Project2

Deadlock Implementation

HEADLINES

- Definition
- Background
- What is Deadlock
- Necessary Conditions
- Deadlock In Resource-allocation Graph
- Deadlock In C#
- Example For Deadlock

DEFINTION

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

BACKGROUND

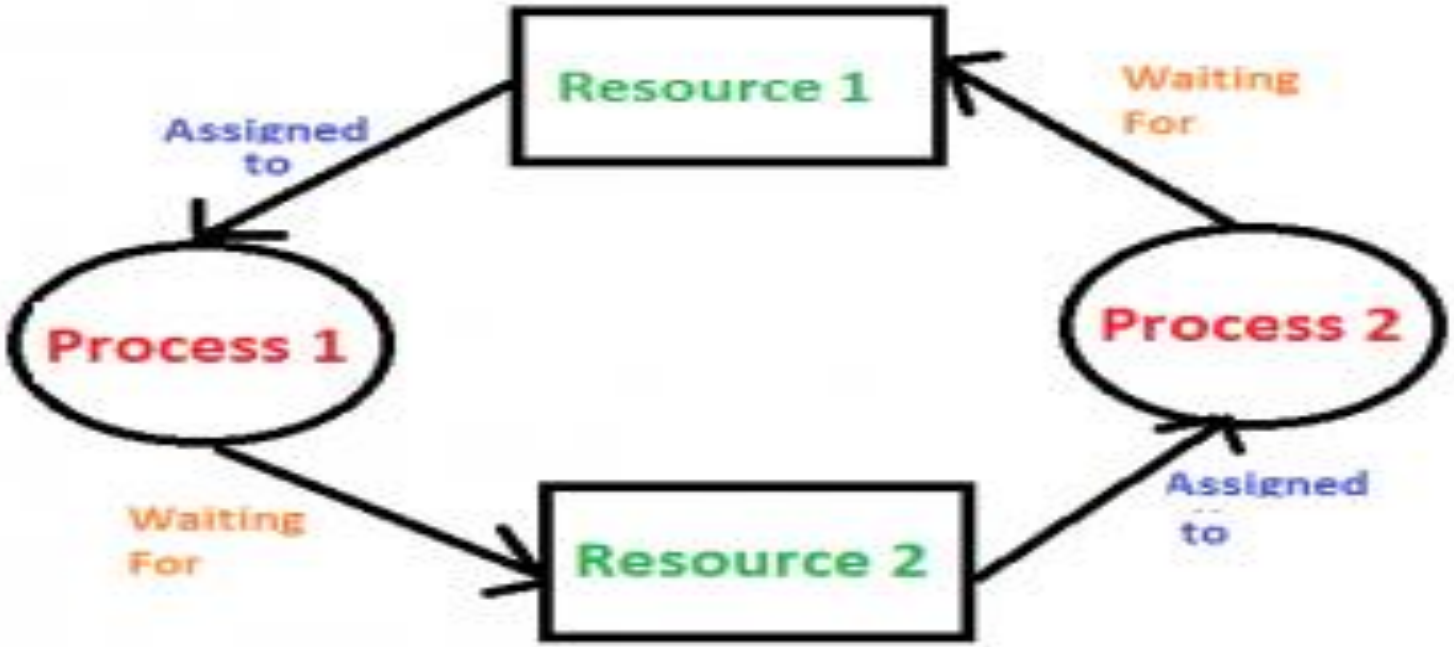
- Under normal operation, a resource allocations proceed like this:

1. Request a resource (suspend until available if necessary).
2. Use the resource.
3. Release the resource.

- The cause of deadlocks: Each process needing what another process has. This results from sharing resources such as memory, devices, links.

WHAT IS DEADLOCK

- A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.



NECESSARY CONDITIONS

- ALL of these four must happen simultaneously for a deadlock to occur: **Mutual exclusion**

One or more than one resource must be held by a process in a non-sharable (exclusive) mode.

- **Hold and Wait**

A process holds a resource while waiting for another resource.

- **No Preemption**

There is only voluntary release of a resource - nobody else can make a process give up a resource.

- **Circular Wait**

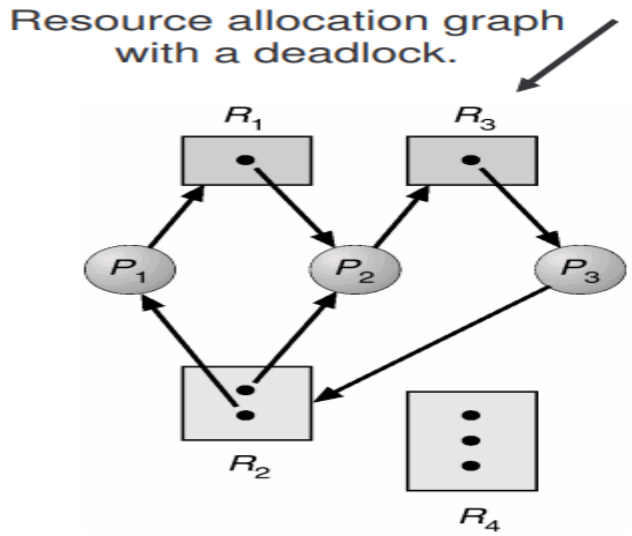
Process A waits for Process B waits for Process C .... waits for Process A

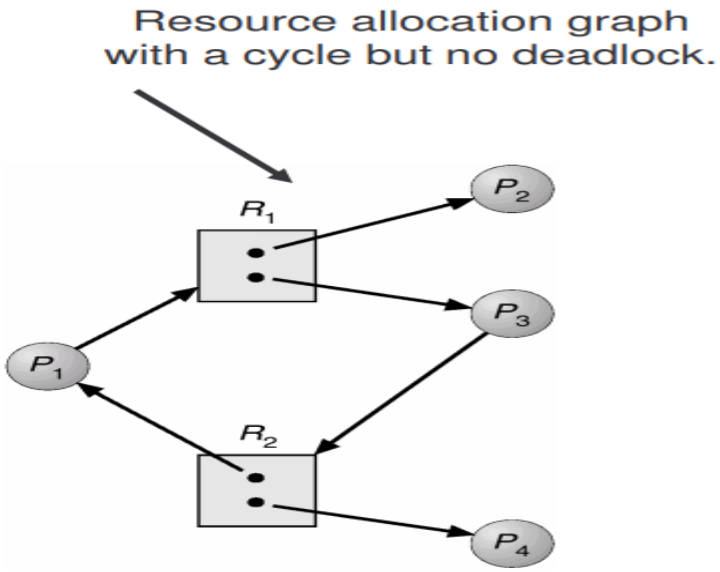
DEADLOCK IN RESOURCE-ALLOCATION GRAPH

- If the graph contains no cycles, then no process is deadlocked. •
- If there is a cycle, then:

a) If resource types have multiple instances, then deadlock MAY exist.

b) If each resource type has 1 instance, then deadlock has occurred.





DEADLOCK IN C#

Lock Method

- The ability to hold one resource and request another, In C#, this is similar to locking on one object and then locking on another before releasing the first lock, for example:

```
lock(a)
    {
      lock(b)
          {
            ....
          }
    }
```

EXAMPLE FOR DEADLOCK

```
using System;
using System.Threading;
using System.Threading.Tasks;
namespace deadlock1
{
    class Program
    {       static void Main(string[] args)
        {    object lock1 = new object();
             object lock2 = new object();
             Console.WriteLine("Starting...");
             var task1 = Task.Run(() =>
             {
                lock (lock1)
                {
                   Thread.Sleep(1000);
                   lock (lock2)
                   {
                      Console.WriteLine("Finished Thread 1");
                   }
                }
             });
             var task2 = Task.Run(() =>
                    {
                       lock (lock2)
                       {
                          Thread.Sleep(1000);
                          lock (lock1)
                          {
                             Console.WriteLine("Finished Thread 2");
                          }
                       }
                    });

                    Task.WaitAll(task1, task2);
                    Console.WriteLine("Finished...");


        }
    }
}
```