Compile and link with -pthread

on Success → 0

on error → return errno.

→ In 3rd argument which function make as a thread this function address pass.

* main thread is running then so sub thread is create as well as running.

Program-1 :-

```
# include " header.h "
Void * thread-1 (Void *p)
{
    while (1)
    Printf (" In t1-----\n") ;
}
Void * thread-2 (Void *p)
{
    while (1)
    Printf (" In t2 --- \n") ;
}
main()
{
    Pthread-t  t1,t2 ;
    Pthread-Create (&t1, NULL, thread-1 , NULL) ;
    Pthread-Create (&t2, NULL, thread-2 , NULL) ;

while (1) ;
}
```

① using 8th :-

main() will receive the thread id of first (sub) thread. in th created thread if we get.

② second argument :-

always put as a NULL.

③ 3rd argument :-

address of function.

Program-2

```
#include "header.h"
Void *thread_1 (void *p)
{
    Printf (" In th ---- %u\n", Pthread_self());
}
Void *thread_2 (void *p)
{
    while(1)
    Printf (" in t2 ---- \n");
}
main()
{
    Pthread_t  t1, t2;
    Pthread_create (&t1, NULL, thread_1, NULL);
    Printf (" In main ---- %u \n", t1);
    while(1);
}
```

Program-3

```
#include "header.h"
Void *thread_1 (void *p)
{
    Printf (" In t1 ---- %s\n", (Char *)p);
}
```

```
void *thread_2 (void *p)
{
    while (1) {
    Printf (" In t2----\n ");
}
main()
{
    pthread_t t1, t2 ;
    pthread_create ( &t1, NULL, thread_1, "hello" ) ;

    Printf (" In main---- %u\n ", t1);
    while(1);
}
```

## ⊛ $ man pthread_join

pthread_join → join with a terminated thread.

```
# include < pthread.h >

int pthread_join ( pthread_t thread , void ** retval) ;
```

Compile and link with : -pthread

on Success → return 0 ;

on error → return errno

→ The thread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() return immediately.

## ⊛ $ man pthread_exit()

pthread_exit → terminate calling thread.

```
# include < pthread.h>

Void pthread_exit (void * retval);
```

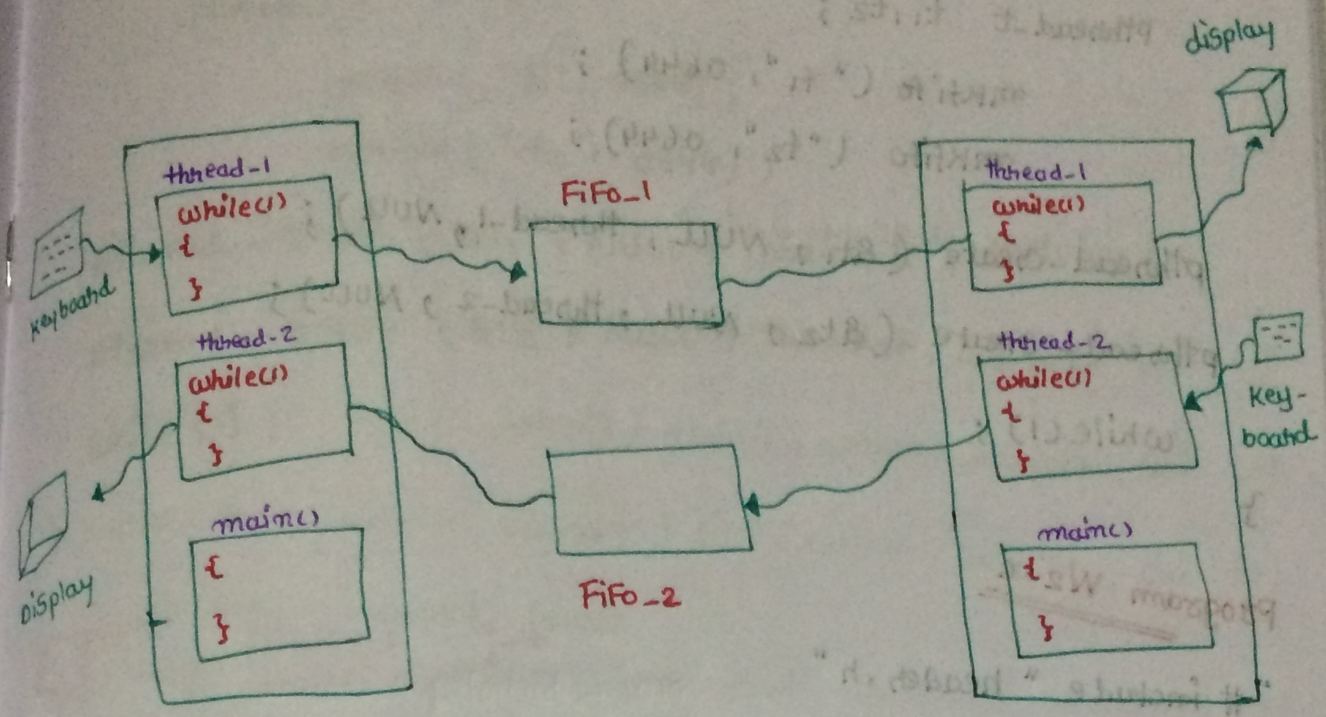
Compile and Link with -thread

⇒ The pthread_exit() function terminates the calling thread and returns a value via retval that is available to another thread in the same process that calls pthread_join().

**Program – ***

```
# include "header.h"
Void * thread_1 (void *p)
{
    Printf (" in t1---- %s\n", (char *) p);
    Sleep (10);
    Pthread_exit ("bye");
}
main()
{
    Char *p;
    Pthread_t t1;
    Pthread_create (&t1, NULL, thread_1, "Hello");
    Printf (" In main t1 = %u\n", t1);
    printf (" Before----\n");
    pthread_join (t1, &p);
    Pthread printf (After ---- %s\n", p);
}
```

## Full - duplex System :-




## Program W1.C :-

```
#include "header.h"

void * thread_1 (void *p)
{
    char a[20];
    int fd;
        fd = open ("f1", O_WRONLY);
        while(1)
        {
            scanf (" %s", a);
            write ( fd, a, strlen(a)+1);
        }
}
void * thread_2 (void *p)
{
    char b[20];
    int fd;
        fd = open ("f2", O_RDONLY);
        while(1)
        {
            read (fd, b, sizeof(b)); printf("Data : %s\n",b);
        }
}
```

```
main()
{
    pthread_t  t1,t2;
        mkfifo ("f1", 0644);
        mkfifo ("f2", 0644);
  pthread_create (&t1, NULL, thread_1, NULL);
  pthread_create (&t2, NULL, thread_2, NULL);
   while(1);
}
```

Program W2.c

```
# include " header.h"
Void *thread_1 (void *p)
{
    char a[20];
    int fd;
        fd = open ("f2", O_WRONLY);
        while(1)
        {
            Scanf ("%s", a);
            write (fd, a, strlen(a)+1);
        }
  }
Void *thread_2 (Void *p)
{
    char b[20];
    int fd;
        fd = open ("f1", O_RDONLY);
        while(1)
        {
            read (fd, b, sizeof(b));
            Printf (" Data : %s\n", b);
        }
}
```

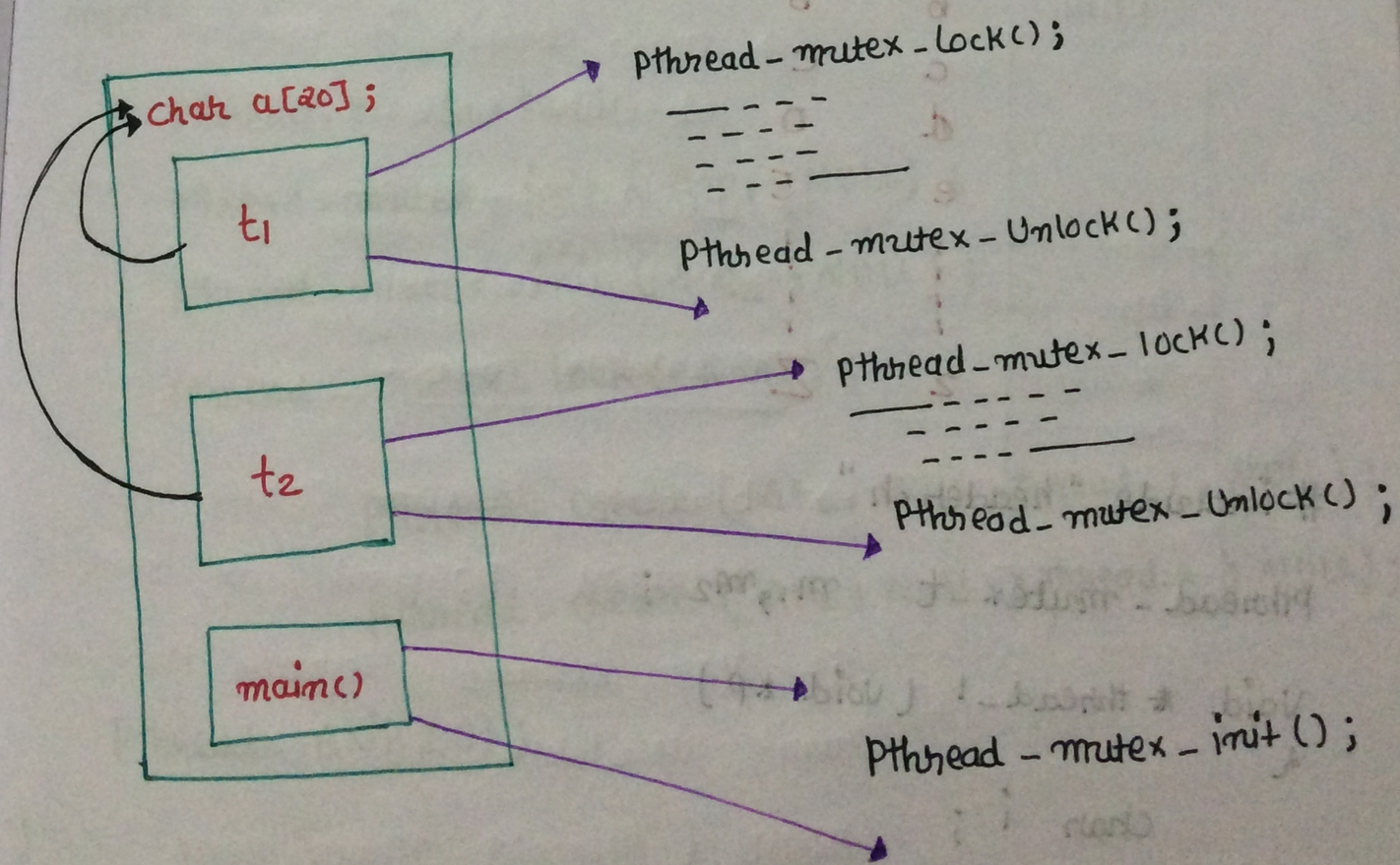
```
main()
{
    pthread_t  t1,t2 ;
        mkfifo ("f1", 0644) ;
        mkfifo ("f2", 0644) ;
pthread_create (&t1 , NULL , thread_1 , NULL) ;
pthread_create (&t2 , NULL , thread_2 , NULL) ;
while (1) ;
}
```

## Mutex :→

1. pthread_mutex_init() ;
2. pthread_mutex_lock() ;
3. pthread_mutex_Unlock() ;

Thread Synchronization is achive by mutex.

**Mutex :-** Mutual exechusive



① **Semaphore** :- It is a signally mechanism.

② **mutex** :- It is called as a locking mechanism.

```
#include <pthread.h>
```

① `int pthread_mutex_lock (pthread_mutex_t *mutex);`

② `int pthread_mutex_unlock (pthread_mutex_t *mutex);`

③ `int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);`

> Return 0 on success, or a positive error number on error.

**Program** :- In a monitor screen

```
a   A
b   B
c   C
d   D
e   E
:   :
z   Z
```

```
#include "header.h"
pthread_mutex_t m1,m2;
Void *thread_1 (Void *p)
{
    char i;
```