|  |  |
| --- | --- |
| **[ Catch Bear (캐치 베어) ]** | |
| 3주 | 2022. 1. 9 ~ 2022. 1. 15 |
| 1.9 Lock-Based Stack 을 공부하며 Stack을 사용할 때 마다 Lock을 거는 것이 아닌 사용할 때 Lock이 저절로 적용이 되는 Stack을 만들어봄. Push, TryPop, WaitPop이라는 클래스 함수를 만들어주고 value를 넣어 주면 멤버 변수인 stack컨테이너에 원자적으로 값을 넣어주는 매커니즘  1.10 Stack을 사용할 때 Lock을 사용하면 처리할 연산 량이 많아져서 속도가 느려질 수 있음. 이때 Lock을 사용하지 않고도 스레드가 다른 스레드의 영항을 받지 않고 데이터를 관리할 수 있는 Lock-Free Stack / Queue에 대해 공부함. Push를 할 때는 보통 여러 스레드가 동시에 push를 해도 문제가 없지만 pop을 할때는 delete할 때 다른 스레드와 동시에 연산을 하면 문제가 생길 수 있음. 해결 방법으로는 예전에 다이렉트 엔진 만들 때 RefernceCount로 장치들을 관리하던 것처럼 popCount를 만들어 현재 pop을 실행중인 쓰레드 개수를 세어서 하나의 스레드만 pop을 할 수 있도록 만들어줌.  1.11 이제부터 ThreadManager를 이용하여 쓰레드를 관리함. 쓰레드 매니저를 만드는 이유는 TLS를 만들고 날리는 부분을 만들 수 있다. 아직까지는 TLS가 정확히 뭔지는 잘 모르겠다. 이를 위해서 ServerCore에 여러 함수를 추가하였다. 쓰레드를 만들때마다 전역으로 쓰레드 아이디를 부여하여서 잘 작동하는것까지 확인함  (1.12 : TLS란 thread local stage. 쓰레드마다 있는 공간. 앞에서 공부했는데 잠깐 까먹어서 다시 공부함)  1.12 Reader-Writer Lock에 대해 공부함. 지금까지 사용하던 표준 mutex를 사용하면 재귀적으로 락을 잡을수 없다는 문제가 있다. 또한 낮은 확률로 상호배타적인 lock이 걸리지 않을 수 있어 Reader-Writer Lock을 구현해봄. 비트 연산을 통해 하위 16비트는 공유해서 사용하는 리드카운트, 상위 16비트는 락을 획득하고 있는 스레드 아이디로 설정하여 unorderedmap으로 관리하여서 현재 스레드 혼자만 있다면 write를 할 수 있게 만듬. Read는 여러 스레드가 동시에 해도 문제없음. 사실 완벽히 이해가 가지 않아서 필요시 나중에 다시 공부할예정임  또한 DeadLock을 탐지하기 위해 DeadLockProfiler를 만들어봄. lock이 걸려있는 함수를 하나의 정점으로 생각하여 Dfs를 이용한 그래프를 만들어서 데드락 상태인지 확인하는 작업. 완전히 이해는 가지 않아서 차차 다시 공부할 예정.    1.13 본격적으로 네트워크 들어가기 전에 메모리 관리 공부중. Reference Counting을 이해하기 위해 SharedPtr을 직접 구현해보고, 스마트포인터에 대해 공부함. 하지만 이 SharedPtr은 네트워크 환경에서의 데드락처럼 순환 문제가 일어날 수 있다. (서로 참조하는 경우에..?) weak\_ptr은 shared\_ptr을 보충하여 사용한다. Weak은 refcountblock을 참조하여 객체가 사라졌나 확인할 수 있지만 객체의 생명주기에는 영향을 주지 않음 (reference count를 직접 건드리지 않는다) | |

|  |  |
| --- | --- |
| **[ Catch Bear (캐치 베어) ]** | |
| 4주 | 2022. 1. 16 ~ 2022. 1. 22 |
| 1.16 Memory라는 클래스를 만들어서 메모리 생성과 소멸을 따로 관리하는 클래스를 만듬. 내가 만든 xnew같은 함수는 메모리에 공간은 만들지만 생성자를 호출하지 않으므로 placement new라는 문법을 사용해줌. 이제 할당하고 해제하는 부분을 우리가 다양한 옵션을 주어서 메모리 할당을 어떻게 할지 가로챌 수 있다.  1.17 또한 메모리 관련 이슈들(버그)를 잡는 데 유용한 기능인 Stomp Allocator에 관해 공부함. 메모리 풀 공부할땐 기존에 사용하던 new delete대신 운영체제에 직접 메모리를 요청하는 VirtualAllock, VirtualFree을 사용할건데 이걸 사용하면 날린 메모리에 값을 수정할 수 없게 된다. 물론 실행 속도는 떨어질 수 있지만 메모리 침범 이슈는 잡을 수 있음. 실행 속도가 떨어지는 문제는 VirtualAllock 특성상 페이지 단위로 메모리를 할당받기 때문인데 할당된 메모리 끝쪽에 객체를 넣게 되면 오버플로우 문제는 잡을 수 있음  1.18 Memory pool과 object pool에 대해 공부함. 자주 사용하는 오브젝트의 경우 생성했다 삭제할때 아예 날리지 말고 다음에 다시 사용할 수 있도록 하는 것. 근데 비트 연산에 익숙하지 않아서 자세한 내용은 이해가 잘 안갔음. 서버 구축하면서 필요시 다시 들을 필요가 있을것같다. Typecast에 대해 자세하게 배움. Static cast는 메모리 오염 위험이 있고 dynamic cast는 부하가 큰 단점이 있어 템플릿을 이용해 컴파일 단계에서 타입캐스트를 도와주는 함수를 만들어 보았다. | |