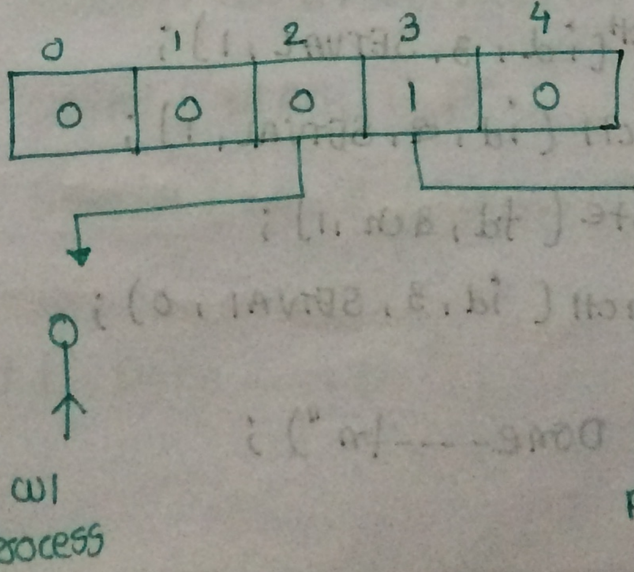
```
    fd= open (" data ", O_RDWR | O_APPEND| O_CREAT ; 0644);
if ( fd <0)
{
      Perror (" open ");
      return ;
}
V. Sem_num = a ;
V. sem_op = 0 ;
V. sem_ flg = 0;
        Printf (" Before ---- \n ");
        Semop (id , &V, 1) ;
/* Start to Enter the Critical Section */
        semctl (id, a, SETVAL , 1) ;
        Printf (" After ---- \n ");
        for (i=0 ; a[i] ; i++)
        {
            write (fd, &a[i] ,1) ;
            Sleep (1) ;
        }
        Printf (" Done ---- \n") ;
        semctl ( id, a, SETVAL,0) ;
}
```

Program :- 2 process access using a semaphore

Value.



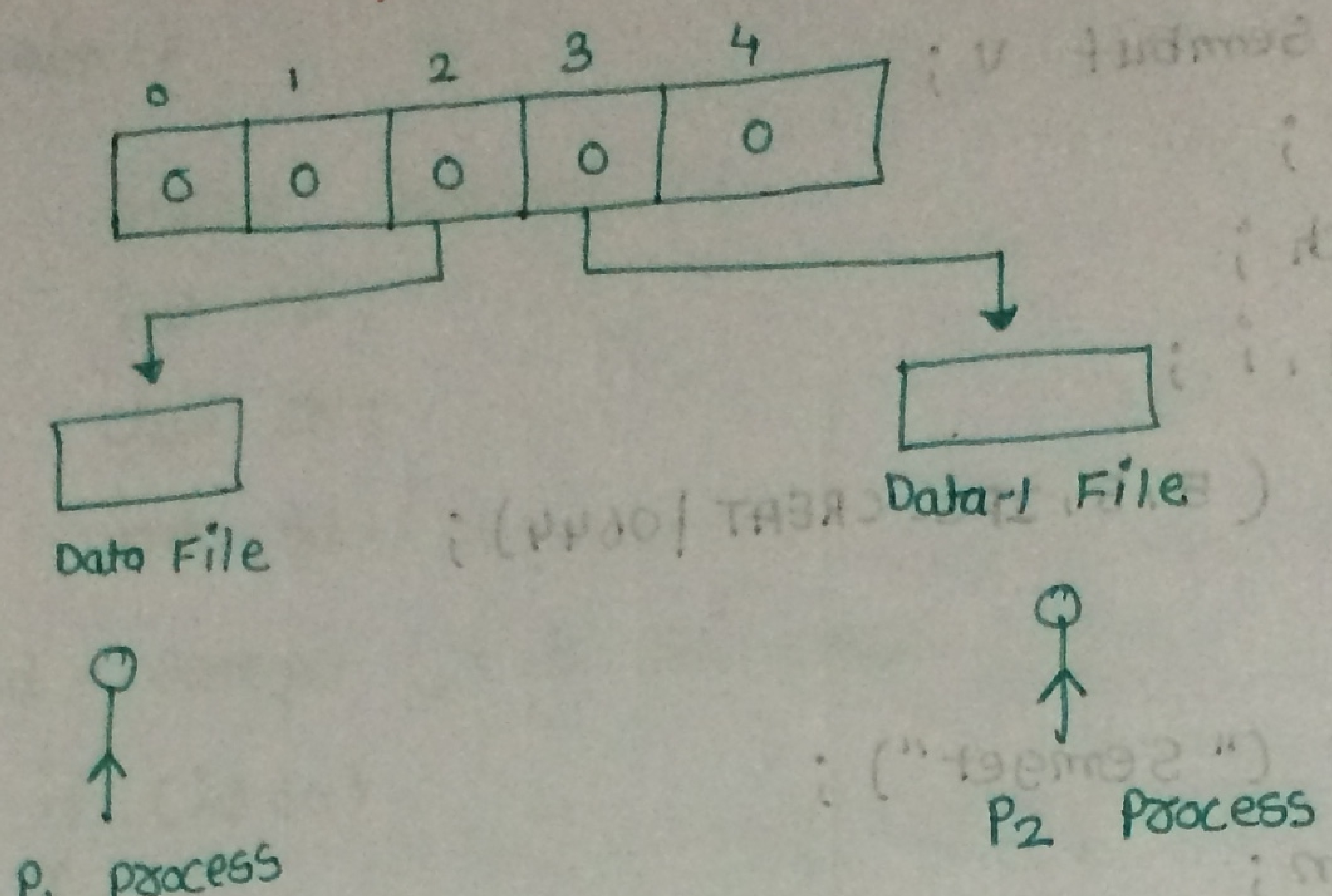
W1.c

```
#include "header.h"
main()
{
    struct sembuf V;
    int id;
    char ch;
    int fd, i;
id = semget (5,5, IPC_CREAT |0644);
 if (id<0)
 {
     perror ("semget");
     return;
 }
fd = open ("data", O_RDWR |O_APPEND |O_CREAT, 0644);
    if (fd<0)
    {
        perror("open");
        return;
    }
V.sem_num = 2;
V.sem_op = 0;
V.sem_flg = 0;
    for (ch = 'a'; ch<='z'; ch++)
    {
        semop(id, &V, 1);
        semctl(id, 3, SETVAL, 1);
        semctl (id, 2, SETVAL, 1);
        write (fd, &ch, 1);
        semctl (id, 3, SETVAL, 0);
    }
    Printf(" Done----\n");
}
```

W2.c

```
#include "header.h"
main()
{
    struct sembuf V;
    int id;
    char ch;
    int fd, i;
id = semget (5,5, IPC_CREAT|0644);
if(id<0)
{
    Perror ("Semget");
    return;
}
fd = open ("data", O_RDWR|O_APPEND|O_CREAT, 0644);
    if (fd<0)
    {
        Perror ("open");
        return;
    }
    V.sem_num = 3;
    V.sem_op = 0;
    V.sem_flg = 0;
for (ch='A'; ch<='Z'; ch++)
{
        Semop(id, &V, 1);
        Semctl(id, 2, SETVAL, 1);
        Semctl(id, 3, SETVAL, 1);
        write (fd, &ch, 1);
        Semctl (id, 3, SETVAL, 0);
    }
Printf (" Done----\n");
}
```

program Two different - different file access using Semaphore Then Dead - Lock is occurs.



→ avoid the dead_Lock follow the below step :-

```
struct Sembuf V[2];

V[0].Sem_num = 2;        ||   V[1].Sem_num = 3;
V[0].Sem_op = 0;         ||   V[1].Sem_op = 0;
V[0].Sem_flg = 0;        ||   V[1].Sem_flg = 0;

Semop(id, &V, 2);
```

2 means both Semaphore.
(no. of sembuf structure Variable).

→ both Semaphore are free than processes are access it. (one is free and other is not free then wait for other process and after access it).

access using
is occurs.

eloop Step :-

um = 3 ;
p = 0 ;
lg = 0 ;

).

s are
not free
access it).

***** Threads *****

⇒ It is a light wait process.

⇒ It is single execution is stream in a process which can be independently schedul by Kernel and share the same address space.

Que :- what is need of Creating threads ?

① Contax Switch betn thread are faster compare to the contax Switch between to processes.

(NOTE :- Loading and unloading the process is called Contax Switch).

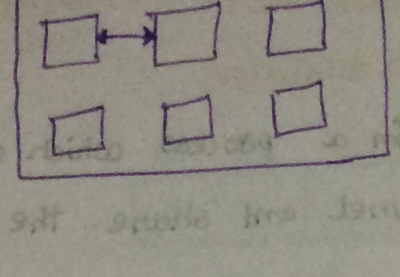
② Contax Switching between two thread then common thing is not Loaded and unloaded.

⇒ whenever process is created into the PCB is [created into the PCB is created into] in threads TCB (Thread Control Block) is created.

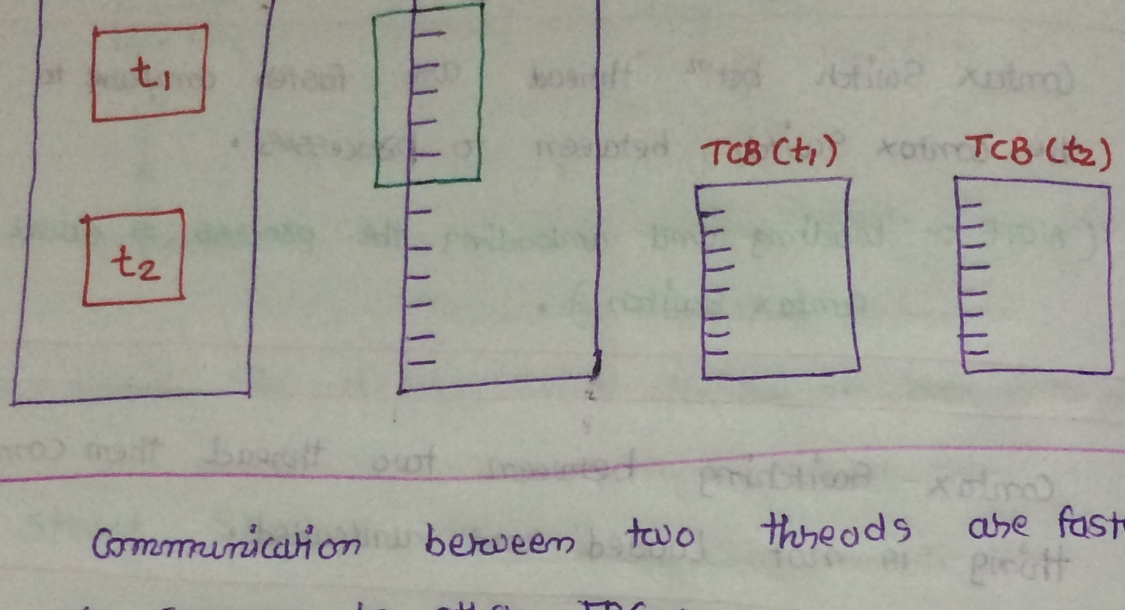
```
main()
{
    abc();
    def();
}

Void abc (Void)
{
    while(1);
    Printf("In abc---\n");
}

Void def (Void)
{
    while(1);
    Printf("In def--\n");
}
```

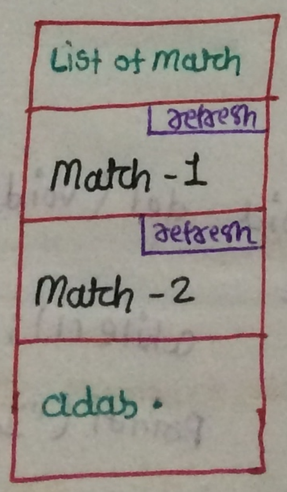


Context switch time is reduce so contox switch between thread is faster.



(3) Communication between two threads are faster as compare to other IPC.

⇒ no need to create IPC only we have to take global variable for two threads which are running in same address spaces.



⇒ 1- process but in 1 process multiple light wait process all process working your work.

⇒ To do a multiple process in process instead of process creat threads.

* disadvant

(1) To create

(2) Synchroni

⇒ It is difficult the parent a

⇒ Sharing inf is just a

⇒ Thread Cre ten-time fa

* following

1) PID &

2) process

3) controlli

4) process

5) open

6) record

7) signa

8) interv

9) resour

10) CPU

11) resou

12) nice

* follo

1) th

2)

text switch time is
uce so contax switch
een thread is faster.

TCB (t2)

ods are faster

ave to take global
running in

1 process
wait process
orting your

ple process
tead of
eads.

(*) disadvantages :-

(1) To create a threads, we need process.

(2) Synchronization problem.

⇒ It is difficult to share information between processes. Since the parent and child don't share memory.

⇒ Sharing information between threads easy and fast. It is just a matter of copying data into shared variables.

⇒ Thread creation is faster than process creation - typically ten-time faster or better.

(*) following attributes not shared.

1) PID & PPID
2) process group ID and session ID
3) Controlling terminal
4) process credentials
5) open file descriptors
6) record locks
7) signal dispositions
8) interval timers
9) resource limits
10) CPU time consumed
11) resources consumed
12) nice value

(*) following attributes are shared.

1) thread ID
2) signal mask
3) thread-specific data
4) alternate signal stack

5) The errno Variable
6) floating point environment.
7) CPU affinity
8) Capabilities
9) Stack

(*) Pthreads data types :-

1) pthread_t ⟶ Thread identifier
2) pthread_mutex_t ⟶ Mutex
3) pthread_mutexattr_t ⟶ Mutex attributes object
4) pthread_cond_t ⟶ Condition Variable
5) pthread_condattr_t ⟶ Condition Variable attributes object
6) Pthread_key_t ⟶ Key for thread-specific data
7) pthread_once_t ⟶ one time initialization
8) pthread_attr_t ⟶ Thread attributes object.

① pthread_create();
② pthread_self();
③ pthread_Join();
④ pthread_Exit();

$ man pthread_create()

pthread_create ⟶ Create a New threads.

#include <pthread.h>

int pthread_create (pthread_t *thread, const pthread_attr_t *attr, Void *(*start routine)(Void *), Void *arg);

Compile and li
on success →
on error →
→ In 3rd argumen
this function a
* main thread is
is create an

Program-1 :-

```
# include " header
Void * thread-1 (
{
    while(1)
    Printf (" In
}
Void * thread-2
{
    while(1)
    Printf (" I
}
main()
{
    pthread_t
    Pthread_cr
    Pthread_cre
    While (1);
}
```