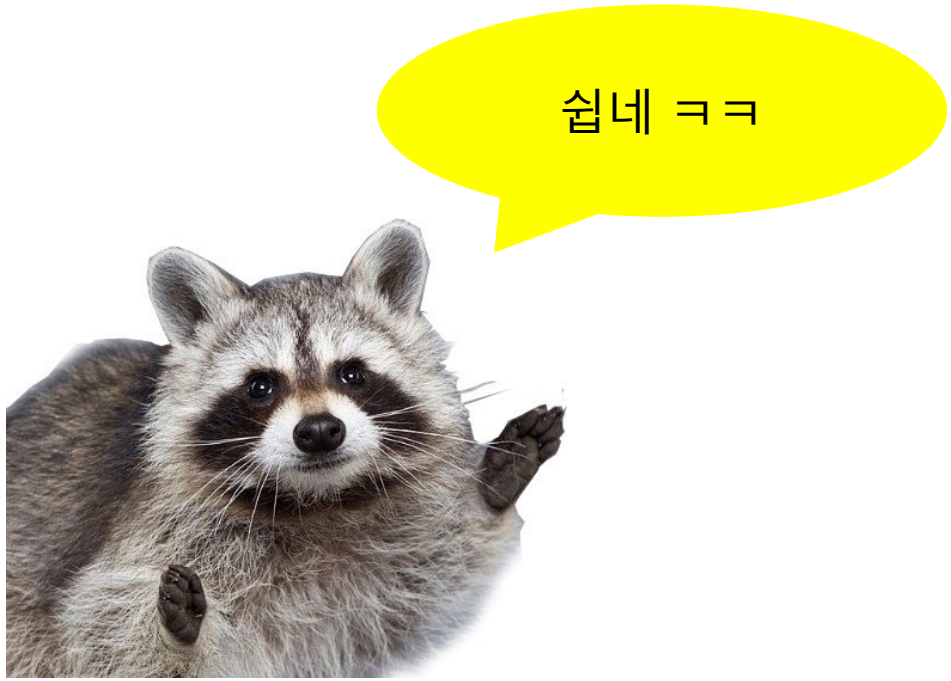
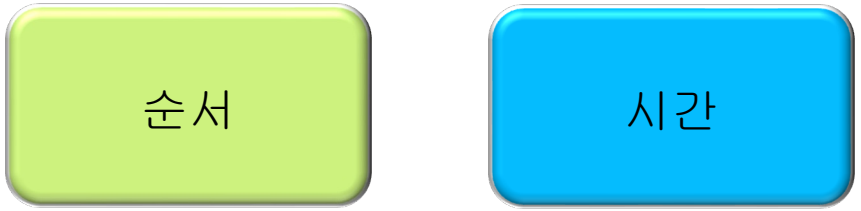
벼리와 함께하는 cpu프로세스 스케줄링



스케줄링이란?

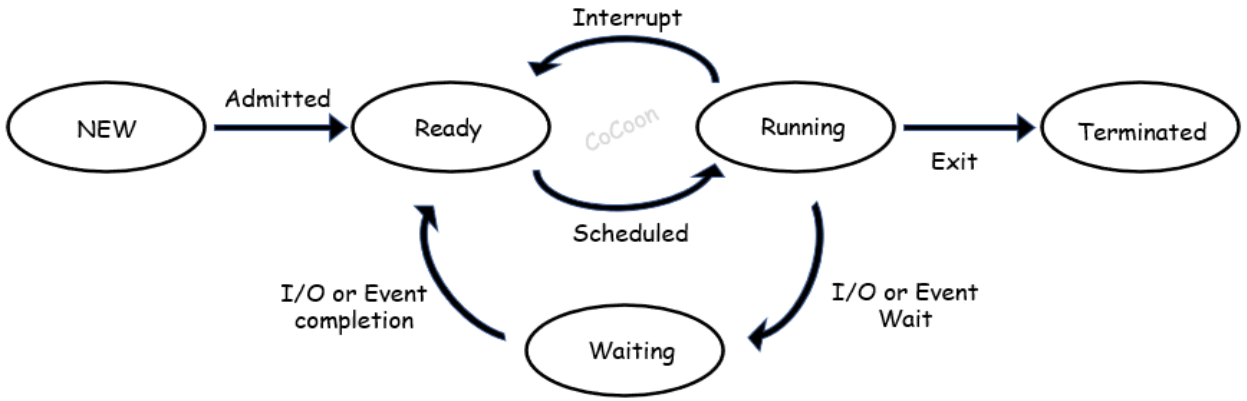
- 프로세스가 작업을 수행을위해 스케줄러로부터 cpu를 할당 받음
- 정해진 규칙에 의해 할당을 받을수 있음



- 할당은 -> 운영체제에서 구현 (프로세스에게 효율적으로 자원을 할당하기 위한 정책)

스케줄링이란?

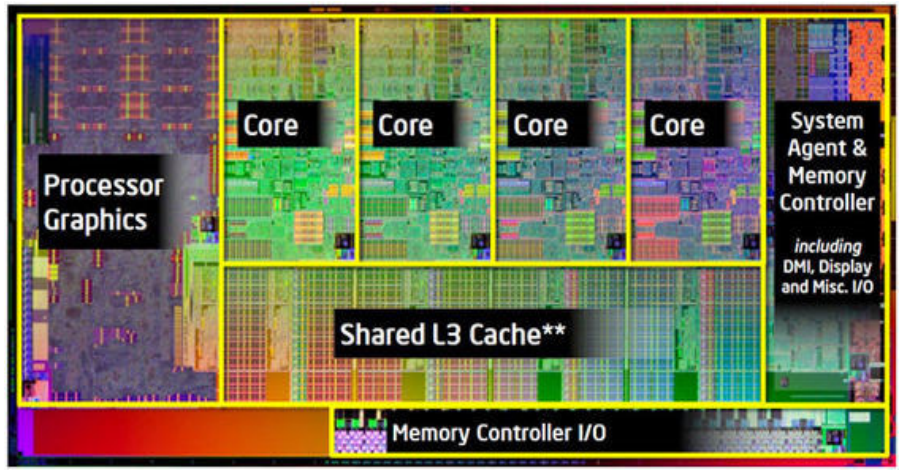
- Cpu

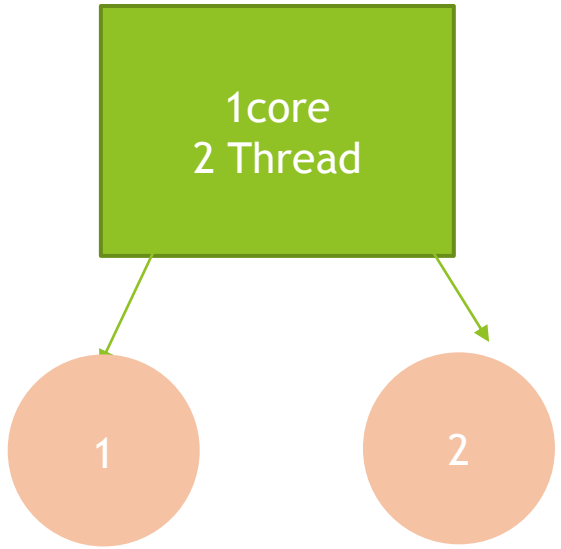


위와 같이 프로세스의 시행을 요청하면 레디 큐에 들어옴
시행-> 러닝
중단-> 웨이팅
재시행 레디큐-> 러닝
종료 러닝->종료

스케줄링이란?

- 왜필요할까?





- 코어

코어는 CPU 역할을 수행하는 블록

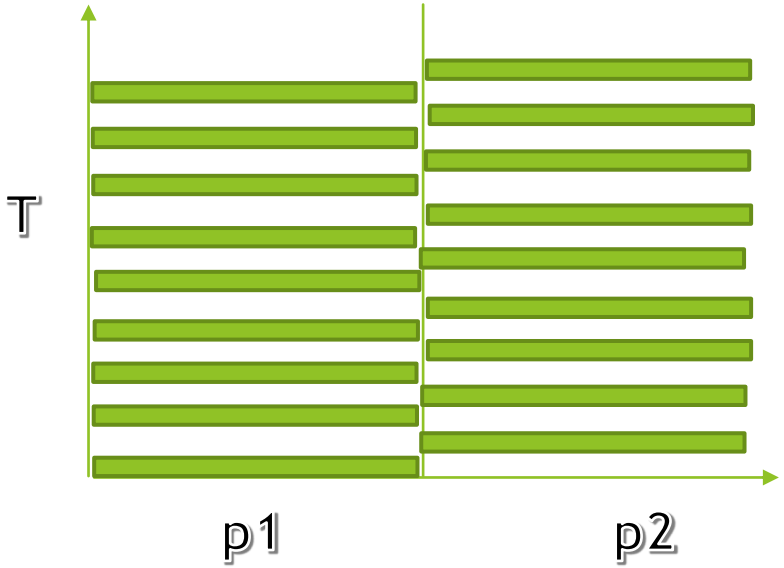
한 개의 CPU 칩 안에 한개의 코어=싱글코어라 2개 듀얼코어 4개 쿼드코어

- 쓰레드

-소프트웨어 관점에서 논리적 작업 처리단위. Core안에 연산을 처리해주는 작업단위.

스케줄링이란?

- Cpu의 병렬처리



Cpu가 1개의 코어를 가졌을때
어떤식으로 병렬처리를 할까?

사실은 한개씩만 처리하고 있다
너무 빠른 연산으로 우리는 느끼지
못햇을뿐 …

Ex) 오늘도 쇼핑을하면서, 유튜브 노래를
듣는 당신.

- 하이퍼 스레딩

- 시대가 발전하면서 일반적인 cpu는 멀티 코어를 가지게 됨.
- 병렬다운 병렬연산으로 -> 코어1개당 1개의 프로세스를 작업시키면서
- 코어안에 있는 쓰레드로 1개의 코어의 연산을 병렬연산을 해주는 기술
- Ex 1코어당 3쓰레드가 있을때 3개의 쓰레드가 프로세서를 병렬연산해줌
- 하이퍼 쓰레딩라는 기술은 인텔에서 만든 기술

목적

공정한 스케줄링	모든 프로세스에게 공정하게 할당을 해야함
응답시간 최소화	대화식 사용자에게는 최대한 응답시간(response time)을 빠르게 함
반환시간 최소화	프로세스를 제출한 시간부터 완료시까지 걸리는 반환시간(turn around time)을 최소화 한다.
대기시간 최소화	프로세스 준비 상태 큐에서 대기하는 시간을 최소화 해야함 앞에서 처리가 늦어지면 뒤에서 부하가 생기기 때문에 빠르게 처리해야함.
우선 순위 제도	먼저 처리해야 하는 것에 우선 순위를 부여해서 먼저 처리 함.
처리량 극대화	단위시간당 할 수 있는 처리량을 최대화 한다.
균형 있는 자원 사용	자원들이 유휴 상태에 놓이지 않도록 골고루 사용하게 함.
무한 연기 회피	자원을 사용하기 위해 무한정 연기하는 경우를 회피

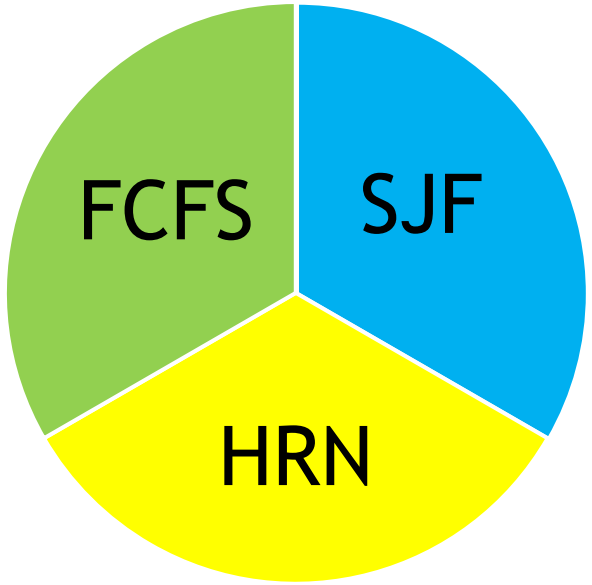
용어 정리

- 1)응답 시간(Response Time)
  - 대화식 시스템에서 요청후 응답이 오기 시작할때까지의 시간

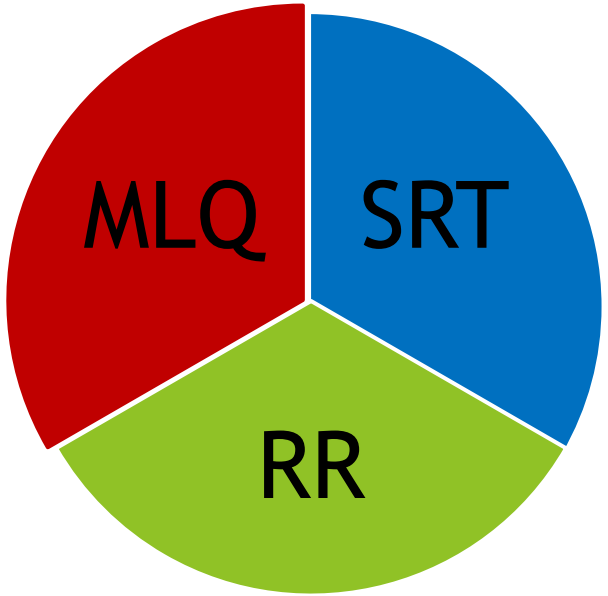
- 2)반환시간(Turnaround Time)
  - 프로세스가 시작해서 끝날 때까지 걸리는 시간

- 3)대기시간(Waiting Time)
  - 프로세스가 준비 큐 내에서 대기하는 시간의 총합
- 4)처리량
  - CPU의 단위 시간당 처리하는 프로세스 개수
- 5)유휴상태
  - 장치는 조작 가능한 상태에 있으나 사용되고 있지 않은 동안의 시간(idle time)이나, 장치가 작업을 개시하기 위한 명령을 대기하고 있는 상태

스케줄링 알고리즘



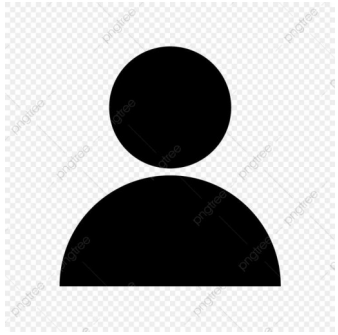
비선점 알고리즘



선점 알고리즘

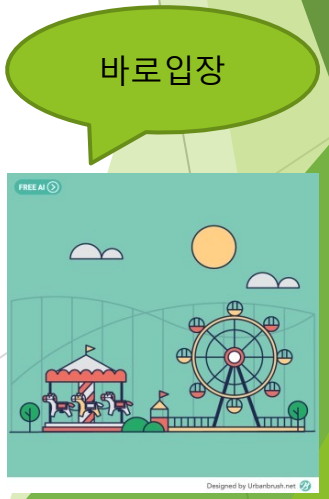
자유이용권

선점



패스트 패스

비선점



비선점 알고리즘

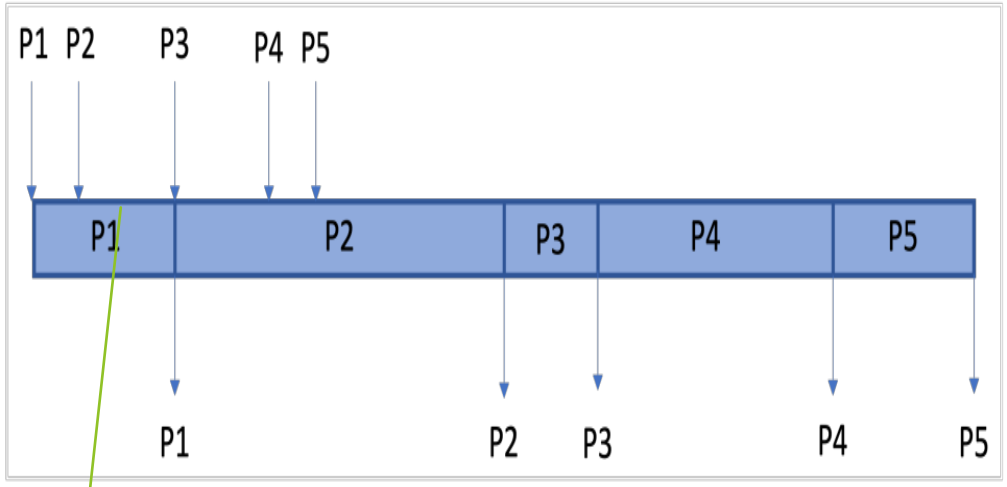
- 1)FCFS

-선입 선출 알고리즘

단점

-평균 응답시간이 김

--중요하지 않은 작업이 중요한 작업을 기다리게 할 수 있다.



2시간

PID	Arrival Time	Burst Time (BT)	Waiting Time (WT = TT- BT)	Turnaround Time (TT)
P1	0	3	0	3
P2	1	7	2	9
P3	3	2	7	9
P4	5	5	7	12
P5	6	3	11	14

평균대기시간 = (0+2+7+7+11) / 5 = 8.8

평균 반환시간 = 11.75

비선점 알고리즘

- 2)SJF스케줄링

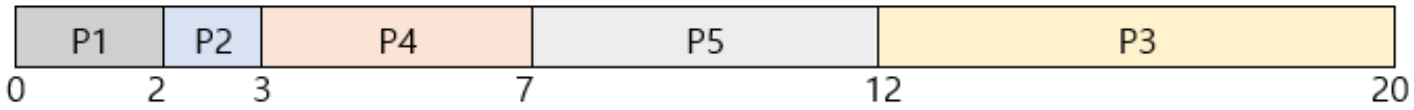
평균대기시간이 줄음

점유 시간이 가장 짧은 프로세스에 먼저 할당

단점

기아상태(Starvation)에 빠질수 있다.

	도착시간	수행시간	대기시간	소요시간
P1	0	2	0	2
P2	0	1	2	3
P3	0	8	12	20
P4	0	4	3	7
P5	0	5	7	12

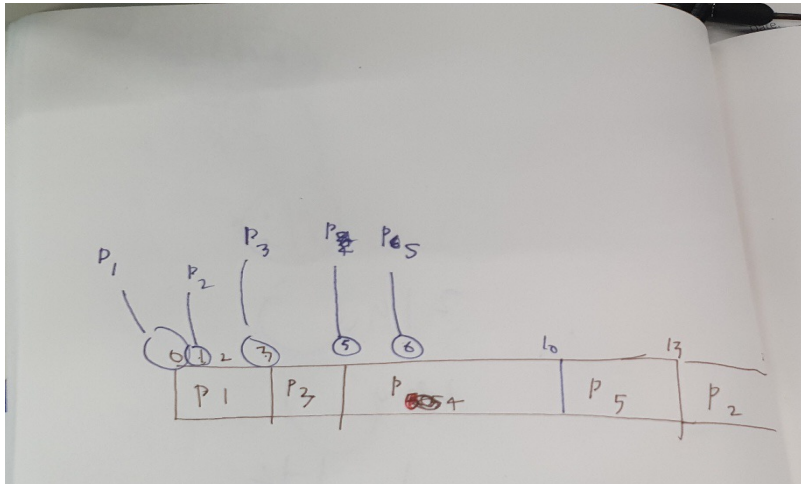


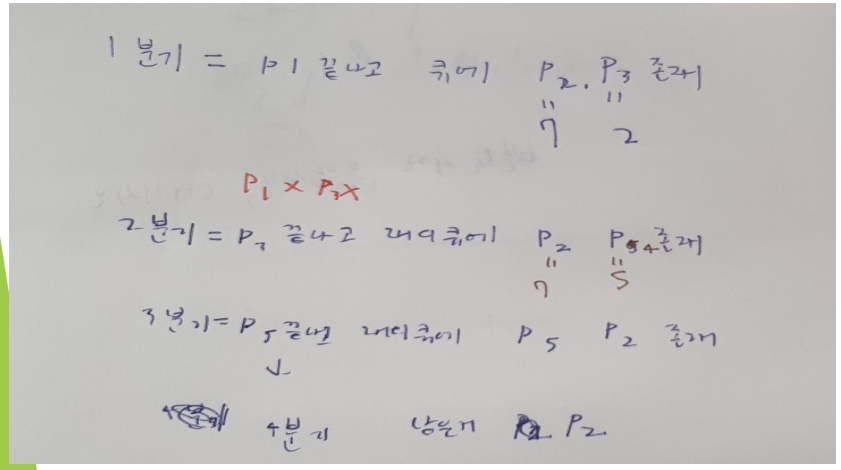
평균 소요(반환)시간: 8.8
평균 대기시간: 4.8

비선점 알고리즘

- 2)SJF스케줄링

Fcfs처럼 응답시간이 다르면 어떻게 될까?





PID	Arrival Time	Burst Time (BT)	Waiting Time (WT = TT- BT)	Turnaround Time (TT)
P1	0	3	0	3
P2	1	7	12	12+7=19
P3	3	2	0	2
P4	5	5	0	5
P5	6	3	4	4+3=17

평균대기시간 = (0+12+0+0+4) / 5 = 3.2

평균 소요(반환)시간: 11.5

비선점 알고리즘

- 3)HRN스케줄링

-SJF 스케줄링 기법의 약점인 긴 작업과 짧은 작업의 지나친 불평등을 보완한 스케줄링 기법

- 대기 시간이 큰 경우에는 우선 순위가 높아짐
- 서비스 받을 시간이 분모에 있으므로 짧은 작업의 우선 순위가 높아짐

	도착시간	수행시간	대기시간
P1	0	2	0
P2	0	1	2
P3	0	8	12
P4	0	4	3
P5	0	5	7

$$\text{응답비율} = \frac{\text{대기시간} + \text{예상 실행시간}}{\text{예상 실행시간}}$$

P1=1, P2=1.5, P3= 2.5
P4= 1.75 P5= 2.4

순서
P1 ->P2 - > P4 -> P5-> P3

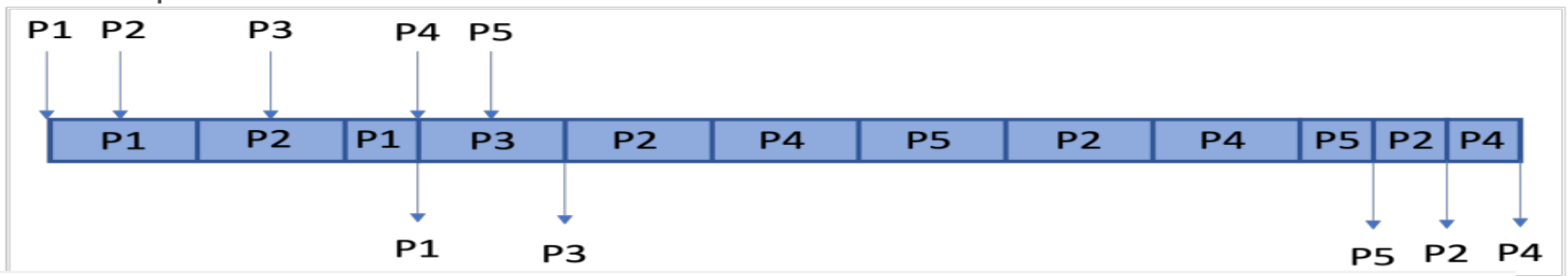
평균 소요(반환)시간: 8.8
평균 대기시간: 4.8

선점 알고리즘

Time quantum이란
RR에 사용되는 규칙을 말함

- 1)RR

- 사용 제한 시간을 두어 프로세스가 돌아가며 선점하는 알고리즘
- Time quantum이 매우 크면 FCFS와 동일하게 동작한다.



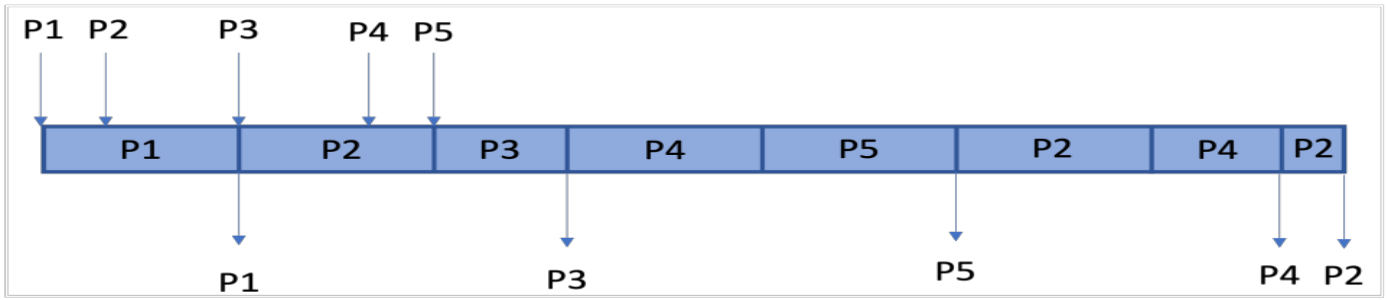
Time quantum=2

PID	Arrival Time	Burst Time (BT)	Waiting Time (WT = TT- BT)	Turnaround Time (TT)
P1	0	3	2	5
P2	1	7	11	18
P3	3	2	2	4
P4	5	5	10	15
P5	6	3	9	12

평균대기시간 = (0+2+7+7+11) / 5 = 6.8
평균 반환시간 = **(5 + 18 + 4 + 15 + 12) / 5 = 10.8**

선점 알고리즘

- Time quantum=3



PID	Arrival Time	Burst Time (BT)	Waiting Time (WT = TT- BT)	Turnaround Time (TT)
P1	0	3	0	3
P2	1	7	12	19
P3	3	2	3	5
P4	5	5	9	14
P5	6	3	5	8

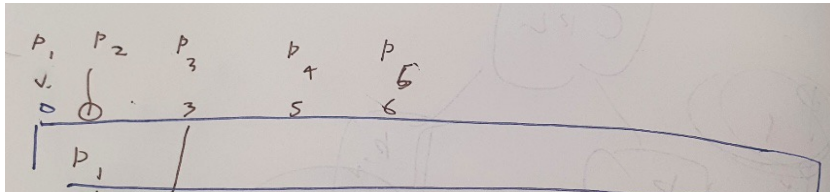
평균대기시간 = 5.8

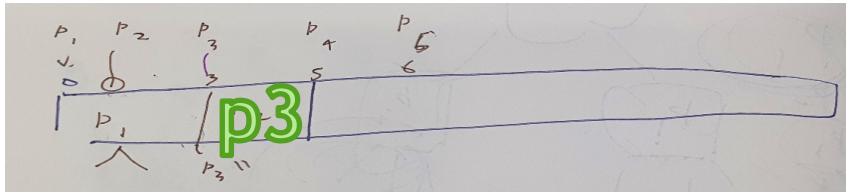
평균 반환시간 = **(3 + 19 + 5 + 14 + 8) / 5 = 9.8**

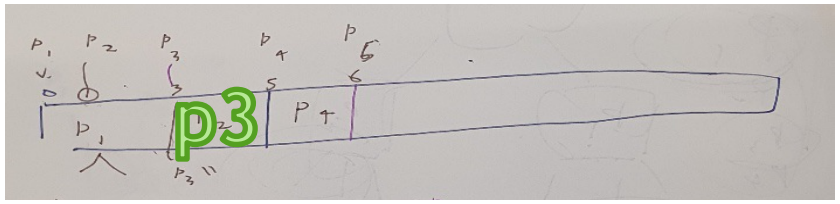
선점 알고리즘

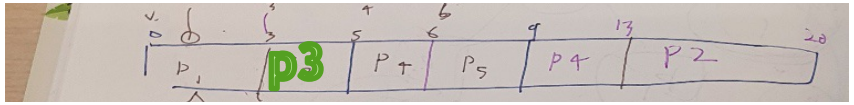
- 2)SRT

- SJF의 선점형 버전
- 최단 잔여시간을 우선으로 하는 스케줄링.
- 진행 중인 프로세스가 있어도, 최단 잔여시간인 프로세스를 위해 sleep 시키고 짧은 프로세스를 먼저 할당한다.





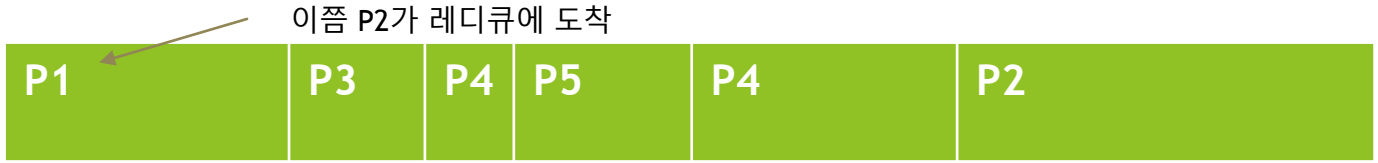




시간	시행중 프로세스	레디큐 남은시간
0	P1	P1=3
1	P1	p1=2,p2=7
3	P3	P3=2,p2=7
5	P4	P4=5,p2=7
6	P5	P4=4,p2=7, p5=3
9	P4	P4=4, p2=7
13	p2	P2=7

선점 알고리즘

- 기아상태
- SRT알고리즘을 생각해보자



PID	Arrival Time	Burst Time (BT)	Waiting Time (WT = TT- BT)	Turnaround Time (TT)
P1	0	3	0	3
P2	1	7	12	19
P3	3	2	0	2
P4	5	5	3	(3+1+4)=7
P5	6	3	0	3

만약 7시간 미만인 프로세스가 계속 들어오면?

버근가....

선점 VS비선점

비선점

- 프로세스가 CPU를 할당 받으면 해당 프로세스가 완료될 때까지 CPU를 사용
- 모든 프로세스에 대한 요구를 공정하게 처리할 수 있음
- 일괄 처리 방식에 적합, 중요한 작업(짧은 작업)이 중요하지 않은 작업(긴 작업)을 기다리는 경우가 발생할 수 있음

선점

- 하나의 프로세스가 실행 중 일 때, 우선 순위가 높은 다른 프로세스가 CPU를 강제로 빼앗아 사용할 수 있음
- 우선 순위가 높은 프로세스를 빠르게 처리할 수 있음
- 빠른 응답 시간을 요구하는 대화식 시분할 시스템이나 처리 시간이 제한 되어 있는 실시간 시스템에 유용.
- 많은 오버헤드를 초래.

오버헤드:는 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간 · 메모리 등을 말함
(RR,SRT의 작업을 생각해보자!!)