Thread creation is easy. Call  std::thread, and a new thread will be created. The thread gets a work package and starts it immediately. The creator of the thread (the Parent) has to take care of the created thread (the child). The Parent should wait until its child is done with its task or has to detach itself from its child. The child thread can get its payload task arguments by copy or by reference.

Creation and execution of a thread

Now, more formal approach: a thread gets a Callable and starts it immediately.

This sentence needs a few notes.

* A Callable  is an entity which behaves like a function. It can be a function, a function object or a lambda function.
* A[function object](http://en.cppreference.com/w/cpp/utility/functional) is an instance of a class, for which the call operator is overloaded. The key difference between functions and function objects is, that a function object can have state.
* A [lambda function](http://en.cppreference.com/w/cpp/language/lambda) (anonymous function) is a pure function body, without a name. It can be invoked just in place. A lambda function can capture its calling context. That's  why they are often called closures.
* // createThread.cpp
* #include <iostream>
* #include <thread>
* void helloFunction(){
* std::cout << "Hello C++11 from function." << std::endl;
* }
* class HelloFunctionObject {
* public:
* void operator()() const {
* std::cout << "Hello C++11 from a function object." << std::endl;
* }
* };
* int main(){
* std::cout << std::endl;
* // thread executing helloFunction
* std::thread t1(helloFunction);
* // thread executing helloFunctionObject
* HelloFunctionObject helloFunctionObject;
* std::thread t2(helloFunctionObject);
* // thread executing lambda function
* std::thread t3([]{std::cout << "Hello C++11 from lambda function." << std::endl;});
* // ensure that t1, t2 and t3 have finished before main terminates
* t1.join();
* t2.join();
* t3.join();
* std::cout << std::endl;
* };