Understanding Callbacks and Signals in C++

With the Alarm System Example

Mohamed KHABOU

February 1, 2025

Objective

- Understand how to implement a callback system in C++.
- Learn how to connect signals to actions using lambdas.
- Explore a practical example: An Alarm System.

What is a Callback?

A **callback** is a function that is passed as an argument to another function and is executed after some event or action.

- Callbacks are used to handle asynchronous events.
- In C++, callbacks can be implemented using `std::function` and lambdas.
- They are commonly used in event-driven programming.

The Alarm System Example

We will create an **Alarm System** where:

- An `Alarm` class has a signal (`onTrigger`) that is triggered when the alarm goes off.
- A `SecuritySystem` class handles the alarm by displaying a notification.

- `Alarm`: Represents the alarm with a `trigger` method.
- `SecuritySystem`: Handles the alarm event.
- `connect`: A function to connect the signal to the callback.

Code: Alarm Class

The `Alarm` class has a signal (`onTrigger`) and a method to trigger the alarm.

```cpp
class Alarm {
public:
    std::function<void()> onTrigger;

    void trigger() {
        std::cout << "The alarm was triggered!" << std::endl;
        if (onTrigger) {
            onTrigger(); // Call the callback
        }
    }
};
```

Code: SecuritySystem Class

The `SecuritySystem` class handles the alarm by displaying a notification.

```
class SecuritySystem {
public:
    static void sendAlert(const std::string& message) {
        std::cout << "Security Alert: " << message << std::endl;
    }
};
```

Code: Connecting the Signal

We use a connect function to link the onTrigger signal to a lambda function.

```
1 template <typename Sender, typename Signal, typename Functor
      >
2 void connect(Sender* sender, Signal signal, Functor functor)
       {
3     sender->*signal = functor;
4 }
```

Code: Main Function

The `main` function creates an alarm, connects it to the security system, and triggers the alarm.

```
int main() {
    Alarm alarm;
    connect(&alarm, &Alarm::onTrigger, []() {
        SecuritySystem::sendAlert("Intrusion detected!");
    });

    alarm.trigger(); // Simulate the alarm being triggered
    return 0;
}
```

The connect Function

The connect function links a **signal** (e.g., onTrigger) to a **callback** (e.g., a lambda function).

```
1 template <typename Sender, typename Signal, typename Functor
      >
2 void connect(Sender* sender, Signal signal, Functor functor)
       {
3     sender->*signal = functor;
4 }
```

- sender: A pointer to the object emitting the signal (e.g., &alarm).
- signal: A pointer to the signal member (e.g., &Alarm::onTrigger).
- functor: The callback function (e.g., a lambda function).

How `sender->*signal = functor` Works

The expression `sender->*signal` uses the **pointer-to-member operator** (`->*`) to access the signal member of the `sender` object.

```
1 sender->*signal = functor;
```

How sender->*signal = functor Works

- `signal` is a pointer to the `std::function<void()>` member of the `sender` object.
- `functor` is a lambda function or any callable object.
- The assignment `sender->*signal = functor` sets the `std::function` member to the provided `functor`.

```
connect(&alarm, &Alarm::onTrigger, []() {
    SecuritySystem::sendAlert("Intrusion detected!");
});
```

Program Output

When the program runs, the output demonstrates how the alarm triggers the security system.

```
1 The alarm was triggered!
2 Security Alert: Intrusion detected!
```

Key Takeaways

- Callbacks are powerful tools for handling events in C++.
- Use `std::function` and lambdas to implement callbacks.
- The Alarm System example demonstrates how to connect signals to actions.