

# OS Basic Inner Structure



IMHOJEONG

2022.1.26 ImHoJeong

# Contents

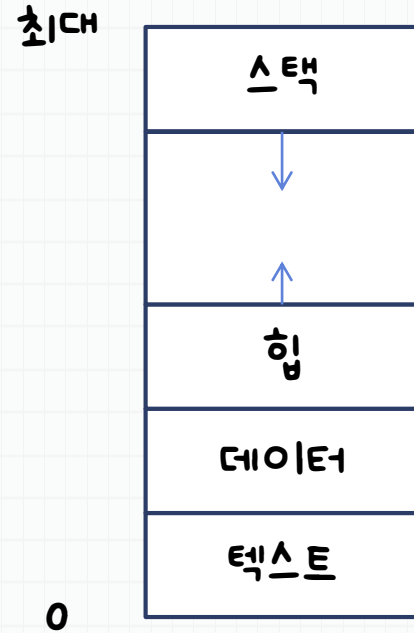
1. Process

2. Thread

3. Scheduling

**Process**

# Process



스택 섹션 - 함수 호출 시 임시 데이터 저장소  
(함수 매개변수, 복귀 주소 및 지역 변수)

힙 섹션 - 프로그램 실행 중 동적으로 할당되는 메모리

데이터 - 전역 변수

텍스트 - 실행 코드

# Process

그러면 직접 코드를 작성해서 알아보자!

상위  
메모리



하위  
메모리

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int x;
5 int y = 15;
6
7 int main(int argc, char *argv[]){
8
9     int *values;
10    int i;
11
12    values = (int *)malloc(sizeof(int)*5);
13
14    for(i = 0 ; i < 5 ; i++){
15        values[i] = i;
16    }
17
18    return 0;
19
20 }
```

## 참고

# Process

실행파일 이름 + size 명령어

상위  
메모리



하위  
메모리

```
D:\wtset>size test.exe
```

text	data	bss	dec	hex	filename
14408	1536	116	16060	3ebc	test.exe

data필드 : 초기화된 데이터

bss필드 : 초기화되지 않은 데이터

dec, hex : text + data + bss 영역의 합  
(각각 10진수, 16진수)

filename : 실행 파일 이름

```
C:\Users\임호정>gcc --version
gcc (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
C:\Users\임호정>d:
```

```
D:\>cd tset
```

```
D:\wtset>gcc test.c -o test
```

```
D:\wtset>size test
size: 'test': No such file
```

```
D:\wtset>size test.exe
  text  data  bss   dec   hex filename
14408  1536   116  16060  3ebc test.exe
```

```
D:\wtset>ls
'ls'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.
```

```
D:\wtset>dir
D 드라이브의 볼륨: T7
볼륨 일련 번호: A844-104E
```

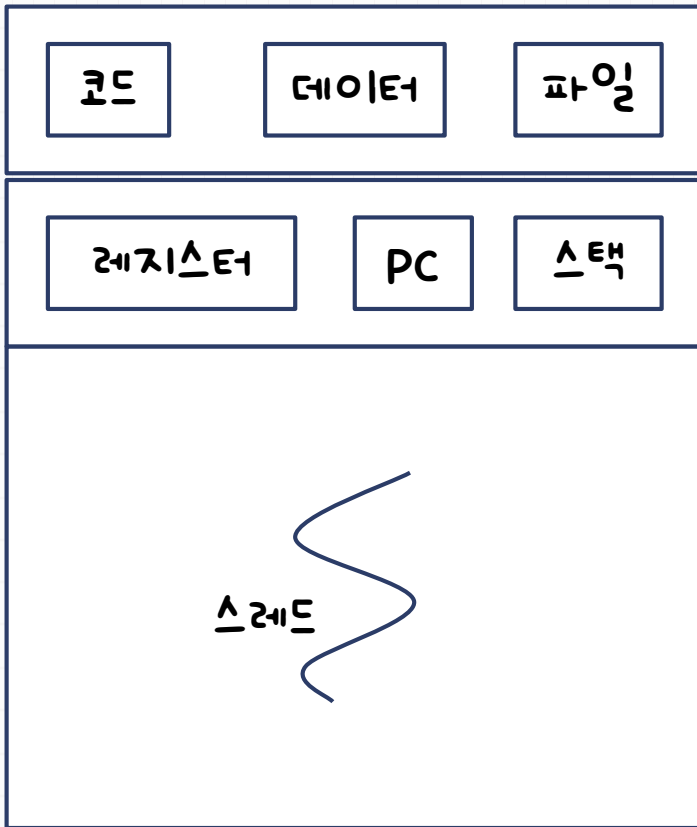
```
D:\wtset 디렉터리
```

2022-01-26	오후 02:32	<DIR>	.
2022-01-26	오후 02:32	<DIR>	..
2022-01-26	오후 02:33		238 test.c
2022-01-26	오후 02:44		40,680 test.exe
		2개 파일	40,918 바이트
		2개 디렉터리	491,742,035,968 바이트 남음

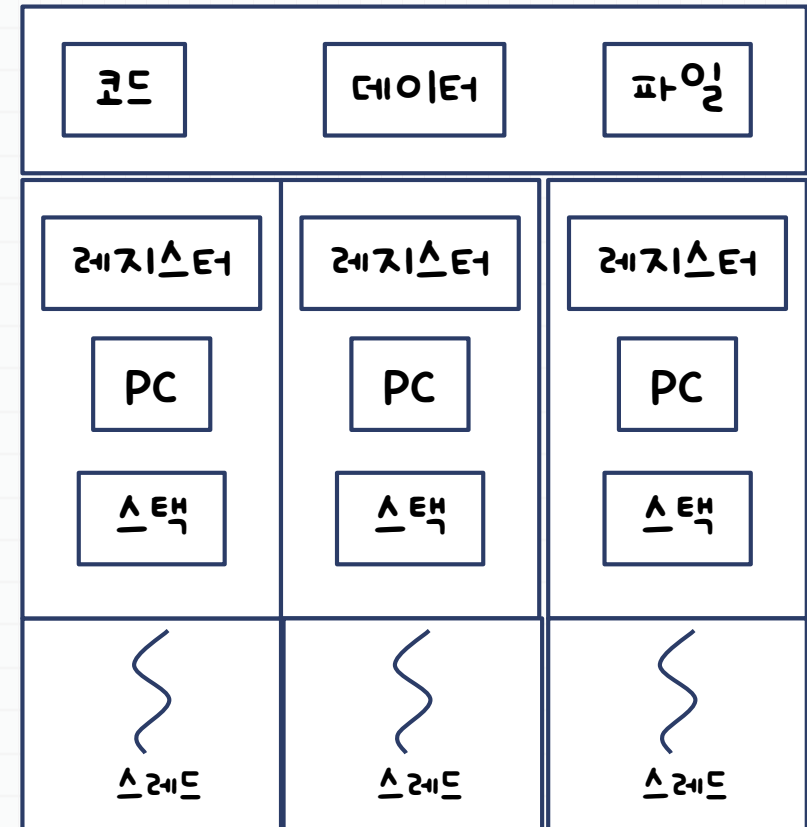
```
D:\wtset>
```

Thread

# Thread



단일 스레드 프로세스

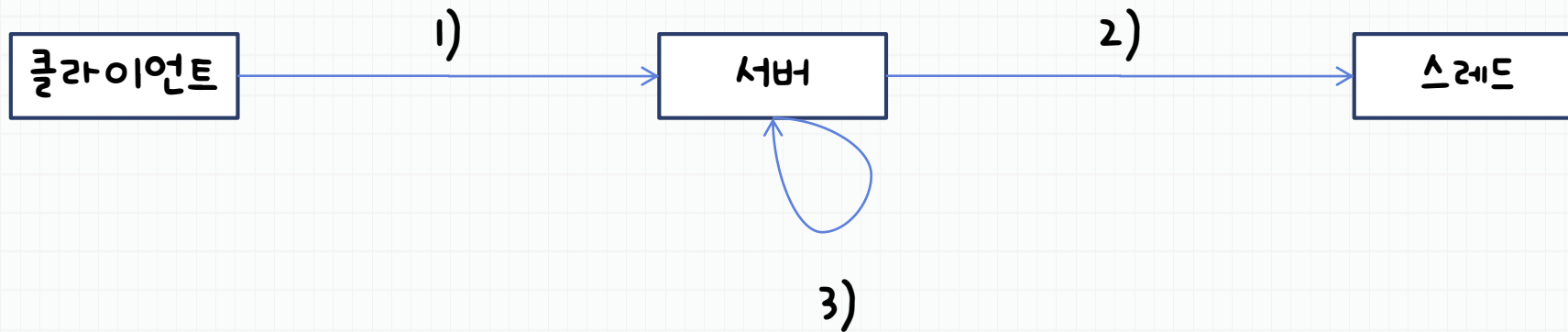


다중 스레드 프로세스



# Thread

다중 스레드 서버 구조



1. Server는 클라이언트의 요청을 listen하는 별도의 스레드를 생성

! - 요청이 들어오면 다른 프로세스 생성 X

2. 요청을 서비스할 새로운 스레드를 생성

3. 추가적인 요청을 listen하기 위한 작업을 재개

# Thread

운영체제 커널

Linux 시스템에서 시스템을 부트하는 동안 여러 커널 스레드가 생성

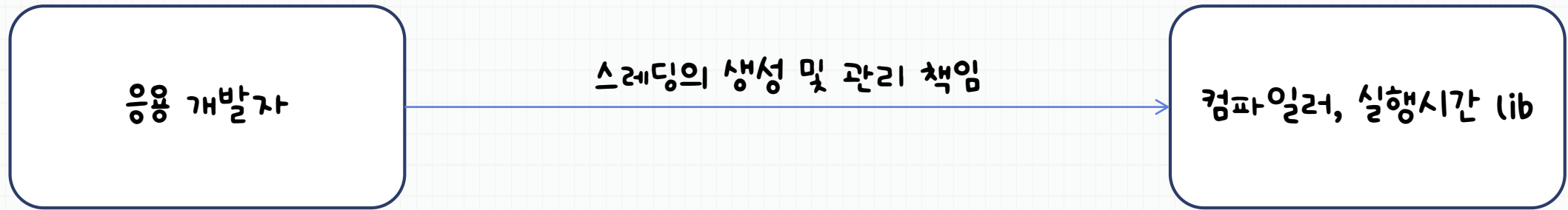
장치 관리, 메모리 관리, 인터럽트 처리 (특정 작업 수행)

왜 다중 스레드인가?

1. 응답성 - 시간이 오래 걸리는 연산이 별도의 비동기적 스레드 => 여전히 응답 가능
2. 자원공유 - 프로세스는 공유 메모리, 메시지 전달 기법(programmer에 의해서만!), 스레드는 자동으로 속한 프로세스의 자원, 메모리를 공유!
3. 경제성 - 문맥 교환 : 스레드가 더 빠름 & 프로세스 생성을 위한 메모리, 자원 할당은 비용이 많이 듦
4. 규모 적응성 - 각각의 스레드가 다른 처리기에서 병렬로 수행될 수 있음

# Thread

암묵적 스레딩



응용 프로그램 개발자가 병렬로 실행할 수 있는 스레드가 아닌 task를 식별해야 함

- 장점 -

개발자는 병렬 작업만 식별하면 되고,

라이브러리는 스레드 생성 및 관리에 대한 특정 세부 사항을 결정

# Thread

암묵적 스레딩

웹 브라우저 : 웹 서버는 요청을 받을 때마다 그 요청을 위해 새로운 스레드 생성

다중 스레드 서버

1. 서비스할 때마다 스레드를 생성하는 데 소요되는 시간  
(오해? 곧장 용도 폐기되기 때문)

2. 모든 요청마다 새 스레드를 만들어 서비스해 준다?

=> 시스템에서 동시에 실행할 수 있는 최대 스레드 수는 몇개?

스레드를 무한정 만들면 언젠가는 CPU 시간, 메모리 공간 같은 시스템 자원이 고갈

# Thread

## 스레드 풀

**IDEA** : 프로세스를 시작할 때 아예 일정한 수의 스레드들을 미리 만들어두는 것

① 평소에는 하는 일 없이 일감 대기

② 서버는 스레드를 생성 X → 요청을 받으면 thread pool에 제출 + 추가 요청 대기

- 풀에 사용 가능한 스레드가 있으면 깨어나고 요청이 즉시 서비스됨

- 풀에 사용 가능한 스레드가 없으면 사용 가능한 스레드가 생길 때까지 작업이 대기

- 스레드가 서비스를 완료 → pool로 돌아가 더 많은 작업을 기다림

# Thread

## 스레드 풀 장점

1. 기존 스레드로 서비스 > 새 스레드 만들어주기 (속도 측면)

2. 임의 시각에 존재할 스레드 개수에 제한을 둠

- 많은 수의 스레드를 병렬 처리할 수 없는 시스템에 도움이 됨

3. Task를 생성하는 방법 --- Task와 분리 => Task를 실행을 다르게 할 수 있음

- Thread pool 내 Thread 개수 -

- CPU 수, 물리 메모리 용량, 동시 요청 클라이언트 최대 개수 등 고려

ex) 시스템 부하에 따라 작은 풀을 유지하도록 함 => 메모리 등 소모를 줄일 수 있음

# CPU Scheduling

# CPU Scheduling

Purpose : CPU 이용률을 최대화하기 위함

WHY? 1개의 Process - I/O 요청 완료..... → CPU 놀지 (대기시간 낭비)

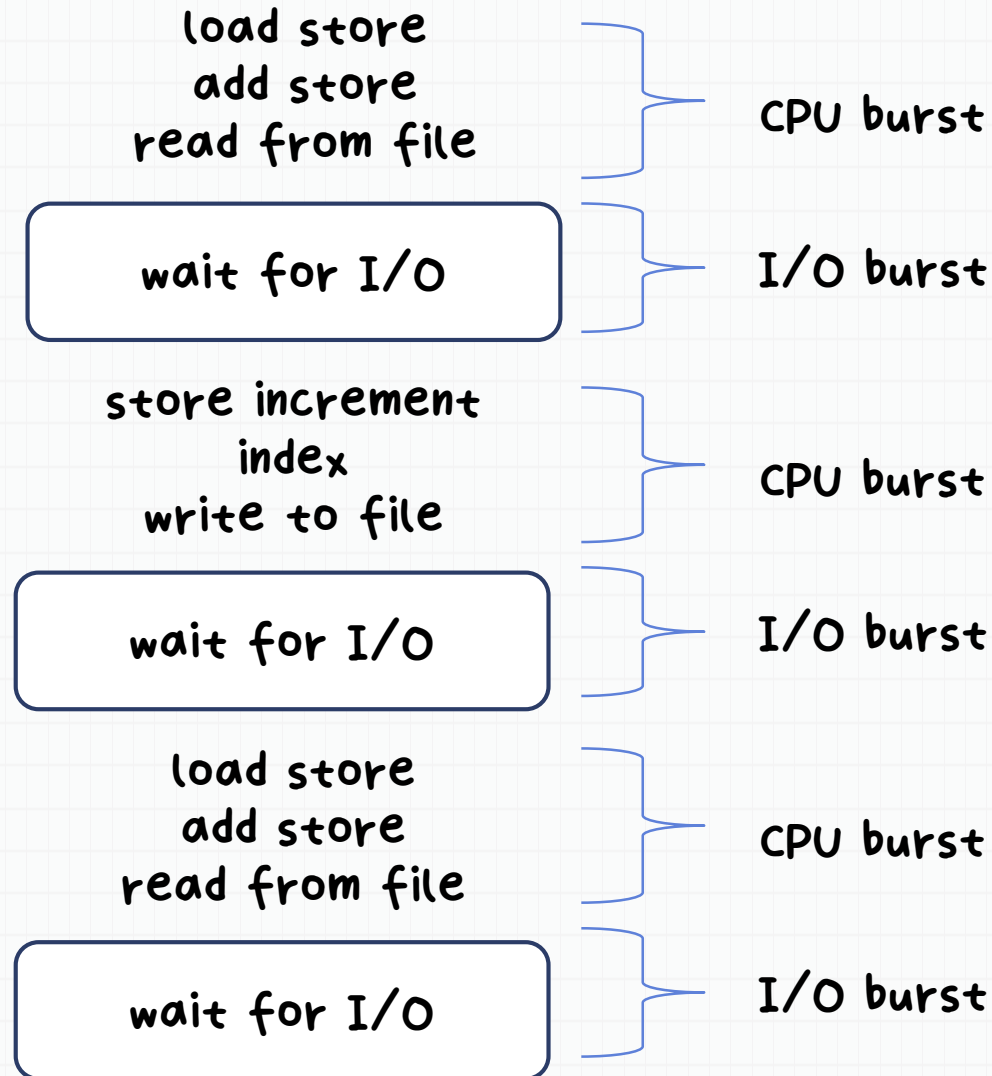
한 순간에 다수의 프로세스를 메모리 내에 유지

- 프로세스가 대기 → OS는 CPU를 회수해 다른 프로세스에 할당

프로세스 실행 = CPU 실행 + I/O 대기의 사이클



# CPU Scheduling



프로세스 실행 = CPU 실행 + I/O 대기의 사이클

CPU Burst로 시작 후, 뒤이어 I/O 버스트가 발생

..... 하다가

마지막 CPU 버스트는 또 다른 I/O 버스트가 뒤따르는 대신

실행을 종료하기 위한 시스템 요청과 함께 끝남

I/O 중심의 프로그램 - 전형적으로 짧은 CPU 버스트를 많이 가짐

CPU 지향 프로그램 - 다수의 긴 CPU 버스트를 가질 수 있음

# CPU Scheduling

## CPU Scheduler

CPU가 유휴 상태가 될 때마다, OS는 준비 큐에 있는 프로세스 중 하나 선택해 실행

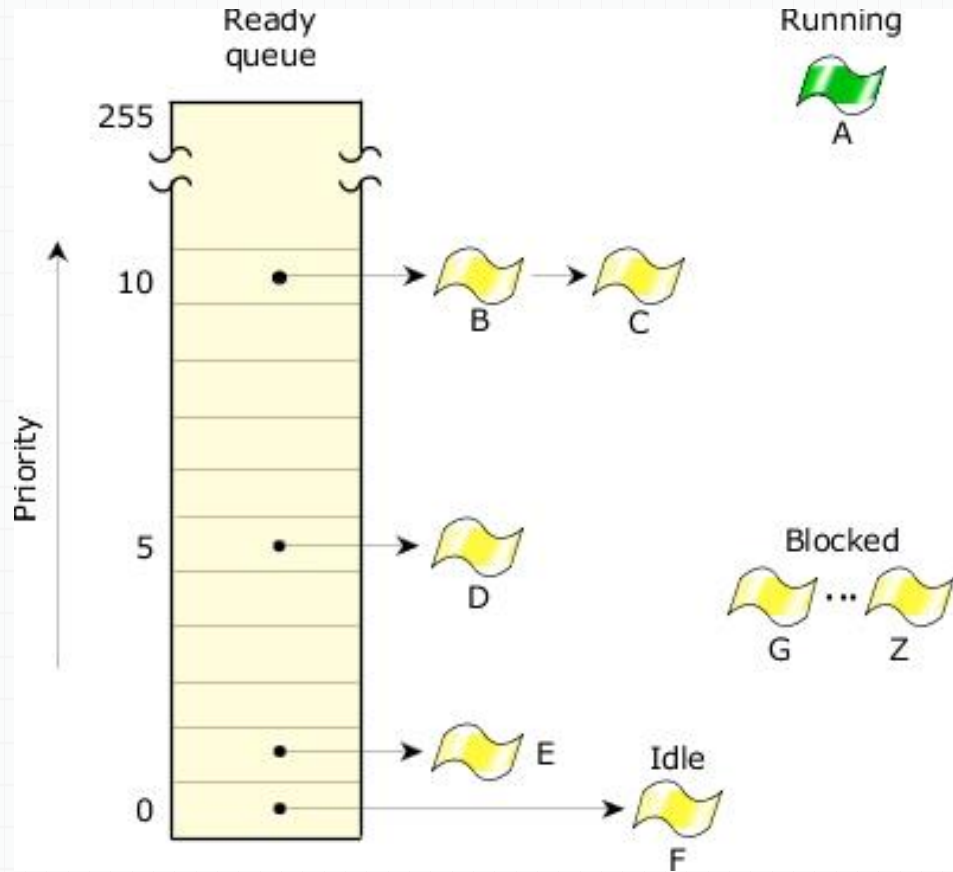
선택 절차는 CPU 스케줄러에 의해 수행됨

스케줄러 - 실행 준비가 되어 있는 메모리 내 프로세스 중에서 선택해, 이들 중 하나에게 CPU를 할당

참고

# CPU Scheduling

## Ready Queue



준비 큐 : 반드시 FIFO 방식의 큐가 아니어도 됨

FIFO, Priority Queue, LinkedList로 구현 가능

Ready Queue에 있는 모든 Process

=> CPU에서 실행될 기회를 기다리며 대기

Queue에 있는 record

=> 일반적으로 프로세스들의 프로세스 제어 블록(PCB)

PCB

PCB

PCB

PCB

PCB

Thank you

39ghwjd@naver.com