연재(마지막회) Embedded System에의 RTOS포팅

TCP/IP Network Stack의 탑재와 응용

삼성 소프트웨어 멤버쉽 Digital System Design Lab.
김종수(promise@secsm.org)

최근 Embedded System에서 네트워크 사용이 빈번해지고 있다.
네트워크의 개념과 구성 자체가 계층화돼 있고, 이를 구현하기 위해 복잡한 프로토콜을 사용하는 관계로 시스템 프로그래머가 직접 모든 상황을 고려하는 프로그램 짜기가 수월치 않다.
가장 간단하게 보면 Network Connection을 한 개만 가지도록 하여 해당 프로토콜을 순차적으로 수행하는 방식이 있을 수 있다. 그러나 기본적으로 네트워크에서는 Multi Session이 가능해야 하기 때문에 OS의 힘을 빌려 처리하는 것이 가장 편리한 방법이 될 것이다.
이번 회에서는 네트워크 구성에서 가장 많이 사용되고, 기본이 되는 프로토콜인 TCP/IP를 살펴보자.

TCP/IP

전자, 정보통신 등에 조금이라도 관계된 개발자라면 TCP/IP(Transmission Control Protocol/Internet Protocol)가 친숙한 용어일 것이다. 가장 보편화된 망들에서(Ethernet 등) 사용되는 기본 프로토콜이므로 가장 부각되는 것이 당연하다.

사실 이 TCP/IP 프로토콜을 제대로 이해하기 위해서는 두꺼운 책 한 권쯤 필요할 것이다. 그러나 이 연재에서는 이를 자세히 설명할 수 있는 지면이 없으므로 개념정도와 구현에 초점을 맞춰 설명할 것이다.

네트워크 관련 지식을 습득하려 할 때 가장 먼저 접하게 되는 것이 OSI 7 Layer라는 것이 있다. 네트워크 시스템을 특징별로 구분하여 계층을 정의한 것이다. 사실 프로그램의 입장에서 보면 계층화의 의미가 그리 중요시되지 않을 수 있다. 계층화된 프로그램이 필요한 이유는 하위 프로토콜보다 상대적으로 개수가 많은 상위 프로토콜들이 공통적인 방법으로 하위 프로토콜을 사용할 수 있도록 하기 위해서이다. 편한 프로그램 작성 방법이라고 생각하면 된다. 사실 이 편한 프로그램 기법이 가장 적절하면서도 효율적이기 때문에 계층화를 프로그램에 도입하는 것이기도 하다.

Application	Telnet, FTP, e-mail
Transport	TCP, UDP
Network	IP, ICMP, IGMP
Link	Device driver & card

그림 1. TCP/IP 프로토콜 4 계층도

위의 그림에서 보면 프로토콜 측에서 바라본 OSI 7 Layer의 특징에 따른 관계를 나타내고 있다. Physical Layer는

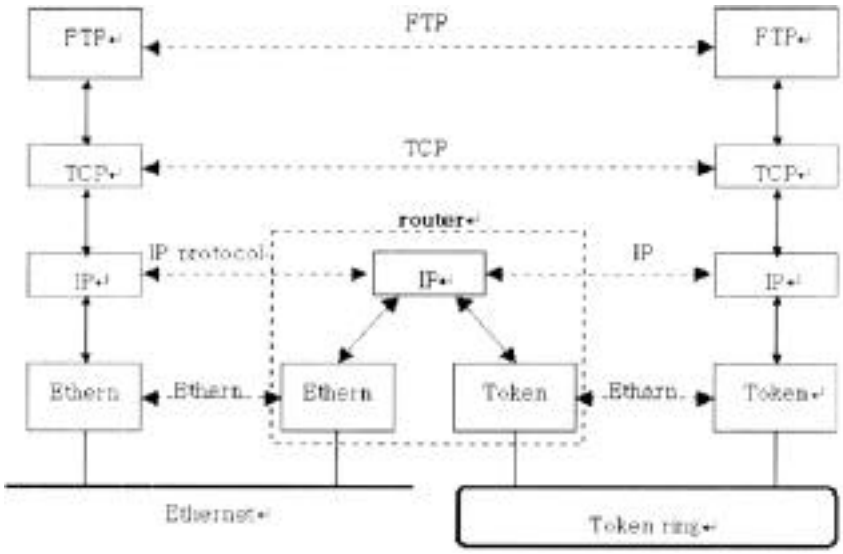

그림 2. TCP/IP 프로토콜 계층화 예제

인터페이스를 구성하는 하드웨어에 관계된 사항이므로 고려에서 제외하였다. 중요 프로토콜들의 엮이는 관계는 이런 Layer 개념을 이용한 것이다. 각각의 layer는 다음과 같은 역할을 담당한다.

· Link layer : OS에서의 Device driver에 해당하는 사항으로 하드웨어에 따라 크게 변하는 부분이다.

· Network layer : 실제 네크워크 상의 packet의 움직임을 다룬다. Packet routing같은 일을 수행한다.

· Transport layer : 상위에 있는 Application layer를 위해 두 host사이의 data flow를 제공한다. TCP는 두 host사이의 Reliable flow를 제공하고, UDP는 보다 간단한 서비스를 제공한다.

· Application layer : 특정 애플리케이션에 대한 서비스를 제공한다.

위의 그림에서와 같이 FTP(File Transfer Protocol)를 사용하여 파일을 전송하는 예를 보자. FTP 응용 프로그램에서 전송할 파일을 선택해서 상대에게 전송하려 할 때 거치는 프로토콜은 다음과 같다. 전송할 파일의 정보가 FTP 프로토콜에 싸여 TCP로 내려오면 TCP의 부가 정보를 붙여 IP로 내려보내고, 다시 IP는 IP의 부가정보를 씌우고, Ethernet Packet 부가 정보를 붙여 라인을 통해 신호로 내보낸다. 이 그림에서는 망의 형태가 다른 곳으로 Packet을 전송하는데 이의 경우 경로 설정을 담당하는 라우터에 의해 Packet이 전송되기 때문에 이의 고려는 안해도 되며, 이 후에는 반대 방식으로 부가정보를 보아 각 계층에서 데이터 처리를 하고 정보를 상위로 올려 보낸다. 마지막에 FTP 응용 프로그램은 순수 파일 데이터를 전달해 주는 것으로 작업은 끝난다.

전에 필자가 받은 선물 중에 재미있는 것이 있었는데, 조그만 상자 속에 또 작은 상자가 있고, 안에 또 상자, 또 상자가 있더니 마지막에 초콜릿 한 조각이 있는 것이었다. 무척이나 황당했다.

이것은 네트워크에서 Packet을 전달하는 방식과 유사하다. 초콜릿은 앞의 예에서 전송하려는 파일과 같다. 선물을 해준 사람은 초콜릿이라는 내용을 상자로 싸고, 싸고 했을 것이고, 이는 전송 측에서 하는 일과 같다. 소포로 배달되는 것은 물리 망에 해당하는 것인데 나에게까지 잘 전달되는 훌륭한 우편 주소 시스템과 이를 관장하는 우체국은 MAC Address or IP Address, Router에 비유될 수 있다. 상자를 풀고 또 푸는 필자는 수신측에 해당하며 상자를

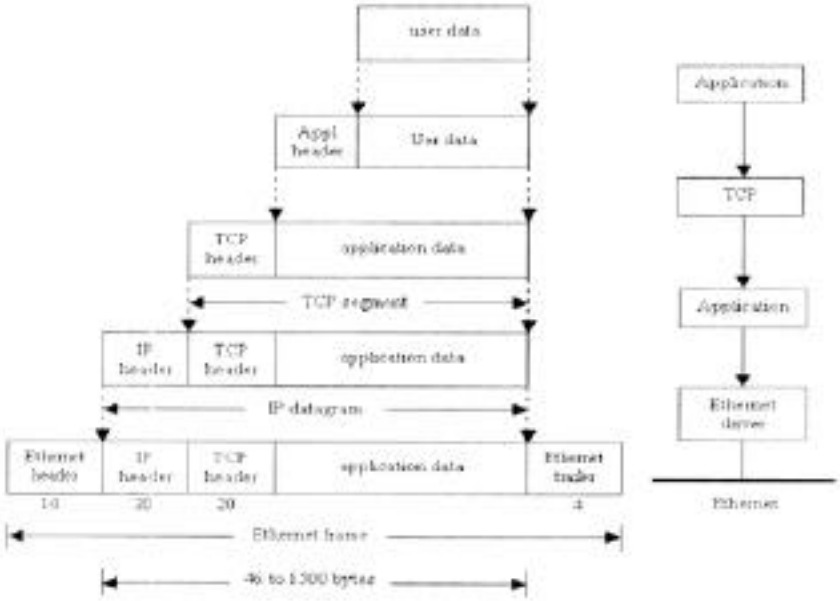

그림 3. Ethernet Frame 계층 구조

하나씩 풀 때마다 프로토콜의 계층이 하나씩 높아지는 것과 같다. 마지막 상자는 애플리케이션 계층이 사용하는 프로토콜과 같으며 초콜릿을 발견하는 순간 전송의 모든 작업은 끝나게 된다.

위의 작업에서 보듯이 정말 귀찮은 작업들의 연속이다. 상자를 겹겹이 포장하는 것과 같이 많은 노력(Overhead)이 드는 게 사실이다. 그러나 정성이 많이 느껴지는 것과 같이 네트워크에서도 많은 이점과 안정성을 보장받게 된다. 또한 각 상자에 해당하는 프로토콜들은 저마다 가지고 있는 Data Flow 제공 등의 작업을 수행함으로써 독특한 제 역할을 수행하게 된다. 나중에 우리는 위의 계층도에서 중간 단계에 있는 2번째, 3번째 상자인 TCP와 IP를 구현하는 것을 살펴보도록 할 것이다.

Target System 구성

먼저 Target 시스템을 구성하여 보자. Target 시스템은 연재 4회에서 다루었던 MC68302시스템을 이용하도록 할 것이다.

68K Series 중에는 IEEE 802.3을 CPU의 한 Peripheral Module로 만들어 내장함으로써 Ethernet 구성을 더욱 쉽게 할 수 있는 MC68EN302나 MC68EN360(CPU32 core) 등이 있다. 그러나 쉽게 구해 사용할 수 없는 것들이 많고, 일반적인 Processor들이 Ethernet 전용으로 구성되어 나오지는 않으므로 좀더 일반적인 경우를 고려하여 설계하였다.

시스템은 MC68302에 Ethernet Controller, RealTek 사의 RTL 8019를 사용하여 설계하였다. 이러한 Ethernet Controller를 사용하면 8051과 같은 Processor들에도 적용이 가능할 것이다.

MC68302의 메인 시스템에 대해서는 구성 설계 등을 4회에서 다루었으므로 여기서는 Ethernet controller 부분만 설명을 할 것이다.

그림 4에서는 Ethernet 시스템을 구현하기 위한 일반적인 물리 계층 설계 모습을 나타내고 있다.

일반 프로세서에 Ethernet을 가능하도록 하기 위해서

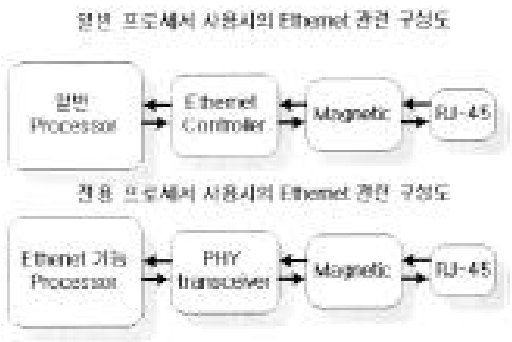

그림 4. Ethernet 구성도

는 프로세서의 BUS에 Ethernet controller의 Address, Data BUS를 연결하고 제어 신호선들을 접속해야 한다. 프로세서의 입장에서 보면 이 BUS를 통하여 DATA의 전송과 수신이 일어나는 것은 일반 외부 디바이스를 접속했을 때와 전혀 다를 바가 없다. 이 Ethernet controller에서는 입, 출력되는 데이터를 Ethernet Frame 형태로 변형시킨다거나, 전송을 위한 데이터 변형 등의 Transceiver 기능, 물리 라인의 상태에 따른 대응, 확인용 LED의 점 등들의 일을 모두 처리한다.

반면 Ethernet 모듈이 내장된 프로세서의 경우 프로세서 내부에 이미 MAC layer에서 처리해야 할 일들을 하고 있으므로 PHY Transceiver는 데이터 변형과 송수신 관련 기능만을 이용하고 있다. 보통 많이 사용하는 10 BASE-T혹은 10/100 BASE-T의 경우 주요 데이터 전송 신호선은 TX+, TX-, RX+, RX- 이며, 여기에 제어 신호를 사용하게 된다.

최근 출시되는 Ethernet 가능 프로세서들 중에 Ethernet의 성능을 중시하고, 구성될 시스템에서의 호환성을 위해 10/100 BASE T를 사용할 수 있도록 IEEE 802.3u를 사용하기도 하는데, 이 때 프로세서와 PHY Transceiver 사이의 인터페이스로 주로 사용되는 것이 7-wire, MII(Media Independent Interface)들이다. CPU의 Ethernet처리 속도(10 Base-T, 10/100 Base-T)에 따라 뒷단의 부품 선정들도 따라서 달라지므로 유의해야 한다. Magnetic 혹은 Transformer라고 불리는 부

10/100Base-T? ??		
Pin	Name	Description
1	TX+	Tranceive Data+
2	TX-	Tranceive Data-
3	RX+	Receive Data+
4	n/c	Not connected
5	n/c	Not connected
6	RX-	Receive Data-
7	n/c	Not connected
8	n/c	Not connected

그림 5. 10, 10/100 Base-T 신호선

품은 Ethernet controller / PHY Transceiver와 RJ-45 connector 사이에서 물리 신호의 완충 작용을 한다. 실제 내부에는 코일이 있어 변압기와 같은 구조를 가지고 있어 외부 라인과 시스템 사이를 분리시키고 있다. 보통 Ethernet, Magnetic관련 칩을 생산하는 회사들은 Spec.에 자사의 제품과 연계 사용 가능한 목록들을 잘 정리해 놓는 편이므로 이를 활용하기 바란다. 일부 제품들은 Magnetic과 RJ-45를 결합하여 만들어진 것도 있다.

RJ-45를 통한 Ethernet 망 접속 신호는 다음과 같다.

이제 일반 프로세서와 Ethernet 부분의 연결을 실제로 구성하여보자.

회로도상의 RTL8019AS의 상위 왼쪽부터 살펴보면 5bit의 Address, 8bit Data 선이 System BUS에 연결되어 있으며, 프로세서와의 /CS(Chip Select), /RD(Read), R/W(Write), Reset 등의 친숙한 신호선 접속을 보이고 있다. INT의 경우 통신 장애, Data 수신 등의 이벤트 발생시 프로세서로 Interrupt를 보낸 후에 발생한 상황에 대한 정보를 보내게 된다.

Ethernet Controller나 PHY Transceiver 들은 동작 클록 및 Ethernet Data 송수신을 위해 20MHz(10 Base-T), 25MHz(10/100 Base-T)의 Oscillator 혹은 X-Tal을 사용한다.

RTL8019AS는 10 Base-T용이므로 역시 20Mhz의

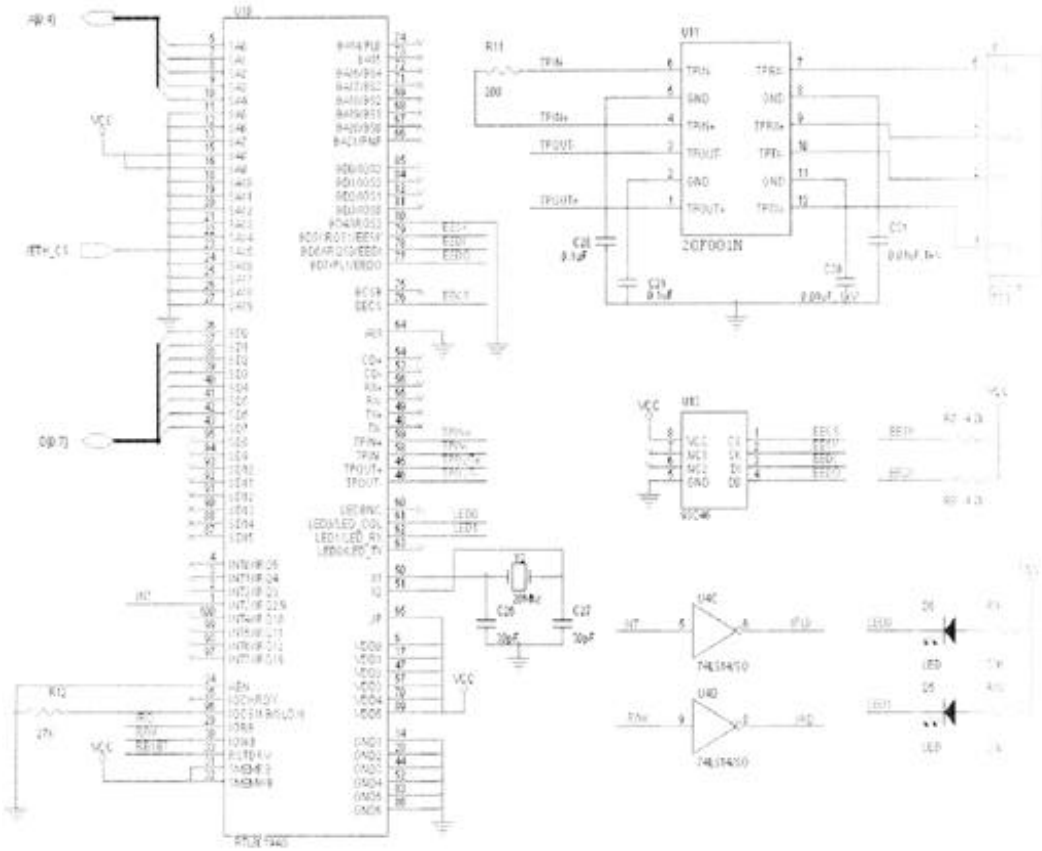

그림 6. Ethernet 관련 회로도

X-Tal을 사용하였다. 회로에서 보이는 LED는 Ethernet controller의 상태를 표시하기 위한 것으로 대부분의 Ethernet controller나 PHY Transceiver에는 이러한 기능을 내장하고 있다.

TPIN+, TPIN-, TPOUT+, TPOUT-는 실제로 네트워크를 통해 들어오거나 나가는 데이터들의 실제 신호들이다. 이 신호선들이 Magnetic을 통해 변화되었겠지만, 실제로는 Magnetic과 같은 신호라고 보아도 무방할 것이다. Magnetic에 대한 설명은 생략하겠다.

Magnetic과 연결된 RJ-45는 다음 그림과 같은 일반적인 형태이다. 단 Female용 소켓은 종류에 따라 접속 번호 순서가 다르므로 주의하여야 한다.

마지막으로 바로 위의 EECS, EESK, EEDI, EEDO의 신호 선들은 Serial EEPROM과의 인터페이스를 위해 사용된다. Ethernet 장치들은 고유의 MAC ADDRESS 정보를 보유하고 있어야 하므로 이를 위한 배려라고 할 수 있다. 회로에서 사용된 EEPROM은 ATMEL사의 작은

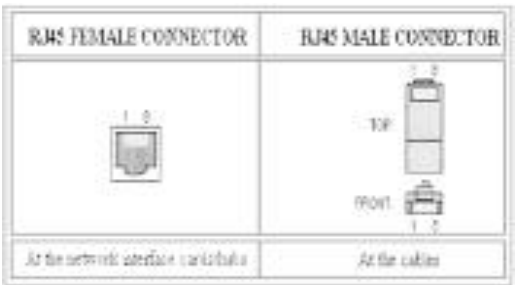

그림 7. RJ-45 Connector

사이즈의 제품으로 Chip Select, 데이터 전송 속도를 위한 클록, Data Input/ Output 등의 신호선을 가지고 있다. 이제 Ethernet 관련 부분의 하드웨어 제작에 대해서 설명을 마쳤으므로 다음에는 탑재할 TCP/IP Software에 대해 설명하도록 하겠다.

TCP/IP STACK

이번 시스템에서 사용할 TCP/IP Stack은 TinyTCP/IP라고 이름 붙여진 소스를 사용하여 만들어 질 것이다. TinyTCP의 저자는 Geoffrey H. Cooper이며, 코드 사용에 대한 제약이 없다. 이 코드는 실제로 TCP/IP의 접속을 순차적이면서 간단하게 구성하고 있다. 코드의 크기가 작아 보통의 작은 시스템에서 사용할 수 있는 크기이다. 그러나 제목 그대로 Tiny한 코드를 가지고 있으므로 많은 예외 상황에 대해서는 대응 처리를 하지 못한다. 이를 이번 Protocol Stack 탑재용으로 선정한 이유는 코드의 크기가 작고, TCP/IP에 대한 기본적인 사항만을 다루고 있어서 최소 기능 요소만을 갖고 있는 uC/OS와 이용 성격이 같은 곳에 사용할 수 있을 것이기 때문이다.

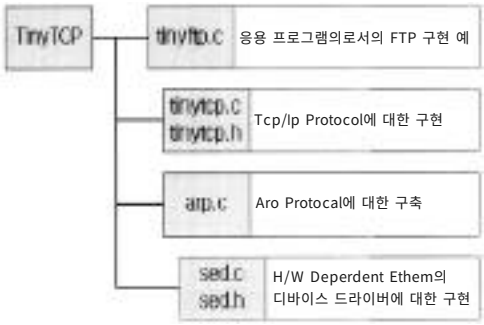

그림 8. TinyTCP 파일 구조

TinyTCP의 코드 내용을 살펴보면 크게 ARP(Address Resolution Protocol) 관련 코드, TCP/IP관련 코드, 이의 응용을 위한 간단한 FTP 코드들로 되어 있다. 파일의 구조는 다음과 같다.

ARP 프로토콜을 TCP/IP의 구현에서 사용하여야 하는 이유는 32bit Internet Address를 48bit Ethernet

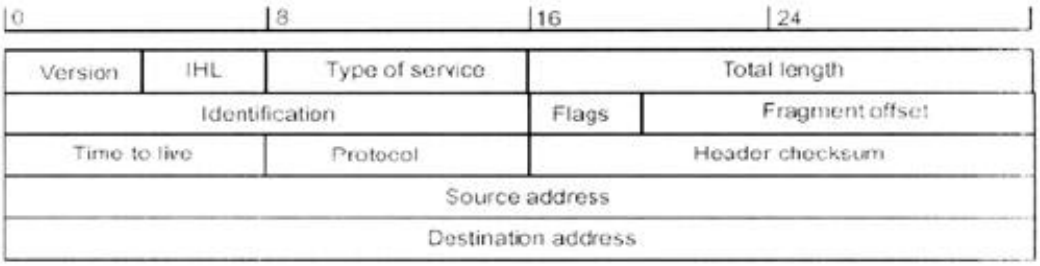

그림 9. IP Packet Header구조

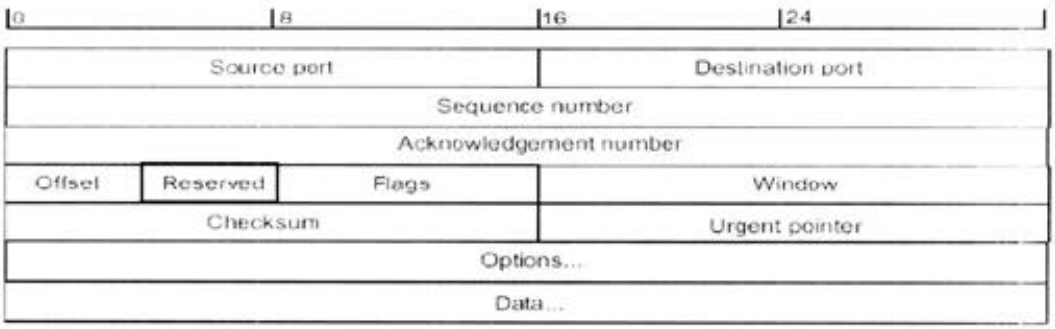

그림 10. TCP Packet 구조

Address(MAC Address)로 바꾸어야 하기 때문이다.

인터넷 망 개념에서는 흔히 IP라고 하는 xxx.xxx.xxx.xxx(예 192.168.123.23 등) 형태의 주소는 MAC Layer에서는 하드웨어마다 할당된 yy:yy:yy:yy:yy:yy(05:12:3A:34:4B:F6 등) 형태의 Ethernet 주소로써 전달되어지므로 원하는 목적지와 보내는 곳과의 맵핑을 위해 사전 작업 차원에서 필요한 정보를 확립하는 프로토콜로 사용된다.

Sed.c, Sed.h 파일은 RealTak에 맞춰 제 기능을 할 수 있도록 수정을 하여야 하는 부분이다. 기본 골격만으로 이루어진 이 파일에 적절한 살을 붙이는 것은 개발자의 몫이 된다. 한편 FTP를 구현한 파일은 일종의 테스트 파일이므로 사용은 자유이다.

전체 Ethernet 파일 구조에 대해 설명하였다. 염두에 두어야 할 사항은 동작 시에 다음과 같은 제약사항을 지켜야 한다.

· Timer를 필요로 한다.

· 순서를 벗어난 segment는 무조건 무시한다.

· 상대의 요청 Window Size는 무조건 무시하며, 우리의 Window Size는 1024로 고정했다.

· Urgent Pointer등의 Option은 전혀 지원하지 않으며, 최대 segment 크기를 1500으로 제한했다.

· Big-Endian으로 작성되었기 때문에, x86 같은 Little-Endian에서 사용하기 위해서는 수정이 필요하다.

이보다 우수한 TCP/IP Stack Source를 사용할 수 있다면 그것을 적극 권장한다. uC/OS 사이트에 있는 TCP/IP 관련 소스들도 모든 것을 다 지원하지는 않는다. 요즘에는 TCP/IP Stack이 많이 공개되어 있으므로 이를 잘 활용하기 바란다.

1. TCP/IP Header 및 Socket 구조

TCP/IP Packet을 전송하기 위한 사전 준비 작업으로써 TCP/IP Packet 구조를 맞추는 작업을 하여보자. 그림 9, 그림 10에 사용할 Packet의 구조를 보이고 있다. 이러한 형태의 Packet을 전송해야만 상대측에서도 인식할 수 있다는 이야기이다. 통신에 가장 중요한 것은 약속이기 때문에 이를 꼭 지켜줘야 한다. 약속을 안 지켜도 상관은 없지만 후에 완전 따가 되는 것은 이쪽 세상이나 네트워크 세상에서나 마찬가지 인 것 같다.

```
typedef struct {
    word            vhl;
    word            length;
    word            identification;
    word            frag;
    word            ttlProtocol;
    word            checksum;
    in_HwAddress    source;
    in_HwAddress    destination;
} in_Header;

#define in_GetVersion(ip) ((ip)->vhl >> 12) & 0xf)
#define in_GetHdrLen(ip)  ((ip)->vhl >> 8) & 0xf)
#define in_GetHdrLenBytes(ip)  (((ip)->vhl >> 6) & 0x3c)
#define in_GetTos(ip)     ((ip)->vhl & 0xff)
#define in_GetTTL(ip)     ((ip)->ttlProtocol >> 8)
#define in_GetProtocol(ip) ((ip)->ttlProtocol & 0xff)
```

코드 1. IP Header Structure

```
typedef struct {
    word            srcPort;
    word            dstPort;
    longword        seqnum;
    longword        acknum;
    word            flags;
    word            window;
    word            checksum;
    word            urgentPointer;
} tcp_Header;
```

코드 2. TCP Header Structure

위의 그림 9의 IP 구조에서 Header영역의 정보를 담기 위한 프로그램상의 Structure 구조는 위와 같다. 각 영역에 대한 쓰임새 정보는 지면상 설명할 분량이 아니므로 관계 도서를 참조하기 바란다.

마찬가지로 그림 10의 TCP 구조에서 Header에 대한 Structure 구조는 위와 같다.

한편 TCP Packet이 만들어지고, IP Packet의 Payload로써 TCP Packet이 들어가 IP layer 까지 내려왔다면, 이후에는 Ethernet header를 붙여서 Packet을

```
typedef longword in_HwAddress;
typedef word eth_HwAddress[3];

typedef struct {
    eth_HwAddress   destination;
    eth_HwAddress   source;
    word            type;
} eth_Header;
```

코드 3. Ethernet Header Structure

```
typedef struct _tcp_socket {
    struct _tcp_socket *next;
    short           state;
    procref         dataHandler;
    eth_HwAddress   hisethaddr;
    in_HwAddress    hisaddr;
    word            myport, hisport;
    longword        acknum, seqnum;
    int             timeout;
    BOOL            unhappy;
    word            flags;
    short           dataSize;
    byte            data[tcp_MaxData];
} tcp_Socket;
```

코드 4. Socket Structure

망에 내보내야 할 것이다. 이때 필요한 Ethernet Header로써 6Byte씩의 Source, Destination MAC Address와 2Byte의 Type/Length정보가 필요할 것이다. 이의 Structure 구조도 아래 기술되어 있다.

이번에는 Socket의 구조에 대해 살펴보자. Socket이란 S/W입장에서 전송하고자 하는 데이터의 구조를 압축 표현한 것으로 볼 수 있다. 실제 하위 계층에서는 이 Socket의 정보를 이용하여 자신의 Layer에 알맞은 형태로 변형하는 과정을 수행하게 된다.

Socket 구조를 간단히 살펴보면 다음과 같다. State는 connection 상태를 나타내는 변수로 TCP/IP 흐름의 한 위치에서의 정보를 나타내게 된다. DataHandler는 데이터 입력시 처리를 위한 루틴을 의미하며, hisethaddr, hisaddr는 상대측의 MAC, IP Address를, myport, hisport는 Connection시에 필요한 Port를 나타낸다. Acknum, seqnum은 Data의 전송 번호를, timeout, unhappy는 ms단위의 전송 제한 시간과 재전송에 대한 정보를 제공한다. Flag, dataSize, Data는 각각 전송시의 마지막 flag 정보, 전송 정보의 크기와 실제 데이터를 의미하는 것이다.

2. TCP/IP 함수 및 구조

이제 TinyTCP.C 파일에 있는 함수들의 상관 관계를 살펴보자.

TCP/IP의 함수가 이정도로 작다는 것을 보면 정말 Tiny인 것만은 사실인 것 같다. 그림 11에서 함수의 관계가 일견 복잡해 보이지만, 사실 의미상으로 이해하면 어렵지 않다. 참고로 전 함수의 실제 코드를 일일이 다 분석하지는 않을 것이다. 전체적인 흐름을 이해하는 것이 더욱 중요하다.

사용되는 함수의 상관 관계는 그림 11에서와 같이 정리된다.(참고로 함수명 중에 sed는 Simple Ethernet Driver를, sar는 Simple Address Resolution을 뜻한다.) 앞으로 나오는 함수에 대한 설명을 참조하기 바란다. 함수의 이름을 보고 직관적으로 함수의 기능을 유추하면 훨씬 이해가 쉽다.

일단 각각 함수의 기능을 살펴보고 난 후 이들의 함수 상관 관계를 예를 들어 설명할 것이다.

(1) Application & Socket Layer

아래 함수들은 실제 Application에서 사용할 수 있는 실제 함수라고 할 수 있다.

· tcp(procref application) : TCP를 위한“ busy-wait ”loop 방식의 루틴을 호출한다. 실제적인 TCP/IP의 접속의 시작이라고 할 수 있다. (TinyTCP에서는 인터럽트가 발생할 때마다 packet을 처리하는 방식이 아니라, 계속 packet이 있는지 확인하는 방식으로 구현되었다. 즉, packet이 있으면 이을 처리하고(tcp_Handler(), sar_checkPacket()), 없으면 재전송 루틴(tcp_Retransmitter())과 사용자가 만든 application()을 호출하는 형식이다.)

· tcp_Write(tcp_Socket *s, byte *dp, int len) :

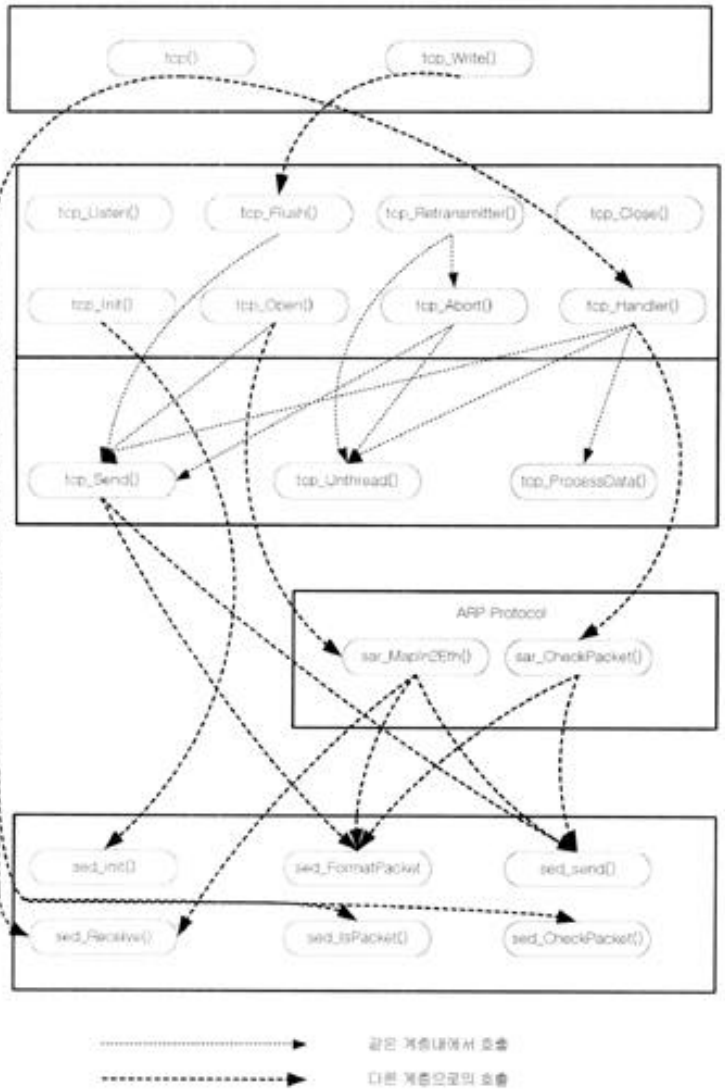

그림 11. Tiny TCP 함수 구조

data를 보내는 루틴이다. 상태는 꼭 ESTABLISHED 상태이어야 하며, tcp_Flush()를 호출하여 보낸다. 앞에서도 언급했듯이 모든 접속 상황을 고려한 예외처리 하나하나가 코드를 크게 만들기 때문에 이를 하지 않는다.

(2) Protocol Layer(상위 계층에서 호출)

실제적인 접속, 절단, 전송, 재전송 등의 프로토콜 과정을 보이는 부분이다.

· tcp_Listen() : Passive Open. 상태를 LISTEN으로 한다.

· tcp_Flush(tcp_Socket *s) : Flag, PUSH를 세트하여, tcp_Send()를 호출한다.

· tcp_Retransmitter() : 주기적으로 호출되어야 하

는 것으로 재전송을 필요로 하는 Scoket을 검색하여, 전송한다.

· tcp_Close() : 상태만 FIN_WAIT_1으로 바꾸며, tcp_send()는 부르지는 않는다.

· tcp_Init() : sed_Init()함수를 호출하고, socket 초기화와 Ethernet address를 설정한다.

· tcp_Open() : Active open. ARP를 요청하여, Ethernet address를 얻은 후, SYN을 세트하여, Packet을 송신한 후 상태를 SYN_SENT로 한다.

· tcp_Abort() : Abort connection. LISTEN, CLOSED 상태가 아니면 RST 비트를 세트시켜 송신한다. socket을 socket list에서 제거한다.

· tcp_Handler() : 우선 packet의 목적지 socket을 찾아서 (demux) 처리를 시작한다. 만약 해당 packet에 대응하는 socket이 없으면 packet을 버린다. RST 비트를 우선적으로 처리한 후, 각 socket의 상태(LISTEN, SYNSENT, SYNREC, ESTAB, FINWT1, FINWT2, CLOSING, LASTACK)에 적합한 처리를 한다.

이 TinyTCP에서는 받은 데이터는 따로 저장되지 않는다. Tcp_Handler() 에서 데이터가 있으면, tcp_ProcessData를 호출하고, tcp_ProcessData()에서 Packet의 data의 첫 포인터를 사용자 정의 함수(dataHandler)에 넘긴다. 데이터는 Ethernet 보드의 버퍼에 저장되어 있으며, tcp loop의 맨 끝에서 버퍼의 내용을 삭제한다.

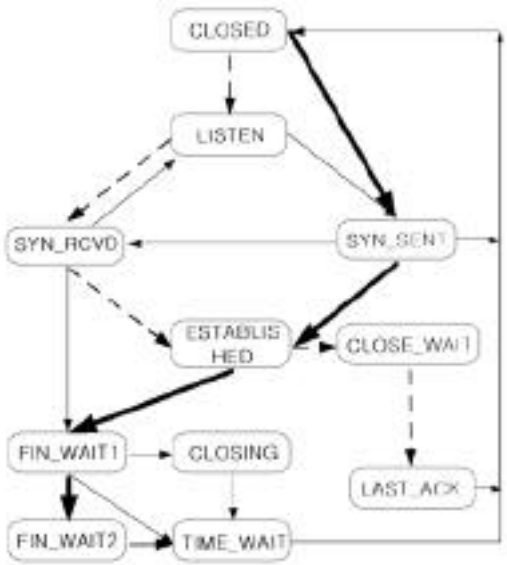

그림 12. TCP State 전이 구조

그림 12에서는 TCP의 상태 전이를 나타낸 것으로 TCP/IP를 서술한 책 중 유명하다고 할 수 있는 W. Richard Stevens의 TCP/IP IIUSTRATED, Volume I의 TCP state transition diagram과 같다. 사실 TinyTCP 코드에서 상태 전이를 나타내고 있는 State는 이를 기본으로 하고 있다고 할 수 있을 만큼 잘 들어맞는다.

약간의 설명을 곁들이자면, 앞에서 정의된 함수들은 이 상태 전이를 조정하는 기능을 하고 있다. 그림에서 점선으로 표시된 상태 전이는 Server 측(접속 요구를 받는 측)을, 굵은 실선은 Client 측(접속 요구를 하는 측)을 나타낸다.

이 시스템이 Server인 경우 (점선을 따라가 본다)초기화 이후 LISTEN 상태에서 접속신호를 기다리게 된다. 이후 Client 측에서의 접속 요구(SYN)가 오면 이에 대한 응답(ACK)과 SYN를 주며 SYN_RCVD 상태로 전이한다. 다시 이에 대한 Client측에서의 SYN 신호가 온다면 ESTABLISHED 상태로 전이하며, 이 후로는 실제 데이터 송 수신이 이루어지게 된다.

반대로 Client라면 (굵은 실선을 따라가 본다)SYN 신호로써 Server측에 접속 신호를 보내고 SYN_SENT 상태로 전이한다. 이 후 서버 측에서의 응답을 주면 다시 SYN 신호와 함께 ESTABLISHED 상태로 전이한다. 이 후는 Server와 같다.

tcp_Listen(), tcp_Open(), tcp_Handler() 등등의 함수들이 이러한 접속을 관계를 형성한다. 이때 필요한 데이터 전송, 수신 데이터 처리를 담당하는 것은 아래 (3)에서 정의되고 있는 함수들을 호출하여 이루어진다.

(3) Protocol Layer (상위계층 호출 않음)

· tcp_Send(tcp_Socket *s) : Socket의 데이터를 전송하는 실제 함수로 sed_FormatPacket()을 호출하여, 버퍼에 Ethernet Packet 형태를 만든다. 그리고,

sed_Send()를 호출하여, 전송한다.

· tcp_Unthread(tcp_Socket *ds) : 원하는 Socket을 Socket list에서 제거한다.

· tcp_ProcessData(tcp_Socket *s, tcp_Header *tp, int len) : 들어온 packet의 tcp 부분의 데이터를 처리하는 루틴이다. len은 tcp packet의 길이이다. socket에서 유지하는 acknum와 들어온 packet의 seqnum를 비교하여, 처리하며, 재전송 packet은 처리하지만, 순서가 다른 packet은 무조건 버린다. seq. no와 ack. no를 계산하여, 처리해야 할 데이터만 socket에 정의되어 있는 dataHandler()를 호출하여 처리하도록 한다. 그리고, FIN이 세트되어 있는 경우, CLOSE 상태(LASTACK, CLOSING, TIMEWT)로 전환한다. 받은 packet은 무조건 ACK를 위해 tcp_Send()를 호출한다.

(4) Protocol Layer (ARP)

앞에서 언급한데로 상대측과의 통신을 위해서는 전화 번호와 같은 상대의 연결 정보를 알아내야 하며, 자신의 연결 정보도 알려줘야 한다. 이를 위해 TinyTCP에서는 ARP 관련 2개의 함수를 제공하고 있다. 실제로 이 코드에서 이 함수를 사용할 때는 접속을 시도할 때이다. 접속할 IP Address는 이미 정해져 있을 것이므로, ARP를 통해 Network에 Broadcast로 이 IP Address를 가지고 있는 사람의 MAC Address를 주세요 라는 요청신호를 보낸다. 그러면 해당 IP Address 정보를 가지고 있는 측에서는 이에 대해 응답을 줄 것이다. 이것이 ARP/RARP의 관계를 이용한 주소 찾기이다. 이 때 사용되는 메시지의 Structure는 다음과 같다.

```
typedef struct {
    word            hwType;
    word            protType;
    word            hwProtAddrLen;
    word            opcode;
    eth_HwAddress   srcEthAddr;
    in_HwAddress    srcIPAddr;
    eth_HwAddress   dstEthAddr;
    in_HwAddress    dstIPAddr;
} arp_Header;
```

코드 5. ARP Header

다음이 이 구조를 이용하는 함수들이다.

· sar_MapIn2Eth() : TCP loop에서 호출되는 ARP Packet Handler 루틴이다. 현재 단순히 ARP REQUEST packet에 대한 처리만을 지원하며, 그 외의 packet은 처리하지 않는다. ARP REQUEST의 경우, ARP REPLY packet을 만들어 보낸다. (sed_Send())

· sar_CheckPacket() : tcp_open에서 호출되는 루틴으로, 상대의 hardware Address(MAC Address)를 얻기 위해 ARP_REQUEST packet을 보내고, ARP REPLY를 기다린다. ARP REPLY를 받기 전까지 블록되며, timeout이 되거나, REPLY packet을 받으면 리턴한다.

(5) Interface Layer

이 부분에 관계된 함수들은 H/W Dependent한 부분으로 현 시스템에서는 RealTek 8019AS에 관계하여 작성하여야 한다. 이 부분에 대한 함수 기능은 다음과 같다.

· sed_init() : Etherent interface를 초기화하는 함수. tcp_init()에서 호출한다.

· sed_FormatPacket(octet *destEAddr, int ethType) : 송신 버퍼에 Ethernet header를 destEAddr, ethType으로 생성한다. 이 함수를 통해 Ethernet Frame 형태가 갖춰진다.

· sed_Send(int pkLengthInOctets) : 송신 버퍼에 있는 packet을 보낸다.

· sed_Receive(octet *recBufLocation) : 수신 버퍼를 Enable하는 함수. 포함된 Ethernet driver인 sed.c에서 사용한 3COM Lan Card는 packet을 받으면, 자동적으로 해당 버퍼가 disable됨으로써, overrun을 막도록 했다. 따라서 packet을 받은 후, 다시 받기 위해서는 Enable 해야 한다. 그러나 현재 적용된 시스템에서는 사용하지 않는다.

· sed_IsPacket(void) : 수신 버퍼에 packet이 있는지 조사하여, 있으면 packet 내의 datagram의 첫 포인터를 반환한다.

· sed_CheckPacket(word *recBufLocation, word expectedType) : 수신 버퍼에 받은 packet의 type을 검

사하는 것으로 recBufLocation은 알아보고자 하는 수신 버퍼의 포인터이고, packet의 type이 expectedType와 같은지를 비교한다.

위의 함수들은 시스템 환경에 따라 달라져야 하는 함수들이므로 기본적으로 PC용 3COM LAN Card 형식으로 되어 있는 드라이버를 변경해서 사용하기 바란다. 이 디바이스의 드라이버에 대한 좋은 정보는 Linux Source를 참조하면 쉽게 구현할 수 있다.

Linux Kernel에서 Device Driver 관련 Source 중에(NE2000 Driver) 이를 참고할 수 있는 부분이 많이 있다. (참고로 위의 것은 Interrupt Handling 방식을 사용했으며, 들어온 packet은 EthRxBuf에 저장하도록 하고, 나가는 packet은 EthTxBuf에 저장하도록 한다. 작성된 함수의 원형은 tinytcp에서 불려지는 형태로 작성되어야 할 것이다.)

(6) 기타 함수

지금까지의 함수들 이외에 사용되는 함수들은 다음과 같다. 이들은 프로그램 내에서 필요한 부가 함수들이다.

- checksum() : 체크섬 계산
- tcp_DumpHeader() : tcp 정보 출력
- Move(byte *src, byte *dest, int numbytes) : src dest (numbytes만큼)
- clock_ValueRough(void) : timeout을 위한 timer function으로 단순히 ms 단위의 Clock을 반환한다. 꼭 정확할 필요는 없다. 몇 개의 clock을 세지 못한다고 해서, 문제가 되지 않는다. 단순히 clock 값을 증가시키고만 있으면 된다.

위에 설명된 각 함수들은 결국 위의 TCP() 루틴이 원활히 진행되도록 하기 위해 필요한 것들이었으며, 이를 상위 프로그램에서 사용하게 되면 목적하고자 한 TCP/IP Stack이 탑재된 것이다.

이제까지 TCP/IP Stack의 H/W, S/W구성 요소들을 갖추는 작업을 해왔다. 이제 이들을 사용하여 실험을 해 보도록 하자.

uC/OS와의 연계

모든 시스템이 마찬가지겠지만 Embedded System에서는 특히 생산가와 시간에 구애를 많이 받으며, 기능 또한 전용 장비에 사용되므로 무시할 수 없는 요인이 된다.

전에 필자가 Linux를 Embedded System의 OS로 사용하여 네트워크 장비를 개발한 적이 있었다. Linux 정도의 OS Kernel이 올라가 작동할 정도이었으므로 Embedded System 치고는 중간 규모 이상인 Flash,

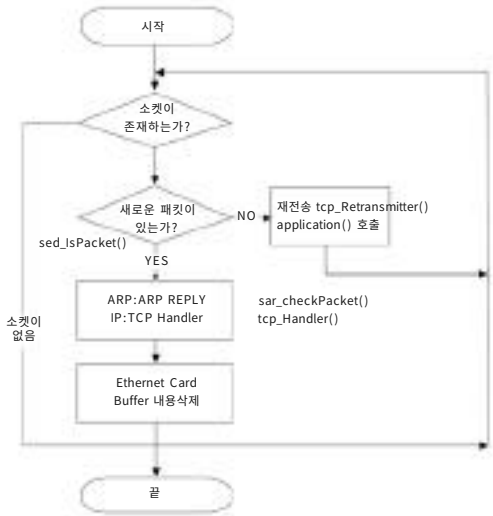

그림 13. TCP() Flow Chart

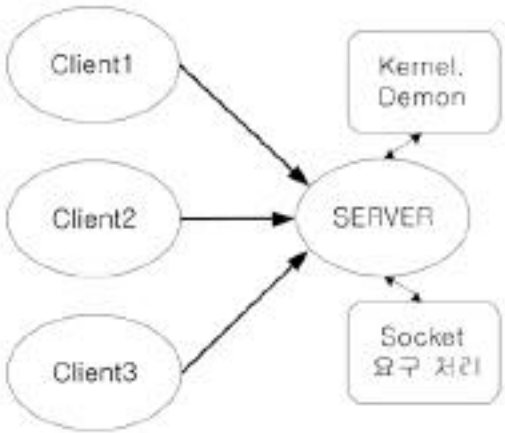

그림 14. Connection

SDRAM, Fast Ethernet을 장착한 시스템이었다. 특히 네트워크 기능이 막강한 Linux인 만큼 소켓 프로그램의 하위 기능이 잘 지원되어 상당히 좋아했던 적이 있었다. 물론 이런 정도의 시스템에서라면 큰 OS를 사용해서 큰 프로그램을 실행시키는 것이 맞겠지만, 보통의 ROM, RAM을 사용한 작은 시스템이라든지, OS의 overhead가 우려될 때는 이런 uC/OS나 작은 TCP/IP를 사용하는 것이 좋을 것이다.

원격지에서의 작업을 수행하는 시스템에 네트워크를 통해 접속하여 작업 상황을 관찰한다든지, 작업을 수행시키는 작업들은 얼마든지 이러한 시스템 셋트를 적용시킬 수 있을 것이다. 이러한 요구를 해결할 수 있도록 환경을 만드는 작업은 그들의 몫이다.

Unix 시스템 하에서 telnet 프로그램을 작성하는 것을 생각해 보자. 그림 14와 같이 Server 측(우리주변의 그런 대형 서버라기보다는 접속 요구를 받는 측이라고 생각하자)에서 서비스를 제공하기 위해서는 커널급의 레벨에서 받아들인 데이터를 보고 서비스마다 사용하는 포트 번호를 통해, 이 Socket 정보를 등록된 관계 Demon에게 전달하여야 한다.

해당 Demon은 Client측 포트 번호 등의 정보를 이용하여 Session을 생성하고 작업을 수행한다. 이러한 시뮬레이션을 통해 데이터 송수신이 이루어지므로 Telnet Server 프로그램은 이를 처리할 수 있게끔 작성하면 된다. 범용적으로 사용하려면 역시 기본 모델을 참고하여 작성하여야 호환이 된다.

이러한 작업을 우리의 Embedded System으로 맞춰보면 uC/OS는 Schedule, Task 관리를 수행하여 Multi-Tasking과 같은 역할을 수행하고, 몇 개의 Task들은 Ethernet 접속에서 응답 처리까지의 작업을 수행할 수 있도록 만들어져야 한다.

간단하게 예를 들면 한 개의 Task는 Ethernet Driver로부터 데이터 수신 여부를 검사하는 Task이다. 또 다른 Task는 Telnet 데이터를 보고, 대응 처리를 하는 Task이다. 다른 Task는 FTP를 담당한다. 또 다른 Task는 외부 디바이스와 인터페이스를 하면서 Database를 저장하는

```
void Main(void){

    Uart_Init();
    put_string("System Start... ");
    System_Init();
    put_string("OK.\r\n");

    put_string("OSIni...");
    OSInit();
    Put_string("OK.\r\n");

    Sem1 = OSSemCreate(1);
    Sem2 = OSSemCreate(1);

    put_string("OSTaskCreate...");
    OSTaskCreate(Init,                    &Id,
&Stack3[STACKSIZE - 1], 3);
    put_string("OK.\r\n");

    OSStart();
}
```

코드 6. main Routine

Task이다. 이러한 구조를 가지고 있을 때 외부로부터 Packet이 들어오면, 첫 Task는 데이터 내부의 정보를 검색하여 Telnet정보임을 검출하고 IPC를 이용해 담당 Task로 정보를 넘겨준다. 이 Task는 대응 처리로써 다른 Task에서 만들어 놓은 정보를 반송하게 된다.

이러한 작업들 자체가 우선순위에 기반을 두고 실시간으로 동작하게 된다. 마치 Multi-Taking 시스템에서 작업을 처리하는 것과 같다. 추가로 Session을 여러 개 열 수 있도록 하기 위해 중간에 Socket과 Data Pointer를 관리하는 Task를 두어 여러 Task를 관리하도록 하는 방법을 사용할 수도 있다. Overhead가 좀 크겠지만, 아예 10개 정도의 Session을 위해 10개의 Telnet Task를 만들고 이에 연결해주는 Task를 만들면 구현은 쉽게 할 수 있다. 이제 간단한 Test Program을 작성해보고 마치도록 하겠다.

전에도 많이 보았을 구조이다. Main 함수가 호출되려면 이미 Startup Routine을 통해서 이미 시스템의 기본이 확립되었다는 이야기이다. 이 이전의 과정들은 생략한다.

일단, Main() 함수 내에서 System에서 해야 할 기본

```
Void Init(void *i) {
    char Id1 = '1';
    char Id2 = '2';

    put_string("In Init Task ... \r\n") ;
    OS_ENTER_CRITICAL() ;
    OSSchedLock() ;

    put_string("IRQInstall() ... ") ;
    if    (    (int)IRQInstall(    &OSTimeTick,
N_TIMER1 ) < 0 ) {
        put_string("IRQInstall()\r\n") ;
    }
    put_string("OK\r\n") ;

    Enable_Int( N_TIMER1 ) ;
    Timer1_Start() ;
    ethernet_init() ;

    put_string("OSTaskCreate() ... ") ;
    OSTaskCreate(Task1,                    &Id1,
&Stack1[STACKSIZE - 1], 1);
    OSTaskCreate(Task2,                    &Id2,
&Stack2[STACKSIZE - 1], 2);
    put_string("OK\r\n") ;

    OSSchedUnlock() ;
    OS_EXIT_CRITICAL() ;

    OSTaskDel( OS_PRIO_SELF ) ;

    for (;;);
}
```

코드 7. Init Task Routine

```
Void Task1(void *i) {
    put_string("Ethernet test\r\n") ;
    put_string("tcp_init() ... ") ;
    tcp_init() ;
    put_string("OK\r\n") ;
    for (;;) {
        tcp_tick( 0 ) ;
        if ( khbit() ) {
            if ( get_char() == 'q' ) {
              put_string("return\r\n ") ;
              /* Interrupt Disable */
            } else {
              put_string("Alive.\r\n") ;
            }
        }
        OSTimeDly( 10 ) ;
    }
}
```

코드 8. Task1 Routine

```
Void Task2(void *i) {

    for (;;) {
      tcp_Listen();
      while (tcp_establish) {}
      put_string("client connect !\r\n");
      tcp_Send("Welcome to Server");
      tcp_Send("PassWord : " );
      OSSemPend(Sem2);
      If(S = xx) tcp_Send("Nice day ! " );
      else tcp_Send("Retry ! " );
      ....
}
```

코드 9. Task2 Routine

적인 작업들을 수행하였다. uart, put_string 등은 개발시의 Debugging을 위해서 PC측에서 메시지를 볼 수 있도록 하기 위한 함수이므로 분석에서는 그리 신경 쓰지 않아도 된다. OSInit()을 통해 OS 기본 설정을 하였고, Task간의 통신을 위한 IPC 자원으로써 Semaphore를 2개 생성하였다. 다음 OSTaskCreate()를 통해 Init 이라는 Task를 등록하였고, OSStart()로 Multi-Task를 시작하여 프로그램은 계속 수행될 것이다. 이러한 구조는 앞에서 많이 언급된 사항들이므로 이해하리라 믿는다.

Main Routine에서 생성한 유일한 Task, Init Task에서는 Timer Interrupt Handler를 등록하여 TimeTick으로 사용할 수 있도록 하였다. 그 다음 2개의 Task를 등록하였는데, Task1과 Task2이다. 이 후 현재의 Init Task는 1회성 Task로 생각하여 지워버리는 작업을 하였다. 이로써 기본적인 작업이 수행되었고 실제로 사용자가 쓰려는 Task가 수행되게 될 것이다.

Task1은 단순한 Task이다. 단지 키보드 입력을 받아 데이터를 처리하면서 시스템 Fault와 Multi-Tasking 여부만을 측정하고 있다.

코드 9에서 보이는 Task2의 코드는 pseudo code이다. 지면상 표시할 분량이 아니므로 내용으로 이해하도록

한다. 이 Task2는 실제 Telnet Server의 역할을 한다. 이는 Telnet 접속 초기의 과정만을 표현한 것으로 Tcp_Listen() 상태에서 메시지를 기다리고 있다가 메시지에 의해서 다음 과정을 수행해 나간다.

프로그램 수행에 필요한 함수들은 TCP/IP Stack 함수들과 OS 관련 함수들을 이용하여 만들면 된다. Serial, Key Scan, LCD 등등의 하위 드라이버 레벨 함수들이 추가로 잘 구축되면 이제 필요한 상위 함수들로 멋진 RTOS 시스템을 구축할 수 있을 것이다.

마치며

이번 회까지 6회에 걸쳐 RTOS에 대한 사항을 살펴보았고 포팅을 시도했었다.

실시간성을 중시여기는 시스템에 또한 OS로써 사용되어 프로그램들이 좀더 쉽게 작성될 수 있기를 바라는 시스템 엔지니어, 프로그래머들에 의해 사용될 것들이다. 가장 간단한 형태의 H/W와 OS와 Stack을 기준으로 정리하였다. 처음 OS를 접하는 사람들을 기준으로 작성한 글이어서 너무 미흡하게 생각되기도 한다.

기회가 된다면 다음에는 더욱 나은 정보를 공유할 수 있도록 하겠다. EE

— 정보 —

TCP/IP Stack은 Web 상에 많이 공개되어 있다. 쓸모 있는 자료를 찾아 내는 일도 그리 만만치 않은 작업이다. 그래서 TCP/IP에 관한 필자가 괜찮다고 느낀 것 몇 가지만 선보인다.

● 해외 도서

1. TCP/IP illustated Volume Ⅰ,Volume Ⅱ(1),Volume Ⅱ(2)(Prentice Hall)
2. UNIX Network Programming(Prentice Hall)
3. Internet Core Protocols(O'Reilly)
4. TCP/IP Network Administration

● 추천 홈페이지

1. TCP/IP에 관련된 resource 제공
   (http://www.landfield.com/faqs/internet/tcpip/resource-list/)
2. Waterloo TCP/IP관련 정보
   (http://www.wattcp.com)
3. TinyTCP관련 정보
   (http://www.csonline.net/bpaddock/tinytcp/default.htm)
4. Embedded TCP/IP관련 paper
   (http://www.tordivel.no/etcp.htm)
5. Xinu OS관련 정보
   (http://willow.canberra.edu.au/~chrisc/xinu.html)
6. uC/CO연계 TCP/IP관련 정보
   (http://www.ucos-ii.com)