**[Chapter 5]**

소프트웨어융합학과/2020105742/한지훈

1. CPU-bound process, I/O-bound process
   A. I/O-bound process
      I. 정의: I/O burst time(프로세스가 I/O 처리 완료될 때까지 대기하는 시간)이 CPU burst time보다 긴 프로세스
      II. 예시: 유튜브, 게임 등(User과 상호작용이 많으며, 자주 사용하는 유형)
      III. 스케줄링 우선순위: 높음
         - 이유: 보통 디코딩 작업 등의 CPU burst보다 영상 data를 서버에서 받아오는 I/O burst이 훨씬 김
            ➢ CPU 자원 자주 못쓰면 안 되기 때문
   B. CPU-bound process
      I. 정의: CPU burst time(프로세스가 CPU를 사용하고 있는 시간)이 I/O burst time보다 긴 프로세스
      II. 예시: 시뮬레이션 코드(실행 알아서 하고 완료되면 결과 파일 txt/csv 등으로 출력) 등
      III. 스케줄링 우선순위: 낮음
         - 이유: I/O 요청 없이 연산만 진행하기 때문에 time quantum 가득 채워서 사용하면 됨
            ➢ 결국 사용자와 주고받는 동작이 없기 때문에 백그라운드에서 돌려도 됨
   C. 구분 이유: 프로세스마다 CPU 사용 시간이 다르기 때문에 스케줄링 정책 달라질 수 있음

2. Dispatcher
   A. Dispatch: 실행되어야 할(스케줄링 된) 프로세스의 실행되기 위한 PCB 정보를 CPU에 restore하는 것
      I. 다른 이름: switching context, switching to user mode
   B. Dispatch latency
      I. 정의: 커널 모드로 전환해서 기존state를 PCB에 저장(save)하고 새 PCB restore(dispatch)하기까지 걸리는 시간
      II. 특징: 스케줄링 시간을 포함하는 개념이며, 커널 오버헤드라고 할 수도 있음
   C. Scheduling vs. Dispatch
      I. Scheduling: 좀 더 이론적인 개념(ready 큐 프로세스 중 CPU에 누구를 올릴 것인지)
      II. Dispatch: Scheduling을 실현시키기 위한 실질적인 방법

3. Idle 프로세스: CPU는 놀 수 없어 로딩 시 생성되는 프로세스로, 내용 없는 무한루프 돌다가 임무 내려지면 수행

4. Preemptive vs. Non-preemptive
   A. Non-preemptive scheduling(비선점형 스케줄링)
      I. 정의: 프로세스 A 수행 중 B 프로세스가 ready 큐에 올라오면 A 프로세스 다 끝난 후에 B 프로세스 실행
      II. 특징: 일상생활에서 Preemptive scheduling에 비해 더 일반적으로 사용
   B. Preemptive scheduling(선점형 스케줄링)
      I. 정의: 프로세스 A 수행 중 B 프로세스가 ready 큐에 올라오면 A 프로세스 중지하고 B 프로세스 실행
      II. 특징: 스케줄링에서 Non-preemptive scheduling에 비해 더 일반적으로 사용
         - 이유: CPU-bound process의 우선순위가 더 높아야 하는데 Non-preemptive scheduling에서는 I/O-bound process 길어지면 CPU-bound process가 오랫동안 실행 안되는 경우 발생

5. Scheduling Criteria
   A. CPU 자원의 효율성(성능) 관련
      I. CPU utilization: CPU가 노는 시간을 적게 하기(CPU를 바쁘게 하기)
      II. Throughput: 단위시간에 실행되는 프로세스의 수를 크게 하기
   B. 시간 관련

I. Turnaround time: 프로세스가 실행가능해질 때까지 걸리는 시간(waiting 큐에 삽입되어 스케줄링 될때까지)

II. Waiting time: I/O 처리를 기다리는 시간으로, 실질적으로 돌아갈 수 있지만 기다리는 시간

III. Response time: 로딩이 매끄러운 것 처럼 사용자에게 결과값을 얼마나 빠르게 보여주는지

C. A VS B

I. 관계: Trade-off 관계(A: 높을수록 좋음, B: 낮을수록 좋음, 양립할 수 없음)

- 이유: A는 CPU-bound process에, B는 I/O-bound process에 각각 중요하기 때문

II. 스케줄링 알고리즘 목표: A, B를 동시에 만족하는 알고리즘

6. Scheduling Goals/Non-goals

A. Scheduling Goals

I. All systems: Fairness(공평하게 CPU 자원 분배) & Balance(그러면서도 CPU, I/O 다 바쁠 수 있게 밸런스 중요)

II. Batch systems: 돌아가는 하나의 프로세스 동작 시간 줄이기 최우선

- Throughput/CPU utilization 높이기, Turnaround time 낮추기

III. Interactive systems: Proportionality(유저의 기대치 충족) 최우선

- Response time/Waiting time 낮추기

IV. Real-time systems: Deadline 충족이 최우선

- Predictability: 프로세스 얼마나 걸릴지 예측해서 deadline 최대한 맞추기(불가능한 개념)

B. Scheduling Non-goals

I. Starvation: 프로세스가 CPU 자원을 써야 하는데 쓰지 못하고 있는 상황

- 이유: 스케줄링 policy(알고리즘)문제/ Synchronization(동기화) 문제

7. Single-Processor Scheduling Algorithms

A. FCFS (First Come First Served)

I. 개요: 프로세스들이 도착한 순서대로 스케줄링됨(선착순)

II. 방식: 비선점 방식(non-preemptive)

III. 장점: 모든 임무는 공평하게(fair) 다뤄지고, starvation 발생 X

IV. 단점:

- Convoy effect(Burst time 긴 애들 뒤에 짧은 애들이 오는 문제 > waiting time 증가 > 성능 감소
- waiting time: ready queue 들어왔을 때부터 dispatch까지 걸리는 시간

B. SJF (Shortest Job First)

I. 개요: 프로세스들 중 burst time 가장 작은 것 먼저 스케줄링

II. 방식:

- 비선점 방식(non-preemptive): SJF (Shortest Job First) - burst time 최소인 것 먼저 스케줄링
- 선점 방식(Preemptive): SRTF (Shortest Remaining Time First) – 남은 burst time 최소인 것 먼저 스케줄링

III. 장점: FCFS에 비해 average waiting time이 optimal(시간 더 작은 것 없음)

IV. 단점:

- Burst time 알아야 사용하는데 burst time을 예측하기가 어려움 > ideal(이상향)
- Starvation 발생 가능(time 100 queue 올라온 상태에서 time 1 이 계속 올라오면 발생)

C. Priority

I. 개요: 각 프로세스에 int priority number(우선순위) 할당 > 우선순위 가장 높은 프로세스가 스케줄링

II. 방식:

- 비선점 방식(non-preemptive): 프로세스들이 다 동시에 도착했다고 가정
- 선점 방식(Preemptive): 프로세스 제각기 다른 시간에 도착 가능(arrival time 존재)

III. 구현: multi-level queue(동일한 우선순위끼리 queue 관리, 우선순위 높은 queue empty시 아래꺼 스케줄링

IV. 장점: 이전 방식들과 달리 실제 OS에 적용할 수 있는 개념

V. 단점:

- Starvation 발생 가능(낮은 우선순위의 프로세스는 절대 동작 안 할 수 있음)
- > 해결책: Aging(시간이 지남에 따라 우선순위를 점점 증가시킴)
- Response time 아주 김

D. Round Robin (RR)

I. 개요: 프로세스들의 response time을 줄이기 위해 모든 프로세스가 CPU를 자주 쓰게 하자(각각 q동안)

II. 방식: 선점 방식(Preemptive): time quantum q 지나면 해당 프로세스 내려오고 다음 프로세스 올라감

III. 구현: q(time quantum)값에 따라 경향 달라짐

- 큰 q: FCFS(FIFO) 방식과 똑같아짐
- 작은 q: context switch 시간에 비해 커야 함(커널 오버헤드 너무 커지는 것 방지)

IV. 장점: SJF 비해 response time 짧음, Starvation 발생하지 않음

V. 단점:

- 커널 오버헤드가 이전 방법들보다 큼
- > 이유: 프로세스 하나여도 time quantum 바뀌면 dispatch 다시 일어나서 PCB 저장/restore
- 상황에 따른 q값 정하기 어려움 > 프로세스 80%의 burst가 q보다 짧게 하기(rule of thumb)

E. Multi-Level Queue

I. 개요: Round Robin은 Fair하고, Priority scheduling은 unfair한 대신 중요한 작업 먼저 처리 가능 > 두개 결합

II. 방식:

- Foreground queue: 우선순위 높, response time 높은 I/O bound 프로세스 위주 > RR 적용
- Background queue: 우선순위 낮, response time 낮은 CPU bound 프로세스 위주 > FCFS 적용

III. 장점: Round Robin과 Priority scheduling의 장점을 모두 가지고 있음

IV. 단점:

- 우선순위가 고정되어 있어 Starvation 발생 가능(foreground queue에만 계속 들어오면)
- 해결법: Time slice(단위시간의 80%는 Foreground, 20%는 Background가 사용)

F. Multi-Level Feedback Queue (MLFQ)

I. 개요: 프로세스들의 우선순위를 고정하지 않고 aging 개념을 도입

II. 방식:

- 프로세스가 ready queue에 도착했을 때 실행시 q = 8(예시) 할당
- > 만약 8 다 쓰기 전에 내려오면(I/O 요청) I/O bound라고 판단하여 q 값 낮춰줌
- > 만약 8 다 쓰면 CPU bound라고 판단하여 q 높여줌

III. 장점:

- Multi-Level Queue와 달리 프로세스를 직접 돌려봐서 I/O bound인지 CPU bound인지 구분 가능
- Multi-Level Queue의 우선순위 고정 문제 해결 가능

IV. 특징: 현재 Unix에 실제로 사용됨(3~4 class, 170 priority level로 q를 높이고 낮추면서 유연하게 구현

8. Multiple-Processor Scheduling

A. Load balancing: 각 코어의 일을 어떤 방식으로 배분할 것인지

I. Push: 수동적인 방식으로, 바쁜 코어가 노는 코어에게 balancing을 걸어 일을 배분

II. Pull: 능동적인 방식으로, 노는 코어가 바쁜 코어에게 balancing을 걸어 일을 가져옴

B. Processor affinity: ready queue에 있는 프로세스들을 core들에게 어떻게 줄 것인지

I. Soft: queue를 common하게 1개만 사용

- 장점: 오버헤드가 작음(단순하여 복잡도가 작음)
- 단점: 캐싱 관점에서 효율 낮음(프로세스를 무작위로 분배하기 때문에 캐싱 불가)

II. Hard: core마다 queue를 따로 배분
- 장점: 비슷한 프로세스끼리 묶이기 때문에 캐시 효율 증가
- 단점: 오버헤드가 큼(queue를 여러 개 관리해야 하고, 스캐줄링 시 dependency 등 고려 factor 증가

III. Soft VS Hard: Trade-off 관계로, 2개 다 쓰임

9. Real-Time Scheduling

A. 핵심: deadline 맞추기(hard real-time, soft real-time에서 모두 중요)

B. Rate-Monotonic algorithm

I. 방식: Static(우선순위는 실행 시간이 짧을수록 높으며, sw 설계시 미리 결정됨)

II. 특징:
- deadline을 못 맞추는 경우 발생
- Dynamic 방식보다 더 많이 사용됨(embedded 환경에서는 하드웨어에 제약 있음)

C. EDF (Earliest Deadline First) algorithm

I. 방식: Dynamic(우선순위는 deadline이 임박할수록 높으며, 런타임에 실시간으로 결정됨)

II. 특징: Static 방식에 비해 deadline 맞출 확률 더 높음

10. Operating System Examples

A. Linux scheduling

I. 방식: foreground(real-time), background(normal)로 우선순위가 크게 나뉘고, 안에서 세분화
- Multi-level feedback queue에 의해서 우선순위가 실시간으로 바뀌고, 같은 우선순위끼리는 RR 적용

II. 특징: CFS(Completely Fair Scheduling)이라고도 불림

B. Windows scheduling

I. 방식: 2차원 우선순위 테이블을 바탕으로 하며, Linux와 마찬가지로 우선순위 실시간으로 바뀜(alt tab, …)

C. Solaris scheduling

I. 방식: interrupt thread는 불변의 최우선 우선순위로 놓고, 나머지 thread들은 가변적인 우선순위로 관리

11. Algorithm Evaluation

A. Queueing models

I. 방식: code를 짜지 않고, 시간 복잡도 등을 이용해서 수학적으로 알고리즘 효율을 분석

B. Simulation

I. 방식: 실제 버전의 주요 factor만 모은 단순화된 버전으로 단기간에 결과값을 확인

II. Vs Emulation: Emulation은 실제와 비슷한 환경을 만들고, 시간도 실제와 동일하게 돌리는 것이 차이점

C. Implementation

I. 방식: 알고리즘을 code로 직접 구현하여 벤치마킹 등으로 알고리즘 효율을 분석