are officially allocated to the process, and the process can proceed with its execution. Otherwise, the process must wait until enough resources become available to ensure a safe state can be achieved.

5. **Release Phase**

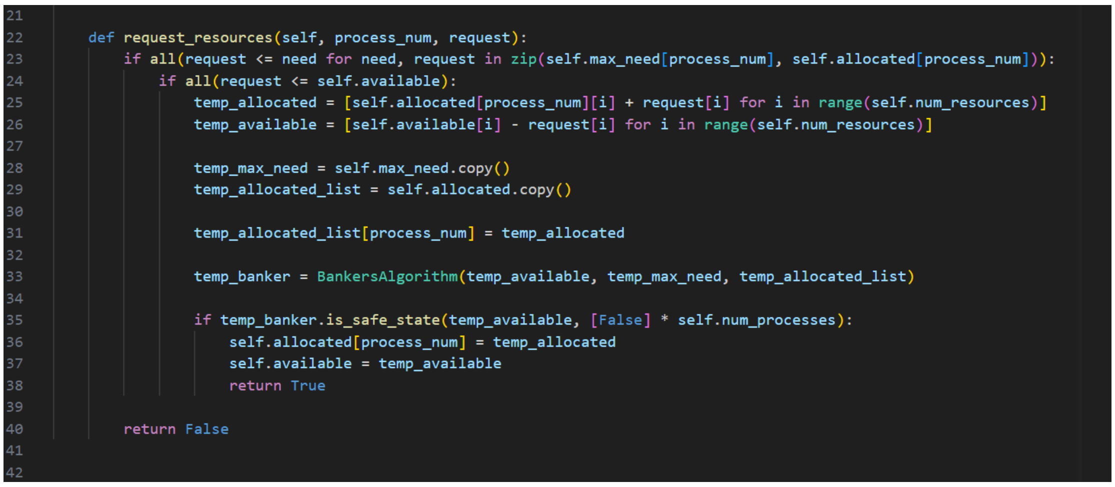
   When a process completes its execution or voluntarily releases resources, update the available resources and allocated resources accordingly. This makes the released resources available for allocation to other processes.

<u>**CODE**</u>

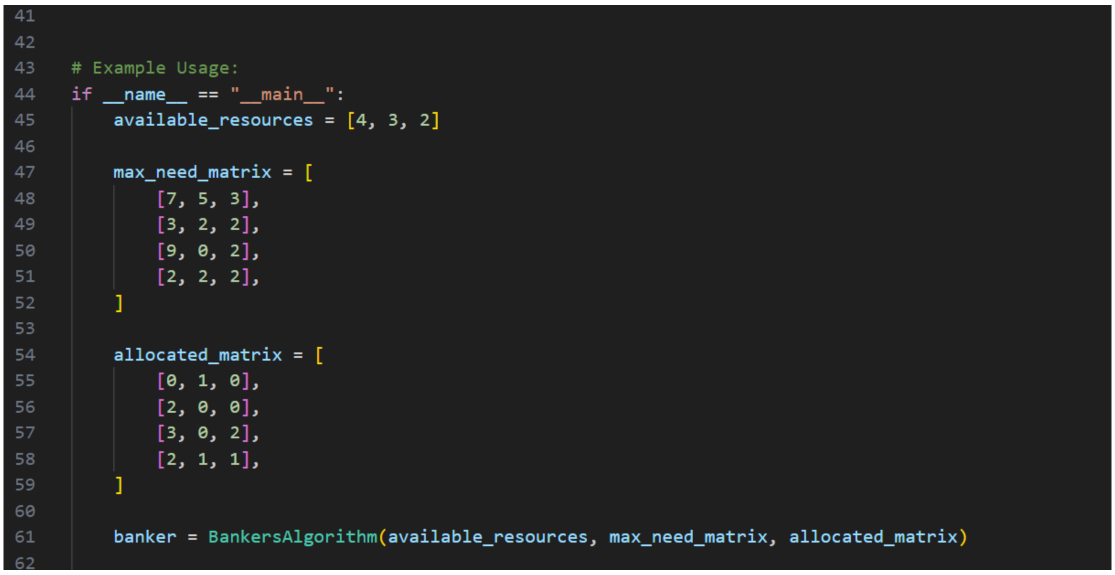


Code > bankersalgo.py

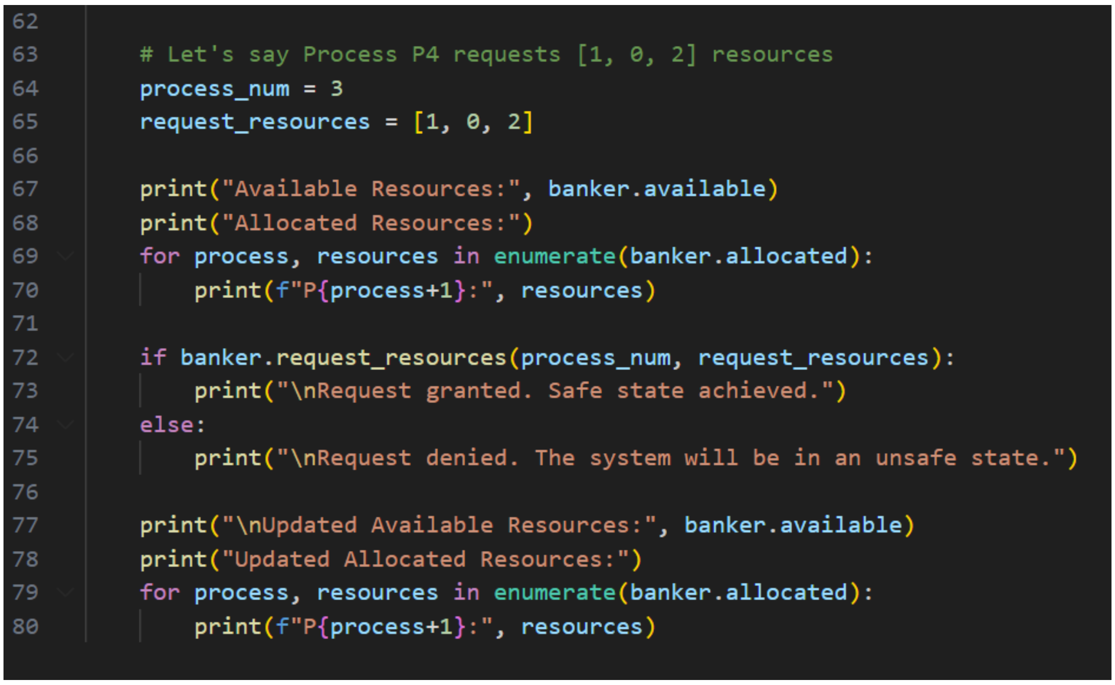
```python
class BankersAlgorithm:
    def __init__(self, available, max_need, allocated):
        self.available = available
        self.max_need = max_need
        self.allocated = allocated
        self.num_processes = len(allocated)
        self.num_resources = len(available)

    def is_safe_state(self, work, finish):
        while True:
            found = False
            for i in range(self.num_processes):
                if not finish[i] and all(need <= work for need, work in zip(self.max_need[i], work)):
                    work = [work[j] + self.allocated[i][j] for j in range(self.num_resources)]
                    finish[i] = True
                    found = True
            if not found:
                break

        return all(finish)
```



```python

    def request_resources(self, process_num, request):
        if all(request <= need for need, request in zip(self.max_need[process_num], self.allocated[process_num])):
            if all(request <= self.available):
                temp_allocated = [self.allocated[process_num][i] + request[i] for i in range(self.num_resources)]
                temp_available = [self.available[i] - request[i] for i in range(self.num_resources)]

                temp_max_need = self.max_need.copy()
                temp_allocated_list = self.allocated.copy()

                temp_allocated_list[process_num] = temp_allocated

                temp_banker = BankersAlgorithm(temp_available, temp_max_need, temp_allocated_list)

                if temp_banker.is_safe_state(temp_available, [False] * self.num_processes):
                    self.allocated[process_num] = temp_allocated
                    self.available = temp_available
                    return True

        return False


```

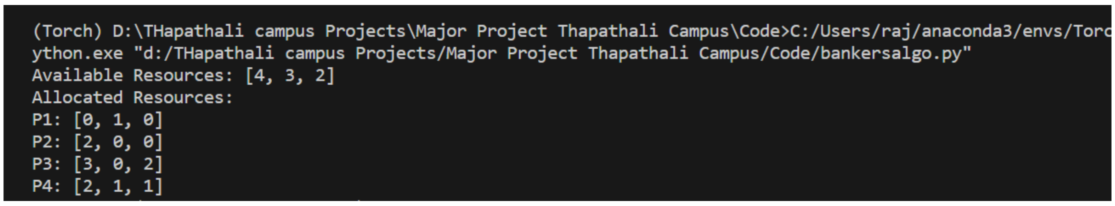


```python


# Example Usage:
if __name__ == "__main__":
    available_resources = [4, 3, 2]

    max_need_matrix = [
        [7, 5, 3],
        [3, 2, 2],
        [9, 0, 2],
        [2, 2, 2],
    ]

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

    banker = BankersAlgorithm(available_resources, max_need_matrix, allocated_matrix)
```



```python

    # Let's say Process P4 requests [1, 0, 2] resources
    process_num = 3
    request_resources = [1, 0, 2]

    print("Available Resources:", banker.available)
    print("Allocated Resources:")
    for process, resources in enumerate(banker.allocated):
        print(f"P{process+1}:", resources)

    if banker.request_resources(process_num, request_resources):
        print("\nRequest granted. Safe state achieved.")
    else:
        print("\nRequest denied. The system will be in an unsafe state.")

    print("\nUpdated Available Resources:", banker.available)
    print("Updated Allocated Resources:")
    for process, resources in enumerate(banker.allocated):
        print(f"P{process+1}:", resources)
```

OUTPUT



```
(Torch) D:\THapathali campus Projects\Major Project Thapathali Campus\Code>C:/Users/raj/anaconda3/envs/Torc
ython.exe "d:/THapathali campus Projects/Major Project Thapathali Campus/Code/bankersalgo.py"
Available Resources: [4, 3, 2]
Allocated Resources:
P1: [0, 1, 0]
P2: [2, 0, 0]
P3: [3, 0, 2]
P4: [2, 1, 1]
```

CONCLUSION

In conclusion, the Banker's Algorithm is a crucial technique for efficiently managing resource allocation and preventing deadlock in sophisticated operating systems. Its dynamic safety-checking approach before granting resource requests ensures system stability and prevents processes from getting stuck. The algorithm's utilization of arrays and data structures simplifies resource tracking and management, making it a valuable tool for multi-process environments. By implementing the Banker's Algorithm, we can enhance the efficiency of resource utilization and ensure smooth execution of processes while avoiding potential deadlock scenarios.