소개

BMM80051-01

고급게임서버프로그래밍

정내훈

2020년도 1학기

한국산업기술대학교 디지털엔터테인먼트 학과

내용

- 강좌 소개
- 주제 리스트

강사 소개 : 정내훈

- 현재 : 한국산업기술대학교 게임공학부 부교수
- 경력
  - 1990년부터 온라인게임 개발
    - LPMUD, Archmage
  - 2002년 3월 – 2008년 2월 NCSoft
    - MMORPG 개발 : Lineage forever, Alterlife, Blade & Soul
  - 2015,2016,2017 모바일 게임 서버 개발 Netmarble
    - GSF, FinalShot, Lineage2Revolution
- 전공
  - Parallel Processing
- 관심분야
  - 차세대 게임 서버 구조
- 연락처
  - nhjung@kpu.ac.kr 공학관 E동 314호

개요 – 고급 게임서버 프로그래밍

- 목적
  - 실제 상용 게임서버에서 발생하는 이슈들인, 멀티쓰레드 문제, DataBase연결 문제, 클라우드 상에서의 성능 문제, Seamless 공간분할 문제, 물리엔진 연동 문제 등에 대해 심도 있게 연구한다

개요 – 고급 게임서버 프로그래밍

- 주제
  - RIO (Registered I/O)를 사용한 최적화
  - Boost/ASIO 서버 제작
  - Flat Buffer, Protocol Buffer
  - 엘릭서를 사용한 분산 서버 구현
  - 물리엔진 연동
  - Seamless 월드 분할 구현
  - REDIS 연동

개요 – 고급 게임서버 프로그래밍

- 교재
  - 게임 서버 프로그래밍 교과서, 배현직, 길벗, 2019
    - 가장 최근에 나온 게임서버 프로그래밍 교재
    - 연구 주제 중 Seamless, REDIS에 대한 소개가 있음.
    - 게임 서버에 관심이 있으면 읽어 보는 것이 좋음.

성적 산출

- 토론: 10%
  - 수업 시간 토론 참여도
- 과제 : 50%
  - 과제물 완성도
- 발표 : 30%
  - 과제 발표 내용 평가
- 출석 : 10%
  - 결석 1번에 1% 감점, 지각 3번에 1% 감점, ¼이상 결석 F)

선수 과목

- 선수과목
  - 게임 서버 프로그래밍
    - 학부 4학년이 과목을 듣거나 교재를 이해하고 있어야 함.
  - C++ 프로그래밍 (학부)
    - 모든 예제와 실습은 **C++11**으로 이루어 짐

강의 편성

- 각 이슈에 대한 소개 이후에 과제를 부여하고 진행 과정을 발표.
- 매주 진행 상황을 발표하고 질의 응답 및 토론 진행

개발 환경

- Windows의 Visual Studio 2019에서 수행
  - C++11사용
- ASIO와 Boost는 LINUX에서 수행
  - 수업용 실습 머신을 활용

차례

- RIO (Registered I/O)를 사용한 최적화
- Boost/ASIO 서버 제작
- Flat Buffer, Protocol Buffer
- 엘릭서를 사용한 분산 서버 구현
- 물리엔진 연동
- Seamless 월드 분할 구현
- REDIS 연동

RIO

- Registered Input/Output
- Windows에서 제공하는 Network API
  - Windows 8.1과 Windows Server 2012 R2 부터 지원
  - Windows Sockets API
- 목적
  - Network Latency 감소
  - 다량의 메시지 처리 성능 향상
    - 반응시간 안정화, 처리 속도 향상
  - Hyper-V 가상머신에서의 높은 성능

History

- 태초에 Socket API가 존재
  - Bill Joy가 UNIX BSD 버전에 구현, 1983
  - send, receive, accept, select
- Windows에 Overlapped I/O라는 이름으로 Asynchronous I/O 구현 (Windows NT)
  - Boost/ASIO에도 있음
- IOCP(Input Output Completion Port)구현
  - Windows NT 3.5부터. Linux는 2.5.44부터 epoll
  - Kernel에서 Thread와 I/O를 직접 control
- RIO
  - windows 8.1
  - Linux는 5.1부터 io_uring

RIO

- 특징
  - 운영체제 호출 감소
    - System Call없이 I/O 동작의 종료를 알 수 있다.
  - NUMA node에서 affinity설정 가능
    - thread 뿐만 아니라 buffer에도 적용 가능

RIO

- 추천 (by Microsoft)
  - 메세지당 CPU사용량을 줄여서 서버 용량을 늘리고 싶을 때
  - 네트워크 스택의 latency를 줄이고, jitter를 감소시키고 싶을 때
  - 많은 개수의 multicast나 UDP 트래픽이 필요할 때

RIO

- 이득 (by Microsoft)
  - 모든 버전의 Windows Server 2012에서 동작
  - 특별한 네트워크 카드가 필요하지 않고, 일반 네트워크 카드에서도 동작
  - 기존의 Windows네트워크 기능들과 완벽히 호환
    - RSS(Receive Segment Coalescing): 네트워크 카드에서 패킷 합체
    - RCS(Receive Side Scaling): 네트워크 카드레벨에서의 멀티 CPU, 멀티 Core load balancing 기능
    - 등등
  - Hyper-V와의 연동
  - 표준 TCP/IP, UDP와 연동

RIO

- API
  - WSASocket
    - WSA_FLAG_REGISTERED_IO 플랙이 생김
  - RIOCreateCompletionQueue
    - RIO 완료 신호를 받는 방법을 설정
  - RIOCreateRequestQueue
    - Request Queue를 생성
  - RIORegisterBuffer
    - PAGE를 RIO용 data buffer로 설정
  - RIOSend
    - WSASend를 대체
  - RIOReceive
    - WSARecv를 대체
  - RIODequeueCompletion
    - RIO 완료 결과 들을 가져옴

완료 방식

- RIOCompletionQueue 폴링
  - 최고의 성능과, 최저의 대기시간을 가지나 높은 CPU 사용률
- I/O CompletionPort
  - IOCP 기반의 코드에 적용하기가 쉬움
- Windows Event를 통한 통지
  - Windows Event기반의 코드에 적용하기 쉬움

완료 방식

- RIOCompletionQueue 폴링

```
ULONG NResults = 0;
RIORESULT Results[MaxResults];

// Poll the completion queue for completions

while (0 == (NResults = RIODequeueCompletion(CQ,
                &Results[0], MaxResults))) {
    YieldProcessor();
}
```

완료 방식

- IOCP 연동

```
RIO_NOTIFICATION_COMPLETION NotificationCompletion;

NotificationCompletion.Type = RIO_IOCP_COMPLETION;
NotificationCompletion.Iocp.IocpHandle = Iocp;
NotificationCompletion.Iocp.Overlapped = &Overlapped;
NotificationCompletion.Iocp.CompletionKey = NULL;

CQ = Rio.RIOCreateCompletionQueue(QueueSize, &NotificationCompletion);
```

```
// Wait for one or more completions, and
// get them all in one operation

  GetQueuedCompletionStatus(IocpHandle …)

  NResults = RIODequeueCompletion(CQ, &Results[0], MaxResults);
```

서버 프로그램 구성

- Polling 방식

1. RIO함수 table설정
2. Buffer 할당 및 등록
3. CompletionQueue 작성
4. Accept
5. RequestQueue 작성
6. RIORecv
7. RIODequeueCompletion
8. goto 7

서버 프로그램 구성

1. RIO함수 table설정

- RIO API는 일반 함수가 아니라 런타임에 함수포인터를 가져와서 사용해야 한다.
- WSAIoctl함수를 사용한다, gRIOFuncTable에 함수들이 등록된다.

```
WSAIoctl(listenSocket,
  SIO_GET_MULTIPLE_EXTENSION_FUNCTION_POINTER,
  &functionTableId, // = WSAID_MULTIPLE_RIO;
  sizeof(GUID),
  (void**)&gRIOFuncTable,
  sizeof(gRIOFuncTable),
  &dwBytes, NULL, NULL);
  clientSocket);
```

서버 프로그램 구성

- 2. Buffer 할당 및 등록
  - RIO에 사용되는 버퍼는 Page 단위로 등록되어야 하므로 Page할당 함수를 통해 할당 받아야 한다.
  - Page를 어떻게 분할하고 어떻게 재사용할 지는 다 프로그래머의 몫
    - Page는 크기때문에 쪼개서 사용해야 한다.
    - buffer의 크기는 프로그램 전체에서 사용할 수 있을 만큼 크게 잡는다.
    - 멀티쓰레드 프로그램의 경우 쓰레드마다 따로 잡는 것이 좋다.
  - RIORegisterBuffer로 등록한 후 반환된 ID를 사용한다.

```
g_buffer = VirtualAllocEx(GetCurrentProcess(), 0,
  buff_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE));
g_bufid = gRIOFuncTable.RIORegisterBuffer(g_buffer, buff_size);
```

서버 프로그램 구성

3. CompletionQueue 작성

- 완료된 IO들의 정보를 받아올 Queue를 만든다.
- Recv용과 Send용을 따로 만들 수도 있고, 하나를 두 곳에 같이 사용할 수도 있다.
- 여러 개의 socket이 하나의 CompletionQueue를 공유한다.
  - 따라서 queue_size는 충분히 커야 한다. 대응되는 모든 RequestQueue의 크기를 합한 값과 같아야 한다.

```
RIO_CQ compQueue = gRIOFuncTable.RIOCreateCompletionQueue(
     queue_size, 0);
```

서버 프로그램 구성

4. Accept

- 일반 TCP 서버와 거의 같다.
- listenSocket을 만들 때 WSA_FLAG_REGISTERED_IO 값을 주어야 한다.
  - WSA_FLAG_OVERLAPPED_IO를 넣을 필요는 없다.
- Non-Blocking Accept는 불가능하다. non-blocking을 원하면 AcceptEx를 사용해야 한다.

서버 프로그램 구성

5. RequestQueue 작성

- socket마다 하나씩 존재해야 하며, 모든 RIO 데이터 전송은 socket이 아닌 이 queue를 통해 이루어 진다.
- socket과 completion queue들을 모아 RIO 구조를 완성한다.
  - queue크기는 최대 저장가능한 event수이고, max_bufs는 gather_scatter용 버퍼 개수이다.
  - info는 사용자가 완료 쪽에 넘겨주는 정보. (보통 socket 값)
  - rq_size보다 compQueue의 크기가 더 커야 한다.
    - 보통 compQueue_size = rq_size * max_sockets

```
reqQueue = gRIOFuncTable.RIOCreateRequestQueue(
   clientSocket,
   recv_rq_size, recv_max_bufs,
   send_rq_size, send_max_bufs,
   recvCompQueue, sendCompQueue, (PVOID) info);
```

서버 프로그램 구성

6. RIORecv

- RIO용 Overlapped Recv 함수
  - rioBuffers를 등록하고 나중에 완료를 받는다.
  - buf_count는 rioBuffers에 들어있는 버퍼들의 개수
- rioBuffer는 등록된 버퍼 ID와 버퍼의 offset, 크기를 등록한다.
  - 하나의 buffer를 여러 클라이언트와 여러 IO에서 나누어 써야 하므로 서로 겹치지 않도록 offset이 필요하다.
  - op_info는 사용자가 완료에게 전달하는 정보이다.

```
RIO_BUF rioBuffers[buf_count];
rioBuffers[i].BufferId = g_bufid;
rioBuffers[i].Length = buf_length;
rioBuffers[i].Offset = my_position;
gRIOFuncTable.RIOReceive(reqQueue, rioBuffers, buf_count,
        NULL, (PVOID)op_info);
```

서버 프로그램 구성

6. RIODequeueCompletion

- RIO의 main loop를 구성하는 함수
- 완료된 IO들을 수집하여 처리한다.
- compQueue마다 따로 다 해야 한다.
- RequestContext, BytesTransferrd, SocketContext가 전부니까 이것들을 가지고 무엇이 완료되었는지 판단해야 한다.

```
RIORESULT results[queue_size];
ULONG numResults = gRIOFuncTable.RIODequeueCompletion(
            compQueue, results, queue_size);
for (ULONG i = 0; i < numResults; ++i) {
   op_info      = results[i].RequestContext;
   ULONG bytes = results[i].BytesTransferred;
   info         = results[i].SocketContext);
```

1주차 과제

- Sample로 제공된 프로그램을 RIO로 동작하도록 수정하시오.
  - Overlapped I/O로 되어 있는 소스코드 수정
  - Client/Server 둘 다 수정
- 제출 E-Class로 제출
- 제출 기한 3월 25일 오후 1시