프로젝트 mashiro

서버 개발자 이재원

프로젝트 mashiro - 소개

기술

- 서버: C/C++, 클라: Unity
- IOCP
- 데드 레커닝
- 멀티스레드
- MySQL

기능

- 회원가입/로그인
- 캐릭터 목록/생성/삭제
- 방 CRUD/ 입장/퇴장
- 인게임 동기화 처리
  - 이동, 대쉬, 근거리/원거리 공격, 사망/부활 처리, 채팅

시연 영상: https://youtu.be/6lo47q-b-XQ
깃허브: https://github.com/Section80/mashiro_public

기술적 방향성

WindowsAPI를 적극 사용

- IOCP, SRWLock, Event, WaitableTimer, …

동기화 복잡성 줄이기

- 컨텐츠 코드에서는 락을 직접 걸지 않는다.

느린 동적할당 줄이기

- Update() 함수에서 힙 할당을 매 프레임 하지 않는다.
- 메모리 풀을 사용한다.

MessageQueue

Message = Event + Job

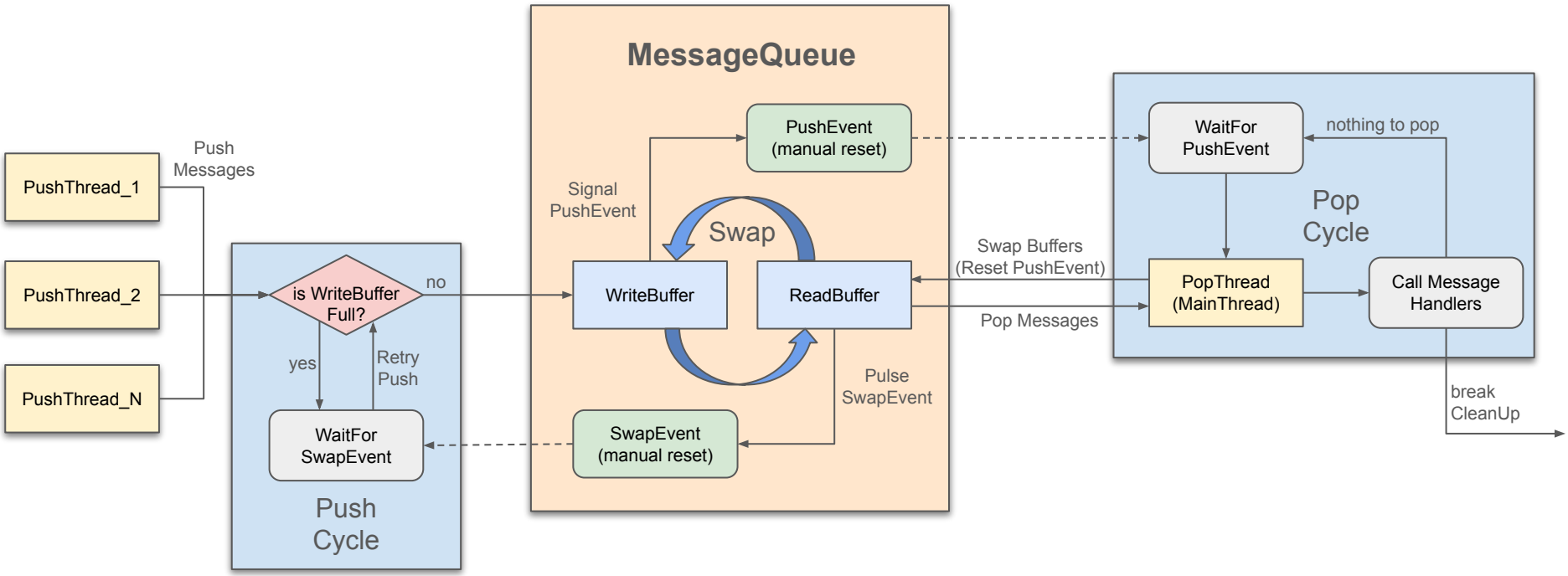
- **Event**: Pop하는 쪽에서 처리 로직을 결정한다.
  - Packet 도착을 알리는 PacketEvent는 직렬화된 Packet을 포함한다.
- **Job**: Push 시점에 나중에 수행할 일이 정해져 있다.

더블 버퍼링으로 작동: Push 스레드가 Pop Thread를 방해하지 않는다.

경쟁적으로 push해도 writeIndex만 증가시키는 **가벼운 락**을 건다.

이벤트(SwapEvent, PushEvent)를 통해 PushThread와 PopThread가 서로 **동기화**된다.

MessageQueue



NetIO

프로토콜은 **TCP**이며 **IOCP**로 구현하였다.

- 패킷이 완성되면 Main MessageQueue에 PacketEvent를 push한다.

반응성을 위해 **Nagle 알고리즘**을 끄는 대신 **FlushSend** 로직을 구현하였다.

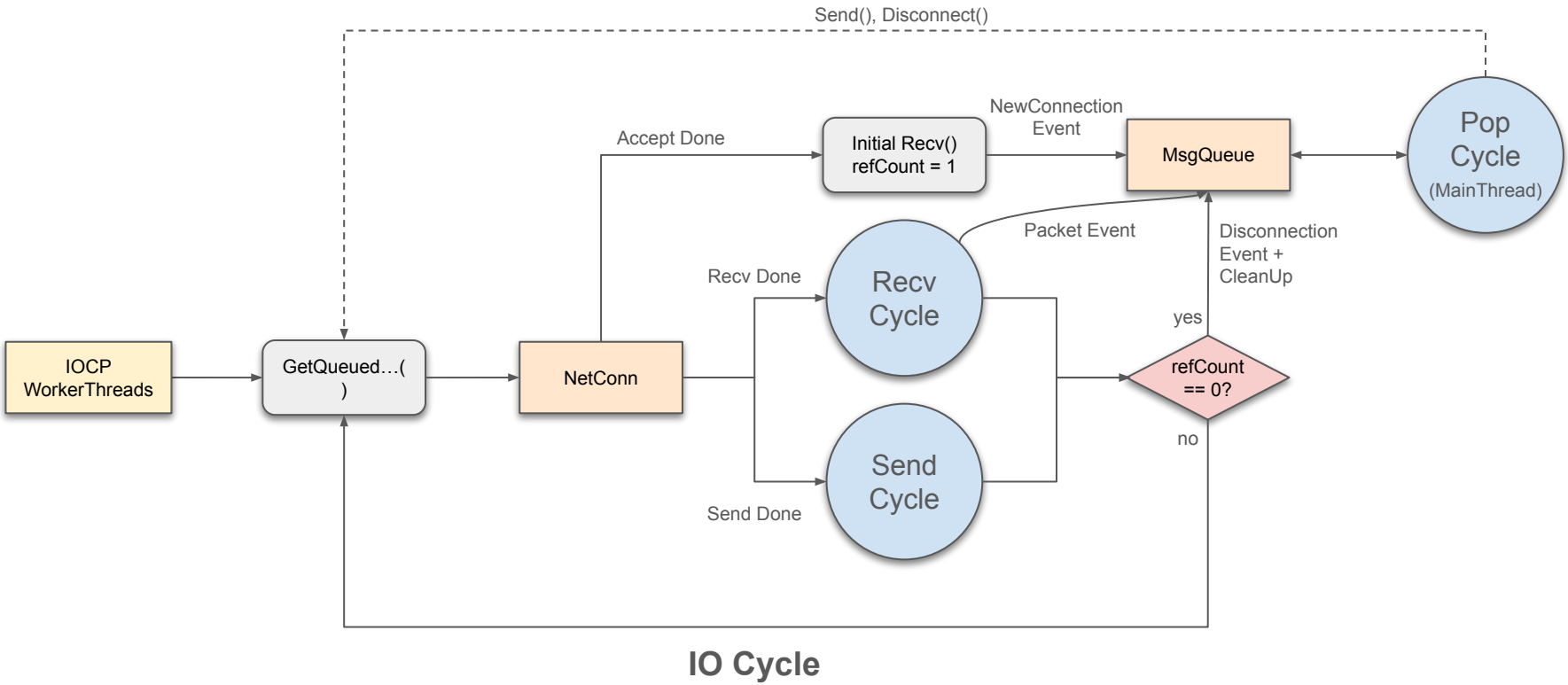
- WSASend() call을 줄여 패킷 헤더 사이즈를 줄인다.
- 즉시 flush하지 않으면 주기적으로 flush된다.

Send/Recv에 동적할당을 하지 않는다.

RefCount를 통해 소켓 수명(pend된 IO 수)를 확인한다.

- shared_ptr처럼 Connection 객체 자체의 수명을 관리하는 것은 아님!

NetIO - NetConn Lifecycle



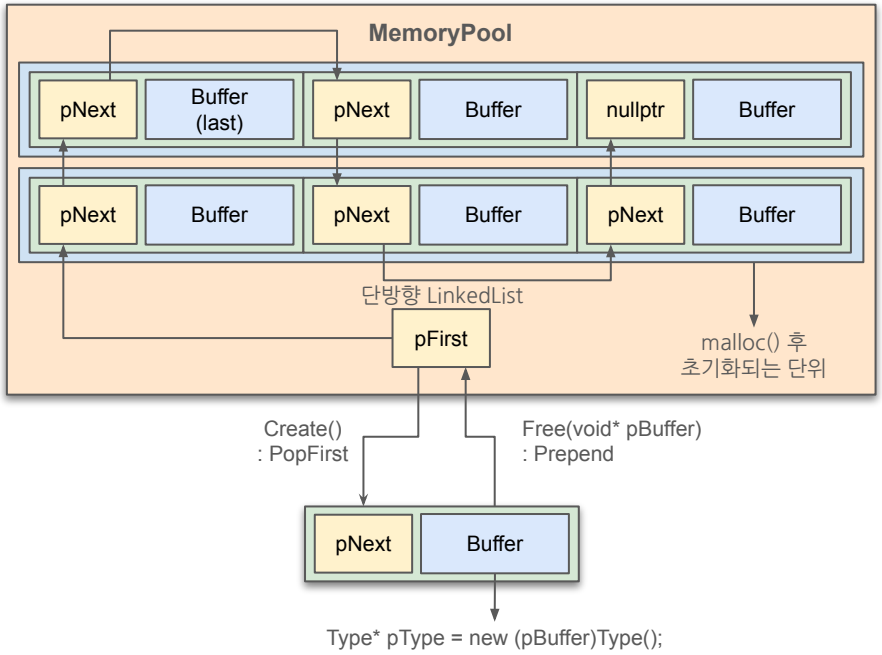
MemoryPool

타입이 없고 byteSize만 있는 **c-style**의 메모리 풀이다.

- v-table을 초기화하기 위해 placement new를 사용한다.

유저에게 돌려줄 **메모리 블럭**은 서로 **단방향 링크**로 연결되어 관리된다.

- malloc()을 통해 한 번에 여러 개의 링크 + 메모리 블럭을 연속해 할당하고 연결해둔다.
  - 8byte align
- 할당 요청 시 항상 pFirst를 돌려준다.
- 반납 시 링크를 다시 연결하고 pFirst로 만든다.
- pFirst가 null이면 로그를 찍고 또 malloc()한다.
- free()는 서버 프로세스를 종료할 때만 한다.



MainLoop

MainThread는 PushEvent, UpdateEvent, UpdateFps 이벤트를 기다린다.

PushEvent: Main Message Queue에 Push됐을 때 signal된다.

UpdateEvent: Update waitable timer에 의해 주기적으로 시그널된다.

- N개의 Update Thread가 signal되어 병렬적으로 update를 수행한다.
- Update Thread는 서로 경쟁하지 않는다.

UpdateFpsEvent: 주기적으로 signal되어 서버 fps를 drop/restore한다.

- 가변 Frame Rate: 60, 30, 15, 8

AsyncQueue - 비동기 처리, DB Query

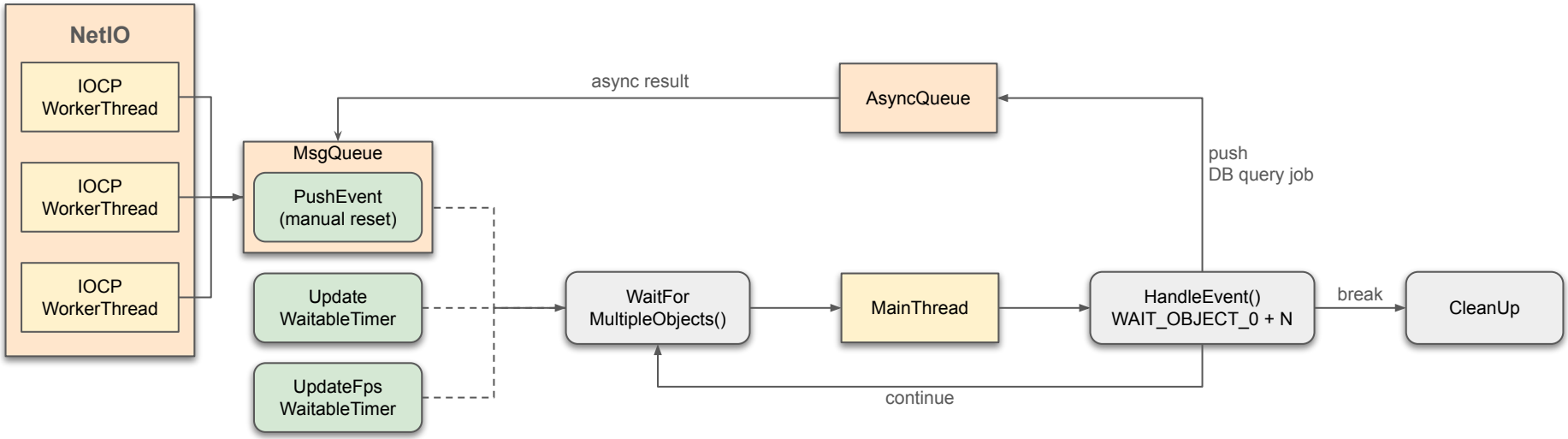
1개의 Disptacher와 N개의 Async Worker Thread로 구성된다.

Dispatcher: PushEvent를 받고 깨어나 Worker Thread들을 깨운다.

Async Worker: ReadBuffer에서 처리할 작업을 받아 처리한다.

- 각 Async Worker는 DB와 연결을 맺는다. (ODBC 사용)
- NetConn마다 Async Worker를 지정하여 쿼리 순서가 꼬이지 않게 한다.
- Round Robbin으로 Async Worker를 지정해준다.
- 쿼리 결과는 Main Message Queue에 Push해서 통지한다.

MainLoop


NetIO
IOCP WorkerThread
IOCP WorkerThread
IOCP WorkerThread
async result
AsyncQueue
MsgQueue
PushEvent (manual reset)
Update WaitableTimer
UpdateFps WaitableTimer
push
DB query job
WaitFor MultipleObjects()
MainThread
HandleEvent() WAIT_OBJECT_0 + N
break
CleanUp
continue

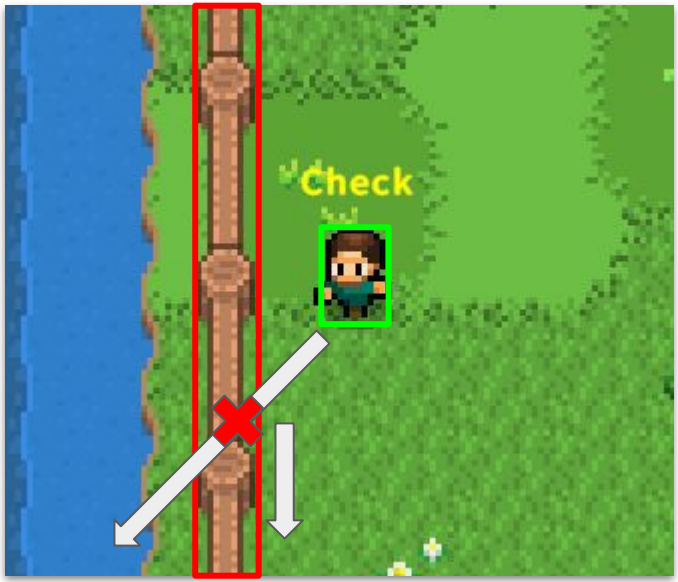
충돌처리 - SweepAABB

충돌 처리는 두 파트로 나뉜다.

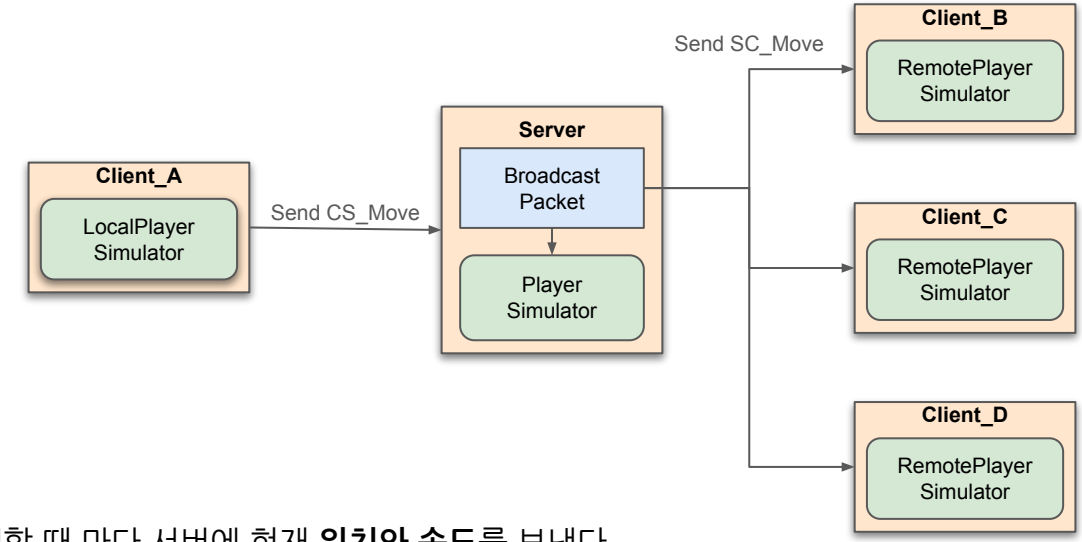
- **물리 처리**: 속도와 충돌체를 고려해 다음 프레임에서의 위치를 결정한다.
- **이벤트 처리**: 로켓이 폭발했을 때 데미지 처리 등 유저 로직 함수를 호출한다.

뚫는 것과 겹치는 것은 다르다.

- **뚫음**: 속도가 너무 빨라 다음 프레임에서 벽을 뚫는다.
  - 알고리즘의 문제다. SweepAABB는 이 문제를 해결해준다.
- **겹침**: 벽까지 이동했지만 float 오차에 의해 벽과 겹치게 된다.
  - **EPSILON** 값을 조절해 문제를 해결하였고, 테스트 결과 겹침 문제가 발생하지 않았다.



데드 레커닝 - 서버사이드 충돌처리



클라는 유저 입력이 발생할 때 마다 서버에 현재 **위치와 속도**를 보낸다.

- **Sampling Rate**를 통해 보내는 패킷량 조절

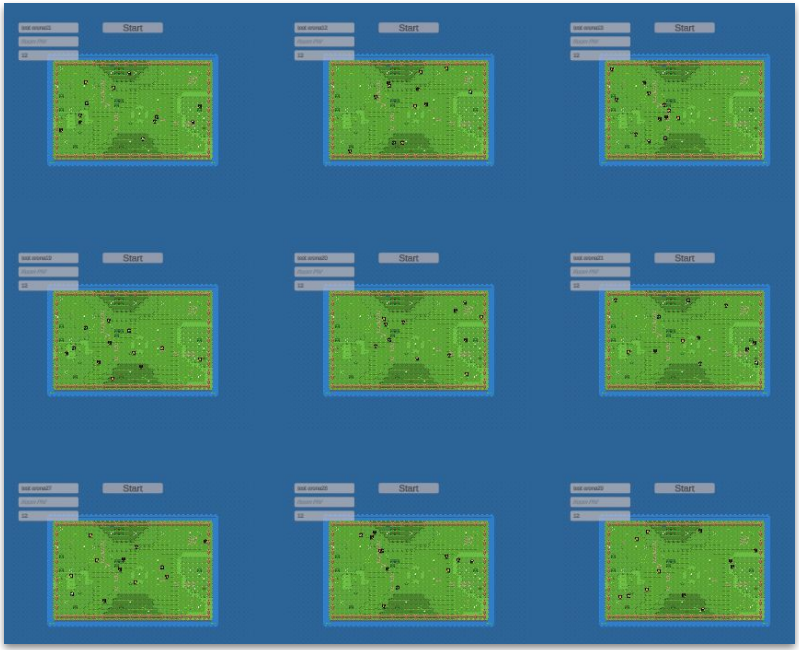
Simuator: 서버에서도 클라와 같은 지형 데이터로 **매 프레임** 충돌 처리를 한다.

물리 처리는 클라/서버에서 동일하게 하지만, **이벤트 처리**는 서버에서 주로 한다.

DummyTest

DummyTestTool 제작하여 테스트

- Azure VM Standard B4ms
- 방 256개 * 16명 = 4,096 수용
  - FPS 60, 낮은 핑 유지
- 서버 정상 종료 확인
  - Disconnect 처리
  - MessageQueue 비우기
  - MemoryPool 반납 확인
  - 메모리 누수 없음



DummyClientTool(Unity)

감사합니다.