Explain deadlock avoidance justifying with example.

Deadlock avoidance is a method of ensuring that the system will never enter an unsafe state, which is a state where there is no possibility of granting all the current requests of all the processes or threads without causing deadlock. [Deadlock avoidance requires some knowledge or prediction of the future requests and releases of each process or thread1](https://www.geeksforgeeks.org/deadlock-prevention/)[2](https://www.baeldung.com/cs/os-deadlock)[3](https://www.studytonight.com/operating-system/deadlock-avoidance-in-operating-system).

**Deadlock Avoidance Algorithm**

Deadlock avoidance uses an algorithm that dynamically examines the resource allocation state before granting any request. The algorithm determines whether granting the request will leave the system in a safe state or an unsafe state. If the request will leave the system in a safe state, it is granted; otherwise, it is denied or delayed until a safe state can be reached.

One of the common algorithms for deadlock avoidance is the banker’s algorithm, which simulates a bank that grants loans to customers and checks whether they can repay them. The banker’s algorithm requires each process or thread to declare in advance the maximum number of each type of resource it may need. The algorithm then keeps track of the available, allocated, and maximum resources for each process or thread, and grants a request only if it satisfies the safety criterion.

**Deadlock Avoidance Example**

Let us consider an example of using the banker’s algorithm to avoid deadlock. Suppose there are five processes P0, P1, P2, P3, P4 and three types of resources A, B, C. The total number of instances of each resource type are 10, 5, 7 respectively. The current allocation and maximum demand of each process are given in the following tables:

| **Process** | **Allocation** | **Maximum** |
| --- | --- | --- |
| P0 | A=0 B=1 C=0 | A=7 B=5 C=3 |
| P1 | A=2 B=0 C=0 | A=3 B=2 C=2 |
| P2 | A=3 B=0 C=2 | A=9 B=0 C=2 |
| P3 | A=2 B=1 C=1 | A=2 B=2 C=2 |
| P4 | A=0 B=0 C=2 | A=4 B=3 C=3 |

The available resources are:

| **Resource** | **Available** |
| --- | --- |
| A | 3 |
| B | 3 |
| C | 2 |

Now suppose P1 requests one additional instance of resource A and two instances of resource C, i.e., Request1 = (1, 0, 2). To decide whether to grant this request, the banker’s algorithm checks if:

* Request1 ≤ Available: This means that the request does not exceed the available resources. In this case, Request1 = (1, 0, 2) ≤ Available = (3, 3, 2), so this condition is true.
* Request1 ≤ Maximum - Allocation: This means that the request does not exceed the maximum demand declared by the process. In this case, Request1 = (1, 0, 2) ≤ Maximum - Allocation = (3 - 2, 2 - 0, 2 - 0) = (1, 2, 2), so this condition is also true.

Since both conditions are true, the banker’s algorithm tentatively grants the request and updates the allocation and available resources as follows:

| **Process** | **Allocation** | **Maximum** |
| --- | --- | --- |
| P0 | A=0 B=1 C=0 | A=7 B=5 C=3 |
| P1 | A=3 B=0 C=2 | A=3 B=2 C=2 |
| P2 | A=3 B=0 C=2 | A=9 B=0 C=2 |
| P3 | A=2 B=1 C=1 | A=2 B=2 C=2 |
| P4 | A=0 B=0 C=2 | A=4 B=3 C=3 |

| **Resource** | **Available** |
| --- | --- |
| A | 2 |
| B | 3 |
| C | 0 |

Then, the banker’s algorithm checks if the new state is safe or not. A state is safe if there exists a sequence of processes that can finish their execution without causing deadlock. To find such a sequence, the banker’s algorithm tries to find a process whose maximum demand can be satisfied by the current available resources. If such a process is found, it is assumed to finish and release all its allocated resources, which are then added to the available resources. This process is repeated until either all processes finish or no more progress can be made.

In this case, the banker’s algorithm can find a safe sequence as follows:

* P1 can finish, as Maximum - Allocation = (3 - 3, 2 - 0, 2 - 2) = (0, 2, 0) ≤ Available = (2, 3, 0). After P1 finishes, Available = Available + Allocation = (2, 3, 0) + (3, 0, 2) = (5, 3, 2).
* P3 can finish, as Maximum - Allocation = (2 - 2, 2 - 1, 2 - 1) = (0, 1, 1) ≤ Available = (5, 3, 2). After P3 finishes, Available = Available + Allocation = (5, 3, 2) + (2, 1, 1) = (7, 4, 3).
* P4 can finish, as Maximum - Allocation = (4 - 0, 3 - 0, 3 - 2) = (4, 3, 1) ≤ Available = (7, 4, 3). After P4 finishes, Available = Available + Allocation = (7, 4, 3) + (0, 0, 2) = (7, 4, 5).
* P0 can finish, as Maximum - Allocation = (7 - 0, 5 - 1, 3 - 0) = (7, 4 ,3) ≤ Available = (7 ,4 ,5). After P0 finishes ,Available = Available + Allocation = (7 ,4 ,5) + (0 ,1 ,0) = (7 ,5 ,5).
* P2 can finish ,as Maximum - Allocation = (9 -3 ,0 -0 ,2 -2) =(6 ,0 ,0) ≤ Available =(7 ,5 ,5). After P2 finishes ,Available=Available+Allocation=(7 ,5 ,5)+(3 ,0 ,2)=(10 ,5 ,7).

The safe sequence is <P1,P3,P4,P0,P2>. Since there exists a safe sequence after granting the request of P1,the banker’s algorithm allows the request and the system remains in a safe state.