병행제어 1

멀티 프로세서 스케줄링

CPU가 여러 개인 경우 스케줄링은 더욱 복잡해진다.

상황

- Homogeneous processor 인 경우
  -
- Load sharing
- Symmetric Multiprocessing
  - 여러 cpu가 대등하게 일을함
- Asymmetric multiprocessing
  - cpu 여러개 중에 하나의 cpu가 대장이 되어 나머지 cpu가 거기에 따르는 것

RealTime Scheduling

Hard real-time systems

- 정해진 시간에 반드시 끝내도록 스케줄링 해야함으로 offline으로 sheduling

Soft real-time systems

- ex) 동영상 스트리밍
- 일반 프로세스에 비해 높은 priority를 갖도록 해야함

Thread Scheduling

설명 : user level thread 인지 kernel level thread 인지에 따라 구분

Local Scheduling

- 사용자 수준 thread library에 의해 어떤 thread를 스케줄할지 결정

Global Scheduling

- 커널의 단기 스케줄러가 어떤 thread를 스케줄할지 결정

CPU 스케줄링 평가 방법

1. Queueing models
    - arrival rate(CPU를 쓰고자 하는 요청들이 큐에 도착률)
    - service rate(단위시간 당 처리율)
    - 이것들의 확률 분포를 통해 수식으로 계산
2. Implementation & Measurement (구현과 성능 측정)
    - 실제 시스템에 알고리즘을 구현하여 실제 작업에 대해 성능을 측정하여 비교한다.
3. Simulation (모의 실험)
    - 알고리즘을 모의 프로그램으로 작성 후 해당 작업을 입력하여 결과를 비교해본다.
    - 여러 경우 상황에 대해 입력을 해보면서 결과를 비교해보는 것이 보다 정확함

Process Synchronize(프로세스 동기화)

동기화 필요성

💡 동기화 필요성 예시

A, B 라는 두 명의 사람이 공유 계좌를 사용하고 있다고 가정

현재 잔고가 2만원인 계좌에 동시에 만원 씩 입금하는 상황
동기화를 하지 않으면 A 입금 후 계좌의 잔고는 3만원, B 입금 후 계좌의 잔고는 3만원

총 입금 금액은 A, B 2만원으로 입금 후 계좌는 4만원이 되어야 하지만 동기화가 되지 않아 생기는 문제

설명 : 동기화의 필요성

운영체제가 끼어드는 경우 시스템 콜을 통해 커널의 데이터를 공유하는 상황에서 컨텍스트 스위칭이 발생하는 경우

공유자원에 대해 동시에 접근하는 경우에 데이터의 불일치 문제의 발생

⇒ 실행 순서를 정해주는 매커니즘 필요하다.

동기화 구현

동기화 : 한 프로세스 혹은 스레드가 공유 데이터에 대한 접근을 마칠 때 까지 다른 프로세스 혹은 스레드가 접근하지 못하도록 제어하는 것

키워드

- 임계구역 : 동시성 문제가 발생하는 코드 영역
- 상호배제 : 임계구역에서 (임계구역 시작 ~ 끝) 하나의 프로세스 혹은 스레드만 독점적으로 실행하도록 관리하는 것

⇒ 즉, 임계구역에서 상호배제하는 것이 동기화 방법

상호배제 구현

목표 : 임계 구역에 오직 한 프로세스 혹은 스레드만 들어가게 하는 것

- 방법
  - Peterson’s 알고리즘

- 인터럽트 금지, 원자 명령

인터럽트 금지

- 임계구역 진입 시 인터럽트 발생하면 상호배제 위반
- 임계구역 진입 시 인터럽트 금지 명령어 실행
- 문제점
  - 임계구역을 실행 시 인터럽트를 막아도 되는가에 대한 문제
  - 멀티 코어 환경에서 인터럽트를 금지한다고 해도 다른 코어의 인터럽트까지 막을 수 있는지에 대한 문제

원자 명령

- lock 0 or 1
  - 임계구역에 들어갈 때 lock 1, 임계구역에 나올 때 lock 0
  - lock 0인 경우만 임계구역에 들어갈 수 있도록 허용
  - 문제점 : lock 변수가 0일 때 lock 변수를 읽고 lock 변수를 1로 바꾸는 과정에서 컨텍스트 스위칭 발생 시 lock 변수는 아직 0이므로 다른 프로세스 혹은 스레드가 임계구역을 실행하여 상호배제에 위배되는 상황이 발생된다.
  - 해결방법 : lock 변수를 읽음과 동시에 lock 변수를 1로 만드는 원자 명령 필요
- 원자명령
  - lock 변수를 읽음과 동시에 lock 변수에 1 저장
  - 두 가지 명령 사이에 컨텍스트 스위칭이 발생하지 않도록 하나의 명령으로 합침

동기화 기법

설명 : 동기화 기법에는 뮤텍스, 스핀락, 세마포 방식이 존재한다.

뮤텍스(Mutex)

- 잠김, 열림의 상태를 가진 락 변수를 이용해서 한 스레드만 임계구역에 진입시키고 다른 스레드는 큐에 대기시킴 (sleep waiting lock)
- 구성요소
  - 락 변수 → true(락 잠그기), false(락 열기)

    - 대기 큐
- 특징
    - 임계구역이 짧은 경우에 비효율적이다.
    - 락이 잠겨있는 경우에 cpu를 빼앗고, 락이 열려있는 경우 cpu를 할당
    - 락이 잠겨있는 시간보다 스레드가 잠자고 깨는데 걸리는 시간낭비 존재

스핀락

- 잠김, 열림 상태의 락 변수를 이용하여 락이 잠겨있으면 락이 풀릴때까지 무한루프를 돌면서 락을 검사 (busy waiting lock)
- 구성요소
    - 락 변수
- 특징
    - 임계구역이 실행시간이 짧은 경우 효율적
    - (락이 열릴때까지 검사하는 것 또한 CPU가 처리하기에) 단일 CPU 혹은 단일 코어를 가진 운영체제에서는 비효율적, CPU 낭비

뮤텍스와 스핀락 차이 비유

식당에서 한칸짜리 화장실을 이용하는데 대기 인원은 화장실에서 대기하는 것이 아닌 자신의 자리에서 대기하며 직원이 명단을 관리하며 직원의 안내에 따라 대기 순서대로 화장실을 사용하는 것이 세마포
화장실을 사용하는 시간이 긴 경우에는 세마포 방식이 체계적일 듯

위의 상황에서 직원의 안내에 따르는 것이 아닌 화장실 앞에서 서성거리며 나왔는지를 확인하면서 대기하는 것이 스핀락
화장실을 사용하는 시간이 짧은 경우에 스핀락 방식이 더 나을 듯

	뮤텍스	스핀락
대기 큐	있음	없음
block 가능여부	락이 잠겨 있으면 블록됨(blocking)	락이 잠겨 있으도 블록되지 않고 계속 락 검사(non-blocking)
lock/unlock 연산 비용	싸다	CPU를 계속 사용하므로 비싸다.
하드웨어 관련	단일 CPU에서 적합	멀티코어 CPU에서 적합
주 사용처	사용자 모드, 사용자 응용 프로그램	커널 모드, 커널 코드, 인터럽트 서비스 루틴

세마포

- 1개 이상의 여러개의 공유 자원을 다수의 스레드가 공유하여 사용하도록 돕는 **자원 관리 기법**
- 관리하는 자원의 수에 따라 세마포는 나뉜다.
  - 이진 세마포 : 관리하는 자원이 1개인 경우 → 뮤텍스와 유사하다.
  - 카운터 세마포 : 자원이 여러 개인 경우
- 구성요소
  - 자원의 개수
  - 대기 큐
  - counter 변수 : 사용 가능한 자원의 개수
    - 음수이면 대기 중인 스레드 수
  - P/V 연산
    - P - wait : 자원 요청 시 실행하는 연산
      - 사용가능 자원의 수를 1감소
    - V - signal : 자원 반환 시 실행하는 연산
      - 사용가능 자원 수 1증가
- 문제점
  - 데드락, 기아 발생 가능성
  - 우선순위 역전 현상