Buffer Management

YoungHand

2017년 8월 15일

차 례

제 1 절 개요

네트워크로 전송과 수신을 위한 데이터를 보관하고 암호화나 프로토콜 처리를 위한 변환 후 처리하기 위한 메모리 버퍼 사용이 필수적이다.

이들 버퍼는 빈번하게 사용되므로 효율적으로 관리되어야 하고 메모리 복사가 메모리 할당 다음으로 느리므로 빠르게 처리되어야 한다.

1.1 목표

- 버퍼의 할당과 해제를 최소화한다
- 버퍼 복사를 최소화한다.
- 버퍼 복사를 최대한 효율적으로 한다.

버퍼 풀을 사용 시 락이 필요하게 된다. 락 충돌을 최소화 할 수 있어야 한다.

제 2 절 전송

2.1 기존 구현

session::on_send로 프로토콜 변환 처리를 한다. on_send 호출 후 `segment_buffer`에 쓰고 `session_mutex_`를 사용하여 전송 상태로 설정하고 `send_segs_mutex_`에 락을 건 상태로 전송 요청을 한다.

전송 요청 과정은 대부분 serialize 된다. 이 부분이 크게 문제가 되지 않는 이유는 전송 중일 경우 `send_segs_::append`만 시리얼라이즈 되기 때문이다.

2.1.1 pool의 thread-safety

`segment_buffer`는 길이 단위로 클래스가 생성되고 풀은 static으로 선언되어 공유된다. pool은 concurrent-queue를 사용하고 다른 공유되는 부분은 없도록 세심하게 처리되어야 하고 그렇게 구현된 것으로 보인다.

2.1.2 개선 방향

petri-net 분석 segment-buffer에 대한 락이 조금 거슬린다. 그 외에 sending 중인 정보에 대한 락도 조금 거슬린다. 확실하게 동작하도록 하고 개선 하는 게 맞을 수 있다. 이 부분에 대한 상태 다이어그램을 petri-net이나 FSM으로 정확하게 그리고 개선 점을 찾아 본다.

복사 segment에 대한 복사가 발생한다. segment 크기에 따라 반복처리되므로 sgement 크기가 애플리케이션에 맞게 구성 가능해야 한다 게임에서는 최소 32K는 되어야 할 것 같다. 다른 구성으로 서로 다른 크기의 버퍼 풀을 갖고 크기에 맞춰 해당 풀을 사용하는 방법도 있다. 어느 쪽이 더 나은가? 어떻게 평가할 것인가?

정리 전송 쪽은 락 충돌이 있는 부분과 버퍼 풀의 크기별 사용이 살펴볼 부분이다. 당장은 확실하게 더 나은 방법이라고 보이는 것은 없다.

제 3 절 수신

3.1 기존 구현

네트워크 단에서 32K 고정 버퍼로 데이터를 받는다. 받은 데이터를 프로토콜 별 세션으로 넘겨준다. session-msgpack을 예로 보면 수신 버퍼에 쓰고 난 후 메세지를 만들어서 넘겨준다.

메세지를 만드는 동안에는 수신을 처리하지 않는다. 이유는 io 쓰레드에서 packetization 등 프로토콜 처리를 해서 application 쓰레드의 부담을 덜어주고 io 쓰레드 풀 자체가 제한적이기 때문이다.

3.2 복사

네트워크 단에서 받은 데이터를 각 세션별 프로토콜 처리 버퍼로 넘길 때 1번 복사가 발생하고 처리 버퍼를 통해 메세지로 만들어 질 때 한번 더 복사가 발생한다.

제 4 절 암호화

per message encrytion w/ session key로 한다. session key는 고정 키에서 데이터에 따라 바꿔 나가는 방식을 사용한다. 최초 키의 생성과 교환이 프로토콜마다 정해질 수 있도록 한다.

암호화된 메세지를 in-place로 풀 수 있도록 한다. 전송 시 암호화 과정에서 align을 해야 한다.

암호화는 application thread에서 이루어진다. 복호화는 io 쓰레드에서 진행한다.

각 프로토콜의 send 함수에서 지정된 암호화 클래스를 사용하여 암호화 하고 on-recv 에서 packetization 과정에서 복화화를 진행한다.