**Processes Vs. Threads**

**Processes**  
Most platforms today allow several processes to execute at the same time. For example, a person might be browsing Facebook, watching a video and listening to music all at the same time.   
**Threads**  
When programming, it is sometimes optimal to allow for several tasks to be performed at the same time. For example, in an email client, you want the user to be able to compose a new email while checking for new messages in the background or that a user should still be able to navigate in a spreadsheet, while the values it contains are being recalculated. One way to do this is to create a separate process for each task but this would use up a lot of memory. In order to simplify this task, programmers use multithreading which means that each thread represents a single task within a process. It is up for the programmer to design the application so that all of the threads cooperate to accomplish the program’s activities.

**Basic Threading Code**

void function\_1()

{

cout << "Hello World" << endl;

}

int main()

{

thread t1(function\_1); //at this point t1 starts running

t1.join(); //the main thread waits for t1 to finish

system("pause");

return 0;

}

But what if t1 is too long and main doesn’t want to wait for t1 to finish because it has something to do?

void function\_1()

{

cout << "Hello World" << endl;

}

int main()

{

thread t1(function\_1); //at this point t1 starts running

t1.detach(); //t1 will run freely on its own which means it will run in the background and once detached, always detached

system("pause");

return 0;

}

**Threading Terms**

**Hardware Concurrency**

A method that gives you an indication of how many threads can be running at the same time

thread::hardware\_concurrency();

**Race Condition**

A race condition is a condition where the outcome of a program depends on the relative execution order of one or more threads. This is not good for the program so try to avoid it.

Example:

void function\_1()

{

for (int i = 0; i >-100; i--)

{

cout << "From t1: " << i << endl;

}

}

int main()

{

thread t1(function\_1);

for (int i = 0; i < 100; i++)

{

cout << "From main: " << i << endl;

}

system("pause");

return 0;

}

In this example, t1 and main is sharing the same output of cout. Therefore, the output of the loop will have a mixture of “From t1” and “From main”.

**Mutex**

A mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously. You can solve the race condition by using mutex. With this example, two threads are never using cout at the same time.

#include <string>

#include <iostream>

#include <thread>

#include <mutex>

using namespace std;

mutex mu;

void shared\_print(string msg, int id)

{

lock\_guard<std::mutex> guard(mu); //this syntax is better than mu.lock and mu.unlock because it handles exceptions

//mu.lock(); //while the msg is being printed, other threads cannot print it

cout << msg << id << endl;

//mu.unlock(); //other threads can lock the mutex and print their own msg

}

void function\_1()

{

for (int i = 0; i >-100; i--)

{

shared\_print(string("From t1: " ), i);

}

}

int main()

{

//this will give an indication of how many threads can be operating at the same time

//cout << thread::hardware\_concurrency();

//t1 starts running

thread t1(function\_1);

for (int i = 0; i < 100; i++)

{

shared\_print(string("From main: "), i);

}

system("pause");

return 0;

}

The output of this is shared between the first thread and the second thread. Therefore in the output console, you will first see t1 thread then main thread then back to t1 thread then main thread etc.

**Example Snippet Code**

#include <string>

#include <iostream>

#include <thread>

using namespace std;

void function\_1()

{

cout << "Goodbye" << endl;

}

int main()

{

thread t1(function\_1); //at this point t1 starts running

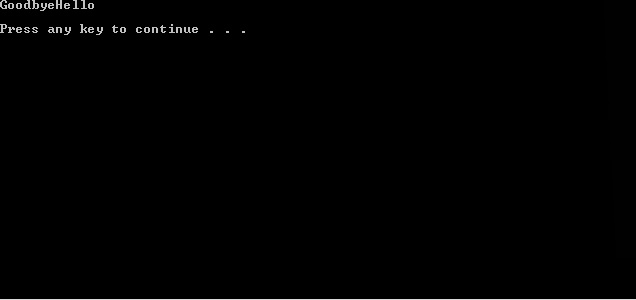
cout << "Hello" << endl;

system("pause");

return 0;

}

In this example, both threads are sharing the same memory so they will be printed out at the same time.



#include <string>

#include <iostream>

#include <thread>

using namespace std;

void function\_1()

{

cout << "Goodbye" << endl;

}

int main()

{

thread t1(function\_1); //at this point t1 starts running

t1.join();

cout << "Hello" << endl;

system("pause");

return 0;

}

But in this example, because we have t1.join(), the main thread must wait on t1 to finish then run its own thread. Therefore, you get the output of…

