Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

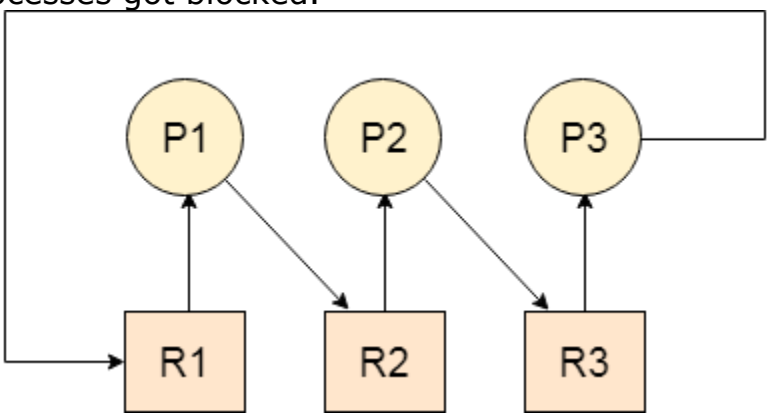
Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	9		

AIM:	Write a multithreaded program for preventing race conditions and deadlock avoidance for the banker's algorithm as follows. Several customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied.
THEORY:	Introduction to 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 grant the resource if it is available otherwise let the process waits. 3. The process uses it and release on the completion. A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another 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. Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3. After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also

stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Necessary conditions for Deadlocks

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two process 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.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

Strategies for handling Deadlock

1. **Deadlock Ignorance**
Deadlock Ignorance is the most widely used approach among all the mechanism. This is being used by many operating systems mainly for end user uses. In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff. There is always a tradeoff between Correctness and performance. The operating systems like Windows and Linux mainly focus upon performance.

2. **Deadlock prevention**
Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

3. **Deadlock avoidance**
In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

4. **Deadlock detection and recovery**
This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.
We will discuss deadlock detection and recovery later in more detail since it is a matter of discussion.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

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.

Following **Data structures** are used to implement the Banker's Algorithm:
Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$
- Need [i, j] = Max [i, j] – Allocation [i, j]

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

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

Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

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

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

EXPERIMENT 1	

CODE:	//C program for Banker's Algorithm
	#include <stdio.h>
	int main()
	{
	// P0, P1, P2, P3, P4 are the names of Process
	
	int n, r, i, j, k;
	printf("\n Enter the number of processes : ");
	scanf("%d", &n); // Indicates the Number of processes
	printf("\n Enter the number of resources : ");
	scanf("%d", &r); //Indicates the Number of resources
	printf("Enter the allocation matrix:\n ");
	for (i=0;i<r;i++)
	printf(" %c",(i+97));
	printf("\n");
	int alloc[n][r];
	for (i=0;i <n;i++) {
	printf("P[%d] ", i+1);
	for (j=0;j<r;j++) {
	scanf("%d",&alloc[i][j]);
	}
	}
	int max[n][r];
	printf(" ");
	for (i=0;i<r;i++) {
	printf(" %c",(i+97));
	}
	printf("\n");
	for (i=0;i <n;i++) {
	printf("P[%d] ",i);
	for (j=0;j<r;j++)
	scanf("%d", &max[i][j]);
	}
	printf("\n");

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

```
int avail[3] = { 2, 3, 2 };
printf("Enter available resources: ");
for(int i=0;i<n;i++)
scanf("%d", &avail[i]);
// These are Available Resources

int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
	f[k] = 0;
}
int need[n][r];
for (i = 0; i < n; i++) {
	for (j = 0; j < r; 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 < r; j++) {
				if (need[i][j] > avail[j]){
					flag = 1;
					break;
				}
			}

			if (flag == 0) {
				ans[ind++] = i;
				for (y = 0; y < r; y++)
					avail[y] += alloc[i][y];
				f[i] = 1;
			}
		}
	}
}
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** OS

```
        printf("The SAFE Sequence is as follows\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);

        return (0);

}
```

OUTPUT:

```
 Enter the number of resources : 3

 enter the max instances of each resources
a= 3
b= 3
c= 2

 Enter the number of processes : 5

 Enter the allocation matrix
      a b c
P[0]  0 1 0
P[1]  2 0 0
P[2]  3 0 2
P[3]  2 1 1
P[4]  0 0 2

Enter the MAX matrix
      a b c
P[0]  7 5 3
P[1]  3 2 2
P[2]  9 0 2
P[3]  2 2 2
P[4]  4 3 3

The SAFE Sequence is as follows
 P1 -> P3 -> P4 -> P0 -> P2
```

RESULT: Learnt about the bankers algorithm and applied it in C language. Understood the deadlock theory and its methods to avoid it. Also learnt how to apply deadlock strategies to real life problems