Thread & Process

<http://you9010.tistory.com/entry/%EB%A9%80%ED%8B%B0-%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4Multi-Process%EC%99%80-%EB%A9%80%ED%8B%B0-%EC%8A%A4%EB%A0%88%EB%93%9CMulti-Thread>

<인터럽트>

운영체제가 자원들을 효율적으로 관리하기 위해 각 자원의 상태를 알아야함. 자원의 상태를 파악할 수 있는 방법은 폴링, 인터럽트가 있음

인터럽트는 각 자원들이 능동적으로 자신의 상태를 cpu에 알리는 것

주변 장치들의 인터럽트 : 하드웨어 인터럽트

경우에 따라서 cpu가 스스로에게 인터럽트를 함 -> 트랩 (시스템콜도 포함됨)

[듀얼모드간 전환은 시스템콜]

인터럽트가 발생하면 cpu는 현재 실행중인 놈의 정보를 시스템 스택(메모리)에 저장함. Pc와 레지스터 값을 메모리에 저장하고 저장한 위치를 스택 포인터에 갱신하여 나중에 알 수 있게함.

* Context switching

<프로세스> : 프로세스의 상태 변화는 인터럽트를 통해서

프로세스의 생성 = PCB 하나의 생성

생성 : 커널에 등록 + PCB생성

준비 : 프로세스가 CPU를 할당 받기 위해 기다리는 상태 (CPU를 할당하는게 디스패치)

대기 : 실행 중 IO요청이나 바로 확보 될 수 없는 자원을 요청한 상태

보류 : 메모리 부족 or 수상한 행동을 보일 때

[스와핑]

보류상태(suspended)는 cpu가 메모리 부족인 상황에서 프로세스의 메모리 공간을 뺏고 디스크로 보내는 것 (swap out) / 나중에 다시 메모리로 들어오는 것(swap in)

인터럽트 처리 전 후 프로세스가 같은 경우보다 다른 경우가 오버헤드가 큼

(모드 스위칭) (프로세스 스위칭)

<스레드>

User level thread

+ 커널이 존재를 모름 -> 스레드 스위칭시 커널의 개입이 필요없어 컨텍스트 스위칭 필요없음

- 스레드가 대기상태로 가면 프로세스도 대기됨

- 다중처리가 된다해도 스레드 단위의 다중처리가 안됨 <- 커널이 스레드를 보지 못하기 때문

Kernel level thread

+ 병렬처리 가능

- 스레드간 스위칭에도 커널의 개입이 필요

<가상메모리> (페이지, 세그먼트)

프로그램을 조각으로 나누어 일부분만 비연속적으로 적재하여 제한적인 메모리를 큰 메모리처럼

가상 주소 – 물리적 주소를 나눈 이유는 메모리 관리의 단순화 (사용자가 알필요 없게)

페이지 테이블 : 가상주소를 실주소로 변환하기 위해 씀, 프로세스당 하나, 크기는 프로세스의 페이지 수에 비례

PCB를 사용하면 PCB에서 주소를 얻어오고 그걸로 실제 워드에 접근해서 두번의 접근이 필요

이 때문에 TLB를 사용. Page cache.

Page의 문제점 : internal fragmentation, 하나의 로직이 두 페이지에 있는데 한 페이지가 디스크에 있다면?

* 세그먼테이션

세그먼테이션 : 프로그램은 서로다른 세그먼트로 이루어짐 / 페이징보다 복잡 / 외부단편화

세그먼트 테이블엔 base, length가 저장되는데 base는 실제 메모리주소 (세그먼트는 길이가 다르니까 계산할 수 없어서)