|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 13주차 | **기간** | 2023.9.26 ~ 2023.10.3 | **지도교수** | (서명) |
| 이번주 한일 요약 | 멀티플레이 서버 책 3장 재정리 | | | | |

<상세 수행내용>

소켓의 유형에는 크게 3가지가있다.

블로킹, 논블로킹, Overlapped 또는 비동기 I/O 소켓이 있다.

먼저 블로킹 소켓은 말 그대로 작업중에 멈춰버린다는 것이다. 송신 함수 호출이 일어났을때 작동 완료 될 때까지 리턴을 기다리며 멈춰 있는다.

하지만 논블로킹 소켓은 멈춰버리지 않는다. 송신 함수 호출이 일어나면 무조건 즉시 리턴하여 리턴값을 확인하여 처리한다.

밑은 네트워크 함수에 관련한 블로킹과 논블로킹 소켓의 차이이다.

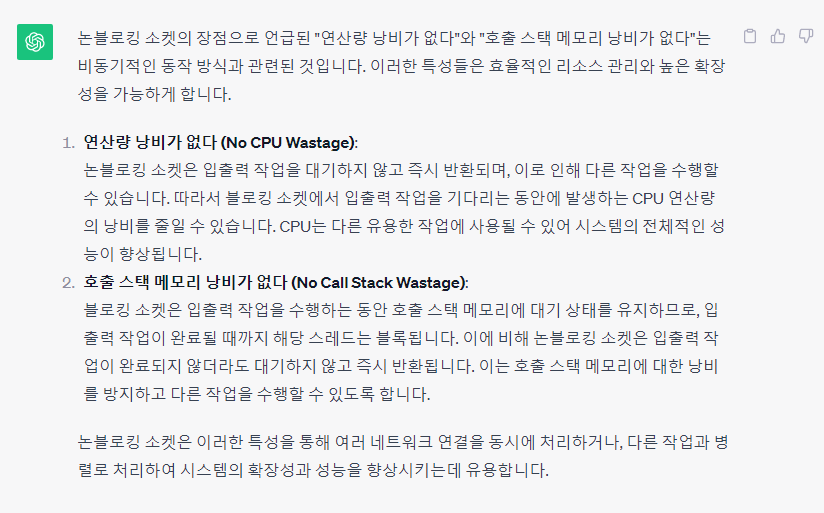
|  |  |  |
| --- | --- | --- |
|  | Blocking | NonBlocking |
| Connect() | Blocking, 상대가 수락, 거절, 존재하지 않다면 리턴 | 무조건 바로 리턴. 리턴 값이 Would Block 이라면 소켓 안에 무슨 일이 있었다 |
| Send() | Tcp: 수신 버퍼가 다 차 있다면 Blocking | Would Block or 데이터 크기가 바로 리턴 |
| Udp: 수신 버퍼가 다 차 있다면 보내는 족족 버려짐 |
| Receive() | 수신 버퍼가 비었다면 Blocking | 비었다면 Would Block 리턴 후 잠시 후 재호출 |
| Accept() | Blocking 후 연결이 들어오면 리턴 | 연결이 없다면 Would Block 리턴 |

논블로킹 소켓의 장점은

1. 스레드 블로킹이 없으므로 취소같은 통제가 가능

2. 스레드 개수가 1개거나 적어도 소켓을 여러개 다룰 수 있음.

3. 연산량 낭비가 적음, 호출 스텍 메모리 낭비가 없음.



단점은

1. 소켓 I/O 함수가 would block을 리턴한 경우 재시도 호출 낭비

2. 소켓 I/O 함수 호출시 입력하는 데이터 블럭에 대한 복사 연산 발생

3. connect() 함수는 재시도 호출을 하지 않지만 send()나 receive() 함수는 재시도 호출해야 하는 API가 일관되지 않음.

단점 1번을 더 자세히 설명하자면, 송신 데이터가 5바이트, 남은 수신 버퍼가 1바이트라고 할 때, TCP에선 문제가 없지만, UDP인 경우 데이터 일부만 보낼 수 없기에 계속해서 would block을 호출하는 상황이 발생한다.

단점 2번을 더 자세히 설명하자면, 보내기 위해선 송신 버퍼에 데이터를 채워야하는데 이를 복사 연산으로 한다. 하지만 송신 버퍼는 운영채제 커널 내에 있으므로 RAM에 존재하고, 보내야할 사용자 프로세스 내 데이터 블럭은 CPU에 있으므로 둘 속도 차이로 연산 속도가 느려진다.

텍스트, 폰트, 친필, 그래픽이(가) 표시된 사진

자동 생성된 설명

이러한 두 단점을 해결해주는것이 Overlapped 또는 비동기 I/O이다.

Overlapped 전용 함수가 존재하기 때문에 그걸 사용하면 된다.

하지만 I/O 실행이 별도로 동시간대에 진행되기 때문에, 호출한 Overlapped I/O 전용 함수가 비동기로 하는일이 완료되기 전에 Overlapped status객체와 데이터 블럭을 건들지 말아야한다. 운영체제가 데이터 블럭을 백그라운드에서 액세스하기 때문이기도 하다.

|  |  |
| --- | --- |
| **장** | **단** |
| 소켓 I/O 환수 호출 후 would block 값인 경우 재시도 호출낭비가 없다. | 완료되기 전까지 Overlapped Status 객체가 데이터 블록을 중간에 훼손하지 말아야 한다. |
| 소켓 I/O 환수를 호출할 때 입력하는 데이터 블록에 대한 복사연산을 없앨 수있다. | 윈도 플랫폼에서만 제공하는 기능이다. |
| send. receive, connect, accept 함수를 한 번 호출하면 이에 대한 완료 신호는 딱 한 번만 오기 때문에 프로그래밍 결과물이 깔끔하다. | accept, connect 함수 계열의 초기화가 복잡하다. |
| I/O completion port(IOCP)와 조합하면 최고 성능의 서버를 개발할 수 있다. |  |

참고로 논블럭 소켓은 상태 확인 후 행동하기에 '뒤늦게'라는 의미의 "리액터 패턴",

Overlapped I/O 소켓은 행동 후 결과 확인하기에 '미리'라는 의미의 "프로액터 패턴"이라고 부른다.

하지만 두 소켓 다 소켓 수에 비례하여 루프를 돌기 때문에 성능문제가 있다. 이 부분을 해결하기 위해 epoll과 IOCP가 등장한다.

epoll방식은 리눅스에서 제공하는 논블로킹 소켓을 대량으로 보유중일때 효율적으로 처리해주는 API이다.

여러 논블로킹 소켓을 모니터링하여 I/O가능한것들을 꺼내와 작업하는 방식으로 이벤트 기반으로 작동한다.

IOCP는 epoll과 다르게 논블로킹이 아닌 Overlapped I/O 소켓을 대상으로 윈도우에서 효율적으로 처리해주는 API이다. Overlapped I/O가 완료되면 감지하여 사용자에게 알려 주는 역할을 함. 완료된 소켓을 IOCP 내장 큐에 푸시하여 사용자에게 완료 신호를 꺼낼 수 있게 함. 이렇게 완료된 것들만 IOCP를 이용해서 사용하기 때문에 모든 소켓에서 루프를 돌지 않아도 됨.

|  |  |  |
| --- | --- | --- |
| **구분** | **IOCP** | **Epoll** |
| **블로킹을 없애는 수단** | Overlapped I/O | 논블럭 소켓 |
| **블로킹 없는 처리 순서** | 1. Overlapped I/O를 건다 2. 완료 신호를 꺼낸다 3. 완료 신호에 대한 나머지 처리를 한다 4. 끝나고 나서 다시 Overlapped I/O를 건다 | 1. I/O이벤트를 꺼낸다 2. 꺼낸 이벤트에 대응하는 소켓에 대한 논블럭 I/O를 실행한다. |
| **지원 플랫폼** | 윈도우 | 리눅스, 안드로이드 |

진행 상황 유튜브 링크:

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** | 시골이라 온라인으로 작업가능한 것이 없다. | | |
| **해결방안** | 일단은 프로토타입 완료까지 1주일 남았으니 거기에 신경쓴다. | | |
| **다음주차** | 15주차 | **다음기간** | 2023.10.4 ~ 2023.10.10 |
| **다음주 할일** | 프로토타입 개발에 착수하기 | | |
| **지도 교수**  **Comment** |  | | |