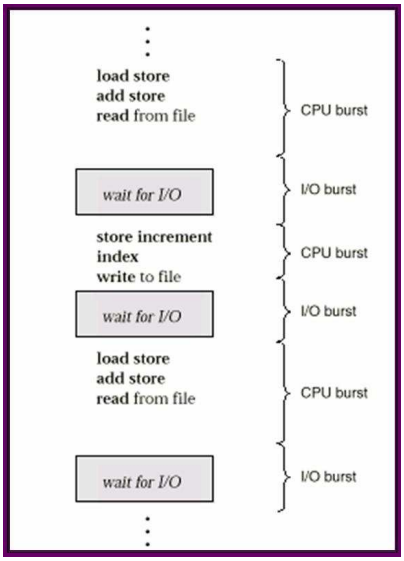
Slide 0

2부 프로세스 관리
(6장. CPU 스케줄링)

기본 개념

Slide 1

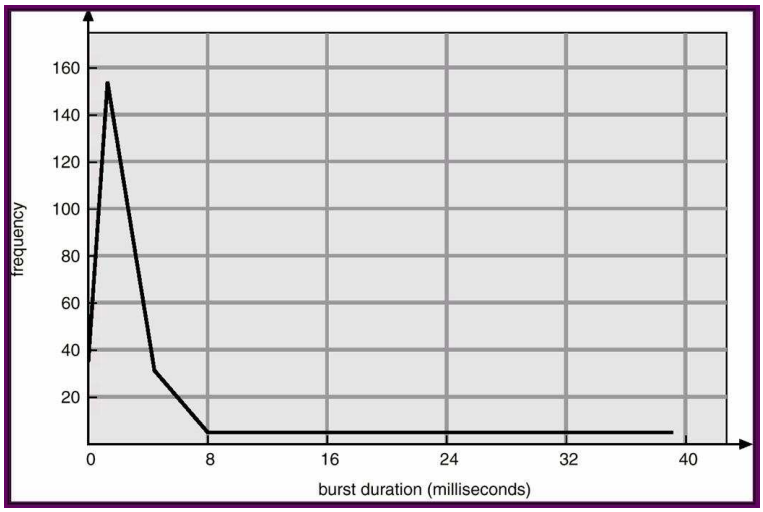
- CPU 스케줄링 목적
  - 항상 실행할 수 있는 프로세스가 있도록 하여 CPU 사용 효율을 극대화함
- CPU-I/O 버스트 주기
  - 프로세스는 수행 중 CPU 실행과 입출력대기를 반복



기본 개념 (Cont'd)

Slide 2

- 계산중심 프로세스 →적은 수의 긴 CPU 버스트
- 입출력중심 프로세스 →많은 수의 짧은 CPU 버스트
- CPU 버스트 빈도수



기본 개념 (Cont'd)

Slide 3

- CPU 스케줄러
  - CPU 스케줄러 (혹은 단기 스케줄러)는
  - CPU 가 유휴 상태가 될 때마다 준비큐에 있는 프로세스중 하나를 선택해서 실행
- 선점 스케줄링 (Preemptive scheduling)
  - CPU 스케줄링이 일어나는 조건
    (1) 프로세스가 실행상태에서 대기상태로 전환될 때 (예로 입출력 요청이나 자식 프로세스의 종료를 기다림)
    (2) 프로세스가 실행상태에서 준비상태로 전환될 때 (예로 인터럽트가 발생할 때)
    (3) 프로세스가 대기상태에서 준비상태로 전환될 때 (예로 입출력의 종료)
    (4) 프로세스가 종료할 때
  - 비선점 스케줄링
    * (1), (4) 일때만 스케줄링이 일어남
  - 선점 스케줄링
    * (1), (2), (3), (4) 에서 스케줄링이 일어남

기본 개념 (Cont'd)

Slide 4

- 선점 스케줄링과 비선점 스케줄링
  - 선점 스케줄링(Preemptive)
    - 프로세스가 수행중일 때 시스템이 임의의 순간에 강제적으로 수행을 중단시키고 다른 프로세스를 수행 시킬 수 있음
    - 임의의 프로세스가 CPU를 오랜 시간동안 점유하는 것을 방지할 수 있음
    - race condition을 유발 할 수 있음 ➙동기화 기법이 요구됨
  - 비선점 스케줄링(Non-preemptive)
    - 현재 수행 중이 프로세스가 자발적으로 CPU를 양도할 때까지 다른 프로세스들은 수행될 수 없음
    - 단순한 알고리즘
    - 구현이 용이함
    - 다중 사용자 시스템에 부적합

기본 개념 (Cont'd)

Slide 5

- 디스패처(Dispatcher)
  - CPU 스케줄러가 선택한 프로세스를 수행하게 해 주는 모듈
  - 기능
    - 문맥교환
    - 사용자 모드로 전환
    - 사용자 프로그램을 다시 시작하기 위해 사용자 프로그램의 재시작 부분으로 점프하는 기능
  - 디스패처 지연 (dispatch latency): 하나의 프로세스를 종료하고 다른 프로세스를 실행하기까지의 소요 시간

스케줄링 기준(Scheduling criteria)

Slide 6

- CPU 스케줄링 알고리즘을 선택할 때 고려되는 기준
  - CPU 사용효율 (utilization): 가능한 CPU가 계속 유용한 작업을 하도록
  - 처리율 (throughput): 단위 시간당 완료되는 프로세스 개수
  - 반환시간 (turnaround time): 어떤 프로세스가 시작한 시간부터 종료될 때까지 걸린 시간
  - 대기시간 (waiting time): 어떤 프로세그가 준비큐에서 대기한 총 시간
  - 응답시간 (response time): 어떤 요청에 대하여 요청부터 첫 반응이 나올 때 까지의 시간
- 선택
  - 사용효율, 처리율은 최대화하고 반환, 대기, 응답시간은 최소화하는 알고리즘을 선택

스케줄링 알고리즘

Slide 7

- 선입선처리 (FCFS: First-Come First-Service) 스케줄링
  - 가장 간단한 알고리즘
  - FIFO(First-In First-Out) 큐로 관리
  - 비선점 방식으로 시분할 시스템에서는 사용하기 어려움
  - 평균 대기 시간이 길어질 수 있음

Process	Burst Time
P1	24
P2	3
P3	3

    * 평균 대기시간(P1 →P2 →P3): 0(P1) + 24(P2) + 27(P3) / 3 = 17ms
    * 평균 대기시간(P2 →P3 →P1): 0(P2) + 3(P3) + 6(P1) / 3 = 3ms

스케줄링 알고리즘 (Cont'd)

Slide 8

- 최소 작업 우선 (SJF: Shortest-Job-First) 스케줄링
  - 가장 작은 CPU 버스트를 가진 프로세스를 우선으로 할당
  - 동일한 버스트 프로세스의 경우 FCFS로 처리
  - 예)

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

    * P4 →P1 →P3 →P2 로 스케줄링
    * 평균대기 시간: $(0 + 3 + 9 + 16)/4 = 7$ms (거의 최적)
    * FCFS 스케줄링에서는 10.25ms

스케줄링 알고리즘 (Cont'd)

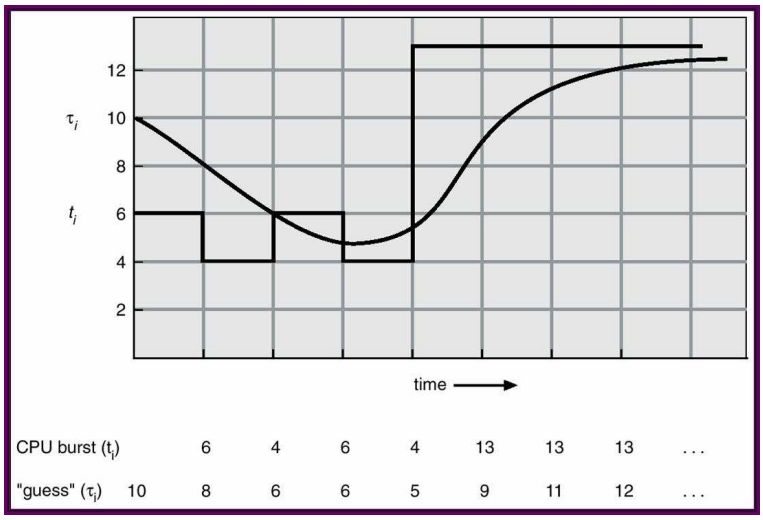
Slide 9

  - 문제점
    * 다음 CPU 버스트의 길이를 알 수 없음
    * 해결 →다음 CPU 버스트를 예측하여 사용
      · 가정
        ➙ 다음 CPU 버스트가 앞의 버스트와 길이가 유사하다고 간주
        ➙ 각 프로세스의 다음 CPU 버스트를 예측해서 가장 작은 프로세스를 선택
      · 예측방법
        ➙ 지수평균값을 사용: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$
        ➙ τ_{n+1} 은 예측 값, t_n 은 최근의 실제 값, τ_n은 과거의 예측 값
        ➙ α 는 과거 예측 값과 최근 실제값을 적용 비율을 결정
        ➙ τ_0 는 상수 혹은 전반적인 시스템 평균으로 정의

스케줄링 알고리즘 (Cont'd)

Slide 10

· 예) $\alpha = 0.5$



스케줄링 알고리즘 (Cont'd)

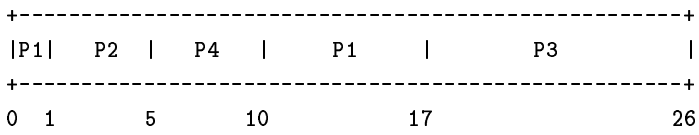
Slide 11

– 구현
  * 비선점형
    · 현재 할당된 프로세스를 끝내도록 허용
  * 선점형
    · 새로 프로세스가 준비큐에 도착하면
    · 실행중인 프로세스의 남은 시간과 새로운 프로세스의 CPU 버스트를 비교
    · 새로 도착한 프로세스의 CPU 버스트 < 실행중인 프로세스의 남은 시간이면
    · 기존 프로세스를 강제 종료하고 새 프로세스 할당
  ➥ shortest remaining time first 스케줄링이라함
    · 예)

스케줄링 알고리즘 (Cont'd)

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Slide 12



- 평균대기시간: ((10-1)+(1-1)+(17-2)+(5-3))/4 = 6.5ms
- 평균반환시간: ((17-0)+(5-1)+(26-2)+(10-3))/4 = 13ms
- 비선점형
  - ➥ 스케줄링: P1 →P2 →P4 →P3
  - ➥ 평균대기시간: (0 + (8-1) + (17-2) + (12-3))/4 = 7.75ms
  - ➥ 평균반환시간: ((8-0) + (12-1) + (26-2) + (17-3))/4 = 14.25ms

스케줄링 알고리즘 (Cont'd)

Slide 13

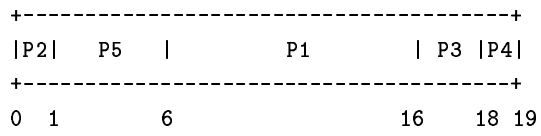
- 우선순위 (Priority) 스케줄링
  - 우선순위가 높은 프로세스에게 먼저 CPU 할당
  - SJF 도 우선순위 스케줄링의 하나 (CPU 버스트를 우선순위의 잣대로 사용)
  - 우선순위는 정수로 0에서 7 혹은 0에서 4095까지로 표현
  - 보통 낮은 수가 높은 우선순위를 갖음

스케줄링 알고리즘 (Cont'd)

Slide 14

- 예)

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



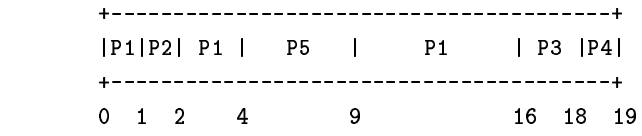
* 평균대기시간: (6 + 0 + 16 + 18 + 1) / 5 = 8.2ms

스케줄링 알고리즘 (Cont'd)

Slide 15

- 선점형으로 구현하거나 비선점형으로 구현 가능
- 선점형 스케줄링 예)

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	1	1	1
P3	2	2	4
P4	3	1	5
P5	4	5	2



* 평균대기시간: (6 + 0 + (16-2) + (18-3) + 0)/5 = 7ms

스케줄링 알고리즘 (Cont'd)

Slide 16

- 문제점
  - 무한대기 (indefinite blocking) 혹은 기아 (starvation) 될 수 있음
    →우선순위가 낮은 프로세스가 CPU 사용을 무한 대기하는 경우
  - 해결 (aging 기법 도입)
    - 대기 시간에 비례해서 우선순위를 높이는 방법
    - 예로 우선순위 0에서 127까지 라고 하면 15분에 1씩 감소시킴

    ☛ 우선순위 127 프로세스도 32시간 뒤에는 우선순위 0 이됨

스케줄링 알고리즘 (Cont'd)

Slide 17

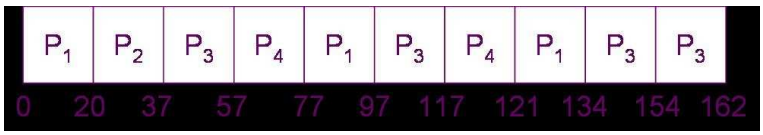
- 라운드로빈(Round-Robin) 스케줄링
  - 시분할 시스템에서 사용하는 스케줄링 알고리즘
  - 시간 할당량(time quantum 혹은 time slice) 라는 시간을 정의
    →보통 10 ∼100ms
  - 시간 할당량이 지나면 현재 프로세스 종료후 다음 프로세스 실행
  - 준비큐는 순환 큐 (circular queue) 사용
  - 스케줄링
    - CPU 버스트 < 시간 할당량
      - 프로세스는 CPU 버스트 시간 후 자발적으로 CPU 사용을 놓음
      - 준비 큐에 있는 다음 프로세스를 실행
    - CPU 버스트 > 시간 할당량
      - timer 종료
      - 운영체제에게 인터럽트
      - 준비 큐에 있는 다음 프로세스로 문맥교환
      - 실행되던 프로세스는 준비 큐 끝에 삽입

스케줄링 알고리즘 (Cont'd)

Slide 18

- 예) time quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24



스케줄링 알고리즘 (Cont'd)

Slide 19

- 예) time quantum = 4

Process	Arrival Time	Burst Time
P1	0	10
P2	1	2
P3	2	6
P4	13	4

```
+-----------------------------------+
|  P1  |P2|  P3  |  P1  |P3|  P4  |P1|
+-----------------------------------+
0      4  6      10     14 16     20 22
```

* 평균대기시간: $(12 + 3 + 8 + 3)/4 = 6.5$ms
* 위에서 P4 의 도착시간이 9 일때 스케줄링 및 평균대기시간은?

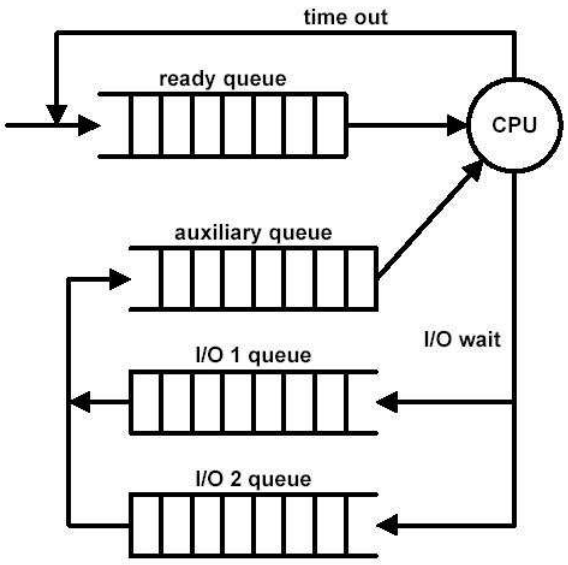
스케줄링 알고리즘 (Cont'd)

Slide 20

- 예)
  - 준비 큐에 n개의 프로세스이 있고 시간할당량이 q이면
  - 각 프로세스는 최대 q시간 단위로 CPU 시간의 $1/n$을 획득
  - 프로세스의 최대대기시간 : $(n-1) \times q$
- 설명
  - 기본적으로 선점형 방식
  - 성능은 시간 할당량에 큰 영향을 받음
    - 할당량이 매우 크면 →선입선처리 스케줄링과 같아짐
    - 할당량이 매우 작으면 →문맥 교환 비용이 커짐
- 문제점
  - 계산 중심 프로세스가 입출력 중심 프로세스보다 많은 시간을 할당 받음
  - 해결
    - 준비 큐외에 별도의 보조큐를 사용
    - 입출력 완료 프로세스를 보조큐로 이동
    - 보조 큐에 프로세스가 있으면 준비 큐보다 먼저 실행

스케줄링 알고리즘 (Cont'd)

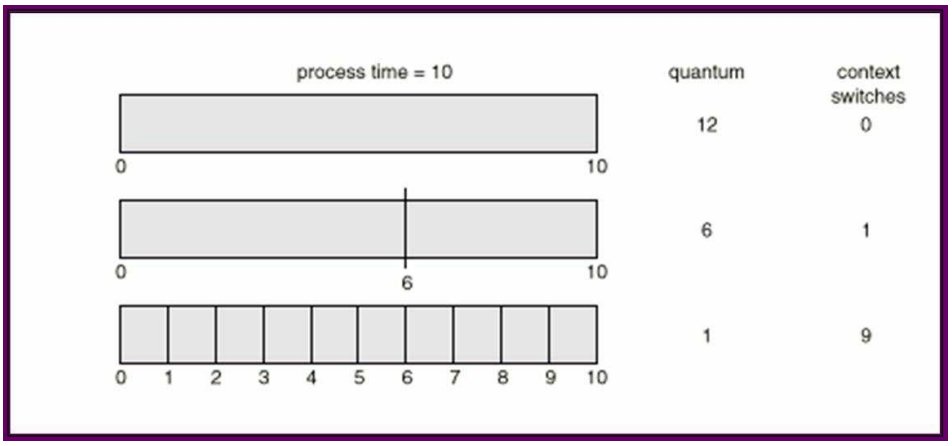
Slide 21



스케줄링 알고리즘 (Cont'd)

– 적은 시간할당량은 문맥교환 횟수를 증대시킴

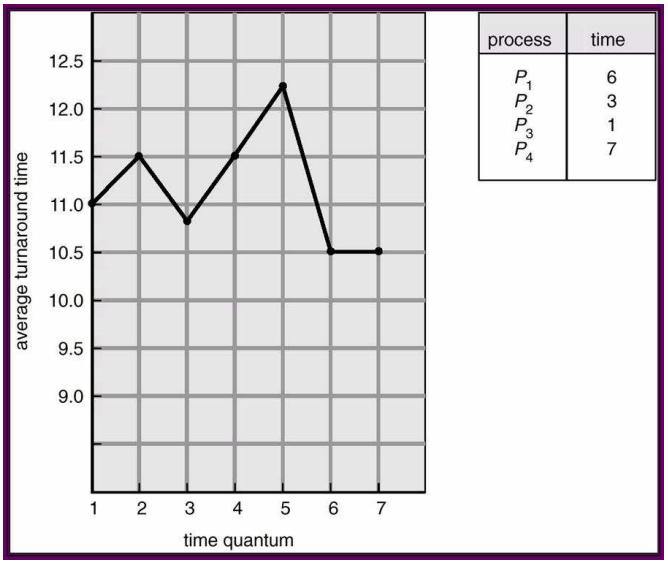
Slide 22



스케줄링 알고리즘 (Cont'd)

– 시간할당량에따른 평균 반환시간

Slide 23



스케줄링 알고리즘 (Cont'd)

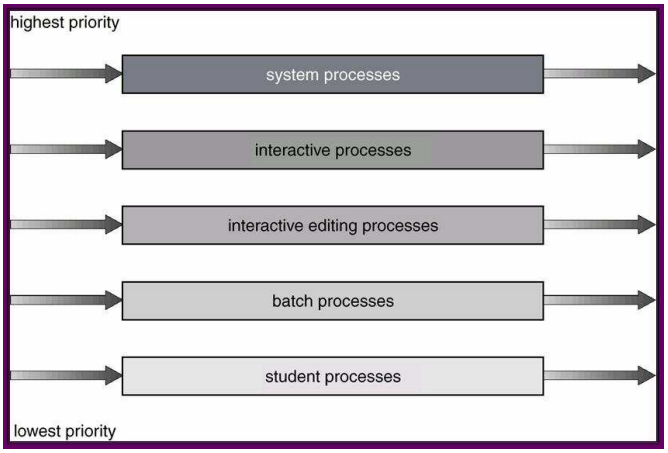
Slide 24

- 다중레벨 큐(Multilevel Queue) 스케줄링
  - 준비 큐를 여러개의 큐로 나누어 사용
  - 프로세스의 특성에 따라 특정 큐에 할당
  - 각 큐는 독자적인 스케줄링 알고리즘을 사용
  - 큐를 구분하는 특성
    * 전면작업 (foreground) 와 후면작업 (background)
    * 기억장치의 요구량
    * 프로세스 우선순위
    * 프로세스 유형

스케줄링 알고리즘 (Cont'd)

Slide 25

- 예)



스케줄링 알고리즘 (Cont'd)

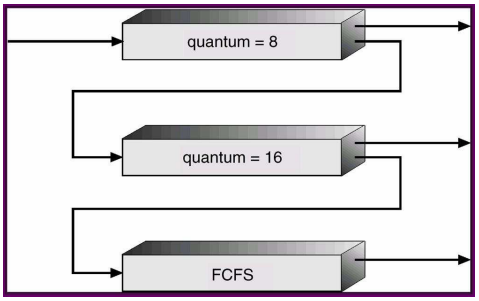
Slide 26

- 다중레벨 피드백 큐 (multilevel feedback queue) 스케줄링
  - 다중레벨 큐 스케줄링의 문제점
    - 하나의 큐에 할당되면 큐사이에 이동을 할 수 없음 (유연성이 떨어짐)
  - 프로세스가 큐사이에 이동 가능
    - 어떤 프로세스가 CPU를 많이 사용하면 낮은 우선순위 큐로 이동
    - 입출력 중심 프로세스와 대화식 프로세스는 높은 우선순위 큐로
    - 낮은 우선 순위큐에 오래 있던 프로세스는 높은 우선순위 큐로 →(기아상태를 예방)
  - 다중레벨 피드백 큐를 결정하는 파라미터
    - 큐의 개수
    - 각 큐를 위한 스케줄링 알고리즘
    - 프로세스를 높은 우선순위 큐로 올려주는 시기를 결정하는 방법
    - 프로세스를 낮은 우선순위 큐로 내려주는 시기를 결정하는 방법
    - 프로세스가 들어갈 큐와 그 프로세스가 서비스를 받는 시기를 결정하는 방법

스케줄링 알고리즘 (Cont'd)

Slide 27

- 예)



  - 상위 큐가 비어있을 때만 하위 큐에서 프로세스 선택 실행 (FCFS)
  - 처음에 모든 프로세스는 최 상위 큐에서 실행
  - 상위 큐에서 프로세스 선택 실행 (8ms) 후 중간 큐로 이동
  - 상위 큐에 프로세스 없으면 중간 큐에서 프로세스 선택 실행 (16ms) 후 하위 큐로 이동
  - 상위 및 중간 큐에 프로세스 없으면 하위 큐에서 프로세스 선택 실행

다중 프로세서 스케줄링

Slide 28

- 단일 프로세서 스케줄링에 비해 복잡
- 다중 프로세서 종류
  - 이종 프로세서로 다중 프로세서 구현
    - 해당 프로세서에서 수행될 수 있는 프로세스만 스케줄 해야함
  - 동일 프로세서로 다중 프로세서 구현
    - 프로세스 스케줄링에 제한 점이 없음
    - 부하 부산 (load sharing) 이 중점
    - 프로세서별로 큐를 사용하면 부하분산이 어려움
    - 공동의 큐를 사용하여 여유있는 프로세서로 스케줄
      - ➙ 각 프로세서가 스케줄하는 방법
      - ➙ 한 프로세서가 스케줄하여 다른 프로세서에게 제공하는 방법

실시간 스케줄링

Slide 29

- 종류
  - 경성 (hard) 실시간 시스템
    - 정해진 시간안에 수행을 완료해야함
    - 작업완료나 입출력 수행에 필요한 시간을 제시
    - 스케줄러는 제시 시간을 보장해 줄 수 있을 경우 스케줄, 아니면 거절
    - 보조기억장치나 가상기억장치를 가진 시스템에서는 제공 불가능
      →(시간을 예측하기 어려움)
    - 일반 범용 컴퓨터에서는 작업의 최대소요시간을 예측하는 것이 어려움
    - 경성 실시간 시스템은 특수 소프트웨어와 작업에 맞게 제작된 특수 하드웨어를 사용하여 구현

실시간 스케줄링 (Cont'd)

Slide 30

- 연성 (soft) 실시간 시스템
  - 중요한 프로세스가 덜 중요한 프로세스보다 높은 우선순위를 갖게 함
  - 우선순위 스케줄링을 사용해야함
  - 실시간 프로세스에게는 가장 높은 우선순위 부여
  - 실시간 프로세스의 우선순위는 낮아져서는 안됨 (aging 을 사용하지 않음)
  - 디스패치 지연이 적어야함
- 디스패치 지연을 줄이는 방법
  - 보통 대부분의 문맥교환 전에 시스템 호출을 수행함
  - 시스템 호출 중단 기능을 구현
    →장기간 수행되는 시스템 호출에 선점 지점 (preemption point)을 삽입
    →이 지점에서는 안전하게 시스템 호출을 중단 할수 있도록
  - 높은 우선순위 프로세스가 필요로하는 자원을 사용하고 있는 낮은 우선순위 프로세스들에게
    →높은 우선순위를 상속하여 빨리 종료하게 함

프로스세 스케줄링 모델

Slide 31

- 솔라리스 2
  - 우선순위에 근거한 알고리즘 사용
  - 우선순위를 네 가지 유형으로 분류
    - 실시간 (선점형 우선순위 스케줄링: 최고 우선순위를 갖음)
    - 시스템 (선점형 우선순위 스케줄링: 커널 프로세스 수행)
    - 시분할 (다중 레벨 피드백 큐)
    - 대화식 (다중 레벨 피드백 큐)
  - 프로세스는 기본적으로 시분할 유형으로 분류
    - 큐마다 다른 길이의 시간할당량을 사용
    - 우선순위가 높을 수록 시간할당량은 작음
    - 상호작용 프로세스 우선순위 > 계산 중심 프로세스 우선순위

프로스세 스케줄링 모델 (Cont'd)

Slide 32

- 윈도우 2000
  - 우선순위에 근거한 선점형 스케줄링 알고리즘 사용
  - CPU 에 할당된 스레드는 다음 상황이 발생될 때까지 계속 실행
    - 자신보다 높은 우선순위 프로세스에 의해 선점된 경우
    - 대기해야 하는 시스템 호출을 하는 경우
    - 시간할당량이 끝난 경우
  - 총 32레벨의 우선순위 사용
    - 가변 클래스 (1에서 15) 와 실시간 클래스 (16에서 31)로 구분
    - 우선순위마다 큐가 있으며 모든 큐가 비어 있으면 idle thread 실행

프로스세 스케줄링 모델 (Cont'd)

Slide 33

- 리눅스
  - 두가지의 스케줄링을 사용
    - 선점형의 시분할 알고리즘
    - 절대적 우선순위를 사용하는 실시간 프로세스를 위한 알고리즘
  - 실시간 프로세스라 하더라도 커널 프로세스를 선점할 수 없음
  - 시분할 알고리즘 →우선순위를 가지는 credit 기반 알고리즘 사용
    - 각 프로세스는 특정 수의 스케줄링 credit 을 가지고 있음
    - 스케줄링 credit 순으로 스케줄링
    - 타이머 인터럽트시 실행중인 프로세스의 credit 하나를 뺌
    - credit 가 0 이되면 프로세스 중단 후 다음 프로세스 실행
    - 실행 가능한 모든 프로세스의 credits 이 모두 0 이면
      →credit 재할당: $credit = \frac{credit}{2} + priority$
  - 두개의 실시간 스케줄링 제공
    - FCFS 방식 (선점 되지 않음)
    - RR 방식 (같은 우선 순위는 공정하게 CPU를 사용)