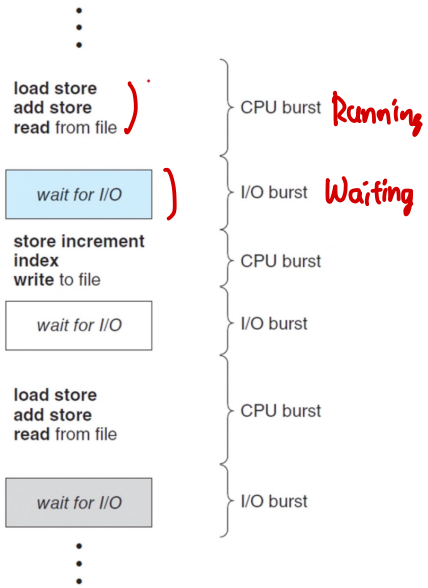
운영체제 - CPU 스케줄링

B889047

윤준호

Scheduler Algorithms



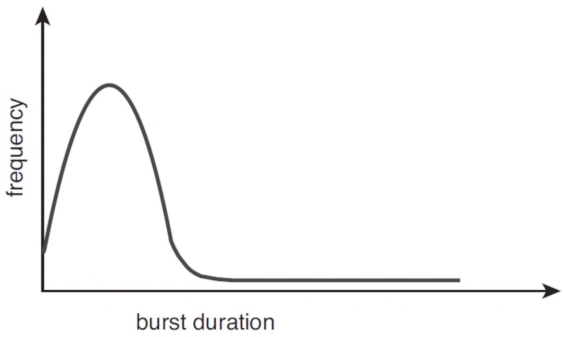
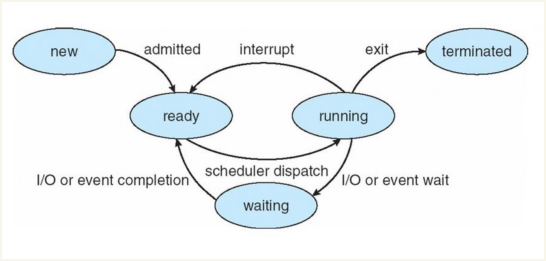


Figure 5.1 *Histogram of CPU-burst duration*

* CPU Burst: CPU연산사용시간 → running 상태
* I/O Burst: I/O에 사용되는시간 → waiting 상태

* I/O Bound vs CPU Bound
  - I/O Bound → I/O Burst Time이 많은것
  - CPU Bound → CPU Burst Time이 많은것

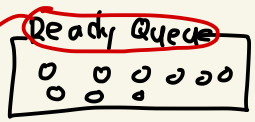
프로세스 상태 다이어그램

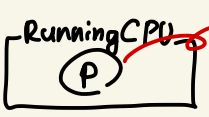


* 항상 CPU Burst → I/O Burst 순서이다.

* CPU 스케줄러 → 메모리에 올라온 프로세스들 어떤프로세스에 CPU할당?

*다음프로세스를 어떻게선택?





→ 얘가나가면? ReadyQueue에서 가져와야함

Linked List, Binary Tree ···

- Preemptive VS Non-Preemptive

Preemptive: 강제로 쫓아낼수 있을

Non-Preemptive: 쫓아낼수 없음 자발적으로 내보내줌

① Non-Preemptive
- 어떤 CPU가 선점하면, 자발적으로 나오기 전까지 점유

② Preemptive
- 스케줄러가 프로세스를 강제로 내보낼수있다.

- Decision Making in CPU-Scheduling

① Process Switches "Running" to "Waiting" state (I/O)

② Process Switches "Running" to "Ready" state (Ready Queue without I/O)

③ Process Switches "Waiting" to "Ready" state (I/O ends. go to Ready Queue)

④ Process Terminates

Preemtive or Non-Preemtive

④ Non-Preemptive, 선택지X

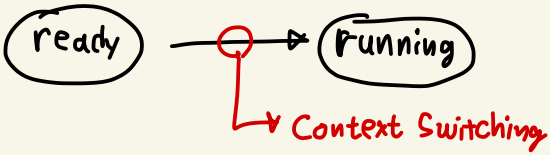
ex) Ready → Wait로갔다가 해당 Process가 CPU에서 뺏어가는 Process보다 높은 Priority를 가진다면?

*현대적인 OS는 모두 Preemptive Scheduling

* Dispatcher

- CPU스케줄러가 선택한 Process에게 CPU제어권을 넘겨주는 모듈이다.

∴ Context Switching을 하게하는 모듈이다.

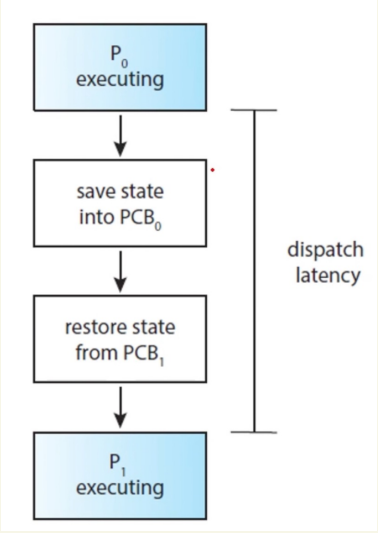


* Function of Dispatcher

- Switching context from one process to another
- Switching to usermode
- Jumping to the proper location to resume program

∴ Scheduler → 선택

Dispatcher → Context Switching } 다름



* Dispatcher는 처리속도가 빨라야함

* Dispatcher Latency

↳ Time to stop process and start another running

초씩 3번

```
root@api_server:/nas_test/nas_test/django-qrcode/SimpleAPI# vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 5  0  16468 505116  79524 8429152    0    0     1     9    3    0  0  1 99  0  0
 0  0  16468 505108  79524 8429136    0    0     0  1466 1642 3075  0  0 98  2  0
 0  0  16468 504256  79524 8429152    0    0     0     0 1626 2933  1  1 98  0  0
```

* Scheduling Criteria : 무엇을 위해 스케줄링을 하는가

- CPU Utilization : CPU 이용률 극대화
- Throughput: 단위시간내에 처리하는 프로세스의 개수늘리기
- **Turnaround time**: 프로세스가 도착한 시간부터 끝난시간까지
- **Waiting Time** : 프로세스가 Ready Queue 에서 대기하고있는 시간
- Response Time

* Solutions for Scheduling Problem

- FCFS: First Come First Served
- SJF: Shortest Job First (SRTF 라고도 부름, Shortese Remaining Time First)
- RR : Round-Robin
  - Time Sharing Scheduling via Time Quantum

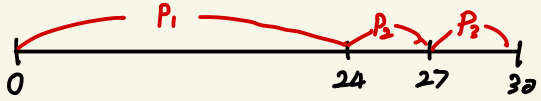
* FCFS Scheduling → Non-Preemptive

- First Come First Served
- The process that requests the CPU first
- Easily implement via FIFO Principle ADT
  Like Queue, Singly L_L
- Problem

Process	Burst Time
P_1	24
P_2	3
P_3	3

↳ Consider P_1, P_2, P_3 Arrival Time 0

- Gant Chart



- Waiting Time

P_1 : 0 P_2 : 24 P_3 : 27

P_2: Executed in 24, Arrived in 0
P_3: Executed in 27, Arrived in 0

∴ Total Waiting Time: 0 + 24 + 27 = 51
Average Waiting Time: 51 / 3 = 17

- Turnaround Time

P_1: 24 P_2: 27 P_3: 30

∴ Total Turn around Time: 24 + 27 + 30 = 81
Average Turn around Time: 81 / 3 = 27

* SJF Scheduling → Non-Preemptive or Preemptive

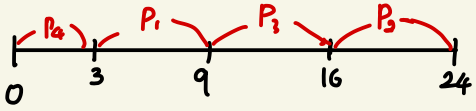
- Shortest Job First
- SJF associate with each process
  - ↳ length of processor's next CPU Burst
- when CPU is available assign it to the process that has the smallese CPU burst
- If two or more Process are even → tie with FCFS
  - ↳ Based on CPU Burst Time
- Problem

Process	Burst Time
P_1	6 ③
P_2	8 ④
P_3	7 ②
P_4	3 ①

.. Burse Time 기준 오름차순정렬을 하였지 괜찮을것같다.

- Gant Chart

→ Consider P_1, P_2, P_3, P_4 Arrival Time 0



- Waiting Time

P_1: 3 P_2: 16 P_3: 9 P_4: 0

- Total Waiting: 28
- Average " : 28/4 = 7

- Turnaround Time

P_1: 9 P_2: 24 P_3: 16 P_4: 3

- Total Turnaround: 52
- Average " : 52/4 = 13

SJF → Probably Optimal

- Burst Time 이 짧은것을 긴것 앞에 놓으면
  - 짧은 Burst time을 가진프로세스 Waiting Time이 줄어든다.
  - ∴ Avg Waiting time decrease
- 하지만 구현이 어렵다.
  - ↳ why? 다음 CPU의 CPU Burst Time을 알 방법X
  - ↳ 대안책: 예측을 한다.
    - ↳ 과거의 CPU Burst Time을 통해 예측한다.
    - 지수평균(exponential average)을 통해

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

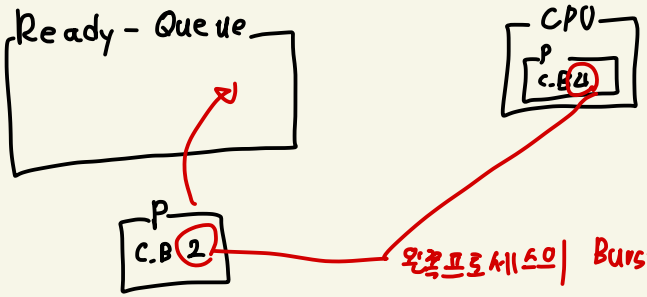
↳ 가중치를 주어서 구하는 방식

- t_n : n번째 CPU Burst Time
- τ_{n+1} : 예측값
- α : $0 \le \alpha \le 1$

⇓

사실상 Non-Preemptive는 구현이 힘들

* Preemptive SJF



완료프로세스이 Burst Time이 더작다.

⇓

CPU에있던 프로세스를 Ready Queue로,

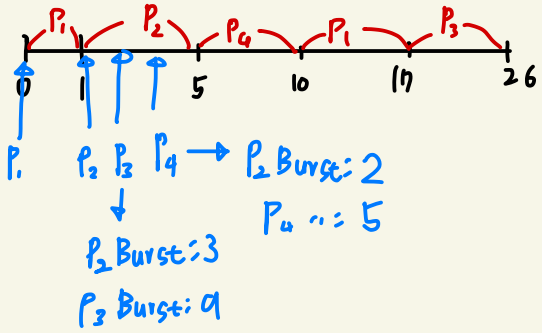
들어온 프로세스를 CPU로

* SRTF Scheduling

- Shortest Remaining Time First Scheduler

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

- Gant Chart



- Waiting Time

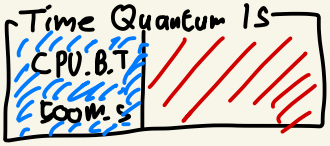
P_1: 10−1 P_2: 1−1 P_3: 17−2 P_4: 5−3

= 9+0+15+2=26

*Round Robin → Preemptive

→ Preemptive FCFS with time Quantum

- Time Quantum : 되게 작은 단위의 시간이다. 일반적으로 10~100ms
  - ↳ 원형 Queue로 구현
- 발생할 수 있는 현상
  - ① Process CPU Burst가 Time Quantum 보다 짧을 수 있을



→ Ready Queue에 Process가 있으면 다음실행, 없으면 IDLE

  - ② Process CPU Burst가 Time Quantum 보다 긴 경우
    - OS에 interrupt를 건다.
    - Context Switching
    - Process는 Ready Queue의 tail로 들어간다.

Process	Burst Time
P_1	24
P_2	3
P_3	3

→ with time Quantum 4ms

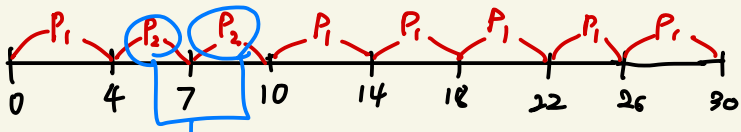
- Waiting time

P_1 : 10 - 4 = 6

P_2 : 4

P_3 : 7

Total 17 Avg: 5.66



→ Time Quantum 보다 짧은 CPU Burst Time 실행 후 Ready Queue의 다음 프로세스가 실행된다