Mutex

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void* trythis(void *arg)
{
pthread_mutex_lock(&lock);
unsigned long i = 0;
counter += 1;
printf("\n Job %d has started\n", counter);
for(i=0; i<(0xFFFFFFFF);i++);
printf("\n Job %d has finished\n", counter);
pthread_mutex_unlock(&lock);
return NULL;
}
int main(void)
{
int i = 0;
int error;
if (pthread_mutex_init(&lock, NULL) != 0)
{
printf("\n mutex init has failed\n");
return 1;
}
```

```
while(i < 2)
{
error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
if (error != 0)
printf("\nThread can't be created :[%s]", strerror(error));
i++;
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_mutex_destroy(&lock);
return 0;
```

} I the above code :

- A mutex is initialized in the beginning of the main function.
- The same mutex is locked in the 'trythis()' function while using the shared resource 'counter'.
- At the end of the function 'trythis()' the same mutex is unlocked.
- At the end of the main function when both the threads are done, the mutex is destroyed.
- Output :
  Job 1 started
  Job 1 finished
  Job 2 started
  Job 2 finished

Semaphore

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
/* Global variables */
int x = 0;
sem_t m;
/* Thread function */
```

```
void *thread(void *arg)
{
/* critical section */
sem_wait(&m); /* lock the mutex m */
x = x + 1;
sem_post(&m); /* unlock the mutex m */
}
void main ()
{
pthread_t tid[2];
int i;
/* semaphore m should be initialized by 1 */
if (sem_init(&m, 0, 1) == -1) {
perror("Could not initialize mylock semaphore");
exit(2);
}
/* create Two threads */
for (i=0; i<2; i++)
{
if (pthread_create(&tid[i], NULL, thread, NULL) < 0) {
perror("Error: thread cannot be created");
exit(1);
}
}
/* wait for all created thread to terminate */
for (i=0; i<2; i++) pthread_join(tid[i], NULL);
printf("Final value of x is %d\n", x);
exit(0);
}
```