BITS, Pilani - Hyderabad Campus
Operating Systems (CS F372)

Tutorial – 7

In this tutorial, you will learn about synchronization mechanisms - Mutex and Semaphores.

Mutex:

Threads use mutex for exclusive access to a shared resource.

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

#define N 5
int sum = 0;
int inc = 5;

// it may work without initialization, but there is no guarantee, so do it
// can be done at run time too; see details after the code
pthread_mutex_t lock= PTHREAD_MUTEX_INITIALIZER;
void *fun (void *val)
{
  pthread_mutex_lock (&lock);
  sum += inc;
  printf ("Value: %d\n", sum);
  inc += 5;
  pthread_mutex_unlock (&lock);

  pthread_exit (NULL);
}

int main (int argc, char* argv[]) {
  pthread_t t[N];
  int errcode, i;
  for (i = 0; i < N; i ++) {
      if (pthread_create (&t[i], NULL, fun, NULL)) {
            printf ("Error creating thread\n");
             return EXIT_FAILURE;
       }
  }
  for (i = 0; i < N; i ++) {
      pthread_join (t[i], NULL);
  }
  return 0;
}
```

Pthread mutex type is defined in /usr/include/bits/pthreadtypes.h (check it)

```
typedef union
{
  struct __pthread_mutex_s
  {
    int __lock;                    //locked? 0/1. Anyone waiting?0/2
    unsigned int __count;          // see recursive kind of locks below; counts it
    int __owner;
#ifdef __x86_64__
    unsigned int __nusers;
#endif
    /* KIND must stay at this position in the structure to maintain
```

```
       binary compatibility.  */
    int __kind;                // kind of mutex
#ifdef __x86_64__
    int __spins;
    __pthread_list_t __list;   // doubly linked list next and prev pointers
# define __PTHREAD_MUTEX_HAVE_PREV      1
#else
    unsigned int __nusers;
    __extension__ union
    {
      int __spins;
      __pthread_slist_t __list;
    };
#endif
  } __data;
  char __size[__SIZEOF_PTHREAD_MUTEX_T];
  long int __align;
} pthread_mutex_t;
```

There are different kind of mutexes in Linux, such as fast, error checking, recursive, etc.

- Fast mutex: No error checking performed (see error checking mutex for the kind of checks)

- Error checking mutex: return EDEADLK if you try to lock it again; return EPERM if you try to unlock a mutex not locked by you. You need to initialise to such a type statically as:

  ```
  pthread_mutex_t mutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;
  ```

  or during run time by setting *mutex_attr* and use pthread_mutex_init() as:

  ```
  pthread_mutex_t mutex;
  pthread_mutexattr_t attr;
  pthread_mutexattr_init (&attr);
  pthread_mutexattr_settype (&attr, PTHREAD_MUTEX_ERRORCHECK_NP);
  pthread_mutex_init (&mutex, &attr);
  ```

- Recursive mutex: error checking but you can lock a mutex many times. Keeps count of how many times a mutex has been locked. Must be unlocked the same number of times before it is fully unlocked. Can be initialized statically as:

  ```
  PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
  ```

  or during run time as:

  ```
  pthread_mutex_t mutex;
  pthread_mutexattr_t attr;
  pthread_mutexattr_init (&attr);
  pthread_mutexattr_settype (&attr, PTHREAD_MUTEX_RECURSIVE_NP);
  pthread_mutex_init (&mutex, &attr);
  ```

Use man pages when in doubt. All details are available there for system calls.

Homework:

1. Use Mutex to fix the thread program of lab-6 so that it always works as desired.
2. Find out what kind of mutex is used by default in your system. Use different types of mutexes by

creating them dynamically.

3. Use a mutex that is shared between different processes (not threads). Use a shared memory segment to store to store a variable that is updated concurrently by different processes. Protect it by a shared mutex. You will need the following code to set the shared attribute of the mutex:

```
pthread_mutexattr_init (&attr);
pthread_mutexattr_setpshared (&attr, PTHREAD_PROCESS_SHARED);
pthread_mutex_init (&mutex, &attr);
```

4. Find out how to print the values of pthread_mutex_t variable in your program. You will need to use a debugger.