1. **Thread synchronization\_ Mutex lock:**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

pthread\_ttid[2];

int counter;

pthread\_mutex\_t lock= PTHREAD\_MUTEX\_INITIALIZER;

void\* trythis(void\* arg)

{

pthread\_mutex\_lock(&lock); **//ensure mutual exclusion between threads**

counter += 1;

printf("\n Job %d has started\n", counter);

sleep(2);

printf("\n Job %d has finished\n", counter);

pthread\_mutex\_unlock(&lock); // **signal the other threads waiting for crictical section access**

return NULL;

}

int main(void)

{

inti = 0;

int error;

/\*

if (pthread\_mutex\_init(&lock, NULL) != 0) {

printf("\n mutexinit 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;

}

1. **Thread\_semaphore:**

**Sem\_wait(semaphore ) to implement wait operation which is studied in theory part**

**Sem\_post(semaphore) to implement signal operation which is studied in theory part**

**Sem\_init()-to initialize semaphore variable**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_tmutex;

void\* thread(void\* arg)

{

//wait

sem\_wait(&mutex);

printf("\nEntered..\n");

//critical section

sleep(4);

//signal

printf("\nJust Exiting...\n");

sem\_post(&mutex);

}

int main()

{

sem\_init(&mutex, 0, 1);

pthread\_t t1,t2;

pthread\_create(&t1,NULL,thread,NULL);

sleep(2);

pthread\_create(&t2,NULL,thread,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

sem\_destroy(&mutex);

return 0;

}

1. **Condition variable:**

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

// Declaration of thread condition variable

pthread\_cond\_t cond1 = PTHREAD\_COND\_INITIALIZER;

// declaring mutex

pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER;

int done = 1;

// Thread function

void\* foo()

{

// acquire a lock

pthread\_mutex\_lock(&lock);

if (done == 1) {

// let's wait on conition variable cond1

done = 2;

printf("Waiting on condition variable cond1\n");

pthread\_cond\_wait(&cond1, &lock);

}

else {

// Let's signal condition variable cond1

printf("Signaling condition variable cond1\n");

pthread\_cond\_signal(&cond1);

}

// release lock

pthread\_mutex\_unlock(&lock);

printf("Returning thread\n");

return NULL;

}

// Driver code

int main()

{

pthread\_t tid1, tid2;

// Create thread 1

pthread\_create(&tid1, NULL, foo, NULL);

// sleep for 1 sec so that thread 1

// would get a chance to run first

sleep(1);

// Create thread 2

pthread\_create(&tid2, NULL, foo, NULL);

// wait for the completion of thread 2

pthread\_join(tid2, NULL);

return 0;

}

**Example 1: Using Condition Variables**  
  
    Example Code - Using Condition Variables  
        This simple example code demonstrates the use of several Pthread condition variable routines. The main routine creates three threads. Two of the threads perform work and update a "count" variable. The third thread waits until the count variable reaches a specified value.   
  
    #include <pthread.h>  
    #include <stdio.h>  
    #include <stdlib.h>  
  
    #define NUM\_THREADS  3  
    #define TCOUNT 10  
    #define COUNT\_LIMIT 12  
  
    int     count = 0;  
    int     thread\_ids[3] = {0,1,2};  
    pthread\_mutex\_t count\_mutex;  
    pthread\_cond\_t count\_threshold\_cv;  
  
    void \*inc\_count(void \*t)   
    {  
      int i;  
      long my\_id = (long)t;  
  
      for (i=0; i<TCOUNT; i++) {  
        pthread\_mutex\_lock(&count\_mutex);  
        count++;  
  
        /\*   
        Check the value of count and signal waiting thread when condition is  
        reached.  Note that this occurs while mutex is locked.   
        \*/  
        if (count == COUNT\_LIMIT) {  
          pthread\_cond\_signal(&count\_threshold\_cv);  
          printf("inc\_count(): thread %ld, count = %d  Threshold reached.\n",   
                 my\_id, count);  
          }  
        printf("inc\_count(): thread %ld, count = %d, unlocking mutex\n",   
           my\_id, count);  
        pthread\_mutex\_unlock(&count\_mutex);  
  
        /\* Do some "work" so threads can alternate on mutex lock \*/  
        sleep(1);  
        }  
      pthread\_exit(NULL);  
    }  
  
    void \*watch\_count(void \*t)   
    {  
      long my\_id = (long)t;  
  
      printf("Starting watch\_count(): thread %ld\n", my\_id);  
  
      /\*  
      Lock mutex and wait for signal.  Note that the pthread\_cond\_wait   
      routine will automatically and atomically unlock mutex while it waits.   
      Also, note that if COUNT\_LIMIT is reached before this routine is run by  
      the waiting thread, the loop will be skipped to prevent pthread\_cond\_wait  
      from never returning.   
      \*/  
      pthread\_mutex\_lock(&count\_mutex);  
      while (count<COUNT\_LIMIT) {  
        pthread\_cond\_wait(&count\_threshold\_cv, &count\_mutex);  
        printf("watch\_count(): thread %ld Condition signal received.\n", my\_id);  
        count += 125;  
        printf("watch\_count(): thread %ld count now = %d.\n", my\_id, count);  
        }  
      pthread\_mutex\_unlock(&count\_mutex);  
      pthread\_exit(NULL);  
    }  
  
    int main (int argc, char \*argv[])  
    {  
      int i, rc;  
      long t1=1, t2=2, t3=3;  
      pthread\_t threads[3];  
      pthread\_attr\_t attr;  
  
      /\* Initialize mutex and condition variable objects \*/  
      pthread\_mutex\_init(&count\_mutex, NULL);  
      pthread\_cond\_init (&count\_threshold\_cv, NULL);  
  
      /\* For portability, explicitly create threads in a joinable state \*/  
      pthread\_attr\_init(&attr);  
      pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);  
      pthread\_create(&threads[0], &attr, watch\_count, (void \*)t1);  
      pthread\_create(&threads[1], &attr, inc\_count, (void \*)t2);  
      pthread\_create(&threads[2], &attr, inc\_count, (void \*)t3);  
  
      /\* Wait for all threads to complete \*/  
      for (i=0; i<NUM\_THREADS; i++) {  
        pthread\_join(threads[i], NULL);  
      }  
      printf ("Main(): Waited on %d  threads. Done.\n", NUM\_THREADS);  
  
      /\* Clean up and exit \*/  
      pthread\_attr\_destroy(&attr);  
      pthread\_mutex\_destroy(&count\_mutex);  
      pthread\_cond\_destroy(&count\_threshold\_cv);  
      pthread\_exit(NULL);  
  
    }  
  
**output:**

Starting watch\_count(): thread 1

inc\_count(): thread 2, count = 1, unlocking mutex

inc\_count(): thread 3, count = 2, unlocking mutex

watch\_count(): thread 1 going into wait...

inc\_count(): thread 3, count = 3, unlocking mutex

inc\_count(): thread 2, count = 4, unlocking mutex

inc\_count(): thread 3, count = 5, unlocking mutex

inc\_count(): thread 2, count = 6, unlocking mutex

inc\_count(): thread 3, count = 7, unlocking mutex

inc\_count(): thread 2, count = 8, unlocking mutex

inc\_count(): thread 3, count = 9, unlocking mutex

inc\_count(): thread 2, count = 10, unlocking mutex

inc\_count(): thread 3, count = 11, unlocking mutex

inc\_count(): thread 2, count = 12 Threshold reached. Just sent signal.

inc\_count(): thread 2, count = 12, unlocking mutex

watch\_count(): thread 1 Condition signal received.

watch\_count(): thread 1 count now = 137.

inc\_count(): thread 3, count = 138, unlocking mutex

inc\_count(): thread 2, count = 139, unlocking mutex

inc\_count(): thread 3, count = 140, unlocking mutex

inc\_count(): thread 2, count = 141, unlocking mutex

inc\_count(): thread 3, count = 142, unlocking mutex

inc\_count(): thread 2, count = 143, unlocking mutex

inc\_count(): thread 3, count = 144, unlocking mutex

inc\_count(): thread 2, count = 145, unlocking mutex

Main(): Waited on 3 threads. Final value of count = 145. Done.