운영체제의 이해를 위한 기초

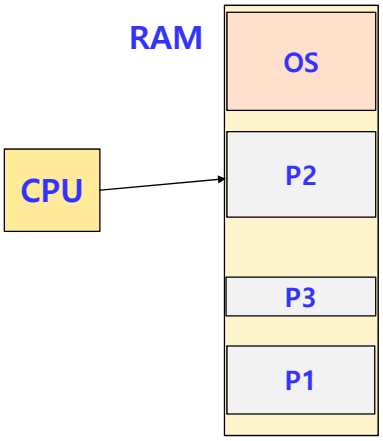
- ✓운영체제의 사용 목적
- ✓운영체제의 구성
- ✓운영체제와 응용프로그램의 실행
- ✓기본적인 용어 정의

이해하고 넘어가야 할 내용들

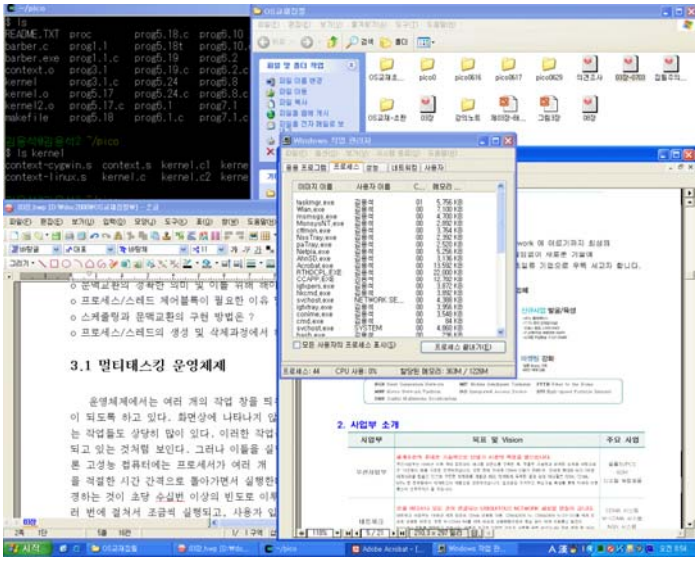
- ✓ 운영체제를 사용함으로써 얻을 수 있는 장점 및 이로 인한 오버헤드
- ✓ 운영체제 / 명령 해석기 / 프로세스 간의 연관성
- ✓ 운영체제 구성요소들 및 이들의 상호 연관성
- ✓ 컴퓨터 부팅 과정
- ✓ 기본적인 용어들

CPU의 프로그램 실행

- ✓ CPU는 메모리에서 명령어를 읽어와서 실행
- ✓ 응용프로세스들과 운영체제 부분을 바꿔가면서
- ✓ 어떻게?
- ✓ CPU의 Core가 여러 개면?
- ✓ 주변장치 구동은?



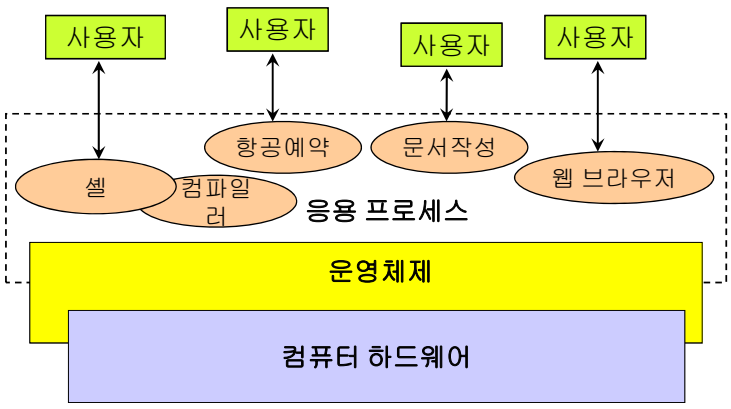
멀티태스킹 운영체제



운영체제의 사용

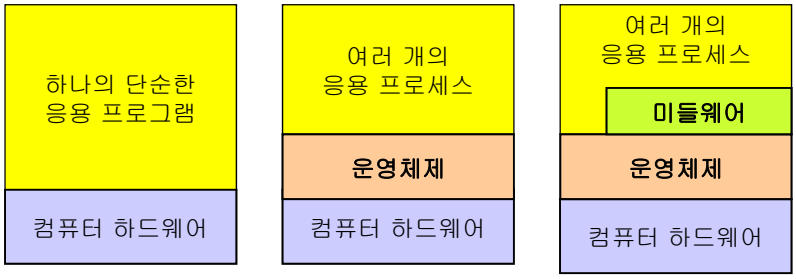
- ✓ 멀티태스킹 운영체제란
  - ✓ 여러 개의 프로세스들이 독립적으로 실행되도록 해주는 운영체제
  - ✓ 스케줄링과 문맥교환 작업 필요
- ✓ 스케줄링
  - ✓ 프로세스들 중에서 CPU가 실행할 프로세스를 선정하는 작업
- ✓ 문맥교환
  - ✓ 현재 실행하던 프로세스를 보류하고 스케줄링에 의해 선정된 프로세스를 실행하도록 전환하는 절차
- ✓ 운영체제의 사용
  - ✓ 사용자 인터페이스
    - ✓ 명령해석기 (command interpreter)
    - ✓ Graphic Desktop Environment (Graphic User Interface)
  - ✓ 사용자가 원하는 프로그램을 지정하여 프로세스로 생성되도록 운영체제에 요청
  - ✓ 프로세스는 실행 중에 운영체제의 각종 기능을 호출하여 사용
  - ✓ 운영체제는 프로세스들이 적절히 돌아가면서 실행되도록 처리

컴퓨터시스템의 개념적인 구성



✓ 하드웨어는 운영체제를 통해서 접근함 ➔ 왜???

운영체제와 미들웨어



- ✓ 미들웨어 (Middleware)
  - ✓ 운영체제에서는 제공하지 않지만 응용프로세스들에서 필요로 하는 고도의 기능 제공

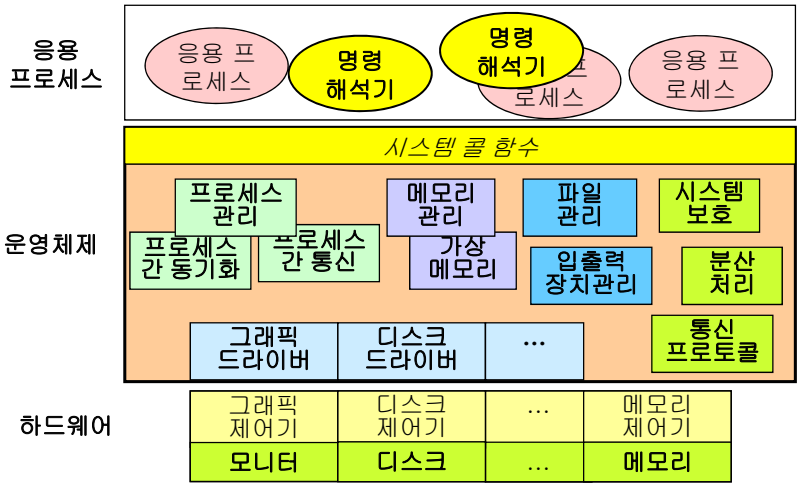
운영체제를 사용하는 목적

목적	설 명
하드웨어 자원의 효율적인 활용	CPU, 메모리, 입출력 장치 등의 자원을 프로세스들 간에 공유
컴퓨터 사용의 편리성 향상	응용 프로그램의 설치 및 실행 방법, 많은 수의 파일들의 접근 방법 등에 있어서 간편하게 명령어나 마우스 클릭으로 처리
프로그램 개발의 편리성 향상	하드웨어의 구체적인 지식 없이 하드웨어 제어 가능 메모리 용량이나 디스크상의 파일 배치 등 고려 불필요 멀티태스킹 프로그램 작성 가능
프로그램의 이식성 향상	하드웨어의 구성이 달라져도 실행 파일을 그대로 사용할 수 있거나, 컴파일만 다시 해주면 실행 가능
컴퓨터의 안전성 향상	프로그램의 오류나 바이러스 프로그램 등으로부터 컴퓨터 시스템을 보호
컴퓨터의 보안 강화	시스템을 부정사용으로부터 보호하기 위해 사용자 ID, 패스워드, 프로세스 및 파일의 접근 권한 등을 활용

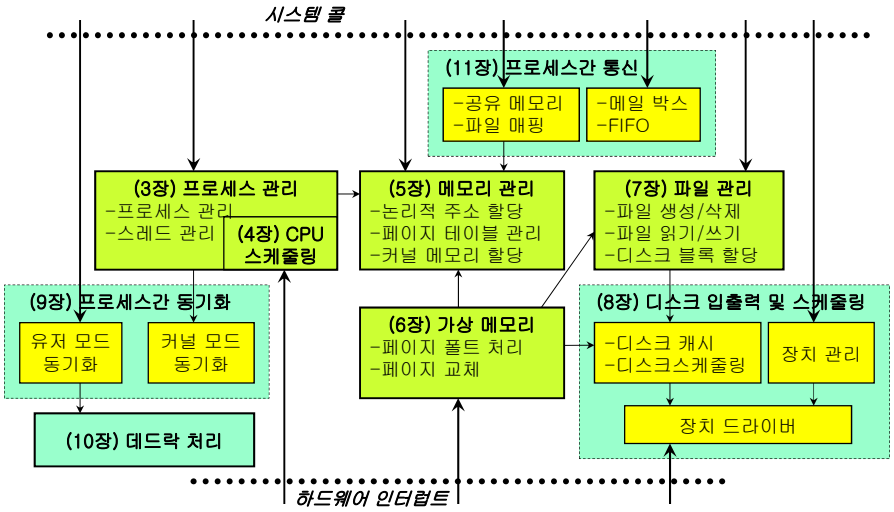
운영체제의 내부를 알면

- ✓ 운영체제의 개발 및 개선
- ✓ 새로운 하드웨어에 운영체제 이식
- ✓ 컴퓨터 주변장치를 위한 장치 드라이버 개발
- ✓ 속도가 빠른 응용 프로그램의 개발
- ✓ 컴퓨터 시스템 관리 능력의 심화

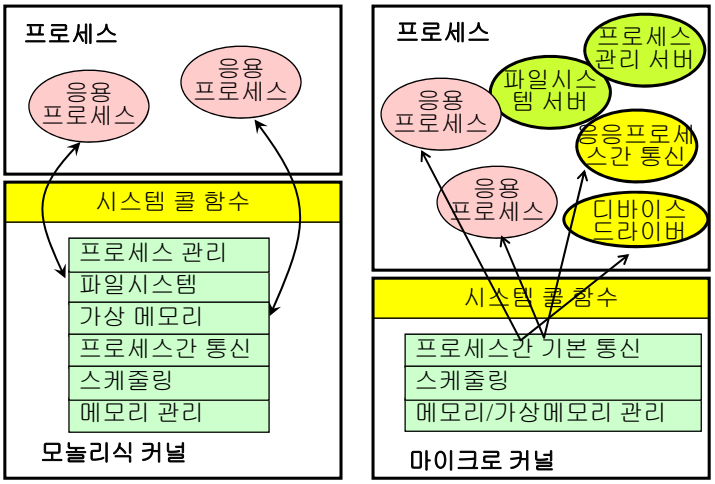
운영체제의 구성



운영체제 모듈간의 호출관계

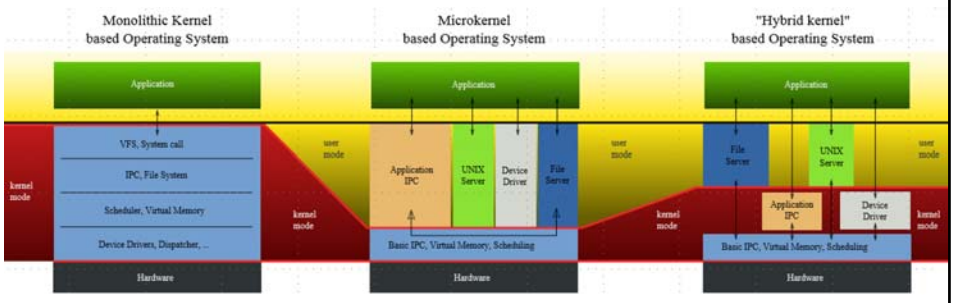


모놀리식커널과 마이크로커널



✓ Monolithic : 하나의 덩어리로 된

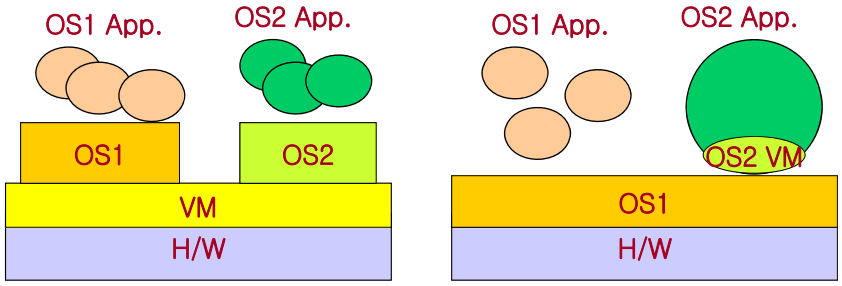
Monolithic, Micro, Hybrid Kernel



(출처: Wikipedia)

버추얼 머신 (Virtual Machine)

- ✓ System Virtual Machine
  - ✓ 하나의 시스템에 복수의 운영체제 실행
  - ✓ VM 위에 사용자 별로 선호하는 운영체제를 실행
- ✓ Process Virtual Machine
  - ✓ 프로세스 마다 독립된 운영체제 환경 제공
  - ✓ 1. System Call을 변환하는 Library 제공 (Cygwin, picoKernel, ...)
  - ✓ 2. 기계어 코드까지 에뮬레이션 (JVM, ...)



System Virtual Machine

- ✓ 여러 개의 운영체제를 동시에 실행
- ✓ 자원의 분할 목적
  - ✓ 하나의 서버 하드웨어를 여러 개의 논리적 서버 (VM)들로 분할
  - ✓ 업무별로 논리적 서버들을 필요에 따라서 조절 (예: 수강신청 시점에는 이를 위해 VM들을 더 많이 할당)
- ✓ 특수한 환경 지원 목적
  - ✓ 특정 응용 소프트웨어에 맞는 OS 적용
  - ✓ 개발중인 운영체제 / 믿을 수 없는 응용 들을 위한 Sandbox
  - ✓ 다른 하드웨어를 위한 / 옛 버전의 운영체제 제공
- ✓ 예
  - ✓ Vmware, Xen, Citrix, Linux+RTOS, IBM z/VM (z/OS+Linux)

Process Virtual Machine (1)

- ✓ System Call Emulation
  - ✓ Guest OS의 System Call을 Host OS 의 System Call 집합을 활용하여 처리
- ✓ Windows
  - ✓ Win32 Application → Win32
  - ✓ Win16 Application + Win16 VDM → Win32
  - ✓ MS-DOS Application + MS-DOS VDM → Win32
  - ✓ POSIX Application + POSIX Subsystem → Win32
- ✓ Cygwin
  - ✓ Linux Application + Cygwin DLL → Win32
- ✓ picoKernel
  - ✓ picoKernel Application + picoKernel → Linux
  - ✓ picoKernel Application + picoKernel + Cygwin DLL → Win32

Process Virtual Machine (2)

- ✓ Code Emulation
  - ✓ Language / application virtual machine
  - ✓ 기능: 별도로 정의된 기계어 코드를 VM이 실행
  - ✓ 목적: 프로그램의 이식성 향상
  - ✓ 방법: code interpreter, just-in-time compilation
- ✓ Java 언어의 Java Virtual Machine (JVM)
  - ✓ Java 언어 프로그램을 Bytecode 로 컴파일 후 JVM으로 실행 (.java→.class)
- ✓ Android의 Dalvik Virtual Machine (DVM)
  - ✓ 메모리 용량이 작고 느린 CPU의 휴대형 단말에 최적화
  - ✓ Java byte code 를 DVM 에 맞게 변환 (.java→.dx)
- ✓ .NET Framework의 Common Language Runtime (CLR)
  - ✓ C#, Visual C, Visual Basic 등의 공통의 CIL (Common Intermediate Language)로 컴파일 후 JIT 컴파일 적용

ART (Android RunTime)

- ✓ 초기 안드로이드는 DVM 적용
  - ✓ Java bytecode 프로그램을 Dalvik VM code로 변환
  - ✓ 실행은 Dalvik VM이 interpret
- ✓ 현재의 안드로이드는 ART 적용
  - ✓ 디바이스에 설치시에 native code로 AOT (Ahead Of Time) 컴파일 하여 ELF 실행파일로 변환
  - ✓ 실행은 ELF 실행파일을 그대로
  - ✓ 안드로이드 4.4 KitKat 이후

응용프로그램의 작성 및 실행

- ✓ 실행파일 생성
  - ✓ 프로그램 작성 및 컴파일
  - ✓ 자체 컴파일러 (native compiler)와 Cross Compiler
- ✓ 프로그램 적재
  - ✓ 프로그램을 CPU가 실행할 수 있도록 메모리에 적재
  - ✓ 파일, ROM, 다른 컴퓨터로부터 down loading 등
- ✓ 프로그램 실행 시작
  - ✓ (문맥교환을 통해서)
  - ✓ CPU의 명령어 주소 레지스터에 적재된 프로그램의 시작주소 기록
  - ✓ CPU는 지정된 주소의 기계어 코드를 읽고 실행

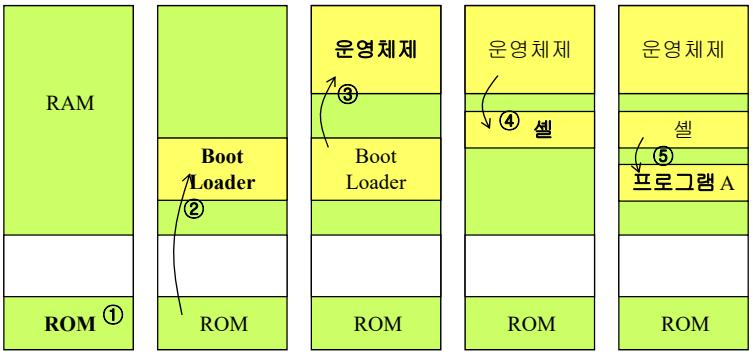
명령 해석기

- ✓ 명령 해석기 (command interpreter)
  - ✓ 사용자로부터 입력 받은 정보에 따라 실행파일을 찾고 이를 실행해 주도록 운영체제에 요청하는 역할 수행
  - ✓ UNIX Shell, Windows cmd / Graphic Desktop Environment
- ✓ UNIX Shell (Windows cmd)의 동작
  - ✓ 1. (shell) 프롬프트 출력 및 명령 줄 읽기 대기
  - ✓ 2. (사용자) 명령 줄 입력 (실행할 프로그램 파일 지정)
  - ✓ 3. (shell) 첫 단어를 파일이름으로 OS에게 실행을 요청
  - ✓ 4. (OS) 메모리 공간을 확보하고 파일을 적재한 후 여기로 실행위치 이동
  - ✓ 5. (프로그램) 실행 후에 종료
  - ✓ 6. (OS) 메모리를 해제하고 shell 로 실행위치 이동
  - ✓ 7. (shell) 1번부터 다시 반복

극히 간단한 UNIX Shell

```
main()
{
  int childpid, exitcode;
  char command[20];
  while (1) {
     printf("shell> ");
     scanf("%s", command);
     if ( ! strcmp(command, "exit") )
        exit(0);
     childpid = fork();
     if (childpid == 0) {
        execlp(command, command, 0);
        printf("command not found\n");
        exit(1);
     }
     wait(&exitcode);
  }
}
```


일반적인 부팅 과정



✓Kernel : 부팅이후 전원을 끌 때까지 메모리에 상주하면서 언제든지 실행될 수 있는 운영체제 부분

기본적인 용어 정의

- ✓ 운영체제 (operating system)
  - ✓ 응용프로그램들을 효율적이고 편리하며 안전하게 실행할 수 있도록 지원하는 소프트웨어
  - ✓ 좁은 의미의 운영체제: 커널만을 의미함
  - ✓ 넓은 의미의 운영체제: 커널뿐만 아니라 기본적인 각종 프로그램들도 포함 (명령 해석기, 시스템 관리 툴 등)
- ✓ 시스템 소프트웨어 (system software)
  - ✓ 넓은 의미의 운영체제에 포함되는 소프트웨어 전부 + SW 개발 툴
- ✓ 응용 소프트웨어 (application software)
  - ✓ 사용자가 원하는 특정 작업을 위한 소프트웨어

기본적인 용어 정의 (2)

- ✓ 프로그램/프로세스/스레드
  - ✓ 프로그램: 컴퓨터가 실행해야 할 내용을 명시적으로 표현한 문서
  - ✓ 프로세스: 특정 프로그램이 실행중인 상태인 능동적인 개체
  - ✓ 스레드: 프로세스를 독립적인 실행단위로 세분화한 개체
- ✓ 멀티태스킹 시스템 (multi-tasking system)
  - ✓ 여러 개의 프로세스들이 동시에 실행되도록 처리
  - ✓ CPU가 하나라면 실제로는 프로세스간에 돌아가면서 실행
- ✓ 멀티유저 시스템 (multi-user system)
  - ✓ 여러 사람이 동시에 사용할 수 있는 시스템
  - ✓ 멀티태스킹 기능이 제공되어야 하고, 계정 관리, 파일/프로세스들을 사용자별로 보호하는 기능 필요
- ✓ (참고)
  - ✓ Multi-programming system
  - ✓ Time sharing system