Test

MM4220 게임서버 프로그래밍

정내훈

Test

- Test의 필요성
  - Bug없는 프로그램을 만들기 위해서
- MMORPG 게임 고유의 Test
  - 부하테스트 (Stress Test)
  - 성능 측정 (동접, HotSpot)

Test

- Test의 단계
  - 작성자 테스트, QA 테스트
  - 알파 테스트
    - 비 관계자를 동원한 테스트
    - 같은 회사의 다른 팀원, 개발자의 가족/친구
  - 클로즈드 베타 테스트
    - 불특정 다수 동원, 실제 부하 테스트의 시작
    - 모집광고 -> 인원선정 -> 클라이언트 배포 -> 테스트
  - 오픈 베타 테스트
    - 모든 사람에게 공개해서 테스트
    - 오픈 베타 테스트 중 이상이 없으면 그대로 상용화

Test (2024-화금)

- Stress Test
  - 많은 인원이 동시에 접속하여 Test
  - 적은 인원 Test시에는 보이지 않았던 bug들을 발견할 수 있다.
    - 특히 멀티쓰레드 버그
  - 서버 프로그램의 bottle-neck을 발견할 수 있다.
    - DB
    - Network Overhead
    - Memory Allocation Overhead
    - NPC AI Overhead

2024년 중간고사 일정

- 화금반
  - 4월 23일 화요일
- 화수반
  - 4월 24일 수요일

Test

- Stress Test의 문제
  - 많은 인원의 동원이 쉽지 않다.
    - 잦은 클베?
    - 동접 5000 서버의 테스트???
  - 원하는 시간의 테스트가 힘들다
  - 통제가 힘들다
    - 원하는 시나리오
  - 해결책
    - 자동 테스트 프로그램

Test

- 자동 테스트 프로그램
  - Dummy Client라고도 불림, 일종의 AUTO
  - No Visual (부하를 줄이기 위해)
  - 하나의 프로그램에서 여러 개의 character로 동시 접속
    - 테스트 비용 감소 (PC 대수 감소, 동시 실행 클라이언트 수 감소)
    - 보통 수백 개의 접속
    - ID를 미리 만들어 놓아야 함
  - 아바타 행동 모방 프로그램을 이용한 테스트
    - 이동
    - 전투

Test

- 자동 테스트 프로그램의 활용
  - PC 여러 대를 사용해 테스트
    - 500 접속 Dummy Client * 10대 PC => 5000동접
- 자동 테스트 프로그램의 한계
  - AI의 한계
    - 복잡한 작업을 수행하기 힘들다
      - 예) 퀘스트, 상거래
    - 실제 인간의 행동과는 차이가 날 수 밖에 없다
      - 실제 서비스에서 나타날 모든 버그를 잡지는 못한다

Test

- 자동 테스트 프로그램 작성시 고려 사항
  - 캐릭터 생성도 자동으로 하게 해 놓으면 편하다.
  - 지형을 입력해서 빈번한 잘못된 이동을 하지 않도록 하는 것이 좋다.
  - 테스트 캐릭터의 공간적 분포를 고려해야 한다
    - Teleport를 통한 분산을 위해 관리자 레벨의 ID발급 필요
  - 서버프로그램의 업그레이드 시 같이 업그레이드 해주어야 한다.
    - 아니면 필요할 때 쓰기가 힘들어진다.

Test

- 자동 테스트 프로그램의 구조
  - IOCP를 사용한 다중 접속
  - FSM을 사용한 접속 및 Test
    - Login id 별로 state가 있고, 서버에서 오는 packet의 종류에 따라 state가 변화 하면서 action을 취한다.
  - 대부분의 서버 packet 무시
    - Status 변경, 다른 캐릭터 이동
  - 간단한 Graphic으로 화면상에 전체 test character들의 분포와 state를 볼 수 있게 한다.

성능 측정

- 서버의 성능은 어떻게 아는가?
  - 부하를 주고 서버가 견디는가 확인
  - 부하 : 동접 또는 HotSpot
  - 견디는가? : 랙이 있는가?
- 랙 측정?
  - 사람이 측정 => 부정확
  - 클라이언트에서 보낸 신호가 서버에 갔다가 되돌아 올 때까지의 시간을 측정
    - 꺼꾸로 하면 클라이언트의 랙 측정이 됨

성능 측정

- 랙 측정
  - 별도의 랙 측정 패킷을 정의해서 주고 받을 수 있음.
    - 최소 랙이 측정 됨
  - 가장 많이 사용되는 패킷을 통해 측정
    - Move Packet에 전송 시간을 추가해서 측정
    - Move Packet에 time stamp를 추가해야 함.

구현

- 사전 작업
  - 월드 확장
    - 8 X 8에서 400 X 400
    - 몇 천이 넘는 동접을 받기 위해서는 월드 크기를 키워야 함.
  - 클라이언트 확장
    - 8 X 8에서 16 X 16
    - 8 X 8은 너무 좁음

실습

- 스트레스 테스트 프로그램 다운로드
  - eClass 강의자료
  - **[실습자료] 스트레스 테스트 프로그램**

실습

- 프로토콜 확장
  - move_time은 millisecond 값을 갖는다.

```
struct sc_packet_move {
    char size;
    char type;
    int  id;
    short x, y;
    unsigned move_time;
};
```

```
struct cs_packet_move {
    char    size;
    char    type;
    char    direction;
    unsigned move_time;
};
```

실습

- 스트레스 테스트 프로그램 구조
  - IOCP 구조 : 게임 서버와 거의 같음
  - Test_Thread에서 플레이어 컨트롤
    - 단순 랜덤 이동
    - 정해진 인원수(MAX_TEST)에 맞춰 접속 추가
      - Adjust_Number_Of_Client()
  - DrawModule.cpp에서는 OpenGL을 사용해 접속한 플레이어들의 위치를 화면에 표시

실습

- 동접 컨트롤

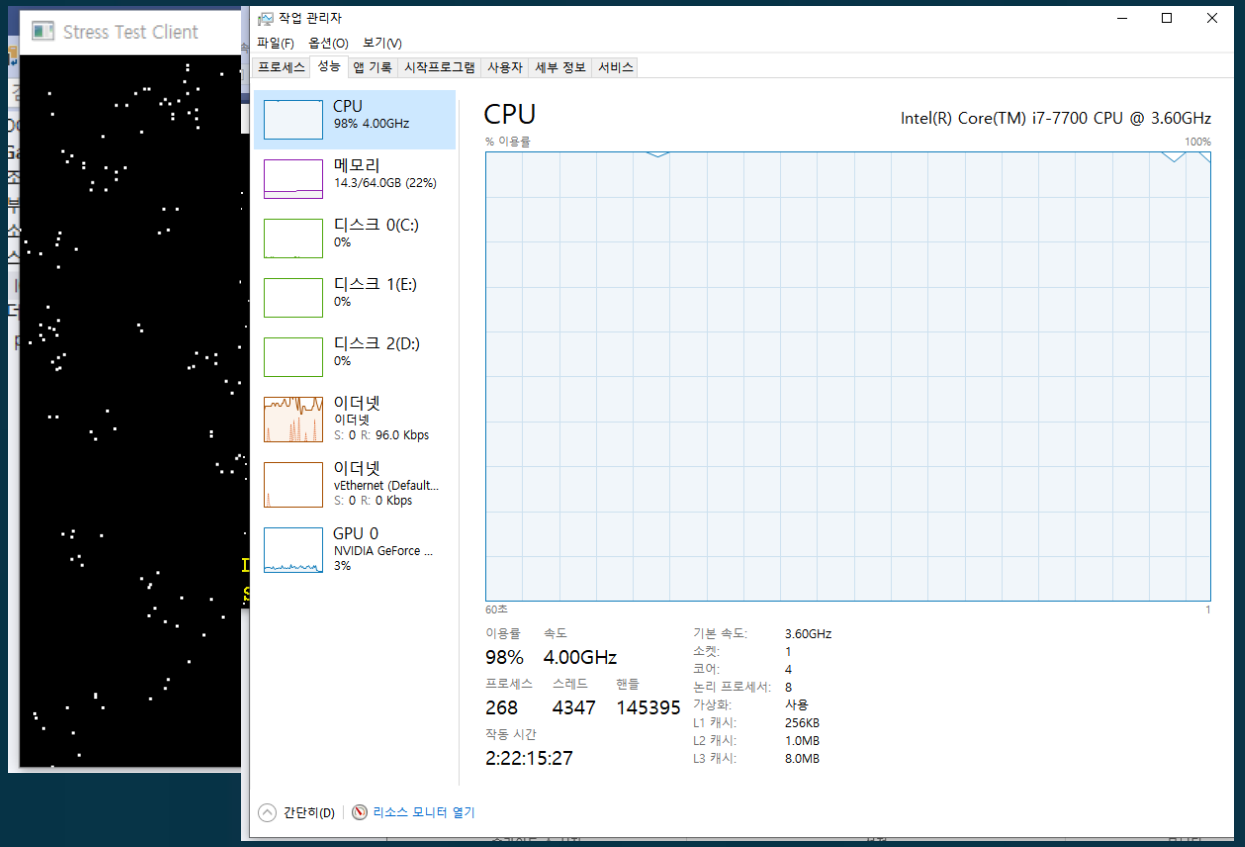
```
constexpr int DELAY_LIMIT = 100;
constexpr int DELAY_LIMIT2 = 150;
constexpr int ACCEPT_DELY = 50;

void Adjust_Number_Of_Client()
{
   static int delay_multiplier = 1;
   static int max_limit = MAXINT;
   static bool increasing = true;

   if (active_clients >= MAX_TEST) return;
   if (num_connections >= MAX_CLIENTS) return;

   auto duration = high_resolution_clock::now() - last_connect_time;
   if (ACCEPT_DELY * delay_multiplier > duration_cast<milliseconds>(duration).count()) return;

   int t_delay = global_delay;
   if (DELAY_LIMIT2 < t_delay) {
      if (true == increasing) {
         max_limit = active_clients;
         increasing = false;
      }
      if (100 > active_clients) return;
      if (ACCEPT_DELY * 10 > duration_cast<milliseconds>(duration).count()) return;
      last_connect_time = high_resolution_clock::now();
      DisconnectClient(client_to_close);
      client_to_close++;
      return;
   } else
   if (DELAY_LIMIT < t_delay) {
      delay_multiplier = 10;
      return;
   }
   if (max_limit - (max_limit / 20) < active_clients) return;

   increasing = true;
   last_connect_time = high_resolution_clock::now();
```

실습

- 테스트 결과
  - 250 정도의 최대 동접
    - 낮다!
      - STRESS 테스트를 한 컴퓨터에서 해서!
      - 노트북은 느림.
    - Debug와 Release모드의 차이는 20정도

실습

- 테스트 결과
  - CPU 사용량 100% => CPU가 bottle neck



분석

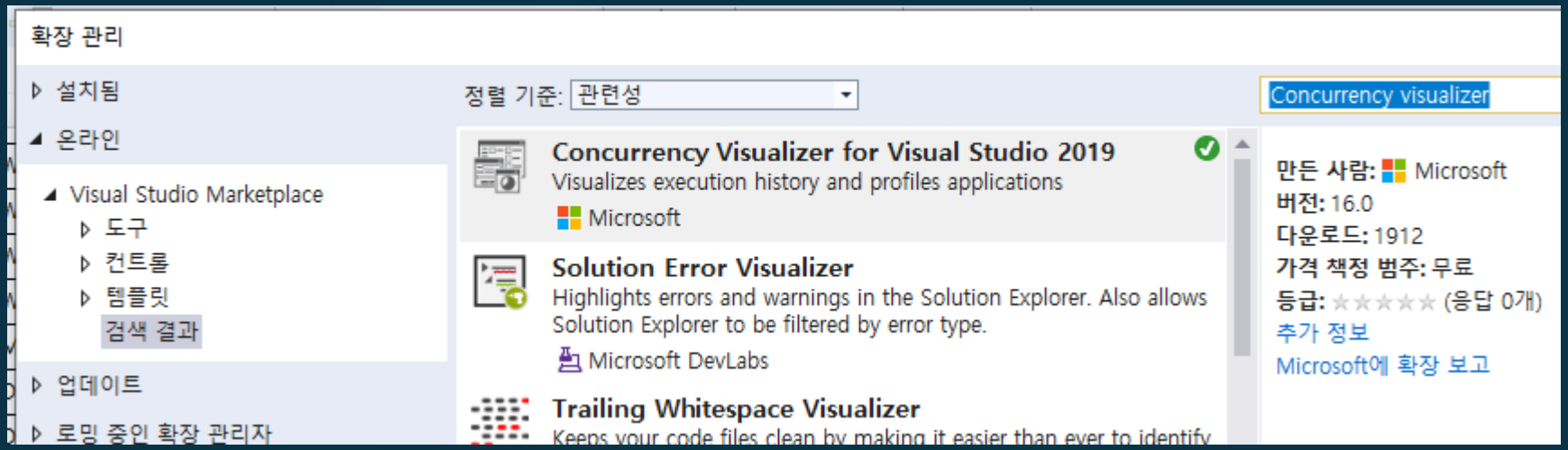
- 동접이 너무 적다.
  - 이런 간단한 서버는 적어도 5000은 나와야 한다.
  - i5 CPU 기준.
- 해결방법?
  - Bottle Neck을 찾아야 한다.
    - 이후 Bottle Neck을 제거
  - 어떻게?

분석

- Profiling을 해야 한다.
- Profiling
  - SW를 실행하고 어느 부분에서 CPU를 많이 사용하는지 계측하는 작업
    - 여러 가지 방법이 있다.
  - 보통은 Kernel에서 주기적으로 CPU의 PC레지스터값을 읽어서 어떤 코드를 실행하고 있는지 기록
    - 프로그램에 함수들의 주소가 있으므로 이를 통해 어떤 함수를 실행 중인지 판단할 수 있다.
    - 이를 통해 어떤 함수가 CPU를 많이 사용하는지 알 수 있다.
  - 지금은 CPU에 프로파일링 기능이 내장되어 있다,

분석

- Profiling 방법
- Visual Studio는 Concurrency Visualizer라는 막강한 Tool이 있다. (공짜)
  - 설치하자
    - 확장(X) -> 확장관리 -> 온라인 -> 검색 (Concurrency Visyualizer) -> 다운로드 -> Visual Studio 재시작

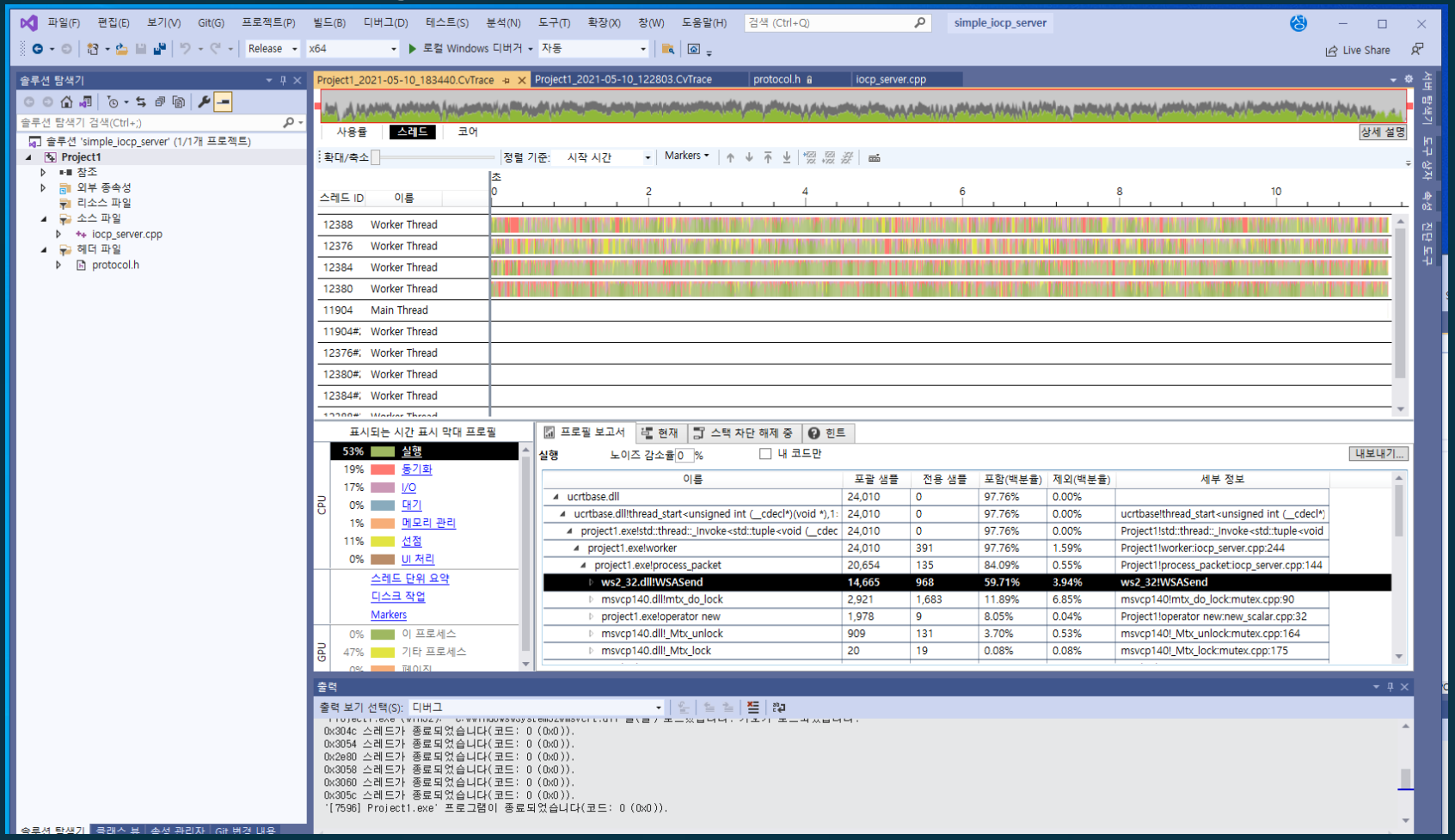


분석

- Concurrency Visualizer사용
  - 서버 실행 -> Stress Test Client 실행 -> 최대 동접이 될 때 까지 대기 -> 분석(N) -> Concurrency 시각화 (V) -> 프로세스에 연결 -> 서버 프로세스 선택 -> << Profiling 시작 >> -> 수집 중지 -> << 보고서 작성>>
  - 오래 수집할 필요는 없다. 1~2초 정도가 적당 너무 오래 수집하면 보고서 작성에 시간이 너무 오래 걸린다.

분석

- Concurrency Visualizer 실행 결과



분석

- Concurrency Visualizer 문제점
  - 2021년 5월 현재 Profiling 불안정
    - 컴퓨터를 리부트 하면 잠깐 동작하다가 불통
    - MS도 원인을 모름, 현재 사용자의 많은 불만
    - Kernel쪽 문제이고, Open Source가 아니므로 MS에게만 의지
    - Visual Studio 2022출시로 MS는 관심이 없는듯.
  - Visual Studio 2022 + Windows 11에서는 문제 없음

분석

이름	포괄 샘플	전용 샘플	포함(백분율)	제외(백분율)	세부 정보
ucrtbase.dll	24,010	0	97.76%	0.00%	
ucrtbase.dll!thread_start<unsigned int (__cdecl*)(void *),1:	24,010	0	97.76%	0.00%	ucrtbase!thread_start<unsigned int (__cdecl*)
project1.exe!std::thread::_Invoke<std::tuple<void (__cdec	24,010	0	97.76%	0.00%	Project1!std::thread::_Invoke<std::tuple<void
project1.exe!worker	24,010	391	97.76%	1.59%	Project1!worker:iocp_server.cpp:244
project1.exe!process_packet	20,654	135	84.09%	0.55%	Project1!process_packet:iocp_server.cpp:144
ws2_32.dll!WSASend	**14,665**	**968**	**59.71%**	**3.94%**	**ws2_32!WSASend**
msvcp140.dll!mtx_do_lock	2,921	1,683	11.89%	6.85%	msvcp140!mtx_do_lock:mutex.cpp:90
project1.exe!operator new	1,978	9	8.05%	0.04%	Project1!operator new:new_scalar.cpp:32
msvcp140.dll!_Mtx_unlock	909	131	3.70%	0.53%	msvcp140!_Mtx_unlock:mutex.cpp:164
msvcp140.dll!_Mtx_lock	20	19	0.08%	0.08%	msvcp140!_Mtx_lock:mutex.cpp:175

분석

- Concurrency Visualizer 실행 결과 분석
  - Do_Move()함수에서 대부분의 시간을 소비 => Bottle Neck
    - WSASend에서 CPU 소비
      - 우리가 작성한 함수가 아니므로 최적화 불가능
      - 호출 회수를 줄여야 한다.
    - mutex lock(), mutex unlock()
      - 멀티 쓰레드 프로그램에서는 역시 lock()이 문제
    - New에서 소비
      - 메모리 관리자를 직접 작성해서 넣어야 한다.
- Bottle Neck을 알았으니 해결해 보자.

최적화

- WSASend()문제 해결 방법 : 호출 회수 감소
  - 시야처리

숙제 (#4)

- 게임 서버 성능 비교 테스트
  - 내용
    - 숙제 (#3)의 프로그램의 성능 비교
      - 지금 까지 구현한 3가지 게임 서버의 성능을 비교하라
        - Overlapped I/O Call Back, Single Thread IOCP, Multi Thread IOCP
      - 다음 2가지 방법으로 구현하고 성능을 비교하라.
        - Non-Blocking I/O, Multi-Thread I/O
    - StressTest 프로그램을 사용해서 최대 동접 측정하기
  - 목적
    - StressTest 프로그램을 사용한 디버깅 및 최적화
    - I/O모델 별 최대 성능 비교
  - 제출 (EClass)
    - 클라이언트/서버/스트레스 테스트 프로그램 소스
    - 컴퓨터의 사양 제출 (CPU, 메모리), 성능 비교 표
    - 사용한 최적화 기법 설명