4강

More on Container

- Container virtualize OS
  - Not hardware
  - Container engine: docker 등
  - 표준: OCI (open container initiative)

    Container는 표준이 있다.

  - Container image
    - Include everything needed to run application

      app은 어디든지 돌아갈 수 있다.

      APP은 docker만 설치되어 있으면 어느 OS에서도 다 돌아갈 수 있다.
    - Program, runtime, system tools, system libraries and settings
    - CI will always run the same, regardless of the infrastructure
      - Write once, run anywhere

        CI만 있으면 한번만 쓰면 하드웨어와 상관없이 어디든지 돌아간다.
      - Deployability

        Container의 가장 큰 장점은 Deployability이다.

        이는 어디든지 보내서 수행시키는 것이 가능하다.

  - Comparison: Java

    java랑 비교해보자

    java는 언어이다. java의 탄생배경은 무엇인가?

    java의 특징이자 장점이 무엇인가? java는 프로그래밍 언어이면서 WORA(Write once, run anywhere)를 지향했다.

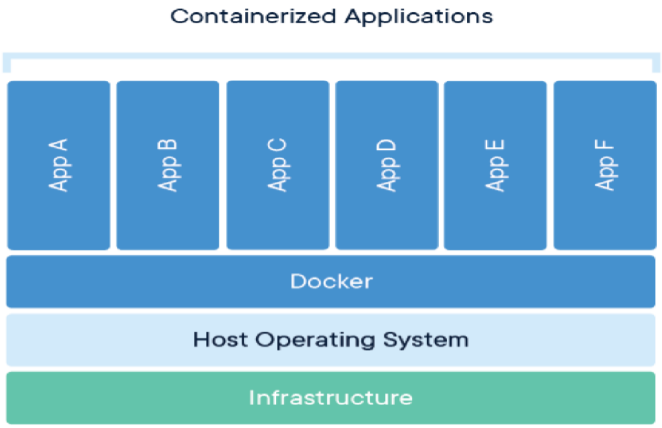
    이것을 해결하기 위해 자바 compiler를 사용하여 이식성을해결해보려고했다.

이 노력은 실패했다. JVM이 환경에 따라 크게 달라졌고, 바이트 코드를 사용하면 성능이 떨어지는 문제가 생겼다.

OS에 대한 프로그래밍의 의존도를 해결하는 것이 아니라, OS자체를 이미지로 만들어서 OS를 WORA로 만들어버린 것이다.

이식성의 문제를 container가 해결했다.

Container image에는 OS에 필요한 모든 것들이 다 들어있다.



Q. Container 는 현재 상용화되는 중인가요?

A. 굉장히 널리 사용된다.

1년 사이에 Container가 보편적으로 사용되기 시작했다.

Q. 우리가 사용하는 window 위에도 docker가 작동하고있나요??

A. NO.

docker를 윈도우에 설치 가능하다.

Q. 호스트 OS의 커널을 공유하는 것으로 아는데, 컨테이너 중 커널이 다른 OS, 이를테면 우분투의 다른 버전 등은 어떻게 구현이 되나요?

A. 우분투의 버전은 상관이 없다.

Q. 답변 감사합니다. 그런데 동일 머신 내에 커널을 달리쓰는 리눅스 배포판이 동시에 구동될 수 있는 원리가 궁금합니다.

A. KVM을 사용하면 쓸 수 있다.

원리는 기술적으로 더 들어가야 한다.

케이블이 hypervisor가 다른 커널들을 virtualize한다.

Q. 제가 과거에 VM에 깔린 Centos에서 다양한 컨테이너들을 돌렸던 경험이 있었는데, 어떤 컨테이너는 다른 OS를 사용하는 것처럼 보이는...? 경험을 했던 것 같습니다!

A. KVM에서 VM안에서 Container를 돌릴 수 있다.

우리가 하는 것도 window 호스트 머신에서 다른 OS를 돌리는 것이다.

이 장의 목표

- 운영체제에서 알아야 할 HW 이슈 정리
  - 컴퓨터 시스템의 전체적인 구성
  - 장치의 성능 비교
  - I/O의 동작원리

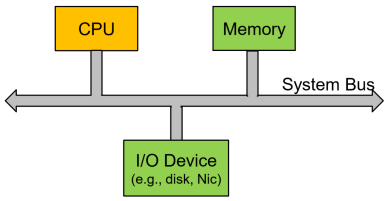
Contents

- 버스 구조
  - 병목현상과 이중화
- I/O basics
  - Event handling Mechanisms
    - Interrupt
    - Trap
  - I/O 처리 기법
    - Polling
      - Direct Memory Access(DMA) DMA-READ I/O Device access 기법 I/O Instruction Memory Mapped I/O

버스구조

- 단일 버스
  - 한 종류의 시스템 버스에 여러가지 모듈이 연결
  - CPU, Memory, I/O의 속도가 비슷했던 초창기에 사용
  - CPU, Memory, I/O의 속도 격차 증가 → 병목 현상 발생

    지금은 CPU가 정말빠르고 그다음이 메모리, 그 다음이 I/O이다.



<단일 버스 구조>

장치의 속도비교

- CPU > Memory >> I/O 로 속도의 격차가 커짐
  - CPU와 메모리 속도 (1000배 정도)
  - 메모리와 disk (1000배 정도)
  - Disk와 네트워크
  - I/O 장치 – CD, Keyboard

- 이로 인한 병목현상 발생

Bottleneck

- 병목현상

  빠른 device가 처리하는 데이터를 느린 device가 제대로 처리를 못하는 현상

  - 같은 버스에 연결된 디바이스들 간의 속도 차이로 인해 발생
  - 빠른 디바이스가 처리하는 양 만큼을 느린 디바이스가 처리하지 못함 → 빠른 디바이스 stall

- 전체 시스템 속도는 느린 디바이스의 속도로 제한됨

  CPU가 아무리 빨라도 소용이 없더라

- Ex) CPU는 초당 5 단위의 일을 처리할 수 있는데 메모리가 초당 3 단위의 일만을 전달해 줄 수 있다고 할 때, 전체적인 시 스템 속도는 메모리의 속도로 제한된다

버스이중화

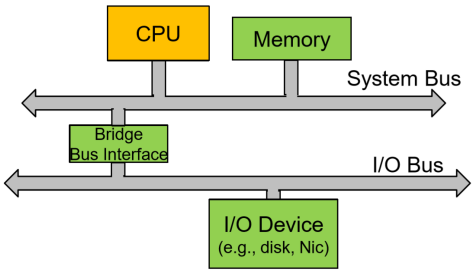
- 계층적 버스 구성
  - 속도 격차로 인한 병목 현상 해결
  - 접근 빈도가 적고, 처리 속도가 느린 장치들은 시스템 버스에 직접 연결 하지 않음(I/O 버스를 거쳐 연결됨) → stall 방지

CPU, Memory의 간격을 가깝게 하고 빠르게 만든다.

I/O는 따로 만든다. 한다.

그리고 버스 사이의 브릿지를 만들어서

PCI버스가 나오게 된 이유 또한 이와 같다.



<이중 버스 구조>

이벤트처리기법: Interrupt

- 비동기적 이벤트를 처리하기 위한 기법
  - 예: 네트워크 패킷 도착 이벤트, I/O 요청

→ 길을 가고 있는데 갑자기 돌맹이가 날아온다. → 비동기적 (외부적사건)

→ 길을 가는데 뭔가에 걸려넘어졌다. → 동기적 (일의 프로세스가 직접적인 원인이 됨)

- 인터럽트 처리 순서
  - 1. 현재 실행 상태(state)를 저장
  - 2. ISR(interrupt service routine)로 점프
  - 3. 저장한 실행 상태(state)를 복원
  - 4. 인터럽트로 중단된 지점부터 다시 시작

  DDOS가 동작하는 방법은 외부에서 패킷을 1, 2, 3개를 보내고 초당 1000만개씩을 보낸다. 그러면 계속 interrupt만 처리하게 된다. 돌아오면 다시interrupt를 수행하는 것이다. 즉, 메인 Program Execution Flow에 머무를 수가 없는 것이다.

- 인터럽트에는 우선순위가 있으며, H/W 장치마다 다르게 설정됨

  패킷이 들어올 때 입력이 들어오면 뭘 먼저 처리해야하는가?

  → 패킷을 먼저 처리한다면 어떻게 먼저 처리하나?

  → 그렇다면 뭐가 네트워크이고, 뭐가 키보드인지를 알아야 한다. 그래서 priority가 필요한 것이다.

- Note
  - ISR은 짧아야 함

    ISR은 interrupt를 처리하는 시간을 의미한다.

    이게 길면 다른 interrupt를 처리 못한다.
    - 너무 길면 다른 인터럽트들이 제시간에 처리되지 못함
  - Timesharing역시 timer interrupt의 도움으로 가능하게 된 기술

    Timesharing에서 어떻게 시간이 지났는지 아는가? → 외부의 타이머가 interrupt를 보낸다. 그렇게 보내면, 수행하는 작업을 멈추고 OS가 잡고, 다른 Process에게 CPU를 넘김

Q. Time slicing인 경우에는 다시 시작되는 프로그램이 다른 프로그램이 되는 건가요?

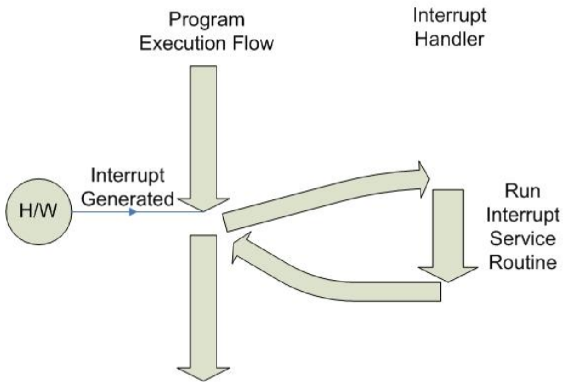
A. 그럴 수도 있고 아닐 수도 있다.

A를 수행하다가 interrupt하다가 다시 돌아와서 판단하게 된다.

A, B, C의 priority가 있으면 달라질 수도 있고, 아니면 그다음 프로세스인 B가 시작한다.

Interrupt은 주변장치가 만든다.

Interrupt 도식



< Flow of handling interrupt >

이벤트처리기법: Trap

- 동기적인 이벤트를 처리하기 위한 기법
  - 예: 시스템 콜, divide by zero, NULL Pointer

    시스템 콜을 호출을 하면 trap이 발생한다. 이것은 어디에서 발생하는가?

    Trap은 모드를 바꿔주는 것이다. User space에서 kernel space로 바꿔주는 것이다.

    Trap역시 interrupt처럼 하드웨어적인 방법이다.

    Trap을 만들어주는 것은 라이브러리이다.

    정확히 말하자면 C 라이브러리이다.

    lib.c라는 것이 있다. 여기에서 trap을 만들어준다. 이것이 complie될 때 이것이 같이 컴파일 된다.

    시스템 번호는 어떻게 넘어가나?

    시스템 번호는 trap의 번호이다.

open이 2번이라고 할때 이 번호가 trap과 함께 넘어가는 것이다.

trap이 넘어가고 모드를 바꾸면서 그 번호를 넘기는 것이다.

- 동기적 - 현재 수행하고 있는 프로그램에 의해 발생
- Trap handler에 의해 처리
- Trap service routine이 있기 때문에 인터럽트와 유사하지만, 인터럽트와 달리 실행 상태(state)를 저장/복원 하지 않음
  - 트랩은 동기적인 이벤트 이므로

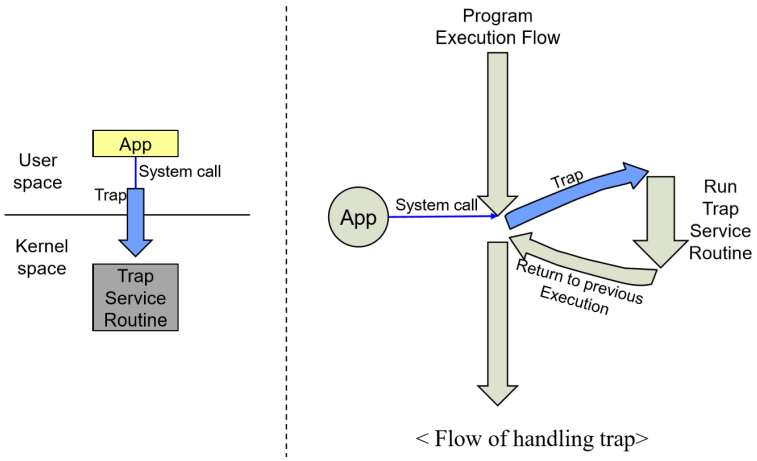
interrupt는 완전히 별도의 무언가를 호출해야 함.

그러나 Trap은 내가 일으키는 것이기에 state를 저장할 필요가 없다.

system call을 발생시키면 program execution을 멈춘다.

우리가 쓰는 모든 CPU는 전부다 어떤 형태의 interrupt와 trap을 제공한다.

Trap 도식



< Flow of handling trap>

system call을 호출하는 함수를 호출한 뒤에 다시 돌아오는 것이다.

Q. 교수님, syscall 은 app이 호출하는것이기 때문에 trap과정도 app을 실행하는 과정에 포함된다고 봐도 되나요??

A. 프로그램 자체에 들어있는 코드가 아닌 다른 코드가 실행되기에 그림이 저렇게 되는 것이다.

Q. divide by zero나 null pointer 참조시에 trap service routine에서는 무슨 일을 수행하는건가요?

A. divide by zero나 null pointer 참조시에 어떤 일이 발생하나? segmentation 오류가 생긴다.

프로그램이 죽는 것이다.

프로그램을 죽이는 것이 trap handler에 들어있다.

프로그램이 이상하다, 그럼 프로그램을 죽이는 것이 TSR(Trap Handler)에서 일어난다.

→ 좋은 질문

Q. interrupt에 대해 찾아본 결과 interrupt가 외부 interrupt와 내부 interrupt로 나뉜다는 글을 봤는데 여기서 내부 interrupt가 trap에 해당하는것인가요?

A. 아마도 그럴 것 같다.

→ 좋아하는 말은 아님.

Q. 그럼 interrupt와 trap의 차이는 프로그램이 멈추는 이유가 app의 외부적요인인지 내부적요인인지에 따라 다르다고 볼 수 있는건가요??

A. 비동기는 외부적, 동기는 내부적 원인이다.

동기적인 이벤트의 대표적인 예시가 syscall이다.

libc가 c를 프로그래밍 할 때 자동으로 컴파일 된다.

libc안에 자동으로 해당 시스템 콜을 받아주는 함수가 있다.

그 함수 안에 들어가면 그 trap을 하는 부분이 있다.

trap을 일으키는 코드도 어셈블리 단계이다.

user → kernel 모드로 변환이 일어나고 kernel의 함수가 실행되는 것이다.

Q. trap 발생시 행동은 libc를 따라간다는거 같은데, 그럼 interrupt 발생시 행동은 어디에 쓰여있나요

A. interrupt는? 어디에도 쓰여있지 않다.

이것 또한 라이브러리에 들어 있는데 어디에도 표현되어 있지 않다.

프로그램이 동작을 하게 되면 runtime과 함께 돌아가게 된다.

프로그램 자체는 interrupt를 인지 못하고 runtime이 인지하고 모드도 변환시킨다. runtime이라는 다른 것이 그것을 사용한다. → 좋은 질문

Q. 교수님 타임 슬라이스가 하나 끝날 때마다 외부의 시계(?)가 cpu에게 알려준다고 하셨는데 인터럽트나 트랩이 발생했을 때는 그 시계도 멈추나요??

A. 시계는 멈추면 안 된다.

시계는 priority와는 관련이 없다.

시간은 안 멈추는데, accounting을 안한다.

ISR이 짧기 때문에 카운팅을 안하고 , 커널로 들어가는 것도 계산을 안한다. 그렇기에 사용자가 받는 시스템의 시간은 안멈춘다. 그렇기에 커널 서비스를 많이 사용하는 프로그램이 유리한 것이다.

시간은 흐르는데, 사용자가 사용하는 시간으로 간주되지 않는다.

→ 좋은 질문

Q. interrupt나 trap routine 도중 trap이 또 다시 발생할 수 있나요?

A. 어려운 문제.

TSR, ISR중간에 interrupt가 발생가능한가?

→ 아마 안 될 것이다.

interrupt 중간에 interrupt가 수행 될 수도 안될 수도 있다.

low priority 에서 higher interrupt가 오면 그 higher를 수행한다.

trap은 interrupt을 발생시킨다. 그래야지 내가 타이핑한 글 같은 것이 화면에 올라온다.

Q. time slice가 끝나기 직전에 interrupt가 발생해서 ISR이 끝나기 전에 time slice가 끝나는 경우에 대해서도 별도의 규칙이 있나요? (ISR 다 마치고 다음 작업으로 스케쥴링 등)

A. ISR은 사용자의 시간으로 카운팅 안한다.

interrupt를 다 마치고 돌아와서 time slice가 남은 것을 수행하고 다른 proccess로 넘어간다.

kernel에서 돌아가는 것은 다 사용자 uncount이다.