Proceedings of the 12th
KACC, October 1997

소형 실시간 커널을 위한 TCP/IP 프로토콜 설계 및 구현

Design and Implementation of TCP/IP Protocol Stack for Small Real-Time Kernels

°윤재식*, 김재양**, 정선태***

* 중앙 소프트웨어(주)(Tel:082-02-3485-2154;Fax:082-02-3485-2100;E-mail:yoon@syscon.soongsil.ac.kr)

** 숭실대학교 전자공학과(Tel:082-02-817-5987;Fax:082-02-821-7653;E-mail:basagi@syscon.soongsil.ac.kr)

*** 숭실대학교 전자공학과(Tel:082-02-820-0638;Fax:082-02-814-1652;E-mail:cst@syscon.soongsil.ac.kr)

Abstracts Many small-sized real-time kernels do not provide memory management and device drivers, not to mention file management. In this paper, we propose a design and implementation of TCP/IP protocol stack for such small real- time kernels based on [6] where we studied issues to be considered for porting the functionalities of TCP/IP for such small real-time kernels.

Keywords Real-Time Kernels, TCP/IP, Real-Time Systems, Porting, Networks

1. 서론

SCADA나 DCS등 실시간 시스템의 말단 장치는 실시간 로컬 제어뿐만 아니라 상부단의 모니터링을 위한 통신 기능을 갖추어야 한다. 디지털 계전기, PLC 등 독자적으로 동작되는 장치들의 경우에 통신 기능이 부가되면 보다 많은 응용을 할 수 있다. 대부분 상용의 실시간 시스템 통신망 구조는 시스템 구성시에 결정되고, 제한된 노드와 통신 용량을 가지고 운영된다. 이러한 실시간 시스템의 경우는 일반적으로 통신망이 외부에 개방되지 않는다. 따라서 대부분의 실시간 시스템에 사용되는 통신은 독자적인 통신 방법(통신 매체, 통신 매체 접근 방법, 통신 프로토콜 등)을 사용하여 왔다. 그러나 응용에 있어서 실시간 시스템의 규모가 커지고 복잡해지고 있어, 통신 용량이 더욱 많이 요구되며 통신 접속 방법도 다양화되고 있다.

대부분의 소규모 실시간 시스템은 OSI(open system interconnection) layer 1(Physical link), 2(Data link layer)[3]만을 지원하여 통신하고 있다. 이러한 기능은 하나의 segment 통신에 적합하다. 그러나 시스템의 규모가 커지면서 하나의 segment를 넘어서 신뢰성 있는 표준(OSI layer 3(Network layer), 4(Transport layer)) 통신 기능이 요구되고 있다. 현재 TCP/IP[1][2][5]를 지원하는 상업용 실시간 O/S(VxWorks, LynxOS 등)들은 범용 O/S인 UNIX의 TCP/IP protocol stack[2][4]에 가까울 정도로 크고 필요 이상으로 기능이 많다. 따라서 소형 실시간 시스템의 커널에 적합한 기능과 크기를 갖는 TCP/IP 이식에 대한 고려가 필요하게 된다.

보통 소형 단말기에 적용할 정도의 소형 실시간 커널은 제한된 수의 task 관리와 task간 동기, Timer 기능을 제공한다. 그러나 일반 O/S 와 같은 파일 시스템, task간 메모리 보호, 할당 등은 제공하지 않는다. 이와 같은 소형 실시간 커널의 제한된 기능 때문에 범용 O/S(UNIX)에 사용되는 TCP/IP protocol stack을 그대로 이식하는 데는 여러가지 어려움이 존재한다. 특히, 대부분 소형 실시간 시스템이 적용되는 곳이 embeded system이기 때문에, 경우에 따라서는 충분한 메모리 사용과 고성능 하드웨어 자원 사용에 제약을 받는다. 따라서 소형 실시간 커널을 위한 TCP/IP 통신 프로토콜은 보다 적은 양의 시스템 메모리에서도 실시간 특성이 잘 동작하도록 설계되어야 한다. 그러므로 범용 O/S등의 TCP/IP protocol stack구현에 사용한 구조는 이러한 소형 실시간 시스템에 적합한 구조로 변경 되어야 한다.

이전의 논문[6]에서 소형 실시간 환경의 제한된 요소와 TCP/IP를 이식 하는데 고려되어야 할 내용을 살펴 보았다. 본 논문에서는 이를 바탕으로 소형 실시간 커널에 적합한 TCP/IP 기능을 정의하고 이를 구현하기 위한 TCP/IP 소프트웨어 구조를 설계하고 구현한 결과를 기술한다. 본 논문에서 시도한 연구는 산업 자동화의 핵심인 실시간 네트워킹 기술 발전에 이바지할 수있을 것으로 여긴다.

2. 소형 실시간 커널을 위한 TCP/IP 주요 기능 정의

이절에서는 먼저 본 논문에서 설계하고 구현하고자 한 소형실시간 커널을 위한 TCP/IP 프로토콜 스택이 수행될 환경을 정의하고, 이 환경상에서 필요로 하는 TCP/IP 기능 등을 정의한다.

2.1 소형 실시간 커널을 위한 TCP/IP 프로토콜 스택의 수행 H/W 환경

수행이 될 물리적 H/W 환경은 SCADA 나 DCS의 소형 말단 장치 또는 소형 디지털 계전기, 소형 PLC 등이다. 따라서 실시간 커널과 이에 구현될 프로토콜 스택은 16bit 환경에서도 수행될 수 있는 크기와 기능이어야 한다. 특히 16 bit CPU이면 최대 직접 접근할 수 있는 메모리가 64K byte 이므로 이러한 작은 크기에서도 TCP/IP 기능을 이용할 수 있어야 한다. 이러한 작은 메모리에서는 통신 세그먼트의 크기가 작아야 하고 기능을 알맞게 제한하여야 한다. 대부분의 소형 실시간 커널의 크기는 8-16Kbyte 정도이고 이에 구현될 목표로 하는 TCP/IP 스택의 크기는 20-40Kbyte 정도 이다. 예를 들어 이에 구현될 커널과 TCP/IP 프로토콜의 크기가 30-45Kbyte 정도라면 PLC 코드 크기가 15-20Kbyte 이고 데이터가 크기가 10-15Kbyte 정도 되는 소형 PLC는 쉽게 구현할 수 있다. PLC에 TCP/IP 기능을 추가하여 공장 자동화를 실현하는데 별도의 통신 게이트 웨이 없이 바로 응용 프로그램에서 원하는 정보에 접근할 수 있다.

2.2 O/S platform

본노문에서 고려하는 소형 실시간 시스템을 위한 O/S 기능은 16bit CPU 에서도 수행될 수 있는 것이다. 그리고 MMU(memory management unit)기능을 이용하지 않고 선형 어드레스 공간에 메모리 보호 기능 없이 최소한의 기능에 최대한의 성능을 내는 작고 빠른 커널이여야 한다. 최소한의 기능은 실시간을 지원하는 task 관리 기능과 task 간에 동기화 기능을 가정한다. 본 논문에서는 이러한 특성의 실시간 커널가운데, 공개된 간단하면서 성능이 좋은 실시간 커널인 uC/OS 를 선택하여 이용 하였다.(크기는 10K 미만)

2.3 요구 TCP/IP 기능

표준 TCP/IP 기능 중에 소형 실시간 시스템에 필요한 기능을 정의 한다.

2.3.1 physical layer, data link layer 기능

표준적인 이더넷, 토큰링, 토큰버스 방식 등은 32bit 정도의 CPU 에서 지원할 수 있게 하고 16bit 정도의 CPU 에서는 SLIP 이나 PPP 또는 CAN 과 같은 field bus 를 사용 하거나 독자적으로 아주 간결한 물리적 접근을 취할 수 있다.

2.3.2 ARP/RARP 기능

물리적인 접근 어드레스와 IP 어드레스간의 연결 관계를 해결 하는데 필요하므로 반드시 지원한다.

2.3.3 IP 기능

IP 는 UDP 와 TCP 상위 프로토콜과 data link 사이에 네트웍 계층을 담당 한다. 소형 말단 장치에 장착되는 것을 목적으로 하기 때문에 IP 기능 중에 게이트웨이 기능은 구현할 필요가 없다. IP 의 multihome 기능 또한 이중화를 위해서 최대 2 개의 IP 만을 허용한다. IP 의 multicasting 기능은 본 논문이 목적으로하는 소형 실시간 시스템 환경에서는 broadcasting 기능을 가지고 대부분의 간단한 응용을 해결할 수 있기 때문에 지원하지 않는다. 게이트웨이의 동적인 변화를 지원하지 않고 이중화 만을 지원하여 최대 2 개의 게이트웨이만을 지원 한다.

2.3.4 ICMP 기능

IP, UDP,TCP 에 각각 해당하는 대부분의 ICMP 기능이 지원되지 않을 경우 문제의 원인을 파악하는데 어려움이 있고 시간이 소요된다. 그러나 ICMP 의 일부 기능이 지원되지 않아도 필수적인 통신 기능은 수행이 가능하다. 따라서 ICMP 기능은 echo 요구와 응답, protocol unreachable, port unreachable, source quench 등의 기본적인 기능만을 구현한다.

2.3.5 UDP 기능

UDP 는 각 socket 과 port 를 관리하며, 동시에 지원되는 UDP socket 는 최대 20 개로 제한한다. 그리고 IP 와 유저 인터페이스를 관리한다.

2.3.6 TCP 기능

동시에 TCP 연결은 최대 20 개로 제한한다. URGENT 기능은 현재 전송하는 일반적인 데이터 보다 특정 내용을 긴급하게 응용 프로그램에 전달하고자 하는 것이다. 그러나 URGENT 기능은 대부분의 응용에서 별도의 socket 을 이용하여 해결할 수 있으므로 제공하지 않는다. RTT 에 관련된 사항을 제외한 재 전송 시간들과 window 크기 등은 시스템 초기에 고정한다.

2.3.7 SOCKET 층 기능

User 의 편의를 위해, user 의 socket interface 는 BSD Unix 의 socket interface 와 최대한 유사하도록 구현하였다. socket 을 생성하는 SOCKET, socket 을 어드레스와 연결시키는 BIND, 생성된 socket 를 소멸시키는 CLOSE, UDP 전송을 위한 SENDTO, 수신을 위한 RECV_FROM, TCP 를 위한 ACCEPT, CONNECT, SEND, RECV, 옵션을 조절하는 GET_OPTION, SET_OPTION 등을 지원한다.

3. 소형 실시간 커널용 TCP/IP 스택 소프트웨어 설계 구조

3.1 설계 기본 목표

- 실시간 특성을 지원한다.
- 가능한 한 크기가 작고(20-40Kbyte) 간단하게 한다.
- 소형 실시간커널의 제한된 기능(no memory management, no device drivers, file management, etc.) 에서도 잘 수행되도록 한다.
- 이식성이 좋도록 한다.

3.2 실시간 특성 지원 구조

실시간 특성을 지원 하기 위해서 실제 제어용 task 이 TCP/IP 기능 보다 높은 우선 순위를 가질 수 있다. 이 때문에 TCP/IP 기능을 실시간 커널에 구현 하는 것은 곤란하다. TCP/IP 기능을 하나의 task 으로 구현 하여 TCP/IP task 보다 우선순위가 높은 task 에 시스템 자원을 할당할 수 있게 한다. TCP/IP 를 구현 하는 방법은 하나의 task 로 구현 하거나 각각의 프로토콜 마다 task 로 구현 방법 두 가지가 있다. 이중에 각 프로토콜 마다 task 로 구현 하면 복잡한 여러 프로토콜이 존재할 때 보다 모듈화 되어서 설계하고 구현하기 쉽다. 그러나 이 논문에서 목표로 하는 소형 TCP/IP 는 아주 단순하여 각각의 프로토콜을 task 간에 제어 부담이 모듈화 장점보다 크다. 따라서, TCP/IP protocol stack 를 하나의 task 으로 구현한다.

3.3 디바이스 드라이버 구조

소형 시스템 커널에서는 많은 경우, 디바이스 드라이버 구조를 제공하지 않는다. 그러나 TCP/IP 에서 이용할 디바이스 드라이버가 필요하다. 이를 지원 하기 위해서 커널을 수정하여 일반적인 UNIX 와 같은 일괄된 접근의 인터페이스를 제공하지 않고, 본 논문의 구현에서는 아주 제한적인 구조로 하여, 디바이스 드라이버는 task 형태로 구현 하고 TCP/IP 에서 디바이스와 관련된 필요한 정보를 TCP/IP 내부에서 단 2 개만을 관리 한다. 디바이스 제어는 task 간 통신 기법을 이용하여 구현 한다.

3.4 TCP/IP 인터페이스 방법

상위의 결정에 의해서 TCP 는 task 형태를 가진다. 이에 더해서 USER task 와 TCP task 간에 상호 동작 방법을 설계해야 한다. 두가지 접근이 가능한데 하나는 TCP/IP task 데이터를 인터페이스 함수에서 직접 조작하는 방식이다. 이러한 방식은 초기 구현 접근이 쉬우나 상호배제를 해결하기 위해 많은 사항을 고려 하여야 한다. 다른 방법은 메세지를 이용하여 클라이언트 서버 모델로 구현 한다. 이런 구조를 사용하면 TCP/IP task 데이터에 접근하는 문맥을 자신 하나로 제한 된다. 데이터에 접근하는 문맥이 하나이므로 상호배제 문제를 고려하지 않아도 된다. TCP/IP 유저 인터페이스 함수는 내부적으로 메세지를 이용하여 TCP/IP task 와 동기화 한다.TCP/IP task 내에서는 유저 요구와 네트웍으로부터의 입력 그리고 시간 관리 내용이 모두 메세지를 통하여 기동된다. 실시간성 지원을 위해서 메세지 전송 기능은 우선순위를 갖는 메세지 전달을 하여야 한다. 각각의 task 으로부터의 여러 요구(메세지 전송) 가 서버로부터 처리되기 전에 도착한다면 여러개의 요구중에 가장 우선순위가 높은 task 요구에 응답을 하게 된다. 그리고 TCP/IP 구현 task 보다 우선순위가 더 높은 응용프로그램이 TCP/IP 를 이용 하여야 한다면 응용 task 과 TCP/IP 를 이용하는 task 으로 분리 하고 이를 task 간 통신을 이용하여 데이터를 전달 하면 Priority inversion 문제가 발생 하지 않는다.

3.4 Timer 구조

TCP/IP 의 기능을 구현 하기 위해서는 많은 시간 관리가 필요하다. TCP/IP task 가 메세지를 이용하는 구조 이므로 이러한 시간관리 기능을 내부적으로 완전히 구현할 수 없다. 이러한 시간 관리 기능을 구현 하기 위해서 보조 task 가 필요하다. 그러므로 시간관리만을 전담하는 task 를 두어서 이 task

와 TCP/IP task 간에 메세지 교환을 이용하면 TCP/IP 시간관리 기능을 구현할 수 있다.

이상에서 살펴본 TCP/IP 프로토콜 스택 S/W 설계 전체 구조는 그림 3.1과 같다.

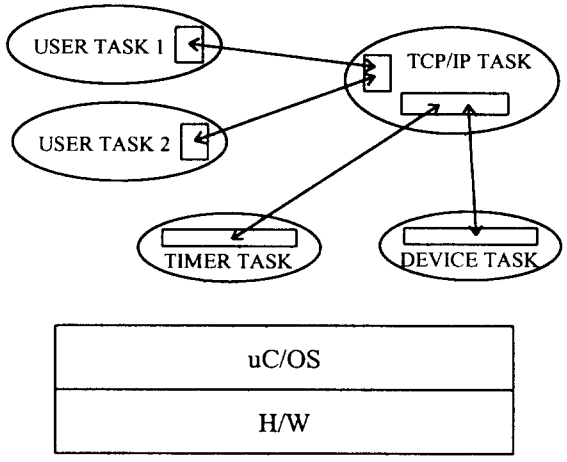

그림 3.1. TCP/IP 프로토콜 스택 S/W 설계 구조

4. 세부 구현 내용

4.1 TCP/IP task 내부 구조

TCP/IP task은 내부에서 기능적으로 여러 모듈로 분리 된다. 각각의 모듈은 다음과 같다. 유저 task과 인터페이스하는 socket 모듈이 있고, 전송을 계층을 위한 UDP 모듈, TCP 모듈, 네트웍 계층을 위한 IP 모듈, ICMP를 위한 모듈, physical, data link 층을 위한 디바이스 관리 모듈이 있다. 재전송 등의 시간을 관리 하기 위한 시간 모듈, 메모리를 할당 하고 제거 하기위한 메모리 관리 모듈이 있다. 이중에서 시간을 관리 하는 모듈은 uC/OS의 구조적인 이유로 하나의 task으로 구현 되었다. 이러한 기능을 수행 하는 모든 함수는 상위에서 결정한 메세지를 이용한 task 구조에 의해서 재 진입과 상호 배제 문제를 고려 하지 않아도 된다. TCP/IP task 내에서 함수는 바로 호출 함으로서 이용되고 일정한 시간의 지연이나 비 동기적인 사건에 의해서 task이 멈추어야 되는 부분은 사후 재 동작을 위하여 모두 메세지를 이용 한다. TCP/IP 기본 골격 함수의 기능은 이러한 메세지를 적절한 함수에 연결시키는 것이다. 메세지에 의해서 처리되는 비 주기적인 사건 종류는 유저와 인터페이스하는 부분, 네트웍 디바이스 드라이버의 출력과 입력, 시간 관리가 필요한 부분 이다. 이를 위해서 적당한 메세지가 정의 되었으며 메세지를 이용하여 동기를 구현 하는 함수가 구현 되었다. TCP/IP task의 내부 구조는 다음 그림 4.1와 같다.

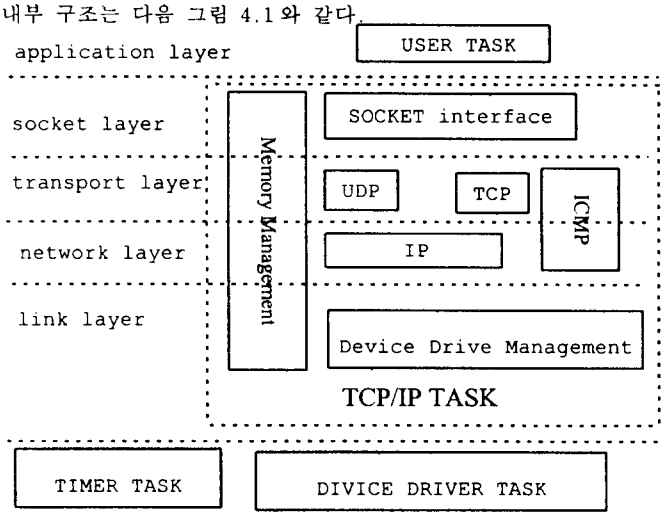

그림 4.1. TCP/IP task 내부구조

4.2 메모리 구조

메모리 관리 성능은 프로토콜의 성능과 많은 관계가 있다. 프로토콜의 성능을 위해서는 데이터의 복사를 최소화 하는 것이 바람직 하나 이를 구현 하기 위해서는 아주 복잡하고 정교한 구현이 요구 된다. 그러나 작고 간단한 구현을 위해서 데이터의 복사를 하더라도 구현을 간결하고 작게 할 수 있는 구조를 선택 하였다.간결한 구조의 제한 내에서 복사를 최소화 하기 위해서 다음과 같은 방법을 이용 한다.

상위로 계층으로 전송될 때는 앞 부분의 헤더나 뒷부분의 테일이 제거되면서 데이터가 가공 되기 때문에 맨 하위(디바이스 드라이버)에서는 버퍼의 데이터 크기와 시작 오프셋을 포함 하도록 한다. 하위 계층으로 데이터가 전송될 때는 현재의 데이터 내용에 앞부분에 데이터(헤더)가 추가 되면서 처리되기 때문에 상위 프로 토콜은 맨 마지막 층까지 얼마나 많은 양의 헤더가 추가되는지 산출 할수 있기 때문에 이를 미리 확보 하여 메모리를 할당 한다.

다음의 두 가지 경우에는 데이터의 복사가 요구된다. IP 계층에서 세그먼트를 분할 하거나 조립 할때 메모리의 복사가 요구된다. TCP의 재전송이 필요할경우 유저로부터 제공된 데이터 블럭에서 다시 복사하여 재전송 세그먼트를 작성 하는 것이 필요하다. 메모리 관리를 위한 정보로 다음의 내용이 필요 하다.

1 전체 버퍼의 크기
2 전체 유효한 데이터 크기
3 시작 오프셋

uC/OS는 메모리 관리 기능이 없으므로 TCP/IP만을 위한 최소한의 메모리 할당과 해제 하는 루틴을 작성 하였다. C표준 함수인 malloc와 free를 구현 하였다. malloc 함수는 요구되는 메모리 크기에 더하여 관리에 필요한 정보량을 할당한다. 관리에 필요한 정보는 전체 메모리 크기 사용 유무 이다. 할당 알고리즘은 사용되지 않는 메모리 영역 중에 요구하는 메모리 보다 큰 영역이 발견되면 이를 초기화 하고 리턴 한다. 메모리를 해제 하는 free 함수는 인자로 넘겨지는 포인터의 앞 부분에서 관리 정보를 참조 하여 비사용으로 표시하고 메모리가 분할되는 것을 방지 하기 위해서 앞뒤의 메모리 영역이 사용 되고 있지 않으면 이를 결합하여 하나의 메모리 영역으로 만든다.

4.3 TIMER 관리 구현

TCP/IP 모듈중에 주로 시간 관리가 요구 되는 기능은 TCP의 다양한 재전송 요구와 IP 모듈에서 재 조립을 위한 기능중에 시간 제한을 위한 요구, ARP에서 캐쉬 관리 요구, 각 socket 인터페이스의 시간 제한 요구 등에 있다. 이렇게 다양하게 많은 시간 관리가 요구 되지만 본 논문의 구현 에서는 UDP와 TCP의 연결이 최대 연결을 각각 20개 내로 제한 하고 있어서 아주 간단한 구조로 이를 구현 하였다. 시간 관리 요구를 구현 하기 위해 고정된 크기의 데이터 구조 배열을 이용한다. 데이터 구조 내용은 사용 유무, 내부 clock tick수와 전달 하고자 하는 메세지 내용이다. 일정한 clock tick마다 TIMER task이 각각의 유효한 데이터의 clock tick를 1씩 감산 한다. 만약 결과가 0 이라면 해당 메세지를 TCP/IP에 전달 하고 무효임을 표시 한다.

4.4 TCP/IP 주요 함수 구현

4.4.1 ARP 층의 함수

ARP 층에서는 IP 어드레스와 물리적 계층의 어드레스 사이를 상호 연결 시킨다. 그러나 이러한 관계가 동적이고 모든 관계를 알 수 없다. 이 정보 내용을 보관하는 캐쉬와 일정한 시간 동안 사용되지 않으면 무효화 시켜서 잘못된 연결을 없애야 한다.

ARP와 IP와의 관계- 상위 계층에서 데이터를 전송할 때 바로 하위 계층의 함수 호출을 통하여 구현 한다. 그러나 ARP 계층은 어드레스 해결을 위한 상당한 시간(상대적으로 CPU 시간에 비해)이 요구되며 결과가 상대방의 응답으로 이루어

지기 때문에 단순히 ARP 패켓을 전송 하고 응답을 대기할 수 없다. 이를 위한 구조가 IP 층에서는 ARP 의 응답을 기다리는 큐를 두어서 이를 처리 하는 함수를 별도로 구현 한다. 이 함수는 ARP 층에서 어드레스를 해결 하거나 실패 했을 경우 둘 다 상위의 호출과 비동기 적으로 처리 한다.

ARP 에 관련된 출력이 ARP 로 부터 요구 되면 해당 캐시 내용을 살펴 보고 이미 내용이 있다면 바로 리턴 하고(ARP 해결 됨을 지신의 메세지 큐에 저장) 해당 내용이 없다면 여분의 케시를 초기화(할당 하고 아직 해결되지 않았음을 표시 하고) 하고 전송 한다. 재 전송 시간을 입력 하고 대기(리턴) 한다.

ARP 의 응답으로 입력이 전송 되면 케쉬의 내용을 검사 하고 해당 대기 중인 내용이 있으면 IP 층에 어드레스가 해결 됫음을 알린다(함수 호출)

만역 재전송 타이머 메세지가 전송 되면 (메세지 내용에 찾고자 하는 IP 어드레스를 포함 한다) 재 전송 회수를 증가 하고 경계치 이상 이면 포기 하고 IP 에 통보 한다(함수 호출).

4.4.2 TCP 층의 함수

TCP 함수들은 대부분 출력과 입력이 아주 밀접하게 연계되 있다. RFC793 에 정의된 TCP 상태도의 변화는 각 상태별 함수를 구현 하여 해당 상태를 처리 하도록 하였다. 그러나 이러한 구분이 명확히 처리되지는 않으며 특히 입력을 처리하는 부분이 굉장히 복잡 하다. TCP 에서 전송은 오히려 단순한 편이다.

전송 함수- TCP 전송 요구는 유저로부터 요구 이므로 내부적으로 TCP 연결 상태를 파악 하고 행동을 예측 하기 쉽다. 오직 데이터의 전송은 ESTABLISHED 상태에서만 가능 하며 그 외의 상태는 연결을 설정하기 위한 전송이나 연결을 해제하기 위한 전송일 뿐이다.

전송 함수는 유저가 요구하는 데이터 크기와 PUSH 조건을 검사한다. PUSH 가 요구되면 요구하는 데이터 크기에 관계없이 바로 전송을 시도 한다. 그러나 PUSH 가 요구 되지않으면 요구 데이터의 크기를 검사하여 바로 보낼 것인지 좀더 많은 양의 데이터를 대기할 것인지를 결정한다. 전송을 시도 한다면 다음 사항을 고려 하여야 한다. 전송 하고자 하는 데이터 크기, 상대방 윈도우 크기, 한번에 전송할 수 있는 최대 크기 등을 검사하여 전송 크기를 결정하고 ACK 할 내용을 검사 하여 TCP 세그먼트를 완성 하여 전송 한다. TCP 전송과 관계되어 여러 가지 시간을 관리 하여야 한다. 재 전송을 위해서 RTT 를 계산, 재 전송 시간, 상대방 window 크기가 0 일때 이를 체크하기 위한 시간, 데이터 전송 요구가 없을때 TCP 연결이 유효한지 검사하는 시간 등이 관리 되어야 한다.

입력 함수- 네트웍으로부터 입력은 비 동기적으로 전혀 예측할 수 없다. 상대방의 전송이 손실되어 도착하지 않을수도 있고 잘못된 내용이 입력될 수도 있다. 이러한 모든 상황에 대한 처리를 하여야 한다. 입력 처리를 상태별로 정리하여 간략하게 표현 하면 다음과 같다.

공통적으로 자주 사용되는 함수는 다음과 같다.

sequence 번호 보정-입력되는 세그먼트의 sequence 를 검사 하여 받아 들일수 있도록 보정 한다. 만약 받아들이수 없다면 ACK 세그먼트를 전송 하고 입력 세그먼트를 버린다.

세그먼트 text 처리- 도착한 데이터를 유저 공간에 복사하고 PUSH 등을 처리 한다.

SYN 처리 함수- window 안에 있지 않다면 RST 를 전송하고 CLOSED 상태로 바꾼다.

주요 상태별 처리는 다음과 같다.

CLOSED-입력된 세그먼트에 해당 연결이 없다면 즉 CLOSED 상태 라면 RST 를 확인하여 RST 가 설정되어 있으면 그냥 무시하고 버린다. 그러나 RST 가 설정되어 있지 않다면 잘못된 것을 상대방에 알리다. 다음 세그먼트를 전송한다. <SEQ=SEG.SEQ><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK> 만약 ACK 가 on 이라면 <SEQ=SEG.ACK><CTL=RST>

LISTEN-ACK 를 검사한다. 어떠한 ACK 든 잘못된 것이다. RST 세그먼트를 전송 한다. <SEQ=SEG.ACK><CTL=RST> SYN 를 검사하여 on 이 아니라면 RST 를 전송 하여야 하다. on 이라면 SYN 세그먼트를 전송 하여야 한다. RECEIVED 상태로 바꾼다. <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SYN-SENT- ACK 를 검사 한다. on 이라면 SEG.ACK=<ISS or SEG.ACK>SND.NXT 라면 RST 세그먼트(<SEQ=SEG.ACK><CTL=RST>) 를 전송 한다. 만약 SND.UNA =< SEG.ACK =< SND.NXT 라면 받아 들이 수 있다. RST 를 검사하여 on 이라면 ACK 범위가 받아들일 수 있는 범위하면 상대방에 RST 를 전송 하고 범위 밖이면 그냥 버린다. socket 상태를 CLOSED 사태로 전환 한다. SYN 를 검사 한다. ACK 가 범위 내에 있다면 ACK 세그먼트를 전송 하고 ESTABLISED 로 변화 시킨다.

SYN-RECEIVED- sequence 번호 보정을 수행하고 RST 를 검사 한다. on 이라면 passive open 이면 LISTEN 으로 되돌리고 active open 이면 CLOSED 상태로 바꾼다. SYN 처리 함수 수행 ACK 를 검사 하여 SND.UNA =< SEG.ACK =< SND.NXT 라면 ESTABLISED 상태로 상태를 바꾸고 처리를 계속 한다.

ESTABLISHED- sequence 번호 보정을 수행하고 RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행 ACK 를 검사 받아들일수 있다면 해당 정보를 갱신 한다. 세그먼트 text 처리를 수행 FIN 를 검사하여 on 이하면 CLOSE-WAIT 로 이동 한다.

FIN-WAIT-1 - sequence 번호 보정을 수행하고 RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행 ACK 를 검사 받아들일수 있다면 해당 정보를 갱신 한다. 세그먼트 text 처리를 수행 만약 FIN 이 ACK 되었다면 FIN-WAIT-2 로 변화하고 처리 계속한다.

FIN-WAIT-2 - sequence 번호 보정을 수행하고 RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행, 세그먼트 text 처리를 수행 재 전송 큐가 비어 있으면 유저에게 OK 리턴. FIN 검사 하여 on 이라면 TIME-WAIT 상태로 변화.

CLOSE-WAIT - RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행

CLOSING- RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행, ACK 검사 FIN 에 대한 ACK 라면 TIME-WAIT 로 변화.

LAST-ACK - RST 검사 on 이라면 CLOSED 상태로 변화. SYN 처리 함수 수행 어떠한 ACK 든 CLOSED 상태로 변화.

5. 결론

본 논문에서는 소형 실시간 시스템에 적합한 TCP/IP 의 기능을 정의 하고 S/W 구조를 설계하고 이를 구현 하였다. 추후의 연구에서는 여기서 설계 구현된 TCP/IP 프로토콜 스택 S/W 를 실제 H/W 환경에 이식하여 그 성능을 분석할 것이며, 그 결과를 추후에 보고할 것이다.

참고문헌

[1]W. R. Stevens, TCP/IP Illustrated Volume 1,Addison-Wesley Publishing Co, 1994

[2]G.R. Wright and W. Richard Stevens, TCP/IP Illustrated Volume II, Addison-Wesley Publishing Co, 1995

[3]W.Stallings.,Handbook of Computer-Communications Standards, Volume 1, 2, 3, 2nd edition,HOWARD W. SAMS & COMPANY

[4]S.J. Leffler, Marshall Kirk McKusick, Michael J. Karels, John S. Quarterman, The Design and Implementation of the 4.4BSD UNIX Operating System, Addison-Wesley Publishing Co, 1996

[5]D.E. COMER and DAVIOD L. STEVENS, Internetworking With TCP/IP vol II: Design, Implementation, and Internals, Second edition, Prentice-Hall InternationalINC

[6]윤재식,김재양,정선태,"소형 실시간 커널에 TCP/IP 이식을 위한 고려",한국통신학회 논문지, VOL.15 NO.2,pp435-438,1996