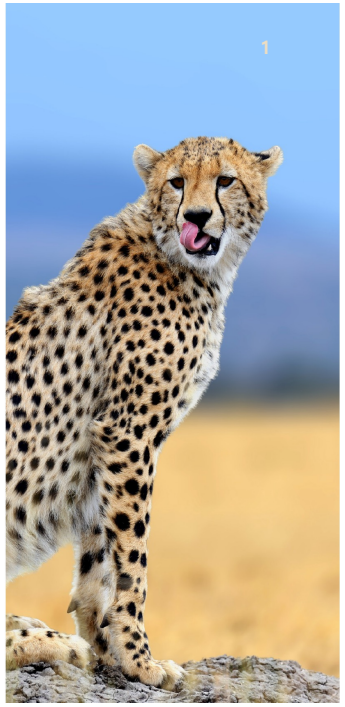
Chapter
05

CPU 스케줄링



강의 목표

1. CPU 스케줄링의 목표와 다양한 평가 기준에 대해 이해한다.
2. CPU 스케줄링이 실행되는 상황과 커널에서 CPU 스케줄링이 일어나는 위치에 대해 이해한다.
3. 선점 스케줄링과 비선점 스케줄링을 구분할 수 있다.
4. 기아와 에이징의 개념을 이해한다.
5. 다양한 CPU 스케줄링 알고리즘에 대해 안다.
   - FCFS, SJF, SRTF, RR, Priority, MLQ, MLFQ
6. (부록)리눅스에서 사용되는 CFS 스케줄링을 구체적으로 이해한다.

3 1. CPU 스케줄링 개요

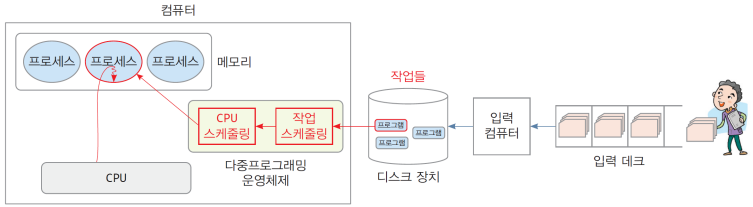
운영체제에서 일어나는 다양한 스케줄링

- 스케줄링은 왜 필요할까?
  - 자원에 대한 경쟁이 있는 곳에서 경쟁자 중 하나 선택
  - 자원 : CPU, 디스크, 프린트, 파일, 데이터베이스 등
  - 컴퓨터 시스템 여러 곳에서 발생
- 컴퓨터 시스템 내 다양한 스케줄링
  - 작업(job) 스케줄링
    - 배치시스템에서
    - 대기중인 배치 작업(Job) 중 메모리에 적재할 작업 결정
  - CPU 스케줄링
    - 프로세스/스레드 중에 하나를 선택하여 CPU 할당
    - 오늘날 CPU 스케줄링은 스레드 스케줄링
  - 디스크 스케줄링
    - 디스크 장치 내에서
    - 디스크 입출력 요청 중 하나 선택
  - 프린터 스케줄링
    - 프린팅 작업 중 하나 선택하여 프린터 할당

다중프로그래밍과 스케줄링

5

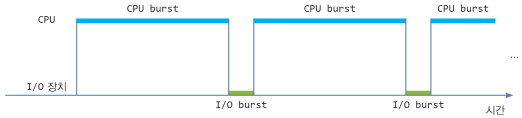
- 다중프로그래밍의 도입 목적 리뷰
  - CPU 유휴 시간 줄여 CPU 활용률 향상 목적
    - 프로세스가 I/O를 요청하면 다른 프로세스에게 CPU 할당
- 다중프로그래밍과 함께 2가지 스케줄링 도입
  - 작업 스케줄링(job scheduling)
    - 디스크 장치로부터 메모리에 올릴 작업 선택(초기 다중프로그래밍 시스템에서)
    - 처음에 혹은 프로세스가 종료할 때마다
  - CPU 스케줄링(CPU scheduling)
    - 메모리에 적재된 작업 중 CPU에 실행시킬 프로세스 선택



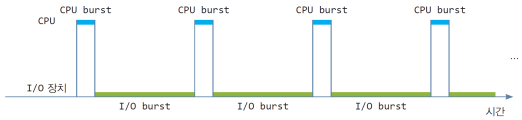
CPU burst와 I/O burst

6

- 프로그램의 실행 특성
  - CPU 연산 작업과 I/O 작업(화면 출력, 키보드, 입력, 파일 입출력 등)이 순차적으로 섞여 있음
  - CPU-burst – I/O burst – CPU-burst – I/O burst의 반복 ...
- CPU burst
  - 프로그램 실행 중 CPU 연산(계산 작업)이 연속적으로 실행되는 상황
- I/O burst
  - 프로그램 실행 중 I/O 장치의 입출력이 이루어지는 상황



(a) CPU 집중 프로세스의 실행 특성



(b) I/O 집중 프로세스의 실행 특성

CPU 스케줄링의 정의와 목표

7

- CPU 스케줄링
  - 정의
    - 실행 준비 상태(Ready)의 스레드 중 하나를 선택하는 과정
  - 기본 목표
    - CPU 활용률 향상 -> 컴퓨터 시스템 처리율 향상
  - 컴퓨터 시스템에 따라 CPU 스케줄링의 목표가 다를 수 있다.

CPU 스케줄링의 기준(criteria)

8

- 스케줄링 알고리즘의 다양한 목표와 평가 기준
  - CPU 활용률(CPU utilization)
    - 전체 시간 중 CPU의 사용 시간 비율, 운영체제 입장
  - 처리율(throughput)
    - 단위 시간당 처리하는 스레드 개수, 운영체제 입장
  - 공평성(fairness)
    - CPU를 스레드들에게 공평하게 배분, 사용자 입장
    - 시분할로 스케줄링
    - 무한정 대기하는 기아 스레드(starving thread)가 생기지 않도록 스케줄
  - 응답시간(response time)
    - 대화식 사용자의 경우, 사용자에 대한 응답 시간, 사용자 입장
  - 대기시간(waiting time)
    - 스레드가 준비 큐에서 머무르는 시간, 운영체제와 사용자 입장
  - 소요 시간(turnaround time)
    - 프로세스(스레드)가 컴퓨터 시스템에 도착한 후(혹은 생성된 후) 완료될 때까지 걸린 시간, 사용자 입장
    - 배치 처리 시스템에서 주된 스케줄링의 기준
  - 시스템 정책(policy enforcement) 우선
    - 컴퓨터 시스템의 특별한 목적을 달성하기 위한 스케줄링, 운영체제 입장
    - 예) 실시간 시스템에서는 스레드가 완료 시한(deadline) 내에 이루어지도록 하는 정책
    - 예) 급여 시스템에서는 안전을 관리하는 스레드를 우선 실행하는 정책 등
  - 자원 활용률(resource efficiency)

타임 슬라이스

9

- 대부분 운영체제에서
  - 하나의 스레드가 너무 오래 CPU를 사용하도록 허용되지 않음

- 타임 슬라이스와 스케줄링
  - 타임 슬라이스(time slice)
    - 스케줄된 스레드에게 한 번 할당하는 CPU 시간
      - 스레드가 CPU 사용을 보장받는 시간
    - 커널이 스케줄을 단행하는 주기 시간
      - 타이머 인터럽트의 도움을 받아 타임 슬라이스 단위로 CPU 스케줄링
      - 현재 실행중인 스레드 강제 중단(preemption), 준비 리스트에 삽입
    - 타임 퀀텀(time quantum), 타임 슬롯(time slot)이라고도 함

10

2. CPU 스케줄링 기본

CPU 스케줄링이 실행되는 4가지 상황

11

- CPU 스케줄링은 언제 시행될까?

1. 스레드가 시스템 호출 끝에 I/O를 요청하여 블록될 때
   - 스레드를 블록 상태로 만들고 스케줄링
   - (CPU 활용률 향상 목적)
2. 스레드가 자발적으로 CPU를 반환할 때
   - yield() 시스템 호출 등을 통해 스레드가 자발적으로 CPU 반환
   - 커널은 현재 스레드를 준비 리스트에 넣고, 새로운 스레드 선택
   - (CPU의 자발적 양보)
3. 스레드의 타임 슬라이스가 소진되어 타이머 인터럽트 발생
   - (균등한 CPU 분배 목적)
4. 더 높은 순위의 스레드가 요청한 입출력 작업 완료, 인터럽트 발생
   - 현재 스레드를 강제 중단(preemption)시켜 준비 리스트에 넣고
   - 높은 순위의 스레드를 깨워 스케줄링
   - (우선순위를 지키기 위한 목적)

CPU 스케줄링과 디스패치(dispatch)

12

- CPU 스케줄링 코드의 위치와 실행 시점
  - 스케줄링 담당하는 커널 스레드나 프로세스가 있는가? 없다.

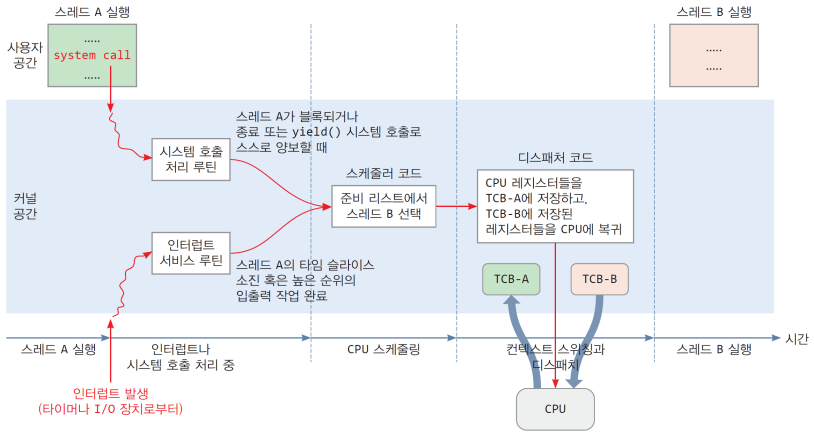
  1) 스케줄링 코드는 어디에 위치? 커널 내 함수로
     - 스케줄링 코드는 커널 코드의 일부로서 호출되어야 실행되는 함수 형태
     - 독립적으로 실행되는 프로세스나 스레드 아님

  2) 스케줄링 코드가 실행되는 시점
     - 시스템 호출이나 인터럽트 서비스 루틴이 끝나는 마지막 단계에서 실행

- 디스패처(dispatcher) 코드 실행
  - 디스패처 코드
    - 컨텍스트 스위칭을 실행하는 커널 코드
    - 스케줄러에 의해 선택된 스레드를 CPU가 실행하도록 하는 작업
    - 커널 모드에서 사용자 모드로 전환

- 스케줄러와 디스패처 모두 실행 시간이 짧도록 작성

CPU 스케줄링과 디스패치(dispatch)

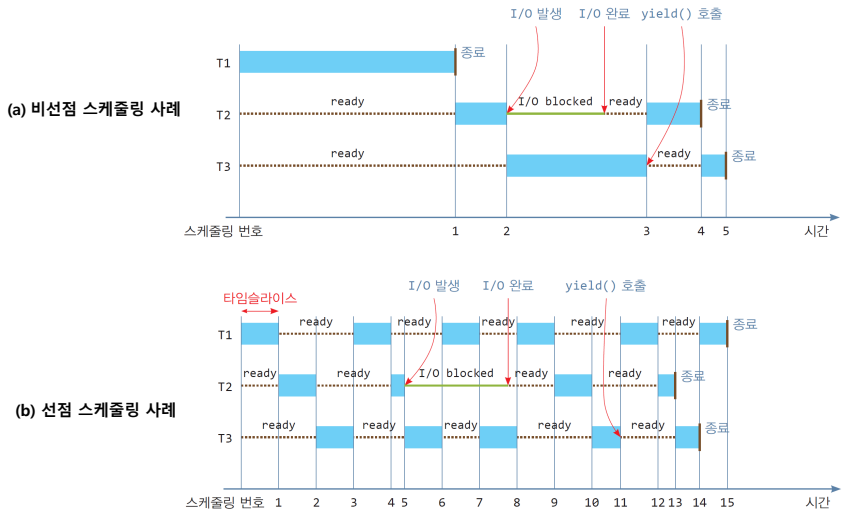


스레드 A에서 스레드 B로 스위칭 되는 사례

선점 스케줄링과 비선점 스케줄링

- 실행중인 스레드의 강제 중단 여부에 따른 CPU 스케줄링
  - **비선점 스케줄링**(non-preemptive scheduling) 타입
    - 현재 실행중인 스레드를 **강제로 중단시키지 않는** 타입
      - 스레드가 CPU를 할당받아 실행을 시작하면, 완료되거나 CPU를 더 이상 사용할 수 없는 상황이 될 때까지 스레드를 강제 중단시키지 않고 스케줄링도 하지 않는 방식
    - 스케줄링 시점
      - CPU를 더 이상 사용할 수 없게 된 경우 : I/O로 인한 블록 상태, sleep 등
      - 자발적으로 CPU 양보할 때
      - 종료할 때
  - **선점 스케줄링**(preemptive scheduling) 타입
    - 현재 실행중인 스레드를 **강제 중단**시키고 다른 스레드 선택
    - 스케줄링 시점
      - 타임슬라이스가 소진되어 타이머 인터럽트가 발생될 때
      - 인터럽트나 시스템 호출 종료 시점에서, 더 높은 순위의 스레드가 준비 상태일 때
- 오늘날
  - 일부 실시간 임베디드 시스템 운영체제 – 비선점 스케줄링
  - 그 외 대부분의 운영체제 - 선점 스케줄링

비선점 스케줄링과 선점 스케줄링 비교



기아와 에이징

- 기아(starvation)
  - 스레드가 스케줄링에서 선택되지 못한 채 오랜 동안 준비 리스트에 있는 상황
  - 사례
    - 우선순위를 기반으로 하는 시스템에서, 더 높은 순위의 스레드가 계속 시스템에 들어오는 경우
    - 짧은 스레드를 우선 실행시키는 시스템에서, 자신보다 짧은 스레드가 계속 도착하는 경우
  - 스케줄링 알고리즘 설계 시 기아 발생을 면밀히 평가
    - 기아가 발생하지 않도록 설계하는 것이 바람직함
- 에이징(aging)
  - 기아의 해결책
  - 스레드가 준비 리스트에 머무르는 시간에 비례하여 스케줄링 순위를 높이는 기법
    - 오래 기다릴 수는 있지만 언젠가는 가장 높은 순위에 도달하는 것 보장

17

3. CPU 스케줄링 알고리즘

기본적인 CPU 스케줄링 알고리즘들

18

- FCFS(First Come First Served)(비선점 스케줄링)
  - 도착한 순서대로 처리
- Shortest Job First(비선점 스케줄링)
  - 가장 짧은 스레드 우선 처리
- Shortest Remaining Time First(선점 스케줄링)
  - 남은 시간이 짧은 스레드가 준비 큐에 들어오면 이를 우선 처리
- Round-Robin(선점 스케줄링)
  - 스레드들을 돌아가면서 할당된 시간(타임 슬라이스)만큼 실행
- Priority Scheduling(선점/비선점 스케줄링 둘 다 구현 가능)
  - 우선 순위를 기반으로 하는 스케줄링. 가장 높은 순위의 스레드 먼저 실행
- Multilevel queue scheduling(선점/비선점 스케줄링 둘 다 구현 가능)
  - 스레드와 큐 모두 n개의 우선순위 레벨로 할당, 스레드는 자신의 레벨과 동일한 큐에 삽입
  - 높은 순위의 큐에서 스레드 스케줄링, 높은 순위의 큐가 빌 때 아래 순위의 큐에서 스케줄링
  - 스레드는 다른 큐로 이동하지 못함
  - 예) background process, Foreground process
- Multilevel feedback queue scheduling(선점/비선점 스케줄링 둘 다 구현 가능)
  - 큐만 n개의 우선순위 레벨을 둠. 스레드는 동일한 우선순위
  - 스레드는 제일 높은 순위의 큐에 진입하고 큐타임슬라이스가 다하면 아래 레벨의 큐로 이동
  - 낮은 레벨의 큐에 오래 있으면 높은 레벨의 큐로 이동

FCFS

19

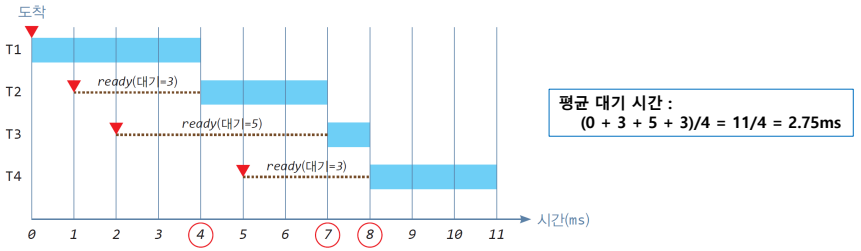
- 알고리즘
  - 선입선처리
    - 먼저 도착한(큐의 맨 앞에 있는) 스레드 먼저 스케줄링
- 스케줄링 파라미터 : 스레드 별 큐 도착 시간
- 스케줄링 타입 : 비선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생하지 않음
  - 스레드가 오류로 인해 무한 루프를 실행한다면, 뒤 스레드 기아 발생
- 성능 이슈
  - 처리율 낮음
  - 호위 효과(convoy effect) 발생
    - 긴 스레드가 CPU를 오래 사용하면, 늦게 도착한 짧은 스레드 오래 대기

FCFS 사례

20

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

실행 시간 동안 입출력은 발생하지 않는다고 가정한다.



평균 대기 시간 :
(0 + 3 + 5 + 3)/4 = 11/4 = 2.75ms

※ 빨간 원은 스케줄링이 일어나는 시점을 나타낸다.

SJF(Shortest Job First)

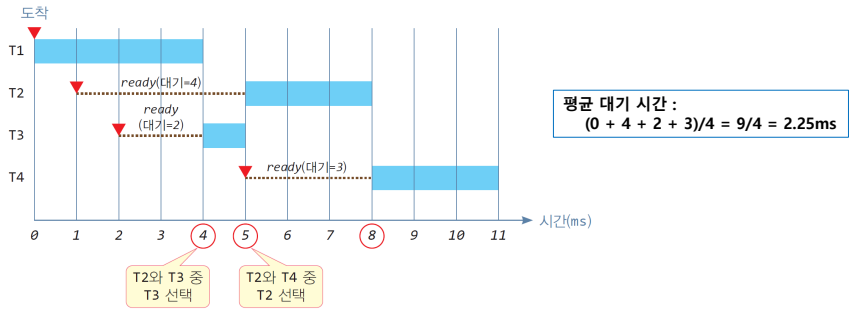
21

- 알고리즘
  - 최단 작업 우선 스케줄링
  - 실행 시간(예상 실행 시간)이 가장 짧은 스레드 선택
  - 스레드가 도착할 때,
    - 실행 시간이 짧은 순으로 큐 삽입, 큐의 맨 앞에 있는 스레드 선택
- 스케줄링 파라미터 : 스레드 별 예상 실행 시간
  - 스레드의 실행 시간을 아는 것은 불가능. 비현실적
- 스케줄링 타입 : 비선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생 가능
  - 짧은 스레드가 계속 도착하면, 긴 스레드는 실행 기회를 언제 얻을 지 예측할 수 없음
- 성능 이슈
  - 가장 짧은 스레드가 먼저 실행되므로 평균 대기 시간 최소화
- 문제점
  - 실행 시간의 예측이 불가능하므로 현실에서는 거의 사용되지 않음

SJF 사례

22

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3



평균 대기 시간 :
(0 + 4 + 2 + 3)/4 = 9/4 = 2.25ms

SRTF(Shortest Remaining Time First)

23

- 알고리즘
  - 최소 잔여 시간 우선 스케줄링
  - 남은 실행 시간이 가장 짧은 스레드 선택
  - SJF의 선점 스케줄링 버전
    - 한 스레드가 끝나거나 실행 시간이 더 짧은 스레드가 도착할 때, 남은 실행 시간이 가장 짧은 스레드 선택
    - 실행 시간에 짧은 순으로 스레드들을 큐에 삽입- > 큐 맨 앞에 있는 스레드 선택
- 스케줄링 파라미터 : 스레드 별 예상 실행 시간과 남은 실행 시간 값
  - 이 시간을 아는 것은 불가능. 비현실적
- 스케줄링 타입 : 선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생 가능
  - 짧은 스레드가 계속 도착하면, 긴 스레드는 실행 기회를 언제 얻을 지 모름
- 성능 이슈
  - 실행 시간이 가장 짧은 스레드가 먼저 실행되므로 평균 대기 시간 최소화
- 문제점
  - 실행 시간 예측이 불가능하므로 현실에서는 거의 사용되지 않음

SRTF 사례

24

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

평균 대기 시간 :
(1 + 4 + 0 + 3)/4 = 8/4 = 2ms

RR(Round-Robin)

25

- 알고리즘
  - 스레드들에게 공평한 실행 기회를 주기 위해
  - 큐에 대기중인 스레드들을 타임 슬라이스 주기로 돌아가면서 선택
  - 스레드는 도착하는 순서대로 큐에 삽입
  - 스레드가 타임 슬라이스를 소진하면 큐 끝으로 이동
- 스케줄링 파라미터 : 타임 슬라이스
- 스케줄링 타입 : 선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 없음
  - 스레드의 우선순위가 없고, 타임 슬라이스가 정해져 있어, 일정 시간 후에 스레드는 반드시 실행
- 성능 이슈
  - 공평하고, 기아 현상 없고, 구현이 쉬움
  - 잦은 스케줄링으로 전체 스케줄링 오버헤드 큼. 특히 타임 슬라이스가 작을 때 더욱 큼
  - 균형된 처리율 : 타임슬라이스가 크면 FCFS에 가까움, 적으면 SJF/SRTF에 가까움
    늦게 도착한 짧은 스레드는 FCFS보다 빨리 완료되고, 긴 스레드는 SJF보다 빨리 완료됨

RR 사례(타임 슬라이스=2ms일 때)

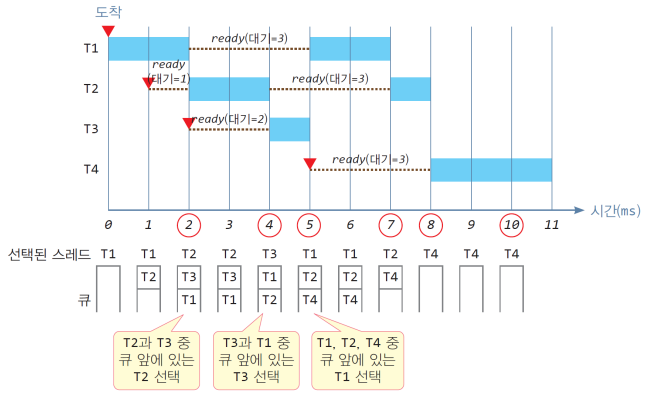
26

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

평균 대기 시간 : (3 + 4 + 2 + 3)/4 = 12/4 = 3ms

스케줄링 : 6번(2, 4, 5, 7, 8, 10ms 때)

컨텍스트 스위칭 : 5번 발생(2, 4, 5, 7, 8ms 때)



RR 사례(타임 슬라이스=1ms일 때)

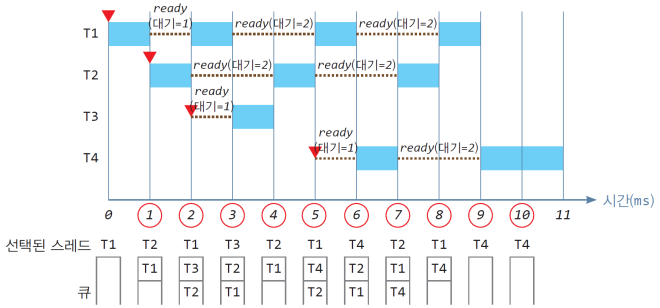
27

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

평균 대기 시간 : (5 + 4 + 1 + 3)/4 = 13/4 = 3.25ms

스케줄링 : 10번(1, 2, 3, 4, 5, 6, 7, 8, 9, 10ms 시점)

컨텍스트 스위칭 : 9번(1, 2, 3, 4, 5, 6, 7, 8, 9ms 시점)



Priority 스케줄링

28

- 알고리즘
  - 우선순위에 따라 스레드를 실행시키기 위한 목적
  - 가장 높은 순위의 스레드 선택
    - 현재 스레드가 종료되거나 더 높은 순위의 스레드가 도착할 때, 가장 높은 순위의 스레드 선택
  - 모든 스레드에 고정 우선순위 할당, 종료 때까지 바뀌지 않음
  - 도착하는 스레드는 우선순위 순으로 큐에 삽입
- 스케줄링 파라미터 : 스레드 별 고정 우선순위
- 스케줄링 타입 : 선점 스케줄링/비선점 스케줄링
  - 선점 스케줄링 : 더 높은 순위의 스레드가 도착할 때 현재 스레드 강제 중단하고 스케줄링
  - 비선점 스케줄링 : 현재 실행 중인 스레드가 종료될 때 스케줄링
- 스레드 우선순위 : 있음
- 기아 : 발생 가능
  - 높은 순위의 스레드가 계속 도착하는 경우, 실행 기회를 언제 얻을 지 예상할 수 없음
  - 큐 대기 시간에 비례하여 일시적으로 우선순위를 높이는 에이징 방법으로 해결 가능
- 성능 이슈
  - 높은 우선순위의 스레드일수록 대기 혹은 응답시간 짧음
- 특징
  - 스레드별 고정 우선순위를 가지는 실시간 시스템에서 사용

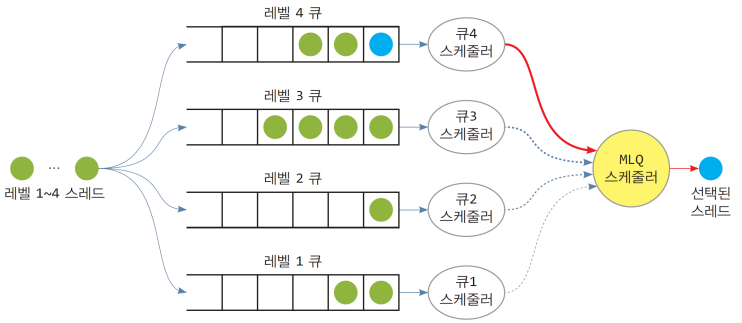
MLQ(Multi-level Queue) 스케줄링

29

- 설계 의도
  - 스레드들을 n개의 우선순위 레벨로 구분, 레벨이 높은 스레드를 우선 처리하는 목적
- 알고리즘
  - 고정된 n 개의 큐 사용, 각 큐에 고정 우선순위 할당
  - 스레드들의 우선순위도 n개의 레벨로 분류
  - 각 큐는 나름대로의 기법으로 스케줄링
  - 스레드는 도착 시 우선순위에 따라 해당 레벨 큐에 삽입. 다른 큐로 이동할 수 없음
  - 가장 높은 순위의 큐가 빌 때, 그 다음 순위의 큐에서 스케줄링
- 스케줄링 파라미터 : 스레드의 고정 우선순위
- 스케줄링 타입 : 비선점/선점 모두 가능
  - 비선점 스케줄링 : 현재 실행중인 스레드가 종료할 때 스케줄링
  - 선점 스케줄링 : 높은 레벨의 큐에 스레드가 도착하면 중단하고 높은 레벨 큐에서 스케줄링
- 스레드 우선순위 : 있음
- 기아 : 발생 가능
  - 높은 순위의 스레드가 계속 도착하는 경우 실행 기회를 언제 얻을 지 예상할 수 없음
- 성능 이슈와 활용 사례
  - 스레드의 고정 순위를 가진 시스템에서 활용
  - 예) 전체 스레드를 백그라운드 스레드와 포그라운드 스레드의 2개의 그룹으로 구성
  - 예) 시스템 스레드, 대화식 스레드, 배치 스레드 등 3개의 레벨로 나누고 시스템 스레드를 우선적으로 스케줄링
  - 예) 대학에서 교수, 교직원, 대학원생, 학부생 등 사용자를 4개의 레벨로 나누고, 사용자에 따라 실행시킨 스레드 레벨로 스케줄링

4개의 우선순위 레벨을 가진 시스템에서 MLQ 스케줄링

30



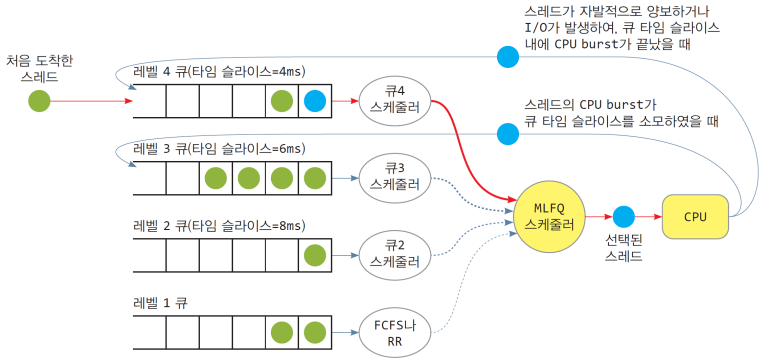
MLFQ(Multi-level Feedback Queue) 스케줄링

31

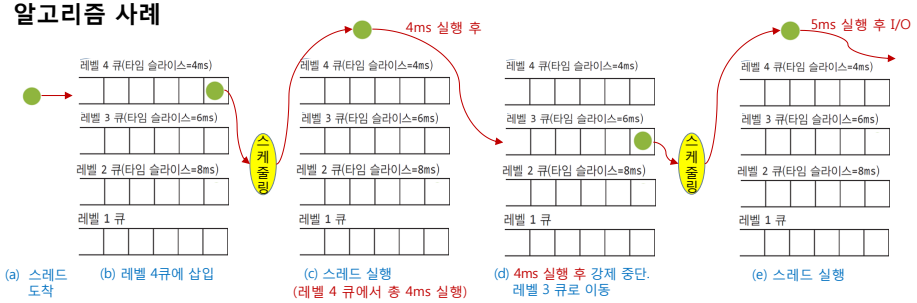
- 설계 의도
  - 1962년에 개발된 알고리즘
  - 기아를 없애기 위해 여러 레벨의 큐 사이에 스레드 이동 가능하도록 설계
  - 짧은 스레드와 I/O가 많은 스레드, 대화식 스레드의 우선 처리. 스레드 평균대기시간 줄임
- n개의 레벨 큐
  - n개의 고정 큐. 큐마다 우선순위 다름. 큐마다 서로 다른 스케줄링 알고리즘
  - 큐는 준비 상태(Ready 상태)의 스레드 저장
  - 큐마다 스레드가 머무를 수 있는 큐 타임슬라이스 있음. 낮은 레벨의 큐일수록 더 긴 타임 슬라이스
  - I/O 집중 스레드(대화식 스레드)는 높은 순위의 큐에 있을 가능성 높음
- 알고리즘
  - 스레드는 도착 시 최상위 레벨 큐에 삽입
  - 가장 높은 레벨 큐에서 스레드 선택. 비어 있으며 그 아래의 큐에서 스레드 선택
  - 스레드의 CPU-burst가 큐 타임 슬라이스를 초과하면 강제로 아래 큐로 이동시킴
  - 스레드가 자발적으로 중단한 경우, 현재 큐 끝에 삽입
  - 스레드가 I/O로 실행이 중단된 경우, I/O가 끝나면 동일 레벨 큐 끝에 삽입
  - 큐에 있는 시간이 오래되면 기아를 막기 위해 하나 위 레벨 큐로 이동
  - 최하위 레벨 큐는 주로 FCFS나 긴 타임 슬라이스의 RR로 스케줄. 스레드들은 다른 큐로 이동 못함
- 스케줄링 파라미터 : 각 큐의 큐 타임슬라이스
- 스케줄링 타입 : 선점 스케줄링
- 스레드 우선 순위 : 없음
- 기아 : 발생하지 않음, 큐에 대기하는 시간에 오래되면, 더 높은 레벨의 큐로 이동시킴(에이징 기법)
- 성능 이슈
  - 짧거나 입출력이 빈번한 스레드, 혹은 대화식 스레드를 높은 레벨의 큐에서 빨리 실행 -> CPU 활용률이 높음

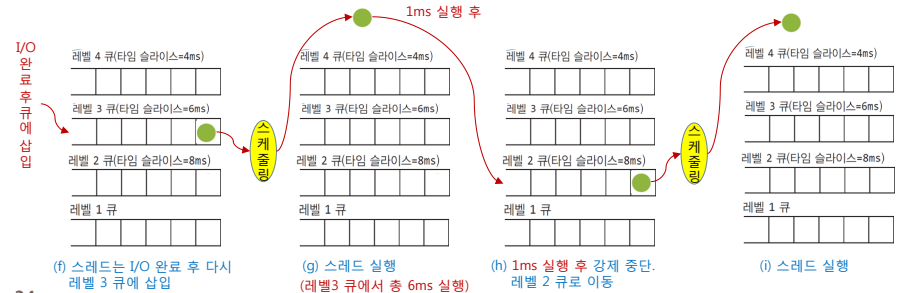
4개의 우선순위 레벨을 가진 MLFQ 스케줄링

32



알고리즘 사례



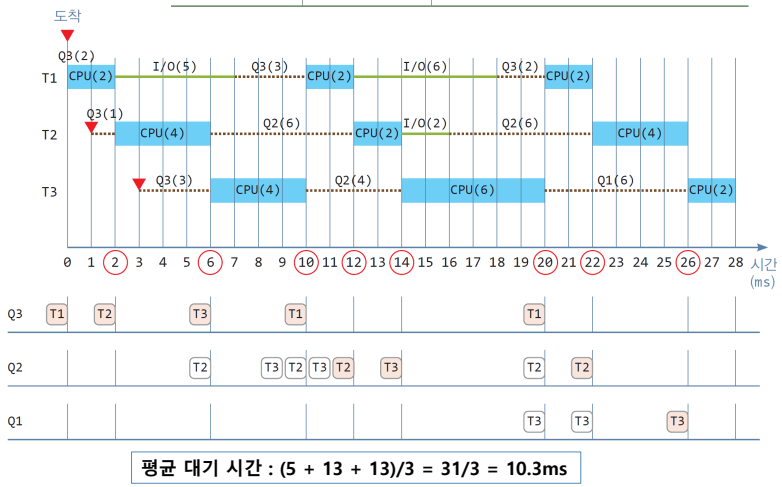


실전 스케줄링 사례

- 3개의 레벨 큐
- 3개의 스레드

큐 타임 슬라이스 : Q3=4ms, Q2=6ms, Q1=무제한

스레드	도착 시간	실행 시간(ms) (CPU-I/O-CPU-I/O-CPU)
T1	0	2 - 5 - 2 - 6 - 2
T2	1	6 - 2 - 4
T3	3	12

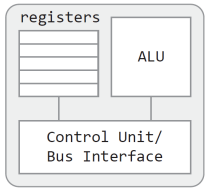


평균 대기 시간 : (5 + 13 + 13)/3 = 31/3 = 10.3ms

36 4.멀티 코어 CPU에서의 스케줄링

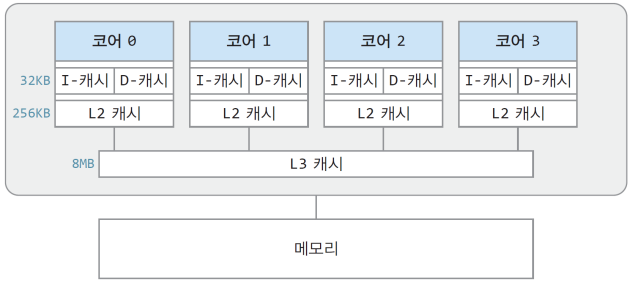
멀티 코어 시스템의 구조

코어 내부 구조



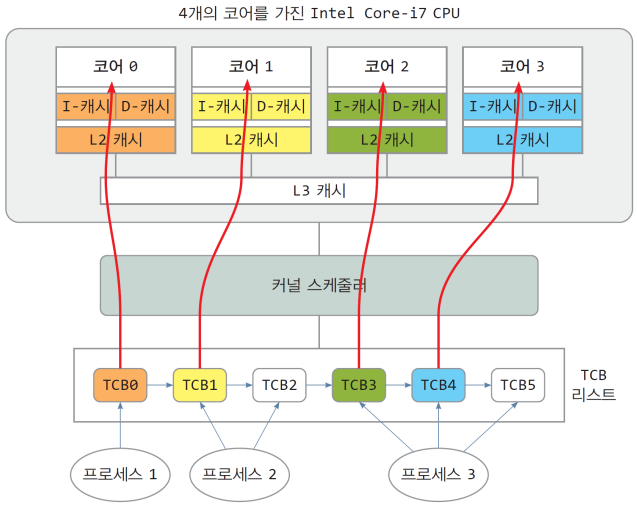
최초의 멀티 코어 CPU - 2001년 IBM에 의해 개발된 2개의 코어를 가진 PowerPC 칩(CPU)

4개의 코어를 가진 인텔 Core-i7 CPU



멀티 코어 시스템에서의 멀티스레딩

38



멀티코어 시스템에서의 CPU 스케줄링

39

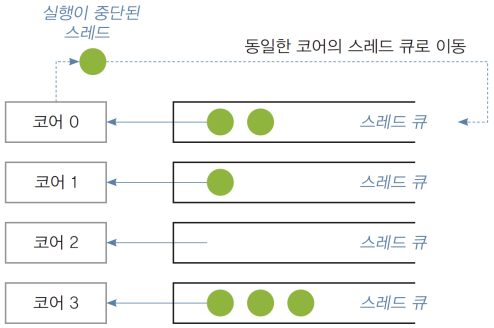
- 멀티코어 시스템에서 싱글 코어 CPU 스케줄링을 사용할 때 문제점
  - (문제 1) 컨텍스트 스위칭 후 오버헤드 문제
    - 이전에 실행된 적이 없는 코어에 스레드가 배치될 때,
    - 컨텍스트 스위칭 후, 실행 중에 새로운 스레드의 코드와 데이터가 캐시에 채워지는 긴 경과 시간
  - (문제 2) 코어별 부하 불균형 문제
    - 스레드를 무작위로 코어에 할당하면, 코어마다 처리할 스레드 수의 불균형 발생
- 컨텍스트 스위칭 후 오버헤드 문제 해결
  - CPU 친화성(CPU affinity) 적용
    - 스레드를 동일한 코어에서만 실행하도록 스케줄링
    - 코어 친화성(Core affinity), CPU 피닝(pinning), 캐시 친화성(cache affinity)라고도 부름
  - 코어 당 스레드 큐 사용
- 코어별 부하 불균형 문제 해결
  - 부하 균등화 기법으로 해결

  1) 푸시 마이그레이션(push migration) 기법
    - 감시 스레드가, 짧거나 빈 큐를 가진 코어에 다른 큐의 스레드를 옮겨놓는 기법

  2) 풀 마이그레이션(pull migration) 기법
    - 코어가 처리할 스레드가 없게 되면, 다른 코어의 스레드 큐에서 자신이 큐로 가져와 실행시키는 기법

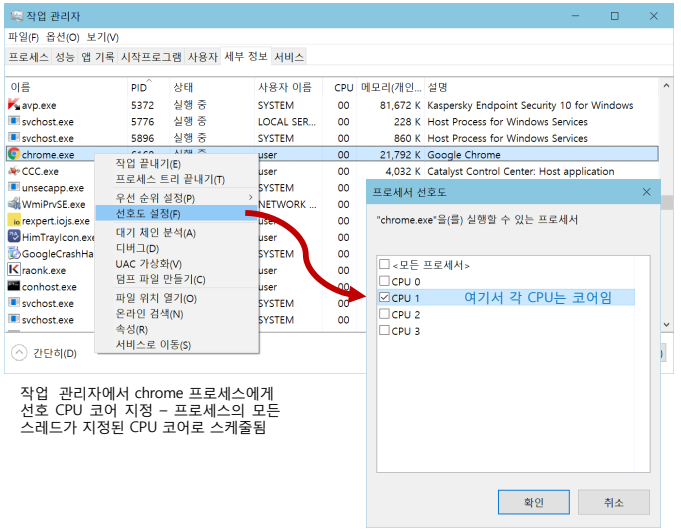
코어 당 스레드 큐

40



Windows 사례 : 작업 관리자에서 프로세스에게 코어 친화성 지정

41



작업 관리자에서 chrome 프로세스에게 선호 CPU 코어 지정 – 프로세스의 모든 스레드가 지정된 CPU 코어로 스케줄됨

chrome 프로세스를 코어 1에 할당