

# MAC 유저의 커널 탐방기

화생방 스터디  
이원영

# 목차

01. Window와 Linux의 명시적인 차이점

02. Window 커널의 프로세스

03. Linux 커널의 프로세스

04. 실제 예제

# 01. Window와 Linux의 명시적인 차이점 (1)

| WINDOWS KERNEL<br>VERSUS<br>LINUX KERNEL                                      |   |
|---|---|
| WINDOWS KERNEL  | LINUX KERNEL  |
| A commercial kernel of Windows operating system developed by Microsoft        | An open source Unix-like computer operating system kernel               |
| Developed by Microsoft  | Developed by Linux Torvalds   |
| There is no access to the source code   | There is full access to the source code                                 |
| Has hybrid architecture   | Has monolithic architecture   |
| Uses Access Control List (ACL) for file access control                        | Uses traditional Unix permissions and POSIX ACL for file access control |
| Includes a GUI stack in the kernel  | GUI stack is in the user-space  |
| Supports multiple users and sessions but depends on the versions and editions | Supports 100 % multi-user environment                                   |
| Maintains a registry to store configurations                                  | Maintains the configurations in files                                   |
| Has different mechanisms for different devices                                | Every device is a file  |
|   | Visit <a href="http://www.PEDIAA.com">www.PEDIAA.com</a>                |

## 1. 유형

1. 윈도우 커널: 마이크로소프트가 상용으로 개발
2. 리눅스 커널: 리누스 토르발스와 리눅스 커뮤니티가 오픈 소스로 개발

## 2. 소스 코드 접근성

1. 윈도우 커널: 소스 코드 접근에 제한이 있음
2. 리눅스 커널: 오픈 소스로 공개되어 누구나 소스 코드에 접근 가능

## 3. 아키텍처:

1. 윈도우 커널: 하이브리드 아키텍처를 채택하여 모놀리식과 마이크로 커널의 특징을 함께 가짐
2. 리눅스 커널: 전통적인 모놀리식 아키텍처를 가짐

운영체제 선에서 커널 모듈을 제공하여 확장성 제공

# 01. Window와 Linux의 명시적인 차이점 (2)

| WINDOWS KERNEL<br>VERSUS<br>LINUX KERNEL                                      |   |
|---|---|
| WINDOWS KERNEL  | LINUX KERNEL  |
| A commercial kernel of Windows operating system developed by Microsoft        | An open source Unix-like computer operating system kernel               |
| Developed by Microsoft  | Developed by Linux Torvalds   |
| There is no access to the source code   | There is full access to the source code                                 |
| Has hybrid architecture   | Has monolithic architecture   |
| Uses Access Control List (ACL) for file access control                        | Uses traditional Unix permissions and POSIX ACL for file access control |
| Includes a GUI stack in the kernel  | GUI stack is in the user-space  |
| Supports multiple users and sessions but depends on the versions and editions | Supports 100 % multi-user environment                                   |
| Maintains a registry to store configurations                                  | Maintains the configurations in files                                   |
| Has different mechanisms for different devices                                | Every device is a file  |
|   | Visit <a href="http://www.PEDIAA.com">www.PEDIAA.com</a>                |

## 4. 액세스 제어:

1. 윈도우 커널: 파일 액세스 제어를 위해 ACL(Access Control List)을 사용
2. 리눅스 커널: 전통적인 유닉스 권한과 POSIX ACL을 사용

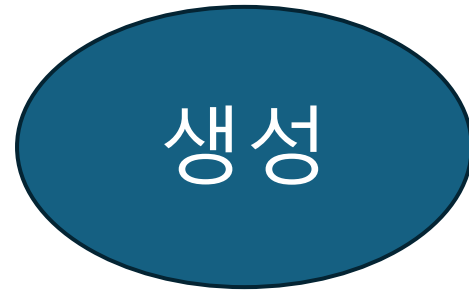
## 5. 그래픽 사용자 인터페이스(GUI) 스택:

1. 윈도우 커널: GUI 스택을 커널 내에 포함
2. 리눅스 커널: GUI 스택을 사용자 공간에 위치

## 6. 설정 관리

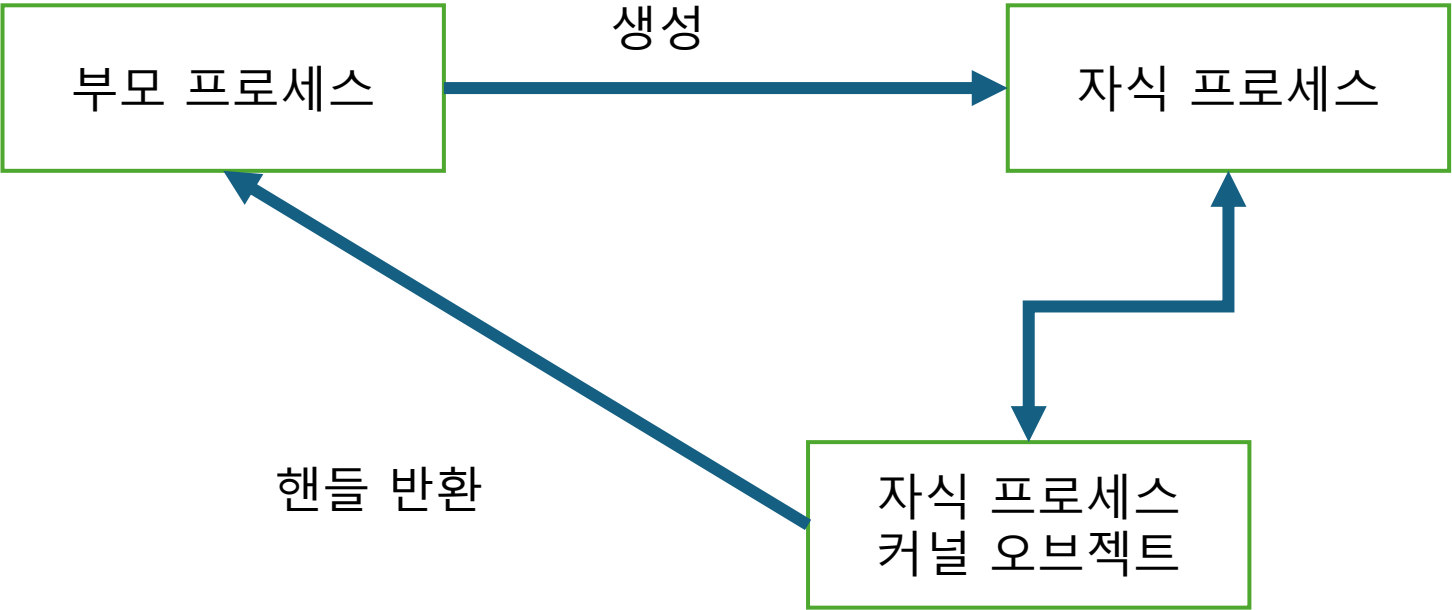
1. 윈도우 커널: 설정을 레지스트리에 저장하고 관리
2. 리눅스 커널: 설정을 파일에 저장하고 관리

## 02. Window 커널의 프로세스



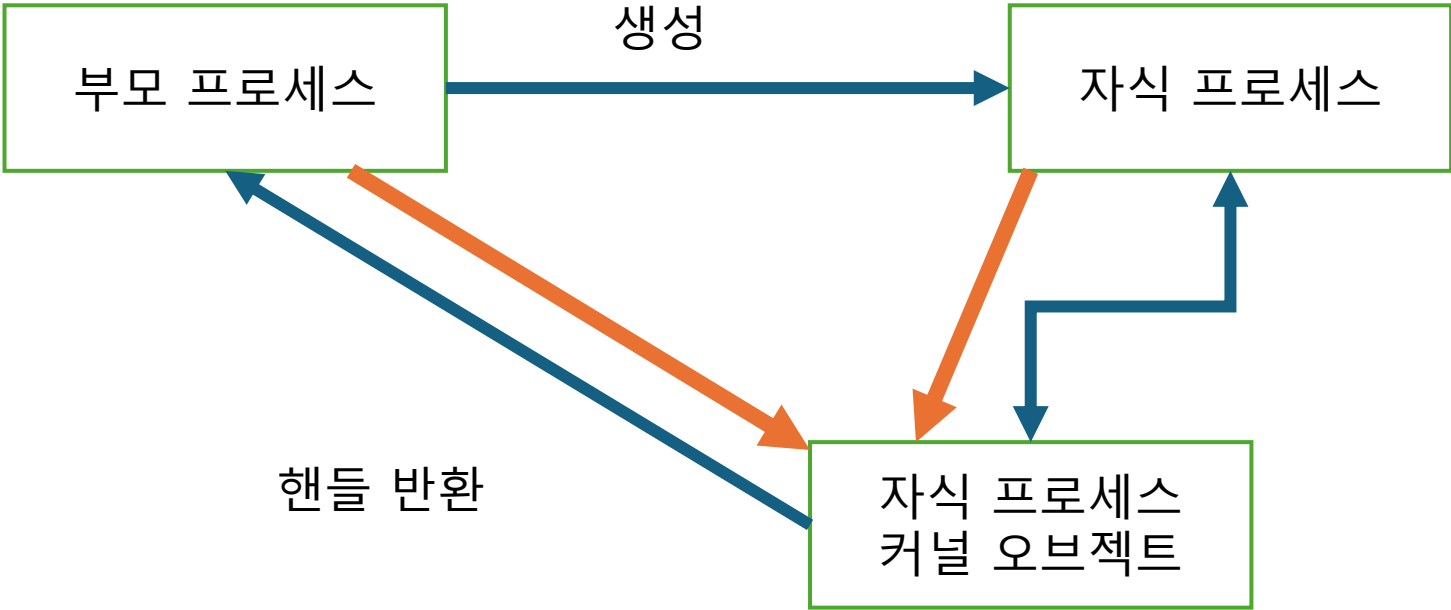
# 02. Window 커널의 프로세스

생성



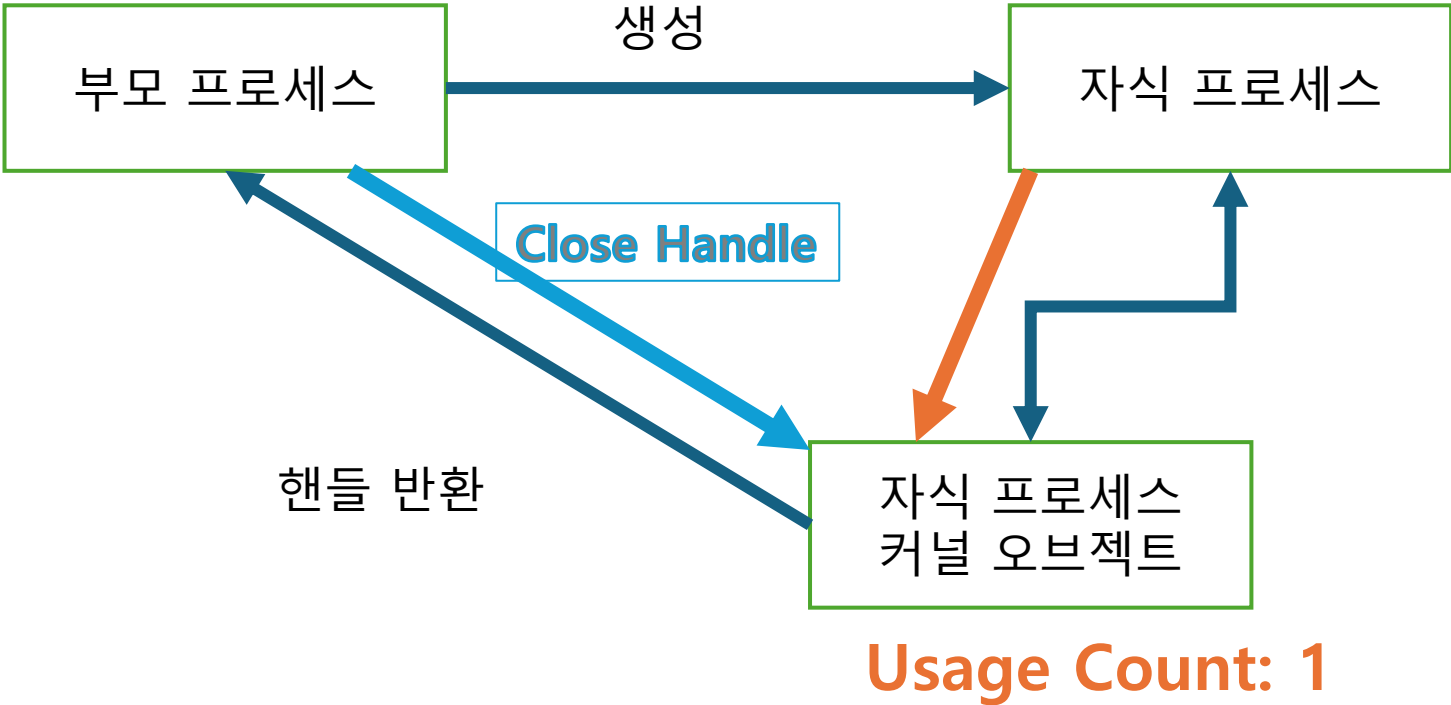
# 02. Window 커널의 프로세스

생성



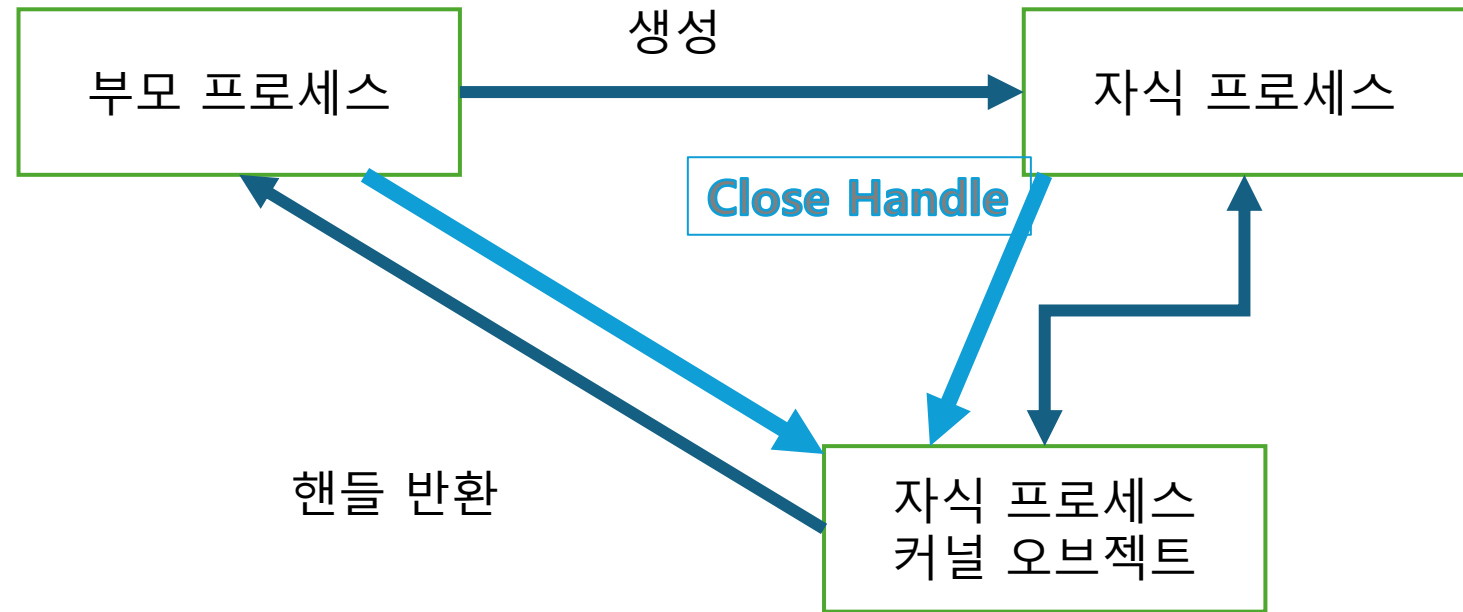
Usage Count: 2

# 02. Window 커널의 프로세스



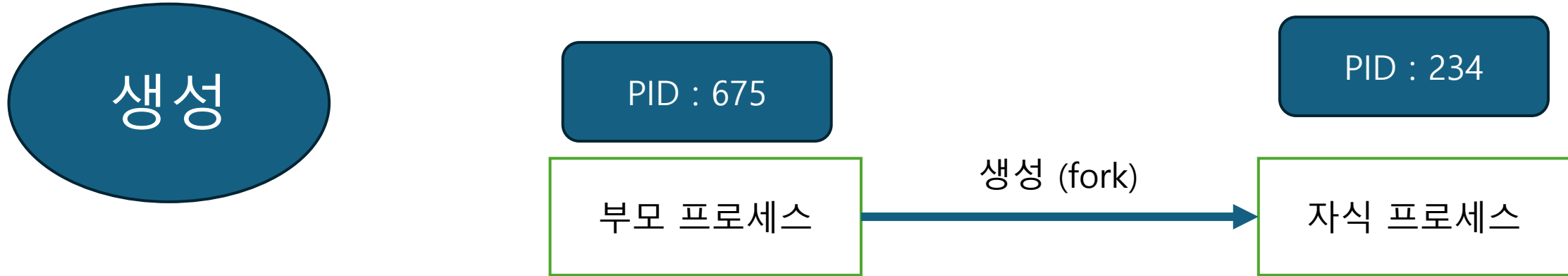


## 02. Window 커널의 프로세스

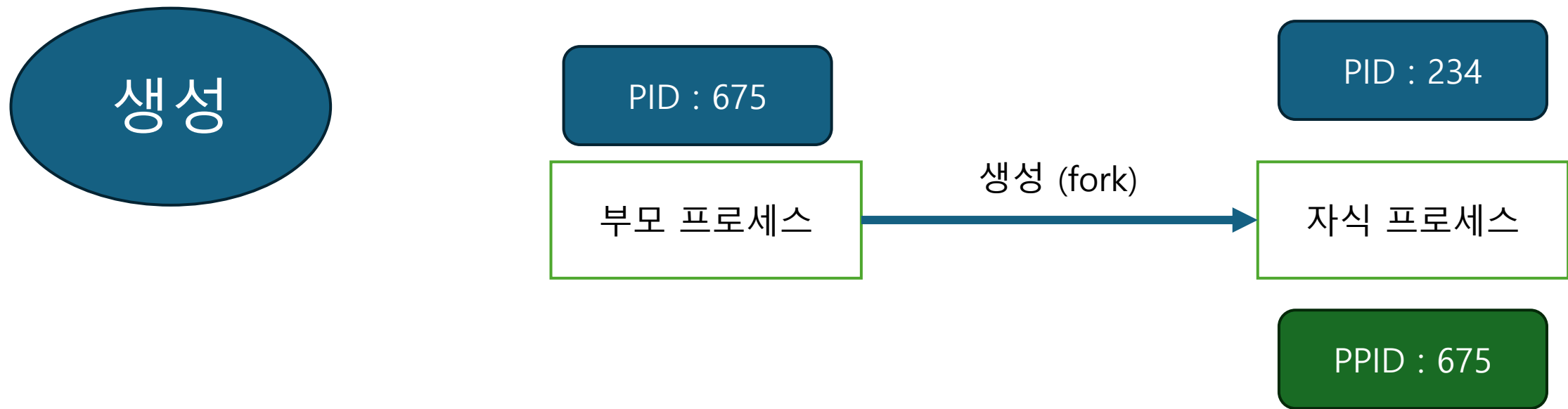


Usage Count: 0  
-> 소멸 !

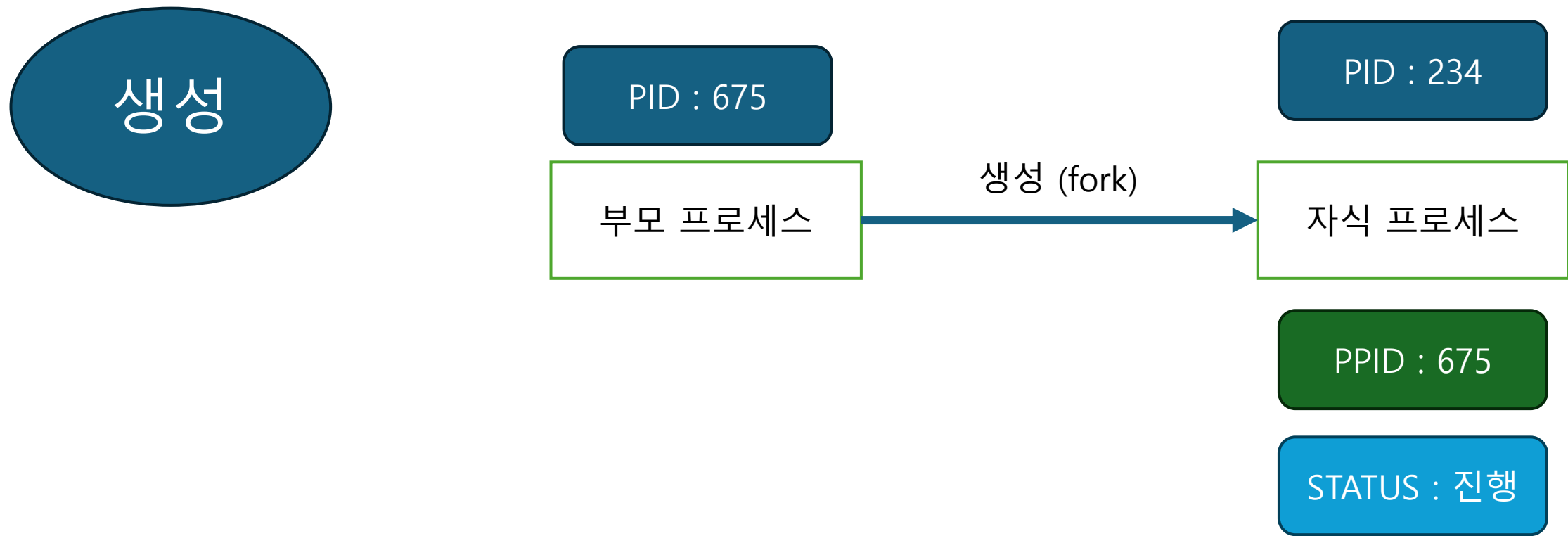
## 03. Linux 커널의 프로세스



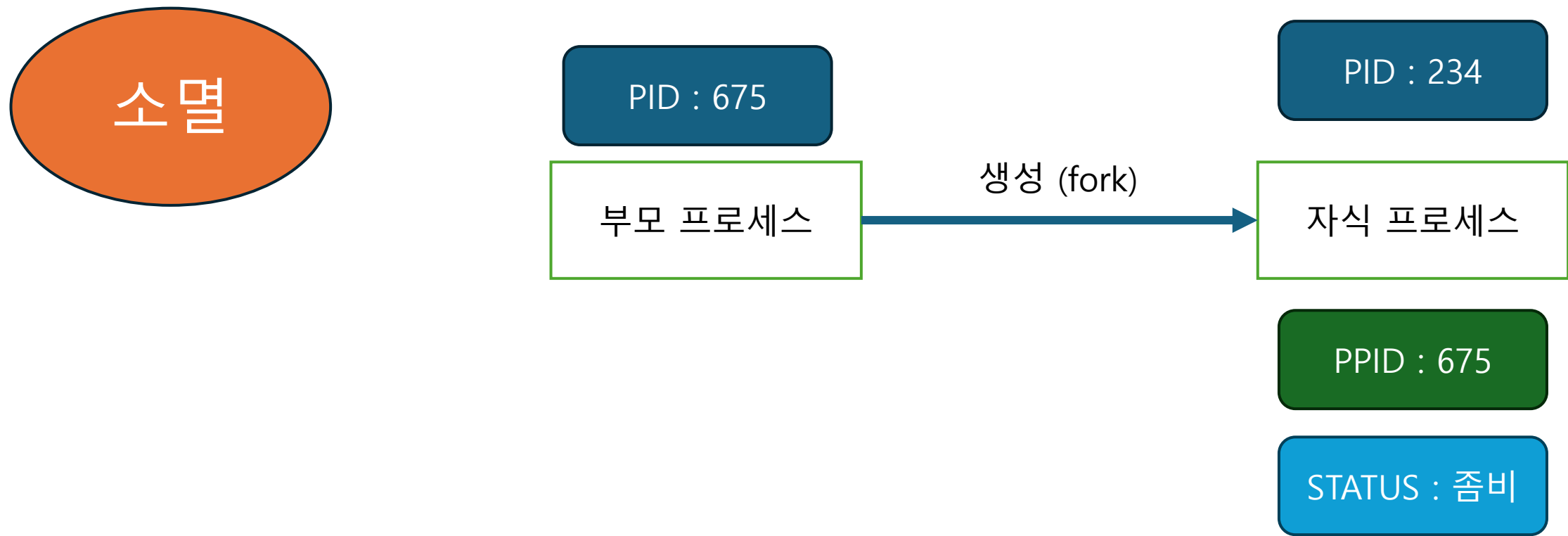
# 03. Linux 커널의 프로세스



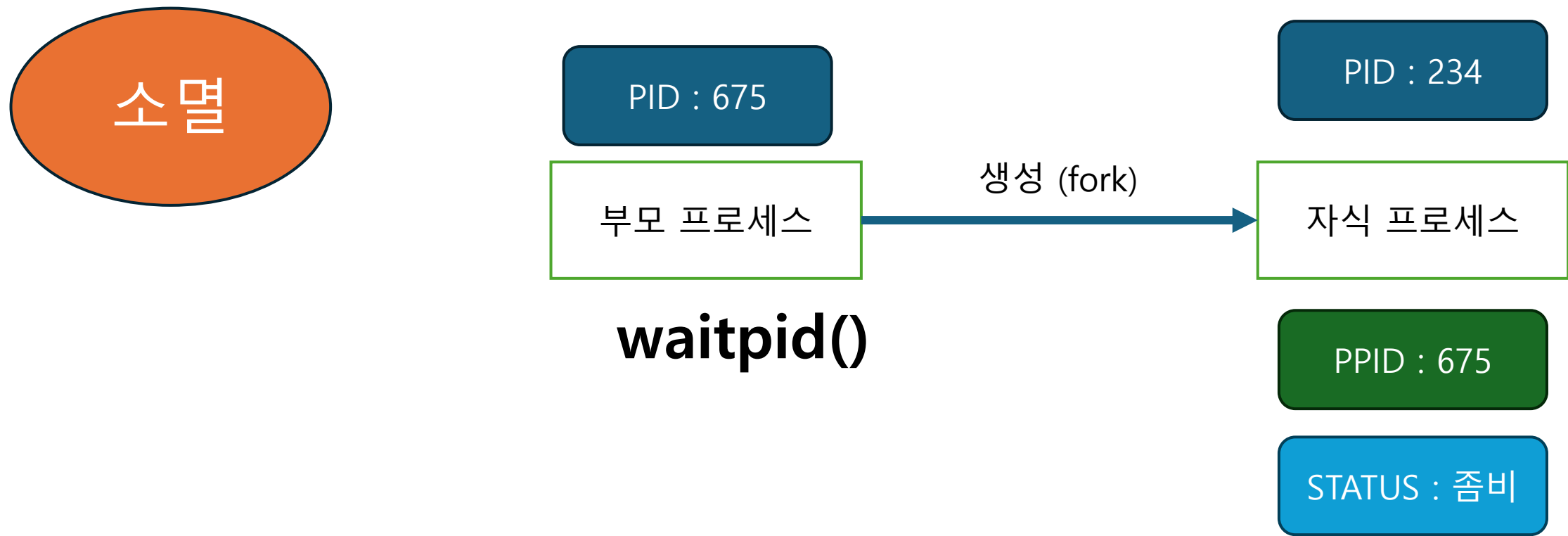
# 03. Linux 커널의 프로세스



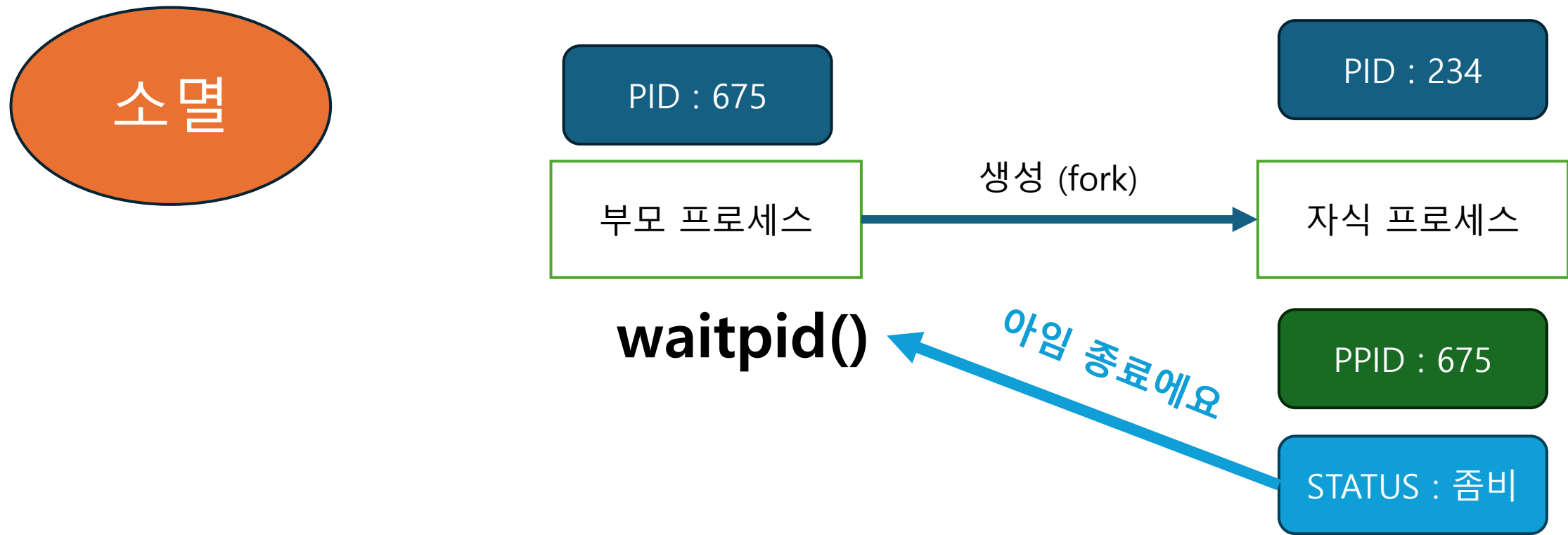
# 03. Linux 커널의 프로세스



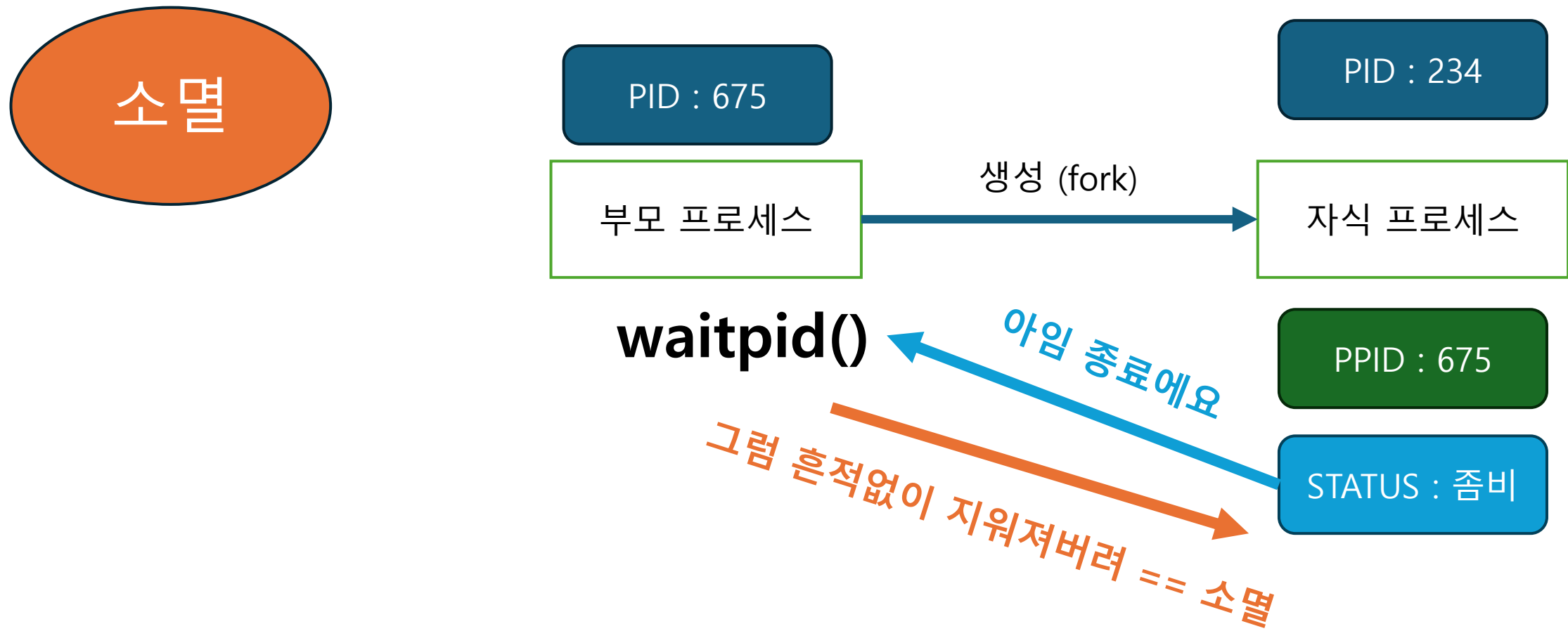
# 03. Linux 커널의 프로세스



# 03. Linux 커널의 프로세스

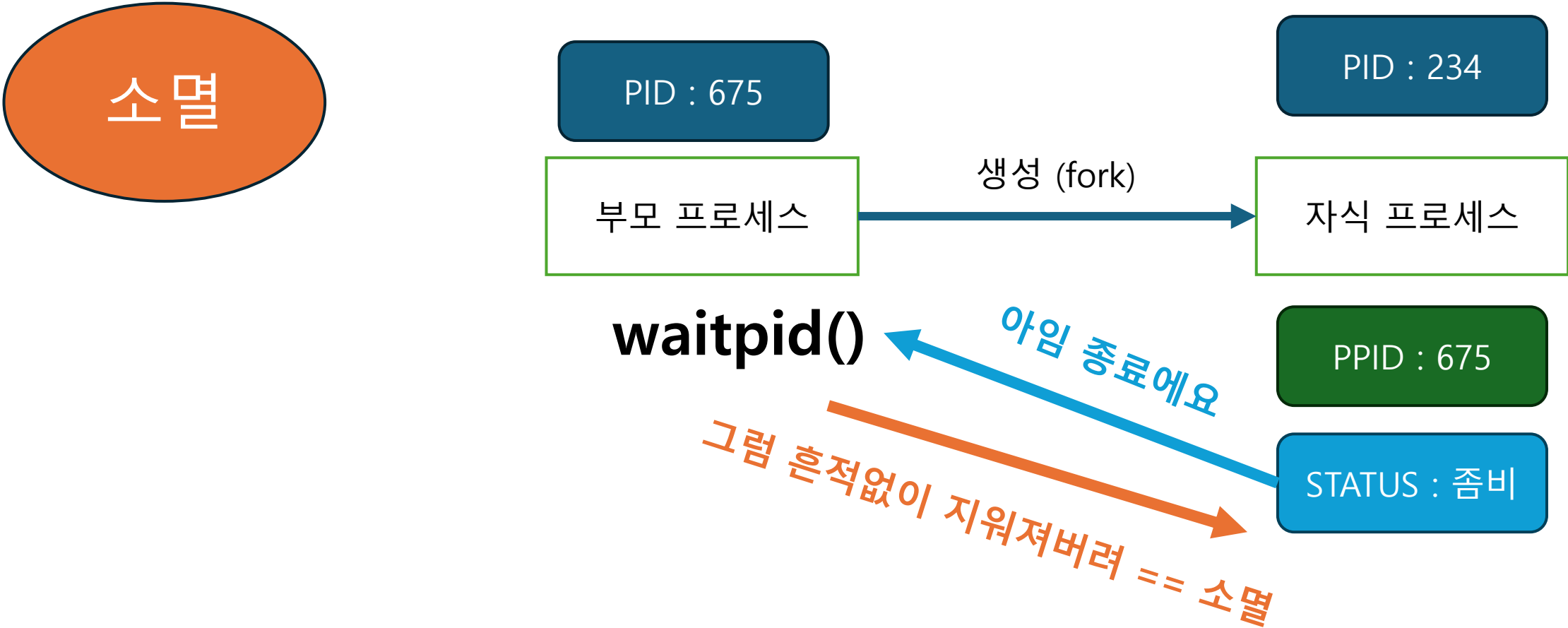


# 03. Linux 커널의 프로세스





# 03. Linux 커널의 프로세스



## 04. 실행 예시

```
int main() {  
    // 자식 프로세스 1 생성 전 상태 출력  
    printStatus("\n자식 프로세스 1 생성 전 상태:");  
  
    pid1 = fork();  
    if (pid1 < 0) {  
        perror("fork");  
        exit(1);  
    } else if (pid1 == 0) {  
        // 자식 프로세스 1 코드  
        execve("child1", NULL, NULL);  
        perror("execve");  
        exit(1);  
    }  
  
    // 자식 프로세스 2 생성 전 상태 출력  
    printStatus("\n자식 프로세스 2 생성 전 상태:");  
  
    pid2 = fork();  
    if (pid2 < 0) {  
        perror("fork");  
        exit(1);  
    } else if (pid2 == 0) {  
        // 자식 프로세스 2 코드  
        execve("child2", NULL, NULL);  
        perror("execve");  
        exit(1);  
    }  
}
```

- 1.fork(): 부모 프로세스로부터 자식 프로세스를 생성
- 2.execve(): 자식 프로세스에서 실행할 프로그램을 지정
- 3.waitpid(): 함수를 사용하여 자식 프로세스가 종료될 때까지 대기

## 04. 실행 예시

```
PID  PPID  STAT
16179 16178 S+
16265 16256 Ss
19402 16265 S+
19404 19402 Z+
19406 19402 Z+
```

### 대기 상태(S):

- Interruptible sleep 상태의 프로세스는 이벤트나 특정 조건을 개시.
- 커널은 이러한 프로세스를 CPU 스케줄링에서 일시적으로 제외시키고, 대기 중인 이벤트가 발생하면 다시 실행 대기 상태로 전환
- 이는 시스템 자원의 효율적 사용을 가능하게 함
- ex) I/O 작업 완료를 기다리는 경우가 여기에 해당합니다.

## 04. 실행 예시

| PID   | PPID  | STAT |
|-------|-------|------|
| 16179 | 16178 | S+   |
| 16265 | 16256 | Ss   |
| 19402 | 16265 | S+   |
| 19404 | 19402 | Z+   |
| 19406 | 19402 | Z+   |

### 포어그라운드 프로세스 그룹(+):

- 사용자와 직접 상호작용하는 프로세스들은 포어그라운드 프로세스 그룹에 귀속
- 커널은 이 프로세스들에게 사용자 입력과 같은 이벤트에 대한 우선권을 부여하며, 사용자의 요청에 따라 이들을 관리
- 이것은 사용자 경험과 시스템의 반응성을 향상시키는 데 중요한 역할

## 04. 실행 예시

```
PID  PPID  STAT
16179 16178 S+
16265 16256 Ss
19402 16265 S+
19404 19402 Z+
19406 19402 Z+
```

### 세션 리더(s):

- 세션 리더는 터미널 세션 또는 사용자 세션의 시작점이 되는 프로세스
- 커널은 세션 리더를 통해 세션 내의 프로세스 그룹을 관리하고, 세션과 관련된 신호나 이벤트를 이 프로세스에 전달
- 세션 리더의 존재는 프로세스 관리와 신호 처리를 간소화하는 데 도움

## 04. 실행 예시

```
PID  PPID  STAT
16179 16178 S+
16265 16256 Ss
19402 16265 S+
19404 19402 Z+
19406 19402 Z+
```

### 좀비 상태(Z):

- 프로세스가 종료되었지만, 부모 프로세스가 아직 그 종료 상태를 회수하지 않은 프로세스
- 커널은 이 프로세스의 메타데이터(예: 종료 상태)를 유지하며, 부모 프로세스가 이 정보를 회수하고 프로세스를 완전히 제거할 때까지 프로세스 엔트리를 유지

## 04. 실행 예시

```
자식 프로세스 1 생성 전 상태 :  
ps: cmd: keyword not found  
  PID  PPID STAT  
16179 16178 S+  
16265 16256 Ss  
19402 16265 S+
```

```
자식 프로세스 2 생성 전 상태 :  
<<첫째 탄생 기념>>  
ps: cmd: keyword not found  
  PID  PPID STAT  
16179 16178 S+  
16265 16256 Ss  
19402 16265 S+  
19404 19402 Z+
```

```
자식 프로세스들 종료 직전 상태 :  
[[옳다구야 둘째로다]]  
ps: cmd: keyword not found  
  PID  PPID STAT  
16179 16178 S+  
16265 16256 Ss  
19402 16265 S+  
19404 19402 Z+  
19406 19402 Z+
```

```
모든 자식 프로세스 종료 후 상태 :  
ps: cmd: keyword not found  
  PID  PPID STAT  
16179 16178 S+  
16265 16256 Ss  
19402 16265 S+
```

**Q & A**