**Three Different ways to Create Threads**

## **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 :

|  |  |
| --- | --- |
| 1 | #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,

|  |  |
| --- | --- |
| 1 | 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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <iostream>  #include <thread>    void thread\_function()  {      for(int i = 0; i < 10000; i++);          std::cout<<"thread function Executing"<<std::endl;  }    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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #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.

|  |  |
| --- | --- |
| 1 | std::thread::get\_id() |

To get the identifier for the current thread use,

|  |  |
| --- | --- |
| 1 | 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()<<std::endl;

}

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 = "<<threadObj1.get\_id()<<std::endl;

    std::cout<<"From Main Thread :: ID of Thread 2 = "<<threadObj2.get\_id()<<std::endl;

    threadObj1.join();

    threadObj2.join();

    return 0;

}

#include <iostream>

#include <thread>

#include <vector>

using namespace std;

#define BEG 0

#define CAP 10000

#define THREADNUM 4

int ThreadSumVector(vector<int> v\_l\_int, int beg, int end, int \*res)

{

// int res = 0;

for (int i=beg; i <=end; i++)

\*res = \*res + v\_l\_int[i];

//return res;

}

int main()

{

vector<int> v\_num (CAP, 1);

int res1, res2, res3;

res1 = res2 = res3 = 0;

thread t1(ThreadSumVector, v\_num, 0, 99 , &res1);

thread t2(ThreadSumVector, v\_num, 100, 9000, &res2 );

thread t3(ThreadSumVector, v\_num, 9001, 9999, &res3 );

t1.join();

t2.join();

t3.join();

cout <<"res1 = "<<res1<<endl;

cout <<"res2 = "<<res2<<endl;

cout <<"res3 = "<<res3<<endl;

return 0;

}