가상머신의 원격 자원 입출력 지원을 위한 디바이스 드라이버 기반 프레임워크의 개발

박세진, 강상우, 박찬익

포항공과대학교 컴퓨터공학과/정보통신대학원
{baksejin, kkangeva, cipark}@postech.ac.kr

요 약

본 논문에서는 가상머신에서 원격 자원 입출력 지원을 위한 디바이스 드라이버의 구현을 소개하고 있다. 일반적으로 가상머신에서 동작하는 가상 데스크탑 환경은 VNC[1], RDP[2] 등의 프로토콜을 통한 원격 접속을 지원하고 있다. 이러한 프로토콜을 통한 접속은 디바이스를 제공하는 측의 변경을 요구하기 때문에, 다양한 디바이스를 지원하기 위한 확장성이 떨어지게 된다. 본 논문에서는 디바이스 드라이버를 이용해 원격지 자원에 대한 접근을 지원하는 프레임워크를 제안하고 있다. 이를 통해 응용 프로그램은 별도의 추가 설치 또는 변경 없이 원격 자원 입출력이 가능해 진다

1. 서론

컴퓨터 장치와 네트워크 환경이 고성능화 되면서, 컴퓨팅 자원의 효율적인 관리 및 배분을 지원하는, Xen[3], VMWare[4] 등의 가상화 기술들이 대두되고 있다. 이러한 가상화 기술을 통해 서버 환경에서는 하나의 물리서버에 복수개의 가상 서버를 운영하여 자원을 효율적으로 사용하고 있으며, 개인 사용자들이 쓸 수 있는 작업 공간인 데스크탑 환경도 가상으로 지원하고 있다.

일반적으로 가상 데스크탑을 사용하기 위해서는 원격지에서 VNC[1], RDP[2] 와 같은 프로토콜을 사용하여 접속하게 된다. 이러한 프로토콜은 운영체제 또는 응용프로그램 내에 설치가 되어(Qemu-dm[5]) 해당 프로토콜이 사용하는 시스템 자원을 중간에서 가로채서 원격지로 보내는 작업을 수행한다.

이러한 프로토콜은 보통 Keyboard / Mouse / Video / Audio 등의 제한적인 자원의 원격화를 지원하며 Legacy 인터페이스를 통한 지원이 아닌 프로토콜 자체적인 인터페이스를 통한 지원이 이루어 지고 있기 때문에 다양한 응용 프로그램의 적용 이나 디바이스의 확장이 용이하지 않게 된다.

본 논문에서는 원격 자원의 지원을 디바이스 드라이버 수준에서 제공함으로써, 응용 프로그램의 수정 없이 다양한 디바이스의 원격 지원을 가능하게 하며, 또한 디바이스 드라이버라는 일관된 수준의 드라이버 원격 지원 계층을 두어 다른 디바이스로의 확장을 쉽게 이루어 질 수 있으며, 또한 개별 디바이스의 세밀한 조정이 가능해 지는 장점을 갖는다.

본 논문은 2 장에서 배경지식을 설명하며, 3 장에서 본 논문에서 제안하고 있는 VIF Driver 에 대한 설계 및 구현 내용을 설명하며, 4 장은 간단한 실험 결과를 나타내고, 5 장에서 향후 연구 방향 및 결론을 내는 것으로 구성되어 있다.

2. 배경 지식

본 논문에서는 Xen[3] 가상화 환경을 기반으로 원격 디바이스 드라이버를 구현하고 있다. 본 절에서는 제시한 프레임워크를 지원하기 위해 필요한 배경지식인 리눅스 디바이스 드라이버와, Xen 가상화 환경을 설명한다.

2.1 리눅스 디바이스 드라이버

2.1.1 디바이스 파일

리눅스에서는 디바이스에 대한 접근을 파일을 통해서 하는데 하드웨어 처리가 필요한 응용 프로그램이 open() / read() / write() 등의 파일 입출력 함수를 이용해 디바이스 파일에 제어 및 입출력을 하면 해당 디바이스 파일과 연동된 실제 디바이스 드라이버가 해당 동작을 받아 하드웨어를 제어한다.

별도의 함수가 아닌 디바이스 파일만 다루는 방법은 하드웨어 관련 함수를 따로 정의할 필요 없이 파일 처리만으로 다양한 하드웨어를 다룰 수 있기 때문에 사용하기가 편리하며, 드라이버 구현 역시 표준화된 입출력에 대응하는 드라이버 함수들만 구현 하면 되기 때문에 통일된 방식으로 접근이 가능하다.

디바이스 드라이버는 문자 디바이스 드라이버, 블록 디바이스 드라이버, 네트워크 디바이스 드라이버 등으로 구별 되는데, 키보드, 마우스와 같이 바이트의 스트림으로 접근이 되는 디바이스를 처리하기 위해서는 문자 디바이스 드라이버를 이용한다.[6]

디바이스 파일은 “/dev” 디렉토리에 위치하며, 각 디바이스 파일은 디바이스를 관리하기 위해 주 번호와 부 번호로 관리되고 있다.

2.2.2 file operation

문자 디바이스 드라이버는 해당 디바이스가 표준 파일 입출력을 통해 처리가 되는 세부 file operation 을 구현함으로써, 디바이스의 입출력 및 제어 부를 처리할 수 있다. 문자 디바이스 드라이버에서 일반적으로 구현해야 할 주요 file operation 은 다음 <표 1> 과 같다.

<표 1 – File operation>

operation	의미
open()	응용프로그램에서 디바이스드라이버를 처음 사용하는 경우 처리
read()	디바이스에서 데이터 읽기를 수행
write()	디바이스에게 데이터 쓰기를 수행
ioctl()	Read/write 가 아닌 다양한 디바이스 드라이버 입출력 컨트롤 처리
release()	응용프로그램이 디바이스드라이버를 모두 사용하고 종료 처리
llseek()	디바이스 드라이버의 파일 포인터 위치 이동 처리
aio_read()	디바이스에서 비동기 읽기를 수행
aio_write()	디바이스에서 비동기 쓰기를 수행
poll()	입출력 다중화를 지원

2.2 Xen[3]

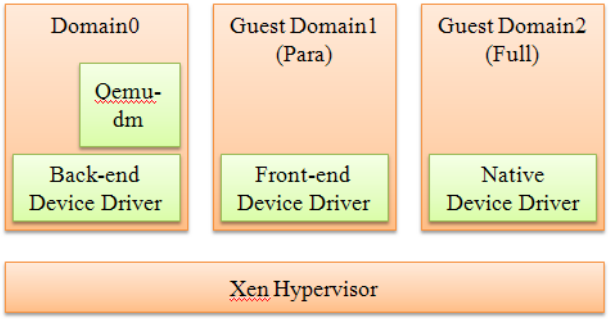
Xen[3] 은 오픈소스 기반 하드웨어 가상화 소프트웨어로 현재 3.4 버전까지 릴리즈 되어 있으며, Para-Virtualization 과 Full-Virtualization 을 동시에 지원하고 있다.

아래 <그림 1> 에 Xen 의 개략도가 나타나 있다. Xen Hypervisor 위에 각 도메인들이 구동되며, Domain0 는 각 Guest Domain 을 관리하며 실제 디바이스에 대한 접근 권한을 가지게 된다.

2.2.1 Para-Virtualization

Para-Virtualization 은 Guest OS 의 일부를 수정하여 가상화 하는 기술을 일컫는데, 위 <그림 1>의 Guest Domain 1 이 Para Virtualization 을 나타낸 그림으로 XenLinux 가 동작된다.

XenLinux 는 Linux 를 수정한 버전으로, 다른 GuestOS 를 변경시키거나 시스템 권한을 가질 수 있는 Privileged level 을 가진 명령어들을 Hypercall interface 로 대치 시킨 후 빌드된다. 커널 일부가 수정이 되며 Userlevel Linux legacy 응용 프로그램은 수정 없이 동작이 된다. 수행 중 Privileged level 을 가진 명령어들의 호출이 발생하면 GuestOS 에서 수행되지 않고 미리 정의된 Hypercall 을 통해 Hypervisor 에서 수행이 이루어 진다. 디바이스 드라이버는 Front-end Device driver 를 가지는데 이는 Domain0 측 Back-end Device driver 와 통신을 하여 실제 디바이스 제어는 domain0 에서 이루어 지게 된다.



<그림 1 – Xen Architecture>

2.2.2 Full-Virtualization

Guest Domain 2 는 Full-virtualization 으로 Guest OS 의 변경이 필요하지 않게 된다. 이 기법은 하드웨어의 도움을 받아서 구현되는데, Privilege level 명령어군들이 수행이 될 경우 CPU 에서 직접 Trap 을 일으켜서 해당 명령어의 수행을 Guest Domain 에서 직접 일어나지 않게 처리 해준다. Intel 의 VT-x 기술[7]이나 AMD 의 SVM 기술[8]을 지원하는 CPU 이용해서 해당 부분의 처리가 가능하다.(HVM : Hardware Virtual Machine)

Full-virtualization 의 경우 GuestOS 의 변경이 없게 되므로, GuestOS 에서는 Native Device Driver 가 동작이 되게 된다. 이때 GuestOS 에게 보여지는 Device 는 Qemu–dm [5] 이라는 디바이스 모델이 에뮬레이션 해주고 있는 가상 디바이스가 된다. Native Device Driver 는 이 가상 디바이스를 제어하게 된다.

2.2.3 Qemu–dm[5]

Qemu-dm[5]은 에뮬레이션 기반 가상화 소프트웨어인 Qemu[9]를 기반으로 Xen 의 Full-Virtualization 을 지원하기 위해 작성된 응용 프로그램이며, Domain0 또는 별도의 Driver Domain 에서 동작이 된다.

Qemu 는 CPU, Memory, Device 등 가상 플랫폼을 구성하는 모든 구성요소들의 에뮬레이션을 지원한다. Xen 환경의 경우 CPU 와 Memory 는 Xen hypervisor 를 통해 지원이 되고 있기 때문에 Qemu-dm[5] 은 Qemu 의 Device 가상화 부분만으로 구성이 되고 있다.

이때 Qemu-dm[5] 내부의 비디오 버퍼와 키보드, 마우스 입력부분을 통해 사용자는 해당 GuestOS 에 접근 할 수 있게 되는데 이는 크게 두 가지 방법을 통해 이루어진다.

- VNC[1] 지원 : 원격지 접속을 위해서 VNC Server 를 내포하고 있다. 즉, Qemu-dm 을 VNC[1] Server 화 시켜서, 실제 Xen HVM 의 접속을 VNC 를 통해서 이루어질 수 있도록 지원하고 있다.

- SDL[10] 지원 : 원격지 접속이 아닌 로컬 접속을 위해서 SDL 출력을 지원하고 있다. 이를 통해 로컬 사용자들은 VNC 보다 더 나은 성능의 Guest OS 접근을 할 수 있다.

3. 설계 및 구현

본 논문에서는 Xen 가상화 기술 기반으로 이루어진 Full-Virtualization Guest OS 의 원격 자원 지원을 프로토타이핑 하고 있다. 2 절에서 설명한 것과 같이 Xen 의 Full-Virtualization 은 qemu-dm 을 통해 자원 가상화가 지원 되고 있기 때문에, GuestOS 가 원격지의 자원을 접근하기 위해서는 qemu-dm 의 가상 디바이스 접근 코드가 수정되어야 한다. Qemu-dm 은 domain 0 에서 동작 되는 응용 프로그램이기 때문에, 본 절에서는 응용 프로그램이 원격 디바이스에 직접 접근할 수 있는 방법에 대한 설계 및 구현을 설명한다.

3.1 디바이스 서버

디바이스 서버는 디바이스 클라이언트에게 실제 디바이스 정보를 제공하는 측으로, 디바이스 클라이언트와 통신하는 VIF 와 이를 동적으로 관리하는 VIF-manager 로 구성되어 있다.

3.1.1 VIF-manager

VIF-manager 는 현재 디바이스 서버에서 가용한 디바이스 리스트를 디바이스 클라이언트에게 알려주는 기능을 수행하며, 디바이스 종류별 VIF 의 생성을 담당 한다.

현재 VIF-manager 에서 지원되는 디바이스는 Keyboard, Mouse, Video, Audio 장치이며, Keyboard, Mouse, Video 와 같이 Hot-plug 가 가능한 장치는 디바이스의 실제 물리적인 연결이 이루어 지면 가용한 디바이스로 등록이 된다.

Keyboard, Mouse 등 USB 디바이스의 경우 물리적인 연결탐지는 Linux HALd 와 dbus[11] 를 통해 이루어 지며, Video 디바이스의 경우는 VESA DDC interface[12] 를 통해 이루어 진다.

3.1.2 VIF

VIF 는 디바이스 클라이언트의 VIF-driver 와 TCP/IP 통신을 통해 실제 디바이스 입출력 및 제어를 처리해 주는 모듈로, 클라이언트의 연결 요청에 의해 동적으로 생성/삭제 된다.

아래 <그림 3>에 도식화 되어 있는 것 처럼 VIF 는 User-level 로 작성되어 있는데 이는 커널 드라이버수준에서 제공하는 디바이스 드라이버의 Multiplexing 기능을 그대로 승계 받을 수 있으며, 또한 User-level 로 구현되어 있기 때문에 네트워크 에러 등 문제상황 발생시에도 디바이스 서버 시스템에는 영향을 미치지 않는다. 디바이스 서버는 특성상 복수개의 디바이스 클라이언트에게 디바이스를 서비스 할 수 있기 때문에 상호 독립적인 구성이 중요하게 된다.

VIF 는 VIF-driver 로부터 넘겨받은 메시지를 해석하여 디바이스 서버의 해당 디바이스 드라이버로 값을 넘겨주고 결과값을 다시 VIF-driver 로 전송하는 기능을 담당한다. 이때, call by reference 에 의한 데이터 전송이 일어난 경우, 해당 데이터 모두를 VIF-driver 으로 전송한다.

3.2 디바이스 클라이언트

디바이스 클라이언트는 원격지의 디바이스를 직접 사용하는 측으로, 디바이스 서버와 연결을 통해 응용 프로그램에게 원격지 디바이스 입출력 및 제어를 가능하게 해주는 인터페이스를 제공한다.

디바이스 클라이언트는 서버와의 연결 설정 및 세션 관리를 담당하는 VIF-Controller 와, 응용 프로그램에게 보여지는 디바이스 드라이버인 VIF-driver 로 구성되어 있다.

3.2.1 VIF-controller

VIF-controller 는 디바이스 서버 측 VIF-manager 와 연결을 해서 VIF-driver 를 생성 시키는 기능을 수행한다. 디바이스 서버에 새로운 가용 디바이스가 등록이 되면. VIF-manager 는 가용 디바이스 정보를 VIF-Controller 에게 전송하게 되는데, 이때 사용자가 특정 디바이스를 사용한다면, VIF-Controller 는 응용 프로그램이 해당 디바이스를 접근할 수 있는 VIF-driver 를 생성한다.

VIF-driver 가 생성이 되면 VIF-driver 는 접속 포트 번호를 VIF-controller 에게 알려주게 되고, 이 포트 번호는 디바이스 서버 측 VIF 에게 전달이 된다.

VIF 는 이 포트 번호를 통해 VIF-driver 와 연결을 맺게 되고 디바이스 제어 및 입출력 기능을 수행한다.

3.2.2 VIF-driver

VIF-driver 는 해당 디바이스 드라이버가 지원하는 모든 file operation 의 인터페이스를 구현해 두어야 한다.

응용 프로그램이 open() 요청 시 실제 VIF, VIF-driver 간 세션이 연결되지 않았으면 error 메시지를 반환 시킨다.

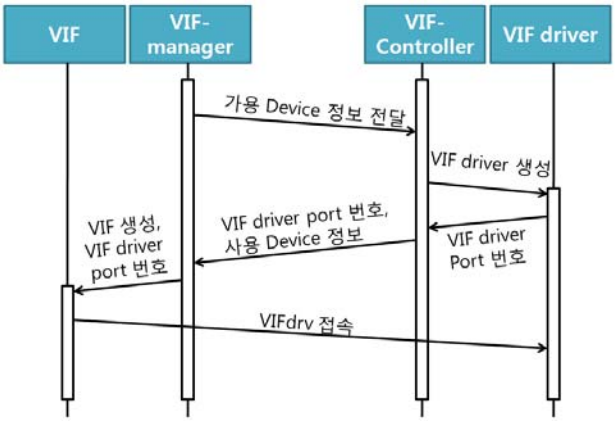
file operation 인터페이스는 디바이스 클라이언트 측 VIF 가 해당 operation 을 똑같이 재 구성해서 수행 할

수 있도록 호출 타입(open, read, write 등)과 User level 로부터 넘겨받은 모든 인자를 VIF 로 전송한다. 포인터의 경우 실제 데이터를 User level 로부터 Kernel level 버퍼로 복사 해온 후 데이터도 함께 VIF 로 전송한다.

3.3 디바이스 서버-클라이언트 세션 연결

VIF, VIF-manager 로 나타나고 있는 디바이스 서버와 VIF-controller, VIF-driver 로 나타나고 있는 디바이스 클라이언트의 세션 연결 과정이 아래 <그림 2> 에 나타나 있다.

1. VIF-manager 는 연결된 디바이스 정보를 VIF-controller 에게 보낸다. 이때 보내는 디바이스 정보는 Vendor ID, Product ID, device type 으로 구성된다.
2. VIF-manager 로부터 가용 device 정보를 전달받은 VIF-controller 는 사용자에게 가용한 정보를 전달하고, 사용자는 특정 디바이스를 선택하게 된다.
3. VIF-controller 는 사용자가 선택한 디바이스 타입과 동일한 타입의 VIF-driver 를 생성하고 이때 VIF-driver 가 생성되면서 Kernel-level socket 을 생성 시키고 포트번호를 반환한다.
4. VIF-controller 는 해당 포트 번호와 사용할 디바이스정보 (Product ID 와 Vendor ID, device type)를 디바이스 서버 측 VIF-manger 로 전송하게 되고, VIF-manager 는 디바이스 클라이언트가 요청한 디바이스를 서비스할 VIF 를 생성 한다.
5. VIF 가 생성되면서 연결될 VIF-driver 의 포트번호로 접속하면서 세션이 연결된다.



<그림 2 - VIF,VIF-driver 세션 연결과정>

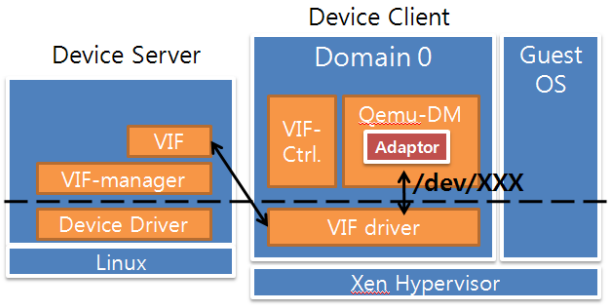
3.4 Qemu-Adaptor

VIF 와 VIF-driver 의 연결을 통해 응용 프로그램은 Legacy interface 인 디바이스 파일을 통해 원격 디바이스를 접근할 수 있다.

Xen 가상화 환경에서 가상 데스크탑 접속을 기존의 VNC, RDP 와 같은 방법이 아닌, VIF-driver 를 통해서 진행 할 수 있게 되는데, 이 기법은 Local 에서 사용할 수 있는 모든 디바이스의 원격화가 가능해 지기 때문에 확장성이 우월하며, 실제 데이터 패킷 크기의 차이도 크게 나지 않기 때문에 네트워크 오버헤드로 인한 성능 차이도 크지 않다.

HVM Domain 은 Qemu-dm 을 통해 VNC 원격 접속을 서비스하고 있는데, VIF-Driver 를 이용한 원격 접속을 지원하기 위해 해당 부분을 수정하였다.

Qemu-dm 내에 기존의 VNC 로 넘어가는 디바이스 데이터(Keyboard, Mouse, Video)를 Local 에서 접속하는 것처럼, 미리 정의된 디바이스 파일로 접근하는 모듈이 추가가 되는데 이 모듈이 Qemu-Adaptor 이다.(<그림 3> 참조) Audio 디바이스의 경우 Qemu-dm 에서는 기본적으로 로컬의 OSS[13]를 지원하고 있다. 이때 로컬 OSS 의 디바이스 파일을(/dev/dsp) VIF-driver 로 등록해 원격지의 오디오 디바이스 접근을 가능하게 한다.



<그림 3 VIF 적용 Xen Architecture>

3.5 디바이스 별 고려사항

3.5.1 VIF-Keyboard / VIF-Mouse

Keyboard, Mouse 는 입력 장치로써, VIF 에서 VIF-driver 측으로의 데이터 이동이 주가 된다. 본 프레임워크의 프로토타입으로 구현된 Keyboard, Mouse 는 VIF 가 Qemu-adaptor 에서 사용하는 자료구조로 패킷을 생성하여 VIF-driver 로 보내준다. 이 자료구조는 VNC 프로토콜의 Keyboard / Mouse 자료구조와 동일하며, 키 코드 값, 버튼 push 상태, wheel 키 상태 등을 전송한다.

3.5.2 VIF-Video

Video 의 경우는 출력 장치로써, VIF-driver 에서 데이터를 생성하여 VIF 로 전송하게 된다. 이 때 전송해야 할 패킷은 매 화면의 픽셀 데이터로 구성되어 있어서 다른 디바이스가 생성하는 패킷 보다 상대적으로 크기가 커지게 된다. 이러한 대용량 데이터 전송 방법에 두 가지 디자인이 고려될 수 있다.

1. 데이터를 동기적으로 매번 VIF 측으로 전송하는 방법

2. 데이터를 VIF-driver 가 내부에 버퍼링 하면서 비동기적으로 VIF 측으로 전송하는 방법

1 번 방법의 경우 User-interact 한 결과를 얻을 수 있지만, 데이터 전송 측에서는 약간의 지연이 발생하게 되고, 2 번 방법의 경우는 데이터 전송 측에서는 응답속도가 좋으나, 데이터 수신 측에서의 응답속도의 약간의 지연이 발생하게 된다.

본 논문에서의 프로토타입으로는 User-interact 한 방법인 1 번 방법으로 구현하였다.

3.5.3 VIF-Audio

Audio 의 경우는 입출력이 모두 일어나는 장치로써, VIF-driver 에서 VIF 로의 Audio-out 전송이, VIF 에서 VIF-driver 로의 Mic-in 전송이 일어나게 된다. Audio device 의 경우도 3.5.2 절의 Video 장치와 마찬가지로, 데이터 전송방법의 두 가지 디자인이 모두 고려될 수 있는데, Qemu-dm 이 생성시키는 Audio packet 은 크기가 작으면서, 매우 빈번히 일어나는 연산이기 때문에 Video 처럼 매번 동기적으로 전송을 하게 되면 Network 전송 오버헤드가 크게 발생하여, User-interaction 을 떨어뜨리는 결과를 보인다.

본 논문에서의 Audio VIF-driver 프로토타입은 내부 버퍼링을 통한 비동기적 전송 방법으로 구현하였다.

4. 평가

본 논문에서 구현된 VIF-VIFdriver 를 테스트한 환경은 다음과 같다.

- 디바이스 서버 : VIA C7 1GHz 프로세서, 1GB 메모리
  - Linux Kernel 2.6.26
- 디바이스 클라이언트 : Intel Dual Core CPU E6300 2.80GHz 프로세서, 4GB 메모리,
  - Hypervisor : Xen 3.1 Official distribution,
  - Domain 0 : XenLinux 2.6.18.8
  - HVM Domain : Windows XP SP2

4.1 디바이스 서버

디바이스 서버의 장치 인식 시간은 실제 디바이스가 디바이스 서버에 물리적으로 연결되어, VIF-manager 에 가용한 디바이스로 인식 되는데 까지 걸리는 시간을 의미한다. 본 논문의 프로토타입에서는 USB-타입 디바이스 인식을 위해 Linux Hald 와 Dbus 를 사용하였고, 약 1.7 초 정도의 인식 시간이 소요되는 것을 확인했다.

4.2 이벤트 응답시간

디바이스 클라이언트 측 VM (Windows XP) 에서 키보드와 마우스가 연결된 후 사용자가 실제 디바이스 서버 측에 연결되어 있는 키보드, 마우스에 이벤트를 입력하게 되면, 이 이벤트는 디바이스 클라이언트 측 VIF-driver 로 전송되고, 해당 이벤트가 Qemu-dm 을 거쳐 VM 으로 전송되어 실제 유저의 입력이 반응하게 된다. 이 때의 키보드, 마우스의 응답시간을 비교하면, 각각 31ms, 20ms 정도가 소요된다. Qemu-dm 에서 내장하고 있는 VNC 의 경우 키보드, 마우스의 경우는 각각 21ms, 14ms 정도 소요되는데, 일반적으로 알려져 있는 키보드, 마우스 액션의 수용 가능한 응답시간은 50~150ms 정도 인 것을 감안하면[14], VNC 보다 응답시간이 늦지만, 충분히 수용 가능한 범위 내에 있음을 확인 할 수 있다.

5. 향후 연구방향 및 결론

본 논문에서는 디바이스 드라이버 기반의 원격 디바이스 접근 프레임워크를 제시하여, 이를 Xen 가상화 환경에서 원격 데스크탑을 접속하는 방법으로 제시하고 있다.

기존의 기법들과 달리 디바이스 드라이버 수준에서 직접 원격지 디바이스와 연결이 되므로, 응용 프로그램들의 수정이 없이 그대로 원격지의 디바이스를 접근 할 수 있는 장점이 생긴다. 또한, 기존의 데스크탑 접속 솔루션과 달리 다양한 디바이스 지원을 쉽게 구현할 수 있기 때문에 확장성의 측면에서도 장점이 생긴다.

6. Acknowledgement

본 연구는 지식경제부 및 정보통신 연구진흥원의 IT R&D 프로그램의 연구 결과로 수행되었음 [2008-S034-01, Development of Collaborative Virtual Machine Technology for SoD (System on-Demand) Service]

참고문헌

[1] Tight VNC, http://www.tightvnc.com.

[2] Microsoft, Remote Desktop Protocol, RDP, http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield, “Xen and the art of virtualization”, "Symposium on Operating Systems Principles - SOSP", 2003

[4] VMWare, ESX server, http://www.vmware.com/

[5] Anthony Liguori ,“Merging QEMU-DM Upstream”, XenSummit Spring 2007

[6] Janathan Corbet, et. al., Linux Device Driver 3rd Ed. O’REILLY, pp.5-8, 2005

[7] Intel, VT-x http://www.intel.com/technology/virtualization/

[8] AMD, SVM
http://www.amd.com/virtualization/

[9] Fabrice Bellard, “QEMU, a Fast and Portable Dynamic Translator”, Usenix Annual Technical Conference, 2005

[10] Simple DirectMedia Layer, SDL
http://www.libsdl.org

[11] HALd, Hal daemon,
http://www.freedesktop.org/wiki/Software/hal
Dbus library,
http://www.freedesktop.org/wiki/Software/dbus

[12] Display data channel (DDC), http://www.vesa.org

[13] Open Sound System, OSS,
http://www.opensound.com/

[14] Shneiderman, B., Designing the User Interface: Strategies for Effective Human-Computer Interaction, 3rd ed., Addison-Wesley, Reading, MA, 1998.