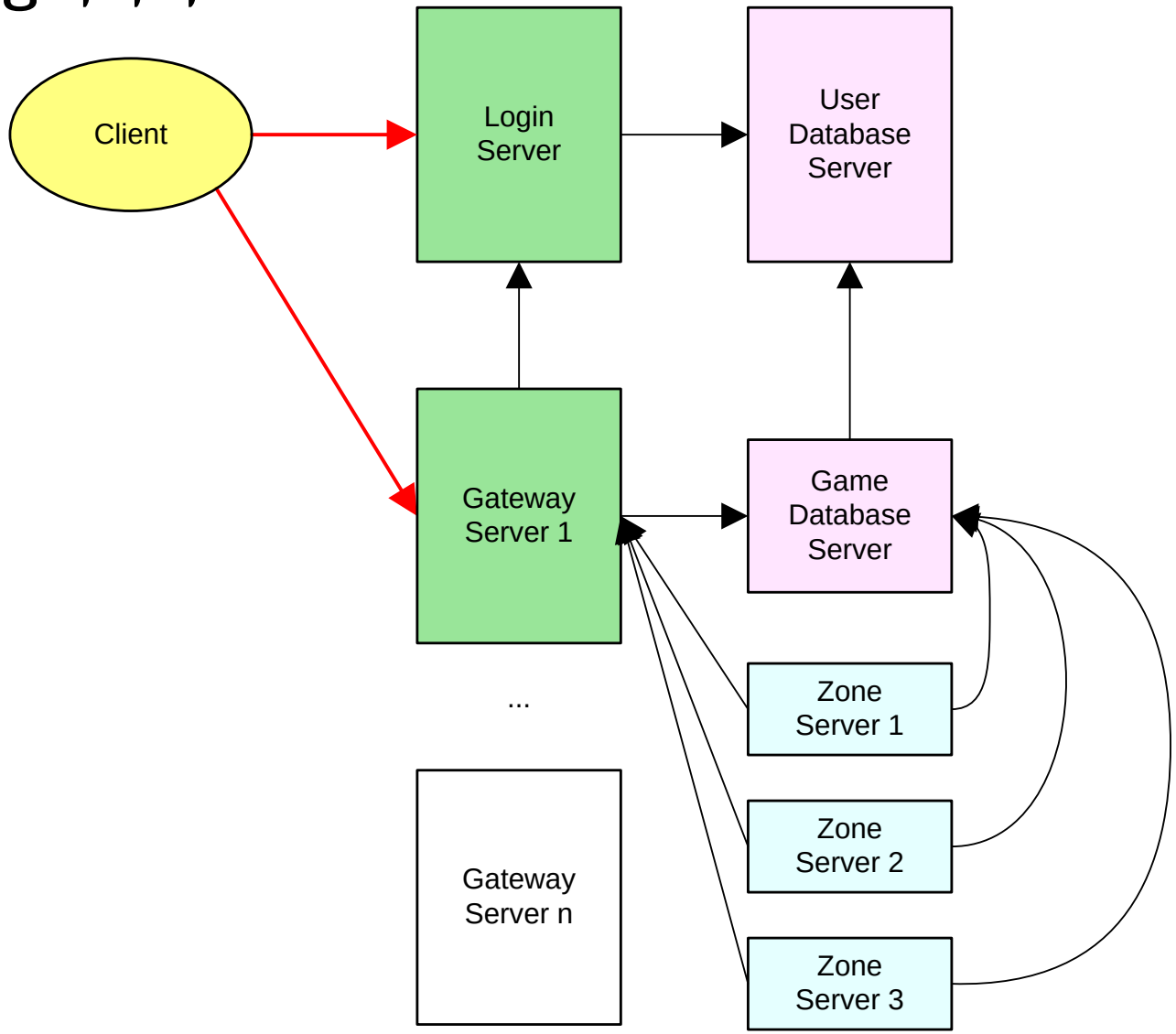
Network Programming - 05

afewhee@gmail.com

- 게임 서버 구조
- Network 추상 클래스
- Network Library
- Socket Option

- 게임서버 구조



```
interface ILcNet
{
    virtual INT Create(...)=0;          // 소켓 생성
    virtual void Destroy()=0;           // 소켓을 소멸
    virtual INT FrameMove()=0;      // 네트워크에서 필요한 데이터 업데이트
    virtual INT Query(char* sCmd, void* pData)=0;
    virtual INT Close()=0;              // 소켓만 소멸
    virtual INT Listen()=0;             // 서버에서 사용되는 리슨
    virtual INT ...)=0;
    virtual INT Send(...)=0;            // 패킷을 보낼 때
    virtual INT Recv(...)=0;            // 패킷을 받을 때
    virtual LRESULT MsgProc(HWND,UINT,WPARAM,LPARAM)=0;  // AsyncSelect용
};
```

- WinCE
  - class CLcNetSlct : public CLcNet
- 게임 크라이언트
  - AsyncSelect
    - class CLcNetSlctA : public CLcNet
  - Event Select
    - class CLcNetSlctE : public CLcNet
- 서버 - IOCP
  - class CLcNetIocp : public CLcNet

- Error Message
  - void LcNet_GetNetworkError(DWORD hr);// Network Error Message Catching
  - INT LcNet_WSAGetError();

- Winsock DLL
  - INT LcNet_WSAStartup(); // 윈속 라이브러리 초기화
  - void LcNet_WSACleanup(); // 윈속 라이브러리 해제

- Socket
  - INT LcNet_SocketErrorCheck(INT hr); // Socket Error Checking
  - void LcNet_SocketAddr(SOCKADDR_IN* pOut, char* sIp, char* sPort); // Setting Socket Address
  - void LcNet_SocketClose(SOCKET* scH); // Socket Close
  - INT LcNet_SocketTcpCreate(SOCKET* pScH, BOOL bOverLapped=FALSE); // Create TCP Socket
  - INT LcNet_SocketUdpCreate(SOCKET* pScH, BOOL bOverLapped=FALSE); // Create UDP Socket
  - INT LcNet_SocketConnect(SOCKET scH, SOCKADDR_IN* psdH); // Connection
  - INT LcNet_SocketBind(SOCKET scH, SOCKADDR_IN* pSdH); // Socket Binding
  - INT LcNet_SocketListen(SOCKET scH); // Socket Listen
  - INT LcNet_SocketAccept(SOCKET* pscOut // Output Socket
  - , SOCKADDR_IN* psdOut // Output Socket Address
  - , SOCKET scListen); // Accept
  - INT LcNet_SocketSelect(FD_SET*, FD_SET*, TIMEVAL*, FD_SET* pFdExcept=NULL);
  - INT LcNet_SocketNonBlocking(SOCKET scH, BOOL bOn=TRUE); // Set NonBlocking
  - INT LcNet_SocketNaggleOff(SOCKET scH, BOOL bOff=TRUE); // Off nagle Algorithm
  - INT LcNet_SocketDirectBuffer(SOCKET scH); // Disable send buffering on the socket

- 쓰레드 관련 함수들
  - HANDLE LcNet_ThreadCreate(...); // _beginthreadex()
  - void LcNet_ThreadEnd(); // _endthreadex(0)
  - void LcNet_ThreadClose(HANDLE* hThread); // Thread Close
  - DWORD LcNet_ThreadResume(HANDLE* hThread); // Thread resume
  - DWORD LcNet_ThreadSuspend(HANDLE* hThread); // Thread Suspend
  - DWORD LcNet_ThreadWait(HANDLE* hThread, DWORD dMilliseconds = INFINITE);

- 이벤트
  - HANDLE LcNet_EventCreate();
  - void LcNet_EventResume(HANDLE hEvent); // 이벤트 활성화
  - void LcNet_EventSuspend(HANDLE hEvent); // 이벤트 비활성화
  - INT LcNet_EventWait(HANDLE hEvent, DWORD dWait=INFINITE); // Event Wait
  - void LcNet_EventClose(HANDLE* hEvent);
  - HANDLE LcNet_WSAEventCreate(); // WSA Event Create
  - void LcNet_WSAEventClose(HANDLE* pEvent); // WSA Event Close
  - INT LcNet_WSAEventSelect(..., lEvents =(FD_ACCEPT|FD_CONNECT|FD_READ|FD_WRITE|FD_CLOSE));
  - INT LcNet_WSAEventWaits(...); // Return is Event Count
  - INT LcNet_WSAEventEnum(SOCKET s, HANDLE e); // WSA Enum Network Event

- 소켓 옵션(socket options)
  - 소켓 함수의 기본 동작을 변경
    - 소켓 코드와 프로토콜 구현 코드에 대한 세부적인 제어 가능

- 소켓 옵션의 종류

① 소켓 코드가 담당하는 부분
  - 옵션을 설정하면 소켓 코드에서 해석하고 처리함

② 프로토콜 구현 코드가 담당하는 부분
  - 옵션을 설정하면 프로토콜 구현 코드에서 해석하고 처리함

소켓 옵션 설정하기

```
int setsockopt (
    SOCKET s,
    int level,
    int optname,
    const char* optval,
    int optlen
);
```

성공: 0, 실패: SOCKET_ERROR

소켓 옵션 얻기

```
int getsockopt (
    SOCKET s,
    int level,
    int optname,
    char* optval,
    int* optlen
);
```

성공: 0, 실패: SOCKET_ERROR

소켓 옵션 - SOL_SOCKET

optname	optval 타입	get	set	설명
SO_BROADCAST	BOOL	•	•	브로드캐스팅 허용
SO_DONTROUTE	BOOL	•	•	데이터 전송시 라우팅 테이블 참조 과정 생략
SO_KEEPALIVE	BOOL	•	•	주기적으로 연결 여부 확인
SO_LINGER	struct linger{}	•	•	보낼 데이터가 있을 경우 closesocket() 함수 리턴 지연
SO_SNDBUF SO_RCVBUF	int	•	•	소켓 송/수신 버퍼 크기 설정
SO_SNDTIMEO SO_RCVTIMEO	int	•	•	send(), recv() 등의 함수에 대한 타임아웃 설정
SO_REUSEADDR	BOOL	•	•	지역 주소(IP 주소, 포트 번호) 재사용 허용

소켓 옵션 - IPPROTO_IP

optname	optval 타입	get	set	설명
IP_HDRINCL	BOOL	•	•	데이터를 보낼 때 IP 헤더를 포함
IP_TTL	int	•	•	IP 패킷의 TTL(time-to-live) 변경
IP_MULTICAST_IF	IN_ADDR{}	•	•	멀티캐스트 패킷을 보낼 인터페이스 설정
IP_MULTICAST_TTL	int	•	•	멀티캐스트 패킷의 TTL 변경
IP_MULTICAST_LOOP	BOOL	•	•	멀티캐스트 패킷의 루프백 여부 설정
IP_ADD_MEMBERSHIP IP_DROP_MEMBERSHIP	struct ip_mreq{}		•	멀티캐스트 그룹 가입과 탈퇴

● 소켓 옵션 - IPPROTO_TCP

optname	optval 타입	get	set	설명
TCP_NODELAY	BOOL	•	•	Nagle 알고리즘 작동 중지

- 용도
  - 해당 소켓을 이용하여 브로드캐스트 데이터 전송 가능
  - UDP 소켓에만 사용 가능

- 용도
  - 데이터 전송시 라우팅 테이블 참조를 생략하고, 곧바로 bind() 함수로 설정한 네트워크 인터페이스로 모든 데이터를 보냄
- 사용 예

```
BOOL optval = TRUE;
if(setsockopt(listen_sock, SOL_SOCKET, SO_DONTROUTE,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
{
    err_quit("setsockopt()");
}
```

- 용도
  - TCP 프로토콜 수준에서 연결 여부를 확인하기 위해 상대 TCP에게 주기적으로(약 2시간 간격) TCP 패킷을 보냄
- 사용 예

```
BOOL optval = TRUE;
if(setsockopt(listen_sock, SOL_SOCKET, SO_KEEPALIVE,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
{
    err_quit("setsockopt()");
}
```

- 용도
  - closesocket() 함수의 디폴트 동작 변경

```
send(sock, ...); // 데이터를 보낸다.
closesocket(sock); // 소켓을 닫는다.
```

- 옵션값

```
struct linger {
    u_short l_onoff; /* option on/off */
    u_short l_linger; /* linger time */
};
typedef struct linger LINGER;
```

- 사용 예

```
LINGER optval;
optval.l_onoff = 1; /* linger on */
optval.l_linger = 10; /* linger time = 10초 */
if(setsockopt(sock, SOL_SOCKET, SO_LINGER,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
{
    err_quit("setsockopt()");
}
```

- 옵션값에 따른 closesocket() 함수의 동작

struct linger{}		closesocket() 함수 동작	추가 설명
l_onoff	l_linger		
0	사용 안함	①과 동일	closesocket() 함수의 디폴트 동작
1	0	②와 동일	
1	양수	③과 동일	

① closesocket() 함수는 곧바로 리턴하고 송신 버퍼의 데이터는 백그라운드로 보낸 후 TCP 연결을 정상 종료

② closesocket() 함수는 곧바로 리턴하고 송신 버퍼의 데이터는 삭제한 후 TCP 연결을 강제 종료

③ 송신 버퍼의 데이터를 모두 보내고 TCP 연결을 정상 종료한 후 closesocket() 함수 리턴. 일정 시간 내에 송신 버퍼의 데이터를 모두 보내지 못하면 TCP 연결을 강제 종료한 후 closesocket() 함수 리턴. 이때 송신 버퍼에 남은 데이터는 삭제함.

- 용도
  - 소켓의 송신 버퍼와 수신 버퍼 크기 변경
- 사용 예

```
int optval;
int optlen = sizeof(optval);
if(getsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    (char *)&optval, &optlen) == SOCKET_ERROR)
    err_quit("getsockopt()");
printf("수신 버퍼 크기 = %d 바이트₩n", optval);

optval *= 2;
if(setsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
    err_quit("setsockopt()");
```

- 용도
  - 데이터 전송 함수(send(), recv(), sendto(), recvfrom())가 작업 완료와 상관없이 일정 시간 후 리턴하도록 함
- 사용 예

```
int optval = 3000;
if(setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
{
    err_quit("setsockopt()");
}
```

- 용도
  - Nagle 알고리즘 작동 여부 결정
- Nagle 알고리즘

① 보낼 데이터가 MSS(maximum segment size)로 정의된 크기만큼 쌓이면, 상대편에게 무조건 보냄

② 보낼 데이터가 MSS보다 작을 경우, 이전에 보낸 데이터에 대한 ACK가 오기를 기다림. ACK가 도달하면 보낼 데이터가 MSS보다 작더라도 상대에게 보냄

- Nagle 알고리즘의 장단점
  - 장점: 작은 패킷이 불필요하게 많이 생성되는 것을 미연에 방지함으로써 네트워크 트래픽을 감소시킴
  - 단점: 데이터가 충분히 쌓일 때까지 또는 ACK가 도달할 때까지 대기하는 시간 때문에 애플리케이션의 반응 시간 (response time)이 길어질 가능성이 있음

- 사용 예

```
BOOL optval = TRUE; // Nagle 알고리즘 작동 중지
if(setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
    (char *)&optval, sizeof(optval)) == SOCKET_ERROR)
{
    err_quit("setsockopt()");
}
```