파이썬 - HW8

임베디드스쿨1기
Lv1과정
2020. 10. 06
박성환

1. Review(Context Switching)

- **현재 진행하고 있는 Task(Process, Thread)의 상태나 레지스터 값을 저장하고 다음 진행할 Task의 상태나 레지스터값을 적용하는 과정**

- **Context Switching Cost**

- Cache 초기화
- Memory Mapping 초기화
- Kernel은 항상 실행되어야 함(메모리 접근을 위해)
- PCB를 저장하고 가져오는 오버헤드 시간

- **Context Switching 발생시점**

- Hardware를 통한 I/O 요청이나,
- OS/Driver 레벨의 Timer기반 Scheduling에 의해 발생

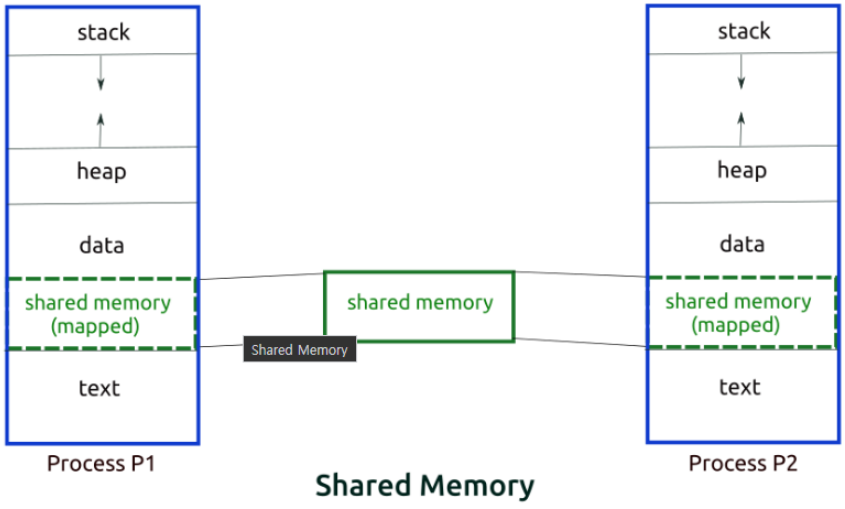
리눅스의 PCB = task_struct이 안에 state vector, 프로세서의 레지스터들의 컨텍스트가 저장됨.
리눅스의 thread_info에 RTOS로 치면 TCB에 해당하는 자료들이 저장됨

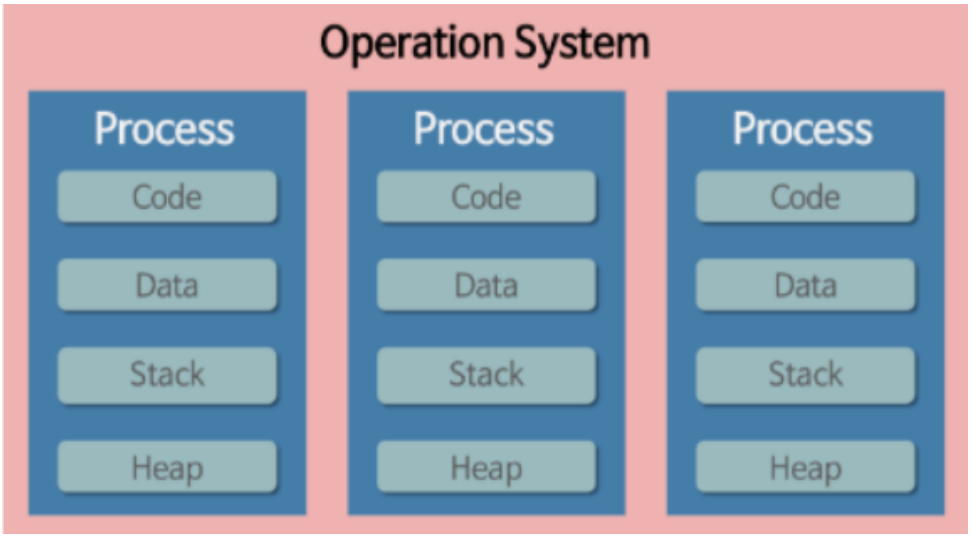
1. Review(공유자원)

Process 사이의 공유 : shared memory(kernel Data Section에 위치) 사용
Thread 사이의 공유 : 일반 메모리 Data Section(전역변수 저장하는...)에 위치하여 사용함

리눅스 : 대규모 데이터 처리에 유리
프로세스는 우선순위 기반이나 인터럽트는 NVIC 구조가 아니기에 중첩되지 않음
범용성 동작에 중점을 둠

RTOS : 하드웨어가 준 신호 즉각 받아서 인터럽트로 주고 응답하는데 유리
NVIC 기반...중첩인터럽트 허용





1. Review(IPC)

1) Message passing

커널(OS)가 memory protection을 위해 대리 전달해주는 것을 말합니다.

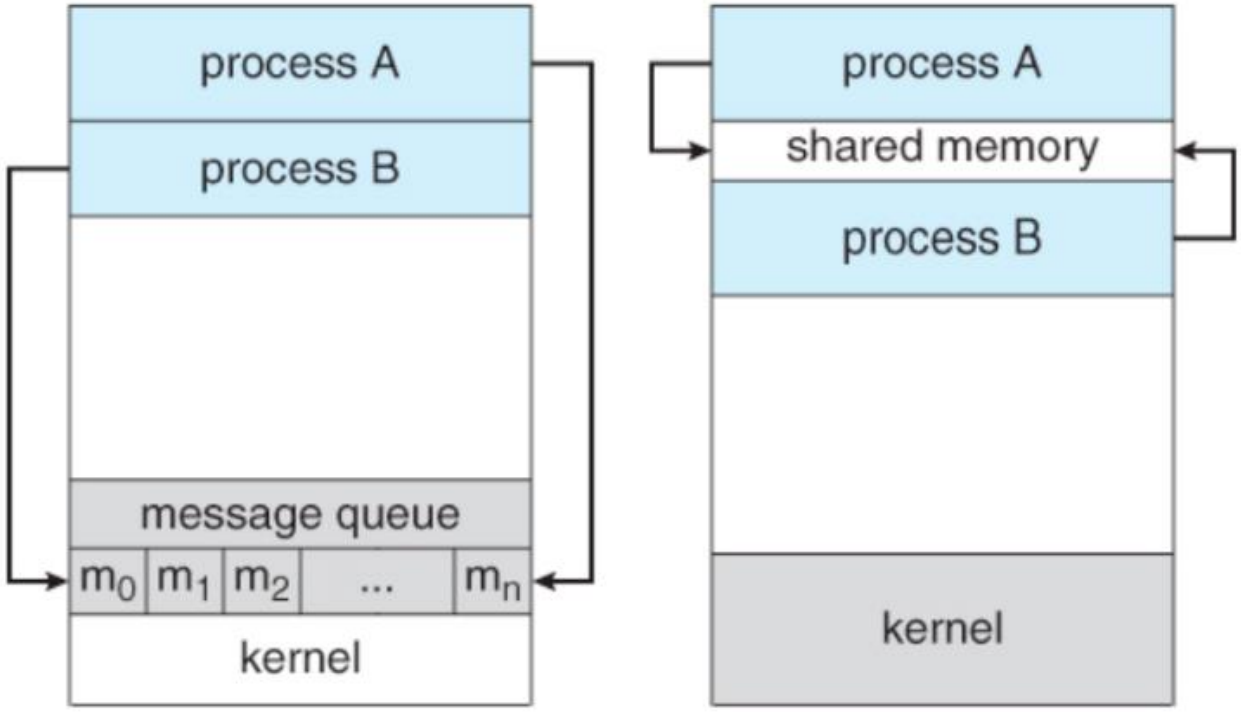
따라서 안전하고 동기화 문제가 없습니다 (OS가 알아서 동기화해주기 때문!) 하지만 성능이 떨어지는 단점을 가지고 있어요

Message passing은 direct communication과 indirect communication이 있습니다.

2) shared memory

두 프로세스간의 공유된 메모리를 생성 후 이용하는 것을 말해요!

성능이 좋지만 동기화 문제가 발생합니다 (App에서 직접 동기화를 해줘야 해요)



(a) Message Passing (b) Shared Memory

2. Spin Lock/Mutex/Semaphore

▪ Spin Lock

- Spin Lock 은 이름이 뜻하는대로, 만약 다른 스레드가 lock을 소유하고 있다면 그 lock이 반환될 때까지 계속 확인하며 기다리는 것이다. "조금만 기다리면 바로 쓸 수 있는데 굳이 컨텍스트 스위칭으로 부하를 줄 필요가 있나?" 라는 컨셉으로 개발된 것

- 크리티컬 섹션에 진입이 불가능할때 컨텍스트 스위칭을 하지 않고 잠시 루프를 돌면서 재시도 하는 것을 말한다.

- Lock-Unlcok 과정이 아주 짧아서 락하는 경우가 드문 경우(즉; 적절하게 크리티컬 섹션을 사용한 경우) 유용하다.

- Lock을 얻을 수 없다면, 계속해서 Lock을 확인하며 얻을 때까지 기다린다. 이른바 바쁘게 기다리는 busy wating이다. 바쁘게 기다린다는 것은 무한 루프를 돌면서 최대한 다른 스레드에게 CPU를 양보하지 않는 것이다.
- Lock이 곧 사용가능해질 경우 컨택스트 스위치를 줄여 CPU의 부담을 덜어준다. 하지만, 만약 어떤 스레드가 Lock을 오랫동안 유지한다면 오히려 CPU 시간을 많이 소모할 가능성이 있다.

- 하나의 CPU나 하나의 코어만 있는 경우에는 유용하지 않다. 그 이유는 만약 다른 스레드가 Lock을 가지고 있고 그 스레드가 Lock을 풀어 주려면 싱글 CPU 시스템에서는 어차피 컨택스트 스위치가 일어나야 하기 때문이다. 주의할 점 스핀락을 잘못 사용하면 CPU 사용률 100%를 만드는 상황이 발생하므로 주의 해야 한다. 스핀락은 기본적으로 무한 for 루프를 돌면서 lock을 기다리므로 하나의 쓰레드가 lock을 오랫동안 가지고 있다면, 다른 blocking된 쓰레드는 busy waiting을 하므로 CPU를 쓸데없이 낭비하게 된다.

2. Spin Lock/Mutex/Semaphore

- **Mutex**

- Mutex는 Binary Semaphore의 특별한 형태로 2개 이상의 Task가 자원을 공유 할 때 사용 함.

- Mutex를 이용하여 자원을 공유하는 경우, Task가 자원을 이용하기 위하여는 먼저 Token을 'Take'하여야 함
  Token을 갖는 Task는 자원의 사용이 종료되면 Token을 'Give'하여 다른 Task가 지원을 사용 할 수 있도록 하여야 함

- **Semaphore**

- 공유 자원에 대하여 둘 이상의 프로세스가 접근할 경우 자원의 값이 변함에 의한 오류나 교착상태(DeadLock)가 생길 수 있는 Critical Section의 동기화 문제를 해결하기 위한 방법

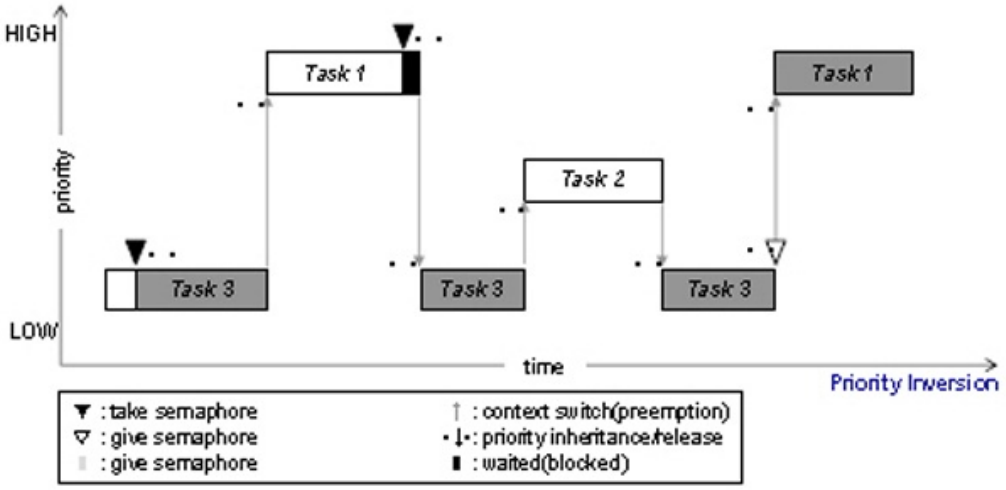
- 하나의 프로세스가 공유자원을 선점하였을 경우에 다른 프로세스들이 접근하지 못하도록 막고, 사용 후 공유자원에 대한 사용권을 풀어줌으로써 자원점유에 대한 중복을 막을 수 있게 함

- 공유자원의 사용권 수에 따라 하나일 경우 Binary Semaphore, 둘 이상일 경우 Counting Semaphore 사용함

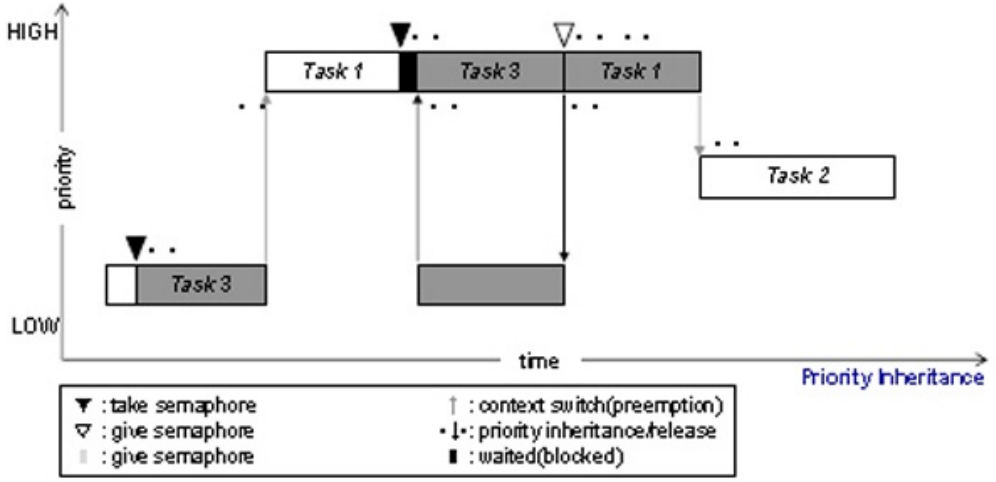
- 각 Semaphore들은 Queue를 이용한 매크로 함수를 사용하여 구현됨

2-1. Binary Semaphore VS Mutex

- **Mutex와 Binary Semaphore의 차이는 Mutex는 Priority 상속 기능을 갖고 있다는 것**
- **Priority 상속개념은 Priority inversion문제를 최소화함(완벽하게 없애지는 못함)**
- **Priority 상속은** 현재 Mutex를 Take 하고 있는 Task3보다 더 높은 Priority를 갖는 Task1가 동일한 Mutex를 Take 하려고 시도 하는 경우에 현재 실행 중인 Task3의 Priority를 임시적으로 Mutex를 Take하려고 시도하는 Task 중에서 가장 높은 Priority를 갖는 Task1와 동일하게 높은 Priority를 부여하는 것으로 현재 Mutex를 갖고 있는 Task3가 Mutex를 Give하는 경우 이 Task3는 원래 자신이 갖고 있는 Priority를 갖게 됨



Priority Inversion 문제



Priority 상속

2-2. Critical Section VS Semaphore/Mutex

Critical Section:
- 커널 객체가 아니고 유저 객체로 user mode에서만 동작, 가볍고 빠르다는 장점이 있음
- 인터럽트 까지 금지 해당 Task만 동작
- 인터럽트 금지하니 우선순위 스케줄링에 영향 줄 수 있으므로 가볍고 빠른 작업에 쓰임
- 한 Process 안의 스레드 사이에서만 동작함(프로세스 사이는 동작 x)

Semaphore/Mutex:
- 다중 프로세스의 스레드 사이에서 동기화 가능
- 사용하는 코드가 길거나 다중 프로세스 사이에서 공유하거나
  동시에 n개 접근을 허용해야 하거나 할 때 사용
- 커널 객체이므로 크리티컬 섹션보다 무거움

감사합니다.