Thread 별도의 실행 흐름

*Stateless 3 : 멀티스레드

Calclient CalServer

연결 >

Thread < 생성

요청 >

< 응답

Calclient CalServer

*connection - oriented vs connectionless

TCP	Connection - Oriented	Statefull Stateless	ServerSocket, Socket
UDP	Connectionless		DatagramSocket, DatagramPacket

추가적인 루틴(절차) 필요 -> 코드에 삽입 해야함.

*URI → URL, URN

URL (Uniform Resource Locator)

	http:	//hong :	IIII	@www.naver.cor	80	cafe/~~~~	no=28 & ~~
URI	프로토콜	사용자 ID	암호	서버주소	포트번호	자원의 경로	
Uniform	scheme	user	password	host	port	path	query string
Resource		생략					
Identifier					http : 80		

URN (Uniforem Resource Name)

*정적자원과 동적자원

Web Browser Web server

HTTP Client 요청 → HTTP / HTTPS ← 응답 HTTP Server 요청 → ← 작업 결과를 HTML / XML / JSON 형태로 리턴 JAVA App - 작업수행

Dynamic Resources

>실행할 때마다 리턴결과가 달라질 수 있다.

↓↑

그대로 읽어서 클라이언트에 보낸다.

HTML
GIF
JS

Static Resources

*멀티 태스킹 - 여러 작업을 동시 수행(순차적 x)

*프로세스(process) : 실행중인 프로그램

1. 멀티 프로세싱

부모 프로세스↘ a.exe → (복제 = fork) (복제 = fork) ↙자식 프로세스

code	code	code
stack	stack	stack
Heap	Heap	Heap
↑ 프로세스	↑ 프로세스	↑ 프로세스

- 프로세스를 복제(fork)해서 멀티태스킹 수행
- code가 중복로딩
- Heap 중복 생성

메모리 많이 사용

2. 멀티 스레딩

Exam0110.exe
———
fork()
↓실행
———
———

→ 복제

Exam0110.exe
———
———
↓실행
———
———

↑

프로세스 종료 = 다른 프로세스로 계속 실행

```
#include <stdio.h>

int main() {
int i = 0;
for (i = 0; i < 100; i++) {
printf("==> %d\n", i);
}

int j = 0;
for (j = 0; j < 100; j++) {
printf("==> %d\n", j);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
int i = 0;
pid_t processId = 0;          //인트값을 받는 변수

processId = fork(); // 현재 실행 싱행 위치에서 프로세스 복제

for (i = 0; i < 10000; i++) {          0~9999까지 반복문
printf("[%d] ==> %d\n", processId, i);
```

```
return 0;
}
```

```
int temp = rand() * rand();
}

return 0;
}
```

*VM(Virtual Machine)
>가상머신은 실제 컴퓨터처럼 H/W 자원을 사용 = 자원 낭비가 심하다.

App	App	App	App	App	
Linux	windows 8	Linux	macOS	Linux	Guest OS
Windows OS					Host OS

*Docker
>App을 실행하는데 필요한 H/W 자원만 사용
>물론 VM처럼 별도의 컴퓨터에서 실행하는 것처럼 동작

Docker		Docker	
	App		App
Windows OS			

*멀티 스레딩
계산기.exe
*Code 와 Heap 메모리를 여러 스레드로 공유 = 멀티 프로세싱 방식에 비해서 메모리 낭비가 적다.

code
Heap
공유 사용
공유 사용
↑사용

main thread	→ 생성	자식 thread	→ 생성	자식 thread
stack		stack		stack

stack은 thread마다 만들어진다.

★프로세스 종료시★
↓
"종속된 스레드 모두 자동 종료"

Exam0120.java 예제

프로세스(process)	JVM		Method Area	Heap
		공유사용		
	"main"Thread	call →	main(){---------->	
	↑		↓	
	스레드(thread)	→	MyThread	
			call →	
			run(){---------->	

*멀티 태스킹의 실체

CPU
↓ ↓ ↓
App App App
왔다갔다 하면서 명령어를 조금씩 실행

*프로세스, 스레드, 멀티태스킹

P
P
L T
L T
CPU
P

프로세스와 스레드를 동일하게 간주하여 동시 실행한다.

★CPU가 프로세스나 스레드의 명령을 실행할 때,
CPU 실행을 분배하는 전략?
(cpu를 어떤 식으로 분배하여 명령을 처리할 것인가?)
★"CPU Scheduling"★ >>>운영체제가 마치 동시에 실행되는 것처럼 보이게 쓰레드나 프로세스를 왔다갔다하면서 명령어를 실행하는 일정(계획)
1.Windows OS -> Round Robin 방식
>프로세스와 스레드의 동일 시간 분배

2. Unix -> Priority + Aging Priority > > 우선 순위가 높은 프로세스나 스레드를 더 많이 실행하는 것.
Aging >> 우선 순위가 낮아 실행이 일절때마다 우선순위를 높혀서 다음번엔 실행될 수 있게 하는것.

*Context Switching
└ CPU가 다른 프로세스나 스레드를 실행하기 위해
└ ★현재 프로세스나 스레드에 실행 상태를 저장하고 > 여기까지 실행했다.
└ ★실행할 프로세스나 스레드의 실행 상태를 로딩하는 것. > 어디서부터 실행하나?

CPU
L1 L2
명령어 data
실행해서 가져오기 실행명령 저장

RAM 명령어 로딩
App1 App2

App1	128줄
App2	1343줄

Context : 실행 상태 정보
를 저장하고 읽어내는 것(바꿔치기) 하는데 시간이 걸림.
*CPU Scheduling 시간을 너무 짧게 잡으면 명령을 실행하는 시간보다
Context Switching 에 더 많은 시간을 소비하는 문제가 발생한다.
>>> 너무 간격을 크게 잡으면 동시 실행 효과가 떨어진다.