**WEEK 7**

**PROGRAM NO-01**

**AIM :**Implement Bankers algorithm for deadlock avoidance . Print the safe sequence if the system is in safe state, Otherwise inform that there no safe sequence. Also implement granting of resource request .

**CODE:**

#bankers algorithm

n=int(5)

alloc=[[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]]

avail=[3,3,2]

need=[]

for i in range(n):

ne=[]

for j in range(3):

ne.append(max[i][j]-alloc[i][j])

need.append(ne)

def bankers(n,avail,need,alloc):

u=int(0)

p=n

process= [0,0,0,0,0]

seq=[]

o=0

while u<n and o==0:

v=0

for i in range(p):

if process[i]==0:

if ((avail[0]>=need[i][0]) and (avail[1]>=need[i][1]) and (avail[2]>=need[i][2]) ) :

avail[0]= avail[0]+ alloc[i][0]

avail[1]= avail[1]+ alloc[i][1]

avail[2]= avail[2]+ alloc[i][2]

process[i]=1

seq.append(i)

u=u+1

v=v+1

if v==0 :

o=1

print("Safe sequence :")

if len(seq)==n:

for i in seq:

print('process '+str(i))

else :

print("None")

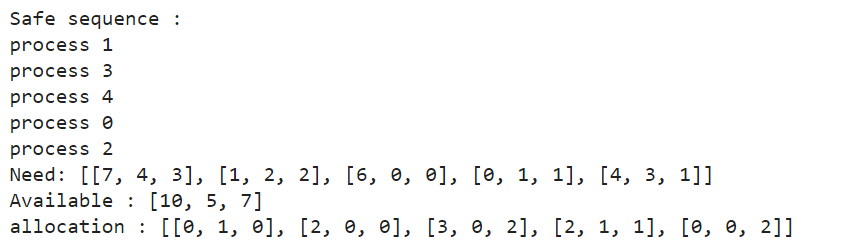
bankers(n,avail,need,alloc)

print("Need:",need)

print("Available :",avail)

print("allocation :",alloc)

**OUTPUT:**



#resource request allocation

def resource\_req(av,need,alloc,rn,n):

if (req[0]<=av[0])and (req[1]<=av[1]) and (req[2]<=av[2]):

xav=0

else :

xav=1

if (need[rn][0]>=req[0])and (need[rn][1]>=req[1]) and (need[rn][2]>=req[2]) :

zne=0

else :

zne=1

if xav==0 and zne==0 :

for i in range(3):

alloc[rn][i]=alloc[rn][i]+req[i]

need[rn][i]=need[rn][i]-req[i]

av[i]= av[i]-req[i]

alloc[rn]=a

print("Need:",need)

print("Available ;",av)

print("allocation :",alloc)

bankers(n,av,need,alloc)

elif xav ==1 and zne==0:

print("Process is under wait , resources are not available")

else :

print("Resource request not possible")

rn=1

alloc=[[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]]

avail=[3,3,2]

need=[]

for i in range(n):

ne=[]

for j in range(3):

ne.append(max[i][j]-alloc[i][j])

need.append(ne)

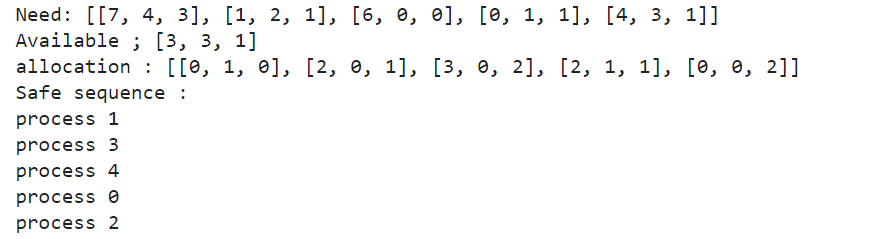
req=[0,0,1]

a=alloc[rn]

n=5

av=[3,3,2]

resource\_req(av,need,alloc,rn,n)



alloc=[[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]]

avail=[3,3,2]

need=[]

for i in range(n):

  ne=[]

  for j in range(3):

    ne.append(max[i][j]-alloc[i][j])

  need.append(ne)

rn=2

req=[0,0,1]

a=alloc[rn]

n=5

av=[3,3,2]

resource\_req(av,need,alloc,rn,n)



alloc=[[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]]

avail=[3,3,2]

need=[]

for i in range(n):

ne=[]

for j in range(3):

ne.append(max[i][j]-alloc[i][j])

need.append(ne)

rn=0

req=[0,4,0]

a=alloc[rn]

n=5

av=[3,3,2]

resource\_req(av,need,alloc,rn,n)



**PROGRAM NO-02**

**Aim:** Implement equal sized fixed partitioning algorithm and calculate total internal fragmentation.

**Code:**

**#**equal sized partitioning

total=int(input("Enter total main memory :"))

partitions=int(input("Enter number of partitions :"))

resources=input("Enter resources memory :").split()

IF=[]

x=total/partitions

s=([x]\*partitions)

for i in range(partitions):

a=resources[i]

if float(a) <=x:

v=s[i]-float(a)

IF.append(v)

elif a>total:

print("For"+str(a)+"use technique similar to overlays technique")

print("Internal Fragmentations :")

sum=0

for i in IF:

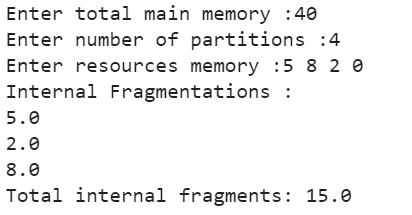
if i!= x:

print(i)

sum += i

print('Total internal fragments:',sum)

**Output:**



**PROGRAM NO-03**

**Aim:** Implement Un equal sized fixed partitioning algorithm and calculate total internal fragmentation.

**Code:**

#unequal partition sizes

p=input("Enter partitions :").split()

r=input("Enter resources memory :").split()

parts=[]

res=[]

for i in p:

parts.append(int(i))

for i in r :

res.append(int(i))

parts=sorted(parts)

x=[]

a=[]

b=[]

for i in range(len(res)):

for j in range(len(parts)):

if res[i]<=parts[j] and (res[i] not in a) and (parts[j] not in b):

x.append(parts[j]-res[i])

parts.pop(j)

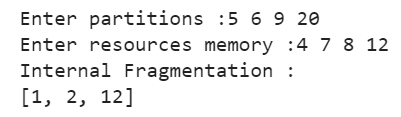
a.append(res[i])

b.append(parts[j])

print("Internal Fragmentation :")

print(x)

**Output:**



**PROGRAM NO-04**

**Aim:** Implement dynamic partitioning algorithm and calculate total external fragmentation.

**Code:**

tm = int(input("Enter total memory size:"))

n = int(input("Enter no. of jobs:"))

summ = 0

for i in range(n):

j\_s = int(input("Enter job size:"))

if j\_s <= tm and summ <= tm-summ:

print("Job",i+1,"allocated in main memory.")

summ+=j\_s

else:

print("Space unavailable.")

print("External fragmentation:",tm-summ)

**Output:**

