DIVYANSHU MAKHARIA
221070036

ASSIGNMENT 6

AIM:

Detection of deadlock

THEORY:

Deadlock detection Algorithm is as follows:

1. **Build a RAG** − The first step is to build a Resource Allocation Graph (RAG) that shows the allocation and request of resources in the system. Each resource type is represented by a rectangle, and each process is represented by a circle.
2. **Check for cycles** − Look for cycles in the RAG. If there is a cycle, it indicates that the system is deadlocked.
3. **Identify deadlocked processes** − Identify the processes involved in the cycle. These processes are deadlocked and waiting for resources held by other processes.
4. **Determine resource types** − Determine the resource types involved in the deadlock, as well as the resources held and requested by each process.
5. **Take corrective action** − Take corrective action to break the deadlock by releasing resources, aborting processes, or preempting resources. Once the deadlock is broken, the system can continue with normal operations.
6. **Recheck for cycles** − After corrective action has been taken, recheck the RAG for cycles. If there are no more cycles, the

system is no longer deadlocked, and normal operations can resume.

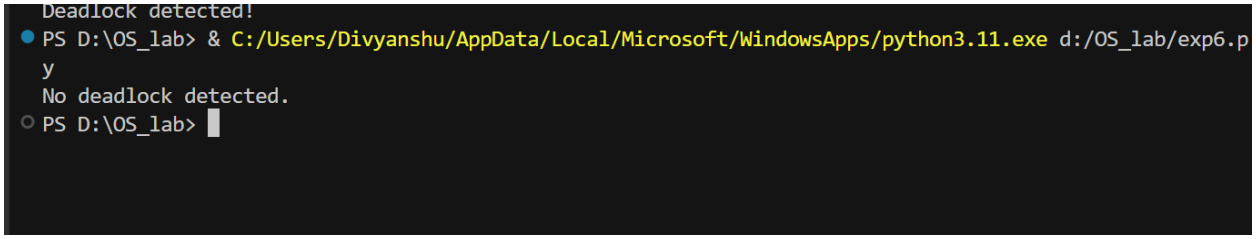
CODE:

```
def add_edge(graph, u, v):
    if u not in graph:
        graph[u] = []
    graph[u].append(v)


def is_cyclic_util(graph, v, visited, marked):
    visited[v] = True
    marked[v] = True

    if v in graph:
        for neighbor in graph[v]:
            if not visited[neighbor]:
                if is_cyclic_util(graph, neighbor,
visited, marked):
                    return True
            elif marked[neighbor]:
                return True

    marked[v] = False
    return False


def is_cyclic(graph, vertices):
    visited = {v: False for v in vertices}
    marked = {v: False for v in vertices}
```

```
    for node in vertices:
        if not visited[node]:
            if is_cyclic_util(graph, node, visited,
marked):
                return True
    return False


def detect_deadlock(processes, resources, allocations,
requests):
    graph = {}

    # Add edges for resource allocation
    for p, r in allocations:
        add_edge(graph, p, len(processes) + r)

    # Add edges for resource requests
    for p, r in requests:
        add_edge(graph, len(processes) + r, p)

    vertices = list(range(len(processes) +
len(resources)))

    if is_cyclic(graph, vertices):
        print("Deadlock detected!")
    else:
        print("No deadlock detected.")


# Example
```

```
if __name__ == "__main__":
    processes = [0, 1, 2,3]
    resources = [0, 1, 2,3]
    allocations = [(0, 0), (1, 1), (2, 2),(3,3)]
    requests = [(0, 3), (1, 2), (2, 0)]

    detect_deadlock(processes, resources, allocations,
requests)
```

OUTPUT:



CONCLUSION:

Thus,we have successfully learnt about Deadlock Detection Algorithm.Deadlock detection algorithms are used to identify the presence of deadlocks in computer systems. These algorithms examine the system's processes and resources to determine if there is a circular wait situation that could lead to a deadlock.