

프로세스의 개념

- "Process is a program in execution".

- 프로세스의 문맥(context)

v CPU 수행 상태를 나타내는 하드웨어 문맥

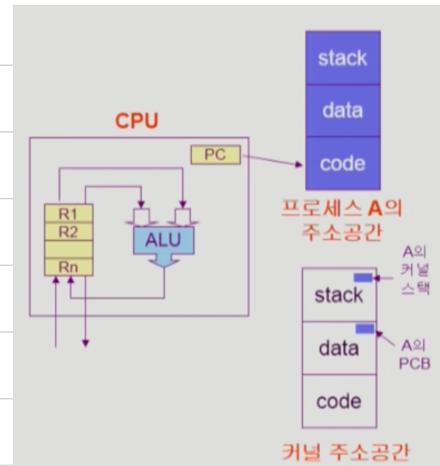
- Program Counter
- 각종 Register

v 프로세스의 주소 공간

• Code, data, stack

v 프로세스 관리 커널 자료구조

- PCB (Process Control Block)
- Kernel Stack



⇒ 현재 시점의 정확한 상태를 파악하기 위해 문맥을 살필.

프로세스의 문맥을 파악하고 있지 않으면 multitasking을 할 때 다시 이 프로세스의 차례일 때 어디까지 수행했는지 파악하지 못하고 처음부터 다시 수행하게 됨.

프로세스의 상태 (Process state)

- 프로세스는 상태(state)가 변경되어 수행된다.

v Running

- CPU를 잡고 Instruction을 수행중인 상태

v Ready

- CPU를 기다리는 상태 (메모리 등 다른 리소스를 모두 만족하고)

v Blocked (Wait, sleep)

- CPU를 주어도 당장 instruction을 수행 할 수 없는 상태

- Process 자신이 요청한 event (예: I/O) 가 즉시 만족되지 않아 이를 기다리는 상태
ex) 디스크에서 file을 읽어야 할 경우

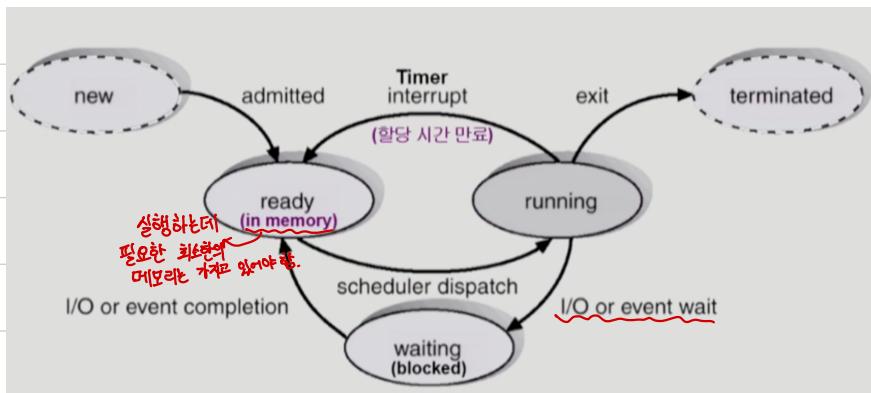
v New: 프로세스가 생성중인 상태

v Terminated: 수행(execution)이 끝난 상태

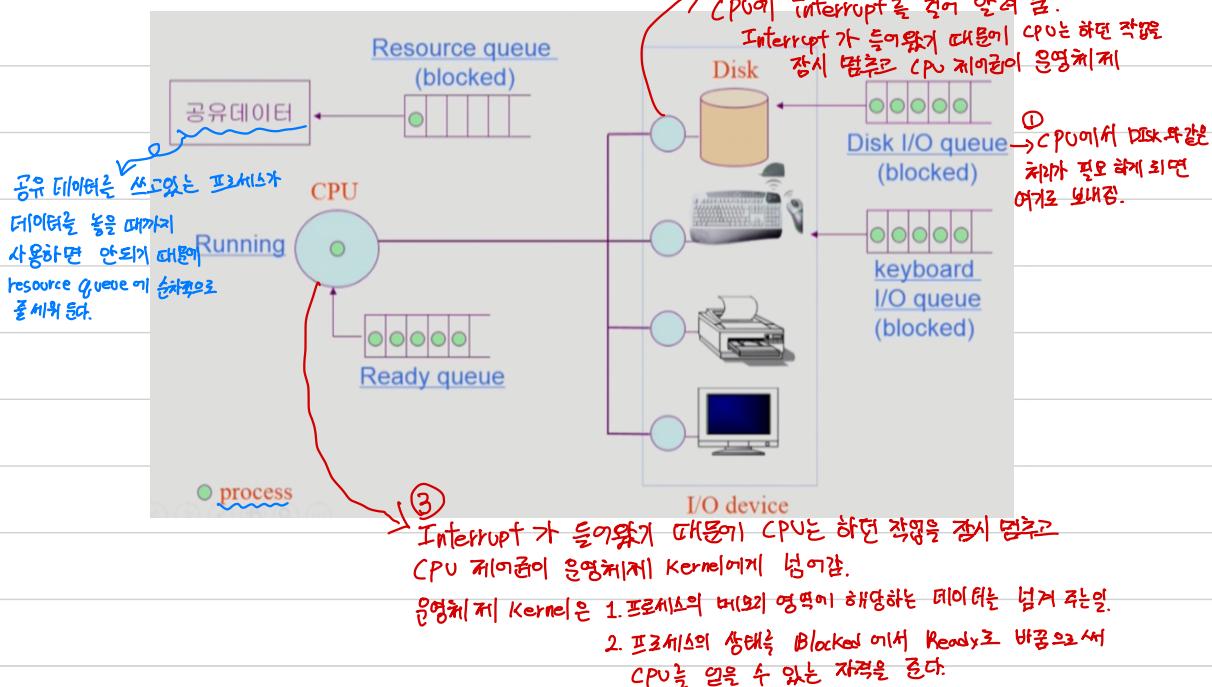
수행이 끝났지만 정지할게 약간 남아있는 상태

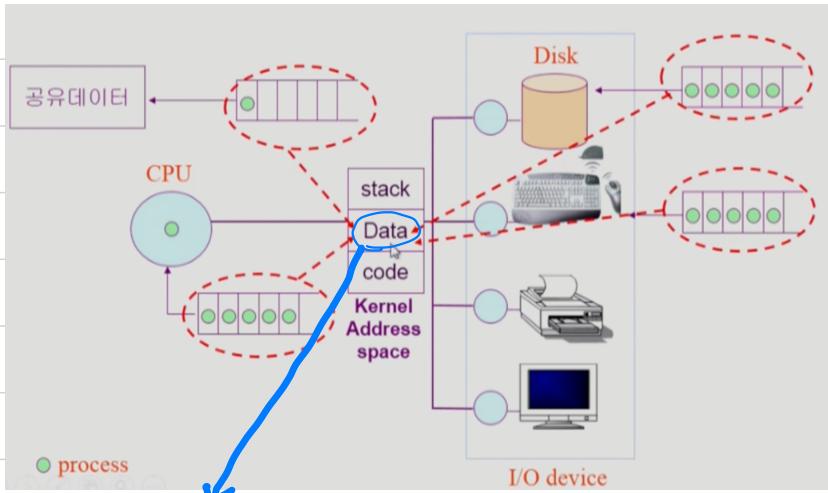
→ 이 두개는 Process state에 포함 시키지 않지만 경우에 따라서 포함 하겠음.

프로세스 상태도



프로세스 구조도





위의 그림에서 각 상태마다 queue가各自 존재하는 것처럼 그렸지만 운영체제 Kernel이 Data 영역에 자료구조로 queue를 만들어 놓고 프로세스 상태를 바꿔 가면서 Ready 상태 프로세스에게는 CPU를 주고 Blocked 상태 프로세스에게는 CPU를 주지 않으면서 운영하는 것.

Process Control Block (PCB)

- 운영체제가 각 프로세스를 관리하기 위해 프로세스당 유지하는 정보

- 다음의 구성 요소를 가진다. (구조체로 유지)

- (1) OS가 관리상 사용하는 정보
 - Process state, Process ID
 - Scheduling information, priority

- (2) CPU 수행 관련 하드웨어 값
 - Program counter, registers

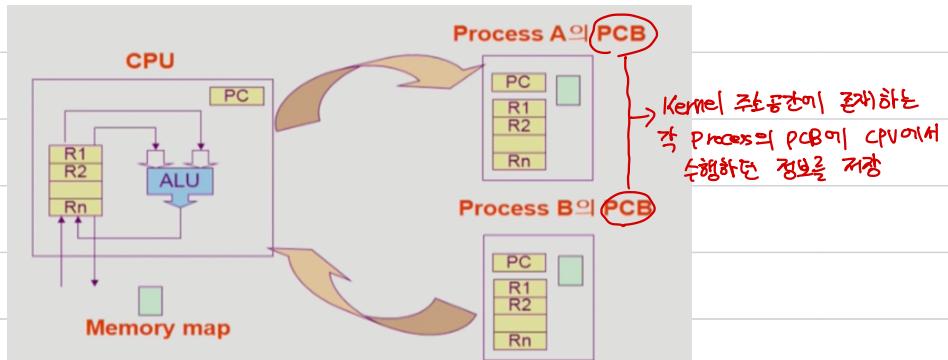
- (3) 메모리 관련
 - Code, data, stack의 위치 정보

- (4) 파일 관련
 - Open file descriptors...

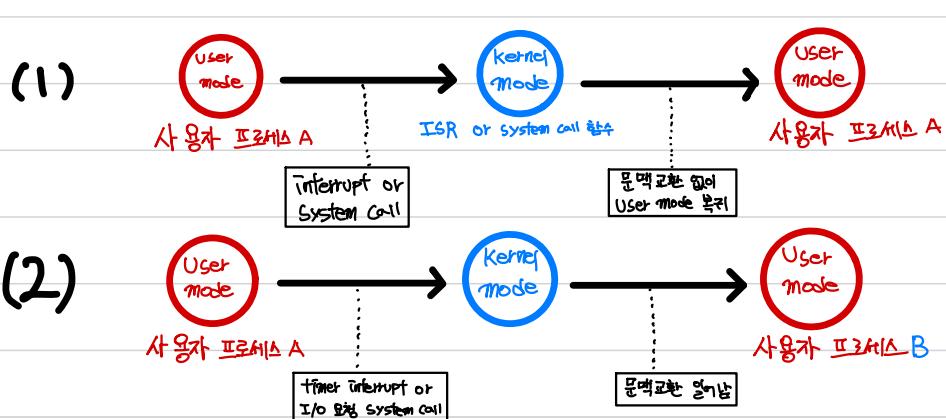
(1)	Pointer	Process state
(2)	Process number	
(3)	Program Counter	
(4)	Registers	
	Memory limits	
	List of open files	
	:	

문맥 교환 (Context switch)

- CPU를 한 프로세스에서 다른 프로세스로 넘겨주는 과정
- CPU가 다른 프로세스에게 넘어갈 때 운영체계는 다음을 수행
 - ✓ CPU를 내어주는 프로세스의 상태를 그 프로세스의 PCB에 저장
 - ✓ CPU를 새롭게 얻는 프로세스의 상태를 PCB에서 일으킴.



- System call off Interrupt 발생시 반드시 Context switch가 일어나는 것은 아님
Process가 운영체계로 오경
운영체계로 넘어온 경우
이 두 경우는 프로세스에서 운영체계로 CPU 사용 권한이 넘어감으로 Context switching이 아니다!!



* (1)의 경우도 CPU 수행 정보를 Context의 일부로 PCB에 Save 해야 하지만

문맥 교환을 하는 (2)의 경우 그 부담이 훨씬 큼 (eg. cache memory flush)

Process가 바뀌면 여기 있는 정보를 다 삭제 해야
하므로 상당히 오버헤드임.

프로세스를 스케줄링하기 위한 큐

- Job queue

- 현재 시스템 내에 있는 모든 프로세스의 집합.

- Ready queue

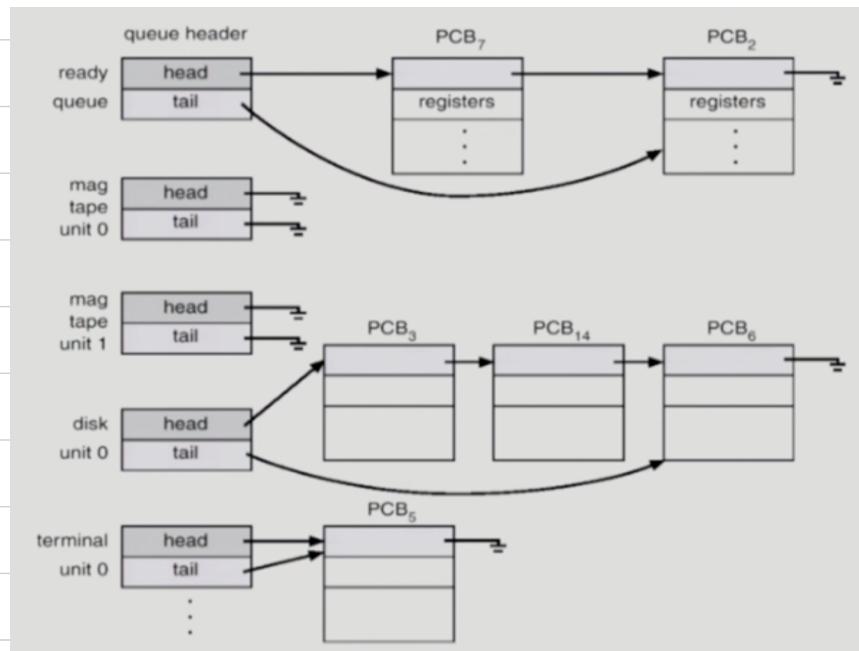
- 현재 메모리 내에 있으면서 CPU를 잡아서 실행 되기를 기다리는 프로세스의 집합

- Device queue

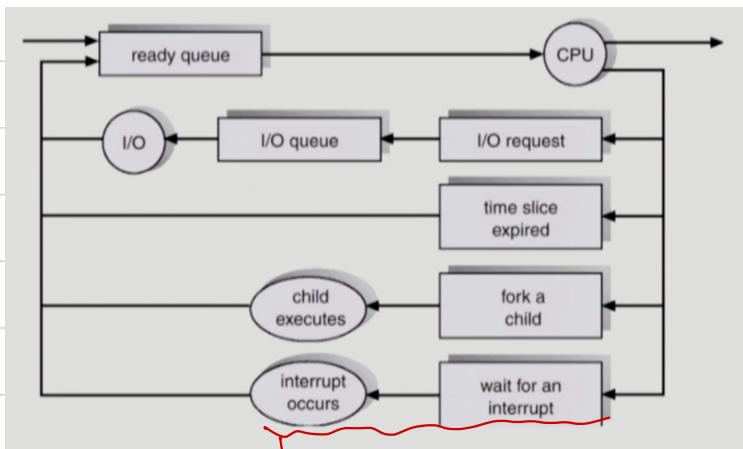
- I/O device의 처리를 기다리는 프로세스의 집합

- 프로세스들은 각 큐들을 오가며 수행된다.

Ready queue와 다양한 Device queue



프로세스 스케줄링 큐의 모습



↳ 고객이선 Interrupt는 ready queue로 돌아가는 것처럼 그렸지만
정확하게는 그렇지 않다!!

스케줄러 (Scheduler)

— Long-term scheduler (장기 스케줄러 or job scheduler)

↳ 시작 프로세스 중 어떤 것을 ready queue로 보낼지 결정

↳ 프로세스에 memory (및 각종 자원)을 주는 문제

↳ degree of Multiprogramming을 제어 \Rightarrow memory에 물려가 있는 Process의 수 제어 \rightarrow 매우 중요한 이유!

↳ Time sharing system에는 보통 장기 scheduler가 없음 (모두 ready) \rightarrow 요즘 운영체계는 long-term scheduler 없음...

— Short-term scheduler (단기 스케줄러 or CPU scheduler)

↳ 어떤 프로세스를 다음번에 running 시킬지 결정

↳ 프로세스에 CPU를 주는 문제

↳ 충돌 | 빌려주 힘 (millisecond 단위)

— Medium-Term Scheduler (중기 스케줄러 or Swapper)

↳ 여유 공간 마련을 위해서 프로세스를 통째로 메모리에서 디스크로 쫓아냄.

↳ 프로세스에게 memory를 뺏는 문제

↳ degree of multiprogramming을 제어

이걸 이용해 제어

프로세스의 상태 2 (Process state)

— 프로세스의 상태

v Running

- CPU를 짊고 Instructions을 수행 중인 상태

v Ready

- CPU를 기다리는 상태 (메모리 등 다른 조건 모두 만족하고)

v Blocked (Wait, sleep)

- I/O 등의 event를 (스스로) 기다리는 상태
예) 디스크에서 읽어올 데 있어화야 하는 경우

v Suspended (Stopped)

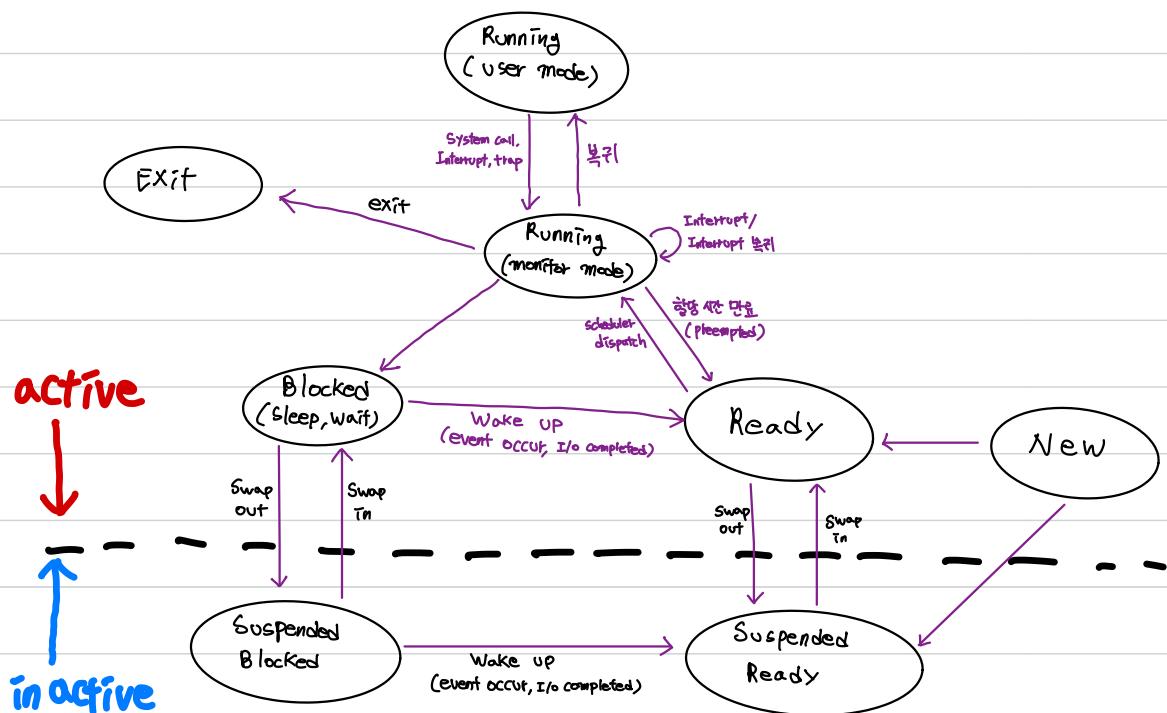
- 외부적인 이유로 프로세스의 수행이 정지된 상태
- 프로세스는 동지로 디스크에 Swap out 된다.
- 사용자가 프로그램을 일시정지 시킨 경우 (break key)
- 시스템이 여러 이유로 프로세스를 잠시 중단 시킴.
(메모리에 너무 많은 프로세스가 몰려와 있을 때)

* Blocked: 자신이 요청한 event가 만족되면 ready

Suspended: 외부에서 resume해 주면 Active

외부에서 다시
재개 해 주어야 시작 가능.

프로세스 상태도 2.



Thread

- A thread (or lightweight process) is a basic unit of CPU utilization

- Thread의 구성

- v Program Counter
- v Register Set
- v Stack Space

* 전통적인 개념의 heavy weight process는 하나의 thread를 가지고 있는 task로 볼 수 있다.

- Thread가 동료 thread와 공유하는 부분 (=task)

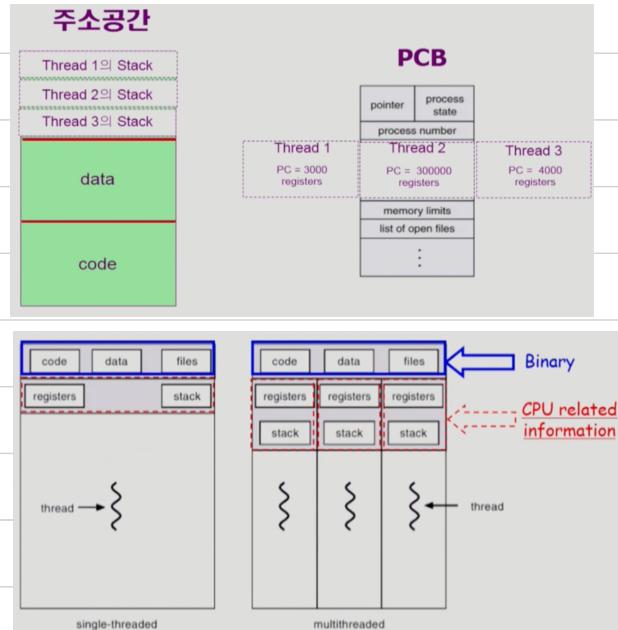
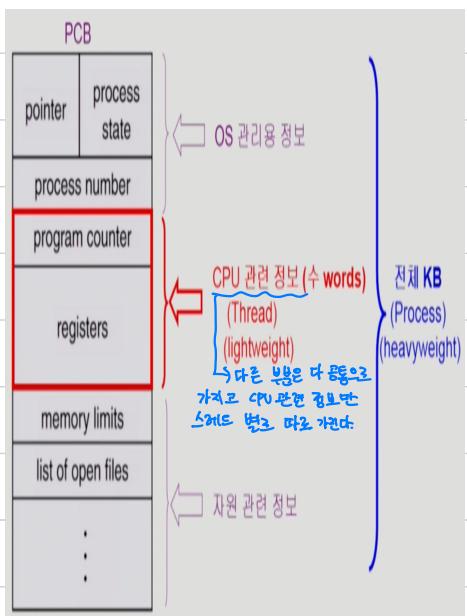
- v Code section
- v Data section
- v OS resources

- [?] 스레드로 구성된 태스크 구조에서는 하나의 서버 스레드가 blocked (Waiting) 상태인 동안에도 동일한 태스크 내의 다른 스레드가 실행 (Running) 되어 빠른 처리를 할 수 있다.

- 동일한 일을 수행하는 다른 스레드가 협력하여 높은 처리율 (throughput)과 성능 향상을 얻을 수 있다.

- 스레드를 사용하면 병렬성을 높일 수 있다.

↳ CPU가 여전히 달린 컴퓨터에서만 활용할 수 있는 장점.



Benefits of Threads

- Responsiveness

- v e.g. multi-threaded Web - if one thread is blocked (e.g. network) another thread continues (e.g. display)

- Resource Sharing

- v n threads can share binary code, data, resource of the process

- Economy

- v Creating & CPU switching Thread (rather than a process)

v Solaris의 경우 의 드라마 overhead가 각각 30㎯, 5㎯

↳ 즉, 같은 일을 한다면 각각의 프로세스로 만드는 것이 아니라

multiprocessor

하나의 프로세스 안에서 각각의 스레드로 만드는게 훨씬 효율적이다.

- Utilization of MP Architectures

- v each thread may be running in parallel on a different processor

Implementation of Threads

- Some are supported by kernel

- v Windows 95/98/NT

- v Solaris

- v Digital UNIX, Mach

Kernel
Threads

Thread가 여러개 있다는 사실을

kernel thread가 알고 있다.

→ 하나의 thread에서 다른 Thread로

CPU가 넘나기는 것을 kernel이 CPU scheduling

을 하듯이 넘겨 준다.

- Some are supported by library

- v POSIX pthreads

- v Mach C-threads

- v Solaris threads

User
Threads

Process 안에 thread가 여러개 있다는 사실을

운영체제는 모르고 유저 프로그램이 스스로 여러개의 threads를

library의 지원을 받아 관리하는 것.

⇒ 운영체제는 모르고 있기 때문에 구현상에 약간의 제약점이

있을 수 있다.

- Some are real-time threads