게임서버 중간 고사

1장

프로그램: 컴퓨터에서 실행되는 명령어와 데이터의 모음이 들어있는 파일

프로세스: 프로그램이 활동을 하는 상태

로딩: 프로그램에 있는 코드와 데이터를 프로세스 메모리로 불러들임

프로세스의 구성: 코드, 데이터, 힙, 스택

멀티스레드 프로그래밍을 해야하는 대표적 상황 – 기기에 있는 cpu를 모두 활용해야 할 때

이런 상황에서는 멀티스레드를 사용하면 안된다

- 오래걸리는 일 하나와 빨리 끝나는 일 여럿을 같이 해야 할 때

이유: 짧은 작업의 종료 시간이 뒤로 밀리는 경우 -> 차라리 오래걸리는 일을 쪼개서 처리

- 어떤 긴처리를 진행하는 동안 다른 짧은 일을 처리해야 할때

- 프로그램을 모듈화 하여 다른쓰레그에서 독립적으로 실행

-> 부작용이 심하다

single core의 사용량이 100퍼센트가 되지 않았다면 멀티스레드 프로그래밍을 절대로 하면 안된다.

이미 멀티스레드에 발을 넣은 경우 적용할수는있으나 이것을 이유로 멀티스레드를 사용하는건 안된다

뮤텍스

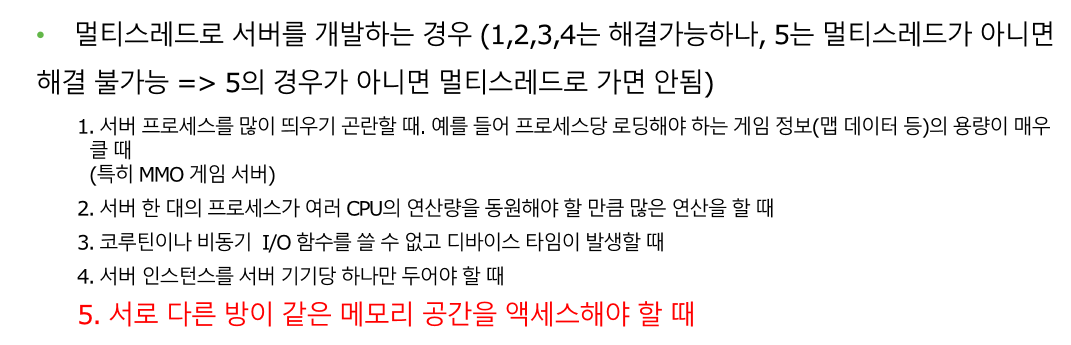
뮤텍스를 너무 잘게 나누면 = 오히려 프로그램 성능이 떨어진다 -> 뮤텍스를 엑세스하는 과정 자체가 무거움 프로그램도 매우 복잡해지고 교착상태가 쉽게 발생할수있다.

병렬성과 시리얼 병목

* 병렬성: 여러 CPU가 각 스레드의 연산을 실행하여 동시 처리량을 올리는 것
* 시리얼 병목: 병렬로 실행되게 프로그램을 만들었는데 정작 한 CPU만 연산을 수행하는 현상

싱글스레드 서버를 구동하는 경우 CPU개수만큼 프로세스를 띄우는 것이 일반적

1. 방 개수만큼 스레드나 프로세스가 있으면 스레드나 프로세스 간 컨텍스트 스위치 횟수가 증가
2. 따라서 같은 동접자를 처리하는 서버라고 하더라도 실제로 처리할 수 있는 동접수를 크게 떨어트린다



스레드풀링

1. 어떤 서버의 주 역할이 CPU연산만 하는 스레드라면 스레드풀의 스레드 개수는 서버의 CPU개수(=모든코어의 개수)와 동일하게 잡아도 충분함
2. 서버에서 데이터베이스나 파일등 다른것에 엑세스하면 디바이스타임이 발생해 스레드개수가 CPU개수보다 많아야한다.
3. 작업이 없으면 스레드들은 스레드풀에서 대기하고 작업이 들어오면 깨어나서 작업을 실행한다.

너무 좁은 잠금범위는 컨텍스트 스위치가 발생할 때 운영체제가 해야할일이 많아지고 처리 병렬성이 떨어지기때문에 멀티스레드 프로그래밍의 이유가 퇴색된다.

Windows IO모델

스레드를 통한 처리 단점 – 과도한 스레드 개수로 운영체제 오버헤드

논블로킹 단점 – recv를 돌아가면서 반복 체크해야한다, cpu낭비->성능저하

Select 단점 – 소켓 개수가 많아질수록 성능 저하, 유닉스나 리눅스에서는 소켓 개수의 한계가 존재

WSAAsyncSelect 단점 – 윈도우가 필요하고, 윈도우 메시지 큐를 사용해서 성능이 느리다

WSAEventSelect 단점 – 소켓 개수 64개 제한 (멀티스레드로 극복가능)

오버렙 이벤트 단점 – 이벤트개수 64개 제한, 개별적으로 작업결과 확이해야함

오버랩 콜백 단점 – 대규모 연결에는 부적합

3장

Overlapped IO

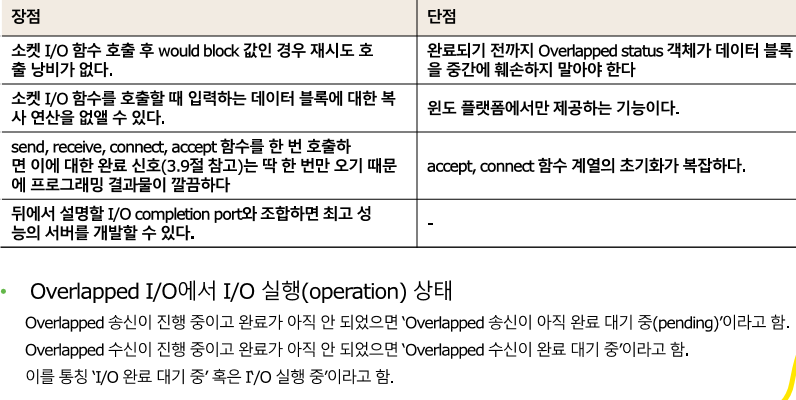
논블록 소켓의 장점

1. 스레드 블로킹이 없어 중도 취소 통제 가능
2. 스레드 개수가 1개이거나 적어도 소켓을 여러 개 다룰수있다
3. 스레드개수가 적거나 1개이므로 연산량의 낭비 X

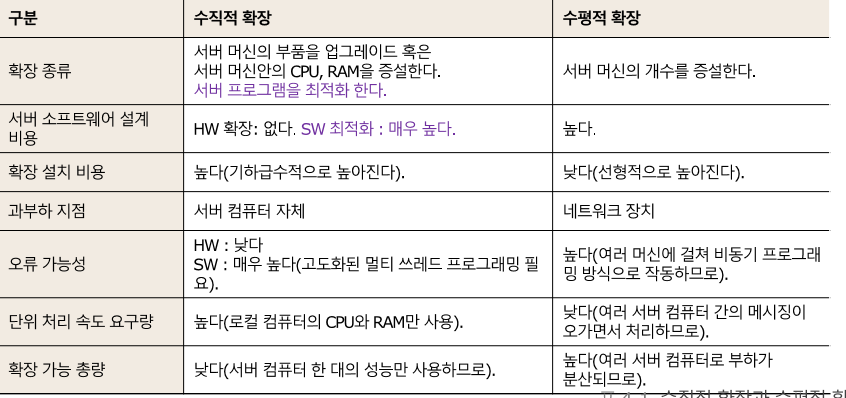
논블록 소켓 단점

1. 소켓 I/O함수가 리턴한 코드가 would block인 경우 재시도 호출 낭비 발생
2. 소켓 I/O함수가 호출할 때 입력하는 데이터 블록에 대한 복사 연산 발생
3. Connect 함수는 재시도 호출을 하지않지만, send 함수나 receive 함수는 재시도 호출해야 하는 api가 일관되지 않은 문제가 있음

Overlapped I/O 장단점



4장

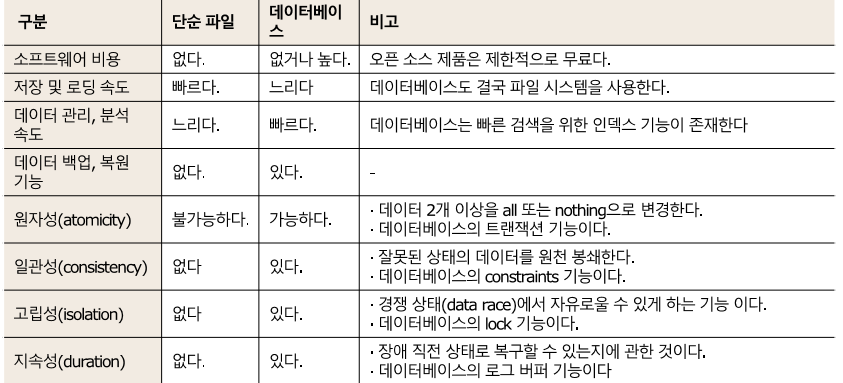


게임서버 성능을 높이는 법:

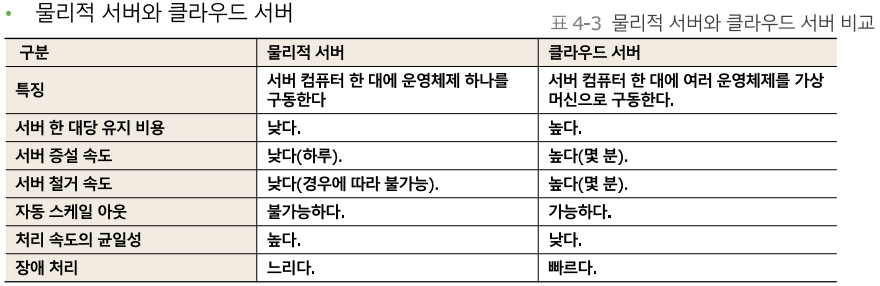
1. 서버의 단위처리 속도 높이기 – 코드최적화나 알고리즘 최적화
2. 서버의 과부하 영역을 분산 – 어떤 함수가 처리 시간을 많이 차지하는지 발견 후 그것에 집중하여 성능개선, 함수의 처리속도를 높이는 방법이없고 실행빈도를 낮추는 방법도 없다면 분산이 필요
3. 네트워크 프로토콜 최적화 – 메시지 양 줄이기
4. 네트워크 전송 시간 줄이기
5. 서버를 거치지 않고 클라끼리 직접 통신 – P2P방식

데이터베이스에 소프트웨어를 이용하여 저장하는 이유

1. 데이터 관리와 분석을 빠르게 함
2. 강력한 데이터 복원기능
3. 전부 아니면 전무 로 데이터를 변경할 수 있다
4. 데이터 일관성 유지
5. 처리가 2개 이상 동시에 실행될 때 한 데이터가 동시에 여러 데이터를 엑세스하면 이상한 결과가 나오는 문제를 막아준다
6. 장애에 대한 내성이 강하다



클라우드 서버: 리얼 머신 위에 구동하는 가상머신의 집합. 물리적머신 한대 안에서 여러서버 운영체제 동시작동



서버 운영체제 용도에 따라 나뉜다

- Linux, Linux tunnig(리눅스 튜닝): 돈이없을떄, 실력이 충분할때 혹은 퍼블리셔가 요구할때(해외)

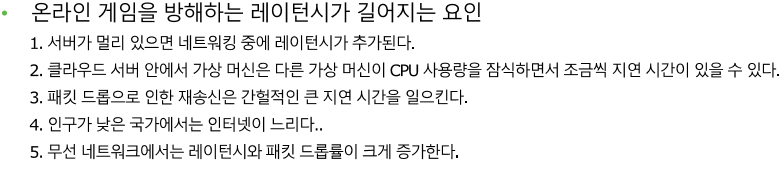
- Windows: 돈이 많을 때, 관리가 용이하고 성능이 좋음

클라우드서버

윈도우10 하이퍼5 가상머신기능제공

mmo게임은 클라우드서버에서 서버를 가동할 수 없다(성능 문제때문)

5장



레이턴시마스킹

1. 추측항법: 상대방 움직임을 어느정도 예상해서 그 위치로 갈 수 있게 보정시키는 방법
   1. 기기 A에서 B에 패킷을 보냄
   2. 기기B는 이를 받으면 기기A에 패킷을 보냄
   3. 기기A는 1 과정의 시간과 현재 시간의 차이를 구하여 2로 나눔
2. 일단 보여주고 나중에 얼렁뚱땅?
   1. 플레이어가 행동을 취하면 동시에 행동을 연출하는 모습 일부를 보여주고 서버에서는 행동명령을 받아 처리하고 플레리어에게 가해야 하는 행동을 클라에게 보낸다. 클라에서 메시지를 받으면 나머지 연출 시행

가시영역필터링: 서버가 가진 월드 전체에서 플레이어 가시영역에 있는 것만 보낸다(mmo서버에선 필수) 클라이언트부하 감소, 네트워크 트래픽 감소, 서버 부하 감소

락스텝: 컴퓨터프로그램의상태에 같은 입력을 주면 같은 결과가 나온다는 원리를 응용

1. 플레이어는 다른 플레이어들에게 입력명령을 내린다
2. 플레이어들의 명령에 따라 모든 클라이언트가 동시에 씬을 업데이트한다

락스텝으로 얻는 효과

1. 각 클라이언트 플레이어의 입력만 주고받으며 씬을 구성하는 캐릭터의 이동상태를 주고받지 않는다
2. 입력 명령은 통신량이 상대적으로 매우 적다

락스텝 동기화 알고리즘 한계

1. 다른플레이어가 플레이하고 있는 게임에 중간에 들어오는걸 만들기 힘들다
2. 물리엔진 등 게임 플레이에 관여하는 연산에 부동소수점을 쓸 수 없다(개발할 때 가장 까다로운점)
3. 플레이어 수가 많아지기 어렵다. 통신량이 플레이어 수에 비례해서 증가하기 때문에
4. 씬 업데이트가 일시 정지할 확률이 높다. 원활한 게임을 진행하려면 플레이어들 중에 레이턴시가 가장 높은 사람을 기준으로 미래시간이 결정 (스타1에서 렉있는 플레이어 추방하는 이유)
5. 입력 명령 속도에 민감한 게임에 부적합(슈팅게임

실제 레이턴시 줄이는 법: UDP 사용, 같은양의 데이터를 보내도 가급적 적은 수의 패킷으로 보낸다, 클라 서버 통신과 클라끼리 직접통신 P2P을 섞어 쓴다.

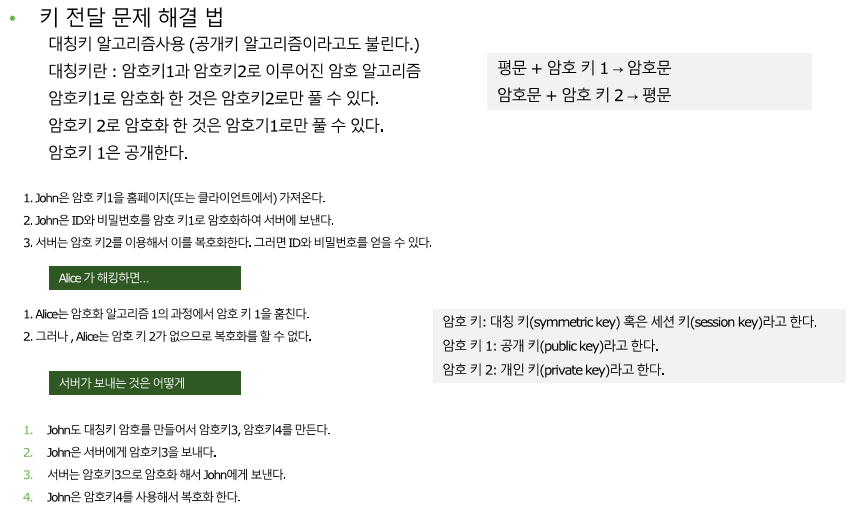
로그인 과정에서 클라와 서버가 대화하는 절차

1. 로그온 요청 메시지 서버로 전송
2. 서버는 파일이나 데이터베이스에서 해당 유저 ID 비밀번호 받아서 식별한다.
3. 식별결과를 클라이언트에 통보
4. 클라에 통보하면서 플레이어정보를 데이터베이스에서 로딩하여 게임 서버 메모리에 보관

해킹과 보안

게임사용자들이 해킹하는 유형

1. 크래킹: 게임 외 분야에서 흔히 이야기하는 해킹을 크래킹이라 한다. 타인의 ID 비밀번호를 도용하거나 비밀정보를 보는 것, 서버에 저장된 데이터를 훼손하거나 훔치는 것도 포함
2. 치트 혹은 조작: 게임에서만 발견되는 형태의 해킹, 내 능력치를 비정상적으로 높이거나 다른사람의 플레이를 망가뜨리는 것을 의미



iocp에서 오버렙 구조체 형태

struct OVER\_EX {

WSAOVERLAPPED over;

WSABUF wsabuf[1];

char net\_buf[MAX\_BUFFER];

bool is\_recv;

};

struct SOCKETINFO

{

OVER\_EX recv\_over;

SOCKET socket;

int id;

short x, y;

};