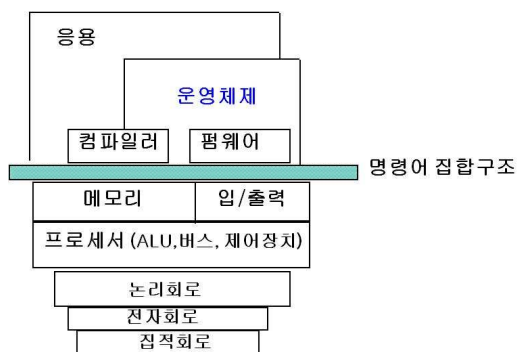


1장. 서론 (Introduction)

순천향대학교 컴퓨터공학과 이 상 정



컴퓨터 구조 복습 1

프로세서는 메모리에 저장된 명령어를
읽고 해석한다

컴퓨터 구조 복습 1: 프로그램 실행

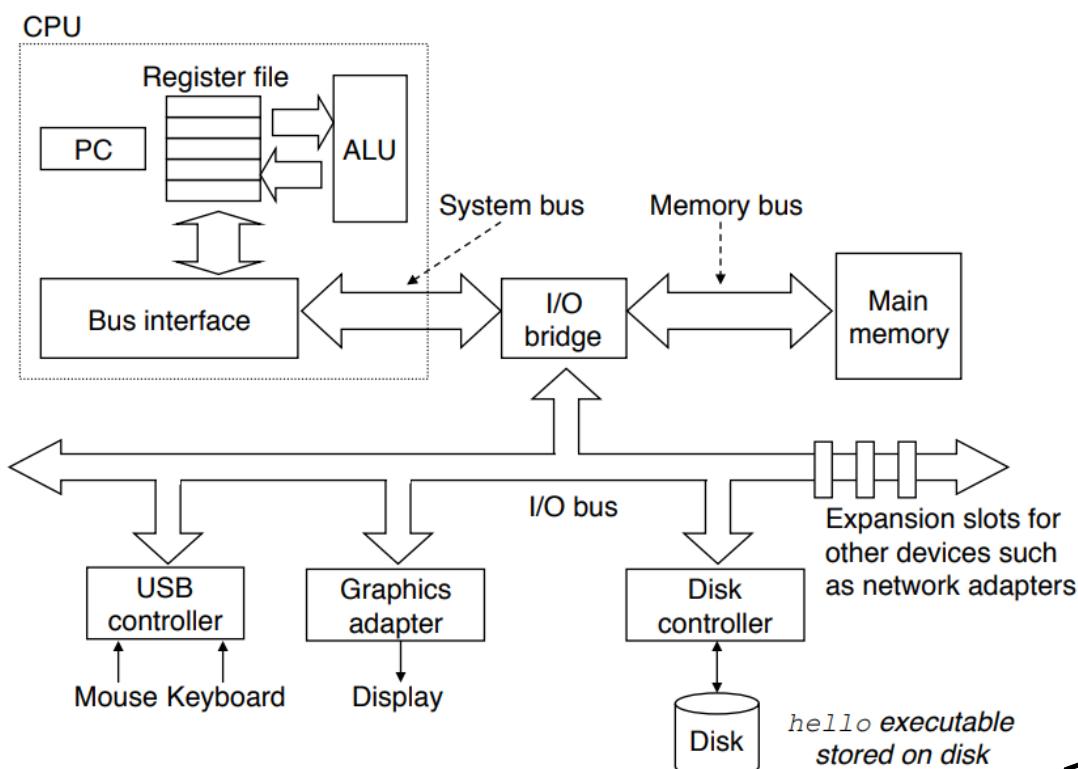
□ 리눅스 시스템에서 실행파일을 수행

- 운영체제가 제공하는 **명령어 해석기(command interpreter)인 셸(shell)** **프로그램**에서 리눅스 명령어 라인을 입력 받아 명령을 실행

```
linux> ./hello
Hello, world
linux>
```

```
linux> ls hello*
hello hello.c hello.o hello.s
linux> ./hello
Hello. world!
linux>
```

컴퓨터 구조 복습 1: 컴퓨터 시스템 하드웨어 구성



<그림 1.4>

컴퓨터 구조 복습 1: 하드웨어 구성 - 버스, 입출력 장치

□ 버스 (Bus)

- 하드웨어 구성 요소들 간의 정보를 전달하는 배선
- 일반적으로 **워드(word)** 단위라는 고정된 크기의 바이트들 단위로 전송
 - 대부분의 현대 컴퓨터 시스템은 4 바이트 (32 비트) 또는 8 바이트 (64 비트)의 워드 크기를 가짐

□ 입출력 장치

- 컴퓨터 시스템과 **외부 장치와의 연결**을 담당
 - 키보드, 디스플레이, 디스크 드라이브 등과 연결
- 입출력 장치는 입출력 버스, **컨트롤러(controller)**, 어댑터(adaptor) 등을 통해 연결
 - 컨트롤러는 디바이스 자체가 칩셋이거나 시스템의 마더보드에 장착
 - 어댑터는 마더보드의 슬롯에 장착되는 카드

컴퓨터 구조 복습 1: 하드웨어 구성 - 메인 메모리, 프로세서

□ 메인 메모리 (Main Memory)

- 실행되는 프로그램의 데이터와 프로그램이 저장
- 물리적으로는 **DRAM** 칩으로 구성되고, 논리적으로는 연속된 바이트의 배열로 고유의 주소(배열의 인덱스)를 가짐

□ 프로세서 (Processor)

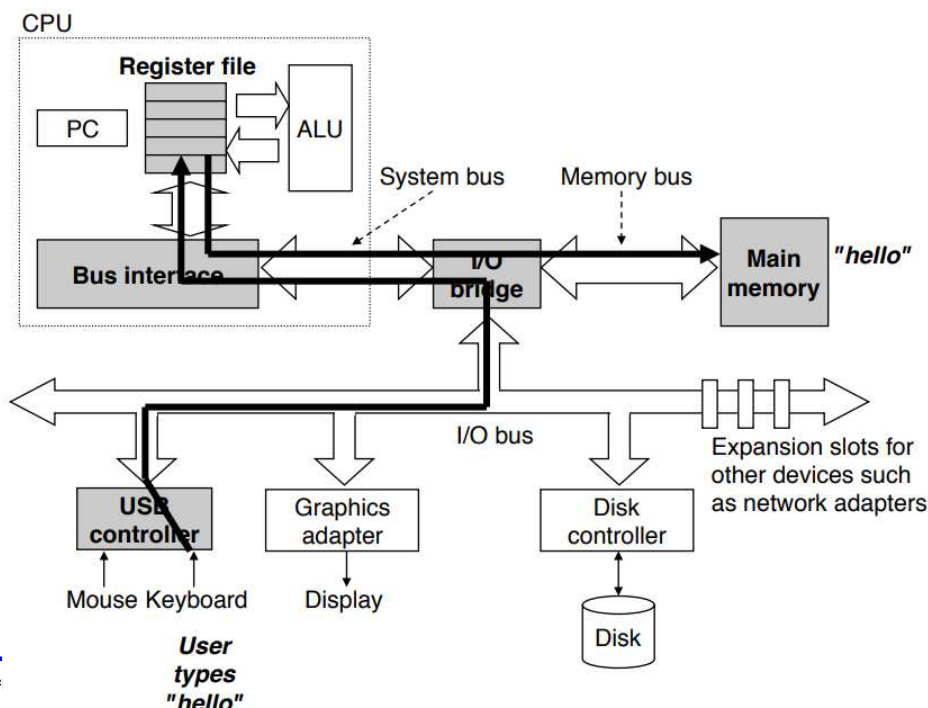
- 중앙처리장치(CPU) 또는 프로세서는 메인 메모리에 저장된 프로그램의 (기계어) 명령어(instruction)를 해독하고 실행
- 프로세서에는 워드 크기의 저장 장치인 레지스터(register) 파일과 현재 실행되는 명령어의 주소를 가리키는 프로그램 카운터(program counter, PC), 산술/논리 연산장치(Arithmetic/Logic Unit, ALU) 등이 있음

컴퓨터 구조 복습 1: 하드웨어 구성 - 프로세서

- 명령어의 요청에 의해 프로세서가 실행하는 작업
 - 적재 (Load)
 - 메인 메모리에서 한 워드 (또는 바이트)를 읽어 레지스터에 저장
 - 저장 (Store)
 - 레지스터에서 메인 메모리로 한 워드 (또는 바이트)를 저장
 - 연산 (Operate)
 - 레지스터에 저장된 데이터를 ALU에서 연산 처리하고 결과를 레지스터에 저장
 - 점프 (Jump)
 - 다음에 실행될 명령어 주소를 지정하기 위해 현재 실행되는 명령어에서 지정된 주소로 PC를 덮어쓰기 함
- 4장에서 프로세서의 구현(마이크로 구조, microarchitecture)을 소개

컴퓨터 구조 복습 1: hello 실행 - 셸 명령 입력

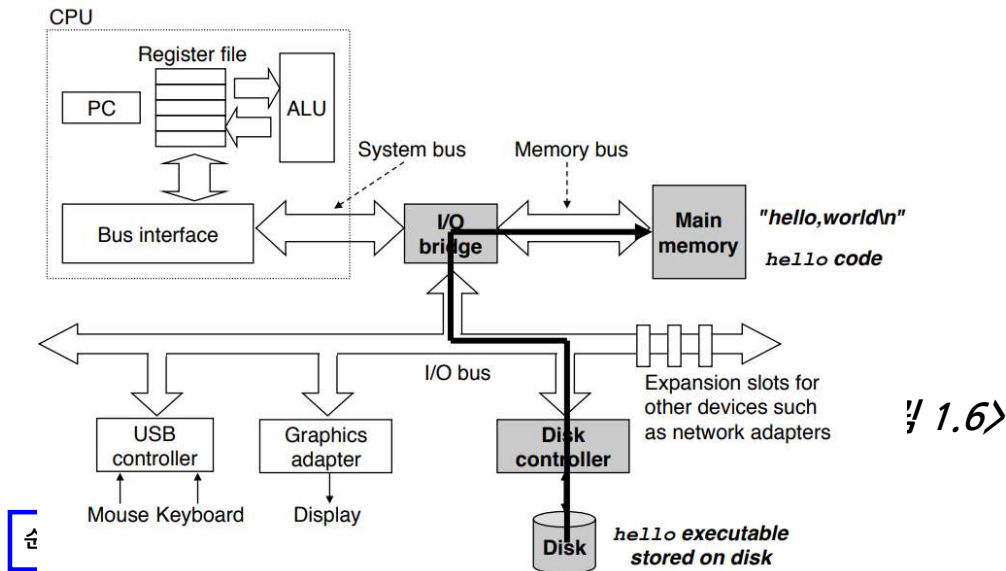
- 셸 프로그램이 “hello” 명령을 읽어 각 문자를 레지스터를 경유하여 메모리에 저장



컴퓨터 구조 복습 1: hello 실행 - 프로그램 로딩

□ 운영체제의 로더(loader)가 쉘에서 입력된 명령을 해석하여 hello 실행파일을 디스크에서 메인 메모리로 로딩

- 프로세서는 직접 메모리 접근(Direct Memory Access, DMA) 기법 - 6장에서 소개 - 으로 프로세서를 거치지 않고 디스크에서 메인 메모리로 직접 이동



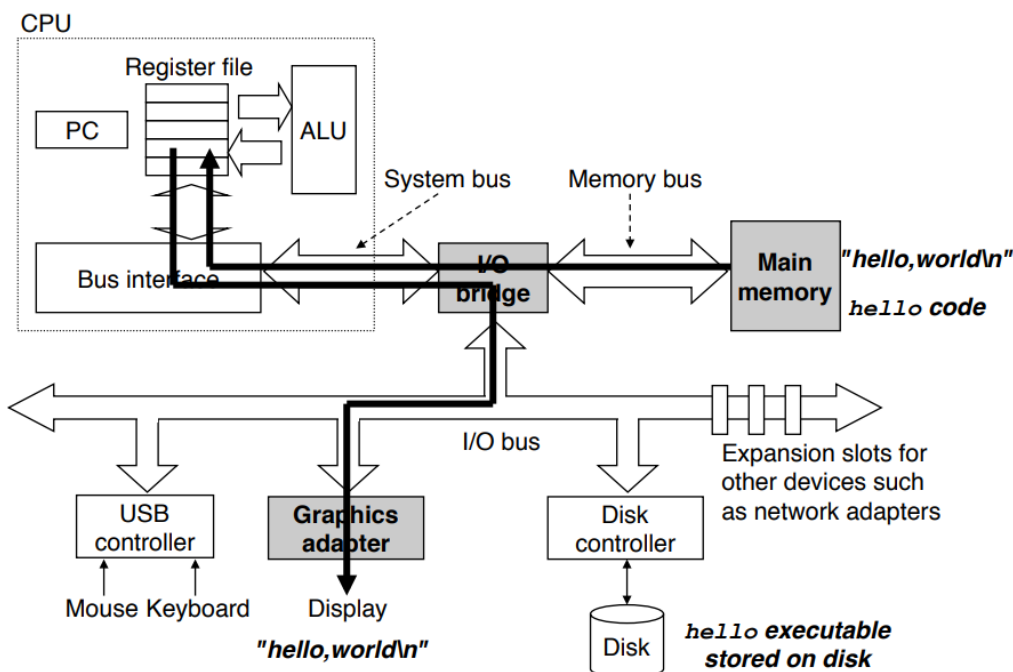
! 1.6>

1. 서론

컴퓨터 구조 복습 1: hello 실행 - hello 프로그램 수행

□ hello 프로그램의 main 루틴의 기계어 명령어를 수행

- "Hello, world\n" 문자열을 메모리에서 레지스터로 복사한 후, 디스플레이 장치로 전송하여 화면에 글자로 표시



<그림 1.7>

1. 서론

컴퓨터 구조 복습 2

캐시가 중요하다
저장장치들은 계층 구조를 이룬다

운영체제

컴퓨터 구조 복습 2: 컴퓨터 시스템에서의 정보 이동

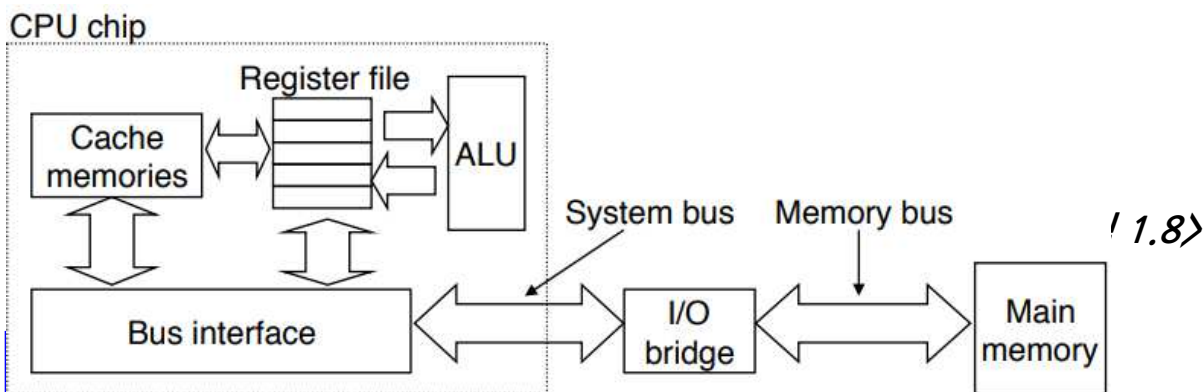
- 앞의 예와 같이 컴퓨터 시스템은 정보의 이동에 많은 시간을 소비
 - 디스크의 hello 프로그램을 메인 메모리로 복사
 - 메인 메모리의 명령어들이 프로세서(레지스터)로 복사
 - “Hello, world\n” 문자열 데이터도 디스크에서 메인 메모리, 프로세서를 경유하여 디스플레이 장치로 복사
- 컴퓨터 시스템에서 이들 이동(복사) 과정을 빠르게 처리하는 것이 중요
 - 저장 장치를 단일 메모리 구조가 아닌 속도와 가격에 따라서 여러 계층의 메모리 설계

컴퓨터 구조 복습 2: 캐시 메모리 (Cache Memory)

□ 레지스터와 메모리

- 프로세서의 레지스터는 작은 용량(수백 바이트)을 저장하지만 메인 메모리보다 100배 이상 빠름
- 메인 메모리는 많은 용량(GB 급)을 저장하지만 느림
- 프로세서와 메모리 간의 속도 격차는 지속적으로 증가

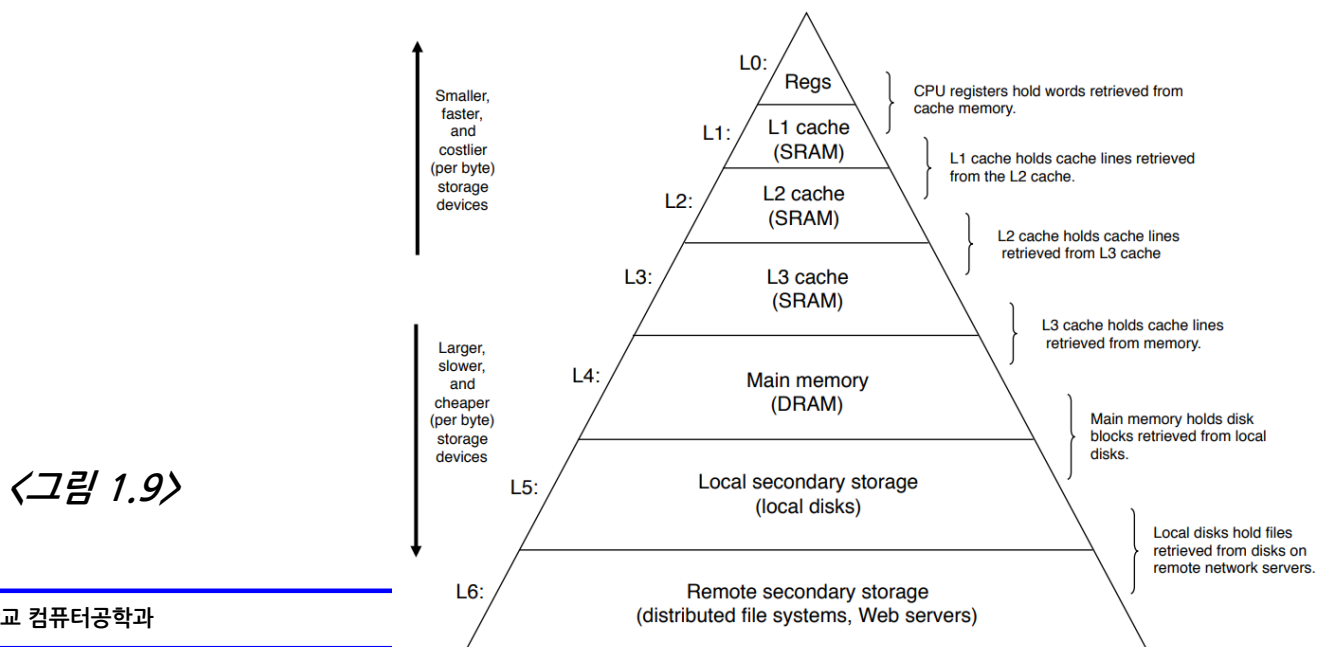
□ 프로세서-메모리 간의 격차에 대응하기 위해 작고 빠른 캐시 메모리(간단히 캐시)를 고안하여 프로세서가 단기간에 사용할 가능성이 높은 정보를 저장



컴퓨터 구조 복습 2: 메모리 계층 구조 (Memory Hierarchy)

□ 프로그램이 특정 영역의 코드와 데이터를 참조하는 경향인 지역성(locality)을 활용하여 메모리 계층 구조를 설계

- 자주 참조할 가능성이 높은 데이터나 코드를 작고 빠른 저장장치(캐시)에 저장하여 접근



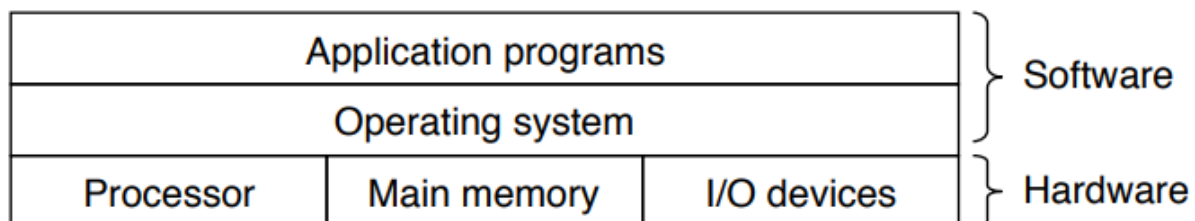
컴퓨터 구조 복습 3

운영체제는 하드웨어를 관리한다

운영체제

컴퓨터 구조 복습 3: 운영체제 (Operating System)

- 운영체제는 하드웨어와 소프트웨어 사이에 위치한 소프트웨어로 하드웨어를 제어하고 조작하는 서비스 제공
 - hello 프로그램이 키보드나 디스플레이, 디스크, 메인 메모리를 직접 접근하지 않고 운영체제가 제공하는 서비스를 활용



〈그림 1.10〉

컴퓨터 구조 복습 3: 추상화

□ 운영체제는 추상화(abstraction)를 통해 서비스를 제공

- 파일 (file)은 입출력 장치의 추상화
- 가상메모리 (virtual memory)는 메인 메모리와 디스크 입출력 장치의 추상화
- 프로세스 (process)는 프로세서, 메인 메모리, 입출력 장치 모두의 추상화

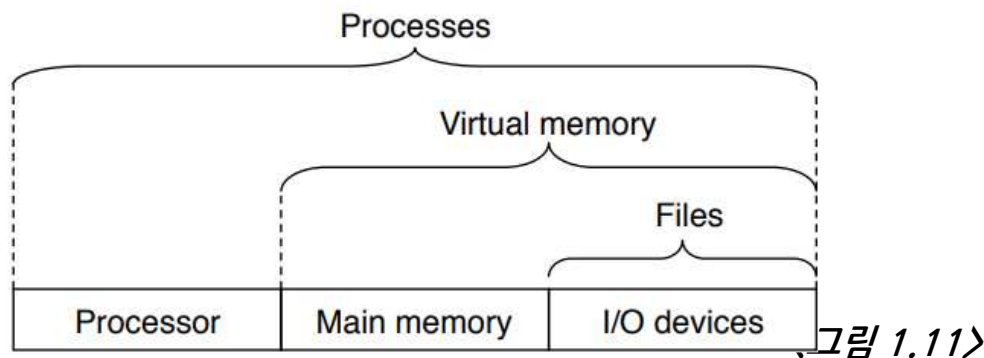


그림 1.11

컴퓨터 구조 복습 3: 프로세스 (Process)

□ 프로세스는 실행 중인 프로그램에 대한 운영체제의 추상화

- 한 개의 CPU에 다 수의 프로세스들이 동시에 실행(병행실행, concurrent execution)하는 것처럼 보임
 - 각 프로세스는 각 하드웨어를 배타적으로 사용하는 것처럼 느껴짐

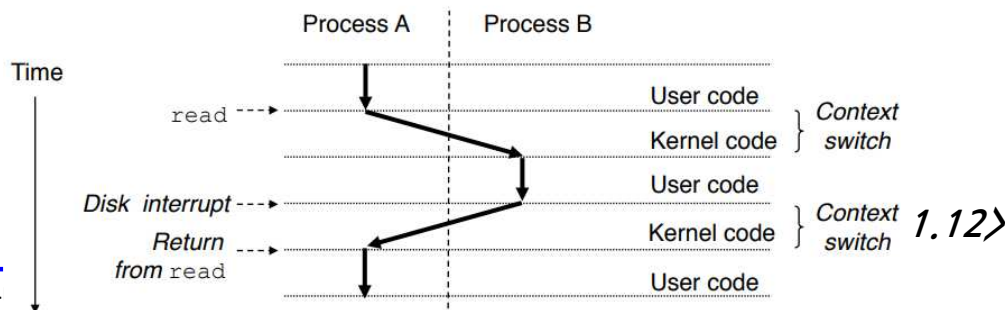
□ 실제로 운영체제는 문맥전환(context switch) 기법으로 여러 프로세스들을 교차 실행

- 프로세스가 실행하는데 필요한 모든 상태정보를 문맥(또는 컨텍스트)라고 함
 - PC, 레지스터 파일, 메모리의 현재 값들
- 현재 프로세스에서 다른 프로세스로 제어를 옮길 때 현재 프로세스의 컨텍스트를 저장하고, 새로운 프로세스의 컨텍스트를 복원하는 것이 문맥 전환

컴퓨터 구조 복습 3: 프로세스 - hello 프로세스 시나리오 예

□ 두 개의 동시성 프로세스 (concurrent process) 존재

- 명령의 입력을 기다리는 쉘 프로세스 (프로세스 A)
- hello 프로세스 (프로세스 B)
- 문맥전환 동작
 - 쉘이 hello 프로그램 실행 명령을 받으면 **시스템 콜(system call)**이라는 특수 함수를 사용해서 제어권을 운영체제에 넘김
 - 운영체제는 쉘 컨텍스트를 저장하고, 새로운 hello 프로세스 컨텍스트를 생성하고 제어권을 hello 프로세스로 넘겨줌
 - hello 운영체제는 프로세스가 종료되면 쉘 프로세스의 컨텍스트를 복구하고 제어권을 쉘에게 넘겨 다음 명령의 입력을 기다림



컴퓨터 구조 복습 3: 스레드 (Thread)

□ 프로세스는 스레드라는 다 수의 실행 유닛으로 구성

- 각각의 스레드는 해당 프로세스의 컨텍스트에서 실행되며 **동일한 코드와 전역 데이터(global data)**를 공유

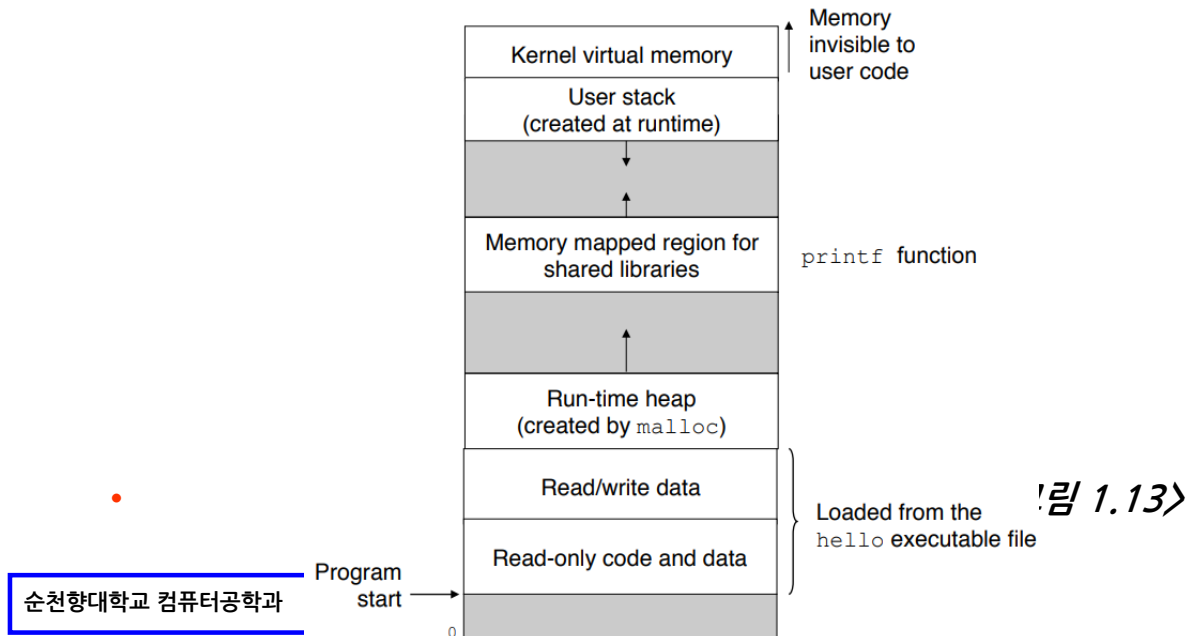
□ 스레드는 프로그래밍 모델로써 중요성이 점점 커지고 있음

- 다 수의 프로세스들 간 보다 스레드들 간에 **데이터의 공유**가 쉬움
- 멀티프로세서(multi-processor) 구조에서 **다중 스레딩(multi-threading) 프로그래밍**은 여러 개의 프로세서를 활용한 병렬처리로 프로그램의 실행 속도를 높임

컴퓨터 구조 복습 3: 가상메모리 (Virtual Memory)

□ 가상메모리는 각 프로세스들이 메인 메모리 전체를 독점적으로 사용하는 것과 같은 환상을 제공하는 추상화

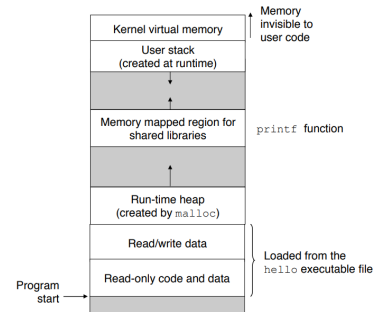
- 각 프로세스는 가상주소 공간이라는 균일한 메모리의 형태를 가짐
- 리눅스 프로세스의 가상주소공간



컴퓨터 구조 복습 3: 가상주소 공간

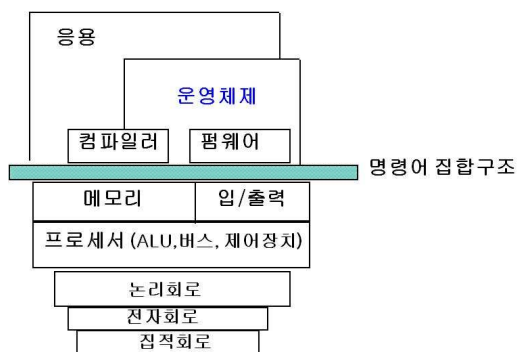
□ 프로세스의 가상주소공간 영역 구분

- 프로그램 코드와 데이터
 - 실행가능 목적파일로 초기화되는 코드와 데이터(전역변수) 영역
- 힙 (heap)
 - 실행 시(런타임)에 동적으로 할당되는 고정된 크기의 영역
- 공유 라이브러리
 - 공유 라이브러리 (C 표준, 수학 등) 코드와 데이터를 저장하는 영역
- 스택 (stack)
 - 함수 호출 및 리턴 시에 사용되는 지역변수, 리턴 값 등이 저장되는 영역
- 커널 가상메모리
 - 운영체제의 커널이 저장되는 영역



□ 파일은 연속된 바이트들로 구성

- 디스크, 키보드, 디스플레이, 네트워크 등 모든 입출력 장치는 파일로 모델링
- 시스템의 모든 입출력은 유닉스 I/O 시스템 콜을 사용하여 접근
- 응용 프로그램에 시스템의 다양한 입출력 장치의 접근 시 통일된 관점을 제공



1장. 서론 (Introduction)

순천향대학교 컴퓨터공학과
이 상 정

강의 목표 및 내용

□ 목표

- 운영체제의 주요 구성 요소 이해
- 컴퓨터 시스템 구조에 대한 기본지식

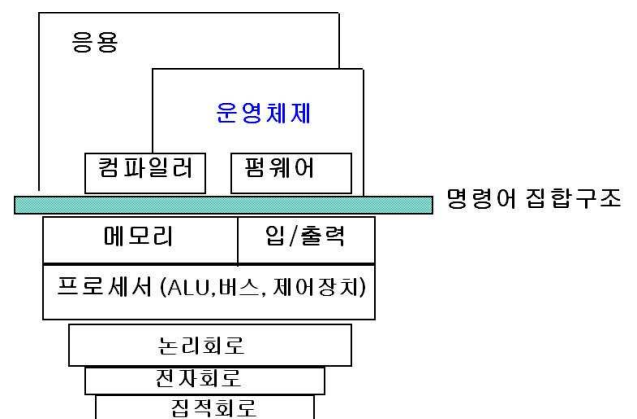
□ 내용

- 운영체제가 하는 일
- 컴퓨터 시스템 구성 및 구조
- 운영체제의 구조 및 연산
- 프로세스 관리, 메모리 관리, 저장 장치 관리
- 보호와 보안
- 커널 자료구조
- 계산 환경
- 오픈 소스 운영체제

운영체제(Operating System, OS)란?

□ 운영체제란?

- 컴퓨터 하드웨어 관리하는 프로그램
- 컴퓨터 사용자와 컴퓨터 하드웨어 사이의 중재자 역할을 하는 프로그램



□ 운영체제 목표

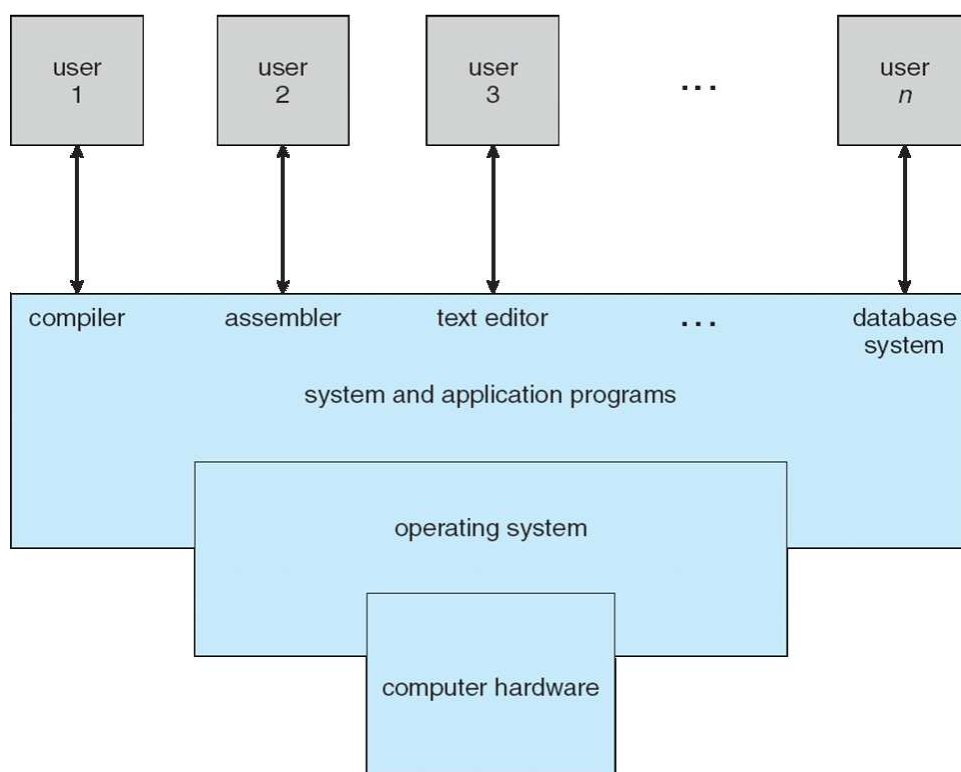
- 사용자 프로그램을 실행
- 사용자가 컴퓨터 시스템을 쉽고 편리하고 사용하여 문제 해결
- 컴퓨터 하드웨어를 효율적으로 관리

컴퓨터 시스템(Computer System) 구성 요소 (1)

□ 컴퓨터 시스템의 4가지 구성 요소

- 하드웨어
 - 기본 계산 자원(computing resource)을 제공
 - 중앙 처리 장치(CPU), 메모리, 입/출력(I/O) 장치
- 운영체제
 - 다양한 응용들 및 사용자들 간의 하드웨어 사용을 제어하고 중재
- 응용 프로그램
 - 사용자의 계산 문제를 해결하기 위해 시스템 자원(system resource)들이 어떻게 사용될 것인지를 정의
 - 워드 프로세서, 컴파일러, 웹 브라우저, 데이터베이스 시스템, 비디오 게임 등
- 사용자
 - 사람, 기계, 다른 컴퓨터 들

컴퓨터 시스템 구성 요소 (2)



운영체제의 정의

- 운영체제는 **자원 할당기 (resource allocator)**
 - 모든 자원들을 관리하고 **스케줄 (schedule)**
 - 서로 상충되는 요청들을 조정하여 효율적이고 공정하게 자원을 사용
- 운영체제는 **제어 프로그램 (control program)**
 - 컴퓨터 부적절한 사용과 에러를 방지하도록 **프로그램들의 실행을 제어**
- 운영체제는 **커널 (kernel)**
 - 컴퓨터 상에서 항상 실행되는 프로그램
 - 항상 **메모리에 상주 (memory resident)**
 - 다른 프로그램들(시스템 프로그램 또는 응용 프로그램)은 운영체제에 종속

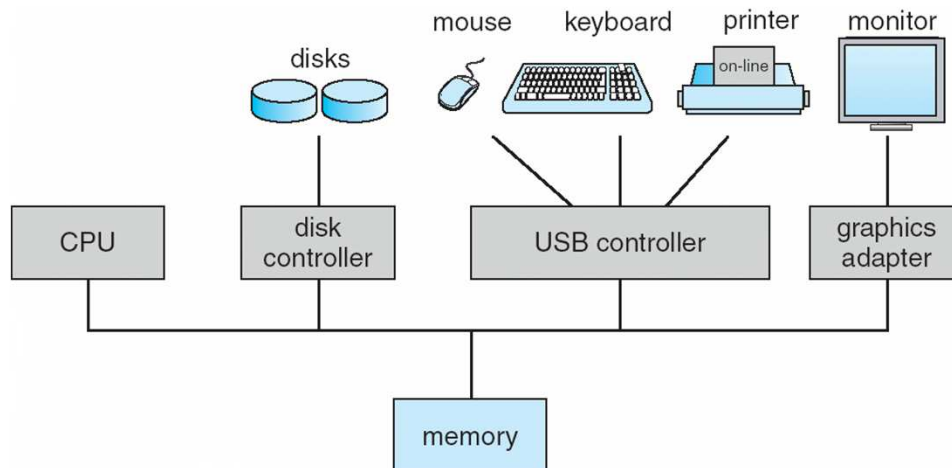
컴퓨터 구동 (Startup)

- **부트스트랩 프로그램 (bootstrap program)**
 - 전원이 켜지거나 재부트(reboot)될 때 실행되는 **초기 프로그램**
 - ROM이나 EEPROM에 저장
 - **펌웨어(firmware)**라고 함.
 - 시스템의 모든 사항을 **초기화**
 - CPU 레지스터, 장치 제어기, 메모리 내용 등
 - 운영체제의 **커널을 찾아 메모리에 적재하고 실행을 시작**

컴퓨터 시스템 구조(Organization)

❑ 컴퓨터 시스템 연산(operation)

- 공유 메모리에 접근 가능한 공통 버스에 연결된 하나 이상의 CPU와 여러 개의 장치 제어기(device controller)들로 구성
- CPU와 장치 제어기는 메모리 사이클을 얻기 위해 경쟁하면서 **병행 수행 (concurrent execution)**



컴퓨터 시스템 연산

- ❑ 입/출력 장치들과 CPU는 **병행 수행 (concurrent execution)**
- ❑ 입/출력 장치 제어기(I/O device controller)
 - 특정 장치 타입에 대해 동작
 - 지역 버퍼(local buffer)를 가짐 (data register)
- ❑ CPU는 주 메모리(main memory)와 지역 버퍼들 사이에 **데이터 송수신을 제어**
- ❑ 입/출력은 장치와 제어기의 **지역 버퍼 사이 수행**
- ❑ 장치 제어기는 입출력 연산 종료 시 **인터럽트(interrupt)**를 사용하여 CPU에 알림

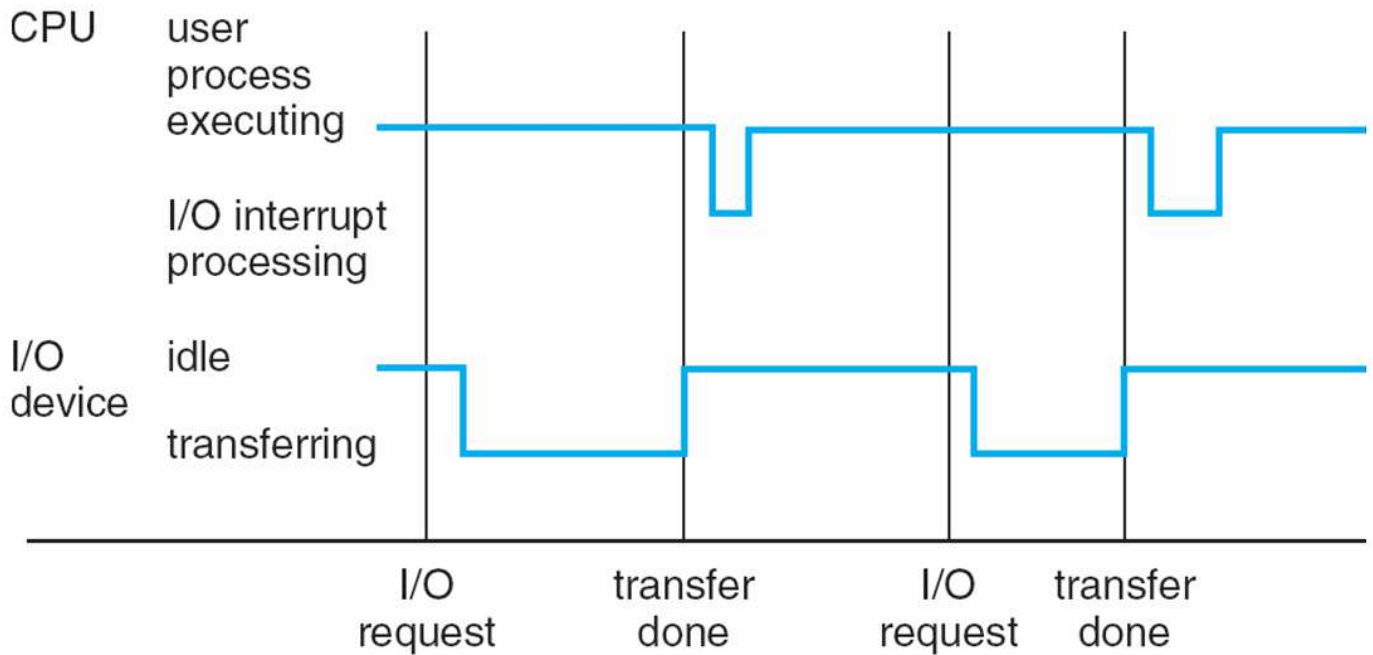
인터럽트 (Interrupt) 소개

- ❑ CPU에 **인터럽트 신호(signal)**가 들어오면 실행 중인 작업을 멈추고 고정된 위치의 **인터럽트 서비스 루틴(interrupt service routine)**으로 이동하여 실행 (p.10 그림 1.4 참조)
 - **인터럽트 벡터(interrupt vector)**가 인터럽트 서비스 루틴들의 시작 주소에 관한 정보 제공 (p.11 그림 1.5 참조)
 - 실행 중인 작업 주소를 저장하여 서비스 루틴 종료 후 복귀
- ❑ 다른 인터럽트 처리 중에 새로이 요청되는 인터럽트는 승인 거부 될 수 있음
 - **인터럽트 우선순위**에 따라 승인 또는 거부 결정
- ❑ **트랩(trap)** 또는 **예외(exception)**은 에러 또는 사용자 요청에 기인한 **소프트웨어가 생성한 인터럽트**
- ❑ 운영체제는 **인터럽트 기반으로 구동**

인터럽트 처리 (Interrupt Handling)

- ❑ 운영체제는 **레지스터와 프로그램 카운터(PC)**를 저장하여 실행 중인 **CPU 상태를 보존**
- ❑ 발생한 인터럽트의 형태를 파악:
 - **폴링 (polling)**
 - **벡터 인터럽트 시스템**
- ❑ **인터럽트의 형태**에 따라 서로 다른 코드에 의해 적용되는 동작을 결정

입/출력 인터럽트 시간 일정 (Time Line)



입출력 구조 (I/O Structure)

- 입출력 명령 실행하면 **입출력 동작 종료 시**에만 사용자 프로그램으로 제어 이동
 - 대기 명령(wait instruction)으로 다음 인터럽트까지 CPU는 쉼(idle)
 - 대기 루프
 - 한 번에 하나의 입출력 요청만 가능하고, 동시에 여러 입출력 처리할 수 없음
- 입출력 명령을 실행하면 입출력 동작 종료를 기다리지 않고 사용자 프로그램으로 리턴
 - **시스템 호출(system call)**
 - 사용자가 입출력 종료 시까지 기다리도록 운영체제에 요청
 - **장치-상태 테이블(device-status table)**
 - 각 입출력 장치의 타입, 주소, 상태 등을 저장
 - 운영체제는 장치의 상태 결정, 인터럽트 등을 포함한 테이블 엔트리 수정을 위해 입출력 테이블을 참조

저장 장치 구조 (Storage Structure)

□ 주 메모리 (main memory)

- CPU가 직접 접근 가능한 유일한 대량의 저장 장치
- 주로 DRAM으로 구현하고 임의 접근(random access)
- 일반적으로 휘발성 (volatile)

□ 보조 저장 장치 (secondary storage)

- 대용량의 비휘발성(nonvolatile) 저장장치로 주 메모리를 확장
- 자기 디스크 (magnetic disk)
 - 디스크의 표면은 논리적으로 트랙(track)과 섹터(sector)로 분할
 - 디스크 제어기(disk controller)는 장치와 컴퓨터 간에 논리적인 상호 작용을 제어
- solid-state disk (drive)
 - 자기 디스크보다 빠름, 비휘발성
 - 최근 많이 사용되고 있음

저장 장치 계층 (Hierarchy) (1)

□ 저장 장치는 계층(hierarchy)으로 구성

- 속도, 비용, 휘발성

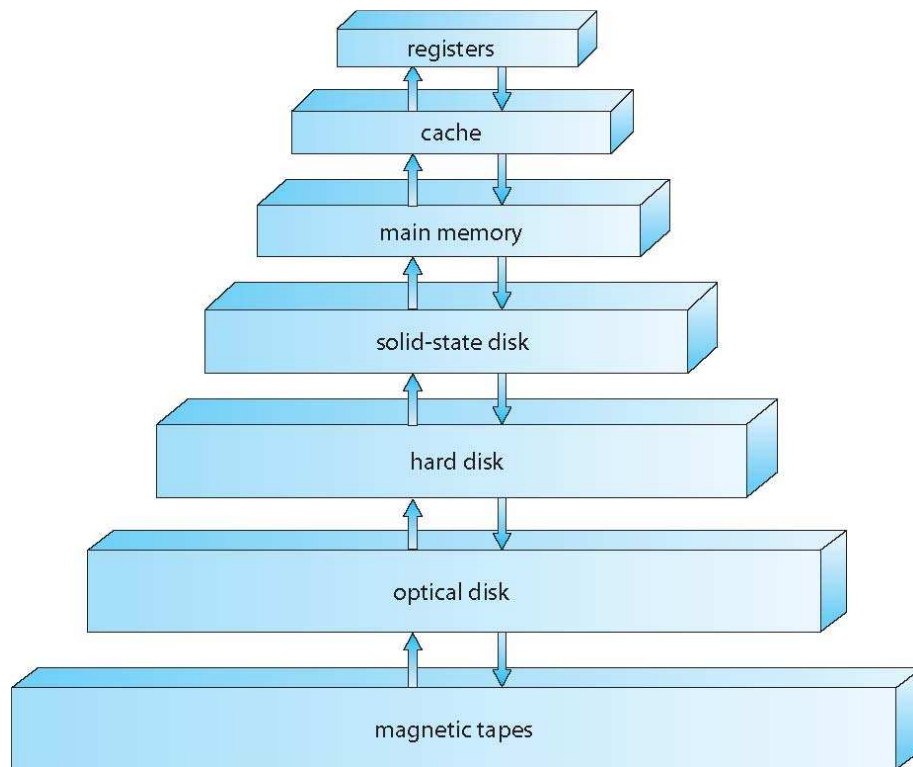
□ 캐싱 (caching)

- 더 빠른 저장 장치로 정보를 복사
- 주 메모리는 보조 저장 장치를 캐싱

□ 운영체제는 각 장치 제어기마다 장치 드라이버(device driver)를 내장

- 장치 드라이버는 장치 제어기의 동작을 이해하고, 제어기와 커널 사이에 일관된 인터페이스 제공

저장 장치 계층 (2)



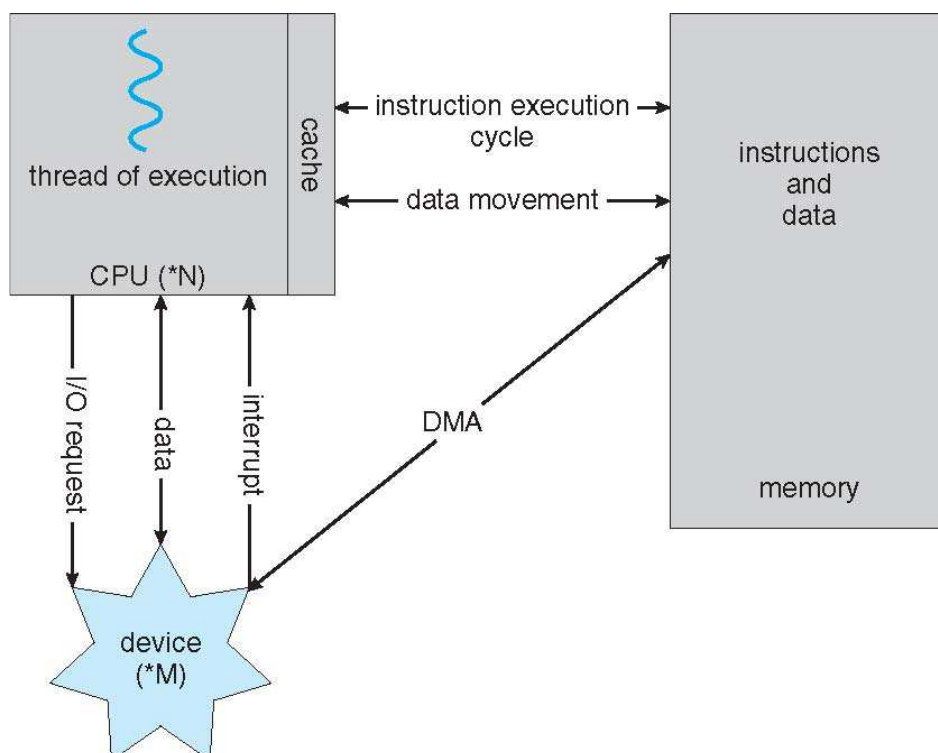
캐싱 (Caching)

- ❑ 컴퓨터의 여러 계층에서 사용되는 중요한 원칙
 - 하드웨어, 운영체제, 소프트웨어
- ❑ 더 느린 저장장치(원본)에서 **사용 중인 정보의 부분**을 더 빠른 저장장치로 복사
- ❑ 사용 중인 정보가 **더 빠른 저장장치(캐시)**에 있는지 조사
 - 있으면 정보는 캐시(더 빠른 저장 장치)로 부터 추출
 - 없으면 캐시로 복사한 후 추출
- ❑ 캐시의 크기는 원본의 저장장치보다 더 작음
 - 캐시의 관리가 중요함
 - 캐시의 크기와 **대체 정책(replacement policy)**

직접 메모리 접근 (Direct Memory Access, DMA)

- ❑ 인터럽트 구동 방식의 입/출력은 디스크 입/출력과 같은 대량의 데이터를 전송하는 데에는 **높은 오버헤드(overhead)**를 초래
- ❑ 직접 메모리 접근(Direct Memory Access, DMA)
 - 고속 전송이 가능한 입/출력 장치에 사용
 - 장치 제어기는 CPU의 개입 없이 메모리로부터 자신의 버퍼 장치로 또는 버퍼로부터 메모리로 **데이터 블록** 전체를 전송
 - 속도가 느린 장치처럼 한 바이트마다 인터럽트가 발생하는 것이 아니라 블록 전송이 완료될 때마다 인터럽트가 발생

컴퓨터 시스템 동작



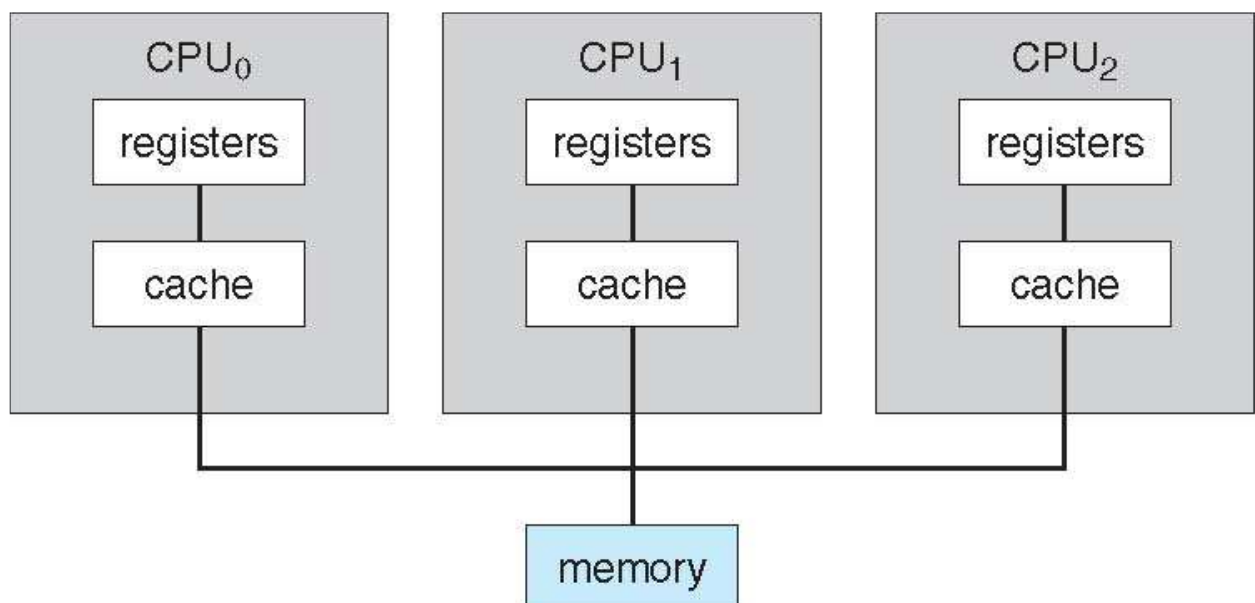
□ 단일 처리기 시스템(Single-Processor System)

- 범용 처리기(processor, CPU)가 하나 있는 시스템
- 디스크나 그래픽 제어기와 같은 특정 장치 처리기

□ 다중 처리기 시스템(Multiprocessor System)

- 다수의 처리기를 갖는 시스템
 - 병렬 시스템, 멀티코어 시스템 이라고도 함
- 주요 장점
 - 증가된 처리량 (throughput)
 - 규모의 경제 (economy of scale)
 - 각 처리기가 주변장치, 저장장치, 전원 등을 공유하여 비용 절감
 - 증가된 신뢰성 (reliability)
- 2가지 유형
 - 대칭적 다중 처리(symmetric multiprocessing, SMP)
 - 비대칭적 다중 처리 (asymmetric multiprocessing)

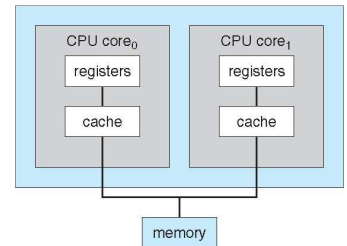
대칭적 다중처리 구조 (Symmetric Multiprocessing Architecture)



UMA vs. NUMA 아키텍처

- 균등, 비균등 메모리 접근 (uniform, non-uniform memory access)

멀티칩(multi-chip) vs. 멀티코어(multicore)



시스템에 모든 칩을 내장 vs. 블레이드 서버(blade servers)

- 블레이드 서버는 한 새시(Chassis)에 다수의 처리기 보드, 입출력 보드, 네트워크 보드 등 장착
- 각 처리기 보드마다 독립적인 운영체제 수행



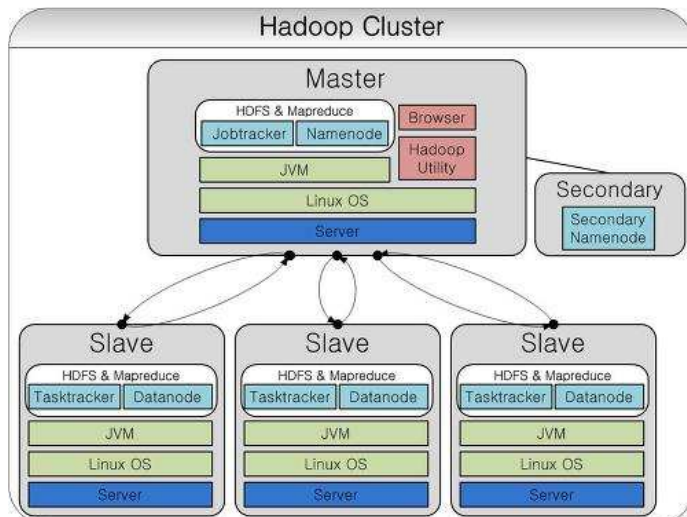
클러스터형 시스템 (Clustered System) (1)

둘 이상의 독자적 시스템(노드)들을 연결하여 협력 동작

- *SAN(storage-area network)*을 사용하여 저장장치 공유하기도 함
- 일부 노드(머신) 고장 시에도 동작 가능한 **고가용성(high-availability)** 제공
 - 비대칭 클러스터링(asymmetric clustering)
 - 다른 머신이 응용을 실행하는 동안 한 머신은 긴급-대기(host-standby) 모드로 상태를 유지하고 동작 중인 머신(서버) 감시
 - 서버 중 하나가 고장 나면 긴급 대기 모드의 서버가 활성화 됨
 - 대칭 클러스터링(symetric clustering)
 - 여러 노드들이 응용을 수행하고 서로 모니터링
- 고성능 컴퓨팅을 위해서도 클러스터링 사용
 - 병렬 응용 프로그램을 작성해야 함
- 공유 데이터에 대한 서로 충돌되는 연산을 피하고 동기화가 필요한 클러스터에서는 분산 잠금 관리자(distributed lock manager, DLM) 사용

클러스터 시스템 (2)

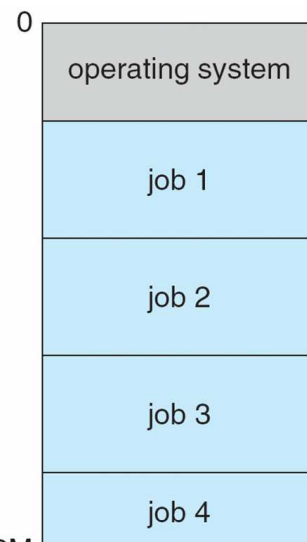
- 최근에 **빅 데이터** 저장 및 처리를 위해 클러스터 구성
 - 분산 파일시스템 구성
 - HDFS (Hadoop Distributed File System)
 - MapReduce 병렬 처리



운영체제의 구조 (OS Structure) - 다중 프로그래밍

□ 다중 프로그래밍(multiprogramming)

- 단일 사용자는 CPU 또는 입/출력 장치를 항상 바쁘게 유지할 수 없음
- **다중프로그래밍(multiprogramming)**은 CPU가 수행할 작업(코드와 데이터)을 항상 하나는 가지도록 **작업(job, process)**을 구성
- 운영체제는 한 번에 **여러 작업을 메모리에 적재**
- 운영체제는 메모리 내에 있는 작업 중에서 하나를 선택해 실행
- 선택된 작업이 입/출력의 종료를 기다리는 동안 운영체제는 다른 작업으로 전환해 수행



운영체제 구조 - 시분할 (Time Sharing)

□ 시분할(time sharing 또는 멀티태스킹: multi-tasking)은 다중 프로그래밍의 논리적 확장

- CPU가 다수의 작업들을 매우 빈번하게 교대가 일어나기 때문에 프로그램이 실행되는 동안 **사용자**들은 각자 자기의 프로그램과 **상호 작용**
- **응답 시간(response time)**이 짧아야 하며, 통상 1초
- 각 **사용자**는 메모리에 최소한 **하나의 프로그램**을 가짐
=> **프로세스(process)**
- 여러 개의 작업이 동시에 실행준비가 되어 있으면, 시스템은 이들 중 하나를 선택 => **CPU 스케줄링**
- 프로세스들이 모두 메모리에 놓일 수 없는 경우 **스와핑(swapping)**
 - 스와핑이란 필요에 의해 프로세스를 주 메모리에서 디스크로, 디스크에서 주 메모리로 옮기는 작업
- **가상 메모리(virtual memory)**를 사용하여 작업 전체가 아닌 메모리에 있는 일부만의 작업 수행을 허용

운영체제 연산 (OS Operations)

□ 운영체제는 인터럽트 구동식(interrupt driven)

□ 트랩(trap)(또는 예외, exception)은 오류 또는 사용자 프로그램의 운영체제 서비스 수행 요청에 의해 유발되는 **소프트웨어에 의해 생성된 인터럽트**

- 0으로 나누기 또는 유효하지 않은 메모리 접근

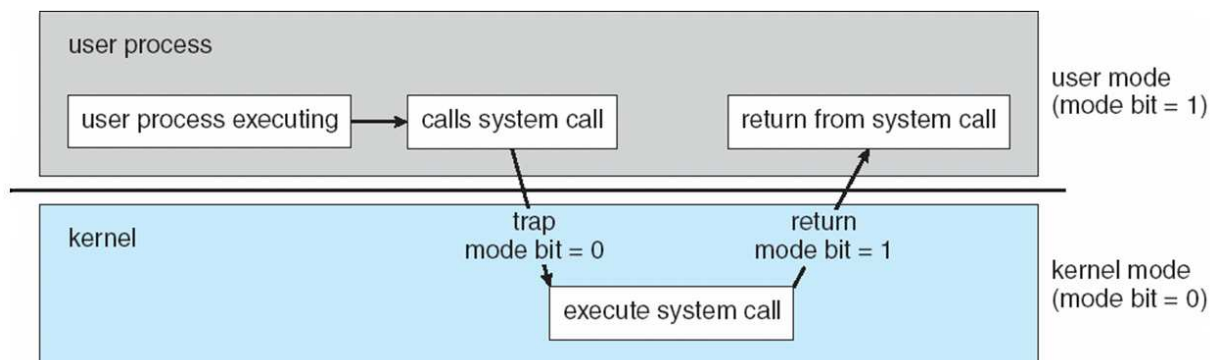
□ 운영체제와 사용자는 컴퓨터 시스템의 **하드웨어와 소프트웨어 자원을 공유**하기 때문에 사용자 프로그램의 오류가 현재 수행중인 프로그램에만 문제를 일으키도록 보장해야 함

운영체제 연산 - 이중 연산 모드

- 이중 연산 모드로 운영체제 코드의 실행과 사용자 정의 코드의 실행을 구분하여 운영체제를 보호
 - 사용자 모드(user mode), 커널 모드(수퍼바이저(supervisor) 모드, 시스템 모드, 특권 모드(privileged mode))
- 하드웨어가 모드 비트(mode bit) 제공
 - 커널 모드(0) 또는 사용자 모드(1)를 나타내는 비트
 - 시스템이 사용자 코드 또는 커널 코드 실행 여부를 구분
 - 일부 명령을 특권 명령(privileged instruction)으로 지정하여 커널 모드에서만 수행되도록 허용
- CPU의 다중 모드(multi-mode) 지원이 증가하는 추세
 - 예, 게스트 가상머신(virtual machine)을 관리하는 가상머신 관리기를 위한 별도의 모드 비트

시스템 호출(System Call)

- 시스템 호출(system call)은 사용자 프로그램이 자신을 대신하여 운영체제가 수행하도록 예약되어 있는 작업들을 운영체제에 게 요청
 - 시스템 호출 시 사용자 모드에서 커널 모드로 변경되고, 복귀 시 사용자 모드로 리셋



사용자 모드에서 커널 모드로 이동

- 무한 루프나 프로세스의 자원 독점을 방지하기 위해 타이머 사용 (타이머 인터럽트)
 - 일정 시간 후에 인터럽트를 발생하도록 타이머 지정
 - 물리적인 클럭으로 감소되는 카운터 관리
 - 운영체제가 카운터 초기값 세트 (특권 명령)
 - 카운터가 0 이되면 인터럽트 발생
 - 프로세스 스케줄링 전에 CPU 제어권을 다시 획득하기 위해서, 또는 할당 된 시간을 초과하여 실행하는 프로그램을 종료하기 위해 설정

프로세스 관리(Process Management) (1)

- 프로세스는 실행 중인 프로그램
 - 시스템의 작업 단위
 - 프로그램은 수동적 개체 (passive entity)
 - 프로세스는 능동적 개체 (active entity)
- 프로세스는 자신의 일을 수행하기 위해 CPU 시간, 메모리, 파일, 그리고 입/출력 장치를 포함한 여러 가지 자원이 필요
 - 프로세스 종료 시 사용 중인 자원들을 해제
- 단일 스레드(thread) 프로세스는 다음 수행할 명령을 지정하는 하나의 프로그램 카운터(program counter)를 가짐
 - 다중 스레드 프로세스는 복수개의 프로그램 카운터를 가짐
- 일반적으로 시스템에서는 여러 프로세스가 동작 중
 - 일부는 운영체제 프로세스들, 일부는 사용자 프로세스들
 - 단일 CPU 코어에서 멀티플렉싱하거나(병행 실행), 여러 CPU 코어들에서 병렬실행

프로세스 관리 (2)

- 운영체제는 **프로세스 관리**와 연관해 다음과 같은 활동에 대한 책임
 - CPU에 **프로세스와 스레드**를 스케줄하기
 - 사용자 프로세스와 시스템 프로세스의 **생성**과 **제거**
 - 프로세스의 **일시 중지**와 **재수행**
 - 프로세스 **동기화(synchronization)**를 위한 기법 제공
 - 프로세스 **통신**을 위한 기법 제공
 - **교착상태(deadlock)** 처리를 위한 기법 제공

메모리 관리 (Memory Management)

- **메인 메모리(main memory)**는 CPU와 입/출력 장치에 의하여 공유되는, 빠른 접근이 가능한 데이터의 저장소
- 실행되는 모든 명령들과 데이터는 **메모리 내에 상주**
 - 폰 노이만 방식 컴퓨터
- **메모리 관리**는 **CPU 이용률**과 **사용자 응답율**을 최적화하는 관점에서 **메모리에 놓일 프로세스를 결정**
- 운영체제는 **메모리 관리**와 관련하여 다음과 같은 일을 담당
 - 메모리의 어느 부분이 현재 사용되고 있으며 누구에 의해 사용되고 있는지를 **추적**
 - 어떤 프로세스(또는 그 일부)들을 메모리에 **적재**하고 **제거**할 것인가를 결정
 - 필요에 따라 메모리 공간을 **할당(allocation)**하고 **회수(release)**

저장 장치 관리 (Storage Management)

- 운영체제는 정보 저장을 위한 일관되고 논리적인 관점을 제공
 - 물리적인 속성을 논리적인 저장 단위로 추상화 - 파일(file)
 - 장치가 각 매체를 제어 (예, 디스크 드라이브, 테이프 드라이브)
 - 접근속도, 용량, 데이터 전송속도, 접근방법(sequential or random) 등 다양한 속성을 가짐
- 파일 시스템 관리 (file system management)
 - 파일은 디렉토리(directory)로 조직화됨
 - 파일의 사용권한을 위한 접근제어 (access control)
 - 운영체제는 파일 관리를 위하여 다음과 같은 일을 담당
 - 파일/디렉토리의 생성 및 제거
 - 파일과 디렉토리를 조작하기 위한 프리미티브(primitive)의 제공
 - 파일을 보조 저장 장치로 사상(mapping)
 - 안정적인(비휘발성) 저장 매체에 파일을 백업

대용량 저장 장치 관리 (Mass-Storage Management)

- 디스크에는 주 메모리 대신 비휘발성 대용량의 데이터를 저장
 - 컴퓨터 연산의 처리 속도는 디스크 서브시스템과 알고리즘에 크게 영향 받음
- 운영체제는 디스크 관리를 위하여 다음과 같은 기능을 담당
 - 자유 공간(free space)의 관리
 - 저장 장치 할당
 - 디스크 스케줄링

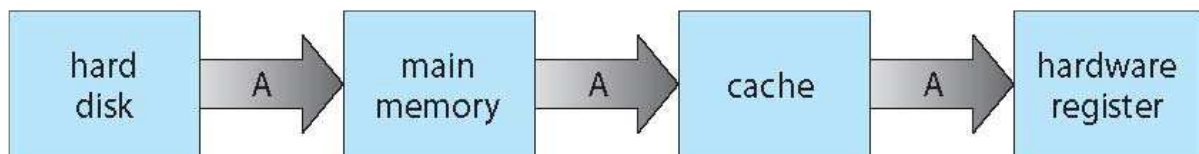
다양한 저장 장치 성능

- 저장 장치 계층 간의 이동은 **명시적(explicit)** 또는 **암묵적(implicit)**으로 이동

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

디스크에서 레지스터로 정수 A의 이동

- **다중 태스킹 환경**에서는 저장 장치 계층에서 가장 최근의 값을 사용해야만 함



- **멀티프로세서 환경**에서는 모든 CPU의 캐시에 가장 최근의 값을 가지도록 하드웨어가 **캐시 일관성(cache coherency)**을 보장해야 함
- **분산 환경**에서는 더욱 복잡해짐
 - 같은 데이터의 여러 복사본이 존재

입출력 서브시스템 (I/O Subsystem)

- 운영체제의 목적 중 하나는 **사용자에게 하드웨어의 상세함을 숨기는 것임**
- **입출력 서브시스템**은 다음과 같은 기능을 담당
 - 다음 입출력 메모리 관리를 수행
 - 버퍼링(buffering): 전송 중 데이터를 임시 저장
 - 캐싱(caching): 데이터의 일부를 고속의 저장장치에 저장
 - 스푼링(spooling): 한 작업의 출력을 다른 작업의 입력과 중첩
 - 일반적인 **장치 드라이버 인터페이스**
 - 특정 하드웨어 장치들을 위한 드라이버

보호와 보안 (Protection and Security)

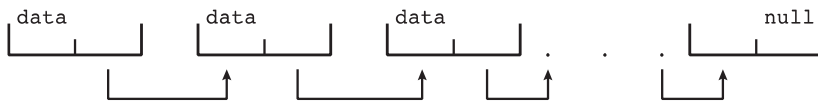
- **보호(Protection)**란 컴퓨터 시스템이 정의한 자원에 대해 프로그램, 프로세스, 또는 사용자들의 **접근을 제어**하는 기법
- **보안(Security)**은 컴퓨터 시스템을 **외부 또는 내부의 공격(attack)**을 **방어**하는 것이 바로 기능
- 보호와 보안을 제공하기 위해서는 **시스템의 모든 사용자들을 구분**
 - 사용자 이름과 연관된 **사용자 식별자(user ID)**의 리스트를 유지
 - 사용자 식별자는 사용자의 모든 **프로세스나 스레드, 파일에 연관**
 - **그룹 식별자**의 리스트로 사용자의 집합을 구분
 - **특권(privilege)**을 사용하여 제한된 장치를 접근

커널 자료 구조 (1)

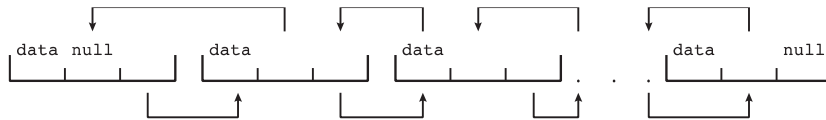
□ 대부분의 운영체제는 **표준 프로그래밍 자료 구조**를 사용하여 구현

□ **연결 리스트**

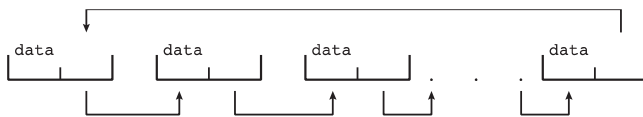
- 단일 연결 리스트 (singly linked list)



- 이중 연결 리스트 (doubly linked list)



- 순환 연결 리스트 (circular linked list)

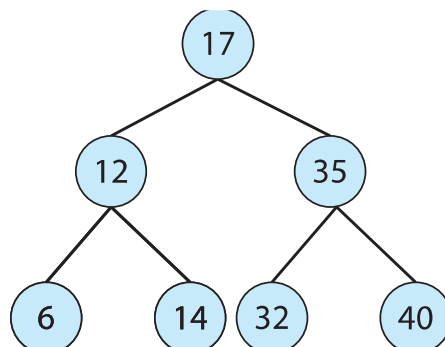


- 예, 연결 리스트는 커널 알고리즘, 스택/큐 등 구현에 사용

커널 자료 구조 (2)

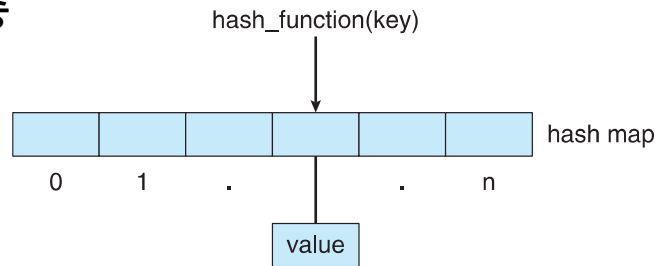
□ **이진 검색 트리 (binary search tree)**

- $left \leq right$
- 검색 성능 $O(n)$
- 균형 (balanced) 이진 검색 트리 $O(\log n)$
- 예, 리눅스 CPU 스케줄링 알고리즘



□ 해시 함수(hash function)은 해시 맵을 생성

- 예, 패스워드 인증



□ 비트맵(bitmap)

- n개 항목의 상태를 나타내는 n개의 이진수의 연속적인 열
- 예, 디스크 블록의 가용 여부 표시

□ 리눅스 자료구조는 include 파일에 저장

- `<linux/list.h>`, `<linux/kfifo.h>`,
`<linux/rbtree.h>`

전통적 계산 (Traditional Computing)

□ 단독의 범용 머신 (stand-alone general purpose machine)

- 현재는 대부분의 머신이 인터넷을 통해 상호 연결되어 이러한 정의는 다소 모호해짐

□ 웹 기술이 전통적인 계산의 경계를 확장

- 포털(portal)은 내부 시스템들의 웹 접근을 허용
- 네트워크 컴퓨터(thin client)는 웹 터미널과 같은 역할
- 모바일 컴퓨터들은 무선 네트워크로 상호 연결

□ 네트워킹은 점점 유비쿼터스 환경이 되고 있음

- 홈 네트워크에서도 인터넷 공격으로 보호하고자 방화벽(firewall)을 사용

이동형 컴퓨팅 (Mobile Computing)

- 스마트폰, 태블릿 등
- 전통적인 노트북과 이들 모바일 기기와의 기능적인 차이는?
- 추가적인 기능
 - 추가적인 기능의 응용 (GPS, 가속도계, 자이로스코프)
- 증강현실(augmented reality)과 같은 새로운 응용 등장
- 네트워크 연결을 위해 IEEE 802.11(WiFi), 셀룰러 데이터 네트워크 사용
- 애플 iOS, 구글 안드로이드

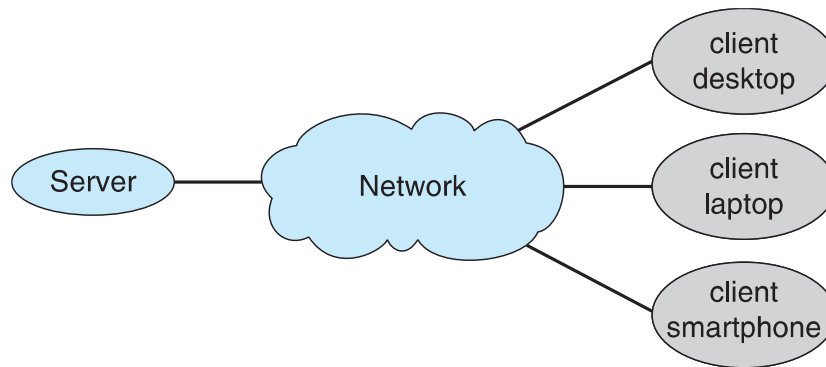
분산 시스템 (Distributed System)

- 분산 계산
 - 네트워크로 연결된 서로 독립된 이질적인 시스템들
 - 네트워크는 통신 경로로 주로 TCP/IP
 - LAN (Local Area Network)
 - WAN (Wide Area Network)
 - MAN (Metropolitan Area Network)
 - PAN (Personal Area Network)
 - 네트워크 운영체제는 네트워크로 연결된 시스템들 간에 필요한 기능 지원
 - 시스템들 간에 메시지를 교환하는 통신 방식 제공
 - 단일 시스템인 것 같은 착각

클라이언트-서버 계산 (Client-Server Computing)

□ 클라이언트-서버 계산(Client-Server Computing)

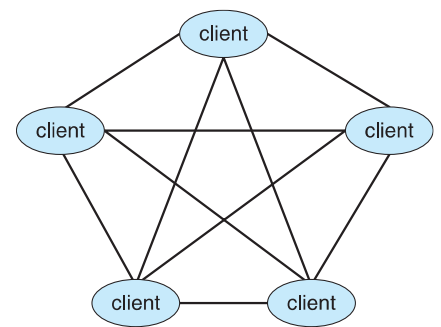
- 클라이언트에 의해 생성되는 요구를 서버가 서비스 응답
- **계산-서버(compute-server)**는 서비스를 요청하는 클라이언트에 인터페이스를 제공 (예, 데이터베이스 서버)
- **파일-서버(file-server)**는 클라이언트가 파일을 생성, 갱신, 읽기 및 제거 할 수 있는 파일 시스템 인터페이스를 제공



피어 간 계산 (Peer-to-Peer Computing)

□ P2P (peer-to-peer)는 클라이언트와 서버가 서로 구별되지 않음

- 모든 시스템 상의 모든 노드가 **피어**로 간주
- 피어는 서비스를 요청하느냐 제공하느냐에 따라 클라이언트 및 서버로 동작
- 노드는 먼저 피어간 네트워크에 **참가(join)**해야 함
 - 노드가 네트워크에 참가할 때 네트워크의 **중앙 검색 서비스(central lookup service)**에 자신이 제공하는 서비스를 등록 또는
 - **서비스 발견 프로토콜(service discovery protocol)**을 사용하여 네트워크 상의 모든 노드에게 서비스 요청 메시지를 방송(broadcast)
- BitTorrent와 같은 파일 공유 서비스 등이 예



가상화 (virtualization) (1)

- 한 운영체제에서 다른 운영체제의 응용들을 실행하고자 하는 요구 증가
 - 다양하고 많은 운영체제들과 응용들 존재
- **에뮬레이션(emulation)**은 소프트웨어로 컴퓨터 하드웨어를 시뮬레이션
 - 소스 CPU 유형이 대상 CPU 유형과 다른 경우 사용
 - i.e. PowerPC to Intel x86
 - 소스 CPU의 명령어를 대상 CPU 명령어로 변환하여야 하므로 원해 소스 CPU 보다 훨씬 느리게 수행

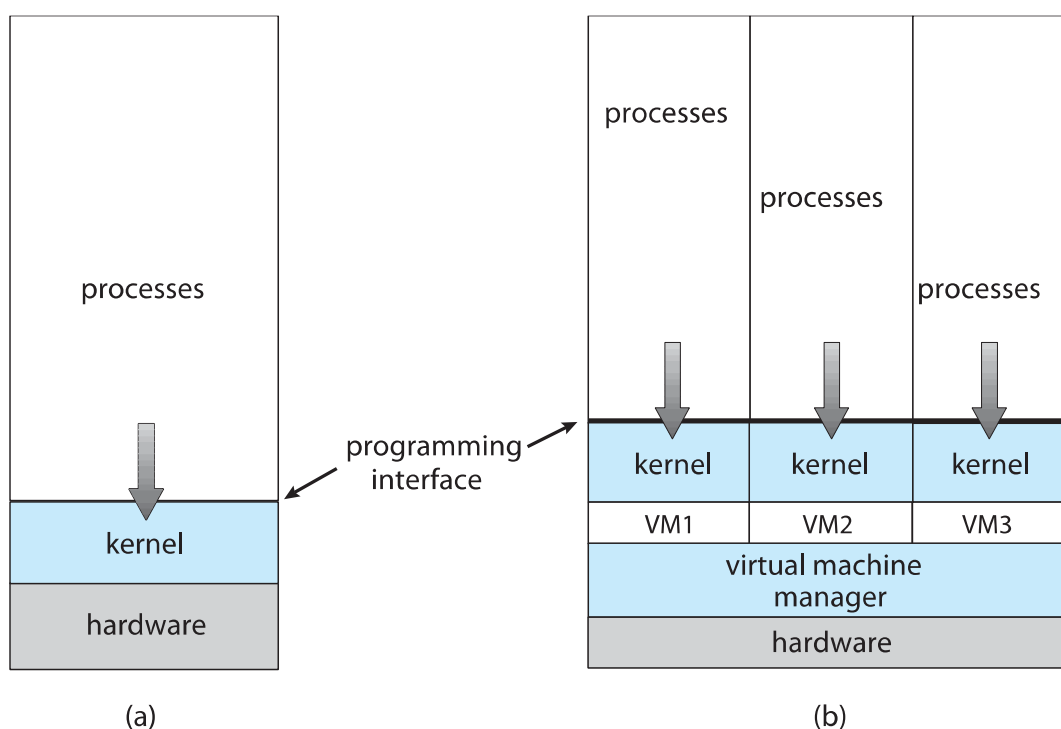
가상화 (Virtualization) (2)

- **가상화**는 특정 CPU를 위해 컴파일된 운영체제가 동일 CPU의 다른 운영체제 내에서 실행
- 한 컴퓨터의 하드웨어(CPU, 메모리, 디스크 드라이브, 네트워크 인터페이스 카드 등)가 **다수의 다른 실행 환경(운영체제 등)**을 제공하도록 추상화
 - 각각 **개별적인 실행 환경이 자신만의 독립된 컴퓨터를 사용하는 환상(illusion)**을 제공
 - 운영체제는 각 프로세스가 자신의 전용(가상) 메모리를 갖는 전용 처리기에서 수행되는 것처럼 환상(illusion)을 제공
 - **VMM (virtual machine manager)**이 가상화 서비스를 제공

가상화 (Virtualization) (3)

- 호환성, 테스트 등을 위해 다수의 운영체제를 실행하는 데스크탑, 노트북의 사용 사례
 - 윈도우 호스트 PC에 게스트로 리눅스를 실행
 - 한 시스템에서 여러 운영체제의 응용 프로그램 개발
 - 한 시스템에서 많은 시스템의 소프트웨어 품질 보증(QA, Quality Assurance) 테스트
 - 데이터 센터 내에서 계산 환경을 실행하고 관리
- VMM이 호스트 운영체제에서 실행되는 것이 아니라 가상 머신 프로세스에게 서비스 및 자원 관리를 제공하는 **호스트 운영체제 그 자체**인 경우도 있음
 - Vmware ESX와 Citrix XenServer

가상화 (4)

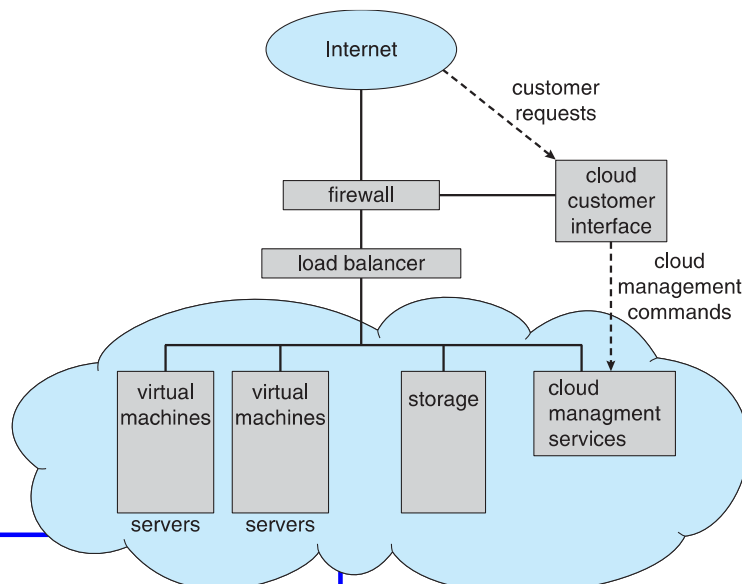


클라우드 컴퓨팅 (Cloud Computing) (1)

- ❑ 계산, 저장장치, 응용 등을 **네트워크 상의 서비스**로 제공하는 계산 유형
- ❑ 가상화를 기반으로 하기 때문에 **가상화의 논리적인 확장**
 - **아마존 EC2**는 인터넷 상에서 수천대의 서버들, 수백만의 가상기계, 페타 바이트 (PB)의 저장장치를 운영하고, 사용량에 기반하여 지불
- ❑ 다양한 유형
 - **공중 클라우드 (public cloud)** - 사용료를 지불할 의사가 있는 누구나 사용
 - **사유 클라우드 (private cloud)** - 기업이 구축하여 사용
 - **혼합 클라우드 (hybrid cloud)** - 공중과 사유 부분을 모두 포함
 - **소프트웨어 서비스 (SaaS, Software as a Service)** - 인터넷 상에서 사용 가능한 하나 이상의 응용 (예, 워드 프로세서, 스프레드시트)
 - **플랫폼 서비스 (PaaS, Platform as a Service)** - 인터넷 상에서 사용하도록 응용에 맞게 준비된 소프트웨어 스택 (예, 데이터베이스 서버)
 - **하부구조 서비스 (IaaS, Infrastructure as a Service)** - 인터넷 상에서 사용 가능한 서버나 저장장치 (예, 백업을 위한 저장장치)

클라우드 컴퓨팅 (2)

- ❑ **클라우드 계산환경은 기존 운영체제, VMM (Virtual Machine Manager), 클라우드 관리 툴로 구성**
 - 방화벽과 같은 안전한 인터넷 연결 필요
 - 여러 응용들을 분배하는 부하 균형기 (load balancer)



실시간 내장형 시스템 (Real-Time Embedded Systems)

- ❑ 실시간 임베디드 시스템은 가장 널리 사용되는 컴퓨터의 형태
 - 아주 다양하고, 특수 목적, 제한된 목적의 운영체제, **실시간 운영체제**
 - 계속 사용이 증대
 - 자동차 엔진, 공장용 로봇, 가전제품
- ❑ 다양한 계산 환경
 - 일부는 운영체제가 있고, 일부는 운영체제 없이 태스크 실행
- ❑ 실시간 운영체제는 잘 정의된 **고정된 시간 제약**을 가짐
 - 제한된 시간 내에 **반드시** 처리되어야 함
 - 시간 제약을 만족해야만 올바른 동작

오픈 소스 운영체제

- ❑ **소스 코드**를 제공하는 운영체제
- ❑ 저작권과 DRM (Digital Rights Management)에 대항
- ❑ **GPL**(GNU Public License)의 **copyleft**를 갖는 **FSF**(Free Software Foundation)에 의해 시작
 - copyleft는 copyright(저작권)의 반대 개념으로 저작권에 기반을 둔 사용 제한이 아니라 저작권을 기반으로 한 정보의 공유를 위한 조치
- ❑ **GNU/Linux, BSD UNIX** (Mac OS X에 포함된 코어) 등
- ❑ **VirtualBox, VMware Player** 같은 VMM을 사용할 수 있음
 - 시험 및 탐색을 위한 게스트 운영체제의 실행을 위해 사용

□ 다음 계산 환경 중 하나를 선택하여 조사하고, 사용 예를 기술하라.

- 분산 시스템 (Distributed System)
- 클라이언트 서버 계산 (Client-Server Computing)
- 피어 간 계산 (Peer-to-Peer Computing)
- 가상화 (Virtualization)
- 클라우드 컴퓨팅 (Cloud Computing)