게임 서버 프로그래밍 - 라이브러리 연습

지산

February 4, 2024

Contents

Part I

준비하기

Chapter 1

왜 C++ 인가?

1.1 현재의 언어들을 간략하게 살피기

llvm의 출현 이후 컴파일러의 마지막 단계인 코드 생성이 매우 수월해지면서 아주 다양한 언어들이 많이 개발되었습니다. 그리고 대체로 개발자 효율을 더 중요시 하고 있어서 go 언어, C#, java, kotlin 등 메모리 관리를 garbage collection으로 자동화 한 언어들이 많이 쓰이고 있습니다.

C++은 C만큼 빠르게 실행되는 코드를 만들 수 있고, C와 호환되면서 클래스와 템플릿을 포함한 더 강력한 언어를 목표로 만들어졌습니다. 하지만, 언어가 오래 되면서 미묘하게 복잡한 부분들이 늘어나고, 템플릿 프로그래밍이 강력하기는 하지만 사람이 이해하기가 쉽지 않은 면이 있어 근래 시스템 프로그래밍을 하는 곳이 아니면 잘 배우지 않고 잘 쓰이지 않는 경향이 있습니다.

프로그래밍의 근간은 모두 동일합니다. 메모리를 직접 바꿀 수 있는 명령형(Imperative) 언어들은 가장 강력한 기능을 메모리를 바꾸는 사이드 이펙트를 통해 처리합니다. 그런 면에서 튜링 기계의 원시 모델에서 별반 앞으로 나아가지 못 했다고 할 수 있습니다.

Lisp, Ocaml, Haskell, F# 과 같은 함수형 언어들은 사이드 이펙트를 지양하고 순수한 함수들의 조합으로 처리하려고 합니다. 같은 입력에 대해 같은 결과를 돌려주는 순수 함수들로 프로그램을 구성하면 증명 가능하고 버그가 적은 결과물을 만들 수 있기 때문입니다.

이런 함수형 언어의 영향을 명령형 언어들도 많이 받고 있는데 C++의 템플릿 프로그래밍이 순수 함수형 언어의 하나이고, 람다와 같은 함수형 언어의 기능들이 많이 도입되어 있습니다.

그래서, C++이 더 복잡해진 원인이기도 합니다.

1.2 게임 개발에서 사용하는 언어들

게임 개발에서는 클라이언트와 서버를 모두 포함하여 C++을 가장 많이 사용해 왔습니다. 지금도 C++로 개발한 게임들이 많고, 언리얼은 C++로 개발한 클라이언트 엔진입니다.

제가 알기로 거의 대부분의 엔씨소프트 게임들은 C++ 서버이고, 넥슨의 많은 서버들도 C++서버입니다. 웹젠의 게임들도 거의 모두 C++이고 일부 게임들이 C#으로 개발되었습니다.

마비노기를 만든 사람들은 C#을 극한까지 활용하여 멀티플레이도 C# 서버로 제공하고 있다고 알고 있습니다. 크레이지 아케이드 B&B도 C++ 서버이니 대부분의 게임들이 C++이라고 할 수 있습니다.

약간 상황이 바뀐 계기는 초기 모바일 게임들이 멀티플레이를 제공하지 않고도 성공한 경우들이 생기고, C++이 다루기 어렵고 버그가 생기면 치명적인 경우들이 많아 멀티플레이를 제공하지 않는 모바일 게임들은 node.js 기반으로 javascript를 쓰는 경우도 생겼습니다.

이런 게임들 중 대표적인 경우가 쿠키런 같은 게임들인데 개발사인 데브시스터즈가 현재 경영 위기를 겪고 있는 이유 중 하나가 너무 말랑한 게임들 위주로 포트폴리오를 구성한 것 때문일 수 있고, 근간에는 멀티플레이를 잘 제공할 수 있는 기술적인 기반의 준비가 약했던 면도 영향을 미쳤다고 생각합니다.

go로 멀티플레이 게임들 서버를 만들려는 시도들도 있었습니다. go는 가비지컬렉션을 갖는 언어이긴 하지만 컴파일된 코드가 매우 빠르게 동작하는 것으로 알려져 있고, 암호화폐 (이더리움) 에 사용해서 많은 암호화폐 프로젝틀들이 사용하고 있고, 쿠버네티스를 포함한 다양한 도구 개발에도 사용하고 있는 좋은 언어입니다.

근래는 러스트로 서버를 만들려는 노력들이 있고 매우 빠른 웹 서버까지는 개발하여 여러 곳에서 서비스 하고 있지만, 아직 게임 서버 개발에까지 도입한 경우는 많지 않아 보입니다.

1.3 C++의 중요한 변화

C++은 현재 언어의 위기 상황을 어느 정도 넘어선 상태로 보입니다. 가장 큰 위기는 수동으로 메모리 할당과 해제를 하면서 메모리 관련 오류가 발생하기 쉽고, 메모리 오류는 서버가 크래시 (프로세스가 멈춤) 되기 때문에 서비스에 치명적이라는 점에서 기인했습

니다.

C++ 발전에 중요한 기여를 하고 있는 boost 프로젝트에서는 shared_ptr, unique_ptr 이 오래전에 도입되었으나 언어에 채택되는데까지 너무 오랜 시간이 걸렸습니다.

템플릿 프로그래밍의 복잡도 문제까지 포함하여 위기가 심화되고 있었는데 이들은 C++ 11 표준에서 상당한 수준으로 해결했습니다. 이 때 많은 반성이 있었고, 지금은 3년마다 표준화를 진행하여 언어를 빠르게 발전시키고 있습니다.

여전히 언어가 복잡하고 사용이 어려운 면이 있다는 점은 분명하지만 개선 속도가 빨라져서 다른 언어들과 다시 경쟁할 수 있게 되었다고 봅니다.

1.4 왜 여전히 C++인가?

1.4.1 빠른 코드 생성과 세밀한 제어가 가능하다

C++은 시스템 프로그래밍 언어로 세밀한 제어가 가능하고 빠른 코드를 생성합니다. 그래서, 날카로운 칼이다라고 할 수 있습니다. 잘 쓰면 유용하지만 잘못 쓰면 매우 위험합니다. 잘 쓰는 연습을 많이 해야 합니다. 그럼에도 불구하고 OS (Operating System, 운영체제)를 만들 수 있는 언어로 생성된 코드가 매우 빠릅니다. 리눅스는 C 언어로 개발되었지만 윈도우는 C++로 개발되었습니다. clang과 같은 컴파일러, SQL 서버와 같은 데이터베이스 등이 C++로 개발된 이유는 세세하게 제어할 수 있고 빠르기 때문입니다.

게임 클라이언트와 서버는 모두 빠른 처리를 필요로 합니다. 그래서, 초기부터 현재까지 많이 쓰이고 있고 앞으로도 많이 쓰일 것입니다.

1.4.2 다양한 안정적인 라이브러리가 있다

게임 개발을 위해서는 다양한 라이브러리가 필요합니다. 2장에서 정리한 라이브러리 외에도 많은 라이브러리들이 있고 C++로 개발되어 오래 쓰이면서 안정된 것들이 상당히 많습니다.

유니티도 내부 엔진은 C++로 개발되었고 IL2CPP처럼 C++로 다시 변환한 후 컴파일해서 사용할 만큼 C++은 빠른 코드를 만들 수 있습니다. 언리얼이 C++로 개발된 이유이기도 합니다. 두 엔진 모두 다양한 외부 라이브러리를 사용하고 있습니다.

1.5 C++을 도구로 포함하기

C#이나 javascript, python으로 게임 서버에 필요한 기능을 빠르게 구현해보면서 다양한 실험을 할 수 있습니다. 모델링 언어로서 매우 좋습니다. 그리고, 아주 빠른 성능을 요구하지 않는 경우 서비스 할 수 있는 게임 서버를 제작하는데도 사용할 수 있습니다.

그래서, 하나의 모델링 언어이자 가벼운 게임을 만드는 도구로 쉽고 빠르게 개발 가능한 언어를 하나 갖고 있으면 좋습니다. C#은 유니티의 영향이 크겠지만 다양한 라이브러리가 잘 포팅되거나 사용 가능한 형태로 제공이 되므로 좋은 언어라고 할 수 있습니다.

하지만 여전히 C#으로는 만들기가 어려운 게임들이 있습니다. C++과 실질적인 성능 차이가 몇 배라고 할 수 있을 정도로 아직은 많이 나기 때문입니다. 또 C++로 잘 작성하면 C#보다 더 잘 동작하는 경우도 많습니다. 템플릿이 어렵지만 잘 쓰면 매우 강력하기도 합니다.

그래서, C#과 C++ 또는 python과 C++ 같이 두 개의 짝을 이루는 언어를 나의 도구로 갖추고 있으면 성장을 지속하는데에 큰 도움이 됩니다.

1.6 C++ 연습하기

C++ 표준 문서는 수천 페이지에 달합니다. 이렇게 복잡한 언어를 프로그래머가 전부 다 이해하기는 불가능에 가깝습니다. 따라서, 효율적으로 이해하고 활용할 방법이 필요합니다.

회사에서 한번 진행한 방법으로 참고할만 합니다.

- Tour of C++ 읽으면서 코드 따라하기
- C++ 표준 라이브러리를 STL 중심으로 연습하기
- C++ 표준 라이브러리 코드 분석하기

Tour of C++은 C++ 개발자인 비야네 스트루스트룹이 여행 안내서처럼 쓴 책으로 대부분의 C++ 기능을 가볍게 안내하듯이 설명하고 있습니다. 씹어먹는 C++ 같은 좋은 사이트들을 참고하면서 제시된 코드들을 연습하면 좋습니다.

위 과정이 원활하지 않을 수도 있습니다. 특히, C++ 표준 라이브러리 코드 분석하기는 매우 어려울 수 있습니다. 최고의 C++ 프로그래머들이 오랜 시간 만들어낸 결과물이기 때문입니다. 아름다운 수학 증명이 매우 어려운 경우는 역사 속에서 매우 천천히 많은 노력들이 쌓인 결과물이기 때문인 것과 비슷합니다.

1.7 안내하고 정리하기

게임 서버 프로그래밍 책을 시리즈로 쓰려고 준비한 이유가 위의 내용들입니다.

많은 프로그래머들과 프로젝트들을 진행하면서 사실 우리가 잘 이해하지 못하거나 더 나은 방법이 있는데 잘 몰라서 어려움을 겪는 경우가 많기 때문입니다.

기본은 연습입니다. 목표를 정하고 필요한 것들을 찾아서 꾸준히 연습할 때만 완전해집니다. 인공지능의 시대에도 프로그래밍은 여전히 창의적인 일이고 함께 새로운 세계를 만드는 게임 개발은 지속될 것입니다.

이런 목표로 첫 걸음을 뗍니다.

Chapter 2

어떤 라이브러리가 필요할까요?

게임 서버를 만들 때 필요한 라이브러리는 다양하게 많습니다. 많은 사람들이 모여서 함께 살아가는 새로운 가상의 세계를 만들려면 통신, 데이터베이스가 필요합니다. 또 안정적으로 동작하는지 확인하려면 로그를 잘 남기고 서버 상태를 알 수 있어야 합니다.

그래서 필요한 도구들을 현재까지 사용해 온 경험을 살펴서 정리하면 다음과 같습니다.

- doctest 단위 테스트
- C++ 표준 라이브러리 (Container, Thread 등)
- fmt 또는 format
- spdlog 로깅
- asio 통신과 처리 (Proactor)
- nanodbc 데이터베이스 ODBC 라이브러리
- boost에서 필요한 라이브러리들

여기까지가 기본 중의 기본이 되는 라이브러리입니다. asio로 서버 처리와 TCP 통신을 작성할 수 있고 DB에 저장하고 읽을 수 있다면 기본은 된 상태라 할 수 있습니다.

이 상태에서는 가상의 물리적인 월드 구성이 없는 모든 게임을 만들 수 있습니다. 예를 들어, 멀티플레이어 테트리스, 마비노기 영웅전 (물리적인 세상이 있지만 클라이언트에서 처리), 카트 라이더 (물리적인 세상이 있지만 클라이언트에서 처리) 등이 있습니다.

그 위에 세상을 만들기위해 이동과 길찾기, 충돌 처리 라이브러리를 필요로 합니다. 주로 쓰이는 라이브러리는 다음과 같습니다.

- RecastDetour 이동과 길찾기
- bulletphysics 충돌과 물리

RPG (Role Playing Game) 장르의 게임들은 서버에서 대부분 위 라이브러리나 유사한 기능을 서버에서 구현해야 합니다. 게임마다 선택은 다르지만 위 두 가지를 기초로 합니다.

이동과 길찾기, 충돌과 물리 처리는 유니티나 언리얼과 같은 클라이언트 엔진에 잘 구현되어 있습니다. 게임 서버에서는 더 빠르게 처리해야 많은 사용자를 처리할 수 있으므로 여러 방법을 동원하여 가볍게 만들려는 노력을 합니다.

Part II

단위 테스트

Part III

C++ 표준 라이브러리

Chapter 3

준비

3.1 무엇을 연습하나요?

C++ 표준 라이브러리는 C++ 컴파일러들과 함께 배포되고 있고, 이들 라이브러리 개발자들은 최고 수준의 C++ 프로그래머들입니다. 따라서, 제공되는 인터페이스와 설명 문서로 잘 사용하는 연습을 통해 여러 곳에서 활용할 수 있을뿐만 아니라 C++ 언어에 대한 이해를 높이는데 매우 효과적입니다.

다양한 라이브러리 구성 요소들이 있습니다. 그 중에서 가장 많이 사용하는 컨테이너 자료구조와 알고리즘을 어떻게 쓰는지 배우는 것에서 시작합니다. 필요한 라이브러리 사용법들을 추가로 좀 더 살펴보고 연습한 후에 코드를 읽는 연습을 합니다.

좀 익숙해지면 빅오(Big O) 기호라고 불리는 알고리즘의 성능 특성을 실제 사용 코드와 프로파일링을 통해 확인합니다. 그 다음은 C++이 어떻게 얼마나 최적화를 잘 하는지 어셈블리 코드를 읽으면서 확인합니다.

- std::array
- std::vector
- std::map
- std::unordered_map
- std::set

3.2 어떤 환경에서 연습하나요?

Part IV

통신

Part V

데이터베이스