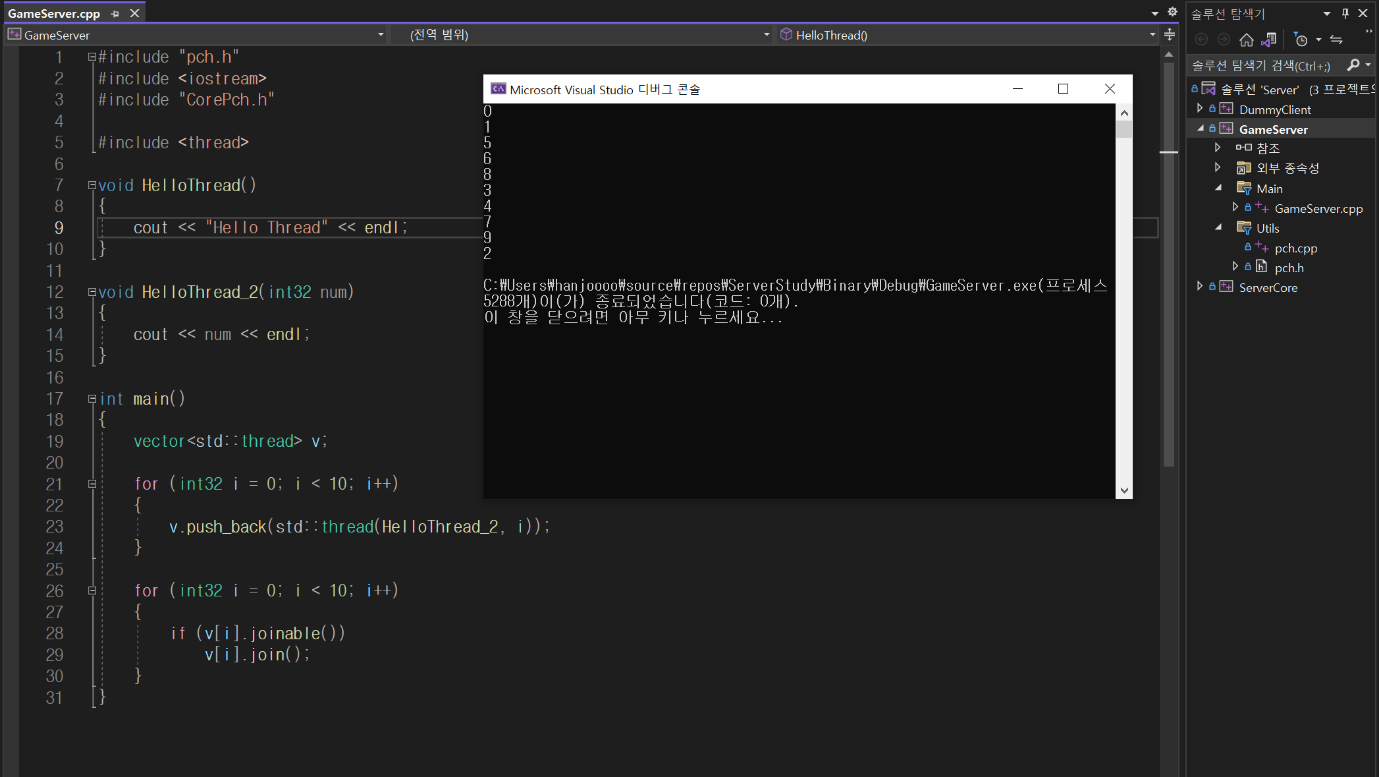
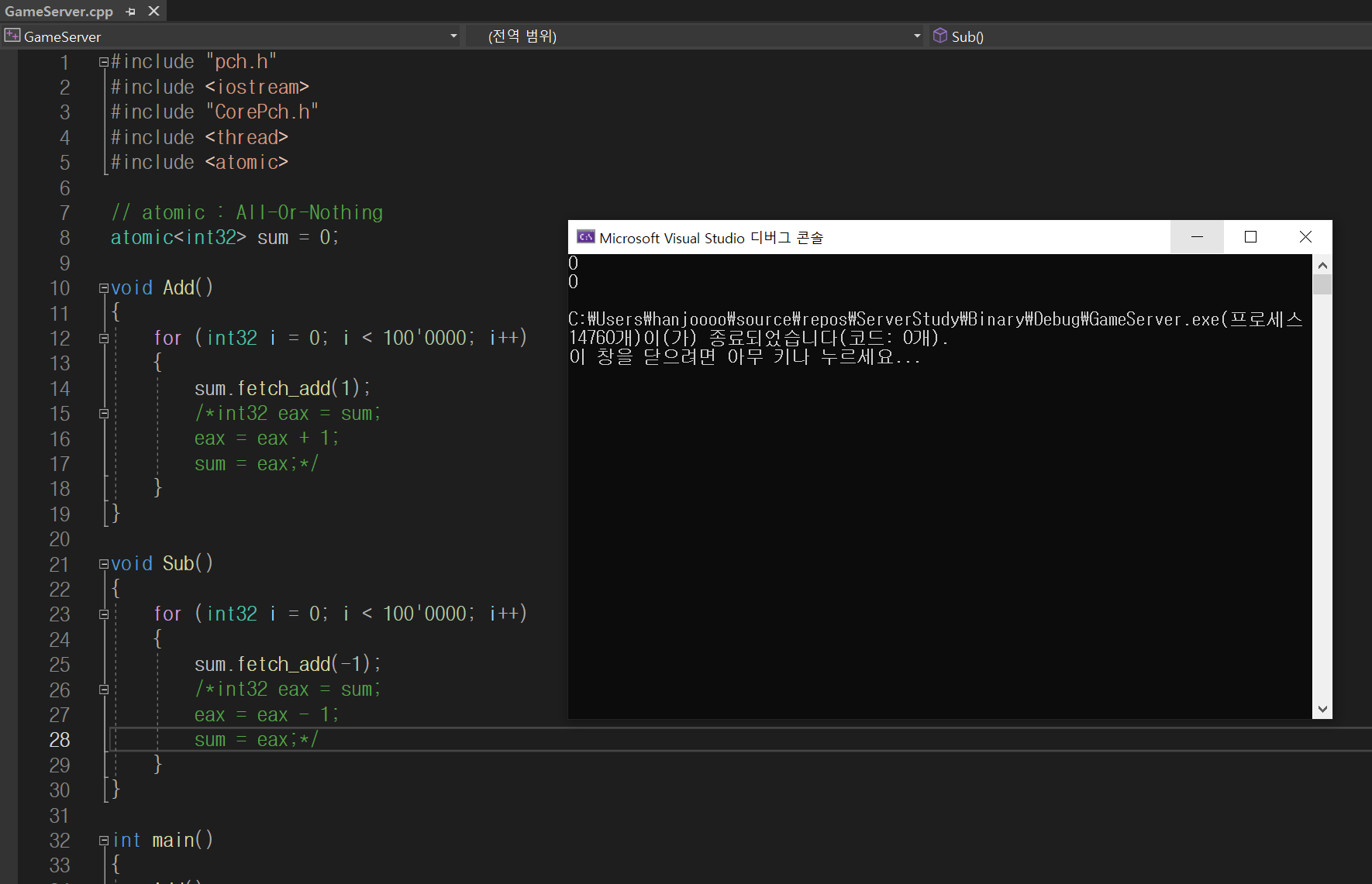
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 3주차 | **기간** | 2023.7.10~ 2023.7.16 | **지도교수** | (서명) |
| 이번주 한일 요약 | - 게임 서버 공부(멀티스레드 프로그래밍) | | | | |

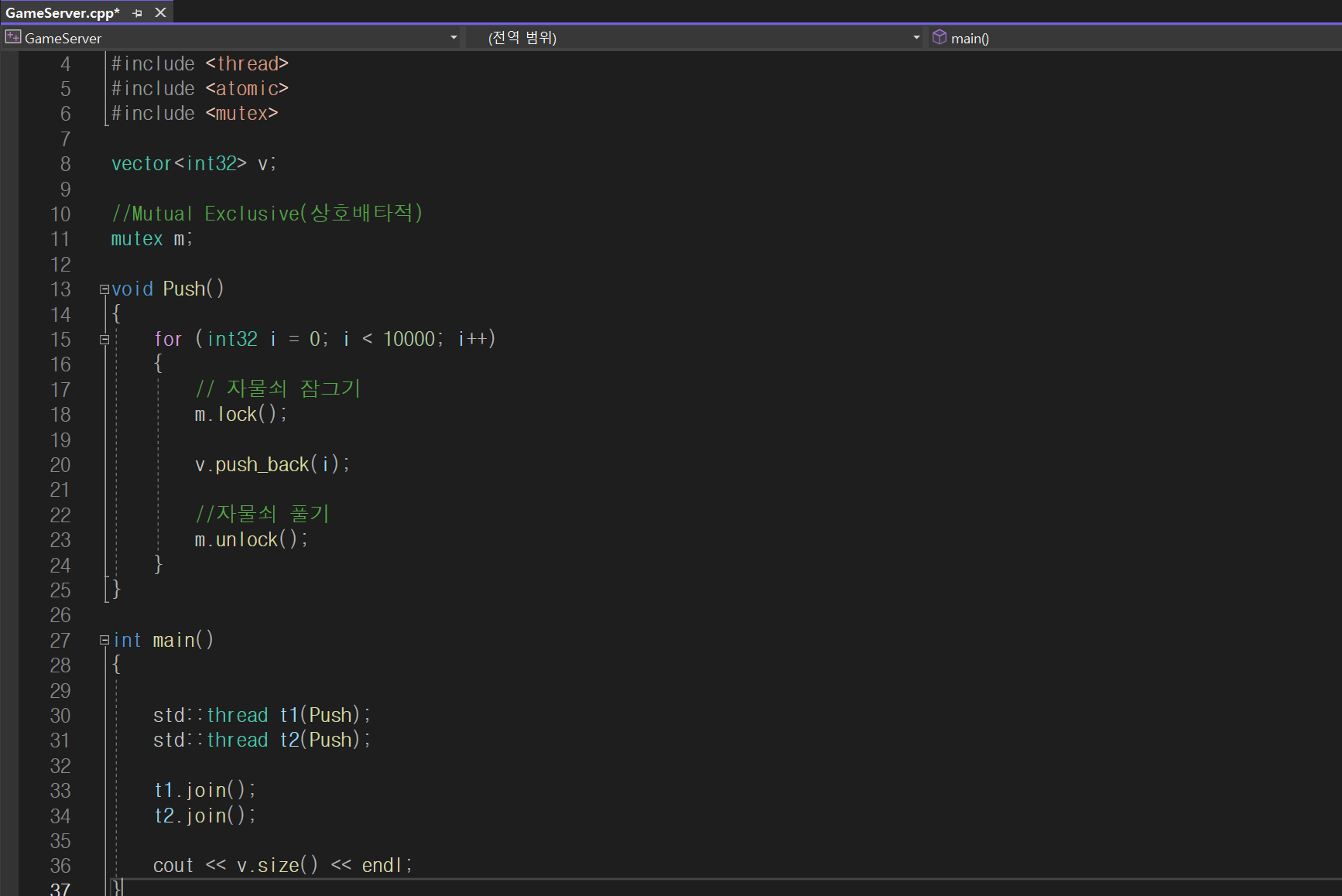
<상세 수행내용>



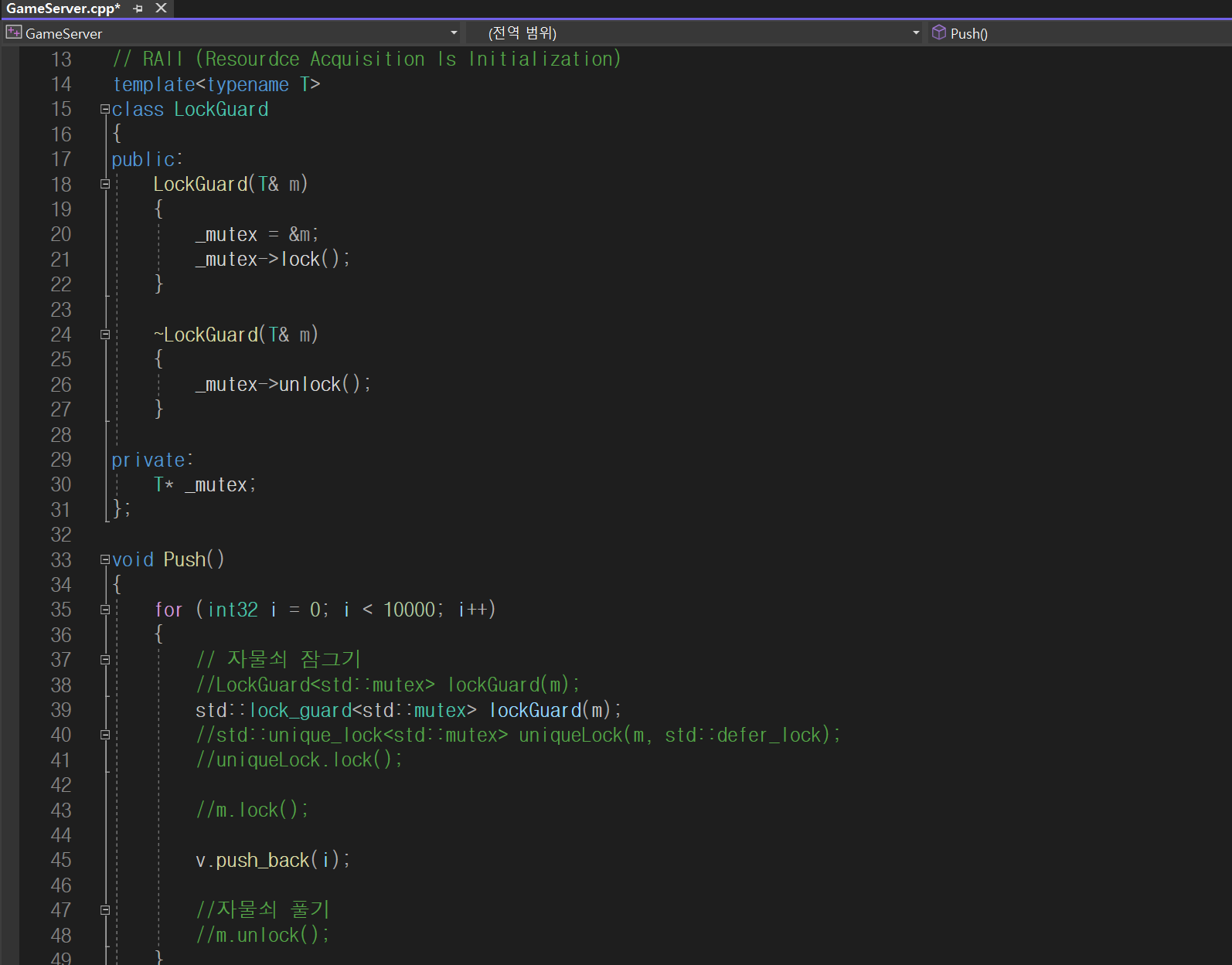
기본적인 스레드의 생성방법에 대해서 먼저 공부하였다. 스레드(thread)는 프로세스 내에서 실행되는 독립적인 실행 흐름이다. 스레드는 프로세스 내에서 코드 실행의 단위이며, 동시에 여러 작업을 수행하거나 병렬로 작업을 처리하는 데 사용된다. 스레드를 사용하면 프로그램의 응답성과 성능을 향상시킬 수 있다.



다음으로는 다중 스레드 환경에서 동기화 문제를 해결하기 위해 사용되는 atomic에 대해서 배웠다. 원래 sum = sum++; 와 같은 연산은 sum의 데이터를 읽어오고, 1을 더하고, sum을 수정하는 3개의 단계를 거치는데, 이경우에는 sum을 다른 스레드도 함께 수정한다면 문제가 생길 수 있다. 서로 다른 스레드에 의해 1을 더하는 함수와 1을 빼는 함수가 거의 동시에 실행된다고 하면, sum의 값을 불러올 때는 둘 다 0을 불러와 계산을 하고, 두 스레드 중 조금이라도 빨리 계산이 끝난 값이 먼저 쓰여진 다음 두번째로 계산이 끝난 값이 덮어 씌워져 문제가 생길 것이다. atomic을 쓰면 해당 연산이 한 번에 완전히 실행되어 다른 스레드의 간섭을 받지 않는다. 즉, atomic은 어떤 연산이 다른 스레드에 의해 동시에 수정되지 않음을 보장하고 이를 통해 데이터 경쟁이나 비일관성 문제를 방지할 수 있다.



또 다른 동기화 문제를 해결하기 위한 방법인 lock에 대해서도 학습하였다. Mutex는 Mutual Exclusion(상호 배제)의 약어로 여러 스레드가 공유된 자원에 동시에 접근하는 것을 방지하기 위해 사용된다. Lock과 unlock을 이용해 해당 작업이 끝날 때까지 다른 스레드가 간섭하지 못하게 할 수 있다.



코드가 복잡해질 경우 lock과 unlock을 해주는 게 번거로워질 수 있기 때문에, lock\_guard를 사용한다. Lock\_guard는 잠금과 해제를 자동으로 처리해주므로 이를 통해 예외 처리 등에서도 잠금을 안전하게 해제할 수 있으며, 코드의 일관성과 안정성을 향상시킨다.

DeadLock을 유의하자. 데드락(Deadlock)은 멀티스레드 또는 다중 프로세스 환경에서 발생하는 동기화 문제로, 두 개 이상의 작업이 서로 상대방이 가지고 있는 자원을 기다리며 무한히 대기하는 상황을 말한다. 데드락을 완전히 방지할 수 있는 만능의 방법은 없으나, lock을 획득하는 순서에 유의하도록 하자. 2개의 lock을 모두 획득해야 하는 상황이 있다고 할 때, 1번 프로세스는 1번 lock을 먼저 획득하고, 2번 프로세스는 2번 lock을 먼저 획득했다면, 두 프로세스 모두 교착상태에 빠지게 된다.

Lock 구현 이론

1. 무작정 앞에서 기다린다 (SpinLock)
2. 일단 자리로 돌아간 뒤 나중에 다시 시도 (Sleep)
3. 알려줄 직원을 만들어서 비어 있을 때 알려달라고 하기 (Event)

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** | 4주차 | **다음기간** | 2023.7.17 ~ 2023.7.23 |
| **다음주 할일** | - 게임 서버 공부(메모리 관리) + (멀티스레드 프로그래밍)  - 언리얼 소스코드 분석(Actor) | | |
| **지도 교수**  **Comment** |  | | |