* **POSIX -**#include <pthread.h> **Thread**: A lightweight process that runs concurrently with other threads within the same program. Threads share the same memory space but have separate stacks.
* **pthread\_t**: This is the data type used to declare a thread identifier in C.
* **pthread\_create**: This function is used to create a new thread.
* **pthread\_join**: This function waits for a specific thread to finish execution.
* **pthread\_exit**: This function is called by a thread to terminate its execution.
* **pthread\_mutex\_t**: A mutex is used to ensure that only one thread can access a resource (like a variable or memory) at a time, avoiding race conditions.
* Syntax: pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arg);
  + attr is usually NULL for default attributes.
  + start\_routine is the function that the thread will execute.
  + arg is the argument passed to the thread function.
* Syntax: pthread\_join(pthread\_t thread, void \*\*retval);
  + **\*\*retval:** This is a pointer to a pointer where the exit status of the thread will be stored.
  + If you don't care about the return value of the thread, you can pass NULL.
* //prg1

#include <stdio.h>

#include <pthread.h>

**// Function to be executed by the thread**

void\* thread\_function(void\* arg) {

printf("Hello from the thread!\n");

return NULL;

}

int main() {

pthread\_t thread; **// Declare a thread identifier**

**// Create the thread**

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

**// Wait for the thread to finish**

pthread\_join(thread, NULL);

printf("Thread has finished execution.\n");

return 0;

}

//prg2 ---thread created and passing the value   
#include <stdio.h>

#include <pthread.h>

**// Thread function definition**

void\* thread\_function(void\* arg) {

int\* num = (int\*)arg; **// Cast the void\* to the appropriate type**

printf("Thread received value: %d\n", \*num);

return NULL;

}

int main() {

pthread\_t thread;

int value = 42;

**// Create a thread and pass the address of `value` as the argument**

pthread\_create(&thread, NULL, thread\_function, &value);

**// Wait for the thread to finish**

pthread\_join(thread, NULL);

return 0;

}

//prg3—2 threads created  
#include <stdio.h>

#include <pthread.h>

**// Function to be executed by each thread**

void\* thread\_function(void\* arg) {

int thread\_number = \*((int\*)arg);

printf("Hello from Thread %d!\n", thread\_number);

return NULL;

}

int main() {

pthread\_t thread1, thread2; **// Declare thread identifiers**

int thread\_arg1 = 1;

int thread\_arg2 = 2;

**// Create the first thread**

pthread\_create(&thread1, NULL, thread\_function, &thread\_arg1);

**// Create the second thread**

pthread\_create(&thread2, NULL, thread\_function, &thread\_arg2);

**// Wait for both threads to finish**

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

printf("Both threads have finished execution.\n");

return 0;

}

//prg 4-using for loop multiple thread created

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 2 // Define the number of threads

**// Function to be executed by each thread**

void\* thread\_function(void\* arg) {

int thread\_number = \*((int\*)arg);

printf("Hello from Thread %d!\n", thread\_number);

return NULL;

}

int main() {

pthread\_t threads[NUM\_THREADS**]; // Array to hold thread identifiers**

int thread\_args[NUM\_THREADS]; **// Array to hold thread arguments**

// Create threads using a for loop

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_args[i] = i + 1; **// Set the thread argument (1, 2, ...)**

pthread\_create(&threads[i], NULL, thread\_function, &thread\_args[i]);

}

**// Wait for all threads to finish**

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

printf("All threads have finished execution.\n");

return 0;

}

//prg5 –without capturing return value-join mtd  
#include <stdio.h>

#include <pthread.h>

// Thread function

void\* thread\_function(void\* arg) {

printf("Thread is running...\n");

return NULL;

}

int main() {

pthread\_t thread;

// Create a thread

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

// Wait for the thread to finish

pthread\_join(thread, NULL);

printf("Thread has finished execution.\n");

return 0;

}

//prg 6 –with capturing value -join mtd

#include <stdio.h>

#include <pthread.h>

// Thread function

void\* thread\_function(void\* arg) {

printf("Thread is running...\n");

return (void\*) "Thread finished successfully!";

}

int main() {

pthread\_t thread;

void\* thread\_return\_value;

// Create a thread

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

// Wait for the thread to finish and capture its return value

pthread\_join(thread, &thread\_return\_value);

// Use the return value from the thread

printf("Thread has finished execution. Return value: %s\n", (char\*)thread\_return\_value);

return 0;

}