

10. 프로세스와 스레드

10-1 프로세스 개요

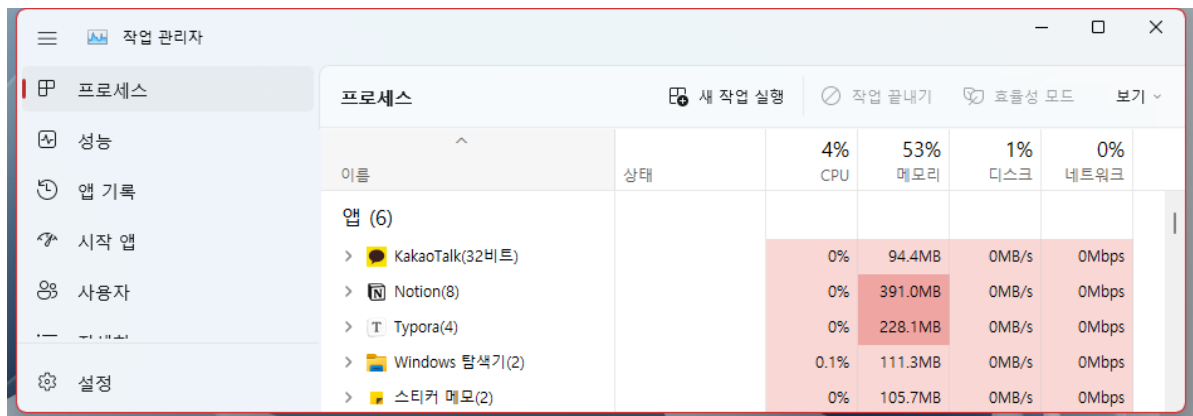
프로세스

프로세스 = 실행중인 프로그램

실행 전에는 @보조기억장치에 저장

실행 중에는 @메모리에 적재되며 프로세스를 생성

프로세스를 확인하는 쉬운방법? 작업관리자의 프로세스 탭 (유닉스는 ps명령어로 확인)



프로세스		4%	53%	1%	0%
		CPU	메모리	디스크	네트워크
이름	상태				
앱 (6)					
> KakaoTalk(32비트)		0%	94.4MB	0MB/s	0Mbps
> Notion(8)		0%	391.0MB	0MB/s	0Mbps
> Typora(4)		0%	228.1MB	0MB/s	0Mbps
> Windows 탐색기(2)		0.1%	111.3MB	0MB/s	0Mbps
> 스티커 메모(2)		0%	105.7MB	0MB/s	0Mbps

프로세스에는

1. 사용자가 볼 수 있는 포그라운드 프로세스(background process)
2. 사용자가 볼 수 없는 백그라운드 프로세스(background process)

백그라운드 프로세스는

1. 사용자와 상호작용을 하는 프로세스와
2. 사용자와 상호작용을 하지 않는 프로세스로 나뉘며,
 - ↳ 이를 유닉스에서는 데몬, 윈도우에서는 서비스라고 불린다.
 - ↳ 윈도우의 서비스 또한 작업관리자의 서비스 탭에서 확인 가능

프로세스 제어블록

배경

프로세스 실행을 위해서는 CPU가 필요. 하지만 CPU 자원은 한정적.

=> 프로세스들이 돌아가며 한정된 시간만큼만 사용하기로 함

- 이 때 한정된 시간이 다 되었음을 알리는 것이 인터럽트! (타이머 인터럽트)
- 인터럽트가 발생한다면, 다음 내 차례가 올 때까지 줄을 기다려야함
- 운영체제는 여기서 프로세스의 실행 순서를 관리하고, 자원을 배분하는 역할을 하는 것 (이를 프로세스 제어블록 PCB, Process Control Block 이라고 부름)

프로세스 제어블록

- 프로세스와 관련된 정보를 저장하는 자료구조 (상품 태그와 같다.. 제품을 식별하기 위한 정보를 저장)
- 메모리의 커널 영역에 생성
- 프로세스 생성 시 만들어지고, 끝나면 폐기

프로세스 제어블록에 담긴 정보들

- 프로세스 ID (PID)
: 프로세스의 주민번호와 같은 식별 번호, 같은 프로그램을 여러번 수행해도 각각의 고유번호 존재.
- 레지스터 값
: 진행했던 작업을 이어서 진행하기 위해 레지스터의 중간값들을 모두 복원
- 프로세스 상태
: 입출력장치를 사용하기 위해 대기중인지, CPU 사용 대기중인지, CPU 이용중인지, 등의 상태정보
- CPU 스케줄링 정보
: 언제 어떤 순서로 CPU를 할당받을지에 대한 정보
- 메모리 관련 정보
: 프로세스마다 메모리에 저장된 위치가 다르기 때문에, 어느 주소에 저장되어있는지에 대한 정보를 저장
- 사용한 파일과 입출력장치 목록
: 입출력장치나 파일을 사용한 내용 명시

=> 위 모든 정보가 모든 프로세스마다 가격표처럼 붙어있다고 생각하면 됩니다!

문맥교환

프로세스A의 실행시간이 도달했을 때, 다음 차례 때 이어서 실행을 위한 중간정보 데이터들 (레지스터값, 메모리 정보, 열었던 파일, 입출력 장치 등)을 백업해야함

이렇게 기억해야할 정보 (중간정보)를 **문맥** 이라고 하며 PCB에 표현되어있다.

PCB에 저장되는 정보가 곧 문맥이라고 봐도 무방.

=> 그렇기 때문에 타이머 인터럽트가 아닌 다른 예기치 못한 인터럽트가 발생했을 때도 PCB를 보면 이어서 작업을 진행할 수 있다.

문맥 교환의 과정

그렇다면 프로세스 A의 실행 시간이 끝났을 때 백업을 해야겠죠?

=> PCB에 문맥을 백업

이제 새로운 프로세스 B를 실행하려면 기존 작업 내용 (즉, 문맥)을 가져와야겠죠?

=> PCB에서 문맥을 가져오기

OK! PCB는 메모리의 커널영역에 생성돼. 그렇다면 사용자 영역에는 뭐가 저장돼?

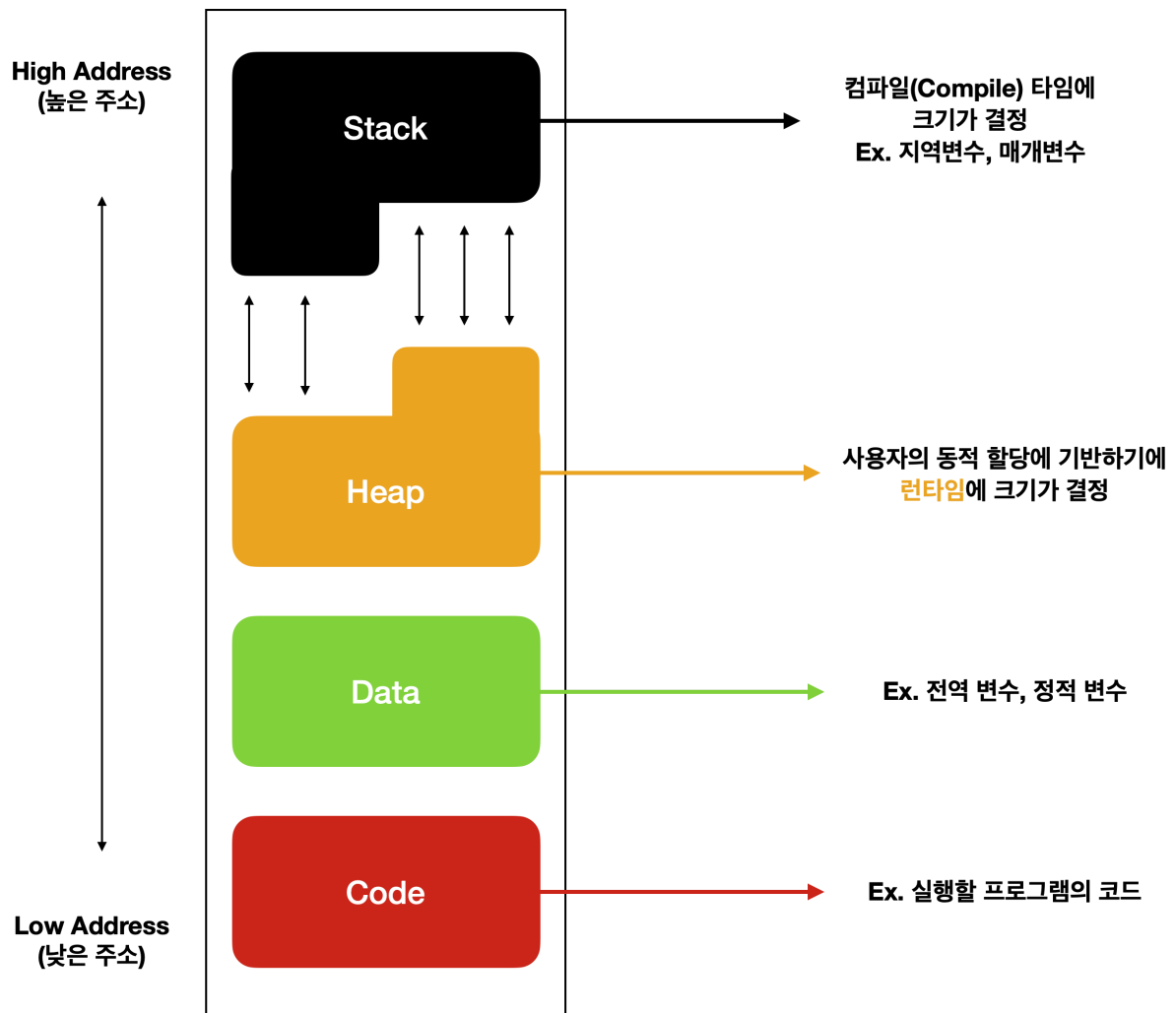
*** 프로그래밍에 있어 매우 중요한 내용이니 특별히 집중 !!

정적 할당 영역 (크기가 고정)

- 코드 영역 (=텍스트 영역)
: 기계어로 이루어진 명령어. 읽기 전용 공간으로 쓰기가 금지되어있음
- 데이터 영역
: 프로그램이 실행되는 동안 유지할 데이터가 저장되는 공간
 - 전역변수: 프로그램 실행동안 유지되며, 프로그램 전체에서 접근할 수 있는 변수

동적 할당 영역 (크기가 유동적)

- 힙 영역
: 프로그래머가 직접 할당할 수 있는 저장 공간.
할당을 했다면 반환도 해야함. => 더이상 사용하지 않는다 (인터럽트)고 운영체제에 알림
그렇지 않으면 메모리만 초과시킬 뿐.. (현재 실행중인 프로세스가 아니니까)
이렇게 반환하지 않아서 메모리 낭비를 초래하는걸 **메모리 누수** 라고 합니다
- 스택 영역
: 데이터를 일시적으로 저장. 담기는 값말고 잠깐 쓰다 말 값들.
e.g.) 함수의 실행이 끝나면 사라지는 매개변수, 지역변수가 대표적



10-2 프로세스 상태와 계층 구조

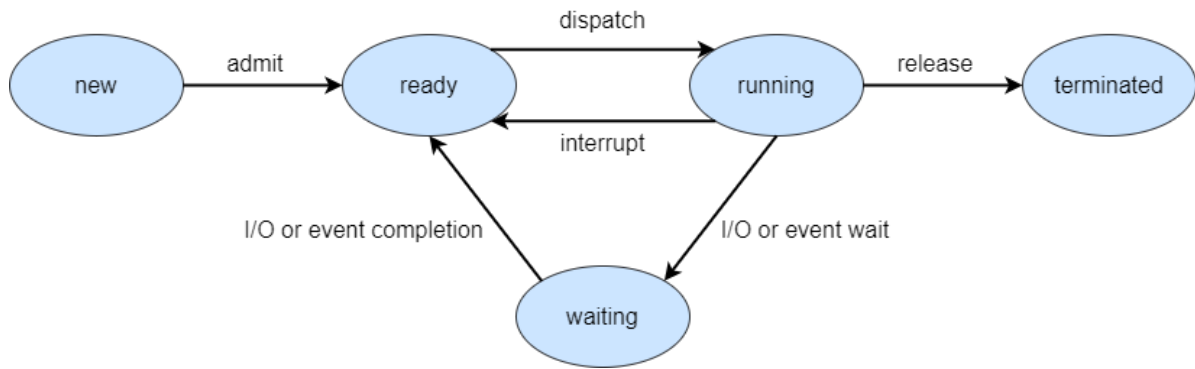
작업관리자 프로세스탭에 가면 상태를 볼 수 있다 (실행중, 일시정지, 등)

운영체제는 그런 상태를 PCB에 기록하고 동시에 실행되고있는 많은 프로그램들을 계층적으로 관리함

프로세스 상태

- 생성 new
: 생성중인 상태로 CPU 할당을 기다림
- 준비 ready
: 실행 차례를 기다리는 상태. (이 때 내 차례가 되어 실행상태로 전환되는 것을 **dispatch** 라고함)
- 실행 running
: CPU를 할당받아 실행중인 상태. 일정 시간 동안만 사용 가능하며 타이머 인터럽트 발생 시 다시 준비상태로 돌아감
실행 도중 입출력장치를 사용하여 입출력 장치의 작업이 끝나길 기다리는 동안은 대기상태
- 대기 blocked (waiting)
: 위와 같이 입출력이 끝날 때까지 (입출력 완료 인터럽트) 기다리는 상태
- 종료 terminated

: 프로세스 종료된 상태. 운영체제는 PCB와 프로세스가 사용한 메모리를 정



프로세스 계층 구조

프로세스 실행 도중 시스템 호출을 통해 다른 프로세스를 생성할 수 있음

이 때, 기존 실행 프로그램: 부모 프로세스, 부모 프로세스에 의해 생성된 프로세스를: 자식 프로세스

물론 각기 다른 프로세스이기 때문에 아까 언급된 프로세스ID (PID)는 다르다.

(일부 운영체제에서는 자식 프로세스에 부모 프로세스 아이디-PPID를 기록하기도 함)

자식 프로세스들은 부모프로세스가 되어 또다른 자식프로세스를 생성할 수 있다

=> 그렇게 생성되는 것이 프로세스 계층 구조 (트리라고 생각하면 편할까?)

e.g.) 최초의 프로세스 > 로그인 프로세스 > bash 셸 실행 프로세스 > Vim 문서 편집기 실행 프로세스

프로세스 생성 기법

10-3 스레드

프로세스를 구성하는 실행 흐름의 단위

프로세스당 여러개의 스레드가 있을 수 있음

웹브라우저 프로세스안에

- 화면 출력 스레드
- 입력 스레드
- 검색 스레드

와 같이 여러개 스레드가 있을 수 있다는 말씀 !!

여러개의 스레드는 동시 실행 가능! => 멀티 스레드 프로세스라고 부른다

프로그래밍 언어로 여러 스레드를 만든다면, 역시 동시에 실행 가능하다!

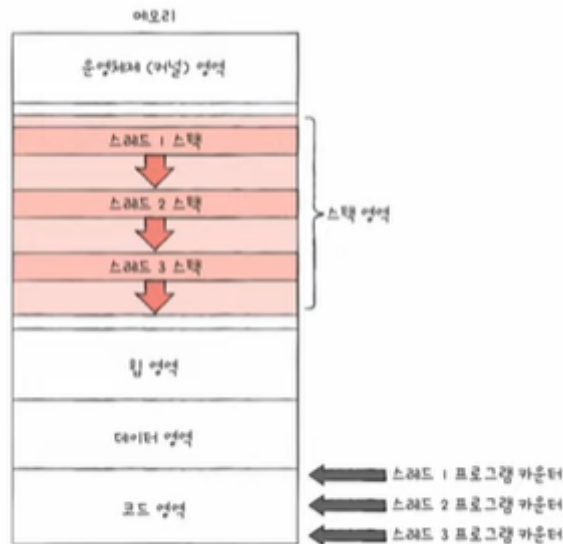
어떻게 스레드로 나누어서 실행할 수 있지?

스레드의 구성 요소

스레드 ID, 프로그램 카운터, 레지스터 값, 스택 등 => 실행에 필요한 최소한의 정보만 유지

프로세스의 자원을 공유하며 실행이 된다..

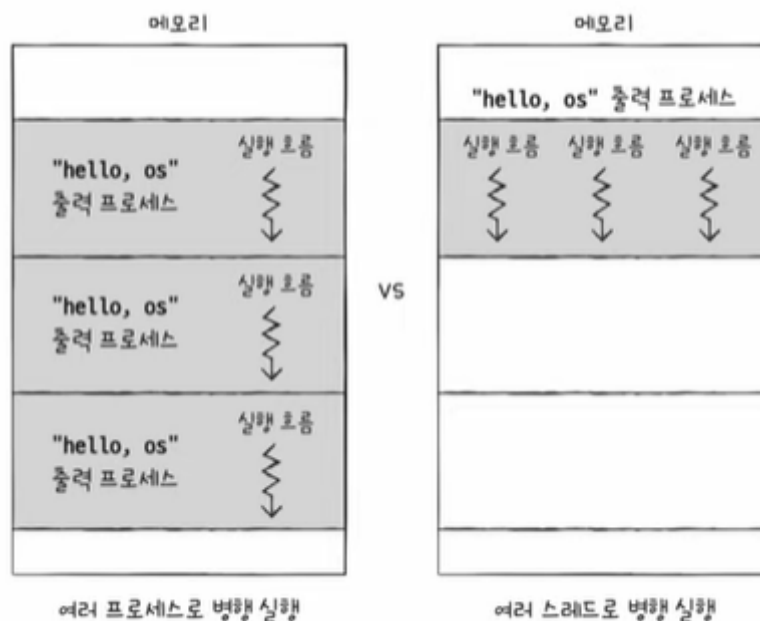
각 스레드의 영역이 나누어져있는 것이 아니라 프로세스의 자원을 공유하여 실행하는 것. (이미지 참고)



- 리눅스는 프로세스와 스레드를 구분하지는 않는다;

리눅스 토르발즈 왈: 스레드는 그냥 문맥이야;; 뭘 구분해;;

멀티프로세스 vs 멀티 스레드



프로세스끼리는 자원을 공유하지 않지만, 스레드는 같은 프로세스 내에서 자원을 공유한다.

=> 여기서 큰 차이점이 있는 것.

멀티 프로세스 예시.

fork를 하면 모든 자원이 복제되어 저장됨

=> 모든것이 동일한 저장소가 2개가 되는 셈. 포크를 할 수록 같은 저장소가 늘어만 가겠조?

멀티 프로세스: 같은 PCB 가격표를 여러개 가지고 있음 -> 남남처럼 독립적으로 실행

멀티 스레드: PCB 값을 공유해서 하나를 가지고 씬 -> 협력과 통신에 유리

근데 웃긴거?

프로세스 간에도 자원을 주고받을 수 있다 (프로세스간 통신 IPC)

단지 스레드보다 협력/통신이 유리하지 않을 뿐