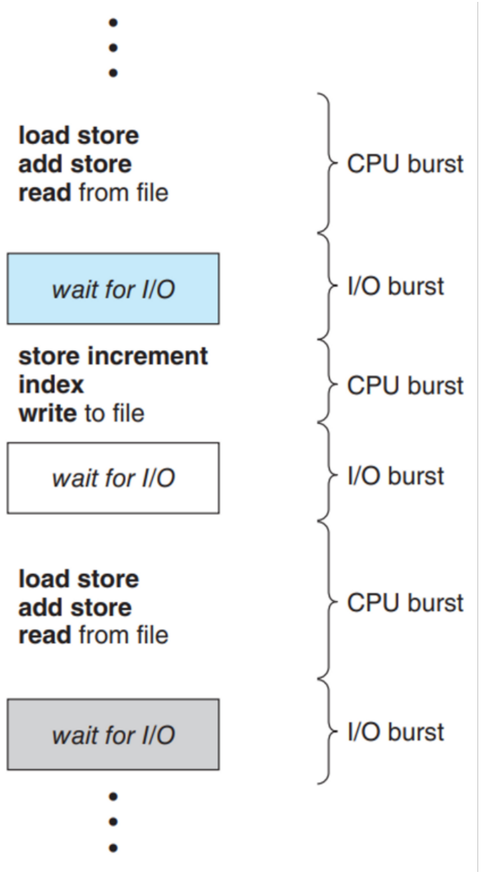
5. CPU Scheduling

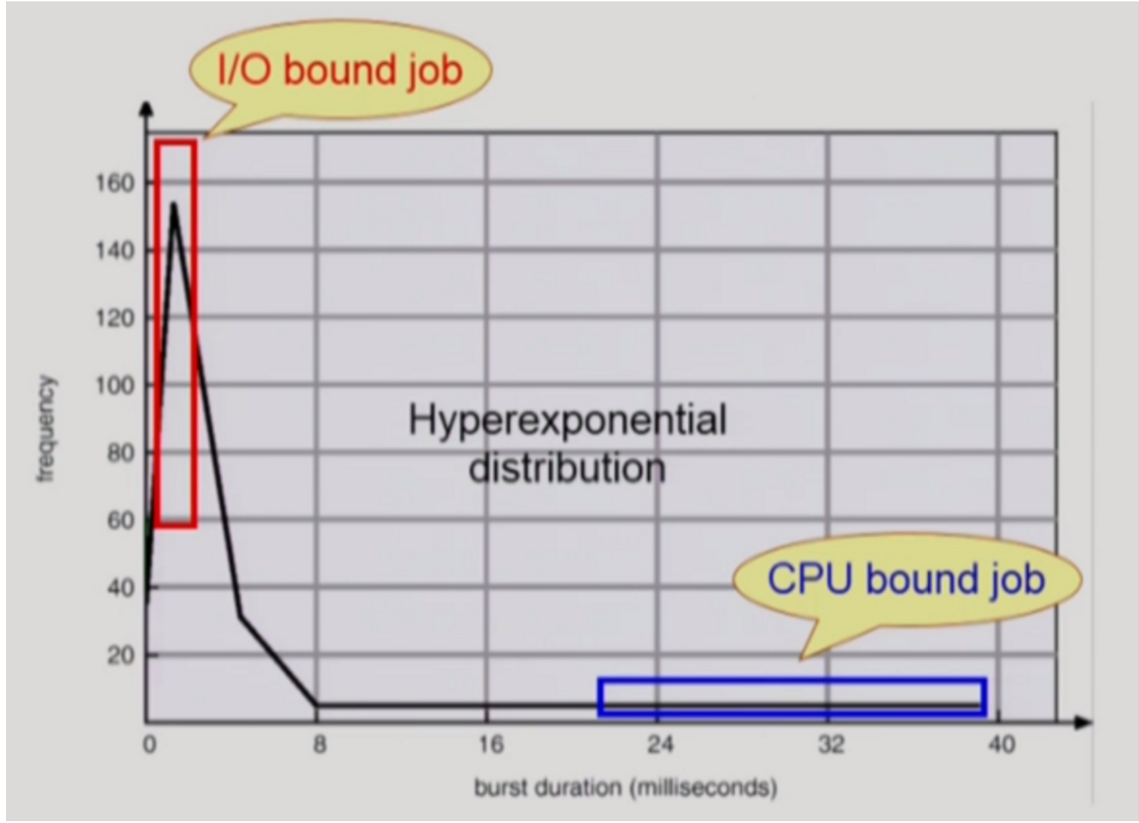
6. Process Synchronization

2023.06.23 **황재천 신석철 박영서**

5. CPU Scheduling

01 Review


load store
add store
read from file
wait for I/O
store increment
index
write to file
wait for I/O
load store
add store
read from file
wait for I/O
CPU burst
I/O burst
CPU burst
I/O burst
CPU burst
I/O burst


I/O bound job
CPU bound job
Hyperexponential
distribution
frequency
160
140
120
100
80
60
40
20
0
8
16
24
32
40
burst duration (milliseconds)

02 Scheduling Criteria(성능 척도)

☆ 매 CPU Burst마다 계산하는 것(프로세스가 시작되고 종료되는 개념 X)

- 시스템 입장에서의 성능 척도
  - CPU utilization(이용률)
  - Throughput(처리량)

- 프로그램(고객)입장에서의 성능 척도
  - Waiting time(대기시간)
  - Turnaround time(소요시간, 반환시간)
  - Response time(응답시간)
  - 영서 : Turnaround Time, Waiting time, Response Time에 대한 상세한 설명을 해 주실 수 있으실지?

02 시스템 입장에서의 성능 척도

- CPU utilization(이용률)
  - 전체 시간 중에서 CPU가 놀지 않고 일한 시간을 의미
  - 가능한 한 CPU를 가장 많이 사용하고자 합니다. 개념적으로 CPU 이용률은 0에서 100%까지 범위를 가질 수 있습니다. 실제 시스템에서는 가벼운 부하 시스템의 경우 40%부터 많은 부하 시스템의 경우 90%까지 범위 내에서 유지되어야 합니다. (Linux, macOS 및 UNIX 시스템에서 top 명령을 사용하여 CPU 이용률을 얻을 수 있습니다.)

- Throughput(처리량)
  - 주어진 시간동안 얼만큼의 일을 처리했는지는 나타내는 척도
  - CPU가 프로세스를 실행하는 동안 작업이 수행됩니다. 작업의 측정 단위로는 단위 시간당 완료된 프로세스의 수인 처리량이 있습니다. 긴 프로세스의 경우 몇 초에 하나의 프로세스일 수 있고, 짧은 트랜잭션의 경우 초당 수십 개의 프로세스가 될 수 있습니다.

02 프로그램(고객)입장에서의 성능 척도 시간과 관련된 성능 척도

- Waiting time(대기시간)
  - CPU 스케줄링 알고리즘은 프로세스의 실행 시간이나 I/O 수행 시간에 영향을 주지 않습니다. 그저 프로세스가 준비 큐에서 대기하는 시간만 영향을 받습니다. 대기 시간은 준비 큐에서 대기하는 기간의 합입니다.
- Turnaround time(소요시간, 반환시간)
  - 특정 프로세스의 관점에서 가장 중요한 기준은 해당 프로세스의 실행에 소요되는 시간입니다. 프로세스 제출 시간부터 완료 시간까지의 간격을 반환 시간이라고 합니다. 반환 시간은 준비 큐에서 대기하는 시간, CPU에서 실행되는 시간 및 I/O를 수행하는 시간의 합입니다.
  - CPU 쓴 시간 + 중간중간 기다리는 시간 = 들어와서 나가기 까지
- Response time(응답시간)
  - 대화형 시스템에서는 반환 시간이 가장 적합한 기준이 되지 않을 수 있습니다. 종종 프로세스는 초기에 어떤 출력을 생성하고 이전 결과가 사용자에게 출력되는 동안 새로운 결과를 계산하는 동안 계속해서 작업을 수행할 수 있습니다. 따라서 다른 측정 항목으로는 요청이 제출된 시점부터 첫 번째 응답이 생성되는 시간을 나타내는 응답 시간이 있습니다. 응답 시간은 응답을 시작하는 데 걸리는 시간이며 응답을 출력하는 데 걸리는 시간이 아닙니다.

02 Waiting time *VS* Response time

- Waiting time
  - 특정 프로세스가 준비큐에서 대기하는 시간 + CPU Burst 중간중간 일어나는 인터럽트로 인한 대기시간까지 포함

- Response time
  - 특정 프로세스가 준비큐에서 대기하다가 처음 CPU Burst로 넘어가기까지 걸리는 시간

03 Scheduling Algorithms

- FCFS(First-Come First-Served)
- SJF(Shortest-Job First)
- Priority Scheduling
- Round Robin(RR)
- Multilevel Queue
- Multilevel Feedback Queue
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling

03 FCFS(First-Come First-Served)

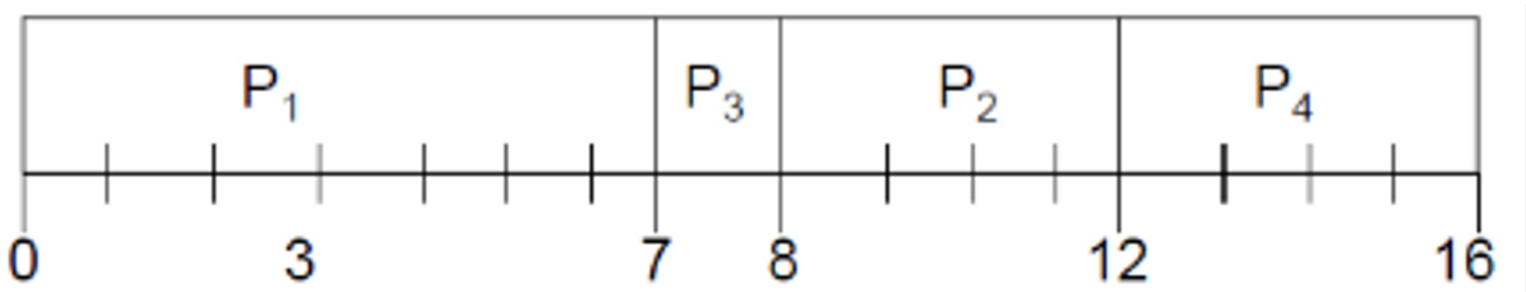
P1 : 24 → P2 : 3 → P3 : 3

- 프로세스의 도착 순서대로 먼저 진행하는 스케줄링 알고리즘
- 비선점형 방식(Nonpreemptive)
- 효율적이진 않음(평균 대기시간 = 17)
- 바로 직전 프로세스의 상태에 따라 효율성이 매우 많이 달라짐(Convoy effect)

03 SJF(Shortest-Job First) - Nonpreemptive

- CPU Burst가 가장 짧은 프로세스를 우선적으로 처리하는 스케줄링 알고리즘
- 평균 대기 시간이 가장 짧아짐
- Nonpreemptive
  - 일단 CPU를 잡으면 이번 CPU Burst가 완료될 때까지 CPU를 선점 당하지 않음

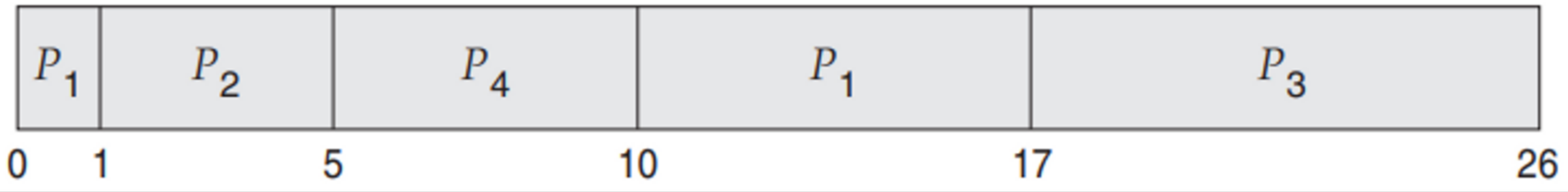
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



03 SJF(Shortest-Job First) - Preemptive

- CPU Burst가 가장 짧은 프로세스를 우선적으로 처리하는 스케줄링 알고리즘
- 평균 대기 시간이 가장 짧아짐
- Preemptive
  - 현재 수행중인 프로세스의 남은 burst time보다 더 짧은 CPU Burst time을 가지는 새로운 프로세스가 도착하면 CPU를 빼앗김
  - 이 방법을 Shortest-Remaining-Time-First(SRTF)라고도 부름

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



03 SJF(Shortest-Job First) - 문제점

- Starvation : CPU Burst가 매우 긴 프로세스의 경우 실행되지 않을 수 있는 문제 발생

석철 : SJF의 Statvation 문제를 해결하기 위한 Aging 기법에 대해 설명해주실 수 있나요?

- Aging은 SJF 알고리즘의 Statvation 문제를 해결하기 위한 기법 중 하나로, 작업의 대기 시간이 증가할수록 우선순위를 높여주는 방법입니다. Aging은 대기 시간을 측정하는 방법으로 사용되며, 대기 시간이 증가할수록 우선순위를 높여 작업이 빨리 실행될 수 있도록 합니다
  1. 각 작업이 대기열에 도착할 때마다 대기 시간을 추적한다.
  2. 대기 시간이 일정 기준을 넘어서면 우선순위를 증가시킨다.
  3. 우선순위가 증가한 작업은 이전보다 더 높은 우선순위를 갖게 되므로, 더 높은 우선순위의 작업이 먼저 실행되도록 스케줄링된다.
  4. 이러한 우선순위 조정은 작업의 대기 시간이 증가함에 따라 계속해서 수행된다.

03 SJF(Shortest-Job First) - 문제점

- CPU Burst time을 미리 알 수 없음
- 과거의 실행했던 시간을 토대로 추측해서 사용
  - exponential averaging

1. t_n =actual length of n^{th} CPU Burst
2. τ_{n+1} = predicted value for the next CPU Burst
3. $\alpha, 0 < \alpha \leq 1$
4. Define : $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

  - 요약하면 최근 실행된 Bust 시간은 많이 반영하고, 오래된 Burst 시간일 수록 덜 반영하는 식 완성

03 Priority Scheduling

- 프로세스마다 적절한 우선순위를 부여하여 우선순위가 높은 프로세스부터 실행하는 알고리즘

프로세스별로 priority number (integer)를 할당하여 우선순위를 나타낼 수 있으며 일반적으로 priority number가 '작을수록' 높은 우선순위를 뜻한다.

- 우선순위는 프로세스 생성 시에 사용자에 의해 지정되기도 하고

운영체제에서 내부적으로 메모리 용량이나 실행 시간 등을 기준으로 하여 결정하기도 한다.

- SJF, SRTF도 Priority Scheduling의 일종이다

예를 들어 SJF는 우선순위를 CPU 실행 시간으로 둔 것

- nonpreemtive & preemtive 두 가지 방식으로 구현할 수 있다.

→ 이 알고리즘 역시 우선순위가 높은 프로세스가 계속해서 ready queue에 들어올 경우 우선순위가 낮은 프로세스는 영원히 실행되지 않을 수 있는 문제가 있다 Starving 문제

→ 이러한 문제를 해결하기 위해 Aging 이라는 기법을 사용한다

아무리 우선순위가 낮더라도 오래 기다렸다고 하면 우선순위를 조금씩 높여주는 것

03 Priority Scheduling

Q. 영서
-> Priority Scheduling에서 priority로 선정할 만한 것들은 무엇인가?

보통 프로세스 우선순위의 기준은 운영 체제 개발자 또는 시스템 관리자에게 달려있다.

우선순위의 기준 예시

1. 작업의 중요도
   특정 작업이 시스템의 전체적인 성능이나 비즈니스 목표에 얼마나 중요한지를 고려하여 우선순위 할당 가능
2. 요청된 서비스의 종류
   입출력 (I/O) 작업이 많은 프로세스에게 더 높은 우선순위를 부여하여 디스크나 네트워크 등의 자원을 효율적으로 활용할 수 있음
3. 프로세스의 실행 시간
   실행 시간이 긴 프로세스에게 더 낮은 우선순위를 부여하여 다른 프로세스들이 빠르게 실행될 수 있도록 할 수 있음

03 Round-Robin (RR)

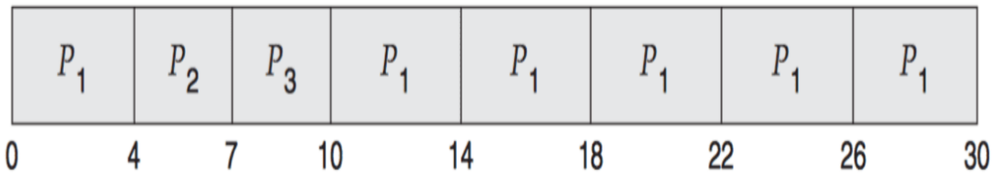
- 현대적인 컴퓨터 시스템에서 사용하는 알고리즘
- ready queue의 프로세스들을 일정한 할당 시간 단위(quantum)로 돌아가면서 실행한다. 선택된 프로세스는 실행을 위해 할당된 시간을 가지며 이 시간이 만료되면 CPU를 빼앗기고 강제로 ready queue에 삽입

1. 할당 시간 만큼 CPU를 줬다가 뺐는 작업을 반복하기 때문에 응답시간이 짧아지는 효과
2. CPU 실행 시간을 예측할 필요가 없고 모든 프로세스가 CPU를 얻을 수 있음. 어떤 프로세스도 할당 시간 단위(q) x (프로세스 개수(n) - 1)만큼의 시간 이상 기다릴 필요가 없다. 이 시간 내에 반드시 실행할 기회가 주어짐

- 일반적으로 SJF보다 Turnaround time, Waiting time은 길지만 Response Time은 더 짧음

Process	Burst Time
P1	24
P2	3
P3	3

- Gantt chart:

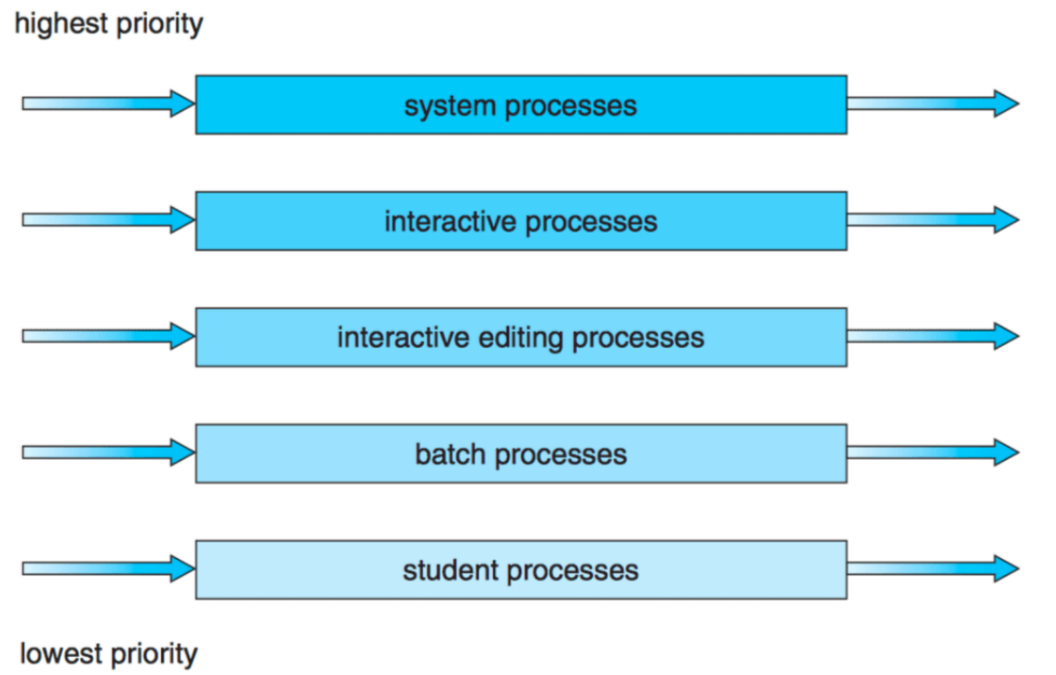


03 Multilevel Queue

- ready queue를 프로세스들의 특성에 따라 여러 레벨로 분리

- 각 queue는 각자에 맞는 독립적인 스케줄링 알고리즘을 가짐
  - foreground(대화형) - RR_ 사람과 interaction 하기 때문에 RR 사용
  - background - FCFS_ 어차피 CPU만 오래 사용하는 것이기 때문에 Context switching overhead 줄이기 위해 FCFS 사용

- queue에 대한 스케줄링이 필요
  - Fixed priority scheduling
    높은 레벨의 queue를 절대적으로 먼저 처리하는 스케줄링
    즉, 높은 레벨의 queue가 비었을 때만 낮은 레벨의 queue에 있는 프로세스를 실행함 → 낮은 레벨에 있는 queue는 영원히 실행되지 않을 수 있음 (Starvation 문제)
  - Time slice
    각 queue에 CPU 시간을 적절한 비율로 할당
    (ex CPU의 80%를 높은 레벨의 queue에 할당하고 나머지 20%는 낮은 레벨의 queue에 할당)



03 Multilevel Feedback Queue

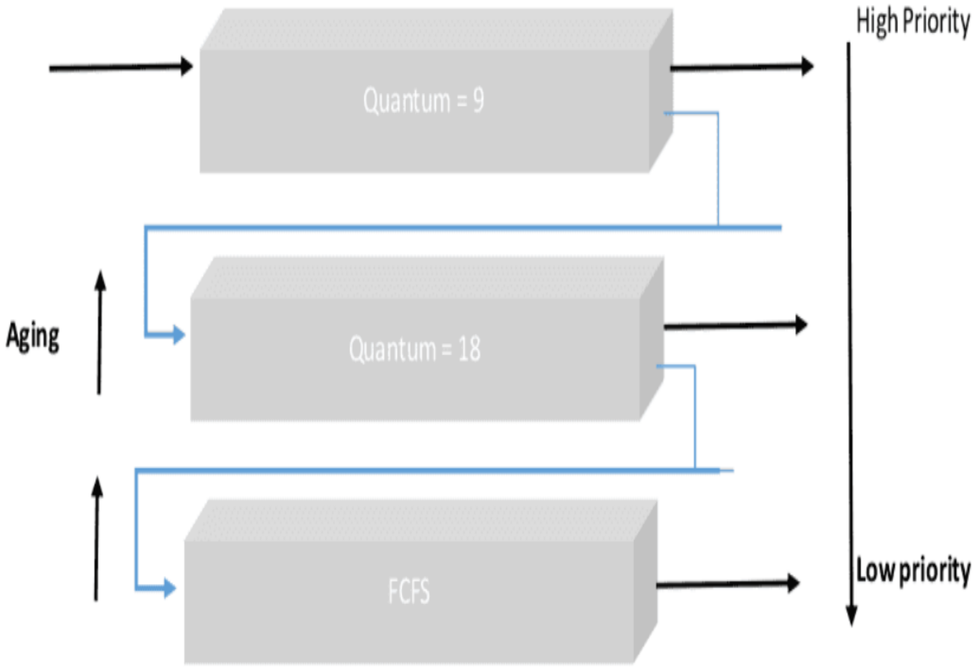
Multilevel Feedback Queue의 일반적인 운영방식

- 프로세스가 queue로 들어갈 때 어떤 queue로 들어갈 것인가
  가장 먼저 도착한 프로세스를 가장 상위 queue로 배치

- 각 queue별로 어떤 스케줄링 알고리즘을 적용할 것인가
  - 상위 queue로 갈수록 짧은 quantum 시간을 주고 RR 적용
  - 해당 큐에서 완료되지 못한 프로세스는 아래로 강등
  - 가장 하위의 queue는 FCFS

- 프로세스를 상위(또는 하위) queue로 보내는 기준은 무엇인가
  상위 queue 할당 시간이 만료될 경우 바로 다음 하위 queue로 강등
  상위 queue가 빌 때 까지 대기
  결국 CPU 사용 시간이 짧은 프로세스한테 우선순위를 많이 주는 방식

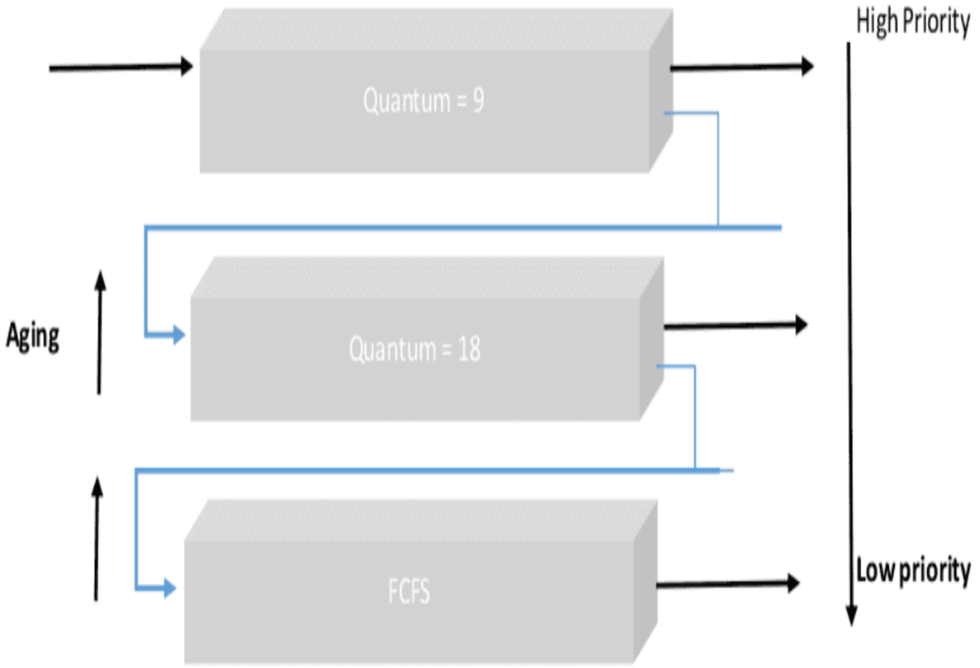
queue 하나에 대해 RR을 적용하는것 보다 더 CPU 실행이 짧은 프로세스를 우대해주는 스케줄링



03 Multilevel Feedback Queue

Q. 재천
-> Multilevel Feedback Queue에서 상위 큐로 올라가는 방법

1. 시간 할당량 제한
   보통 하위 큐에서 실행중인 프로세스가 시간 할당량을 모두 소모한 경우, 해당 프로세스는 하위 큐에 남게 되지만 MFQ에서는 상위 큐로 올라가는 기회를 주는 경우도 있음
2. 우선순위 상승
   하위 큐에서 실행 중인 프로세스가 우선순위 상승 조건을 충족하는 경우, 해당 프로세스는 상위 큐로 올라갈 수 있음
   - 상승 조건은 운영체제 마다 다를 수 있지만 일반적인 두 가지 조건
   a. 실행 시간 초과
      하위 큐에서 실행 중인 프로세스가 시간 할당량을 초과한 경우에는 상위 큐로 올라갈 수 있음
   b. I/O 요청의 유무
      하위 큐에서 실행 중인 프로세스가 I/O 작업을 요청하는 경우, 해당 프로세스는 상위 큐로 올라갈 수 있음



03 Multiple processor scheduling

- Homogeneous processor인 경우
  - queue에 한 줄로 세워서 각각을 processor가 알아서 꺼내가게 할 수 있다.
  - 반드시 특정 프로세서에서 수행되어야 하는 job이 있는 경우,
  해당 job을 특정 프로세서에 먼저 할당하고 나머지를 할당할 수 있다.
- Load Sharing이 잘되어야 한다
  - 일부 프로세서에 job이 몰리고 나머지 프로세서는 놀고 있지 않도록 부하를 적절히 공유하는 메커니즘이 필요하다.
  - 각각의 CPU별로 별개의 queue를 두는 방법도 있고 공동 queue를 사용하는 방법도 있다

- Symmetric Multiprocessing (SMP)
  - 모든 CPU들이 대등한 것
  - 대등하기 때문에 각 프로세서가 각자 알아서 스케줄링 결정
- Asymmetric multiprocessing (AMP)
  - 여러 개의 CPU가 있는데 특정 CPU가 전체적인 컨트롤을 담당
  - 따라서 하나의 프로세서가 시스템 데이터의 접근과 공유를 책임지고 나머지 프로세서는 거기에 따름

03 Real-time Scheduling

Soft real-time systems

: 서비스가 완료되었으면 하는 시간이 주어지기는 하지만, 이 시간을 지키지 못하더라도 critical하지는 않다.
: 서비스가 주어지는 데드라인까지 반드시 완료될 것이라는 것을 보장하지 못한다.
: 주어진 데드라인을 바탕으로 일반 프로세스보다 **높은 priority를 갖도록 스케줄링**한다.
: 예) 일반적으로 사용하는 PC, 휴대폰 등

Hard real-time systems

: 서비스가 **반드시** 데드라인까지 완료되어야 하는 task들을 다룬다.
: 데드라인까지 완수되지 못한 것은 아예 하지 않은 것과 다름 없다.
: 주어진 데드라인까지 반드시 완료되도록 스케줄링한다.
: 예) 원자력 발전소 시스템 등

03 Thread Scheduling

Local Scheduling
: User-level thread의 경우, 사용자 수준의 thread library에 의해 어떤 thread를 어떻게 스케줄할지가 결정된다.

Global Scheduling
: Kernel-level thread의 경우 일반 프로세스와 마찬가지로 커널의 단기 스케줄러가 어떤 thread를 스케줄할지를 결정

04 Algorithm Evaluation

특정 시스템에서 사용할 CPU Scheduling 알고리즘을 고르기 위해 쓰이는 평가 방법들이다.

Deterministic Modeling

: 이미 주어졌다고 가정된 특정 workload를 가지고서 각 알고리즘들을 평가하는 방법
: 강의에서 계속해서 써왔다. (프레젠테이션에서는 RR 슬라이드의 오른편에 있는 그것)
: 쉽고 빠른 분석이 가능하지만, 주어지지 않은, 다른 종류의 workload에 대해서도 마찬가지일지는 알 수 없다.

Queueing Models

: 현대 컴퓨터 시스템에서는 고정된 특정 프로세스 집합이 주어지지 않는다.
: 하지만 CPU-burst의 분포, 프로세스의 arrival-time 분포는 정해져 있다고 할 수 있다.
: 위와 같은 분포들을 바탕으로 알고리즘의 평균 throughput, utilization, waiting time 등등을 계산하는 방법이다.
: 여러 알고리즘들을 비교하는 데에는 유용할 수 있다.
: 다뤄질 수 있는 알고리즘, 분포가 제한적이며, 분석도 복잡하고 어려우며, 결국에는 근사치에 불과하다.

04 Algorithm Evaluation

Implementation & Measurement

: 실제 시스템에 알고리즘을 구현해보고 실제 작업에 대해 성능을 측정, 비교한다.

: 정확하지만 비용이 크고, 적용 환경 또한 바뀔 수 있다.

Simulations

: 컴퓨터 시스템의 모델 및 알고리즘을 모의 프로그램으로 작성해 성능을 측정한다.

: 이때 input data로는 *trace files*등을 사용한다.

*trace files*란?

- 실제 시스템을 모니터링하고 그 이벤트의 순서를 기록함으로써 만들어 낸 파일
- 완전히 같은 입력 집합을 가지고서 여러 알고리즘들을 비교할 수 있으므로 매우 유용하다.

: 시뮬레이션 또한 수 시간이 걸릴 수 있을 만큼 비용이 비싸고, trace files 또한 매우 큰 저장 용량을 필요로 할 수 있다.

6. Process Synchronization

01 Data Access Pattern in Computer Systems

컴퓨터 시스템 내에서의 데이터 접근 패턴

(1) 컴퓨터의 어떤 스토리지(Storage-Box)에 데이터가 저장되어 있음
(2) 데이터를 읽어옴
(3) 연산(Execution-Box)
(4) 연산 결과를 재저장

E-Box	S-Box
CPU	Memory
컴퓨터 내부	디스크
프로세스	프로세스의 주소 공간

02 Race Condition

Race Condition

: 여러 E-box가 한 S-box에 동시에 접근하려 할 때, 최종 출력 결과가 그 접근 순서에 의해 결정되는 상황

: 한 S-box를 공유하는 E-box가 여럿 있는 경우 경쟁 상태(Race Condition)에 빠질 가능성이 있다.

예) 메모리를 공유하는 여러 CPU(멀티프로세서 시스템에서)

한 주소 공간을 공유하는 여러 프로세스

- shared-memory의 경우
- 커널 내부 데이터에 접근하는 루틴의 경우
  - 예) 커널 모드 수행 중 인터럽트로 커널 모드의 다른 루틴 수행 시

02 OS에서의 Race Condition

Kernel 수행 중 인터럽트 발생시

(1) 커널모드로 수행 중 인터럽트가 발생해 해당 인터럽트의 핸들러가 수행됨
(2) 기존에 수행되던 것과 인터럽트 핸들러 양쪽이 모두 커널 코드이므로 커널 주소 공간을 공유함
(3) 기존의 것과 인터럽트 핸들러가 동일한 주소 공간에 접근하려 하는 경우 경쟁 상태 발생
- 해결
중요한 변수 값을 건드리는 동안에는 인터럽트가 들어와도 핸들러로 넘어가지 않고 작업이 끝난 후에 핸들러로 넘어가기

유저 프로세스가 system call을 해 kernel mode로 수행 중인데 context switch가 일어나는 경우

(1) 두 프로세스의 주소 공간에는 data sharing이 없음
(2) 그러나 system call을 하는 동안에는 커널 주소 공간의 데이터를 공유하게 됨
(3) 이 작업 중 context switch가 일어나 다른 프로세스가 CPU를 선점하려 한다면 race condition 발생
- 해결
커널 모드로 수행 중에는 CPU를 다른 프로세스가 선점하지 못하도록 하고, 사용자 모드로 돌아갈 때 넘겨주기.

02 OS에서의 Race Condition

멀티프로세서에서 공유 메모리 내의 커널 데이터에 접근

여러 프로세서가 메모리의 동일한 곳에 동시에 접근하려 하는 경우
멀티프로세서일 경우 Interrupt enable/disable로는 상황 해결이 불가하다

(철썩 : 웨않되?)
한 프로세서의 Interrupt를 disable하더라도 다른 프로세서가 해당 영역을 읽는 것을 막을 수는 없기에

- 해결 방법 1) 한 번에 하나의 CPU만이 커널 모드로 들어갈 수 있게 하기
  CPU가 여럿 있어도 커널에 접근할 수 있는 것은 단 하나의 프로세서 뿐. 비효율적

- 해결 방법 2) 커널 내부에 있는 각 공유 데이터에 접근할 때마다 그 데이터에 대한 lock/unlock을 설정
  한 프로세서가 공유 데이터에 들어가면 data에 대한 접근을 lock하고, 해당 작업이 끝나면 unlock

03 Process Synchronization Problem

Process Synchronization Problem

공유 데이터에 대한 동시 접근은 데이터의 불일치 문제를 발생시킬 수 있다.
일관성의 유지를 위해서는 협력 프로세스 간의 **실행 순서를 정해주는 메커니즘**이 필요하다

Race Condition

여러 프로세스들이 동시에 공유 데이터에 접근하는 상황.
데이터의 최종 연산 결과는 마지막에 그 데이터를 다룬 프로세스에 따라 달라질 수 있다.
Race Condition을 막기 위해서 concurrent process는 동기화되어야 한다.

Critical-Section Problem

n개의 프로세스가 공유 데이터를 동시에 사용하고자 하는 경우, 각 프로세스의 code segment에는 공유 데이터에 접근하는 코드인 **critical section**이 존재한다.

한 프로세스가 critical section에 있을 때, 다른 모든 프로세스는 critical section에 들어갈 수 없어야 한다.

03 Process Synchronization Problem

프로그램적 해결법의 충족 조건

Mutual Exclusion
: 한 프로세스가 critical section 부분을 수행 중이면 다른 모든 프로세스들은 그들의 critical section에 들어가면 안 된다.
: 중복 접근을 방지

Progress
: 아무도 critical section에 있지 않은 상태에서 critical section에 들어가고자 하는 프로세스가 있으면 들어가게 해야 한다

Bounded Waiting
: 프로세스가 critical section에 들어가려 요청한 후로부터 그 요청이 허용될 때까지 다른 프로세스들이 critical section에 들어가는 횟수에 한계가 있어야 한다.
: Starvation에 빠지지 않도록