1주차 Computer System Overview / OS Overview / Process Management

▼ Lecture 1. Computer System Overview

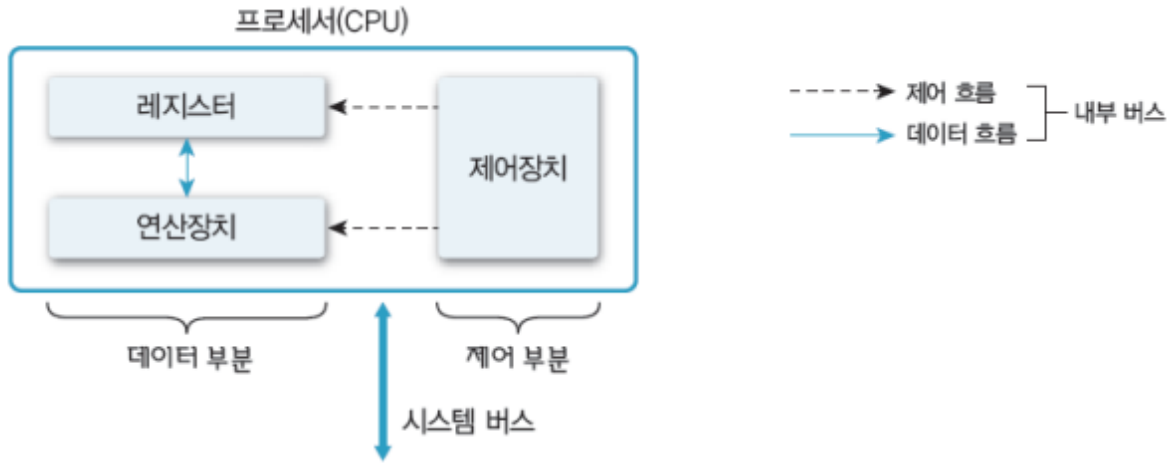
os란

- **하드웨어**를 효율적으로 관리해서 사용자 혹은 응용 프로그램에게 서비스를 제공

하드웨어 종류

- 프로세서 (Processor) -
  - CPU
  - 그래픽카드 (GPU)
  - 응용 전용 처리장치 등
- 메모리 (Memory) - 저장 담당
  - 주 기억장치
  - 보조 기억장치 등
- 주변장치
  - 키보드/마우스 -입력 장치
  - 모니터, 프린터 - 출력 장치
  - 네트워크 모뎀 등 - 네트워크 장치

프로세서 (Processor)



- 계산, 컴퓨터 모든 장치의 동작 제어
- 레지스터
  - 프로세서 내부에 있는 메모리
  - 프로세서가 사용할 데이터 저장
  - 컴퓨터에서 가장 빠른 메모리
  - 종류
    - **용도**에 따른 분류
      - 전용 레지스터, 범용 레지스터 (용도에 따른 분류)
    - 저장하는 **정보의 종류**에 따른 분류
      - 데이터 레지스터, 주소 레지스터, 상태 레지스터
    - 사용자가 **정보 변경 가능 여부**에 따른 분류
      - 사용자 가시 레지스터, 사용자 불가시 레지스터

표 1-1 사용자 가시 레지스터

종류	설명	
데이터 레지스터 DR, Data Register	함수 연산에 필요한 데이터를 저장한다. 값, 문자 등을 저장하므로 산술 연산이나 논리 연산에 사용하며, 연산 결과로 플래그 값을 저장한다.	
주소 레지스터 AR, Address Register	주소나 유효 주소를 계산하는 데 필요한 주소의 일부분을 저장한다. 주소 레지스터에 저장한 값(값 데이터)을 사용하여 산술 연산을 할 수 있다.	
	기준 주소 레지스터	프로그램을 실행할 때 사용하는 기준 주소 값을 저장한다. 기준 주소는 하나의 프로그램이나 일부처럼 서로 관련 있는 정보를 저장하며, 연속된 저장 공간을 지정하는 데 참조할 수 있는 주소이다. 따라서 기준 주소 레지스터는 페이지나 세그먼트처럼 블록화된 정보에 접근하는 데 사용한다.
	인덱스 레지스터	유효 주소를 계산하는 데 사용하는 주소 정보를 저장한다.
	스택 포인터 레지스터	메모리에 프로세서 스택을 구현하는 데 사용한다. 많은 프로세서와 주소 레지스터를 데이터 스택 포인터와 큐 포인터로 사용한다. 보통 반환 주소, 프로세서 상태 정보, 서브루틴의 임시 변수를 저장한다.

표 1-2 사용자 불가시 레지스터

종류	설명
프로그램 카운터 PC, Program Counter	다음에 실행할 명령어의 주소를 보관하는 레지스터이다. 계수기로 되어 있어 실행할 명령어를 메모리에서 읽으면 명령어의 길이만큼 증가하여 다음 명령어를 가리키며, 분기 명령어는 목적 주소로 갱신할 수 있다.
명령어 레지스터 IR, Instruction Register	현재 실행하는 명령어를 보관하는 레지스터이다.
누산기 ACC, ACCumulator	데이터를 일시적으로 저장하는 레지스터이다.
메모리 주소 레지스터 MAR, Memory Address Register	프로세서가 참조하려는 데이터의 주소를 명시하여 메모리에 접근하는 버퍼 레지스터이다.
메모리 버퍼 레지스터 MBR, Memory Buffer Register	프로세서가 메모리에서 읽거나 메모리에 저장할 데이터 자체를 보관하는 버퍼 레지스터이다. 메모리 데이터 레지스터 MDR, Memory Data Register라고도 한다.

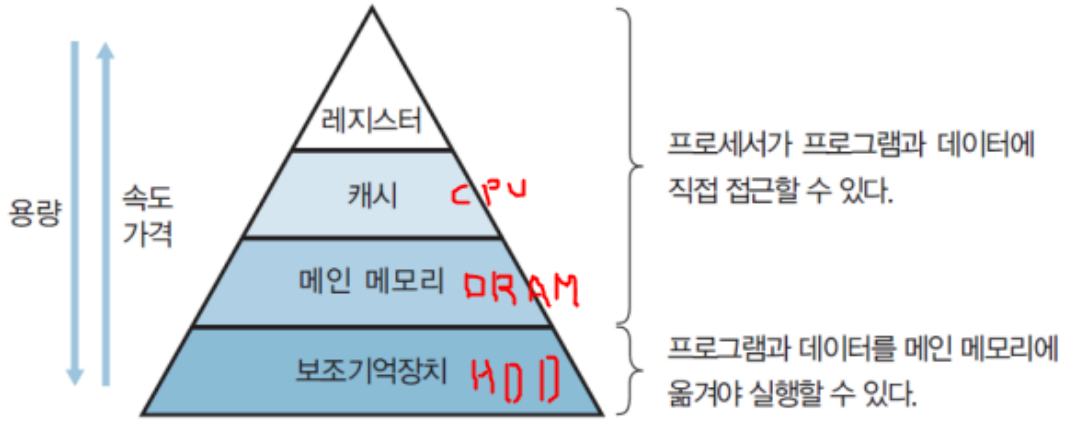
운영체제와 프로세서

- 프로세서에게 처리할 작업 할당 및 관리
  - 프로세스(Process) 생성 및 관리
- 프로그램의 프로세서 사용 제어
  - 프로그램의 프로세서 사용 시간 관리
  - 복수 프로그램간 사용 시간 조율 등

메모리 (Memory)

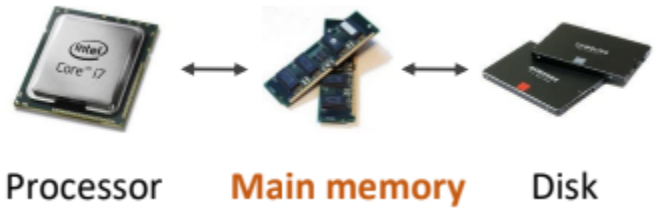
- 데이터를 저장하는 장치 (기억장치)

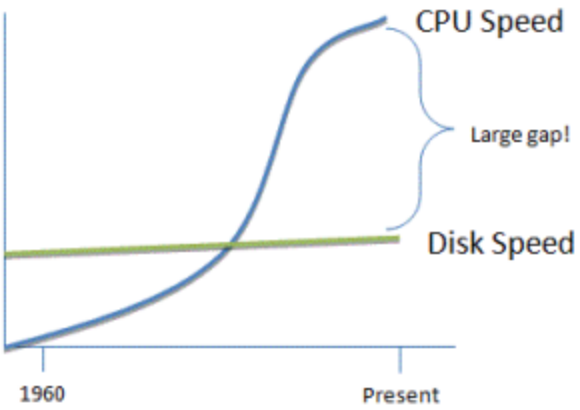
- 프로그램(OS, 사용자SW 등), 사용자 데이터 등



주기억장치 (Main memory)

- 프로세서가 수행할 프로그램과 데이터 저장
- DRAM을 주로 사용
  - 용량이 크고, 가격이 저렴
- `디스크 입출력 병목현상(I/O bottleneck) 해소`
- 왜 프로세서는 디스크에 직접 접근하지 않고 메인 메모리를 거쳐갈까?

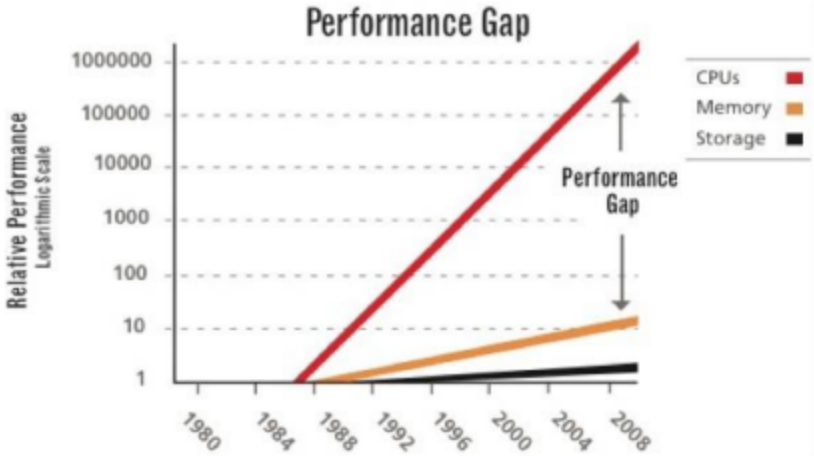




디스크가 CPU의 발전 속도를 못 쫓아감 ⇒ 디스크보다 용량은 작지만 속도는 빠른 메인 메모리 사용

캐시 (Cache)

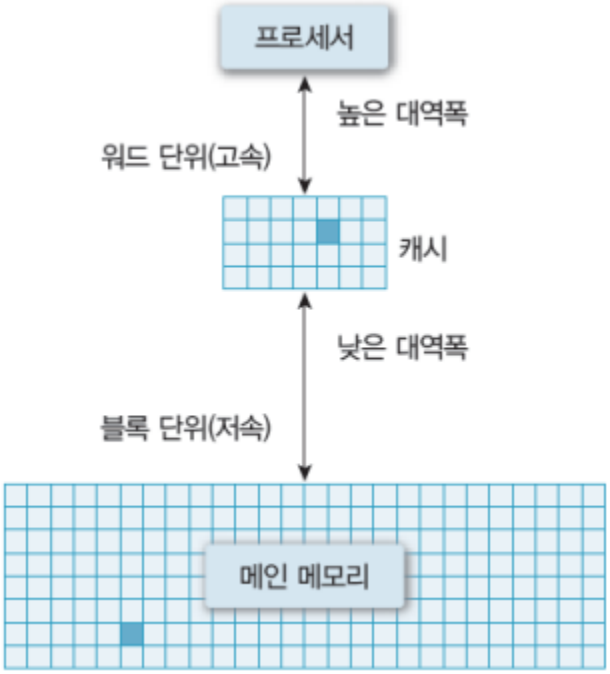
- 프로세서 내부에 있는 메모리 (L1, L2 캐시 등)
  - 속도가 빠르고, 가격이 비쌈
- `메인 메모리의 입출력 병목현상 해소`



메인 메모리를 넣어도 속도 차이가 나서 메인 메모리보다 용량은 작지만 속도는 빠른 캐시 사용

- **캐시의 동작**
  - 일반적으로 HW적으로 관리 됨
  - **캐시 히트 (Cache hit)**
    - 프로세서에서 필요한 데이터가 캐시에 존재
  - **캐시 미스 (Cache miss)**

- 프로세서에서 필요한 데이터가 캐시에 없음
- 캐시가 메인 메모리에 가서 데이터를 캐시로 가져간 후에 프로세서에 전달



- **지역성**
  - 공간적 지역성 (Spatial locality)
    - 참조한 주소와 인접한 주소를 참조하는 특성
    - 예) 순차적 프로그램 수행
  - 시간적 지역성 (Temporal locality)
    - 한 번 참조한 주소를 곧 다시 참조하는 특성
    - 예) For 문 등의 순환 문
  - `지역성은 캐시 적중률(cache hit ratio)과 밀접`
    - 알고리즘 성능 향상 위한 중요한 요소 중 하나

보조기억장치(Auxiliary memory / secondary memory / storage)

- 프로그램과 데이터를 저장
- 프로세서가 직접 접근할 수 없음 (주변장치)

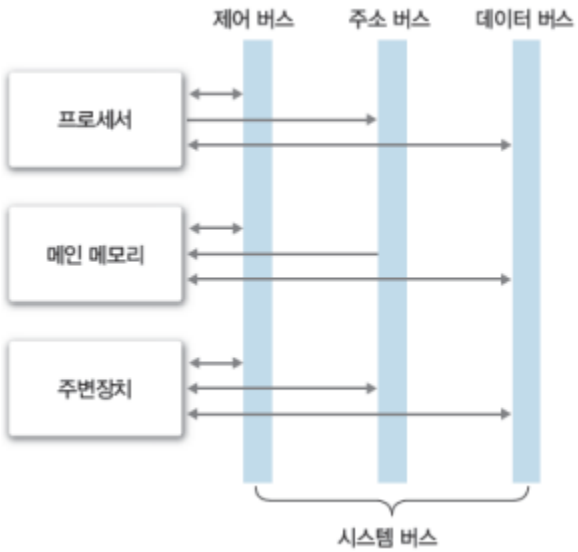
    - 주기억장치를 거쳐서 접근
    - (프로그램/데이터 > 주기억장치) 인 경우는 ?
    - 가상 메모리(Virtual memory
- 용량이 크고, 가격이 저렴

메모리와 운영체제

- 메모리 할당 및 관리
    - 프로그램의 요청에 따른 메모리 할당 및 회수
    - 할당된 메모리 관리
- 가상 메모리 관리
    - 가상메모리 생성 및 관리
    - 논리주소 → 물리주소 변환

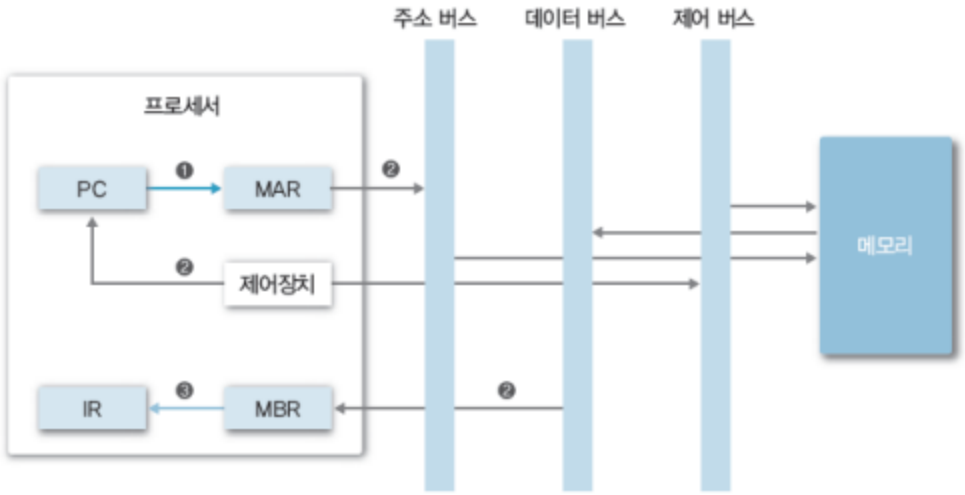
시스템 버스 (System Bus)

- 하드웨어들이 데이터 및 신호를 주고 받는 물리적인 통로



- 데이터 버스 : 데이터 전송
- 주소 버스 : 주소 정보 전송

- 제어 버스 : 제어 신호 전송, 시스템 구성 요소를 제어하는데 사용



시간	레지스터 동작	설명
❶	PC → MAR	PC에 저장된 주소를 프로세서 내부 버스를 이용하여 MAR에 전달한다.
❷	Memory^{MAR} → MBR	MAR에 저장된 주소에 해당하는 메모리 위치에서 명령어를 인출한 후 이 명령어를 MBR에 저장한다. 이때 제어장치는 메모리에 저장된 내용을 읽도록 제어 신호를 발생시킨다.
	PC + 1 → PC	다음 명령어를 인출하려고 PC를 증가시킨다.
❸	MBR → IR	MBR에 저장된 내용을 IR에 전달한다.

- PC : 프로그램 카운터
- MAR : 메모리 주소 레지스터
- MBR : 메모리 버퍼 레지스터
- IR : 명령어 레지스터

- PC → 메모리
  - PC → 메모리 주소 레지스터 → 주소버스 → 메모리 → 제어장치 → 제어신호 발생 → 제어장치 → 메모리 → 데이터버스 → 메모리 버퍼 레지스터 → 명령어 레지스터

주변장치와 운영체제

- 장치 드라이버 관리
  - 주변 장치 사용을 위한 인터페이스 제공
- 인터럽트 (Interrupt) 처리
  - 주변 장치의 요청 처리
- 파일 및 디스크 관리

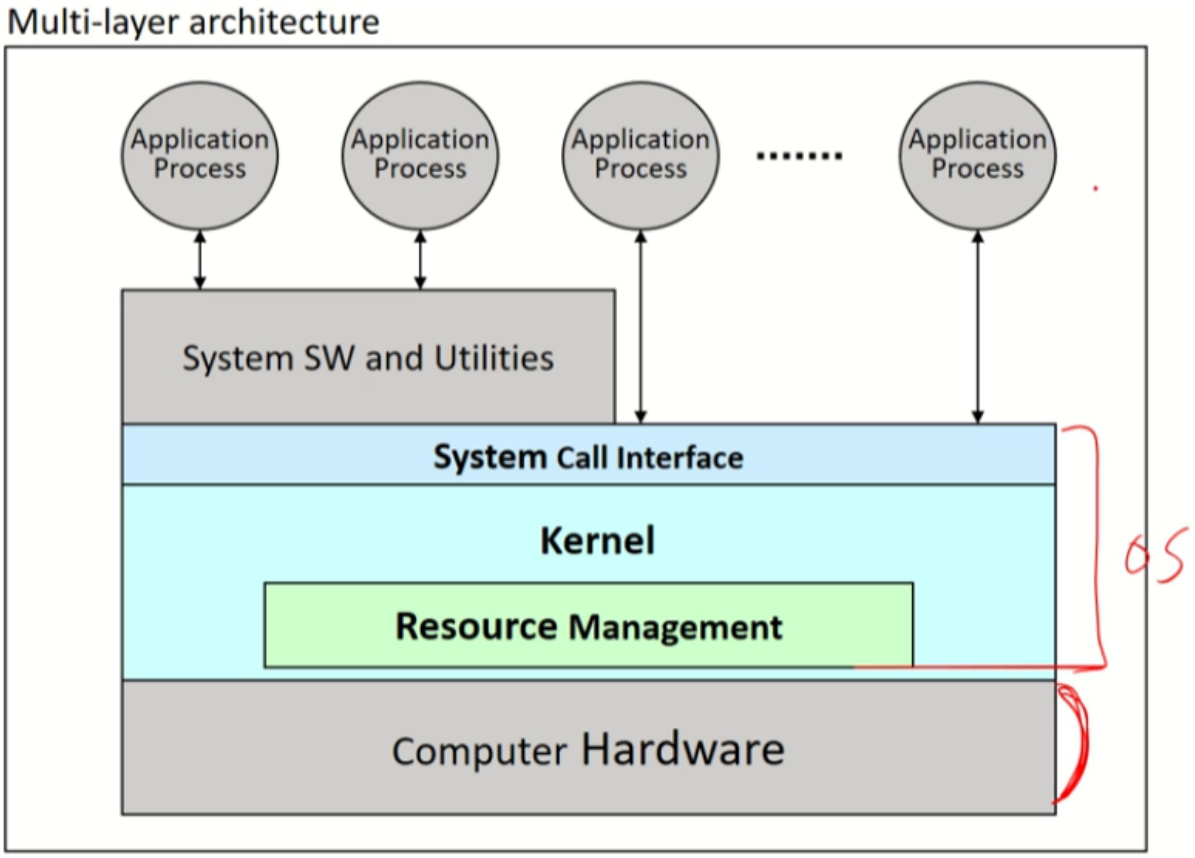
    - 파일 생성 및 삭제, 디스크 공간 관리 등

▼ Lecture 2. OS Overview

운영체제의 역할

- **User Interface (편리성)**
    - CUI (Character user interface)
    - GUI (Graphical User interface)
    - EUCI (End-User Comfortable Interface)
        - ex) MP3
- **Resource management (효율성)**
    - HW resource (processor, memory, I/O devices, Etc.)
    - SW resource (file, application, message, signal, Etc.)
- **Process and Thread management**
    - process : 실행의 주체
- **System management (시스템 보호)**

컴퓨터 시스템의 구성



- system call interface
  - kernel은 사용자가 직접 제어 불가능하므로 필요한 기능이 있으면 os에 요청해야 하는데 그 통로가 system call interface
  - kernel이 제공하는 기능 중에서 사용자가 사용할 수 있는 기능을 모아둔 것

운영체제의 구분

- **동시 사용자 수**
  - 단일 사용자 (Single-user system)
    - 한 명의 사용자만 시스템 사용 가능 ⇒ 모든 시스템 자원 독점
    - 자원관리 및 시스템 보호 방식이 간단 함
    - 개인용 장비(PC, mobile) 등에 사용
    - Windows 7/10, android, MS-DOS 등
  - 다중 사용자 (Multi-user system)

        - 동시에 여러 사용자들이 시스템 사용 ⇒ 각종 시스템 자원에 대한 소유권 관리 필요
        - 기본적으로 Multi-tasking 기능 필요
        - OS의 기능 및 구조가 복잡
        - 서버, 클러스터(cluster) 장비 등에 사용
        - Unix, Linux, Windows server 등
- **동시 실행 프로세스 수**
    - 단일작업 (Single-tasking system)
        - 시스템 내에 하나의 작업(프로세스)만 존재
        - 하나의 프로그램 실행을 마친 뒤에 다른 프로그램의 실행
        - 운영체제의 구조가 간단
        - 예) MS-DOS
    - 다중작업 (Multi-tasking system)
        - 동시에 여러 작업(프로세스)의 수행 가능
        - 작업들 사이의 동시 수행, 동기화 등을 관리해야 함
        - 운영체제의 기능 및 구조가 복잡
        - 예) Unix/Linux, Windows 등
- **작업 수행 방식 (사용자가 느끼는 사용 환경)**
    - Batch processing system (일괄처리 시스템)
    - Time-sharing system (시분할 시스템)
    - Distributed processing system (분산처리 시스템)
    - Real-time system (실시간 시스템)

운영체제 발전 역사

순차 처리 (No OS, ~1940s)

- **운영체제 개념 존재하지 않음**

    - 사용자가 기계어로 직접 프로그램 작성
    - 컴퓨터에 필요한 모든 작업 프로그램에 포함
        - 프로세서에는 명령어 저장 방법, 계산 대상, 결과 저장 위치와 방법, 출력 시점, 위치 등
- 실행하는 작업 별 순차 처리
    - **각각의 작업에 대한 준비 시간이 소요**

Batch Systems (1950s~1960s)

- 순차 처리 방식의 단점인 준비 시간을 줄이기 위한 방법
- 모든 시스템을 중앙(전자계산소 등)에서 관리 및 운영
- 사용자의 요청 작업(천공카드 등)을 일정 시간 모아 두었다가 한번에 처리 ⇒ 일괄 처리
- 시스템 지향적 (System-oriented)
- 장점
    - 많은 사용자가 시스템 자원 공유
    - 처리 효율(throughput) 향상
- 단점
    - 생산성(productivity) 저하
        - 같은 유형의 작업들이 모이기를 기다려야 함
    - 긴 응답시간 (turnaround time)
        - 약 6시간 (작업 제출에서 결과 출력까지의 시간)

Time Sharing Systems (1960s~1970s)

- 여러 사용자가 자원을 동시에 사용
    - OS가 파일 시스템 및 가상 메모리 관리
- 사용자 지향적 (User-oriented)
    - 대화형(conversational, interactive) 시스템 ⇒ 반응이 있음
    - 단말기(CRT terminal) 사용

- 장점
  - 응답시간(response time) 단축 (약 5초)
  - 생산성(productivity) 향상
    - 프로세서 유휴 시간 감소
- 단점
  - 통신 비용 증가
    - 통신선 비용, 보안 문제 등
  - 개인 사용자 체감 속도 저하
    - 동시 사용자 수 ↑ ▶️ 시스템 부하 ↑ ▶️ 느려짐 (개인관점)

Personal Computing

- 개인이 시스템 전체 독점
- CPU 활용률(utilization)이 고려의 대상이 아님 ⇒ CPU를 최대한 많이 쓸 필요가 없어짐
- OS가 상대적으로 단순함
  - 하지만, 다양한 사용자 지원 기능 지원
- 응답시간이 빠르지만 성능이 낮음

Parallel Processing System

- 단일 시스템 내에서 둘 이상의 프로세서 사용
  - 동시에 둘 이상의 프로세스 지원
- 메모리 등의 자원 공유 (Tightly-coupled system)
  ⇒ CPU는 여러개지만 기타 지원들은 공유해서 사용
- 사용 목적
  - 성능 향상
  - 신뢰성 향상 (하나가 고장 정상 동작 가능)
- 프로세서간 관계 및 역할 관리 필요

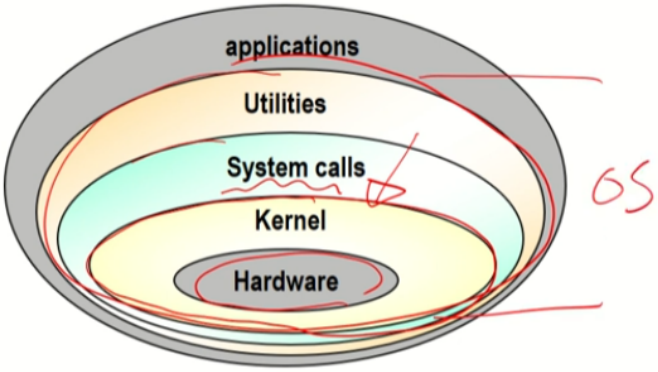
Distributed Processing Systems

- 네트워크를 기반으로 구축된 병렬 처리 시스템 (Loosely-coupled system)
  - 물리적인 분산, 통신망 이용한 상호 연결
  - 각각 운영체제 탑재한 다수의 범용 시스템으로 구성
  - 사용자는 분산 운영체제를 통해 하나의 프로그램, 자원처럼 사용 가능 (은폐성, transparency)
  - 각 구성 요소들 간의 독립성 유지, 공동 작업 가능
  - Cluster system, client-server system, P2P 등
- 장점
  - 자원 공유를 통한 높은 성능, 고신뢰성, 높은 확장성
- 단점
  - 구축 및 관리가 어려움

Real-time Systems

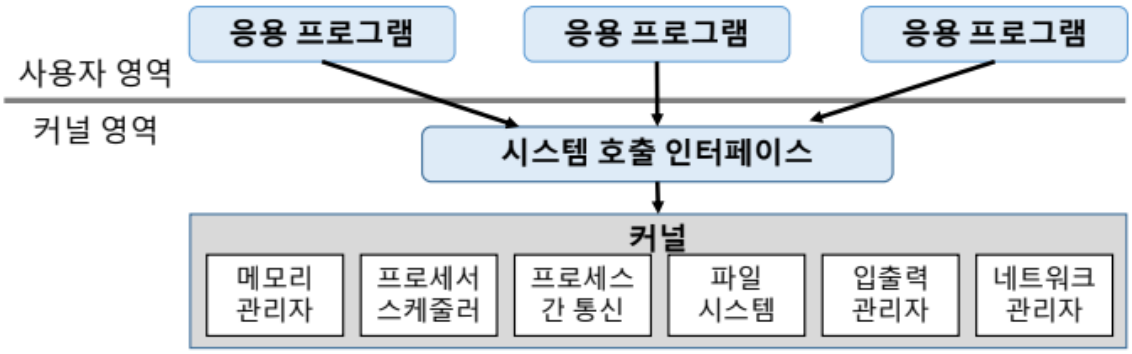
- 작업 처리에 **제한 시간(deadline)**을 갖는 시스템
  - 제한 시간 내에 서비스를 제공하는것이 자원 활용 효율보다 중요
- 작업(task)의 종류
  - Hard real-time task
    - 시간 제약을 지키지 못하는 경우 시스템에 치명적 영향
    - 예) 발전소 제어, 무기 제어 등
  - Soft real-time task
    - 동영상 재생 등
  - Non real-time task

운영체제 구조



- 커널 (Kernel)
  - OS의 핵심 부분 (메모리 상주), 가장 빈번하게 사용되는 기능들 담당
    - 시스템 관리(processor, memory, Etc) 등
  - 동의어: 핵 (neucleus), 관리자 (supervisor) 프로그램, 상주 프로그램 (resident program), 제어 프로그램 (control program) 등
- 유틸리티 (Utility)
  - 메모리 비상주 프로그램

단일 구조



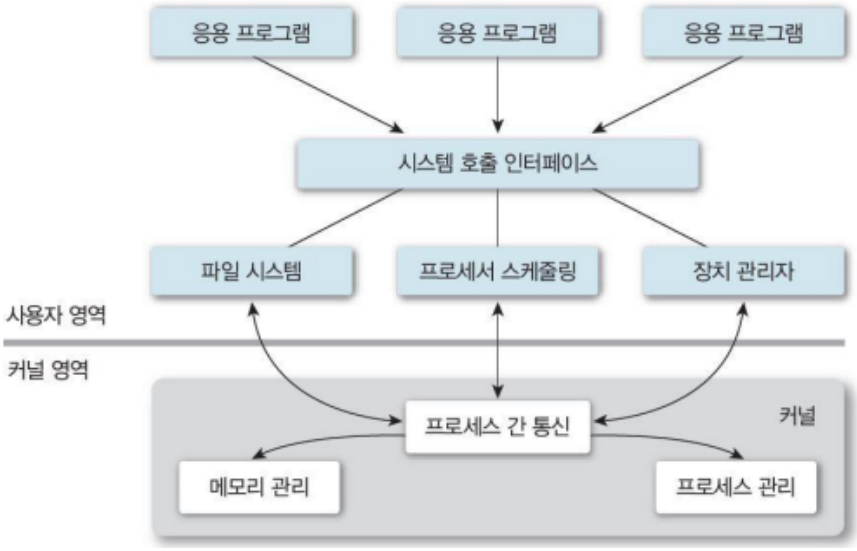
- 장점
  - 커널 내 모듈간 직접 통신
  - 효율적 자원 관리 및 사용
- 단점

    - 커널의 거대화
    - 오류 및 버그, 추가 기능 구현 등 유지보수가 어려움
    - 동일 메모리에 모든 기능이 있어, 한 모듈의 문제가 전체 시스템에 영향 (예, 악성 코드 등)

계층 구조

- 장점
    - 모듈화
        - 계층간 검증 및 수정 용의
    - 설계 및 구현의 단순화
- 단점
    - 단일구조 대비 성능 저하
    - 원하는 기능 수행을 위해 여러 계층을 거쳐야 함

마이크로 커널 구조



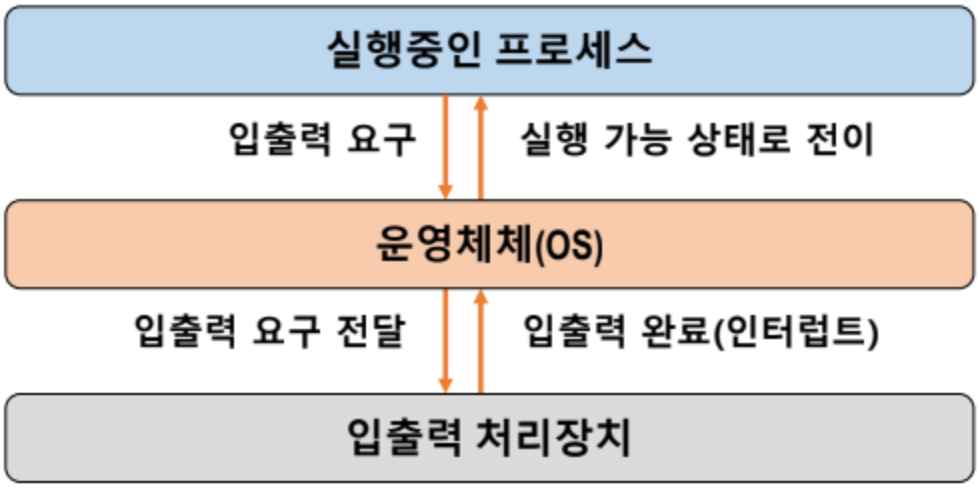
- 커널의 크기 최소화
  - 필수 기능만 포함
  - 기타 기능은 사용자 영역에서 수행

운영체제 기능

- 프로세스(Process) 관리
  - 프로세스 (Process)
    - 커널에 등록된 실행 단위 (실행 중인 프로그램)
    - 사용자 요청/프로그램의 수행 주체(entity)
  - OS의 프로세스 관리 기능
    - 생성/삭제, 상태관리
    - 자원 할당
    - 프로세스 간 통신 및 동기화(synchronization)
    - 교착상태(deadlock) 해결 (여러 개의 프로세스가 동시에 하나의 자원을 쓰려고 할 때)
  - 프로세스 정보 관리
    - PCB (Process Control Bloc)

- 프로세서(Processor) 관리
  - 중앙 처리 장치(CPU)
    - 프로그램을 실행하는 핵심 자원
  - 프로세스 스케줄링(Scheduling)
    - 시스템 내의 프로세스 처리 순서 결정
  - 프로세서 할당 관리
    - 프로세스들에 대한 프로세서 할당
    - 한 번에 하나의 프로세스만 사용 가능
- 메모리(Memory) 관리
  - 주기억장치
    - 작업을 위한 프로그램 및 데이터를 올려 놓는 공간
  - Multi-user, Multi-tasking 시스템
    - 프로세스에 대한 메모리 할당 및 회수
    - 메모리 여유 공간 관리
    - 각 프로세스의 할당 메모리 영역 접근 보호
  - 메모리 할당 방법(scheme)
    - 전체 적재
      - **장점**: 구현이 간단 / **단점**: 제한적 공간
    - 일부 적재 (virtual memory concept)
      - 프로그램 및 데이터의 일부만 적재
      - **장점**: 메모리의 효율적 활용 / **단점**: 보조기억 장치 접근 필요
- 파일(File) 관리
  - 파일: 논리적 데이터 저장 단위
  - 사용자 및 시스템의 파일 관리
  - 디렉토리(directory) 구조 지원
  - 파일 관리 기능

        - 파일 및 디렉토리 생성/삭제
        - 파일 접근 및 조작
        - 파일을 물리적 저장 공간으로 사상(mapping)
        - 백업 등
- 입출력(I/O) 관리
    - 입출력(I/O) 과정
    - OS를 반드시 거쳐야 함

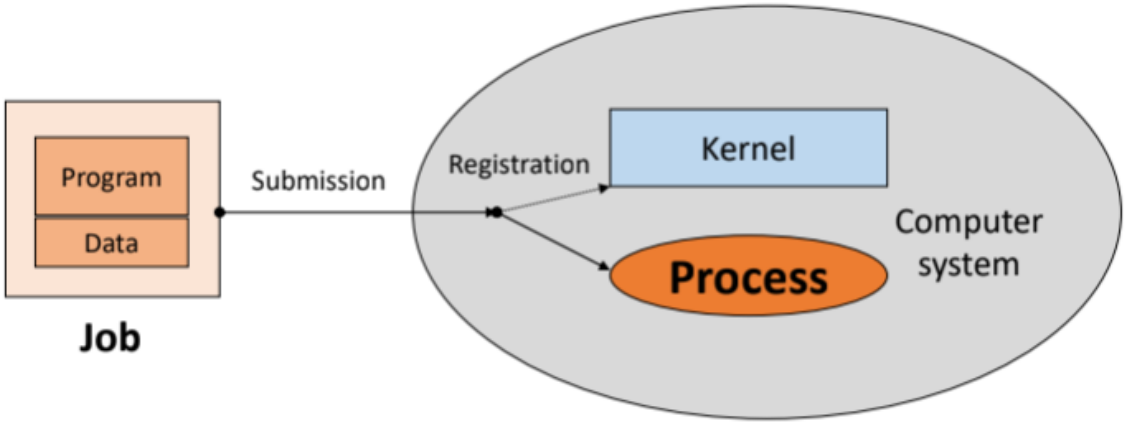


- 보조 기억 장치 및 기타 주변장치 관리 등
    - Disk
    - Networking
    - Security and Protection system
    - Command interpreter system
    - System call interface
        - 응용 프로그램과 OS 사이의 인터페이스
        - OS가 응용프로그램에 제공하는 서비스

▼ Lecture 3. Process Management

Job vs Process

- 작업 (Job) / 프로그램 (Program)
  - 실행 할 프로그램 + 데이터
  - 컴퓨터 시스템에 실행 요청 전의 상태
- 프로세스 (Process)
  - 실행을 위해 시스템(커널)에 등록된 작업
  - 시스템 성능 향상을 위해 커널에 의해 관리 됨



프로세스의 정의

- **실행중인 프로그램**
  - 커널에 등록되고 커널의 관리하에 있는 작업
  - 각종 자원들을 요청하고 할당 받을 수 있는 개체
  - 프로세스 관리 블록(PCB)을 할당 받은 개체
  - 능동적인 개체(active entity)
    - 실행 중에 각종 자원을 요구, 할당, 반납하며 진행
- Process Control Block (PCB)
  - 커널 공간 (kernel space) 내에 존재

- 각 프로세스들에 대한 정보를 관리

프로세스 종류

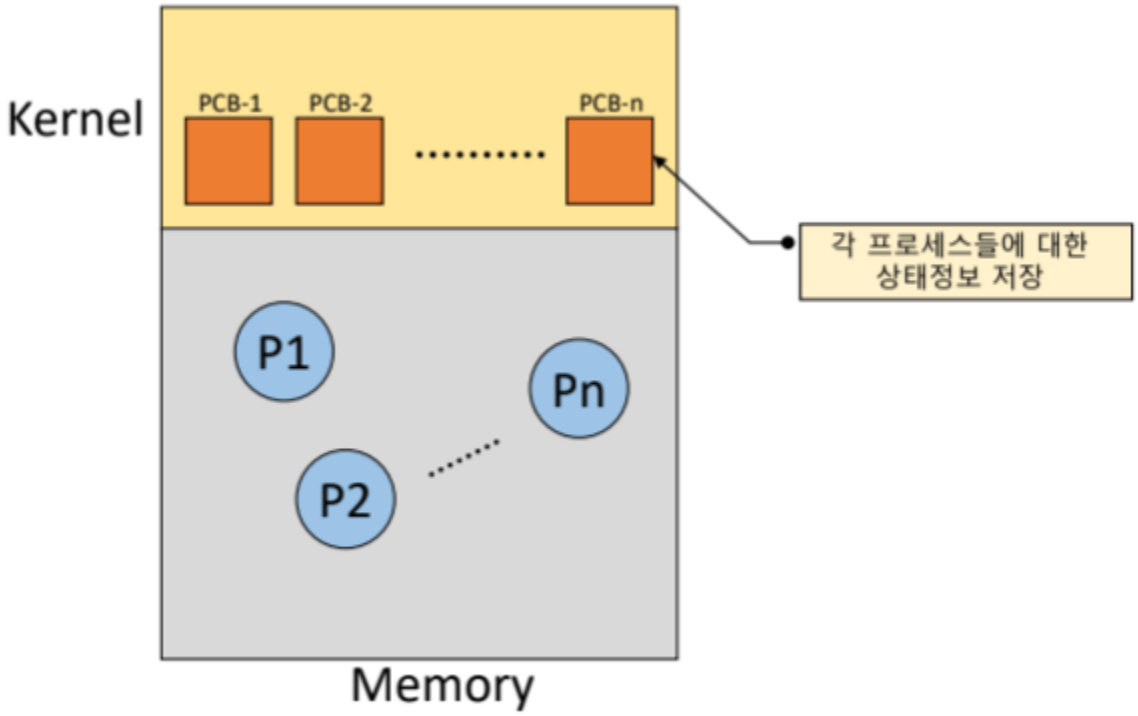
- 역할에 따라
  - 시스템 프로세스 : 시스템에서 사용
  - 사용자 프로세스 : 사용자가 사용
- 병행 수행 방법에 따라
  - 독립 프로세스 : 혼자 작업 처리
  - 협력 프로세스 : 여러 개의 프로세스가 하나의 작업 처리

자원(Resource)의 개념

- 커널의 관리 하에 프로세스에게 할당/반납 되는 수동적 개체(passive entity)
- 자원의 분류
  - H/W resources
  - S/W resources

Process Control Block (PCB)

- OS가 프로세스를 컨트롤 하기 위해 필요한 정보를 모아 놓은 것
- 프로세스 생성 시, 생성 됨



kernel이 관리하는 영역에 저장됨

- PCB 정보는 OS 별로 서로 다름
- PCB 참조 및 갱신 속도는 OS의 성능을 결정 짓는 중요한 요소 중 하나

PCB가 관리하는 정보

- PID : Process Identification Number
  - 프로세스 고유 식별 번호
- 스케줄링 정보
  - 프로세스 우선순위 등과 같은 스케줄링 관련 정보들
- 프로세스 상태
  - 자원 할당, 요청 정보 등
- 메모리 관리 정보
  - Page table, segment table 등
- 입출력 상태 정보
  - 할당 받은 입출력 장치, 파일 등에 대한 정보 등

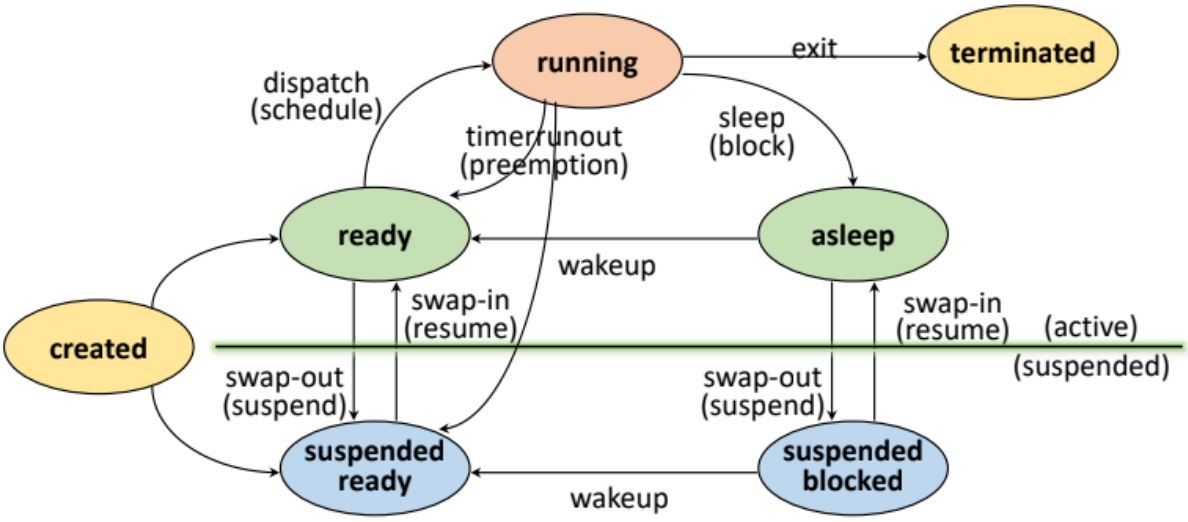
- 문맥 저장 영역 (context save area)
  - 프로세스의 레지스터 상태를 저장하는 공간 등
- 계정 정보 - 다중 사용자 시스템일 경우
  - 자원 사용 시간 등을 관리

프로세스의 상태 (Process States)

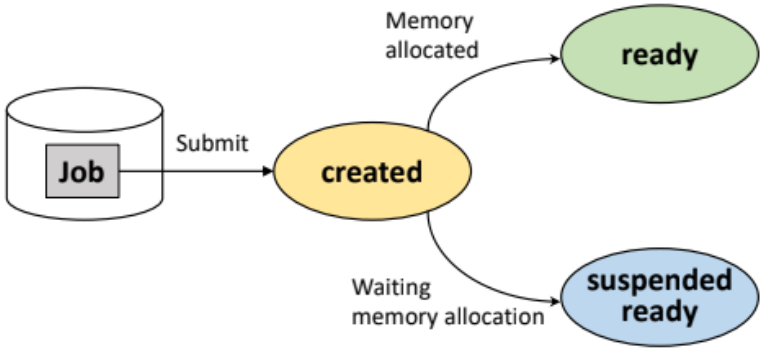
- 프로세스 – 자원 간의 상호작용에 의해 결정
- 프로세스 상태 및 특성

상태		자원 할당 상태	
Active (swapped-in)	Running	프로세서 O	메모리 O
	Ready	프로세서 X, 기타 자원 O	
	Blocked, asleep	프로세서 X, 기타 자원 X	
Suspended (Swapped-out)	Suspended ready	프로세서 X	메모리 X
	Suspended block	프로세서 X, 기타 자원 X	

Process State Transition Diagram

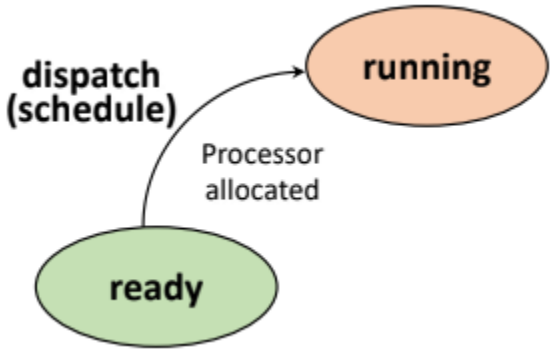


Created State



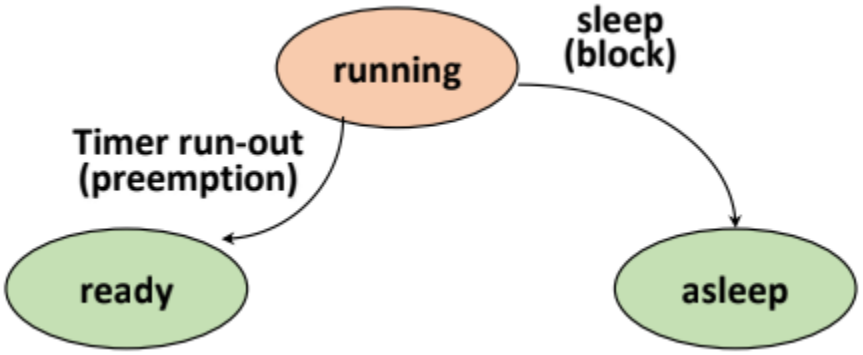
- 작업(Job)이 커널에 등록된 상태
- PCB 할당 및 프로세스 생성
- 커널
  - 쓸 수 있는 메모리 공간이 **있다** ⇒ Ready
  - 쓸 수 있는 메모리 공간이 **없다** ⇒ Suspended ready

Ready State



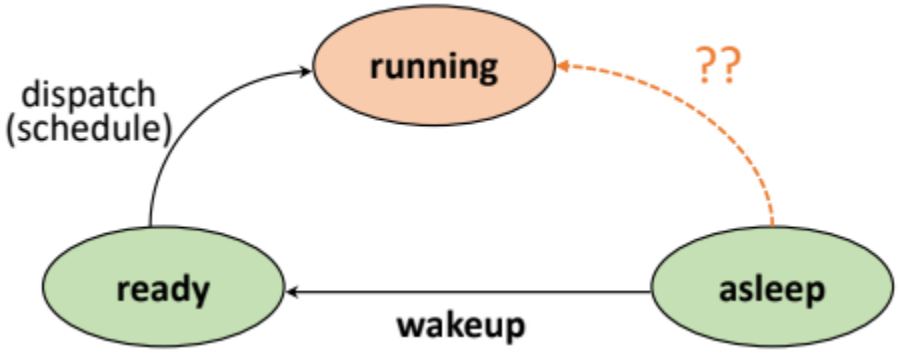
- 프로세서(cpu)를 기다리는 상태
  - 즉시 실행 가능 상태
- Dispatch (or Schedule) ← cpu를 할당 받는 것
  • Ready state ⇒ running state

Running State



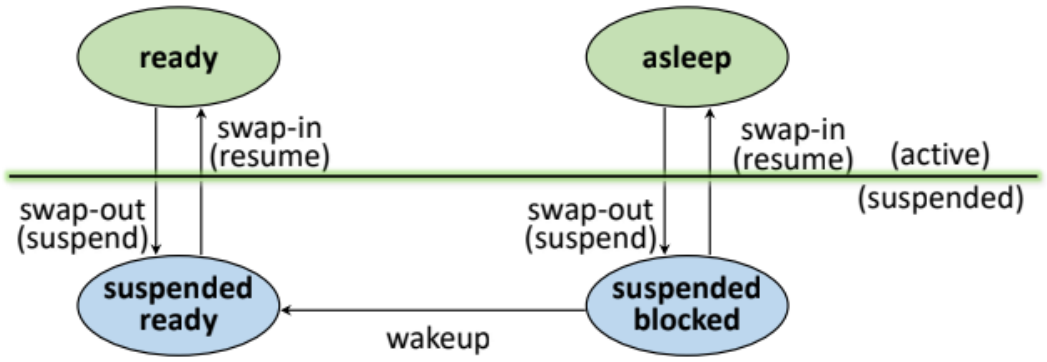
- 프로세서와 필요한 자원을 모두 할당 받은 상태 (실행)
- Preemption ← 프로세서를 뺏기는 것
  - Running state ⇒ ready states
  - 프로세서 스케줄링 (e.g, time-out, priority changes)
- Block/sleep ← I/O를 기다리는 상태
  - Running state ⇒ asleep state
  - I/O 등 자원 할당 요청

Blocked/Asleep State



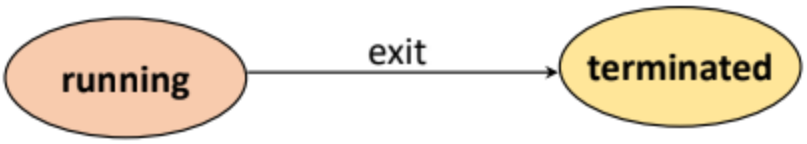
- 프로세서 외에 다른 자원을 기다리는 상태
  - 자원 할당은 System call에 의해 이루어 짐
- Wake-up
  - Asleep state ⇒ ready state

Suspended State



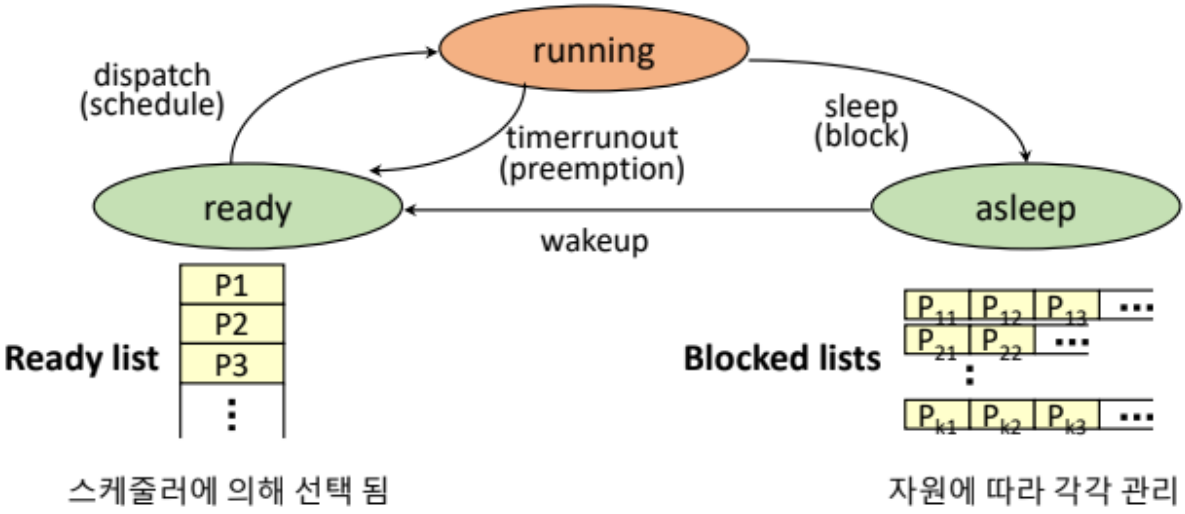
- 메모리를 할당 받지 못한(빼앗긴) 상태
  - Memory image(메모리 상태)를 swap device에 보관
    - Swap device: 프로그램 정보 저장을 위한 특별한 파일 시스템
  - 커널 또는 사용자에 의해 발생
- Swap-out(suspended) : 메모리를 뺏기면서 swap device에 메모리 이미지를 저장
- Swap-in(resume) : 다시 메모리를 할당 받아서 복구하는 것

Terminated/Zombie State



- 프로세스 수행이 끝난 상태
- 모든 자원 반납, 커널 내에 일부 PCB 정보만 남아 있는 상태
  - 이후 프로세스 관리를 위해 정보 수집

프로세스 관리를 위한 자료구조

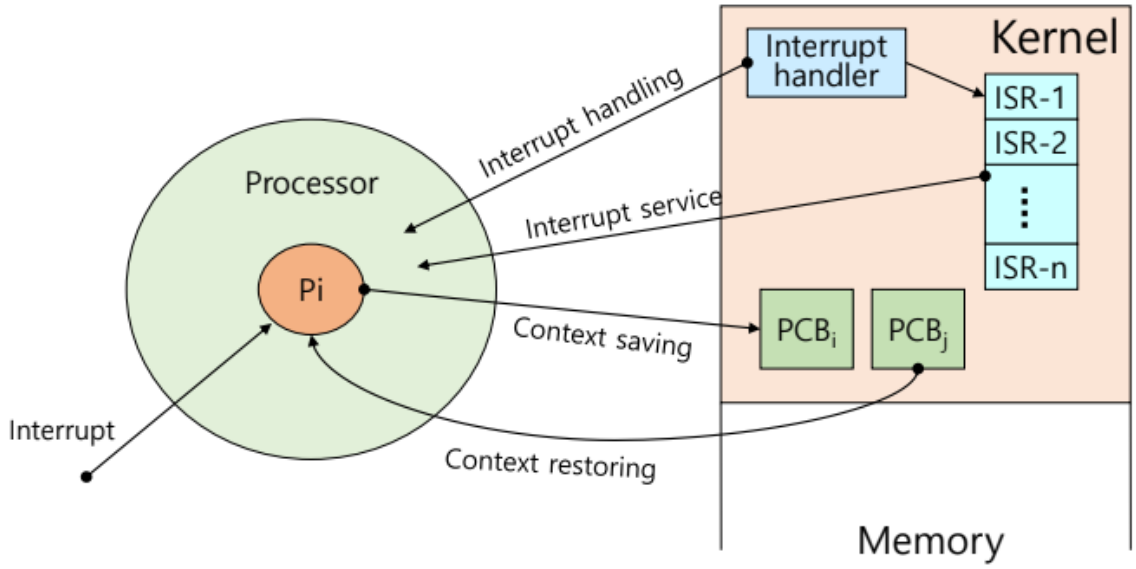


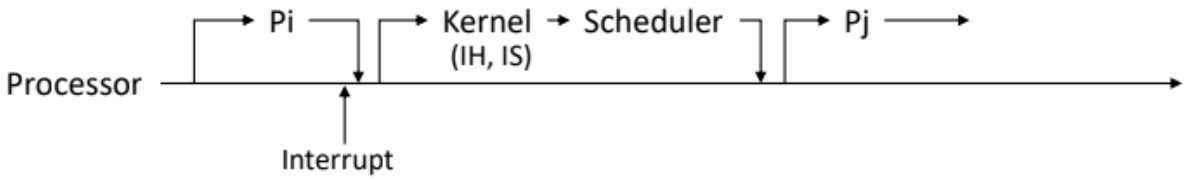
- Ready list
  - Ready Queue
- blocked list
  - I/O Queue, Device Queue

인터럽트(Interrupt)

- 예상치 못한, 외부에서 발생한 이벤트
- 인터럽트의 종류
  - I/O interrupt, Clock interrupt, Console interrupt, Program check interrupt, Machine check interrupt, Inter-process interrupt, System call interrupt

인터럽트 처리 과정





- 프로세스 Pi가 실행 중에 인터럽트 발생됨
- context saving 발생 : 흐름 저장
- 인터럽트 핸들링 (어디서, 왜 발생했는지 / 해결하기 위해 어떤 서비스를 할 것인지)
- 인터럽트 서비스 (인터럽트 서비스 루틴 호출)
- 서비스가 프로세서 안에 들어가서 수행됨
- 서비스가 종료되면 비어있는 프로세서에 ready 상태에 있던 것 중 하나를 넣음
  ⇒ 꼭 Pi가 들어오는 것은 아니다.

Context Switching (문맥 교환)

- Context
  - 프로세스와 관련된 정보들의 집합
    - CPU register context => in CPU
    - Code & data, Stack, PCB => in memory

- Context saving
  - 현재 프로세스의 Register context를 저장하는 작업
- Context restoring
  - Register context를 프로세스로 복구하는 작업
- Context switching
  - 실행 중인 프로세스의 context를 저장하고, 앞으로 실행 할 프로세스의 context를 복구 하는 일 ← 커널의 개입으로 이루어짐

Context Switch Overhead

- Context switching에 소요되는 비용
  - OS마다 다름, OS의 성능에 큰 영향을 줌
- 불필요한 Context switching을 줄이는 것이 중요