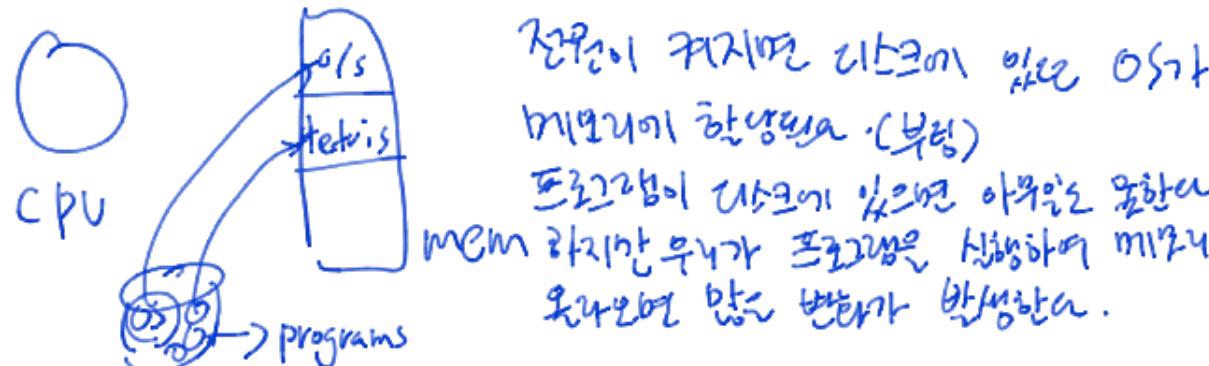


프로세스 관리



Process Management

→ 컴퓨터에서 파일, 중요한 자원인 CPU를 프로세서들에게 어떻게 나눠줄까?



프로세스

- **프로그램 vs 프로세스 (program vs process)**

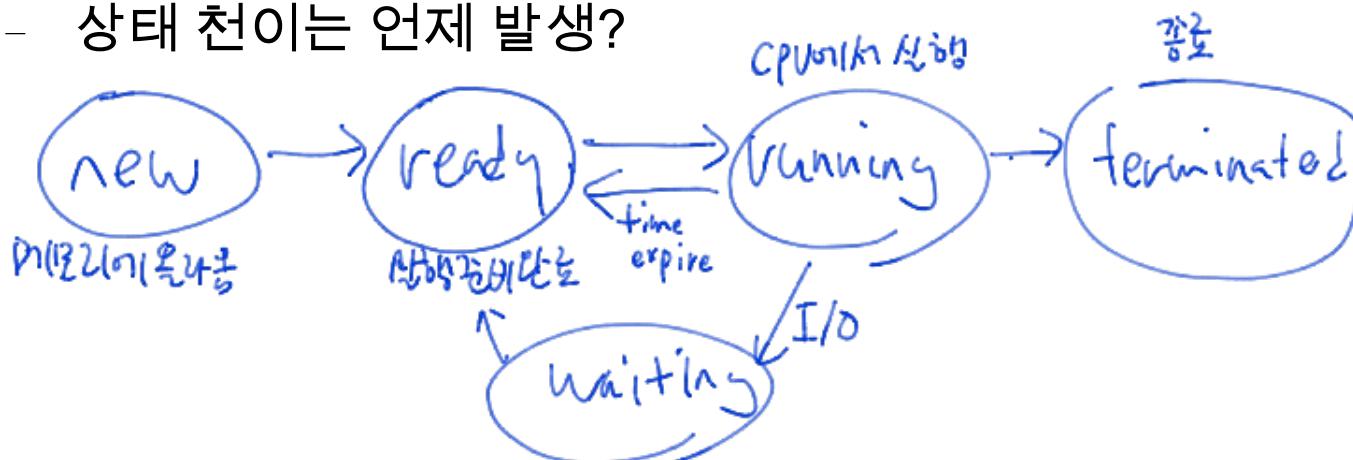
- process, task, job ...
- 실행중인 프로그램
- program in execution: text + data + stack, pc, sp, registers, ...
- 무덤 속 프로그램, 살아 움직이는 프로세스

program counter

↳ stack pointer

- **프로세스 상태**

- new, ready, running, waiting, terminated (그림)
- 프로세스 상태 천이도 (process state transition diagram)
- 상태 천이는 언제 발생?



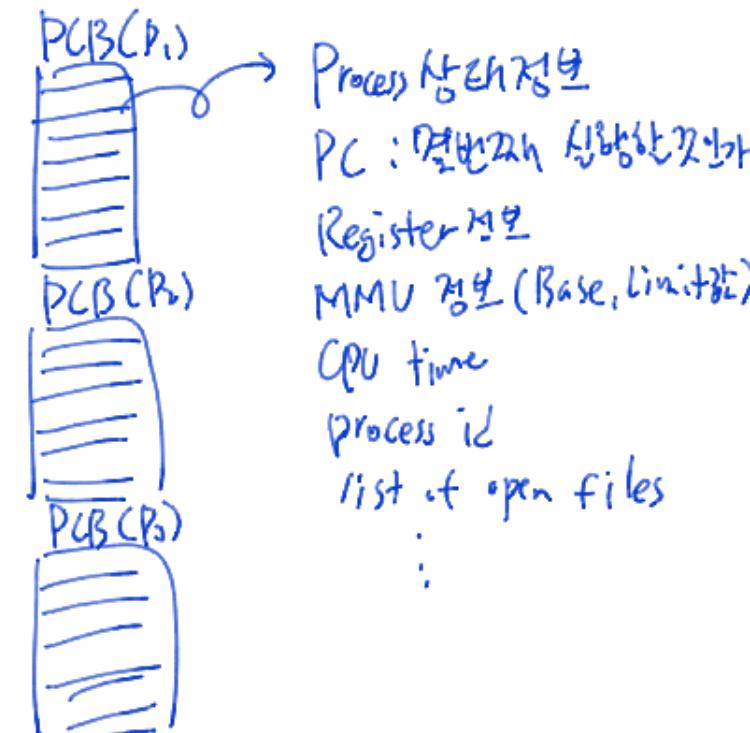
PCB

- Process Control Block (PCB); 프로세스 제어 블록, 프로세스당 하나씩 할당된다.

- Task Control Block (TCB)
- 프로세스에 대한 모든 정보가 들어있다.
- process state (running, ready, waiting, ...), PC, registers, MMU info (base, limit), CPU time, process id, list of open files,
- ...

- 사람과 비유? process = 사람
PCB = 주민등록 정보, 신상정보

- PCB는 OS의 프로세스 management 내부에 위치



메인 메모리는 하드디스크에 비해 상대적으로 작다.

따라서 디스크에서 데이터를 읽어가기 위해서는 Job Queue에서 대기해야 한다.

● Job Queue

- Job scheduler
- Long-term scheduler

● Ready Queue

- CPU scheduler
- Short-term scheduler

● Device Queue

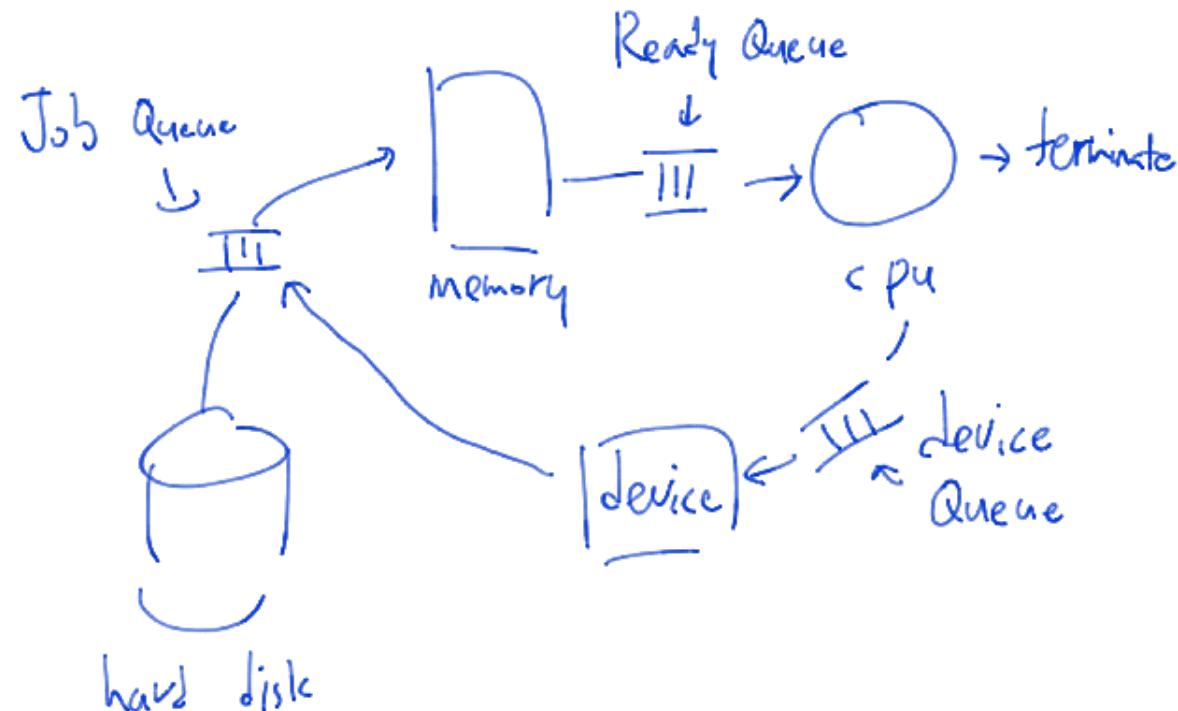
- Device scheduler

Scheduler

Queues

(os in)

process management 포함



Multiprogramming

메인 메모리에 프로세스가 몇개 동시에 있는가;

- Degree of multiprogramming

각 I/O 사용

각 CPU 사용

- i/o-bound vs CPU-bound process

- Medium-term scheduler

- Swapping

- Context switching (문맥전환)

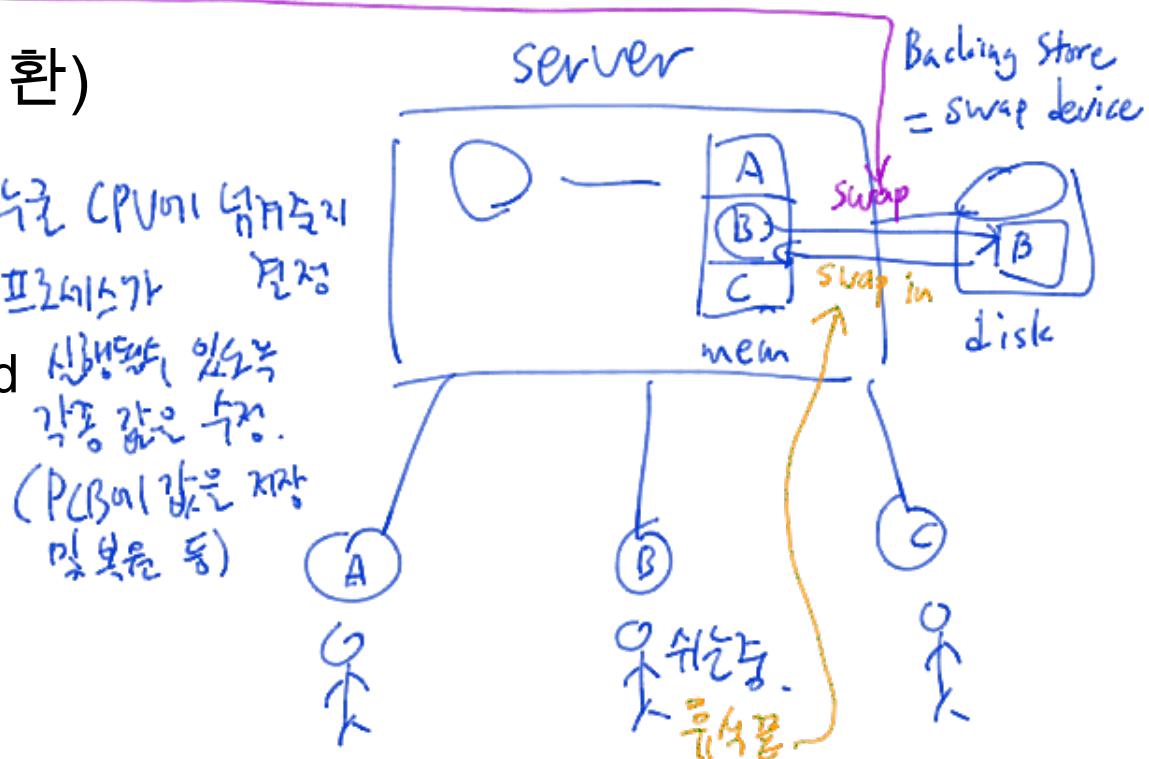
- Scheduler: Ready Queue에서 누군 CPU에 넘겨주기

- Dispatcher: scheduler가 선택한 프로세스가 결정

- Context switching overhead
 - 시간복잡도, 인도록
 - 각 프로세스 수장
 - (PCB와 같은 자료
 - 및 복사를 등)

; 메인 메모리에 여는 개의 프로세스를
 - 동시에 실행하는 것.

→ 스케줄러는 이 두 가지 종류의 프로세스를
 - 적절히 스케줄링하는 것이 중요하다.



CPU 스케줄링

CPU = 의사

Eg) 병원의 응급환자.

병원의 일반환자.

현재 실행중인 프로세스가
끝나거나 I/O는 만나지 않아도
잘아내고 CPU는 점유할 수 있는가?

현재 실행중인 프로세스가

끝나거나 I/O는 만나지 않으면

종료 ; context switch가 발생할 때
종료되면 다음에는 프로세스를 실행하지 않음
context switch.

CPU Scheduling

- Preemptive vs Non-preemptive

- 선점(先占) : 비선점(非先占)

- Scheduling criteria ; 다양한 스케줄링 알고리즘의 좋은 나쁨을 구별하는 척도.

- CPU Utilization (CPU 이용률)
 - Throughput (처리율) ; 시간당 몇개의 작업을 처리하는가?
 - Turnaround time (반환시간) ; 어떤 작업이 끝난 때까지의 시간.
 - Waiting time (대기시간) ; Ready Queue에서 얼마나 기다렸는가.
 - Response time (응답시간) ; 첫 응답이 나온 때까지 시간.
 - ...

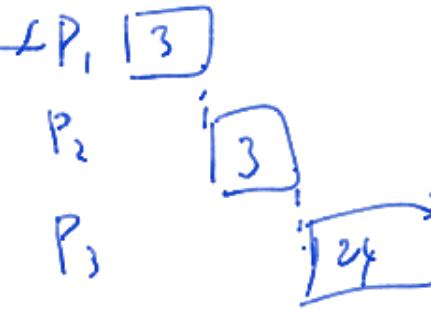
CPU Scheduling Algorithms

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
 - Shortest-Remaining-Time-First
- Priority
- Round-Robin (RR)
- Multilevel Queue
- Multilevel Feedback Queue

First-Come, First-Served

- Simple & Fair 간단하고 공정한 차-1 원리 성능을 의미하는 걸 아는다.
- Example: Find Average Waiting Time
 - $AWT = (0+24+27)/3 = 17 \text{ msec}$ cf. 3 msec! P₁은 뒤쪽으로 옮기면 평균 3msec를 감소가능.
- Gantt Chart

Process	Burst Time (msec)
P_1	24
P_2	3
P_3	3



- Convoy Effect (호위효과) : P_1 과 같이 시간이 오래걸리는 프로세스가 있으면 P_2, P_3 가 호위하는 듯하다. 즉 뒤에 있는 프로세스가 더 늦어지게 된다. 이런 현상을 호위효과라 함. FCFS의 단점
- Nonpreemptive scheduling

Shortest-Job-First (1)

설명: $P_4 \rightarrow P_1 \rightarrow P_3 \rightarrow P_2$: 실행시간이 짧은 것 순으로 우선적으로 실행.

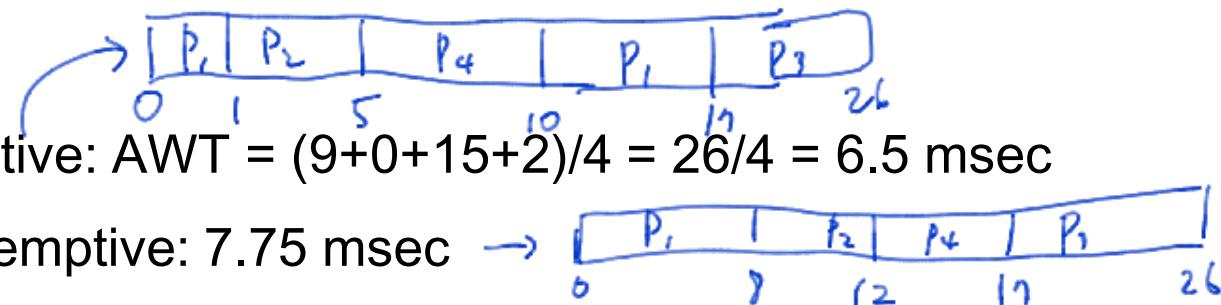
- Example: AWT = $(3+16+9+0)/4 = 7$ msec
 - cf. 10.25 msec (FCFS)
- Provably *optimal* : 증명의 최적화된 방법이다.
- *Not realistic*; prediction may be needed 하지만 비현실적 실례로 CPU Time이 얼마나 걸리지는 알 수 있다.
그래서 예상은 해석 서버하는데 예상하는데
오류율이 높을 수 있다.

Process	Burst Time (msec)
P_1	6
P_2	8
P_3	7
P_4	3

Shortest-Job-First (2)

- Preemptive or Nonpreemptive: *누가21 양방향 모두 가능*
cf. Shortest-Remaining-Time-First (최소잔여시간 우선)

- Example



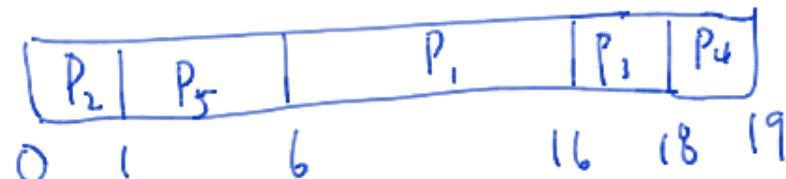
- Preemptive: AWT = (9+0+15+2)/4 = 26/4 = 6.5 msec
- Nonpreemptive: 7.75 msec → Gantt chart showing processes P₁, P₂, P₄, and P₃ running sequentially without preemption.

Process	Arrival Time	Burst Time (msec)
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

Priority Scheduling (1)

- Priority (우선순위): typically an integer number
 - Low number represents high priority in general (Unix/Linux)
- Example
 - AWT = $\frac{6+0+16+18+1}{5} = 8.2$ msec

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



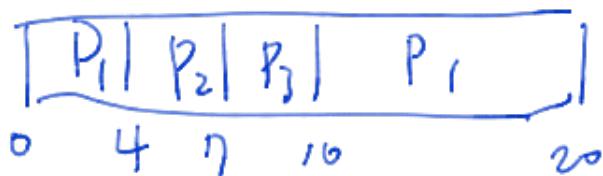
Priority Scheduling (2)

- Priority
 - Internal: time limit, memory requirement, i/o to CPU burst, ...
 - External: amount of funds being paid, political factors, ...
- Preemptive or Nonpreemptive (동의 가능성)
- Problem
 - Indefinite blocking: *starvation* (기아) ; 우선순위가 있고서 아예 더 가다려운 차례가 오지 않는 것.
 - Solution: aging ; 시간이 지나면 차례가 있는 것.
aging? 우선순위는 올바른가.

Round-Robin (1)

- Time-sharing system (시분할/시공유 시스템)
- Time *quantum* 시간양자 = time slice (10 ~ 100msec) ↗ 1주: 10msec
- Preemptive scheduling (비정정형을 없음)
- Example
 - Time Quantum = 4msec
 - AWT = $17/3 = 5.66 \text{ msec}$ $= \frac{6+4+7}{3}$

Process	Burst Time (msec)
P_1	24
P_2	3
P_3	3



Round-Robin (2)

- Performance depends on the size of the time quantum

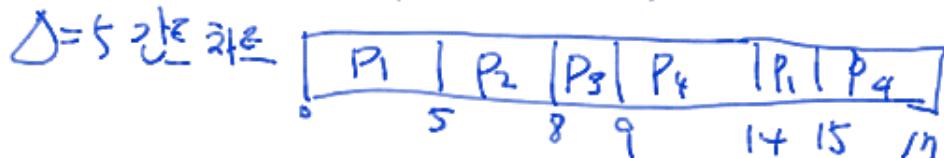
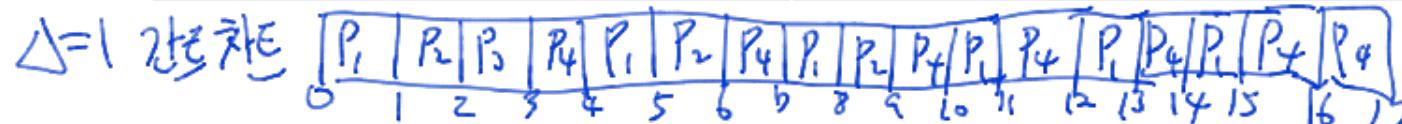
- $\Delta \rightarrow \infty$ FCFS

- $\Delta \rightarrow 0$ Processor sharing (* context switching overhead) ; 수시로 자주 일어나
프로세스들이 동시에
실행되는 것처럼 보임.

- Example: Average turnaround time (ATT)

- ATT = 11.0 msec ($\Delta = 1$), 12.25 msec ($\Delta = 5$)

Process	Burst Time (msec)
P_1	6
P_2	3
P_3	1
P_4	7



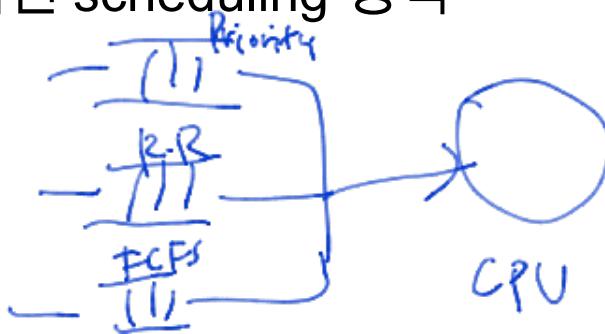
Multilevel Queue Scheduling

- Process groups

- System processes : OS가 쓰는 프로세스
- Interactive processes : 사용자와 상호작용하는 프로세스 (게임 등)
- Interactive editing processes : 워드, PPT 등
- Batch processes : 컴퓨터가 일정 시간에
- Student processes

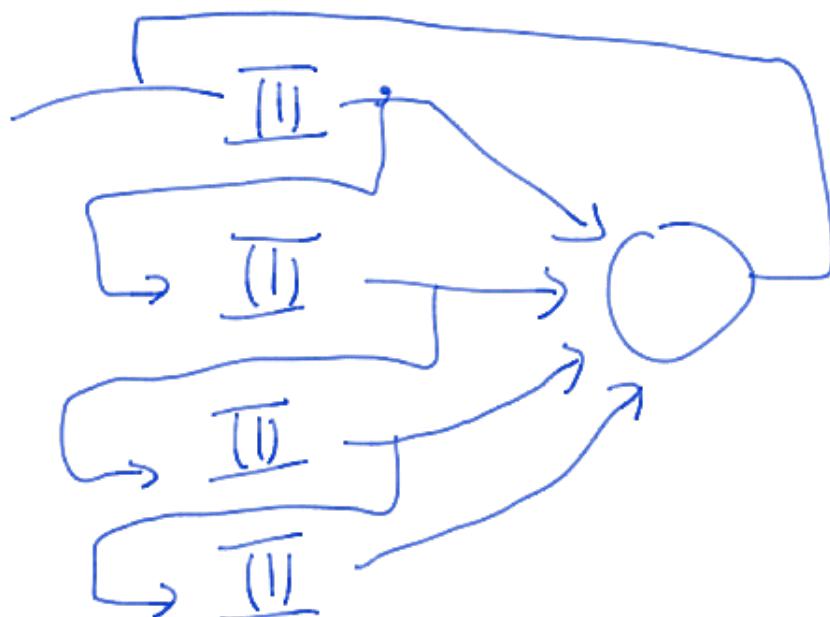
- Single ready queue → Several separate queues

- 각각의 Queue에 절대적 우선순위 존재
- 또는 CPU time을 각 Queue에 차등배분
- 각 Queue는 독립된 scheduling 정책



Multilevel Feedback Queue Scheduling

- 복수 개의 Queue
- 다른 Queue 로의 점진적 이동
 - 모든 프로세스는 하나의 입구로 진입
 - 너무 많은 CPU time 사용 시 다른 Queue 로
 - 기아 상태 우려 시 우선순위 높은 Queue 로



상용 OS는 여전까지 배운 내용을 복합적으로 사용.

프로세스 생성과 종료

Process Creation

- 프로세스는 프로세스에 의해 만들어진다!

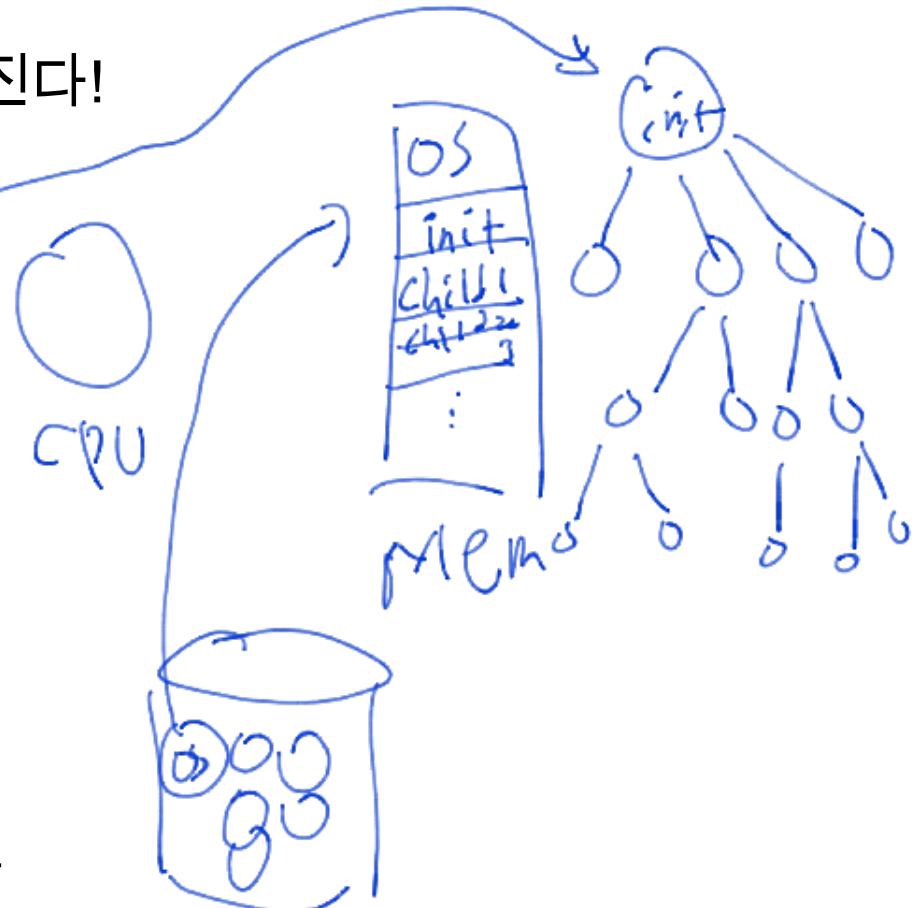
- 부모 프로세스 (Parent process)
- 자식 프로세스 (Child process)
 - cf. Sibling processes 형제프로세스
- 프로세스 트리 (process tree)

- Process Identifier (PID)

- Typically an integer number
- cf. PPID: Parent PID

- 프로세스 생성

- *fork()* system call – 부모 프로세스 복사
- *exec()* – 실행파일을 메모리로 가져오기



예제: Windows 7 프로세스 나열



process status

예제: Ubuntu Linux 프로세스 나열

```
hjyang@rm303:~$ ps -1
F S  UID   PID  PPID C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S 1000  2197  2189 0 80    0 - 1799 wait    pts/0    00:00:00 bash
0 R 1000  2368  2197 0 80    0 - 1177 -      pts/0    00:00:00 ps

hjyang@rm303:~$ ps -axl
F  UID   PID  PPID PRI  NI   VSZ   RSS WCHAN  STAT TTY          TIME COMMAND
4   0     1     0 20   0 3536 1948 poll_s Ss   ?          0:00 /sbin/init
1   0     2     0 20   0     0 0 kthrea S   ?          0:00 [kthreadd]
1   0     3     2 20   0     0 0 run_ks S   ?          0:00 [ksoftirqd/0]
1   0     4     2 20   0     0 0 worker S   ?          0:00 [kworker/0:0]
5   0     5     2 20   0     0 0 worker S   ?          0:00 [kworker/u:0]
.....
1 1000  1820     1 20   0 55944 3992 poll_s S1   ?          0:00 /usr/bin/gnome-...
4 1000  1831  1658 20   0 50924 9096 poll_s Ss1  ?          0:00 gnome-session --sessio...
0 1000  2196  2189 20   0 2404  724 unix_s S   ?          0:00 gnome-pty-helper
0 1000  2197  2189 20   0 7196 3572 wait    Ss   pts/0    0:00 bash
0 1000  2370  2197 20   0 4708  708 -      R+   pts/0    0:00 ps -axl

hjyang@rm303:~$
```

Process Termination

- 프로세스 종료

- *exit()* system call
- 해당 프로세스가 가졌던 모든 자원은 O/S에게 반환
(메모리, 파일, 입출력장치 등)