Peer간의 커뮤니케이션

p215 WebFuncCall로 WebMethod를 호출한다. num에 가변인자 개수를 넣고 가변인자에는 WebMethod의 인자이름과 값을 넣는다. 그래서 num은 WebMethod의 인자개수 × 2이다.

예) // GetAreaDistance WebMethod를 호출하는데 인자로 index1, index2를
// 값은 str1, str2를 넣는다. Num의 값은 4(인자 2개 × 2)이다.
char* Ret = Web->WebFuncCall("GetAreaDistance", 4, "index1", str1, "index2", str2);

기반기술 위주로 설명하려다 보니 다음 회에서 설명해야 할 내용이 미리 언급되어 다소 이해가 어려운 부분도 있을 것이다. 하지만 여기서 설명한 기술 중 IOCP나 XML 관련 내용은 우리의 P2P 애플리케이션 뿐 아니라 현재 일반적인 P2P 기술 개발에 많이 쓰이는 기본 정책이기도 하다. P2P 관련 프로젝트를 실제 수행하는 독자가 있다면, 이번 내용이 많은 도움이 되었기를 바란다. 다음 회에서는 서버와 클라이언트의 역할에 대해 알아보고, 블로그를 P2P에 접목시켜 본다.

〈리스트 8〉

```
class WebMethod
{
        ...
public:
        ...
        WebMethod(char* _host, char* _address); // 생성자
        char* WebFuncCall(char* FuncName, int num, ...);
// WebMethod 호출
        char* SoapMessage(char* FuncName, const char*
argu); // SoapMessage 생성
        char* Header(char* FuncName, char* StrXML); //
HTTP Header 붙이기
};

// Soap Message 생성
char* WebMethod::SoapMessage(char* FuncName, const char*
argu)
{
        static char str[1024];
        char cr[3] = "\r\n";
        wsprintf(str,
"<?xml version=\"1.0\" encoding=\"utf-8\"?>%s"
"<soap:Envelope xmlns:xsi=\"
http://www.w3.org/2001/XMLSchema-instance\""
"xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\""
" xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/
\">%s"
"<soap:Body>%s"
"<%s xmlns=\"http://tempuri.org/\">%s"          //
FuncName" %s"       // argu_str 인수
"</%s>%s"           // FuncName
"</soap:Body>%s"
"</soap:Envelope>",cr, cr, cr,FuncName, cr,argu,//s1,
s1_value, s1, cr,  FuncName, cr, cr);
        return str;
}

// HTTP Header를 붙인다.
char* WebMethod::Header(char* FuncName, char* StrXML) //
StrXML == ASCII
{
        char str[1024];
        char cr[3] = "\r\n";

        static char utf_str[1024];
        convertCharToUTF8(StrXML, utf_str, 1024); // UTF8
로 변환

        wsprintf(str,
                "POST /%s HTTP/1.1%s"        // full
address
                "Host: %s%s"                          //
host
                "Content-Type: text/xml; charset=utf-
8%s"
                "Content-Length: %d%s"       // xml_str
length
                "SOAPAction:
\"http://tempuri.org/%s\"%s" // FuncName
                "%s"
                "%s" // xml_str
                "%s%s%s",
                address, cr, host, cr, cr,
strlen(utf_str), cr, FuncName, cr, cr,
                StrXML, cr, cr, cr);
        convertCharToUTF8(str, utf_str, 1024);
        return utf_str; // return UTF-8형식
}
```

NDIS의 이해와 활용법

p223 PassThru

PassThru 예제는 DDK가 제공해 주는 예제 프로그램이다. %NTDDK%\src\network \ndis\passthru에서 찾아볼 수 있다. Intermediate 드라이버를 만든다고 하면 보통 이 예제 프로그램을 조금 수정해서 만든다. 이 PassThru 예제는 네트워크 패킷이 Intermediate 드라이버에서 어떤 핸들러(함수)를 통해, Transport 드라이버 혹은 NIC 드라이버로 이동하는지 보여주는 예제다. 하지만, 실제로 패킷에 아무 작업을 하지 않으므로, 필터링이나 모니터링 등의 작업을 하기 위해서는 개

발자가 추가로 코드를 작성해야 한다.

IMSamp

IMSamp는 MSDN에서 제공하는 예제 프로그램이다(DDK에 있는 샘플이 아니다). 이는 http://support.microsoft.com/kb/q178322/에서 다운로드 받을 수 있다. 이 예제 프로그램은 Intermediate 드라이버의 I/O Contorl을 할 수 있는 인터페이스를 제공해 준다. 즉, 어떤 패킷을 필터링 할지 여부를 윈도우 UI에서 결정하고, 그 결과를 드라이버에 넘겨주는 역할을 한다. 기본적으로 드라이버에 값을 입력하는 방식은 파일에 값을 쓰는 것과 동일하다. 다만, DeviceIoContorl이라는 함수를 통해 값을 주고받는 것이 다를 뿐 이다.

NetCfg

드라이버를 설치할 수 있도록 만들어주는 프로그램이다. DDK에서 제공 해주는 예제 프로그램으로, %NTDDK%src\network\config\netcfg에서 찾아볼 수 있다. 드라이버는 다른 프로그램과 달리, Install Shield를 통해 설치되지 않고, 자체 설치 프로그램을 통해 설치된다(물론, Install Shield에서 설치가 가능하다). 이 설치 프로그램을 이용하지 않고, 윈도우가 제공하는 드라이버 설치 UI를 이용해도 상관없다. 또한, 직접 네트워크 카드의 등록정보에서 설치하는 것도 가능하다.

이번 호에서 크게 3 가지를 언급했다. NDIS를 포함한 윈도우 커널의 구조와 네트워크 구조, NDIS 개발에 필요한 도구들, 실제로 만들게 될 프로그램에 대한 소개를 다루었다. 실제 코드를 만지면서 프로그래밍하는 부분이 없어 그다지 어렵지 않았으리라 생각 된다.

필자가 생각하는 연재 목표는 최대한 쉽게 드라이버 프로그래밍에 접근하는 것이다. 처음엔 필자도 경험이 많지 않아, 드라이버 프로그래밍에 접근하기까지 많은 어려움을 겪었다. 영문으로 된 자료는 많지만, 하나하나 쉽게 따라서 할 수 있도록 만들어진 자료들은 많지 않다. 독자들은 본문에 소개한 프로그램을 설치해 윈도우 내부 구조를 실제적으로 익히길 바란다. 다음 호부터 NDIS 드라이버를 만들어 보자.

OOP로 만드는 워드 프로세서 1

p247

완성 및 테스트

이제 폼 위에 TEditor를 동적으로 생성해 올려본다. 물론 키보드도 눌러보자. 텍스트가 입력되는 모습을 볼 수 있다. TEditor의 Font를 세팅해주면, 그 후로는 새로운 폰트의 텍스트가 입력된다. 지금까지 일차적인 워드 프로세서 구현을 마쳤다. 다음 호에서는 여기에 그림과 글 상자를 추가하는 과정을 함께 해본다.

〈리스트 6〉 TEditor의 WM_CHAR 메시지

```
procedure TEditor.WMChar(var Message: TWMChar);
var
  Line: TLine;
  Text: TText;
begin
case Message.CharCode of
//엔터 키가 눌러졌을 때 새 라인을 생성해 추가한다(줄 바꿈).
  VK_RETURN: begin
               Line := TLine.Create( FCurrentSeat );
               Line.AdjustBound;

               Inc( FCurrentSeat.FCurrentLine );
               FCurrentSeat.Insert(
FCurrentSeat.CurrentLine, Line );
               FCurrentSeat.AdjustBound;
             end;
//백스페이스를 눌렀을 때 마지막 캐릭터를 삭제한다.
    VK_BACK: begin
               Line := FCurrentSeat.Lines[
FCurrentSeat.CurrentLine ];
               if Line.CurrentCharacter > 0 then
                begin
                  Line.Delete( Line.CurrentCharacter - 1
);
                  Line.AdjustBound;
                  Dec( Line.FCurrentCharacter );

                  FCurrentSeat.AdjustBound;
                end;
             end;
//기타 키가 입력되었을 때는 TText를 생성해 현재 라인에 추가한다(문자 입력).
        else begin
               Line := FCurrentSeat.Lines[
FCurrentSeat.CurrentLine ];
               Text := TText.Create( Line );
               Text.Text := Char( Message.CharCode );
               Text.AssignFont( Font );
               Text.AdjustBound;

               Line.Insert( Line.CurrentCharacter, Text );
               Line.AdjustBound;
               Inc( Line.FCurrentCharacter );

               FCurrentSeat.AdjustBound;
             end;
  end;

  Repaint;
end;
```

웹 브라우저로 구현하는 '안티 피싱' 테크닉

p287 의심 불가?

얼마 전 하버드대와 버클리대의 정보보호 전문가들이 모여 '피싱이 왜 효과가 있는가' 라는 주제로 연구하고, 이용자 집단의 성향과 피싱 효과를 분석한 논문을 발표한 바 있다. 논문 내용 가운데 가장 흥미 있는 것은 다수의 실험자들에게 피싱 메일을 가려내볼 것을 요구했을 때 90%에 달하는 실험자가 감쪽같이 속았다는 점이다.

전문가들이 실험에 이용한 메일은 BOW(Bank Of the West) 은행의 메일 형식을 정교히 본딴 것으로, 이를 수신하면 피싱 사이트인 'www.bankofthevvest.com'에 접속하게 된다. BOW 은행의 실제 도메인(www.bankofthewest.com)에 있는 'w' 대신 'vv'를 넣어 실험자들의 눈을 속인 셈이다.

실험자가 받아 본 이메일 콘텐츠에는 보안 표시 자물쇠와 가짜 로고, 자격 확인 표시, 소비자 안전 경고 팝업 등이 완비돼 있어 실험자가 이를 의심하기란 불가능했다. 이 실험은 결국 피싱의 수법이 너무나 교묘해, 그에 따른 피해를 고스란히 이용자 부주의로 돌릴 수 없음을 말하려 한 것이다.

커널 내부 접근방법

p305

prev_syscall을 사용해 원래 시스템콜의 주소를 저장하도록 했다. 모듈을 커널에서 제거할 때 저장한 시스템콜을 다시 되돌려 놓는 것을 볼 수 있다.

〈리스트 10〉의 소스를 Makefile을 만들어 컴파일을 한 후에 insmod를 사용해서 커널에 적재하자. 그리고 시스템콜 추가하고 동작확인을 위해 구현한 예제를 실행시키면 'Hello, Kerne' 이 아니라 'Hello, Module' 이 출력된다면 성공이다. rmmod를 사용해 커널에서 제거한 후에 다시 예제를 실행시키면 원래의 'Hello, Kernel' 이 출력될 것이다.

본문에서 언급했듯이 리눅스에서는 대부분의 기능을 모듈로 구현할 수 있다. 그 중 하드웨어를 다루는 기능을 구현한 것이 바로 디바이스 드라이버라고 할 수 있다. 다음 호에서는 커널 내부를 제어하는 방법으로 디바이스 드라이버와 커널에서 메모리를 사용하는 방법을 살펴 볼 것이다.

〈리스트 11〉 리스트 10을 컴파일 하기 위한 Makefile

```
KERNELDIR= /lib/modules/$(shell uname -r)/build
CFLAGS = -D__KERNEL__ -DMODULE -I$(KERNELDIR)/include -O

all : test2.o

clean :
 rm -rf *.o
```

```
〔root@localhost root〕# ./a.out
Hello, Kernel!
〔root@localhost root〕# indMod test2.o
〔root@localhost root〕# lsMod
Module  Size  Used by                    Tainted : p
test2                                    86          0  (unused)
〔root@localhost root〕# .a/.out
Hello, Module!
〔root@localhost root〕# rMMod test2
〔root@localhost root〕# ./a.out
Hellod,Kernel!
〔root@localhost root〕# _
```

〈화면 8〉 모듈의 적재/해제에 따른 시스템콜의 대체/복귀