임베디드 시스템으로의 Point-to-Point Protocol(PPP) 소프트웨어 이식

최 성 종†

요 약

디지털 셋톱박스와 같은 임베디드 시스템이 활발히 개발됨에 따라, 제한된 자원에서 동작하는 네트워크 소프트웨어 개발이 필요하다. 본 연구에서는 UNIX용 Point-to-Point Protocol(PPP) 소프트웨어인 PPPD를 임베디드 시스템으로 이식하기 위해 풀어야 할 문제점을 찾고, 이의 해결 방법을 제시하였다. PPP 소프트웨어는 커널 코드이고, 하드웨어를 직접 제어하기 때문에 이식을 하는데 많은 문제점이 있다. 또한, 임베디드 시스템으로 이식으로 인한 문제점도 있다. 이러한 문제점을 해결하기 위해 PPP 소프트웨어를 논리적인 모듈들로 분리한 후, 각 모듈을 분석하여 이식을 위해 해결해야 할 문제점을 찾았다. 이러한 문제점을 외부와의 인터페이스로 정의한 후, 각 인터페이스를 임베디드 시스템으로 이식하였다. 또한, 임베디드 시스템으로 이식함으로써 간략화 된 입출력 모듈에 대해 기술하였다. 마지막으로, 테스트 베드를 구성하여 서버와의 패킷을 네트워크 패킷 분석기를 사용하여 이식의 결과를 검증하였다.

Porting Point-to-Point Protocol(PPP) Software to an Embedded System

Seong-Jong Choi†

ABSTRACT

Developing network software in embedded systems, such as digital set-top boxes, requires coding under limited computing resources. This paper presents the porting of Point-to-point Protocol (PPP) software, PPPD, to an embedded system. The PPP is the most popular link layer protocol for the information appliance, to an embedded system. In order to achieve this, problems to be solved for the porting were identified and methods to solve the problems were described. First, PPP source codes were divided into modules. Next, functions of each module were analyzed and interfaces between the modules were delineated. With the analysis results, porting to the embedded system was described. The normal operation of the ported software was verified with the help of a network packet analyzer. Finally, experiences during the porting were presented. The method developed in the paper can be applied to the porting of software to an embedded system as well as the porting of network software.

1. 서 론

Digital TV를 수신하기 위한 디지털 셋톱박스, 핸드폰, Personal Digital Assistant(PDA)와 같은 가전정보기기(Information Appliance)의 보급이 증가하고 있다. 이러한 가전 정보기기는 여러 종류의 하드웨어을 기반으로 하고, 다양한 임베디드 운영체제(Embedded Operating System : EOS)을 사용하고 있다. EOS는 Windows 또는 UNIX 기반의 범용 OS 환경보다는 제한된 자원 하에서 자원의 효율적 운영과, 편리한 프로그램 개발 환경을 지

† 정 회 원 : 서울시립대학교 전자전기공학부 교수
논문접수 : 2000년 4월 7일, 심사완료 : 2000년 6월 22일

원한다. 최근 들어, 많은 가전정보기기가 인터넷과 연결되고 있는 추세이다. 예를 들어, 과거의 셋톱박스의 주된 목적은 디지털 방송의 수신이었지만, 현재 네트워크 기능이 추가되어 웹브라우저, 전자우편과 같은 인터넷 서비스를 직접 제공 받을 수 있는 시스템으로 기능이 확장되고 있다.

임베디드 시스템에서 사용되는 EOS의 기능 또한 제한적이고 모듈별로 고가를 지불해야 된다, 특히, TCP/IP와 같은 네트워크 소프트웨어는 제공되지 않거나 추가의 사용료를 부담해야 한다. 따라서, EOS를 위한 네트워크 소프트웨어를 직접 개발해야 할 필요가 있다. 이러한 네트워크 소프트웨어 개발 방법으로 기존의 소프트웨어를 이식하는 방법이 많이 사용되고 있다.

소프트웨어 이식은 기존 프로그램을 플랫폼 종속적인 부분과 플랫폼 독립적인 부분으로 정리하여 종속적인 부분을 타겟 플랫폼에서 동작할 수 있도록 개발하는 것이다. 여기에서 플랫폼은 하드웨어 및 OS를 포함한다. 이식의 대상 소스코드에 따라 이식에는 여러 종류가 있을 수 있다. 소프트웨어 이식 중 사용자 모드의 응용 소프트웨어를 이식할 경우 플랫폼 종속적인 부분은 주로 OS에서 제공하는 시스템 호출(system call)이다. POSIX와 같은 표준을 준수하여 응용 프로그램을 구현하면 이러한 시스템 호출 이식에 대한 문제는 최소화할 수 있다[1]. 네트워크 소프트웨어 이식의 문제를 근원적으로 해결하기 위해 JAVA를 사용한 네트워크 소프트웨어 개발이 시도되었지만 아직은 성능과 필요한 자원면에서 문제점을 갖고 있다[2]. 프로그램 설계 단계에서부터 여러 상이한 시스템에 쉽게 이식 될 수 있도록 여러 가지 연구가 진행되어 왔지만, 마찬가지로 임베디드 시스템에서 매우 중요한 메모리 자원을 희생해야 한다[3, 4]. 따라서, 기존의 네트워크 소프트웨어를 효율적으로 이식하는 방법과 임베디드 시스템을 타겟으로 하는 소프트웨어 이식 방법에 대한 연구가 필요하다.

본 연구에서는 UNIX용 Point-to-Point Protocol(PPP) 소프트웨어(PPPD[5])를 임베디드 시스템(IBM OS Open Realtime Operating System)으로 이식하기 위해 풀어야 할 문제점을 찾고, 이의 해결 방법을 제시하였다. PPP 소프트웨어는 커널 코드이고, 하드웨어를 직접 제어하기 때문에 이식을 하는데 많은 문제점이 있다. 또한, 임베디드 시스템으로 이식으로 인한 문제점도 있다. 이러한 문제점을 해결하기 위해 PPP 소프트웨어를 논리적인 모듈들로 분리한 후, 각 모듈을 분석하여 이식을 위해 해결해야 할 문제점을 찾았다. 이러한 문제점을 외부와의 인터페이스로 정의한 후, 각 인터페이스를 임베디드 시스템으로 이식하였다. 또한, 임베디드 시스템으로 이식함으로써 간략화 된 입출력 모듈에 대해 기술하였다. 마지막으로, 테스트 베드를 구성하여 서버와의 패킷을 네트워크 패킷 분석기를 사용하여 이식의 결과를 검증하였다.

본 연구에서는 다음과 같은 절차로 PPPD 소프트웨어를 이식 하였다. 2장에서는 이식에 사용된 PPPD 소프트웨어, 이식에 관련된 PPP의 내용 및 임베디드 시스템의 특징에 대해 기술하였다. PPP에 대한 내용으로, 프레이밍 기법, PPP 헤더 처리, 파라미터를 결정하는 상태 전이도에 대하여 자세히 기술하였다. 이외의 여러 옵션의 의미(semantics)와 패킷 오류와 같은 비정상적인 연결에 대한 설명은 이식에 중요하지 않기 때문에 생략하였다. 3장에서는 소스코드를 분석하였다. 이에 대한 분석은 디바이스 드라이버, 초기화, 네트워크 출력, 네트워크 입력, PPP daemon 및 큐의 관리에 대한 모듈로 구분하여 기술하였다. 입출력의 경우 주로 패킷의 흐름에 대해 분석하였다. 이러한 분석을 기초로 외부 프로그램 모듈과의 인터페이스를 분석하였다. 4장에서는 앞에서 분석한 인터페이스와 모듈을 타겟 플랫폼에 이식하는 방법에 대해 기술하였다. 5장에서는, 서버와의 통신을 네트워크 패킷 분석기를 사용하여 이식의 결과를 검증하였다. 6장은 이식 과정의 중요한 경험과, 사용자 모드와 커널 모드의 소스코드 이식에 대한 비교 분석에 대해 기술하였다. 마지막으로 7장에서는 결론과 본 연구 결과의 활용 방안에 대해 제시하였다.

2. Background

2.1 PPPD 소프트웨어

본 연구에서는 FreeBSD 3.1에 구현된 PPPD 소프트웨어를 임베디드 시스템에 이식하였다[5]. 네트워크 계층 이상의 상위 계층의 프로토콜은 이미 존재한다고 가정한다. PPPD 소프트웨어는 많은 UNIX 시스템에서 사용되고 있는 PPP 프로그램이다. Berkeley Software Distribution(BSD) 운영체제 환경에서의 소스코드를 선택한 이유는 많은 네트워크 소프트웨어가 BSD 환경에서 개발되기 때문이다. PPPD 소프트웨어를 선택한 이유는 사용자 모드의 소스코드와 커널 모드의 소스코드

를 동시에 이식함으로써 각각의 차이점을 비교할 수 있다는 것이다. PPPD 소프트웨어는 커널 모드인 PPP 디바이스 드라이버와 사용자 모드인 PPP daemon 두개의 부분으로 구성되어 구분할 수 있다. PPP 디바이스 드라이버는 PPP를 링크 계층에서 사용하기 위한 초기화와, 직렬 포트를 사용하여 PPP 패킷을 입출력 하는데 필요한 모듈이 구현되어 있다. 사용자 모드에서 동작하는 PPP daemon에는 링크를 열기위해 필요한 Link Control Protocol(LCP), 인증(authentication), Network Control Protocol(NCP)와 같은 옵션 결정 기능이 구현되어 있다 PPPD 소프트웨어는 자세한 설명 자료가 없다. BSD 환경하의 SLIP 소프트웨어는 [6]에서 자세하게 설명이 되어 있다. SLIP의 구현 방법과 PPP의 구현 방법은 매우 유사하지만, PPP는 보다 많은 기능을 갖고 있기 때문에 알고리듬이 더욱 복잡하다.

2.2 PPP의 Framing

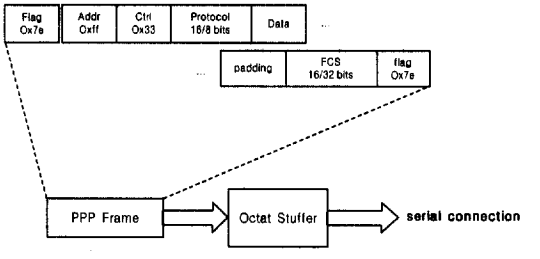
PPP는 TCP/IP 프로토콜 계층에서 링크 계층에 해당되는 프로토콜로 TCP/IP를 비롯하여 Novell IPX, AppleTalk, DECnet, 등의 여러 가지 network 계층의 프로토콜을 동시에 다중화할 수 있다[7, 8]. 이러한 기능은 기존의 Serial line IP(SLIP)이 갖고 있는 여러 가지 단점을 보완하는 것이다. 즉, 오류 검출, 여러 네트워크 프로토콜의 다중화, 연결 시 IP 주소 전달, 사용자 인증과 같은 여러 기능을 제공하는 프로토콜이다.

PPP의 기본적인 기능은 다음 세가지로 구분할 수 있다.

- **Framing Format** : 패킷의 시작과 끝을 정의.
- **Link Control Protocol(LCP)** : 링크 계층의 전송 옵션 결정,
- **Network Control Protocols(NCP)** : 여러 네트워크 계층 프로토콜에 필요한 파라미터 결정.

RFC 1662에서는 세 가지 PPP 프레이밍 방법을 정의하고 있다[9]. Asynchronous HDLC(AHDLC), bit-synchronous HDLC, Octet-synchronous HDLC의 세 방법 중 본 논문에서는 모뎀을 사용한 전화 접속방법으로 사용되는 AHDLC 프레이밍 방법에 대해 기술한다. (그림 1)은 AHDLC 프레이밍 포맷이다. 각 필드의 내용은 다음과 같다.

각 프레임을 구분하기 위해, 처음에는 0x7E의 고정된 프레임 구분자(frame delimiter)를 사용한다. 두 번째와 세 번째 필드인 address와 control 은 각각 0xFF와 0x03으로 고정하여 사용한다. 네 번째 필드의 프로토콜은 PPP에서 정의된 프로토콜 및 상위계층에서 사용되는 프로토콜을 구분하기 위한 정보로 사용된다. <표 1>은 많이 사용되는 프로토콜의 필드 값이다. 다음 다섯 번째 필드는 실제 전송 데이터이고, 여섯 번째 필드는 오류 검출을 위한 CRC 값이고 마지막으로 프레임의 끝을 알리는 프레임 구분자(0x7E)이다.



(그림 1) AHDLC Framing Format

<표 1> PPP Frame Protocol Field Values

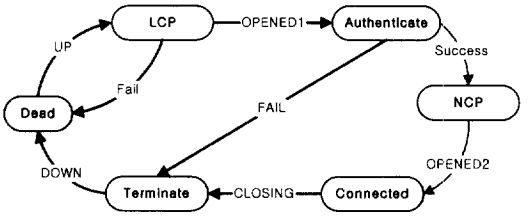
Value	Protocol	Type
0x0021	Internet Protocol (IP)	Network layer
0x002d	Van Jacobson Compressed TCP/IP	Network layer
0x002f	Van Jacobson Uncompressed TCP/IP	Network layer
0x00fd	Compressed Datagram	
0x8021	Internet Protocol Control Protocol (IPCP)	
0x80fd	Compression Control Protocol	
0xc021	Link Control Protocol	
0xc023	Password Authentication Protocol (PAP) Authentication	
0xc223	Challenge Handshake Authentication Protocol	

프레임 구분자(0x7E)나 XON/XOFF(0x0B/0x0D)흐름 제어를 사용할 경우와 같이 특별한 의미가 있는 제어 문자가 실제 데이터에 존재할 경우 이를 구분을 하기 위해 예외 문자(Escape character), 0x7D를 사용한다. 예를 들어, 데이터 값에 0x7E가 있게 되면 전송측에서는 0x7D 값을 보내고 0x7E와 0x20의 exclusive-OR를 취한 값 0x5E를 전송한다. 제어 문자를 많이 정의하게 되면 전송 효율에 낮게 된다. 따라서, 가급적 최소한의 제어 문자를 사용해야 한다. AHDLC에서의 흐름 제어는 위에서 언급한 소프트웨어적인 방법(XON/

XOFF)을 사용하거나 하드웨어 제어(RTC/CTS)를 사용할 수 있다. 소프트웨어적인 방법을 사용할 경우 앞에 설명한 예외 문자 알고리듬을 적용한다.

2.3 PPP Phase Diagram

RFC 1661에서는 회선을 사용하기 위해 거쳐야할 여러 phase를 정의하고 있다[10]. (그림 2)는 간략화한 phase 전이도이다. 우선 전화 접속을 시도하기 전, 즉 어떠한 연결도 되어 있지 않은 상태를 Dead phase로 정의한다. 사용자가 모뎀을 구동하여 상대방 모뎀과 접속에 성공하면 LPC phase로 전이한다.



(그림 2) Phase Transition Diagram for PPP

LCP phase에서는 LCP를 사용하여, 회선을 연결하기 위해 필요한 여러 옵션을 결정한다. 예를 들어, 예외 문자 정의(Asynchronous Control Character Map : ACCM), 전송프레임의 최대값(Maximum Receive Unit : MRU), 인증 프로토콜의 선택과 같은 여러 옵션을 정한다. 이러한 옵션의 결정은 통신 양측에서 각각 독립적으로 요청한다. 이러한 결과, A가 B로 전송할 때의 옵션과, B가 A로 전송할 때의 옵션이 다를 수가 있다. 양측에서 요청된 옵션이 승인이 되면(OPENED1), 사용자 인증(Authenticate) Phase로 전이된다.

사용자 인증 프로토콜은 여러 가지를 사용할 수 있는데, 어떠한 프로토콜을 사용할 것인지는 LPC Phase에서 결정된다. 현재 많이 사용하고 있는 프로토콜로서 PAP[11], MD5 CHAP[12], Microsoft CHAP등이 있다. 사용자 인증이 실패하면 회선을 닫을 Terminate Phase로, 성공하게 되면 NCP Phase로 전이한다.

NCP Phase에서는 NCP를 사용하여, 양측이 사용할 네트워크 계층의 프로토콜 파라미터들을 결정한다. PPP를 사용할 수 있는 네트워크 계층 프로토콜은 IP, IPV6, IPX, NetBIOS등이 있다. 이러한 프로토콜을 위해 IPCP[13], IPV6CP[14], IPXCP[15], NBFCP[16]와 같은 NCP를 사용한다. IP를 사용할 경우, IPCP에서 결정하는 중요한 파라미터로 IP주소가 있다. NCP Phase에서 사용할 네트워크 계층의 파라미터를 결정한 후(OPENED2) 비로소 양측간의 연결이 완성되게 된다 Connected Phase에서는 실제 사용자의 데이터가 전송된다. IP를 사용할 경우 HTTP를 사용한 웹브라우징이 가능하다. 필요한 전송을 끝내고 더 이상 회선을 사용하지 않을 경우, Terminate Phase로 전이한다. 이 Phase에서는 LCP의 Terminate Packet을 전송하여 회선을 닫은 후, 물리 계층에 사용된 디바이스(모뎀)을 초기화 한다(DOWN). 이후, 다시 Dead Phase로 전이한다.

2.4 임베디드 시스템

임베디드 시스템은 자원이 제한되고, 특정한 목적을 위한 시스템이다. 이는 Windows나 Unix를 기반으로 하는 범용 시스템과 많은 차이점을 갖고 있다. 대부분의 임베디드 시스템은 보다 큰 시스템의 한 요소이다. 마이크로웨이브 오븐 제어장치, 미사일 제어장치, 자동차 제동 장치 등이 이러한 임베디드 시스템의 예이다. 이러한 임베디드 시스템은 범용 시스템과 유사한 구조를 갖고 있다. 즉, 1) 마이크로프로세서와 메모리를 포함하는 하드웨어, 2) 이러한 하드웨어의 효율적인 관리 및 개발의 편의성을 위한 운영체제, 3) 마지막으로 특정 기능을 위한 응용 소프트웨어로 구성되어 있다. 하지만, 시스템 구성의 유사성과는 달리 시스템 설계의 방법은 뚜렷한 차이를 갖고 있다.

첫 번째, 범용 시스템에서는 사용자가 선택할 수 있는 다양한 하드웨어 및 소프트웨어의 호환성이 요구되지만 Embedded 임베디드 시스템은 개발자가 제한할 수 있는 특정의 응용 프로그램들만 사용하기 때문에 범용 시스템에 비해 호환성에 대한 요구가 크지 않다. 두 번째, 범용 시스템에서는 다양한 응용 프로그램의 동작을 위해 시스템 자원에 대한 최적화가 어렵지만, 임베디드 시스템에서는 특정한 응용 프로그램만 동작하기 때문에, 이를 위한 시스템 자원의 최적화를 꾀할 수 있다. 위와 같은 시스템 설계의 결과로 임베디드 시스템은 최소한의 하드웨어 자원과 최소한의 소프트웨어의 크기가 매우 중요하다. 물론, Time-to-Market이 중요한 소량, 고 부가가치의 임베디드 시스템에서는 위와 같은 설계 방법을 활용하기 어렵다. 또한, 추후의 시스템 업그레이드 기능을 추가하려면 모듈화 된 소프트웨어 설계에 더욱 치중해야 한다. 하지만 본 논문에서 주로 다루는 가전 정보기기에서는 최적의 설계

는 최소한의 자원으로 주어진 기능을 수행할 수 시스템을 설계, 구현해야 하는 것이 중요하다.

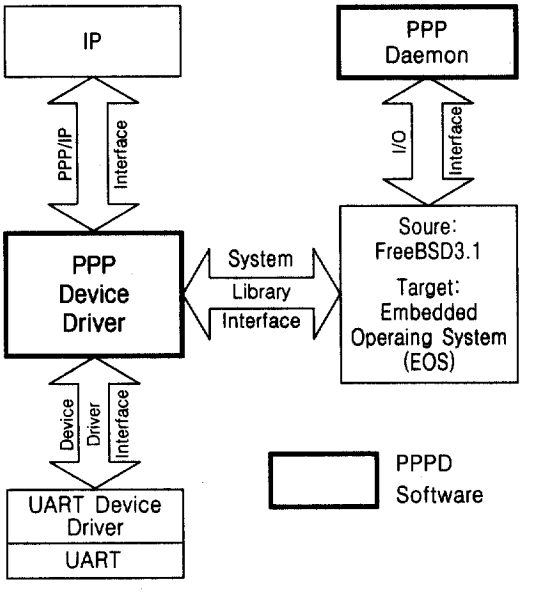
EOS는 구조적으로 범용 시스템과 많은 차이를 갖고 있다. EOS는 다수의 쓰레드(thread)를 갖는 하나의 프로세스로 운영된다. 쓰레드를 사용함으로써, 범용 멀티 프로세스 시스템보다 빠른 Context Switch이 가능하고, 프로그램 제어 블록(Program Control Block : PCB)을 위한 메모리 공간을 줄일 수 있기 때문이다. 따라서, EOS 개발 환경에서는 쓰레드를 기본으로 하는 프로그램 개발 툴이 제공된다. EOS에서는 커널 모드와 사용자 모드의 구분이 없다. 프로그램 수행 시 발생하는 오류로 인해 전체 시스템이 멈추게 되는 단점이 있지만, 커널 메모리와 사용자 메모리와의 차이가 없기 때문에 효율적인 메모리 관리와 빠른 프로그램 수행이 가능하게 된다. 이외의 파일 시스템, 가상 메모리, 터미널 처리와 같은 시스템 서비스는 EOS에서는 제공하지 않는다. 하지만, 쓰레드 관리, 힙 메모리 관리, 세마포와 같은 기본적인 시스템 서비스는 제공된다.

본 연구에서 사용된 EOS는 IBM OS Open Realtime Operating System이다. 이 운영체제는 PowerPC를 기반으로 하는 하드웨어에 POSIX 호환, 실시간 처리와 같은 기능을 제공한다.

3. PPP 소스코드 분석

PPP 소스코드 분석의 가장 중요한 목적은 이식에 필요한 플랫폼 종속적인 부분을 정의하는 것이다. PPP 소프트웨어에서 종속적인 부분은 외부로 제공하거나 제공 받는 서비스에 대한 것이다. 이러한 외부와의 연결은 본 이식에서 인터페이스라고 정의하였다. (그림 3)은 PPP 이식을 위해 해결해야 하는 네 가지 인터페이스이다.

PPP/IP 인터페이스는 네트워크 계층의 IP 모듈과 링크 계층의 PPP 모듈이 연동되기 위해 필요하다. 이 인터페이스를 사용하여 IP에서부터의 데이터그램을 PPP로 출력하고, 직렬 포트에서 입력된 패킷 중 IP 데이터그램을 IP 모듈로 입력한다. 디바이스 드라이버 인터페이스는 직렬 포트로의 PPP 프레임 입출력에 필요한 인터페이스이다. 다음으로, System Library 인터페이스는 주로 커널에서 제공하는 서비스로 구성되어 있다. 마지막으로, PPP daemon에서 필요한 입출력함수를 위한 I/O 인터페이스이다. PPP daemon에서는 PPP/IP 인터페이스와는 독립적으로 패킷을 입출력 한다. 이러한 경우의 입출력은 궁극적으로 PPP 디바이스 드라이버에서 제공하는 함수를 사용한다.



(그림 3) PPP 이식을 위한 외부 인터페이스

본 절에서는 네 가지 인터페이스를 구체적으로 분석하기 위해 PPP 소프트웨어를 PPP discipline, 초기화, 입력, 출력, 큐의 관리, PPP daemon의 모듈로 나누어 분석하였다. PPP는 직렬 포트를 사용하기 때문에 직렬 포트를 위한 디바이스 드라이버를 우선 분석하였다. 다음에는 PPP를 위한 시스템 초기화 모듈에 대해 분석하였다. 상위 네트워크계층에서부터 오는 출력 데이터를 어떻게 처리하고, 직렬 포트로 입력된 개별 바이트가 어떠한 처리과정을 거쳐 상위 네트워크계층으로 전달되는지 분석하였다.

3.1 PPP Discipline

본 논문의 PPP 소프트웨어는 비동기 직렬 포트를 사용한다. UNIX 계열의 OS에서 비동기 직렬 포트용 디바이스 드라이버를 TTY라고 한다. 이러한 TTY는 두 개의 계층으로 나눌 수 있는다. 하위에는 직렬 포트 하드웨어 디바이스 드라이버가 있고, 상위에는 line discipline이 있다[17]. 하드웨어 디바이스 드라이버는 Universal Asynchronous Receiver/Transceiver(UART)

의 레지스터(Status, Rx, Tx)를 접근하고, UART에서 발생하는 하드웨어 인터럽트를 처리하는 함수로 구성되어 있다. Line discipline은 라인 편집과 같은 기존의 터미널을 위해 필요한 여러 기능을 제공하는 함수들로 구성되어 있다. 즉, Line discipline은 사용자 I/O 시스템 호출과 하드웨어 디바이스 드라이버 사이에 존재하여 터미널 입출력에 필요한 여러 기능을 제공한다. BSD에서 Line Discipline은 8개의 함수로 구성되어 있다. PPP 디바이스 드라이버는 이 8개의 함수를 구현하였는데, <표 2>는 PPP를 위한 PPP Discipline이다.

〈표 2〉 PPP Discipline

함수명	Called from	기 능
pppopen()	above	PPP discipline을 열어줌
pppclose()	above	PPP discipline을 닫음
pppread()	above	PPP daemon에서 패킷 입력
pppwrite()	above	PPP daemon에서 패킷 출력
pppioctl()	above	PPP discipline의 제어 값을 읽거나 변경
ppprintr()	Below	직렬 포트에 새 바이트 입력될 때 수행되는 interrupt handler
pppstart()	Below	큐의 전송이 완료될 때 수행되는 interrupt handler
ttymodem()	Below	모뎀 제어 신호가 변경될 때 수행되는 interrupt handler

처음 다섯 개의 함수는 시스템 함수로서 사용자에게 제공된다. 다음 세 개의 함수는 하드웨어 인터럽트 처리함수로서 비동기적인 event가 발생되었을 경우 수행된다. Pppread()와 pppwrite() 함수는 PPP daemon에서 LPC, Authenticate, NCP와 같은 패킷의 입출력에 사용된다.

3.2 PPP Initialization

PPP를 시스템에 등록하기 위해 시스템 부트 시 PPP 초기화를 한다. PPP 초기화는 1) PPP 인터페이스를 위한 ifnet 구조를 등록 2) 소프트웨어 인터럽트 등록 3) PPP discipline 등록의 세단계로 이루어진다. Ifnet 구조는 모든 인터페이스 하드웨어마다 정의되어져야 하는 구조로 각 인터페이스의 종류, 관련된 함수들의 포인터, 데이터를 저장할 수 있는 큐들에 대한 정보를 갖고 있다. 한 시스템이 여러 인터페이스를 갖고있을 경우 이들을 Linked List로 연결하여 관리한다. 따라서, 초기화 시에 PPP에 대한 ifnet 구조의 모든 정보를 구성한 후, 이를 if_attach() 함수를 사용하여 시스템에 등록한다.

하드웨어 인터럽트 처리함수는 새로운 데이터를 큐에 저장하고 패킷에 대한 구체적인 처리는 하위 priority의 루틴이 처리하게 된다. 이러한 하위 priority의 처리를 위해 소프트웨어 인터럽트를 사용한다. Pppintr() 함수가 이러한 소프트웨어 인터럽트 처리함수의 역할을 수행하게 되는데, register_netisr() 함수를 사용하여 이를 시스템에 등록한다. 마지막으로, PPP를 위한 PPP discipline, 즉 pppopen(), pppread()와 같은 8개의 함수들을 등록함으로써 초기화를 완료한다.

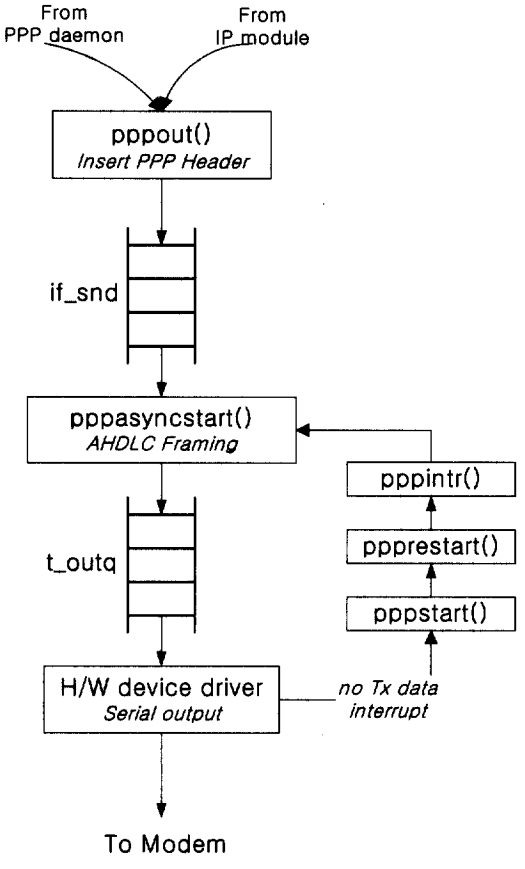
3.3 PPP Output Processing

직렬 포트로의 PPP 출력은 다음 두 가지 방법에 의해 사용된다. 첫째, PPP daemon 프로그램에서 LCP, Authenticate, NCP 패킷 전송이다. 이 경우, 사용자 프로그램에서 write() 함수를 호출하면 PPP discipline에서 정의된 함수 pppwrite()가 호출된다. 두 번째의 경우는, 네트워크 계층인 IP 모듈에서 데이터그램을 전송이다. pppout() 함수는 상위 계층에서부터 전달된 데이터에 PPP 헤더를 붙여 큐(if_snd)에 저장한다. 즉, address (0xFF), control(0x03), protocol 필드를 헤더로 삽입한다. Pppasyncstart() 함수는 큐에 저장된 패킷을 AHDLC 프레임으로 만들어 이를 TTY의 큐(t_outq)에 저장한다. 이 함수에서는 0x7E를 프레임 맨 앞과 맨 뒤에 삽입하고, 0x7D를 사용하여 필요한 바이트를 예외 처리한다. 만약 t_outq에 데이터를 삽입할 수 없을 경우, pppasyncstart() 함수는 종료한다. 나머지 데이터는 t_outq의 데이터가 없을 경우 발생하는 인터럽트(no Tx data interrupt)에 의해 다시 pppasyncstart()에 의해 출력된다. (그림 4)는 PPP 출력을 위한 흐름도이다.

IP 데이터그램의 출력은 함수 호출 방법에 의해 수행된다. 우선, PPP 초기화에서 PPP를 링크 계층으로 등록하게 되는데, 이 때 인터페이스 구조체의 멤버 중 if_output이라는 멤버를 PPP로 출력하는 함수, pppout()로 정의한다. IP 모듈에서 데이터그램을 완성한 후, PPP로 출력하려면, (*ifp->if_output)(ifp, m, dst, rt)의 방법으로 pppout() 함수를 호출한다.

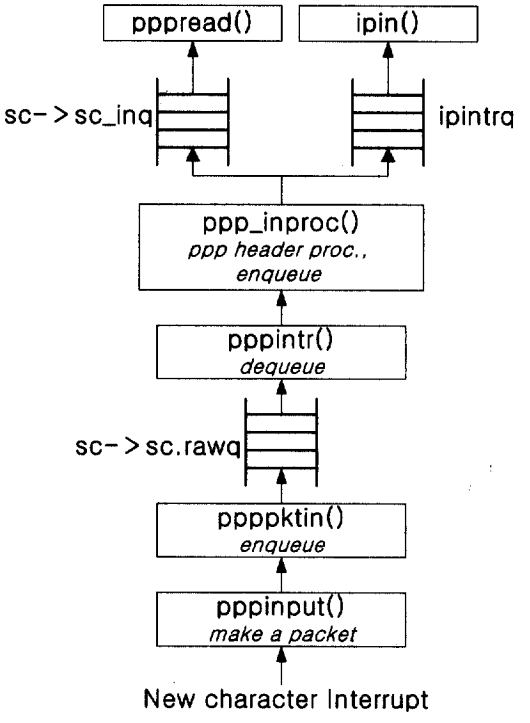
3.4 PPP Input Processing

직렬 포트를 통해 입력되는 데이터는 시스템과 비동기이다. 즉, 입력 모듈 하드웨어 인터럽트 처리함수가 트리거 되면서 시작된다. 새로운 바이트가 입력되면



(그림 4) PPP 디바이스 드라이버의 출력 흐름도

앞에서 정의된 PPP discipline에 의거하여 pppinput() 함수가 시작된다. 이 함수에서의 가장 중요한 기능은 개별적으로 입력된 바이트들로부터 하나의 PPP 패킷을 완성하는 것이다. 하나의 PPP 패킷이 완성되면 ppppkin() 함수를 호출하여 이 패킷을 큐(sc->sc_rawq)에 저장하고(enqueue) 소프트웨어 인터럽트를 구동한다. 이로서 하위 계층의 인터럽트 처리함수는 종결되고, 하위 priority의 소프트웨어 인터럽트 처리함수인 pppintr() 함수가 적절한 시기에 호출된다. Pppintr()에서는 큐에 저장된 새로 도착한 패킷을 꺼내서 ppp_inproc() 함수를 호출한다. Ppp_inproc() 함수는 PPP 패킷의 헤더를 분석하여 적절한 큐에 저장하고, IP의 경우, 소프트웨어 인터럽트로 IP 입력 모듈에게 새로운 데이터그램의 도착을 알린다 예를 들어, PPP 헤더의 분석 결과 IP 데이터그램이면 ipintq에 저장하고, schednetisr(NETISR_IP) 함수를 호출하여 소프트웨어 인터럽트를 발생시킨다. (그림 5)는 위와 같은 내용을 흐름도로 표현하였다.



(그림 5) PPP 디바이스 드라이버의 출력 흐름도

3.5 System Library

앞 절의 입출력 처리과정에서와 같이 네트워크 프로그램에서는 모듈 간의 메시지 전달을 위해 메시지 큐를 많이 사용한다. 일반적으로 모듈간의 메시지 전달 방법은 크게 두 가지로 구분할 수 있다[18]. 첫 번째 방법은 함수 호출 방법이다. 메시지를 함수의 입력 인자로 정의하여 전달한다. 두 번째 방법으로는 메시지 큐로 메시지를 전달하는 방법이다. 메시지 큐 방법은 함수 호출 방법보다 시스템 자원이 더 많이 필요하다. BSD의 네트워크 계층과 링크 계층간의 출력은 함수 호출 방법을 사용하는 반면, 입력은 메시지 큐를 사용한다. 네트워크 계층으로의 입력을 메시지 큐로 하는 이유는 다음 두 가지이다. 첫 째, 다수의 링크 계층으로부터 데이터를 받기 위해서 필요하다. Ethernet, PPP와 같은 다수의 링크 계층을 갖고 있을 경우 이를 하나의 경로로 입력할 필요가 있다. 둘 째, 하드웨어로부터의 데이터를 잃지 않기 위해 필요하다. 새로운 데이터가 하드웨어 입력되면 이를 처리하는 함수는 신속하게 처리를 하고

다음 데이터를 기다려야 된다. 만약 이의 처리가 지체되면 다음에 입력되는 데이터를 잃어 버릴 수 있다. 따라서, 최소한의 처리과정 후 이를 메시지 큐에 저장하여 다음 데이터를 위해 준비해야 한다.

이러한 메시지 큐를 관리하기 위해서는 Serialization의 문제를 해결해야 한다. Serialization의 문제는 여러 쓰레드가 하나의 큐를 접근할 때 발생한다. 큐에 메시지를 저장하는 도중에 context switching이 발생하여 다른 쓰레드가 큐에 접근할 경우 예측할 수 없는 오류가 발생할 수 있다. 이러한 Serialization 문제를 해결하기 위해 BSD에서는 splimp(), splx() 함수로 모듈이 큐에 접근할 경우 다른 모듈이 큐에 접근할 수 없도록 한다.

마지막으로, 메시지 큐를 관리하기 위해 소프트웨어 인터럽트가 필요하다. PPP 입력 처리 모듈에서 패킷이 정상적으로 처리되어 이 메시지가 IP 입력 모듈의 큐에 저장되면, PPP 입력 처리 모듈은 IP 입력 모듈에게 새로운 메시지의 도착을 알려야 한다.

네트워크 프로토콜에서 상대방의 응답을 기다리기 위해 타이머 서비스가 필요하게 된다. 이러한 타이머 서비스는 PPP 디바이스 드라이버와 PPP daemon 두 모듈에서 사용하고 있다.

3.6 PPP Daemon

PPP daemon은 (그림 2)의 closed phase에서 connected phase로 전이하기 위해 필요한 프로토콜들이 구현되어 있다. 일반적인 컴퓨터 시스템에 구현된 PPP를 분석할 때 고려해야 할 중요한 점은, PPP는 대칭형 Peer-to-Peer 프로토콜이라는 것이다. 즉, 클라이언트/서버프로토콜에서와 같이 접속을 요청하는 시스템과 요청된 접속을 제공하는 시스템의 구분이 없다. PPP daemon을 구현하기 위해서는 RFC1661에 있는 state transition diagram에 대한 이해가 필요하다. 이러한 state transition diagram은 Finite State Machine으로 구현되어져 있는데, 이 알고리듬은 매우 복잡하게 설계되어져 있고, 이식에 커다란 영향을 주지 않기 때문에 자세한 분석은 필요하지 않다.

이식을 위해 분석해야 할 가장 중요한 사항은 직렬 포트로의 입출력 함수이다. 하지만, 이러한 함수는 read(), write(), iocntrl()과 같이 매우 소수이다. PPP daemon은 터미널 사용자 인터페이스를 제공하기 때문에, 임베디드 시스템으로 이식할 경우 터미널 입출력에 대한 부분을 수정해야 한다.

사용자 모드에서의 read(), write() 함수를 호출하기 위해서 우선 open() 함수로 디바이스 드라이버의 file descriptor를 얻은 다음, 이 file descriptor를 입력 인자로 하여 read(), write() 함수를 호출한다. 이 후, OS에서 적절한 처리 과정을 거쳐 PPP 디바이스 드라이버의 함수를 호출한다. PPP discipline과 같은 character device에서는 사용자 영역(user space)의 데이터를 디바이스로 직접 전달한다. Block device의 경우는 커널에 캐쉬(cache)를 사용해 데이터를 전달한다. PPP 이식을 위한 분석 시 가장 중요한 사항은 함수의 입력 인자들의 처리이다. 예를 들어, 데이터를 디바이스에 쓰기 위해 write()를 호출하면 입력 인자들은 처리를 거쳐 uio 구조체에 저장되고, uio구조체를 입력인자로 pppwrite()가 호출된다. 이러한 데이터 처리를 하는 이유는, 1) 여러 디바이스에 통일된 함수 이름, 즉 read, write을 사용할 수 있고, 2) 여러 조각의 사용자 버퍼를 동시에 접근할 수 있고, 마지막으로 3) 커널 모드에서 사용자 메모리 영역에 접근하기 위해서이다. 커널에서는 uio 구조체를 접근하기 위해서 uiomove() 함수를 사용한다.

4. PPP 이식

범용 시스템과 임베디드 시스템의 차이점과 임베디드 시스템에서의 개발 환경의 특징은 2.4절에서 설명하였다. 4장에서는 앞 장에서 분석한 결과를 기초로 하여 임베디드 시스템으로 이식에 대해 기술하였다. 우선, (그림 3)에서 정의된 외부 모듈과의 인터페이스의 이식 방법을 설명하였다. 다음으로, BSD보다 간략화 된 입출력 모듈의 이식과 PPP daemon의 이식에 대해 기술하였다.

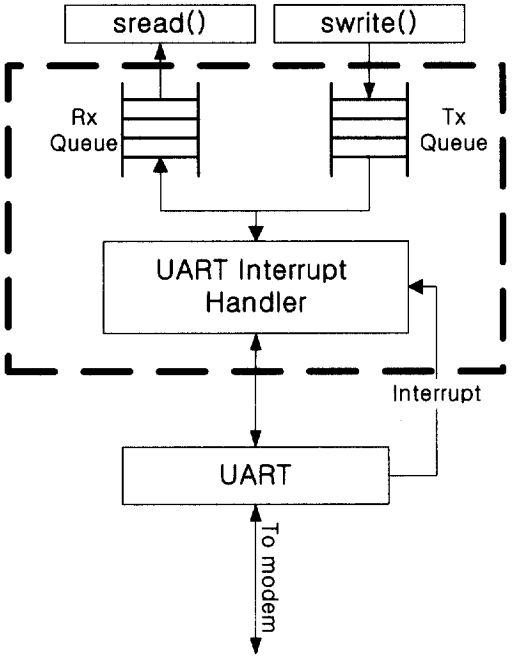
4.1 PPP/IP 인터페이스

본 논문에서 사용한 임베디드 시스템의 IP 모듈은 BSD와 호환이 되는 인터페이스를 갖고 있기 때문에 PPP/IP 인터페이스는 특별한 이식이 필요하지 않다. 따라서, 임베디드 시스템에서도 PPP 출력을 위한 함수 호출과 IP 입력을 위한 메시지 큐를 BSD와 동일하게 사용하였다.

4.2 디바이스 드라이버 인터페이스

BSD에서는 Line discipline과 하드웨어 디바이스 드

라이버를 포함하는 TTY 디바이스 드라이버를 사용하지만, 임베디드 시스템에서는 이와 같은 복잡한 디바이스 드라이버가 필요하지 않다. 직렬 포트는 PPP를 위해서만 사용하기 때문이다. 따라서, BSD의 PPP 소스코드에서 TTY에 관련된 부분을 삭제, 수정해야 된다. 본 논문에 사용된 임베디드 시스템의 직렬 포트용 디바이스 드라이버는 BSD보다 간략화 되어 있다. (그림 6)은 임베디드 시스템에서의 직렬 포트용 디바이스 드라이버의 동작 원리이다. 이 디바이스 드라이버는 두개의 입출력용 큐와 UART에서부터의 하드웨어 인터럽트를 처리하기 위한 인터럽트 처리함수로 구성되어 있다. 직렬 포트 출력용 swrite() 함수는 데이터를 출력 큐에 저장하고, 직렬 포트 입력용 sread() 함수는 데이터를 입력 큐에서 읽는다. 실제 UART로의 데이터 입출력은 인터럽트 처리 함수가 수행한다.



(그림 6) 임베디드 시스템의 직렬 포트 디바이스 드라이버 구조

PPP를 위한 인터페이스 구조체(ifnet)와 TTY 구조체는 포인터로 연결이 되어 있는데, 둘 중에 하나만 알아도 나머지 하나의 포인터를 찾을 수 있다. PPP 디바이스 드라이버에서는 주로 TTY 구조체가 함수의 입력 인자로 많이 사용된다. 본 이식에서는 TTY를 사용하지 않으므로, PPP의 인터페이스를 전역 변수로 정의하여 소스코드 어디에서나 접근할 수 있도록 하였다.

4.3 I/O 인터페이스

BSD에서는 디바이스 드라이버를 시스템에 등록하여 read(), write()와 같은 함수를 디바이스 드라이버함수로 매핑 하는 메커니즘을 제공한다. 임베디드 시스템에서는 이러한 메커니즘을 직접 구현하였다. 이를 위해 두개의 보조 함수, readuio(), writeuio()를 구현하였다. 출력의 경우, write() -> writeuio() -> pppwrite()의 순서로 함수가 호출된다. writeuio()에서는 write()에서의 입력 인자로 uio 구조체를 만든 후, 이 uio 구조체로 pppwirte() 함수를 호출한다. 입력의 경우에도 마찬가지로 readuio() 보조 함수로 입력 인자를 처리하였다. uiomove() 함수는 uio 구조체를 참조하여 데이터를 쓸 것인지 아니면 읽을 것인지 결정한다. BSD에서는 커널 메모리와 사용자 메모리를 접근하기 때문에 매우 복잡하지만, 임베디드 시스템에서는 커널과 사용자의 차이가 없으므로 구현이 매우 용이하다.

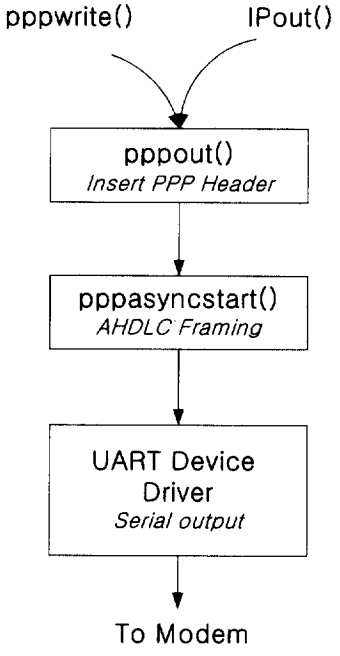
4.4 System Library 인터페이스

2.3절에서 설명한 것과 같이, 임베디드 시스템의 운영체계는 기본적인 시스템 라이브러리를 제공하고 있다. PPP daemon과 PPP 디바이스 드라이버에는 10 msec간격의 타이머를 필요로 하는데 이러한 타이머 서비스도 임베디드 시스템이 제공하고 있다. 메시지 큐의 구현과 Serialization 문제 해결을 위해 필요한 쓰레드와 세마포도 임베디드 시스템에서 제공하고 있다. 쓰레드를 사용할 경우 네트워크 하위 계층이 상위 계층보다 쓰레드 priority를 높게 해주어야 한다. Serialization 문제 해결을 위한 splimp(), splx() 함수는 세마포로 구현하였다. Splimp() 함수는 세마포의 상태를 확인하여, 다른 쓰레드가 큐를 사용할 경우, 접근을 제한하도록 하였다. 큐에 접근이 가능하여 사용을 마친 후, 다른 쓰레드가 큐에 접근할 수 있도록 splx() 함수로 세마포 신호(signal)를 전달한다. 이 세마포는 전역 변수로 정의하고 시스템 초기화에 이 세마포를 초기화하였다.

4.5 PPP Output Processing Module

본 논문에서의 임베디드 시스템에서는 BSD에서와 같이 다수의 큐가 필요하지 않다. 링크 계층의 프로토콜 중 PPP 하나만 사용하고, 비교적 느린 직렬 포트를

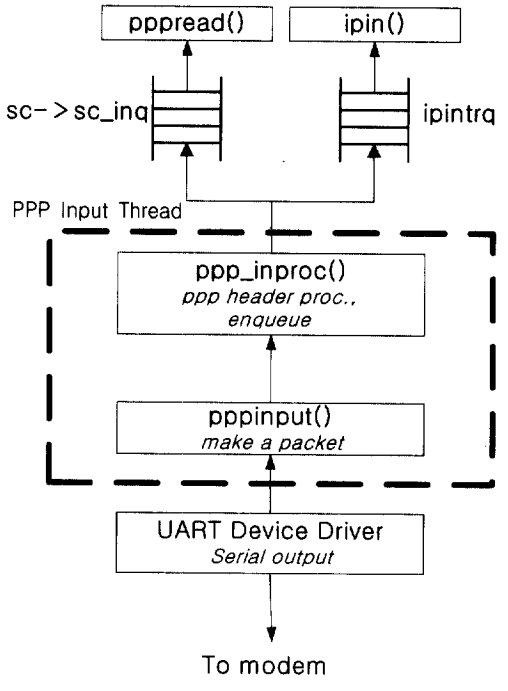
사용하기 때문이다. 따라서, 본 이식에서는 (그림 4)에서의 pppout()과 pppasyncstart() 사이의 큐(if_snd)를 제거하고 pppout() 함수에서 직접 pppasyncstart() 함수를 호출하도록 하였다. (그림 7)은 임베디드 시스템에서 구현된 PPP 출력 흐름도이다. 이와 같이 구현하여 코드의 크기와 메시지 큐를 위해 필요한 메모리 자원을 줄일 수 있다.



(그림 7) 임베디드 시스템의 PPP 출력 흐름도

4.6 PPP Input Processing Module

PPP 출력은 함수 호출에 의해 시작되는 반면, 입력 처리는 새로운 데이터의 입력에 의해 시작된다. 따라서, 입력 처리는 쓰레드로 구현하였다. 새로운 데이터가 직렬 디바이스 드라이버의 큐에 입력되면, 소프트웨어 인터럽트로 이 쓰레드에 새로운 데이터의 도착을 알린다. 이러한 방법으로 쓰레드가 시작되면, 직렬 디바이스 드라이버에 있는 입력 큐의 모든 데이터를 처리한다. 큐에 저장된 데이터를 전부 처리하면 다시 디바이스 드라이버에서의 소프트웨어 인터럽트를 기다리도록 구현하였다. PPP 입력 처리에서도 출력처리와 같은 방법으로 ppppktin()과 pppintr() 함수 사이의 큐(sc->sc.rawq)를 제거하였다. (그림 8)은 이러한 입력 처리 모듈의 흐름도이다.



(그림 8) 임베디드 시스템의 PPP 입력 흐름도

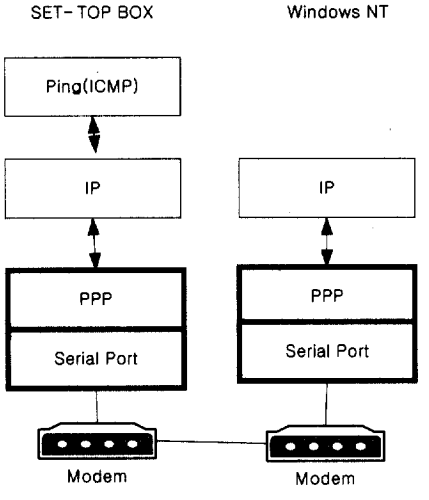
4.7 PPP Daemon

PPP daemon은 사용자 모드에서 실행되고, 터미널로 사용자의 입력과 이에 대한 출력을 한다. user verification, multi-user와 같은 이식의 문제점은 임베디드 시스템의 single user, single process의 특성으로 용이하게 해결할 수 있었다. 사용자 모드이기 때문에 코드 분석 시 디버깅 툴을 손쉽게 사용할 수 있어서 분석에 용이한 점도 있지만, 시스템에 종속된 코드가 많지 않아, 커널 소스코드와 같이 상세한 분석이 필요 없다. read(), write()와 같은 I/O 시스템 호출의 문제는 앞 절의 시스템 라이브러리 인터페이스에 기술하였다.

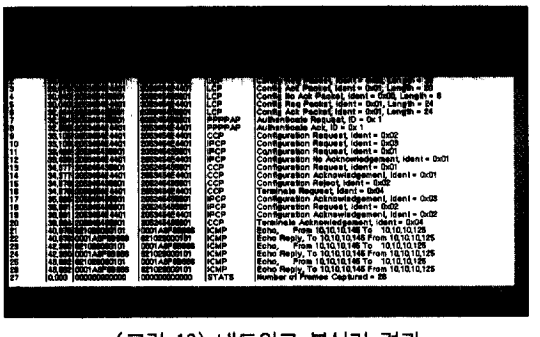
5. 이식 테스트 및 검증

본 장에서는 임베디드 시스템에 이식되어진 PPP 소프트웨어가 정상적으로 동작하는 것을 검증하기 위해 (그림 9)와 같은 테스트 베드를 구성하여 전송되는 네트워크 패킷을 분석하였다. 임베디드 시스템은 위성방송 수신용 셋톱박스를 사용했다. 이 셋톱박스에 U.S Robotics사에서 제공하는 외부 모뎀을 사용했다. 서버 시스템은 Windows NT의 원격접속 서버 프로그램을

이용했다. (그림 10)은 셋톱박스와 Windows NT와 (B)의 패킷을 네트워크 모니터 프로그램으로 갈무리 한 결과이다.



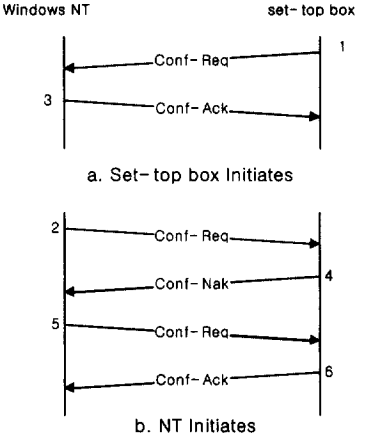
(그림 9) 테스트 베드의 구성도



(그림 10) 네트워크 분석기 결과

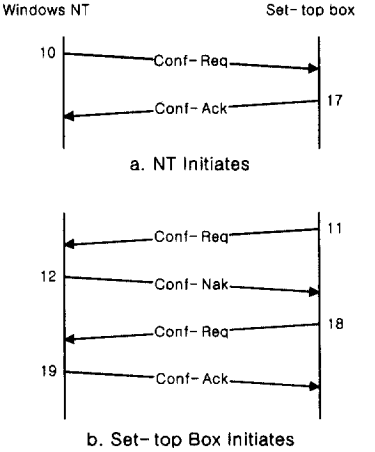
(그림 11)은 이러한 LCP 패킷 교환에 대한 요약이다. 셋톱박스는 패킷 1을 전송하여 옵션을 요청하면, 서버는 패킷 3으로 이 옵션을 승인한다. 서버가 원하는 옵션을 패킷 2로 요청하면, 셋톱박스는 패킷 4로 인증 프로토콜을 CHAP가 아닌 PAP로 수정을 요구한다. 서버는 이러한 수정 요구를 받아들여 인증 프로토콜을 PAP로 하는 패킷 5를 전송한다. 마지막으로 셋톱박스는 패킷 6을 전송하여 패킷 5의 옵션을 승인한다.

셋톱박스와 서버가 LCP로 옵션을 결정한 후, 사용자 인증이 진행된다. 패킷 7은 LCP로 결정된 프로토콜인 PAP를 사용하여 셋톱박스가 사용자 인증을 요청하고, 서버는 패킷 8로 사용자 인증을 승인한다.



(그림 11) LCP Negotiation

사용자 인증이 끝나면, 양측은 IP를 사용하기 위해 필요한 옵션을 결정한다. 우선 서버는 패킷 10을 사용하여 Van Jacobson Compression 옵션과 서버의 IP주소를 셋톱박스로 전송한다. 셋톱박스는 이에 대한 승인을 패킷 17을 사용하여 전송한다. 패킷 11은 셋톱박스가 원하는 IP 옵션의 요청이다. 이 때, 서버는 IP주소를 요청하는 의미로 IP 주소를 0.0.0.0으로 전송하게 된다. 이에 대한 응답으로, 서버는 Nak으로, 셋톱박스가 사용할 수 있는 특정한 IP 주소를 패킷 12를 사용하여 전송한다. 셋톱박스가 사용할 IP 주소를 사용하여 패킷 18은 다시 요청하여, 서버는 이에 대한 승인을 패킷 19로 전송한다. 이러한 IPCP 패킷 교환을 (그림 12)에서 정리하였다.



(그림 12) IPCP Negotiation

CCP는 셋톱박스와 서버양측이 Terminate를 함으로서(패킷 16, 20) 사용하지 않게 된다. 이 후, 패킷은 셋톱박스에서 요청하는 ping(패킷 21, 23, 25)과 이에 대한 서버로부터의 응답(패킷 22, 24, 26)이다. 이와 같은 절차로 본 이식에서 개발한 PPP 소프트웨어가 정상적으로 동작하는 것을 검증할 수 있었다.

6. Discussion

본 논문에서 다룬 소프트웨어는 크게 두 종류로 구분이 된다. 하나는 사용자 모드에서 실행되는 PPP daemon이고, 다른 하나는 커널 모드에서 실행되는 PPP 디바이스 드라이버이다. PPP daemon은 사용자 모드에서 실행되기 때문에 PPP 디바이스 드라이버 보다 이식은 매우 용이하였다. PPP daemon의 알고리듬은 PPP 디바이스 드라이버보다 더욱 복잡하지만 플랫폼에 종속적인 부분이 매우 적고, 사용자 모드에서 동작하기 때문에 디버거를 사용한 분석과 이식 테스트가 가능하였기 때문이다. 이와는 반대로, PPP 디바이스 드라이버는 이식의 여러 문제점을 갖고 있었다. 우선 프로그램의 분석이 용이하지 않다. 커널 모드에서 수행되기 때문에 디버깅이 매우 어렵기 때문이다. 또한, PPP 프로토콜은 링크 계층의 디바이스 드라이버를 포함하는 프로그램이기 때문에 하드웨어와 밀접한 관계를 갖고 있다. 즉, 임베디드 시스템의 직렬 포트 디바이스 드라이버와 PPP 프로그램과의 연동을 위해 기존의 인터페이스의 분석이 매우 중요하다. 따라서, 본 연구에서는 PPP 디바이스 드라이버를 상세하게 분석하였고, 이의 이식에 대해 중점적으로 설명하였다.

BSD 네트워크 소프트웨어의 메모리 관리는 Mbuf라는 구조체와 이에 대한 도구함수를 주로 사용한다[6]. PPP 디바이스 드라이버를 분석하기 위해서 Mbuf의 구조와 이에 연관된 함수 및 매크로를 자세히 이해해야 한다. 또한, PPP를 구현하기 위해 기존의 구현과 Mbuf에 대한 정의가 유사한 프로그램을 사용하여 이식하는 것이 매우 중요하다. Mbuf의 정의가 매우 상이하면 프로토콜 헤더의 처리과정을 크게 수정해야 하기 때문이다. 또한, 기존의 네트워크계층과의 인터페이스가 문제가 된다. IP 계층에서 내려오는 데이터그램은 Mbuf의 포맷이고 PPP 헤더를 처리한 후의 IP 데이터그램도 Mbuf의 포맷이기 때문이다.

본 이식을 통해 코드의 크기와 힙 메모리(Heap Memory) 사용량을 줄일 수 있었다. PPP Daemon에서는 터미널 입출력을 위한 코드가 제거되었고, PPP 디바이스 드라이버에서는 TTY와의 인터페이스 부분과 입출력 처리 과정이 상당히 간략화 되었다. 전체적으로는, 사용자 인증과 시스템 보안에 대한 코드를 제거하였다. 또한, PPP 패킷의 입출력 처리과정에서 큐의 사용을 최소화하여 힙 메모리 사용량을 줄일 수 있었다.

네트워크 소프트웨어 이식을 위해 여러 가지의 툴이 필요하다. 우선 모든 프로그램 개발에 필수인 디버거가 매우 중요하다. 프로그램의 분석과 이식 테스트를 위해, 최소한 Break, Step과 메모리의 내용을 볼 수 있는 디버거가 필요하다. 만약 이러한 디버거가 없다면, 이식된 프로그램을 emulation 할 수 있는 환경을 초기에 구성해야 한다. 또 하나의 중요한 툴은 네트워크 패킷 분석기이다. 이는 네트워크 프로그램 디버깅에 필수적이고, 프로그램 이식의 최종적인 검증을 위해 매우 중요하다. 물론, 패킷 분석기를 사용하기 위해서는 TCP, IP등과 같은 상위 계층의 프로토콜에 대한 자세한 지식이 필요하다. 본 이식에 사용된 임베디드 시스템은 적절한 직렬 포트 디바이스 드라이버가 없었다. 따라서, 직렬 포트 디바이스 드라이버를 개발하였다. 이 때, AHDLC의 프레이밍과 직렬 포트의 hardware 디바이스 드라이버를 테스트하기 위해서 실제 직렬 선(Rx, Tx)에 연결하여 전송되는 데이터를 분석할 수 있는 툴이 필요하였다.

본 이식의 과정은 소스분석, 이식(컴파일, 링크), 테스트의 세 단계를 하나의 주기로 하여 여러 번 반복되었다. 첫 주기는 주로 네트워크 출력 부분에 대한 프로그램에 대해 수행되었다. PPP 출력부분은 모두 함수 호출로 구성되어 있기 때문에 분석이 용이하고 또한, 출력되는 내용을 서버의 네트워크 모니터로 검증할 수 있기 때문이다. 이러한 출력 프로그램이 완성된 후, 네트워크 입력 부분에 대해 수행되었다. 네트워크 출력에 대한 응답(입력 패킷)은 네트워크 모니터를 통해 미리 알고 있기 때문에 디버깅이 용이하였다.

7. 결 론

본 논문에서는 UNIX 계열의 OS에서 동작되는 PPP

소프트웨어를 임베디드 시스템으로 이식하기 위해 풀어야 할 문제점을 찾고 이를 해결하는 방법을 제시하였고, 이를 실제 이식에 사용함으로써 그 실용성을 증명하였다. 본 연구에 사용된 PPPD 소프트웨어는 사용자 모드에서 실행되는 PPP daemon과, 커널 모드에서 실행되는 PPP 디바이스 드라이버로 구성되어 진다. PPP daemon은 사용자 모드의 소스코드이기 때문에 이식이 매우 용이하다. 반면에, PPP 디바이스 드라이버의 이식은 커널 모드의 소스 코드이고, 하드웨어와 밀접한 연관성을 갖고 있기 때문에 많은 문제를 해결해야 했다. 이와 같은 문제들을 해결하기 위해, 본 논문에서는 PPP 소프트웨어를 기능적으로 여러 개의 모듈로 구분하여 소스 코드를 분석하였다. 다음으로 이식에 필요한 외부 모듈과의 인터페이스에 대해 정의하였고, 프로그램 이식에 대해 설명하였다. 이러한 과정을 거쳐 이식된 프로그램으로 Windows NT의 원격 액세스 서비스와 PPP로 연결하였고, 시리얼 라인으로 교환되는 패킷을 네트워크 패킷 분석기로 정상적인 동작을 확인하였다.

앞으로 본 연구는 다음과 같은 분야에 활용될 수 있다.첫째, 소형 정보 가전기기와 같은 임베디드 시스템의 링크 계층으로서 PPP를 구현하는 데 활용할 수 있다.둘째, PPP 소스 코드에 대한 분석은 PPP 패킷 분석기를 구현하는 데 활용될 수 있을 것이다. 셋째, 제시한 이식 방법은 UNIX에서 구현된 네트워크 소프트웨어를 이식할 경우 활용할 수 있다. 마지막으로, 현재 활발히 연구 중인 사용자 모드 네트워크 프로토콜 구현에 활용 될 수 있다[19, 20].

참 고 문 헌

[1] IEEE, "Information Technology - Portable Operating System Interface(POSIX) - Part 1 : System Application Program Interface(API) [C Language]," *IEEE Std 1003.1*, 1996 Edition, IEEE, N. J., 1996.

[2] Krupczak, B., Kalvert, K. L., and Ammar, M. H., "Implementing Communication Protocols in Java," *IEEE Comm. Mag.*, Vol.36, No.10, pp.93-99, 1998.

[3] Hanish, A. A. and Dillon, T. S., "Object-Oriented Modelling of Communication Protocols for Re-Use," *IEEE Proceedings, Fourth International Conference on Computer Communications and Networks*, pp.18-26, 1995.

[4] Krupczak, B., Kalvert, K. L., and Ammar, M. H., "Increasing the Portability and Re-Usability of Protocol Code," *IEEE/ACM Trans. Networking*, pp. 445-459, 1997.

[5] ftp : //cs.anu.edu.au/pub/software/ppp/.

[6] Wright, G. R., and Stevens W. R., *TCP/IP Illustrated, Volume 2 The Implementation*, Addison-Wesley, 1997.

[7] Stevens W. R., *TCP/IP Illustrated, Volume 1 The Protocols*, Addison-Wesley, 1998.

[8] Carlson, J., *PPP Design and Debugging*, Addison-Wesley, 1998.

[9] Simpson W., *PPP in HDLC-like Framing*, RFC 1662, 1994.

[10] Simpson W., *The Point-to-Point Protocol(PPP)*, RFC 1661, 1994.

[11] Lloyd B., Simpson W., *PPP Authentication Protocols*, RFC 1334, 1992.

[12] Simpson W., *PPP Challenge Handshake Authentication Protocol(CHAP)*, RFC1994, 1996.

[13] McGregor, G., *The PPP Internet Protocol Control Protocol(IPCP)*, RFC1332, 1996.

[14] Haskin, E. and Allen, E., *IP Version 6 over PPP*, RFC2023, 1996.

[15] Simpson, W., *The PPP Internetworking Packet Exchange Control Protocol(IPXCP)*, RFC 1552, 1993.

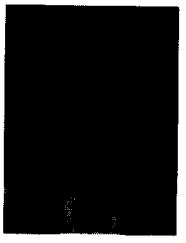
[16] Pall, G., *The PPP NetBIOS Frames Control Protocol(NBFCP)*, RFC 2097, 1997.

[17] McKusick, M. K., Bostic, K., Karels, M. J., and Quarterman, J. S., *The Design and Implementation of the 4.4 BSD Operating System*, Addison-Wesley, 1996.

[18] Schmidt, D. C., and Suda T., "Transport System Architecture Services for High-Performance Communications Systems," *IEEE Jour. on Selected Areas in Comm.*, Vol.11, No.4, pp.489-506, 1993.

[19] Thekkath, C. A., Nguyen, T. D., Moy E., and Lazowska, E.D., "Implementing network protocols at user level," *IEEE/ACM Trans. Networking*, Vol.1, No.5, pp.554-565, 1993.

[20] Bhoedjang, R. A. F., Ruhl, T., and Bal, H.E., "User-Level Network Interface Protocols," *IEEE Computer*, Vol.31, No.11, pp.53-60, 1998.



최 성 종

e-mail : chois@uoscc.uos.ac.kr

1982년 서울대학교 전기공학과 졸업(학사)

1984년 서울대학교 대학원 전기공학과(석사)

1992년 University of Florida, Dept. of Electrical Eng.(Ph.D.)

1993년~1996년 강릉대학교 전자공학과 전임강사

1996년~현재 서울시립대학교 전자전기공학부 조교수

관심분야 : 멀티미디어 시스템, 네트워크 소프트웨어, 임베디드 시스템