2장. 운영체제 구조
(Operating System Structure)

순천향대학교 컴퓨터공학과
이 상 정

강의 목표 및 내용

- 목표
  - 사용자, 프로세스 및 다른 시스템에게 제공하는 서비스
  - 운영체제 구성 요소
  - 운영제제 설치 및 부팅 과정
- 내용
  - 운영체제 서비스
  - 사용자 운영체제 인터페이스
  - 시스템 콜
  - 시스템 프로그램
  - 운영체제 설계 및 구현
  - 운영체제 구조
  - 운영체제 디버깅
  - 운영체제 생성
  - 시스템 부팅

운영체제 서비스 (1)

- 운영체제는 프로그램과 프로그램의 사용자에게 아래와 같은 서비스를 제공
  - 사용자 인터페이스 (user interface)
    - 명령어 라인 인터페이스(Command-Line Interface : CLI)
    - 그래픽 사용자 인터페이스(Graphics User Interface: GUI)
    - 일괄처리(batch)
  - 프로그램 수행 (program execution)
    - 프로그램을 메모리에 적재해 실행
    - 정상적이든 혹은, 비정상적이든 (오류를 표시하면서) 실행을 종료
  - 입/출력 연산 (I/O operation)
    - 운영체제가 입/출력 동작의 수단을 제공
  - 파일 시스템 조작 (file system manipulation)
    - 파일 및 디렉토리 생성 및 삭제, 읽고 쓰기, 파일 검색, 접근 허가

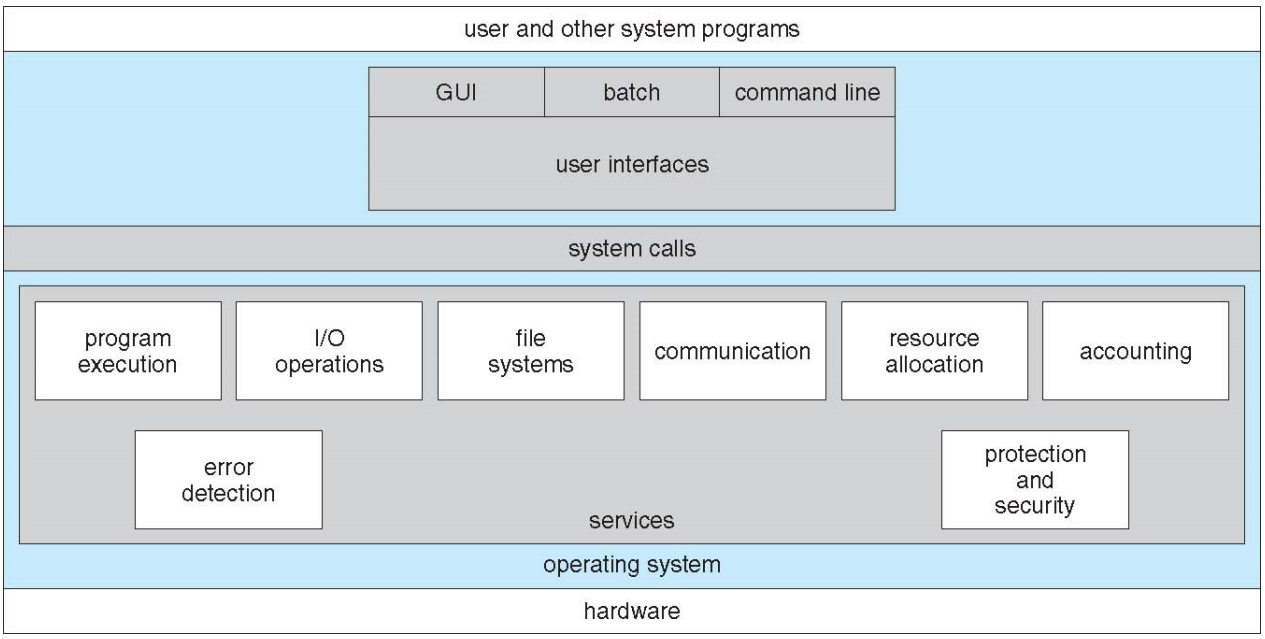
운영체제 서비스 (2)

  - 통신 (communication)
    - 동일한 컴퓨터 또는 네트워크 에 연결된 컴퓨터 상에서 한 프로세스가 다른 프로세스와 정보를 교환하는 수단 제공
    - 통신은 공유 메모리(shared memory)나 메시지 전달(message passing) 기법 등에 구현
  - 오류 탐지 (error detection)
    - 모든 가능한 오류를 탐지하고 적절한 조치
    - CPU, 메모리 하드웨어(메모리 오류, 정전 등처럼), 입/출력 장치(테이프의 패리티 오류, 네트워크의 접속 실패, 또는 프린터의 종이 부족) 오류
    - 사용자 프로그램(연산의 오버플로우, 불법적인 메모리 위치에의 접근 시도, 또는 CPU 시간의 지나친 사용 등) 오류
    - 사용자와 프로그래머에게 디버깅 기능 등을 제공하여 시스템을 효율적으로 사용

운영체제 서비스 (3)

- 운영체제는 자원을 공유하여 시스템 자체의 효율적인 동작을 보장
  - 자원 할당 (resource allocation)
    - 다수의 프로세스나 다수의 작업들이 동시에 실행될 때, 그들 각각에 자원을 할당
    - 자원의 유형: CPU 사이클, 메인 메모리, 파일 저장장치, 입출력 장치
  - 기록 작성 (logging)
    - 각 사용자가 어떤 종류의 컴퓨터 자원을 얼마나 많이 사용하는지를 추적
  - 보호 (protection)와 보안(security)
    - 다중 사용자 컴퓨터 시스템 또는 네트워크로 연결된 컴퓨터 시스템에 저장된 정보의 소유자는 정보의 사용을 통제하길 원함
    - 보호는 시스템 자원에 대한 모든 접근이 통제되도록 보장
    - 보안은 각 사용자가 자원에의 접근 허용을 위한 인증(authentication), 외부 입/출력 장치들을 부적합한 접근 시도로부터 지키고, 침입의 탐지를 위해 모든 연결을 기록하도록 확장

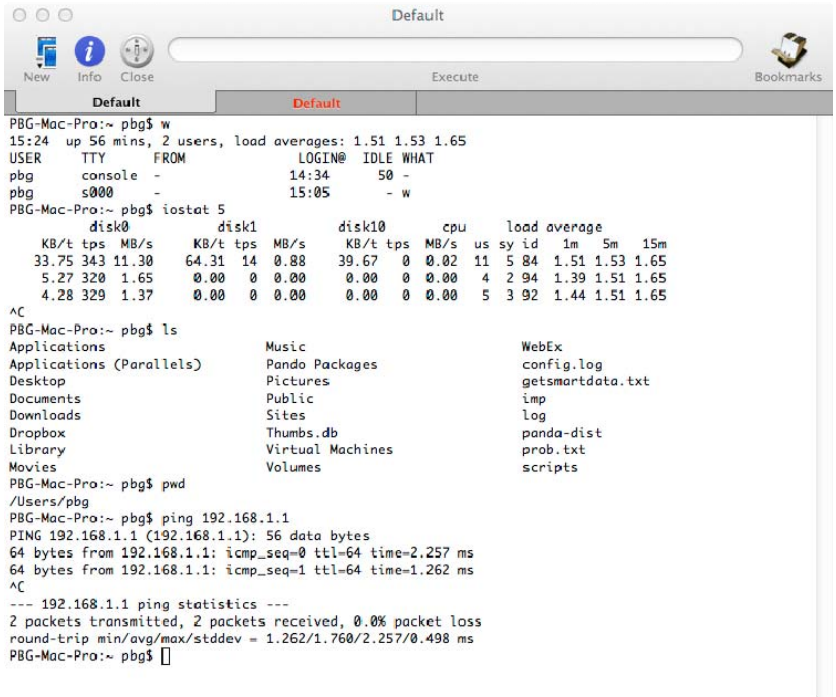
운영체제 서비스 관점



사용자 운영체제 인터페이스 - CLI

- 명령어 인터프리터(command interpreter, CLI)
  - 운영체제가 수행할 명령어를 직접 입력하는 수단 제공
  - 명령어 인터프리터의 주요 기능은 사용자가 지정한 명령을 해석하고 수행하는 것
  - 커널에 구현되거나 시스템 프로그램으로 제공
  - 선택할 수 있는 여러 명령어 인터프리터를 제공하는 시스템에서 이러한 인터프리터는 셸(shell)이라고 함
    - UNIX나 Linux시스템에서는 Bourne shell, C shell, Bourne-Again shell, Korn shell
  - 두 가지 구현 방법
    - 명령어 인터프리터 자체가 명령을 실행할 코드를 내장
    - 시스템 프로그램에 의해 대부분의 명령을 구현
      - 명령어 인터프리터는 명령에 해당하는 시스템 프로그램을 찾아서 실행

Bourne 셸 명령어 인터프리터



사용자 운영체제 인터페이스 – GUI

- 그래픽 사용자 인터페이스(Graphic User Interface, GUI)
  - 마우스를 기반으로 하는 윈도우 메뉴 시스템
    - 아이콘이 파일, 프로그램, 시스템 기능 등을 표현
  - 1970년 대 초 Xerox PARC 연구 센터 개발
    - 1973년에 출시된 Xerox Alto 컴퓨터에 처음 등장

- 대부분의 시스템이 CLI와 GUI 모두 제공
  - Microsoft Windows는 CLI "command" 와 함께 GUI 제공
  - Apple Mac OS X 는 UNIX 커널의 하부구조와 CLI 셸과 함께 " Aqua " GUI 제공
  - UNIX/Linux
    - 다양하고 강력한 셸 인터페이스 제공
    - Common Desktop Environment(CDE), K Desktop Environment(KDE), GNU 프로젝트의 GNOME

Mac OS X GUI

사용자 운영체제 인터페이스 – 터치스크린 (Touchscreen)

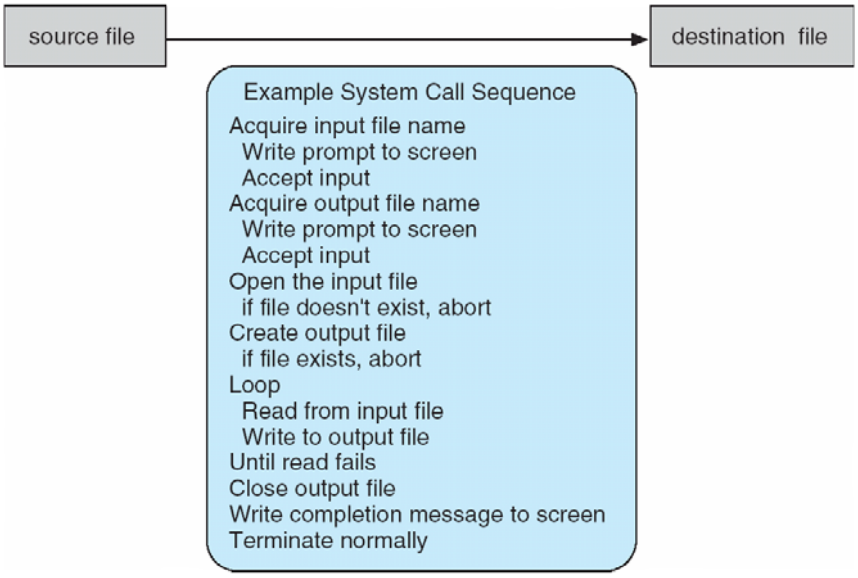
- 터치 스크린 디바이스
  - 마우스 필요 없음
  - 제스처 기반 동작 및 선택
  - 텍스트 입력을 위한 가상 키보드
  - 음성 명령

시스템 콜 (System Call)

- 운영체제에 의해 사용 가능하게 된 서비스에 대한 인터페이스를 제공
- 일반적으로 C와 C++ 언어로 작성된 루틴 형태로 제공
- 프로그램이 직접 시스템 콜(호출)을 사용하기 보다는 응용 프로그래밍 인터페이스(Application Programming Interface, API)를 거쳐서 접근
  - 프로그램의 호환성
  - 사용이 용이
- 응용 프로그래머가 사용 가능한 가장 흔한 세 가지 API
  - Windows 시스템을 위한 Win32 API
  - POSIX (Portable Operation System Interface) 기반 시스템을 위한 POSIX API(거의 모든 버전의 UNIX, Linux 및 Mac OS X를 포함)
  - Java 가상 기계에서 수행될 수 있는 프로그램을 위한 Java API

시스템 콜 예

- 한 파일로부터 데이터를 읽어서 다른 파일로 복사하는 연속된 시스템 콜 예



표준 API의 예

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

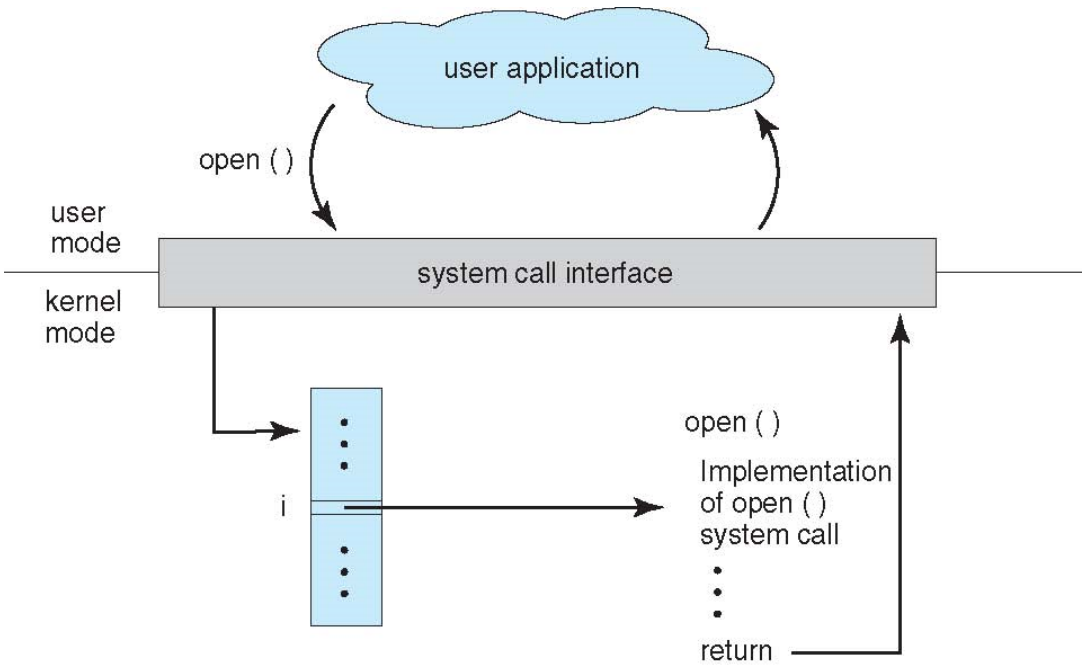
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

시스템 콜 구현

- 통상 각 시스템 콜에는 번호가 할당
  - 시스템 콜 인터페이스는 이 번호에 따라 인덱스되는 테이블을 관리
- 시스템 콜 인터페이스는 의도하는 시스템 콜을 부르고 시스템 콜의 상태와 반환 값을 리턴
- 호출자는 시스템 콜이 어떻게 구현되고 실행 중 무슨 작업을 하는지 아무 것도 알 필요가 없음
  - API를 준수하고 시스템 콜의 결과로 운영체제가 무엇을 해 줄 것인지만 이해하면 됨
- 운영체제 인터페이스에 대한 대부분의 자세한 내용은 API에 의해 프로그래머로부터 숨겨짐
  - 실행시간 지원 라이브러리(run-time support library)에 의해 관리

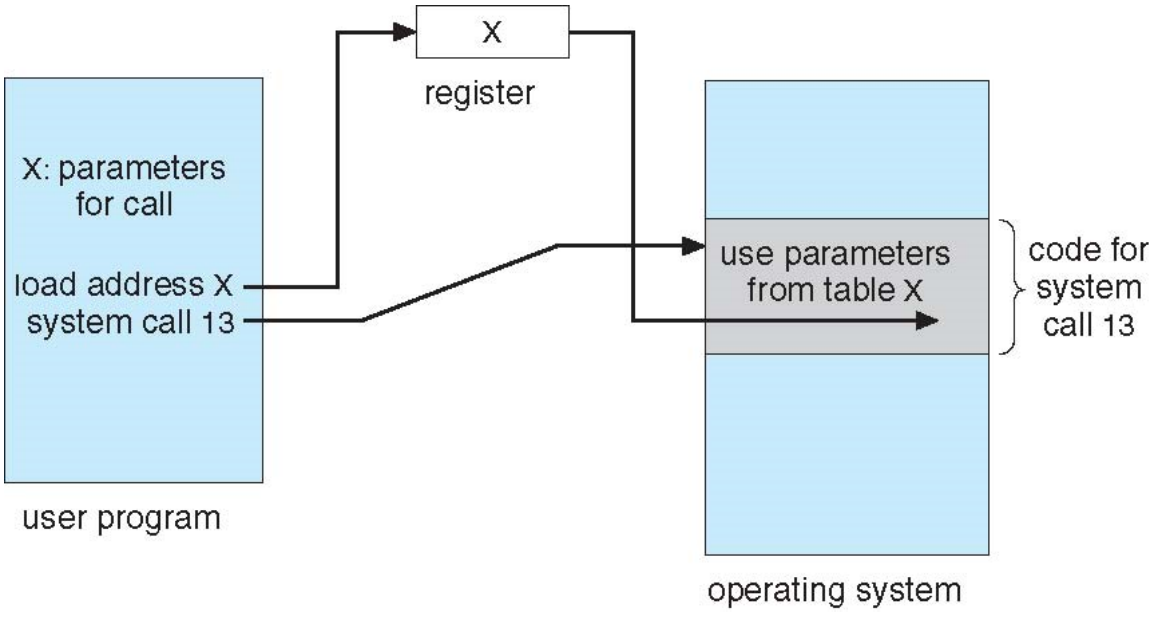
사용자 응용의 open() 시스템 콜 처리 예



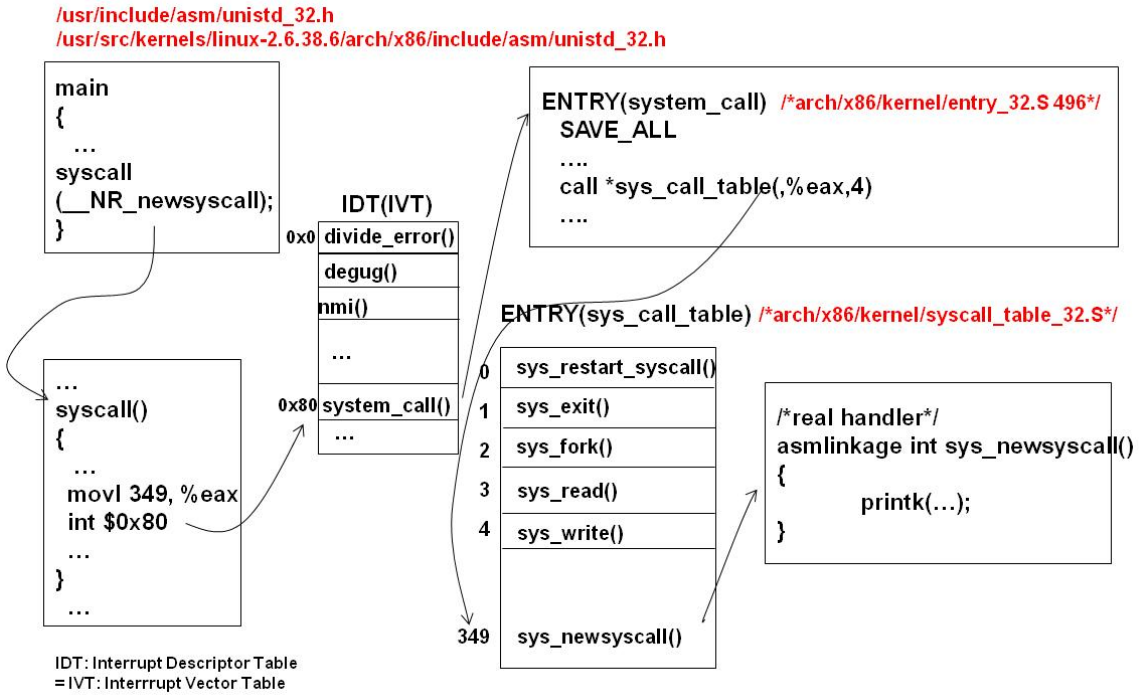
시스템 콜 매개변수 전달

- 시스템 콜 시 운영체제에 매개변수를 전달하기 위해서 세 가지 일반적인 방법
  - 가장 간단한 방법은 매개변수를 레지스터 내에 전달
  - 메모리 내의 블록이나 테이블에 저장되고, 블록의 주소가 레지스터 내에 매개변수로 전달
    - Linux
  - 프로그램에 의해 스택(stack)에 넣어질(push) 수 있고, 운영체제에 의해 꺼내짐(pop off)
  - 블록이나 스택 방법은 매개변수의 개수나 길이에 제한이 없음

테이블로 매개변수 전달



리눅스 시스템 콜 과정



시스템 콜의 유형 – 프로세스 제어

- 시스템 콜은 프로세스 제어, 파일 조작, 장치 조작, 정보 유지 보수와 통신 등으로 분류

- 프로세스 제어(process control)
  - 끝내기(end), 중지(abort)
  - 적재(load), 실행(execute)
  - 프로세스 생성(create process or submit job), 프로세스 종료
  - 프로세스 속성(attribute) 획득, 프로세스 속성 설정
  - 일정 시간을 기다림 (wait time)
  - 이벤트를 기다림(wait event), 이벤트를 알림(signal event)
  - 메모리 할당 및 자유화
  - 오류를 찾기 위한 디버거
  - 프로세스 간 공유 데이터 접근을 관리하는 락(lock)

시스템 콜의 유형 – 파일조작, 장치관리

- 파일 조작 (file manipulation)
  - 파일 생성(create file), 파일 삭제(delete file)
  - 열기(open), 닫기(close)
  - 읽기, 쓰기, 위치 변경(reposition)
  - 파일 속성 획득 및 설정
- 장치 관리(device management)
  - 장치를 요구(request device), 장치를 방출(release device)
  - 읽기, 쓰기, 위치 변경(reposition)
  - 장치 속성 획득, 장치 속성 설정
  - 장치의 논리적 부착(attach) 또는 분리(detach)

시스템 콜의 유형 – 정보 유지, 통신

- 정보 유지 (information maintenance)
  - 시간과 날짜의 설정과 획득
  - 시스템 데이터의 설정과 획득
  - 프로세스, 파일, 장치 속성의 획득
  - 프로세스, 파일, 장치 속성의 설정
- 통신 (communication)
  - 프로세스 간 통신 연결의 생성, 제거
    - 메시지 전달 모델: 호스트 이름, 프로세스 이름으로 메시지 전달
    - 공유 메모리 모델: 특정 공유 메모리 영역 접근으로 통신
  - 메시지의 송신, 수신
  - 상태 정보 전달
  - 원격 장치의 부착 및 분리

시스템 콜의 유형 – 보호

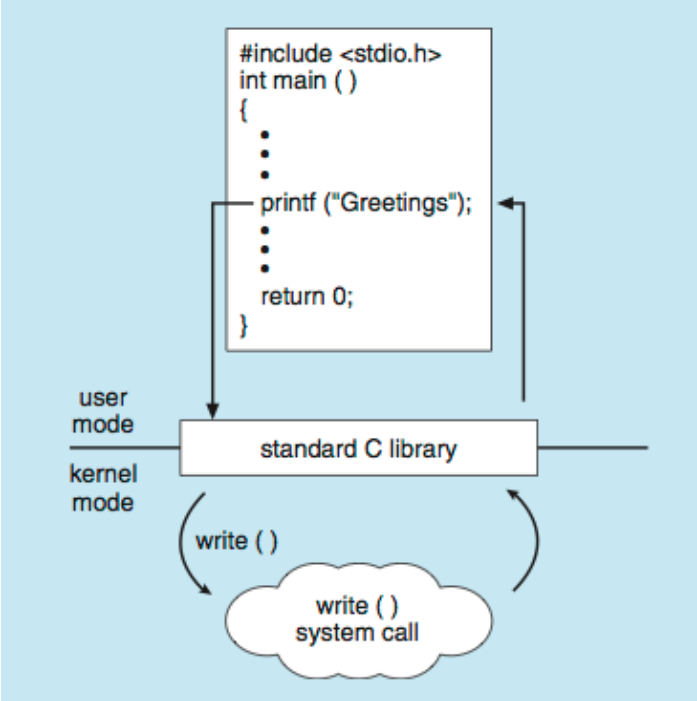
- 보호 (protection)
  - 자원에 대한 접근 제어
  - 사용 권한(permission) 획득
  - 사용 권한 설정

윈도우즈와 Unix 시스템 콜 예

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

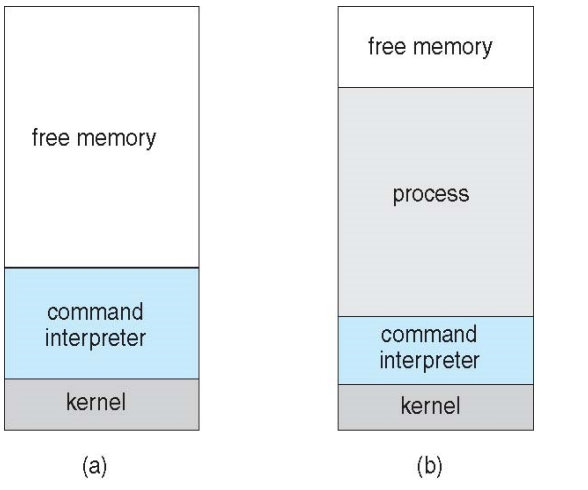
표준 C 라이브러리의 예

- printf() 문을 호출하는 C 프로그램을 예
  - write() 시스템호출



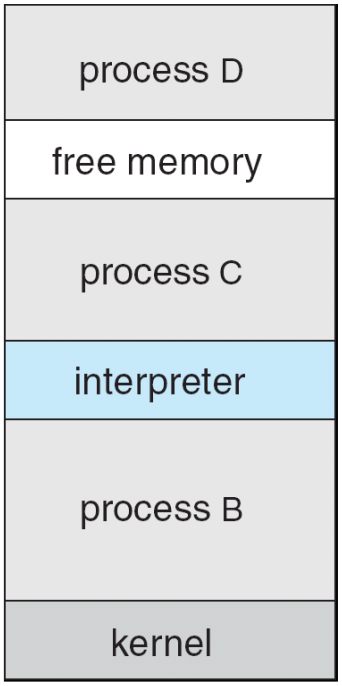
MS-DOS 실행 예

- 단일 태스킹 시스템
  - MS-DOS, Arduino
- MS-DOS
  - 시스템 부팅 시 셸 실행
  - 단일 프로그램 실행
    - 새로운 프로세스 생성 안됨
  - 단일 메모리 사용 영역
  - 새 프로그램은 기존 프로그램의 메모리 영역에 적재



FreeBSD 실행 예

- Unix 변형
- 다중 태스킹(multitasking) 시스템
- 사용자 로그인하면 선택된 셸 실행
- 셸은 프로세스 생성을 위해 fork() 시스템 콜
  - 프로그램을 프로세스로 적재하기 위해 exec() 실행
  - 셸은 프로세스가 종료 되기를 기다리거나 사용자 명령을 수행
- 프로세스 종료
  - code = 0 => 에러 없음
  - code > 0 => 에러 발생



시스템 프로그램 (System Program)

- 시스템 프로그램은 프로그램 개발과 실행을 위해 보다 편리한 환경을 제공
  - 파일 관리: 파일과 디렉토리를 생성, 삭제, 조작
  - 상태 정보: 시스템 상태 정보 (메모리,디스크 공간, 사용자, 성능, 디버깅 …)
  - 파일 변경: 문장 편집기(text editor) 등
  - 프로그래밍 언어: 컴파일러, 어셈블러, 인터프리터 (interpreter)
  - 프로그램 적재와 수행: 적재기(loader)
  - 통신: 프로세스, 사용자 그리고 다른 컴퓨터 시스템들 사이에 가상 접속을 이루기 위한 기법을 제공
- 대부분의 사용자가 보는 운영체제의 관점은 실제의 시스템 콜에 의해서 보다는 시스템 프로그램과 응용에 의해 정의

운영체제 설계 및 구현 (1) (OS Design and Implementation)

- 각 운영체제의 내부구조는 아주 다양

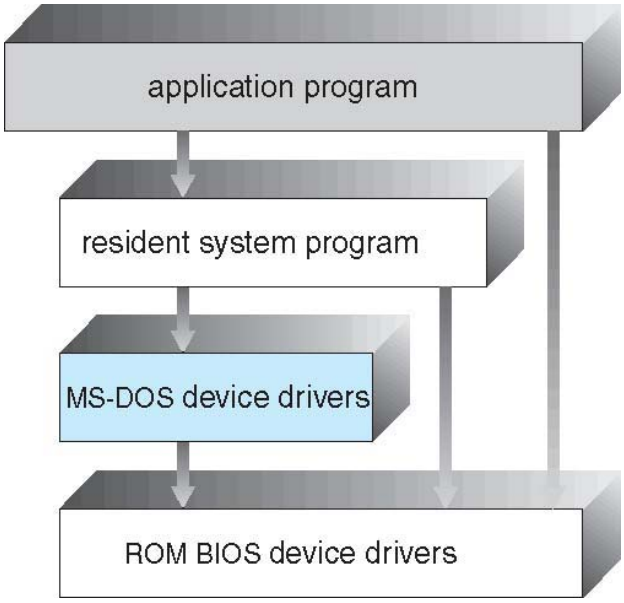
- 설계 목표 (design goal)
  - 설계는 시스템의 목표(goal)와 명세(specification) 정의부터 시작
  - 설계는 하드웨어와 시스템 유형(일괄 처리, 시분할, 단일 사용자,다중 사용자, 분산, 실시간, 혹은 범용)의 선택에 의해 영향
  - 사용자 목표 (user goal)
    - 운영체제는 사용하기 쉽고 편리하며, 배우기 쉽고, 믿을 수 있고, 안전하고, 신속해야 함
  - 시스템 목표 (system goal)
    - 운영체제는 설계, 구현, 유지 보수가 쉬워야 하며 또한, 적응성, 신뢰성, 무오류, 효율성을 가져야 함

운영체제 설계 및 구현 (2)

- 메커니즘과 정책 (mechanism and policy)
  - 메커니즘(기법)은 어떤 일을 어떻게(how) 할 것인가를 결정하는 것이고, 정책은 무엇(what)을 할 것인가를 결정하는 것
    - 예를 들면, 타이머 구조는 CPU 보호를 보장하기 위한 메커니즘이지만, 특정 사용자를 위해 타이머를 얼마나 오래 동안 설정할지를 결정하는 것은 정책적 결정
  - 중요한 원칙은 메커니즘으로부터 정책을 분리하는 것
    - 융통성(flexibility)을 위해 아주 중요
    - 정책은 장소가 바뀌거나 시간이 흐름에 따라 변경될 수 있음
    - 최악의 경우, 정책의 각 변경이 저변에 깔려 있는 메커니즘의 변경을 요구
    - 정책의 변경에 민감하지 않은 일반적인 메커니즘이 보다 바람직
- 초기 운영체제는 어셈블리어로 구현되나, 최근에는 C나 C++와 같은 고급 언어로 구현

운영체제 구조 (OS Structure) – 간단한 구조

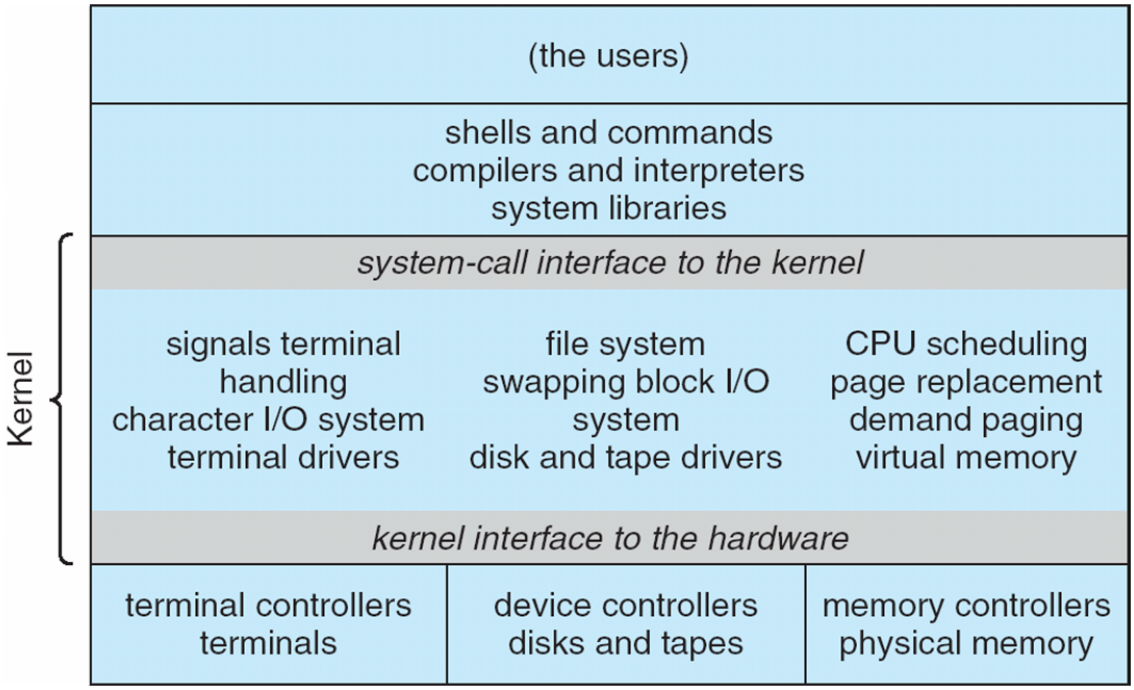
- 소형이면서 간단하고 제한된 시스템의 운영체제
- MS-DOS
  - 최소의 공간에 최대의 기능들을 제공하도록 작성
  - 모듈들로 분할되지 않음
  - 인터페이스와 기능 계층이 분리되지 않음



모노리식 구조 (Monotithic Structure)

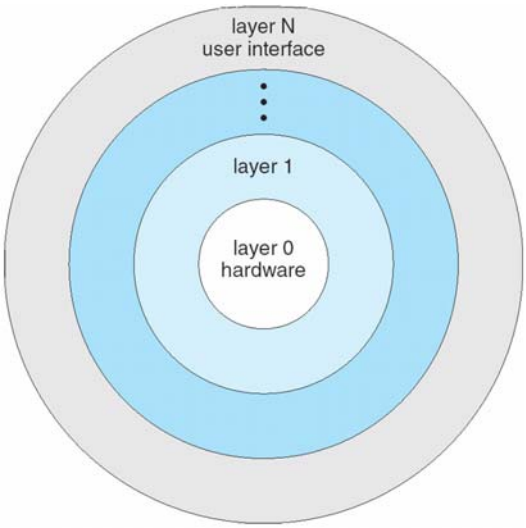
- 특정한 구조 없이 커널의 모든 기능을 단일 주소 공간에서 실행되는 단일 정적 이진 파일로 구현
  - 시스템 콜 인터페이스 오버헤드가 거의 없고, 커널 안에서의 통신 속도 빠름
  - 운영체제의 모든 기능을 하나의 계층으로 결합하여 구현 및 확장, 유지보수가 어려움
- 모노리식의 예로 초기의 전통적인 UNIX 운영체제
  - UNIX는 커널과 시스템 프로그램으로 구성
  - 커널 (kernel)
    - 시스템 콜 인터페이스 아래와 물리적 하드웨어 위의 모든 것이 커널
    - 커널은 시스템 콜을 통해 파일 시스템, CPU 스케줄링, 메모리 관리 그리고 다른 운영체제 기능을 제공

모노리식 구조– 전통적인 UNIX 시스템 구조



계층적 접근 (Layered Approach)

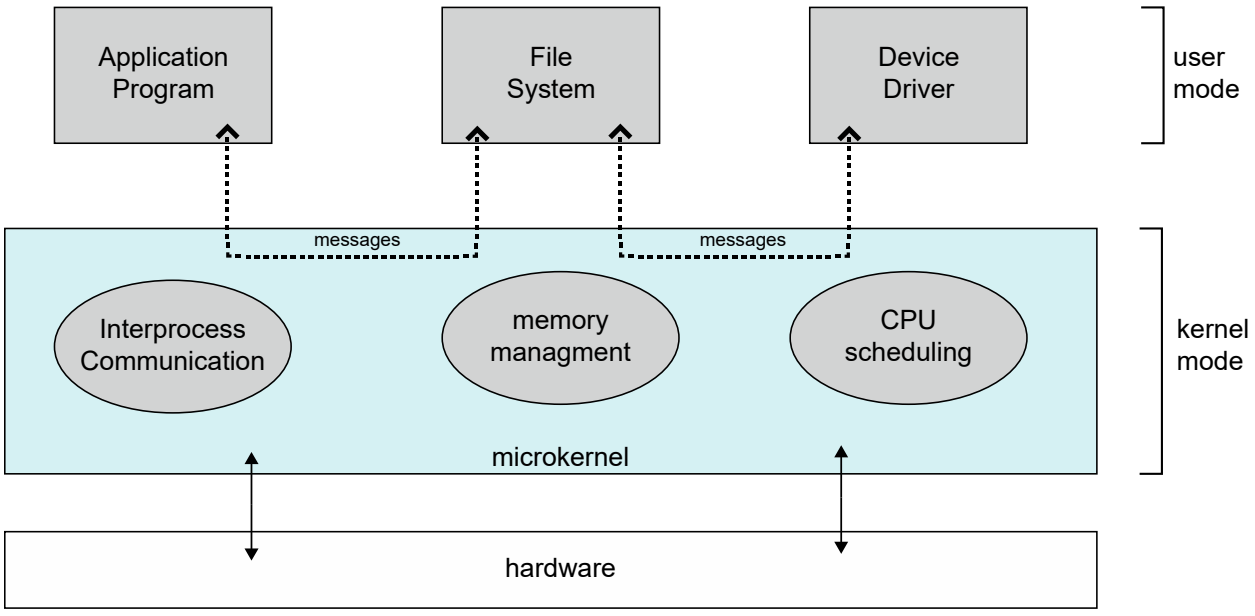
- 운영체제를 여러 개의 계층 분할
- 최하위 계층(계층 0)은 하드웨어이고 최상위 계층(계층 N)은 사용자 인터페이스
- 계층적 접근 방식의 주된 장점은 구현과 디버깅의 간단함
  - 계층들은 단지 자신의 하위 계층들의 서비스와 기능(연산)들만을 사용하도록 선택
- 계층적 접근 방법의 가장 어려운 점은 여러 계층을 적절히 정의하는 것



마이크로커널 (Microkernel)

- 모든 중요치 않은 구성 요소를 커널로부터 제거하고, 그들을 시스템 및 사용자 수준 프로그램으로 구현하여 운영체제를 구성
  - 간소화된 커널
  - 메시지 전달(message passing)을 사용자 공간에서 수행되는 다양한 서비스간에 통신
  - 1980년대 중반 카네기멜론 대학의 Mach OS
    - Mac OS X 커널(Darwin)의 부분이 Mach 적용
  - 장점
    - 운영체제의 확장이 용이
    - 한 하드웨어로부터 다른 하드웨어로 이식이 용이
    - 높은 보안성과 신뢰성을 제공
  - 단점
    - 사용자 공간과 커널 공간과의 통신 오버헤드 때문에 성능이 감소

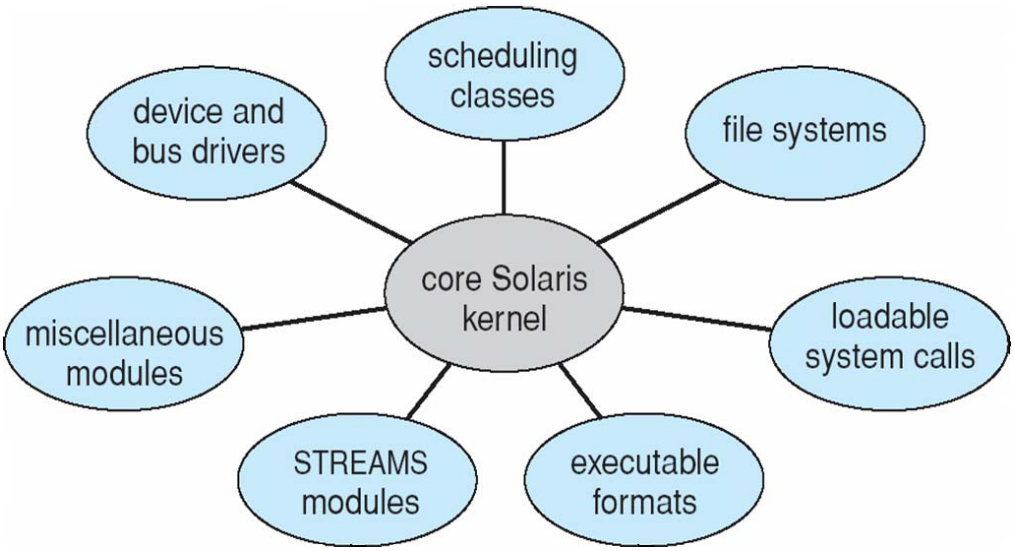
마이크로커널 시스템 구조



모듈 (Module)

- 최근의 운영체제는 적재가능 커널 모듈(loadable kernel module)로 구현
  - 객체 지향 프로그래밍 기법을 사용
  - 커널은 핵심적인 구성 요소의 집합을 가지고 있고 부팅 때 또는 실행 중에 부가적인 서비스들을 링크
    - 동적 적재 모듈을 사용
- 커널의 각 부분이 정의되고 보호된 인터페이스를 가진다는 점에서 계층적 구조와 비슷하지만 임의의 모듈을 호출할 수 있다는 점에서 계층적 구조보다 유연함
- 중심 모듈은 단지 핵심 기능만을 가지고 있다는 점에서는 마이크로커널과 유사하지만 통신하기 위하여 메시지 전달을 호출할 필요가 없기 때문에 더 효율적
- Solaris, Linux 및 Mac OS X 등의 현대 UNIX의 구현에 일반적인 추세

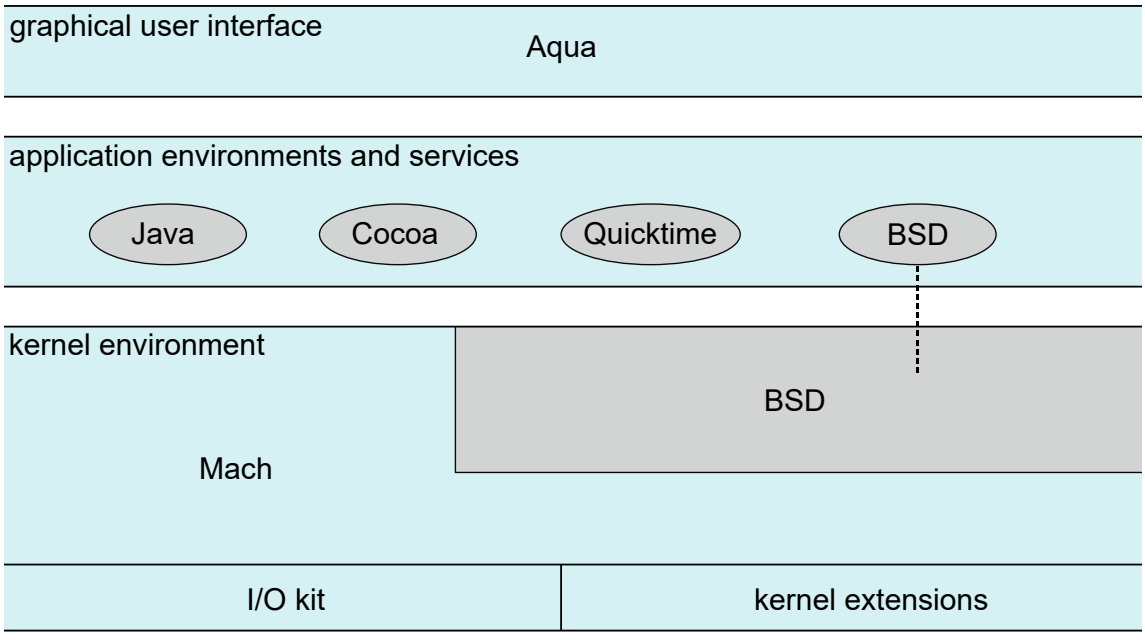
Solaris 적재 가능 모듈



하이브리드 시스템 (Hybrid System)

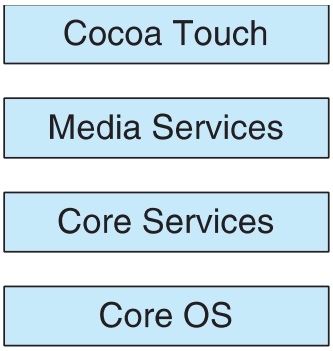
- 현대 운영체제는 하나의 순수한 모델이 아닌 혼성 구조
  - 성능, 보안, 사용 요구 등을 충족하기 위해 하이브리드 구조를 채택
  - 리눅스, 솔라리스는 커널 주소 공간에 있는 커널들의 모노리딕에 기능들을 동적으로 적재하는 모듈을 사용
  - 윈도우즈는 모노리식 구조에 분리된 서브시스템을 지원하는 마이크로커널 형태를 유지
  - Apple Macintosh Mac OS X (Darwin) 운영체제는 혼성 구조를 사용
    - 최상위 계층은 응용 환경과 응용에 그래피컬 인터페이스를 제공하기 위한 서비스의 집합
      - Aqua UI + Cocoa 프로그래밍 환경
    - 커널은 Mach 마이크로커널과 BSD Unix로 구성
      - Mach는 메모리 관리, 원격 함수 호출(remote procedure call, RPC)과 메시지 전달을 포함한 프로세스간 통신 설비 그리고 스레드 스케줄링을 제공
      - BSD구성 요소는 BSD 명령어 라인 인터페이스와 네트워킹과 파일 시스템 지원, Pthread를 포함한 POSIX API의구현을 제공

Mac OS X 구조



iOS

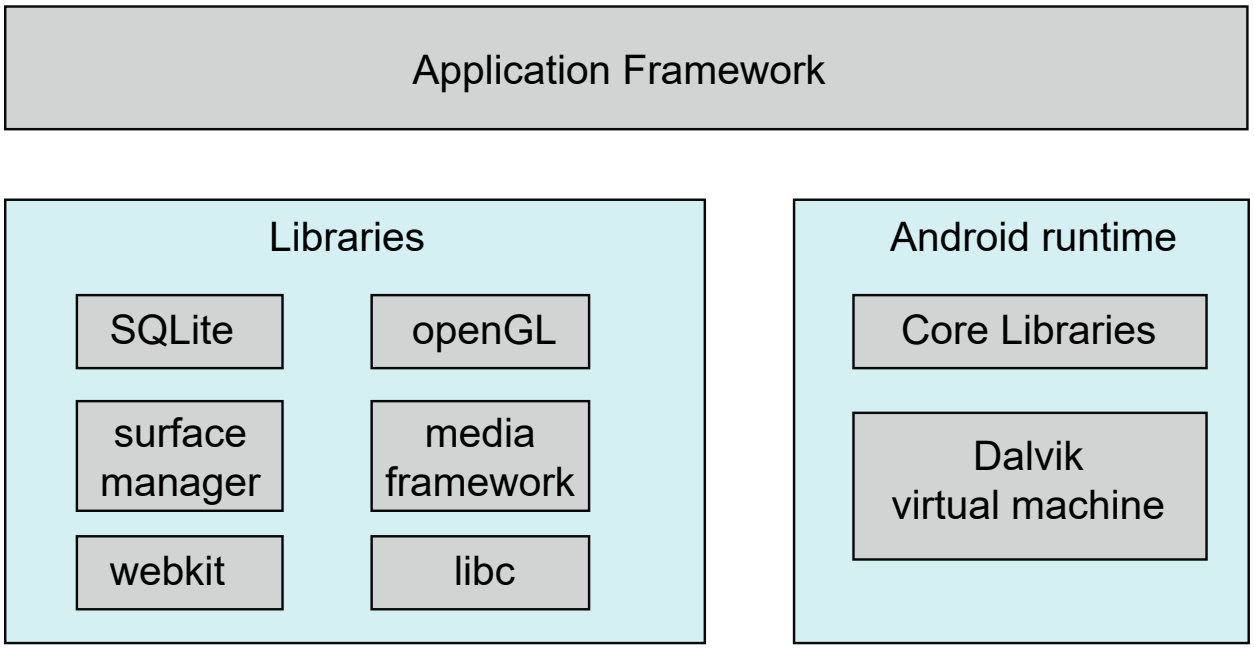
- iPhone, iPad를 위한 애플의 모바일 운영체제
  - Mac OS에 기능(터치 스크린 등) 추가
  - 코코아 터치는 앱 개발을 위한 Objective-C API
  - 미디어 서비스는 그래픽, 오디오, 비디오 계층
  - 코아 서비스 클라우드 컴퓨팅, 데이터베이스 제공
  - 코아 운영체제는 Mac OS X 커널 기반 운영체제



안드로이드 (Andriod)

- 모바일 장치를 위한 소프트웨어 스택
  - 리눅스 운영체제, 미들웨어와 핵심 응용 프로그램들로 구성
  - 안드로이드 SDK 제공
    - 자바 언어로 안드로이드 응용 프로그램을 개발을 위한 API와 툴 제공
    - Dalvik 가상머신
- 오픈소스
  - 기존의 다양한 오픈소스를 기반으로 개발
  - 기존의 오픈소스 라이선스
    - GPL, GPL2, LGPL
    - Apache
    - BSD
    - MIT
    - SGI OpenGL

안드로이드 구조



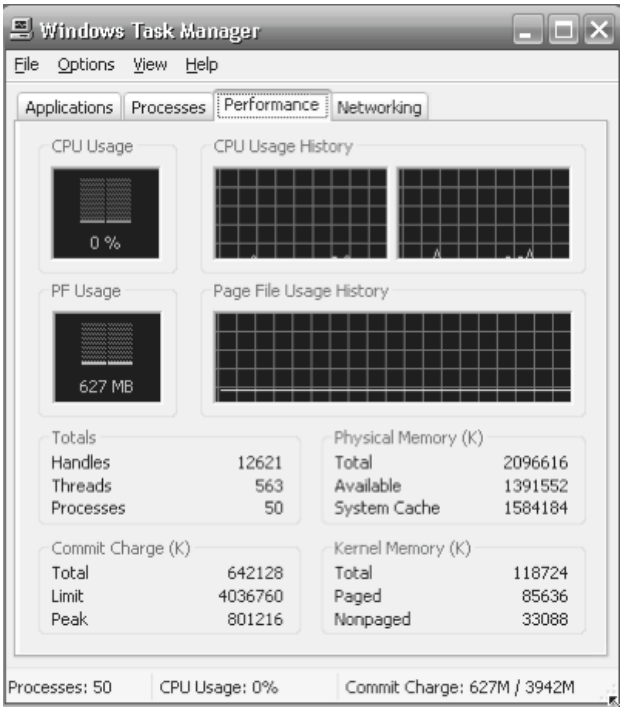
운영체제 디버깅

- 디버깅(debugging)은 오류(error, bug)를 발견하고 수정하는 행위
- 운영체제는 오류 정보를 로그 파일에 기록
- 응용의 오류는 프로세스가 사용하던 메모리를 캡처한 코어 덤프(core dump)를 생성
- 운영체제의 오류는 커널 메모리가 포함된 충돌 덤프(crash dump)를 생성

성능 조정 (performance tuning)

- 병목(bottlenecks) 현상을 제거하여 성능을 향상
- 운영체제는 병목 지점 발견을 위해 시스템 성능을 감시하고, 시스템 동작을 측정할 수 있는 수단을 제공

- 예
  - 유닉스 명령어 top
  - 윈도우 작업 관리자



strace (1)

- 리눅스에서 프로그램의 실행을 추적하는 디버깅 툴
  - 특정 프로그램의 시스템 콜과 시그널을 감시
  - 프로그램 소스 코드를 갖고 있지 않거나, 프로그램의 시작에서 종료될 때까지의 실행 시퀀스를 디버깅할 때 유용하게 사용
- 예: 리눅스 명령어 ls 실행 추적
  - strace ls

```
lee@leeVB:~$ strace ls
execve("/bin/ls", ["ls"], [/* 39 vars */]) = 0
brk(0)                                  = 0x9f17000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb772c000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=61708, ...}) = 0
mmap2(NULL, 61708, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb771c000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0@A\0\0004\0\0\0"..., 512) = 512
```

strace (2)

- strace –e trace=open,read ls
  - open, read 시스템 콜만 추적

```
lee@leeVB:~$ strace -e trace=open,read ls
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/i386-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0@A\0\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320\30\0\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/libacl.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\200\24\0\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0`\n\0\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p[\0\0004\0\0\0"..., 512) = 512
open("/lib/i386-linux-gnu/libattr.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0@\f\0\0004\0\0\0"..., 512) = 512
open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tb"..., 1024) = 326
read(3, "", 1024)                       = 0
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
open("/usr/lib/i386-linux-gnu/gconv/gconv-modules.cache", O_RDONLY) = 3
examples.desktop  hi  hostshare  test  공개  다운로드  문서  바탕화면  비디오  사진  음악  템플릿
lee@leeVB:~$
```


strace (3)

- strace –c ls
  - 통계 정보 표시

```
lee@leeVB:~$ strace -c ls
examples.desktop  hi  hostshare  test  공개  다운로드  문서  바탕화면  비디오  사진  음악  템플릿
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  -nan    0.000000           0         9           read
  -nan    0.000000           0         1           write
  -nan    0.000000           0        11           open
  -nan    0.000000           0        14           close
  -nan    0.000000           0         1           execve
  -nan    0.000000           0         9         9 access
  -nan    0.000000           0         3           brk
  -nan    0.000000           0         2           ioctl
  -nan    0.000000           0         3           munmap
  -nan    0.000000           0         1           uname
  -nan    0.000000           0         9           mprotect
  -nan    0.000000           0         2           rt_sigaction
  -nan    0.000000           0         1           rt_sigprocmask
  -nan    0.000000           0         1           getrlimit
  -nan    0.000000           0        26           mmap2
  -nan    0.000000           0        12           fstat64
  -nan    0.000000           0         2           getdents64
  -nan    0.000000           0         2         1 futex
  -nan    0.000000           0         1           set_thread_area
  -nan    0.000000           0         1           set_tid_address
  -nan    0.000000           0         1           statfs64
  -nan    0.000000           0         1           openat
  -nan    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   114        10 total
lee@leeVB:~$
```

운영체제 생성 (OS Generation)

- 운영체제는 다양한 주변 구성을 가진 다수의 기계에서 수행되도록 설계
  - 따라서 운영체제가 특정 컴퓨터 사이트를 위해 구성되거나 또는 생성되어야 함
  - 이 절차를 시스템 생성(SYSGEN)이라 함
- SYSGEN 프로그램은 하드웨어 시스템의 특정 구성에 관한 정보를 운영자에게 요구하거나 주어진 파일로부터 읽어 들임
  - CPU, 사용 가능한 메모리의 크기, 주변 장치, 운영체제 옵션
- 시스템 정보 결정 후 운영체제 컴파일하여 생성
  - 미리 컴파일된 라이브러리로부터 모듈을 선택하거나 테이블 생성

시스템 부트(boot)

- 부팅(booting)
  - 커널을 적재하여 컴퓨터를 시동하는 절차
- 부트스트랩 프로그램(bootstrap program)
  - 부트스트랩 적재기 (bootstrap loader)라고도 하고 ROM에 저장
  - 커널을 찾고, 그것을 주 메모리에 적재하고, 수행을 시작
  - 기계의 상태를 진단
  - CPU 레지스터, 장치 제어기, 주 메모리의 내용 등 시스템 전반에 걸쳐 초기화
  - GRUB은 리눅스 시스템을 위한 오픈소스 부트스트랩 프로그램
- Windows, Mac OS X 및 Linux 같은 일반적인 운영체제
  - 부트스트랩 적재기는 펌웨어에 있고 운영체제는 디스크에 위치
  - 부트스트랩은 진단 절차를 수행하고 고정된 위치(예를 들면 블록 0, 부트블록)의 디스크 블록 하나를 읽어 메모리에 적재하고 실행

과제: 연습문제

- 연습문제 2.1
- 연습문제 2.4
- 연습문제 2.6

- 풀이는 PPT로 작성하고, 문제도 기술

실습과제 – 파일 복사

1. VirtualBox, 우분투 Linux 설치
   - 보충자료 참고, 설치 과정 포함
   - 자신의 신분(이름, 학번) 표시: 호스트 이름, 프롬트 등
2. Linux 운영체제 상에서 파일을 복사하는 다음 2개의 C 프로그램(표준함수 버전, 시스템 콜 버전)을 작성, 컴파일, 실행한 후 프로그램의 내용분석 설명
   - 프로그램 작성, 컴파일, 실행 과정
   - 프로그램 실행 결과
3. 위의 프로그램을 수정하여 명령행 인수(command line argument)로 파일 이름을 입력받는 프로그램(표준함수 버전, 시스템 콜 버전)을 작성하고 내용 분석 설명
   $ mycp in.txt out.txt
   - strace를 사용하여 프로그램 실행 추적하고 두 개의 결과 비교
     - 사용된 시스템 콜 설명하고 비교

표준함수 버전 (1)

```
#include <unistd.h>
#include <stdio.h>

#define NAME_LENGTH 25

int main(void)
{
    char in_file[NAME_LENGTH], out_file[NAME_LENGTH];
    FILE *in, *out;
    int c;

    printf("Enter source file name: ");
    scanf("%s",in_file);
    printf("Enter destination file name: ");
    scanf("%s",out_file);
```


표준함수 버전 (2)

```
    if ( (in = fopen(in_file,"r")) == NULL) {
        fprintf(stderr,"Cannot open %s for reading₩n",in_file);
        return -1;
    }
    if ( (out = fopen(out_file,"w")) == NULL) {
        fprintf(stderr,"Cannot open %s for writing₩n",out_file);
        return -1;
    }

    while ( (c = getc(in)) != EOF)
        putc(c,out);

    fclose(in);
    fclose(out);

    return 0;
}
```

시스템 콜 버전 (1)

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

#define NAME_LENGTH 25
int main(void){
   char in_file[NAME_LENGTH], out_file[NAME_LENGTH];
   int in, out;
   char c;

   printf("Enter source file name: ");
   scanf("%s",in_file);
   printf("Enter destination file name: ");
   scanf("%s",out_file);
```


시스템 콜 버전(2)

```
   if ( (in = open(in_file,O_RDONLY)) == -1) {
            fprintf(stderr,"Cannot open %s for reading₩n",in_file);
            return -1;
   }
   if ( (out = open(out_file, O_CREAT|O_WRONLY|S_IRWXU)) == -1) {
            fprintf(stderr,"Cannot open %s for writing₩n",out_file);
            return -1;
   }

   while (read(in,&c,1) != 0)
            write(out,&c,1);

   close(in);
   close(out);

   return 0;
}
```