**정책들의 설명과 장단점 적기**

\* 샤딩 : 데이터베이스를 조각내는 것

- 장점 : 서버를 확장할 때 서버 머신의 대수를 증설하기 때문에 확장 비용이 낮다, 여러 서버 컴퓨터로 부하가 분산되므로 처리 가능 총량이 높다.

- 단점 : 여러 머신에 걸쳐 비동기 프로그래밍 방식으로 작동하므로 오류 가능성이 높고, 여러 서버 컴퓨터 간 메시징이 오가면서 처리하므로 단위 처리 속도가 낮다. 다른 월드로 이동 불가

\* 기능별 분산 : 한 머신이 처리해야할 데이터의 처리 단계를 세분화해서 여러 머신들이 나누어서 처리, 사용자 분리 없이 확장, 기능별로 분리된 여러 종류의 서버 조합

- 장점 : 어느 한 서버가 병목이 되는 것을 피할 수 있다.

- 단점 : 너무 세세하게 분산한면 통신 오버헤드의 증가를 고려해야 함

접속 분할의 경우 반드시 게임에서 로그아웃을 해야 채널 이동이 가능

공간 분할을 Seamless로 구현하는 것이 Portal방식으로 구현하는 것보다 어렵다.

**#pragma pack(push, 1)을 사용하는 이유?**

1바이트 단위로 정렬하게 되므로 남는 공간 없이 자료형 크기 그대로 메모리에 올라가기 때문에

**클라이언트와 서버 사이의 프로토콜을 고급 언어의 구조체로 정의하지 않고, 메모리 블록(단순 배열)과 내부 인덱스로 정의하는 경우의 문제점은?**

코드의 가독성이 떨어지고 코드를 수정하기 어렵다.

**서버 컴퓨터에서 사용되는 CPU인 XEON이나 EPYC의 경우 코어의 개수가 일반 데스크탑 CPU의 몇 배가 되는 이유?**

서버는 병렬처리가 기본이므로 일반 CPU보다 서버에 사용되는 CPU가 더 많은 코어를 보유하기 때문이다.

**게임 서버 프로그래밍을 할 때 캐릭터 레벨이 1에서 100까지 라면 이를 저장하기 위해 int를 사용하지 않고 char를 사용하는 이유**

Int는 4byte이고 char는 1byte인데 캐시에 부담이 가고 메모리를 낭비하기 때문에

**게임 서버 프로그래밍을 할 때 주의사항으로 접속을 받기 시작하면 FILE I/O를 하지말라는 이유**

게임 서버는 recv, send를 계속하는 비동기 I/O인데, FILE I/O를 하면 대기 시간이 길어져 서버에 렉이 걸리고 overhead가 발생할 수 있기 때문에

**Process와 Thread의 차이점으로 통신 오버헤드를 둘 수 있다. Process사이의 통신이 Thread사이의 통신보다 부하가 큰 이유**

System call 해야 하기 때문에

**Overlapped I/O**

- 비동기 I/O라고도 불린다.

- Linux에서는 epoll()를 통해서 Overlapped I/O스타일의 I/O 모델을 구현 가능

- 하나에 소켓에 대해서 여러 개의 send를 동시에 처리할 수 있다.

- 데이터 전송을 위해 send()대신 WSASend()를 사용

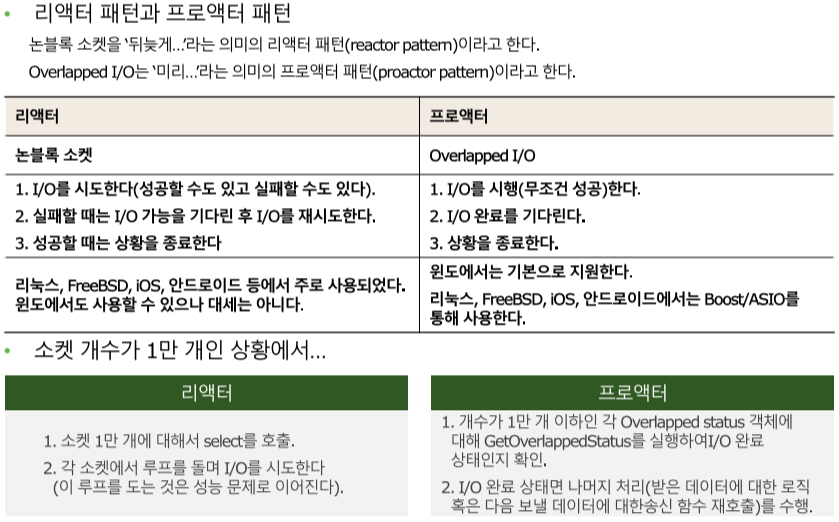
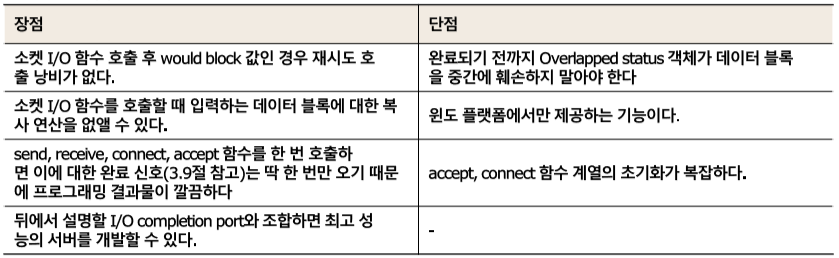
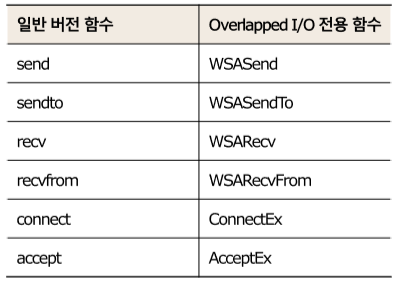
- Overlapped 구조체는 다른 곳에서 사용하지 않고 있다면 재사용 가능

- Overlapped를 통한 send나 recv요청은 Blocking방식으로 처리되지 않는다.

- Overlapped를 통해 데이터 수신을 할 경우 앞에서 한 recv요청이 완료되는 것을 기다렸다가 recv를 해야한다.

- Overlapped를 통한 send나 recv호출은 대기없이 즉시 리턴한다.

- Overlapped를 통해 데이터를 보낼 경우 이전 send함수의 callback을 받지 않아도 send가 가능



**Epoll**

어떤 소켓이 I/O가능 상태가 되면 이를 감지해서 사용자에게 알림을 해주는 역할을 함

**IOCP**

소켓의 Overlapped I/O가 완료되면 이를 감지해서 사용자에게 알려주는 역할을 함

사용자는 IOCP에서 I/O가 완료되었음을 알려 주는 완료 신호(completion event)를 꺼낼 수 (pop) 있다.

소켓 개수가 1만 개라고 하더라도 이 중에서 I/O가 완료된 것들만 IOCP를 이용해서 바로 얻을 수 있어서 모든 소켓에서 루프를 돌지 않아도 됨

전용 API : CreateIoCompletionPort(IOCP를 만들거나 입출력 핸들과 연결하는 함수), PostQueuedCimpletionState(IOCP 큐에 데이터를 입력하는 함수), GetQueuedCompletionStatus (IOCP에서 데이터를 가져오는 함수) 등이 있다.

CreateIoCompletionPort()에서 정의된 개수를 초과하는 스레드는 동시에 실행할 수 없다.

Socket의 close 여부는 GetQueuedCompletionPort()함수의 return 값을 통해 파악, Socket강제종료 여부는 함수의 에러값으로 파악

하나의 소켓에 대해 여러 개의 WSARecv가 동시에 실행되지 않도록 해야한다..

WSAAccept()실행 후, CreateIoCompletionPort(), WSARecv()를 차례대로 실행한다.

**IOCP의 GetQueuedCompletionStatus함수에서 부족한 정보를 확장된 Overlapped구조체를 통해 얻는다 이때 추가로 확보해야 하는 정보는?**

1. 어떤 작업을 수행 중인지(recv, send 여부)

2. 보낸, 받은 데이터가 얼마인지

struct OVER\_EX{ // IOCP 확장된 overlapped구조체

WSAOVERLAPPED over; // overlapped 확인

WSABUF dataBuffer; // overlapped 버퍼

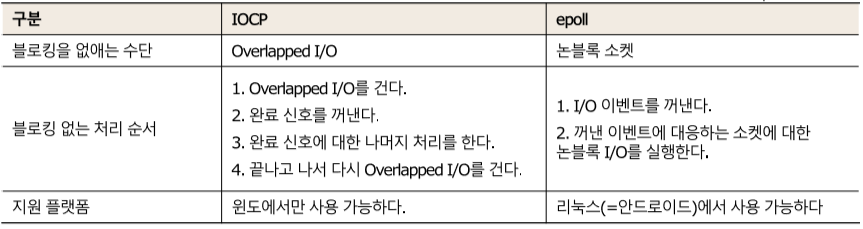
Char messageBuffer[MAX\_BUFFER]; // 송수신 버퍼용

Bool is\_recv; // recv인지 아닌지

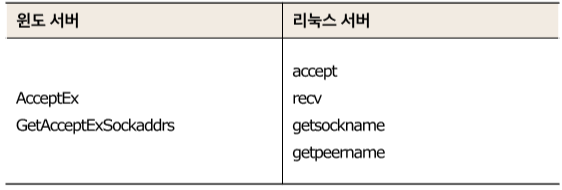
} // recv와 send를 구분하기 위해 사용

**IOCP와 epoll 차이점**

IOCP는 I/O가 완료인 것을 알려주지만 epoll은 I/O 가능인 것을 알려준다.



IOCP를 쓰는 윈도 서버에서는 연결 받기와 수신을 소켓 함수 호출 한번으로 끝낼 수 있고, 부가적으로 연결된 소켓의 끝점 정보를 얻는 것도 같이 끝내 버릴 수 있어 프로그램 최적화에 유리



**OX문제**

Thread사이의 동기화는 시스템 호출을 통해 이루어진다. **X**

Thread사이의 동기화에는 지역변수들을 사용한다. **X**

싱글 코어 컴퓨터에서 멀티쓰레드 프로그램을 실행하면 DataRace가 발생하지 않는다 **X**

싱글 코어 컴퓨터에서 멀티쓰레드 프로그램을 실행하면 메모리 일관성 문제가 발생하지 않는다. **O**

C++11 STL컨테이너들은 mutex만으로 multi-thread에서 atomic하게 동작하게 할 수 있다. **O**

PC에있는 메모리는 mfence명령을 추가하면 Atomic Memory처럼 사용할 수 있다. **O**

피터슨 알고리즘은 Blocking 알고리즘 중에 하나이다. **O**

CAS연산을 HW로 구현하지 않으면 많은 non-blocking 자료구조를 구현하는 것이 불가능하다. **O**

공유메모리의 read/write만으로는 non-blocking의 CAS연산을 구현할 수 없다. **O**

Non-blocking 알고리즘인 lock-free 알고리즘의 수행 시간은 다른 쓰레드의 동작에 영향을 받지 않는다. **X**

**게임 서버 실행 시 모든 지형 Data를 메모리에 읽고 사용해야 하는 이유**

입력 delay를 없애기 위해서 클라이언트에서 먼저 이동을 해야 되기 때문에

**게임 실행 시 벽이나 장애물과의 충돌 검사를 게임 서버내에서만 하는 것이 아니라 클라이언트에서도 해야 되는 이유?**

클라이언트에서 충돌하면 send를 보내고 서버에서 recv를 받은 후, 충돌검사하여 다시 클라이언트에 send를 하는 동안 delay가 발생하여 클라이언트에서 그 delay로 버벅이거나 벽이나 장애물을 뚫고가는 현상을 막아주기 위해 먼저 클라이언트에서 충돌검사를 해야한다.

**게임서버에서 충돌검사를 위해 클라이언트의 폴리곤 모델을 사용하는 경우가 있다. 하지만 2D지형을 사용할 경우 Cell방식의 충돌검사도 많이 사용하는데 Cell방식의 이점은?**

충돌검사가 빠르고 메모리가 절약된다.

**게임서버에서 길 찾기를 할 때 3D게임인 경우 Path node나 Navigation Mesh를 사용한다. 이 방식들은 직접 폴리곤 지형에서 길을 찾을 때와 비교하면 어떤 장점이있는가?**

타겟이 이동할 확률이 높고 지형이 변할 수 있기 때문에 이러한 부분에서 더 우수하다.

**게임서버에서 목표물을 향해 이동할 때 A\*알고리즘을 사용하여 길 찾기를 한다. 이 때 찾은 길의 Path List를 저장하지 않고 매번 이동 시 마다 다시 길 찾기를 하는 이유는?**

목표물에 다가갈 때 목표물이 가만히 있지 않는 경우도 있고, 이동한 길을 다시 돌아갈 필요가 없기 때문이다.

**IOCP의 worker thread에서 recv를 처리하는 프로그램을 작성할 때 주의점**

도착한 패킷이 완전한 패킷인지 모르기 때문에 재조립하는 부분이 꼭 필요하다.

**게임서버**

패키지게임 : offline game

MOG : multi player online game, MMOG : massively MOG

MO게임 종류

- 전용 서버(custom server) : 게임에서 제공하는 전용 서버, 동접을 많이 할 수 없다.

- 데디케이트 서버(dedicated server) : 서버에서 그래픽부분은 처리하지 않고 로직만 처리하는 것, 싱글플레이와 멀티 플레이가 서버와 통신하는건 똑같다. 일부 패키지 게임에서 데디케이트서버를 제공해서 다른 사용자에게 공익을 제공함

MUD게임 (Multi User Dungeon) : 여러 사람이 동시에 던전에 들어가서 모험하는 방식의 게임으로 텍스트로만 진행됨

게임 클라이언트와 서버의 상호작용

- 연결 : 최초로 클라이언트가 서버와 데이터를 주고받을 준비를 하는 것

- 요청-응답 : 연결을 마쳤으면 클라이언트는 서버에 메시지를 보내고, 서버는 이를 처리한 후 결과를 응답해주는 것

- 능동적 통보 : 서버와 클라이언트가 서로에게 요청을 보낸 적도 없이 그냥 통보하는 것

- 연결 해제 : 서버는 클라이언트의 신원을 확인하여 연결을 계속 유지할지 추방할지 판단하는 것

게임서버의 역할

1. 여러 사용자와 상호작용

2. 클라이언트에서 해킹 당하면 안되는 처리

3. 플레이어의 상태 보관

게임서버의 품질

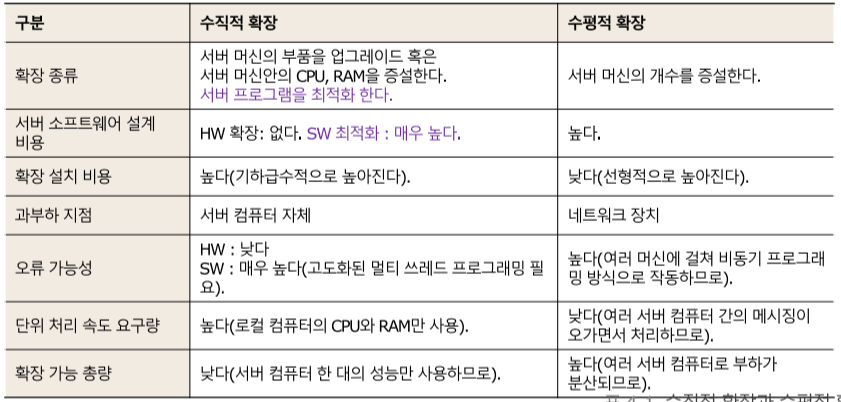
1. 안정성 : 게임 서버가 얼마나 죽지 않는가

- 소프트웨어 측면에서 안정성에 영향을 주는 요인 : 버그

- 극복방법 : 서버가 죽더라도 최대한 빨리 살아나게 함, 서버는 죽더라도 최대한 적은 서비스만 죽게 하기, 서버 오작동에 대해서 기록(로그)을 남기기

2. 확장성 : 서버를 얼마나 많이 설치할 수 있느냐

사용자가 많아지더라도 서버가 처리하는 속도가 하락하지 않게 하는 것



3. 성능 : 기본적으로 얼마나 빨리 처리하는지

- 성능을 높이는 방법

3.1. 코드 최적화나 알고리즘 최적화

3.2. path table 테크닉 사용

3.3. 더 빠른 속도로 실행되는 프로그래밍 언어 사용

3.4. 서버의 과부하 영역을 분산

3.5. 프로토콜 최적화

3.6. 네트워크 전송 시간을 줄이기 위해, 고품질 네트워크 회선을 가진 데이터센터에 서버를 설치

3.7. 지리적으로 가까운 데이터 센터에 서버들을 분산해서 설치 -> 지리적으로 가까운 플레이어끼리 플레이하게 유도

3.8. 서버를 거치지 않고 클라이언트끼리 직접 통신하는 P2P방식 -> 서버가 주고받는 메시지 검증을 할 수 없는 단점 존재

4. 관리 편의성

**데이터베이스 소프트웨어를 이용해서 플레이어 정보를 저장하는 이유**

데이터 관리와 분석을 빠르게 할 수 있다.

강력한 데이터 복원 기능이 있다.

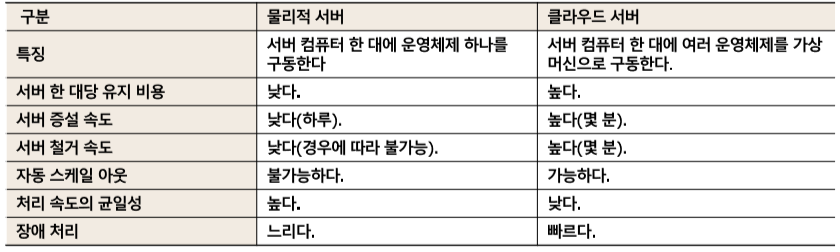
전부 아니면 전무로 데이터를 변경할 수 있다.

데이터 일관성을 유지시켜 준다.

처리가 2개 이상 동시에 실행될 때 한 데이터가 동시에 여러 데이터를 액세스하면서 이상한 결과가 나오는 문제를 막아주는 기능이 있다.

6. 장애에 대한 내성이 강함

**물리적 서버와 클라우드 서버**



**온라인 게임을 방해하는 레이턴시가 길어지는 요인**

서버가 멀리 있으면 네트워킹 중에 레이턴시가 추가됨

클라우드 서버 안에서 가상 머신은 다른 가상 머신이 CPU 사용량을 잠식하면서 조금씩 지연시간이 발생

패킷 드롭으로 인한 재송신은 간헐적인 큰 지연 시간을 발생시킴

인구가 낮은 국가에서는 인터넷이 느림

무선 네트워크에서는 레이턴시와 패킷 드롭률이 크게 증가

**레이턴시를 줄이는 방법**

TCP대신 UDP사용

똑 같은 양의 데이터를 보낸다고 하더라도 가급적 적은 수의 패킷으로 보내기

클라이언트와 서버 간 통신(C/S 네트워킹)과 클라이언트끼리 직접 통신하는 것(P2P)을 섞어 쓰기

**가시영역 필터링**

클라이언트 : 자기 근처에 있는 다른 캐릭터들 정보

서버 : 플레이어 각각에 대해서 각 플레이어가 볼 수 있는 캐릭터 정보, 캐릭터 각각에 대해서 자기 자신을 볼 수 있는 플레이어 목록