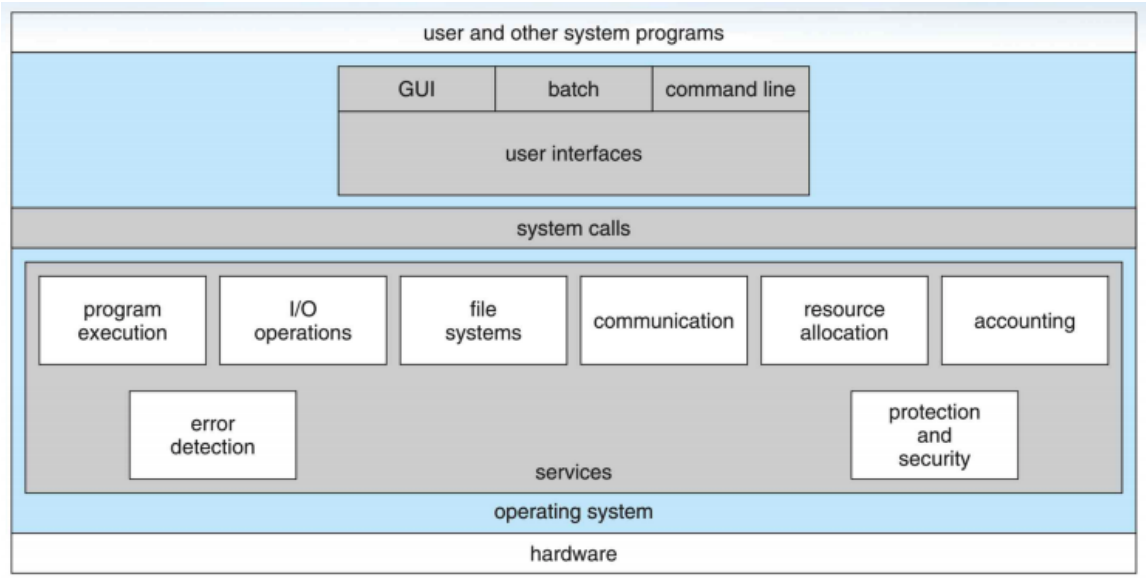
02. 운영체제 구조

운영체제는 프로그래밍이 실행되는 환경을 제공해 준다.

▼ 목차

1. 운영체제 서비스

▼ <Figure> 운영체제 서비스를 바라보는 관점

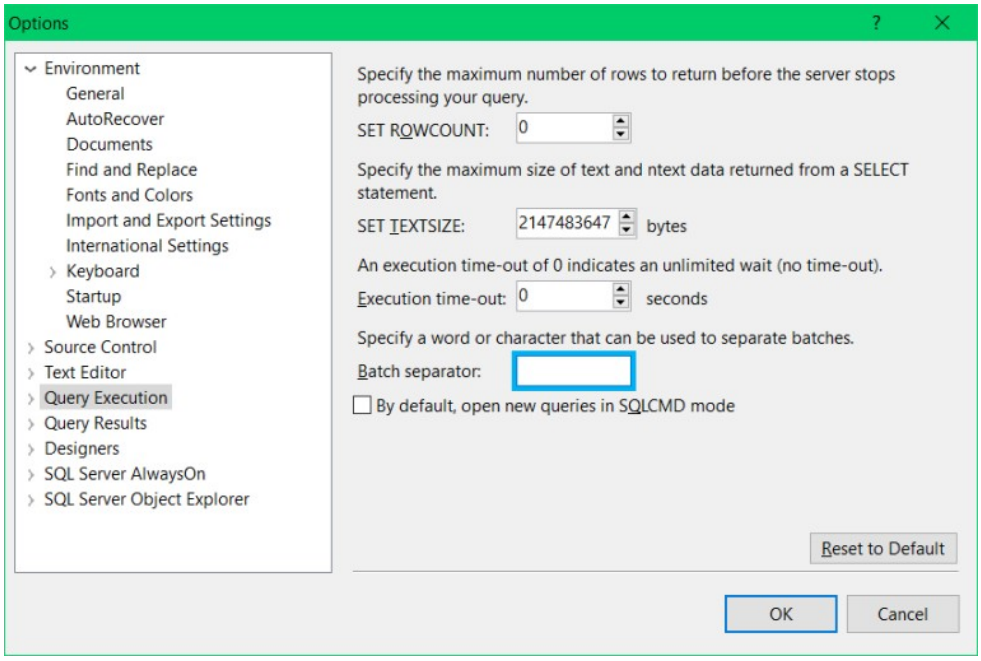


프로그래머의 편의성 제공

💡 운영체제는 프로그램 실행 환경을 제공한다.

- 사용자 인터페이스(User Interface, UI)
  - 거의 모든 운영체제가 UI를 가지고 있으면 여러 형태를 가질 수 있다.
    - 명령어 라인 인터페이스(Command-line interface, CLI)
    - Batch 인터페이스(Batch interface)

▼ <Figure> Batch 인터페이스 예시 → 정확한 의미? 터치?



    - 그래피컬 사용자 인터페이스(Graphics user interface, GUI)
- 프로그램 실행(program execution)
  - 프로그램을 메모리에 적재하여 실행할 수 있다.
  - 프로그램이 정상적이든 혹은 비정상적이든 실행을 끝낼 수 있다.
- 입출력 연산(I/O operaton)
  - 실행중인 프로그램에 의해 요구되는 입출력 동작을 위한 방법을 제공한다.
  - 파일, 입출력 장치
- 파일 시스템 조작
  - 프로그램이 파일을 읽고 쓸 수 있다.

    - 이름에 의해 파일을 생성하고 삭제할 수 있다.
    - 지정된 파일을 찾을 수 있고, 파일의 정보를 열거할 수 있다.
- 통신
    - 프로세스 간에 정보를 교환하는 방법을 제공한다.
        - 동일한 컴퓨터에서 실행되고 있는 프로세스들, 또는 컴퓨터 네트워크에 연결된 다른 컴퓨터 시스템 상에서 실행되는 프로세스들 사이
    - 공유 메모리, 메시지 전달
- 오류 탐지(error detection)
    - 모든 가능한 오류를 항상 탐지해야 한다.
        - 오류는 CPU, 메모리, 입출력 장치, 사용자 프로그램에 의해 발생할 수 있다.
    - 올바르고 일관성 있는 계산을 보장하기 위해 각 유형의 오류에 대해 적절한 조치를 취해야 한다.

시스템 자체의 효율적인 동작 보상

사용자에게 도움을 주는 것이 목적이 아니라, 시스템 자체의 효율적인 동작을 보장하기 위한 기능들을 제공한다.

- 자원 할당(resource allocation)
    - 다수의 사용자나 작업들이 동시에 실행될 때 각각에 자원들을 할당한다.
        - CPU 사이클, 주 메모리, 파일 저장 장치 등
    - 어떤 자원(CPU, 메모리, 입출력장치)들은 특수한 할당 코드를 가질 수 있고, 또 다른 자원들은 일반적인 요청/해제 코드를 가질 수 있다.
- 회계(accounting)
    - 사용자가 어떤 종류의 컴퓨터 자원을 얼마나 많이 사용하는지 등에 대한 기록을 보관한다.
- 보호와 보안
    - 다중 사용자 컴퓨터 시스템의 정보 소유자는 사용의 통제를 원할 수 있다.
    - 보호 : 시스템 자원에 대한 모든 접근이 통제되도록 보장

- 보안 : 사용자가 자원에 대한 접근을 허용 받기 위해 시스템에 자신을 인증하는 것으로부터 시작

2. 사용자 운영체제 인터페이스

- 사용자가 운영체제와 접촉하는 두 가지 방식
  - 명령어 해석기
  - 그래피컬 사용자 인터페이스

명령어 해석기

> Command Interpreter - CLI

💡 어떤 운영체제는 커널에 명령어 해석기를 포함(시스템 프로그램)하고 있고, 어떤 운영체제는 선택할 수 있는 여러 명령어 해석기를 제공(쉘)한다.

- Bourne shell, C shell, Bourne again shell, Korn shell, ...

명령어 해석기의 주요 기능

- 사용자가 지정한 명령들을 가져와서 그것을 실행하는 것
- 제공된 많은 명령들이 파일을 조작(생성, 삭제, 리스트, 프린트, 실행, 복사)한다.

명령어 구현의 두 가지 일반적인 방식

1. 명령어 해석기 자체가 명령을 실행할 코드를 갖고 있는 경우
   - 제공될 수 있는 명령의 수가 해석기의 크기를 결정
2. 시스템 프로그램에 의해 대부분의 명령을 구현하는 경우

- 명령어 해석기 프로그램이 아주 작아질 수 있고, 새로운 명령을 추가하기 위해 변경될 필요가 없다.

그래피컬 사용자 인터페이스

> Graphical User Interaface

💡 데크스톱이라고 특정지어지는 마우스를 기반으로 하는 윈도 메뉴 시스템을 사용

많은 시스템들이 CLI와 GUI 인터페이스를 모두 지원한다.

터치 스크린 인터페이스

> Touch-Screen Interface

💡 터치 스크린을 손가락으로 누르거나 스와이프 등의 제스쳐를 취하여 상호작용 하는 인터페이스

- Ipad, iPhone 둘다 Springboard 터치스크린 인터페이스를 사용한다.

---

3. 시스템 호출

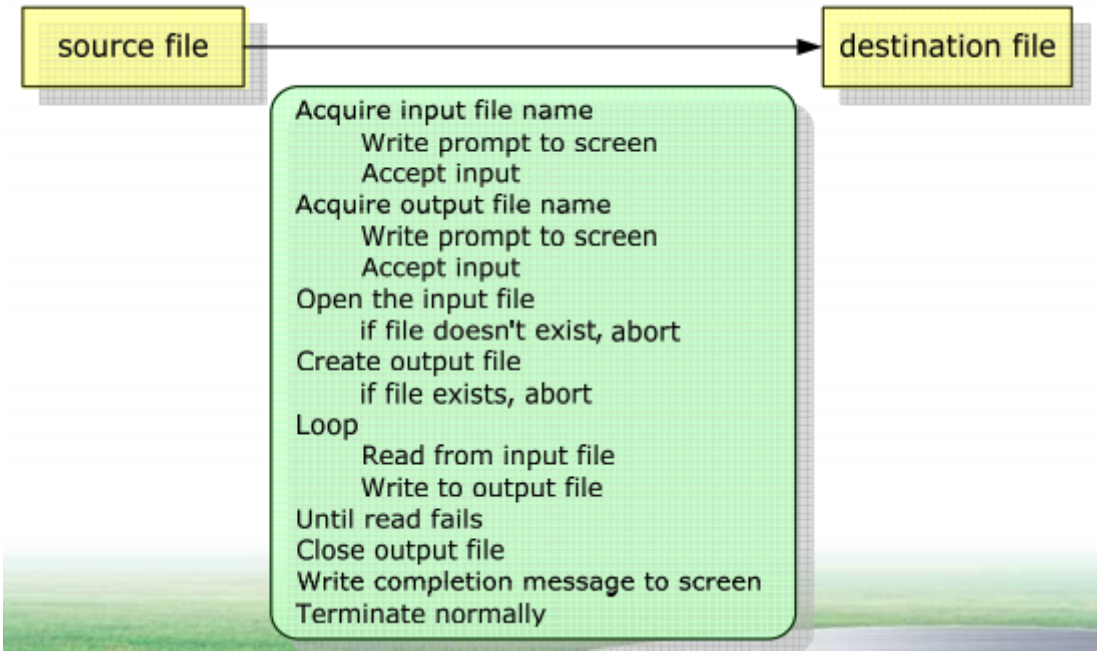
> System Call

시스템 호출의 사용

💡 시스템 호출은 운영체제에 의해 사용 가능한 서비스에 대한 인터페이스를 제공한다.

▼ <Figure> 시스템 호출 사용 예시

한 파일로부터 데이터를 다른 파일로 복사하기 위한 시스템 호출들



응용 프로그래밍 인터페이스

> Application Programming Interface - API

💡 대부분의 응용 개발자들은 응용 프로그래밍 인터페이스에 따라 프로그램을 설계한다. API를 구성하는 함수들은 통상 응용 프로그래머를 대신하여 실제 시스템을 호출한다.

▼ API ?

각 함수에 전달되어야 할 매개 변수들과 프로그래머가 기대할 수 있는 반환 값을 포함하여 응용 프로그래머가 사용 가능한 함수의 집합을 명시한 것

응용 프로그래머가 사용 가능한 가장 대중적인 세 가지 API

1. Windows API for Windows
2. POSIX API for POSIX-based systems
    - 거의 모든 버전의 UNIX, Linux 및 Mac OS X를 포함
3. Java API for the Java vitual machine(JVM)

표준 API 예

- 파일로부터 자료를 읽어 들이는 함수

시스템 호출 인터페이스

> 💡 응용 프로그래머는 실제 시스템 호출을 부르는 것 보다 API에 따라 프로그래밍 하는 것은 선호한다.

- 프로그램의 호환성
    - 프로그램이 같은 API를 지원하는 어느 시스템에서건 컴파일 되고 실행된다는 것을 기대한다.
- 시스템 호출은 API보다 어렵다.

운영체제가 제공하는 시스템 호출에 대한 연결 고리로서 동작하는 시스템 호출 인터페이스를 제공한다.

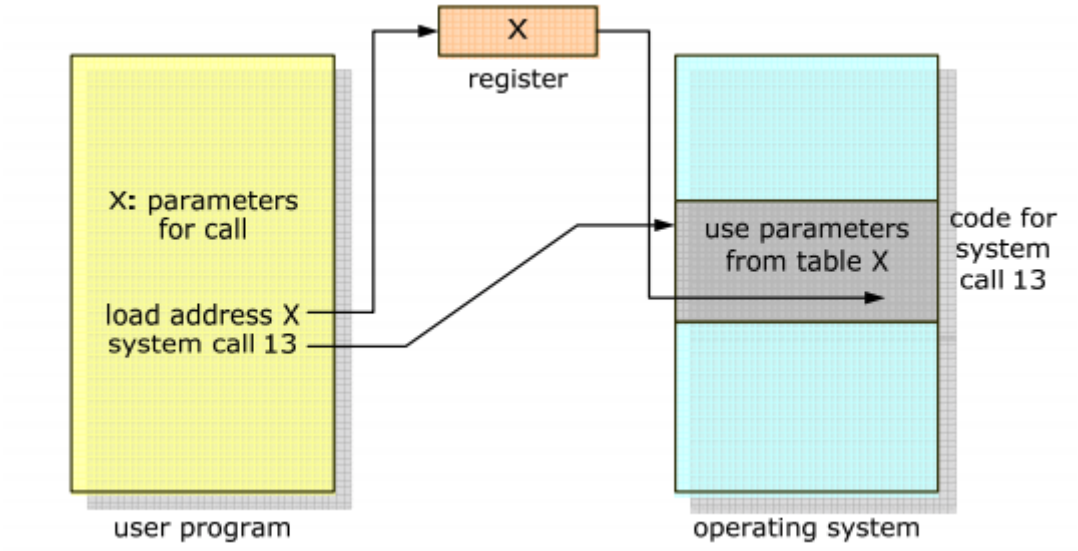
- 시스템 호출 인터페이스 : OS 커널의 해당 시스템 호출이 실행되도록 하며 시스템 호출의 결과 값과 상태를 반환한다.
    - 일반적으로 각 시스템 호출마다 하나의 번호가 연관된다.
        - 시스템 호출 인터페이스는 이들 번호에 따라 인덱스 되는 테이블을 유지한다.
- 호출자는 시스템 호출이 어떻게 구현되고 실행 중 무슨 작업을 하는지 알 필요가 없다.
  → 단지 API를 준수하고 시스템 호출의 결과로 운영체제가 무엇을 해줄 것인지 이해하면 된다.

운영체제에 매개변수를 전달하는 방법

시스템 호출 자체 보다 더 많은 정보를 요구한다면 매개 변수 전달을 요구해야 한다.

- 매개변수를 레지스터 내에 전달

  ▼ <Figure> 매개변수 전달



- 메모리 내의 블록이나 테이블에 저장하고 블록의 주소가 레지스터 내의 매개 변수로 전달
- 스택 방식으로 전송 : 프로그램에 의해 스택으로 저장되고 운영체제에 의해 pop 되면서 사용

4. 시스템 호출의 유형

프로세스 제어, 파일 조작, 장치 관리, 정보 유지보수, 통신

Windows와 Unix시스템 호출

▼ <Figure> 호출 명령어

	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() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

프로세스 제어

프로그램의 끝내기(end) 또는 중지(abort)

- 실행중인 프로그램은 실행을 정상 또는 비정상으로 끝낼 수 있어야 한다.
- 명령어 해석기, GUI를 통해 사용자에게 오류를 알리고 지시를 기다린다.

프로그램의 적재(load)와 실행(execute)

- 한 프로그램을 실행하고 있는 프로세스나 작업이 다른 프로그램을 적재하고 실행하기 원할 수 있다.
- 명령어 해석기가 사용자 명령, 마우스의 클릭 혹은 일괄 처리 명령 등을 통하여 지시된 프로그램을 실행하는 것을 허용한다.

프로세스의 생성(create)과 종료(terminate)

- 새로운 작업이나 프로세스를 생성하고 프로세스가 잘못되었거나 더 이상 필요 없을 때 종료한다.

프로세스 속성(process attribute)의 획득과 설정

- 작업의 우선 순위, 최대 허용 실행 시간 등을 포함하여 작업 혹은 프로세스의 속성들을 제어한다.

새로운 작업이나 프로세스를 생성한 후에는 실행이 끝나기를 기다림

- 시간 기다림(wait time)
- 사건 기다림(wait event)및 알림(signal event)

공유되는 데이터의 일관성 보장을 위해 공유 데이터를 잠글 수 있는 시스템 호출을 제공.

- 데이터가 잠겨 있는 동안에는 다른 프로세스의 접근을 막을 수 있게 한다.

파일 조작

> 💡 파일 관리를 위한 시스템 호출

- 파일 생성과 삭제
- 파일의 열기와 닫기
- 읽기, 쓰기, 위치 변경
- 파일 속성의 획득과 설정
    - 파일 이름, 형태, 보호 코드, 사용료 계산 정보 등

파일 시스템이 디렉토리 구조를 가졌다면 디렉토리에 대해서도 파일과 같은 동작이 필요하다.

장치 관리

프로그램의 작업을 계속 실행하기 위해 추가 자원을 필요로 할 수 있다.

- CPU, 디스크 드라이브, 파일 등 ...
- 운영체제에 의해 제어되는 다양한 자원들은 장치로 간주될 수 있다.

장치 관리를 위한 시스템 호출들

- 장치의 요청(request)와 방출(release)
  - 다수 사용자가 있는 경우에 장치 사용을 독점하기 위해 파일의 개방과 폐쇄에 대응
- 읽기, 쓰기, 위치 변경(reposition)
- 장치 속성의 획득과 지정
- 장치의 논리적 부착(attach) 또는 분리(detach)

정보의 유지

사용자 프로그램과 운영 체제 간의 정보 전달을 위해 존재하는 시스템 호출들

- 현재 시간과 날짜의 설정과 획득
- 시스템 자료의 설정과 획득
  - 사용자 수, 운영 체제 버전 등
- 프로세스, 파일, 장치 속성의 설정과 획득
  - 현재 운영되고 있는 작업과 처리에 관한 정보들

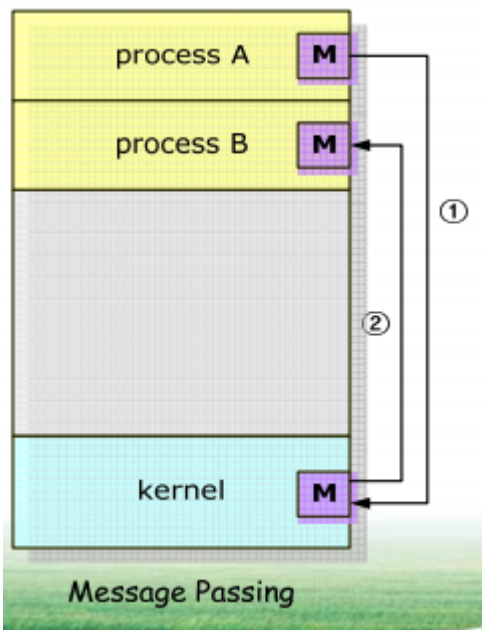
프로그램 디버깅을 위한 시스템 호출들

- 메모리 덤프
- 프로그램 추적
- 프로세스 상태 코드의 반환

통신

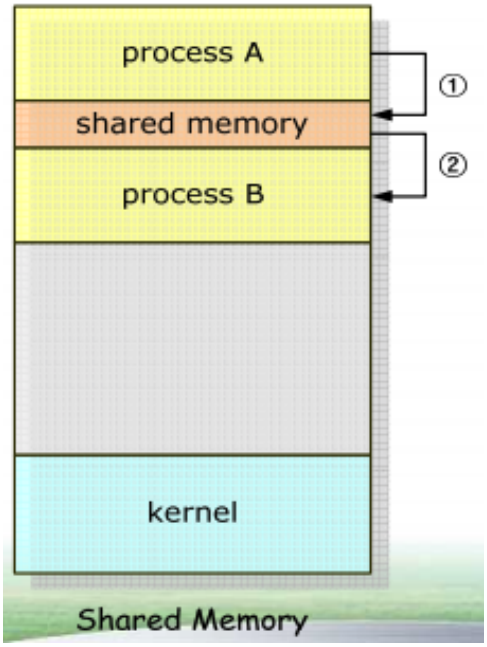
두 가지 일반적인 통신 모델

▼ 메시지 전달(message passing) 모델 : 프로세스 간 통신 기능을 통한 정보 교환



- 통신이 이루어지기 전에 연결이 반드시 열려야 한다.
- 상대 통신자가 동일한 CPU에 있는 프로세스이든지, 통신 네트워크에 의해 연결된 다른 컴퓨터에 있는 프로세스이든지 간에 그 이름을 반드시 알고 있어야 한다.

▼ 공유 기억장치(shared memory) 모델 : 두 개 이상의 프로세스들이 기억장치에 접근



- 정상적인 운영체제는 한 프로세스가 다른 프로세스의 메모리에 접근하는 것을 막으려고 한다. 공유 메모리는 두 개 이상의 프로세스가 이러한 제한을 제거하는 데 동

의할 것을 필요로 한다. 그 이후 공유 영역에서 데이터를 읽고 씀으로써 정보를 교환할 수 있다.

- 프로세서는 동일한 위치에서 동시에 쓰지 않도록 보장할 책임을 진다.

보호

- 컴퓨터 시스템이 제공하는 자원에 대한 접근을 제어하기 위한 기법을 지원

5. 시스템 서비스

> System Service 또는 시스템 프로그램이라고도 한다.

💡 시스템 서비스는 시스템 유틸리티(system utility)로도 알려진, 프로그램 개발과 실행을 위해 더 편리한 환경을 제공한다.

- 몇몇은 단순히 시스템 콜에 대한 사용자 인터페이스이며, 나머지는 훨씬 더 복잡하다.

시스템 서비스 범주

- 파일 관리 : 파일과 디렉토리 생성, 삭제, 복사, 개명, 인쇄, 덤프, 리스트 ...
- 상태 정보 : 시스템의 날짜, 시간, 사용 가능한 디스크 공간, 사용자 수 ...
- 파일 변경 : 텍스트 편집기들
- 프로그래밍 언어 지원 : 컴파일러, 어셈블러, 해석기(intetperter)
- 프로그램 적제와 수행 : 로더, 링키지 에디터, 디버거
- 통신 : 전자 우편, 원격지 로그인, 파일 전송
- 백그라운드 서비스 : 항상 실행되는 시스템 프로그램 프로세스 (서비스, 서브 시스템, 데몬)

→ 사용자 대부분이 보는 운영체제의 관점은 실제의 시스템 콜에 의해서보다는 시스템 프로그램과 응용에 의해 정의된다.

6. 링커와 로더

> Linkers and Loaders

💡 일반적으로 프로그램은 디스크에 이진 파일 형태로 존재한다. CPU에서 실행하려면 프로그램을 메모리로 가져와 프로세스 형태로 배치되어야 한다.
링커와 로더를 사용해서 이진 파일을 프로세스 형태로 만들 수 있다.

- 과정
  1. 소스 파일은 임의의 물리 메모리 위치에 적재되도록 설계된 오브젝트 파일로 컴파일 된다.
  2. 링커는 이런 재배치 가능 오브젝트 파일을 하나의 이진 실행 파일로 결합한다.
  3. 링킹 단계에서 표준 C 또는 수학 라이브러리와 같은 다른 오브젝트 파일 또는 라이브러리도 포함될 수 있다.
  4. 로더는 이진 실행 파일을 메모리에 적재하는 데 사용되며, CPU 코어에서 실행할 수 있는 상태가 된다.
  5. 링크 및 로드와 관련된 활동은 재배치로, 프로그램 부분에 최종 주소를 할당하고 프로그램 코드와 데이터를 해당 주소와 일치하도록 조정하여 프로그램이 실행될 때, 코드가 라이브러리 함수를 호출하고 변수에 접근할 수 있게 한다.

▼ `<Figure>` 링크와 로더의 역할

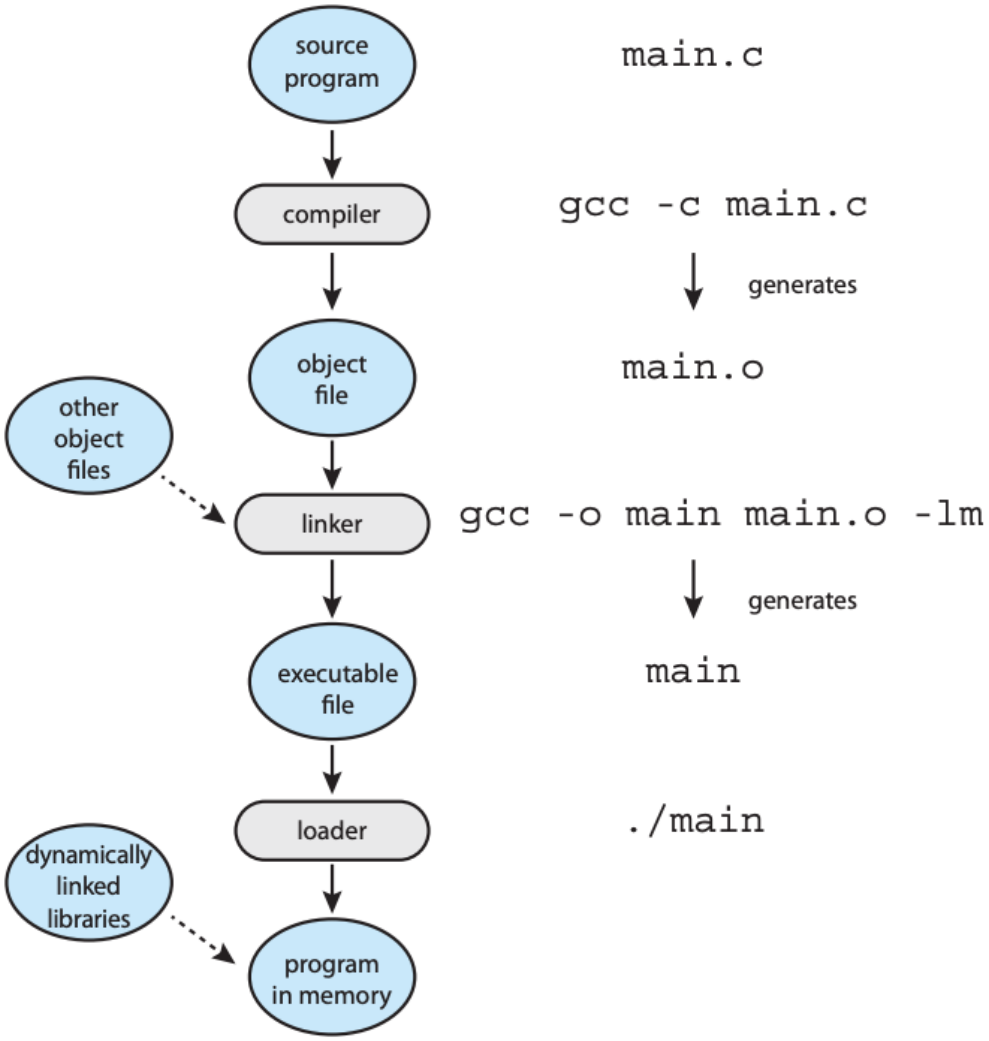


Figure 2.11 The role of the linker and loader.

7. 운영체제 설계 및 구현

> Operating System Design and Implementation

💡 운영체제를 설계하고 구현할 때 문제점을 논의한다.
완벽한 해결챌은 없지만 성공적인 접근 방법들은 있다.

설계 목표

- 시스템의 목표와 명세의 정의
  - 최상위 수준에서는 하드웨어와 시스템 타입의 선택에 의해 영향을 받는다.

    ▼ 응용 프로그램은 운영체제마다 다르다.

    각 운영체제는 고유한 시스템 콜 집합을 제공한다. 시스템 콜은 응용 프로그램이 사용할 수 있도록 운영체제가 제공하는
- 요구 조건의명세
  - 사용자 목적과 시스템 목적의 두 가지 기본 그룹으로 나눔
  - 사용자 그룹
    - 사용하기 쉽고 편리하며, 배우기 쉽고, 믿을 수 있고, 안전하고, 신속해야 한다.
  - 시스템 그룹 : 설계, 생성, 유지, 조작하는 사람들
    - 설계, 구현, 유지 보수가 쉬어야 하고 적응성, 신뢰성, 무 오류, 효율성이 있어야 한다.

기법과 정책

- 한 가지 중요한 원칙은 기법으로부터 정책을 분리하는 것이다.
  - 기법(mechanism) : 어떤 일을 어떻게 할 것인가를 결정하는 것
  - 정책(policy) : 무엇을 할 것인가를 결정하는 것
- 정책과 기법의 분리는 융통성을 위해 매우 중요하다.
  - 정책은 장소가 바뀌거나 시간이 흐름에 따라 변경될 수 있다.
  - 정책 결정은 모든 자원 할당 문제에 있어 중요하다.

▼ <Example> CPU 보호

- 기법 : 타이머 구조
- 정책 : 특정 사용자를 위한 타이머를 얼마나 오랫동안 설정할 것인지를 결정하는 것

구현

💡 초창기 운영 체제는 어셈블리어로 작성되어 왔으나 지금 대부분의 운영체제는 C와 같은 고급 언어나 C++와 같은 더 고수준의 언어로 작성된다.

- 커널의 가장 낮은 단계는 어셈블리어로 구현될 수 있고, 그 보다 높은 단계의 루틴은 C 언어로, 시스템 프로그램은 C, C++, Perl, Python또는 쉘 스크립트 같은 해석형 스크립팅 언어로 구현될 수 있다.

고급 언어로 작성했을 때의 장단점

- 장점 : 코드를 빨리 작성할 수 있고, 더욱 간결하고, 이해하기 쉽고, 디버깅하기 쉽고, 쉽게 이식 가능하다.
- 단점 : 속도가 느리고 기억장치가 많이 소요된다. → 현재의 시스템에서는 더 이상 중요한 문제가 아니다.

→ 현대의 운영체제의 주요 성능 향상은 우수한 어셈블리어 코드보다는 좋은 자료구조와 알고리즘의 결과일 가능성이 크다.

---

8. 운영체제 구조

> Operating System Structures

- 현대의 운영체제와 같이 크고 복잡한 시스템은 적절하게 동작하고 쉽게 변경할 수 있으려면 신중히 설계해야 한드.
- 일반적으로 한 개의 일관된 시스템 보다는 태스크를 작은 구성 요소로 분할한다.

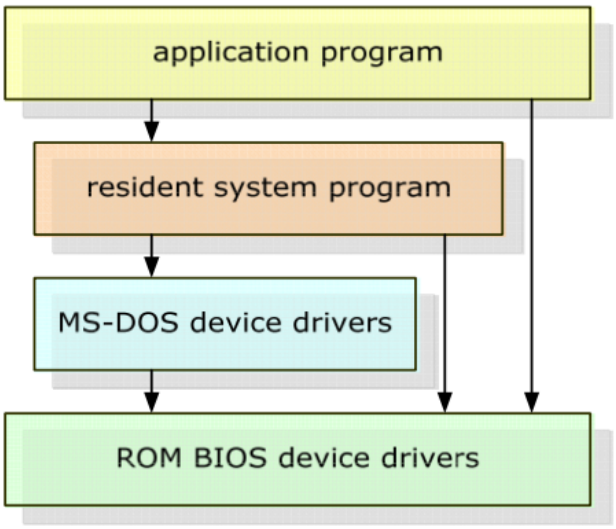
  → 각 모듈은 잘 정의된 입력, 출력, 그리고 기능들을 가지도록 한다.

모놀리식 구조

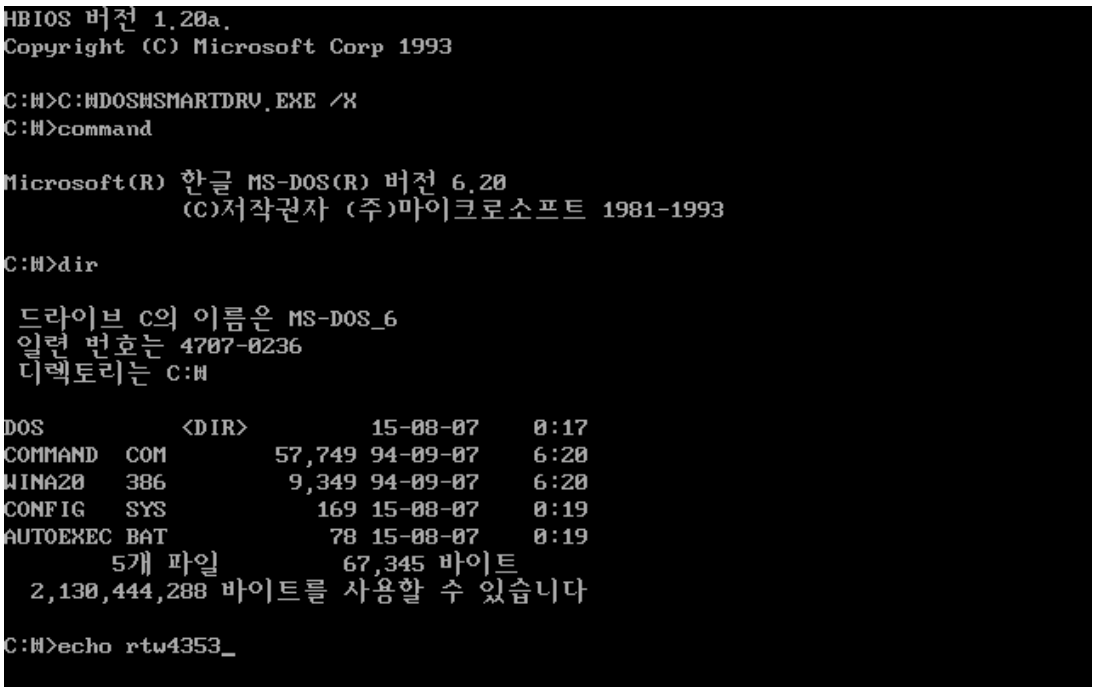
> MS : Monolithic Structure

💡 운영체제를 구성하는 가장 단순한 구조로 커널의 모든 기능을 단일 주소 공간에서 실행되는 단일 정적 이진 파일에 넣은 것

▼ <Figure> MS-DOS



- <Figure> 구동화면



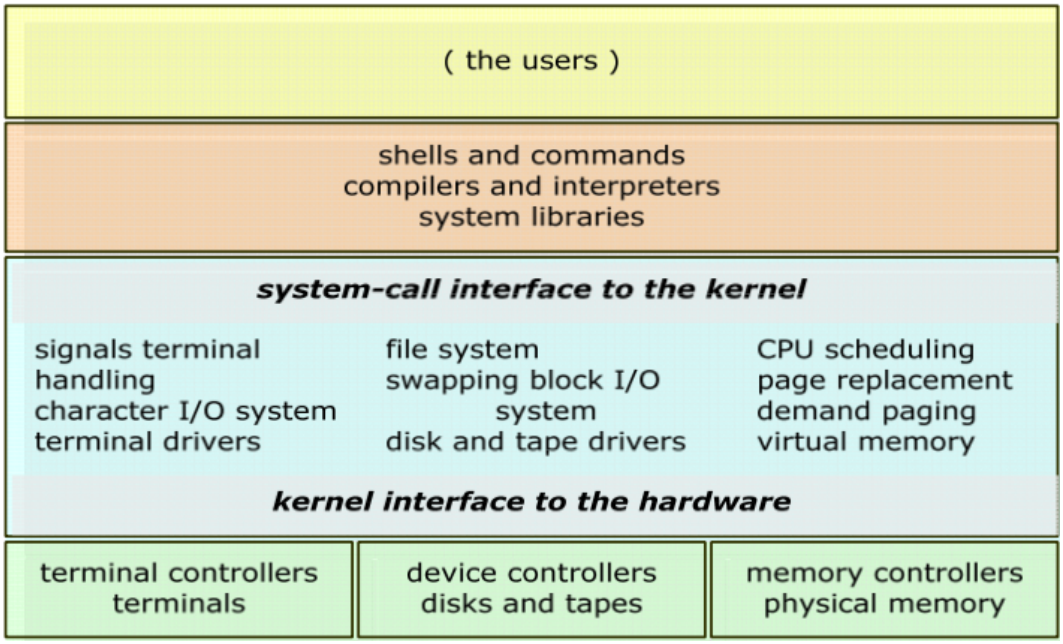
- 최소한의 공간에 최대의 기능들을 제공하도록 구현되었다.
- 모듈들로 효율적으로 분할되어 있지 않다.

- 인터페이스와 기능에 대한 계층이 잘 분리되어 있지 않다.
  - 기본 입출력 루틴을 통하여 디스플레이와 디스크 드라이브에 직접 쓰기가 가능하다.
- 오류가 있는 프로그램으로부터 취약하다.

  → 사용자 프로그램이 고장 나면 시스템 전체가 고장나게 된다.

초기 UNIX는 하드웨어 기능에 의해 제한 받는 또 다른 시스템이었다.

▼ <Figure> 전통적인 UNIX 시스템 구조



→ 모놀리식(monolithic) 구조 : 구현하기 어렵고 유지 보수하기도 어렵다.

- 성능 측면 장점 : 시스템 호출 인터페이스나 커널 안에서 통신하는 경우, 오버헤드가 거의 없다.

계층적 접근

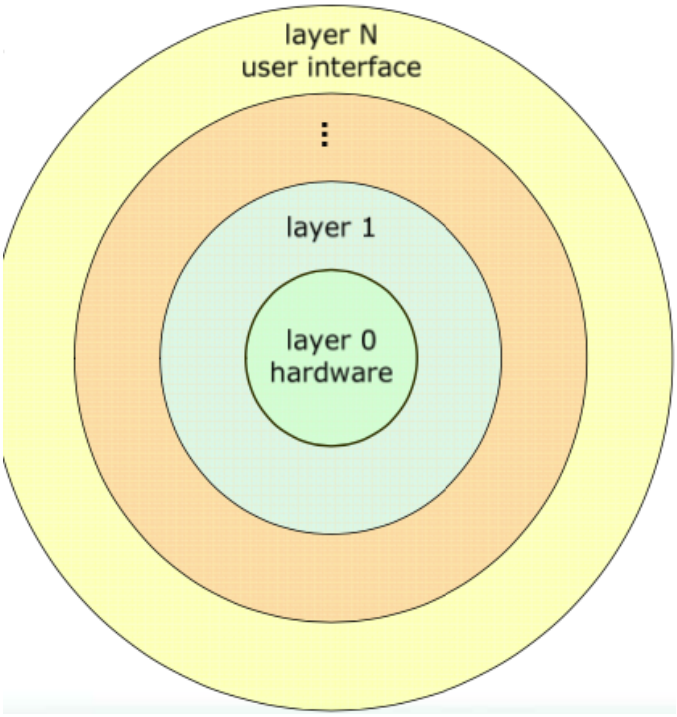
적절한 하드웨어 지원이 있을 경우, 작고 보다 적절한 조각으로 분할될 수 있다.

- 시스템은 다양한 방식으로 모듈화 될 수 있다.

계층적 접근 방식

- 운영체제가 여러 개의 계층으로 나뉘어진다.

  ▼ <Figure> 계층 구조의 운영체제



    - 최하위 층은 하드웨어이고 최상위 층은 사용자 인터페이스이다.
- 장점 : 간단한 구현과 디버깅
    - 각 계층은 자신의 하위 계층 서비스와 기능들 만을 사용하는 방식으로 구성된다.
    - 계층은 하위 계층에서 제공되는 연산들이 어떻게 실행되는지 알 필요가 없고 무엇을 하는지 알면 된다.
- 문제점
    - 여러 층을 적절히 정의하기 어렵다.
    - 다른 유형의 구현 방법보다 효율성이 낮다.

→ 현대의 운영체제에서는 어느정도의 계층화는 이루어지고 있다.

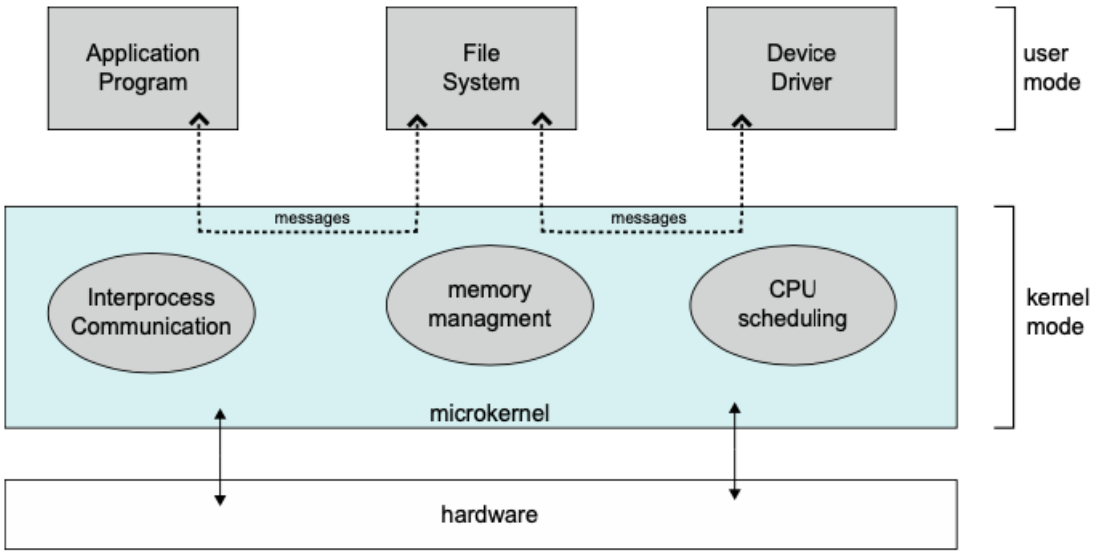
마이크로 커널

> Micro Kernel

> 💡 모든 중요하지 않은 구성 요소를 커널로부터 제거하고 이들을 시스템 및 사용자 수준 프로그램으로 구현하여 운영체제를 구성하는 방법 → 더 작은 커널

- 1980년대 중반, Carnegin Mellon University에서 커널을 모듈화한 마크(Mach) 운영체제 개발
- 클라이언트 프로그램과 사용자 공간에서 실행되는 다양한 서비스 간에 통신 설비를 제공한다.

▼ <Figure> 마이크로 커널 구성



마이크로 커널 접근법의 장단점

- 장점
  - 운영체제 확장에 용의
  - 다른 하드웨어로의 이식이 쉽다.
  - 높은 보안성과 신뢰성 제공
- 단점
  - 가중된 시스템 기능 오버헤드 때문에 성능이 감소된다.

모듈

Module

> 💡 적재 가능 커널 모듈(LKM ; Loadable kernel modules)기법의 사용
> 커널은 핵심적인 구성요소의 집합을 가지고 있고 부팅 때 또는 실행 중에 부가적인 서비스들은 모듈을 통하여 링크할 수 있다.

- 커널은 핵심 서비스를 제공하고 다른 서비스들은 커널이 실행되는 동안 동적으로 구현한다.
- 모듈에서 임의의 다른 모듈을 호출할 수 있다는 점에서 계층 구조보다 유연하다.

하이브리드 시스템

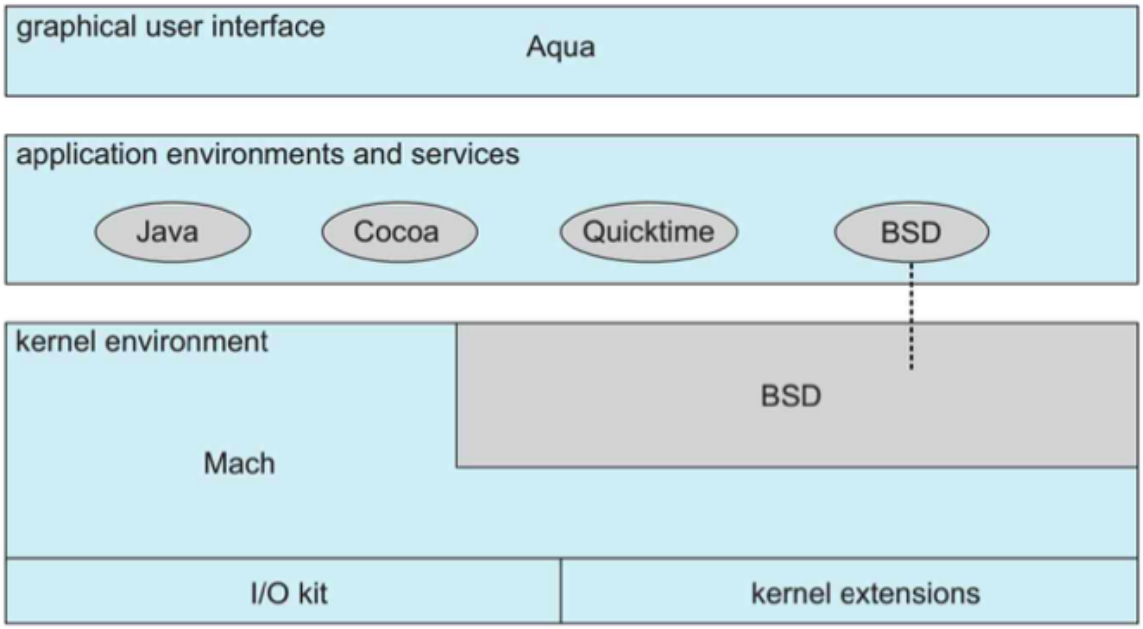
> 💡 엄격하게 정의된 하나의 구조를 채택한 운영체제는 거의 존재하지 않는다. 대신 다양한 구조를 결합하여 성능, 보안 및 편의성 문제를 해결하려는 혼용 구조로 구성된다.

- Linux, Solaris : 모놀리식 + 모듈
  - 운영체제 전부가 하나의 주소 공간에 존재하여 효율적인 성능 제공
  - 모듈을 사용하여 새로운 기능을 동적으로 커널에 추가할 수 있다.
- Windows : 모놀리식 + 마이크로 + 모듈
  - 대체적으로 모놀리식 구조
  - 사용자 모드 프로세스로 실행되는 분리된 서브 시스템 지원
  - 동적으로 적재 가능한 커널 모듈 지원
- MacOS X, iOS, Android, ...

Mac OS X

> 💡 최상위 층은 응용 환경과 응용에 그래피컬 인터페이스를 제공하기 위한 서비스의 집합을 포함

▼ <Figure> Mac OS X Operating System



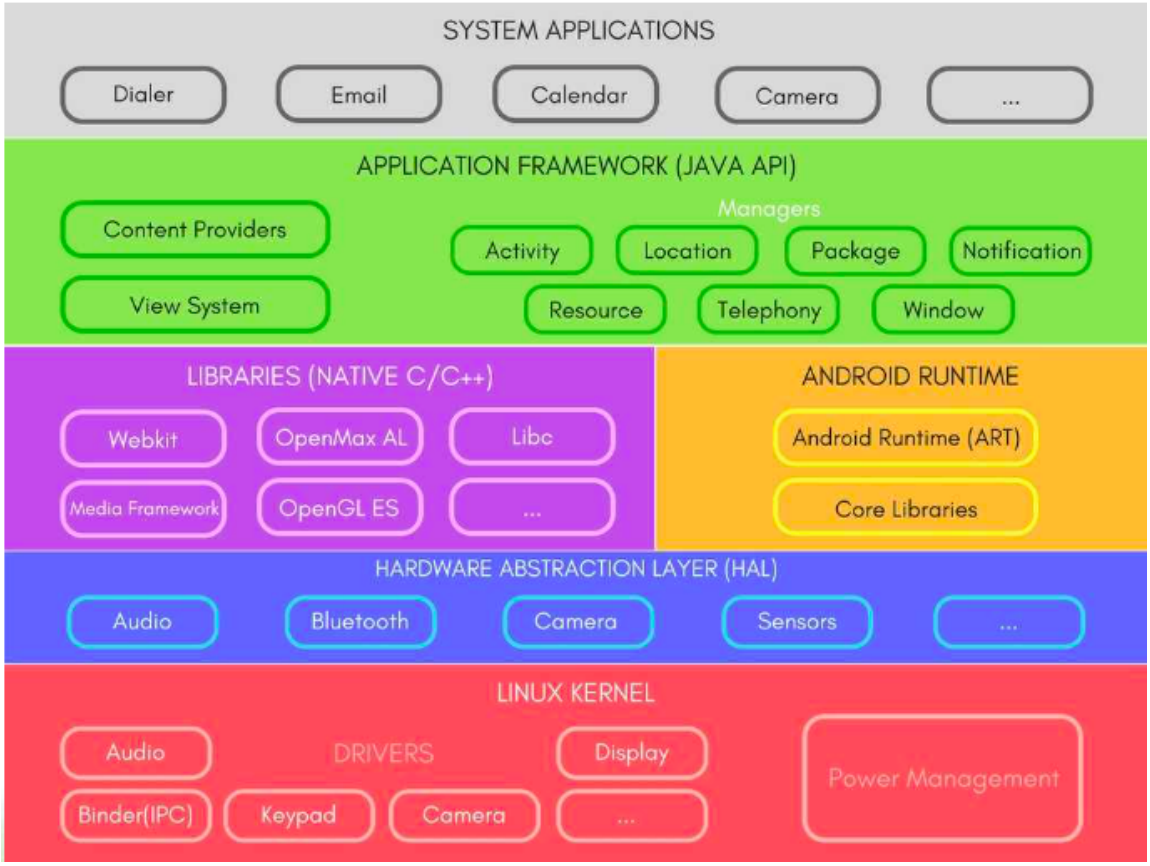
- Mach 마이크로 커널
  - 메모리 관리, 메시지 전달과 함께 원격 함수 호출(RPC)과 프로세스 간 통신 설비 지원
- BSD 커널
  - BSD 명령어 라인 인터페이스, 네트워킹 및 파일시스템 지원, POSIX SPI 구현 제공

Android

💡 모바일 응용을 개발하기 위한 풍부한 프레임워크를 제공하는 소프트웨어의 계층 구조로 구성

- 제일 아래 층에 리눅스 커널 존재
  - 프로세스, 메모리 및 하드웨어를 위한 장치 드라이버 지원에 주로 이용되며 전력 관리 기능 추가
- Android 실행 환경
  - 핵심적인 라이브러리 집합과 Android 런타임을 포함

▼ <Figure> Android Operating System



9. 운영체제 빌딩과 부팅

> Building and Booting an Operating System

하나의 특정 기기 구성에 맞게 운영체제를 설계, 코딩 및 구현할 수 있지만 일반적으로 운영체제는 다양한 주변장치 구성을 가진 모든 종류의 컴퓨터에서 실행되도록 설계한다.

운영체제 생성

> Operating-System Generation

💡 컴퓨터에 적절한 운영체제를 설치하고 사용할 수 있도록 구성하는 몇 가지 옵션

1. 운영체제 소스 코드를 작성한다. (이전에 작성된 소스 코드를 확보한다.)
2. 운영체제가 실행될 시스템의 운영체제를 구성한다.
3. 운영체제를 컴파일한다.
4. 운영체제를 설치한다.
5. 컴퓨터와 새 운영체제를 부팅한다.

시스템을 구성하려면 어떤 기능이 포함되는지 명시해야 하며 이는 운영체제에 따라 다르다. 일반적으로 시스템 구성 방법을 설명하는 매개변수는 특정 유형의 구성 파일에 저장되며 이 파일을 만든 후에는 여러가지 방법으로 사용할 수 있다.

운영체제 생성 방식

운영체제 생성은 생성된 시스템의 크기 및 일반성과 하드웨어 구성이 변경될 때 얼마나 변경이 쉬운가에 달려 있다.

1. 시스템 관리자가 운영체제 소스 코드의 사본을 수정할 수 있다. 그런 다음 운영체제가 컴파일 된다. 컴파일 시 주어진 데이터 선언, 초기화 및 상수는 구성 파일에 설명된 시스템에 맞는 운영체제의 출력-오브젝트 버전을 생성한다.

   → 임베디드 시스템의 경우 많이 사용한다.

2. 상세한 조정을 할 수 없는 부분에서는 시스템 설명을 통하여 기존 라이브러리에서 사전 컴파일된 오브젝트 모듈을 선택할 수 있다. 이 모듈들이 서로 링크되어 새 운영체제가 생성된다.
   조금 극단적으로는, 완전히 모듈 방식으로 시스템을 구성할 수 있다. 여기서 선택은 컴파일 또는 링크 시간이 아닌 실행 시간에 일어난다.

   → 데스크톱 및 랩톱 컴퓨터와 모바일 장치를 지원하는 대부분의 최신 운영체제가 선택한다.

시스템 부트

> System Boot

커널을 적재하여 컴퓨터를 시작하는 과정을 부팅이라고한다.

대부분의 시스템의 부팅 과정

1. 부트스트랩 프로그램 또는 부트 로더라고 불리는 작은 코드가 커널의 위치를 찾는다.
2. 커널이 메모리에 적재되고 시작된다.
3. 커널은 하드웨어를 초기화 한다.
4. 루트 파일 시스템이 마운트 된다.