**OPERATING SYSTEMS**

LAB EXPERIMENT - 5

Submitted To: Prashant Giridhar Shambharkar Sir.

Submitted By: Manav Agrawal (2K19/CO/216)

---------------------------------------------------------------------

Aim:

Write a C program to simulate Bankers Algorithm for DeadLock Avoidance.

Introduction:

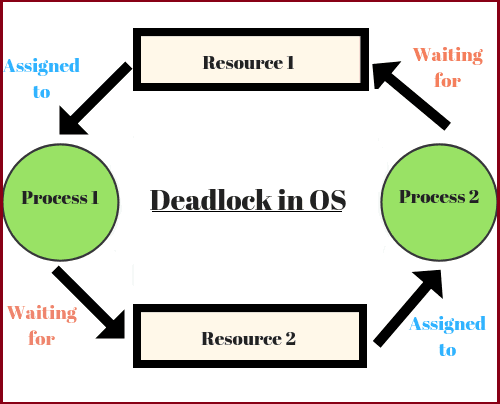
### Deadlock:

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grants the resource if it is available otherwise let the process waits.
3. The process uses it and releases on the completion.

A Deadlock is a situation where each of the computer processes waits for a resource which is being assigned to some other process. In this situation, none of the process gets executed since the resource it needs is held by some other process which is also waiting for some other resource to be released.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Necessary conditions for Deadlocks:

1. Mutual Exclusion: A resource can only be shared in a mutually exclusive manner. It implies, if two processes cannot use the same resource at the same time.
2. Hold and Wait: A process waits for some resources while holding another resource at the same time.
3. No preemption: The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.
4. Circular Wait: All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

**Deadlock Avoidance:**

Simplest and most useful model requires that each process declare

the maximum number of resources of each type that it may need.

The deadlock-avoidance algorithm dynamically examines the

resource-allocation state to ensure that there can never be a

circular-wait condition. Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

**Bankers 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.

Data Structures required for Banker’s Algorithm

* Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available.
* Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj.
* Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj.
* Need: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task.

Need [i,j] = Max[i,j] – Allocation [i,j]

Algorithms:

**Banker’s Algorithm:**

Banker’s algorithm consists of Safety algorithm and Resource request algorithm:

Safety Algorithm:

1- Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4….n

2- Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

3- Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4- if Finish [i] = true for all i then the system is in a safe state.

Resource-Request Algorithm:

1- If Requesti <= Needi

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2- If Requesti <= Available

Goto step (3); otherwise, Pi must wait, since the 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 = Available – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi– Requesti

Implementation:

**Banker’s Algorithm:**

The following is the input values and attributes for the processes with their requested resource.

----------------------------------------------------------------------------------------------------------------

Process | Allocation | Max Request | Available Resource |

| X Y Z | X Y Z | X Y Z |

----------------------------------------------------------------------------------------------------------------

Process0 | 0 1 0 | 6 6 4 | 3 2 4 |

Process1 | 3 0 0 | 3 2 2 | |

Process2 | 2 0 2 | 9 0 2 | |

Process3 | 1 1 1 | 2 2 2 | |

Process4 | 0 0 2 | 4 3 3 | |

---------------------------------------------------------------------------------------------------------------

PROGRAM:

#include <iostream>

using namespace std;

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int max\_matrix[5][3] ={ { 6, 6, 4 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int available\_resources[3] = { 3, 2, 4 };

int allocation\_matrix[5][3] = { { 0, 1, 0 },

{ 3, 0, 0 },

{ 2, 0, 2 },

{ 1, 1, 1 },

{ 0, 0, 2 } };

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need\_matrix[n][m];

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++)

need\_matrix[i][j] = max\_matrix[i][j] - allocation\_matrix[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\_matrix[i][j] > available\_resources[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

available\_resources[y] += allocation\_matrix[i][y];

f[i] = 1;

}

}

}

}

cout << "Following is the SAFE Sequence" << endl;

for (i = 0; i < n - 1; i++)

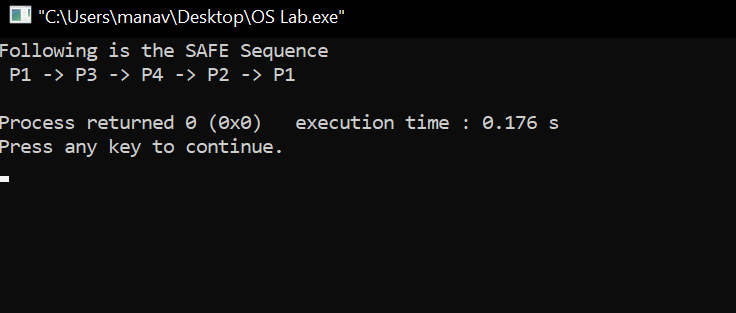
cout << " P" << ans[i] << " ->";

cout << " P" << ans[n - 1] <<endl;

return (0);

}

Output:

**Banker’s Algorithm:**

Learning From The Experiment:

Banker’s algorithm avoids deadlock and it is less restrictive than deadlock prevention. It is not necessary to preempt and rollback processes, as in deadlock detection.

However, the restrictions are that it only works with a fixed number of resources and processes. It only guarantees finite time - not reasonable response time. It needs advanced knowledge of maximum needs. It is not suitable for multi-access systems. Maximum resource requirement for each process must be stated in advance. Processes under consideration must be independent and with no synchronization requirements. There must be a fixed number of resources to allocate. No process may exit while holding resources.

***THANK YOU!***