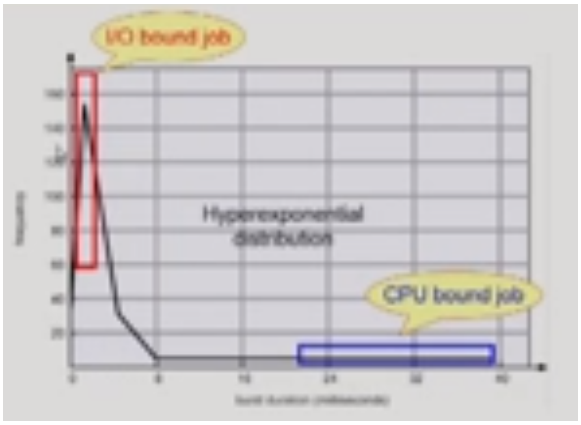
5. CPU Scheduling 1

속성

1. CPU and I/O Bursts in Program Execution

- 컴퓨터 프로그램이 실행 될때는 CPU에서 instruction(지침) 실행 단계(CPU Bursts) ↔ I/O 같이 오래 걸리는 작업 이 번갈아가며 작업 (모든 프로그램의 경로; 프로그램 종류마다 조금씩 다르다.)
    - I/O가 빈번이 끼어들지 않는 프로그램은 CPU Burst가 길게 나타남
    - 사람과 interection하는 경우 중간중간 화면출력, 키보드 입력 등에 의해 CPU를 연속적으로 쓰는 단계가 짧아지게 된다.

2. CPU-burst Time의 분포



- CPU사용 시간이 짧다는 것은 I/O가 중간에 빈번이 끼어든 프로그램이라고 할 수 있다.
  - 사람하고 interection 하는 작업이기 때문에 이런 작업한테 CPU를 늦게 주게 되면 사람이 오래 기다려야하고, 답답함을 느낄 수 있다.
- CPU Bursts가 길다는 것은 I/O같은 작업이 거의 없거나 마지막에 한번 하는 등의 프로그램이라고 할 수 있다.

3. CPU Scheduler & Dispatcher

- `CPU 스케줄러`라는 것은 현재 Ready queue에 있는 CPU를 얻고자 하는 프로세스 중에서 어떤 프로세스에 CPU를 줄 것인지 정하는 매커니즘이다.
  - **누구?**
  - **CPU를 계속 쓰게 할 것인가 뺏어올 것인가?**
    - CPU Bursts가 긴 프로세스 하나가 CPU 하나를 잡고 안놔주면 뒤에 작업들이 기다리게 된다.

4. Scheduling Algorithms

1. non-primitive 스케줄링(비선점형)

- 강제로 CPU를 빼앗지 않는 방법 : 그 프로그램이 다 쓰고 나갈 때 까지 CPU를 자진 반납할 때까지 보장해주는 방법

2. Primitive 스케줄링(선점형)

- timer interrup 등

5. Scheduling Criteria (성능 척도)

- 스케줄링 알고리즘이 어떤 것이 좋은 건지 평가할 수 있는 방법이 필요 : 성능 척도

1. 시스템 입장에서 성능 척도

1. **utilization 이용료** : CPU 하나로 최대한 일을 많이 시키는 것
2. **Throughput 처리량, 산출량** : 주어진 시간동안 몇개의 작업을 완료 했는지

2. 프로그램(사용자, 프로세스) 입장에서 성능 척도

> 가능하면 내(사용자, 프로세스)가 최대한 빨리 CPU를 받고, I/O를 하러 가고 싶음.

1. **Turnarround Time(소요시간)** : 대기하다가 CPU에 들어와서 다 쓰고 나갈 때까지의 시간
2. **Waiting Time(대기시간)** : 줄 서서 기다린 시간
3. **Response Tme(응답시간)** : 처음 CPU를 얻기까지의 시간

   ❗ 선점형의 경우 CPU를 얻었다가 뺏겼다가 얻었다가 뺏겼다가를 반복 이런 경우 Waiting Timt은 계속 올라가지만, Response Time은 동일

   ❗ Ready Queue에 들어온 프로세스만 관심 대상임.

   ex) 중국집으로 비유 :

   - 이용료 : 주방장에게 최대한 많이 일을 시킴
   - 처리량 : 근무 시간동안 몇개의 짜장면을 만들었는지
   - 소요시간 : 대기하고 음식이 나와서 다 먹는데 나오는 시간
   - 대기시간 : 기다린 시간
   - 응답시간 : 첫번 째 음식이 나올 때까지 기다리는 시간

6. Scheduling Algorithms

1. FCFS(First- Come First-Served)_비선점형

- 먼저 온 고객을 먼저 서비스 해주는 스케줄링 방법
- 사람의 세계에서는 FCFS 방법을 사용 (은행, 커피숍 등)
- 상황에 따라서 효율성이 뛰어나진 않음 (CPU를 오래 쓰는 프로세스가 있으면 대기 시간이 길어짐)

2. SJF(Shortest-Job-First)

- CPU를 짧게 쓰는 프로세스에게 CPU를 먼저 주는 스케줄링 방법
- 비선점형 방식의 경우 : 나(프로세스)보다 짧은 애가 와도 CPU를 뺏어가지 않음
- 선점형 방식의 경우 : 나(프로세스)보다 짧은 애가 오면 CPU를 뺏김.
- **Example of SJF**
  - 프로세스 도착시간이 다르고, CPU를 연속으로 쓰고자 하는 시간이 주어짐.
  - **비선점형(Non-Preemptive)** 방식의 경우 : 0초 시점에 도착한 프로세스가 CPU를 차지함. 그 후 같은 시간에 프로세스가 여러개 도착하면 짧은 시간을 가지고 있는 프로세스가 CPU를 얻는다.
  - **선점형(Preemptive)** 방식 : 0초에 도착하고, 7초의 CPU 사용 시간을 가진 P1과 2초에 도착하고 4초의 CPU 사용 시간을 가진 P2가 있다. P1은 도착과 동시에 CPU를 얻게 되지만 2초에 P2가 도착하고 나서 남은 시간을 비교하면 P1(5초) > P2(4초)이기 때문에 P1은 P2에게 CPU를 넘겨 주어야 한다.

    ➡️ 스케줄링이 이루어지는 시점이 다름! 비선점형은 현재 CPU를 가지고 있는 프로세스의 작업 이 끝나면 스케줄링이 이루어지지만 선점형은 Ready Queue에 새로운 프로세스가 도착하면 일어난다.
- **[단점]**
  - ***스타베이션 현상*** CPU 사용 시간이 짧은 걸 선호 : CPU 사용 시간이 길다면 CPU를 얻지 못할 수도 있음
  - 현재 내가 CPU를 얼마나 쓰고 나갈지 시간을 알 수 없다.
- **[다음 CPU Burst Time의 예측]**
  - 과거 CPU burst time를 이용해 추정
  - **Exponential averaging**

$$\tau(n+1) = \alpha . Tn + (1-\alpha) . \tau(n)$$

다음번 CPU 사용시간을 예측하는데 있어서 이전 사용시간을 (1-a) 만큼 반영하고, 이전전 사용 시간을 (1-a)**1/2 만큼 반영하는 듯 가중치를 다르게 한다.

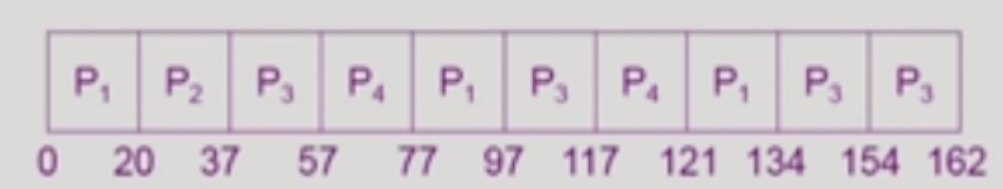
➡️ 예측치가 가장 적은 프로세스에 CPU를 할당

3. Priority Scheduling

- 우선순위 스케줄링 : 우선 순위가 높은 프로세스에게 CPU를 주겠다
- 비선점형 방식: 현재 프로세스가 작업을 끝낼 수 있도록 보장해 줌
- 선점형 방식: Ready Queue에 도착한 프로세스의 우선순위가 높다면 현재 CPU를 사용하는 프로세스가 CPU를 빼앗기게 된다.
- **[단점]**
  - ***스타베이션 현상*** 우선순위가 낮은 프로세스가 영원히 CPU를 얻지 못할 수도 있다.
- **[해결방안]**
  - ***aging*** 아무리 우선순위가 낮아도 대기시간이 길다면 우선순위를 높여준다.

4. Round Robin(RR)

- 현대 CPU는 주로 Round Robin 스케줄링 방식을 기반으로 하고 있다. CPU 사용시간이 짧은 프로세스와 긴 프로세스가 얼마나 CPU를 사용할지 알 수 없는 상황에서 섞여 있을 때 사용하기 적합하다.
- 선점형 방식이다.
- 프로세스에 CPU를 줄때 할당 시간(time quantum)을 주고, 할당 시간 후에는 timer interrupt에 의해 CPU를 빼앗기고, Ready queue에 들어간다.
- **[장점]**
  - 응답시간(response time)이 빠르다. CPU를 얻기 까지 시간이 빠르다.
  - q : 10 ~ 100 ms가 적당
- **[단점]**
  - q large : FCFS과 비슷하다.
  - q small : context switch가 빈번하게 발생
- 일반적으로 SJF 보다 `average turnaround`(작업이 끝나는 시간, 대기 시간도 포함)는 길지만 `response time`(처음으로 CPU를 받는 시간)은 짧아진다.


P1 P2 P3 P4 P1 P3 P4 P1 P3 P3
0 20 37 57 77 97 117 121 134 154 162