```
// 터미널에서 입력
Ps f
Ps -l
Ps -l f
Ps -e -l | more
Ps -e -l f | more
Pstree
```

```
(base) 202020827@cslinux:~$ ps f
    PID TTY      STAT   TIME COMMAND
  58419 pts/22   Ss     0:00 /bin/bash -l
 516223 pts/22   R+     0:00  \_ ps f
  41571 pts/14   Ss+    0:00 /bin/bash -l
  38039 pts/29   Ss     0:00 /bin/bash -l
  38134 pts/29   S+    68:08  \_ htop
  37939 pts/12   Ss     0:00 /bin/bash -l
  38037 pts/12   S+    25:02  \_ top
(base) 202020827@cslinux:~$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1096   58419   37891  0  80   0 -  3177 do_wai pts/22   00:00:00 bash
0 R  1096  516350   58419  0  80   0 -  3496 -      pts/22   00:00:00 ps
```

1) ps 명령어 출력에서 PPID(부모 프로세스 ID)와 PID(프로세스 ID)를 확인하면, 부모-자식 관계를 파악할 수 있다.

2) 시스템의 최상위 조상 프로세스는 PID 1인 systemd이다.

Fork2.c

```c
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>

int main(void)
{
    int pid, status;
    pid = fork();
    if (pid > 0) {
        printf("PARENT: Child pid = %d₩n", pid);
        waitpid(pid, &status, 0);
        printf("PARENT: Child exited (parent is still running)₩n");
        do {} while (1);
    } else {
```

```
        printf("CHILD : Child process is running.₩n");
        do {} while (1);
    }
}
```

```
(base) 202020827@cslinux:~/os/fork$ ./fork2 &
[1] 517758
(base) 202020827@cslinux:~/os/fork$ PARENT: Child pid = 517759
CHILD : Child process is running.
ps -l f
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
0 S  1096   58419   37891  0  80   0 -  3177 do_wai pts/22     0:00 /bin/bash -l
0 S  1096  517758   58419  0  80   0 -   694 do_wai pts/22     0:00  \_ ./fork2
1 R  1096  517759  517758 98  80   0 -   694 -      pts/22     0:11  |   \_ ./fork2
0 R  1096  517873   58419  0  80   0 -  3496 -      pts/22     0:00  \_ ps -l f
0 S  1096   41571   37891  0  80   0 -  3332 do_sel pts/14     0:00 /bin/bash -l
0 S  1096   38039   37891  0  80   0 -  3177 do_wai pts/29     0:00 /bin/bash -l
0 R  1096   38134   38039  2  80   0 -  3702 -      pts/29    68:12  \_ htop
0 S  1096   37939   37891  0  80   0 -  3177 do_wai pts/12     0:00 /bin/bash -l
0 S  1096   38037   37939  0  80   0 -  3835 do_sel pts/12    25:03  \_ top
```

1) 부모를 먼저 종료해도 자식은 함께 종료되지 않는다. 자식 프로세스의 부모가 init(systemd) 프로세스로 변경된다.

2) 자식을 먼저 종료해도 부모 프로세스는 함께 종료되지 않는다. 자식이 매우 빠르게 종료되더라도, 부모가 waitpid()를 호출하면 자식의 종료 사실을 정상적으로 회수할 수 있음

3) 만약 부모가 waitpid()를 실행하기 전에 자식이 종료된다면, 자식은 좀비(zombie) 상태가 된다. 이후 부모가 waitpid()를 호출하면 커널이 자식의 종료 상태를 반환해주므로, 부모는 자식의 종료 사실을 인지하고 waitpid()를 성공적으로 마친다.

Fork1.c

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
int main(void)
{
    int pid, status;
    pid = fork();
    if (pid > 0) {
```

```
    /* parent process */
        printf("PARENT: child=%d₩n", pid);
        do { } while (1);
        } else { /* pid == 0 */
        /* child process */
        printf("CHILD: child process is running.₩n");
        do { } while (1);
    }
}
```

```
(base) 202020827@cslinux:~/os/fork$ ./fork1 &
[1] 520577
(base) 202020827@cslinux:~/os/fork$ PARENT: child=520578
CHILD: child process is running.
ps -l f
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
0 S  1096  518442   37891  0  80   0 -  3177 do_wai pts/83     0:00 /bin/bash -l
0 R  1096  520577  518442 99  80   0 -   694 -      pts/83     0:03  \_ ./fork1
1 R  1096  520578  520577 99  80   0 -   694 -      pts/83     0:03  |   \_ ./fork1
0 R  1096  520601  518442  0  80   0 -  3496 -      pts/83     0:00  \_ ps -l f
0 S  1096   41571   37891  0  80   0 -  3332 do_sel pts/14     0:00 /bin/bash -l
0 S  1096   38039   37891  0  80   0 -  3177 do_wai pts/29     0:00 /bin/bash -l
0 S  1096   38134   38039  2  80   0 -  3702 do_pol pts/29    68:18  \_ htop
0 S  1096   37939   37891  0  80   0 -  3177 do_wai pts/12     0:00 /bin/bash -l
0 S  1096   38037   37939  0  80   0 -  3835 do_sel pts/12    25:05  \_ top
```

1) 자식은 종료되었지만 좀비(Zombie) 상태이다. defunct 또는 Z 상태는 자식 프로세스가 이미 종료되었으나, 부모가 wait() 또는 waitpid()를 통해 자식의 종료 상태를 회수하지 않아 커널 프로세스 테이블에서 제거되지 않은 상태를 의미한다.

2) 좀비 상태에서는 kill 명령이 무의미하다. 프로세스가 이미 종료되었으므로 추가적인 시그널을 보내도 반응이 없고, 좀비 상태에서 벗어나려면 부모가 종료 상태를 수거해야 한다.

3) 부모를 kill 로 종료하면 자식(좀비)도 사라진다. 부모가 사라지면 좀비 프로세스의 부모가 init(systemd) 프로세스로 바뀌고, 최종적으로 init 이 자식 프로세스의 종료 상태를 회수하여 프로세스가 완전히 제거된다.