221070055
Raj Sapale

EXPERIMENT NO. 06

Aim:

Implement an algorithm to detect the deadlock in given input resource allocation graph.

Theory:

The provided code implements the Banker's algorithm, a deadlock avoidance algorithm used in resource allocation systems. Deadlock occurs in resource allocation systems when processes wait indefinitely for resources held by other processes, resulting in a state where no process can proceed. The Banker's algorithm prevents deadlock by ensuring that resource allocations are made in a safe manner, such that the system can always recover to a safe state even if all processes request their maximum resources simultaneously.

The algorithm works by simulating resource allocation and checking if granting a resource request would leave the system in a safe state. It maintains information about the resources currently allocated to processes, the maximum resources each process may need, and the available resources. It then iterates through the processes, attempting to allocate resources to each process in a way that avoids deadlock. If a safe allocation sequence is found, the system proceeds accordingly. Otherwise, if deadlock is detected, appropriate action can be taken to resolve the deadlock situation.

Code:

```
def user_input():

p = int(input("Enter number of processes: "))
r = int(input("Enter number of resources: "))

alloc = []
```

```
max_claim = []
avail = []
for i in range(p):
temp1 = []
temp2 = []
for j in range(r):
x = int(input(f"Enter resource {j+1} allocated for process {i+1}: "))
y = int(input(f"Enter resource {j+1} max for process {i+1}: "))
temp1.append(x)
temp2.append(y)
alloc.append(temp1)
max_claim.append(temp2)

for i in range(r):
avail.append(int(input(f"Enter available resource {i+1}: ")))
return alloc, max_claim, avail, p, r



def detect_deadlock(alloc, max_claim, avail, p, r):
f = [0] * p
for k in range(p):
f[k] = 0
need = [[0 for i in range(r)] for i in range(p)]
for i in range(p):
for j in range(r):
need[i][j] = max_claim[i][j] - alloc[i][j]
deadlock_detected = False
for k in range(p):
for i in range(p):
if f[i] == 0:
flag = 0
for j in range(r):
if need[i][j] > avail[j]:
flag = 1
break
if flag == 0:
for y in range(r):
avail[y] += alloc[i][y]
f[i] = 1

if all(f):
break

if any(f) and not all(f):
deadlock_detected = True
```

```
    break

return deadlock_detected

def main():
choice = input("User Inputs(Y/n): ")
if choice.lower() == 'y':
alloc, max_claim, avail, p, r = user_input()
else:
alloc = [[1, 0, 2, 1],
[0, 1, 1, 0],
[1, 1, 0, 1]]
max_claim = [[2, 2, 3, 2],
[1, 2, 2, 1],
[2, 2, 1, 2]]
avail = [1, 2, 1, 1]
p = 3
r = 4

if detect_deadlock(alloc, max_claim, avail, p, r):
print("Deadlock detected!")
else:
print("No Deadlock detected")


if __name__ == "__main__":
main()
```

Output:

```
raj@raj-IdeaPad-5-14ALC05:~/Desktop/Assignments/sy/os-lab$ /bin/python
3 /home/raj/Desktop/Assignments/sy/os-lab/deadlock.py
User Inputs(Y/n): n
No Deadlock detected
raj@raj-IdeaPad-5-14ALC05:~/Desktop/Assignments/sy/os-lab$ /bin/python
3 /home/raj/Desktop/Assignments/sy/os-lab/deadlock.py
User Inputs(Y/n): y
Enter number of processes: 3
Enter number of resources: 4
Enter resource 1 allocated for process 1: 1
Enter resource 1 max for process 1: 2
Enter resource 2 allocated for process 1: 0
Enter resource 2 max for process 1: 2
Enter resource 3 allocated for process 1: 2
Enter resource 3 max for process 1: 3
Enter resource 4 allocated for process 1: 1
Enter resource 4 max for process 1: 2
Enter resource 1 allocated for process 2: 0
Enter resource 1 max for process 2: 2
Enter resource 2 allocated for process 2: 1
Enter resource 2 max for process 2: 2
Enter resource 3 allocated for process 2: 1
Enter resource 3 max for process 2: 2
Enter resource 4 allocated for process 2: 0
Enter resource 4 max for process 2: 1
Enter resource 1 allocated for process 3: 1
Enter resource 1 max for process 3: 2
Enter resource 2 allocated for process 3: 1
Enter resource 2 max for process 3: 2
Enter resource 3 allocated for process 3: 0
Enter resource 3 max for process 3: 1
Enter resource 4 allocated for process 3: 1
Enter resource 4 max for process 3: 2
Enter available resource 1: 1
Enter available resource 2: 2
Enter available resource 3: 1
Enter available resource 4: 2
No Deadlock detected
```

Conclusion:

The provided code offers a solution to detect deadlock using the Banker's algorithm. It allows users to input their own resource allocation graph and available resources or use a predefined example.