22BAI10288

AROMAL.S.K

14th May 2024

O.S. Lab Practical

(*) Banker's Algorithm (.py).

code:

```
def is_safe(processes, allocation, max_res, available):
    need = [[max_res[i][j] - allocation[i][j]
             for j in range(len(available))]
            for i in range(processes)]
    finish = [False] * processes
    work = available.copy()
    Safe seq = []

    while any(not f for f in finish):
        found = False
```

```
for i in range(processes):
    if (not finish[i] and all(need[i][j]
        <= work[j] for j in range(len(available))):
            for j in range(len(available)):
                work[j] += allocation[i][j]
            finish[i] = True
            safe_seq.append(i)
            found = True
    if not found:
        return None
return safe_seq
```

example

Processes = 5

resources = 3

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

max_res = [[7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 2]]

available = [3, 3, 2]

safe_seq = is_safe (processes, allocations, max_res, available).

if safe_seq :

    print("Following is the SAFE Sequence :")
    for processes in safe_seq :
        Print("P", process, end = " → ")
    Print ("P", safe_seq [-1])

else :
    print("The following system is not safe")

Output :

Following is the safe sequence :

P1 → P3 → P4 → P0 → P2 → P2.

Code :

```
def is_safe(processes, allocation, max_resources, available):

  need = [[max_resources[i][j] - allocation[i][j] for j in range(len(available))]
for i in range(processes)]
  finish = [False] * processes
  work = available.copy()
  safe_seq = []

  while any(not f for f in finish):
    found = False
    for i in range(processes):
      if (not finish[i]) and all(need[i][j] <= work[j] for j in
range(len(available))):
        for j in range(len(available)):
          work[j] += allocation[i][j]
        finish[i] = True
        safe_seq.append(i)
        found = True
    if not found:
      return None

  return safe_seq

# Example usage
processes = 5
resources = 3
allocation = [
  [0, 1, 0],
  [2, 0, 0],
  [3, 0, 2],
]
max_resources = [
  [7, 5, 3],
  [3, 2, 2],
  [9, 0, 2],
  [2, 2, 2],
  [4, 3, 2]
]
available = [3, 3, 2]

safe_sequence = is_safe(processes, allocation, max_resources, available)

if safe_sequence:
  print("Following is the SAFE Sequence:")
  for process in safe_sequence:
    print(" P", process, end=" -> ")
  print("P", safe_sequence[-1])
else:
  print("The following system is not safe")
```

Output :

```
PS C:\Users\skaro> & C:/Users/skaro/AppData/Local/Microsoft/WindowsApps/python3.11.exe f:/Coding/Codes/Python/Bankers.py
Following is the SAFE Sequence:
 P 1 ->  P 3 ->  P 4 ->  P 0 ->  P 2 -> P 2
PS C:\Users\skaro>
```

(*) Resource Allocation graph...

```
def check_deadlock(processes, res, allo, req):
    st_req = [[0]*res for _ in range(processes)]
    st_pro = [[0]*res for _ in range(processes)]

    while True:
        found = False
        for i in range(processes):
            if all(allo[i][j]>0 for j in range(res)):
                for j in range(res):
                    if req[i][j]==1:
                        st_req[i][j]+=1
                        if st_req[i][j]>1 and flag==1:
                            return True
                        found = True
                        break
                if found:
                    st_pro[i] = allocation[i].copy()
                    flag = 1
                    break
        if not found:
            break
```

return False.

#Example:

```
Processes = 3
res = 3
allo = [[1,0,0], [2,1,1], [0,1,2]]
req = [[0,1,0], [0,1,0], [2,0,1]]
if check_deadlock(processes, res, allo, req):
        print("Deadlock Detected")
else:
        print("NO Deadlock Detected")
```

Output:

Deadlock Detected.

Code :

```
def check_deadlock(processes, resources, allocation, request):


    st_req = [[0] * resources for _ in range(processes)]  # Request status matrix
    st_pro = [[0] * resources for _ in range(processes)]  # Process status matrix
    flag = 0

    while True:
        found = False
        for i in range(processes):
            if all(allocation[i][j] > 0 for j in range(resources)):
                for j in range(resources):
                    if request[i][j] == 1:
                        st_req[i][j] += 1
                        if st_req[i][j] > 1 and flag == 1:
                            return True  # Deadlock detected
                        found = True
                        break
                if found:
                    st_pro[i] = allocation[i].copy()
                    flag = 1
                    break

        if not found:
            break

    return False  # No deadlock detected

# Example usage (assuming you have the allocation and request matrices)
processes = 3
resources = 3
allocation = [
    [1, 0, 0],
    [2, 1, 1],
    [0, 1, 2]
]
request = [
    [0, 1, 0],
    [0, 1, 0],
    [2, 0, 1]
]

if check_deadlock(processes, resources, allocation, request):
    print("Deadlock detected")
else:
    print("No deadlock detected")
```

Output :

```
PS C:\Users\skaro> & C:/Users/skaro/AppData/Local/Microsoft/WindowsApps/python3.11.exe f:/Coding/Codes/Python/RAG.py
Deadlock detected
PS C:\Users\skaro>
```