Assignment-5

Assumptions:

1. The resources are available globally to all the processes of every site.
2. We have maintained two status table for each site; one showing the resources being held by the respective processes and the other showing the resources the processes are requesting respectively.
3. We have represented the wait for graph using a list.

Source Code:

```
# This represents a site. It can contain its own processes and resources.
class Site:
    def __init__(self, resource: int):
        self.process_count: int = 0
        self.resource_count: int = resource
        self.process_holding_resource: "list[list[int]]" = []
        self.process_waiting_resource: "list[list[int]]" = []

    def setupStatusTables(self):
        # Take number of processes in this site as input from user
        try:
            self.process_count = int(input("Enter Number of Processes: "))
        except:
            print("Invalid Input")
            exit()

        # Take the process_holding_resource table from user
        print("Enter the Matrix representing Processes holding Resources:")
        for _ in range(self.process_count):
            try:
                temp_row: "list[int]" = list(map(int, input().split(" ")))
            except:
                print("Invalid Input")
                exit()
            # The length of each row should be equal to the total number of
resources in the site
            if(len(temp_row) != self.resource_count):
                print("Incorrect Number of Values Provided")
                exit()
            # All the values in the row should be either a 0 or a 1
            for value in temp_row:
                if(value != 0 and value != 1):
                    print("Invalid Input")
                    exit()
            # If all tests passed, add the row to the list
            self.process_holding_resource.append(temp_row)

        # Take the process_waiting_resource table from user
```

```
        print("Enter the Matrix representing Processes waiting for
Resources:")
        for _ in range(self.process_count):
            try:
                temp_row: "list[int]" = list(map(int, input().split(" ")))
            except:
                print("Invalid Input")
                exit()
            # The length of each row should be equal to the total number of
resources in the site
            if(len(temp_row) != self.resource_count):
                print("Incorrect Number of Values Provided")
                exit()
            # All the values in the row should be either a 0 or a 1
            for value in temp_row:
                if(value != 0 and value != 1):
                    print("Invalid Input")
                    exit()
            # If all tests passed, add the row to the list
            self.process_waiting_resource.append(temp_row)


# Represents a single node in the graph. This node can be a process or a
resource in any of the sites.
class Node:
    def __init__(self, type: str, site_id: int, type_id: int):
        self.type: str = type
        self.site_id: int = site_id
        self.type_id: int = type_id
        self.next: int = None


# Apart from setting up the program and maintaining the global number of
resources (in this program for simplicity), the main block also
# acts as the centralized controller which generates the wait-for-graph and
performs calculations for detecting deadlock in the system.
if(__name__ == "__main__"):
    # Get the number of sites from the user
    try:
        sites = int(input("Enter Number of Sites: "))
        resources = int(input("Enter Number of Resources in all the Sites
combined: "))
    except:
        print("Incorrect Input")
        exit()

    # List contains instances of all the sites
    sites_list: "list[Site]" = []
    # Create the sites
```

```
    for index in range(sites):
        print(f"Site {index + 1}:")
        new_site: Site = Site(resources)
        # Setup the status tables in the corresponding site
        new_site.setupStatusTables()
        # Add the newly created site to the sites list
        sites_list.append(new_site)

    # # Print the status tables per each site
    # for index, site in enumerate(sites_list):
    #     print(f"Site {index}:")
    #     site.showStatusTables()

    # Construct the wait-for-graph. Here, all the processes and resources are
represented as nodes in the graph.
    # Here, it is stored as a list where each index represents a node in the
graph. The first portion of the list
    # contains all the processes and the second part of the list contains all
the resources in all the sites.
    # Each node points to some other node, representing a directed edge
between the two.

    graph: "list[Node]" = []

    # First, get all the processes in all the sites and add them as nodes to
the graph
    # In this stage, no edges are added. Just the ndoes are added.
    processes: int = 0
    for index, site in enumerate(sites_list):
        processes += site.process_count
        for process in range(site.process_count):
            new_node = Node("P", index, process)
            graph.append(new_node)

    # Now, add the resources to the graph. Again, no edges are added and only
the nodes are added to the graph.
    # Also, the site_id for the resources are set to -1 because as of the
current implementation of this program,
    # we are not tracking which resources are present in which site for
simplicity. It is assumed that all the
    # sites have information about all the resources in the entire system.
    for index in range(resources):
        new_node = Node("R", -1, index)
        graph.append(new_node)

    # Setup the edges within the graph
    elapsed_processes: int = 0
    for site in sites_list:
```

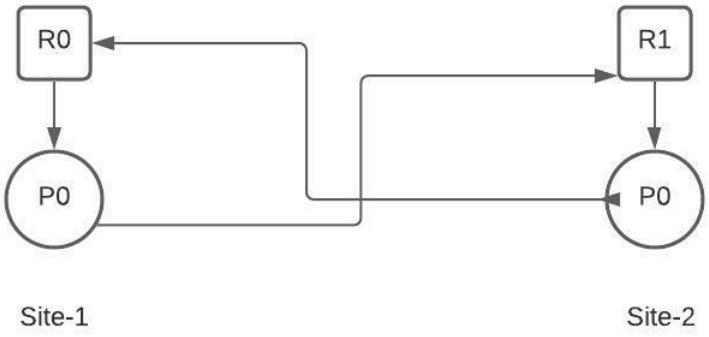
```
        # Directed edges from resources to processes indicating these
resources are taken be the
        # corresponding processes
        for row_index, row_value in enumerate(site.process_holding_resource):
            for resource_index, resource_value in enumerate(row_value):
                if(resource_value == 1):
                    graph[processes + resource_index].next = elapsed_processes
+ row_index

        # Directed edges from processes to resources indicating these
processes are waiting
        # for the corresponding resources
        for row_index, row_value in enumerate(site.process_waiting_resource):
            for resource_index, resource_value in enumerate(row_value):
                if(resource_value == 1):
                    graph[elapsed_processes + row_index].next = processes +
resource_index

        elapsed_processes += site.process_count

    # Detect cycle within the network by starting from every node and checking
whether loop comes back to the same node (index) or not
    deadlock_detected = False
    for track_index in range(len(graph)):
        index = track_index
        # We iterate over the diameter of the graph because that is the
largest deadlock cycle possible.
        # This stops the program from entering an infinite loop if the
starting node is not in deadlock but there is a deadlock
        # among the processes it iterates over.
        for _ in range(len(graph)):
            if(graph[index].next == None):
                break
            index = graph[index].next
            if(graph[index].next == track_index):
                deadlock_detected = True
                break
        if(deadlock_detected == True):
            print("Deadlock Detected.")
            print("Deadlocked Nodes :- ")
            index = track_index
            while(True):
                if(graph[index].type == "P"):
                    print(f"Process {graph[index].type}{graph[index].type_id}
at Site S{graph[index].site_id+1}")
                index = graph[index].next
                if(index == track_index):
                    break
```

```
            break
    else:
        print("No Deadlock Detected")
```

Dataset used:

Set 1:

We have taken 2 sites:



Status table-1 for site-1:

(Resources holding)

	R0	R1
P0	1	0

Status table-1 for site-2:

(Resources holding)

	R0	R1
P0	0	1

Status table-2 for site-1:

(Resources Requesting)

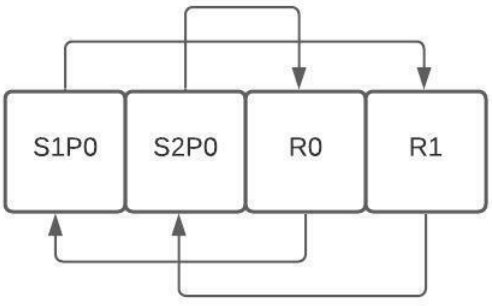
	R0	R1
P0	0	1

Status table-2 for site-2:

(Resources Requesting)

	R0	R1
P0	1	0

Wait for graph:

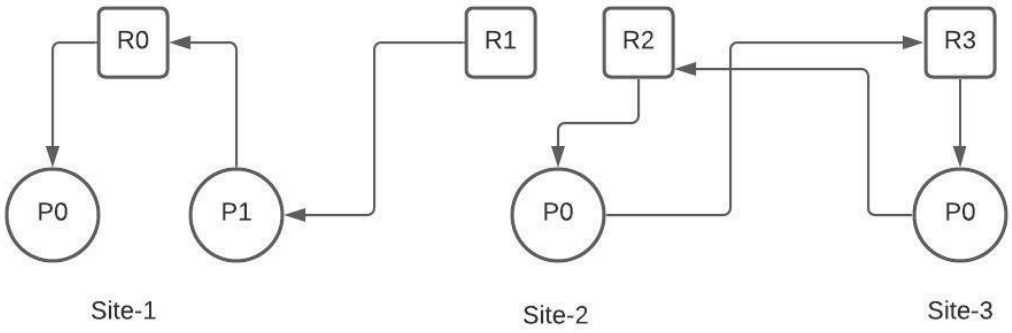


Input-Output:

```
PS C:\Users\VAISHALI MONDAL\Desktop\Calcutta University\Sem-3\Distributed Operating System> python Ho-Ramamurthy.py
Enter Number of Sites: 2
Enter Number of Resources in all the Sites combined: 2
Site 1:
Enter Number of Processes: 1
Enter the Matrix representing Processes holding Resources:
1 0
Enter the Matrix representing Processes waiting for Resources:
0 1
Site 2:
Enter Number of Processes: 1
Enter the Matrix representing Processes holding Resources:
0 1
Enter the Matrix representing Processes waiting for Resources:
1 0
Deadlock Detected.
Deadlocked Nodes :-
Process P0 at Site S1
Process P0 at Site S2
PS C:\Users\VAISHALI MONDAL\Desktop\Calcutta University\Sem-3\Distributed Operating System>
```

Set-2:

We have taken 3 sites:



Status table-1 for site-1:

(Resources holding)

	R0	R1	R2	R3
P0	1	0	0	0
P1	0	1	0	0

Status table-2 for site-1:
(Resources requesting)

	R0	R1	R2	R3
P0	0	0	0	0
P1	1	0	0	0

Status table-1 for site-2:

(Resources holding)

	R0	R1	R2	R3
P0	0	0	1	0

Status table-2 for site-2:

(Resources requesting)

	R0	R1	R2	R3
P0	0	0	0	1

Status table-1 for site-3:

(Resources holding)

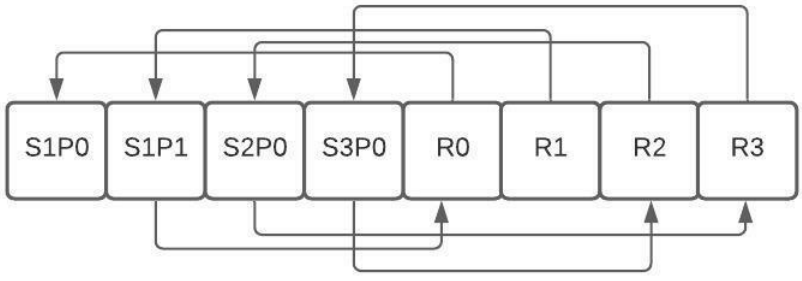
	R0	R1	R2	R3
P0	0	0	0	1

Status table-2 for site-3:

(Resources requesting)

	R0	R1	R2	R3
P0	0	0	1	0

Wait for graph:



Input-Output:

```
PS C:\Users\VAISHALI MONDAL\Desktop\Calcutta University\Sem-3\Distributed Operating System> python Ho-Ramamurthy.py
Enter Number of Sites: 3
Enter Number of Resources in all the Sites combined: 4
Site 1:
Enter Number of Processes: 2
Enter the Matrix representing Processes holding Resources:
1 0 0 0
0 1 0 0
Enter the Matrix representing Processes waiting for Resources:
0 0 0 0
1 0 0 0
Site 2:
Enter Number of Processes: 1
Enter the Matrix representing Processes holding Resources:
0 0 1 0
Enter the Matrix representing Processes waiting for Resources:
0 0 0 1
Site 3:
Enter Number of Processes: 1
Enter the Matrix representing Processes holding Resources:
0 0 0 1
Enter the Matrix representing Processes waiting for Resources:
0 0 1 0
Deadlock Detected.
Deadlocked Nodes :-
Process P0 at Site S1
Process P0 at Site S2
PS C:\Users\VAISHALI MONDAL\Desktop\Calcutta University\Sem-3\Distributed Operating System>
```