

# CPU 스케줄링(=프로세스 스케줄링)

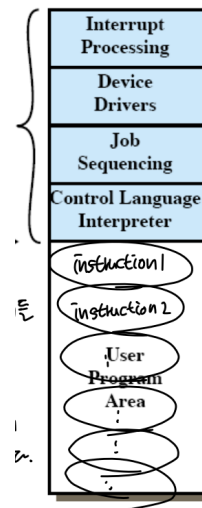
## OS 발전과정

### Serial Processing

- display lights, toggle switches, input device and printer과 같은 콘솔들로 기계가 작동했다.
- 운영체제가 프로세스를 생성하고 수행하는 것이 아닌 인간이 실행하는 것이었다.

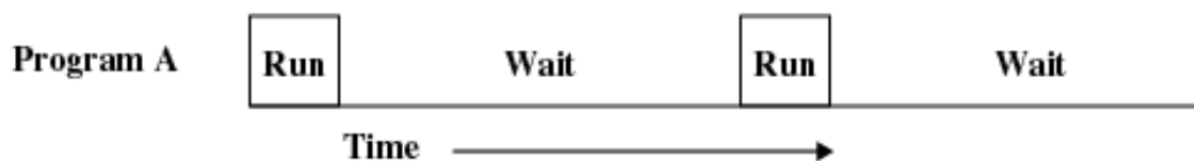
### Simple Batch Systems

- Serial Processing이 발전한 형태
- 비슷한 작업끼리 묶인 형태
- Monitor에 의해 컨트롤되었다.
  - Monitor : 일련의 events를 컨트롤하는 소프트웨어



### Uni-processing

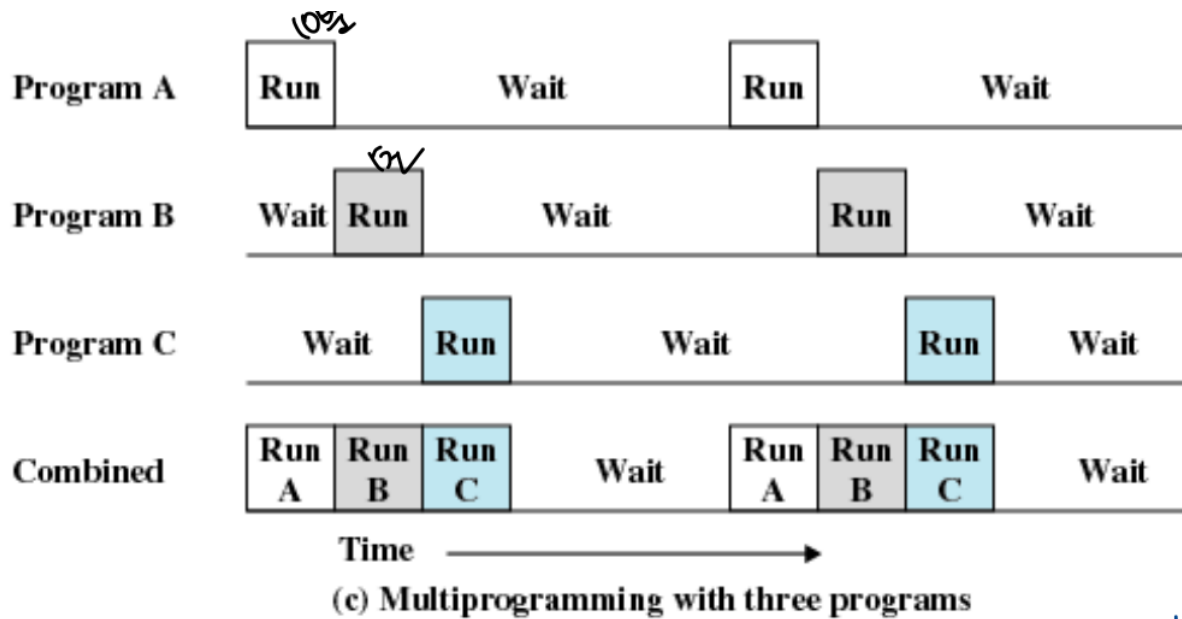
- Monitor가 Program을 실행시키는 방법
- 프로세서는 입출력 명령이 완료될때까지 기다리고 다시 실행하고를 반복
- 성능이 매우 안 좋았음



## Multiprogramming vs Timesharing

### Multiprogramming Systems

한 작업이 입출력을 기다리고 있으면 프로세서가 다른 작업으로 switch한다. 그러기 위해서는 메모리에 실행되는 프로그램이 메모리에 모두 적재되어야하고, 큰 메모리 공간이 필요하다.



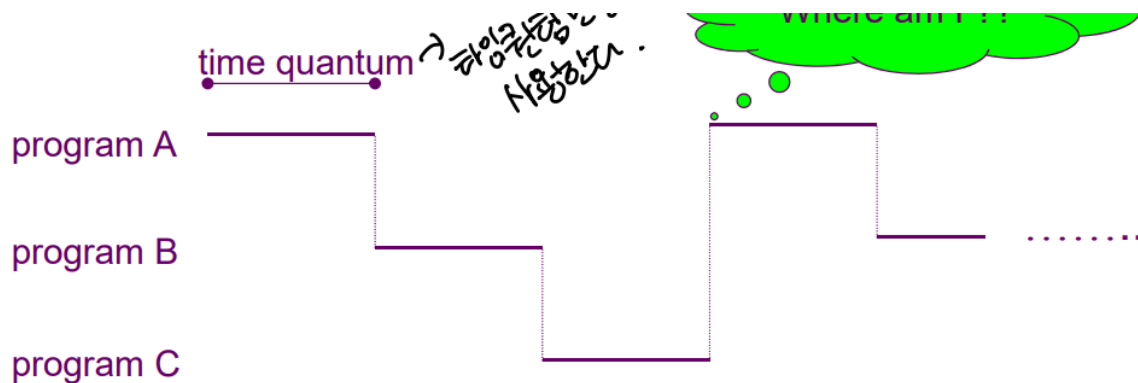
작업 A, B, C가 모두 메모리에 적재되므로 메모리의 크기가 커야한다.

### Time Sharing Systems (시분할 시스템)

: 멀티프로그래밍을 논리적으로 확장한 개념으로 프로세서가 다중의 프로그래밍을 동일한 시간만큼 번갈아가면 실행한다.

하나의 CPU는 같은 시점에서 여러 개의 작업을 동시에 수행할 수 없기 때문에, CPU의 전체 사용시간을 작은 작업 시간량으로 쪼개어 그 시간량 동안만 번갈아가면서 CPU를 할당하여 각 작업을 처리한다. 동일 시간만큼 각 프로그램이 실행되므로 평균 응답시간이 줄어드는 특징이 있다.

(시간간격은 time slice, time quantum이라고 불린다.)



### Multiprogramming VS Timesharing

현대 컴퓨터에서는 둘 다 사용되고 장단점이 있다.

#### Multiprogramming

- timesharing보다 overhead가 적다
- 처리율(시간동안 처리되는 작업의 수)이 좋고 낭비되는 시간이 적다.
- 하지만 응답시간(프로그램 실행 명령 ~ 실제로 실행되는 시간)은 Timesharing에 비해 크다.

#### Timesharing

- Multiprogramming보다 overhead가 크다
- 응답시간이 적다.
- 낭비되는 시간이 많아진다. (시간간격보다 처리시간이 짧아도 시간간격만큼 해당 프로그램이 CPU 제어권을 가지고 있기때문)

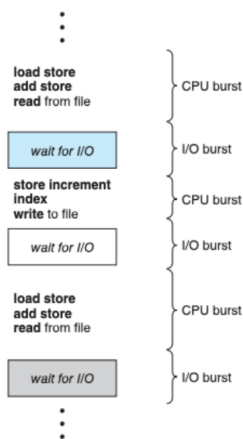
## CPU 스케줄링

다중 프로그래밍의 목적은 CPU 스케줄러를 통해 항상 실행 중인 프로세스를 가지게 하여 CPU 이용률을 최대화하는 것이다. 운영 체제는 CPU 제어권을 프로세스끼리 넘겨주면서, 컴퓨터를 보다 효율적으로 만든다. 어떤 프로세스가 I/O 입출력을 기다리고 있을 경우, 운영체제는 CPU 제어권을 회수하고 다른 프로세스에게 할당한다.

Process Scheduling = CPU Scheduling = Job Scheduling

### 스케줄링 단위

프로세스 실행은 CPU 실행과 I/O 대기의 사이클로 구성된다. CPU Burst로 프로세스의 실행 시작되고 뒤이어 I/O Burst가 발생하고 CPU Burst와 I/O Burst가 반복된다. 마지막 CPU Burst는 실행을 종료하기 위한 시스템 요청과 함께 프로세스가 마무리된다.-



- CPU Burst : 사용자 프로그램이 CPU를 직접 가지고 빠른 명령을 수행하는 단계. 즉 스케줄링 단위
- I/O Burst : 커널에 의해 입출력 작업을 진행하는 비교적 느린 단계. 프로세스는 Block되는 구간

이 CPU Burst의 지속시간을 확인해보면 컴퓨터 시스템 내에서 수행되는 프로세스의 대부분은 CPU Burst기간이 짧고 극히 일부만 긴 CPU Burst를 가진다.

**CPU를 한번에 오래 사용하기보다는 잠깐 사용하고 I/O 작업을 수행하는 프로세스가 많다.**

사용자에게 빠른 응답을 주기 위해서는 CPU Burst가 짧은 프로세스에게 우선적으로 CPU를 할당하는 게 바람직하다. 이때 이러한 문제를 해결하기 위해 CPU 스케줄링이 필요하다..

## CPU 스케줄러

CPU 스케줄러는 CPU 자원 관리를 담당하며, 프로세스 실행 우선순위를 결정하여 CPU 자원을 할당하는 역할을 수행한다.

**CPU 스케줄러가 CPU가 유휴 상태가 될 때마다, Ready Queue에 있는 프로세스 중 하나 선택해서 CPU를 할당한다.**

Ready Queue는 FIFO 방식일 수도, 우선순위 큐 등등으로 구현될 수 있다. 일반적으로 큐에 있는 레코드들은 프로세스의 프로세스 제어 블록(PCB)들이다.

## 스케줄링 기법

## 선점 및 비선점 스케줄링

cpu 스케줄링 결정은 4가지 상황에서 발생할 수 있다.

1. 한 프로세스가 실행 상태에서 대기 상태로 전환될 때 (I/O 발생)
2. 프로세스가 실행 상태에서 준비 완료 상태로 전환될 때 (인터럽트 발생)
3. 프로세스가 대기 상태에서 준비 완료 상태로 전환될 때 (I/O 종료)
4. 프로세스가 종료될 때

### 비선점 스케줄링

: 일단 CPU가 한 프로세스에 할당되면 프로세스가 종료하든지, 또는 대기 상태로 전환해 CPU를 방출할 때까지 점유한다. 위에서 1,4번의 경우에 해당된다.

ex) FCFS(first come first service), SJF(shortest job first), 우선 순위, HRN(highest response next)

### 선점 스케줄링

: 시분할 시스템에서 타임 슬라이스가 소진되었거나, 인터럽트나 시스템 호출 종료시에 더 높은 우선 순위 프로세스가 발생되었음을 알았을 때, 현 실행 프로세스로부터 강제로 CPU를 회수한다. 위에서 2,3번 경우에 해당된다.

ex) round robin, SRT, 선점 우선 순위 등의 알고리즘

## 스케줄링 기준

여러 cpu 스케줄링 알고리즘 중 하나를 선택하기 위한 CPU 스케줄링 비교 기준

- CPU 이용률 : 특정 기간동안의 CPU 이용시간
- 처리량 : 단위 시간당 완료된 프로세스의 개수
- 총 처리시간 : 프로세스 제출 시간과 완료 시간의 간격
- 대기시간 : 프로세스가 준비 큐에서 대기하면서 보낸 시간의 합
- 응답시간 : 하나의 요청을 보낸 후 첫번째 응답이 나올 때까지의 시간

대부분 알고리즘은 Trade-off 임으로 상황에 맞춰서 선택해야한다.

## 디스패처(Dispatcher)

: CPU 코어의 제어를 CPU 스케줄러가 선택한 프로세스에게 주는 모듈

### 디스패처의 역할

스케줄링 대상인 프로세스의 상태 전환을 수행하기도 하며, 이전 실행 프로세스의 컨텍스트(context)를 저장하고 새로운 프로세스의 컨텍스트를 로드하는 등의 작업을 수행합니다.

1. 한 프로세스에서 다른 프로세스로 문맥 교환하는 일
2. 사용자 모드로 전환하는 일
3. 프로그램을 다시 시작하기 위해 사용자 프로그램의 적절한 위치로 이동하는 일

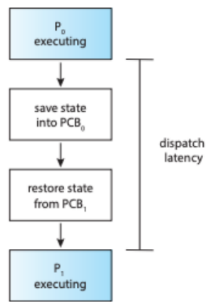


Figure 5.3 The role of the dispatcher.

### 디스패치 지연(Dispatch latency)

: 디스패처가 하나의 프로세스를 정지하고 다른 프로세스를 수행하는 데 걸리는 시간

### 문맥교환(Context Switch)

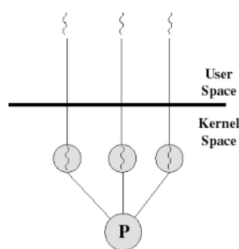
- 자발적 문맥 교환 : 현재 사용 불가능한 자원을 요청했을 때 프로세스가 CPU 제어를 포기한 경우 발생
- 복귀 문맥 교환 : 인터럽트나 예외 상황이 발생하여 프로세스가 CPU를 반환한 후, 이전에 실행되던 프로세스로 돌아가기 위한 문맥 교환이며, 대개 운영체제의 커널 모드에서 실행된다.
- 비자발적 문맥 교환 : 타임 슬라이스가 만료되었거나 우선순위가 높은 프로세스에 의해 선점되는 경우와 같이 CPU를 빼앗겼을 때 발생한다.

## 스레드 스케줄링

현대 운영체제에서는 프로세스가 아닌 **커널 수준 스레드**가 스케줄링된다.

사용자 수준 스레드는 스레드 라이브러리에서 관리되고 커널은 사용자 수준의 스레드를 알 지 못한다.

그래서 사용자 수준 스레드도 CPU상에서 실행되기 위해서는 LWP(Light Weight Process)를 통한 간접적인 방식일지라도 커널 수준 스레드에 매핑되어야한다.



### 경쟁범위

- 프로세스 경쟁 범위(Process Contention Scope, PCS)
  - 다대일과 다대다 모델에서 사용자 수준 스레드를 가용한 LWP에 스케줄링한다. 이때 동일한 프로세스 내 스레드 간 CPU를 경쟁하기 때문에 프로세스 경쟁범위라 한다.
- 시스템 경쟁 범위(System Contention Scope, SCS)
  - 일대일 모델에서 커널 수준 스레드를 CPU에 스케줄링한다. 이때 시스템상의 모든 스레드에서 CPU를 경쟁하기 때문에 시스템 경쟁 범위라 한다.

우리가 스레드 라이브러리가 사용자 수준 스레드를 가용한 LWP 상에서 스케줄 한다고 말하는 경우, 실제로 스레드가 CPU 상에서 실행중이라는 것을 의미하지 않는다. **실제로 CPU상에서 실행되기 위해서는 운영체제가 LWP의 커널 스레드를 물리적인 CPU 코어로 스케줄링 하는 것을 필요로 한다.** 이때, CPU상에 어느 커널 스레드를 스케줄 할 것인지 결정하기 위해서 커널은 시스템 경쟁 범위를 사용하여 커널 스레드를 스케줄링 한다.

#### 사용자 수준 스레드와 커널 수준 스레드의 연관관계

##### ▼ ChatGPT에 질문한 것들 ( 정확한 정보가 아닐수도 있습니다,,)

- 스레드는 CPU이용의 기본단위야?
  - 네, 스레드는 CPU 이용의 기본 단위 중 하나입니다. CPU는 한 번에 하나의 작업만 처리할 수 있으므로, 멀티스레딩을 통해 여러 작업을 동시에 처리할 수 있습니다. 스레드는 하나의 프로세스 내에서 실행되는 실행 흐름으로, 각각의 스레드는 CPU 시간을 할당받아 작업을 수행합니다. 따라서, 스레드는 CPU 이용의 기본 단위 중 하나로 볼 수 있습니다.
- LWP가 뭐야?
  - LWP는 Light-weight Process의 약어로, 가벼운 프로세스를 의미합니다. 이 용어는 주로 유닉스 및 유닉스 계열 운영 체제에서 사용되며, 스레드를 구현하는 방식 중 하나입니다. LWP는 스레드를 모방하여 구현되었으며, 사용자 수준에서 스케줄링되기 때문에 프로세스보다 빠르게 생성 및 종료될 수 있습니다. LWP는 스레드와 유사한 기능을 제공하며, 여러 개의 LWP가 하나의 프로세스 내에서 동작할 수 있습니다. 이렇게 하나의 프로세스 내에서 여러 개의 LWP가 동작하면, 각각의 LWP는 별도의 스레드처럼 동작하면서도 다른 LWP와 공유하는 자원(예: 메모리, 파일 등)에 접근할 수 있습니다.
- 다대일과 다대다 모델에서 사용자 수준 스레드를 가용한 LWP에 스케줄링한다. 여기서 가용한 LWP가 무슨 뜻이야?
  - 여기서 가용한 LWP는 사용 가능한(lightweight process) LWP를 의미합니다. 다대일 모델에서는 하나의 프로세스에 여러 개의 사용자 수준 스레드가 존재하며, 이러한 스레드들은 하나의 LWP에 할당되어 실행됩니다. 반면에 다대다 모델에서는 여러 개의 프로세스가 있으며, 각 프로세스 내부에는 여러 개의 LWP가 존재할 수 있습니다. 여기서 사용 가능한 LWP란, 현재 시스템에 생성 가능한 LWP 중에서 이미 다른 스레드에 할당되어 실행 중이지 않은 LWP를 의미합니다. 이러한 LWP는 새로운 사용자 수준 스레드에게 할당될 수 있어, 프로세스의 병렬성과 처리 성능을 높일 수 있습니다.
- 프로세스 경쟁범위에서 다대일과 다대다 모델이 뭐야?
  - 프로세스 경쟁 범위는 여러 개의 프로세스나 스레드가 동시에 실행되는 환경에서, 이들이 경쟁하는 범위를 의미합니다.
  - 다대일 모델은 하나의 프로세스에 여러 개의 스레드가 존재하며, 이러한 스레드들이 하나의 중앙처리장치(CPU)를 공유하며 동작합니다. 이 모델에서는 프로세스의 주소공간과 자원을 스레드들이 공유하기 때문에, 프로세스 내부에서 스레드 간의 데이터 공유가 용이하고, 스레드 간의 커뮤니케이션도 비교적 쉽게 구현할 수 있습니다. 하지만, 스레드의 수가 증가하면 스레드 간의 경쟁이 치열해져 병목현상이 발생할 가능성이 높아집니다.
  - 다대다 모델은 여러 개의 프로세스가 존재하며, 이들이 서로 다른 중앙처리장치(CPU)를 공유하면서 동시에 실행됩니다. 각 프로세스는 독립된 주소공간과 자원을 가지고 있어, 프로세스 간의 데이터 공유가 어렵고, 커뮤니케이션도 복잡하게 구현해야 합니다. 하지만, 다수의 CPU를 이용해 병렬 처리를 할 수 있기 때문에, 대규모 서버 시스템 등에서 많이 사용됩니다.

## 다중 처리기 스케줄링(Multiple Processor Scheduling)

여러개의 CPU가 사용 가능하면, 여러 스레드가 병렬로 실행될 수 있어 **부하 공유(Load Sharing)**가 가능해진다. 하지만 이에 따라 스케줄링 문제는 그에 상응해 더욱 복잡해진다.

- 비대칭 다중처리(asymmetric multiprocessing)
  - 마스터 서버

: 다중 처리기 시스템의 CPU 스케줄링에 관한 한 가지 해결방법은 **마스터 서버**라는 하나의 프로세서가 모든 스케줄링 결정과 I/O 처리 그리고 다른 시스템의 활동을 취급하게 하는 것이다.

**하나의 코어만 시스템 자료구조에 접근하여 자료 공유의 필요성을 배제하기 때문에 간단하다**

하지만 마스터 서버가 전체 시스템 성능을 저하할 수 있는 병목이 된다.

- **대칭 다중 처리(symmetric multiprocessing, SMP)**

**다중 처리기를 지원하기 위한 표준 접근 방식이다.** 거의 모든 최신 운영체제는 SMP를 사용한다.

각 프로세서는 스스로 스케줄링 할 수 있다. 각 프로세서의 스케줄러가 준비 큐를 검사하고 실행할 스레드를 선택하여 스케줄이 진행된다.

스케줄 대상이 되는 스레드를 관리하기 위해 2가지의 방식이 있다.

- 공통 준비큐에 모든 스레드가 있다. (race condition이 발생할 수 있다.)
- 각 프로세서는 자신만의 스레드 큐를 가질 수 있다. (일반적인 방식)

- ▼ 좀 깊게 알아보고 싶은 주제들

**다중 코어 프로세스**

**부하 균등화**

**처리기 선호도**

**이기종 다중처리**

## 퀴즈

- ▼ Multiprogramming / Timesharing은 각각 어떤 프로그램에 적합할까?

**Multiprogramming** : batch processing, background processing과 같이 사용자에게 보이지 않은 프로그램에 적합하다.

- timesharing보다 overhead가 적다
- 처리율(시간동안 처리되는 작업의 수)이 좋고 낭비되는 시간이 적다.
- 하지만 응답시간(프로그램 실행 명령 ~ 실제로 실행되는 시간)은 Timesharing에 비해 크다.

**Timesharing** : interactive processing, foreground processing과 같이 사용자에게 보여지는 프로그램에 적합하다

- Multiprogramming보다 overhead가 크다
- 응답시간이 적다.
- 낭비되는 시간이 많아진다. (시간간격보다 처리시간이 짧아도 시간간격만큼 해당 프로그램이 CPU 제어권을 가지고 있기 때문)

- ▼ CPU 스케줄러는 어디에 위치할까?

CPU 스케줄러는 운영체제의 커널 내부에 위치합니다. 커널은 운영체제의 핵심 부분으로, 시스템 자원 관리, 입출력 처리, 파일 시스템 등 다양한 기능을 담당합니다. CPU 스케줄러는 이 중에서도 CPU 자원 관리를 담당하며, 프로세스 실행 우선순위에 따라 CPU 자원을 할당하는 역할을 수행합니다.

#### ▼ 문맥교환의 종류는 무엇이 있을까?

- 자발적 문맥 교환 : 현재 사용 불가능한 자원을 요청했을 때 프로세스가 CPU 제어를 포기한 경우 발생
- 복귀 문맥 교환 : 인터럽트나 예외 상황이 발생하여 프로세스가 CPU를 반환한 후, 이전에 실행되던 프로세스로 돌아가기 위한 문맥 교환이며, 대개 운영체제의 커널 모드에서 실행된다.
- 비자발적 문맥 교환 : 타임 슬라이스가 만료되었거나 우선순위가 높은 프로세스에 의해 선점되는 경우와 같이 CPU를 빼앗겼을 때 발생

## 참조

### [Operating System - Chapter 5] CPU 스케줄링

이 포스팅은 공룡책으로 알려진 Operating System Concepts의 5장인 CPU Scheduling를 공부하면서 정리한 포스팅이다.

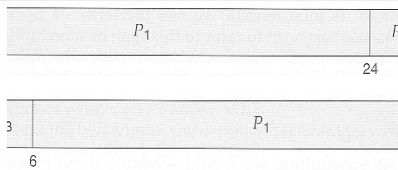
[https://imbf.github.io/computer-science\(cs\)/2020/10/18/CPU-Scheduling.html](https://imbf.github.io/computer-science(cs)/2020/10/18/CPU-Scheduling.html)



### [반효경 운영체제] CPU Scheduling 1

operating-system-study에서 스터디를 진행하고 있습니다. CPU burst와 I/O burst CPU burst CPU 버스트는 사용자 프로그램이 CPU를 직접 가지고 빠른 명령을 수행하는 단계이다. 이 단계에서 사용자 프로그램은 CPU 내에서 일어나는 명령 (ex. Add)이나 메모리(ex. Store, Load)에 접근하는 일반 명령을 사용할 수 있다.

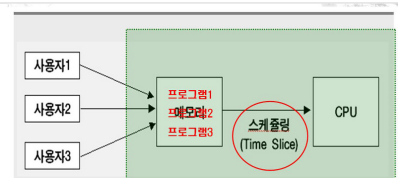
<https://steady-coding.tistory.com/530>



### 시분할 방식 및 시스템에 대하여

[시분할 시스템] 시분할 시스템(TSS, Time Sharing System)은 다중 프로그래밍을 논리적으로 확장한 개념으로 프로세서를 통해 다중 작업을 교대로 수행합니다. 컴퓨터를 대화식으로 사용하려는 시도에서 탄생하였습니다. 시분할 운영체제는 CPU 스케줄링과 다중 프로그래밍을 이용해서 각 사용자들에게 컴퓨터 자원을 시간적으로

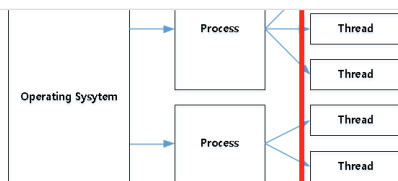
<https://akdl911215.tistory.com/219>



### [2020.09.16] (운영체제) 사용자 수준 스레드와 커널 수준 스레드의 차이

운영체제 공부 중에 사용자 수준 스레드와 커널 수준의 스레드에 대해 개념이 이해가 되지 않았다. 다른 블로그 글을 찾아봐도 뭔가 애매모호하게 설명하고 말아 버린 느낌으로 받아들여져서 이해하는데 시간이 꽤 걸렸다. 혹시나 나 같은 사람이 있을까봐 직접 최대한 쉽게 정리를 해보려고 한다. 스레드 우선 스레드부터 정리해 보자면,

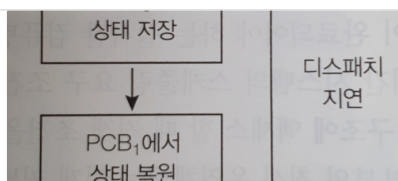
<https://helloinyong.tistory.com/293>



### 공룡책 Chapter5 CPU 스케줄링

CPU가 유휴 상태가 될 때 마다, 운영체제는 준비 큐에 있는 프로세스에서 하나를 선택해 실행해야 한다. 준비 큐에 있는 레코드들은 일반적으로 프로세스들의 프로세스제어블록(PCB)이다. CPU 스케줄링은 다음의 네 가지 상황에서 발생할 수 있다. running → wait


<https://velog.io/@koseyeon/공룡책-Chapter5-CPU-스케줄링>





#### [운영체제] CPU 스케줄링: 스레드 스케줄링

현대의 운영체제에서는 프로세스가 아닌 커널 수준의 스레드가 스케줄링된다. 사용자 수준의 스레드는 스레드 라이브러리에 의해 관리되고 커널은 사용자 수준 스레드를 알 지 못한다. 사용자 수준 스레드는 CPU상에서 실행되기 위해서 커널 수준 스레드에 매핑되어야 한다. contention scope(경쟁 범위) 사용자 수준 스레드와 커널

 <https://coding-gongbu.tistory.com/20>

