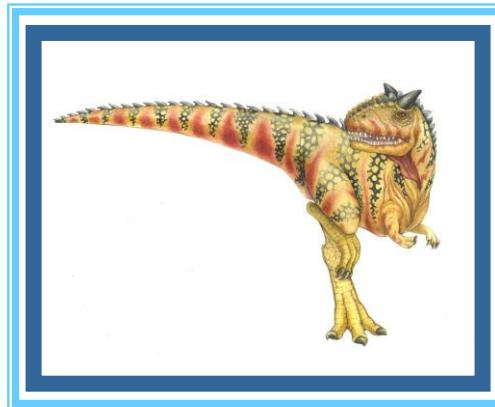


Chapter 2: Operating-System Structures (구조)





Chapter 2: Operating-System Structures

- ❖ 운영체제 서비스 (Operating System Services)
- ❖ 운영체제 사용자 인터페이스 (User Operating System Interface)
- ❖ 시스템 콜 (System Calls)
- ❖ 시스템 콜 유형 (Types of System Calls)
- ❖ 시스템 프로그램 (System Programs)
- ❖ 운영체제 설계 및 구현 (Operating System Design and Implementation)
- ❖ 운영체제 구조 (Operating System Structure)
- ❖ 운영체제 디버깅 (Operating System Debugging)
- ❖ 운영체제 생성 (Operating System Generation)
- ❖ 시스템 부팅 (System Boot)





목표 (Objectives)

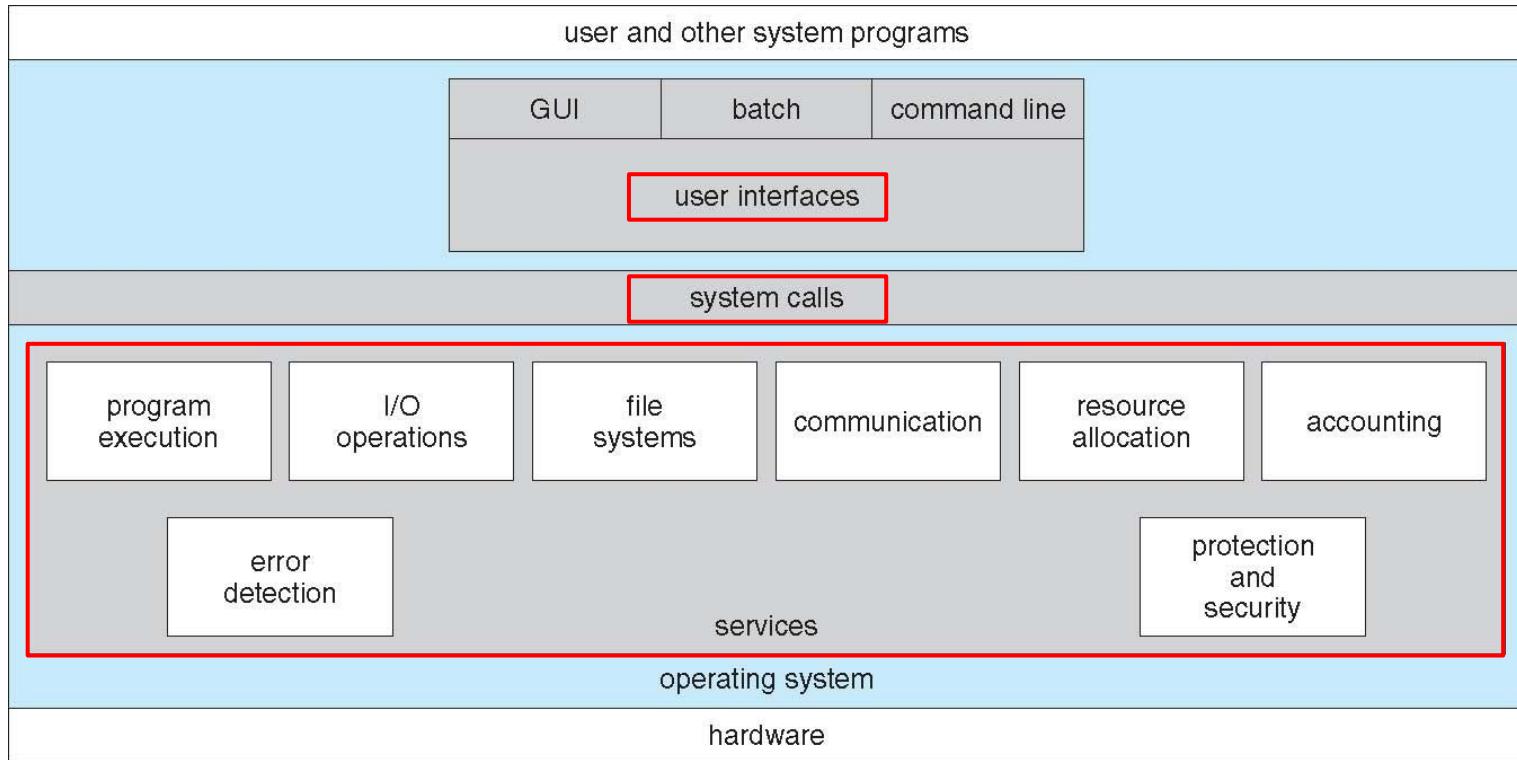
- ❖ 운영체제가 사용자, 프로세스, 다른 시스템에 제공하는 서비스 설명
 - ❖ 운영체제를 구성하는 다양한 방법에 대해 논의
 - ❖ 운영체제 설치, 맞춤화 과정(customization), 부팅하는 방법
-
- ❖ To describe the services an operating system provides to users, processes, and other systems
 - ❖ To discuss the various ways of structuring an operating system
 - ❖ To explain how operating systems are installed and customized and how they boot





운영체제 서비스 : View

- ❖ 운영체제는 사용자(또는 프로그램)들에게 프로그램의 실행 환경을 제공한다.
- ❖ 사용자 프로그램의 적절한 실행과 시스템의 효율적 운용을 담당한다.





운영체제 서비스 (Operating System Services) 1

❖ 사용자 업무를 수행하는 데 필요한 기능을 제공하는 운영체제 서비스

| 사용자 인터페이스

- ▶ 명령어 라인 인터페이스 (Command Line Interface)
 - 문자열 명령 입력
- ▶ 배치 인터페이스 (Batch Interface)
 - 명령어 또는 명령어 집합을 파일로 제공
- ▶ GUI (Graphical User Interface)



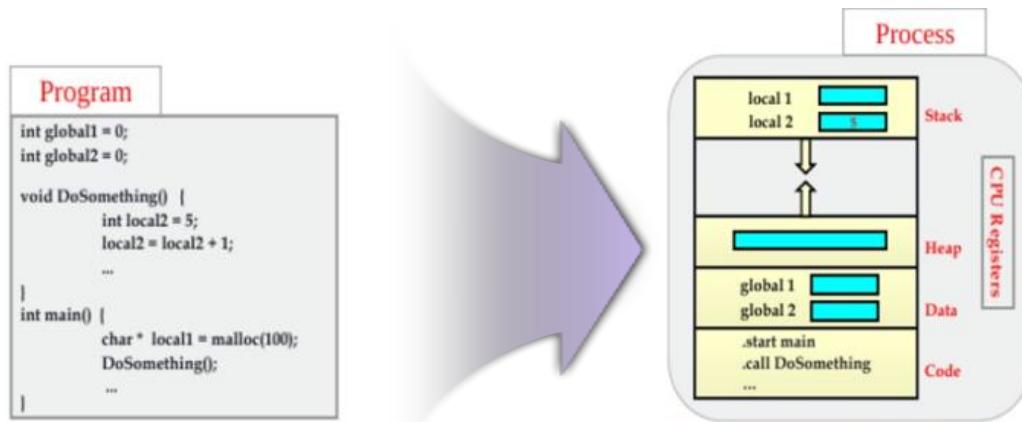
```
//FMBAT EXEC PGM=FILEMGR
//STEPLIB DD
DISP=SHR,DSN=HLQ.SFMNMOD1
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
$$FILEM TPIMP DSNIN=FMN.XMLINP,
$$FILEM MEMBER=*, 
$$FILEM DSNOUT=FMN.IMPORT.TEMPLATE,
$$FILEM REPLACE=YES
/*
```

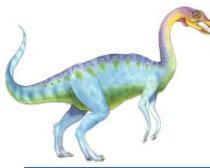




운영체제 서비스 (Operating System Services) 2

- ❖ 사용자 업무를 수행하는 데 필요한 기능을 제공하는 운영체제 서비스
 - | 프로그램 실행 (Execution)
 - ▶ 프로그램을 메모리에 적재, 실행, 실행 종료 (정상 또는 비정상)
 - | 입출력 연산 (I/O)
 - ▶ 프로그램 실행에서 요구하는 파일 또는 입출력 장치 등의 입출력 서비스

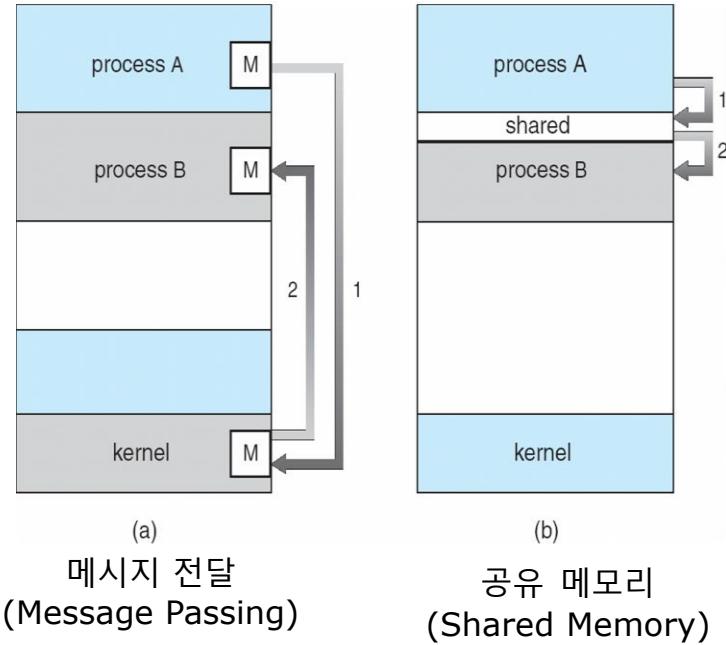




운영체제 서비스 (Operating System Services) 3

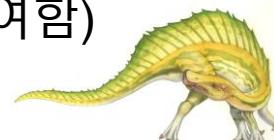
❖ 사용자 업무를 수행하는 데 필요한 기능을 제공하는 운영체제 서비스 | 파일 시스템

- ▶ 파일과 디렉토리 생성과 삭제
- ▶ 파일과 디렉토리 읽고 쓰기
- ▶ 파일과 디렉토리 찾기
- ▶ 파일 정보 보기
- ▶ 파일 접근 권한 관리



| 통신

- ▶ 동일한 컴퓨터 또는 네트워크로 연결된 컴퓨터에 있는 프로세스 간 정보 교환
- ▶ 통신은 **메시지 전달** 기법 (운영체제가 패킷을 이동하는데 관여함)
또는 **공유 메모리를** 이용





운영체제 서비스 (Operating System Services) 4

- ❖ 사용자 업무를 수행하는 데 필요한 기능을 제공하는 운영체제 서비스
 - | 오류 탐지 (Error Detection)

- ▶ 운영체제는 가능한 모든 오류를 인지하고 있어야 함
 - ▶ 오류는 CPU, 메모리, 입출력 장치, 사용자 프로그램 등에서 발생 가능
 - ▶ 운영체제는 올바르고 일관된 계산을 보장하기 위해 각 타입의 오류에 대해 올바른 조처를 취해야 한다.
 - ▶ 디버깅 장치는 사용자와 개발자의 효율적인 시스템 사용을 향상시킬 수 있다.

The screenshot shows a debugger window with the following details:

- Registers:**

```
PSW=0F01 AD=4014A44C AI=4014A3F4 A2=4000058E A3=00000000  
PC=4000058E MDR=400002CD LIR=F1700B20 LAR=40005E42 SP=4012E744  
MDRQ=00000000 MCRH=00000000 MCRL=00000000 MCWF=00000000
```
- Task List:**

```
task4:  
TEST.C:00100: [  
move [d2,d3,a2,a3],(sp)  
Break  
>od task  
id taskentry pri bpri sts wait wobj timeout act wup sus exinf  
1 _task1 1 1 RAI SLP 0 0 0 0  
2 _task2 2 2 RAI SEM 1 0 0 0 0  
3 _task3 3 3 RAI FLG 1 0 0 0 0  
4 _task4 4 4 RUN 0 0 0 0 0  
5 _task5 DRT 0 0 0 0 0 0 0
```
- Semaphores:**

```
>od sem  
id attr semcont isemcont maxsem wtasklist  
1 TF 0 0 1 _task2( 2)  
2 TF 0 0 1  
3 TF 0 0 1
```
- File Tables:**

```
>od fil  
id attr IFLGPTN FLGPTN wtasklist MFLGPTN mode  
1 TF|MU 00000000 00000000 _task3( 3) 00001111 OR  
2 TF|MU 00000000 00000000  
3 TF|MU 00000000 00000000
```





운영체제 서비스 (Operating System Services) 5

❖ 시스템 자원을 공유하여 시스템의 효율적인 운영을 보장하는 서비스

- | 시스템 자원을 공유하여 시스템의 효율적인 운영을 보장하는 서비스
 - | 자원 할당(resource allocation)
 - ▶ 다수의 사용자나 다수의 작업들이 동시에 실행될 때 각각에 자원을 할당해야 함
 - 자원 종류 - CPU cycles, main memory, file storage, I/O devices.
 - ▶ 자원의 일부는 특별한 할당 코드를 가지며, 동일한 컴퓨터 또는 네트워크로 연결된 컴퓨터에 있는 프로세스 간 정보 교환 가능
 - | 회계(accounting)
 - ▶ 어떤 사용자가 어떤 자원을 얼만큼 쓰는지 기록하고 관리





운영체제 서비스 (Operating System Services) 6

❖ 시스템 자원을 공유하여 시스템의 효율적인 운영을 보장하는 서비스

- | 보호와 보안(protection and security)
 - ▶ 보호는 시스템 자원에 대한 모든 접근이 통제되도록 보장하는 것
 - ▶ 외부(external)로부터 시스템 보안을 유지하기 위해 사용자 인증(User Authentication)이 필요하고 부적합한 접근(invalid access attempts) 시도로부터 입출력 장치를 보호해야 한다.
 - ▶ 시스템 전체에 예방책이 제정되어 한다.

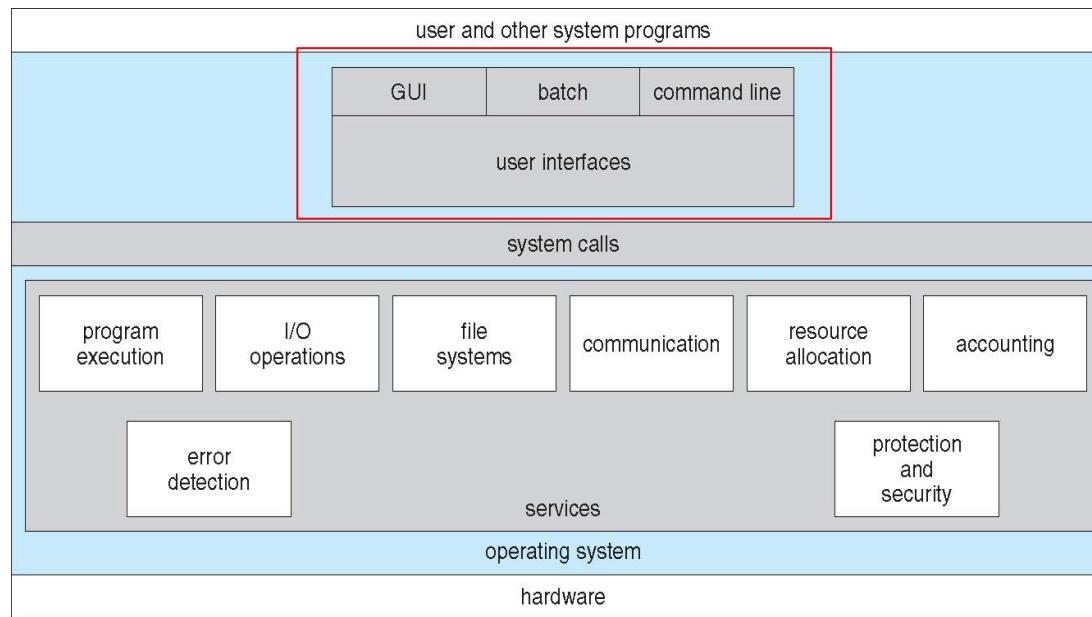




사용자 인터페이스 – CLI

❖ 명령어 해석기 (Command interpreter)

- | CLI, 또는 명령어 해석기를 통해 명령어 직접 입력
- | 커널 또는 시스템 프로그램으로 구현
- | 선택 가능한 여러 개의 명령어 해석기 (**shells**) 제공 – UNIX, Linux

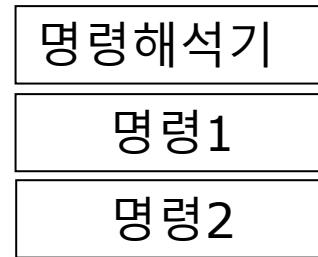
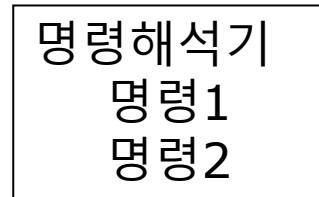


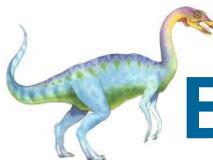


사용자 인터페이스 – CLI

❖ 명령어 해석기 (**Command interpreter**)

- | 역할 : 사용자의 명령어를 가져와 그것을 실행한다.
 - ▶ 명령어 해석기가 명령을 실행하는 코드를 포함하는 경우도 있음
 - 명령어 개수가 프로그램 크기를 결정
 - ▶ 시스템 프로그램으로 명령 구현 (UNIX)
 - 명령 해석기 프로그램 크기가 작아짐
 - 새로운 명령을 추가해도 셸을 변경시킬 필요가 없다.
 - 예 : rm file.txt (p65)





Bourne Shell Command Interpreter

```
Default
New Info Close Execute Bookmarks
Default Default
PBG-Mac-Pro:~ pbgs$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER TTY FROM LOGIN@ IDLE WHAT
pbgs console - 14:34 50 -
pbgs s000 - 15:05 - w
PBG-Mac-Pro:~ pbgs$ iostat 5
          disk0      disk1      disk10      cpu      load average
 KB/t tps MB/s   KB/t tps MB/s   KB/t tps MB/s us sy id 1m 5m 15m
 33.75 343 11.30 64.31 14 0.88 39.67 0 0.02 11 5 84 1.51 1.53 1.65
  5.27 320 1.65 0.00 0 0.00 0.00 0 0.00 4 2 94 1.39 1.51 1.65
  4.28 329 1.37 0.00 0 0.00 0.00 0 0.00 5 3 92 1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbgs$ ls
Applications           Music           WebEx
Applications (Parallels) Pando Packages config.log
Desktop                Pictures         getsmartdata.txt
Documents              Public          imp
Downloads              Sites           log
Dropbox                Thumbs.db       panda-dist
Library                Virtual Machines prob.txt
Movies                 Volumes         scripts
PBG-Mac-Pro:~ pbgs$ pwd
/Users/pbgs
PBG-Mac-Pro:~ pbgs$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbgs$ 
```

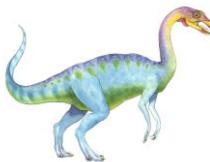




사용자 인터페이스 – GUI

- ❖ 사용자 친화적인 상징 인터페이스(metaphor interface)
 - | 마우스, 키보드와 모니터 사용
 - | **아이콘**이 파일, 프로그램, 액션 등을 표시
 - | 인터페이스 내의 대상에 대한 다양한 마우스 버튼으로 다수의 행위가 발생한다(정보 제공, 선택사항, 실행, 폴더 열기 등)
 - | 1970년대 초 Xerox PARC 연구에서 시작
- ❖ 많은 시스템은 CLI와 GUI 인터페이스를 모두 가지고 있다.
 - | Microsoft 윈도우는 CLI 명령어 셸을 가지고 있는 GUI이다.
 - | Apple Mac OS X는 UNIX 커널과 셸을 가지고 있는 Aqua GUI이다.
 - | Unix와 Linux는 선택적인 GUI(CDE, KDE, GNOME)를 가진 CLI이다.





터치스크린 인터페이스

- ❖ 터치스크린 디바이스는 새로운 인터페이스가 필요
 - | 마우스 사용이 불가능하고 바람직하지 않음
 - | 액션과 선택은 손동작에 근거
 - | 글자 입력을 위해 가상 키보드 (Virtual keyboard for text entry) 사용

iPad touch screen



Springboard





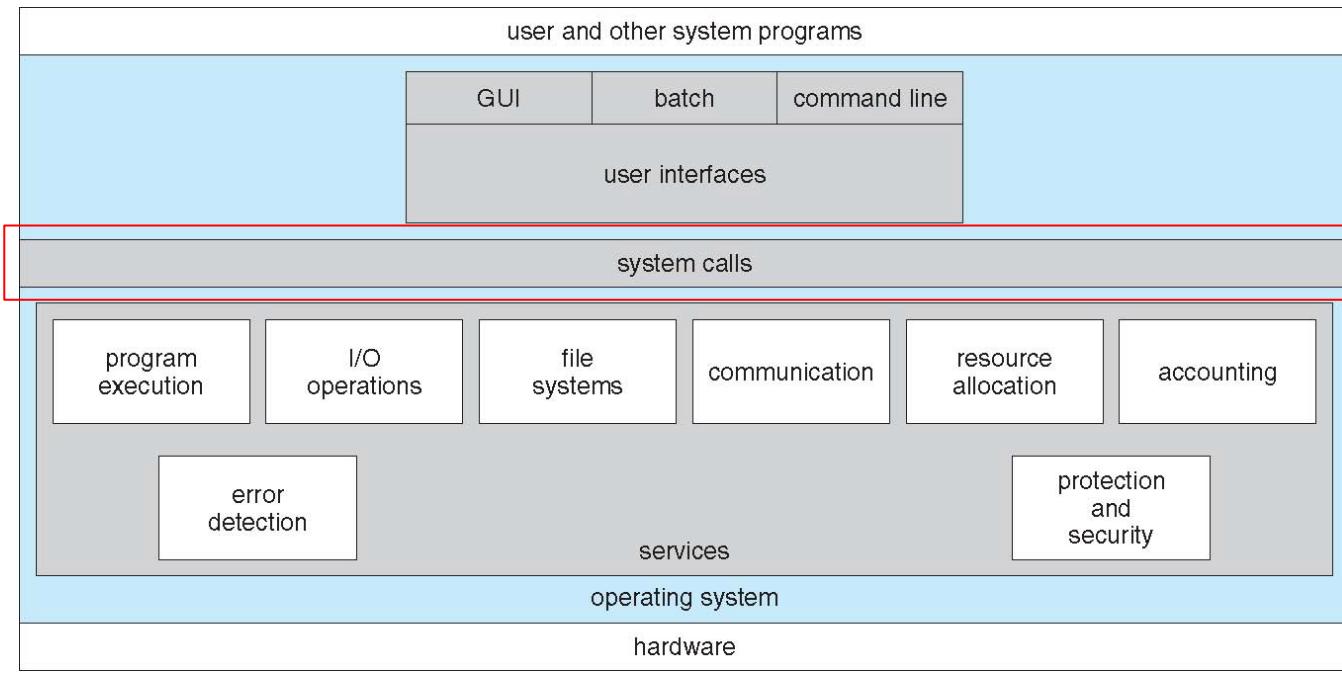
The Mac OS X GUI





시스템 콜 (System Calls)

- ❖ 시스템 콜(호출)은 운영체제가 제공하는 서비스에 대한 인터페이스를 사용자에게 제공한다
- ❖ 일반적으로 고급 언어(C 또는 C++)로 작성된 루틴 형태로 제공
- ❖ 대부분 직접적인 시스템 콜 사용보다는 고급의 **Application Program Interface (API)**를 통해 접근한다.
 - | 사용자 프로그램 -> API 사용 -> 시스템콜

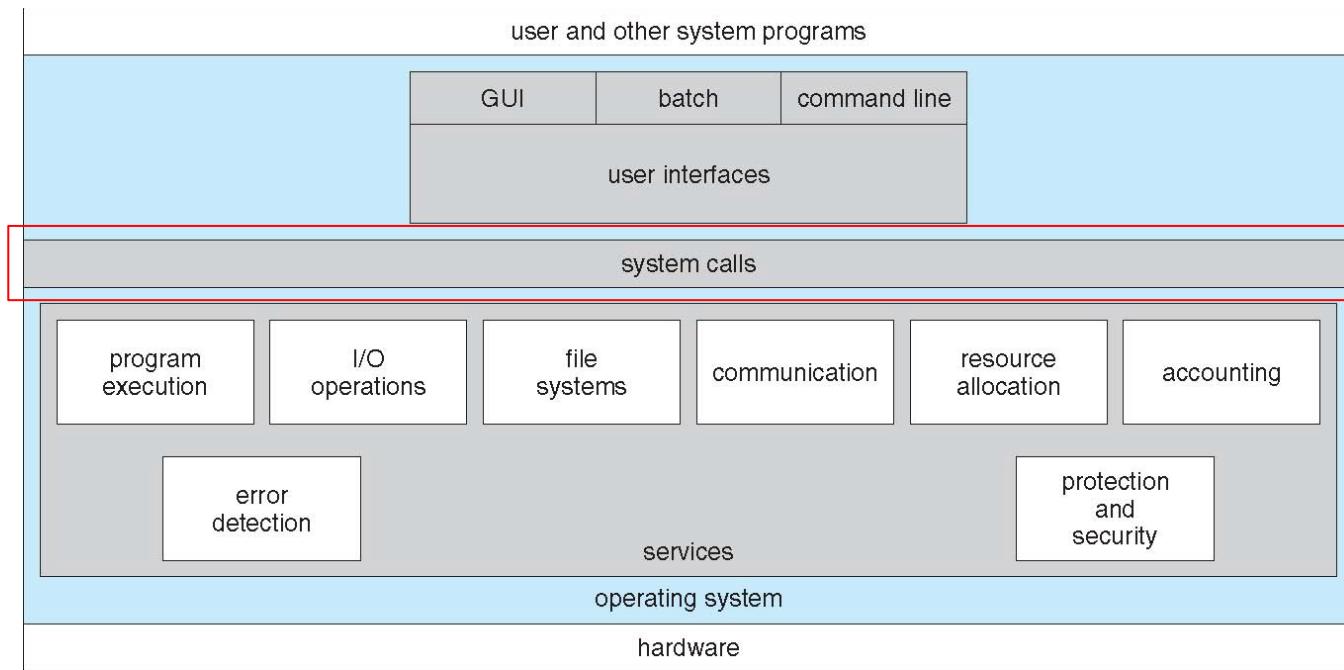




시스템 콜 (System Calls)

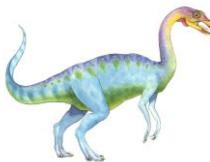
❖ 가장 많이 사용되는 APIs

- | 원도우의 Win32 API
- | POSIX API (모든 버전의 UNIX, Linux, and Mac OS X)
- | Java Virtual Machine(JVM)의 Java API



POSIX (portable operating system interface) : 이식 가능한 운영 체제 인터페이스. 서로 다른 [UNIX](#) OS의 공통 [API](#)를 정리 → [이식성](#)이 높은 [유닉스 응용 프로그램](#)을 개발하기 위한 목적

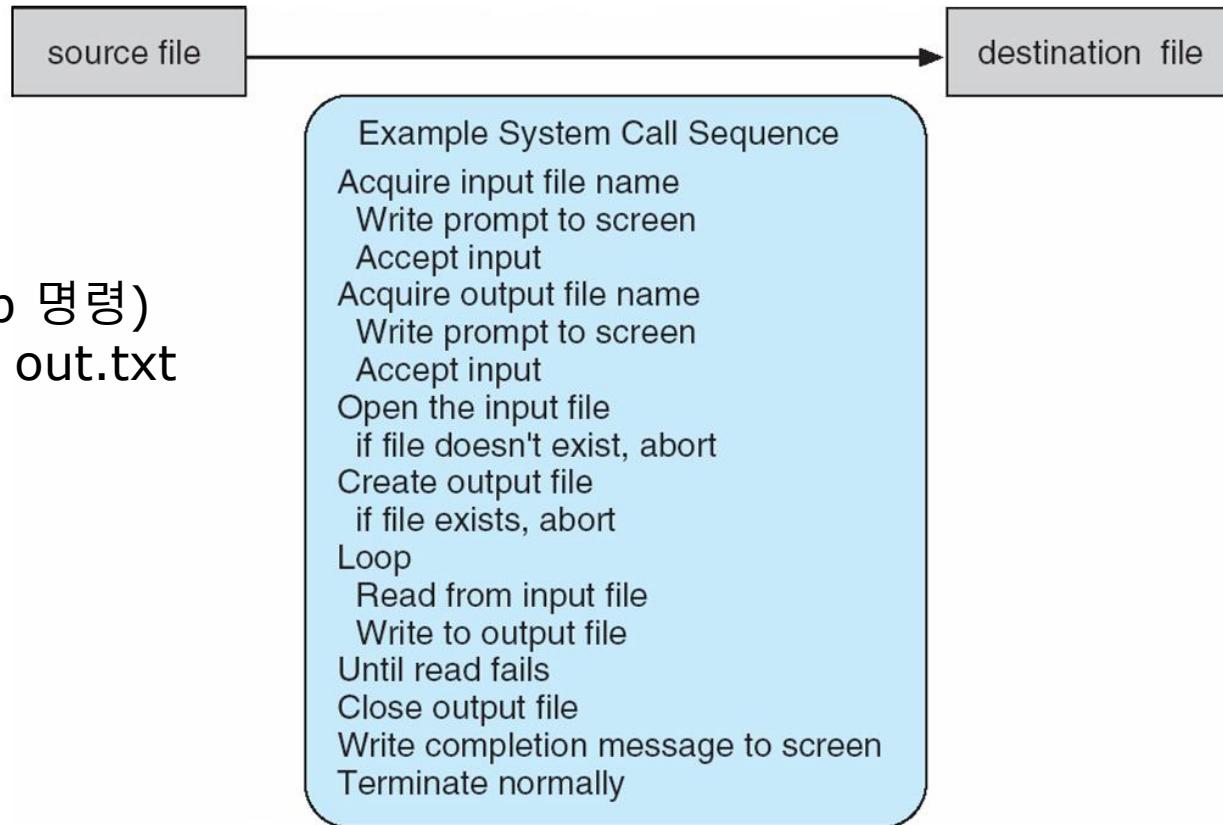




Example of System Calls

- ❖ 파일의 내용을 다른 파일에 복사하는 시스템 콜의 순서(p70)
- ❖ System call sequence to copy the contents of one file to another file

(UNIX cp 명령)
cp in.txt out.txt





Example of Standard API (p71)

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 function parameters
value name

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

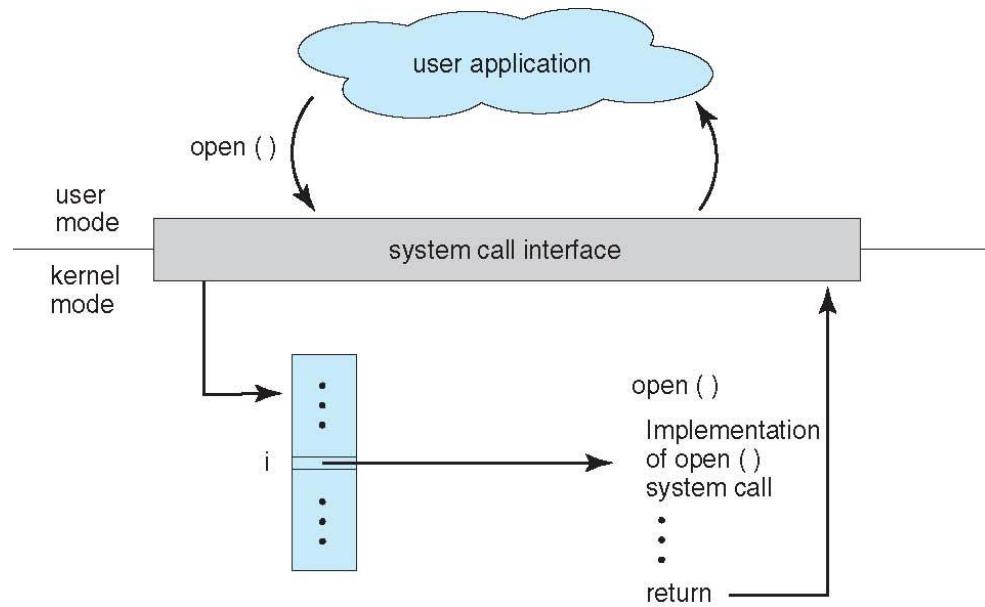
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns -1.





시스템 콜 구현 (System Call Implementation) 1

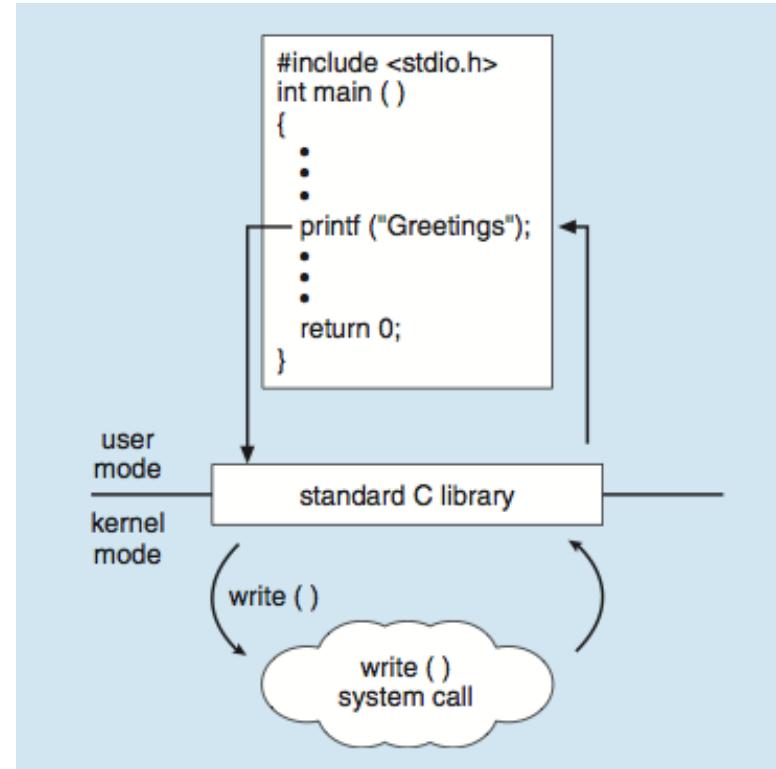
- ❖ 통상적으로, 각 시스템 콜에 대하여 숫자가 부여된다
 - | 시스템 콜 인터페이스가 이 숫자로 된 인덱스를 가진 테이블을 관리한다.
- ❖ 시스템 콜 인터페이스가
 - | 운영체제 커널 내에 있는 요청된 시스템 콜을 하고
 - | 시스템 콜의 반환 상태와 해당 반환 값을 반환한다.





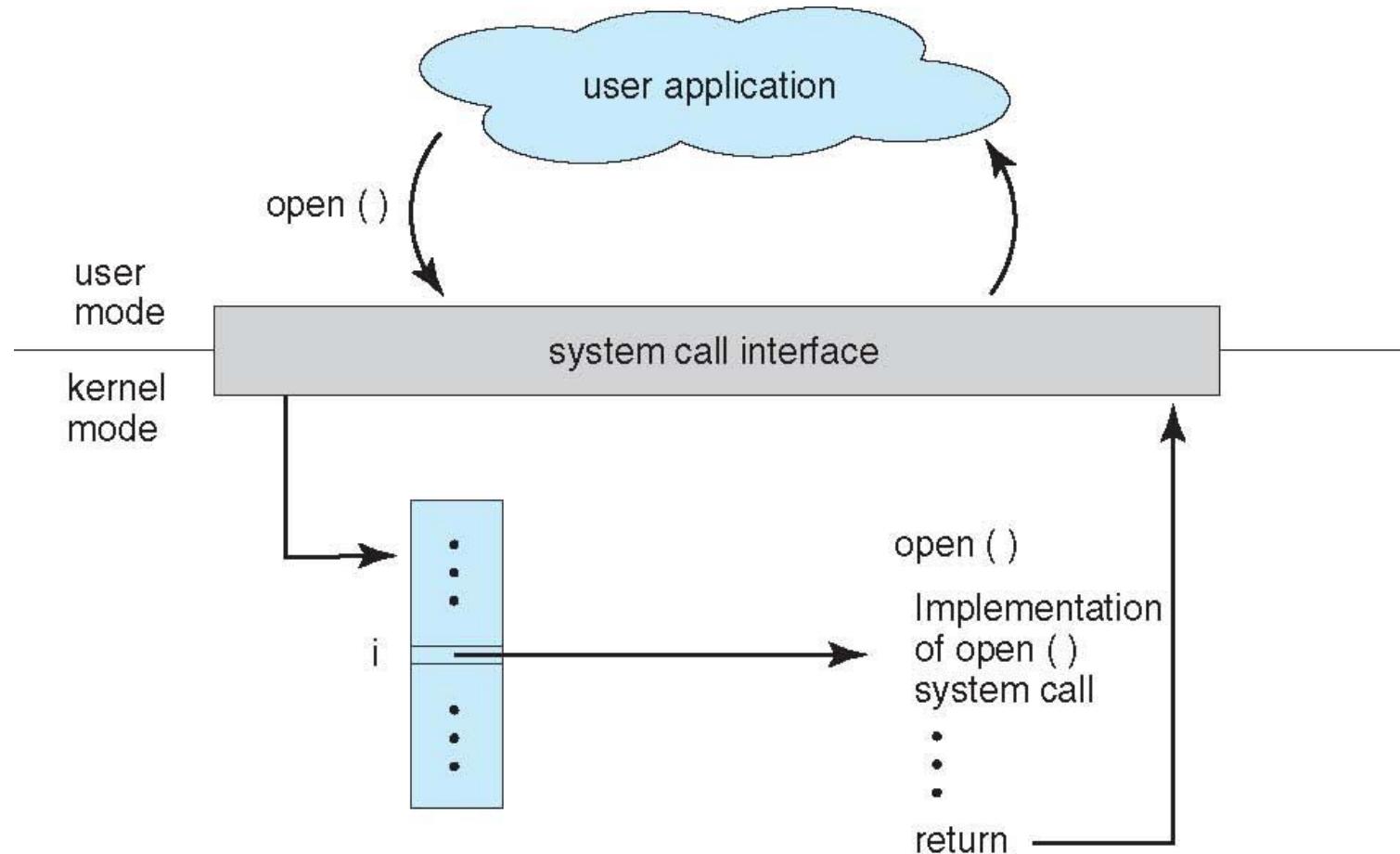
시스템 콜 구현 (System Call Implementation) 2

- ❖ 개발자는 시스템 콜이 어떻게 구현되는지 알 필요 없다.
 - | 개발자는 API를 제대로 사용하고, 운영체제가 시스템 콜의 결과로 무슨 일을 하는지 알면 된다.
 - | 운영체제 인터페이스의 자세한 내용은 API에 의해 숨겨진다
 - ▶ 실행시간 지원 시스템 (컴파일러에 포함된 라이브러리에 내장된 함수의 집합)이 운영체제의 시스템 콜을 연결해 주는 시스템 콜 인터페이스를 제공한다.





API – System Call – OS 관계





시스템 콜 파라미터 전달

- ❖ 원하는 시스템 콜 이름 외에 많은 정보가 일반적으로 필요하다
 - | 필요한 정보의 종류와 양은 운영체제와 시스템 콜에 따라 다르다.
(* 시스템콜은 일반적인 용어다. 시스템 별로 시스템 콜에 해당하는 용어가 있다.)
- ❖ 운영체제에 매개변수를 전달하는 세가지 일반적인 방법
 1. 가장 간단한 방법: 매개변수를 레지스터에 전달
 - ▶ 문제는 레지스터 숫자보다 더 많은 매개변수가 있는 경우는 레지스터만으로는 부족하다.

Calling program

```
Move N, R1  
Move M, R2  
Call NumAdd  
Move R0, Sum
```

.

.

.

Subroutine

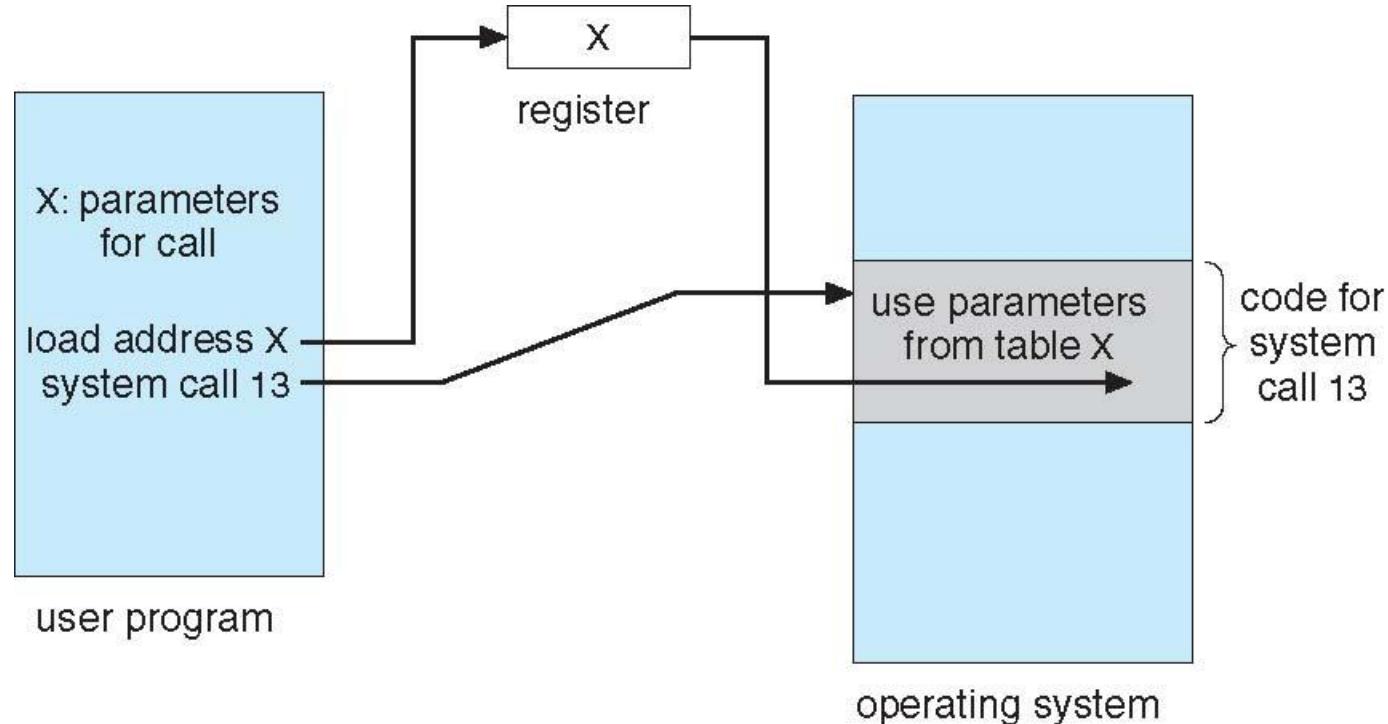
```
NumAdd    Clear R0  
          Add (R1, R2) R0  
          Return
```





시스템 콜 파라미터 전달 : 테이블

- ❖ 운영체제에 매개변수를 전달하는 세가지 일반적인 방법 ...
 2. 매개변수가 메모리에 있는 블록, 테이블에 저장된다.
 - ▶ 블록의 메모리 주소가 레지스터에 매개변수로 전달된다.
 - ▶ Linux와 Solaris에서 사용하는 방법





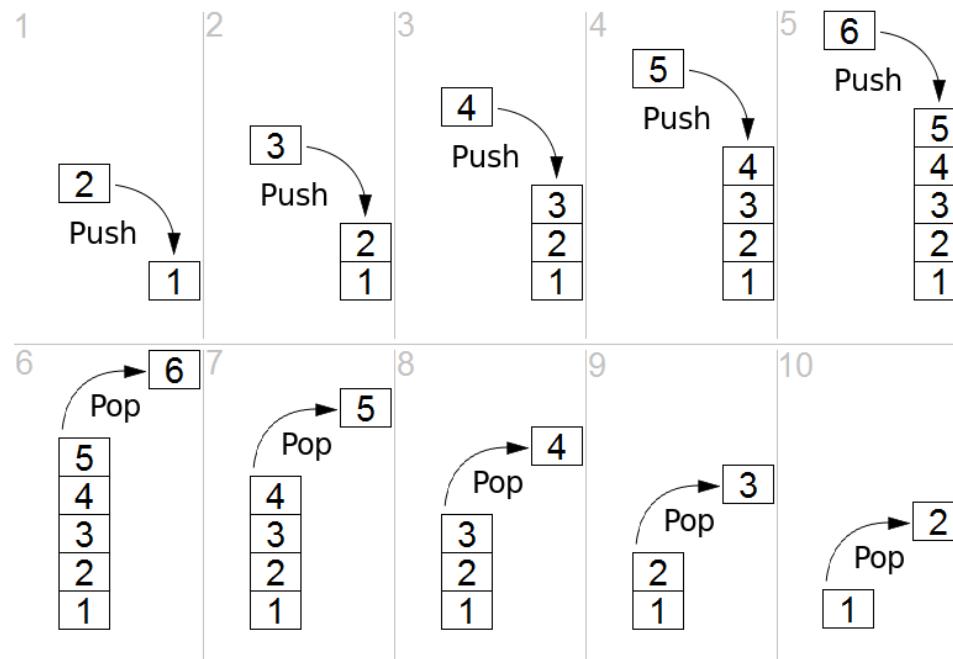
시스템 콜 파라미터 전달 : 스택

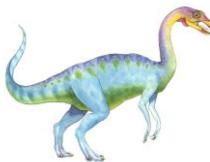
- 운영체제에 매개변수를 전달하는 세 가지 일반적인 방법 ...

3. 프로그램이 매개변수를 스택에 넣는다

- 운영체제가 스택으로부터 매개변수를 꺼낸다

| 블록과 스택 방법은 매개변수 개수나 길이에 제약을 가하지 않는다





시스템 콜 유형 1

❖ 시스템 콜 : 5종류

❖ 프로세서 제어

- | 프로세스 생성(create process), 프로세스 종료(terminate process)
- | 정상적 종료(end), 비정상적 종료(abort)
- | 적재(load), 실행(execute)
- | 프로세스 속성(attributes) 획득(get), 프로세스 속성 설정(set)
- | 시간 대기
- | 사건 대기(wait event), 사건 알림(signal event)
- | 메모리 할당 및 제거(allocate and free memory)
- | 장애 시 메모리 덤프
- | **버그** 찾는 **debugger**, 한 단계씩 실행
- | 프로세스 간에 공유 자료 접근을 관리하는 **locks**





시스템 콜 유형 2

❖ 파일 관리 (File management)

- | 파일 생성 (create file), 파일 삭제 (delete file)
- | 파일 열기 (open), 파일 닫기 (close file)
- | 읽기, 쓰기, 위치 변경 (read, write, reposition)
- | 파일 속성 획득 및 설정 (get and set file attributes)

❖ 장치 관리 (Device management)

- | 장치 요청, 장치 방출 (request device, release device)
- | 읽기, 쓰기, 위치 변경 (read, write, reposition)
- | 장치 속성 획득 및 설정 (get device attributes, set device attributes)
- | 논리적 장치 부착 및 분리 (logically attach or detach devices)





시스템 콜 유형 3

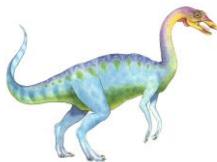
❖ 정보 관리(Information maintenance)

- | 시간과 날짜의 설정과 획득(get time or date, set time or date)
- | 시스템 데이터의 설정과 획득(get system data, set system data)
- | 프로세스, 파일, 장치 속성의 획득과 설정(get and set process, file, or device attributes)

❖ 통신(Communications)

- | 통신 연결 생성, 제거
- | 메시지의 송신, 수신
- | 공유메모리 모델 생성, 메모리 영역 접근
- | 상태 정보 전달
- | 원격 장치의 부착(attach) 및 분리(detach)





시스템 콜 유형 4

❖ 보호

- | 자원 접근을 제어(Control access to resources)
 - ▶ 다중 프로그래밍 시스템
 - ▶ 네트워킹과 인터넷
- | 접근 허용 반환 및 설정(Get and set permissions)
- | 사용자 접근 허용 및 거절(Allow and deny user access)





윈도우와 UNIX 시스템 콜

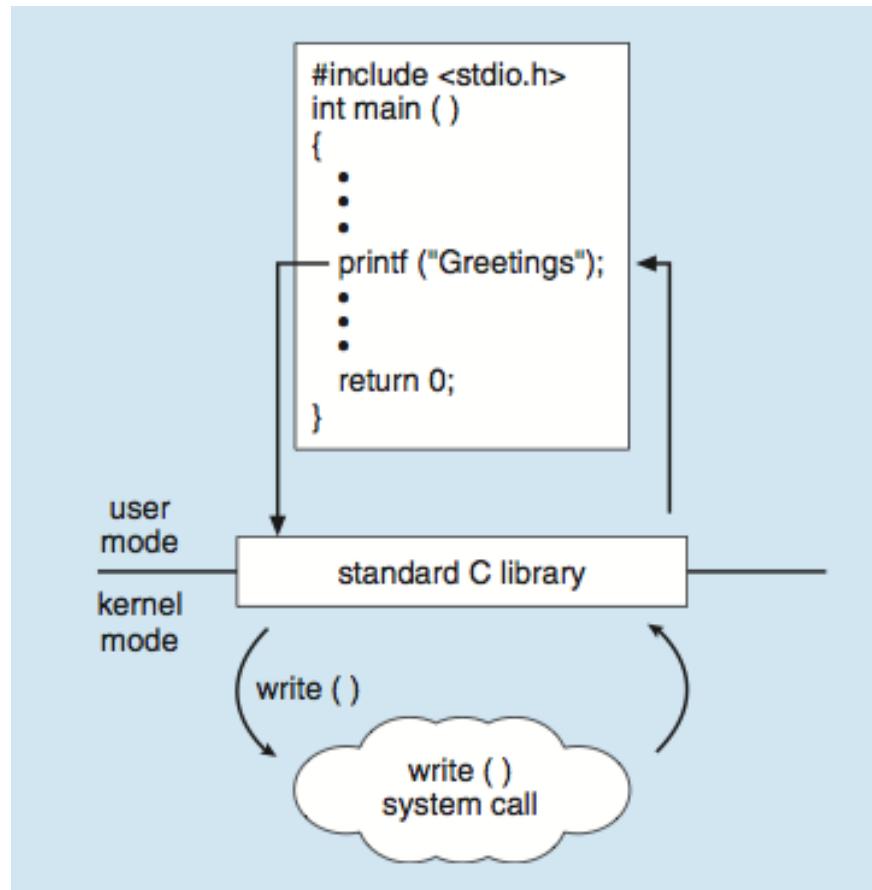
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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





Standard C Library Example

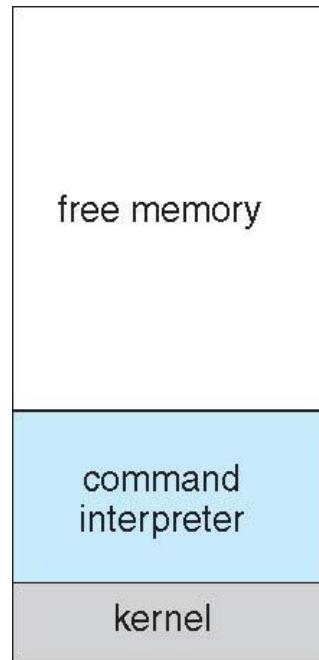
- ❖ C 프로그램이 printf() 호출 → 표준 “C 라이브러리”가 write() 시스템 콜 수행
- ❖ C program invoking printf() library call, which calls write() system call





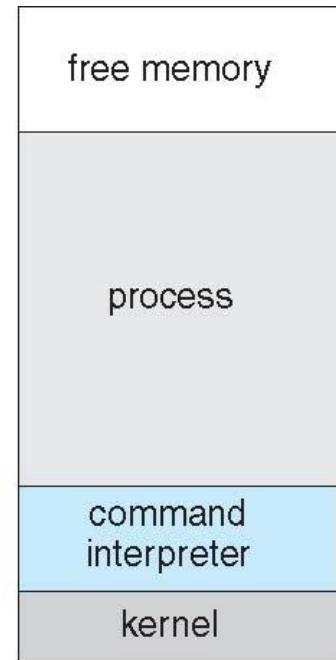
Example: MS-DOS

- ❖ 단일 태스킹(monoprogramming)
- ❖ 시스템이 시동될 때 하나의 명령해석기 호출
- ❖ 하나의 프로그램을 실행하는 간단한 방법 제공
 - | 프로세스 생성하지 않는다
- ❖ 단일 메모리 공간 (Single memory space)
- ❖ 커널용 메모리를 제외한 메모리 공간을 덮어쓰며 프로그램을 적재
- ❖ 프로그램 종료 -> 쉘을 재적재



(a)

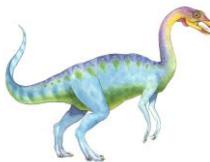
At system startup



(b)

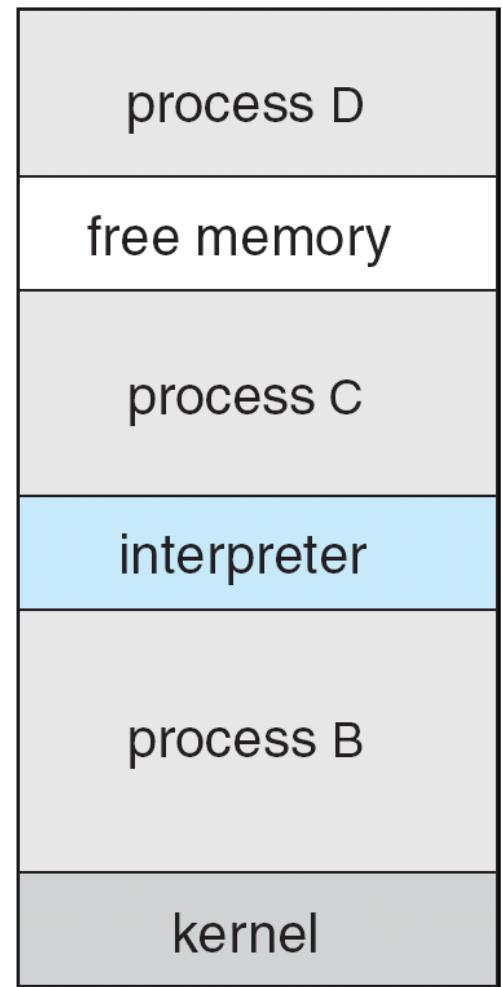
running a program

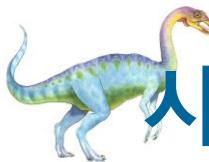




Example: FreeBSD

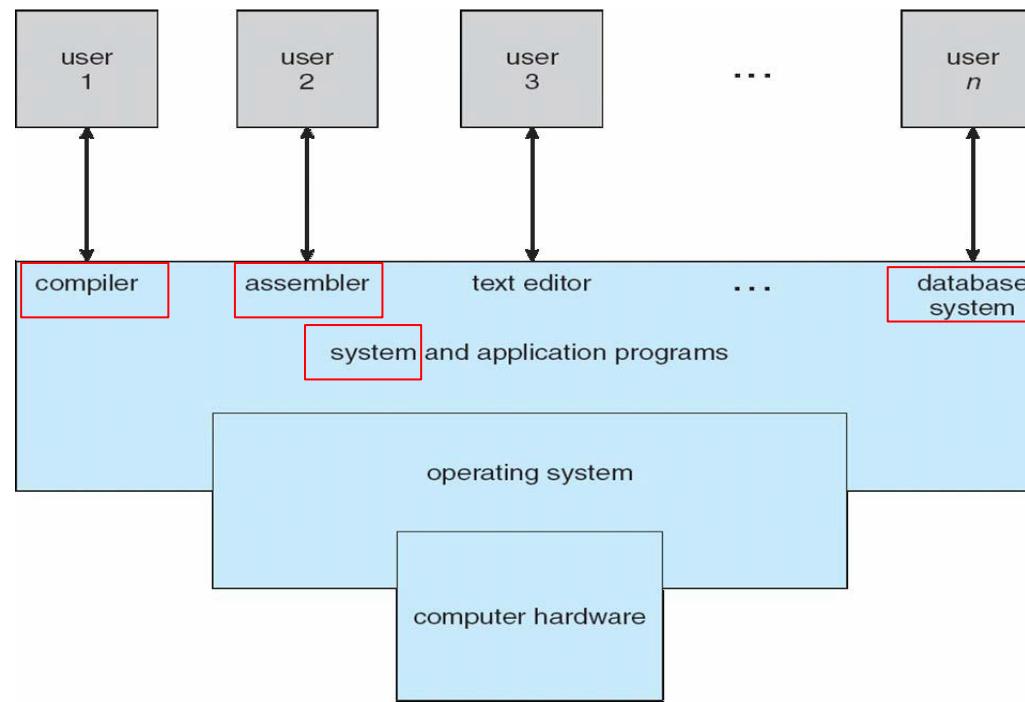
- ❖ UNIX 의 한 종류
- ❖ 다중 태스킹(multiprogramming)
- ❖ 사용자가 로그인할 때, 사용자가 선택한 쉘 (명령 해석기)이 실행
- ❖ 쉘이 fork() 시스템 콜을 실행하여 프로세스 생성
 - | Exec()을 실행하여 프로그램을 프로세스에 적재함
 - | 쉘은 프로세스 종료를 기다리거나 또는 다른 사용자 명령을 수행한다
- ❖ 프로세스는 exit() 시스템 콜을 실행하여 종료
 - | 상태 코드 0 : 정상
 - | 상태 코드 0 아닌 값 ← 오류 코드

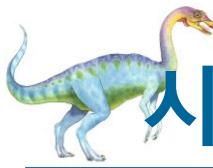




시스템 프로그램 (System Programs) 1

- ❖ 시스템 프로그램(시스템 유틸리티라고도 함)은 프로그램 개발과 실행에 편리한 환경을 제공한다.
 - | 일부는 시스템 콜에 대한 사용자 인터페이스, 다른 것들은 이보다 훨씬 더 복잡하다.
- ❖ 운영체제에 대한 대부분 사용자의 인식은 시스템 콜 보다는 시스템 프로그램에 의해 결정된다.

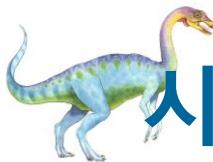




시스템 프로그램 (System Programs) 2

- ❖ 종류는 아래와 같다
 - | 파일 관리
 - | 상태 정보
 - | 프로그래밍 언어 지원
 - ▶ (C, C++, Java 등)
 - | 프로그램 적재와 실행
 - | 통신
 - | 백그라운드 서비스 (Background services)





시스템 프로그램 (System Programs) 3

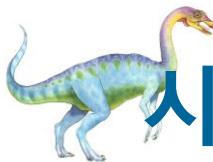
❖ 파일 관리 (File Management)

- | 일반적으로 파일과 디렉토리에 대하여 생성, 삭제, 복사, 개명 (rename), 프린트, 덤프, 리스트 등의 작업을 한다.

❖ 상태 정보 (Status Information)

- | 어떤 프로그램들은 시스템에 날짜, 시간, 가용한 메모리 용량, 디스크 용량, 사용자의 수 등을 요청한다.
- | 다른 프로그램들은 자세한 성능, 로깅과 디버깅 정보를 제공한다.
 - ▶ 통상 이 프로그램들은 정보를 단말기나 다른 출력 장치 등에 서식을 갖추어 인쇄하거나 또는 GUI 윈도우에 표시한다.
- | 어떤 시스템은 레지스트리(**registry**)를 만들어 환경 설정 정보를 저장하고 검색한다.





시스템 프로그램 (System Programs) 4

❖ 파일 변경

- | 파일을 생성하고 수정하는 텍스트 에디터 제공
- | 파일 내용을 검색하거나 변환하기 위한 특수 명령어 제공

❖ 프로그래밍 언어 지원

- | 컴파일러, 어셈블러, 디버거와 인터프리터 등이 제공된다

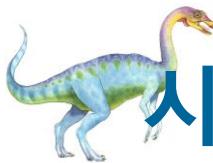
❖ 프로그램 적재와 실행 (p84~85)

- | 절대 로더(Absolute loaders), 재배치 가능 로더(relocatable loaders), 링키지 에디터(linkage editors), 중첩 로더(overlay-loaders), 고급언어와 기계어를 위한 디버깅 시스템 등이 제공된다

❖ 통신

- | 프로세스, 사용자, 컴퓨터 시스템 간에 가상 접속을 이루기 위한 기법을 제공한다
- | 다른 사용자 화면에 메시지를 전송하거나, 웹 페이지를 읽거나, 이메일 메시지를 보내거나, 원격 로그인하거나, 한 기계에서 다른 기계로 파일 전송을 할 수 있게 한다.



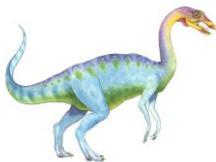


시스템 프로그램 (System Programs) 5

❖ 백그라운드 서비스 (Background Service)

- | 백그라운드 서비스 (Background Service)
 - | 부팅 시 살아남
 - ▶ 일부는 시스템 시작 시 살아나고 끝난다
 - ▶ 일부는 시스템 시작해서 종료 시까지 살아있다
 - | 디스크 검사, 프로세스 스케줄링, 예외 기록, 프린트 기능을 제공한다.
 - | 사용자 모드에서 수행
 - | 서비스, 서브시스템, 데몬(daemon) 등으로 알려져 있음





운영체제 설계 및 구현

- ❖ 운영체제의 설계와 구현에 완전한 해결책은 없지만, 성공적인 접근 방법들이 존재한다
- ❖ 운영체제 별로 내부 구조는 매우 다르다
- ❖ 첫 번째로 목표와 명세를 정의하는 것이다
- ❖ 하드웨어와 시스템 타입(일괄처리, 시분할, 단일사용자, 다중 사용자, 분산, 실시간 등)의 선택에 따라 다르다
- ❖ **사용자 목표와 시스템 목표**
 - | 사용자 목표 – 운영체제는 사용이 편리, 배우기 쉽고, 안정되고, 안전하며 신속해야 한다
 - | 시스템 목표 – 운영체제는 설계, 구현, 유지가 쉬워야 하며, 또한 적응성, 신뢰성, 무오류, 효율성을 가져야 한다

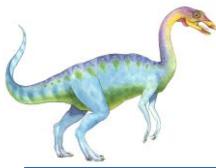




운영체제 설계 및 구현 : 정책과 기법

- ❖ 중요한 원칙 : 정책과 메커니즘의 분리
 - | 정책 (Policies): 무엇을 할 것인가?
 - | 기법 (Mechanism): 어떻게 할 것인가?
- ❖ 정책과 기법을 분리
 - | 나중에 정책이 변경될 때 최대한의 융통성이 생긴다.
- ❖ 정책은 장소가 바뀌거나 시간의 흐름에 따라 변경될 수 있다.
- ❖ 정책에 민감하지 않은 기법이 바람직하며 정책의 변경은 시스템의 일부 매개변수를 재정의할 수 있으면 좋다.
- ❖ 예
 - | CPU를 계속 점유하는 것을 방지하기 위해 Timer 사용 ()
 - | 특정 사용자에게 타이머를 얼마나 오랫동안 설정할 것인지 결정 ()





구현 (Implementation)

- ❖ 다양성
 - | 초기 운영체제는 어셈블리 언어로 구현
 - | 그 후에 Algol과 PL/1 같은 시스템 프로그래밍 언어 사용
 - | 현재는 C, C++ 사용
- ❖ 실제는 언어를 혼합 사용
 - | 하위 수준은 어셈블리어
 - | 중요 부분은 C
 - | 시스템 프로그램은 C, C++, PERL, Python과 같은 스크립트 언어
- ❖ 고급 언어(High-level Language)로 된 운영체제
 - | 코드 작성이 빠르며, 간결하고, 이해하기 쉽고 디버그도 쉽다.
 - | 다른 하드웨어로 이식이 쉽다 (easier to port).
 - | 느리다 (slow)
- ❖ **Emulation**으로 운영체제를 다른 하드웨어에서 수행되는 것도 가능하다





운영체제 구조 (structure)

- ❖ 범용 운영체제는 매우 큰 프로그램이다
- ❖ 다양한 방법으로 운영체제의 구성을 할 수 있다

- ❖ General-purpose OS is very large program
- ❖ Various ways to structure ones
 - | Simple structure – MS-DOS
 - | More complex -- UNIX
 - | Layered – an abstraction
 - | Microkernel -Mach

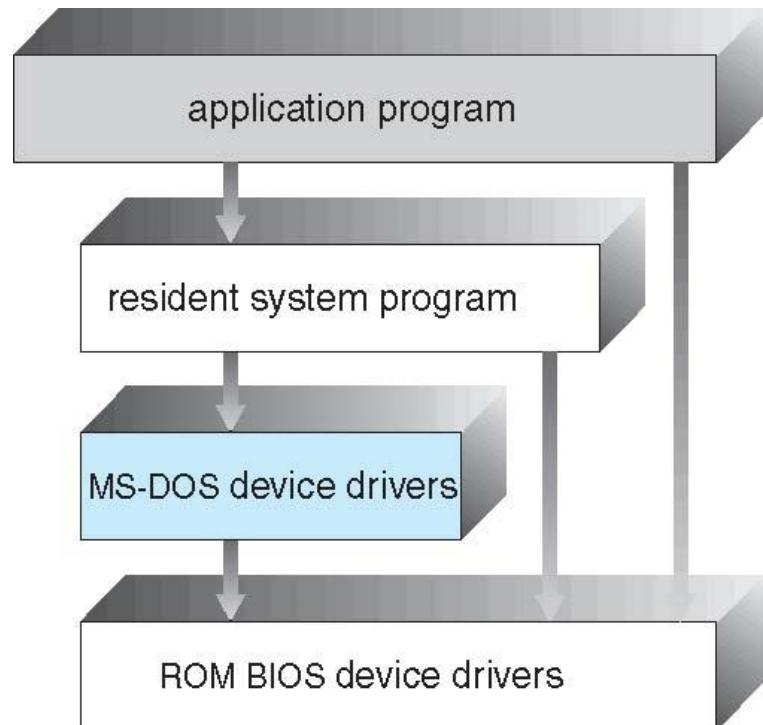




간단한 구조 -- MS-DOS

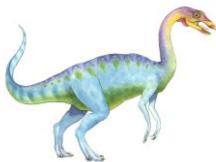
- ❖ MS-DOS – 최소의 공간에 최대의 기능을 제공하도록 작성
 - | 모듈로 제대로 분할되지 않음
 - | MS-DOS가 구조를 갖추고 있지만, 인터페이스와 기능 계층이 잘 분리되지 않음

✓ Monolithic Structure



- BIOS (Basic Input/Output System)
 - Initialization, runtime services for OS and Program
- Unified Extensible Firmware Interface (UEFI) is a successor to BIOS





좀더 복잡한 구조 - UNIX

❖ 최초의 UNIX

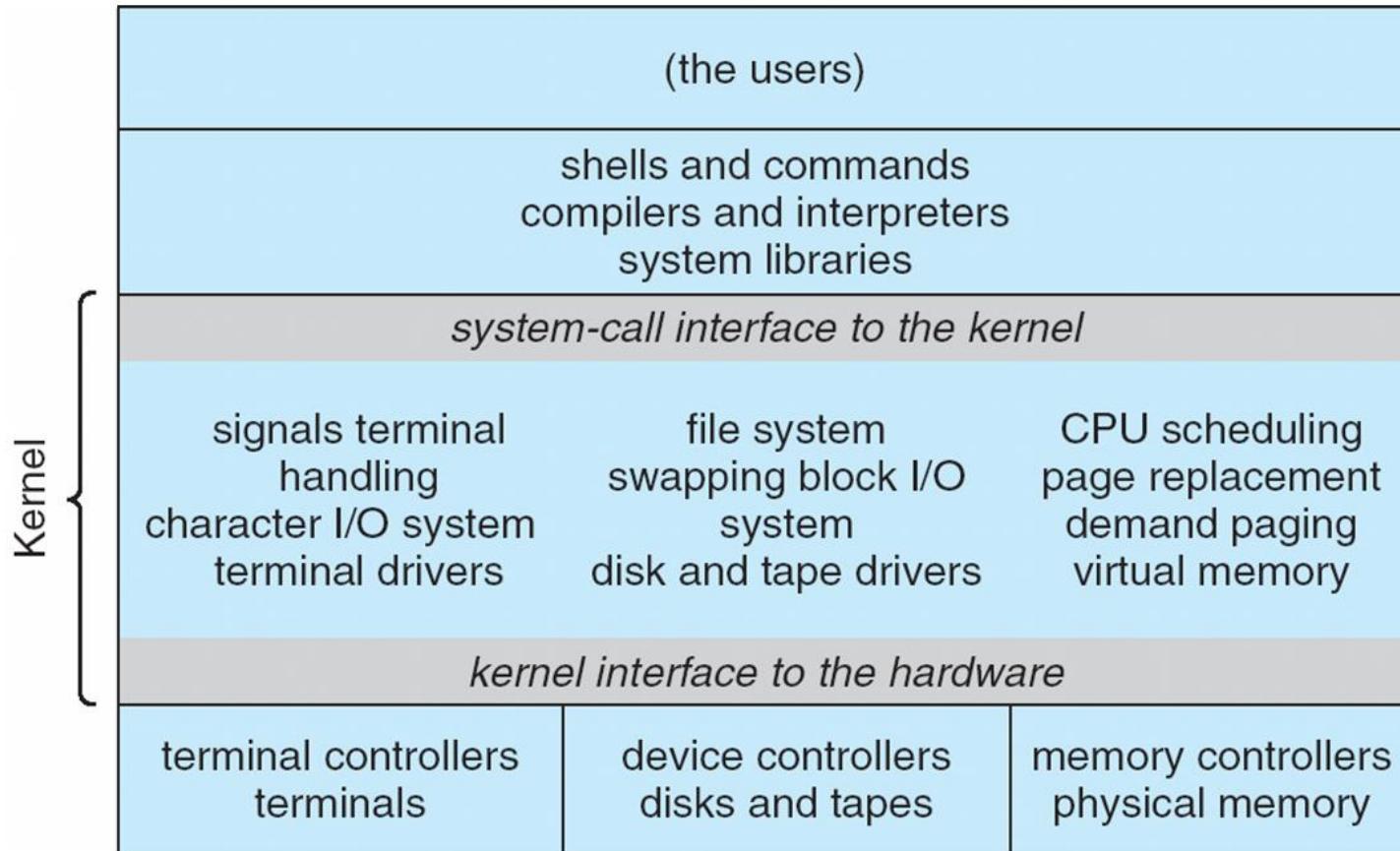
- | 하드웨어 기능의 제약
- | 최초 UNIX는 제한적인 구조를 가졌다.
- | UNIX 운영체제는 두 가지 부분으로 분리
 - › 시스템 프로그램
 - › 커널(kernel)
 - 시스템 콜 인터페이스 아래와 물리적 하드웨어 위에 있는 모든 것
 - 파일 시스템, CPU 스케줄링, 메모리 관리와 다른 운영체제 기능들을 제공 : 한 계층에서 많은 기능을 제공함





전통적인 UNIX 시스템 구조 (p91)

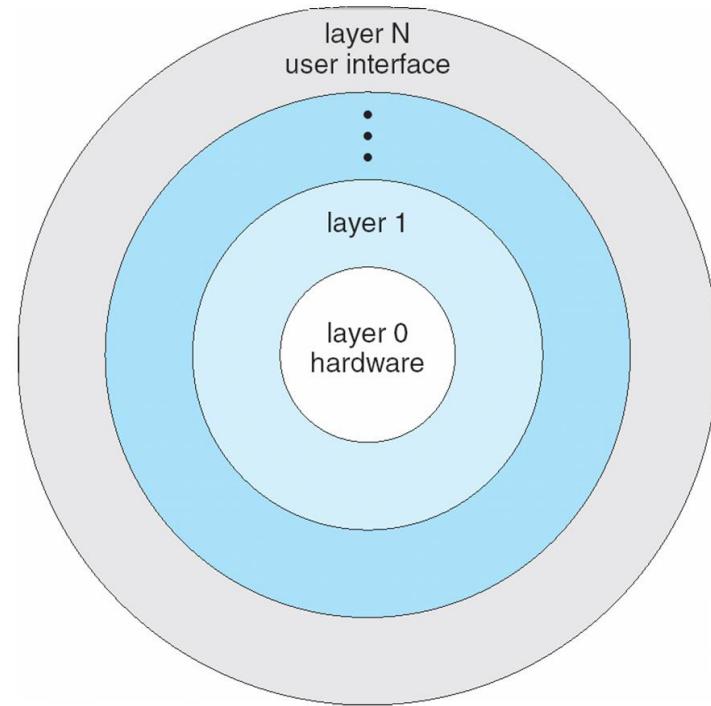
단순하지도 않지만 계층 구조도 없는 형태

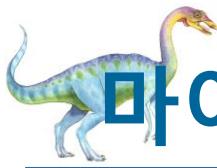




계층적 접근 (Layered Approach)

- ❖ 운영체제는 여러 개의 계층(layer, level)으로 구분
 - | 각 계층은 하위 계층 위에 만들어짐
 - | 최하위(계층 0) 계층은 하드웨어
 - | 최상위 계층(계층 N)은 사용자 인터페이스
- ❖ 모듈화 함으로, 각 계층은 하부 계층의 함수(오퍼레이션)와 서비스만을 사용한다
 - | 설계, 구현과 디버깅 간단
- ❖ 효율성이 낮다



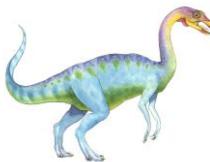


마이크로 커널 시스템 구조

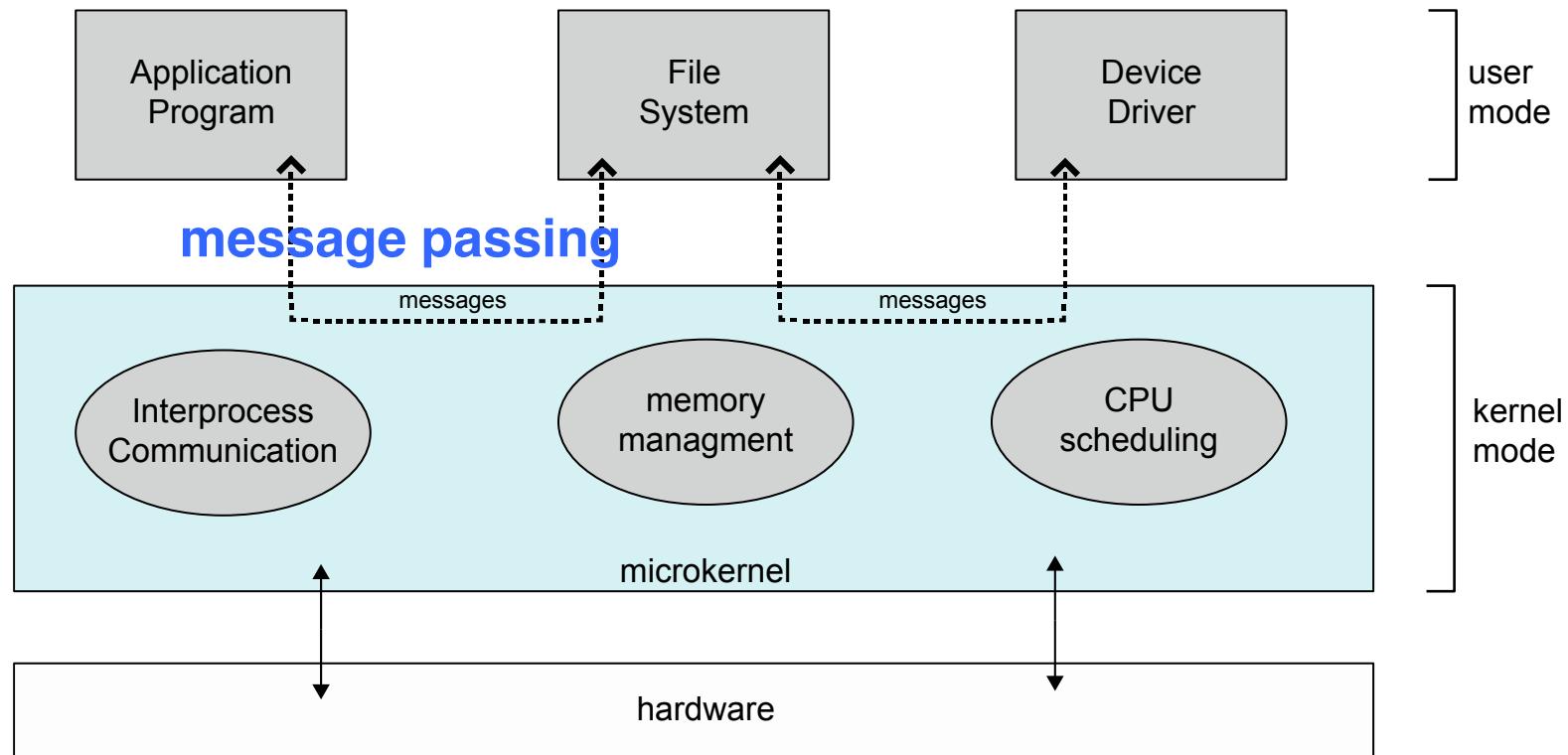
(Microkernel System Structure)

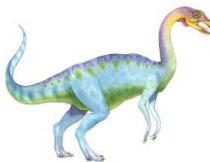
- ❖ 커널이 커져 관리가 어려워져 가능한 많은 기능을 커널에서 사용자 공간으로 옮겨 커널을 최소화
- ❖ **Mach** : 카네기 멜론 대학교에서 개발한 첫 번째 **microkernel**
 - | Mac OS X 커널(**Darwin**)은 부분적으로 Mach에 기초함
- ❖ 사용자 모듈 간의 통신은 **message passing**을 사용한다
- ❖ 장점:
 - | 마이크로커널은 확장하기 좋음
 - | 새로운 하드웨어에 이식이 쉽다
 - | 신뢰성 향상 (커널 모드가 작다)
 - | 보안성 향상
- ❖ 단점
 - | 사용자와 커널 간의 통신으로 인한 성능 오버헤드





マイクロカーネル システム構造

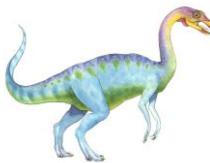




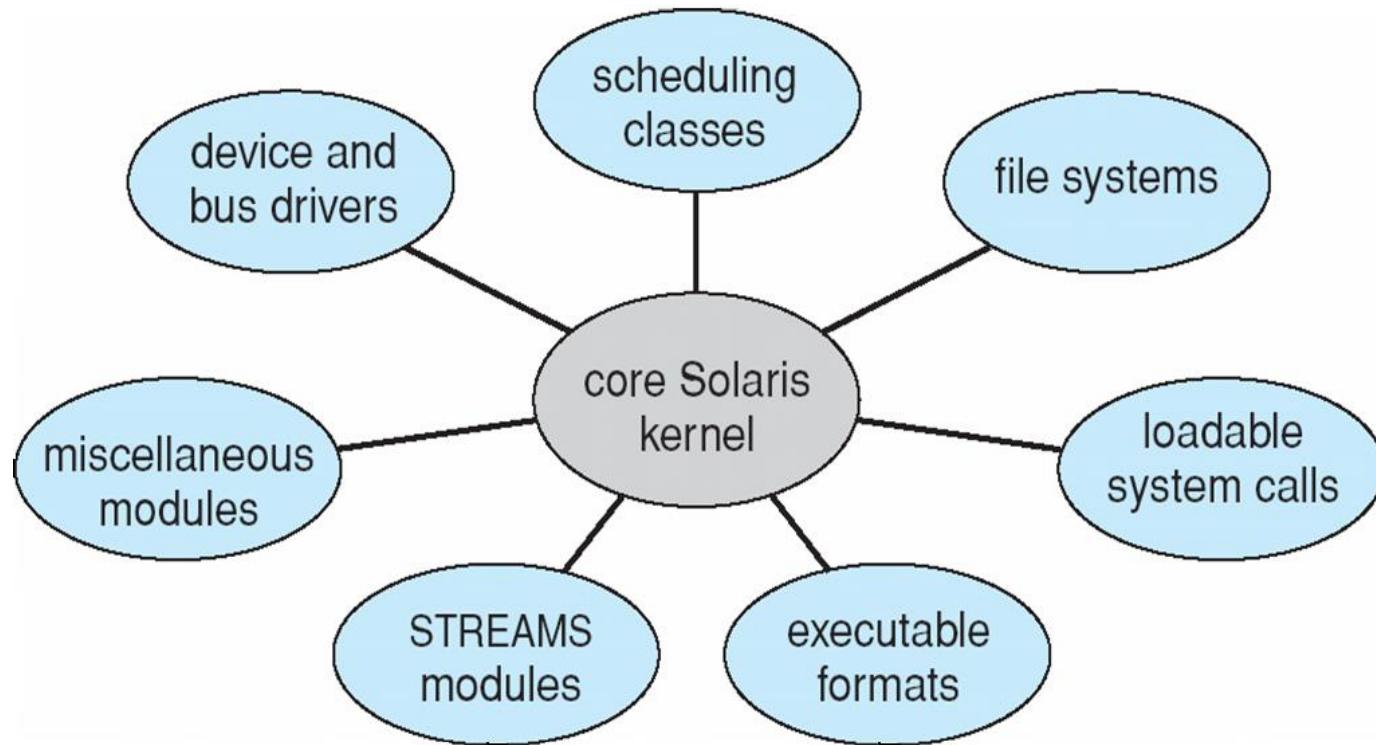
모듈 (Modules)

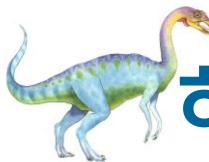
- ❖ 대부분의 현대 운영체제 시스템은 적재 가능한 모듈 (LKM, Loadable Kernel Modules)로 구현한다
 - | 객체 지향 기법 사용
 - | 각 핵심 컴포넌트가 분리되어 있음
 - | 각 모듈이 알려진 인터페이스를 사용하여 다른 모듈과 통신
 - | 각 모듈은 필요한 시점에 커널 내에 적재
- ❖ 대체로, 계층 구조와 비슷하나 융통성이 있음
 - | Linux, Solaris, Mac OS X





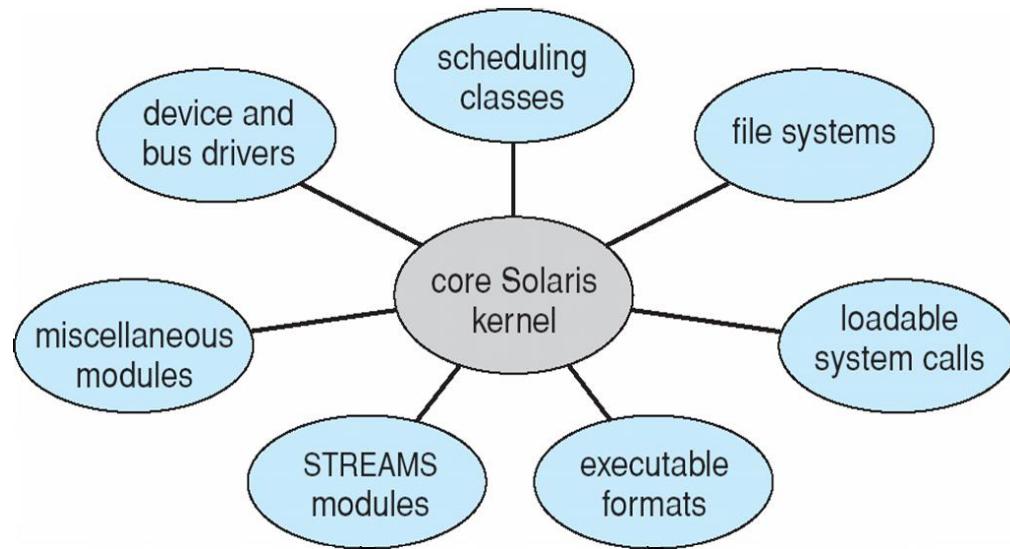
모듈화 방식의 예 : Solaris

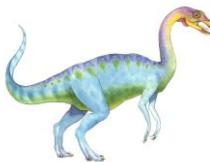




하이브리드 시스템 (Hybrid Systems)

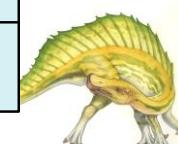
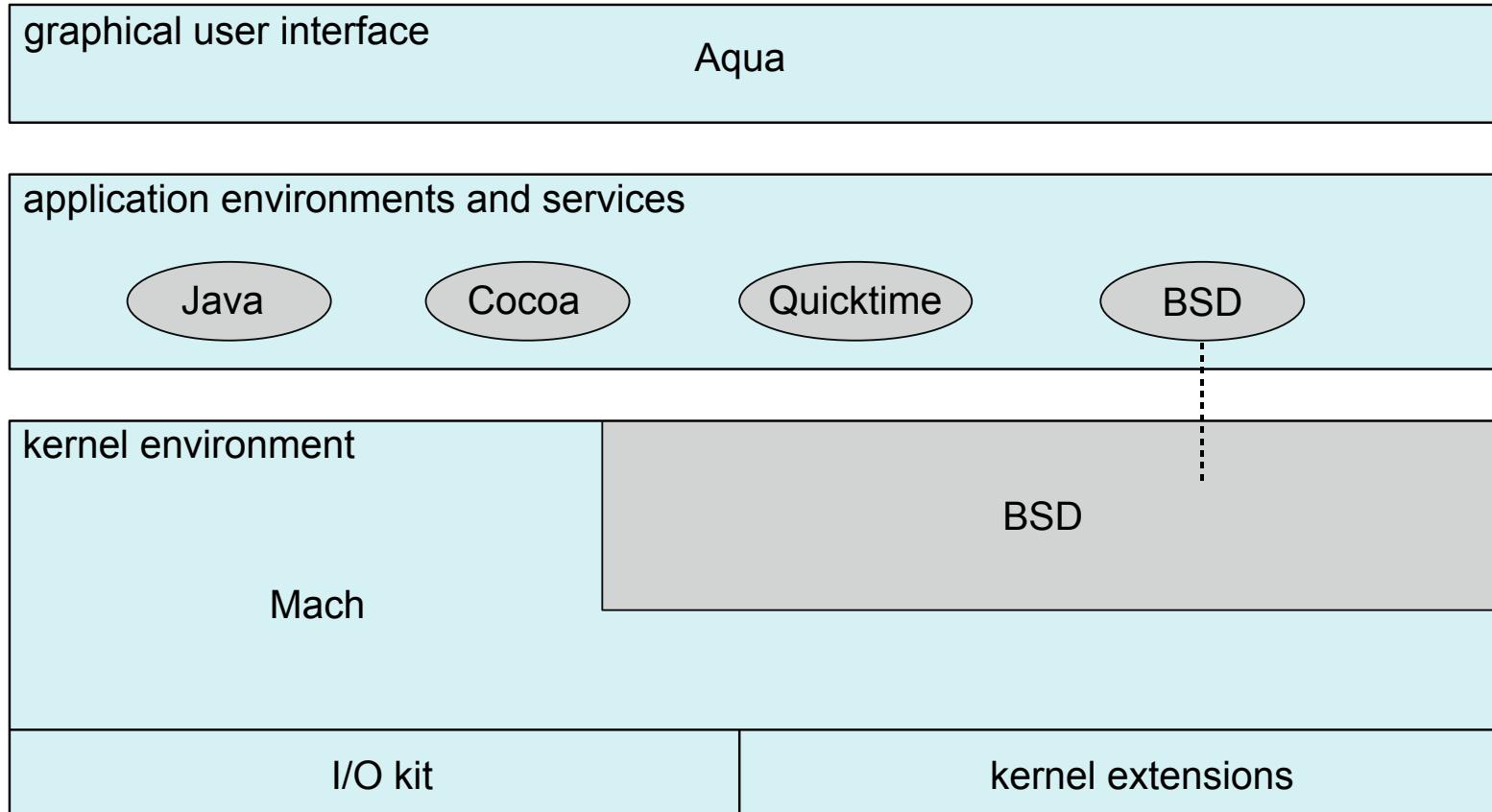
- ❖ 대부분의 현대 운영체제는 순수 한가지 모델만을 사용하지 않는다
 - | 혼용 구조는 여러 기법을 사용하여 성능, 보안, 사용성 요구를 해결한다
- ❖ Linux와 Solaris의 커널은 단일구조(monolithic)이지만, 함수의 동적 적재는 모듈화 기법을 사용한다
- ❖ 윈도우도 대부분이 단일구조이지만, 다른 서브시스템의 특징에 따라 마이크로커널을 채택





Mac OS X Structure

- ❖ 애플 Mac OS X는 혼용으로, 계층구조를 사용하고, **Aqua** 사용자 인터페이스와 **Cocoa** 프로그래밍 환경을 사용한다
 - | 커널 구성 : Mach 마이크로 커널, BSD Unix 일부, 입출력 도구, 동적 적재 모듈인 **커널 확장**





iOS

- ❖ ***iPhone, iPad***를 위한 애플 모바일 운영체제

- | Mac OS X 커널에 기초한 계층 구조
- | OS X 애플리케이션을 네이티브 모드로 지원하지 않는다
 - ▶ 다른 CPU 구조에서도 실행된다 (ARM 대 Intel)
- | 앱 개발을 위한 **Cocoa Touch**는 Objective-C를 위한 API
- | 그래픽, 오디오, 비디오를 위한 **Media services** 계층
- | **Core services**가 클라우드 컴퓨팅과 데이터베이스 제공

Cocoa Touch

Media Services

Core Services

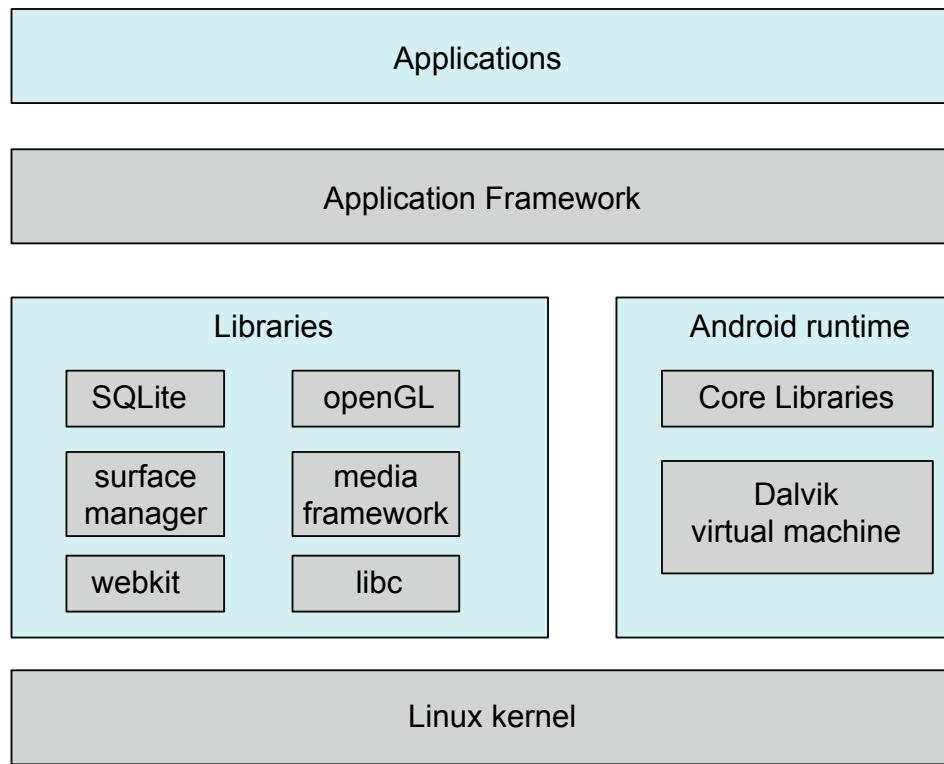
Core OS

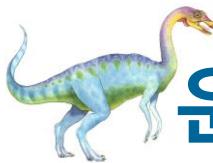




Android Architecture (구조)

- ❖ Open Handset Alliance (Google 주도)에서 개발 : 오픈 소스
- ❖ iOS와 유사한 계층 구조
- ❖ Linux 커널을 기초로 수정
 - | 프로세스, 메모리, 장치 드라이버 관리 제공, 파워 관리 추가
- ❖ 실행환경이 핵심 라이브러리 집합과 Dalvik 가상 머신 포함





운영체제 디버깅 (Operating-System Debugging)

- n Debugging은 오류나 bugs를 발견하고 고치는 행위이다
- n 운영체제는 오류 정보를 포함하는 log files 을 생성한다
- n 애플리케이션이 죽으면 프로세스의 메모리를 캡처한 코어 덤프(core dump)를 생성한다
- n 운영체제가 죽으면 커널 메모리를 캡처한 비정상 종료 덤프(crash dump)를 생성한다
- n 성능 조정(performance tuning)은 시스템 성능을 최적화한다
 - | 분석을 위해 기록된 시스템 동작의 추적 목록(trace listings)을 사용한다
 - | 통계적인 경향을 찾기 위해 프로파일링(Profiling)은 주기적으로 명령어 포인터를 채취한다

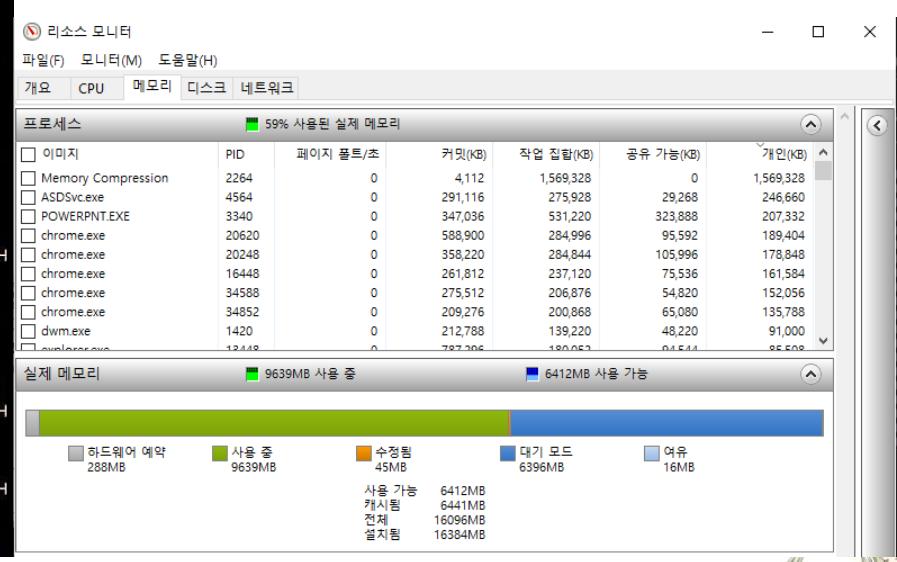
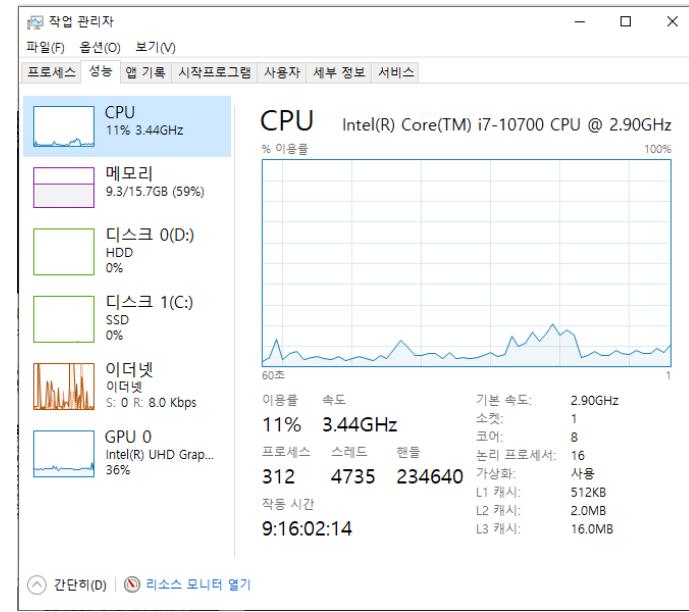
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."





Performance Tuning

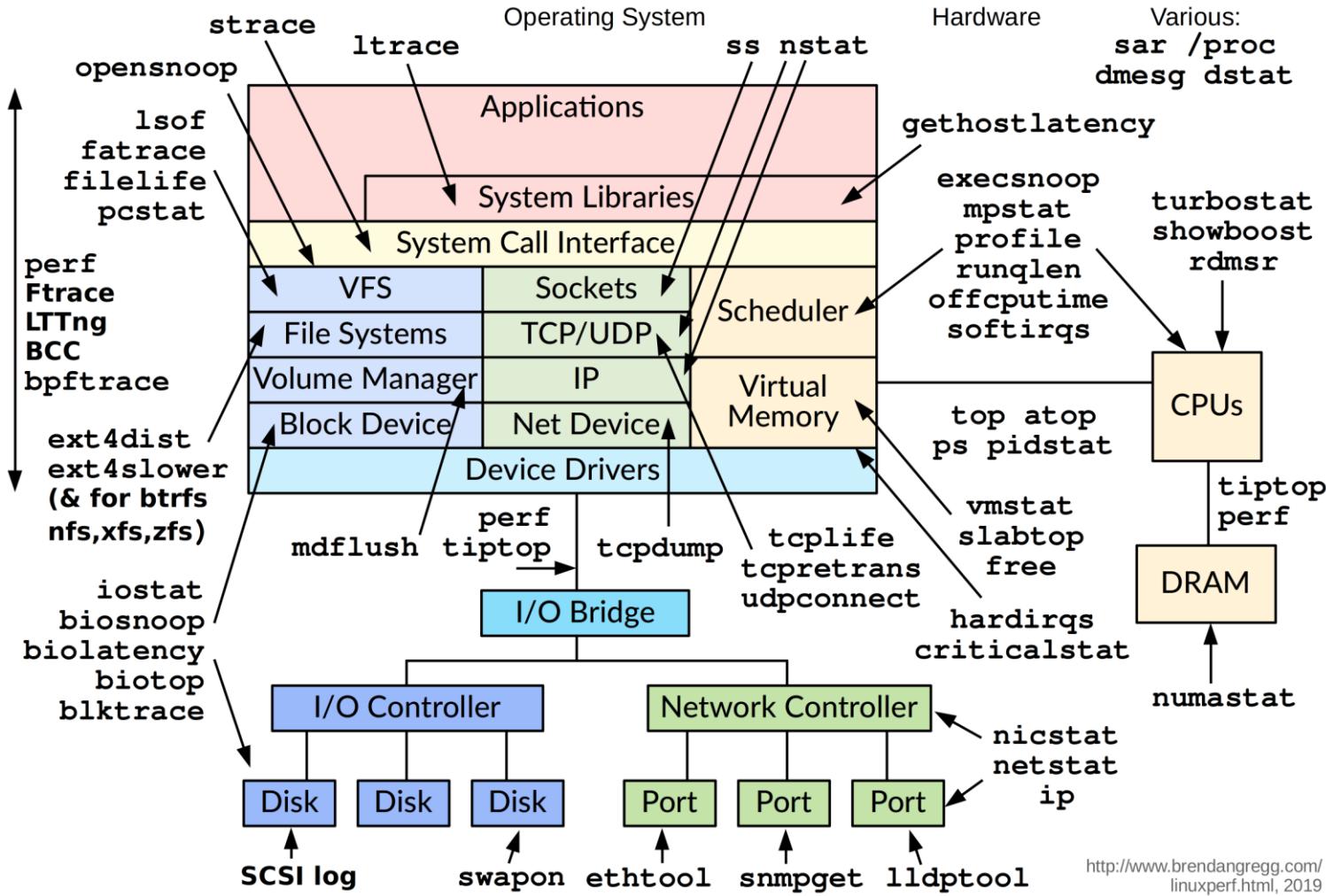
- ❖ 병목 현상을 제거하여 성능을 향상시킨다
- ❖ 운영체제는 시스템의 행동을 계산하고 알려주는 도구를 제공해야 한다
- ❖ 예로, UNIX의 “top” 프로그램 또는 Windows Task Manager





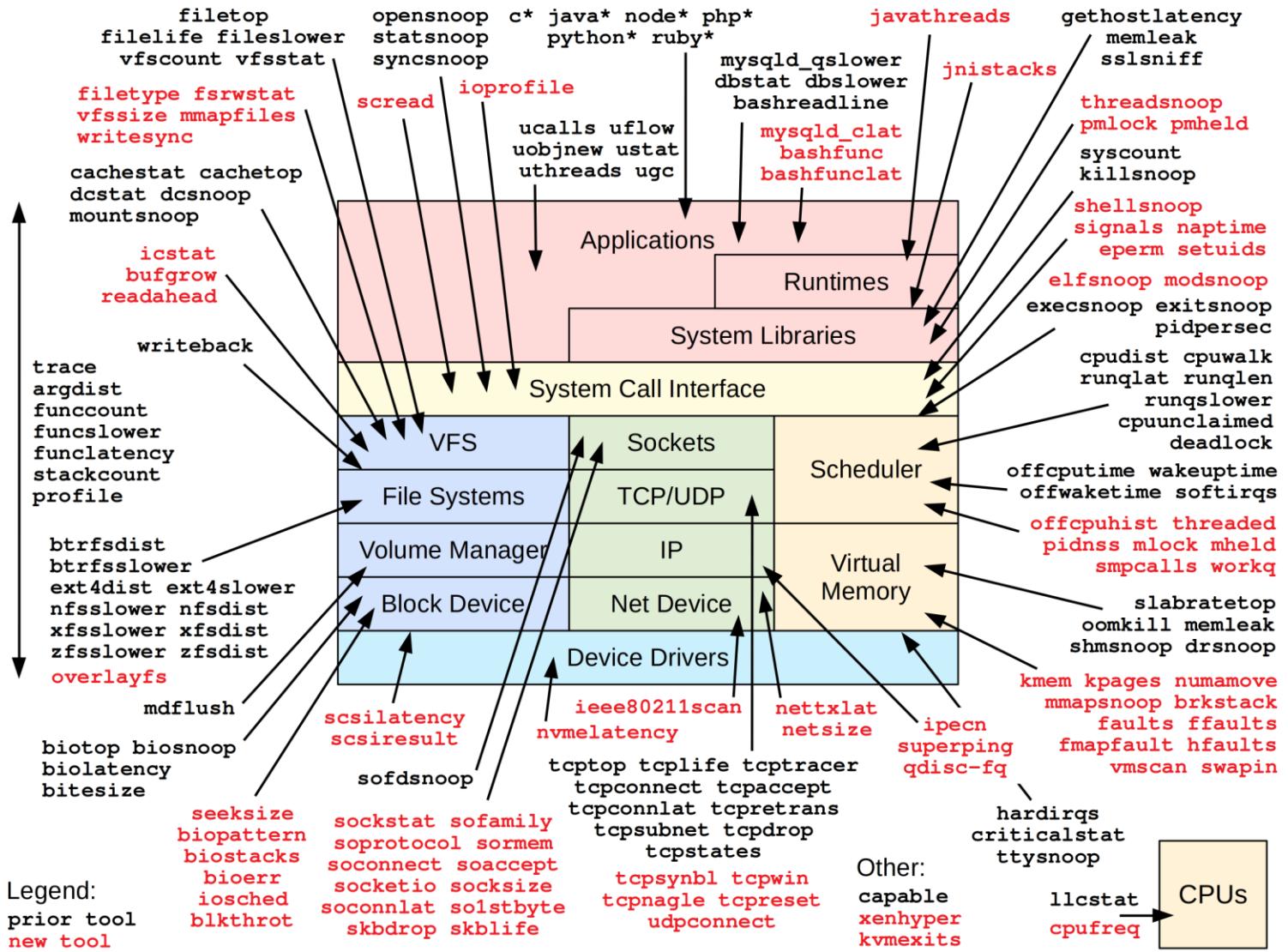
Linux Performance : example

Linux Performance Observability Tools





Linux Performance : example





DTrace

- n DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- n **Probes** fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes
- n Example of following **XEventsQueued** system call move from libc library to kernel and back

```
# ./all.d `pgrep xclock` XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                                U
  0  -> _XEventsQueued                            U
  0  -> _X11TransBytesReadable                     U
  0  <- _X11TransBytesReadable                     U
  0  -> _X11TransSocketBytesReadable              U
  0  <- _X11TransSocketBytesreadable              U
  0  -> ioctl                                      U
  0    -> ioctl                                 K
  0    -> getf                                  K
  0      -> set_active_fd                      K
  0      <- set_active_fd                      K
  0    <- getf                                 K
  0    -> get_udatamodel                     K
  0    <- get_udatamodel                     K
...
  0    -> releasef                           K
  0      -> clear_active_fd                 K
  0      <- clear_active_fd                 K
  0      -> cv_broadcast                      K
  0      <- cv_broadcast                      K
  0      <- releasef                         K
  0    <- ioctl                            K
  0    <- ioctl                           U
  0  <- _XEventsQueued                     U
  0 <- XEventsQueued                      U
```





Dtrace (Cont.)

- n DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

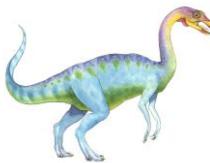
```
 sched:::on-cpu
uid == 101
{
    self->ts = timestamp;
}

 sched:::off-cpu
self->ts
{
    @time[execname] = sum(timestamp - self->ts);
    self->ts = 0;
}
```

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
      gnome-settings-d          142354
      gnome-vfs-daemon          158243
      dsdm                      189804
      wnck-applet                200030
      gnome-panel                 277864
      clock-applet                374916
      mapping-daemon              385475
      xscreensaver                514177
      metacity                     539281
      Xorg                         2579646
      gnome-terminal                5007269
      mixer_applet2                7388447
      java                        10769137
```

Figure 2.21 Output of the D code.





운영체제 생성 (Operating System Generation)

- ❖ 운영체제는 같은 종류의 어떤 기계에서도 수행되도록 설계된다; 운영체제는 특정한 컴퓨터 사이트에 대해 구성되어야(configure) 하는데 이 절차를 운영체제 생성(System Generation)이라고 한다
- ❖ **SYSGEN** 프로그램은 하드웨어 시스템의 구체적인 구성에 관한 정보를 얻는다
 - | 특정한 시스템에 맞추어 컴파일된 커널 또는 튜닝된 시스템을 만드는데 사용
 - | 한 개의 일반적인 커널보다 더 효율적인 코드를 생성할 수 있다





시스템 부트 (System Boot)

- ❖ 시스템에 전기가 들어오면, 고정된 메모리 위치에서 실행이 시작된다
 - | 펌웨어(firmware) ROM(Read Only Memory)이 초기 부팅 코드를 가지고 있다
- ❖ 운영체제를 하드웨어가 시작할 수 있도록 하드웨어에게 알려야 한다
 - | **ROM** 또는 **EPROM(Erasable Programmable ROM)**에 있는 **bootstrap loader**라 불리는 작은 코드가 커널을 찾아서 메모리에 적재하고 커널을 구동시킨다
 - | 두 단계로 나눠지는 경우도 있음- ROM의 특정한 위치에 있는 **부트블록(boot block)**이 메모리에 로드되고 이것이 디스크에서 부트스트랩 로더를 메모리에 적재한다
- ❖ **GRUB(GRand Unified Bootloader)** : Linux 시스템을 위한 오픈소스 부트스트랩 프로그램
- ❖ 커널이 적재되어 시스템이 실행(**running**)되게 된다



End of Chapter 2

