

**WIA2004 OPERATING SYSTEM**

**LAB 7 PROJECT REPORT**

**BANKERS ALGORITHM FOR THE PURPOSE OF DEADLOCK AVOIDANCE**

**GROUP MEMBERS**

| **No.** | **Member Name** | **Matric Number** |
| --- | --- | --- |
| **1.** | **MUHAMMAD IZZAT HAKEEM BIN ROSDI** | **17204856** |
| **2.** | **WAN MUHAMMAD AZIB BIN WAN ABDUL RAHIM** | **17207678** |
| **3.** | **MUHAMMAD AMIRUL AMIN BIN ARMAN** | **17202969** |
| **4.** | **JONE WAN GAN** | **S2031153** |
| **5.** | **LAWRENCE LEROY CHIENG TZE YAO** | **S2018935** |

**OBJECTIVE**

Write a program to simulate Bankers algorithm for the purpose of deadlock avoidance.

**DESCRIPTION**

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker’s algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

**Banker Algorithm**

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

**Advantages of Banker algorithm deadlock avoidance :**

Avoids deadlock and it is less restrictive than deadlock prevention.Bankers algorithm is safe and effective algorithm

**Disadvantages of Banker algorithm deadlock avoidance :**

-Only works with a fixed number of resources and processes.

-Guarantees finite time - not reasonable response time

-Needs advanced knowledge of maximum needs

-Not suitable for multi-access systems

-Unnecessary delays in avoiding unsafe states which may not lead to deadlock

**CODE**

# Banker's Algorithm

# LAB 7 OS

if \_\_name\_\_ == "\_\_main\_\_":

# P0, P1, P2, P3, P4 are the Process names here

n = 5 # Number of processes 5

m = 3 # Number of resources 3

print('The number of process :' + str(n))

print('The number of resources :' + str(m))

# Allocation Matrix

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

print('The allocation matrix : \n' + alloc.\_\_str\_\_())

#

# MAX Matrix

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

print('The max matrix : \n' + max.\_\_str\_\_())

#

# Available Resources

avail = [3, 3, 2]

print('Available resources: \n ' + avail.\_\_str\_\_())

f = [0] \* n

ans = [0] \* n

ind = 0

for k in range(n):

f[k] = 0

need = [[0 for i in range(m)] for i in range(n)]

for i in range(n):

for j in range(m):

need[i][j] = max[i][j] - alloc[i][j]

y = 0

for k in range(5):

for i in range(n):

if (f[i] == 0):

flag = 0

for j in range(m):

if (need[i][j] > avail[j]):

flag = 1

break

if (flag == 0):

ans[ind] = i

ind += 1

for y in range(m):

avail[y] += alloc[i][y]

f[i] = 1

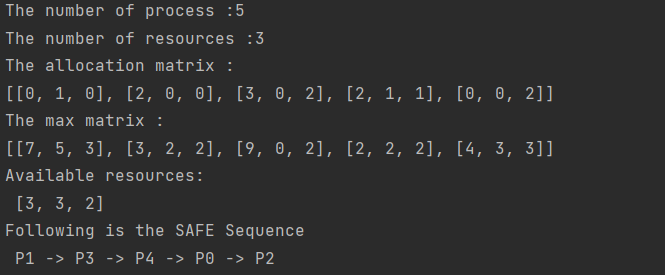
print("Following is the SAFE Sequence")

for i in range(n - 1):

print(" P", ans[i], " ->", sep="", end="")

print(" P", ans[n - 1], sep="")

**OUTPUT**

****