파이썬 - HW8

임베디드스쿨1기

Lv1과정

2020. 10. 6

박하늘

1. IPC(Inter Process Communication) Mechanism

1) IPC 가 필요한 경우

- 여러 사람이 하나의 파일을 공유할 때 (동시 접근 필요시)
- 작업을 세부 작업으로 쪼개서 속도를 높일 때
- 여러 프로세스나 스레드로 나누는 등의 모듈화 작업시
- 한번에 많은 일을 진행할 때 멀티태스킹 시

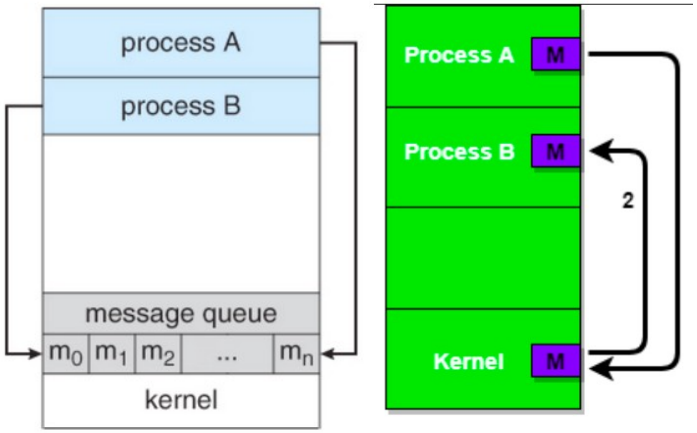
2) IPC 두 가지 방법

2-1) Message Passing: 메시지 교환

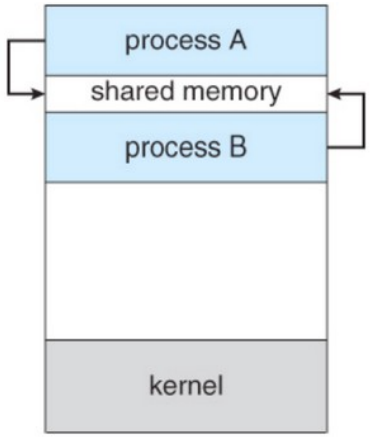
프로세스간 접근을 막기 위해서(memory protection) OS가 메시지를 direct or indirect로 교환해주는 방식
direct: 커널이 직접 관여 / indirect: 커널이 또다른 메세지 박스에 넣어놓으면 그것을 읽어오는 방식

2-2) Shared Memory : 데이터 공유

프로세스 A와 B를 모두 읽고 쓸수 있는 공유 메모리를 만들고 거기서 주고 받음. 성능 좋음
프로세스 간 동기화 문제 발생함 (즉, A에서 쓰고 B에서 쓰는 건데 B가 조금 일찍 읽히면 누가 쓴건지 모름)



< Message Passing >
(왼쪽: indirect , 오른쪽: direct)



< Shared Memory >

2. Critical Section

1) Critical Section (임계영역)

: 공유되는 자원, 동시접근하려는 그 포커싱 자원에서 문제가 발생하지 않게 독점을 보장해주는 코드 영역

2) Critical Section 특징

- 한순간 프로세스 하나만 진입하게 보장해줌. 임계 구역은 지정된 시간이 지난 후 종료된다.
- 병렬 컴퓨팅에서 둘 이상의 스레드가 동시에 접근해서는 안 되는 공유자원에 접근하는 코드의 일부로도 쓰임
- 스레드가 공유자원의 배타적인 사용을 보장받기 위해서 임계구역에 들어가거나 나올때에는 semaphore 같은 동기화 메카니즘이 필요하다. (lock에서 발전된 개념)

3) lock 이란 ?

: 자원을 사용하는 동안 어떤 프로세스도 못들어오게 하는 방식
예) shared data(공유자원):get_balance, put_balance 함수에 사용되는 account 데이터베이스
race condition: 그 shared data를 스레드 A,B가 동시 접근하는 상황
critical section: 이 race condition을 유발시키는 code segment 영역
lock: 한번에 한개의 스레드가 수행되도록 lock을 걸어 보호하고, 나올때 lock을 풀어줌
즉, 스레드 A가 lock이 걸린 상태에서 아무리 interrupt가 걸려도 lock이 해제된 후에 스레드 B수행함

```
int withdraw(account, amount) //해당 account에서 amount만큼 인출하는 함수
{
    balance = get_balance(account); //계좌에서 현재 잔액을 가져옴
    balance = balance -amount; //남은 잔액은 기존 잔액에서 amount만큼 뺀 금액
    put_balance(account, balance); //account에 변경된 잔액을 갱신함
    return balance;
}
```

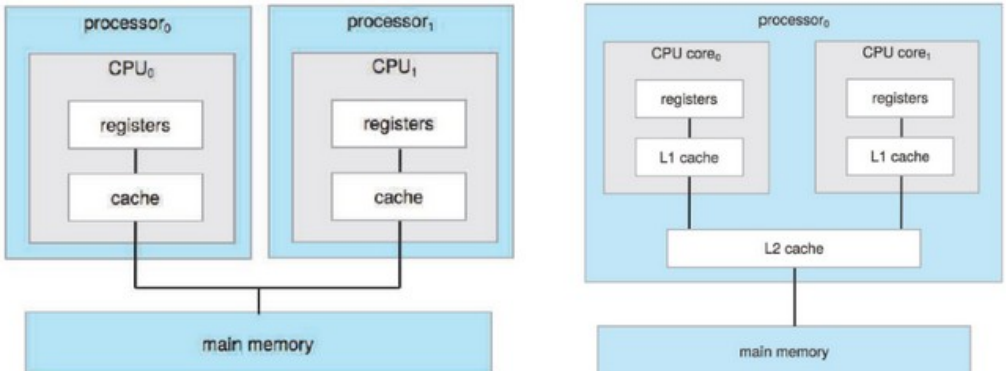
lock

```
int withdraw(account, amount) //해당 account에서 amount만큼 인출하는 함수
{
    lock(lock);
    balance = get_balance(account); //계좌에서 현재 잔액을 가져옴
    balance = balance -amount; //남은 잔액은 기존 잔액에서 amount만큼 뺀 금액
    put_balance(account, balance); //account에 변경된 잔액을 갱신함
    unlock(lock);
    return balance;
}
```

3. Operating system – multiprocessing/programming

1) 멀티 프로세싱 (Multi-processing)

:다수의 프로세서가 서로 협력적으로 일을 처리하는 것을 의미
(컴퓨터는 1대인데 프로세서(CPU)는 2개 이상이다. 보통 멀티코어시스템 포함함)



2) 멀티 프로그래밍 (Multi-programming)

: 프로세서가 입출력 작업의 종료를 대기할 동안 하나의 프로세서에서 다른 프로그램을 수행할 수 있도록 하는 것
: 프로세서의 자원 낭비를 막음

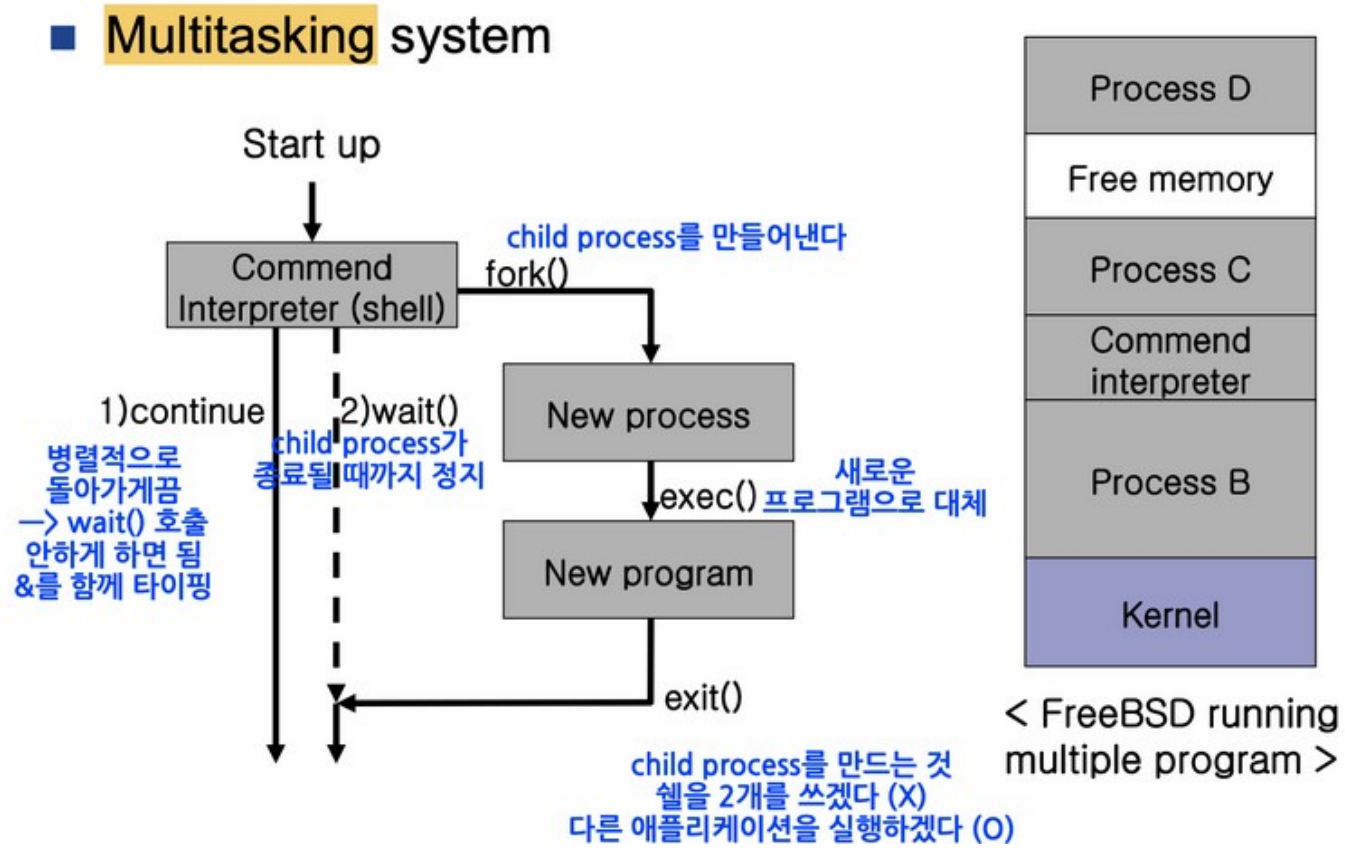
3) 멀티 태스킹 (Multi-tasking)

: Task란 process보다 확장된 개념, 프로세스나 스레드를 리눅스에서 스케줄링 할 수 있는 최소 단위가 태스크이다
: task를 OS의 **스케쥴링**에 의해 task를 번갈아가며 수행하는 것, 동시 수행하는 것처럼 느낌

3. Operating system – Multitasking

3) 멀티 태스킹 (Multi-tasking)

: 동시에 2개의 태스크를 하나의 CPU에서 구동을 시키면 이론적으로는 그 CPU에서 50%의 파워로 2 개의 태스크가 동시에 병렬로 동작해야 하는데 실제 하드웨어 CPU는 한 번에 하나의 태스크 밖에 수행할 수 없으므로 시간을 분할(time slice) 한 그 가상 시간(virtual runtime) 만큼 교대로 수행을 하여 만족시킨다.



3. Operating system – Multithreading

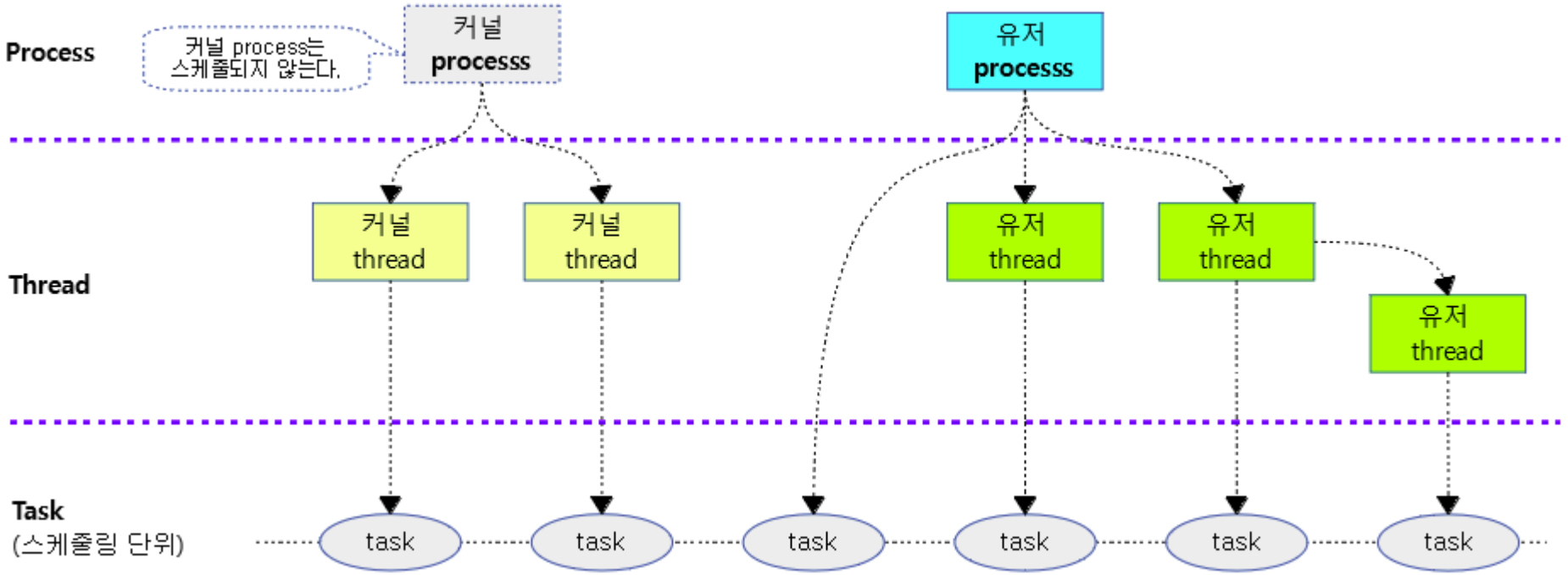
4) 멀티 스레딩 (Multi-threading)

: 하나의 프로세스를 여러 개의 실행 단위이며, 여러 개의 스레드끼리 자원을 공유하는 것

(참고)

프로세스: 실행될 때 os로부터 자원을 할당받아 실행되는 프로그램

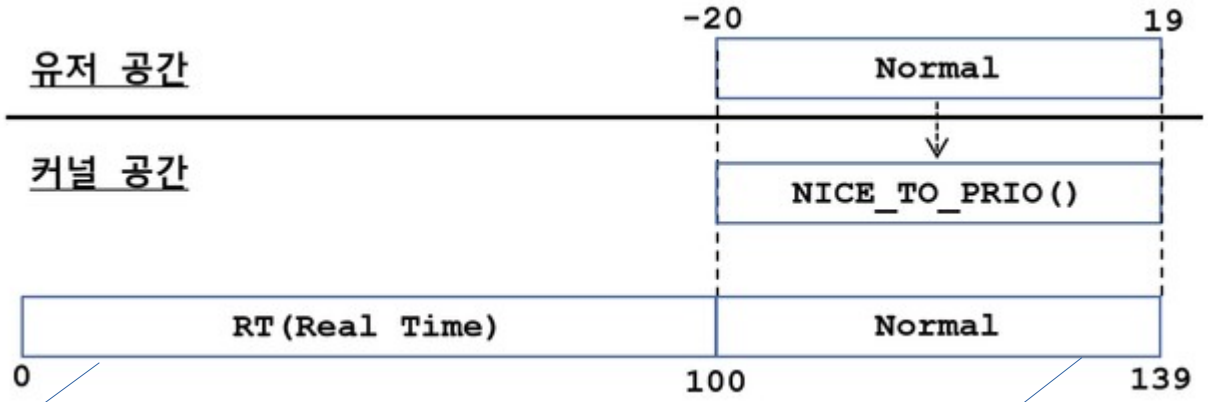
스레드: 한 프로세스 내에서 동작되는 여러 실행의 흐름



4. 우선순위 (nice)

1) 스케줄링:우선순위(nice)

: 정수형 nice 범위는 -20~19인데 이 값은 커널 공간에서 100~139 사이의 값으로 변환되어 관리됨
: 커널 공간에서 RT 스케줄러 클래스에 등록한 RT 프로세스 우선 순위 범위는 0~99이고,
CFS 스케줄러 클래스에서 구동하는 일반 프로세스 우선 순위 범위는 100~139입니다.



RT 프로세스

0~99 사이 우선권을 갖고 있습니다.

일반 프로세스에 비해 높은 우선권으로 실행하므로 빠르고 신속히 처리하도록 구현해야 합니다.

스케줄러 정책은 SCHED_FIFO 이며 우선 순위가 더 높은 RT 프로세스가 없으면 계속 CPU를 점유해 사용합니다. 런큐 내 서브 런큐 중 RT 프로세스를 관리하는 RT 프로세스 런큐가 있으며 struct rt_rq 구조체로 구현돼 있습니다.

일반 프로세스

100~139 사이 우선권을 갖고 있습니다. CFS 스케줄러 클래스로 실행되며 대부분 프로세스가 이 범주에 속해 있습니다.

CFS 스케줄러 정책에 따라 SCHED_NORMAL 스케줄링 정책으로 CFS 스케줄러는 시분할 방식으로 프로세스를 관리합니다.

4. 스케쥴러

- 참고이미지 및 스케줄러 추가 공부 필요.