DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

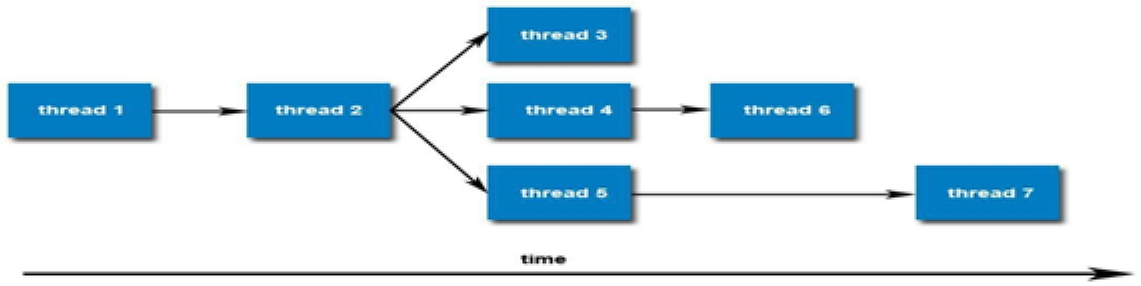
Lab 08 – Thread Creation and Management

Objective(s):

- Learn about Threads
- Learn about Creating Threads
- Learn about Multithreading in C++

What is a Thread?

A thread is the smallest unit of processing that can be performed in an OS. It is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. It is generally referred as a light weight process. In most modern operating systems, a thread exists within a process. A single process may contain multiple threads. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. Each thread executes different parts of a program and shares memory, file descriptors and other system resources. Once created, threads are peers, and may create other threads. There is no implied hierarchy or dependency between threads.



How Threads differ from Processes?

Threads differ from traditional multitasking operating system processes in that:

- Processes are typically independent, while threads exist as subsets of a process.
- Processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources.
- Processes have separate address spaces, whereas threads share their address space.
- Processes interact only through system-provided inter-process communication mechanisms.
- Context switching between threads in the same process is typically faster than context switching between processes.

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

What is Multithreading

Multithreading means two or more threads running concurrently where each thread is handling a different task. When you login to your Facebook profile, on your news feed, you can see live videos, you can comment or hit a like button, everything simultaneously. This is the best example of multithreading. Multithreading environment allows you to run many activities simultaneously; where different threads are responsible for different activities.

Uses of Multithreading

There are various uses of Multithreading, some of them are:

- Better resource utilization.
- Simpler program design.
- More responsive programs.

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 threadObject(<CALLBACK>);
```

New Thread will start just after the creation of a 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.

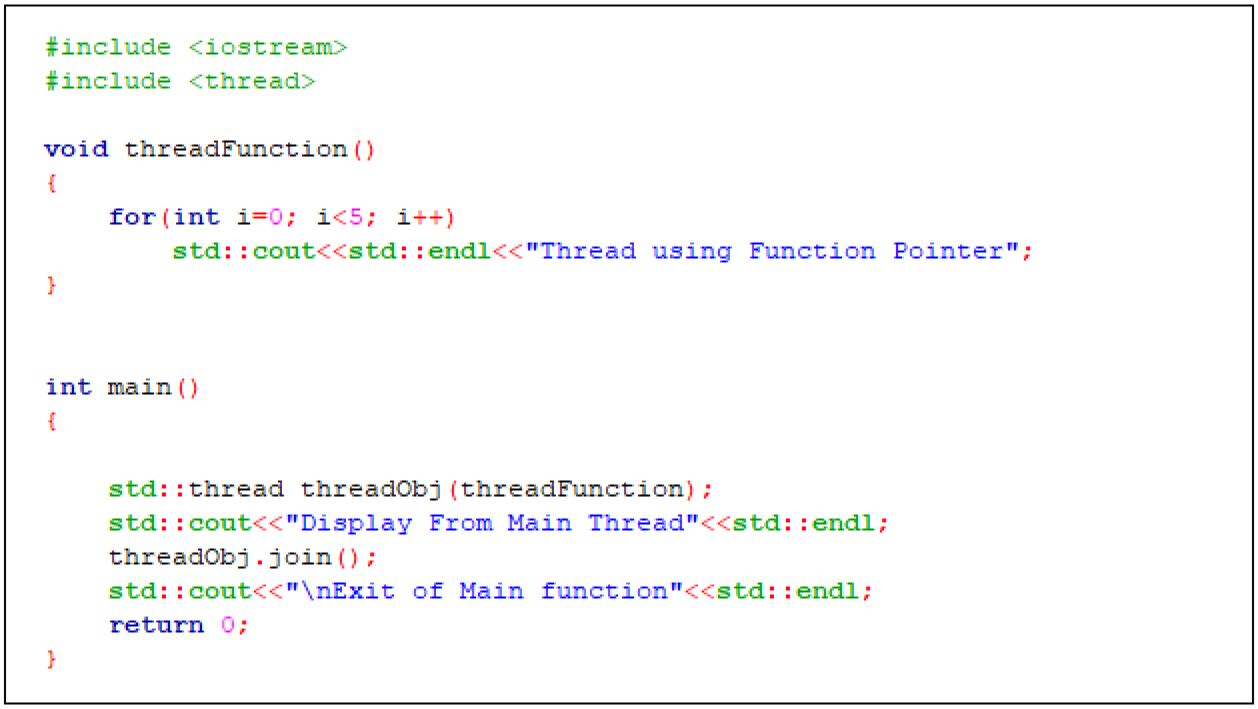
How to compile on Linux:

```
g++ –std=c++11 sample.cpp -o sample -lpthread
```

How to compile on Windows Operating System:

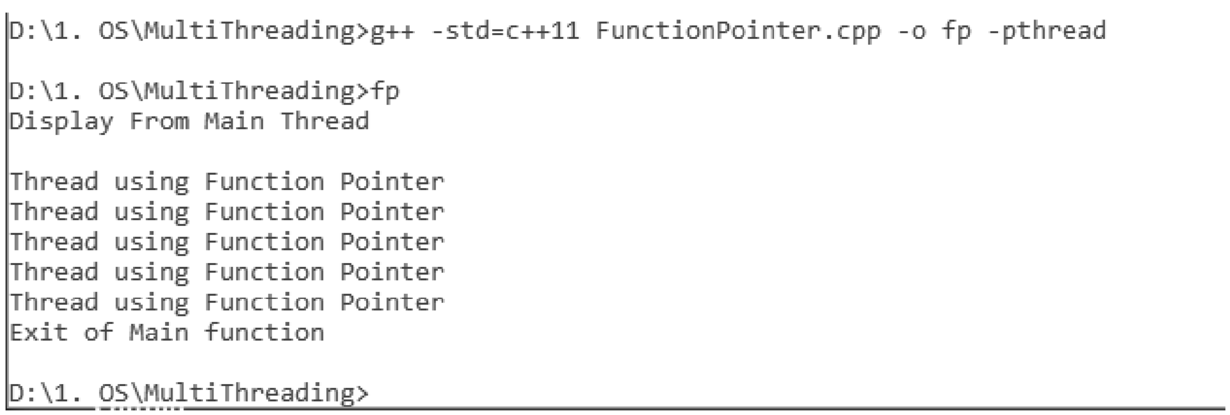
```
g++ –std=c++11 sample.cpp -o sample -pthread
```

Example 01 - Creating a thread using Function Pointer



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

void threadFunction()
{
    for(int i=0; i<5; i++)
        std::cout<<std::endl<<"Thread using Function Pointer";
}


int main()
{

    std::thread threadObj(threadFunction);
    std::cout<<"Display From Main Thread"<<std::endl;
    threadObj.join();
    std::cout<<"\nExit of Main function"<<std::endl;
    return 0;
}
```

Output:

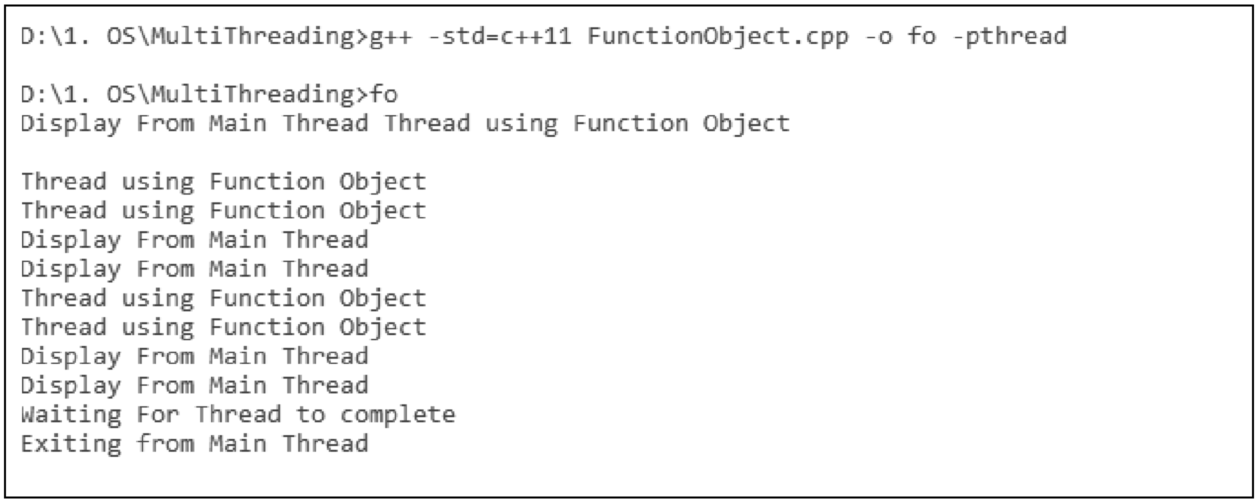


```
D:\1. OS\MultiThreading>g++ -std=c++11 FunctionPointer.cpp -o fp -pthread

D:\1. OS\MultiThreading>fp
Display From Main Thread

Thread using Function Pointer
Thread using Function Pointer
Thread using Function Pointer
Thread using Function Pointer
Thread using Function Pointer
Exit of Main function

D:\1. OS\MultiThreading>
```

Example 02 - Creating a thread using Function Objects

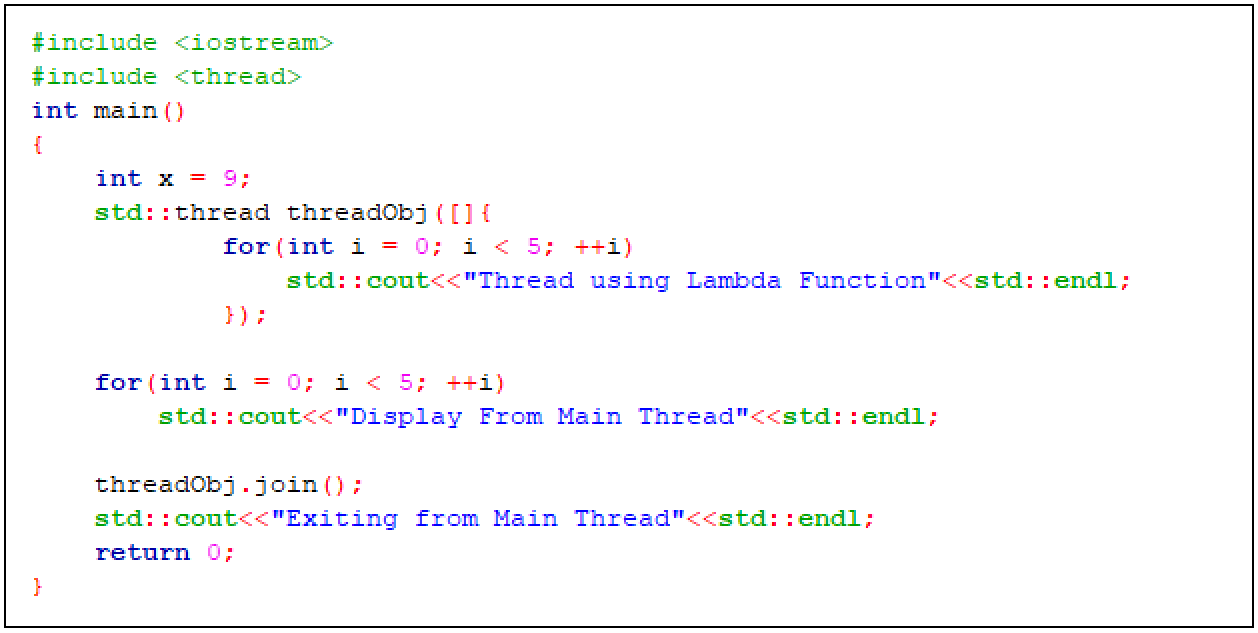
```
#include<iostream>
#include <thread>
class DisplayThread
{
public:
    void operator()()
    {
        for(int i = 0; i < 5; ++i)
            std::cout<<"Thread using Function Object"<<std::endl;
    }
};

int main()
{
    std::thread threadObj( (DisplayThread()) );
    for(int i = 0; i < 5; ++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;
}
```

Output:



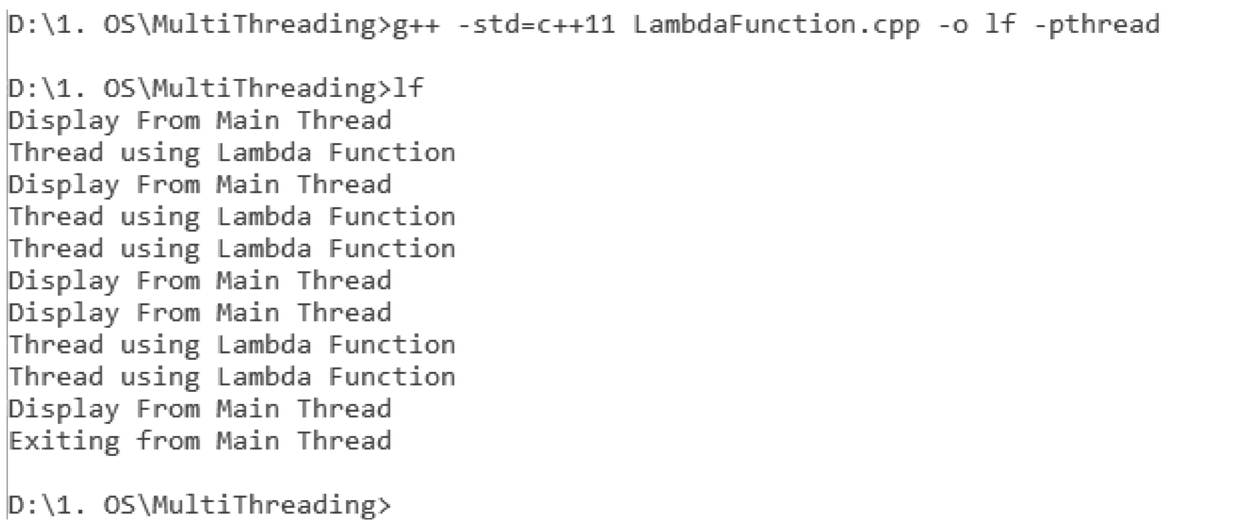
```
D:\1. OS\MultiThreading>g++ -std=c++11 FunctionObject.cpp -o fo -pthread

D:\1. OS\MultiThreading>fo
Display From Main Thread Thread using Function Object

Thread using Function Object
Thread using Function Object
Display From Main Thread
Display From Main Thread
Thread using Function Object
Thread using Function Object
Display From Main Thread
Display From Main Thread
Waiting For Thread to complete
Exiting from Main Thread
```

Example 03 - Creating a thread using Lambda functions



```
#include <iostream>
#include <thread>
int main()
{
    int x = 9;
    std::thread threadObj([]{
            for(int i = 0; i < 5; ++i)
                std::cout<<"Thread using Lambda Function"<<std::endl;
            });

    for(int i = 0; i < 5; ++i)
        std::cout<<"Display From Main Thread"<<std::endl;

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

Output:



```
D:\1. OS\MultiThreading>g++ -std=c++11 LambdaFunction.cpp -o lf -pthread

D:\1. OS\MultiThreading>lf
Display From Main Thread
Thread using Lambda Function
Display From Main Thread
Thread using Lambda Function
Thread using Lambda Function
Display From Main Thread
Display From Main Thread
Thread using Lambda Function
Thread using Lambda Function
Display From Main Thread
Exiting from Main Thread

D:\1. OS\MultiThreading>
```

Passing arguments to a std::thread:

To Pass arguments to thread's associated callable object or function just pass additional arguments to the std::thread constructor.

Example 04 - Passing Arguments to a thread

```
#include <iostream>
#include <string>
#include <thread>
 class DisplayThread
{
    public:
        void operator()(int x, std::string str)
        {
              std::cout<<"Passed Number = "<<x<<std::endl;
              std::cout<<"Passed String = "<<str<<std::endl;
        }
};
```

```cpp
void threadFunction(int x, std::string str)
{
    std::cout<<"Passed Number = "<<x<<std::endl;
    std::cout<<"Passed String = "<<str<<std::endl;
}
int main()
{
    int x = 10;
    std::string str = "Sample String";
    std::thread threadObj1(threadFunction, x, str);
    std::thread threadObj2((DisplayThread()), x, str);
    std::thread threadObj3([](int x, std::string str){
            std::cout<<"Passed Number = "<<x<<std::endl;
            std::cout<<"Passed String = "<<str<<std::endl;
    },x,str);

    threadObj1.join();
    threadObj2.join();
    threadObj3.join();
    return 0;
}
```

<u>Output:</u>

```
D:\1. OS\MultiThreading>g++ -std=c++11 PassingArgument.cpp -o pa -pthread

D:\1. OS\MultiThreading>pa
Passed Number = 10Passed Number =
Passed Number = 1010

Passed String = Passed String = Sample StringPassed String =
Sample StringSample String


D:\1. OS\MultiThreading>
```

Differentiating between threads

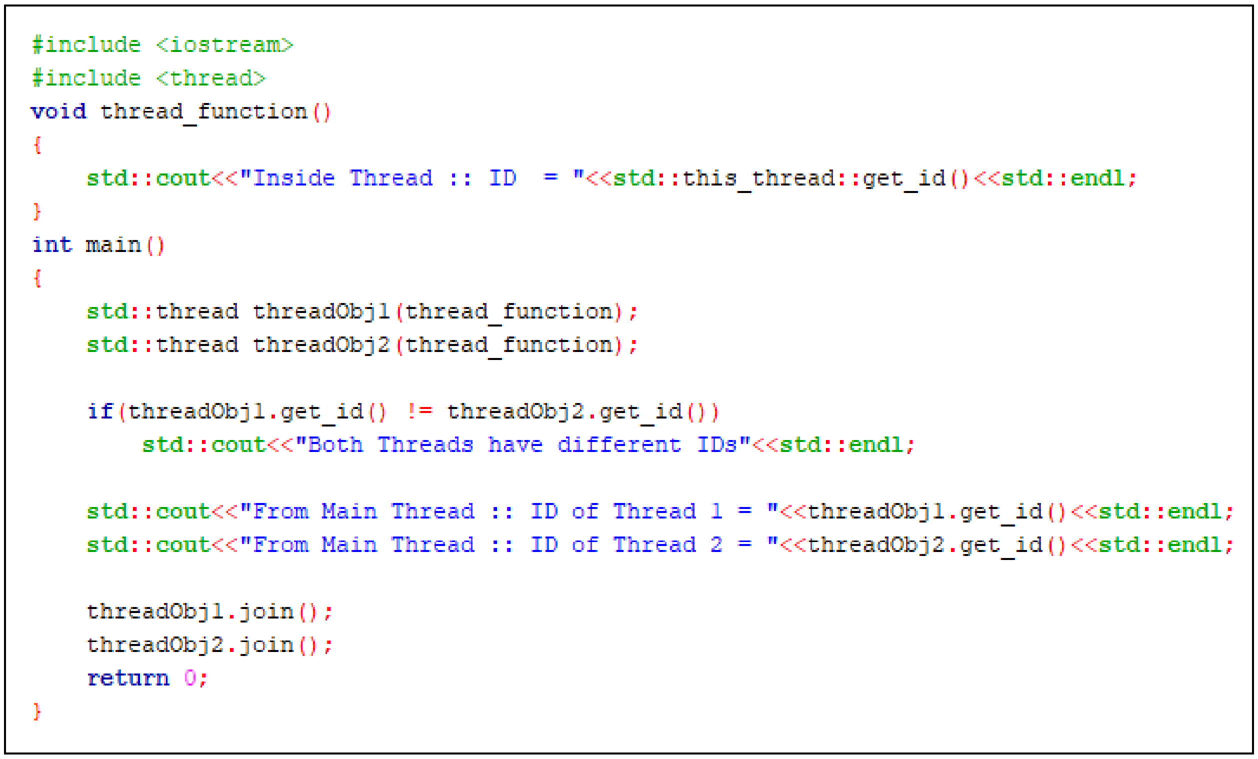
Each of the std::thread object has an associated ID and we can fetch using Member function, it gives id of associated thread object i.e.

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

To get the identifier for the current thread use,

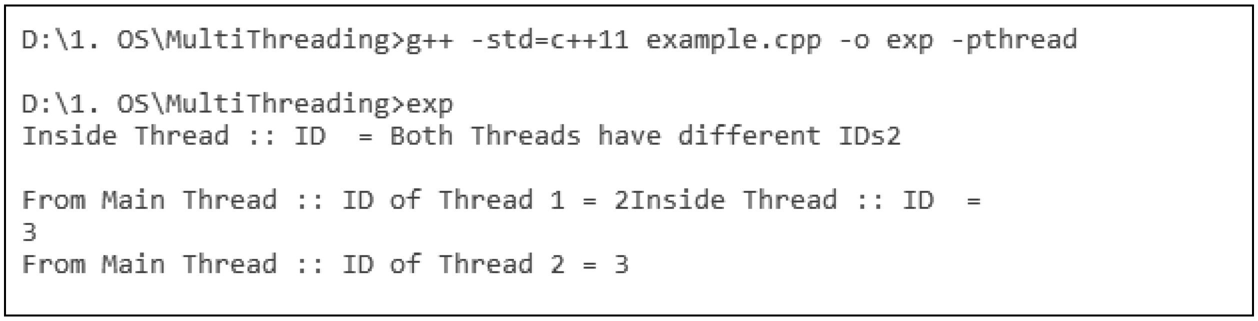
```
std::this_thread::get_id()
```

Example 05 - Using get_id()



```cpp
#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;
}
```

Output:



```
D:\1. OS\MultiThreading>g++ -std=c++11 example.cpp -o exp -pthread

D:\1. OS\MultiThreading>exp
Inside Thread :: ID  = Both Threads have different IDs2

From Main Thread :: ID of Thread 1 = 2Inside Thread :: ID  =
3
From Main Thread :: ID of Thread 2 = 3
```

Joining and Detaching Threads

There are two methods which we can use to join or detach threads.

join() function

Joining of a thread is done by using the join() function of the thread class. It makes main thread and child thread interdependent. Main thread terminates only after child thread terminates i.e. main thread waits for child thread to complete execution.

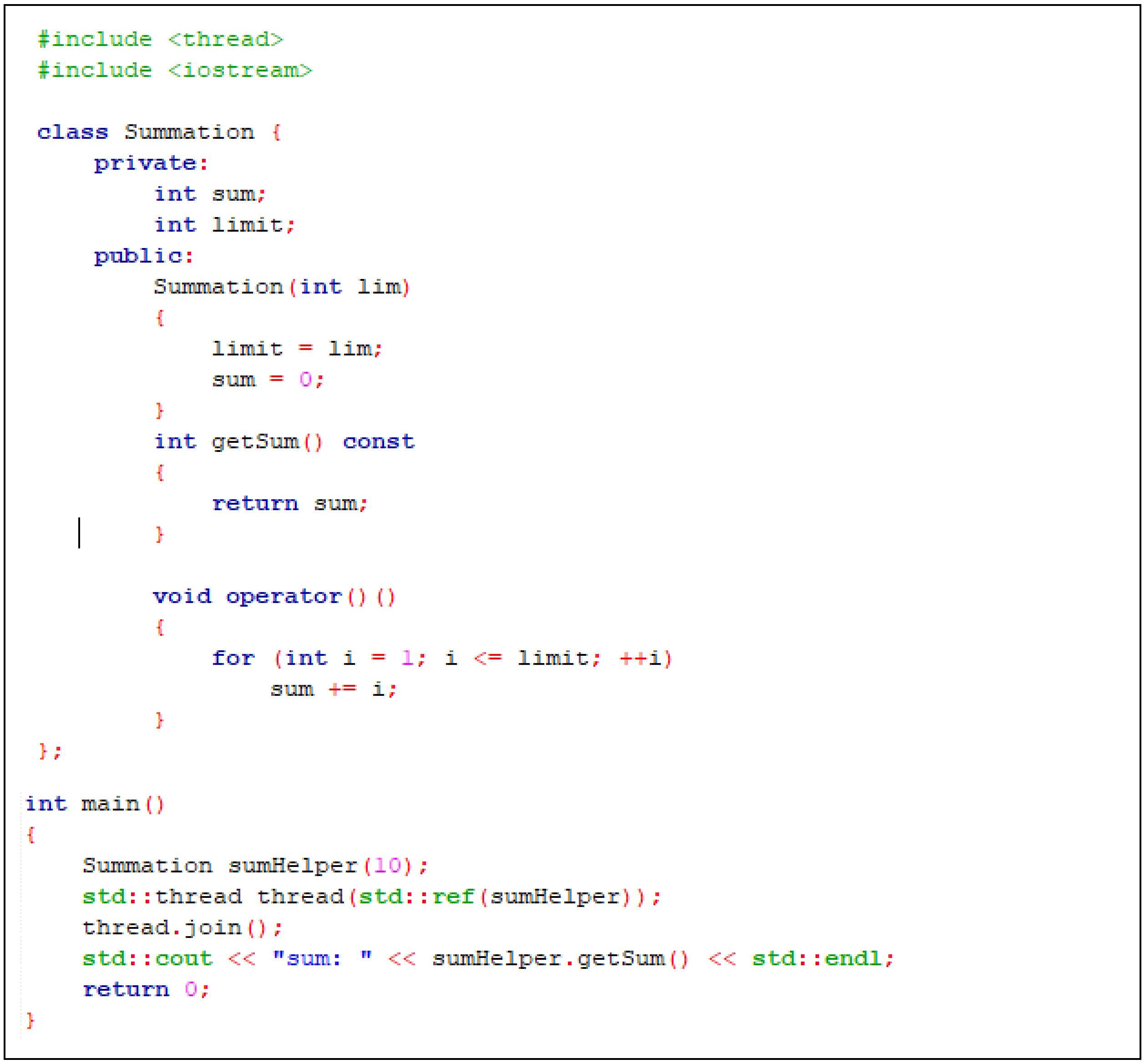
Syntax

```
threadname.join();
```

It returns once all functions are completed.

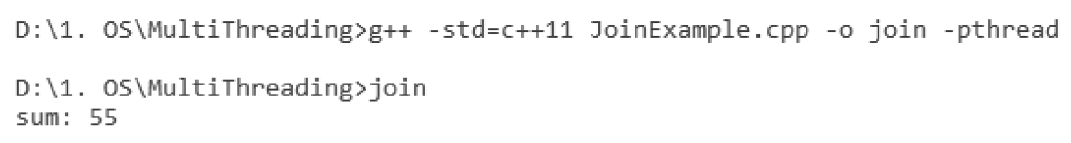
DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

<u>Example 06 - Joining Thread</u>



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

class Summation {
    private:
        int sum;
        int limit;
    public:
        Summation(int lim)
        {
            limit = lim;
            sum = 0;
        }
        int getSum() const
        {
            return sum;
        }

        void operator()()
        {
            for (int i = 1; i <= limit; ++i)
                sum += i;
        }
};

int main()
{
    Summation sumHelper(10);
    std::thread thread(std::ref(sumHelper));
    thread.join();
    std::cout << "sum: " << sumHelper.getSum() << std::endl;
    return 0;
}
```

Output:



```
D:\1. OS\MultiThreading>g++ -std=c++11 JoinExample.cpp -o join -pthread

D:\1. OS\MultiThreading>join
sum: 55
```

detach() function

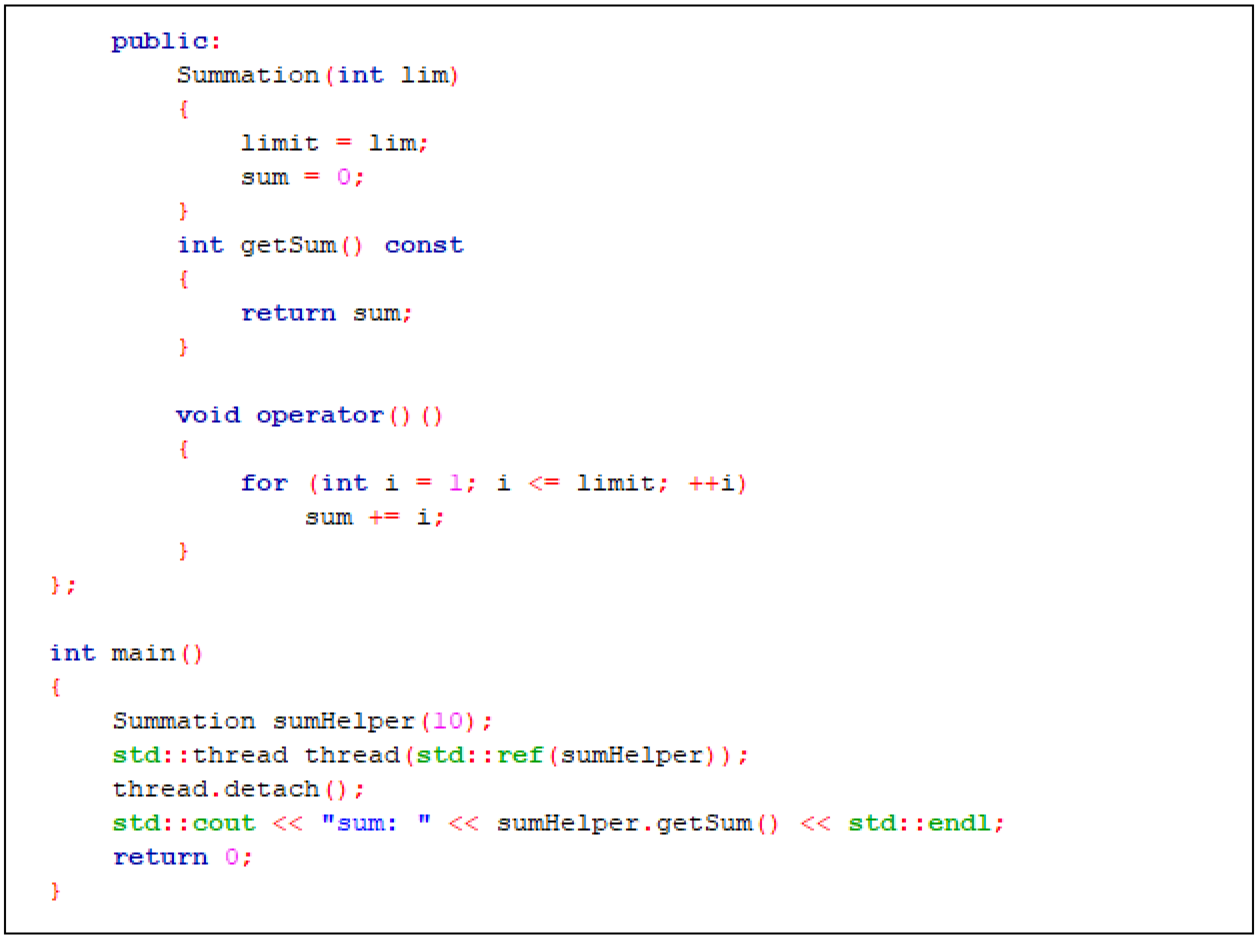
The detach() function detaches a thread from the parent thread. It allows both main thread and child thread to execute independently. Calling **detach()** on a **std::thread** object leaves the thread to run in the **background**. The actual thread of execution has no direct communication with the detached thread, thus there is no control over waiting for that thread to complete. In other words, if a thread becomes detached, it's impossible to obtain that **std::thread** object, so it can no longer be joined. A detached thread runs in the background. The ownership and control are passed over to the C++ Runtime Library. The C++ Runtime Library ensures that the resources of the thread are correctly reclaimed when the thread exits.

Syntax:

threadname.detach();

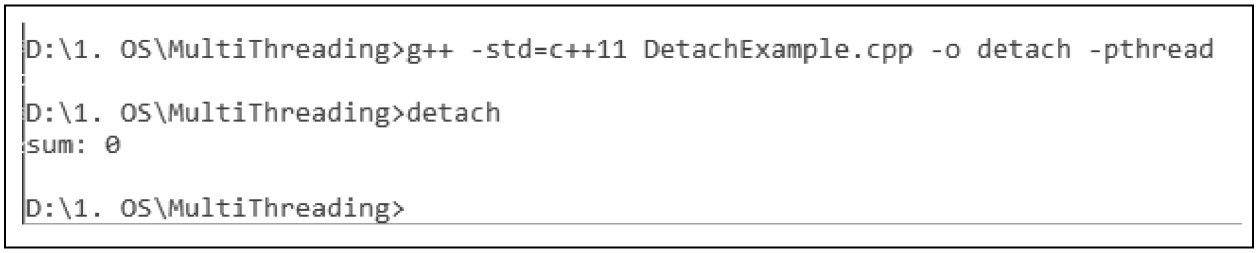
Example 07 - Detaching Thread

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

class Summation {
    private:
        int sum;
        int limit;
```



```cpp
    public:
        Summation(int lim)
        {
            limit = lim;
            sum = 0;
        }
        int getSum() const
        {
            return sum;
        }

        void operator()()
        {
            for (int i = 1; i <= limit; ++i)
                sum += i;
        }
};

int main()
{
    Summation sumHelper(10);
    std::thread thread(std::ref(sumHelper));
    thread.detach();
    std::cout << "sum: " << sumHelper.getSum() << std::endl;
    return 0;
}
```

Output:



```
D:\1. OS\MultiThreading>g++ -std=c++11 DetachExample.cpp -o detach -pthread

D:\1. OS\MultiThreading>detach
sum: 0

D:\1. OS\MultiThreading>
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Lab Tasks:

1. **Submit all the examples given in the lab manual.**

ASSIGNMENT # 08

1. Write a program that generates arithmetic series using std::threads.
2. Write a program that creates two threads, first thread is for generating the product and the second is for generating the sum of numbers from 1 to the number the user has entered.

SUBMISSION GUIDELINES

- Take a screenshot of each task(code and output).
- Place all the screenshots in a single word file labeled with Roll No and Lab No. e.g. **'cs181xxx_Lab01'.**
- Convert the file into PDF.
- Submit the file at LMS