MAKERERE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES (COCIS)

SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY

GROUP H

NAME	REGISTRATION NUMBER	STUDENT NUMBER
NALUBEGA SHADIAH	24/U/08715/EVE	2400708715
MUGOLE JOEL	24/U/07060/EVE	2400707060
NANSWA PATRICIA	24/U/27188/EVE	2400727188
SUUBI BAKER	24/U/26049/EVE	2400726049
NAKITTO ROSEMARY	24/U/26589/EVE	2400726589
AMANYIRE CINDY	23/U/05985/EVE	2300705985

Bankers Algorithm

With reference to the example in class;

5 processes P_0 through P_4;

3 resource types:

A (10 instances), B (5instances), and C (7 instances)

Snapshot at time T_0:

Processes	Allocation	Max	Available	Need
	ABC	ABC	ABC	ABC
P_0	010	753	332	743
P_1	200	322		122
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431
Total	725			

Available = Available instances - Total Allocation

A = 10 – 7 = 3

B = 5 - 2 = 3

C = 7 – 5 = 2

Available = 332

Need = Max – Allocation

P_0 = 753 – 010 = 743

A = 7 – 0 = 7

B = 5 – 1 = 4

C = 3 – 0 = 3

P_1 = 322 – 200 =122

A = 3-2 = 1

$B = 2-0 = 2$

$C = 2 – 0 =2$

$P_2 = 902 – 302 = 600$

$A = 9-3 = 6$

$B = 0-0 = 0$

$C = 2-2 = 0$

$P_3 = 222 – 211= 011$

$A = 2-2 – 0$

$B = 2-1 = 1$

$C = 2-1 = 1$

$P_4 = 433 – 002 = 431$

$A = 4-0 = 4$

$B= 3-0 =3$

$C=3-2 = 1$

Safe Algorithm

With reference to the example in class;

5 processes P_0 through P_4;

3 resource types:

A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time T_0:

Processes	Allocation	Max	Available	Need
	ABC	ABC	ABC	ABC
P_0	010	753	332	743
P_1	200	322		122
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431
Total	725			

Rules;

I. Work = Available where finish (i) = False, for (i = 0,1, ...n-1)
II. Need <= Work
III. If finish = (True) Work = Work + Allocation

P_0

for Need<= Work of instances ABC

743 <= 332: Finish [0] = F

P_1

for Need<= Work of instances ABC

122 <= 322 : Finish[1] = T

If true we implement work

Work = Work + Allocation

332+200

New Work at P_1 = 532

P_2

for Need<= Work of instances ABC

600 <= 532: Finish [2] = F

Work cannot be implemented

P_3

for Need<= Work of instances ABC

011 <= 532: Finish [3] = T

If true we implement work

Work = Work + Allocation

532+211

New Work at P_3 = 743

P_4

for Need<= Work of instances ABC

431 <= 743: Finish [4] =T

If true we implement work

Work = Work + Allocation

743+002

New Work at P_4 = 745

Checking for P_0 again

P_0

for Need<= Work of instances ABC

743 <= 745: Finish [0] = T

If true we implement work

Work = Work + Allocation

745+010

New Work at P_0 = 755

Checking P_2 again

for Need<= Work of instances ABC

600 <= 755: Finish [2] = T

If true we implement work

Work = Work + Allocation

755+302

New Work at P_2 = 10 5 7

Therefore, the Safe sequence is < **P_1, P_3, P_4, P_0, P_2** >

Deadlock Detection Algorithm

Uses Allocation, Requests and Available where;

Available is a vector m indicates the number of available resources of each type.

Allocation is the matrix n * m that defines the number of resources of each type currently allocated to each process.

Request is a matrix n * m that indicates the current request of each process.

Example

5 processes P_0 through P_4;

3 resource types:

A (7 instances), B (2 instances), and C (6 instances)

Suppose that, at timeT_0, we have the following resource-allocation state

Processes	Allocation	Request	Available
	ABC	ABC	ABC
$\mathbf{P_0}$	010	000	000
$\mathbf{P_1}$	200	202	
$\mathbf{P_2}$	303	000	
$\mathbf{P_3}$	211	100	
$\mathbf{P_4}$	002	002	
Total	726		

Let Work and Finish be vectors of length m and n, respectively.

Initialize Work=Available. For i = 0, 1, ..., n–1,

Properties

1. if Allocation is ≠ 0, then Finish[i]= false. Otherwise, Finish[i]=true.

2. Find an index i such that both

a. Finish[i]==false

b. Request ≤ Work

If no such i exists, go to step 4.

3. Work=Work + Allocation

i Finish[i]=true Goto step 2.

4. If Finish[i]==false for some i, ,0≤i<n, then the system is in a deadlocked state. Moreover, if Finish[i]==false, then process Pi is deadlocked.

Using for property 2;

For P_0

Allocation ≠ 0; 010≠0; Finish [0] = False

Request <= Work

000<=000; Finish [0] = True

New Work = Work + Allocation

000+010 = 010

For P_1

Allocation ≠ 0, 200≠0; Finish [0] = False

Request <= Work

202<= 010; Finish [1] = False

For P_2

Allocation ≠ 0, 302≠0; Finish [2] =False

Request<=work

000<=010; Finish [2] = True

New Work = Work + Allocation

303+010 = 313

For P_3

Allocation ≠ 0; 211≠0; Finish [3] = False

Request <= Work

100<=313; Finish [3] =True

New work = Work + Allocation

211+313 = 524

For P_4

Allocation ≠0,002≠0; Finish [4] = False

Request<=Work

002<=524; Finish [4] =True

New work = Work + Allocation

524+002 = 526

Checking for P_1 again

Allocation ≠ 0, 200≠0, Finish [1] =False

Request<=Work

202<=526; Finish [1] = True

New Work = Work+ Allocation

200+526 = 726

Safe Sequence is $<P_0, P_2, P_3, P_4, P_1>$

NOTE:

Kindly install numpy use this command for the code to run.

- Pip install numpy in the terminal

Code is as below;

```
import numpy as np

import os


class BankersAlgorithm:

    def __init__(self, allocation, max_claim, instances):

        self.allocation = allocation

        self.max_claim = max_claim

        self.instances = instances

        self.need = self.calculate_Need()

        self.available = self.calculate_Available()


    def calculate_Need(self):

        """Calculate the Need matrix based on Max and Allocation."""

        return [

            tuple(self.max_claim[i][j] - self.allocation[i][j] for j in
range(len(self.max_claim[i])))

            for i in range(len(self.max_claim))

        ]
```

```python
    def calculate_Available(self):
        """Calculate the Available resources based on total instances and allocation."""
        allocated = np.array(self.allocation).sum(axis=0)
        return [
            self.instances["A"] - allocated[0],
            self.instances["B"] - allocated[1],
            self.instances["C"] - allocated[2]
        ]

    def can_request_be_granted(self, process, request):
        """Check if a request can be granted without modifying the system state."""
        print(f"\nChecking if request {request} by Process {process} can be granted...")

        if any(request[i] > self.need[process][i] for i in range(len(request))):
            print(f"Request cannot be granted: Process {process} has exceeded its maximum claim.")
            return False

        if any(request[i] > self.available[i] for i in range(len(request))):
            print(f"Request cannot be granted: Process {process} must wait (Not enough resources available).")
            return False
```

```
        print(f"Request {request} by Process {process} can be granted.")
        return True

    def request_resources(self, process, request):
        """Request resources for a process and check if the request can be granted."""
        print(f"\nProcessing request {request} by Process {process}...")

        if not self.can_request_be_granted(process, request):
            print(f"Request {request} by Process {process} is denied.")
            return False

        # Step 2: Temporarily allocate resources for safety check
        temp_available = [self.available[i] - request[i] for i in range(len(request))]
        temp_allocation = [list(self.allocation[i]) for i in range(len(self.allocation))]
        temp_need = [list(self.need[i]) for i in range(len(self.need))]

        for i in range(len(request)):
            temp_allocation[process][i] += request[i]
            temp_need[process][i] -= request[i]

        temp_system = BankersAlgorithm(temp_allocation, self.max_claim,
self.instances)

        # Step 3: Check system safety after temporary allocation
        if temp_system.is_safe_state():
```

```
            self.available = temp_available
            self.allocation = [tuple(row) for row in temp_allocation]
            self.need = [tuple(row) for row in temp_need]
            print(f"Request granted for Process {process}.")
            return True
        else:
            print(f"Request denied for Process {process}.")
            return False

    def is_safe_state(self):
        """Determine if the system is in a safe state."""
        work = self.available[:]  # Copy Available resources
        finish = [False] * len(self.allocation)
        safe_sequence = []

        while len(safe_sequence) < len(self.allocation):
            progress = False
            for i in range(len(self.need)):
                if not finish[i] and all(self.need[i][j] <= work[j] for j in range(len(work))):
                    work = [work[j] + self.allocation[i][j] for j in range(len(work))]
                    finish[i] = True
                    safe_sequence.append(i)
                    progress = True
                    break
```

```
            if not progress:
                print("System is NOT in a safe state.")
                return False

        print("System is in a safe state.")
        print("Safe Sequence:", safe_sequence)
        print("Available Resources:", work)
        return True

    def deadlock_detection(self):
        """Deadlock Detection Algorithm: Find if any process is in deadlock."""
        work = self.available[:]
        finish = [False] * len(self.allocation)
        deadlocked_processes = []

        while True:
            progress = False
            for i in range(len(self.need)):
                if not finish[i] and all(self.need[i][j] <= work[j] for j in
range(len(work))):
                    work = [work[j] + self.allocation[i][j] for j in range(len(work))]
                    finish[i] = True
                    progress = True
                    break
```

```
        if not progress:
            break

    # Collect all processes that are not finished, indicating deadlock
    for i in range(len(finish)):
        if not finish[i]:
            deadlocked_processes.append(i)

    if deadlocked_processes:
        print("Deadlock detected in the following processes:",
deadlocked_processes)
    else:
        print("No deadlock detected.")

  def make_request(self, process, A, B, C):
    """Simplified function to make a request dynamically."""
    request = (A, B, C)
    self.request_resources(process, request)

  def clear(self):
    """Clear the terminal screen."""
    os.system('cls' if os.name == 'nt' else 'clear')

def main():
  data = {
```

```
    "Allocation": [
        (0, 1, 0), (2, 0, 0), (3, 0, 2), (2, 1, 1), (0, 0, 2)
    ],
    "Max": [
        (7, 5, 3), (3, 2, 2), (9, 0, 2), (2, 2, 2), (4, 3, 3)
    ],
    "Available": [],
    "Need": []
}

instances = {"A": 10, "B": 5, "C": 7}

bankers = BankersAlgorithm(data["Allocation"], data["Max"], instances)

while True:
    print("\n===== Banker's Algorithm Menu =====")
    print("1. Check if the system is in a safe state")
    print("2. Make a resource request")
    print("3. Check for deadlock")
    print("4. Clear the screen")
    print("5. Exit")

    choice = input("Enter your choice (1/2/3/4/5): ")
```

```
if choice == '1':
    bankers.is_safe_state()

elif choice == '2':
    try:
        process = int(input("Enter process number (0 to 4): "))
        A = int(input("Enter request for resource A: "))
        B = int(input("Enter request for resource B: "))
        C = int(input("Enter request for resource C: "))
        bankers.make_request(process, A, B, C)
    except ValueError:
        print("Invalid input. Please enter valid integers for the request.")

elif choice == '3':
    bankers.deadlock_detection()

elif choice == '4':
    bankers.clear()  # Clear the screen

elif choice == '5':
    print("Exiting...")
    break

else:
```

```
            print("Invalid choice, please try again.")


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

Output after running Code;

```
Ch
===== Banker's Algorithm Menu =====
1. Check if the system is in a safe state
2. Make a resource request
3. Check for deadlock
4. Clear the screen
5. Exit
Enter your choice (1/2/3/4/5): 1
System is in a safe state.
Safe Sequence: [1, 3, 0, 2, 4]
Available Resources: [np.int64(10), np.int64(5), np.int64(7)]
```

Making a request

for P1 (1,0,2)

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5): 2

Enter process number (0 to 4): 1

Enter request for resource A: 1

Enter request for resource B: 0

Enter request for resource C: 2

Processing request (1, 0, 2) by Process 1...

Checking if request (1, 0, 2) by Process 1 can be granted...

Request (1, 0, 2) by Process 1 can be granted.

System is in a safe state.

Safe Sequence: [1, 3, 0, 2, 4]

Available Resources: [np.int64(10), np.int64(5), np.int64(7)]

Request granted for Process 1.

Enter your choice (1/2/3/4/5): 2

Enter process number (0 to 4): 4

Resource request for **P_4 (3,3,0) ;**

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5): 2

Enter process number (0 to 4): 4

Enter request for resource A: 3

Enter request for resource B: 3

Enter request for resource C: 0

Processing request (3, 3, 0) by Process 4...

Checking if request (3, 3, 0) by Process 4 can be granted...

Request (3, 3, 0) by Process 4 can be granted.

System is NOT in a safe state.

Request denied for Process 4.

Resource request for **P_0 (0,2,0);**

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5): 2

Enter process number (0 to 4): 0

Enter request for resource A: 0

Enter request for resource B: 2

Enter request for resource C: 0

Processing request (0, 2, 0) by Process 0...

Checking if request (0, 2, 0) by Process 0 can be granted...

Request (0, 2, 0) by Process 0 can be granted.

System is in a safe state.

Safe Sequence: [3, 1, 0, 2, 4]

Available Resources: [np.int64(10), np.int64(5), np.int64(7)]

Request granted for Process 0.

Testing for Deadlock

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5): 3

No deadlock detected.

Clearing Screen

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5):

Exiting

===== Banker's Algorithm Menu =====

1. Check if the system is in a safe state

2. Make a resource request

3. Check for deadlock

4. Clear the screen

5. Exit

Enter your choice (1/2/3/4/5): 5

Exiting...