



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2008-0103725
(43) 공개일자 2008년11월28일

(51) Int. Cl.

G06F 3/14 (2006.01) G06F 9/06 (2006.01)
G06F 3/033 (2006.01)

(21) 출원번호 10-2007-0050816

(22) 출원일자 2007년05월25일

심사청구일자 2007년05월25일

(71) 출원인

한림대학교 산학협력단

강원 춘천시 한림대학길 39

(72) 발명자

김용범

서울 송파구 문정동 115-11 대영빌라 가동 202호

심민섭

서울 노원구 월계2동 롯데캐슬 120동 103호

(뒷면에 계속)

(74) 대리인

박준석

전체 청구항 수 : 총 6 항

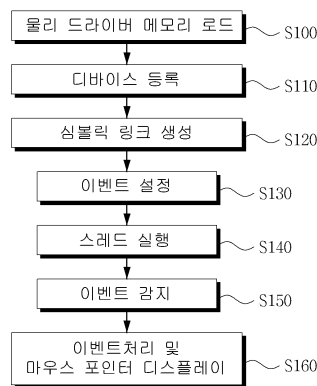
(54) 멀티 마우스 포인터 디스플레이 방법

(57) 요약

본 발명은 멀티 마우스 포인터 디스플레이 방법에 관한 것으로서, 보다 구체적으로는 논리 드라이버를 사용하여 1개 이상의 추가된 마우스의 포인터를 생성하고 이를 디스플레이 함으로써 2개 이상의 마우스를 동작시키는 방법에 관한 것이다.

본 발명에 따른 멀티 마우스 포인터 디스플레이 방법은 컴퓨터 시스템의 물리 드라이버의 메모리를 로딩(loading)하는 단계, 상기 로딩한 물리 드라이버의 메모리를 이용하여 추가된 마우스 디바이스를 등록하는 단계, 상기 마우스 디바이스에 의하여 발생하는 이벤트를 설정하는 단계, 상기 마우스 디바이스에 대한 스레드(thread)를 실행하는 단계, 멀티 마우스가 적용되는 어플리케이션을 구동하고, 상기 구동된 어플리케이션 상에서 발생한 이벤트를 감지하되, 상기 감지되는 이벤트는 상기 설정된 이벤트인 것인 이벤트 감지 단계 및 상기 감지된 이벤트에 기반하여 전체 윈도우 사이즈와 상기 멀티 마우스가 적용되는 어플리케이션 윈도우 사이즈를 동적으로 계산하여 마우스 커서를 이미지로 출력하여 마우스 포인터를 디스플레이하는 단계를 포함하는 멀티 마우스 포인터 디스플레이 방법을 제공한다.

대표도 - 도2



(72) 발명자

송창근

강원 춘천시 퇴계동 그린타운아파트 108동 501호

김유섭

서울 강남구 청담2동 삼성1차아파트 101동 811호

특허청구의 범위

청구항 1

멀티 마우스 포인터 디스플레이 방법에 있어서,

컴퓨터 시스템의 물리 드라이버의 메모리를 로딩(loading)하는 단계,

상기 로딩한 물리 드라이버의 메모리를 이용하여 추가된 마우스 디바이스를 등록하는 단계,

상기 마우스 디바이스에 의하여 발생하는 이벤트를 설정하는 단계,

상기 마우스 디바이스에 대한 스레드(thread)를 실행하는 단계,

멀티 마우스가 적용되는 어플리케이션을 구동하고, 상기 구동된 어플리케이션 상에서 발생한 이벤트를 감지하되, 상기 감지되는 이벤트는 상기 설정된 이벤트인 것인 이벤트 감지 단계 및

상기 감지된 이벤트에 기반하여 전체 윈도우 사이즈와 상기 멀티 마우스가 적용되는 어플리케이션 윈도우 사이즈를 동적으로 계산하여 마우스 커서를 이미지로 출력하여 마우스 포인터를 디스플레이하는 단계

를 포함하는 멀티 마우스 포인터 디스플레이 방법.

청구항 2

제1항에 있어서,

상기 이벤트 감지는 컴퓨터 시스템의 운영 체제(operating system, OS)가 상기 등록된 마우스 디바이스에 대한 이벤트를 감지하는 경우 이를 후킹(hooking)함으로써 수행되는 멀티 마우스 포인터 디스플레이 방법.

청구항 3

제1항에 있어서,

상기 등록된 마우스 디바이스와 상기 마우스 디바이스에 대하여 사용자가 인식 가능한 이름(user visible name)을 연결하는 심볼릭 링크를 생성하는 단계

를 더 포함하는 멀티 마우스 포인터 디스플레이 방법.

청구항 4

제1항에 있어서,

상기 설정된 이벤트는 상기 마우스 디바이스의 왼쪽 마우스 버튼 다운, 왼쪽 마우스 버튼 업, 오른쪽 마우스 버튼 다운, 오른쪽 마우스 버튼 업, 중축 마우스 이동, 횡축 마우스 이동, 마우스 휠 다운, 마우스 휠 업, 마우스 휠 스크롤 다운 및 마우스 휠 스크롤 업 중 하나 이상을 포함하는 것인 멀티 마우스 포인터 디스플레이 방법.

청구항 5

제1항 내지 제4항 중 어느 한 항에 있어서,

상기 마우스 디바이스 등록 단계는

상기 마우스 디바이스를 위한 필터 디바이스 객체를 생성하는 단계 및

상기 생성된 필터 디바이스 객체를 디바이스 스택의 최상단에 위치시키고 마우스 디바이스 객체의 포인터를 획득하는 단계

를 포함하는 것을 특징으로 하는 멀티 마우스 포인터 디스플레이 방법.

청구항 6

멀티 마우스의 포인터를 생성하고 동작시키는 프로그램이 기록된 컴퓨터로 판독 가능한 기록 매체에 있어서,

제1항 내지 제4항 중 어느 하나의 항에 기재된 각각의 단계를 수행하는 프로그램이 기록된 컴퓨터 판독 가능 기록 매체.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

- <6> 본 발명은 멀티 마우스 포인터 디스플레이 방법에 관한 것으로서, 보다 구체적으로는 논리 드라이버를 사용하여 1개 이상의 추가된 마우스 포인터를 생성하고 이를 디스플레이 함으로써 2개 이상의 마우스를 동작시키는 방법에 관한 것이다.
- <7> 현재 컴퓨터의 보급으로 인하여 컴퓨터는 생활 필수품이 되었다. 이러한 컴퓨터를 작동시키고 운용하기 위한 운영 체제(operating system, OS)의 사용을 위하여, 마우스는 키보드와 더불어 필수적인 입력 장치로 사용되고 있다.
- <8> 현재 사용되는 마우스는 주로 볼 마우스 또는 광 마우스이며, 최근 광 마우스의 보급이 활발하게 진행되고 있다. 마우스는 마우스에 장착된 볼 또는 광 센서를 통하여 마우스의 움직임을 감지하고 이를 모니터 상에서 생성된 포인터를 통하여 나타냄으로써 사용된다. 이러한 마우스의 포인터는 마우스의 횡축 및 종축에 대한 움직임에 응답하여 각각의 움직임에 대하여 모니터 상에서 마우스 포인터를 연속적으로 디스플레이 함으로써 구현된다.
- <9> 일반적으로, 이러한 필수적 입력장치로 사용되고 있는 마우스는 오직 하나만이 사용되고 있다. 다시 말해, 컴퓨터에 2개 이상의 마우스를 연결하여도 컴퓨터 운영 체제(operating system, OS)는 하나의 마우스 포인터만을 모니터 상에 생성하고 컴퓨터에 연결된 2개 이상의 마우스의 움직임을 하나의 포인터로서만 구현한다. 따라서 2개의 마우스를 사용하여 동시에 두 개의 포인터를 동작시키는 것은 현재 OS에서는 구현되어 있지 않다.
- <10> 그러나 컴퓨터의 다양한 활용, 예를 들어, 의료에서의 수술 시뮬레이션, 스포츠 게임 또는 유도식 유아 교육 등에서의 활용을 위하여 2개 이상의 독립된 포인터를 이용할 필요가 있다.

발명이 이루고자 하는 기술적 과제

- <11> 본 발명은 상술한 문제점을 해결하기 위한 것으로서, 2개 이상의 마우스를 동시에 사용하기 위하여 모니터 상에서 각각의 마우스에 대한 포인터를 생성하고 이를 조작하는 방법을 제공하는 것을 목적으로 한다.

발명의 구성 및 작용

- <12> 상기와 같은 목적을 달성하기 위해, 본 발명의 일 실시예는 멀티 마우스 포인터 디스플레이 방법에 있어서, 컴퓨터 시스템의 물리 드라이버의 메모리를 로딩(loading)하는 단계, 상기 로딩한 물리 드라이버의 메모리를 이용하여 추가된 마우스 디바이스를 등록하는 단계, 상기 마우스 디바이스에 의하여 발생하는 이벤트를 설정하는 단계, 상기 마우스 디바이스에 대한 스레드(thread)를 실행하는 단계, 멀티 마우스가 적용되는 어플리케이션을 구동하고, 상기 구동된 어플리케이션 상에서 발생한 이벤트를 감지하되, 상기 감지되는 이벤트는 상기 설정된 이벤트인 것인 이벤트 감지 단계 및 상기 감지된 이벤트에 기반하여 전체 윈도우 사이즈와 상기 멀티 마우스가 적용되는 어플리케이션 윈도우 사이즈를 동적으로 계산하여 마우스 커서를 이미지로 출력하여 마우스 포인터를 디스플레이하는 단계를 포함하는 멀티 마우스 포인터 디스플레이 방법을 제공한다.
- <13> 또한, 본 발명의 일 실시예에서 상기 이벤트 감지는 컴퓨터 시스템의 운영 체제(operating system, OS)가 상기 등록된 마우스 디바이스에 대한 이벤트를 감지하는 경우 이를 후킹(hooking)함으로써 수행될 수 있다.
- <14> 또한, 본 발명의 일 실시예에서 상기 마우스 디바이스 등록 단계는 상기 마우스 디바이스를 위한 필터 디바이스 객체를 생성하는 단계 및 상기 생성된 필터 디바이스 객체를 디바이스 스택의 최상단에 위치시키고 마우스 디바이스 객체의 포인터를 획득하는 단계를 포함할 수 있다.
- <15> 또한, 본 발명의 일 실시예는 등록된 마우스 디바이스와 상기 마우스 디바이스에 대하여 사용자가 인식 가능한 이름(user visible name)을 연결하는 심볼릭 링크를 생성하는 단계를 더 포함할 수 있다.
- <16> 또한, 설정된 이벤트는 상기 마우스 디바이스의 왼쪽 마우스 버튼 다운, 왼쪽 마우스 버튼 업, 오른쪽 마우스 버튼 다운, 오른쪽 마우스 버튼 업, 종축 마우스 이동, 횡축 마우스 이동, 마우스 휠 다운, 마우스 휠 업, 마우스

스 휠 스크롤 다운 및 마우스 휠 스크롤 업 중 하나 이상을 포함할 수 있다.

- <17> 아래에서는 첨부한 도면을 참고로 하여 본 발명의 실시예에 대하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.
- <18> 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한, 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미한다.
- <19> 도 1은 본 발명의 일 실시예에 따른 필터 드라이버의 동작을 설명한 도면이다.
- <20> 필터 드라이버(Filter Driver)는 일종의 인터미디에트 드라이버(intermediate driver)로서, 파일 시스템 드라이버나 디스크 드라이버 등과 같이 이미 상용화되어 있는 드라이버에 전달되는 I/O 요청을 가로채어 기존의 드라이버가 제공하는 기능을 보완하거나 새로운 기능을 추가할 수 있는 기회를 제공한다. 예를 들어, USB 포트에 전달되는 데이터를 모니터링해야 한다면 운영체제에서 USB 드라이버를 제공하기 때문에 USB 필터 드라이버를 제작해야 한다. 개인용 컴퓨터에 설치된 각 포트나, 특정 작업을 모니터링하기 위해 사용하는 툴들은 대부분 필터 드라이버를 이용하여 제작한다.
- <21> 필터 드라이버를 이용한 다른 예로서 파일 시스템 필터 드라이버를 들 수 있다. 파일 시스템 드라이버(FAT, NTFS, CDFS, LAN 관리자 리더렉터 등)는 파일을 디스크에 저장하거나 읽을 때 파일에 대한 암호화 또는 복호화 기능을 제공하지 않는다. 따라서 파일에 대한 암호화 또는 복호화 기능을 추가하거나 바이러스 검사와 같은 기능을 수행하기 위하여 파일 시스템 필터 드라이버를 사용할 수 있다.
- <22> 필터 드라이버는 드라이버 계층에서 드라이버에 전달되는 I/O 요청을 어디서 가로채느냐에 따라 상위(Upper) 필터 드라이버와 하위(Lower) 필터 드라이버로 나뉜다.
- <23> 상위 필터 드라이버는 사용자 프로세스의 요청이 파일 시스템 드라이버에 도달하기 전에 I/O 요청을 가로채어 필요한 작업을 하는 드라이버이다. 하위 필터 드라이버는 파일 시스템 드라이버 아래에 위치한다. 파일 시스템 드라이버에서 처리한 I/O 요청을 하드 디스크와 같은 보조 저장 장치를 관리하는 드라이버에 전달할 때 이 I/O 요청을 가로채어 필요한 작업을 한다.
- <24> 필터 드라이버는 전체 드라이버를 다시 작성하지 않고 현재 있는 드라이버에서 몇 가지 측면에서만 동작을 수정하는 방식으로 동작한다. SCSI 필터 드라이버가 이런 방식으로 동작한다. 필터 드라이버는 하위 레벨 디바이스 드라이버가 갖는 한계를 감추어 주는 역할을 수행한다.
- <25> 예를 들어, 하위 드라이버가 데이터 전송 크기의 한계를 갖고 있다면 필터에서 큰 용량의 데이터 전송을 작은 크기로 쪼개어 하위 레벨로 전달할 수 있을 것이다. 필터 드라이버는 압축이나 암호화와 같은 새로운 특징을 하위 디바이스 드라이버의 또는 디바이스를 사용하는 프로그램의 수정 없이 디바이스에 추가시킬 수 있다. 필터 드라이버는 드라이버가 언제나 수행되지 않는 비용이 많이 드는 동작(성능 모니터링과 같은)을 추가하거나 제거시킬 수 있다.
- <26> 필터 드라이버와 다른 계층 드라이버들간의 주된 차이점은 그들이 생성한 디바이스 객체에 있다. 계층 드라이버는 디바이스 객체를 자신의 유일한 이름으로 익스포트(export)시키는 데 반해, 필터 드라이버의 디바이스 객체는 이름을 갖지 않는다. 이렇게 필터 드라이버들은 하위 레벨 드라이버에 의해 생성된 디바이스 객체에 자신의 이름 없는 디바이스 객체들 중 하나를 붙여 동작하게 된다.
- <27> FLTDRIVER는 FDO DRIVER의 디바이스 객체 중 하나인 FDO에 필터 디바이스 객체가 부착되어 있다. FDO로 보내지는 모든 IRP들은 자동적으로 FLTDRIVER 안의 디스패치 루틴으로 라우팅된다. 이와 같은 과정은 다음에 설명되어 있다.
- <28> 필터 드라이버의 AddDevice 루틴에서 보이지 않는 디바이스 객체를 생성하고, 그것을 자신의 하위 드라이버가 갖고 있는 명명된 디바이스 객체에 붙인다. 하위 레벨 드라이버에 접근하는 클라이언트는 FDO의 연결을 오픈(open)한다. 이것은 일반적으로 Win32 CreateFile 메소드(method)를 통해 핸들을 획득하는 것으로 수행되고, 커널 모드 클라이언트들은 IoGetDeviceObjectPointer 함수를 사용하여 수행할 수 있다. 이와는 상관없이, I/O 관

리자는 실질적으로 클라이언트와 필터 드라이버의 디바이스 객체간의 연결을 성립시킨다.

- <29> 클라이언트가 I/O 요청을 FD0로 보내면, I/O 관리자는 이 요청을 필터 드라이버의 이름없는 디바이스 객체로 대신 보낸다. 그리고 I/O 관리자는 필터 드라이버 객체의 MajorFunction 테이블을 사용하여 적절한 디스패치 루틴을 선택한다. 필터 드라이버 내의 디스패치 루틴들은 자신이 IRP를 직접 처리하거나 즉시 완료시킨다. 또는 IoCallDriver를 통해 아래의 FD0로 IRP를 내려 보낸다. 만약 필터 드라이버가 하위 레벨 드라이버에서 IRP에 대한 처리가 완료된 이후에 제어를 다시 얻고자 한다면, 필터에서 IRP에 I/O 완료 루틴을 지정시킨다.
- <30> 필터들은 다른 필터 위에 또 계층적으로 형성될 수 있다. 이미 필터링되는 디바이스 객체에 다시 새로운 필터를 올리면 이미 형성된 계층의 최상단에 새로운 필터가 올라가는 결과를 가져온다. 본질적으로, 단일 디바이스 객체는 몇 개의 필터라도 계층으로 형성시킬 수 있다.
- <31> 필터 드라이버는 DriverEntry라는 이름의 루틴을 익스포트(export)시켜야 한다. 이 루틴은 드라이버의 데이터 구조체를 초기화하고, 다른 모든 드라이버의 구성 요소들을 위한 환경을 준비한다. DriverEntry 루틴 상에서 밝히는 과정은 다음과 같다.
- <32> (1) DriverEntry는 드라이버에서 제어할 하드웨어를 획득한다. 그 하드웨어는 이 드라이버의 제어 하에 있다고 지정되도록 할당된다.
- <33> (2) 드라이버 객체는 드라이버 내부의 다른 함수들의 시작점을 지정함으로써 초기화된다 이러한 지정은 드라이버 객체 내에 함수들의 포인터를 저장하는 것으로 처리된다.
- <34> (3) 만약 드라이버가 멀티 유닛이나 멀티 펄스 컨트롤러를 관리한다면, IoCreateController를 사용하여 컨트롤러 객체를 생성하고, 컨트롤러 익스텐션을 초기화한다.
- <35> (4) IoCreateDevice를 사용하여 드라이버의 제어 하에 있는 각 물리적 또는 논리 디바이스에 대한 디바이스 객체를 생성하고, 디바이스 익스텐션을 초기화한다.
- <36> (5) 이렇게 생성된 디바이스 객체를 IoCreateSymbolicLink를 호출하여 Win32 서브시스템에게 보이도록 만든다.
- <37> (6) 디바이스를 인터럽트 객체에 연결한다. 만약 ISR이 DPC 객체를 요구한다면, 이 과정에서 생성하고 초기화한다.
- <38> (7) 과정 (4)에서 (6)을 이 드라이버의 제어 하에 있는 각 물리적 또는 논리 디바이스에 대해 반복 수행한다.
- <39> (8) 만약 성공하면, DriverEntry는 I/O 관리자에게 STATUS_SUCCESS를 리턴한다.
- <40> 이 경우 과정 (1) 및 (3) 내지 (6)은 WDM 드라이버의 DriverEntry 루틴에 의해 수행되지 않고, AddDevice 루틴에 의해 처리된다.
- <41> 만약 DriverEntry가 어떤 이유로 인해 초기화 단계에서 실패한다면, 할당했던 시스템 자원을 해제해야 하고, I/O 관리자에게 정확한 NTSTATUS의 실패 상황에 대한 코드를 리턴한다.
- <42> 다른 모든 커널 모드 드라이버에서처럼, 필터 드라이버는 DriverEntry로 불리는 주 진입점을 갖고 있다. 또한, 여타 WDM 드라이버와 같이, AddDevice, RemoveDevice 및 Unload 루틴을 갖고 있어야 한다.
- <43> AddDevice 루틴에서는 디바이스를 추가시키며, AddDevice 루틴에서 디바이스를 추가시키는 PnP 요청을 위해 필터 드라이버에서 수행해야 하는 초기화 순서는 간단하다.
- <44> 필터는 IoCreateDevice 함수를 호출하여 타겟 디바이스를 위한 필터 디바이스 객체를 생성한다. 필터 디바이스 객체는 내부 이름을 갖고 있지 않고, 심볼릭 링크 이름도 갖고 있지 않다.
- <45> 흔히 쓰이는 IoAttachDeviceToDeviceStack 함수를 호출하여 하위 레벨 드라이버의 상단에 자신을 쌓아 올리고 타겟 디바이스 객체의 포인터를 획득한다.
- <46> 필터 디바이스 객체의 디바이스 익스텐션 안에 타겟 디바이스 객체의 주소를 저장한다. 필터 드라이버의 다른 루틴에서 이 포인터를 사용하여 타겟 드라이버를 호출하게 된다.
- <47> 다음은, AddDevice에서 타겟 디바이스 객체로부터 필터 디바이스 객체로 DeviceType과 Characteristics 필드를 복사한다. 또한, 타겟 디바이스 객체의 Flags 필드로부터 DO_DIRECT_IO, DO_BUFFERED_IO, DO_POWER_INRUSH 그리고 DO_POWER_PAGABLE 비트를 복사한다. 이 과정을 통해 필터가 타겟 드라이버와 동일한 버퍼링 정책을 갖게 된다는 것을 뜻한다.

- <48> RemoveDevice 루틴에서는 반드시 필터와 타겟 디바이스 객체의 연결을 끊어야 한다. 이것은 IoDetachDevice를 호출하여 타겟 디바이스 객체의 포인터를 전달하는 것으로 수행한다. 필터 디바이스 객체가 연결 해제되면, RemoveDevice 루틴은 IoDeleteDevice를 호출하여 이름없는 필터 디바이스 객체를 삭제한다.
- <49> 필터 드라이버가 타겟 드라이버에 부착되면, 타겟 드라이버로 오는 모든 I/O 요청은 먼저 필터 드라이버의 디스패치 루틴을 통해 전달된다. 만약 필터 드라이버 객체의 MajorFunction 테이블이 타겟 드라이버와 동일한 IRP_MJ_XXX 코드를 지원하지 않는다면, 필터 드라이버가 부착된 이후 클라이언트에게 문제가 발생할 것이다. 엄밀히 얘기하면, 필터가 없을 때 잘 수행되던 드라이버가 필터가 붙은 이후 잘못된 동작을 하게 되는 것이다.
- <50> 이러한 문제점을 피하기 위해 필터 드라이버의 MajorFunction 테이블은 타겟 드라이버가 지원하는 모든 IRP_MJ_XXX 함수를 지원해야 하는 것이다. 또한, 필터 드라이버가 일부의 특정 MajorFunction 코드에 대해 수정을 가하길 원치 않는다고 하더라도, 필터 드라이버의 디스패치 루틴에서는 하위 드라이버로 IRP를 단순히 넘겨주기만이라도 해야 한다.
- <51> 필터 드라이버에서 이런 문제점을 해결할 수 있는 가장 간단한 방법은 필터 드라이버의 객체의 MajorFunction 테이블 내의 모든 디스패치 루틴을 pass-through 루틴(IRP를 단순히 하부로 넘겨주는 루틴)으로 제공하는 것이다. 만약 MajorFunction 엔트리 중 필터에서 수정을 가하길 원한다면, 해당 부분만 단순히 넘기지 않는 루틴으로(non-pass-through) 제공해야 할 것이다.
- <52> 도 2는 본 발명의 일 실시예에 따른 멀티 마우스 포인터를 디스플레이하는 방법을 설명하기 위한 순서도이다.
- <53> 단계(S100)에서 컴퓨터 시스템의 물리 드라이버의 메모리를 로딩(loading)한다. 물리 드라이버는 하드웨어와 컴퓨터 시스템 사이의 연결을 관리하는 드라이버이다.
- <54> 단계(S110)에서는, 단계(S100)에서 로딩한 물리 드라이버의 메모리를 이용하여 추가된 마우스 디바이스를 등록한다. 마우스 디바이스의 등록은 마우스 디바이스의 생성 및 마우스 디바이스를 디바이스 스택에 추가하는 것을 포함한다.
- <55> 단계(S120)에서는, 단계(S110)에서 등록된 마우스 디바이스와 그 마우스 디바이스에 대하여 사용자가 인식 가능한 이름(user visible name)을 연결하는 심볼릭 링크를 생성한다.
- <56> 단계(S130)에서는 단계(S110)에서 등록된 마우스 디바이스에 의하여 발생하는 이벤트를 설정한다. 본 발명의 일 실시예에서 설정되는 이벤트는, 예를 들어, 왼쪽 마우스 버튼 다운, 왼쪽 마우스 버튼 업, 오른쪽 마우스 버튼 다운, 오른쪽 마우스 버튼 업, 종축 마우스 이동, 횡축 마우스 이동, 마우스 휠 다운, 마우스 휠 업, 마우스 휠 스크롤 다운 및 마우스 휠 스크롤 업 중 하나 이상을 포함할 수 있다.
- <57> 단계(S140)에서 단계(S110)에서 등록된 마우스 디바이스에 대한 스레드(thread)를 실행하고 단계(S130)에서 설정된 이벤트를 감지할 준비를 한다. 스레드란 컴퓨터 프로그램을 수행할 때, 프로세스(process) 내부에 존재하는 수행 경로, 즉 일련의 실행 코드를 말한다. 컴퓨터 프로그램을 수행하는 경우 프로세스는 단순한 껍데기일 뿐, 실제 작업은 스레드가 담당한다. 프로세스 생성 시 하나의 주 스레드가 생성되어 대부분의 작업을 처리하고 주 스레드가 종료되면 프로세스도 종료된다. 하나의 OS에서 여러 개의 프로세스가 동시에 실행되는 환경이 멀티태스킹이고, 하나의 프로세스 내에서 다수의 스레드가 동시에 수행되는 것이 멀티스레딩이다.
- <58> 단계(S150)에서는, 멀티 마우스가 적용되는 어플리케이션을 구동하고, 구동된 어플리케이션 상에서 발생한 이벤트를 감지한다. 이 경우 감지하는 이벤트는 단계(S130)에서 설정된 이벤트이다. 이러한 이벤트 감지는 컴퓨터 시스템의 OS가 등록된 논리 드라이버의 마우스 이벤트를 감지하는 경우 이를 후킹(hooking)함으로써 수행된다.
- <59> 단계(S160)에서는, 단계(S150)에서 감지한 이벤트에 기반하여 전체 윈도우 사이즈와 멀티 마우스 디바이스가 적용되는 어플리케이션 윈도우 사이즈를 동적으로 계산하여 단계(S150)에서 구동된 어플리케이션의 윈도우에 마우스 커서를 이미지로 출력하여 마우스 포인터를 디스플레이한다.
- <60> 도 3은 본 발명의 일 실시예에 따른 멀티 마우스 포인터를 디스플레이하는 방법에 있어서 마우스 디바이스의 등록 단계의 세부 과정을 도시한 순서도이다.
- <61> 단계(S112)에서 마우스 디바이스를 위한 필터 디바이스 객체를 생성한다. 필터 디바이스 객체는 내부 이름을 갖고 있지 않고, 심볼릭 링크 이름도 갖고 있지 않다.
- <62> 단계(S114)에서는, 단계(S112)에서 생성된 필터 디바이스 객체를 디바이스 스택의 최상단에 위치시키고 마우스

디바이스 객체의 포인터를 획득한다.

- <63> 스택이란 자료 구조의 하나로써, 자료의 삽입과 삭제가 한 쪽 끝에서만 일어나는 선형 목록을 말한다. 자료의 삽입 및 삭제가 일어나는 곳은 스택의 톱(top)이고, 자료를 스택에 넣는 것을 푸시(push), 스택에서 자료를 꺼내는 것을 팝(pop)이라 한다. 스택은 나중에 들어간 자료가 먼저 꺼내는 후입선출(LIFO) 방식을 사용하며 주로 어떤 내용을 기억시켰다가 다시 이용하고자 할 때 사용된다.
- <64> 이상과 살펴본 바와 같이 본 발명의 일 실시예에서 멀티 마우스 포인터를 생성하기 위하여 기존 마우스 드라이버와 동일한 모듈을 사용하며, 다만 마우스의 고유아이디, 즉 범용 식별자(globally unique identifier, GUID)를 통하여 새로운 드라이버로 인식하도록 레지스트리에 등록한다. 새로운 드라이버의 마우스 이벤트, 예를 들어, 왼쪽 마우스 버튼 다운, 왼쪽 마우스 버튼 업, 오른쪽 마우스 버튼 다운, 오른쪽 마우스 버튼 업, 종축 마우스 이동, 횡축 마우스 이동, 마우스 휠 다운, 마우스 휠 업, 마우스 휠 스크롤 다운 및 마우스 휠 스크롤 업 등을 후킹하고 전체 윈도우 사이즈와 멀티 마우스 디바이스가 적용되는 어플리케이션 윈도우 사이즈를 동적으로 계산하여 마우스커서를 이미지로 출력함으로써 2개 이상의 마우스를 각각 사용할 수 있게 한다.
- <65> 도 4는 본 발명의 일 실시예에 따른 멀티 마우스의 사용을 위하여 포인터를 생성하는 방법에 있어서 객체 관리자 네임스페이스 내의 내부 이름과 심볼릭 링크 이름의 관계를 도시한 도면이다.
- <66> 디바이스들은 하나 이상의 이름을 가질 수 있다. IoCreateDevice에 의해 내부적으로 지정된 이름은 OS에게 자신이 어떤 디바이스인지를 알려주는 역할을 한다. 이 내부 이름은 Win32 사용자 애플리케이션에게는 감춰져 있다. 새로운 디바이스는 디바이스를 Win32 서브시스템, Win16 서브 시스템 또는 가상 Dos 환경에게 알리기 위해서 자신의 내부 이름 이외에 심볼릭 링크 이름(Symbolic link Name)이 주어져야 한다.
- <67> 이러한 두 종류의 이름은 객체 관리자의 서로 다른 네임스페이스에 존재한다. 객체 관리자는 운영체제에 의해 관리되는 모든 자원들에 대한 이름 디렉토리(Name directory)를 관리하는 역할을 한다. 내부 디바이스 이름들은 디렉토리 트리(tree)의 \Device 섹션 아래에 저장된다. 심볼릭 링크 이름은 \?? 트리 아래에 존재한다. 또한, IoCreateDevice를 사용할 때에 \Device 이름이 반드시 주어져야 한다. 예를 들어, "\Device\Minimal0"의 형태가 적당한 디바이스 이름 문자열이 될 것이다. 백슬래쉬(\)는 C 언어에서 \t와 같이 특수 문자이므로, 이중 백슬래쉬(\\)로 표기하여야 한다.
- <68> 심볼릭 링크 이름은 내부 디바이스 이름과는 다른 디바이스 명명법을 따른다. 내부 디바이스 이름은 보통 긴 편이고, 항상 이들은 0을 기준으로 한 숫자로 끝난다(예를 들어, FloppyDisk0 또는 FloppyDisk1). 심볼릭 링크 이름은 보통 파일 시스템 디바이스들의 경우 A에서 Z의 형태를 따르고, 다른 디바이스들의 경우는 1을 기준으로 한 숫자로 끝난다(LPT1 또는 LPT2).
- <69> 심볼릭 링크명을 생성하기 위해서는 IoCreateSymbolicLink를 이용한다. 이 함수는 이미 존재하는 디바이스의 이름과 새로운 심볼릭 이름(UNICODE_STRING 데이터 형식)을 인자로 취한다.
- <70> 마지막으로, 디바이스 이름의 선택은 필수 선택 사항이 아니라는 점에 유의해야 한다. WDM에서는 버스 드라이버, 클래스 드라이버와 상당수의 PnP 디바이스들이 그들의 이름을 정의한다.
- <71> 본 발명은 컴퓨터에 의해 실행되는 프로그램 모듈과 같은 컴퓨터에 의해 실행가능한 명령어를 포함하는 기록 매체의 형태로도 구현될 수 있다. 컴퓨터 판독 가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 가용 매체일 수 있고, 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체를 모두 포함한다. 또한, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 모두 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함한다. 통신 매체는 전형적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈, 또는 반송파와 같은 변조된 데이터 신호의 기타 데이터, 또는 기타 전송 메커니즘을 포함하며, 임의의 정보 전달 매체를 포함한다.
- <72> 본 발명의 방법은 특정 실시예와 관련하여 설명되었지만, 그것들의 구성 요소 또는 동작의 일부 또는 전부는 범용 하드웨어 아키텍처를 갖는 컴퓨터 시스템을 사용하여 구현될 수 있다.
- <73> 도 5는 상술한 하나 이상의 구성 요소 또는 동작을 실시하기 위하여 사용될 수 있는 컴퓨터 시스템 아키텍처의 일례를 도시한 도면이다. 일 실시예에서, 하드웨어 시스템(800)은 프로세서(810), 캐쉬(815), 메모리(815) 및 상술한 기능에 관련된 하나 이상의 소프트웨어 어플리케이션 및 드라이버를 포함한다.
- <74> 추가적으로, 하드웨어 시스템(800)은 고성능 입/출력(I/O) 버스(840) 및 표준 I/O 버스(870)를 포함한다. 호스

트 브릿지(820)는 프로세서(810)를 고성능 I/O 버스(840)에 결합시키고, I/O 버스 브릿지(850)는 두 개의 버스(840 및 870)를 상호 결합시킨다. 시스템 메모리(860) 및 네트워크/통신 인터페이스(830)는 고성능 I/O 버스(840)에 결합된다. 하드웨어 시스템(800)은 비디오 메모리(도시 생략) 및 그 비디오 메모리에 결합된 디스플레이 장치를 더 포함할 수 있다. 대용량 기억장치(880) 및 I/O 포트(890)는 표준 I/O 버스(870)에 결합된다. 하드웨어 시스템(800)은 키보드 및 포인팅 디바이스(pointing device), 그리고 표준 I/O 버스(870)에 결합된 디스플레이 장치(도시 생략)를 선택적으로 포함할 수 있다. 총괄적으로, 이러한 요소들은 넓은 범위의 컴퓨터 하드웨어 시스템을 표현하도록 의도되며, 적절한 프로세서에 기반하는 범용 컴퓨터 시스템을 포함하나 이에 제한되지는 않는다.

<75> 하드웨어 시스템(800)의 구성 요소들은 아래에서 더 상세하게 설명된다. 보다 구체적으로, 네트워크 인터페이스(830)는 하드웨어 시스템과 이더넷(예를 들어, IEEE 802.3) 네트워크 등과 같은 임의의 넓은 범위의 네트워크 사이의 통신을 제공한다. 네트워크 인터페이스(830)는 하드웨어 시스템(800)과 네트워크 사이를 접속하여 하드웨어 시스템(800)이 그것들의 데이터베이스들을 관리하도록 한다. 대용량 기억장치(880)는 데이터와 프로그래밍 명령을 위한 영구 기억장치를 제공하여 본 발명에서 구현되는 상술한 기능을 수행하고, 시스템 메모리(860)(예를 들어, DRAM)는 프로세서(810)에 의하여 수행될 때 데이터 및 프로그래밍 명령을 위한 임시 기억장치를 제공한다. I/O 포트(890)는 하드웨어 시스템(800)에 결합될 수 있는 추가적인 주변 장치 간의 통신을 제공하는 하나 이상의 직렬 및/또는 병렬 통신 포트이다.

<76> 하드웨어 시스템(800)은 다양한 종류의 시스템 아키텍처를 포함할 수 있고, 하드웨어 시스템(800)의 다양한 구성요소가 재배치될 수 있다. 예를 들어, 캐쉬(815)는 프로세서(810)에 내장될 수 있다. 선택적으로, 캐쉬(815) 및 프로세서(810)는 "프로세서 모듈"로써 함께 묶여질 수 있고, 이 때 프로세서(810)는 "프로세서 코어"로서 지칭될 수 있다. 또한, 본 발명의 특정한 실시예는 상술한 구성 요소의 전부를 요구하거나 포함하지 않을 수 있다. 예를 들어, 표준 I/O 버스(870)에 결합되는 것으로 도시된 주변 장치들은 고성능 I/O 버스(840)에 결합될 수 있다. 추가적으로, 임의의 실시예에서 단지 하나의 버스만 존재할 수 있고, 하드웨어 시스템(800)의 구성 요소들은 그 하나의 버스에 결합될 수 있다. 더욱이, 하드웨어 시스템(800)은 추가적인 프로세서, 기억 장치 또는 메모리와 같은 추가적인 구성요소를 포함할 수 있다. 이하에서 논의되는 것처럼, 일 실시예에서, 본 발명의 동작은 하드웨어 시스템(800)에 의해 구동되는 일련의 소프트웨어 루틴으로서 실시될 수 있다. 이러한 소프트웨어 루틴들은 프로세서(810)와 같은 하드웨어 시스템에서 프로세서에 의하여 실행될 수 있는 복수의 또는 일련의 명령들을 포함한다. 우선, 일련의 명령들은 대용량 기억 장치(880)와 같은 기억 장치에 저장된다. 그러나 일련의 명령들은 디스켓, CD-ROM, ROM, EEPROM 등과 같은 임의의 적절한 기억 매체에 저장될 수 있다. 더욱이, 일련의 명령들은 국소적으로 저장될 필요가 없고, 네트워크/통신 인터페이스(830)를 통하여 네트워크 상의 서버 등과 같은 원격 기억 장치로부터 수신될 수 있다. 그 명령들은 대용량 기억 장치(880)와 같은 기억 장치로부터 시스템 메모리(860)로 복사되고, 프로세서(810)에 의하여 액세스되고 실행된다.

<77> 운영 시스템은, 소프트웨어 어플리케이션(도시 생략)과의 데이터 입/출력을 포함하는 하드웨어 시스템(800)의 동작을 관리하고 제어한다. 운영 시스템은 시스템 상에서 실행되는 소프트웨어 어플리케이션과 시스템의 하드웨어 구성 요소 사이의 인터페이스를 제공한다. 본 발명의 일 실시예에 따른 운영 시스템은 마이크로소프트사의 Windows 95/98/NT/XP/VISTA 운영 시스템일 수 있다. 그러나, 본 발명은 애플 컴퓨터사의 애플 맥킨토시 운영 시스템, 유닉스(UNIX) 운영 시스템, 리눅스(LINUX) 운영 시스템 등과 같은 다른 적절한 운영 시스템에서도 사용될 수 있다.

<78> 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적인 것이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

<79> 이상에서 본 발명의 실시예에 대하여 상세하게 설명하였지만 본 발명의 권리범위는 이에 한정되는 것은 아니고 다음의 청구범위에서 정의하고 있는 본 발명의 기본 개념을 이용한 당업자의 여러 변형 및 개량 형태 또한, 본 발명의 권리범위에 속하는 것이다.

발명의 효과

<80> 본 발명에 따르면, 2개 이상의 마우스가 발생시키는 이벤트를 감지하고 이에 응답하여 각각의 마우스에 대한 마

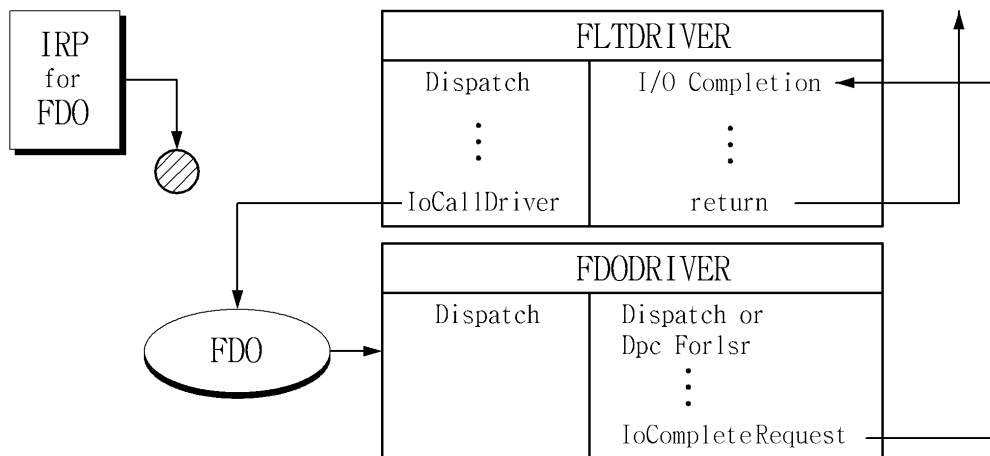
우스 포인터를 디스플레이함으로써 2개 이상의 마우스를 다양한 어플리케이션에 적용할 수 있는 효과가 있다.

도면의 간단한 설명

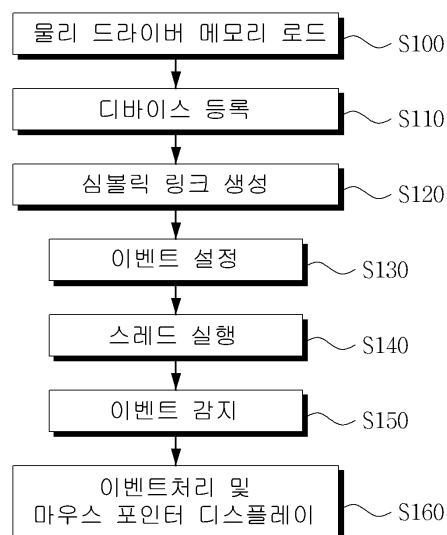
- <1> 도 1은 본 발명의 일 실시예에 따른 필터 드라이버의 동작을 설명한 도면,
- <2> 도 2는 본 발명의 일 실시예에 따른 멀티 마우스 포인터를 디스플레이하는 방법을 설명하기 위한 순서도,
- <3> 도 3은 본 발명의 일 실시예에 따른 멀티 마우스 포인터를 디스플레이하는 방법에 있어서 마우스 디바이스의 등록 단계의 세부 과정을 도시한 순서도,
- <4> 도 4는 본 발명의 일 실시예에 따른 멀티 마우스 포인터를 디스플레이하는 방법에 있어서 객체 관리자 네임스페이스 내의 내부 이름과 심볼릭 링크 이름의 관계를 도시한 도면,
- <5> 도 5는 본 발명을 실시하기 위하여 사용될 수 있는 컴퓨터 시스템 아키텍처의 일례를 도시한 도면.

도면

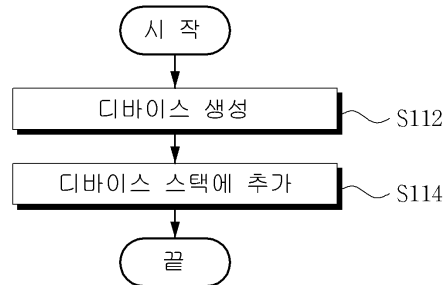
도면1



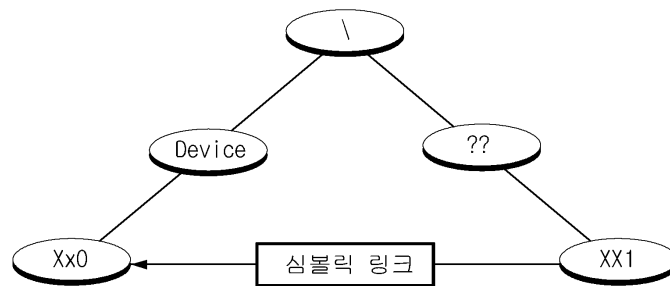
도면2



도면3



도면4



도면5

