Process Syn

2022년 4월 25일 월요일 오후 4:13

Critical Section Problem

Race condition : 공유 메모리 모델에서 한정된 공유 데이터들을 두고 프로세스/스레드가 동시에 접근하여 사용하는것이 race같다고 붙여진 이름

하나의 공우 자원을 동시에 접근하게 되면 하나로 일관적이지 않다 그래서 **동기화**가 필요

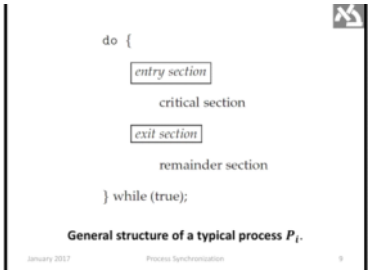
문제: 다수의 프로세스들끼리 데이터가 공유되는 경우 프로세스들 간 임계 구역의 이용 순서는 반드시 동기화 되야 함
프로세스들간의 동기화는 어떻게 적절하게 구현 해야 하는가?에 대한 문제

즉 일관성 유지 하기위해 어떻게 해야하는가

Critical Section

한 프로세스 내에서 타 프로세스들과 공유하는 영역(데이터,변수등)에 접근하는 코드상의 영역

해결하기 위한 기본 뼈대



화면 캡처: 2022-04-25 오후 7:51

Critical section -> cs로 줄여서 사용할께요

입장 구역(Entry section) : cs 진입전 진입에 대한 요청을 하고 허가가 날때까지 대기하는 코드 상의 영역

나머지 구역(remainder section) : 입장,퇴장 ,임계구역들을 제외한 모든 코드상의 영역

퇴장 구역(Exit section) : 임계구역을 빠져 나왔음을 알리는 코드상의 영역

Critical section problem 해결 방안 반드시 만족 되야하는 특성

1. **Mutal Exclusion**
   a. 하나의 프로세스가 임계구역에 진입 중이라면 해당 프로세스와 데이터를 공유하는 다른 프로세스들(협동 관계)은 임계구역 진입 불가 하게 하는거
   b. Non Sharable공유 자원에 한해서만 존재
      i. -> write only처럼 둘이상의 스레드 /프로세스가 동시에 접근할수없다 (상호배타성이 필요하다)
   c. Read only 같은 경우 스레드 /프로세스가 동시에 접근해도 무방하므로 sharable공유 자원이다 (상호배타성 X)
2. **Progress**
   a. 임계 구역에 진입중인 프로세스가 없다면 그 즉시 입장 구역에서 대기중인 프로세스들 중 하나를 적절히 선택해서 임계구역에 진입하도록 해야한다 -> 이 선택과정은 지연 되어서는 안됨!
   b. 즉 퇴장한다면 그 즉시 다른 프로세스/스레드가 임계구역(CS)에 진입 해야한다 만약 퇴장 했는데 아무도 진입하지 않는다면 -> progress 즉 진행성을 위반한 것이다
3. **Bounded Waiting (한정된 대기 시간)**
   a. 모든 협동관계 프로세스들은 임계구역에 한정된 시간 내에 진입해야한다 즉 하나의 프로세스가 무한한 시간동안 임계구역에 진입하는 경우가 발생해서는 안된다. 이를 위해 한번 임계구역에 진입하고 퇴장한 프로세스는 다음번 진입시 적절히 진입에 제한을 둔다
      ==> 즉 기근 현상이 발생해서는 안되는다는 의미이다

커널 영역에서 발생 가능한 임계구역 문제

오히려 interrupt막아놓으면 악영향 끼친다 -> 하나의 인터럽트 막아 놓는다 해서 임계구역 문제가 해결되는것은 아니다
-> 멀티 CPU 코어 시스템에서는 말 그대로 명령어들이 동시에 실행 되기에 한 CPU 코어에서 인터럽트를 막음으로서 임계구역에서 데이터를 수정하던 도중 cs를 강제로 빠져나오게 되는 일은 막을수있더라도 여전히 다른 CPU 코어에서는 cs접근 가능하기 때문이다. 그리고 인터럽트를 한번에 막았다가 다시 푸는것도 시간 낭비가 크다

2가지 해경 방안

Nonpreemptive kernel

- 한번 프로세스가 시스템 호출 등 그 외에도 많은 인터럽트를 통해 커널 모드로 진입하면 해당 프로세스가 ISR수행을 마치고 커널 모드에서 나올때까지 계속 수행
- 현 시점에 하나의 프로세스만이 커널 영역에서 실행 가능

- 커널 영역에서의 cs문제를 완전히 해소 가능
- 그러나 시스템 동작 및 반응속도는 처참하다

Preemptive kernel

- 프로세스가 커널 모드에서 실행중이더라도 인터럽트가 발행한 다른 프로세스에 의해 선점될수있는 커널
- 도중에 다른 ISR 실행 가능

소프트웨어적으로 처리한 대표적인 알고리즘

Lamport's Bakery Algorithm

각각의 같은 데이터들을 공유하는 프로세스/스레드들에게 먼저 cs에 대한 진입요청을 한 순서대로 일종의 번호를 부여하고 그 번호가 가장 낮은(가장 우선순위가 높은) 프로세스/스레드 부터 cs에 진입시키는 알고리즘
마치 빵집에 찾아온 순서대로 번호표 부여하여 번호가 가장 빠른 사람부터 입장토록하는 모습과 비슷하다
만약 동일한 번호 부여 받았다면 pid가 더 작은 것을 먼저 cs에 진입시키도록 한다

하드웨어로 처리하기

Mutex

가상의 잠금장치인 lock을 두어 cs에 들어가기 전 해당 lock획득하는 연산을 통해 lock을 획득하는데 성공하면 cs에 진입lock을 획득하지 못하면 대기 그리고 cs에서 최장하는 프로세스 / 스레드가 lock을 다른 협동 프로세스/스레드에 양보토록 하여 cs 문제를 해결하는 기법
Lock과 해당 lock을 획득하는 연산 lock을 양도하는 연산 이 3가지가 필요 하다

절대 둘 이상의 프로세스 /스레드가 동시에 실행할수없는 원자적 연산이여한다 (atomic 명령어)
즉 lock을 획득하는 함수는 그 내부가 반드시 CPU에서 제공하는 원자적 명령어(atomic 명령어)로 구성된다

Atomic 명령어

- 실행 중간에 간섭받거나 중단 되지 않는다
- 같은 메모리 영역에 대해 동시에 실행되지 않는다
- Context switching 일어나지 않음을 보장한다

Lock을 하는 함수가 atomic 명령어로 이루어져 있기 때문에 두개 이상의 스레드가 동시에 진행이 되어도
CPU의 도움을 받아서 먼저 하나를 실행시키고 다음을 시킨다 즉 동기화 시켜서 명령어 수행하게 한다 -> 둘다 동시에 수행할수없게 해준다

Priority inheritance 속성

P2가 P1보다 우선순위가 높음에도 불구하고 lock()을 가지고있지 않아서 끝날때까지 기다려야하는 의존성 문제가 발생한다. 그래서 P2의 우선순위를 P1 만큼 올려서 p2를 실행시키는것을 priority inheritace라고 한다

Lock을 획득할 수있는지 계속 확인한다 -> spin lock -> CPU 명령어 사이클 낭비

Pros

- 멀티 코어 환경이고 만약 cs이용하는 평균적인 시간이 짧다면 오히려 lock획득을 위해 계속 while돌게하는것이 context swtiching발생시키지 않으므로 이와 같은 경우에는 전체적인 오버헤드가 감소한다

Cons

- Busy wating
  - 해당 lock을 획득하기를 기다리는 프로세스들은 무한정 while돌게 되는데 이로 인해 분명히 lock획득을 위해 기다리는 프로세스 /스레드들은 cs진입을 위해 대기중인 상태임에도 불구하고 계속 실행 상태인 기이한 현상
  - 즉 대기하는데 실제로는 CPU에서 실행중 -> CPU 명령어 사이클 낭비 된다
  - Spin lock 구조보다 개선 한 mutex구조라면 waiting상태있다가 wake up하는 방식을 사용하는데
  - 이때 context switching발생한다

```
while (true) {
    acquire lock
        critical section
    release lock
        remainder section
}
```

화면 캡처: 2022-04-25 오후 9:32

화장실이 하나 뿐이 없는 식당과 비슷하다 화장실을 가기위해서는 key를 받아야한다
내가 화장실에 있다면 다른 사람이 아무리 급해도 열쇠가 없으므로 화장실에 들어올수없다.

semaphore

Signal mechanism을 가진 하나 이상의 프로세스/스레드가 critical section에 접근 가능하도록 하는 장치 이다

Mutex 보다 더 복잡 하지만 견고한 방식으로 작동하는 동기화 기법 이다.

대기, 신호 발생 연산 각각은 원자적 연산으로 구현됨
하드웨어에서 제공하는 원자적 명령어를 활용하여 구현 한다

멀티 프로그래밍 환경에서 공유된 자원에 대한 접근을 제한하는 방법이다

작업간의 실행 순서를 동기화 해야한다면 세마포어를 사용하자

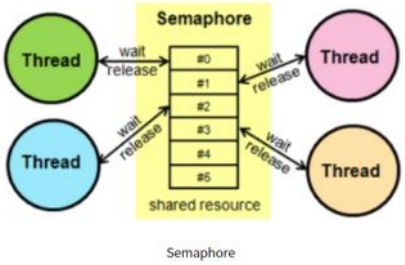
S : 0과 양의 정수값만 가진다 두개의 atomic operation로만 조작이 된다 -> wait,signal 또는 P,V

S는 1로 초기화 되있다

P()[wait()]
세마 포어 값을 감소 시킴
현대 해당 대기 연산을 실행중인프로세스/스레드가 공유 데이터에 접근하겠다는 것을 의미한다

세마포어 값이 0이하라면 해당 프로세스 /스레드는 대기(waiting)상태로 전환되며 해당 프로세스/스레드는 해당 세마포어의 대기 큐에 넣어진다

V() [signal]: 퇴장 구역에서 실행되는 연산으로 현재 세마포어의 값을 1증가



Semaphore

화면 캡처: 2022-04-25 오후 9:49

계수 세마포어(counting semaphore)

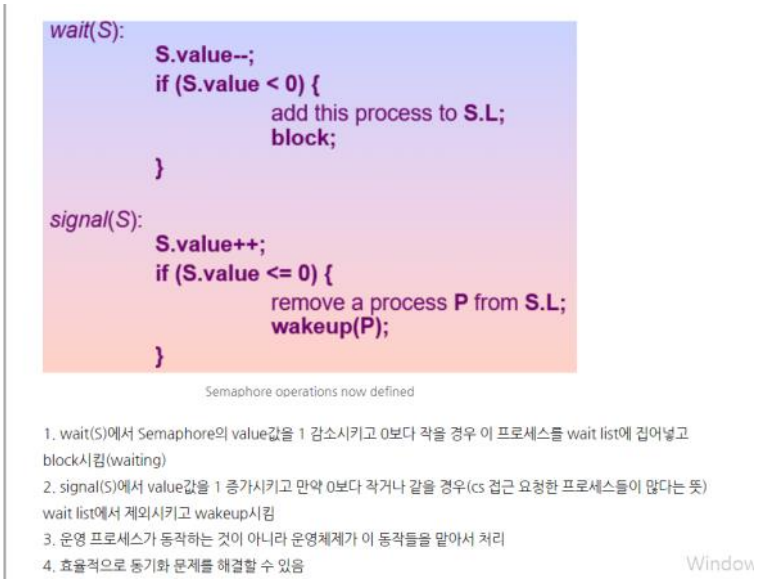
- 세마 포어가 가질수있는 값이 무한정이다
- 여기서 세마포어는 공유되는 자원들의 개수로 초기화 된다

이진 세마포어(binary semaphore)

- 0 혹은 1의 값만 가진다 (시용가능한 자원의 개수를 최대 1개로 간주)
- Mutex와의 차이점
  - Mutex는 lock을 가진애만 unlock할수있다 세마포는 그렇지 않다

공유자원에 접근할수있는 프로세스의 최대 허용치 만큼 동시에 사용자가 접근할수있다
각 프로세스는 세마포어의 값을 확인하고 변경할수있다

Cs에 한개 이상의 프로세스/스레드가 접근하는데 이거 맞나? -> 세마포어는 예외적이라고 생각하면 될 듯



Semaphore operations now defined

1. wait(S)에서 Semaphore의 value값을 1 감소시키고 0보다 작을 경우 이 프로세스를 wait list에 집어넣고 block시킴(waiting)
2. signal(S)에서 value값을 1 증가시키고 만약 0보다 작거나 같을 경우(cs 접근 요청한 프로세스들이 많다는 뜻) wait list에서 제외시키고 wakeup시킴
3. 운영 프로세스가 동작하는 것이 아니라 운영체제가 이 동작들을 맡아서 처리
4. 효율적으로 동기화 문제를 해결할 수 있음

Window

화면 캡처: 2022-04-25 오후 9:59

-> busy waiting으로 인해 cpu사이클이 낭비되는것을 막기 위해 위와 같은 순서로 진행이 된다

화장실 여러칸있다 화장실 입구에는 비어있는 화장실 칸 수를 확인 할 수있다
내가 화장실을 가고 싶다면 화장실 빈칸수가 1개 이상일때 한 개를 빼고 들어갈수있다
나올때는 하나를 더해주고 나온다
만약 0이면 양수가 될때까지 wait한다

화장실 -> 공유자원
사람들 -> 스레드,프로세스
화장실 빈칸의 개수 -> 현재 공유자원에 접근할수있는 스레드 , 프로세스의 개수

Mutex와 Semaphore차이

1. 동기화 대상의 개수
   a. Mutex -> 1개
   b. Semaphore -> 1개 이상

2. Mutex는 자원소유 가능 그러나 semaphore는 자원 소유 불가

Monitor

협동 프로세스 , 스레드들 간에 공유되는 데이터와 그에 대한 접근 함수 그리고 유한개의 조건 변수 그리고 해당 조건 변수에 대한 2가지 연산으로 구성되는 추상적 자료 구조이다

언제 사용?

- 한번에 하나의 스레드만 실행되야할때
- 여러 스레드와 협업이 필요할때