RR Eg:

Consider 3 processes as shown table:

Processes	CPU time
P1	10
P2	5
P3	2

Q= 2ms

GANTT chart:

```
| P1 | P2 | P3 | P1 | P2 | P1 | P2 | P1 | P1 |
0    2    4    6    8    10   12   13   15   17
```

Waiting time for P1: 0+ (6-2) + (10-8) + (13-12) =7ms
Waiting time for P2: 2+ (8-4) + (12-10) =8ms
Waiting time for P3:4ms

⇒ AWT=(7+8+4)/3=19/3

TAT for P1=17
TAT for P2=13
TAT for P3=6

⇒ ATAT=(17+13+6)/3=12ms

Q. Consider 3 processes as shown in table:

Processes	CPU time
P1	30
P2	6
P3	8

Q= 5ms

Find: AWT, ATAT

```
| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P1 | P1 | P1 |
0    5    10   15   20   21   24   29   34   39   44
```

Waiting time for P1: 0+ (15-5) + (29-24) =14ms
Waiting time for P2: 5+ 10 =15ms
Waiting time for P3:10+6=16ms

⇒ AWT=(14+15+16)/3=15ms

TAT for P1=44
TAT for P2=21
TAT for P3=24

⇒ ATAT=(44+21+24)/3=29.6ms

Q.

Processes	**CPU time**
P1	**10**
P2	**15**
P3	**7**
P4	**9**

Q= 3ms

Find: AWT, ATAT

Solution,

```
 | P1 | P2 | P3 | P4 | P1 | P2 | P3 | P4 | P1 | P2 | P3 | P4 | P1 | P2 | P2 |
0     3     6     9    12    15   18    21   24    27    30   31   34    35    38   41
```

Waiting time for P1: 0+9+9+7 =25ms
Waiting time for P2: 3+9+9+5 =26ms
Waiting time for P3:6+9+9=24ms
Waiting time for P4:9+9+7=25ms

- AWT=(25+26+24+25)/4=25ms

TAT for P1=35
TAT for P2=41
TAT for P3=31
TAT for P4=34

- ATAT=(35+41+31+34)/4=35.25ms

Process Synchronization

- The processes are changing context from an executing program to an interrupt handler is called as context switching.
- Tasking may be (i) Implicit or (ii) Explicit
- Implicit tasking means processes are define system.
- Explicit tasking means processes are defined by the programmer or user application.

Let C, L, S & R are four instructions each requiring 3, 4, 2 & 3 unit of time we execute or 2 way of executing as:

Way I: Sequential process:

```
While (true)
{
   C; 3
   L; 4
   S; 2
   R; 3
}
```

Way II: Multitasking

$$\begin{tikzcd}
 & C1 \arrow[dl] \arrow[dr] & & \\
L1 \arrow[dr] & & S1 \arrow[dl] \arrow[dr] & \\
 & C2 \arrow[dl] \arrow[dr] & & R1 \arrow[dl] \\
L2 \arrow[dr] & & S2 \arrow[dl] \arrow[dr] & \\
 & C3 & & R2
\end{tikzcd}$$

So, we use race condition to make process synchronization to make performance of system and application system.

Race Condition/Race Problem

- When two or more process are reading or writing some shared data and the final result depends upon the sequence in which they access the data such process is called **race condition**.
- Such data are passed in critical condition or region to read/write and share data.
- Critical region is a section of code or a set of operation in which process may change shared variables, updating common files or a table, etc.

Condition for race condition

- No two processes may be simultaneously inside their critical region.
- No assumption may be made about speed or no. of CPU.
- No process running outside its critical region may block other process.
- No process should have to wait forever to enter its critical region.

Mutual Exclusion Using Critical Region

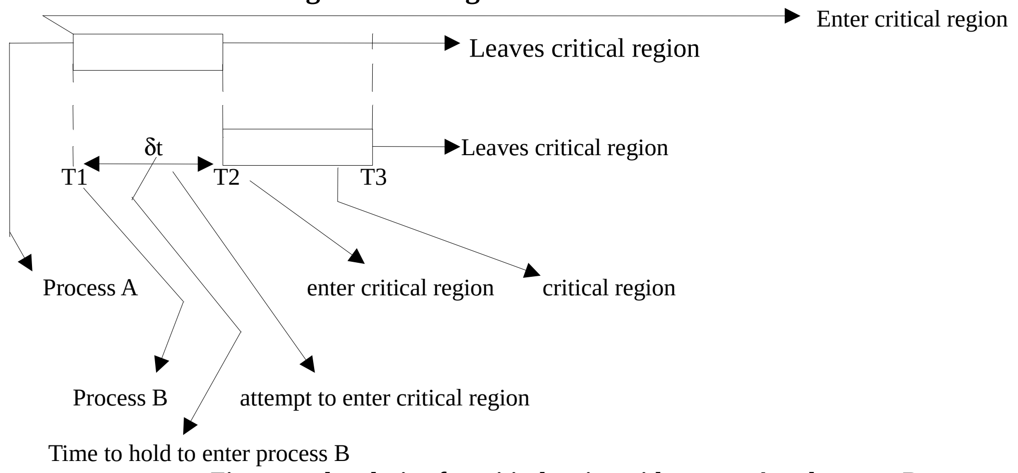


Fig: mutual exclusion for critical region with process A and process B

If one process is in critical region other competing process must be excluded to enter their critical region.

- According to above figure two process A and process B are m mutual exclusion which enter critical region. Consider process A enter first critical region which take its time to execute for certain time, during this time process B has to wait δt time i.e. (t2-t1) to enter critical region. This time called as attempt to enter critical region. After process A leaves from critical region process B take critical region to execute its job within given time.

Critical section Algorithm

- Some of the algorithm related to CS to most the mutual exclusion, program, bounded waiting and no. assumption.

I. Peterson's Algorithm

- In 1981, GL Peterson discovered much similar way to achieve mutual exclusion.

- **Algorithm**

```
FALSE=0;
TRUE=1;
N=2;
Turn;
Interested [process];
 Enter in region
Other;
Other=1-process;
Interested [process] =TRUE;
Turns==process && interested [other] ==TRUE
Leave _region (int process)
Interested [process] =FALSE;
```

- Initially, neither process is in true state.
  Now consider process 0 calls enter region. It indicates its interest by setting array Element and sets turn to 0. Since process 1 is not interested, enter region return FALSE immediately. If process 1 now makes a call to enter region, it will hang there until interested [0] goes to FALSE.

- Now consider the case that processes call enter region almost simultaneously. Both will store their process number in turn. The first one is overwritten and lost data and initial process run for zero (0) time and exit from critical region.

Q. What is process? Explain different state of a process with the help of state diagram?

Swap Algorithm

- Void swap(Boolean * key, Boolean *fly)

```
{
        Boolean temp=*flag;
        *flag=*key;
        *key=temp;
}
        Boolean key;
Do
```

```
{
        Key=true;
        While (key)
          {
             Swap (&key, flag);
        }
          Cs;
          Flag=false;
          Rs;
        }
        While (1);
```

Dead lock

- Computer system will support no of resources like printer, plotter, tap driver, hard disk, files, etc.
- Os have the ability to grant the exclusive access of resources to a process temporarily.
- The permanent blocking of a set of processes that either complete for system resource or communicate with each other is a **Dead lock.**
- It is also called as a situation where a process or a set of processes are blocked, waiting for some resources that is held by some other waiting processes.

Eg:

- Real life situation, consider a four way crossing of traffic:

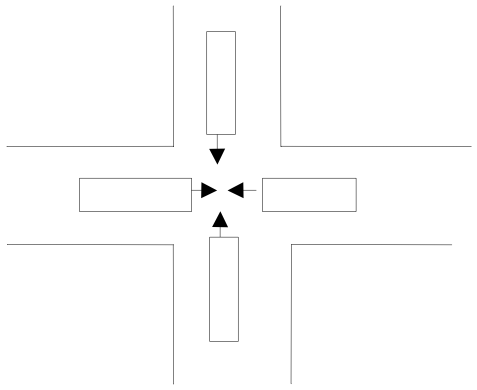
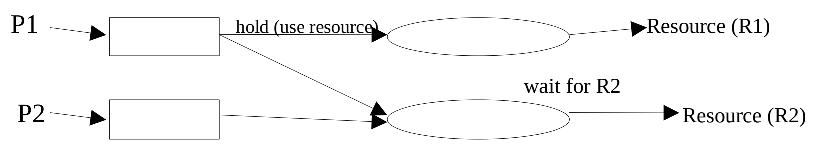


Fig: real- life situation for dead lock condition

Condition for Dead Lock

- Dead lock is an undesirable state of the system. There are four condition as follows:
  - **Mutual condition**
    - A resource can be used by only one process at a time. If another process mutually request for that resource then requesting process must delay until the resource has been released.
  - **Hold and Wait**
    - Some process must be holding some resources in a non-sharable or non pre-emptive to acquire some more resource, which is currently held by other process in non sharable mode or state.


P1
hold (use resource)
Resource (R1)
wait for R2
P2
Resource (R2)