Solution pseudocode:

Producer

```
while (true) {
 /* produce an item in next produced */
while (counter == BUFFER_SIZE) ;
/* do nothing */
 buffer[in] = next_produced;
in = (in + 1) % BUFFER_SIZE;
counter++;
 }
```

consumer

```
while (true) {
while (counter == 0) ;
/* do nothing */
next_consumed = buffer[out];
out = (out + 1) % BUFFER_SIZE;
counter--;
/* consume the item in next consumed */
}
```

****

Deadlock

A resource allocation graph tracks which resource is held by which process and which process is waiting for a resource of a particular type. If a process is using a resource, a directed link is formed from the resource node to the process node. If a process is requesting a resource, a directed link is formed from the process node to the resource node. If there is a cycle in the Resource Allocation Graph then the processes will deadlock. In the following Resource Allocation Graph R stands for a resource and P stands for a process. Here one of the deadlock cycles is

There are 3 processes $P1$, $P2$, $P3$

• There are 4 resources:

$R1$ (1 instance) , $R2$ (2 instances) , $R3$ (1 instance) , $R4$ (3 instances) . $R4$ (3 instances)

$P1$ holding 1 instance from $R2$

$P1$ request 1 instance from $R1$

$P2$ holding 1 instance from $R2$

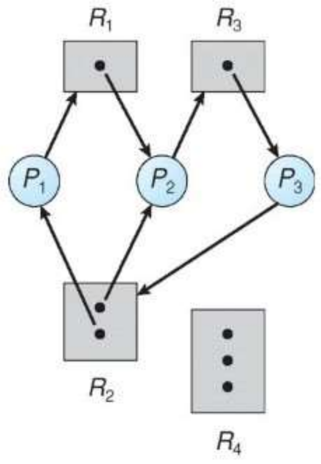
$P2$ holding 1 instance from $R1$

$P2$ request 1 instance from $R3$

$P3$ holding 1 instance from $R3$

$P3$ request 1 instance from $R2$

And this is a deadlock:



Because $P1$ is waiting for $P2$ and,

$P2$ is waiting for $P3$ and,

$P3$ is waiting for $P1$ and $P2$

The solution by deadlock prevention

>>hold and wait :

A process requests a resource only if it does not hold any other resources.

A process requests and is allocated all its resources before it begins execution

>>No Preemption

If a process holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

Preempted resources are added to the list of resources for which the process is waiting.

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

>>Circular Wait

Impose a total ordering on all resource types, and Require that each process requests resources in an increasing order of enumeration.

****

Starvation

Explanation for real world application:

The bound buffer idea for real world is the producer is the waiter will produce for example meals to customer and the consumer will consume this meals until buffer be empty.

The problem that the producer will produce foods with out stop,the consumer not to able to finish the food and the leftover food will go to waste as soon as if the producer will just produce limited amount of food so, the supply of food will eventually run out and if it happening the consumer will be in starvation case.

The solution of this problem:

that that the producer will produce food until the buffer full ,and the consumer will eat until the buffer is empty,,,,,,,,,,, and the consumer will be wait for the buffer to be full again ,,

when the buffer empty again the producer will produce food again so, the process will be repeat .