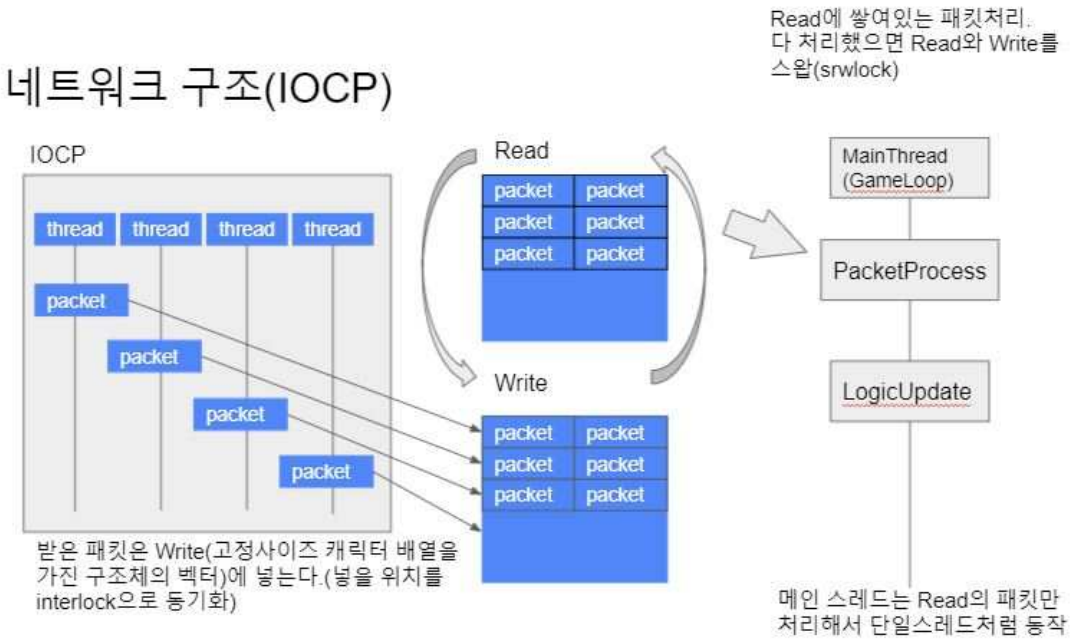
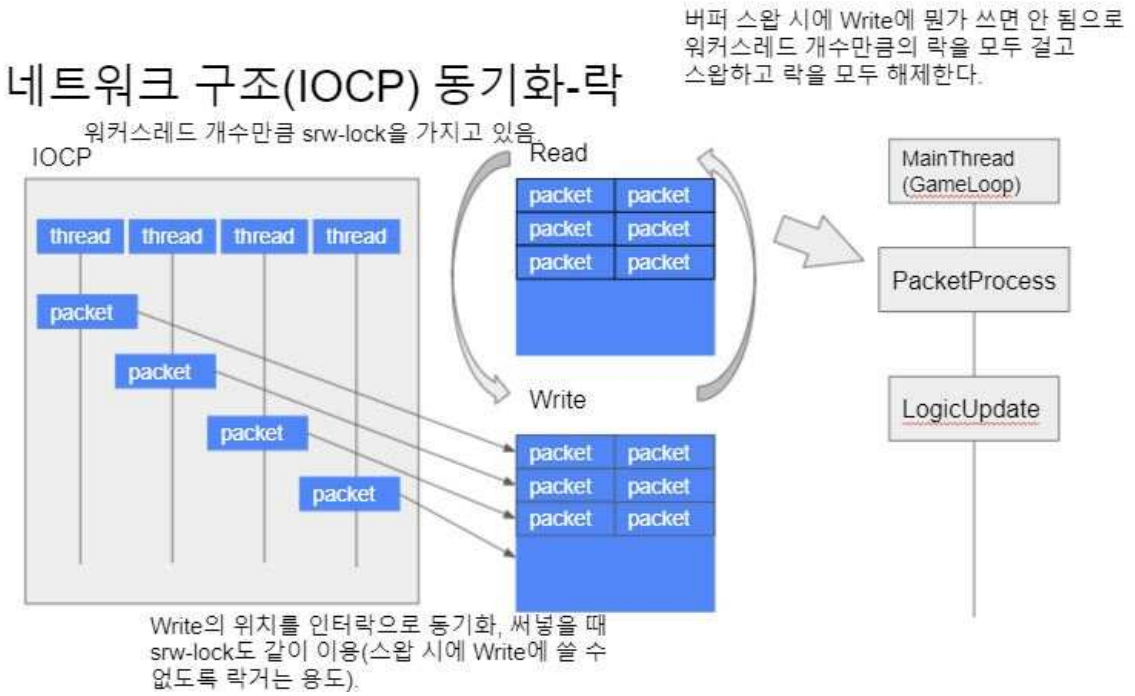
포트폴리오

네트워크 라이브러리(IOCP)

개발인원: 1
개발기간: 2020.12 ~ 2021.06
개발환경: VisualStudio 2019
링크 : https://github.com/YUKI-SOS/SRIOCP



IOCP를 이용한 네트워크 라이브러리 입니다. 주로 화면을 그릴 때 깜빡임을 방지하는 더블버퍼링의 개념을 응용하여 패킷을 Recv하는 것이 특징입니다. Read와 Write 두 개의 자료구조를 두고 워커스레드들이 GQCS에서 패킷을 받으면 Write쪽에 패킷을 넣고, 메인스레드 쪽에서는 Read에 있는 패킷들을 처리하고 Read에 있는 패킷을 다 처리했을 경우 두 자료구조 간에 서로 Swap하는 구조입니다. 해당 구조는 유영천님의 '실시간 게임서버 최적화 전략' 이라는 강의 자료를 보고 구현하였습니다. (https://www.slideshare.net/dgtman/ss-228096175)



스레드 간 동기화를 위해서 인터락Interlock과 SRW-Lock을 이용하였습니다. IOCP 워커스레드의 개수만큼의 SRW-Lock을 가지고 있으며, Read와 Write 스왑할 때에 Write에 패킷을 써넣을 수 없도록 하는 용도입니다. 각 워커스레드는 패킷을 받으면 패킷을 Write에 넣을 때 인터락을 이용하여 써넣을 위치를 동기화하고, 스왑 할 때 쓰지 못하도록 자신이 가진 SRW-Lock을 걸고 있습니다. Read와 Write의 자료구조는 인터락을 통해 써넣을 위치를 동기화하기 때문에 인덱스를 통해 접근 가능한 STL의 Vector를 사용하였습니다.

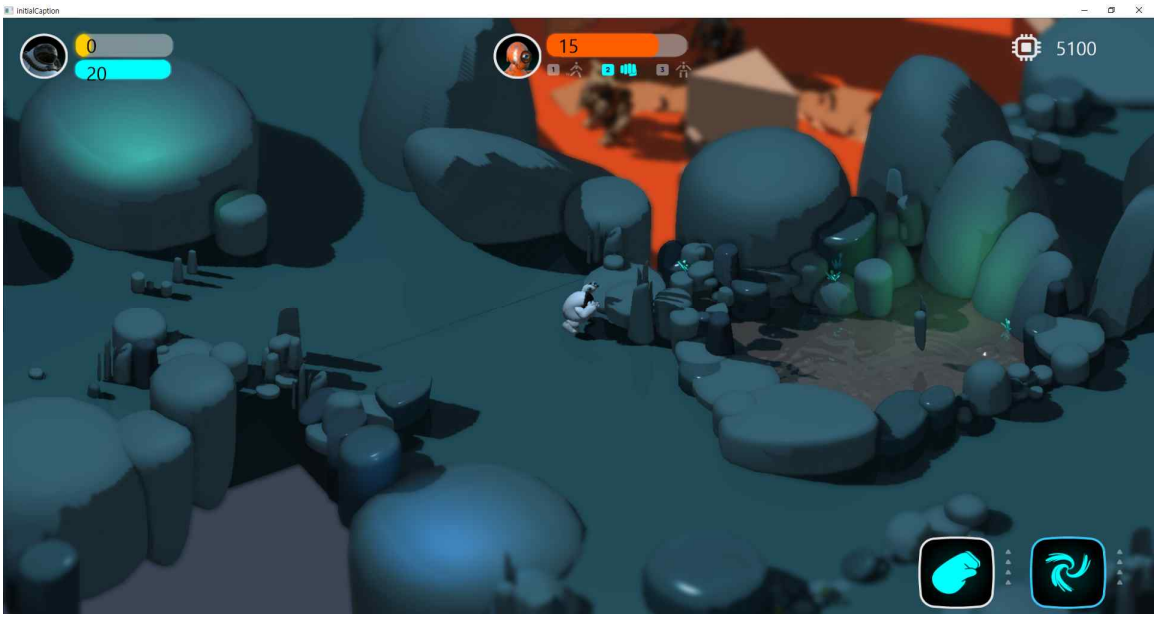
메인스레드에서 Read자료구조에 있는 모든 패킷을 처리한다면 워커스레드가 가진 락의 개수만큼 락을 걸고 스왑하고 다시 모든 락을 해제하는 과정을 통하여 스레드 간 동기화를 구현하고 있습니다. 이를 통하여 라이브러리를 이용하는 프로세스의 메인스레드는 동기화를 생각하지 않고 단일스레드처럼 이용할 수 있도록 설계하였습니다.

게임 Escort The Pi(룸 구조 기반 멀티플레이 서버)

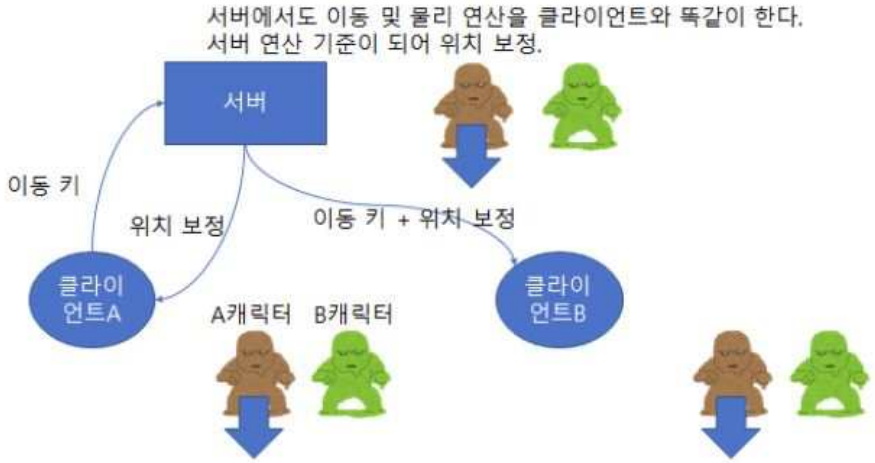
개발인원: 10 (프로그래밍 4 기획 4 아트 2)

개발기간: 2020.12 ~ 2021.06

개발환경: VisualStudio 2019

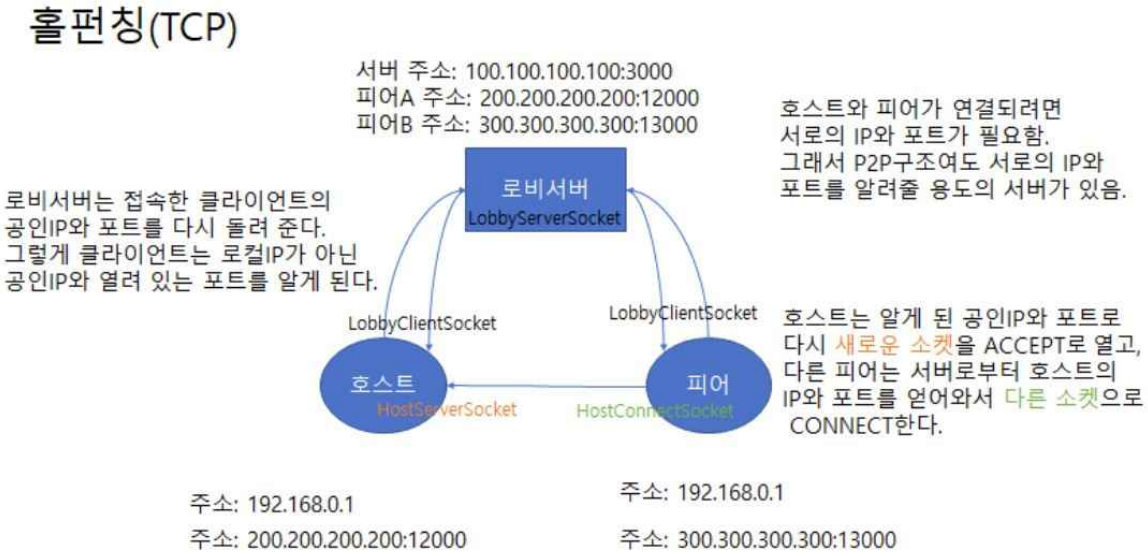


팀 Crio에서 게임 Escort The Pi의 서버를 담당하였습니다. Escort The Pi는 호위대상을 지키면서 목적지까지 도달해야 하는 호위 액션 게임입니다. 로비에서 룸을 골라 접속하여 하나의 룸에서 2명이 협력하여 플레이 하는 형태의 룸 구조 형태의 멀티플레이를 요청받고 개발을 진행하였습니다. 서버는 직접 만든 IOCP 네트워크 라이브러리를 이용하여 구현하였으며, 클라이언트 또한 서버와 같은 라이브러리를 이용하여 패킷을 송수신 하고 있습니다.

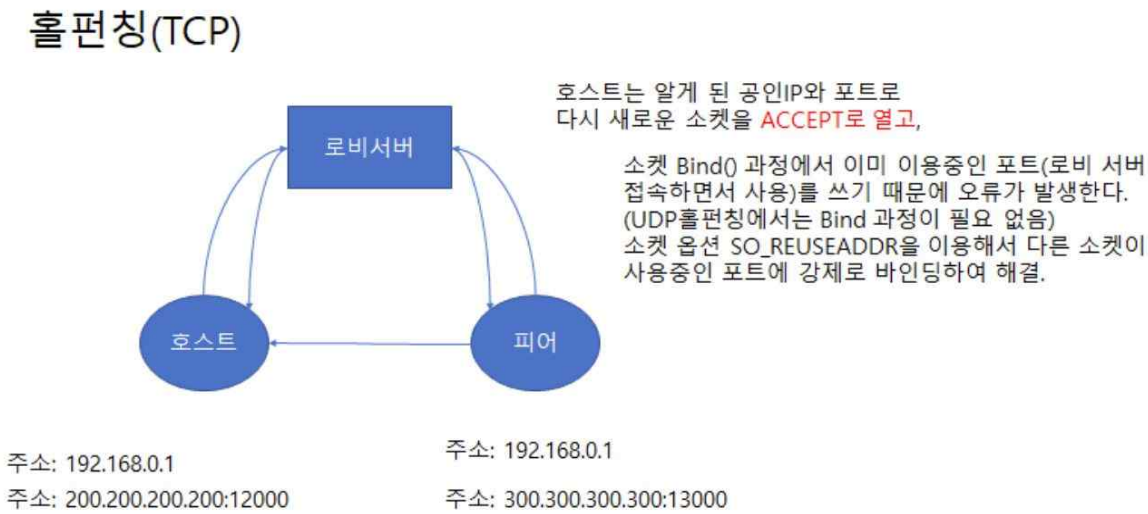


처음에는 하나의 서버가 여러 개의 룸의 로직을 모두 담당하여 처리하는 클라이언트-서버 구조로 개발을 진행하였습니다. 서버에서 클라이언트의 동작을 그대로 시뮬레이션 하는 형태로, 이동 및 충돌처리와 전투판정 등을 서버에서도 하도록 하였는데 문제가 발생하였습니다. 클라이언트 파트에서 NVIDIA PhysX SDK를 이용하여 물리를 구현하였는데, 테스트해보니 몬스터 30마리 정도만 필드를 뛰어다녀도 게임 루프가 60프레임 밑으로 떨어졌습니다. 프레임을 유지하면서 여러 개의 룸에서 동시에 게임을 돌리기 위한 방법으로 P2P를 생각하였으나 홀펀칭이 문제가 되었습니다.

서버를 별도로 두고 클라이언트가 서버에 접속하면 IP와 펀칭된 PORT를 클라이언트에게 돌려준 뒤 다른 클라이언트에게도 알려주어 접속하는 형태로 기초적인 홀펀칭을 구현하였으나 성공률이 높지 않았습니다. 최종적으로는 룸마다 로직의 처리를 전담하는 서버를 하나씩 두는 데디케이트 형태로 서버를 구성하였습니다.

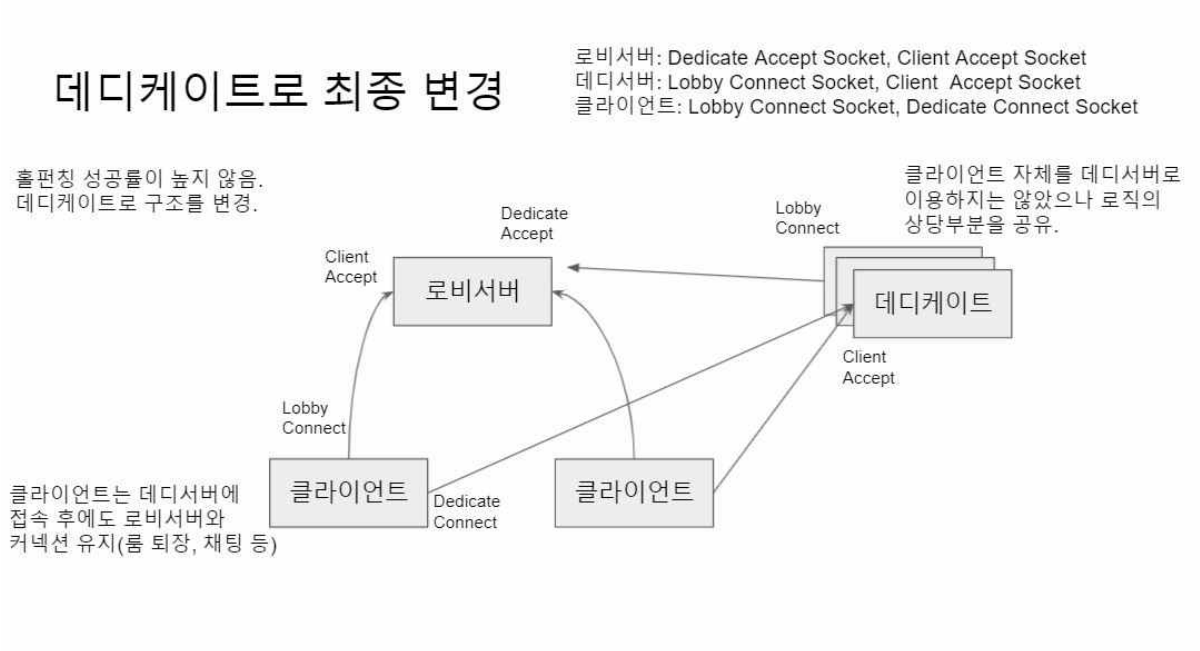


P2P구조에서는 한 룸에 2명이 들어가 플레이하는 형태의 게임이기 때문에 간단히 방을 개설한 유저가 호스트가 되어 다른 클라이언트의 접속을 받도록 설계하였습니다. NAT환경 안에서는 자신의 IP가 공인 IP인지 사설 IP인지 알 수 없기 때문에 각 피어들의 NAT환경 외부에 존재하는 서버에 패킷을 보내고, 서버는 접속한 클라이언트의 공인 IP와 접속한 포트를 돌려주도록 하였습니다.



Tcp를 이용했기 때문에 호스트는 알아낸 자신의 공인 IP와 펀칭한 포트를 이용해서 새로운 소켓을 Accept하고, 룸에 있는 다른 피어는 서버로부터 호스트의 IP와 포트를 받아 Connect 합니다. 호스트는 새로운 소켓을 열면서 Bind() 과정에서 서버에 접속하기 위한 소켓이 이미 사용 중인 포트를 이용하려고 하여 에러가 발생하는데, 소켓에 SO_REUSEADDR 옵션을 주어 재사용할 수 있도록 처리하였습니다.

위와 같은 방법으로 홀펀칭을 구현하였으나 성공률이 높지 않아 P2P 방식을 포기하고 최종적으로 데디케이트 방식으로 서버 구조를 변경하였습니다.



여러 개의 룸에서 동시에 게임이 돌아가면서 공유기를 이용하더라도 접속에 문제가 없어야 하는 조건을 만족하기 위해서 데디케이트 방식을 이용하였습니다. 룸마다 게임의 로직을 전담하는 서버를 하나씩 두어 이동 및 충돌, 전투 판정, 몬스터 AI, 길 찾기 등의 일들을 처리하도록 하였습니다. 클라이언트 자체를 데디케이트 서버로 쓰지는 않았으나, 클라이언트와 상당부분 코드를 공유하여 서버에서 로직을 다시 개발해야 되는 일을 최대한 지양하였습니다.