##### Multi-Threading

“A programmable approach to achieve multi-tasking in known as multi-threading. A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.”

For multi-threading, the Built-in support was first introduced in C++11. But in order to create multi-threaded C++ programs we have to include the Header file <thread>.

Before C++ 11, we had POSIX thread support. But this feature had serious portability issues as it worked only on the Linux or UNIX operating system. Thus from C++ 11 onwards, we have a single class std:: thread which defines all the functionality for threads. The classes and functions are defined in the header file.

Working Of <thread>

Using std:: thread we simply need to create a new thread object and pass it a callable. A callable is an executable code that we want to execute when the thread is running. So whenever we want a new thread, we just create an object of std:: thread and pass a callable as an argument to its constructor.

Once the std::thread object is created, a new thread is launched and the code provided by callable is executed.

**A callable can be defined in three ways.**

**#1) Using The Function Object**

We can use a function object as a callable in the thread object. For using the function object, we need to have a class and in that class, we overload the operator (). This overloaded function contains the code to be executed when the thread is created.

*class functionObject\_class {*

*// Overload () operator*

*void operator()(params)*

*{*

*// code to be executed*

*}*

*};*

*// Create thread object*

*std::thread thread\_object(functionObject\_class (), params)*

Note the way in which the thread object is defined. As the first parameter to the constructor of thread object, we provide the overloaded function and then specify its arguments (params) as the second argument.

**#2) Using Function Pointer**

A callable using function pointer can be defined in the following way.

*void funct\_call(params)*

*{*

*//code to be executed*

*}*

Once we define this function, we can create a thread object with this function as callable, in the following way.

*std::thread thread\_obj(funct\_call, params);*

Note that the arguments (params) passed to the function, are provided after the function name in the thread object.

**#3) Using A Lambda Expression**

We can also have a callable as a lambda expression and pass it to the thread object for execution. The code snippet for the same is shown below.

*// Define a lambda expression*

*auto f = [](params) {*

*// code for execution*

*};*

*std::thread thread\_object(f, params);*

In the above code, we have defined a lambda expression f and we then pass it to the thread object constructor as the first argument followed by its parameters (params) as the second argument.

**std::thread join method**

In some cases, we might want the currently executing thread to finish before we start another action.

A classic example is when we open the GUI application. The moment we open the application, a thread to load and initialize the GUI is started and we cannot perform any action unless loading and initializing is done correctly so as to ensure that the GUI functions properly.

Another reason for joing threads is that, when we are running threads and main function finishes execution but a thread is still running, it will result in an error. All the threads must be terminated before main function is terminated.

The class std::thread provides a join() method which ensures that the current thread (pointed by \*this) finishes first before any other action is taken.

Take the following example,

*int main()*

*{*

*std::thread t1(callable\_code);*

*…..*

*t1.join();*

*…..*

*}*

In the above example, the main function will have to wait to continue until thread t1 finishes. In general, the join function of thread blocks other actions/functionality until the thread calling finishes its execution.

**Example Of Thread:**

We present a complete coding example for creation and execution of the thread in the program shown below.

#include <iostream>

#include <thread>

using namespace std;

// a callable function

void call\_function(int x)

 {

   for (int i = 0; i < x; i++) {

   cout << "This is Thread 1 using function pointer\n";

   }

 }

// a callable object

class thread\_obj {

 public:

   void operator()(int x) {

       for (int i = 0; i < x; i++)

           cout << "This is Thread 2 using function object\n";

   }

};

int main()

{

// Define a Lambda Expression

auto f = [](int x) {

   for (int i = 0; i < x; i++)

   cout << "This is Thread 3 using lambda expression\n";

   };

// launch thread using function pointer as callable

thread th1(call\_function, 3);

// launch thread using function object as callable

thread th2(thread\_obj(), 3);

//launch thread using lambda expression as callable

thread th3(f, 3);

// 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

*This is Thread 1 using function pointer*

*This is Thread 1 using function pointer*

*This is Thread 1 using function pointer*

*This is Thread 3 using lambda expression*

*This is Thread 3 using lambda expression*

*This is Thread 3 using lambda expression*

*This is Thread 2 using function object*

*This is Thread 2 using function object*

*This is Thread 2 using function object*

**Detach()**

If the execution of threads has not yet been joined, then the thread is join-able. And also the thread will not join-able if it is default constructed or is moved or assigned to another thread or join() or detach() member function is called.

The detach() is the member function of class thread, and used to detaches a thread from the parent thread.

So it’s always better to check if the thread is joinable or not. We can check if a thread is joinable or not using the joinable() member function of class thread.

*#include <iostream>*

*#include <thread>*

*using namespace std;*

*void function()*

*{*

*cout<<”Hello world: “<<endl;*

*}*

*main()*

*{*

*//call function using thread Object:*

*thread call\_function(function);*

*if(call\_function.joinable())*

*{*

*//main thread block, until child thread execution finish:*

*call\_function.join();*

*}*

*else*

*cout<<” Call\_function is not join-able at the moment: “<<endl;*

*return 0;*

*}*