Objectives

- To demonstrate the creation of threads using the POSIX threads library.
- To perform thread creation by dynamic memory allocation.
- To show synchronization among the threads using Binary Semaphore.

Pre-Lab Theory:

Thread Creation:

#include <pthread.h>

synopsis:

```
int pthread_create(pthread_t *tid,pthread_attr_t * attr,doprocess,void* arg)
```

Description:

tid *: Thread id*

attr_t: *thread attribute object*

doprocess: *Thread routine*

arg *: The argument to be passed to the thread routine*

return: *negative value if thread not created.*

Thread Routine:

Synopsis:

```
void* doprocess(void * arg)
```

Threads Joining:

Synopsis:

```
int pthread_join(pthread_t * tid, void* status)
```

Description:

The main thread (process) will send a join call to the specific thread ID

Thread Id:

pthread_t pthread_self()

Description: *The thread can call this function to display its thread id.*

Threads Synchronization:

Thread synchronization is defined as a mechanism that ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section.

Binary Semaphore:

Binary semaphores are synchronization mechanisms that have integer values that range from 0 (zero) to 1 (one). As a result, this type of semaphore gives a single point of access to a key portion. It signifies that only one thread will have simultaneous access to the critical section.

Mutex:

A mutex is a locking mechanism used to synchronize access to a resource. Only one thread can acquire the mutex. It means there is ownership associated with a mutex, and only the owner can release the lock (mutex).

```
pthread_mutex_t  mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t * mutex);
int pthread_mutex_unlock(pthread_mutex_t * mutex);
```

In-Lab Tasks:

Task 1: *Create a thread with default attributes and the thread will display its thread id.*

steps:

1. *Define a doprocess() function above the main() function that will use pthread_self() function to print the thread id.*
2. *Call the pthread_create() function in main() to create a thread.*
3. *Call the pthread_join() function to send a join call to the newly created thread.*

Output:

Task 2: *Create 5 threads with default attributes and each thread will display its thread ID.*

Steps:

1. *Declare an array of pthread_t tid[5]*
2. *Use the same doprocess() function in Task1*
3. *call pthread_create() function in the for loop*
4. *call pthread_join() to send a join call to the newly created threads*

Output:

Task 3: *Create **n** threads based on input entered by the user and allocate dynamic memory for **n** threads using calloc() function*

```
pthread_t * tid;
```

```
tid = (pthread_t* )calloc(n,sizeof(pthread_t))
```

Steps:

1. *prompt user in main() to enter the number of threads*
2. *In the main() function do the dynamic memory allocation for thread ids*
3. *Same steps as in the task 2*

Code:

Output:

Task 4: *Display the thread sequence number as well before the thread id in task 3 by passing loop variable i to the thread routine(doprocess).*

Steps:

In doprocess() do the necessary changes to typecast the argument

int i;

*i = *((int*)(arg))*

Modified doprocess() function:

write pthread_create() call below:

Output:

Highlight the problem in the output with the reason

Task 5: *Correct the problem in Task 4 by taking a global integer array for sequence numbers.*

Task 5(a) *Static allocation for integer sequence array for 5 threads and initialize it from 1 to 5*

Steps:

1. *Declaration and initialization of integer sequence array*
2. *Pass the sequence number of thread from the integer sequence array with the index of for loop variable i to the doprocess() function.*

integer array declaration and initialization code

Modified pthread_create() call

Output:

Highlight the problem (if any) in the output

Threads Synchronization:

Task 6: *Introduce a global integer counter variable initialized to zero value. Each thread will increment the counter variable and print it with its thread id.*

Steps:

Introduce a delay using a for loop with 1 billion iterations in the doprocess() function

for(int i=0;i<1000000000;i++);

Modified code of doprocess()

Output:

Highlight the problem in the output and ***Identify*** *the* ***critical section***

Task 7: *Modify the program in Task 6 to declare and initialize a global mutex(Binary Semaphore) variable to synchronize the thread for critical section in Task 6*

Steps:

use mutex lock and unlock functions to protect the critical section and write output.

Modified doprocess() function

Output: