

<pre>#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); } }</pre>	<pre>#include <stdio.h> #include <stdlib.h> #include <unistd.h> int main() { int i = 0; pid_t processid = 0; processid = fork(); // 현재 실행 상황 위치에 프로세스 복제 for (i = 0; i < 10000; i++) { printf("[%d] ==> %d\n", processid, i); } }</pre>	<p>//인트값을 받는 변수</p> <p>0~9999까지 반복문</p>
--	--	---

```
return 0;
}

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

return 0;
}
```

*VM(Virtual Machine)

>가상머신은 실제 컴퓨터처럼 H/W 자원을 사용 = 자원 낭비가 심했다.



Guest OS

Host OS

*Docker

>App를 실행하는데 필요한 H/W 자원만 사용

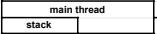
>물론 VM처럼 별도의 컴퓨터에서 실행하는 것처럼 동작



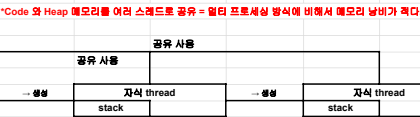
*멀티 스레딩
계산기.exe



사용



stack은 thread마다 만들어진다.



프로세스 종료시

"종속된 스레드 모두 자동 종료"

프로세스(process)

Exam0120.java 예제



"main"Thread

스레드(thread)

공유 사용

call →

↓

→

↓

call →

run() {

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

*멀티 태스킹의 실행

CPU

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

↓

App

왔다갔다 하면서 명령어를 조금씩 실행

*프로세스, 스레드, 멀티태스킹

P

P

L

L

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

CPU

P

P

L

L

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

P

*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 스레드나 프로세스를 왔다갔다하면서 명령어를 실행하는 일종(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 밀릴때마다 우선 순위를 높여서 다음번엔 실행될 수 있게 하는 것.

*Context Switching

↓ CPU가 다른 프로세스나 스레드를 실행하기 위해

↓ *현재 프로세스나 스레드에 실행 상태를 저장하고

↓ *실행할 프로세스나 스레드의 실행 상태를 로딩하는 것.

> 여기까지 실행했다.

> 어디서부터 실행하나?

CPU

L1

L2

영역

data

실행에서 가져오기

실행영역 저장

RAM

영역이 로딩

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

Context : 실행 상태 정보

를 저장하고 읽어내는 것(바꿔치기) 하는데 시간이 걸림.

*CPU Scheduling 시간을 너무 짧게 않으면 명령을 실행하는 시간보다

Context Switching 에 더 많은 시간을 소비하는 문제가 발생한다.

>>> 너무 간격을 크게 않으면 동시 실행 효과가 떨어진다.

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

App1

App2

App1

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

1343줄

128줄

