introduction

작성일시	@2023년 12월 15일 오전 8:58
복습	☐
진행률	Not started

<오늘의 목표는?>

p.54까지 정리

Chapter 1 Introduction

1.1 운영체제가 하는 일

- 컴퓨터 시스템은 대략 **하드웨어, 운영체제, 응용 프로그램, 사용자**의 네 가지 구성 요소로 나눌 수 있다.
    - **하드웨어(cpu), 메모리, 입출력(I/O) 장치** 등은 시스템에 **기본 컴퓨팅 리소스를 제공**한다.
    - 워드 프로세서, 컴파일러, 웹 브라우저 등의 **응용 프로그램**은 이러한 **리소스(하드웨어)를 사용하여 사용자의 컴퓨팅 문제를 해결하는 방식**을 정의.
    - 운영체제는 **하드웨어를 제어하고 다양한 사용자를 위해 다양한 응용 프로그램 간의 하드웨어 사용을 조정.**
- 또한 컴퓨터 시스템은 **하드웨어, 소프트웨어, 데이터**로 구성되어 있다고 볼 수 있다.
- 운영체제는 **컴퓨터 시스템 작동 시 이러한 리소스를 적절하게 사용하기 위한 수단을 제공**.
- 운영 체제의 관점을 더 완전히 이해하기 위해 **사용자 관점**과 **시스템 관점**이라는 두 가지 관점에서 운영 체제를 살펴보겠음.

1.1.1 사용자 관점

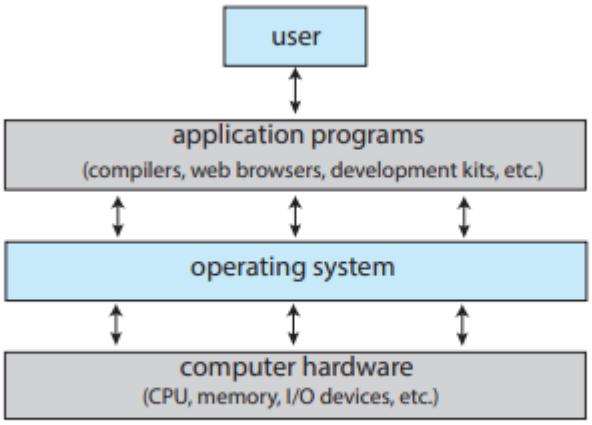

Figure 1.1 Abstract view of the components of a computer system.

- 컴퓨터에 대한 사용자의 시각은 사용되는 인터페이스에 따라 달라진다.
- 사용자 관점의 경우 주로 **사용하기 쉽도록 설계 되었으며,** 성능과 보안에 약간의 주의를 기울이고 **리소스 활용(다양한 하드웨어 및 소프트 웨어가 공유되는 방식)에는 신경을 쓰지 않았다.**
- 일부 컴퓨터에는 사용자 보기가 거의 혹은 전혀 없다.
    - 예를 들어, 가정용 기기와 자동차에 내장된 컴퓨터에는 숫자 키패드있고 표시등을 켜거나 꺼서 상태를 표시하지만, 컴퓨터와 해당 운영체제 및 애플리케이션은 기본적으로 사용자 개입 없이 실행되도록 설계.

1.1.2 시스템 보기

- 컴퓨터의 관전에서 보면 운영 체제는 하드웨어와 가장 밀접하게 관련된 프로그램이다.
- 이러한 맥락에서 우리는 운영 체제를 **자원 할당자**로 솔 수 있다.
- 컴퓨터 시스템에는 문제를 해결하는 데 필요한 cpu 시간, 메모리 공간, 저장 공간, I/O 장치 등 많은 리소소가 있고 운영 체제는 컴퓨터 시스템을 효율적이게 운영할 수 있도록 특정 프로그램과 사용자에게 리소스를 할당하는 방법을 결정하는 관리자 역할을 한다.

1.1.3 운영 체제 정의

- 운영 체제는 컴퓨터의 수많은 디자인과 용도 때문에 많은 역할을 가진다. 시스템의 기초이다. ⇒ **즉, 운영 체제에 대한 완전히 적절한 정의는 없다.**

- 컴퓨팅은 수행할 수 있는 작업을 결정하기 위한 실험으로 시작되었고 코드 해독 및 궤적 플로팅과 같은 군사적 용도와 인구 조사 계산과 같은 정부 용도를 위한 고정 목적 시스템으로 빠르게 이동했다.
- 초기 컴퓨터는 범용 다기능 메인프레임으로 발전했고, 이때 운영 체제가 탄생.
    - 1960년대 무어의 법칙은 집적 회로의 트랜지스터 수가 18개월마다 두 배로 늘어갈 것이라고 예측. 컴퓨터 기능은 향상되고 크기는 줄어들면서 사용 횟수가 늘어나고 운영체제도 다양해졌다.

무어의 법칙

- 우리가 운영체제를 주문할 때 포함되는 기능은 시스템마다 크게 다르다.
    - 일부 시스템은 1MB 미만의 공간을 차지하고 전체 화면 편집기조차 없는 반면, 다른 시스템은 기가바이트의 공간이 필요하고 전적으로 그래픽 창 시스템을 기반으로 한다.
- 보다 일반적인 정의이자 우리가 일반적으로 따르는 정의는 운영체제가 컴퓨터에서 항상 실행되는 하나의 프로그램, 즉 일반적으로 커널이라는 것이다.
    - 운영 체제와 연관되어 있지만 반드시 커널의 일부는 아닌 시스템 프로그램과 시스템 작동과 연관되지 않은 모든 프로그램을 포함하는 응용 프로그램이다.
- 모바일 운영 체제를 살펴보면 운영 체제를 구성하는 기능의 수가 다시 증가하고 있음을 알 수 있다. 모바일 운영 체제에는 핵심 커널뿐만 아니라 응용 프로그램 개발자에게 추가 서비스를 제공하는 소프트웨어 프레임워크 집합인 미들웨어도 포함되는 경우도 있다.
    - apple의 ios와 google의 android가 다음의 기능들을 제공한다.

요약하자면, 우리의 목적을 위해 운영 체제에는 항상 실행되는 커널, 애플리케이션 개발을 용이하고 기능을 제공하는 미들웨어 프레임위크, 실행 중인 시스템 관리에 도움이 되는 시스템 프로그램이 포함이 된다.

1.2 컴퓨터 시스템 구성

- 현대의 범용 컴퓨터 시스템은 하나 이상의 CPU와 구성 요소와 공유 메모리 간의 액세스를 제공하는 공통 버스를 통해 여러 연결된 여러 장치 컨트롤러로 구성된다.
- 각 장치 컨트롤러는 특정 유형의 장치(디스크 드라이브, 오디오 장치 등)를 담당한다.
- 컨트롤러에 따라 두 개 이상의 장치가 연결된다.
    - 하나의 시스템 USB 포트를 USB 허브에 연결하고 여러 장치를 연결.
- 장치 컨트롤러는 일부 로컬 버퍼 저장소와 특수 목적 레지스터를 유지 관리한다. 장치 컨트롤러는 자신이 제어하는 주변 장치와 로컬 버퍼 저장소 간에 데이터를 이동하는 역할을 담당한다.

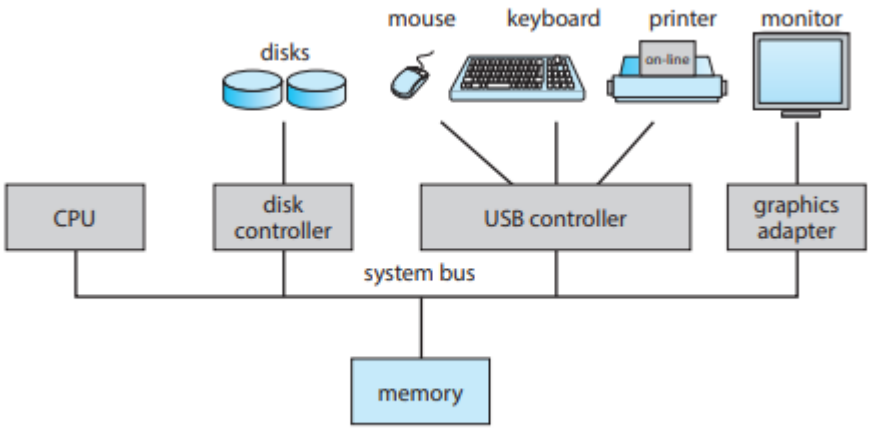

Figure 1.2 A typical PC computer system.

- 일반적으로 운영 체제에는 각 장치 컨트롤러에 대한 장치 드라이버가 있다.
- 장치 드라이버는 장치 컨트롤러를 이해하고 운영 체제의 나머지 부분에 장치에 대한 균일한 인터페이스를 제공한다.
- CPU와 장치 컨트롤러는 병렬로 실행되어 메모리 주기를 놓고 경쟁한다.
- 공유 메모리에 대한 순차적인 액세스를 보장하기 위해 메모리 컨트롤러는 메모리에 대한 액세스를 동기화 한다.

1.2.1 인터럽트

1. I/O 작업을 시작하기 위해 장치 드라이버는 장치 컨트롤러에 적절한 레지스터를 로드한다.

2. 장치 컨트롤러는 이러한 레지스터의 내용을 검사하여 수행할 작업(예시: 키보드에서 문자읽기)를 결정한다.
3. 컨트롤러는 장치에서 로컬 버퍼로 데이터 전송을 시작한다. 데이터 전송이 완료되면 장치 컨트롤러는 장치 드라이버에게 작업이 완료되었음을 알린다.
4. 그런 다음 장치 드라이버는 운영체제의 다른 부분에 제어권을 부여하고, 작업이 읽기인 경우 데이터 또는 데이터에 대한 포인터를 반환한다.
5. 다른 작업의 경우 장치 드라이버는 “쓰기가 성공적으로 완료되었습니다” 또는 “장치 사용 중”과 같은 상태 정보를 반환한다.
6. 그러면 컨트롤러는 장치 드라이버에 작업이 완료되었음을 인터럽트를 통해 수행한다.

- 하드웨어는 일반적으로 시스템 버스를 통해 CPU에 신호를 보내 언제든지 인터럽트를 트리거 할 수 있다.
    - 컴퓨터 시스템에는 많은 버스가 있을 수 있지만 시스템 버스는 주요 구성 요소 간의 주요 통신 경로이다.
- CPU 가 중단되면 수행 중인 작업을 중지하고 즉시 고정된 위치로 실행을 전공한다. 고정 위치에는 일반적으로 인터럽트에 대한 서비스 루틴이 위치한 시작 주소가 포함된다.
- 인터럽트는 적절한 인터럽트 서비스 루틴으로 제어를 전달해야 한다. 해당 전송을 관리하는 간단한 방법은 일반 루틴을 호출하여 인터럽트 정보를 검사하는 것이다.

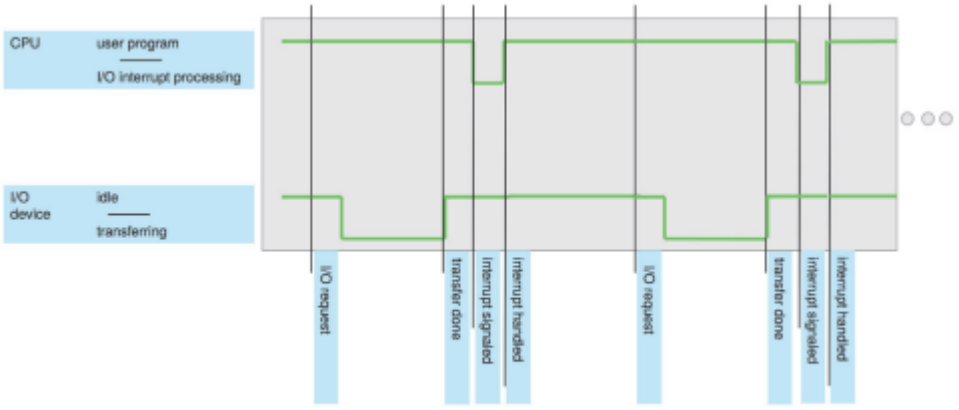

Figure 1.3 Interrupt timeline for a single program doing output.

인터럽트 서비스 루틴은 다음과 같다.

1. 인터럽트 관련 핸들러를 호출. 단, 인터럽트는 매우 자주 발생하므로 빠르게 처리.

2. 인터럽트 루틴에 대한 포인터 테이블을 대신 사용하여 필요한 속도를 제공할 수 있다.
    a. 인터럽트 루틴은 중간 루틴이 필요 없이 테이블을 통해 간접적으로 호출된다.
3. 일반적으로 포인터 테이블은 낮은 메모리(처음 100개 정도의 위치)에 저장된다. 이러한 위치는 다양한 장치에 대한 인터럽트 서비스 루틴의 주소를 보유한다.
4. 이 주소 배열 또는 인터럽트 벡터는 인터럽트 요청과 함께 제공되는 고유 번호로 색인화되어 인터럽트 장치에 대한 인터럽트 서비스 루틴의 주소를 제공.
5. 인터럽트 아키텍처는 인터럽트 처리 후 이 정보를 복원할 수 있도록 인터럽트된 모든 항목의 상태 정보도 저장해야 한다.
    a. 인터럽트 루틴이 프로세서 상태를 수정해야 하는 경우(예: 레지스터 값 수정) 현재 상태를 명시적으로 저장한 다음 반환하기 전에 해당 상태를 복원해야 함.

1.2.1.2 구현 기본

인터럽트 메커니즘은 다음과 같이 작동한다.

1. CPU 하드웨어에는 모든 명령을 실행한 후 CPU가 감지하는 인터럽트 요청 라인이라는 와이어가 있다.
2. CPU는 컨트롤러가 인터럽트 요청 라인에 신호를 보낸 것을 감지하면 인터럽트 번호를 읽고 해당 인터럽트 번호를 인터럽트 벡터에 대한 인덱스로 사용하여 인터럽트 처리 루틴으로 간다.
3. 그런 다음 해당 인덱스와 연관된 주소에서 실행을 시작한다. 인터럽트 핸들러는 작동 중에 변경될 모든 상태를 저장하고, 인터럽트의 원인을 파악하고, 필요한 처리를 수행하고, 상태 복원을 수행하고, 인터럽트 명령에서 복귀를 실행하여 CPU를 인터럽트 이전의 실행 상태로 되돌린다.

⇒ 장치 컨트롤러가 인터럽트 요청 라인에 신호를 지정하여 인터럽트를 발생시키고, CPU가 인터럽트를 포착하여 인터럽트 핸들러에 전달하고, 핸들러가 장치를 서비스하여 인터럽트를 지운다고 한다.

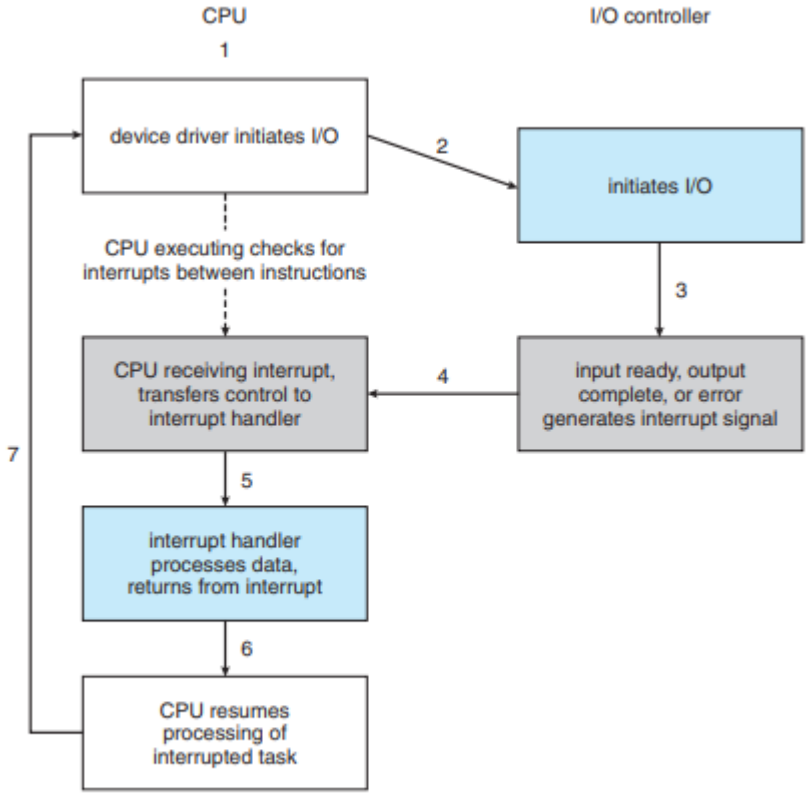

Figure 1.4 Interrupt-driven I/O cycle.

해당 그림은 인터럽트 기반 I/O 주기가 요약되어 있다.

- 대부분 cpu에는 두개의 인터럽트 요청 라인이 있는데, 하나는 복구할 수 없는 메모리 오류와 같은 이벤트를 위해 예약된 **nonmaskable과 maskable**이 가능한 인터럽트 라인이 있다.
- 인터럽트되어서는 안되는 중요한 명령 시퀀스가 실행되기 전에 cpu에 위해 꺼질 수 있으며 maskable이 가능한 인터럽트는 장치 컨트롤러가 서비스를 요청하는데에 사용이 된다.

방금 설명한 기본 인터럽트 메커니즘을 사용하면 장치 컨트롤러가 서비스 준비가 될 때와 같이 CPU가 비동기 이벤트에 응답할 수 있지만, 최신 운영 체제에서는 보다 정교한 인터럽트 처리 기능이 필요.

1. 중요한 처리 중에 인터럽트 처리를 연기하는 기능이 필요.
2. 적절한 인터럽트 핸들러로 디스패치하는 효율적인 방법이 필요.
3. 운영체제가 우선 순위가 높은 인터럽트와 낮은 우선수위 인터럽트를 구별하고 적절한 긴급도로 응답할 수 잇도록 다중 레벨 인터럽트가 필요하다.

다음은 Intel 프로세서의 인터럽트 벡터 설계를 보여준다.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 1.5 Intel processor event-vector table.

maskable할 수 없는 0부터 31까지의 이벤트는 다양한 오류 조건을 알리는데 사용이 되고 maskable이 가능한 32번부터 255번까지의 이벤트는 장치 생성 인터럽트 등의 용도로 사용이 된다.

1.2.2 저장구조

- 범용 컴퓨터는 **주메모리(RAM)** 이라는 반도체 기술로 구현이 된다.
- 컴퓨터는 다른 형태의 메모리도 사용을 한다.
  - 예를 들어, 컴퓨터를 부팅시 실행되는 첫 번째 프로그램은 **부트스트랩 프로그램**이고, 이후 운영체제를 로드한다. **RAM은 휘발성이므로 부트스트랩을 보유한다고 볼 수 없다.**
  - 대신, 목적과 다른 목적을 위해 컴퓨터는 전기적으로 지울 수 있는 프로그래밍이 가능한 읽기 전용 메모리(EEPROM) 와 다른 형태의 펌웨어를 사용한다.

- 폰 노이만 아키텍처를 사용하는 시스템에서 실행되는 일반적인 명령 실행 주기는 먼저 메모리에서 명령을 가져와 해당 명령을 **명령 레지스터**에 저장한다.
- 메모리 장치는 메모리 주소 스트림만 볼 수 있고, 그것들이 어떻게 생성되는지 (명령어 카운터, 인덱싱, 간접지정, 리터럴 주소 또는 기타 수단에 의해) 또는 그 용도(명령어 혹은 데이터)를 알지 못한다.
    - 따라서 프로그램이 메모리 주소를 생성하는 방법을 무시할 수 있다.
    - 우리는 실행 중인 프로그램에 의해 생성된 메모리 주소 시퀀스에만 관심이 있다.

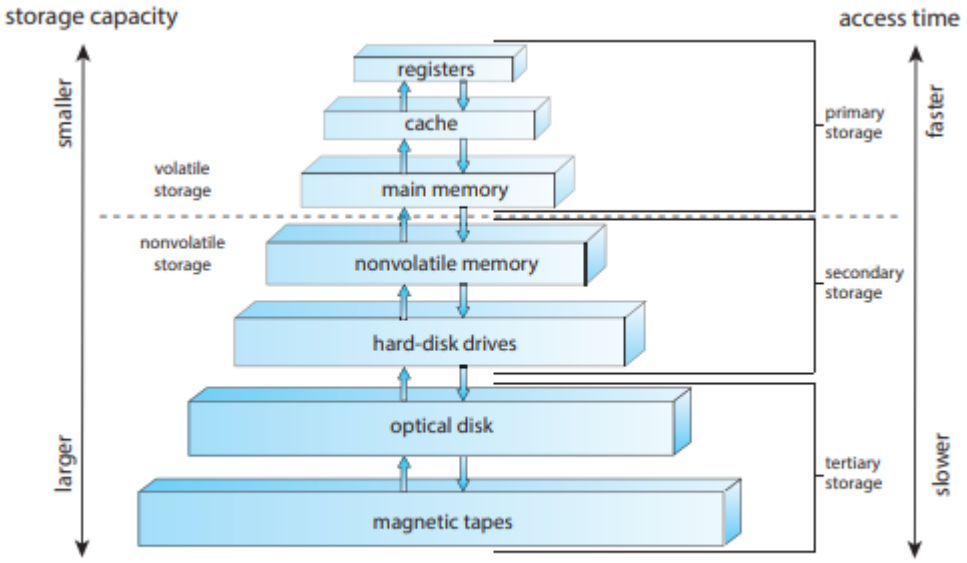

Figure 1.6 Storage-device hierarchy.

- 주 메모리는 일반적으로 필요한 모든 프로그램과 데이터를 저장하기엔 너무 작고, 휘발성이므로 다른 보조 저장소를 제공한다.
- 보조 저장소의 주요 요구사항은 대량의 데이터를 영구적으로 보관하는 것이다.
    - 가장 일반적인 보조 장치에는 하드 디스크 드라이브, 비휘발성 메모리 장치 등이 있다.
- 다른 많은 저장 시스템이 있는데, 다양한 스토리지 시스템 간의 주요 차이점은 속도, 크기 및 변동성에 있다. (위의 그림 참조.)