1장 (Introduction)

- OS는 무엇인가? : 유저(사람 or 응용프로그램)와 하드웨어(CPU, I/O, 메모리 등) 사이의 중간 역할

- OS의 목적

1. 유저 프로그램을 실행, 유저의 문제를 쉽게 해결하도록 만드는 것

2. 컴퓨터 시스템을 사용하기 편리하게 만드는 것

3. 하드웨어를 (자원 면에서)효율적인 방식으로 사용하도록 하는 것

- 컴퓨터 시스템의 구조

1. H/W : CPU, memory, I/O devices

2. OS : 다양한 응용프로그램과 유저 사이의 H/W 제어, 관리, 통제 (편리, 효율 중요)

3. 시스템, 응용 프로그램 : 유저의 Computing Problems 해결하기 위한 자원의 사용을 정의하는 프로그램

4. Users : 사람 또는 다른 컴퓨터

- 관점에 따른 OS 역할

1. 유저 : 사용의 편의성, 편리성 (자원 활용은 몰라도 됨)

2. Shared Computer : 미니 컴퓨터, 미니 프레임(은행 전산) 등은 하나의 H/W에 대해 사용자 수가 많으므로 자원 효율성까지도 중요함

3. 시스템 : 서버로부터 자원을 공유, Workstations 등은 자원 효율성 필요

4. 스마트폰 : H/W 사양이 낮으므로 자원 효율성 더욱 중요

5. 임베디드 : 사용자 인터페이스가 존재하지 않는 컴퓨터도 있음. OS가 알아서 잘 해줘야 함

🡪 정리 : 용도, 사용 주체(관점)에 따라 중요도의 기준이 다를 수 있음

🡪 편리성, 효율성은 함께 만족시키기 어렵다.

- OS의 정의

1. 자원 할당자 : 여러 요청들의 충돌, 효율적이고 공평한 사용을 위한 자원 할당

2. 제어 프로그램 : H/W의 편리, 효율적인 사용을 위한 S/W

🡪 정리 : 컴퓨터가 동작하는 동안 항상 메모리에 존재하는 하나의 프로그램이 OS(Kernel)

- 컴퓨터 시스템의 동작

1. CPU, I/O devices는 서로 동시에, 독립적으로 동작. 서로 하는 일을 알 필요가 없다. 둘 다 메모리와의 관계는 중요하다.

🡪 Cycle Stealing : I/O와 CPU가 동시에 메모리에 접근하려 할 때, I/O가 메모리에 대한 우선권을 갖는 것

2. 모든 I/O devices는 Device Controller를 갖고, 그 안에는 어느 정도의 메모리가 있다. (Local Buffer)

3. I/O는 Device Controller와 Data를 주고 받고, 결국 Device Controller가 CPU에 인터럽트를 보낸다.

🡪 발생한 인터럽트를 OS가 처리한다.

4. CPU는 Device Controller의 Local Buffer와 메모리 사이에서 Data를 이동시킨다.

🡪 인터럽트 발생 시 하던 일을 멈추고 인터럽트 동작을 처리해야 한다.

- 인터럽트 기능

1. 인터럽트 벡터 : 인터럽트 도착 시 종류에 따라 적절히 실행해야 할 함수 포인터(OS 코드)를 갖는 표

🡪 어디서 온 인터럽트인지 알 수 있다.

2. 인터럽트 아키텍처는 인터럽트 발생 시에 실행을 멈췄던 명령을 기억하도록 설계되었다.

🡪 인터럽트 처리 후 다시 실행해야 하니까(복귀 주소 저장)

3. Trap, Exception : error 또는 user 요청에 의해 CPU가 자가 발생시킨 S/W 인터럽트

🡪 정리 : OS의 동작은 인터럽트에 의해 구동된다. (원동력)

- DMA (Direct Memory Access)

1. 메모리 버퍼, 포인터, 카운터를 사용하여 Device Controller가 CPU의 도움 없이 DMA컨트롤러를 이용하여 데이터를 직접 메모리로 전송하는 입출력 방식

🡪 CPU의 주업무인 프로그램 실행이기 때문에 “인터럽트 오버헤드” 방지하는 다량, 고속 I/O에 적합한 방법

2. CPU는 상태정보, 제어정보만을 교환하고 데이터 전송은 I/O와 메모리 간의 직접 교환

3. DMA컨트롤러가 버스를 제어하고 I/O와 메모리가 정보를 직접 전송

4. CPU에게 PIO작업을 할당하지 않고 DMA라는 특수 프로세서에게 위임하여 메모리와 직접 데이터 전송할 수 있도록 하는 방법

5. DMA의 동작은 Cycle Stealing을 통해 이루어짐. 고속의 CPU 🡪 저속의 DMA에게 BUS의 사용을 허가하여 고속 I/O가 가능하도록 함.

- 채널(Channel)

1. I/O 작업만을 전담하는 한대의 컴퓨터 또는 처리기

2. 보조 CPU : 커다란 컴퓨터 옆에 조그마한 컴퓨터를 장착하여 조그마한 컴퓨터로 입출력 작업을 전담하는 방식으로 사용하기 위해 만들어진 입출력 통로로 활용 🡪 입출력 전담하는 처리기로 점차 변화

3. DMA와 채널의 공통점 : 공통점은 입출력 작업을 원활하게 수행하기 위하여 사용되는 개념이라는 것

4. DMA와 채널의 차이점 : DMA의 경우 주변장치에 장착된 DMA 제어기에 의해 입출력 작업이 수행되고 채널은 입출력 작업 전담 컴퓨터 또는 처리기에 의해 수행

- 메인 메모리

1. 오직 CPU만이 메인 메모리에 접근할 수 있다.

2. Address를 가지므로 Random Access 방식이다. (순차 접근과 반대)

3. 휘발성이다.

🡪 속도 : 레지스터 > 캐시 > 주기억장치 > 보조기억장치

- 저장 시스템의 구분 기준

1. 접근 속도

2. 가격

3. 휘발성 유무

4. 용량

5. CPU 직접/간접 접근 : Secondary 부터는 메인메모리로 가져와서 간접 접근해야 함.

- 캐싱

1. 캐시의 데이터는 일반적으로 RAM(Random Access Memory)과 같이 빠르게 액세스할 수 있는 하드웨어에 저장됨

2. 장치 간의 속도 차 극복 : 캐시의 주요 목적은 더 느린 기본 스토리지 계층에 액세스해야 하는 필요를 줄임으로써 데이터 검색 성능을 향상하는 것입니다. (한 번 사용된 데이터를 캐시에 저장해뒀다가 다음 번에 빠르게 사용하기 위함)

3. 캐시는 CPU와 RAM사이에서 중간 저장소 역할을 하는 메모리

4. CPU에 내장 되어 있다고 표현하기도 하는데, 캐시 메모리는 원래 CPU의 일부로 존재하는 메모리 개념이 아니다. CPU에 근접해 있는 메모리 개념이다.

🡪 캐시의 단점

1. 캐시로 인해 복잡해지는 부분이 존재한다.

2. 캐시 일관성 문제(Consistency가 깨질 수 있음) : 공유 메모리 시스템에서 클라이언트 1, 2가 각각 캐시를 가진 상태에서 클라이언트 1이 데이터를 수정하게 되면, 클라이언트 2는 다시 이 데이터를 읽을 때 캐시에 있는 다른 값을 사용하게 되는 문제



3. 최신 값이 갱신되다가 CPU가 꺼지는 경우, 캐시와 메인메모리의 내용이 달라지게 된다. (Consistency가 깨짐)

- 멀티 프로세서(Multiprocessors)

1. 한 컴퓨터 내에 공통의 메모리를 사용하고 CPU가 여러 개 존재, 각 CPU는 레지스터와 캐시를 가짐

🡪 “프로세스” 개념과 다르기 때문에, 프로세서는 하나의 CPU를 갖는 것을 말함.

2. 병렬 시스템(Parallel Systems), 강 결합 시스템(Tightly-coupled System)이라고도 알려져 있다.

3. 멀티 프로세서의 장점

🡪 처리율 : 한 컴퓨터에 CPU가 여러 대인 병렬 시스템이므로 성능이 높아진다. But, n배 증가했다고 해서 성능이 n배가 되지는 않는다.

🡪 이유 : CPU가 늘어난 만큼 복잡도가 증가하기 때문, 각 CPU마다 캐시를 가지므로 메모리의 원본 Data의 수정에 대해 복사본 n개의 일관성을 맞춰야 하기 때문

🡪 규모 경제적(크기가 작아지니까)

🡪 의존성 증가 : 성능 저하, 내부 결함에 대해 유리

🡪 이유 : CPU 하나가 고장 나더라도 다른 CPU가 존재하기 때문에 성능 저하가 적은 편

4. 멀티 프로세서의 두 가지 종류

🡪 Symmetric Multiprocessing (대칭 멀티프로세서) : 하나의 컴퓨터 내에서 메모리 자원을 공유하며 각 CPU는 레지스터와 캐시를 따로 갖는다. 일관성(동기화)을 맞춰야 할 필요가 있다. 여러 프로세서가 프로그램을 수행한다.

🡪 asymmetric Multiprocessing (비대칭 멀티프로세서) : 각 프로세서가 각자 특정된 일을 맡아서 한다. 주 프로세서가 전체 시스템을 통제하고 관리한다. 종 프로세서들을 스케줄링한다.

- Dual-Core Design : 멀티 코어는 CPU 내에 n개의 코어가 있는 것으로 n개의 CPU를 갖는 멀티 프로세서와 다르다.

- Clustered Systems

1. 멀티 프로세서와 비슷하지만, 여러 대의 PC를 한 대의 큰 컴퓨터처럼 연결하여 사용하는 것(디스크도 공유)

2. Storage-Area Network (SAN) 을 통해 저장소를 공유한다.

3. 두 가지 종류

🡪 Symmetric Clustering (대칭 클러스터) : 하나의 작업을 1/n씩 나눠서 수행하는 것

🡪 Asymmetric Clustering (비대칭 클러스터) : 한 대 일할 때, 나머지는 쉬면서 대기하는 것

- 멀티 프로그래밍(Multiprogramming)

1. 특정 프로세스에 대해서 프로세서가 작업을 처리할 때 낭비되는 시간 동안 다른 프로세스를 처리하도록 하는 것

Ex) A라는 프로세스를 처리 중에 있을 때 입출력 이벤트가 발생하여 프로세서가 입출력 이벤트에 대한 응답을 위해 무작정 대기하고 있다면 프로세서의 자원을 낭비하는 결과를 초래한다.

🡪 멀티 프로그래밍은 낭비되는 시간 동안 프로세서가 다른 프로세스를 수행할 수 있도록 하는 것을 말한다.

🡪 장점 : CPU를 100% 활용. Throughput, 생산성이 높다.

🡪 문제점 : 각 프로세스 마다의 CPU 사용 시간을 OS가 알 수 없다. 하나의 프로세스가 CPU를 장시간 독점하게 되면 다른 프로세스를 사용하는 유저는 느리다고 느낄 수 있다. (유저가 1명이면 다른 프로그램이 버벅거리겠지)

- 멀티 태스킹(Multitasking)

1. 다수의 Task(프로세스보다 확장된 개념)를 OS의 스케줄링에 의해 번갈아 가면서 수행하는 것

2. 프로세서가 각각의 Task(작은 개념으로는 프로세스를 말하기도 함) 를 조금씩 자주 번갈아 가면서 처리하기 때문에 사용자는 마치 동시에 여러 Task가 수행되는 것처럼 느끼게 된다.

🡪 멀티 프로그래밍과의 공통점 : 모든 프로세스가 일단 메모리에 모두 올라가 있어야 한다. 🡪 메모리가 부족할 수 있다는 단점이 존재한다.

🡪 멀티 프로그래밍과의 차이점 : 멀티 프로그래밍은 프로세서의 자원이 낭비되는 것(I/O 이벤트로 인해 CPU가 대기하는 것)을 최소화하기 위해 다른 프로세스를 실행시키는 것을 말하며, 멀티 태스킹은 일정하게 정해진 시간(Time Sharing) 동안 번갈아 가면서 각각의 Task 를 처리하는 것

🡪 만약 정해진 시간이 되지 않았는데 I/O 이벤트 발생하면 자진 반납

- 다른 Task(또는 프로세스)로의 전환은 OS가 개입하게 된다. 두 가지가 존재

1. 시간 제한에 의한 ‘커널 모드’(but, 상당히 빠르게 다시 유저 모드로 전환됨)

2. 자발적 System Call에 의한 ‘커널 모드’

🡪 프로세스의 실행은 ‘유저 모드’라고 할 수 있다.

- Dual Mode : OS는 1. 커널 모드 와 2. 유저 모드로 나뉘어 구동된다.

- OS의 주요 기능 중 보호와 보안

1. 보호(Protection) : 다른 사용자 또는 프로세스로부터 자원의 사용을 제한하는 것

2. 보안(Security) : 다른 시스템으로부터 오는 공격 등을 막음