Deepesh Tank
1807501.

{ Experiment - 5 }.

Deadlock managment techiques:

Aim-(1).

DATE / /
PAGE

Aim:-

Write a C program to stimulate banker's algorithm for purpose of deadlock avoidance.

Theory:-

Bankers algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources and then make a 's-state' check to test for possible deadlock conditions for all other pending activities. before adding deciding whether allocation should be continued.

Data Structure used:-

1) Let n be number of processes
2) let m be total number of resources types in system
3) Available :- A vector of length m It show available resources of each type
4) if Available [i] = k k instance of resource R_i is available
5) Max: An n x m matrix that contains maximum demand of each process if Max[i][j] = k,
   Then process P_i can request maximum k instance of resource R_j
6) Allocation:- An n x m matrix that contains no of resources of each type currently allocated to each process.
   if Allocation[i,j] = k, then P_i is currently allocated k instance of resource R_j
7) Need :- A (n x m) matrix that shows the remaining resources need of each process,
   if Need [i,j] = k then process P_i may need k instance of Resource R_j to complete the task

Algorithm:-

Banker's algorithm consists of safety algorithm and Resource request algorithm.

safety algorithm:-

for finding if system is in safe state or not

1) Let Work and Finish[i] ~~= false~~ be vectors of length m and n respectively

initialize : Work = Available.

Finish[i] = false ; {1≤i≤n.}

2) Find an i such that both.

a) Finish[i] = false.

b) Needi ≤ Work.

if no such i exsist go to step 4

3) work += Allocation[i]

Finish[i] = true

go to step (2)

4) if Finish[i] == true for all i

then system is in safe state

Resource-Request Algorithm:-

let Requesti be the request array for process Pi Requesti[j] = k

means process Pi wants k instance of resource Rj when a request for resource is made by process Pi the following actions are taken:

1) If Request$_i$ <= Need$_i$
go to step (2) otherwise
raise an error condition since the process has exceeded its maximum Claim

2) If Request$_i$ ≤ Available go to step (3) otherwise
Pi must wait since Resources are not available

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows.
Available -= Request$_i$
Allocation$_i$ += Request$_i$
Need$_i$ -= Request$_i$

program code:

```
#include <stdio.h>
int main()
{ int n, m, i, j, k;
  n = 5;
  m = 3;
  int allocation [5][3] = { { 0, 1, 0 },
                            { 2, 0, 0 },
                            { 3, 0, 2 },
                            { 2, 1, 2 } };
  int max [5][3] = { { 7, 5, 3 },
                     { 3, 2, 2 },
                     { 9, 0, 2 },
                     { 2, 2, 2 },
                     { 4, 3, 3 } };
  int available [3] = { 3, 3, 2 };
int f[n], ans[n], ind = 0
  for ( k=0; k < n; k++) {
      f[k] = 0; }
int need [n][m];
  for ( i=0; i<n; i++) {
     for(j=0; j<m; j++) {
                need[i][j] = max[i][j] - alloc[i][j]; } }
int y = 0;
for (k = 0; k<5; k++){
  for (i=0; i<n; i++) {
     if (f[i] == 0) {
          int flag=0;
       for (j=0; j <m; j++) {
          if (need[i][j] > available[j]) {
```

```
                flag = 1;
                break; } }
if (flag == 0) {
    ans[ind++] = i;
for (y = 0; y < m; y++)
        available[y] += allocation[i][y];
        f[i] = 1;
    }
  }
 }
printf ("Following is the safe sequence \n");
for (i=0; i<n-1; i++)
  {printf (" P%d -> ", ans[i]); }
printf (" P %d \n", ans[n-1]);
return 0;
}
```

output :-

Following is the safe sequence

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$.

Aim (2).

DATE / /
PAGE

Aim :-

write a C program to stimulate safety algorithm.

Theory :-

Safety algorithm is used for finding out whether a system is in safe state or not.

Data structure is used as that of banker's algorithm and in actual it is a part of banker's Algorithm and works as follows.

Algorithm :-

1) let Work and Finish be vector of length m and n respectively

initialize : Work = Available

Finish [i] = false; $1 \leq i \leq n$

2) Find an i such that

a) Finish [i] = false;

b) Need_i $\leq$ work;

if no such i exist go to step 4

3) Work + = Allocation;

Finish [i] = true

go to step (2)

4) if Finish [i] = true for all i

then the system is in safe state.

Program code:-

```
#include <stdio.h>
const int P = 5;
const int R = 3;
void calculateneed (int need[P][R], int maxm[P][R],
    int allot[P][R])
{ for (int i=0; i<P; i++)
      for (int j=0; j<R; j++)
          need[i][j] = maxm[i][j] - allot[i][
}
bool issafe (int processess[], int avail[], int maxm[]
          int allot[][R])
{   int need[P][R];
    calculateneed (need, maxm, allot);
    bool finish[P] = {0};
    int safeseq[P];
    int work[R]; int i=0;
    for (i=0; i<R; i++)
      {  work[i] = avail[i];
    int count=0;
while (count<P)
    {   bool found = false;
    for (int p=0; p<P; p++)
      if (finish[p]==0)
          {   int j;
              for (int j=0; j<R; j++)
                  if (need[p][j] > work[j])
                      break;
    if (j==R)
      {
```

```
for (int k = 0; k<R; k++) {
        work [k] += allot [p][k]; }
safeseq [count++] = p;
finish [p] = 1;
found [p] = 1;
found = true;
}
}
if (found == false)
{ printf ("Not safe state");
return false; } }
printf (" safe state \n sequence will be \n ");
for ( int a = 0; a < P; a++)
    printf (" %d", safeseq [i]);
return true;
}
int main ( ) {
int processes = {0, 1, 2, 3, 4};
int avail [] = { 3, 3, 2}
int maxm [ ] [R] = { { 7, 5, 3 },
                     { 3, 2, 23,
                     { 9, 0, 2 }
                     { 2, 2, 2 }
                     { 4, 3, 3 } };
int alloc [ ] [R] = { { 0, 1, 0 },
                     { 2, 0, 0 },
                     { 3, 0, 2 }
                     { 2, 1, 1 }
                     { 0, 0, 2 } };
```

```
issafe(process, avail, maxm, allot);
return 0;}
```

output:-

safestate

sequence will be

1 3 4 0 2.