Starvation:

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. In starvation resources are continuously utilized by high priority processes. Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.

Example of Starvation:

- In priority-based scheduling algorithms, a major problem is an indefinite block or Starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low-priority processes waiting indefinitely. A steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

How did solve starvation? :

Some solutions that can be implemented in a system to handle Starvation are as follows:

- An independent manager can be used for the allocation of resources. This resource manager distributes resources fairly and tries to avoid Starvation.
- Random selection of processes for resource allocation or processor allocation should be avoided as they encourage Starvation.
- The priority scheme of resource allocation should include concepts such as Aging, where the priority of a process is increased the longer it waits, which avoids Starvation.

Example to solve starvation:

```
Class MyThread extends Thread {
        Public void run() {
                String threadName = Thread.currentThread().getName();
                System.out.println(threadName + " Started");
               Synchronized(MyThread.class) { // lock
                        // doing some useful work
                        Try {
                                Thread.sleep(2000); // 2 sec }
                        catch (InterruptedException ie){}
               }
                System.out.println(threadName + " End");
        }
 }
Public class Test {
        Public static void main(String[] args)
               { System.out.println("Start of Main thread");
               MyThread mt[] = new MyThread[10];
                 For (int i=0; i<mt.length;i++){
                        Mt[i]=new MyThread();
                        Mt[i].start();}
               System.out.println("End Of Main thread")

}
```

Deadlock

Deadlock happens when every process holds a resource and waits for another process to hold another resource. In other words, a deadlock occurs when multiple processes in the CPU compete for the limited number of resources available in the CPU. In this context, each process keeps a resource and waits for another process to obtain a resource.

Examples of Deadlock

It is a common issue in multiprogramming OS, parallel computing systems, and distributed systems. There is a deadlock issue when one process requires a process that is requested by another process.

Four conditions may occur the condition of deadlock. These are as follows:

1. **Mutual Exclusion**
2. **Hold and Wait**
3. **No preemption**
4. **Circular Wait**

Mutual Exclusion

Only one process can utilize a resource at a time; if another process requests the same resource, it must wait until the process that is utilizing it releases it.

Hold and Wait

A process should be holding a resource when waiting for the acquirer of another process's resource.

No Preemption

The process holding the resources may not be preempted, and the process holding the resources should freely release the resource after it has finished its job.

Circular Wait

In a circular form, the process must wait for resources. Let's suppose there are three processes: **P0**, **P1**, and **P2**. **P0** must wait for the resource held by **P1**; **P1** must wait for process **P2** to acquire the resource held by **P2**, and **P2** must wait for **P0** to acquire the process.

Although several applications may detect programs that are likely to become deadlocked, the operating system is never in charge of preventing deadlocks. It is the responsibility of programmers to create programs that are free of deadlocks, and it is possible to avoid deadlock by avoiding the conditions listed above.

How did solve Deadlock? :

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

```
// Thread-1

Runnable block1 = new Runnable() {

  public void run() {

    synchronized (b) {

      try {

        // Adding delay so that both threads can start trying to

        // lock resources

        Thread.sleep(100);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      // Thread-1 have A but need B also
      synchronized (a) {
        System.out.println("In block 1");
      } }
  } }

// Thread-2
Runnable block2 = new Runnable() {
  public void run() {
    synchronized (b) {
      // Thread-2 have B but need A also
      synchronized (a) {
        System.out.println("In block 2");
      }  }
  }  }
```

Real world application :

To be honest it wasn't that hard to find a real world application because in every major u will see at least one who suffers from this problem and unfortenatly it wastes a lot of time and money it may make business closes because of this simple problem so the hard thing to get a real world application is to find the best one which need it so we found one which is for wooden industry

In wooden industry they get wood called mdf, hpl, polylac and so on they get it and save it in place which mustn't be sunny and are considered by amount and quantity which must not get more than it because it will be damaged because of sun or rain or so on

And as we know produce consumer problem has 2 special things mustn't happen which is producer won't produce till the buffer has empty space and the consumer won't consume till the buffer has items

So the first thing which producer won't produce we explained it

And the second one is for industries who works with this industry they can't buy thing aren't available

And the last thing which is produce and consumes in the same time ,, in the industry they get the panel in large machine which will make it perfect so we can't see if it done or not till it is out 100% some workers think it is done so the enter another panel so the machine damage both of them which if we blocked them in the first we would save the wood and money