디지털컨버전스 기반 UI/UX Front 전문 개발자양성 과정(비트캠프)

강 사 : 이 승 훈

수 강 생 : 오 진 욱

- Java -

Input / Output
컴퓨터 구조

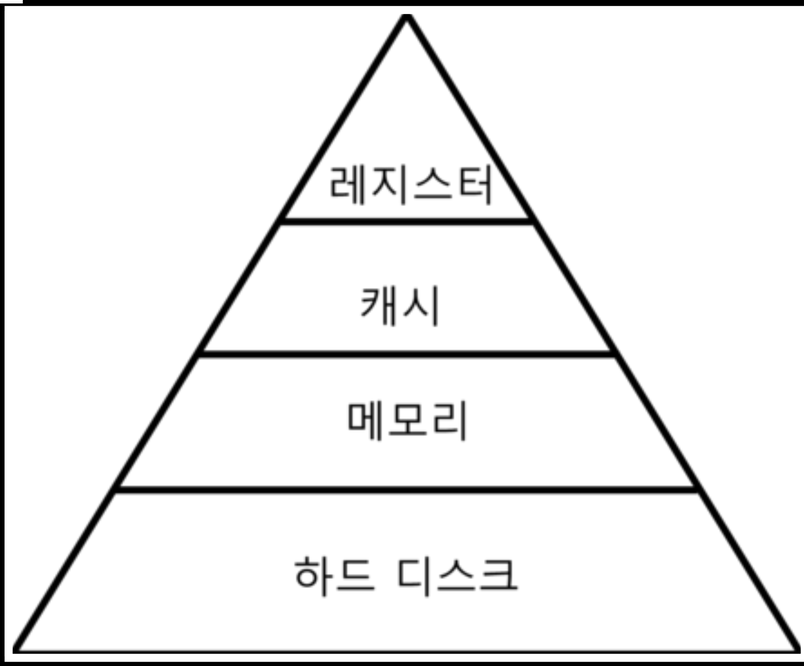
- 목적 달성을 위해 CPU 또는 메모리와 외부 장치간에 정보를 주고 받는 것
- 메모리의 데이터들을 일련의 과정에 걸쳐 인간이 볼 수 있는 형태로 입력과 출력해 주는 것
- 마찬가지로 외부장치와 통신하기 위해 레지스터 필요 —> 이것도 경쟁 발생?
- 과거 프로세스들은 단순 —> CPU 클럭 속도만 향상하면 성능 UP
- 현재 프로세스는 복잡 —> CPU + I/O 퍼포먼스 향상이 엄청난 속도 UP
- —>코드 쪽에서 가장 쉬운 향상 방법은 한번에 모아서 출력하기

Memory Hierarchy
메모리 계층 구조

- 메모리들은 프로세스 동안 데이터의 입력 및 출력을 담당
- CPU와 가까울수록 빠르고 용량이작고 멀수록 느리고 용량이 큼
- 높은 성능을 위해서는 앞서 말한 것처럼 I/O 컨트롤이 중요

Memory Hierarchy
메모리 계층 구조

- 레지스터와 캐시는 CPU 내부에 존재 -> 빠르고 가벼움(용량이 작음)
- 메모리는 CPU 밖에 존재 -> 위보다 느림
- 하드디스크는 CPU에 직접 접근 조차 불가능->메모리 통해 접근
- 용량이 크면 클 수록 속도가 느리다 이를 표현한 것이
- 오른쪽 Memory Hierarchy 이다



Synchronised Issue

Critical Section

- 각 Thread들은 stack / resister만 독립적 , 나머지 가상메모리들은 공유함
- 여러 태스크가 접근 할 수 있는 영역(공유영역)을 Critical Section이라고 함
- 이 공유 자원에 여러 Thread들이 동시에 자원을 사용(Race Condition)하려 드는 문제가 발생함
- 이 부분에서 여러 문제가 발생함
- 이렇듯 동기화가 안되어 있는 상태라면 결과 값은 전혀 예측 불가
- 그렇기 때문에 이 Critical Section에는 무조건 하나의 Thread만 접근을 허용해야함

Synchronised Issue

상호 배제 방법

- 공유 자원(CPU 메모리 할당)관리 방법
- 대표적인 Critical Section 관리 방법
- 이 문제를 해결하기 위해서 한번에 하나의 Thread가 접근할 수 있도록 제한하는 것

Synchronised Issue

상호 배제 방법 - Mutex

- Mutual Exclusion의 약자로 Critical Section 문제를 해결할 수 있는 개발 도구
- 동기화함에 있어 동시에 하나의 Thread만 실행 되게 하는 도구(Kernel객체)
- 프로세스 / 쓰레드 다중 실행 방지
- 1개의 방 (Singled) ㅡ-> 문잠금(Lock( )) ㅡ->기달리면서 빠르게 노크하는 상태(Busy Waiting)

Synchronised Issue

상호 배제 방법 - Semaphore

- Mutex확장버전
- 지정된 수 만큼의 Thread가 동시에 실행되도록 동기화 하는 것
- Dead Lock을 피하기 위해 사용
- 10개의 방(2개이상의 진입 O) —->10개문 잠김—-> 비어진 방을 기달리고 있는 Thread가 사용