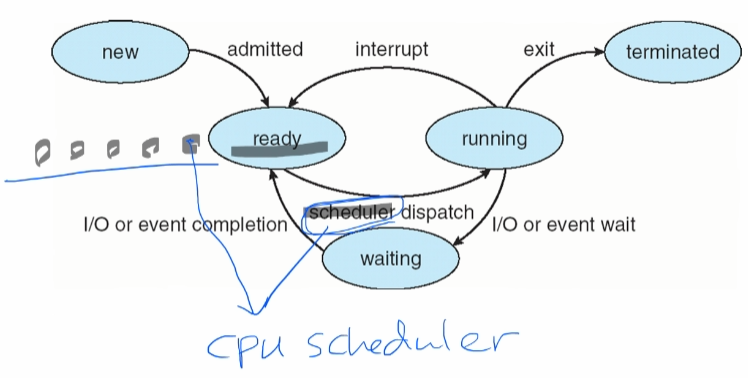
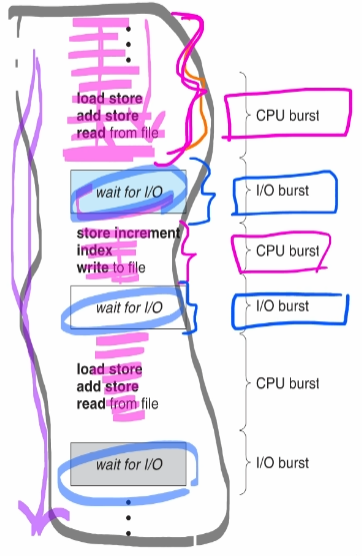
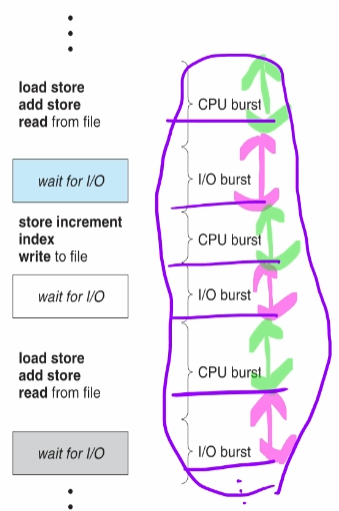
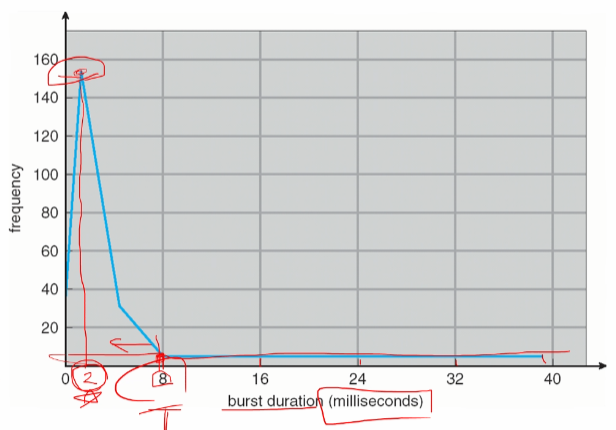
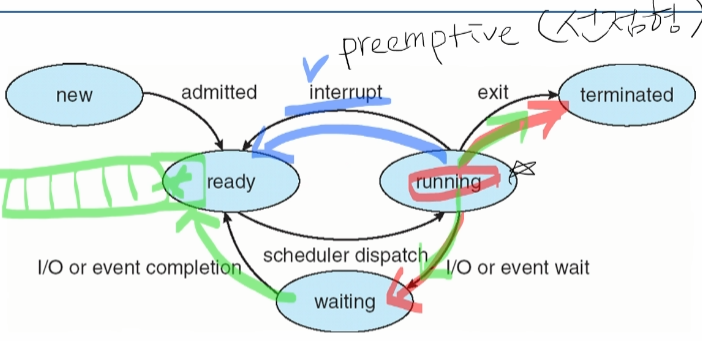
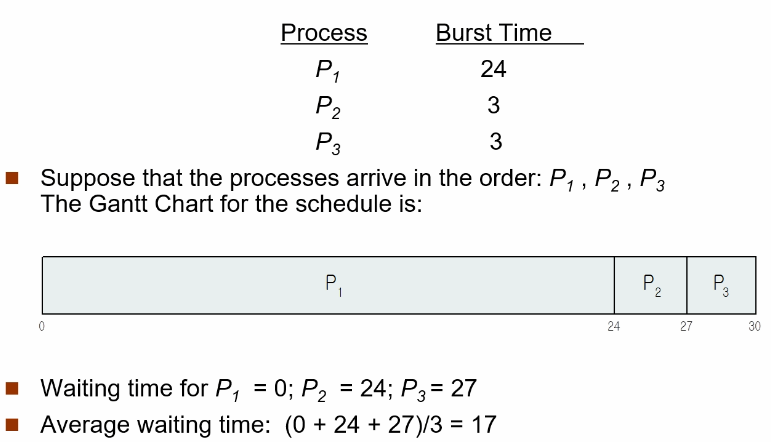
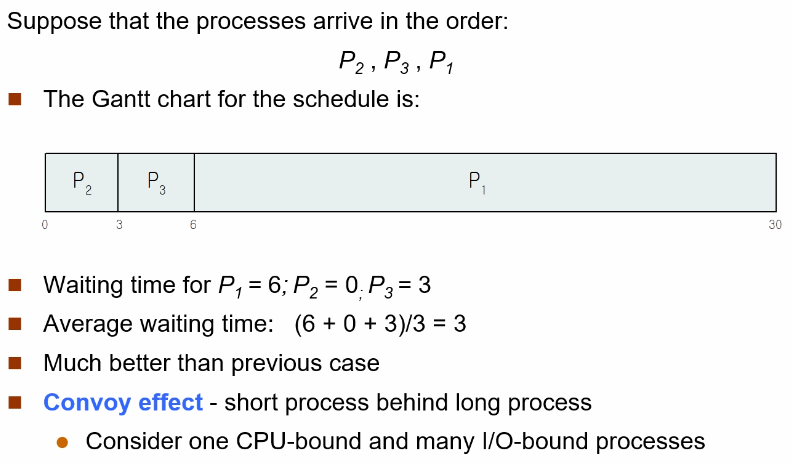
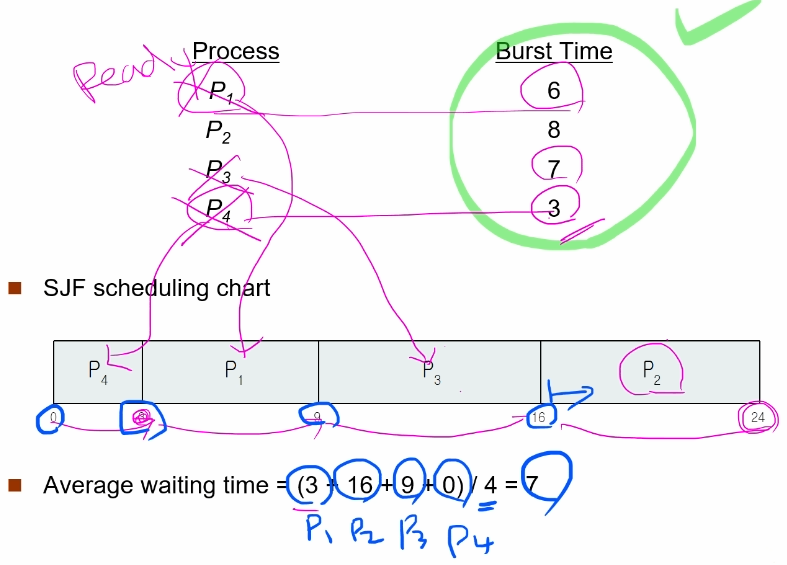
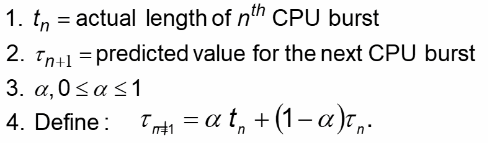
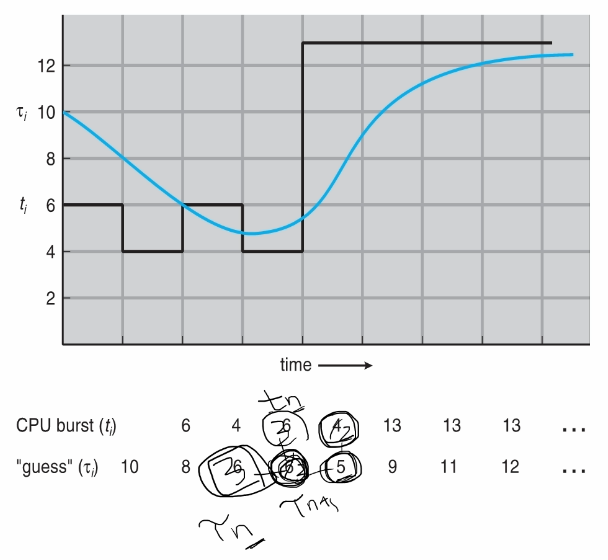
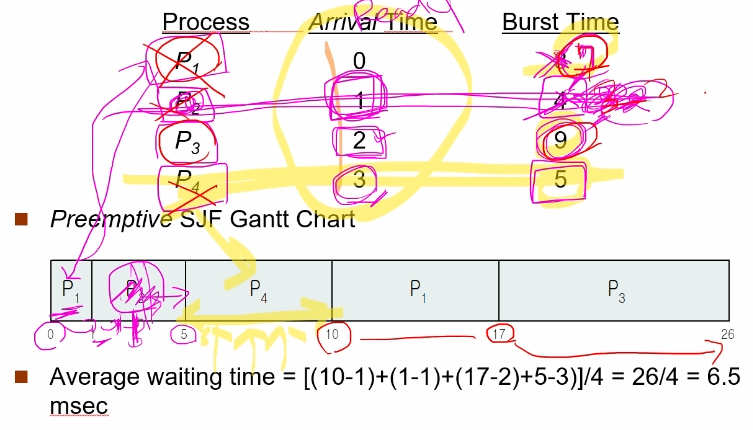
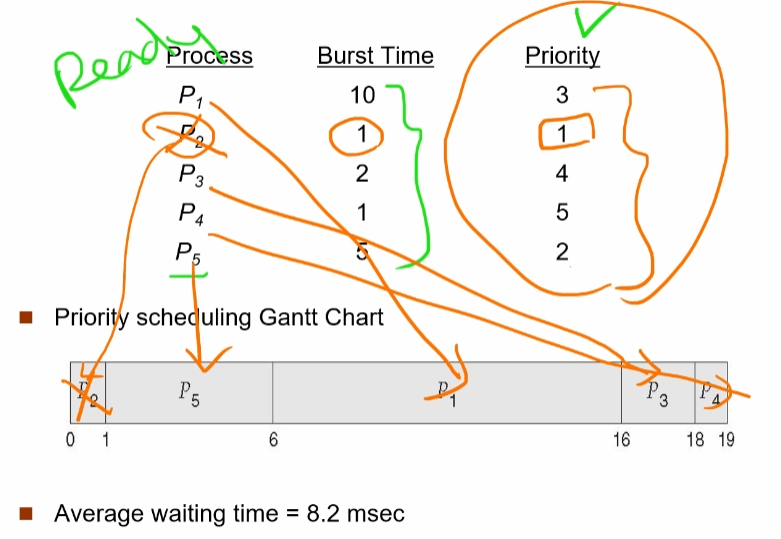
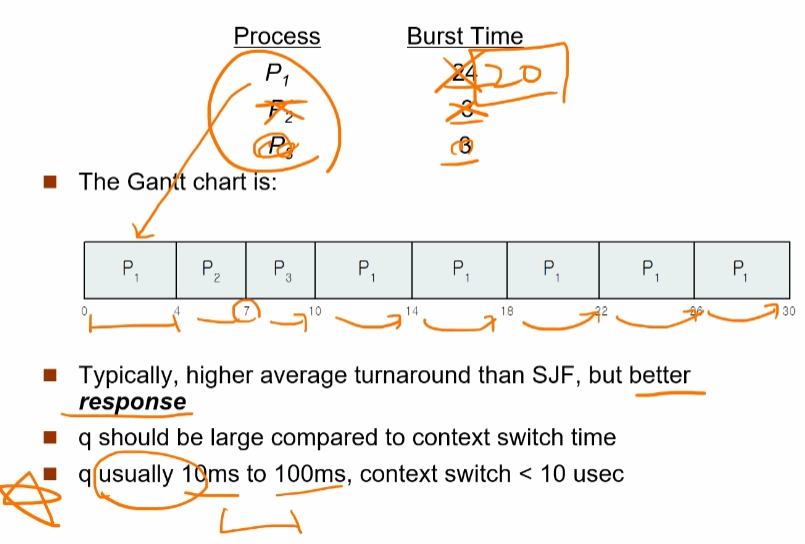
Chapter 5: CPU 스케줄링

Diagram of Process State



Schedulers

* Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU  
  단기 스케줄러 (또는 CPU 스케줄러) – 다음에 실행할 프로세스를 선택하고 CPU를 할당한다
  + Sometimes the only scheduler in a system
  + Short-term scheduler is invoked frequently (milliseconds) => (must be fast)  
    단기 스케줄러는 자주 호출된다 (밀리초) => 빨라야 함
* Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue  
  장기 스케줄러 (또는 잡 스케줄러) – 레디 큐로 가져올 프로세스를 선택한다
  + Long-term scheduler is invoked infrequently (seconds, minutes) => (may be slow)  
    장기 스케줄러는 자주 호출되지 않는다 (초, 분) => 느려야 함
  + The long-term scheduler controls the degree of multiprogramming  
    장기 스케줄러는 다중 프로그래밍의 정도를 제어한다
* Processes can be described as either:  
  프로세스는 다음 중 하나로 설명될 수 있다
  + I/O-bound process – spends more time doing I/O than computations, many short CPU bursts  
    I/O-bound process – 계산보다 I/O를 수행하는 데 더 많은 시간을 할애한다; 많고 짧은 CPU 버스트
  + CPU-bound process – spends more time doing computations; few very long CPU bursts  
    CPU-bound process – 계산하는 데 더 많은 시간을 할애한다; 드물고 매우 긴 CPU 버스트
* Long-term scheduler strives for good process mix

1. Basic Concepts; 기본 개념  
   - Maximum CPU utilization obtained with multiprogramming  
   멀티 프로그래밍으로 얻을 수 있는 최대 CPU 활용도  
   - CPU-I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait  
   CPU-I/O 버스트 주기 – 프로세스 실행은 CPU 실행과 I/O 대기의 주기로 구성된다  
   - CPU Burst followed by I/O burst  
   CPU 버스트 뒤이어 I/O 버스트  
   - CPU burst distribution is of main concern  
   CPU 버스트 분포가 중요점이다  
      
   I/O 버스트가 길면 I/O-bound 프로세스, CPU 버스트가 길면 CPU-bound 프로세스  
   - Histogram of CPU-burst Times; CPU 버스트 지속시간의 히스토그램  
   
   1. CPU Scheduler; CPU 스케줄러  
      
      1. Short-term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them  
         단기 스케줄러는 레디 큐에 있는 프로세스들 중에 선택하고 그들 중 하나에 CPU를 할당합니다
         1. Queue may be ordered in various ways  
            큐는 다양한 방법으로 명령 받을 수 있다
      2. CPU scheduling decisions may take place when a process:  
         CPU 스케줄링 결정은 프로세스가 다음과 같은 경우 발생할 수 있다
         1. Switches from running to waiting state  
            러닝 상태에서 대기 상태로 전환
         2. Switches from running to ready state  
            러닝 상태에서 레디 상태로 전환
         3. Switches from waiting to ready  
            대기 상태에서 레디 상태로 전환
         4. Terminates  
            종료
      3. Scheduling under 1 and 4 is nonpreemptive  
         스케줄링 1과 4는 비선점
      4. All other scheduling is preemptive  
         다른 모든 스케줄링은 선점
         1. Consider access to shared data  
            공유 데이터에 대한 접근 고려
         2. Consider preemption while in kernel mode  
            커널 모드에서 선점 고려
         3. Consider interrupts occurring during crucial OS activities  
            중요한 OS 활동 중에 발생하는 인터럽트 고려
2. Scheduling Criteria; 스케줄링 기준  
   - CPU utilization – keep the CPU as busy as possible  
    CPU 이용률 – CPU를 최대한 바쁘게 유지  
   - Throughput - # of processes that complete their execution per time unit  
    처리량 – 시간 단위당 실행을 완료한 프로세스 수  
   - Turnaround time – amount of time to execute a particular process  
    총 처리 시간 – 특정 프로세스를 실행하는 데 소요된 시간  
   - Waiting time – amount of time a process has been waiting in the ready queue  
    대기 시간 – 프로세스가 레디 큐에서 대기한 시간  
   - Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)  
    응답 시간 – 하나의 요구를 제출한 후 첫 번째 응답이 나올 때까지의 시간, 응답이 시작되는 데까지 걸리는 시간
3. Scheduling Algorithms; 스케줄링 알고리즘
   1. First-Come, First-Served (FCFS) Scheduling; 선입 선처리 스케줄링  
        
      
   2. Shortest-Job-First (SJF) Scheduling; 최단 작업 우선 스케줄링
      1. Associate with each process the length of its next CPU burst  
         각 프로세스에 다음 CPU 버스트 길이를 연관시킨다
         1. Use these lengths to schedule the process with the shortest time  
            프로세스를 가장 짧은 시간으로 스케줄링 하기 위해 이 길이를 사용한다
      2. SJF is optimal – gives minimum average waiting time for a given set of processes  
         SJF가 최적 – 주어진 프로세스 집합에 대한 최소 평균 대기 시간을 준다
         1. The difficulty is knowing the length of the next CPU request  
            어려움은 다음 CPU 요청의 길이를 아는 것
         2. Could ask the user  
            사용자에게 물을 수 있음
      3. Example of SJF  
         
      4. Determining Length of Next CPU Burst  
         다음 CPU 버스트의 길이 결정
         1. Can only estimate the length – should be similar to the previous one  
            길이만 추정 가능 – 이전의 길이와 유사해야 함
            1. then pick process with shortest predicted next CPU burst  
               그리고 다음 CPU 버스트 예측이 가장 짧은 프로세스를 선택한다
         2. Can be done by using the length of previous CPU bursts, using exponential averaging  
            다음 CPU 버스트는 일반적으로 측정된 이전의 CPU 버스트들의 길이를 지수 평균한 것으로 예측한다  
            
         3. Commonly, a set to 1/2  
            일반적으로 a는 1/2
         4. Preemptive version called shortest-remaining-time-first  
            선점형 SJF 알고리즘은 때때로 최소 잔여 시간 우선 스케줄링이라고 불린다
      5. Prediction of the Length of the Next CPU Burst  
         
      6. Example of Shortest-remaining-time-first
         1. Now we add the concepts of varying arrival times and preemption to the analysis  
            이제 다양한 도착 시간과 선점의 개념을 분석에 추가합니다
         2. 
   3. Priority Scheduling; 우선순위 스케줄링
      1. A priority number (integer) is associated with each process  
         우선순위 번호 (integer)는 각 프로세스에 연관되어 있다
      2. The CPU is allocated to the process with the highest priority (smallest integer = highest priority)  
         CPU는 우선순위가 가장 높은 프로세스에 할당된다 (가장 작은 정수 = 가장 높은 우선순위)
         1. Preemptive; 선점
         2. Nonpreemptive; 비선점
      3. SJF is priority scheduling where priority is the inverse of predicted next CPU burst time  
         SJF 알고리즘은 우선순위가 예상되는 다음 CPU 버스트의 역인 단순한 우선순위 알고리즘이다
      4. Problem = Starvation – low priority processes may never execute  
         문제 = 기아 상태 – 낮은 우선순위 프로세스들이 CPU를 무한히 대기하는 경우가 발생한다
      5. Solution = Aging – as time progresses increase the priority of the process  
         해결 = 노화 – 노화는 오랫동안 시스템에서 대기하는 프로세스들의 우선순위를 점진적으로 증가시킨다
      6. Example of Priority Scheduling  
         
   4. Round Robin (RR); 라운드 로빈
      1. Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.  
         각 프로세스는 CPU 시간 단위(time quantum q)를 얻는다, 보통 10-100 밀리초 동안이다. 이 시간이 지나면 프로세스가 선점되어 레디 큐의 끝에 추가된다.
      2. If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.  
         레디 큐에 n개의 프로세스가 있고 시간 할당량이 q이면, 각 프로세스는 최대 q 시간 단위의 덩어리로 CPU 시간의 1/n을 얻는다. 각 프로세스는 자신의 다음 시간 할당량이 할당될 때까지 (n-1)\*q 시간 이상을 기다리지는 않는다.
      3. Timer interrupts every quantum to schedule next process  
         타이머는 다음 프로세스를 스케줄링 하기 위해 모든 퀀텀을 중단한다
      4. Performance
         1. q large => FIFO
         2. q small => q must be large with respect to context switch, otherwise overhead is too high  
            time quantum이 문맥 교환 시간에 비해 커야 하지만 너무 커서는 안 된다
      5. Example of RR with Time Quantum = 4  
         
      6. Time Quantum and Context Switch Time  
         