



# Winsock 함수호출과 TDI 함수호출

## 관계

작성자 : vmmmger

작성일 : 2003-02-04

## 1. 서론

## 2. Winsock함수 동작 설명

### 2.1. 클라이언트인 경우

### 2.2. 서버인 경우

## 3. Winsock함수호출에 대응되는 TDI함수 호출 설명

### 3.1. 연결

### 3.2. 송수신

### 3.3. 연결 종료 1

### 3.1. 연결 종료 2

## 3. 결론

# 1. 서론

TDI함수에 대한 이해를 목적으로 이 문서를 작성했다.

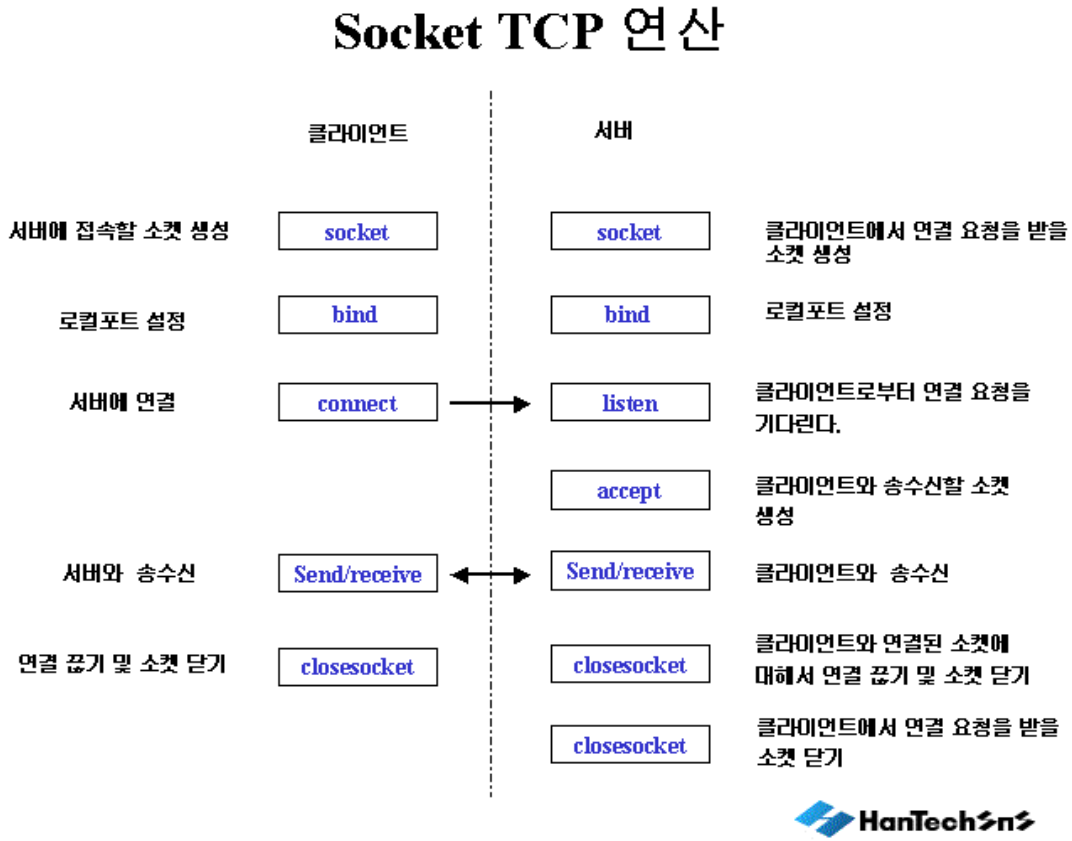
일반적으로 인터넷 프로그램을 개발할 때 사용되는 Winsock 함수에 대해서는 어느 정도 개발자들이 지식을 갖고 있다.

그리고 Winsock함수와 TDI함수는 매우 유사한 부분이 많다.

Winsock함수를 호출하고 그에 따른 TDI함수의 호출을 모니터링 한다면 TDI함수에 대한 이해가 더 쉬울 것이다.

본 문서는 Windows 2000 Pro환경에서 클라이언트와 서버용 Winsock 함수들을 호출하고 [www.sysinternals.com](http://www.sysinternals.com)의 tdimon 프로그램을 통해 모니터링된 정보를 바탕으로 작성되었다.

## 2. Winsock함수 동작 설명



다음은

클라이언트 IP 100.100.100.103 PORT 100 ↔ 서버 IP 100.100.100.105 PORT 10000

의 TCP 연결 및 종료를 Winsock함수로 구현한 코드를 중심으로 설명한다.

### 2.1. 클라이언트인 경우

#### 1) 소켓 생성

```
m_ClientSocket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

#### 2) 바인드

로컬정보(로컬포트와 로컬IP)를 설정한다.

로컬포트가 0인 경우는 운영체제가 임의로 로컬포트를 설정한다.

로컬IP가 0인 경우는 운영체제가 임의로 로컬IP를 설정한다.

```
struct sockaddr_in local;
memset(&local,0,sizeof(local));
local.sin_addr.s_addr=0x67646464; //로컬 IP 를 100.100.100.103으로 설정
local.sin_family=AF_INET;
local.sin_port=htons(100); //로컬 PORT 를 100으로 설정
bind(m_ClientSocket, (struct sockaddr*)&local,sizeof(local))
```

### 3) 연결

연결할 서버 IP와 포트를 인자로 해서 서버에 연결한다.  
TCP/IP프로토콜의 3핸드셰이킹이 이 부분에서 수행한다.

```
struct sockaddr_in remote;
remote.sin_addr.s_addr = 0x69646464; //연결할 원격지 IP를 100.100.100.105으로 설정한다.
remote.sin_family=AF_INET;
remote.sin_port=htons(10000); //연결할 원격지포트를 1000으로 설정한다.
connect(m_ClientSocket,(struct sockaddr*)&remote,sizeof(remote));
```

### 4) 송신

1000바이트를 보낸 경우  
char buff[1000] ;  
**send**(m\_ClientSocket,buff,1000,0);

1000바이트를 송신한다.

### 5) 연결 종료 및 소켓 닫기

```
closesocket(m_ClientSocket);
```

## 2.2. 서버인 경우

### 1) 소켓 생성

```
m_ServerSocket= socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

### 2) 바인드

로컬정보(로컬포트와 로컬IP)를 설정한다.

로컬포트가 0인 경우는 운영체제가 임의로 설정한다.

로컬IP가 0인 경우는 운영체제가 임의로 설정한다.

```
struct sockaddr_in local;  
memset(&local,0,sizeof(local));  
local.sin_addr.s_addr=0x69646464;  
local.sin_family=AF_INET;  
local.sin_port=htons(100);  
bind(m_ServerSocket, (struct sockaddr*)&local,sizeof(local))
```

### 3) listen

클라이언트로부터 연결요청을 기다린다.

```
listen(m_ServerSocket,4);
```

### 4) accept

클라이언트로부터 연결요청을 받아들이고 클라이언트와 통신할 새로운 소켓을 생성한다.

```
sockaddr_in from;  
int fromlen=sizeof(from);  
m_AcceptSocket = accept(m_ServerSocket, (struct sockaddr*)&from, &fromlen);
```

### 5) 연결 종료 및 m\_AcceptSocket소켓 닫기

```
closesocket(m_AcceptSocket);
```

### 6) m\_ServerSocket소켓을 닫기.

```
closesocket(m_ServerSocket);
```

### 3. Winsock함수호출에 대응되는 TDI함수 호출 설명

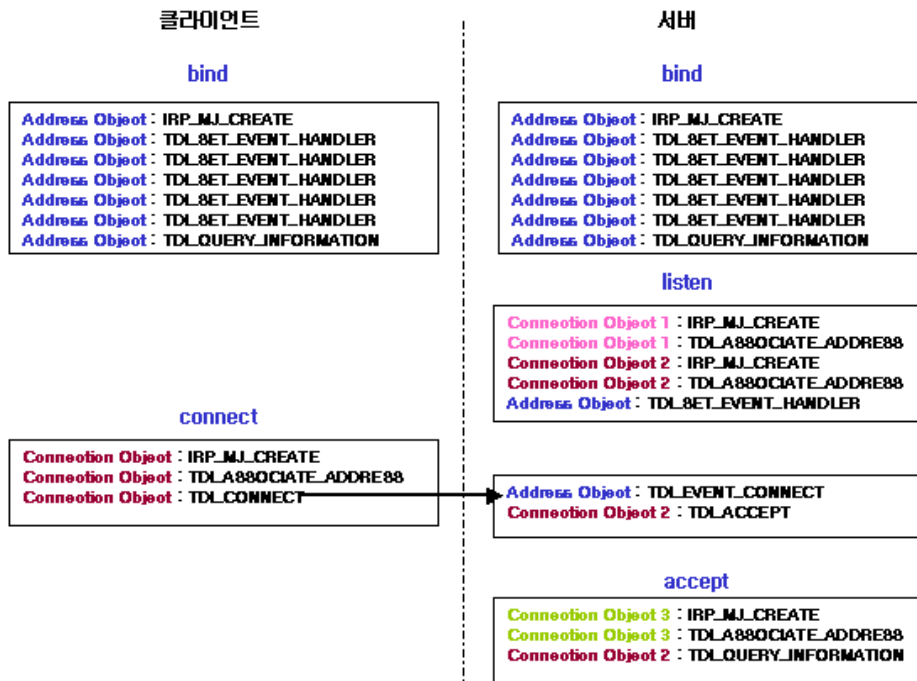
다음은

클라이언트 IP 100.100.100.103 PORT 100 ↔ 서버 IP 100.100.100.105 PORT 10000

의 TCP 연결 및 종료를 Winsock함수호출과 TDI함수호출의 상관관계를 설명한다.

#### 3.1. 연결

### Socket TCP 연산 - 연결



### 3.1.1. 클라이언트인 경우

#### - bind

로컬정보(로컬포트와 로컬IP)를 설정한다.

로컬포트가 0인 경우는 운영체제가 알아서 로컬포트를 설정한다.

로컬IP가 0인 경우는 운영체제가 알아서 로컬IP를 설정한다.

```
struct sockaddr_in local;
memset(&local,0,sizeof(local));
local.sin_addr.s_addr=0x67646464; //로컬 IP 를 100.100.100.103으로 설정
local.sin_family=AF_INET;
local.sin_port=htons(100); //로컬 PORT 를 100으로 설정
bind(m_ClientSocket, (struct sockaddr*)&local,sizeof(local))
```

다음은 Winsock함수의 bind 호출시 TDI함수 호출 내용이다.

object	request	Local	Remote	result
address	IRP_MJ_CREATE	TCP:100.100.100.103:100		SUCCESS Address Open
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Error Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Disconnect Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Receive Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Expedited Receive Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Chained Receive Event
address	TDI_QUERY_INFORMATION	TCP:100.100.100.103:100		SUCCESS Query Address

Address 객체를 IRP\_MJ\_CREATE로 생성한다.

TDI\_SET\_EVENT\_HANDLER 로 이벤트 핸들러들을 등록한다.

등록된 이벤트 핸들러들은 예를 들면, 데이터가 원격지로부터 수신되었을 때나 원격지로부터 연결 종료가 감지되었을 때 호출된다.

유의 사항 : bind시 TDI\_SET\_EVENT\_HANDLER의 connect event는 등록이 안된다. 이 이벤트핸들러 등록은 서버로서 동작할 때 발생한다. 즉, listen 함수 호출 때 발생한다.

**- connect**

연결할 서버 IP와 포트를 인자로 해서 서버에 연결한다.  
TCP/IP프로토콜의 3핸드셰이킹이 이 부분에서 수행한다.

```
struct sockaddr_in remote;
remote.sin_addr.s_addr = 0x69646464; //연결할 원격지 IP를 100.100.100.105으로 설정한다.
remote.sin_family=AF_INET;
remote.sin_port=htons(10000); //연결할 원격지포트를 1000으로 설정한다.
connect(m_ClientSocket,(struct sockaddr*)&remote,sizeof(remote));
```

다음은 Winsock함수의 connect 호출시 TDI함수 호출 내용이다.

object	Request	local	remote	result
connect	IRP_MJ_CREATE	TCP:Connection obj		SUCCESS Context:0x811FCB88
connect	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj		SUCCESS TCP:100.100.100.103:100
connect	TDI_CONNECT	TCP:100.100.100.103: 100	TCP:100.1 00.100.105: 10000	SUCCESS

Connect 객체를 IRP\_MJ\_CREATE로 생성한다.

클라이언트에서 TDI\_CONNECT 호출시 서버쪽에서는 TDI\_EVENT\_CONNECT 함수와 TDI\_ACCEPT함수가 호출이 된다.

이때 클라이언트와 서버가 연결이 이루어진다.

생성된 connect 객체를 통해서 송수신이 발생한다.

주목 : Winsock함수의 bind 호출없이 connect만 호출했을 때, bind시 호출하는 TDI함수인 IRP\_MJ\_CREATE, TDI\_SET\_EVENT\_HANDLER, TDI\_QUERY\_INFORMATION 가 호출이 되고 connect시 호출되는 IRP\_MJ\_CREATE, TDI\_ASSOCIATE\_ADDRESS가 모두 호출이 된다.

이때 로컬IP와 포트와 운영체제가 임의로 설정한 내용이 된다.

### 3.1.2. 서버인 경우

#### - bind

클라이언트에서의 bind와 같은 기능을 한다.

로컬정보(로컬포트와 로컬IP)를 설정한다.

로컬포트가 0인 경우는 운영체제가 알아서 로컬포트를 설정한다.

로컬IP가 0인 경우는 운영체제가 알아서 로컬IP를 설정한다.

```
struct sockaddr_in local;  
memset(&local,0,sizeof(local));  
local.sin_addr.s_addr=0x67646464; //로컬 IP 를 100.100.100.103으로 설정  
local.sin_family=AF_INET;  
local.sin_port=htons(100); //로컬 PORT 를 100으로 설정  
bind(m_ClientSocket, (struct sockaddr*)&local,sizeof(local))
```

다음은 Winsock함수의 bind 호출시 TDI함수 호출 내용이다.

object	request	Local	Remote	result
address	IRP_MJ_CREATE	TCP:100.100.100.105:10000		SUCCESS Address Open
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105:10000		SUCCESS Error Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105:10000		SUCCESS Disconnect Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105:10000		SUCCESS Receive Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105:10000		SUCCESS Expedited Receive Event
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105:10000		SUCCESS Chained Receive Event
address	TDI_QUERY_INFORMATION	TCP:100.100.100.105:10000		SUCCESS Query Address

Address 객체를 IRP\_MJ\_CREATE로 생성한다.

TDI\_SET\_EVENT\_HANDLER 로 이벤트 핸들러들을 등록한다.

등록된 이벤트 핸들러들은 예를 들면, 데이터가 원격지로부터 수신되었을 때나 원격지로부터 연결 종료가 감지되었을 때 호출된다.

유의 사항 : bind시 TDI\_SET\_EVENT\_HANDLER의 connect event는 등록이 안된다. 이 이벤트핸들러 등록은 서버로서 동작할 때 발생한다. 즉, listen 함수 호출 때 발생한다.

**- listen**

`listen(m_ServerSocket,2);`

클라이언트로부터 연결요청을 기다린다.

다음은 Winsock함수의 listen 호출시 TDI함수 호출 내용이다.

object	request	local	remote	result
<b>Connect 1</b>	<b>IRP_MJ_CREATE</b>	<b>TCP:Connection obj</b>		<b>SUCCESS Context:0x81120708</b>
<b>Connect 1</b>	<b>TDI_ASSOCIATE_ADDRESS</b>	<b>TCP:Connection obj</b>		<b>SUCCESS TCP:100.100.100.105:10000</b>
<b>Connect 2</b>	<b>IRP_MJ_CREATE</b>	<b>TCP:Connection obj</b>		<b>SUCCESS Context: 0x8560DCE8</b>
<b>Connect 2</b>	<b>TDI_ASSOCIATE_ADDRESS</b>	<b>TCP:Connection obj</b>		<b>SUCCESS TCP:100.100.100.105:10000</b>
<b>address</b>	<b>TDI_SET_EVENT_HANDLER</b>	<b>TCP:100.100.100.105:10000</b>		<b>SUCCESS Connect Event</b>

Connect 객체를 listen의 두번째인 자인 backlog의 수만큼 생성한다.

여기서는 2이므로 connect 객체가 2개 생성된다.

그리고 클라이언트로 연결 요청을 받기 위한 이벤트 핸들러인 TDI\_EVENT\_CONNECT를 등록한다.

**- 클라이언트에서 연결 요청이 오는 경우**

= 클라이언트가 connect 한 경우

이 경우 소켓함수의 호출이 없더라도 TDI함수가 호출된다.

object	Request	local	remote	result
<b>address</b>	<b>TDI_EVENT_CONNECT</b>	<b>TCP:100.100.100.105:10000</b>	<b>100.100.100.103:100</b>	<b>MORE_PROCESSING_REQUIRED</b>
<b>Connect 2</b>	<b>TDI_ACCEPT</b>	<b>TCP:100.100.100.105:10000</b>	<b>100.100.100.103:100</b>	<b>SUCCESS</b>

클라이언트가 TDI\_CONNECT함수 호출했을 때 서버쪽에서는 TDI\_EVENT\_CONNECT가 발생하고

TDI\_ACCEPT를 호출한다.

이때 address객체를 통해서 TDI\_EVENT\_CONNECT가 호출된다.

그리고 listen시 마지막에 생성된 **Connect 2** 객체를 통해서 TDI\_ACCEPT 가 호출된다.

**TDI\_ACCEPT와 Winsock함수의 accept하고는 호출에 상관이 없다.**

#### - accept

```
sockaddr_in from;
```

```
int fromlen=sizeof(from);
```

```
m_AcceptSocket = accept(m_ServerSocket, (struct sockaddr*)&from, &fromlen);
```

클라이언트로부터 연결요청을 받아들이고 클라이언트와 통신할 새로운 소켓을 생성한다.

object	Request	local	remote	result
<b>Connect 3</b>	IRP_MJ_CREATE	TCP:Connection obj		SUCCESS Context: 0x84B80BA8
<b>Connect 3</b>	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj		SUCCESS TCP:0.0.0.0:200
<b>Connect 2</b>	TDI_QUERY_INFORMATION	TCP:100.100.100.105: 10000	100.100.100.103 :100	SUCCESS Query Address

Connect 객체를 생성한다.

그리고 listen시 마지막에 생성된 **Connect 2** 객체를 통해서 TDI\_QUERY\_INFORMATION 가 호출된다.

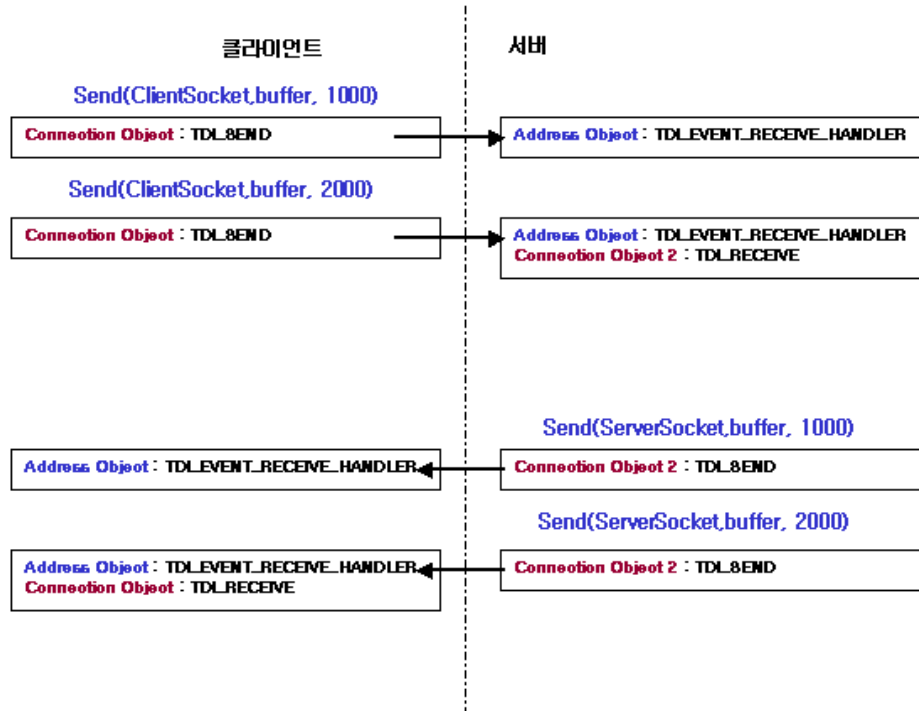
listen시 마지막에 생성된 **Connect 2** 객체를 통해서 클라이언트와 송수신이 이루어진다.

주의 : accept에서 생성된 **Connect 3** 객체로 클라이언트와 송수신이 이루어지는 것이 아니고 listen때 생성된 **Connect 2** 객체로 송수신이 이루어지는 것이다. accept에서 생성된 **Connect 3** 객체는 다른 클라이언트로부터의 연결 요청을 받을 때 사용이 된다.

**TDI\_ACCEPT와 Winsock함수의 accept하고는 호출에 상관이 없다.**

### 3.2. 송수신

## Socket TCP 연산 - 송수신



클라이언트와 서버의 송수신 연산은 같다.

#### 3.2.1. 1000바이트 송수신 연산

- 송신측

send

```
char buff[1000];
send(m_ClientSocket,buff,1000,0);
```

다음은 Winsock함수의 send 호출시 TDI함수 호출 내용이다.

object	Request	Local	remote	result
Connect	TDL_SEND	TCP:100.100.100.103:100	100.100.100.105:10000	SUCCESS Length:1000



- 수신측

데이터를 수신한 경우에 소켓함수 호출이 없이 TDI함수가 호출된다.

object	Request	Local	remote	result
<b>Connect</b>	<b>TDI_EVENT_RECEIVE</b>	<b>TCP:100.100.100.105: 100000</b>	<b>100.100.100.103 :100</b>	<b>Length:1000 Flags: ENTIRE_MESSAGE LOOKAHEAD DISPATCH</b>

3.2.2. 2000바이트 송수신 연산

- 송신측

**send**

```
char buff[2000] ;
send(m_ClientSocket,buff,2000,0);
```

다음은 Winsock함수의 send 호출시 TDI함수 호출 내용이다.

object	Request	Local	remote	result
<b>Connect</b>	<b>TDI_SEND</b>	<b>TCP:100.100.100.103:100</b>	<b>100.100.100.105:10000</b>	<b>SUCCESS Length:2000</b>

- 수신측

데이터를 수신한 경우에 소켓함수 호출이 없이 TDI함수가 호출된다.

object	Request	Local	remote	result
<b>Connect</b>	<b>TDI_EVENT_RECEIVE</b>	<b>TCP:100.100.100.105: 100000</b>	<b>100.100.100.103 :100</b>	<b>MORE_PROCESSING_REQUIRED Length:1452      Flags: LOOKAHEAD DISPATCH</b>
<b>Connect</b>	<b>TDI_RECEIVE</b>	<b>TCP:100.100.100.105: 100000</b>	<b>100.100.100.103 :100</b>	<b>SUCCESS</b>

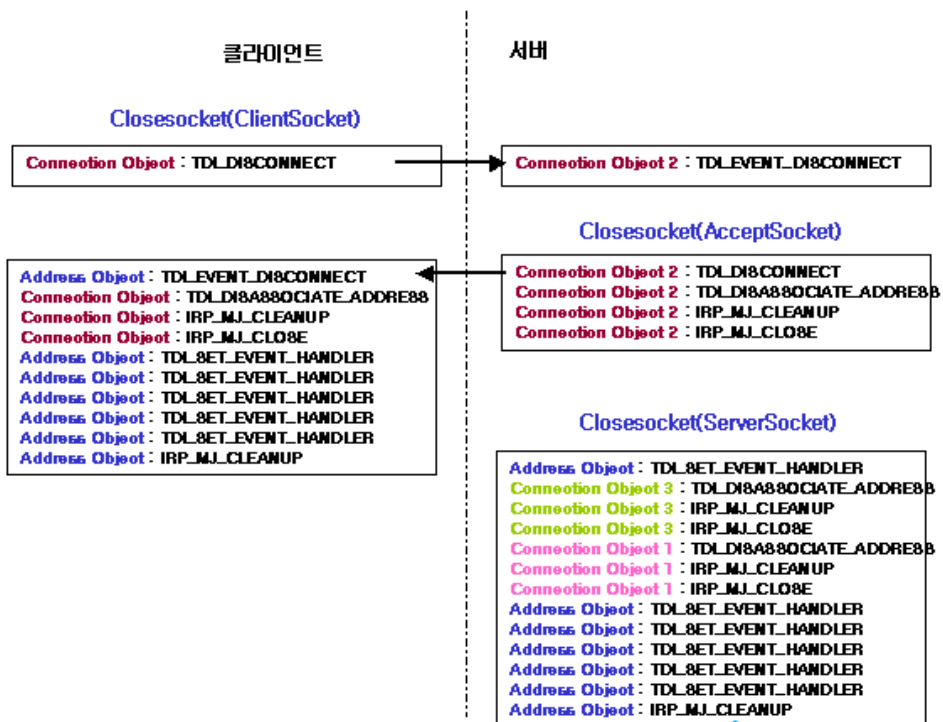
송신데이터가 1452보다 큰 경우에 수신측에서는 TDI\_EVENT\_RECEIVE와 TDI\_RECEIVE로 수신 이 이루어진다. TDI\_RECEIVE와 Winsock함수의 recv하고는 호출에 상관이 없다.

### 3.3. 연결종료 1

연결 종료는 클라이언트가 먼저 종료하느냐 서버가 먼저 종료하느냐에 따라서 TDI함수호출이 다르다.

클라이언트가 먼저 소켓을 닫고 그 후에 서버가 소켓을 닫는 경우

## Socket TCP 연산 – 연결종료 1



### 3.3.1.- 클라이언트에서 소켓을 닫았을 때

`closesocket(m_ClientSocket);`

- 클라이언트 측

Object	Request	Local	remote	Result
connect	TDI_DISCONNECT	TCP:100.100.100.103:100	100.100.100.105:10000	SUCCESS RELEASE

TDI\_DISCONNECT로 연결을 종료한다.

- 서버측

소켓함수 호출이 없이 TDI함수가 호출된다.

object	Request	local	Remote	result
address	TDI_EVENT_DISCONNECT	TCP:100.100.100.105: 10000	100.100.100.105: 100	SUCCESS RELEASE

소켓함수 호출이 없이 클라이언트의 TDI\_DISCONNECT 함수에 대해 TDI\_EVENT\_DISCONNECT 이벤트 핸들러가 호출된다.

### 3.3.2. 서버에서 소켓을 닫았을 때

`closesocket(m_AcceptSocket);`

- 클라이언트 측

소켓함수 호출이 없이 TDI함수가 호출된다.

Object	Request	local	remote	Result
address	TDI_EVENT_DISCONNECT	TCP:100.100.100.103: 100		SUCCESS RELEASE
connect	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.103: 100		SUCCESS
connect	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
connect	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103: 100		SUCCESS Error Event: NULL

address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103: 100		SUCCESS Disconnect Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103: 100		SUCCESS Receive Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103: 100		SUCCESS Expedited Receive Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103: 100		SUCCESS Chained Receive Event: NULL
address	IRP_MJ_CLEANUP	TCP:100.100.100.103: 100		SUCCESS

소켓함수 호출이 없이 서버의 TDI\_DISCONNECT 함수에 대해 TDI\_EVENT\_DISCONNECT 이벤트 핸들러가 호출된다.

서버로부터의 연결 종료를 TDI\_EVENT\_DISCONNECT 핸들러를 통해서 알 수 있다.

이때 connect 객체는 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 호출로 해제된다.

Address 객체를 통해서 이벤트 핸들러들을 NULL로 초기화하고 IRP\_MJ\_CLEANUP로 address 객체를 해제한다.

- 서버측

object	request	local	remote	result
Connect 2	TDI_DISCONNECT	TCP:100.100.100.105:10000		SUCCESS RELEASE
Connect 2	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.105:10000		SUCCESS
Connect 2	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 2	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS

TDI\_DISCONNECT로 클라이언트와 연결을 종료한다.

이때 Connect 2 객체를 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 를 통해서 해제한다.

`closesocket(m_ServerSocket);`

- 클라이언트 측

반응 없음

- 서버측

object	request	local	remote	result
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Connect Event: NULL
Connect 3	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.105: 10000		SUCCESS
Connect 3	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 3	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
Connect 1	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.105: 10000		SUCCESS
Connect 1	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 1	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Error Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Disconnect Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Receive Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Expedited Receive Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.105: 10000		SUCCESS Chained Receive Event: NULL
address	IRP_MJ_CLEANUP	TCP:100.100.100.105: 10000		SUCCESS

Listen때 생성된 connect객체들 중에서 클라이언트와 연결에 사용이 안된 connect 객체(Connect 1)와 accept시 생성된 connect 객체(Connect 3)를 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 호출로 해제한다.

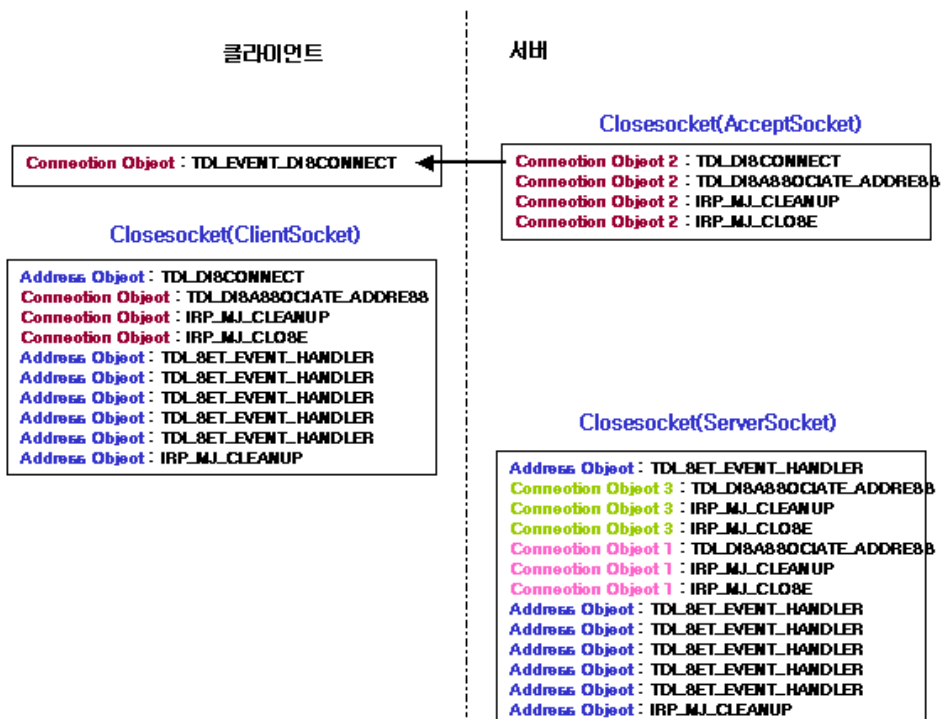
Address 객체를 통해서 이벤트 핸들러들을 NULL로 초기화하고 IRP\_MJ\_CLEANUP로 address 객체를 해제한다.

### 3.4. 연결종료 2

연결 종료는 클라이언트가 먼저 종료하느냐 서버가 먼저 종료하느냐에 따라서 TDI함수호출이 다르다.

서버가 먼저 소켓을 닫고 그 후에 클라이언트가 소켓을 닫는 경우

## Socket TCP 연산 – 연결종료 2



3.4.1. 서버에서 소켓을 닫았을 때  
**closesocket**(m\_AcceptSocket);

- 클라이언트 측

object	Request	Local	remote	Result
address	TDI_EVENT_DISCONNECT	TCP:100.100.100.103:100		SUCCESS      RELEASE



서버로부터 연결 종료를 TDI\_EVENT\_DISCONNECT 이벤트 핸들러를 통해서 알 수 있다.

- 서버 측

object	request	local	remote	result
Connect 2	TDI_DISCONNECT	TCP:100.100.100.103:100		SUCCESS RELEASE
Connect 2	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.103:100		SUCCESS
Connect 2	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 2	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS

TDI\_DISCONNECT로 클라이언트와 연결을 종료한다.

이때 connect 객체를 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 를 통해서 해제한다.

`closesocket(m_AcceptSocket);`

- 클라이언트 측

반응 없음

- 서버 측

object	request	local	remote	result
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Connect Event: NULL
Connect 3	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.103:100		SUCCESS
Connect 3	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 3	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
Connect 1	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.103:100		SUCCESS
Connect 1	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
Connect 1	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Error Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Disconnect Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS

				Receive Event: NUL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Expedited Receive Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Chained Receive Event: NULL
address	IRP_MJ_CLEANUP	TCP:100.100.100.103:100		SUCCESS

TDI\_DISCONNECT로 클라이언트와 연결을 끊는다.

Listen때 생성된 connect객체들을 connect 객체를 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 를 통해서 해제한다.

Address 객체를 통해서 이벤트 핸들러들을 NULL로 초기화하고 IRP\_MJ\_CLEANUP로 address 객체를 해제한다.

### 3.4.2.- 클라이언트에서 소켓을 닫았을 때

`closesocket(m_ClientSocket);`

- 클라이언트 측

object	Request	local	remote	Result
connect	TDI_DISCONNECT	TCP:100.100.100.103:100	100.100.100.105: 10000	SUCCESS RELEASE
connect	TDI_DISASSOCIATE_ADDRESS	TCP:100.100.100.103:100		SUCCESS
connect	IRP_MJ_CLEANUP	TCP:Connection obj		SUCCESS
connect	IRP_MJ_CLOSE	TCP:Connection obj		SUCCESS
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Error Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Disconnect Event: NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Receive Event: NUL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Expedited Receive Event:

				NULL
address	TDI_SET_EVENT_HANDLER	TCP:100.100.100.103:100		SUCCESS Chained Receive Event: NULL
address	IRP_MJ_CLEANUP	TCP:100.100.100.103:100		SUCCESS

TDI\_DISCONNECT 로 연결을 종료한다.

이때 TDI\_DISASSOCIATE\_ADDRESS , IRP\_MJ\_CLEANUP , IRP\_MJ\_CLOSE 로 connect 객체를 해제한다.

Address 객체를 통해서 이벤트 핸들러들을 NULL로 초기화하고 IRP\_MJ\_CLEANUP로 address 객체를 해제한다.

주목 : address객체를 통해서 이벤트 핸들러 들이 관리가 되고 connect 객체를 통해서 TCP의 연결 관리와 송수신 관리가 되는 것을 알 수 있다.

- 서버 측

반응 없음

## 4. 결론

Winsock 함수 호출에 따른 TDI함수호출에 대해서 알아보았다.

TDI 함수에 대한 자세한 설명은 본 문서에서는 생략했다. 자세한 내용은 MSDN의 ddk문서를 참조하기 바란다.

본 문서는 TCP인 경우에만 Winsock함수호출과 TDI함수호출관계를 알아보았는데 UDP에 대해서도 직접 Socket 프로그램을 작성해서 Winsock함수호출과 TDI함수호출관계를 확인해볼 수 있을 것이다.