System Structure & Program Execution 1

상태	완료
생성 일시	@2023년 6월 29일 오후 8:01
완성일	@2023년 6월 29일
유형	정리

들어가기에 앞서

운영체제도 프로그램임 시작되면 메모리에 올라오게 되어있음 그래서 프로그램이 처음에 실행되면

운영체제가 메모리에 올라와서 CPU를 잡고 있다가 다른 사용자 프로그램에 CPU를 넘겨주는 것인 메모리에

올라와 있는 상태로 운영체제(프로그램)가 메모리에 올라와 있는 것을 커널이라고 부름

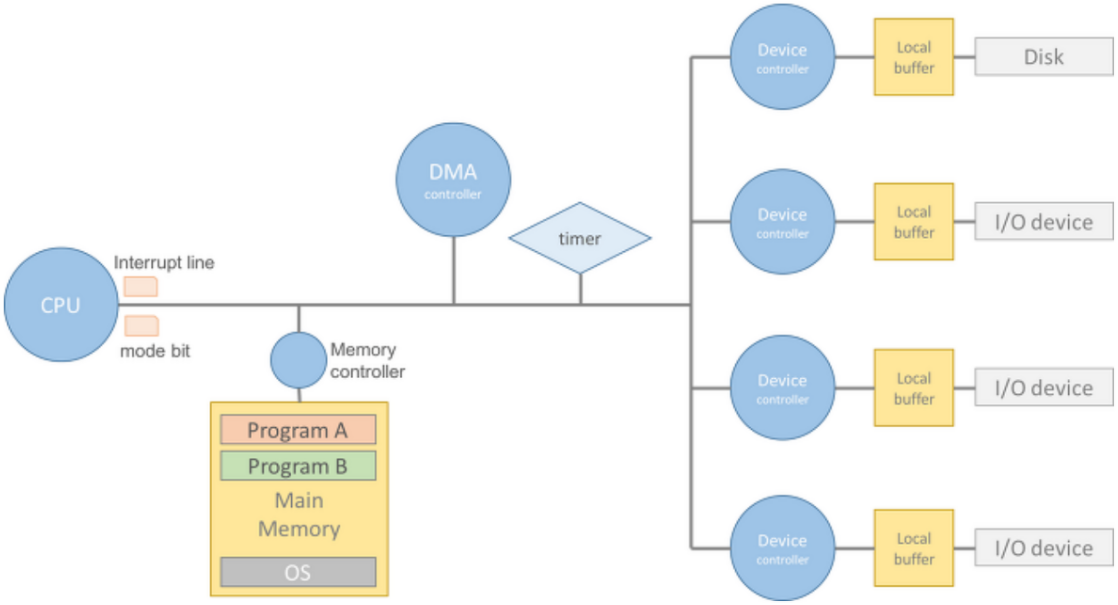
▼ 커널

커널은 운영체제의 핵심 기능을 수행하는데 필요한 코드와 데이터로 구성된 프로그램입니다. 이 프로그램은 메모리에 상주하여 운영체제의 모든 기능을 지원하고, 하드웨어 자원에 직접적인 접근을 할 수 있습니다. 커널은 시스템 호출(System Call)이나 인터럽트(Interrupt)와 같은 메커니즘을 통해 프로세스와 하드웨어 사이의 상호 작용을 조정하고 관리합니다.

커널은 운영체제의 다양한 서비스를 제공하며, 프로세스 관리, 메모리 관리, 입출력 관리, 파일 시스템 관리 등의 기능을 수행합니다. 또한 커널은 시스템의 안전성과 보안을 유지하기 위해 권한과 접근 제어를 관리합니다.

일반적으로 커널은 운영체제의 핵심 부분을 의미하며, 메모리에 상주하여 시스템의 핵심적인 기능을 제공하는 운영체제의 핵심 프로그램입니다.

컴퓨터 시스템의 구조



큰 흐름

컴퓨터 시스템의 구조는 컴퓨터 내부 장치인 CPU, 메모리와 컴퓨터 외부 장치(입출력 장치)인 디스크, 키보드, 마우스, 모니터, 네트워크 장치 등으로 구분된다. 컴퓨터는 외부 장치에서 내부 장치로 데이터를 읽어와 (input) 각종 연산을 수행한 후, 그 결과를 외부 장치로 내보내는(output) 방식으로 업무를 처리한다.

각 요소에 대한 설명

Memory

메모리는 CPU가 직접 접근할 수 있는 내부 기억 장치로서 특정 프로그램이 CPU에서 실행되려면 해당 부분이 메모리에 올라가 있어야 한다. 운영 체제는 컴퓨터가 부팅되었을 때 메모리에 올라가 있는데, 메모리에 상주하고 있는 CPU의 작업 공간을 Main Memory라고 부른다.

Device Controller

메모리 및 입출력 장치 등의 각 하드웨어 장치에는 Device Controller라는 것이 붙어 있다. 컨트롤러는 일종의 작은 CPU로서, 각 하드웨어 장치를 제어하는 역할을 수행한다.

Local Buffer

메인 CPU의 작업 공간인 메인 메모리가 있듯이 디바이스 컨트롤러도 데이터를 임시로 저장하기 위한 작업 공간이 필요한데, 이를 Local Buffer가 그 역할을 한다.

참고로, Device Driver는 CPU가 실행하는 각 디바이스에 접근하기 위한 소프트웨어를 뜻한다(각 하드웨어에 맞게 접근 할 수 있도록 하는).

CPU

CPU는 클럭마다 메모리에서 명령(Instruction)을 하나씩 읽어서 실행하는 역할을 한다. IO가 일어나면 CPU가 직접 접근하지 않고 Device Controller에게 시킨다. 이렇게 디바이스 컨트롤러에게 시키는 이유는 CPU에 비해 디바이스 속도가 매우 느리기 때문이다. (그래서 Device Controller에게 일을 시키고 다른 일을 하는 것임)

접근 범위

CPU는 메인 메모리와 Local Buffer(각 장치별 작업이 끝나거나 어떠한 처리를 했을때 데이터를 저장하는 부분)에 접근이 가능하다.

인터럽트 라인 (Interrupt Line)

CPU가 자신의 작업을 하던 중간에 인터럽트 라인에 신호가 들어오면 하던 일을 멈추고 인터럽트와 관련된 일을 먼저 처리한다.

레지스터 (Register)

CPU 내부에 메모리보다 더 빠르면서 정보를 저장할 수 있는 작은 공간이다. 레지스터 중에 메모리 주소를 가리키는 레지스터인 PC (Program Counter) 레지스터가 있다. CPU는 PC 레지스터가 가리키는 메모리 위치에서 인스트럭션을 읽어서 수행한다.

Mode bit

CPU에서 실행되는 것이 운영 체제인지 사용자 프로그램인지 구분해 준다.

- 1 (사용자 모드) - 사용자 프로그램
  - 사용자 프로그램이 CPU 가질 때는 제한된 인스트러션만 실행시킬 수 있다. 만약 Mode bit가 없다면 사용자 프로그램에서 하드웨어를 직접 접근하여 보안에 취약해질 수 있다.
- 0 (커널 모드) - 운영 체제
  - 운영 체제가 CPU에서 실행 중일 때는 모든 인스트럭션을 실행할 수 있다. (모니터 모드 혹은 시스템 모드라고 부른다.)

▼ 인스트럭션이란?

운영체제에서 "인스트럭션(instruction)"은 컴퓨터 프로세서가 수행해야 할 작업이나 명령을 나타내는 기본적인 단위입니다. 인스트럭션은 프로그램의 명령어(instruction)로 이해할 수도 있습니다.

컴퓨터는 인스트럭션을 이해하고 실행하여 원하는 작업을 수행합니다. 인스트럭션은 CPU(Central Processing Unit, 중앙처리장치)에서 처리되며, 컴퓨터 아키텍처에 따라

다양한 형식을 가질 수 있습니다.

Timer

사용자 프로그램에서 while문으로 무한 루프를 돌게 된다면 특정 프로그램이 CPU를 독점할 수 있다. Timer는 특정 프로그램이 CPU를 독점하는 것을 막기 위한 하드웨이다.

동작

컴퓨터를 시작하면 처음에 운영 체제가 CPU를 가지고 있다가 사용자 프로그램에게 CPU를 넘겨준다. 하지만 그냥 넘겨 주지 않고 Timer에 값을 세팅하고 넘겨 주게 된다. 시간이 지나서 Timer의 값이 0이 되면 타이머 인터럽트가 발생하여 다른 프로그램에게 CPU를 넘겨준다.(즉 ***Timer 또한 인터럽트를 CPU에게 걸수 있는 것임***)

(운영체제가 사용자 프로그램에 CPU를 주면 마땅히 다시 뺏을 방법이 없기 때문에 만든 것임)

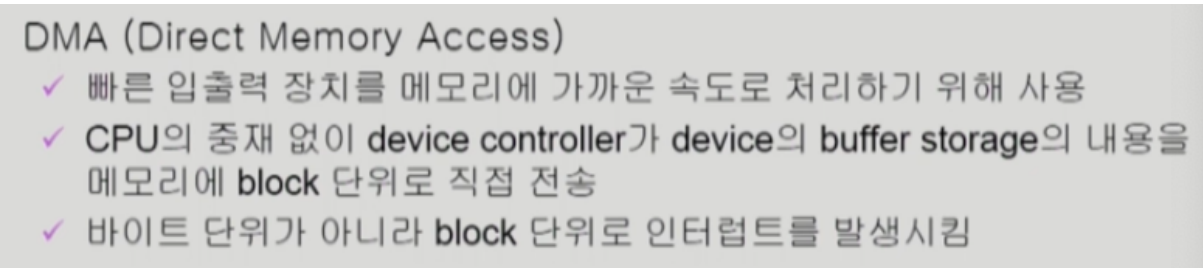
CPU는 계속 명령만 처리하는 게 아니고 명령 처리하다가 인터럽트 라인 체크하고 다시 명령 처리하는 것을 반복함

DMA (Direct Memory Access) Controller

원칙적으로 메모리는 CPU에 의해서만 접근할 수 있는 장치이다. CPU 외의 장치가 메모리의 데이터에 접근하기 위해서는 CPU에게 인터럽트 발생시켜 CPU가 대신 컨트롤러의 로컬 버퍼와 메모리 사이에서 데이터를 옮겨 준다. 하지만, 작업 처리 속도가 매우 빠른 CPU가 인터럽트를 많이 당하면 비효율적이다. 그래서 CPU 이외에 메모리 접근이 가능한 DMA 컨트롤러를 둔다.

DMA를 사용하게 되면 로컬 버퍼에서 메모리로 읽어오는 작업을 CPU 대신 수행해 줄 수 있다. 이때, DMA는 바이트 단위가 아니라 블록이라는 큰 단위로 정보를 메모리로 읽어온 후에 인터럽트를 발생시켜서 작업 완료 신호를 CPU에게 보낸다. 이러한 방식으로 CPU에 발생하는 인터럽트의 빈도를 줄여 CPU를 효율적으로 이용할 수 있다.

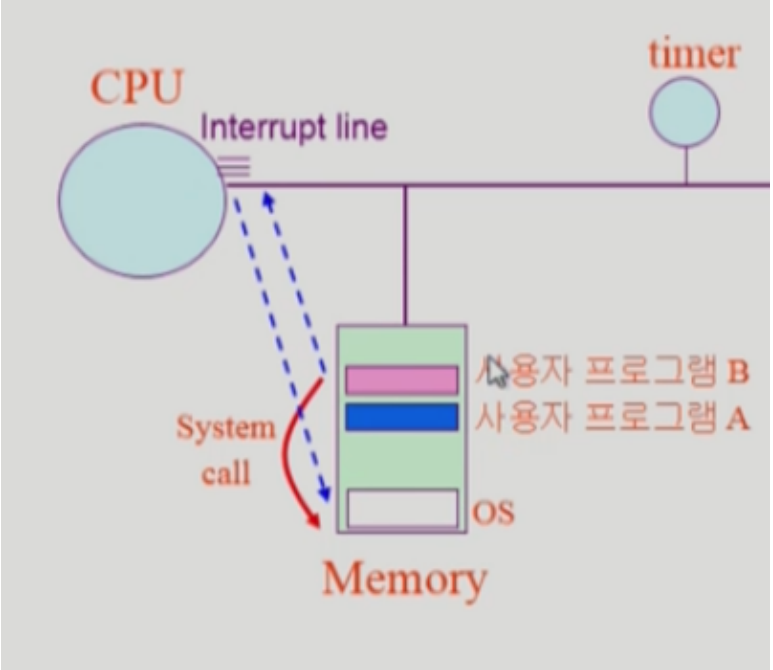
다만, CPU와 DMA가 동시에 메인 메모리에 접근할 수 있다는 문제점이 있어서 누가 메모리에 먼저 접근하게 만들지 Memory Controller가 교통 정리하는 역할을 한다.



입출력 I/O의 수행

모든 입출력 명령은 운영 체제만 사용할 수 있는 특권 명령으로만 가능하다. 그렇다면 사용자 프로그램은 어떻게 I/O를 할 수 있을까? 바로, 시스템 콜을 활용한다.

시스템 콜은 운영 체제에게 I/O를 요청하는 것을 말한다. 운영 체제의 서비스를 받기 위해 커널 함수를 호출하는 것이라고 생각하면 된다.



동작

mode bit가 1인 상태로 사용자 프로그램이 실행되고 있다가 I/O를 해야 하는 상황이 오면 바로 OS의 주소로 점프를 할 수가 없다. 그래서 프로그램이 직접 인터럽트 라인을 세팅 한 후, CPU에게 인터럽트를 보내면, CPU는 다음 인스트럭션을 수행하는 대신 mode bit를 0으로 바꾸고 CPU 제어권을 OS로 넘기게 된다. 이러한 방식으로 사용자 프로그램이 OS에게 I/O 요청을 보낼 수 있다.

▼ 좀더 상세히

c언어로 프로그램을 만들면 메인 함수가 있고 A 함수를 만들어 실행할 수가 있음

예를 들어 CPU는 인스트럭션을 순차적으로 실행하다가 함수호출이나 제어문, 반복문을 만나면

인스트럭션 메모리 주소를 점프할 것임 즉 내 프로그램 안에서 함수호출 하거나 어떠한 동작을 처리하는 것은 내 프로그램 영역에 해당하는 주소 안에서 주소를 바꾸는 거지만 내 프로그램이 실행되다가 I/O를 요청할 때는 그냥 내 프로그램 영역에 메모리 주소를 점프해서 되는 일은 아님 직접 운영체제 메모리 쪽으로 점프할 수 없음

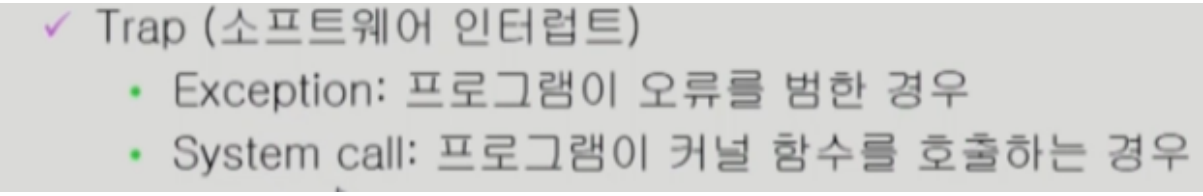
나는 사용자 프로그램이라 mode bit가 1임 그래서 프로그램이 직접 인터럽트 라인을 세팅한 후, CPU에 인터럽트를 보내면, CPU는 다음 인스트럭션을 수행하는 대신 mode bit를 0으로 바꾸고 CPU 제어권을 OS로 넘기게 된다. 이러한 방식으로 사용자 프로그램이 OS에게 I/O 요청을 보낼 수 있다.

즉 I/O를 하기 위해서는 총 두 번의 인터럽트가 발생하는 것임 요청할 때는 소프트웨어 인터럽트를 통해 요청할 거고 I/O가 끝나면 하드웨어 인터럽트를 통해서 알려줌 (I/O ***Device Controller가***)

인터럽트

CPU가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치나 예외 상황이 발생하여 처리가 필요한 경우에 CPU에게 알리는 일종의 이벤트를 뜻한다. 좁은 의미의 인터럽트는 하드웨어 인터럽트를 의미하지만, 넓은 의미의 인터럽트는 소프트웨어 인터럽트까지 포함한다.

소프트웨어 인터럽트는 Trap이라고도 부르며, 프로그램이 오류를 발생하여 Exception이 발생하거나(mode bit 1 인데 이상한 작업을 하려고 할때 CPU가 운영체제로 넘어감), 시스템 콜을 통해 커널 함수를 호출할 때 등이 있다.



하드웨어 인터럽트 vs 소프트웨어 인터럽트

CPU 인터럽트 라인에 신호를 보내서 인터럽트를 알려주는 방식은 똑같다. 다만 하드웨어 인터럽트는 컨트롤러 등 하드웨어 장치가 CPU의 인터럽트 라인을 세팅하는 반면, 소프트웨어 인터럽트는 소프트웨어가 CPU의 인터럽트 라인을 세팅한다.

인터럽트 벡터

운영 체제는 할 일을 쉽게 찾아가기 위해 인터럽트 벡터를 가지고 있는데, 인터럽트 벡터란 인터럽트 종류마다 번호를 정해서 번호에 따라 처리해야 할 코드가 위치한 부분을 가리키고 있는 자료 구조를 말한다.

인터럽트 처리 루틴 (인터럽트 핸들러)

인터럽트 벡터에서 실제 처리해야 할 코드를 인터럽트 처리 루틴 혹은 인터럽트 핸들러라고 부른다.

인터럽트 처리 루틴을 통해 해당하는 인터럽트 처리를 완료하고 나면 원래 수행하던 작업으로 돌아갈 위치를 알아야 하므로 인터럽트 처리 전에 수행 중이던 작업이 무엇이었는지 반드시 저장해야 하는데, 이러한 정보를 저장하기 위해 운영 체제는 PCB라는 공간을 별도로 가지고 있다.