The Linux Plug-and-Play-HOWTO

David S.Lawyer *dave@lafn.org* <mailto:dave@lafn.org> v0.11, May 2000

남상현 (*nsh@asp-linux.co.kr* <mailto:nsh@asp-linux.co.kr>) v0.11 2000년 7월 29일

복잡한 플러그&플레이(Plug-and-Play)에 대해서 이해하고, 이것을 취급한다. 또 Linux 시스템이 플러그&플레이를 지원 하는 방법을 소개한다.

목 차

제 1 절 도입 **3**

1.1 저작권 표시, 등록 상표, 면책 사항과 신용 . . . 3

1.1.1 Copyright . . . 3

1.1.2 Disclaimer . . . 3

1.1.3 Trademarks. . . . 3

1.1.4 Credits . . . 3

1.2 향후 계획: 도와 주십시오. . . . 4

1.3 이 HOWTO 의 최신판에 대해서 . . . 4

제 2 절 PnP 의 할일: "Bus-Resources" 의 배정 **4**

2.1 플러그&플레이(PnP)란 무엇인가? . . . 4

2.2 컴퓨터에 의한 디바이스의 찾는 법(또는 그 반대로) . . . 4

2.3 I/O 어드레스등 . . . 5

2.4 IRQ –개요 . . . 6

2.5 DMA 채널 . . . 6

2.6 메모리 영역 . . . 7

2.7 디바이스와 드라이버의 양쪽에 있어서의 "resources" . . . 7

2.8 문제 . . . 7

2.9 시리얼 포토에 꽂은 디바이스를 PnP에 의해 검출 . . . 8

제 3 절 플러그&플레이(PnP)의 해결 방법 **8**

3.1 PnP의 도입 . . . 8

3.2 PnP 의 동작 (간단한 설명) . . . 9

3.3 PC 의 기동 . . . 9

3.4 버스 . . . 10

3.5 Linux 에서 PnP를 더 잘해야할 필요성 . . . 10

제 4 절 PnP BIOS 의 설정 **10**

4.1 PnP operating system을 갖고 있는가? . . . 11

4.1.1 Interoperability with Windows9x . . . 11

4.2 어떻게 버스 자원을 제어할까? . . . 12

4.3 설정을 리셋트할지? . . . 12

제 5 절 PnP 카드의 취급 방법 **12**

5.1 PnP 카드 취급의 소개 . . . 12

5.2 Disable PnP ? . . . 13

5.3 BIOS에서 PnP 설정 . . . 13

5.3.1 BIOS를 사용해 PnP의 설정을 하는데에 있어 . . . 13

5.3.2 BIOS의 ESCD 데이타베이스 . . . 14

5.3.3 Windows 를 이용한 ESCD 의 설정 . . . 14

5.3.4 새로운 디바이스의 추가 (Linux 또는 Windows 에 있어서) . . . 15

5.4 isapnp (isapnptools 의 일부) . . . 15

5.5 PCI Utilities . . . 16

5.6 Linux PnP를 만들기 위한 커널을 패치하여 주시오. . . . 16

5.7 Windows 에 의한 설정 . . . 16

5.8 디바이스 드라이버에 의한 설정 . . . 17

5.9 PnP 에 관련하는 소프트웨어와 문서 . . . 17

제 6 절 Tell the Driver the Configuration **17**

6.1 소개 . . . 17

6.2 시리얼 포트 드라이버: setserial . . . 18

6.3 사운드 카드 드라이버 . . . 18

6.3.1 OSS-Lite . . . 18

6.3.2 OSS (Open Sound System) 와 ALSA . . . 18

제 7 절 현재의 설정을 알기 위해서는? **18**

7.1 Boot-time Messages . . . 19

7.2 디바이스 드라이버는 어떻게 설정되고 있는가? . . . 19

7.3 하드웨어 디바이스는 어떻게 설정되고 있는가? . . . 19

제 8 절 부록 **20**

8.1 어드레스 . . . 20

8.1.1 ISA 버스의 설정 어드레스 (Read-Port등) . . . 20

8.1.2 어드레스의 범위 . . . 20

8.1.3 어드레스 공간 . . . 21

8.1.4 어드레스 범위 체크 (ISA에 있어서 I/O 어드레스의 충돌 검사) . . . 21

8.1.5 메모리를 통한 직접 통신 21
8.2 인터럽트 –상세 21
8.3 PCI 의 인터럽트 22
8.4 Isolation 23

제 1 절 도입

1.1 저작권 표시, 등록 상표, 면책 사항과 신용

역주: 라이센스 관계에 대해서는 원문을 그대로 나타냅니다.

1.1.1 Copyright

Copyright (c) 1998-2000 by David S. Lawyer *dave@lafn.org* <mailto:dave@lafn.org>

Please freely copy and distribute (sell or give away) this document in any format. Forward any corrections and comments to the document maintainer. You may create a derivative work and distribute it provided that you:

1. Send your derivative work (in the most suitable format such as sgml) to the LDP (Linux Documentation Project) or the like for posting on the Internet. If not the LDP, then let the LDP know where it is available. Except for a translation, send a copy to the previous maintainer's url as shown in the latest version.
2. License the derivative work in the spirit of this license or use GPL. Include a copyright notice and at least a pointer to the license used.
3. Give due credit to previous authors and major contributors.

If you're considering making a derived work other than a translation, it's requested that you discuss your plans with the current maintainer.

1.1.2 Disclaimer

While I haven't intentionally tried to mislead you, there are likely a number of errors in this document. Please let me know about them. Since this is free documentation, it should be obvious that I cannot be held legally responsible for any errors.

1.1.3 Trademarks.

Any brand names (starts with a capital letter) should be assumed to be a trademark). Such trademarks belong to their respective owners.

1.1.4 Credits

Daniel Scott proofread this in March 2000 and found many typos, etc.

1.2 향후 계획: 도와 주십시오.

사실, 의견, 논리, 단어의 철자법, 문법, 문장의 이해, 링크 등에 문제가 있으면, 필자에게 알려주세요. 그리고 1 개월이상의 오래된 버전의 문서라면, 먼저 최신판인지를 확인하기 바란다. 또, 이 문서에 관련한다고 생각되는 정보가 있으면 알려주세요.

필자는 isapnptools 에 대해서나 David Howells 씨의 커넬 패치에 대해서도 자세하게 조사하지 못했다 (조사하려고 생각은 하고있다). BIOS는 PnP 를 어느 것처럼 초기화하는지(이것은 BIOS 에 의해 다릅니다)와, Windows9x 에 있는 ESCD 의 갱신 방법에 대해서도 완전하게는 이해하고 있지 못하다. 따라서, 이 HOWTO 는 미완성이고, 부정확할지도 모른다(실수가 있으면 알려 주세요.) 이 HOWTO 에서는, 필자가 정확한 답을 모르는 것을 나타내는 기호로서 「??」를 몇개정도 사용 하고있다.

1.3 이 HOWTO 의 최신판에 대해서

Plug-and-Play-HOWTO 의 최신판은 대체로 매월나오고, LDP 와 그미 러사이트에서 열람또는 입수할 수 있다. 미러사이트의 일람은 *linuxdoc.org/mirrors.html* <http://linuxdoc.org/mirrors.html> 에 있다. 이 문서는 다양한 포맷으로 존재한다. 최신판의 날짜를 확인하고 싶을 뿐이라면, *linuxdoc.org/HOWTO/Plug-and-Play-HOWTO.html* <http://linuxdoc.org/HOWTO/Plug-and-Play-HOWTO.html> 을 보면 좋겠다. 당신이 읽고 있는 이 문서의 버젼은 v0.11(2000년 5월판)이다. 이번새롭게 추가된 항목은 scanport 유틸리티, 많은 오자 수 정, setpci 의 간단한 사용법이다.

제 2 절 PnP 의 할일: "Bus-Resources" 의 배정

2.1 플러그&플레이(PnP)란 무엇인가?

매우 간단하게 설명하면, 플러그&플레이는 모뎀과 네트워크 카드, 사운드 카드등의 각종 하드웨어(장치)를 찾아낸 장소를 자동적으로, 소프트웨어에 알려준다. 플러그&플레이의 일은, 물리적 장치와 이것을 조작하는 소프트웨어(디바이스 드라이버)와 일치시키고, 디바이스와 드라이버 사이에 통신「채널」을 만드는 것이다. 이것을 실현하기 위해서, PnP는 아래의 "bus-resources" 을 드라이버와 하드웨어의 양쪽에 할당한다: I/O 어드레스, IRQ, (ISA 패스만)DMA 채널, 메모리 영역이다. 이것들 4 개의 bus-resources가 어떤 것인지 모르면, 뒤에 기술한 I/O 어드레스, IRQ, DMA 채널, 메모리 영역의 절을 읽어 주세요. 또 이것들의 bus-resources 중 3 개에 관한 Linux Gazette 의 기사가 Introduction to IRQs, DMAs and Base Addresses 에 있다. 한 번 이것들의 bus-resources를 분배시키면 (그리고 옳바른 드라이버가 인스톨되면), /dev 디렉토 리에 있는 디바이스의 파일을 사용할 수 있게 된다.

이러한 bus-resources PnP 배분을「설정(configuring)」라고 불리우는 것도 있지만, 이것은 저레벨의 종류 설정에 지나지 않는다. 즉, PnP 를 최대한 이용한 경우라도 디바이스 설정의 대부분은 PnP 이외에서 행해진다. 예를 들면 모뎀의 설정이라면,「초기화문자열」이 I/O 어드레스의 "channel" 을 사용해 모뎀에 보내진다. 모뎀에 문자열을 보내기 위해서 사용되는 "channel"은 PnP를 할당한 것입니다만,「초기화문자렬」그것은 PnP와는 무관하다. 시리얼 포트의 속도(및 다른 파라미터의 다수)의 설정은, 사용자가 실행된(부팅시에 자동적으로 실행되는 것도 꽤 있다.) 프로그램으로부터 디바이스 드라이버로 메세지를 보내는 것에 의해 실행된다. 이 설정도 역시 PnP과는 관계 없다. 이와 같이, PnP의 화제 에 있어서는「설정」은 특정 종류의 설정에 지나지 않는다. 다른 문서 (MS Windows 용의 문서등)에서는 bus-resources을「resources 」라고 읽고 있는 것도 있지만, 이 문서에서는 다른곳에도 많이 있는 resources 자원과 구별하기 위해서, "bus-resources" 라는 용어를 사용하는 것이다.

2.2 컴퓨터에 의한 디바이스의 찾는 법(또는 그 반대로)

컴퓨터는 계산을 하는 CPU 와 데이터를 저장하는 메모리로 구성되고 있다. 이것에 더해, 각종 디스크 드라이버와 비디오 카드, 키보드, 네트웍 카드, 모뎀 카드, 사운드 카드, 시리얼 포트, 패러렐 포토등이 있다. 또,

전력을 공급하는 전원 유니트, 디바이스를 CPU 에 접속하는 마더보드상의 버스, 그리고 이것들 모두를 장착하는 케이스가 있다.

옛날 거의 모든 디바이스에는 전용 plug-in card(프린트된 회로기판) 가 있었다. 최근 많은「디바이스」는, plug-in card로서뿐만이 아닌 ,「마더보드」에 붙여진 작은 칩 으로도 제공되고 있다. 마더보드에 plug-in card는 1 개이상의 디바이스가 들어가 있는 것도 있다. 메모리칩을 디바이스로서 생각하는 것도 있지만 , 본 HOWTO 에 있어서의 의미에서의 plug- and-play가 아니다.

컴퓨터 시스템을 바르게 동작시키기 위해서는, 각각의 디바이스는「디바이스 드라이버」의 제어밑에 있어야만 한다. 디바이스 드라이버는 운영 시스템의 일부로(모듈로서 로드되는 것도 있다), CPU 상에서 동작하는 소프트웨어이다. 디바이스 드라이버는 /dev 디렉토리에 있는「특수 파일」에 관련되어져 있다 . 이 파일 은 실제의 파일이 아니다. 이 파일의 이름은 hda1(하드디스 크 a 의 최초의 파티션), ttyS0(최초의 시리얼 포토), eth1 (2번째 의 이더넷 카드) 같게 되어 있다 . 복잡한 이야기로 되지만 , 선택되는 디바이스 드라이버(예를 들면 eth1라고 합시다)는 사용하고 있는 이더넷 카드의 종류에 의해 결정된다. 즉, 모든 이더넷 드라이 버가 eth1 를 할당되는 것은 아니다. 단 이더넷 의 기종에 대응한다, 어떤 특정 드라이버가 할당되지 않는다. 디바이스를 제어하기 위해서, (디바이스 드라이버를 제어하고 있다)CPU 는 각 종 디바이스에 커멘드 (및 데이터)를 보내거나, 정보를 읽기 내거나 한다. 이러한 통신을 하기 때문에, 각 바이스 드라이버는 제어하는 디바이스 유 일의 어드레스값을 알고 있지 않으면 안 된. 이러한 어드레스값을 알수있는 것은 (통신 채널을 설정한다)인 것이다. 이「채널」이 실제로는 PC 내부의 데이터 버스이고, 다른 모든 것에 공유되어 있는 경우라도 같다. 실제의 통신 채널은 여기서의 설명보다 조금 복잡하다.「주소값」은 실제로는 폭을 가진 어드레스 이고, 채널에 는(plug-and-play로 알려져 있다.)역방향 분배가 있고, 디바이스는 이것을사용해서 긴급의「헬프」요구를 디바이스 드라이버에 보낼 수 있다.

2.3 I/O 어드레스등

PC 에는 3 개의 어드레스 공간(I/O, 메인 메모리, (PCI 버스 한정)설정) 이 있다. 이들 3 종류의 어드레스는, PC 내부에서는 같은 버스를 공유하고 있다 . 그러나, 어떤 어드레스가 어느 공간(I/O, 메인 메모리, 설정)의 것인지 는, PC 의 버스상에 있는 전용의 배선의 전압을 사용해 전해진다. 자세한것은 "어드레스" 절 참조해 주세요. 최초의 디바이스는 I/O 어드레스 공간 에 배치되고 있었지만, 현재는 메인 메모리내의 공간을 사용하는 것도 있다 . I/O 어드레스는 단순히 "I/O", "IO", "i/o", "io" 라고 불리우는 것도 있다 . 또「I/O 포트」라고 말하는 것도 사용된다. I/O 어드레스 (또는 인터럽트 등의 다른 패스 자원)은 이하의 2 개의 주요한 스텝에 의해 나누어진다.

1. I/O 어드레스등을 카드(레지스터의 1 개)에 설정한다.
2. 이 I/O 어드레스등을 디바이스 드라이버에 알린다.

위의 2개 스텝 처리는, 길에서 누군가의 집의 번지를 찾아내는 문제를 2 개 로 나눈 것과 닮아 있다. 당신은 번지를 찾아내지 않으면(그리고 write down)않되고, 이 집의 소유자는, 찾아낼 수 있도록 집 앞 에 번지를 표시하지 않으면 안된다. 컴퓨터의 경우에, 디바이스 드라이버는 어드레스를 취득하지 않으면 안 되고, 디바이스 하드웨어는 같은 어드레스를 특정 레지스터에 설정하지 않으면 안된다. 이 작업은 양쪽 모두 하지 않으면 안 되기 때문에, 한쪽만 설정하는 실수를 사용자가 할 수 있다. 이러한 경우에, 컴퓨터는 디바이스를 검출 할 수 없어 곤란해 진다. 예를 들면, 시리얼 포토에 어드레스를 할당할 목적으로 "setserial" 를 사용해도, "setserial" 는 드라이버에만 어드레스를 알려주지 않는다. setserial 는 시리얼 포트 자체의 어드레스는 설정 하지 않는다. 실제로 시리얼 포트가 어드레스와 차이가 나면(혹은 완전히 설 정되어 있지 않으면), 잘못된 어드레스를 드라이버에 설정하고 있는 것에 , 문제가 일어난다.

이 외에도 분명이 필요한 것으로서, 어떤 어드레스를 디바이스 드라이버가 사용하기 전에는, 그 어드레스가 카드에 설정되고 있지 않으면 안된다. 디바이스 드라이버는 컴퓨터가 시작된 직후에 동 작을 시작하는 것이 많기 때문에, PnP 설정 프로그램이 카드 어드레스 설정을 행하기 전에 디바이스 드라이버가(카드가 있을지 어떨지를 조사하는 등을 위해서)카드 에 억세스하려고 하는 것도 때로는 있다.이러한 경우에는, 비록 카드가 있어도(그러나, 아직 어드레스를 가지고 있지 않은), 카드가 발견되지 않는다라는 에러-메세지가 표시된다.

이전의 2 개의 단락의 I/O 어드레스에 관한 설명은, 다른 자원("IRQ –개요", "DMA 채널", "메모리 영역")에 대해서도 같게 의의할 수 있다. 이것들에 대해서는 이하의 3 번째 장에서 설명 합니다 .

2.4 IRQ –개요

이 설명을 읽은 뒤에 좀 더 자세한 것을 알고 싶은 경우에는 "인터럽트 –상세"를 읽어 주세요. 여기서는 설명을 매우 간단하게 한다: 어드레 스 외에도, 취급하지 않으면 안 되는 인터럽트 번호(IRQ 5 등)라는 것도 있다. 이것은 IRQ(Interrupt ReQuest, 인터럽트 요구)번호라고 불리운다. 디바이스 드라이버가 통신을 하기 위해서는 카드의 어드레스를 알고 있을 필요가 있는 것은 이미 기술했다. 그러나, 역방향의 통신에 대해서는 어떨까? 또, 디바이스가 디바이스 드라이버에 즉시 전하지 않으는 안 되는 것이 있다면? 예를 들면 디바이스는, 메인 메모리에 보내지 않으면 안 되는 대량의 바이트열을 받았을지도 모른다. 이러한 경우에, 디바이스는 즉시 이 바이트를 가져오기 위하영 그것의 드라이버를 호출하고, 주기억 장치에 디바이스에서 가까운 전체 버퍼로부터 그들을 옮기는 것이 필요하다.

디바이스는 어떻게 해서 도움을 요청하여야만 하는가? 메인 데이타버스는 이미 사용되고 있는 것이므로, 이것을 사용하는 것은 할 수 없다. 그 대신에, 디바이스는 전용이 인터럽트 선(버스의 일부이다)에 전압을 설정한다. 이 선은 많은 경우, 그 디바이스를 위해서만 예약 되어 진다. 이 신호는 인터럽트라고 불리운다. 인터럽트 선에는 등가인 선이 16 개있어, 이것 들은 각자 (간접적으로)특정 디바이스 드라이버에 연결되어 있다. 각자 의 선에는 고유의 IRQ (Interrupt ReQuest)번호가 붙어 있다 . 디바이스는 인터럽트를 정확한 선으로 보내지 않으면 안 되고, 디바이스 드라이버는 정확한 선에서 인터럽트를 기다리지 않으면 안 된다. 어느 선에서 인터럽트가 일어나는지는 디바이스에 저장되고 있는 IRQ 번호에 의해 결정된다. 감시하는 IRQ 가 디바이스 드라이브에게 알려주는 것과같이, 디바이스 드라이버에는 이것과 같은 IRQ 번호를 설정하지 않으면 안된다.

디바이스 드라이버가 인터럽트(도움 요구)를 받으면, 디바이스 드라이버는 인터럽트가 발생된 이유를 조사하고, 인터럽트를 처리하는 적절한 동작을 하지 않으면 않된다. ISA 버스의 경우에는, 각각 디바이스에 고유의 IRQ 번 호가 필요하다. PCI 버스와, (ISA 에서도)특별한 경우에는 IRQ 를 공유하는 것도 할 수 있다.

2.5 DMA 채널

DMA 채널을 사용하는 것은 ISA 버스뿐이다. DMA 는「Direct Memory Access(직접 메모리 억세스)」라는 의미이다. 이것은 디바이스가 CPU 로부터 컴퓨터 메인 버스를 계승하고, 바이트열을 메인 메모리에 직접 전송을 하는 것이 허가되고 있는 장소이다. 보통의 CPU 는 이러한 전송을 2 스 텝으로 처리로 한다:

1. 디바이스의 I/O 메모리 공간으로부터 읽은 바이트열을 CPU 그것에송신한다.
2. 이 바이트열을 CPU 로부터 메인 메모리에 보낸다.

DMA 를 사용하면 이 처리는, 디바이스로부터 메모리로 직접 바이트열을 보내는 1 개의 스텝이 된다. 그러나, 디바이스의 하드웨어에 이 기 능이 넣어져 있지 않으면 안 되기 때문에, 반드시 모든 디바이스에서 DMA 를 사용할 수 있는 것은 아니다. 또 메인 버스가 DMA 전송에 사요되고 있기 때문에, DMA가 동작하고 있을 때에는 CPU의 동작이 제한된다.

실제로 PCI 버스에는 DMA가 없지만, 그 대신에 DMA 보다도 좀 더 좋은 기능이 있다. 그것은 bus mastering이다. bus mastering의 동작은 DMA 와 닮아서, DMA 라고 불리우는 것도 있다 (예를 들면, "UltraDMA" 로 불리우는 하드 디스크 드라이브). 이 기능을 사용하면, 디바이스는 일시적으로 버스 의 소유자(bus master)가 되고, bus master가 CPU에 있는것 같이 바이트열 을 전송할 수 있다. bus mastering은 채널 번호를 전부 사용하지는 않는다. 왜냐하면 PCI 버스의 구조에서, PCI의 하드웨어는 현재의 bus master와 bus master가 되려하는 요구를 하고 있는 디바이스를 알수 있기 때문이다. 따라서, PCI 버스에 대한 DMA 채널 할당 은 없다.

ISA 버스상의 디바이스가 DMA 를 하려고 할 때, 이 디바이스는 인터럽트 요구처럼 많은 DMA 요구를 전용 요구선을 사용해 발생한다. 실제로 DMA 는 인터럽트를 이용해 처리하는 것도 할 수 있지만, 그러나 지

연되기 때문에, DMA 요구라고 불리우는 특별한 타입의 인터럽트를 사용하는 편이 빠르다. 인터럽트와 같이, DMA 요구에는 번호가 붙여져 있고, 요구를 한 디바이스를 식별할 수 있다. 이 번호는 DMA 채널이라고 불리운다. DMA 채널은 메인 버스를 모두 사용(또 동시에 1 개밖에 동작할 수 없다.)하기 때문에, 실제로는 이것들모두가 같은 채널을 사용하는 의미이지만, 「DMA 채널」번호를 사용하면 「채널」을 사용하고 있는 디바이스를 식별 할 수 있다. 각「채널」의 현재 상태를 저장하고 있는 하드웨어 레지스터가 마더보드상에 있다. 이와 같이, DMA 요구를 하기 위해서는, 디바이스는 물리 디바이스의 레지스터에 저장되고 있는 DMA 채널 번호를 알고 있지 않으면 안 된다.

2.6 메모리 영역

일부의 디바이스에는 메인 메모리내의 어드레스 공간이 할당해져 있다. 이것은 많은 경우「공유 메모리(shared memory)」또는 「메모리맵 I/O(memory mapped I/O)」이다. 디바이스상의 ROM에 있는 것도 있다. bus-resources를 론을 할 때에는, 이것은 단순히 「메모리」라고 불리운다. 이 같은 디바이스도 I/O 어드레스 공간을 사용 한다

이러한 카드를 꽂았을 때는, 실제로는 (I/O 메모리가 아닌)메인 메모리에 대한 메모리 모듈을 꽂는 것이 된다. 이것은 ROM(Read Only Memory)과 공유 메모리의 어느 쪽에서도 상관하지 않는다. 이 메모리는 디바이스와 메인 메모리간의 직접 데이터「전송」의 수단으로서 사용된다. 그러나, 이것은 실제로는 전송이 아니다. 이것은, 디바이스는 자신 자신의 메모리에 데이 타를 쓰고, 그것이 메인 메모리에도 쓰여지게 하고 있는것뿐이기 때문이다. 카드와 디바이스 드라이버의 어느것도, 이 영역이 어디에 있을지를 의식할 필요는 없다. 메모리 어드레스는 아마 매우 높은 위치에 놓여져 , 낮은 위치에 있는 컴퓨터의 메모리칩 어드레스와 겹쳐질 일은 없다.

ROM의 취급은 다르다. 이것은 프로그램(아마 디바이스 드라이버)으로 있는것이 많아, 디바이스와 함께 사용된다. 아마 이것은 Windows 상뿐만 아니라 Linux 상에서도 동작한다(??). 이것은 shadowed화할 필요가 있을지도 모른다. shadowed화라는 것은, 고속 동작을 하게 하기 위해서 ROM의 내용을 메인 메모리에 복사하는 것이다. 일단 shadowed화를 하면, 이 영역은 이제 「읽기전용」에서는 없어진다.

2.7 디바이스와 드라이버의 양쪽에 있어서의 "resources"

이와 같이, 디바이스 드라이버는 자신이 제어하는 하드웨어에 대해 어떠한 방법으로 「할당」되지 않으면 안된다. 이것은 bus resourse (I/O, 메모리, IRQ, DMA)를 물리적인 디바이스와 디바이스 드라이버의 소프트웨어 양쪽 에 고급하는 것에 의해 행해진다. 예를 들면, 시리얼 포토는(4 개 중)2 개 의 자원, 즉 IRQ 와 I/O 어드레스만을 사용한다. 이 값은 양쪽 모두의 디바이스 드라이버와 물리적 디바이스에 공급하지 않으면 안 된다. 드라이버(와 그 디바이스)는 /dev 디렉토리내에도 이름을 공급한다(ttyS1 등). 어드레스와 IRQ 번호는 물리적 디바이스 카드 레지스터 내(또는 마더보 -드상의 칩내)에 저장된다. 점퍼의 경우에, 이 정보는 반드시 디바이스 의 하드웨어(카드등)에 저장된다. 그러나 PnP 의 경우에는 보통 PC 의 전원을 끄면 레지스터의 데이터는 없어지게 되므로, resource data는 PC 의 전원을 넣을 때마다 매회, 각디바이스에 대해 새롭게 공급하여야 한다.

2.8 문제

PC 의 구조에서는 IRQ, DMA 채널, I/O 어드레스, 메모리 영역 의 수에 제한이 있다. 만일 디바이스의 수가 적고, 이들 모두가 표준화되어있는 bus resourse(고유의 I/O 어드레스와 IRQ 번호)를 가지고 있다면, 디바이스 드라이버를 디바이스에 할당할 때, 문제는 일어나지 않을 것이다. PC 상의 각디바이스는 다른 디바이스와 충돌하지 않는 고정 resourse를 가질 수 있기 때문이다. 다른 디바이스가 같은 I/O 어드레스와 IRQ 를 갖는것도 없을 것이다. 그렇기 때문에, 각 디바이스 드라이버의 프로그램 중에 I/O 어드레스 와 IRQ 를 hard-coded하면 용이하므로. 매우 이야기가 간단하게 된다.

그러나, 현실은 그렇지 않다. 최근에는 다른 디바이스가 많이 있으므로 충돌은 비번하지는 않지만, 같은 타입의 디바이스를 복수로 사용해야만 하는 경우도 있다. 예를 들면, 복수가 다른 디스크 드라이버와 복수의

시리얼 포트를 사용하고 싶은 경우등이다. 이러한 이유 때문에, 디바이스에는 충돌을 회피할 수 있는, IRQ 와 어드레스를 설정할 수 있는 자유도가 필요하게 된다. 그러나, 클럭과 키보드 같은 일부의 IRQ 와 어드레스는 표준 치를 사용한다. 이러한 디바이스에는 이같은 자유도는 필요 없다.

bus resourse 할당에서 충돌 문제는 그외에, 디바이스 드라이버의 버스리소 스설정을 잘못해서 생기는 문제가 있다. 예를 들면, 사실은 디바이스에 IRQ 5 가 설정되고 있는데 설정 파일에는 IRQ 4 를 쓰고 있는 것 같은 경우이다. 이것은 다른 형식의 버스 자원 배정 에러이다.

버스자원 배정을 바르게 하면, 물리적 디바이스와 여기에 대응하는 디바이스 드라이버사이에 통신 채널이 확립된다. 예를 들면, 어떤 범위의 I/O 어드레스(자원)가 디바이스 드라이버와 하드웨어에 할당되어진 경우, 양자간에 일방통행의 통신 채널이 확립된 것으로 된다. 드라이 버는 명령어와 정보를 디바이스에 보내는것이 가능하다. 드라이버는 레지스터를 읽는 것에 의해 디바이스로부터 정보를 얻을 수 있으므로, 실제로 단순히 일방통행이라는 것이 아니다. 그러나, 이 방법으로는 디바이스 측에서 통신을 시작하는 것을 할 수 없다. 쌍방향 통신 채널을 만들기 위해서, 디바이스 IRQ 를 필요로 한다. 쌍방향 통신 채널이란, 디바이스 드라이버 에서도 디바이스에서도 통신을 시작할 수 있는 것이다.

2.9 시리얼 포토에 꽂은 디바이스를 PnP에 의해 검출

시리얼 포트에 cable로 접속한 외부 디바이스(외부부착 모뎀등)도 Plug-and-Play 라고 불리운다. 버스자원(IRQ 와 I/O 포토)를 필요로 하는 것은 시리얼 포토 자신뿐이므로, 이러한 접속 디바이스 자체에 버스 자원 을 배당하지 않는다. 그렇기 때문에, 실제로는 이들 디바이스에 PnP는 필요 하지 않다. 설령 그렇더라도, 이러한 외부 시리얼 디바이스에 대해서 PnP의 사양이 정해져 있다 .

PnP OS는 이러한 외부 디바이스를 검출하고, 그 디바이스의 모델 번호등을 읽는다. 이것에 의해, 그 디바이스용 디바이스 드라이버를 찾아내는 것이 가능해게 되므로, 특정 디바이스(예를 들면 /dev/ttyS1)를 사용하고 있는 것을 응용 프로그램에 대해 지시할 필요가 없게 된다. 디바이스가 접속되고 있는 시리얼 포트를(설정 파일등을 사용해)수동으로 지정하는 것도 가능하므로, (디바이스의 모델 번호를 지정하는 것이 있을지도 모른다) 어쨌든, PnP 의 「시리얼 포트」기능이 필요하다는 것은 아니다.

제 3 절 플러그&플레이(PnP)의 해결 방법

3.1 PnP의 도입

플러그&플레이라는 단어에는 여러가지 의미가 있다. 넓은 의미에서는, 디바이 스를 연결하면 그 설정이 행해지는 단순한 자동 설정을 가리킨다. 본 HOWTO 에서의 의미로는, 설정이라는 것은 PnP 버스자원 설정과, 디바이스 드라이버에 그 설정 내용을 알리는 것만을 가리킨다. 좀 더 좁은 의미에서는, 하드웨어 디바이스의 버스 자원을 설정하는 것만을 가리킨다. 이것은 PnP 의 사양을 가리키는 것도 있다. 이 사양은(다른 것도 쓰여지고 있지만 특히) ISA 버스상의 디바이스(많은 경우 카드이다)에 대해 PnP 자원 데이 타를 읽고 쓰기 하는 방법의 사양을 정한다 . 표준 PCI(PnP 에는 없다)의 사양은, 같은 것을 PCI 버스로 할 수 있도록 만들어져 있다 .

PnP 는 디바이스와 디바이스 드라이버의 설정을 일치시키고, 양쪽의 통신 채널 을 지정 한다. 플러그&플레이가 사용되기 전의 ISA 버스의 경우에는, 점퍼 를 사용해 하드웨어 디바이스의 버스 자원을 설정하고 있었다. 소프트 웨어 드라이버로의 버스 자원 배당은, 설정 파일(등)또는 디바이스 가 존재한다고 생각되는 어드레스를 조사하는 것에 의해 행해졌다. PCI 버스는 만들어졌을 때로부터 PnP와 유사 했으므로, PCI 버스용으로 PnP 를 실현하는 것은 매우 간단했다. PCI버스의 사양에서는 PnP 라는 용어 가 사용되고 있지 않기 때문에, PCI 버스를 PnP 라고 불러도 좋을가는 분명히 하고 있지 않다(그렇지만, PCI 버스는, 현재 PnP 라고 불리우고 있는 것을 하드웨어 적으로 지원하고 있다).

3.2 PnP 의 동작 (간단한 설명)

PnP 동작의 개요를 매우 간단하게 설명 합다. PnP 설정 프로그램(아마 BIOS 안에 있는 프로그램이다)은 모든 PnP 디바이스를 검출하고, 각 디바이스가 필요 로 하는 버스 자원을 요구한다. 다음으로, 이 설정 프로그램은 알려주어야만하는 버스 자원(IRQ 등)을 체크 한다. 당연하지만 non-PnP (legacy) 디바이스가 사용하고 있는 예약제 버스 자원이 있다면(설정 프로그램 이 알고 있으면), 이러한 예약제 자원은 PnP 디바이스에 알려주지 않는다. 다음으로, 설정 프로그램은(PnP 사양에서는 정해지고 있지 않은)어떠한 기준 을 이용해, 충돌이 일어나지 않게, 또한 모든 디바이스에 필요한 버스 자원을(가능 하다면)할당한다. 그래서, 설정 프로그램은 할당된 버스 자원 을 각 물리 디바이스에 설정하고, 디바이스는 할당되어진 버스 자원을 사용하는 것과 같이 자신 자신의 설정을 한다. 그리고 디바이스 드라이버는 제어하는 디바이스 가 사용하는 버스 자원을 어떠한 방법으로 검출하고, 이것에 의해 디바이스를 효율적으로 통신이 할 수 있게 된다.

예를 들면, 인터럽트(IRQ 번호) 1 개와 공유 메모리 1MB를 필요로 하는 카드가 있다고 하자. PnP 프로그램은 이 요구를 카드로부터 받는다. PnP 프로그램은 IRQ5와, 어드레스 0xe9000000 로부터 시작되는 1MB 의 어드레스 공간을 할당한다. 언제나 이와 같이 간단하게 설정할 수 있는것은 아니다. 이러한, (ISA 의 경우는)특정 IRQ 번호밖에 사용할 수 없는 것과, 1MB 의 메모리 영역이 특정 범위 어드레스에 들어 있지 않으면 안 되는 것을 카드가 지정되어 오는 경우가 있기 때문이다. 상세한 부분은 PCI 버스와 ISA 버스에서 다르지만, ISA 버스 쪽이 문제가 좀더 복잡하다.

PnP 설정 프로그램에서 사용할 수 있는 단축키 설정 방법이 몇개 있다. 그 하나는 마지막 설정시(컴퓨터를 마지막으로 사용했을 때)의 버스 자원 할당을 보존하여, 이것을 재이용하는 것이다. Windows9x 와 PnP BIOS 의 조합 은 이 동작을 하지만, 표준의 Linux 에서는 이것을 하지 않는다. Windows9x는 이 정보를 하드 디스크상의「레지스터리」에 보존 하고, PnP BIOS는 이 정보를 PC의 불휘발 메모리(이것은 ESCD로 알려져 있다.) “BIOS 의 ESCD 데이타베이스” 를 참조하시오) 에 보존한다.

Linux에 있어서, 이 처리는 각각의 디바이스가 자력으로하며, 자원의 할당을 집중 관리하는 불휘발성의 레지스터리는 없다. 디바이스 드라이 버에 의해서는, 마지막에 이용한 설정을 저장하고, 다음에 전원을 켰을 때 그 설정을 사용하는 것도 있다. 이러한 디바이스는, 나머지 하드웨어가 버스자원을 필요로 하지 않는 것을 암묵적으로 가정하고 있다.

디바이스 하드웨어가 전의 설정을 기억하고 있으면, 다음 기동시에는 대부분 아무것도 설정하지 않아도 되지만, 디바이스는 전원을 끊으면 설정을 잊어 버린다. 디폴트 설정을 가지고 있는 디바이스도 있다 (그렇지만, 이것은 반드시 마지막으로 사용한 설정이라고 한정하지 않는다). 따라서, PnP 설정 프로그램 은 PC를 기동할 때마다 매회 실행할 필요가 있다. 또, 새로운 디바이스 를 추가하면, 디바이스 설정을 할 필요가 있다. 이 새로운 디바이스에 버스 자원을 할당할 때에는, 이미 존재하는 디바이스부터 일부의 버스 자원 을 제거하고, 대신에 사용할 수 있는 다른 버스 자원을 그 디바이스에 할당하는 것일 지도 모른다.

3.3 PC 의 기동

컴퓨터의 전원을 처음 켰을때, BIOS 칩은 컴퓨터를 시작 시키기 위해서 BIOS 프로그램을 실행한다(최초의 스텝은 하드웨어의 체크이다). operating system이 하드 디스크에 저장되어 있는 경우(보통은 그렇다), BIOS는 하드 디스크의 정보를 취해야만 한다. 하드 디스크가 PnP 이면, BIOS 는 이것을 찾아내기 위해 PnP 를 사용할 수 있다. 또, 컴퓨터 시작시에 사용자가 BIOS 의 CMOS 를 수동으로 설정할 수 있도록 하거나, 에러-메세지를 보낼 수 있도록 하기 위해서는, 스크린(비디오 카드)와 키보드가 필요가 된다. 이들 디바이스가 있다면 BIOS 는 PnP 설정을 해야만 한다.

한 번 BIOS가 하드 디스크, 비디오 카드, 키보드를 인식하면, BIOS 에 의한 부팅(하드 디스크로부터 메모리로 operating system를 로 드하는 것)개시 준비는 완료된다. PnP 대응 오퍼레이팅 시스템 (PnP OS)를 사용하고 있는 것을 BIOS에서 지정하고 있는 경우에는, BIOS는 전에 말한 것처럼 PC의 부트를 개시하고, operating system에 PnP 설정 을 해야만한다. 그렇지 않을 경우는, (부팅 전에)PnP BIOS 자신이 나머지 디바이스 PnP 설정을 한다.(그러나, 드라이버의 설정은 하지 않는다).

3.4 버스

ISA은 오래된 IBM-PC 버스이고, PCI는 인텔이 제안한 새롭운 고속의 버스이다. PCI버스는, 현재 PnP 라는 기능을 실현할 수 있도록 설계되고 있다. PCI 버스에서는, PnP 버스 자원이 하드웨어 디바이스의 어느것에 할당되는지의 조사가(ISA 버스와 비교해)간단하다. 어떤 설정이 되어 있는지를 알기 위해서는 lspci 커멘드를 사용하거나, /proc/pci 또는 /proc/bus/pci 파일을 보면 좋다. 시작시에 화면에 표시되는 메세지(전의 표시를 보기위해서는 Shift-PageUp을 사용한다)도 유용하다. "Boot-time Messages"를 참조하시오.

ISA 버스의 경우에는, PnP의 실현에 관한 어려운 문제가 있다. 이 이유는 ISA 버스 설계 당시에는 누구도 PnP를 고려하지 않았던 것과, 설정 정보 를 물리 디바이스에 보내기 위해서 PnP가 사용할 수 있는 I/O 어드레스가 거의 없다는 것이다. 결국, ISA 버스상에서 PnP를 하는 방법은 매우 복잡하게 되었다. 이것에 관한 책이 몇권 쓰여져있다 . "PnP Book" 를 보시오. 특히, 각각의 PnP 디바이스에 PnP 프로그램용의 일시적인「핸들 (handle)」를 할당하고, 프로그램이 PnP 설정을 할 때에 디바이스를 특정할 수 있도록 하는 것이 필요하다. 이「핸들」을 할당하는 것을「Isolation (isolation)」이라고 말한다. 상세한 것에 대해서는 부록 "Isolation"을 참조하시오.

ISA 버스는 언젠가는 없어지게 된다. 그렇게 되면, PnP는 BIOS가 어떻게 하드웨어를 구성했는지를 간단히 알수있기 때문에 좀 더 간단해 질 것이다. 그런데도, 디바이스 드라이버를 디바이스와 어울리게 구성할 필요가 있고, PC의 시작·실행시에는 추가된 디바이스를 설정할 필요도 있다. 이러한 필요성은, Linux가 PnP operating system이였다면 만족될수 있는 것이다.

3.5 Linux 에서 PnP를 더 잘해야할 필요성

PnP 규격(ISA 버스용)은 Compaq, Intel, Phoenix 가 만들었다. Microsoft 는 선두에서 PnP의 보급을 진행시켰다. PnP가「발명」되지 않았다면 , Linux는 좀 더 좋았을것이다. 언젠가 ISA 버스는 쓸모없게 되고, PnP 에 닮은 기능을 가진 PCI 버스가 보급되면, 실현이 용이한 PnP를 실제로 사용할 수 있을 것이다. 그러나, 그것을 좋아하거나 그렇지 않거나, 최근의 새로운 ISA 하드웨어는 거의 모든 것이 PnP 이므로, Linux는 PnP와 잘 맞도록하는 이외의 길은 없다. 그러나, 표준 Linux (1999 년 초기의 시점)에서 는, PnP 취급이(특히 ISA버스의 경우에는)복잡하게 되어 있다 . PnP 원래의 목적은 설정을 간단하게 하는 것이었지만 ….

어떤 의미로, Linux는 PCI 버스에 대해서는 이미 어느정도 PnP 기능을 가지고 있다고도 말할 수 있다. PC를 부팅했을 때, 스크린에 표시되는 메세지 로부터 일부의 디바이스 드라이버가 자신이 제어하는 하드웨어 디바이스(및 BIOS가 이것들에 할당한 버스 자원)을 검출하는 한 것을 독자 여러분 도 기억할지 모른다. 그러나, PnP operating system이라면 좀 더 능숙하게 처리할 수 있는 상황이 몇개쯤 있다 :

- 버스자원 부족의 경우.
- 1 개의 물리 디바이스에 복수의 드라이버가 있는 경우.
- 활동하게 된 디바이스가 물리 디바이스를 찾아내지 못한 경우.
- 디바이스 설치(docking 등)의 경우.

Linux 사용자는, 사용하고 싶은 ISA PnP 디바이스의 설정을 하기 위해서 PnP, 그것에 대해서 자세하게 조사할 필요는 없을 것이다. 해결 방법의 하나는, 표준화된 버젼의 Linux가 ISA 버스와 PCI 버스, 그 외의 버스에 있어서 프 래그&플레이를 지원하는 것이다. 커넬에 패치가 쓰여져 있지만, 대부분의 드라이버는 이 패치에 대응하고 있지 않다. 이 패치는 표준 Linux의 부분이 아니다. "Patch Kernel".를 참조 하시오.

제 4 절 PnP BIOS 의 설정

컴퓨터의 전원을 켜면, operating system가 로드되기 전 에 BIOS 가 실행된다. 최근 BIOS는 PnP 대응이고, 일부 혹은 전부 의 PnP 디바이스를 설정 한다. 대부분의 PnP BIOS 에서는 PnP 를 무효로 할 수 없기 때문

에, PnP와 잘 공존해 갈 수 밖에 없다. BIOS 의 CMOS 메 뉴에 있을지도 모르는 선택권을 아래에 몇개 나타낸다:

- "Do you have a PnP operating system?(PnP operating system 를 가지고 있는가?"
- "How are bus-resources to be controlled?(어떻게 버스 자원을 제어 하는가?)"
- "Reset the configuration?(설정을 리셋트할 것인가?")

4.1 PnP operating system을 갖고 있는가?

이것에「yes」를 설정하고 있는 경우, PnP BIOS 는 하드 디스크·비디오카 드·키보드의 PnP 설정을 하고, 시스템을 시작할 수 있도록 한다 . 그러나, PnP BIOS는 설정 작업 마무리를 operating system에 맡긴다. BIOS 는 ISA 버스상에서 "Isolation" 을 하고, 디바이스는 무효 로되면 operating system에서 설정할 수 있는 상태이다. Linux의 경우에는 필히, PnP operating system를 가지고 있지 않은 BIOS 에 설정해야 한다. 이와 같이 답하지 않으면, BIOS는 설정하지 않은 ISA 디바이스를 무효 상태로 할지도 모른다(??). PCI 디바이스 도 설정되지 않을지도 모른다(??).

PnP OS 를 가지지 않도록 BIOS 에 설정한 경우, BIOS는 자신 자신으로 디바이스 설정을 한다. 새로운 PnP 디바이스를 추가하지 않는 한, BIOS는 불휘발성 메모리(ESCD)에 보존되고 있는 설정을 사용한다. "BIOS 의 ESCD 데이터 베이스" 를 참조하시오. 컴퓨터의 마지막 세션에서 Linux 를 사용했다면 설정은 바뀌지 않을 것입니다. "BIOS에서의 PnP 설정" 을 참조 하시오. 그러나, 마지막 세션에서 (PnP OS이다) Windows9x를 사용 한 경우, Windows가 ESCD 를 변경였을지도 모른다. 이것이 실행되는 것 은 아마,「설정」을 강제로 시키거나, legacy 디바이스를 인스톨했을 때 뿐이다. "Windows를 이용한 ESCD의 설정이 문제를 일으키는 경우" 를 참조하시오. 독자의 여러분이 isapnp 와 PCI Utilities 등의 프로그램을 사용해서 설정 한 경우, 이들의 프로그램은 BIOS 실행후에 실행되고, 사용자가 지시한 대로 PnP의 설정을 변경 한다.

4.1.1 Interoperability with Windows9x

Linux 와 Windows를 같은 PC 상에서 사용하고 있는 경우에는, BIOS의「PnP OS 를 사용하고 있는가?(Do you have a PnP OS?)」라는 질문에 어떻게 답하면 좋을까? 통상(그리고 정확하게)은, 표준의 Linux에 대해서는「no」라고 답하고 , Windows9x 에 대해서는「yes」라고 답해야 한다. 그러나, OS 를 바꾸려고 할 때에, BIOS CMOS 메뉴를 수동으로 설정해야 하는 것은 매우 불편하다. 이것을 해결하는 방법의 하나는, Windows 사용시 도 포함해서「PnP OS 를 가지고 있지 않다」라고 CMOS 에 설정하는 것이다. Windows 는 이 상황(BIOS가 주어진 하드웨어를 완전하게 설정하고 있다) 에 대응 할 수 있는 것을 기대할 수 있다. 한편, 하드웨어가 이미 설정되어 있는 것 을 Windows가 인식할 수 없어도, Windows를 다시 한번 설정 하여 훌륭히 동작하는 것이 기대할 수 있다. 그러나, 그다지 좋지 않은것 같다. Windows는 단순히, 레지스트리에 정장하고 있는 정보를 드라이버에 전하는 것뿐이다. 그러나 (BIOS가 실행된)실제 하드웨어 설정은, ESCD에 저장되어 있는 설정에 있고, 레지스터리와 다를수도 있어 , 문제를 일으킬지도 모른다.

CMOS의 설정과 레지스트리의 설정을 맞추는 방법의 하나는, BIOS 설정을「PnP OS 를 가지고 있지 않은(not a PnP OS)」인 상태에서 Windows 를 설치 (또는 재인스톨)하는 것이다. 이렇게 하면, Windows에는 BIOS가 설정한 시스템이 보여지게 된다. 이 설정에 있어서 자원 의 충돌이 없다면, Windows는 아마 설정을 유지하고 , 이것을 레지스터리에보 존 할 것이다. 이렇게 해서 ESCD와 레지스트리가 동기된다. 이것으로서 바르게 동 작하면(그리고 이 HOWTO 가 최신판이면), 필자에게 알려 주시오. 필자는 바르게 동작했다고 말한 보고를 1 건밖에 받지 못했지 때문이다.

다른 방법은, Windows 에서 문제를 일으키는 디바이스를 디바이스 매니저상에서「삭 제」하는 것이다. 그리고「PnP OS 를 가지고 있지 않은(Not a PnP OS)」의 상태에서 PC를 재시작 한다(설정은 시작시에 CMOS에서 한다). Windows 상에서 디바이스 드라이버를 재인스톨되어, 잘 하면 BIOS가 설정 한 버스 자원을 이 때에 사용 할 수 있다. Windows는 아마 Windows의 설치 용 CD 를 요구하는 것에 주의 하시오. Windows는 드라이버 파일(의 종류)이 남아 있어도, 이것을 찾아내지 않는 경우가 있기 때문이다. 테스트로, 필자는 Novell 호환의 드라이버를 갖고있는 NIC 카드를「삭제」했다. 재시작 시, Windows는 Novell이

아닌 Microsoft의 네트워크 드라이버를 사용해 재인스톨을 했다. 즉, Novell 클라이언트를 재인스톨할 필요가 있는 것이다. 이 방법으로 문제가 발생했다면 필자에게 알려주시오 (이 HOWTO 가 최신판의 경우에 한정한다).

4.2 어떻게 버스 자원을 제어할까?

이 항목은 IRQ 버스자원과 DMA 버스자원 할당 방법만을 결정한다. 이 항목에 「auto(자동)」을 설정하면, BIOS가 할당을 실행한다. 메뉴얼(수동)」을 설정하면, 사용자의 입력에 의해 「legacy」(non-PnP) 카드를 위한 IRQ를 몇개 정도 예약할 수 있다. 지정을 하지 않으면, 카드가 Iegacy카드에 있는지는 BIOS가 인식할 수 있는 것도 인식할 수 없는 것도 있다. BIOS가 Iegacy카드의 정 보를 알 수 있는 것은, 사용자가 Windows 상에서 ICU(또는 비슷한것)를 실행하고 BIOS에 그 정보를 주고 있는 경우뿐이다. BIOS가 이것을 알고 있으면 "auto" 를 시험해 주시오. 이해되지 않으면, Iegacy ISA 카 드용 IRQ는 수동으로 예약하고, 나머지의 IRQ를 BIOS의 PnP로 할당하도록 한다.

4.3 설정을 리셋트할지?

이 항목은 PnP 디바이스의 설정에 관한 BIOS ESCD 데이타베이스를 소거한다. 또한, Iegacy(non-ISA)의 디바이스 설정에 관한 리스트도 소거한다. 데이타베이스가 잘못되어져, 작성, 수정 할 필요가 있다고 확신하고 있는 경우라면, 소거하지 말아야한다. 소거를 하는 것은, 사용자가 컴퓨터를 시작 할 수 없는 경우에 한한다, 라고 어딘가에 쓰여져 있다고 생각한다. BIOS Iegacy 디바이스의 데이터를 잃은 경우에는, 사용자가 Windows에서 ICA 를 실행하고 이 데이터를 작성, 수정 할 필요가 있을 것이다.

제 5 절 PnP 카드의 취급 방법

5.1 PnP 카드 취급의 소개

현재는 새로운 내장 보드(카드)의 거의 모든 것이 플러그&플레이 (PnP)이다. Linux 에도 PnP를 취급하는 소프트웨어는 있지만, 이것은 반드시 사용이 용이한것은 아니다. PnP와 조화롭게 공존하기 위한 방법을 아래에 6개 보였다. (상황에 의해서는 사용할 수 없는 것도 있다). 이 중의 어떤 것을 사용하는 것이 마땅한지는 목적에 의해 다르다. 결국, 지금은 없을지도 모르지만 가장 간단하고 편리한것이 좋다. 간단하다고 생각되는 방법은, 스스로는 아무것도 하지 않고, PnP-BIOS 에 설정을 하게 하는 것이지만, 추후에 당신은 BIOS가 행한 설정을 찾아 보는것이 필요할 수 있다. 이들 방법을 비교하는 것은, 누구가가 전부를 시험해서 그것을 글로 옮길 필요가 있다. 작업을 할때에 복수의 방법을 사용할 필요가 있을지도 모른다.

- "Disable PnP". 이것에는 점퍼 또는 DOS/Windows 용 소프트웨어를 사용한다(이것을 할 수 없는 카드도 많다).
- "BIOS 에 PnP를 설정한다"(PCI 버스의 경우에 필요한 것은 PCI BIOS 뿐이다. 그 이외의 경우에는 PnP BIOS가 필요가 된다).
- "isapnp" 는 오직 ISA 버스 상에서만, PnP 디바이스를 설정 할 수있는 프로그램이다.
- "PCI Utilities" 는 PCI 버스를 설정하기 위한 유틸리티이다.
- "Windows 를 사용한 설정"을 하고, Windows/DOS 내부에서 Linux 를 시작한다. 마지막 수단으로서 사용 하시오.
- "Patch Kernel", Linux를 PnP operating system로 변환한다.
- "디바이스 드라이버를 설정한다".그러나, 이것을 하는 일은 거의 없다.

위의 방법중 어느것을 사용해도, 하드웨어 내의 버스 자원이 설정될 것이다. 그러나, 실행된 설정을 디바이스 드라이버에 전한 것은 마지막 2 개 뿐이다. 확실하게 전하는 것은 마지막 1개 뿐이다.(그것 자신이 드라이버 이기 때문이다.) 드라이버에 정보가 어떻게 해서 전해지는 지는 드라이 버에 의존하고, 정보를 전하기 위해서는 사용자가 무엇인가를 하지 않으면 안 되는 것 도 있다. 자세한 것은, "Tell the Driver the Configuration" 의 장을 보시오.

5.2 Disable PnP ?

많은 디바이스는 PnP 전용이고, PnP를 무력화 할 수 있는 것은 없다. 그러나, 일부의 디바이스에서는, 점퍼와(점퍼가 없는 구성의 경우에는)디바이스 부속의 Windows 용프로그램을 사용해 이것을 무력화 할 수 있다. 이것에 의해, PnP 설정의 불필요한 작업을 회피할 수 있는 것도 많이 있다. 이러한 버스 자원이 예약되어 있는 것을, 잊지 말고 BIOS 에 설정하시오.

PnP 를 무력화시키는 편이 좋은 이유는 이 외에도 몇개 있다 :

1. 같은 머신에 MS Windows가 있는 경우, PnP를 사용할 수 있도록 해서 Windows 의 설정을 Linux와는 다른 설정로 하는 것이 있을지도 모른다.
2. PnP 를 사용하지 않는다면, IRQ 번호(와 포트 주소)등의 선택 범위가 크게 제 한되어 버릴 수 있다.
3. 제어하는 디바이스를 찾기 위해서 PnP 방법을 이용하는 Linux 디바이스 드라이버도 있다.
4. 앞으로 기기 구성을 바꾸는 것이 필요하게 된 경우, 디바이스가 PnP라면 변경이 용이할 것이다.(점퍼를 설정하거나, DOS/Windows 용 프로그램을 실행할 필요가 없다).
5. 다른 PnP 디바이스가 있을 수도 있지만(또는 있을 수 있다), PnP를 사용할 수 있도록 해 둘(혹은 그 방법을 알아 둔다)필요가 있을지도 모른다.

한 번 non-PnP 디바이스로 설정하면, (고쳐 점퍼로 설정을 변경하는가, DOS/Windows의 설정 프로그램을 실행하지 않는 한)그 디바이스는 PnP 설정 프로그램과 BIOS 에서는 설정할 수 없게 된다.

5.3 BIOS에서 PnP 설정

5.3.1 BIOS를 사용해 PnP의 설정을 하는데에 있어

PnP BIOS를 사용한다면, 하드웨어 설정을 하는 것이 가능하다. 즉, BIOS가 모든 디바이스를 필요로 하는 자원을 전부 읽어보고, 이것을 설정한다(버스자원을 디바이스에 할당한다). 이것은 PnP OS의 대용이 되지만, BIOS는 디바이스를 설정하지 않는 점과, 실행된 설정을 드 라이버에 알리지 않는 점이 다르다. 보통은 불휘발성 메모리(ESCD)내에 보존되어 있는 설정을 사용해야 한다. 새로운 디바이스를 검출한 경우와, 자원이 충 돌한 경우, BIOS는 필요에 따라서 설정을 변경해야 마땅하고, ESCD내의 설정 을 그대로 사용할 수는 없다.

사용되는 BIOS는 이러한 설정을 지원하고 있지만, BIOS를 바르게 설정 하지 않는 경우와 불완전한 설정을 하는 경우가 있다. BIOS를 사용하는 이점은 단순한 것이다. 대부분의 경우 설정하는 것이 없기 때문이다. (BIOS의 CMOS 메뉴에서「PnP OS가 아니다」라고 설정한 점을 제거한다.). 디바이스 드라이버에 의해서는 BIOS가 행한 설정을 자동적으로 검출할 수 있는 것도 있지만, 어떤 경우에 있어서는 BIOS가 행한 설정을 사용자가 조사할 필요 가 있다. (항상 쉽지만은 않다). 상세한 것은 "현재의 설정은 어떻게 되어 있는가?" 를 참조 하시오. 달리 생각되는 이점으로서는, BIOS 는 Linux가 시작하기 전에 동작하므로, 뒤에 시작하는 디바이스 드라이버가 모든 버스 자원을 사용할 수 있는(그리고 검출할 수 있다)점이 거론된다. MS에 의하면, PnP BIOS가 디바이스의 PnP 설정을(MS Windows의 지원 없은에) 할 수 있는 기능은 옵션에 지나지 않는다.(필수가 아니다). 그러나, 1996년(??)무렵 보다 후에 만들어진 BIOS의 대부분은 이 기능을 가지고 있다 . 이것이 바르게 동작한다면 , 우리들은 그들에게 (*보답)답례의 메모를 보내야 할 것 이다. 이러한 BIOS는 PCI 버스도 ISA 버스도 설정하지만, 일부 오래된 BIOS는 PCI 밖에 설정할 수 없다고 한다. 사용

하는 BIOS에 대해서 좀더 조사하고 싶으면 WWW 를 봐 주시오. 필자도 이것에 관한 데이터를 가지고 있는 것이 아니기 때문에, 필자에게 문의는 피하기 바랍니다. BIOS 에 대해서 독자 여러분이 알고 싶다고 생각하고 있는 것의 자세한 정보는 입수가 곤란(혹은 입수할 수 없는)할지도 모른다. 일부의 BIOS는 최소한의 PnP 기능밖에 가지고 있지 않고, 설정 작업의 어려운 부분은 Windows 유티리티 에 맡기려고 한다. 이런 경우에는, 다른 방 법(isapnptools 등)을 찾거나, 혹은 BIOS에 ESCD 데이타베이스가 있다면 이것을 이용해야만 할 것이다. 자세한 것은 다음 장을 참조 하시오.

5.3.2 BIOS의 ESCD 데이타베이스

BIOS 는 PnP 설정 정보를 기록해 있는 비휘발성 데이타베이스를 관리하고 있다. 이것은 ESCD(Extended System Configuration Data)라고 한다. 또한, ESCD에 관한 규정은 옵션이지만, 대부분의 PnP BIOS에는 이 기능이 있다. ESCD는 PnP 디바이스의 자원 설 정정보를 저장하는 것만이 아니라, non-PnP 디바이스의 설정 정보도 저장하고 있으므로, (non-PnP 디바이스에 있는 것도 기록하고 있다), 충돌을 피할 수 있다. ESCD 데이터는 보통 칩에 보존되기 때문에 전원을 꺼도 없어지지 않지만, 하드 디스크에 데이터를 보관 유지하는 타입도 있다 (??).

ESCD는 마지막으로 사용한 설정을 보관 유지하기 위한 것이지만, Linux의 isapnp 와 PCI Utilities 같은(ESCD를 갱신하지 않는)프로그램을 사용한 경우에는, ESCD는 이러한 프로그램을 모르기 때문에, 이 설정은 ESCD 내에 보존 되지 않는다. 좋은 PnP OS는 ESCD를 갱신하므로, 후에(표준 Linux 같은)non-PnP의 OS로 설정을 사용할 수 있다. Windows는 특별한 경우에만 이것을 한다. “Windows 를 이용한 ESCD 의 설정” 을 참조 하시오.

ESCD에 설정되어 있는 정보를 이용면, 반드시 BIOS의 CMOS에서 ”Not a PnP OS” 혹은 비슷한 설정을 해 주시오. BIOS가 시작할 때에(Linux OS 가 로드되기 전), BIOS는 ESCD를 사용해 설정을 한다.그러나, BIOS가 ESCD에 기록되고 있지 않은 새로운 PnP 카드를 검출하면, BIOS는 그 카드에 버스 자원을 할당하고 ESCD를 갱신하여야만 한다. 기존 의 PnP 카드에 할당하고 있는 버스 자원을 변경하고, 그것에 동반해 ESCD 를 수정해야만 하는 것도 있다.

각각의 디바이스가 마지막 설정을 하드웨어에 보존하고 있다면, PC 를 시작할 때에 하드웨어 설정을 할 필요는 없다. 그러나 실제 동 작은 그렇지 않다. 따라서, BIOS를 PnP로 사용하고 있는 경우에는 모든 ESCD 데이터를 항상 정확한 상태로 유지해야만 한다. ESCD를 가지고 있지 않은 BIOS도 몇개 있지만, 이러한 BIOS도 비휘발 메모리를 몇개 가지고 있어, non-PnP 카드가 사용하기 위해서 예약하고 있는 버스 자원을 저장할 수 있다. 많은 BIOS는 양쪽 모두를 가지고 있다.

5.3.3 Windows 를 이용한 ESCD 의 설정

BIOS가 하는 ESCD의 설정에 사용자의 원하는 바가 없는(또는, 꼭 있어야만 하는) 경우에는, Linux용 유틸리티를 사용해 ESCD를 설정 가능하다면 좋다고 생각할 수 있다. 1999초의 시점에서는, 이러한 툴 은 아무것도 없었다. 따라서, 이것을 실행하는 마지막 수단으로서 Windows를 사용하게 되었던 것이다(같은 PC 에 Windows가 들어 있는 경우에는).

Windows에서 ESCD를 설정또는 수정시키는 방법은 3가지가 있다. 최초의 방법 은, DOS 또는 Windows 3.x용으로 만들어진 ICU 유틸리티를 사용하는 것이다. 이것은 Windows 9x/2k 에서도 동작한다(??). 두번째의 방법은, Windows 9x/2k 상에서 수동으로(「강제적인」)설정 하고, Windows의 정상 종료시 에 이 정보를 ESCD에 보존시키는 방법이다. 세번째 방법은 플러그&플레이에 없는 legacy디바이스에서만 이용할 수 있다. Windows가 이러한 디바이스의 존 재와 사용하고 있는 버스 자원을 알고 있으면, Windows는 이 정보를 ESCD 에 저장한다.

Windows가 디바이스의 설정을 자동적으로 한후, 사용자가「강제적으로 」설정 을 바꾸지 않으면, 이러한 설정은 어쩌면 ESCD에는 반영되지 않을 것이다. 물론, Windows는 자신 자신의 설정을 ESCD에 보존되고 있는 설정 에 맞추는 것 이므로, 결과적으로 간혹 일치하는 것도 있다.

Windows9x는 PnP operating system이고, 자동적으로 디바이스의 PnP 설정을 한다. Windows는 레지스터리(이것은 binary인 Windows 파일들에 저장 되어 있다)의 속에서 PnP 데이타베이스를 관리하고 있다. PnP

버스 자원 되에도 많은 설정 정보가 레지스터리에 들어 있다. PnP 버스 자원은, 현재 메모리에 기억되어 있는 것과(아마 대부분 같지만)하드 디스크에 저장되어 있는 것중 어느 것에나 있다. Windows98 에서 PnP 버스 자원(메모리에 보관 유지하고 있는 방법?)을 간접적으로 탐색하거나, 강제적으로 변경 하려면, 디바이스 매니저를 사용한다.

Windows98에서 디바이스 매니저를 호출하는 방법은 2 가지가 있다 :

- (1)My Computer –> Control Panel –> System Properties –> Device Manager
- (2)(right-click) My Computer –> Properties –> Device Manager

그리고 디바이스 매니저 내에서 디바이스를 선택한다.(같은 클래스의 디바이스가 복수개 있는 경우에는 복수 스텝의 조작이 되수도 있다.) 다으로 특성」,「자원」을 차례로 클릭 한다. 자원 설정을 수동으로 변경 하려면,「자동 설정을 사용한다」의 체크를 지우고,「설정의 변경」을 클릭 한다. 그러면, 설정을 변경 할 수 있지만, 설정을 변경할 수 없는 것도 있다. 변경할 수 있다면, 변환을 강제로 시킨것이 된다. 설정이 강요된 것을 나타내는 메세지 가 표시될수 있다. 그러나, Windows가 표시한 기존의 설정을 계속 사용하고 싶은데「강제적으로 」바꾸어 버린 경우에는, 일단 무엇인가 다른 설정 으로 한후 원래 설정으로 돌리지 않으면 안된다.

Windows98 에 있어서 어떤 설정이「강요」되고 있는지를 조사하려면 다음과같이 조작을 한다:Start –> Programs –> Accessories –> System Tools –> System Information –> Hardware Resources –> Forced Hardware. Windows에서 버스 자원을「강제적」으로 바꾼 경우에는 그 변경을 ESCD에 저장하지 않으면 안 된다.(Windows를 정상 종료 시키면 그렇게 된다).「시스템 정보」윈도우를 보면, Windows 상 에서 IRQ 와 I/O 포트가 어떻게 할당되어져 있는가를 조사 할 수 있다.

Windows에서 버스 자원의 충돌이 일어나지 않아도, Linux에서 충돌이 일어나는 경우가 있다. 그 이유는, Windows는 ESCD 와 다른 방법으로 버스 자원을 할당하고 있기 때문이다. 드문 경우 이지만, Windows 상의 모든 디바이스가 legacy 디바이스 이거나, 설정이「강요」되고 있는 경우에는, Windows 와 ESCD 의 설정은 완전히 같게 되는 것이다.

5.3.4 새로운 디바이스의 추가 (Linux 또는 Windows 에 있어서)

새로운 PnP 디바이스가 추가되고, 또한 BIOS의 설정이 "not a PnP OS"인 경우, PnP BIOS는 자동적으로 디바이스의 설정을 하고, 그 설정을 ESCD에 저장 하여야만 한다. non-PnP legacy 디바이스(또는 점퍼등으로 PnP를 무효로한 디바이스)의 경우에는, 이것을 처리하기 위한 옵션이 몇개 있다.

특정의 버스 자원(IRQ 등)이 예약되고 있어, PnP에서 할당해서는 안 되는 것을(CMOS 설정 메뉴를 사용해)직접 BIOS에 지정하는 것도 할 수 있다. BIOS가 이 정보를 ESCD에 기록하지 않는다. 그러나, 충돌이 일어난 경우에, 미리 선택해 있던 CMOS의 설정으로 ESCD의 설정을 덮어 쓸지를 BIOS의 메뉴로 선택할 수 있게 되어 있을지도 모른다. 다른 방법은, DOS 또는 Windows에서 ICU를 실행하는 방법이 있다. 보다 특별한 방법으로서, ICU를 Windows9x/2k 에 수동으로 인스톨하고 , 그 설정을「강요」하는(앞의절을 참조)방법도 있다. 설정이「강요」되면, Windows는 PC가 숏다운될 때에 ESCD를 갱신 할 것이다.

5.4 isapnp (isapnptools 의 일부)

유감이지만, PnP의 기초를 모르는 사람에는, isapnp에 대한 많은 문장 서의 내용은 알기 어려울 것이다. 본 HOWTO 에서는, isapnp 부속의 FAQ 와 같은 isapnp의 이해를 돕기 위한 설명을 한다. isapnp를 사용할 수 있는 것 은 ISA 버스 상의 PnP 디바이스뿐이다(PCI 버스에서는 사용할 수 없습니다). Linux 시작시에 "isapnp" 프로그램을 실행하면, /etc/isapnp.conf 파일 에서 지정된 자원값이 PnP 디바이스에 설정된다. 이 설정 파일 은 자동적으로 만들 수 있지만, 각종 옵션을 설정하려면 직접 편집해야 한다. isapnp를 사용하는 경우, 커넬 일부에 있는 디바이스 드라이버는 isapnp가 어드레스등을 하드웨어에 설정하는 것보다도 좀도 앞에서 실행된다. 결과적으로, 디바이스 드라이버는 디바이스를 찾아내는 것이 불가능하다. 드라이버가 정확한 어드레스에 억세스 되어도, 그 어드레스 는 아직 하드웨어에 설정되고 있지 않기 때문이다.

당신의 Linux 배포본이 isapnptools를 자동적으로 인스톨 할 경우에, isapnp는 시스템 시작시에 이미 실행되고 있을수도 있다. 이 경우에 해야 할 것은, "man isapnp.conf" 의 출력을 보면서 /etc/isapnp.conf 를 편집하는 것뿐이다. 그러나, 이 작업은 수동으로 PnP 를 설정하는 것과 그다지 다르지 않는 점에 주의해 주시오. 사용자는 어떠한 설정으로 할지를 정하고서 설정 파일을 편집해야 하기 때문이다. 설정 파일을 편집하는것에는, "pnpdump" 프로그램이 편리하다. 이 프로그램은 설정 파일을 대체로 만들어 주지만, 이 설정을 사용하기 전에 정확하게 편집 하지 않으면 안 된다. 이 파일에는 편집의 참고가 되는 코멘트가 포함되어 이다 . "isapnp" 를 설정에 이용하고, 또한 PnP BIOS가 있는 경우에, BIOS에는(설정시에)「PnP OS를 가지고 있지 않다」라고 지정하지 않으면 안 된다. 왜냐하면, 이 경우에도 PCI 디바이스의 설정을 BIOS로 하게 하고 싶기 때문이다. BIOS가 ISA 디바이스의 설정을 할 수 있을지도 모르지만 가, isapnp는 그것을 다시 재설정 한다

/etc/isapnp.conf 에서 사용되고 있는 용어가 처음에는 이상하게 생각될지도 모른다. 예를 들면, I/O 어드레스 0x3e8를 표현하는데, "(IO 0 (BASE 0x3e8))" 을 보았을 수 있다. "IO 0" 라는 것은, 이것이 디바이스가 사용하는 최초의 (0 번째의) I/O 어드레스 영역이라는 의미다. 이것을 표현하는 다른 방법으로는 "IO[0] = 0x3e8" 가 있지만, isapnp는 이 방법을 사용하지 않는다. "IO 1" 는, 이것이 이 디바이스가 사용하는 2 번째의 I/O 어드레스 영역이라는 의미다. 이하도 같다. "INT 0" 도 같은 의미이지만, IRQ(인터럽트)에 대해 용이하다. 1 개의 카드가 복수의 물리 디바이스를 갖는 것도 있지만, 상기의 설 명은 이들 디바이스 중 1개에만 해당한다.

5.5 PCI Utilities

새로운 패키지인 PCI Utilities (=pciutils, 잘못해 "pcitools" 로 불리우는 것도 있다.)는 PCI 버스의 PnP 설정을 수동으로 할 수 있도록 한다. "lspci" 는 버스 자원을 리스트 표시하고, "setpci" 는 하드웨어 디바이스의 자원 할당을 설정 한다. setpci는 주요한 스크립트로 사용되는 것을 상정하고 있는 것이다. 현시점에서 이것을 이용하기에는 PCI 설정 레지스터를 상세하게 이해하고 있을 필요가 있지만, 그 점에 관해서, 이 문서와 setpci의 메뉴얼에서는 자세하게 설명되어 있지 않다.

5.6 Linux PnP를 만들기 위한 커널을 패치하여 주시오.

David Howells 씨는 "Linux Kernel Configuration/Resource Manager" 라고 불리운다. Linux 커넬을 PnP 대응으로 하기 위한 패치를 작성했다. ("Hardware Configuration Manager" 라고 불리우는 것도 있다). 1999 년말 의 시점에서는, 이 패치를 그의 페이지로부터 입수할 수 없었다. 아마도 , 최근 버젼의 커넬에 대해서 사용할 수 있는 패치는 없을 것이다.

사전의 패치에 대해서는, 패치를 하여 얻어지는 커넬은 안정되어 있다고 저자는 주장하고 있었지만, 버그가 몇개 보고 되어 있다. 이 패치에 는, 시리얼 포트를 취급하는 방법을 설명하는 serial.txt 등의 문서가 추가되어 있다. 이 패치를 하면 /proc 디렉토리에「파일」이 만들어진다. 이들 파일에서 현재 상황을 조사하고, 이들 파일에 명령어를 보내는 것에 의해 설정을 바꾸는 것도 할 수 있다. 그러나, 문제가 한가지 있다. 디바이스 드라이버의 다수는 이 패치를 상정하고 있지 않기 때문에, 설정 을 하기 위해서는 지금까지 전통적인 설정 파일등을 사용 해야만한다. 이 패치의 WWW 페이지는 *www.astarte.free-online.co.uk* <http://www.astarte.free-online.co.uk>이다.

5.7 Windows 에 의한 설정

Windows9x (또는 Windows 2k)가 Linux와 같이 PC에 들어 있으면, 단순히 Windows를 시작해서 PnP를 설정 하시오. 그 후, Windows(또는 DOS)로부터 Linux를 시작 한다. Windows가 PCI 디바이스 레지스터로부터 IRQ 를 제거해 버리는 것이 보고 되고 있다. 이 경우에 Linux는 0 인 IRQ가 검출되었다고 에러를 출력한다. 따라서, 이 분법은 사용할 수 없을지도 모른다.

5.8 디바이스 드라이버에 의한 설정

몇개의 드라이버는, PnP의 방법을 사용해 하드웨어 내에 버스 자원을 설정하지만, 이것은 그 드라이버가 제어되는 디바이스에 대한것 뿐이다. 드라이버가 설정을 끝내 이후, 드라이버는 분명하게 설정을 알고 있으므로, 사용자는 이 정보를 드라이버에게 알릴 필요는 없다.

이 방법에 동반되는 문제는 설정의 중복이다. 이 정보 모두를 드라이버에 내장시키는 것은 어렵다. 그리고, 드라이버는 다른 디바이스가 필요로 하고 있는 버스 자원을 잡아 챌지도 모른다. 이것에 의해 사용자의 설정은 간단하게 되지만, PnP 대응 Linux 커넬을 사용하는 방법이 좀 더 좋을 것이다. 자세한 것은 "Linux에서 PnP를 능숙하게 대처해야 할 필요성" 을 보시오.

5.9 PnP 에 관련하는 소프트웨어와 문서

- isapnptools 의 홈 페이지 *www.roestock.demon.com.uk/isapnptools/* <http://www.roestock.demon.com.uk/isapnptools/>
- Linux 커넬 PnP를 대치 하는 패치 *www.astarte.free-online.co.uk* <http://www.astarte.free-online.co.uk>
- PnP 드라이버 프로젝트 *www.io.com/~cdb/mirrors/lpsg/pnp-linux.html* <http://www.io.com/~cdb/mirrors/lpsg/pnp-linux.html>
- Microsoft에 의한 PnP의 사양 *www.microsoft.com/hwdev/respec/pnpspecs.htm* <http://www.microsoft.com/hwdev/respec/pnpspecs.htm>
- 서적: PCI System Architecture, 3rd ed., 저자 Tom Shanley 외, MindShare, 1995년. PCI 버스가 가지고 있는 PnP 적인 기능의 설명이 있다.
- 서적: Plug and Play System Architecture, 저자 Tom Shanley, MindShare, 1995년. ISA 버스에서 PnP에 대한 자세한 해설이 이다. PCI 버스에서 PnP에 대한 개요밖에 쓰여져있지 않다 .
- 서적: Programming Plug and Play, 저자 James Kelsey, 1995년. PnP BIOS에서 통신을 실행하는 프로그래밍에 대한 자세한 해설이 있다. ISA, PCI, PCMCIA 버스를 담당하고 있다.

제 6 절 Tell the Driver the Configuration

6.1 소개

이것을 하는 방법은 완전히 드라이버에 의존 한다. 드라이버에 의해서 물리 디바이스가 어떻게 설정 되는지를 조사하는 방법은 여러개 가지고 있다 . 가장 그단적인 경우에는, 버스 자원을 커넬의 hard-code로 recompile 을 해야한다. 이에 반대되는 경우, 설정은 모두 자동적으로 행해지고, 사용자는 아무것도 할 필요가 없다. 하드웨어로 버스 자원 설정조차도 PnP 구조를 사용해 행해진다.

이 중간의 경우는, 자원 정보를 드라이버에 주거나, 파일에 설정 하는 경우이다. 경우에 따라서는, 디바이스가 있는 경우 어드레스에 대한 드라이버를 탐색 하는 것도 있다. 그렇기 때문에 다양한 IRQ를 조사하고, 어느 IRQ를 사용할 수 있을지를 조사 한다. 이것은 자동으로 행해지는 경우도 그렇지 않은 경우도 있다. 이 외에, 드라이버가 PnP의 구조를 사용해 디바이스 존 재와 버스 자원의 설정을 조사하는 것이 있지만, 실제로 설정은 행해지지 않는다. 이 정보는 /proc 디렉토리 내의 몇개의 파일에도 나타난다.

버스 자원을 커넬과 loadable 모듈에 파라미터로 줄 필 요가 있는 것도 있다. 지정가능한 파라미터에 대해서는 /usr/lib/modules_help/descr.gz를 보시오. 로드되는 모듈 은, /etc/modules에 파라미터 첨부로 열거 되어 있다. 경우에 따라서, 버스자원은 파라미터로서 커넬에 주어지는 것이 있다. 파라 미터는 lilo.conf 파일에 append="..." 라는 형태로 지정 한다. 이 파일을 편집한 후에는, 설정을 커넬의 부트 코드에 보존하기 위해서 lilo를 실행 해야만 한다.

드라이버가 버스 자원을 찾아내기 위한 방법은 모두 제각각이지만, 최종적 인 목적은 같다. 하드웨어와 그것에 대응하는 드라이버는 매우 많이 있으므로, 드라이버가 버스 자원을 찾아내는 방법과 드라이버가 필요로 하는 정 보를 확실하게 입수하기 위해서 해야 할것에 대해서는, 드라이버 문서를 볼 필요가 있을 것이다. 몇개의 드라이버에 관한 간단한 정보를 아래에서 설명 한다.

6.2 시리얼 포트 드라이버: setserial

표준 시리얼 포트 드라이버(멀티 포트 카드를 제외한)에 대해서는, setserial를 사용해 드라이버의 설정을 한다. setserial는 많은 경우, 시작파일로부터 실행 된다. 새로운 버젼의 setserial는 /etc/serial.conf 파일을 사용한다. 이 파일은 seetserial 명령어를 단순히 사용하는 것으로「편집」되고, setserial를 사용해 설정한 내용이 설정 파일 setserial에 등록된다. serial.conf 파일 은, setserial 명령어가 시작 파일로부터 실행되었을 때에 참조되는 것이다. 이러한 설정이 되어 있는지는 배포본마다 다를지도 모른다.

setserial 명령어는 주는 옵션에 따라서, 서로 다른 두가지 사용법이 가능하다. 사용법의 하나는 수동으로 드라이버에 설정을 알리는 것이다. 또 하나 의 사용법은, 지정된 어드레스를 조사하고, 그곳에 시리얼 포트가 있을지 를 알려주는 것이다. 이 어드레스를 조사하고, 이 포트에 대해 어느 IRQ가 사용되고 있는지의 검출을 시험도 가능하다. 드라이버는 시작시에 setserial에 적당한 것을 실행하지만, IRQ 탐색은 하지 않고, 오류가 있을지도 모르는「표준」IRQ를 할당하는 것뿐이다. 드라이버는 포트가 존재하는지를 탐색 한다. 자세한 것은 Serial-HOWTO를 보시오.

6.3 사운드 카드 드라이버

6.3.1 OSS-Lite

I/O 어드레스, IRQ, DMA 채널을 파라미터로서 모듈에 알려주거나, 커넬에 컴파일 해야만 한다. 그러나, 일부 PCI 카드는 자동적으로 검출된다(아마, lspci 명령어등을 사용하는 것에 의해 행해질 것이다). RedHat에는 "sndconfig" 라는 프로그램이 공급되고 있다. 이 프로그램은 ISA PnP 카드를 검출하고, 검출된 버스 자원을 사용해 모듈의 로드 설정을 한다.

6.3.2 OSS (Open Sound System) 와 ALSA

이것들은 PnP의 구조를 사용해 카드를 검출하고, 적절한 드라이버의 선택과 로드를 한다. ISA PnP 카드의 자원 설정도 실행한다. 버스 자원의 충돌을 피하기 위해서 수동으로 조정 할 필요가 있을지도 모른다. ALSA 드라이버의 경우에는, ISA PnP의 기능은 생략이 가능하고, 원하면 isapnptools를 사용하는 것도 가능하다.

제 7 절 현재의 설정을 알기 위해서는?

여기서의「설정」라는 것은 PnP 버스 자원(어드레스, IRQ, DMA)의 할당일 것이다. 디바이스가 어떻게 설정이 되어 있는가? 」라는 질문은, 디바이스와 디바이스 드라이버의 모두에 대한 질문이다. 각 부분은 같은 대답은 하여야 한다.

1. 디바이스 드라이버의 소프트웨어의 설정은 어떻게 되어 있는가? (즉, 드라이버는 하드웨어 설정이 어떻게 되고 있다고 생각하고 있는가?)
2. 디바이스 자체에는 어떤 설정이 되고 있는가?

당연히, 디바이스 하드웨어와 그 드라이버의 설정은 같아야만 한다. (그리고 대부분은 같다). 정상적으로 동작하지 않는 경우는, 어딘가에 서로 다른것이 있는 것이다. 즉, 실제의 하드웨어 설정에 대해서 드라이버가 가지고 있는 정 보가 잘못되었다는 것이다. 이것은 문제를 일으킨다. 사용하는 소프트 웨어가 틀림 점을

적절히 지적해 주지 않으면(또는 바른 설정을 해 주지 않으면), 하드웨어 디바이스와 그 드라이버가 어떻게 설정 되고 있는 지를 조사할 필요가 있다. Linux의 디바이스 드라이버는「모든 정보를 표시한다」경우도 있지만, 하드웨어의 설정을 조사하는 것 은 간단하지 않을지도 모른다.

이것과는 다른 문제도 있다. 즉, 화면에 설정에 관한 메세지가 표시 되었을 때, 이것이 디바이스 드라이버의 설정인가, 디바이스 하드웨어의설 정인가, 혹은 그 양쪽 모두인지 분명치 않는 경우가 있다는 문제이다. 디바이스 드라이버를 설정한 후, 하드웨어가 이것과 같은 설정이 되어있는지를 디바이스 드라이버가 확인하고 있다면, 드라이버가 보고하는 설정은 하드웨어와 드라이버 공통의 설정이 되어 있는 것이다.

그렇지만 이것을 하지 않는 일부의 드라이버는, 확인하고 되지 않은 설정을 받아들일지도 모른다. 예를 들면, "setserial" 는 확인되지 않은 설정도 받아 들인다. (비록 버스 자원을 조사하도록 지시한 경우에 있어서도). 이와같이 "setserial" 이 표시하는 것은 드라이버의 설정뿐이고, 하드웨 어의 설정은 표시하지 않는다.

7.1 Boot-time Messages

설정 정보의 일부는, 컴퓨터를 시작했을 때 BIOS의 메세지와, Linux의 시작 메세지를 읽으면 알수 있다. 이들 메세지는 보여지는 시간이 너무 빠라 읽을수 없는 경우가 많지만, 메세지가 멈춘 뒤에 Shift- PageUp을 몇 번정도 누르면 메세지를 스크롤백시킬 수 있다. forward하려면 Shift-PageDown를 사용한다. 쉘 명령어로서 "dmesg" 라고 입력하면, Linux의 커넬 메세지라면 언제나 표시 할 수 있다. 그러나, (BIOS 메세지를 포함한)매우 중요한 메세지는 나타나지 않는 것도 있다. Linux가 출력한 메세지는, 디바이스 드라이 버가 알고 있는 설정만을 표시하는 것이다. 이 설정은 혹시 , 오류가 있는 설정 파일로부터 읽혀진 것일수도 있다.

BIOS로부터의 메세지는 그 시점에서 하드웨어 설정을 표시하지만, PnP OS, isapnp, PCI Utilities 등이 후에 설정을 변경할 수도 있다. BIOS의 메세지는 최초에, Linux의 메세지보다도 앞에 표시된다. 뒤부터 Shift-PageUp를 사용해 메세지를 읽는 방법 대신에, "Pause" 키를 눌러서 메세지 출력을 멈추는 방법을 시도해 보시오. 어떤 키를 누르면 다시 동작을 시작한다. 그러나, Linux 로부터 메세지가 출력되기 시작하면, "Pause" 키는 더 이상 사용할 수 없다. "Pause" 키는 Linux 로부터의 메세지 출력을 멈추지 못하지 때문이다.

7.2 디바이스 드라이버는 어떻게 설정되고 있는가?

커멘드 라인에서 실행하여 (시리얼 포트용 "setserial" 처럼), 설정을 표시하는 프로그램이 있을지도 모른다. /proc 디렉토리 트리 는 편리하다. /proc/ioports는 드라이버가 사용하고 있는 I/O 어드레스를 표시 한다. (혹은, 어드레스가 오류가 있는지를 체크한다). 이것을 사용해 하드웨어의 I/O 어드레스를 설정하는 것은 할 수 없다.

/proc/interrupts는 현재 사용 중인 인터럽트만을 표시 한다. 드라이버에 할당되어 있는 인터럽트의 다수는, 사용 중인 것이 없을 때문에 완전히 표시되지 않는다. 예를 들면 floppy 드라이브에 floppy disk가 들어있어 언제라도 사용할 수 있는 상태가 되고 있어도, 실제로 사용되고 있지 않으면 인터럽트는 표시되지 않는다. 반복하지만, 여기에 표시되는 것은, 인터럽트가 하드웨어내에 존 재한다는 것이 아니다. 하드웨어에 존재하지 않는 것을 알수 있는 것은, 이 인터럽트가 발생된 특별한 인터럽트가 0 개인지를 조사한다. 그러나 인터럽트가 몇개쯤 발생되었다고 나타나고 있어도, 이 인터럽트가 그 디바이스상에서 일어났다는 보증은 없다. 현재 사용중 아닌 디바이스상에서 일어난 것일지도 모르고, (커넬 경유로)사용되는 것이 없는 디바이스에서도, 어떤 이유로 인터럽트를 발행할지도 모른다.

7.3 하드웨어 디바이스는 어떻게 설정되고 있는가?

"lspci" 명령어를 사용해, PCI 버스상의 디바이스에 대한 버스 자원의 할당을 조사하는 것은 간단하다. 버젼 2.2 보다 전의 커넬에서는, /proc/pci 와 /proc/bus/pci 를 참조 하시오. 또 /proc/pci의 IRQ는 16진수 표기로 표시되는 것에 주의 하시오. /proc/bus/pci/devices 의 해독에 대해서 걱정할 필요는 없다. "lspci" 가 대신 행해준다 .

ISA 버스의 경우에는 pnpdump –dumpregs를 사용하려고 할지도 모르지만, 이것은 확실한 방법이 아니고, 결과는 판단독하기 어려울지도 모른다. 또 pnpdump 가 찾아낸「read-port 어드레스」과 디바이스가 찾아낸「 I/O 어드레스」를 혼동하지 말아 주세요. 이것들은 같은 것이 아니다. ISA 버스상(혹은 PnP의 옛것)에서 발견되지 않는 하드웨어를 검출하려면, "scanport" 프로그램을 시험해 보시오. 그러나, PC 가 위험하다는 것에 주의합시다. 또, 이것은 IRQ 표시 를 나타내지 못하고, 확실하게 하드웨어를 인식하는 것도 아니다.

BIOS가 시작시에 출력하는 메세지를 보면, 하드웨어 설정이 어떻게 되어 있는지를 알 수 있다. BIOS에 설정을 맡기고 있다면, 이것은 이전과 동일한 것이어야 한다. Linux의 시작시에는, 드라이버나, 하드웨어가 있을지를 확 인(그리고 IRQ와 DMA 도 설정되고 있으면 그것도 확인)한 메세지를 출력 한다. 물론, 디바이스가 바르게 동작하고 있으면, 디바이스와 드라이버 의 설정은 동일하다.

제 8 절 부록

8.1 어드레스

어드레스에는 3 개의 타입, 즉 메인 메모리 어드레스, I/O 어드레스, 설정 어드레스가 있다. PCI 버스에서, 설정 어드레스는 I/O 어드레스와 닮은 다른 어드레스 공간을 구성 한다. 어려운 ISA의 설정 어드레스의 경우를 제외하고, 버스상의 어드레스가 메모리 어드레스, I/O 어드레스, 설정 어드레 스로 있는지는, 버스의 다른 선(traces)의 전압에 의해서만 결정된다.

8.1.1 ISA 버스의 설정 어드레스 (Read-Port등)

ISA 버스의 경우, 기술적으로 설정 어드레스 공간은 없지만, CPU 가 PnP 카드의 PnP 설정 레지스터에 억세스하기 위한 특별한 방법이 있다. 의 목적을 위해서, 3 개의 @ I/O 어드레스가 할당되어 있다. 그러나, 각각의 카드에 3개씩 어드레스가 할당되었다는 것이 아니라, 모든 카드에서 3 개의 어드레스를 공유 한다.

3 개의 어드레스 이름은, 각각 read-port, write-port, address-port 이다. 각 포트의 사이즈는 정확히 1 바이트이다. PnP 카드는 제각기 레지스터를 많이 가지고 있기 때문에, 이 3 개의 어드레스에서는 카드 1 매분의 레지스터에 대한 것도 충분하지 않다. 특정의 카드와 통신 하기 위해서는, 특별히 할당되어진 카드의 번호(핸들)을 write-port 어드레스를 사용해 모든 카드에 보낸다. 그러면, 이 핸들을 가진 카드만이 감시 상태가 된다. 다음으로(이 카드의)설정 레지스터의 어드레스 포트에 보낸다(어드레스 포트는 모든 카드가 공유되어 있지만, 현재 포트의 감시를 하고 있는 것은 1 개뿐이다). 그런 다음, 통신은 그 카드의 설정 레지스터 중의 하나에서 일어난다. 이것은 read-port로부터 읽어 내거나, write-port에서의 쓰기에 의해 행해진다.

write-port의 어드레스는 반드시 A79로, 어드레스 포트의 어드레스는 반드시 279이다(16 진수). 그러나, read-port 포트의 어드레스는 고정이 아니며, 그 외 의 ISA 카드와 겹치지 않다고 생각되는 어드레스에 설정 프로그램이 설정한다. 어드레스가 겹친 경우는 변경된다. read-port의 어드레스를 사용해, 모든 PnP 카드를「프로그램한다」는것이 가능하다. 그렇지만 , isapnp 등을 사용해 데이터의 설정과 체크를 하는 경우에는, 이 read-port 어드레스를 정하지 않으면 안 된다.

8.1.2 어드레스의 범위

본문서에서는「어드레스」라는 단어에서 어드레스에 인접하고 있는 범위를 가리키는 것으로 가끔 사용되었다. 어드레스는 바이트 단위로 주어지므로, 1 개의 어드레스 는 1 바이트의 용량밖에 없다. 그러나, I/O(와 메인 메모리)어드레스 에는 이것보다 큰 사이즈가 필요하다. 따라서, I/O 어드레스에는 예를 들면, 8 바이트의 범위가 사용되는 것이 많고, 디바이스에 할당되어진 메인 메모리 어드레스의 범위는 이것보다도 좀 더 커야한다. 시리얼 포트 (I/O 디바이스)의 경우에는, 디바이스 최초의 I/O 어드레스(3F8 등)을 부여하는 것으로 충분하다. 왜냐하면, 이 디바이스에 대한 어드레스의 범위는 8 바이트 밖에 없는 것으로 상식화 되어 있기 때문이다. 이 범위 중 최초의 어드레스는「베이스어드레스(base address)」라고 불리운다.

8.1.3 어드레스 공간

ISA 버스의 경우, I/O 어드레스와 (메인)메모리 어드레스의「공간」에 접근할 때에는, 같은 어드레스 버스가 사용된다.(어드레스에 사용되는 배선이 공 유된다). 그러면, 디바이스는 어드레스 버스에 나타나는 어드레스가 메모리 어드레스인지, I/O 어드레스인지를 어떻게 알수 있을까? 사실는 버스상에는 4 개의 전용 선이 있어, 이러한 정보를 전하는 것이다. 4개의 선 중의 특정한 것에 신호가 흐른 경우, 이것은 CPU가 I/O 어드레스 로부터 데이터를 읽으려하고 있는 것과, 메인 메모리는 버스 상의 어드레스를 무시하는 것을 의미 한다. 나머지의 3 개의 선도 비슷한 목적으로 사용된다. 간단하게 설명하면, 읽기와 쓰기의 선이 메인 메모리와 I/O 어드레스 의 양쪽을 위해서 존재한다(선은 전부 4 개이다).

PCI 버스의 경우도 기본적인 사고방식은 같이 4 개의 선을 사용하지만, 그 사용 방법은 조금 다르다. 즉, 4 개 중의 1개에 신호를 흘리는 것이 아니고, 4 개의 선을 전부사용해 2 진수를 흘린다.(16 가지로 조합가능하다). 이렇게 하는 것에 의해, 보다 많은 정보를 보낼 수 있다. 16개 중 4 개는 앞 에서 말한 I/O 공간과 메모리 공간을 위해서 사용한다. 추가로, 설정 어드레스 공간이 2 개를 더 사용한다. 나머지의 10 개는 다른 목적으로 사용할 수 있도록 남겨진다.

8.1.4 어드레스 범위 체크 (ISA에 있어서 I/O 어드레스의 충돌 검사)

ISA 버스의 경우, 같은 어드레스를 사용하는 다른 카드가 없는 지를 체크하기 위한 방법이 각 PnP카드에 구축 되어있다 . 복수의 카드가 같은 I/O 어드레 스를 사용하고 있으면, 어느 쪽의 카드도 바르게 동작하지 않을 것이다. 효과적인 PnP 설정 프로그램은 이러한 충돌이 일어나지 않도록 버스 자원을 할당하지만, 이 경우도 숨어 있는 legacy 카드가 중복되는 어드레스 를 가고 있을지도 모른다.

이 테스트는, 카드가 자신의 I/O 레지스터에 테스트 번호를 설정하는 것에 의해 행해진다. 다음으로 PnP 설정 프로그램은 이것을 읽고, 같은 테스트 번호가 읽는 것을 검사 한다. 이것은 다른 경우에는 무엇인가 문제가 있다. (예를 들면, 다른 카드가 같은 어드레스를 사용하고 있다). 이것은 다른 테스트 번호로 같은 테스트를 반복한다. 이 테스트는 실제로는 카드에 할당된 I/O 어드레스의 범위를 체크 하는 것으로「범위 체크」라고 불리운다. 이것은 어드레스 충돌 테스트라고 부르는 편이 좋을지도 모른다. 어드레스 충돌이 있으면 사용자에게 에러 메세지가 보내므로, 사용자는 스스로 이것을 해결하지 않으면 안 된다.

8.1.5 메모리를 통한 직접 통신

전통적으로, 대부분의 I/O 디바이스는 CPU 와 통신할 때에 I/O 메모리 만을 사용하였다다. 예를 들면, 시리얼 포트가 이것에 해당 한다. CPU 상에서 동작하고 있는 디바이스 드라이버는, I/O 어드레스 공간과 메인 메모리에 대해 읽기 와 쓰기를 한다. 보다 빠른 방법으로서, 디바이스가 직접 메인 메모리 에 데이터를 쓰는 방법이 있다. 이것을 할수 있느 방법의 하나가, "DMA 채널" 혹은 bus mastering의 이용이다. 또, 메인 메모리 공간의 일부를 디바이스에 할당하는 방법도 있다. 이와 같이 해서, 디바이스는 일부러 DMA 와 bus mastering을 사용하지 않아도 직접 메인 메모리에 데이터 쓰기를 할 수 있다. 이러한 디바이스는 보통, I/O 어드레스도 사용할 수 있다.

8.2 인터럽트 –상세

인터럽트는 많은 정보를 전달 할 수 있지만, 간접적이다. 인터럽트 신호(배선상의 전압)는, 어떤 디바이스가 처리를 필요 로 하고 있는 것을 인터럽트 콘트롤러라고 불리우는 칩에 전달한다. 그러면 인터럽트 콘트롤러는 CPU에 신호를 보낸다. CPU는 이 디바이스의 드라 이버를 찾아내어,「인터럽트 서비스 루틴」(또는「인터럽트 핸들러 」)라고 불리우는 드라이버의 일부분을 실행 한다. 이「루틴」은 무언가 동작 하는지를 조사하고, 디바이스와의 데이터 전송등의 문제를 처리하려고 한다. 이 프로그램(루틴)에서, 무엇이 일어난지에 대해서 간단하게 조사 할 수 있다. 왜냐하면, 드라이버는 자신이 알고 있는 어드레스 내에, 조사하기 위한 레지스터를 가지고 있기 때문이다.(디바이스의 IRQ 번호와 I/O 어드레스가 바르게 설정되어 있는 경우에는) 이들 레지스터에는 디바이스에 관한 상태 정보가 저장 되어 있다. 드라이버는 이 레지스터의 내용을 읽고, 이것을 조사해서, 무엇이 발생했는가를 찾아 적합한 동작을 한다.

따라서, 각 디바이스 드라이버는 감시하는 인터럽트 번호(IRQ)를 알고 있을 필요가 있다. PCI 버스(또 커넬 2.2 이후에서는 ISA 버스상의 시리얼 포트) 의 경우에는, 복수의 디바이스가 같은 IRQ 번호를 공유하는 것이 가능 하다. 이러한 인터럽트가 발생되면, CPU는 그 인터럽트를 사용하고 있는 모든 디바이스 인터럽트 서비스 루틴을 전부 실행한다. 최초의 서비스 루틴이 먼저 하는 일은, 인터럽트가 정말로 그 디바이스에 대해 실행되는지를 확인하는 것이다. 인터럽트가 없으면(false alarm) 이 루틴은 종료하고, 다음 서비스 루틴을 개시 한다. 이후도 같은 형태이다.

8.3 PCI 의 인터럽트

PCI 의 인터럽트는 ISA의 인터럽트와는 다른 것이지만, 보통은 IRQ에 mapping되기 때문에, 동작은 대부분 같다. 주요한 차이는 PCI 에서는 인터럽트 를 공유할 수 있는 점이다. 이 공유는 자동적으로 행해진다. 따라서, 특별한 하드웨어와 소프트웨어는 필요 없다. 이전에는 인터럽트의 공유가 제대로 동작하지 않다는 보고도 있었지만, 이것은 대부분 디바이스 드라이버 소프트웨어의 문제 이다. PCI 용 디바이스 드라이버는 모든 인터럽트 공유 의 기능을 가지고 있는 것으로 되어있다. 단, 같은 인터럽트를 PCI 버스 와 ISA 버스에서 공유하는 것은 할 수 없는 점에 주의 하시오. 그러나, 인터럽트가 중복되는 디바이스를 동시에 사용하는 것이 없으면, 부적절하게 인터럽트를 공유하고 있어도 가끔 동작할 수 있다. 여기서「사용한다」라는 것은, 동작하고 있는 프로그램이(C 언어 프로그램의 의미로)디바이스를 오픈하는 것이다.

BIOS의 CMOS를 설정하거나, 낡은 PCI 카드의 점퍼를 설정 하려면, 어쩌면 PCI의 인터럽트 시스템에 대해서 자세한 지식이 필요할 것이다. 각 PCI 카드는 INTA# 로부터 INTD# (A, B, C, D)까지의 4 개의 인터럽트를 사용할 수 있다. 따라서, 슬롯이 7 개있는 시스템에서는 7 x 4 = 28 개의 인터럽트 선을 별개로 가지는 것이 된다.그렇지만, 사양에서는 인터럽트 선의 수는 이것보다 적어도 좋다고 되어 있다 . 또한, 인터럽트는 공유할 수 있으므로, 이것은 그다지 제한하지 않는다. 여기서, 이것들 의 선(배선과 traces)을 W, X, Y, Z 라고 부른다. 또, 슬롯 3 으로부터의 인터럽트 B를 인터럽트 3B라고 한다. 그러면, 선 W 를 사용해 인터럽트 1A, 2B, 3C, 4D, 5A, 6B, 7C를 공유할 수 있다. 공유는, 선 W 를 물 리적으로 1A, 2B 등에 접속하는 것으로서 실현 한다. 같이, 선 X 를 사용해 인터럽트 1B, 2C, 3D, 4A, 5B, 6C, 7D를 공유할 수 있다. 그리고, 시작시에 BIOS가 W, X, Y, Z를 IRQ 에 map 한다. 그 후 BIOS는, 각각 의 디바이스가 map된 IRQ를 각각의 디바이스의 하드웨어 레지스터에 쓴 후에, 디바이스 문의를 하는 모든 것은 디바이스가 사용하는 IRQ를 알 수 있다.

PCI 사양에서는, 먼저 언급된 배선 W, X, Y, Z 에 INTA#, INTB#, INTC#, INTD# 라는 라벨이 붙어 있다. 그렇지만, 이 정식 기법은 혼동하기 쉽다. 왜냐하면, 슬롯과 PCI 버스의 어느 쪽에 주목하고 있는지에 대한 INTA# 의 의미가 두가지로 바뀌기 때문이다. 예를 들면 3C가 X 에 맵되고 있는 경우, 슬롯 3의 INTC# 가 PCI 버스의 INTA# (X) 에 배선되고 있다고 말하는 것이다. 혼동하기 쉬운 기법이다.

필요 사항은 다른것도 있다. PCI 슬롯에서는, 하위 문자의 인터럽트로부터 사용해야 한다. 따라서, 슬롯이 1개밖에 인터럽트를 사용하지 않는다면, 인터럽트 INTA# 이어야만 한다. 인터럽트를 2 개사용한다면 , INTA# 와 INTB# 이어야 한다. 이후도 같다. 슬롯내의 카드는 디바이스를 8 개까지 가질 수 있지만, PCI 인터럽트의 할당 은 4 개밖에 없다. 인터럽트는 공유할 수 있기 때문에, 이것으로 문제는 없어지고, 8 개 의 디바이스는 각자 인터럽트를 가질 수 있다. 디바이스의 PCI 인터럽트 문자는, 고정값으로서 디바이스에 하드웨어적으로 결선되고 있는 것도 자주 있다.

BIOS 는 ISA 버스에 설정되어 있는 IRQ(인터럽트)와 충돌하지 않도록, PCI 의 IRQ 를 할당한다. CMOS 의 BIOS 메뉴에서 사용자가 IRQ를 PCI 버스에 할당하는 것도 가끔 있다. (그러나, 이것이 간단하지 않다는 것은 이미 기술되었다.) IRQ의 mapping를 설정한 후에는, Windows가 PCI 카드 의 IRQ를 모두 0 으로 만들어 버리는 경우도 있다. 따라서, Windows 를 사용하고 있는 사람이 Windows에서 Linux를 시작한 경우, Linux 에서는 IRQ가 0 라는 틀린 결과만을 얻는 경우도 있다.

독자의 여러분은 PCI가 IRQ(ISA 버스)를 사용하고 있기 때문에 늦는 이유를 생각 할지도 모른다. 그렇지만, 그것은 바른것이 아니다. ISA의 인터럽트 제어칩은 CPU 에 직결하고 있는 배선을 가지고 있으므로, 곧바로 CPU 에 신호를 보낼 수 있다. ISA 어드레스와 데이터 버스상의 신호는 PCI 버스 경유로 CPU 에 닿게 되지만, IRQ의 인터럽트 신호는 대부분 직접 CPU 까지 보낸다.

8.4 Isolation

Isolation은 ISA 버스에서만 사용할 수 있다. 이것은 ISA 버스상의 각 PnP 디바이스에 일시적인 핸들(ID 번호또는 카드 선택 번호(Card Select Number, CSN))를 할당하기 위한 복잡한 방법이다. 이것보다도 효율적인 방법 (그러나, 한층 복잡하다)이 있으므로, 그것이 단순한 방법이라고 말해질 수도 있다. Isolation에서는, 감시를 하고 있는 모든 PnP 디바이스에게 PnP 쓰기 대해서, 쓰기 어드레스 1개만이 사용된다. 이 쓰기 어드레스는, 각각의 PnP 디바이스에 고유의 핸들을 보내기 위해(할당한다.) 사용된다. 이 핸들의 할당은, 핸들이 공통의 어드레스에 보내진(쓰여졌다)때에 디바이스 1개만이 대기하고 있는 것 이 필요하다. 모든 PnP 디바이스는 Isolation의 처리로 사용하는 고유의 시리얼 번호를 가지고 있다 . Isolation의 동작은 게임과 비슷하다. 이것은, 모든 PnP 디바이스가 연결되어 있는 오직 1개의 공통 버스 배선과 Isolation 프로그램이 가지고 있는 값을 동등한 것으로 수행한다.

「게임」 최초의 라운드에서는, 모든 PnP 디바이스는 이 선을 감시하고, 여기로 비트열을 동시에 보낸다. 허용되는 비트값은 1 (positive voltage)또는 전압 없음의 "open 0" (open circuit or tri-state)의 어느 것이다. 계속해서, 각각 의 PnP 디바이스는 이 선으로 시리얼 번호를 비트마다 상위 비트로부터 전송을 시작한다. 어떤 디바이스가 1를 보내면, 접속되어 있는 다른 디바이스는 모두 1을 받는다. 모든 디바이스가 "open 0"을 보내면, 접속 되어있는 디바이스는 아무것도 받지 않는다. 이 목적은, (최초의 라운드가 종료될때 까지) 가장 높은 시리얼 번호를 가지는 것 이외의 것을 제거하는 것이다.「제거한다」라는 것은, 이 디바이스는 쓰기 어드레스의 감시를 그만두지만, 게임에 이겨 남아 있는 모든 디바이스는 이 어드레스의 감시를 계속한다는 것이다. 이것은 "dropping out" 라고도 말한다. (시리얼 번호의 길이는 모든 같다는 점에 주의해 주시오.)

먼저, 아직 핸들을 받고 있지 않은 디바이스 모두가 처음으로 배선에 놓인 시리얼번호의 가장 상위 비트에 대해서만 생각 보자. 어떤 PnP 디바이스가 0 (open 0)를 보냈는데 1 을 받은 경우, 이것은 다른 PnP 디바이스가 보다 높은 시리얼 번호를 가지고 있다는 것이다. 즉, 이 디바이스는 이 라운드로부터 일시적으로 탈락하고, 이 라운드가 끝날 때까지는 비트열을 읽을 수 없게 된다. (라운드가 끝난 시점에서 승자, 즉 가장 큰 시리얼 번호를 가지고 있는 디바이스에 핸들이 할당되어진다.) 이 때, 게임에 남아 있는 디바이스는 모두 같은 선두 비트(1)을 가지고 있다. 그런데, 이 라운드의 연속되는 자리수를 제거하고 요구한 "stripped serial number" 만을 생각하면 좋을 것이다. 그 후에는 이 단락 의 선두로 돌아오고, 시리얼 번호 전체를 조사할 때까지 반복한다(모든 것이 0 의 경우에 대해서는 아래를 참조해 주시오).

가장 높은 시리얼 번호가 게임으로부터 제거되지 않는 것은 분명하다. 그러나, (손상된 것도 포함해)시리얼 번호의 선두의 자리수가 모두 0 이었던 경우 는 어떻게 될 것인가 ? 이 경우에는「open 0」이 배선에 보내지고, 모든 디바이스는 게임에 참가했던 그대로 된다. 모든 디바이스의 선두의 자리수가 0 이라면 무승부가 되고, 앞의 단락에서 1을 제거한 것과 같게 0를 제거한다. 그리고 게임은 계속되고, (시리얼 번호의)다음 자리수가 보내진다.

라운드가 끝나면(참가 디바이스가 남아 있는 시리얼 번호의 하위 비트를 계속 보낸 후에), 가장 높은 시리얼 번호를 가진 PnP 디바이스가 1개 만 남는다. 이 디바이스에는 핸들이 할당되어지고, 그 이후는 게임 에 참가할 수 없다. 그리고, 전의 라운드 도중에서 탈락한(아직 핸들을 할당되어지지 않은)모든 디바이스는 게임에 다시 참가하고, 1개 적은 디바이스의 참가로 새로운 라운드를 시작한다. 이와 같이 해 모든 PnP 디바이스에 핸들을 할당한다. 이 알고리즘이 바르게 동작하는 것 은 간단하게 증명할 수 있다.

한 번 할당되어진 핸들은 각 PnP 디바이스를 가리키기 위해서 사용되고, PnP 와 의 디바이스 설정 정보의 교환에 사용된다. 이 핸들은 PnP 설정 으로만 사용하는 것이고, PnP 디바이스와의 통상적인 통신에는 사용되지 않는 점에 주의 하시오. 컴퓨터 시작시에는 핸들은 모두 상실되므로, PC를 시작할 때에 PnP BIOS 는 isolation 처리를 한다.

이상