2강

컴퓨터의 기원

- 전쟁에서 필요한 값을 계산하는 것을 컴퓨터가 대신 하게 만드는 것이다.
- 튜링의 머신이 애니그마를 해독하여 전황을 바꾸었다.

1950년대 초반

- 당시 컴퓨터는 현재의 컴퓨터에 비해 매우 원시적임
- 프로그램은 기계적인 스위치를 이용하여 1bit 단위로 컴퓨터에 입력되어 실행
    - 진공관 기반 - 집채만 한 크기, 많은 열 방출

당시에는 지구상에 10개만 필요하리라 생각되어짐

1960년대 초반

- 모든 프로그램은 기계어로 쓰여짐. → instruction을 사용하고, 저장 장치가 없었기에 프로그램을 매번 입력해야 했음. 뜨거워지면 꺼야했음.
- 플러그 보드 (Plug-board)에 와이어링(wiring)을 통해 컴퓨터의 기능을 제어

플러그 보드는 일종의 키보드 같은 것이다.

- 프로그래밍 언어 및 운영체제라는 존재가 없음
    - 영속적인 저장장치가 없음 - 매번 프로그램 다시 입력

1960년대 중반

- Punch Card가 등장, 카드 하나당 문장하나임
- 프로그래밍한 카드로 컴퓨터 구동
    - 플러그 보드 대체

Mainframe - 일괄처리 (Batch)

Bussiness machinery로써 쓰이면서 가치가 발생

- 계산을 하는데 주로 사용되기 시작함
- 교량(Bridge) 설계

일괄 처리 - 아주 단순한 OS개념

- 일단 시작한 "job"은 끝나야 다음 job이 수행됨
- Punch card를 재출하면, 메모질에 적재, 수행의 순서로 진행
- 결과를 받기까지 중간에 NO User Interaction
- 사람이 job을 scheduling

→ 빨리 결과를 받고 싶으면 담당자에게 커피를 사줘라

CPU는 빈번히 idle 상태로 전환됨

- 기계적인 I/O 장치와 전기 장치인 CPU사이에 현격한 속도 차가 존재

Automatic job sequencing – 좀 더 나은 OS

사람의 관여 없이 여러 개의 프로그램을 컴퓨터에서 순차적으로 실행

이전 작업이 종료되자 마자 다음 작업을 실행하기 때문에, 일괄처리 (batch) 보다 성능이 향상됨

- 일괄처리

  → 사람이 직접 스케줄링
- Automatic job sequencing

  → 스케줄링을 담당하는 소프트웨어에서 프로그램 실행

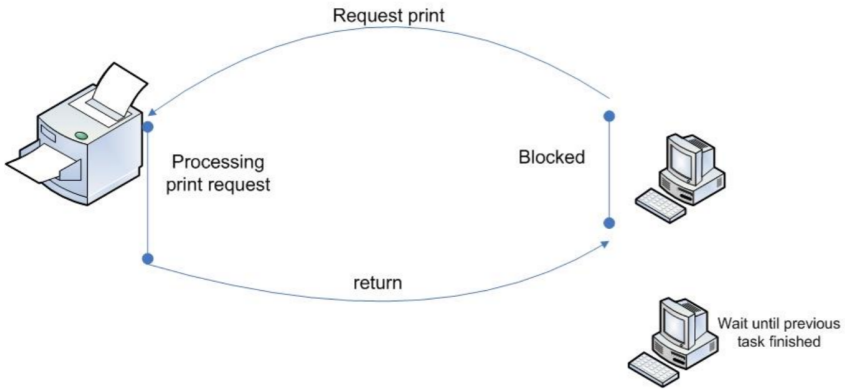
I/O에 의해 CPU가 유휴 상태로 전환되는 문제 해결 X

뭘 출력하려면 CPU가 기다려야 했음.

문제점 - I/O in the middle of job

게다가 프린터도 느렸음

문제점 - I/O in the middle of job



Spooling

Simultaneous Peripheral Operation On-Line

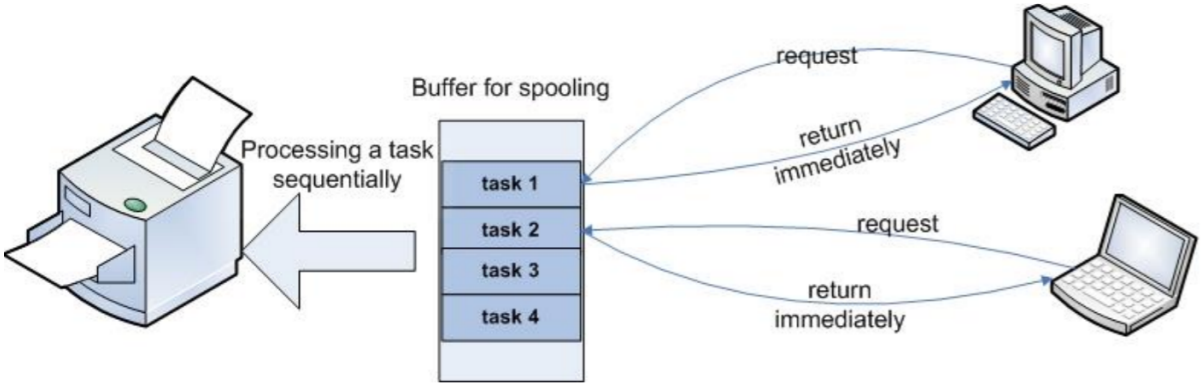
I/O와 computeation을 동시에 진행할 수 있음.

- 인쇄할 문서를 디스트나 메모리에 로드, 저장(데이터를 복사하는 것임)
- 프린터는 버퍼에서 자신의 처리 속도로 인쇄할 데이터를 가져옴
- 프린터가 인쇄하는 동안 컴퓨터는 다른 작업을 수행 할 수 있음

Spooling을 통해서 사용자는 여러 개의 인쇄 작업을 프린터에 순차
적으로 요청할 수 있음.

- 이전 작업의 종료를 기다리지 않고, 버퍼에 인쇄 작업을 로드하여 자신의 인쇄 작업을 요청함

spooling을 하게 되면 CPU가 I/O처리를 하지 않게 된다.



인쇄할 task를 복사하여 Buffer for spooling에 넣어둠

Spooling을 통해서 사용자는 여러 개의 인쇄 작업을 프린터에 순차적으로 요청할 수 있음.

- 이전 작업의 종료를 기다리지 않고, 버퍼에 인쇄 작업을 로드하여 자신의 인쇄 작업을 요청함

Multiprogramming

메모리에 여러개의 일을 올려놓는 것이다.

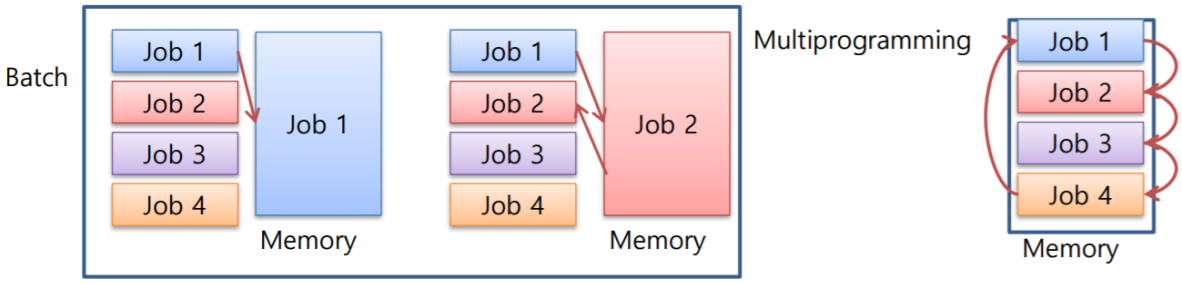
중간에 프로그램이 멈춰야 할 때가 있다.

예를 들어서 C언어 I/O를 할 때 job을 바꾼다.

2개 이상의 작업을 동시에 실행

- 운영체제는 여러 개의 작업을 메모리에 동시에 유지
- 현재 실행중인 작업이 I/O를 할 경우 다음 작업을 순차적 실행
- 스케줄링 고려사항 – first-come, first-served

메모리에 여러개를 올려놓고, I/O를 하게 되면 다른 job을 하게 된다.



Multiprogramming의 목적

- CPU 활용도(utilization) 증가

CPU 활용도는 100%까지 높일 수 있다.

job을 메모리로 가져오는 시간에도 CPU는 일을 못하기에 이를 막아준다.

이 자주자주 바뀌는 Multiprogramming을 담당하는 프로그램이 필요해짐

Multiprogramming의 의미? con-current이다.

→ 어떤 순간에 보면 job1, job2, job3, jo4가 다 수행되는 상태이다.

→ I/O로 인해 넘어가면 동시에 다 수행하는 것이다.

→ 여러개의 job들이 동시에 수행되는 상태가 된다. 물론 partialy이다.

→ 비록 한 순간에는 하나만 수행되지만, 전체적으로 보았을때는 동시에 전부 수행되는 것이다.

단점

- 사용자는 여전히 실행중인 작업에 대해서는 관여할 수 없음

사용자는 job1이 수행되고 있다면, 실행되는 작업이 끝날때까지 사용자는 관여가 불가능

만일 jo4가 무한 루프가 돌고 있다면 할 수 있는게 없음 운영체제도 CPU를 할당받아야 끊는 것이 가능

Issues with multiprogramming

다른 job이 수행되기 위해서는 현재 수행되는 job이 I/O를 해야 함

→ voluntary yield 에 의존 (자발적 양도에 의존.)

→ 의도적으로 I/O 안함 (이기적인 사람들)

공평성을 유지할 필요 발생

누구나 컴퓨터를 오래 많이 쓰고 싶어한다

당시에는 하나의 컴퓨터를 몇백명의 사람들이 사용했다.

High priority로 수행할 필요도 생김

어떤 job들은 빨리 해결해야 하더라

여기서 priority개념은 없음.

Job scheduling으로는 해결 안됨

multiprogramming?

1. 여러개의 프로그램을 메모리에 올려놓고 사용한다.
2. 그 프로그램들이 I/O를 할 때 점유하고 있던 CPU를 다른 프로그램에게 양도한다.

→ 이걸 꼭 써줘야한다.

Timesharing

이때부터 진정한 의미의 OS가 생기는 것임

CPU의 실행 시간을 타임 슬라이스(time slice)로 나누어 실행

- 10ms (mill-seconds, 밀리초)

타임 슬라이스(time slice)가 그 단위이다.

지금은 10ms보다 더 짧아졌다.

모든 프로그램은 타임 슬라이스 동안 CPU를 점유하고, 그 시간이 끝나면 CPU를 양보(relinquish)

어떤 프로그램이든지 10ms가 지나면 강제로 예외없이 넘겨버린다.

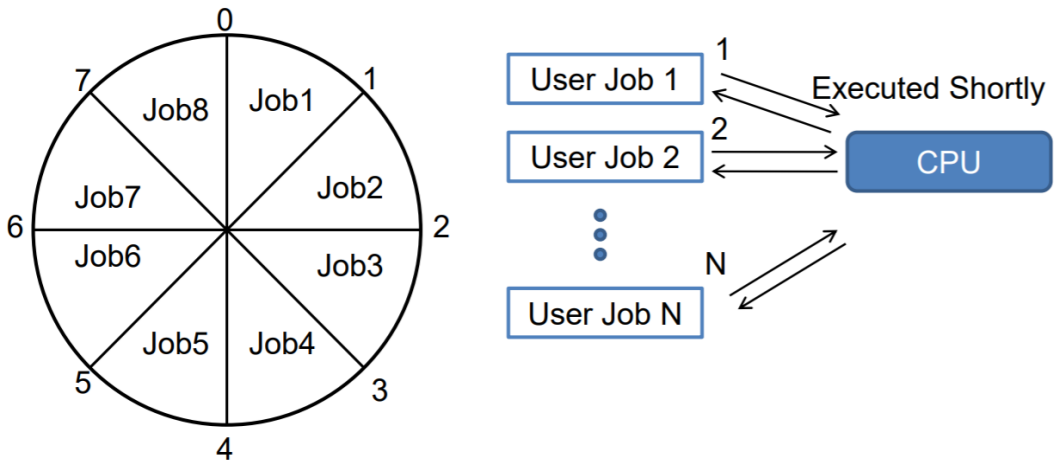
여러 개의 작업들은 CPU 스위칭을 통해서 동시에 실행됨

CPU스위칭이 매우 빈번하게 일어남

- 사용자는 실행중인 프로그램에 관여(interact)가 가능

최소한 1초에 100회 이상의 스위칭이 생김

→ 즉 그 중간에 내가 입력을하고 10ms가 지나면 os가 개입을 하는데 그때 그 os가 스위칭을 하는 것이다.



범용컴퓨터는 다 이렇게 한다.

그러나 슈퍼컴퓨터들은 이 Time Sharing을 사용하지 않는다.

왜냐하면 Time Sharing은 OS가 가져가는 CPU가 overhead가 된다.

OS는 job이 아닌, user를 위한 것이기에 overhead가 된다.

슈퍼컴퓨터들은 오히려 batch를 사용한다.

→ 용도에 따라서 다른 방식을 사용하는 것이다. 무조건 Time Sharing이 좋은 것은 아니다.

전기자동차는 어떤 방식이 좋을까?

→ Time Sharing은 이러이런 특성이 있는데, 전기자동차는 이러이러한 특징이 있다. 그래서 적절하다/적절하기 않다.

이렇게 답변하라.

예) 안전과 관련한 기능이 가장 중요하기 때문에 High priority를 다룰 수 있는 Time sharing이 좋을 것 같습니다.

이전 스마트폰에서는 OS를 넣지 않았는데, 왜냐하면 그 크기가 너무 컸기 때문이다.

Time Sharing에서 강제로 점유권을 가져가면 프로그램 동작에 영향이 안가는가?

→ 멈출때 그 state를 저장한다. 그리고 다시 실행될 때 그걸 불러온다.

→ 즉 스위칭할때마다 그 state를 다 저장해야 한다.

Time Sharing에서 cpu 스위칭이 일어날 때 os도 타임 슬라이스만큼 cpu를 점유하는가?

→ 절대 안 된다. OS 차지하는 시간을 최대한 줄이는 것이 중요하다.

타임슬라이스는 10ms가 굳이 아닐 수도 있다.

상대적인 시간이다.

OS가 10ms 시간이 지났는지 재고 있는가?

타임 슬라이스가 지났는지 어떻게 알 수 있는가?

그 안에 끝나면 어떻게 되는가?

→ 시간을 버리는 방법이 있고, 흠...

Time Sharing에서 I/O가 일어나면 이 job이 제외되고 스위칭된다.

→ yes

Time Sharing에서 priority가 높으면 연속적으로 연속적으로 작업하게 되는가?

→ 이렇게 될 수 있는데, 흠.. 어느게 좋을까? 다른 프로그램이 수행되지 못한다면 그게 좋은 것인가?

→ Policy의 문제이다. 설계의 문제이다. 이것은 design decision이다.

Time Sharing에서 OS가 state를 기록하는데, 어디에 기록하는가?

→ yes, 메모리와 스토리지 두 가지에 저장이 가능한데, 메모리에 일반적으로 저장된다. 기본적으로 메모리이다.

멀티프로그램은 노래들으면서 카톡을 못한다

job의 priority는 os가 결정하거나 job자체에 할당되어 있다.

다음 수업시간까지 윈도우가 차지하는 DISK의 크기와 메모리를 확인해와라

concurrent 수행

- 여러 프로그램들이 동시에 수행 상태에 있는 것
- 멀티프로그래밍 도는 time sharing을 할 때 가능
- 어느 시간 t에서 보면, 하나의 프로그램만 CPU에서 수행되고 있음

Time quantum == time slice

Time sharing을 하면 CPU 스위칭이 발생하면

- 프로그램의 state를 저장해 놓아야 함
- 이 state를 사용하여 CPU를 다시 받아서 수행할 수 있음

운영체제가 상당한 부분의 ram과 disk를 차지하더라

Multitasking

두가지 의미이다.

1. 여러 개의 태스크(task)들이 CPU와 같은 자원을 공유하도록 하는 방법

2. 하나의 작업(job) 은 동시에 실행할 수 있는 태스크로 나뉘어 질 수 있음

   예) 유닉스의 프로세스는 fork()시스템 콜을 이용해서 여러 개의 자식 프로세스를 생성할 수 있음

Multitasking은사용자가 여러 개의 프로그램을 실행 할 수 있도록 하며,

CPU가 유휴 상태일 때는 background 작업을 실행 가능하도록 함

Example of DBMS Multitasking

Issues with multitasking system

복잡한 메모리 관리 시스템

- 동시에 여러 개의 프로그램이 메모리에 상주됨 (A, B, C가 상주함. B가 끝나고 훨씬 큰 D가 올라가지 못한다.)
- 메모리 관리 및 보호 시스템 필요

적절한 응답 시간을 제공

큰 job을 넣을꺼면 기존 job을 내려 놓아야 한다. 그 위치가 swap space이다.

- Job들은 메모리와 디스크로 swap in/out 될 수 있음

Concurrent execution 제공

- CPU 스케줄링이 필요

→ 프로세스 간의 수행순서를 정해야 할 필요성이 있다. - policy 구현이다.

순서를 맞춰서 해야할 필요도 있다.

필요에 따라서 job들간의 orderly execution 이 필요

- 동기화, deadlock

→ 동기화가 필요.

Lineage of well-known operating systems

Multics를 보면 지금에 와서야 사용가능한 기법들이 많이 보였다. 그러나 그 당시의 하드웨어는 그 구현이 거의 불가능하였다.

UNIX : Multics는 여러개고 UNI는 하나가 아닌가? UNIX는 훨씬 단순하고 놀랍게 성공했다.

이때의 가장 큰 차이점은 networking이 가능했다는 것이다. 그것이 BSD이다. 그리고 그 소스코드를 다 공개해버렸다.

그런데 UNIX는 PC로 포팅이 안 되었다. 그리고 UNIX를 사는 사람들은 WINDOW와 INTEL을 배척했다.

그래서 누군가가 이 UNIX를 PC에 올리는데, 그것이 MINIX이다. 이때부터 유닉스가 PC에서 돌아가기 시작한다. 그리고 Linux는 완전히 소스코드부터 싹 바꿔서, 그렇지만 MINIX의 주요장점을 받아서 개발하기 시작했다.

BSD를 개발하던 사람들이 NEXT로 갔다가 애플로 갔다. 그리고 MAC OS가 탄생한다.

그중 가장 중요한 것이 Berkeley UNIX이다. 에서 SVR4를 개발했으나, 사실 상 소멸했다.

오른쪽 PC는 또 이야기가 다르다.

MS-DOS라는 OS가 있었다. 프로그램 크기가 512KB를 넘으면 죽어버렸다. 마이크로소프트 자체에서 만들어 보자는 것이 아니라, 배낀 것이다. 그것이 Windows 3.0이다. Windows 95는 버그가 너무 자주 발생해서 하루에 한번씩 꺼야했다. 유닉스에 대해서 마음이 불편한 VMS를 만드는 기술자들이 조언을 한다. 그리고 회사가 망한다. (BSD의 오픈소스 정책때문에) 이 사람들이 Windows NT를 만들고 이것을 합쳐서 Windows XP를 만든다.

그외 시스템들

Multiprocessor systems

- Symmetric vs. asymmetric multiprocessor

분산시스템

- LAN/WAN으로 연결
- Client-server model, peer-to-peer model
- 질문: bus와의 차이는? - 숙제 bus과 lan의 차이점이 무엇인가? Multiprocessor와 분산시스템의 차이점이 뭐냐?

Clustered Systems

- 공동의 목적을 위해 여러 개의 시스템 네트워크를 통해 작업을 수행

Multiprocessor systems와 분산시스템의 중간 정도 일 것이다.

이 시스템은 하나의 목적을 가지고 있고, 굉장히 빠르다.

Embedded systems

- 특정 목적을 위한 운영체제 및 소프트웨어가 하드웨어에 탑재된 형태의 시스템
- MP3 player, smart phone, PDAs

전화기, 스마트폰, fit 같은 것들, 특정 목적을 위한 소프트웨어와 하드웨어가 타이트하게 붙어서 돌아가는 시스템

Real-time systems

- 시스템에서 수행하는 작업의 완료 시간(deadline)이 정해짐
- Soft real-time, hard real-time

우주선을 띄운다. 미사일을 발사한다. 그것은 시간이 주어진다. 언제까지 어떤 미션을 해야 한다는 것이다. MP3를 듣는다.

우리는 범용시스템에 들어가는 OS에서 추가적인 구현이 들어간다.

이것은 우리가 다루는 영역 밖의 것임.
