Thread functions in C/C++

Difficulty Level : Easy • Last Updated : 23 Jun, 2020

In a **Unix/Linux operating system**, the **C/C++ languages** provide the POSIX thread(pthread) standard API(Application program Interface) for all thread related functions. It allows us to create multiple threads for concurrent process flow. It is most effective on multiprocessor or multi-core systems where threads can be implemented on a kernel-level for achieving the speed of execution. Gains can also be found in uni-processor systems by exploiting the latency in IO or other system functions that may halt a process.

We must include the pthread.h header file at the beginning of the script to use all the functions of the pthreads library. To execute the c file, we have to use the -pthread or -lpthread in the command line while compiling the file.

```
cc -pthread file.c or
cc -lpthread file.c
```

The **functions** defined in the **pthreads library** include:

a. ***pthread_create:*** used to create a new thread
   Syntax:

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- **attr:** pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes.
- **start_routine:** pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used.
- **arg:** pointer to void that contains the arguments to the function defined in the earlier argument

b. ***pthread_exit:*** used to terminate a thread
Syntax:

```
void pthread_exit(void *retval);
```

Parameters: This method accepts a mandatory parameter **retval** which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

c. ***pthread_join:*** used to wait for the termination of a thread.
Syntax:

```
int pthread_join(pthread_t th,
                 void **thread_return);
```

Parameter: This method accepts following parameters:

- **th:** thread id of the thread for which the current thread waits.
- **thread_return:** pointer to the location where the exit status of the thread mentioned in th is stored.

d. ***pthread_self:*** used to get the thread id of the current thread.
Syntax:

Syntax:

```
int pthread_equal(pthread_t t1,
                  pthread_t t2);
```

Parameters: This method accepts following parameters:

- t1: the thread id of the first thread
- t2: the thread id of the second thread

f. ***pthread_cancel:*** used to send a cancellation request to a thread
Syntax:

```
int pthread_cancel(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter **thread** which is the thread id of the thread to which cancel request is sent.

g. ***pthread_detach:*** used to detach a thread. A detached thread does not require a thread to join on terminating. The resources of the thread are automatically released after terminating if the thread is detached.
Syntax:

```
int pthread_detach(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter **thread** which is the thread id of the thread that must be detached.

Example: A simple implementation of threads may be as follows:

```
// C program to show thread functions

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

```
    printf("Inside the thread\n");

    // exit the current thread
    pthread_exit(NULL);
}

void fun()
{
    pthread_t ptid;

    // Creating a new thread
    pthread_create(&ptid, NULL, &func, NULL);
    printf("This line may be printed"
           " before thread terminates\n");

    // The following line terminates
    // the thread manually
    // pthread_cancel(ptid);

    // Compare the two threads created
    if(pthread_equal(ptid, pthread_self())
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");

    // Waiting for the created thread to terminate
    pthread_join(ptid, NULL);

    printf("This line will be printed"
           " after thread ends\n");

    pthread_exit(NULL);
}

// Driver code
int main()
{
    fun();
    return 0;
}
```

Output:

Start Your Coding Journey Now!

Explanation: Here two threads of execution are created in the code. The order of the lines of output of the two threads may be interchanged depending upon the thread processed earlier. The main thread waits on the newly created thread for exiting. Therefore, the final line of the output is printed only after the new thread exits. The threads can terminate independently of each other by not using the *pthread_join* function. If we want to terminate the new thread manually, we may use *pthread_cancel* to do it.

Note: If we use exit() instead of **pthread_exit()** to end a thread, the whole process with all associated threads will be terminated even if some of the threads may still be running.

Related Articles

1. Mutex lock for Linux Thread Synchronization
2. Get the stack size and set the stack size of thread attribute in C
3. Print numbers in sequence using thread synchronization
4. C Program to Show Thread Interface and Memory Consistency Errors
5. Static functions in C
6. Write one line functions for strcat() and strcmp()
7. Can Static Functions Be Virtual in C++?
8. Functions that cannot be overloaded in C++

Start Your Coding Journey Now!

Like 42

Next

Multithreading in C

Article Contributed By :

anindo_7
@anindo_7

Vote for difficulty
Current difficulty : Easy

Easy Normal Medium Hard Expert

Improved By : Akanksha_Rai

Article Tags : C-Functions, C Language, Linux-Unix

Improve Article Report Issue

Start Your Coding Journey Now!

feedback@geeksforgeeks.org

Start Your Coding Journey Now!

Django Tutorial	Improve an Article
HTML	Pick Topics to Write
JavaScript	Write Interview Experience
Bootstrap	Internships
ReactJS	Video Internship
NodeJS	

Please Login To Continue ×

[✓ Sign In] Sign Up

account_circle Username or email

lock Password

[✓ Remember me] Forgot Password

Sign In

email E-mail

lock Password

business Institution/Organization

Sign Up

or
Google
Facebook

LinkedIn
GitHub

Why Create an Account?

By creating this account, you agree to our Privacy Policy & Cookie Policy.
Please enter your email address or userHandle.

account_circle Username/Email

Back to Login

Reset Password