C++11 Multithreading – Part 1 : Three Different ways to Create Threads

Varun January 20, 2015 C++ 11, c++11 Threads, Multithreading

In this article we will discuss how to create threads in C++11 using std::thread.

Introduction to C++11 Thread Library

Original C++ Standard supported only single thread programming. The new C++ Standard (referred to as C++11 or C++0x) was published in 2011. In C++11 a new thread library is introduced.

Compilers Required:
Linux: gcc 4.8.1 (Complete Concurrency support)
Windows: Visual Studio 2012 and MingW

How to compile on Linux: g++ –std=c++11 sample.cpp -lpthread

Thread Creation in C++11

In every C++ application there is one default main thread i.e. main() function. In C++ 11 we can create additional threads by creating objects

of std::thread class.
Each of the std::thread object can be associated with a thread.

Header Required :

```
#include <thread>
```

What std::thread accepts in constructor ?

We can attach a callback with the std::thread object, that will be executed when this new thread starts. These callbacks can be,

1.) Function Pointer
2.) Function Objects
3.) Lambda functions

Thread objects can be created like this,

```
std::thread thObj(<CALLBACK>);
```

New Thread will start just after the creation of new object and will execute the passed callback in parallel to thread that has started it. Moreover, any thread can wait for another to exit by calling join() function on that thread's object.

Lets look at an example where main thread will create a separate thread. After creating this new thread, main thread will print some data on console and then wait for newly created thread to exit.

Lets implement above using three different callback mechanism,

Creating a thread using Function Pointer

```
#include <iostream>
#include <thread>

void thread_function()
{
    for(int i = 0; i < 10000; i++);
        std::cout<<"thread function Executing"<<std::endl;
}

```

C++11 : std::thread as a member variable in class
C++11 : How to Stop a Thread

Std::Tuple
Lambda Functions
C++11 Utilities
Initializer_list Std::Array
Rvalue References
Smart Pointers
Multithreading
Unordered_Set
Unordered_Map

c++11 : std::tuple Tutorial
c++11 : std::make_tuple Tutorial

Popular Categories

Behavioral Design Patterns (6) Boost Date Time Library (6) Boost Library (17) C++ (107) C++ 11 (59) c++11 Threads (16) C++ Interview Questions (30) collections (17) Dataframe (10) Data Science (10) Debugging (6) Debugging Tutorial (6) Design Patterns (9) dictionary (17) Directories (13) FileHandling (18) Functions (10) gdb (6) gdb

```
int main()
{

    std::thread threadObj(thread_function);
    for(int i = 0; i < 10000; i++);
        std::cout<<"Display From MainThread"<<std::endl;
    threadObj.join();
    std::cout<<"Exit of Main function"<<std::endl;
    return 0;
}
```

Creating a thread using Function Objects

```
#include <iostream>
#include <thread>
class DisplayThread
{
public:
    void operator()()
    {
        for(int i = 0; i < 10000; i++)
            std::cout<<"Display Thread Executing"<<std::endl;
    }
};

int main()
{
    std::thread threadObj( (DisplayThread()) );
    for(int i = 0; i < 10000; i++)
        std::cout<<"Display From Main Thread "<<std::endl;
    std::cout<<"Waiting For Thread to complete"<<std::endl;
    threadObj.join();
    std::cout<<"Exiting from Main Thread"<<std::endl;
    return 0;
}
```

Creating a thread using Lambda functions

```
#include <iostream>
#include <thread>
int main()
{
    int x = 9;
    std::thread threadObj([]{
            for(int i = 0; i < 10000; i++)
                std::cout<<"Display Thread Executing"<<std::endl;
            });

    for(int i = 0; i < 10000; i++)
        std::cout<<"Display From Main Thread"<<std::endl;

    threadObj.join();
    std::cout<<"Exiting from Main Thread"<<std::endl;
    return 0;
}
```

Differentiating between threads

Each of the std::thread object has an associated ID and we can fetch using,

Member function, gives id of associated thread object i.e.

```
std::thread::get_id()
```

To get the identifier for the current thread use,

```
std::this_thread::get_id()
```

If std::thread object does not have an associated thread then get_id() will return a default constructed std::thread::id object i.e. □not any thread.□

std::thread::id is a Object, it can be compared and printed on console too. Let's look at an example,

```
#include <iostream>
#include <thread>
void thread_function()
{
    std::cout<<"Inside Thread :: ID  = "<<std::this_thread::get_id(
}
int main()
{
    std::thread threadObj1(thread_function);
    std::thread threadObj2(thread_function);

    if(threadObj1.get_id() != threadObj2.get_id())
        std::cout<<"Both Threads have different IDs"<<std::endl;

        std::cout<<"From Main Thread :: ID of Thread 1 = "<<threadO
    std::cout<<"From Main Thread :: ID of Thread 2 = "<<threadObj2.

    threadObj1.join();
    threadObj2.join();
    return 0;
}
```

Learn more about multithreading in C++11 / 14

- Modern C++ Concurrency in Depth
- Beginning Modern C++ (C++11/C++14)
- C++ Concurrency in Action

- Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14

Other C++11 Multi-threading Tutorials

C++11 Multi-threading Part 2: Joining and Detaching Threads

C++11 Multi-threading Part 3: Passing Arguments to Threads

C++11 Multi-threading Part 4: Sharing Data & Race Conditions

C++11 Multi-threading Part 5: Fixing Race Conditions using mutex

C++11 Multi-threading Part 6: Need of Event Handling

C++11 Multi-threading Part 7: Using Condition Variables to do Event Handling between threads

C++11 Multi-threading Part 8: std::future and std::promise

C++11 Multithreading – Part 9: std::async Tutorial & Example

If you didn't find what you were looking, then do suggest us in the comments below. We will be more than happy to add that.

Do Subscribe with us for more Articles / Tutorials like this,

C++ 11, C++ 11 Multithreading, c++ 11 Threads, Multithreading, std::thread

2 Comments Already

Sorry, comments are closed for this post

« Design Sorting algorithm using Strategy Design Pattern

C++11 Multithreading – Part 2: Joining and Detaching Threads »

Data Analysis in Python

Dataframe in Pandas
Numpy Arrays

Terms and Conditions

Terms and Conditions Policy

C++ / C++11 Tutorials

C++ Tutorials
C++11 Tutorial
C++ Interview Questions
Multithreading
C++17
STL Tutorials

Design Patterns

Behavioral Design Patterns
- Observer Design Pattern
- State Design Pattern
- Strategy Design Pattern

Structural Design Patterns
- Composite Design Pattern
- Flyweight Design Pattern

Creational Design Patterns

Python tutorials

Strings
Lists
Tuple
Dictionary
Functions
File Handling
Lambda Functions