[Chapter 12]

소프트웨어융합학과/2020105742/한지훈

1. I/O operation
   A. I/O controller: CPU의 I/O 관련 역할 대신 수행함으로써 abstraction 가능
      I. 특징: 역할 수행을 위해 IR(Instruction Register), DR(Data Register) 등이 필요함
   B. I/O method
      I. Programmed I/O: 기존에 알고 있는, 사용자의 입력 등으로 인한 I/O
      II. Interrupt, DMA(Direct Memory Access): 작업 후에 CPU에게 끝났음을 알리기 위한 방법

2. I/O 방식
   A. Direct I/O: memory 거치지 않고 CPU가 direct하게 I/O 명령을 하는 방식
      - CPU와 I/O controller 간 매핑이 필요
   B. Memory-Mapped I/O: CPU가 memory를 거쳐서 I/O controller에게 I/O 명령을 하는 방식
      - Memory와 I/O controller 간 매핑이 필요
   C. 비교
      I. Instruction 수
         - Direct: I/O 명령도 결국 read/write이기 때문에 CPU에는 memory와 I/O instruction 분리 필요
         - Memory-Mapped: I/O instruction 필요없이 memory에 쓰면 되기 때문에 RISC 구조 CPU에 많이 쓰임
      II. 성능
         - Direct: direct하게 처리를 하기 때문에 성능이 높음
         - Memory-Mapped: memory를 거치고, memory 관리 차원에서 protection 등 필요해서 성능이 낮음
      III. I/O device 많아졌을 때
         - Direct: 많아지면 커버하지 못함(즉 수용 가능 device 수에 제한이 있음)
         - Memory-Mapped: 핀 수보다 memory 주소체계가 많기 때문에 수용 I/O device 수가 더 많음

3. 각종 controller(graphics, memory 등) 존재 이유: CPU가 잡일 하지 않고 주요 업무에 집중할 수 있도록

4. Storage의 I/O 성능, Network Latency
   A. CPU 캐시, DRAM 등: memory controller 따로 있어서 latency 낮지만 비싸기 때문에 capacity 낮음
   B. SSD, HDD 등: latency는 높지만 대규모로 사용할 수 있어서 capacity 높음
   C. Network Latency: network 인터페이스 성능 증가로 점점 감소하는 추세, SAN 등 사용 가능

5. Kernel I/O Structure
   A. 개요: I/O 장치 많을 때, 자원 관리를 OS가 하기 때문에 커널 딴에서 정보 관리 필요
      I. Device controller: HW로, 각 device에 대한 명령을 실행함
      II. Device driver: device의 스펙이 다 다르기 때문에 각 device 제어 정보를 OS 상에 담고 있음
      III. Kernel I/O subsystem(Device-independent I/O Software)
         - 사용 이유: 모든 driver를 OS가 다 관리하기 어렵기 때문에 abstract layer을 만들자
         - 동작 방식: 커널이 간략한 명령을 subsystem에 내리면 subsystem이 driver/controler등 고려해 처리
         - 장점: 분리하는 것이 구현상 편리(depenency 감소, 유지보수 용이)
      IV. Kernel: I/O 관련 처리를 해야하는 상황에서는 Kernel I/O subsystem에 명령을 내림

6. Life Cycle of An I/O Request

유저가 I/O 요청을 통해 system call을 호출하면 커널로 mode change

- Kernel I/O subsystem한테 이미 읽어 온게 있는지 질문
- 이미 읽어온 게 있다면 data를 return하고 system call return
- 이미 읽어온 게 없다면 Device driver에게 명령하고, Device driver는 Device controller에게 명령
- Device controller는 실제 device에 대한 명령을 수행하고, 완료되면 interrupt 발생
- Device driver에 속한 interrupt handler는 interrupt 확인, Device driver는 subsystem에 보고
- 커널 거치지 않고 system call return

7. 새 I/O 동작이 필요하여 구현시 고려사항

A. 바꿀 수 있는 부분: device code/device controller code/device driver code/kernel code/application code

I. 특징: 왼쪽으로 갈수록 HW의 성격, 오른쪽으로 갈수록 SW의 성격

B. HW 구현시:

- 성능 높음, 비용 높음(유지/설비 등 고려 필요),
- abstraction 잘됨(사용자는 구체적인 기능 알 필요 X), 가변성(flexibility) 낮음

C. SW 구현시:

- 성능 낮음, 비용 낮음
- abstraction 잘 안됨, 가변성(flexibility) 높음(code로 변화 주기 쉬움)

8. Goals of I/O Software

A. Device independence: device 간에 dependency가 있으면 안됨

B. Uniform naming: 이해할 수 있도록 통일된, 직관적인 이름으로 짓기

C. Error handling: I/O 처리 시 발생 가능한 오류에 대한 대처 필요

D. Synchronous vs. asynchronous: 둘다, 혹은 한쪽만 지원할지에 대한 고민

E. Buffering: 유실되지 않도록 전송시 밀려있는 data를 보관하는 방법

F. Sharable vs. dedicated devices: share 가능하지 않은 device에서 발생할 수 있는 문제 해결

9. Buffering

A. 개요: ex) 이미지 받아올 때 (data 들어오는 속도 > 처리속도)가 되면 loss가 발생하고, data가 유실됨

- Data는 1bit만 오류 생겨도 전체를 쓰지 못하기 때문에 버퍼링 작업이 중요함

B. 방식(buffering 처리 장소)

I. Buffered in user space: 커널이 관여 X -> 신뢰성 보장 X, but 성능 높음

II. Buffered in the kernel space: 커널이 관여하기 때문에 신뢰도, 관리 차원에서 좋고, 가장 많이 사용

III. Double buffering in the kernel: 이중 버퍼링을 통해 신뢰성을 더욱 높임

10. Error reporting

A. 개요: device error가 났을 때 처리하는 여러가지 방법

B. 종류

I. Returning the system call with an error code: 애플리케이션 level에서 error 처리하는 방식

II. Retrying a certain number of times: 몇번 재시도하는 방식 ex) 재전송 알고리즘(ARQ)

III. Ignoring the error: error을 그냥 무시하는 방식

IV. Killing the calling proces: I/O 요청을 한 프로세스를 kill하는 방식

V. Terminating the system: system off

C. 특징: 정답 X, 다양한 Case가 있기 때문에 상황에 맞는 방식을 사용

11. I/O Software
    A. Share 불가한 device들: 이미 open되어있으면 open() 막음
    B. Block size: 표준은 4096 byte이지만 상황에 따라 조절할 수 있음
        I. GPU: 블록 size를 크게 함 <-> 키보드 입력 시: 블록 size를 작게 함
    C. System call: 보통 user 직접제공 X, 관련 기능들을 fopen() 등의 라이브러리 함수로 제공
    D. Spooling: 프린터 출력 등 동시에 할 수 없고, 순서가 중요한 경우애 사용하는 buffering