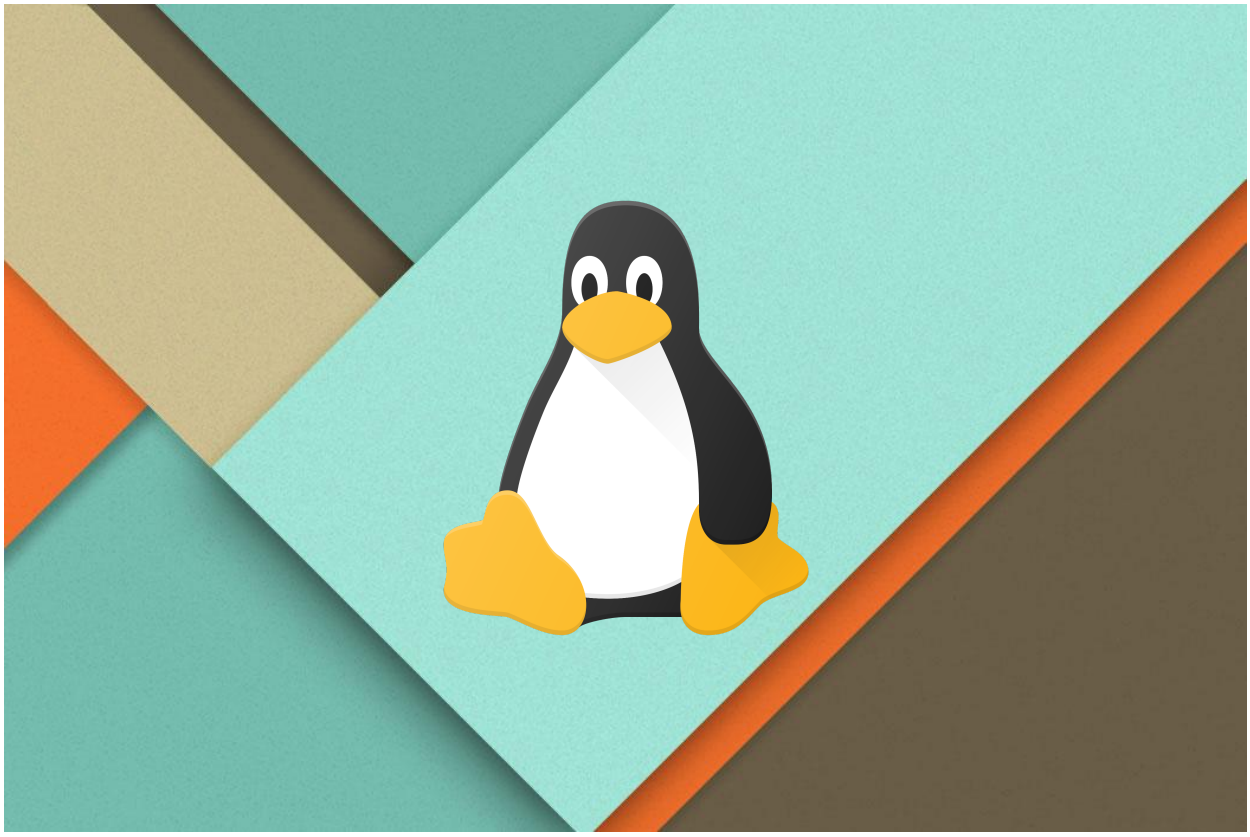


Operating Systems Lab

Experiment No. 7

22.03.2022

Professor - Dr. Shrinivas Khedkar.

Pratham Loya
201080068
IT
prloya_b20@it.vjti.ac.in

Aim

Study and Implement Bankers Algorithm for Deadlock &

Implement algorithm for deadlock Detection in C programming.

Theory

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

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

What is a Deadlock ?

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 processes gets executed since the resource it needs is held by some other process which is also waiting for some other resource to be released.

For Example: Let us assume that there are three processes **P1**, **P2** and **P3**. And 3 different resources assigned to them **R1**, **R2**, and **R3** respectively.

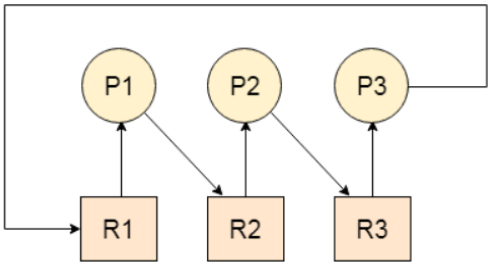
After some time, *P1 demands for R2* which is being used by P2.

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.*



Difference between Deadlock & Starvation

Deadlock	Starvation
Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process gets blocked and the high priority processes proceed.
Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
Every Deadlock is always a starvation.	Every starvation need not be a deadlock.
The requested resource is blocked by the other process.	The requested resource is continuously used by the higher priority processes.
Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Banker's Algorithm for Deadlock Detection

The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources.

Similarly in the case of OS the '**S-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes.

It is a banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system.

When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming *processes, requests for their resources, counting them, and delays*. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as **deadlock avoidance algorithm** or **deadlock detection** in the operating system.

Advantages

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that indicates each process can hold the maximum number of resources in a system.

Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

Working

When working with a banker's algorithm, it requests to know about three things:

1. How much each process can request for each resource in the system. It is denoted by the [**MAX**] request.
2. How much each process is currently holding each resource in a system. It is denoted by the [**ALLOCATED**] resource.
3. It represents the number of each resource currently available in the system. It is denoted by the [**AVAILABLE**] resource.

Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.

1. **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.
2. **Max:** It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.
3. **Allocation:** It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.
4. **Need:** It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work.
   Nedd[i][j] = Max[i][j] - Allocation[i][j].
5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system.

Safety Algorithm

It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Work** and **Finish** of length m and n in a safety algorithm.

   Initialize: Work = Available

   Finish[i] = false; for I = 0, 1, 2, 3, 4... n - 1.

2. Check the availability status for each type of resources [i], such as:

   Need[i] <= Work

   Finish[i] == false

   If the i does not exist, go to step 4.

3. Work = Work +Allocation(i) // to get new resource allocation

   Finish[i] = true

   Go to step 2 to check the status of resource availability for the next process.

4. If Finish[i] == true; it means that the system is safe for all processes.

Resource Request Algorithm

A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Let's create a resource request array R[i] for each process P[i]. If the Resource Request [j] equals 'K', which means the process P[i] requires 'k' instances of Resources type R[j] in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process P[i] exceeds its maximum claim for the resource. As the expression suggests:

If Request(i) <= Need

Go to step 2;

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

If Request(i) <= Available

Else Process P[i] must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

Available = Available - Request

Allocation(i) = Allocation(i) + Request (i)

$Need_i = Need_i - Request_i$

When the resource allocation state is safe, its resources are allocated to the process P(i). And if the new state is unsafe, the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state.

Program

banker.c (Implementation of Banker's algorithm)

farmer.cpp (Solution to programming problem on page 345 in Galvin Book)

Output

```
~/Desktop/OS
❯ gcc .\banker.c -o .\banker.exe

~/Desktop/OS
❯ .\banker.exe
Th SAFE Sequence is as follows
 P1 -> P3 -> P4 -> P0 -> P2
```

```
~/Desktop/OS
❯ g++ .\farmer.cpp -o .\farmer.exe

~/Desktop/OS
❯ .\farmer.exe
 Vill. 1| Vill. 2| Vill. 3| Vill. 4| Vill. 5|0sec
Appeared|        |        |        |        |2sec
  B to A|        |        |        |        |2sec
 Waiting|        |        |        |        |2sec
Got Perm|        |        |        |        |2sec
Crossing|        |        |        |        |2sec
        |Appeared|        |        |        |3sec
        |  B to A|        |        |        |3sec
        | Waiting|        |        |        |3sec
        |Got Perm|        |        |        |3sec
        |Crossing|        |        |        |3sec
        |        |Appeared|        |        |5sec
        |        |  B to A|        |        |5sec
        |        |        |Appeared|        |5sec
        |        |        |  B to A|        |5sec
        |        |        | Waiting|        |5sec
        |        |        |Got Perm|        |5sec
        |        |        |Crossing|        |5sec
Crossed!|        |        |        |        |5sec
        |        | Waiting|        |        |5sec
        |        |Got Perm|        |        |5sec
        |        |Crossing|        |        |5sec
        |        |        |        |Appeared|6sec
```