|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [별지 5호] | | | | | | | |  | |  | | |  |
|  | | | | | | | |  | |  | | |  |
| **캡스톤디자인(Capstone Design) 결과 보고서** | | | | | | | | | | | | | |
| **1. 과제 현황** | | | | | | | | | | | | | |
| 과 제 명 | | 국문 | | MMORPG를 위한 IOCP서버 제작 및 부하분산 | | | | | | | | | |
| 영문 | | IOCP Game Server Delvelopment and LoadBalance | | | | | | | | | |
| 팀 명 | |  | | | | | | | | | | | |
| 연도-학기 | | 2019 - 1 | | | 교과목명 | | 캡스톤 디자인 | | | 학부(과) | 게임 소프트웨어 | | |
| 지도교수 | | 소속 | | | 게임 소프트웨어 | | | | | 성 명 | 김혜영 | | |
| **2. 참여 학생 현황** | | | | | | | | | | | | | |
| 구분 | 학부(과) | | | | 학번 | 이름 | | | 휴대전화 | | | E -Mail | |
| **대 표** | 게임소프트 웨어 | | | | B377004 | 김기남 | | | 01042023400 | | | Kaiose30@gmail.com | |
| 팀 원 |  | | | |  |  | | |  | | |  | |
|  | | | |  |  | | |  | | |  | |
|  | | | | | | | | | | | | |
| **3. 과제 결과** | | | | | | | | | | | | | |
| 작품 유형 | | | □ 시작품 | | | | | | | | | | |
| □ 아이디어 □ 소프트웨어 □ 논문 | | | | | | | | | | |
| 주요 성과 | | | *특허등록 및 출원 /시작품제작 등 포함하여 작성* | | | | | | | | | | |
| 과제 결과 및 개선방안 | | | IOCP를 통한 서버 라이브러리 작성과 대량의 접속 및 패킷처리에 대한 안정성을 확인 한 뒤 분산 서버를 구현하고 분산이 되는 결과를 시각화를 통해 나타냄 | | | | | | | | | | |

| **캡스톤디자인 결과 보고서** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **작품명** | | 서버 구축 및 서버간 방 전달을 통한 트래픽 분산 | | | | |
| <https://plot.ly/~kaiose/5/traffic-level/#/>  서버가 진행되는 동안의 트래픽 송수신량을 실시간으로 체크한 것으로 위의 링크에서 가장 최근의 작업 수행결과의 그래프를 볼 수 있다. | | | | | | |
| **1. 작품 개요** | | | | | | |
| *(캡스톤 디자인 과제의 개요)*  **필요성**  MMORPG와 같은 온라인 게임을 서비스하기 위해서는 비연결지향형(Connectionless) 서버가 아닌 연결지향형(Connection Oriented)의 TCP서버가 유리하다. 또한 TCP 서버라 할지라도 하드웨어의 성능이 게임 내의 트래픽발생을 처리할 정도의 수준이 아니라면 TCP 서버또한 부하가 발생하여 게임내 반응성을 낮추거나 오류가 발생할 우려가 생긴다. 때문에 TCP 서버를 구축하고 게임 내 발생하는 트래픽들을 서버들 간 분산하는 기법이 필요하다.  **개요**  온라인 게임에서도 짧고 많은 패킷들이 발생하는 MMORPG와 같은 경우의 게임에서는 반응성을 중시하며 많은 데이터들이 송수신되기 때문에 비연결지향형(Connectionless) 서버가 아닌 TCP 서버를 사용한다. WINDOW 기반 IOCP 모델을 사용하여 서버를 구축하고 단일서버에 대한 안정성, 신뢰성을 보였다. 그 후 기능분산과 트래픽 분산을 위해 서버들을 나누고 로드밸런서가 아닌 서버내 에이전트로부터 분산이 시행된 모습을 시각화라이브러리를 통해 보였다. | | | | | | |
| **2. 작품의 이론, 기술, 디자인(그래픽) 현황** | | | | | | |
| *(캡스톤 디자인 과제의 이론적․기술적 근거)*  **서버 라이브러리 제작**  1. 소켓을 풀링하며 Accept 및 Connect, Disconnect , 비동기 요청을 처리하는 Server 클래스 제작  2. Server 클래스로부터 Packet이 들어가 있는 Package를 받아와서 연산작업 및 후속 처리작업을 하는 ContentsProcess 클래스 제작  (IOCP에서 큰 연산 및 처리작업을 하지 않기 위함)  3. friend를 이용한 template singletion 제작  4. 클라이언트와의 통신을 담당하는 Session 클래스 제작  5. 다른 서버간 통신을 위해 Session을 상속받아 Terminal 클래스 제작  6. Terminal 클래스들을 관리하기 위한 TerminalManager 클래스 제작  7. Session 클래스와 더불어 게임에 필요한 정보를 저장하기 위한 User클래스 제작  8. User를 관리하기 위한 UserManager 클래스 제작  9. 게임 내 Room에 대한 정보를 저장하기 위한 Room 클래스 제작  10. Room을 관리하기 위한 RoomManager 클래스 제작  11. 비동기 요청을 위해 OVERLAPPED를 상속받는 IO\_DATA 클래스 제작  12. 각 작업을 동기화하여 수행하기 위한 LOCK 클래스 제작  13. 로깅을 위한 Logger클래스 제작  14. 시간을 가져오기 위한 Clock클래스 제작  15. Database와 연동하기 위한 Database 클래스 제작  16. Packet들의 정보를 담고 있는 Packet 클래스 제작  17. 성능 테스트를 위해 system clock 을 가져오는 Profile 클래스 제작  18. Cpu점유율 또는 memory 사용량을 확인 할 수 있는 Work 클래스 제작  19. 패킷송수신량을 통해 로드발생의 유무를 체크하는 Monitor클래스 제작  **서버 사용 기법**   1. Mysql 데이터베이스를 통한 계정관리, agent를 제작한뒤 ODBC를 통해 mysql과 연동하였음 2. 로깅을 위해 비동기 읽기 쓰기를 사용하는 클래스를 제작함 3. 효율적인 객체생성 및 관리를 위해 singleton, pactory, observer 등 디자인패턴을 사용 4. 대용량 요청처리를 위해 IOCP를 사용함 5. 대량의 유저에 대한 접속처리를 효율적으로 사용하기 위해 AcceptEx함수와 소켓을 pooling함 6. Pooling 에 관해 Session을 위한 Socket만 pooling 하는 것이 아닌 session과 두 개의 OV구조체 까지 Pooling 함으로써 접속대기 시간을 줄임 7. 비동기요청을 처리하는 GQCS와 packet을 처리하는 ContentsProcess간에 단일 Queue를 통해 통신하게 되는데 Push 및 Pop간 동기화기법 사용 시 나타나는 병목현상을 병렬 Queue를 사용함으로 완화시킴 8. 동일한 저장소에 대해 접근하고 수정하는 스레드들의 동기화를 최대한 줄여 bottleneck을 없애기 위해 SRWLock 사용 9. PacketQueue에 대해 여러 스레드가 queue를 확인할 때 queue가 비어있다면 자신에게 할당되어있는 시간을 포기할 수 있도록 슬립 또는 yield를 사용함 10. XML파서 라이브러리를 통해 서버를 실행함에 있어서 유동적으로 서버 자원을 변경 할 수 있도록 하였음. 11. 파이썬 라이브러리인 Plotly를 사용하여 서버들간 패킷 송수신상태를 시각화 할 수 있도록 하였음. 12. 유저 및 스레드의 동기화를 위해 mutex , critical\_section, srwlock 등을 사용함. 13. SRWLOCK 을 lock\_guard 와 같이 해제시 따로 선언하지 않게 함으로써 코드의 복잡성을 줄임 14. 패킷 구조는 [길이][패킷번호][데이터] 로 이루어져 있으며 모든 패킷을 오버라이딩을 통해 처리함. 15. 패킷들을 직렬화 하여 보냄으로써 간편성을 확보, 가변길이 패킷을 통해 네트워크 리소스를 줄였다. 16. 한 번에 여러 바이트스트림이 잇다라 들어오는 상황에 대해서 처리함(네이글옵션을 끈 것이 아님) 17. 클라이언트의 비정상 접속 종료시 발생하는 ERROR\_NETNAME\_DELETED, WSAOPERATION\_ABORTED 에러에 대해 예외처리함. 18. 서버간의 유저의 데이터가 이동할 때는 먼저 유저의 데이터를 타 서버에 넘기고 난후 해당 유저에게 이동할 서버를 알림 19. 서버간의 map이동은 map안의 모든 유저들의 이동이다.       AccpetEx의 Pooling과 NoPooling의 비교  소켓을 비동기 Accpet하는 방식에서 Pooling과 Nopooling을 비교했다.  10000개의 세션에 대해 Nopooling시 약 8500milliseconds 가 pooling시 500milliseconds가 걸리는 것으로 보아 Pooling 시의 성능이 훨씬 우수함을 알 수 있다. 또한 pooling 객체 수에 대해 비교했던 자료에서는 100개 이상의 pooling 만 된다면 그 이후부터는 비슷한 성능을 띄었다.  **클라이언트 디자인**   |  |  |  | | --- | --- | --- | | **1** | **2** | **3** | | **4** | **5** | **6** | | **7** | **8** | **9** |     Figure 1. 클라이언트 Figure 2. 맵의 구조    **Figure 3. 로비**  전체적인 맵의 형태는 3\*3 의 정사각형들로 이루어져 있으며 각 맵에는 번호와 트래픽 발생량을 저장하고 있다. 초기에는 각각의 서버가 1~3, 4~6, 7~9 의 맵을 소유하고 있으며 유저들은 서버에 관계없이 인근의 맵으로 이동이 가능하다. 또한 로비에서도 서버와 관계없이 방에 입장이 가능하다.  **서버 성능 테스트**  **테스트 하드웨어 환경**   |  |  | | --- | --- | | CPU | I-5 | | 메모리 | 8GB | | OS | Win 10 | | GPU | GTX960 |   에코에서 한 개의 게임서버에 대해 무작위로 움직이는 더미클라이언트들을 연결하여 다음과 같은 목록에 대해 테스트함.   1. 초당 세션 접속 수 2. 100, 400, 800개의 클라이언트에 대한 처리량 3. 현 하드웨어의 IOCP가 처리할 수 있는 최대 Byte량   **- 초 당 세션접속 수**  에코환경에서 접속만을 시도하는 더미클라이언트 10000개에 대해 대략 0.5초의 시간이 소요됨.  이를 통해 초당 20000개까지의 세션에 대한 접속처리가 가능할 것으로 보임.    **- 100, 400, 800개의 클라이언트에 대한 처리량**  더미클라이언트는 접속과 무작위 움직임에 대한 패킷만을 보낸다. ( 3~5초 간격)  서버에서는 더미클라이언트에게 응답패킷을 보내지 않는다.  아래의 사진들은 각각 100, 400, 800개의 더미클라이언트들을 하나의 서버에 접속시키고 Move패킷만을 전송하는 사진들이며 사진 내에서  Room Number : Room번호  Sending Packet Count : Server가 송신하는 패킷의 총 수  Sending Packet Quantity : Server가 송신하는 패킷의 총 Byte량  Receive Packet Count : Server가 수신 받는 패킷의 총 수  Receive Packet Quantity : Server가 수신 받는 패킷의 총 Byte량  Total Packet Quantity : Sending Packet Quantity + Receiving Packet Quantity 이다.  송수신 Packet에 대한 정보는 25초마다 수집한 것으로  -200명의 클라이언트가 3개의 방에 할당되어 Room당 8000Bytes의 처리량이 발생하며  3개의 Room에 24000Bytes, 즉 대략 초당 950Byte를 처리한다.  -400명의 클라이언트가 3개의 방에 할당되어 Room당 14000Bytes의 처리량이 발생하며  3개의 Room에 42000Bytes, 즉 대략 초당 1700Byte를 처리한다.  C:\Users\kaios\AppData\Local\Microsoft\Windows\INetCache\Content.Word\400 client2.png  -800명의 클라이언트가 3개의 방에 할당되어 Room당 30000Bytes의 처리량이 발생하며  3개의 Room에 90000Bytes, 즉 대략 초당 3600Byte를 처리한다.  C:\Users\kaios\AppData\Local\Microsoft\Windows\INetCache\Content.Word\800Client.png  (각 사진에서 Room 2 의 Packet 송수신 량이 많은 이유는 더미들의 움직임을 2D 그래픽으로 보기 위해 패킷을 수신 받는 클라이언트가 Room2에 접속해 있기 때문이다.)  **- 현 하드웨어의 IOCP가 처리할 수 있는 최대 Byte량**  10000개의 세션에 대해 while 반복문을 통해 에코서버에 딜레이없이 패킷을 보내고 IOCP에서 받는 바이트 량에 대해 프로파일링 한 것이다.  처리량이 일정하지는 않으나 최소 170000Byte 이상은 처리하는 것을 볼 수 있다.  C:\Users\kaios\AppData\Local\Microsoft\Windows\INetCache\Content.Word\제목 없음.png  현재 세션의 접속 수가 10000명이지만 이는 더미세션이므로 실 게임에서는 더욱 적은 유저의 수에 대해 응답을 하게 될 것이다. 때문에 단일 서버에 대한 부담을 덜기 위해 부하분산을 구현하였다. | | | | | | |
| **3. 작품의 개발 방법 및 과정** | | | | | | |
| 1. OS에 따라 TCP 서버의 모델이 다르므로 OS를 선택하고 OS가 제공하는 서버모델중 효율이 좋은 모델을 선택한다.  2. IOCP모델을 중심으로 Server 클래스를 만들고 통신을 담당하는 Session 클래스를 구성하여 기본적인 뼈대를 만든다.  3. 클라이언트와 서버간 통신을 위해 사용할 패킷 클래스를 제작한다.  4. 가변길이 패킷을 위해 Serialize를 하는 Stream 클래스를 만든다.  5. IOCP로직에서 큰작업을 처리 하지 않도록 하기위한 PacketProcess 클래스를 만든다.  6. 게임의 로그인나 서버전환, 동기화 같은 기능들을 위한 부수적인 클래스들을 만든다.  7. 유니티클라이언트를 만든 후 유니티클라이언트와 서버간의 통신을 위한 C#네트워크 클래스들을 만든다.  8. 게임로직에서의 발생하는 패킷들을 만든다.  9. 클라이언트에 대한 서버처리의 비정상 종료, 오류를 디버깅한다.  10. 다수 클라이언트들을 발생시킬 더미프로젝트를 만든다.  11. 다수 클라이언트에 대한 서버의 안정성을 확인한다.  12. 서버라이브러리를 DLL 파일로 제작하여 로그인, 로비, 게임과 같은 기능분산을 시행한 알맞은 형태로 고친다.  13. 서버간의 터미널 통신을 위한 기능을 제작한다.  14. 트래픽분산기능을 위해 게임서버를 분화시키고 기존 단일서버가 가지고 있던 맵들을 분산시켜 소유하도록 만든 뒤 유저가 서버간 이동할 수 있는 기능을 만들고 안정성을 확인한다.  15. 서버간 소유하고 있는 맵과 맵내부 유저들에 대한 정보를 이동시킬 수 있는 기능을 구성한다. | | | | | | |
| 1. C:\Users\kaios\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ClassDiagramAdvanced.jpg**작품 구조도**   Figure 2 ClassDiagram | | | | | | |
| Figure 3. 로그인 SequenceDiagram  위 그림은 유저가 로그인 버튼을 클릭시 LogInServer 서버에서 Database에 쿼리후 그 응답을 통해 유저가 게임서버에 접속할 수 있도록 해주는 로그인 과정에 대한 시퀀스 다이어그램이다.    Figure 4. 유저의 Map이동간 SequenceDiagram  Figure4는 유저가 다른 맵으로 이동을 요청했을 때 일어나는 시퀀스다이어그램으로 현재 위치한 서버에서 요청한 맵의 존재유무를 파악한 뒤 현 서버가 요청맵을 가지고 있지 않다면 맵을 소유한 서버에게 유저의 데이터를 넘겨주는 시퀀스 다이어그램이다.    Figure 5. 부하분산시 패킷시퀀스 다이어그램  위의 그림은 Server1에서 상대적으로 많은 부하량이 발생시 타 서버의 부하량을 체크하고 가장낮은 부하량을 가진 서버2에게 맵을 전이하는 과정이다.  Figure 5. 부하분산시 서버의 상태변화 및 과정  Figure5에서 보이듯이 서버는 각각 SENDER, RECEIVER, VLOCK 상태를 가질 수 있으며 송신자가 RECEIVER 상태를 가진다. 송수신이 시작되면 서버는 VLOCK 상태로 전이하며 전달할 맵과 내부 유저들의 데이터를 미리 옮길 서버에 전송하는 방식을 통해 부하분산이 일어날 시 유저의 딜레이를 줄일 수 있도록 노력했다.    위의 결과보고서 사진은 두 개의 서버의 로드한계치를 13000Byte로 설정해 놓은 뒤 서버1에  100대의 클라이언트를 할당한뒤 일어나는 분산을 보인 것이다.  <https://play-tv.kakao.com/embed/player/cliplink/396363285?service=daum_tistory>의 영상은  과정 1) 더미클라이언트 100대를 서버 1에 룸1:33, 룸2:34 룸3:33 으로 분산함.  과정 2) 서버1에서 로드가 발생하여 패킷송수신량이 가장 많은 룸2를 서버2에 할당함  과정 3) 서버1에서 로드가 발생하여 패킷송수신량이 가장 많은 룸1을 서버2에 할당함.  과정 4, 5) 서버 1에서 로드가 발생하여 패킷송수신량이 가장 많은 룸3을 서버2에 할당함,  서버2에서 로드가 발생하여 패킷송수신량이 가장 많은 룸2를 서버1에 할당함.  최종적으로 서버1 : 룸2 서버2 : 룸1,3 을 소유하게되는 과정을 영상으로 보인 것이다  위 그래프를 통해 100개의 클라이언트에 대해 로드밸런스가 일어난 것을 시각화했다. 이를 통해 특정서버의 트래픽이 감소하여 3대의 서버가 균등한 트래픽을 가져감을 알 수 있다. | | | | | | |
| **5. 기대 효과 및 활용 방안** | | | | | | |
| *(캡스톤 디자인 과제의 시장성 및 기대 효과)*   1. 라이브러리 제작을 통해 목적에 따른 서버들을 제작할 수 있다. 2. IOCP 를 통해 제작한 서버라이브러리들에 대한 벤치마킹책으로 온라인 게임서버 프로그래밍 벤치마크가 있는데 시간이 꽤 지난 이후라 하드웨어의 차이가 심했다. 이를 새로운 하드웨어 환경에서의 테스트를 통해 내 서버에 대한 성능을 평가할 수 있다. 3. 서버라이브러리 제작을 통해 목적에 따른 서버들을 제작할 수 있다. 4. IOCP 를 통해 제작한 서버라이브러리들에 대한 벤치마킹책으로 온라인 게임서버 프로그래밍 벤치마크가 있는데 시간이 꽤 지난 이후라 하드웨어의 차이가 심했다. 이를 새로운 하드웨어 환경에서의 테스트를 통해 내 서버에 대한 성능을 평가할 수 있다. 5. gibhub소스 서버들과 다르게 바이트스트림이 겹쳐오는 처리를 함으로서 클라이언트간 통신처리가 원활하다. 6. 스트레스테스트를 통해 서버의 안정성을 확인하여 대용량 송수신이 가능함을 보였다. 7. 부하분산시 복잡한 연산이 필요없을 때 로드밸런서가 없이 서버들을 수평적으로 이은 후 agent를 통해 분산이 가능한 것을 보였다.   **결론**  IOCP서버 모델의 과정 및 성능을 보이고 부하분산에 대한 결과를 시각화 하였다. | | | | | | |
| **6. 역할 분담표** | | | | | | |
| **연번** | **학부(과)** | | **학번** | **성명** | **역할** | **참여도(%)** |
| 1 | 게임소프트웨어 | | B377004 | 김기남 | 제작 | 100 |
| 20 . . .     |  |  |  | | --- | --- | --- | | 대표학생 |  | (인) | |  |  |  | | 지도교수 |  | (인) | | | | | | | |