Multithreading in C++

Difficulty Level : Easy • Last Updated : 18 Jan, 2023

Multithreading is a feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU. Each part of such a program is called a thread. So, threads are lightweight processes within a process.

Multithreading support was introduced in C++11. Prior to C++11, we had to use POSIX threads or <pthreads> library. While this library did the job the lack of any standard language-provided feature set caused serious portability issues. C++ 11 did away with all that and gave us **std::thread**. The thread classes and related functions are defined in the **<thread>** header file.

Syntax:

```
std::thread thread_object (callable);
```

std::thread is the thread class that represents a single thread in C++. To start a thread we simply need to create a new thread object and pass the executing code to be called (i.e, a callable object) into the constructor of the object. Once the object is created a new thread is launched which will execute the code specified in callable. A callable can be either of the three

- **A Function Pointer**
- **A Function Object**
- **A Lambda Expression**

ter defining the callable, we pass it to the constructor.

A function pointer can be a callable object to pass to the std::thread constructor for initializing a thread. The following code snippet demonstrates how it is done.

Example:

C++

```
void foo(param)
{
  Statements;
}
// The parameters to the function are put after the comma
std::thread thread_obj(foo, params);
```

Launching Thread Using Lambda Expression

std::thread object can also be launched using a lambda expression as a callable. The following code snippet demonstrates how this is done:

C++

```
// Define a lambda expression
auto f = [](params)
{
    Statements;
};

// Pass f and its parameters to thread
// object constructor as
std::thread thread_object(f, params);
```

Launching Thread Using Function Objects

Function Objects or Functions can also be used for launching a thread in C++. The following code snippet demonstrates how it is done:

C++

```
    // Overload () operator
    void operator()(params)
    {
      Statements;
    }
}

// Create thread object
std::thread thread_object(fn_object_class(), params)
```

> ***Note**: We always pass parameters of the callable separately as arguments to the thread constructor.*

Waiting for threads to finish

Once a thread has started we may need to wait for the thread to finish before we can take some action. For instance, if we allocate the task of initializing the GUI of an application to a thread, we need to wait for the thread to finish to ensure that the GUI has loaded properly.

To wait for a thread, use the **std::thread::join()** function. This function makes the current thread wait until the thread identified by ***this** has finished executing.
For instance, to block the main thread until thread t1 has finished we would do:

C++

```
int main()
{
    // Start thread t1
    std::thread t1(callable);

    // Wait for t1 to finish
    t1.join();

    // t1 has finished do other stuff
    Statements;
```

A C++ program is given below. It launches three threads from the main function. Each thread is called using one of the callable objects specified above.

C++

```
// C++ program to demonstrate
// multithreading using three
// different callables.
#include <iostream>
#include <thread>
using namespace std;

// A dummy function
void foo(int Z)
{
  for (int i = 0; i < Z; i++)
  {
    cout << "Thread using function"
            " pointer as callable\n";
  }
}

// A callable object
class thread_obj {
public:
    void operator()(int x)
    {
      for (int i = 0; i < x; i++)
        cout << "Thread using function"
                " object as callable\n";
    }
};

// Driver code
int main()
{
  cout << "Threads 1 and 2 and 3 "
          "operating independently" << endl;

  // This thread is launched by using
  // function pointer as callable
  thread th1(foo, 3);

  // This thread is launched by using
```

```
    // Define a Lambda Expression
    auto f = [](int x)
    {
      for (int i = 0; i < x; i++)
        cout << "Thread using lambda"
                " expression as callable\n";
    };

    // This thread is launched by using
    // lambda expression as callable
    thread th3(f, 3);

    // Wait for the threads to finish
    // Wait for thread t1 to finish
    th1.join();

    // Wait for thread t2 to finish
    th2.join();

    // Wait for thread t3 to finish
    th3.join();

    return 0;
}
```

Output (Machine Dependent)

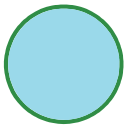
```
Threads 1 and 2 and 3 operating independently
Thread using function pointer as callable
Thread using function pointer as callable
Thread using function pointer as callable
Thread using function object as callable
Thread using function object as callable
Thread using function object as callable
Thread using lambda expression as callable
Thread using lambda expression as callable
Thread using lambda expression as callable
```

Previous Next

Related Articles

1. Packaged Task | Advanced C++ (Multithreading & Multiprocessing)
2. Multithreading in C
3. Maximum Stack Size for C/C+ Program
4. C++ Programming Examples
5. Top C++ STL Interview Questions and Answers
6. How to Use Binder and Bind2nd Functors in C++ STL?
7. 10 C++ Programming Tricks That You Should Know
8. Working and Examples of bind () in C++ STL
9. C++ Program to Print Even Numbers in an Array

Article Contributed By :



Sayan Mahapatra
@Sayan Mahapatra

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : Sayan Mahapatra, DarwinHuang, barkha chauhan, Saksham Sharma 4, harsh_shokeen, suvamtestpurpose, rkbhola5, always_rising

Article Tags : cpp-multithreading, C++

Practice Tags : CPP

Report Issue

Start Your Coding Journey Now!