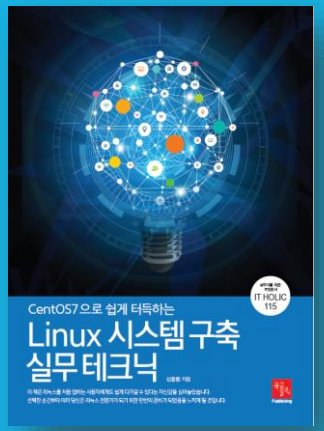
CHAPTER 10

프로세스 관리와 응급 복구



Section 01 | 프로세스와 서비스

Section 02 | 응급 복구

Section 03 | GRUB 부트로더

프로세스와 서비스

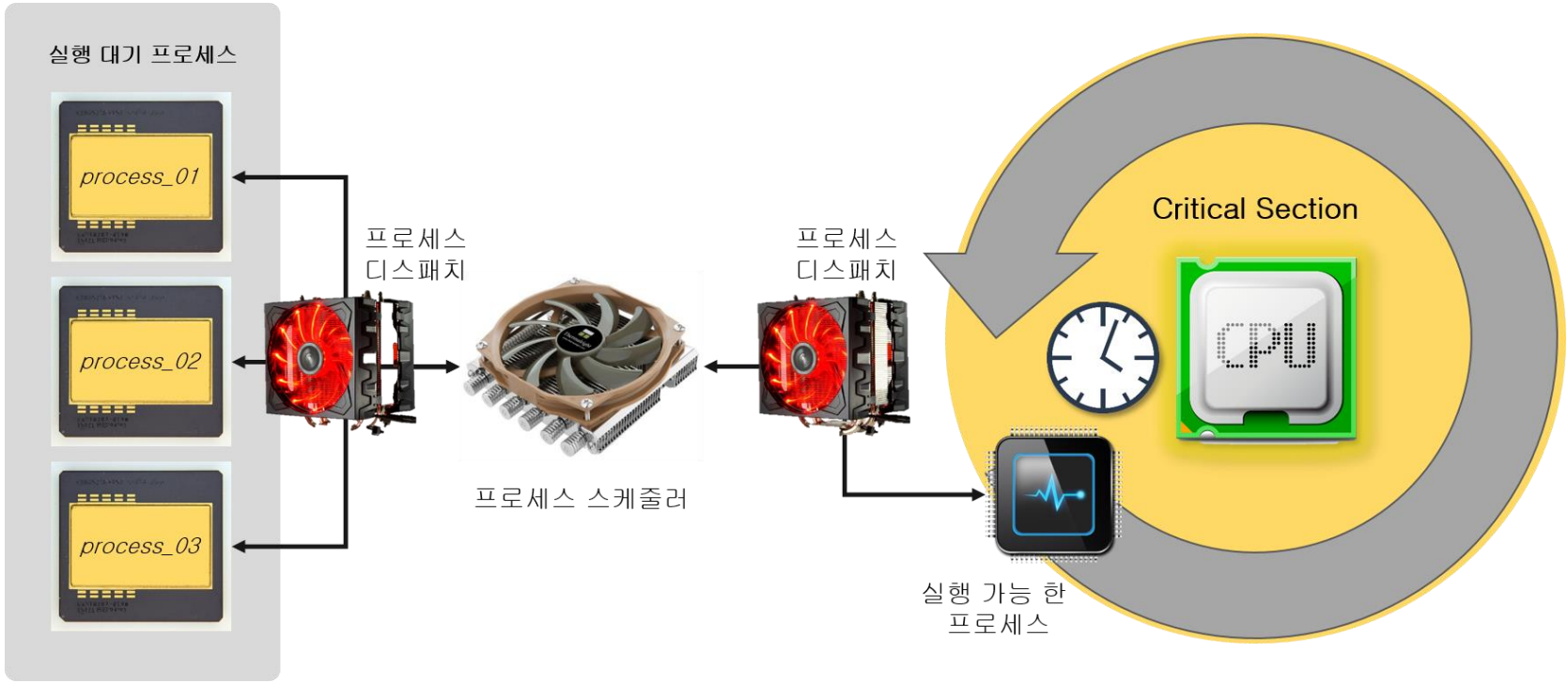
- 프로세스 상태 전이와 서비스 이해하기

1. 프로세스에 대한 개념을 이해합니다.
2. 프로세스의 상태 확인과 제어 명령에 대해 실습합니다.
3. 데몬과 서버 프로세스에 대해 살펴봅니다.

프로세스 개념

- 리눅스 시스템을 구동하기 위해 가장 먼저 활동하는 것은 바로 시작 프로세스이며
- 리눅스 계열 운영체제에서 구동되는 시작 프로세스는 Unix식 시동 프로세스와 여러모로 유사한 점이 많음

- 프로세스 스케줄러와 상태 전이



프로세스 상태

- 프로세스는 컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램을 의미
- 커널은 준비 큐, 대기 큐, 실행 큐 등의 자료구조를 이용하여 프로세스를 관리

● 프로세스 관리

- 생성create : 프로세스가 생성되는 상태
- 실행running : 프로세스가 CPU를 차지하여 명령어들이 실행되고 있는 상태
- 준비ready : 프로세스가 CPU를 사용하고 있지는 않지만 언제든지 사용할 수 있는 상태로 CPU가 할당되기를 기다리고 있으며 일반적으로 준비 상태의 프로세스 중 우선순위가 높은 프로세스가 CPU를 할당받는 상태
- 대기waiting : 보류block라고도 부르며 프로세스가 입출력 완료, 시그널 수신 등 어떤 사건을 기다리고 있는 상태
- 종료terminated : 프로세스의 실행이 종료되는 상태

프로세스 상태 전이

- 하나의 프로그램이 실행되면 그 프로그램에 대응되는 프로세스가 생성되어 준비 리스트의 끝에 들어감
- 준비 리스트 상의 다른 프로세스들이 CPU를 할당받아 준비 리스트를 벗어나면
- 그 프로세스는 점차 준비 리스트의 앞으로 나가게 되고 할당된 순서가 도래되면 CPU를 사용하게 됨
- 이와 같은 과정을 프로세스 상태 전이라고 함

디스패치dispatch

준비 리스트의 맨 앞에 있던 프로세스가 CPU를 점유하게 되는 것으로 준비 상태에서 실행 상태로 바뀌는 과정을 의미하며 다음과 같이 단계를 이동합니다.

```
dispatch (processname) : ready → running
```

보류block

실행 상태의 프로세스가 허가된 시간을 다 쓰기 전에 입출력 동작을 필요로 하는 경우 프로세스는 CPU를 스스로 반납하고 보류 상태로 넘어가는 과정을 의미하며 다음과 같이 단계를 이동합니다.

```
block (processname) : running → blocked
```

깨움wakeup

입출력 작업 종료 등 기다리던 사건이 일어났을 때 보류 상태에서 준비 상태로 넘어가는 과정을 의미하며 다음과 같이 단계를 이동합니다.

```
wakeup (processname) : blocked → ready
```

시간제한timeout

운영체제는 프로세스가 프로세서를 계속 독점해서 사용하지 못하게 하기 위해 clock interrupt를 두어서 프로세스가 일정 시간동안만(시분할 시스템의 time slice) 프로세서를 점유할 수 있게 하며 다음과 같이 단계를 이동합니다.

```
timeout(processname) : running → ready
```

포그라운드 프로세스

포그라운드 프로세스Foreground Process는 화면에서 프로그램이 실행되고 있는 것을 눈으로 직접 확인할 수 있는 상태의 프로세스 구동을 의미합니다. 즉 사용자와 상호작용으로 작업을 수행하도록 해주는 프로세스를 포그라운드 프로세스라고 합니다.

백그라운드 프로세스

백그라운드 프로세스Background Process는 프로세스가 실행되었지만 직접 눈으로 확인되지 않는 프로세스를 의미합니다. 예를 들어 백신 프로그램, 서버 데몬 등과 같이 화면에 나타나 눈에 보이지는 않지만 실행되는 프로세스를 백그라운드 프로세스라고 합니다.

프로세스 번호

CPU에서 처리하고자 하는 작업이 멀티태스킹과 같이 여러 개의 프로세스가 메모리에 로딩되어 있을 경우 프로세스를 구분하려면 각각의 프로세스에 대한 고유 식별방법이 존재해야 합니다. CPU가 프로세스를 구분하기 위해 부여되는 고유번호를 바로 프로세스 번호라고 합니다. 여러 개의 프로세스는 프로세스 스케줄러에 의해 우선적으로 처리해야 할 프로세스와 나중에 처리해야 할 프로세스의 순서가 결정됩니다. 프로세스 스케줄러는 일련의 스케줄링 방식에 의거 스케줄을 조정하기 때문에 프로세스의 상태 전이가 발생하게 되는 것입니다.

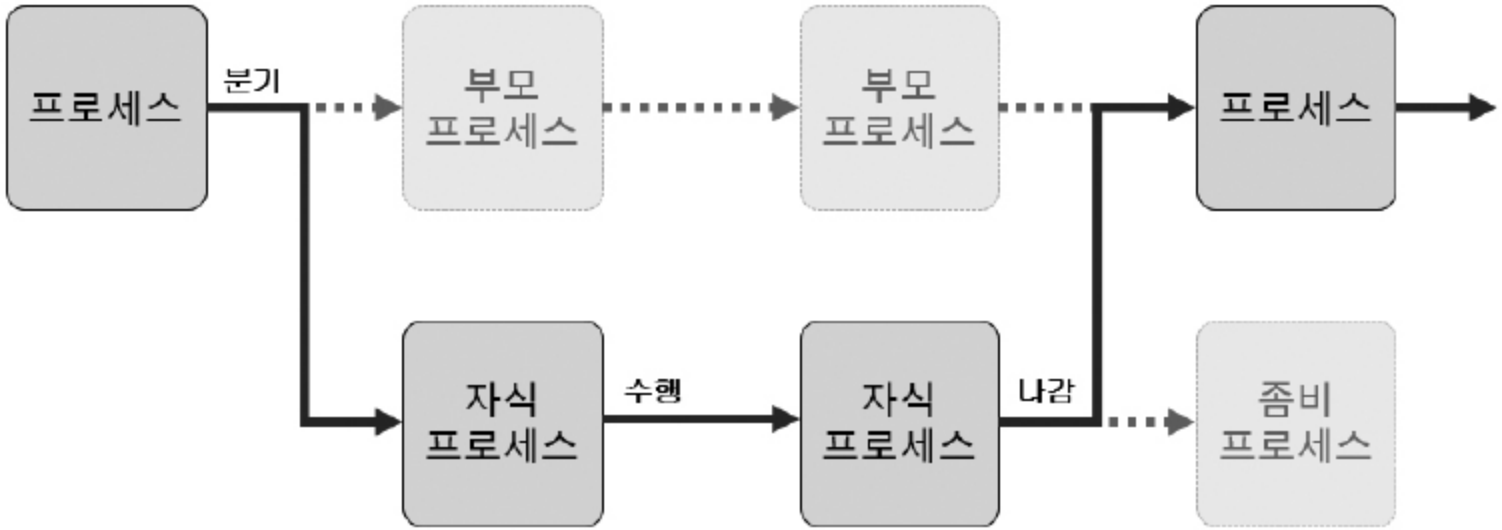
작업 번호

현재 실행되는 백그라운드 프로세스가 CPU를 점유하여 작업을 수행할 때의 순차번호를 작업번호라고 합니다.

부모 프로세스와 자식 프로세스

시스템에서 수행되는 모든 프로세스는 하나의 독립된 프로세스만이 구동되는 것은 아닙니다. 처음에 실행하는 프로그램의 메뉴에서 종속된 프로그램을 수행하게 되면 두 개의 프로세스가 구동하게 됩니다. 먼저 실행한 메인 프로그램을 부모 프로세스라고 하고 나중에 메뉴에서 선택하여 실행한 프로그램을 자식 프로세스라고 합니다.

부모 프로세스는 PPID[Parents Processor ID]를 가지며 자식 프로세스는 PID[Processor ID]를 가지고 있으므로 부모 프로세스인지 자식 프로세스인지가 구분됩니다. 부모 프로세스를 종료하게 되면 종속된 자식 프로세스 또한 강제로 종료됩니다. 부모 프로세스와 자식 프로세스와의 관계를 [그림 10-2]와 같이 정리하였습니다.



[그림 10-2] 부모 프로세스와 자식 프로세스의 관계

좀비 프로세스

자식 프로세스가 정상적으로 종료될 때는 종속된 부모 프로세스에게 종료 정보를 보내게 됩니다. 부모 프로세스가 자식 프로세스의 종료 정보를 받으면 커널에 존재하는 PCB[Process Control Block]에서 자식 프로세스에 대한 정보가 제거되어야 합니다. 하지만 자식 프로세스가 종료하였음에도 불구하고 PCB 목록에 남아 있어 마치 살아 있는 프로세스인 것처럼 간주되는 프로세스를 좀비 프로세스[Zombie Process]라고 합니다. 이와 같이 좀비 프로세스가 발생하게 되는 이유는 부모 프로세스가 자식 프로세스의 종료를 기다린 뒤에 종료를 처리해 주는 정상적인 과정을 거치지 않고 부모 프로세스를 종료할 경우에 발생하게 됩니다.

프로세스 명령어

프로세스 상태 확인 : ps

 # ps

기능 현재 실행 중인 프로세스의 상태 확인

형식 ps [옵션] Enter↵

옵션 -a : 터미널에서 실행한 프로세스의 정보 출력
-e : 시스템에서 실행 중인 모든 프로세스의 정보 출력
-f : 프로세스에 대한 상세한 정보 출력
-m : 프로세스가 사용하는 메모리의 정보 출력
-r : 현재 실행 중인 프로세스의 정보 출력
-u : 프로세스 소유자, CPU와 메모리 사용량 등 상세한 정보 출력
-x : 시스템에서 실행 중인 모든 프로세스 정보 출력
-u uid : 특정 사용자에 대한 모든 프로세스의 정보 출력
-p pid : pid로 지정한 특정 프로세스의 정보 출력
--pid PID목록 : 목록으로 지정한 특정 PID 정보 출력

예제 10-1

터미널 창에서 프로세스의 상태에 대한 모든 정보를 출력하기 위해 ps 명령과 옵션 -aux를 함께 선언하여 사용합니다.

```
# ps -aux
```

root@localhost:~

파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
[root@localhost ~]# ps -aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.1  0.3 128092  6696 ?        Ss   12:47   0:02 /usr/lib/systemd/systemd --sw
root          2  0.0  0.0      0     0 ?        S    12:47   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    12:47   0:00 [ksoftirqd/0]
root          7  0.0  0.0      0     0 ?        S    12:47   0:00 [migration/0]
root          8  0.0  0.0      0     0 ?        S    12:47   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        R    12:47   0:00 [rcu_sched]
root         10  0.0  0.0      0     0 ?        S    12:47   0:00 [watchdog/0]
root         12  0.0  0.0      0     0 ?        S<   12:47   0:00 [khelper]
```

[그림 10-3] 현재 시스템에서 사용 중인 모든 프로세스의 정보 출력

ps 명령과 함께 -aux 옵션으로 출력한 프로세스의 상태결과에 대한 항목들에 대해 자세히 살펴보도록 하겠습니다.

■ ps -aux 출력 결과

```
USER    PID %CPU %MEM     VSZ    RSS  TY   STAT   START   TIME  COMMAND
root     1     0.1     0.3  128092  6696  ?    Ss    12:47   0:02  /usr/lib/...
 :       :      :         :       :    :   :    :      :      :         :
```

출력 결과화면에 나타난 항목들이 어떠한 의미를 가지고 있는지에 대해 [표 10-1]과 같이 정리하였습니다.

[표 10-1] ps -aux 출력 정보

항목	의미
USER	프로세스의 소유자
PID	실행 중인 프로세스를 구별하기 위한 프로세스의 고유 ID
%CPU	프로세스가 CPU를 점유하는 비율
%MEM	프로세스가 메모리를 점유하는 비율

<계속>

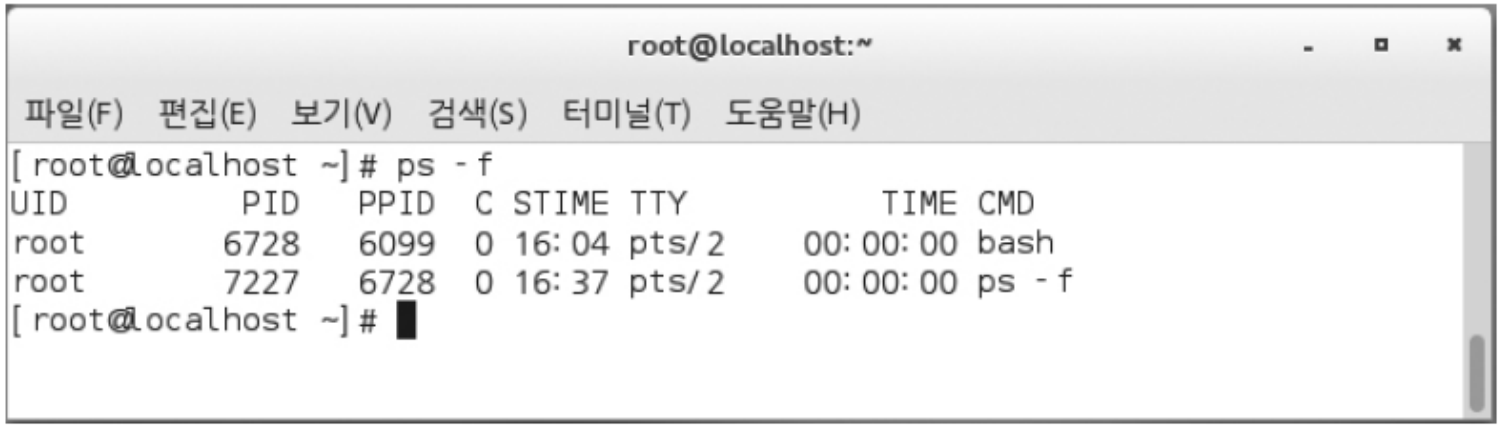
VSZ	프로세스가 사용 중인 가상 메모리의 크기(KB)
RSS	프로세스가 실제 사용 중인 물리적 메모리의 크기(KB)
TY	프로세스가 시작되고 있는 터미널
STAT	프로세스의 현재 상태 R : 현재 실행되고 있는 프로세스(running) S : 잠시 멈춘 상태로 인터럽트가 가능한 상태(sleep) → 20초 이내 D : 디스크의 입출력을 기다리는 상태(In disk wait) T : 작업 제어에 의해 정지된 상태(stopped) Z : 좀비 프로세스(defunct) STIME : 프로세스의 시작 날짜 또는 시간 s : 세션 리더 프로세스 + : 포그라운드 프로세스 그룹 l(소문자 L) : 멀티스레드
START	프로세스가 시작된 시각
TIME	현재까지 사용된 CPU의 시간(분:초)
COMMAND	프로세스가 실행한 명령행

프로세스 상세 정보 출력 : -f 옵션

예제 10-2

터미널 창에서 프로세스의 상태에 대한 상세한 정보를 출력하기 위해 ps 명령과 옵션 -f를 함께 선언하여 사용합니다.

```
# ps -f
```



[그림 10-4] 프로세스의 상세 정보

[그림 10-4]에서 출력된 항목들이 어떤 의미를 가지고 있는지에 대해 [표 10-2]와 같이 정리하였습니다.

[표 10-2] 프로세스의 상세한 정보

항목	의미	항목	의미
UID	프로세스를 실행한 계정 ID	STIME	프로세스의 시작 날짜 또는 시각
PID	프로세스 번호	TTY	프로세스가 실행된 터미널 종류와 번호
PPID	부모 프로세스 번호	TIME	프로세스 실행 시간
C	CPU 사용량 (%로 표시)	CMD	실행되고 있는 프로그램 명령이름

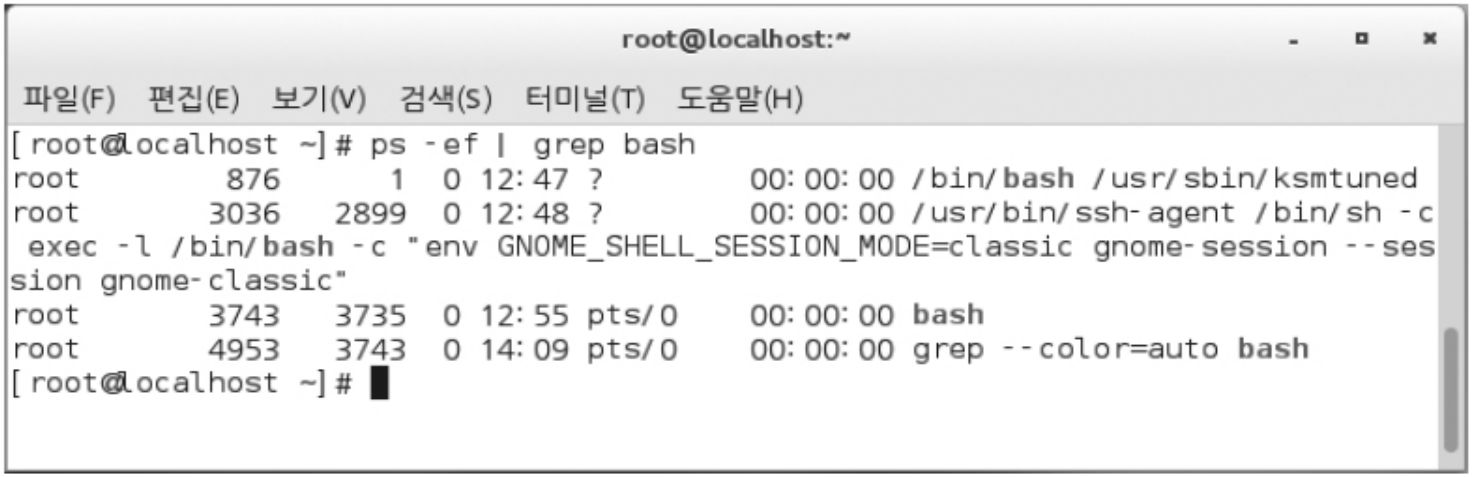
특정 프로세스 정보 검색

ps 명령으로 특정 프로세스 정보 검색

| 예제 10-3 |

bash 셸에 대한 프로세스의 정보를 검색하기 위해 ps -ef | grep 명령을 수행합니다.

```
# ps -ef | grep bash
```



[그림 10-5] bash 셸에 대한 프로세스의 정보 검색

grep 명령으로 특정 프로세스 정보 검색

 # pgrep

기능 패턴 지정으로 특정 프로세스에 대한 정보 출력

형식 pgrep [옵션] [패턴] Enter↵

옵션 -l : PID와 프로세스의 이름 출력

-n : 패턴을 포함하고 있는 최신 프로세스의 정보 출력

-t term : 특정 단말기와 관련된 프로세스의 정보 출력

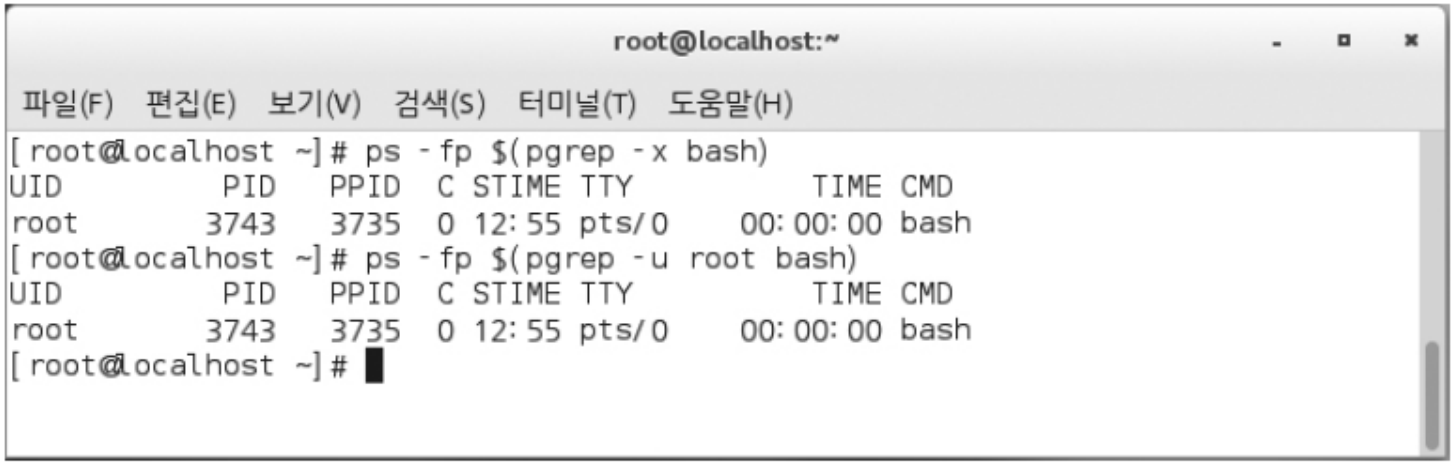
-u 사용자명 : 특정 사용자에 대한 모든 프로세스의 정보 출력

-x : 주어진 패턴과 정확히 일치하는 프로세스의 정보 출력

예제 10-4

패턴을 bash로 지정하여 프로세스의 정보를 검색하기 위해서는 ps 명령과 pgrep 명령을 옵션과 함께 선언하여 명령을 수행합니다.

```
# ps -fp $(grep -x bash)          → 시스템에 접속된 모든 사용자 검색됨
# ps -fp $(grep -x cskisa bash)   → cskisa 계정에 대한 정보만 검색
```



[그림 10-6] 패턴과 사용자 계정을 지정하여 프로세스 정보 검색

프로세스 종료

kill 명령으로 프로세스 종료

프로세스를 종료하는 kill 명령은 종료할 프로세스에 인자로 지정한 숫자 메시지를 시그널로 보내 해당 프로세스를 종료합니다. 프로세스에게 보내는 시그널은 인터럽트, 프로세스 종료, 강제 종료 등의 숫자로 기능이 지정되어 있으며 사용형식은 다음과 같습니다.

 # kill

기능 프로세스 종료를 위해 지정한 시그널을 해당 프로세스에 전달

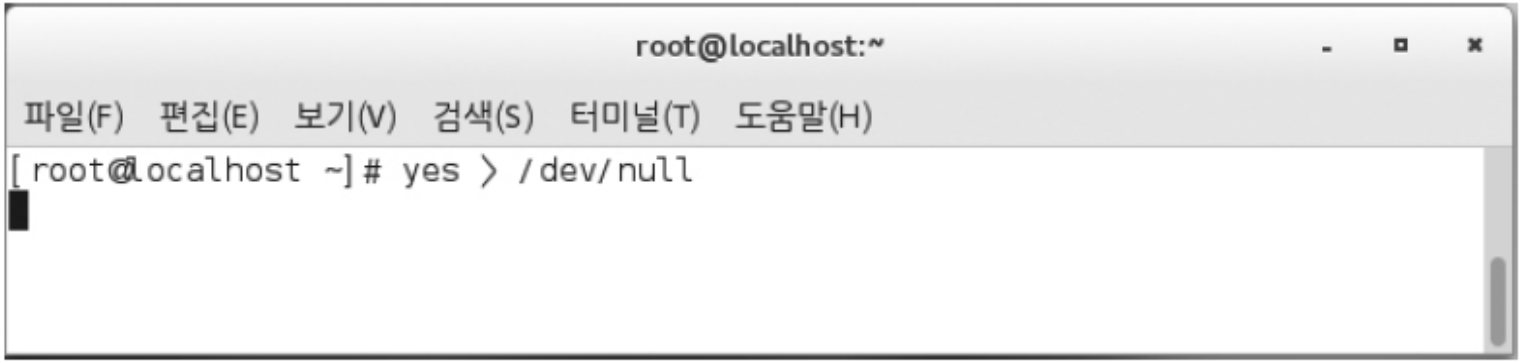
형식 kill [시그널] PID Enter↵

시그널 -2 : 인터럽트 시그널 전송 (Ctrl+C)
-9 : 프로세스 강제 종료
-15 : 프로세스가 관련 파일을 정리 후 종료 (종료되지 않는 프로세스도 있음)

예제 10-5

- **Step 01** | 첫 번째 터미널 창을 열어서 무한루프 명령을 수행하도록 다음과 같이 명령을 실행합니다. yes 명령은 단순히 yes라는 문자열을 화면에 계속 반복해서 출력하라는 의미이고 /dev/null은 아무런 반응을 하지 않는 장치를 선언한 것입니다.

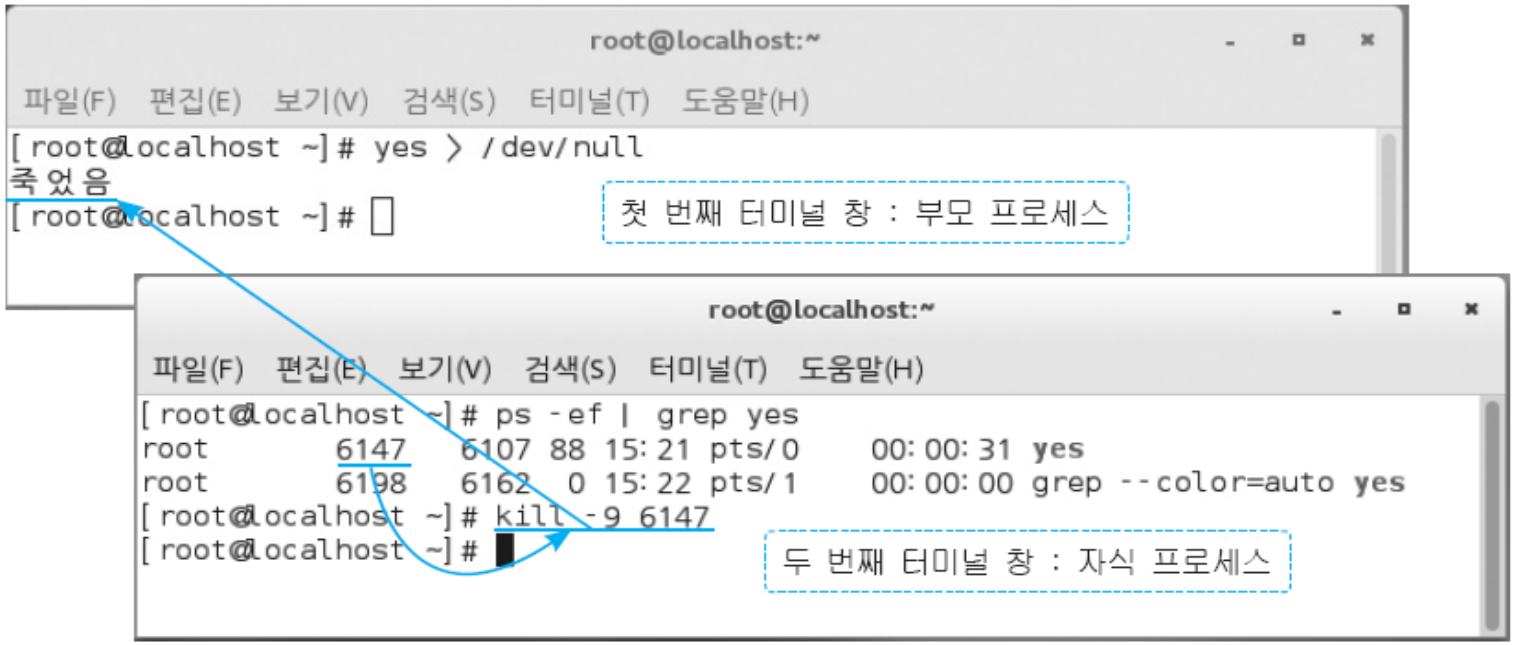
```
# yes > /dev/null
```



[그림 10-7] yes 문자열을 무한반복 출력

- **Step 02** | 두 번째 터미널 창을 열어서 첫 번째 터미널(부모 프로세스)에서 수행하고 있는 프로세스의 PID를 확인한 다음 kill 명령과 시그널 -9 그리고 PID를 지정하여 부모 프로세스를 강제로 종료합니다.

```
# ps -ef | grep yes
```



[그림 10-8] PID를 이용하여 부모 클래스에서 수행한 프로세스 종료

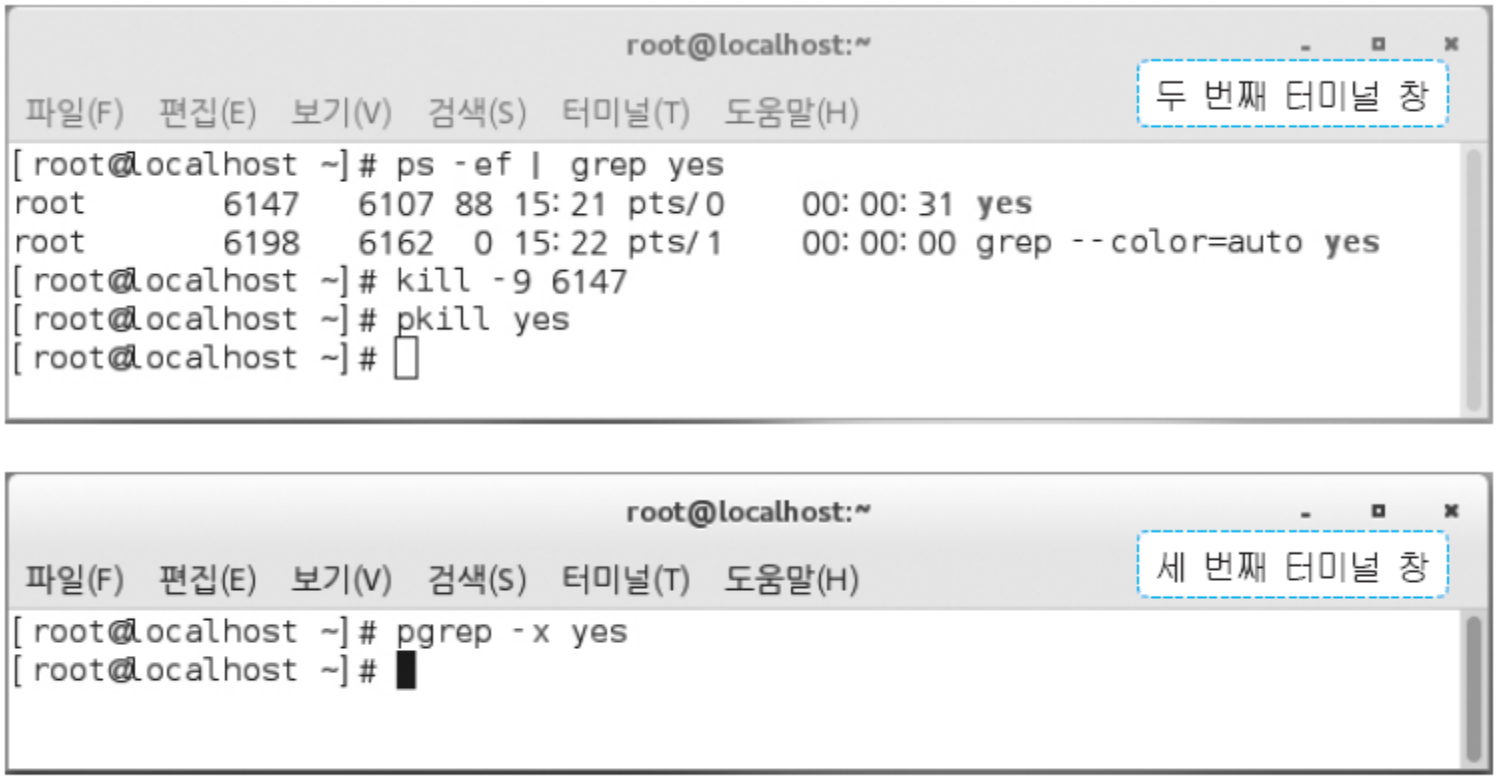
pkill 명령으로 프로세스 종료

pkill 명령은 프로세스를 종료할 때 kill 명령과 같이 시그널을 보내는 방식은 같지만 다른 점이 있다면 pkill 명령은 PID를 보내는 것이 아니라 프로세스의 명령이름[CMD]으로 프로세스를 찾아서 종료해 준다는 점이 kill 명령과 다른 점입니다. pkill 명령은 같은 명령으로 수행한 프로세스를 명령이름으로 찾아주기 때문에 한꺼번에 같은 이름의 명령을 찾아서 모두 종료할 수 있다는 편리성을 제공해 줍니다. 사용방법이 비교적 간단하므로 다음 예제를 통해 살펴보도록 하겠습니다.

예제 10-6

[예제 10-5]에서 수행한 첫 번째 터미널 창과 두 번째 터미널 창에서 pkill yes 명령을 각각 입력하여 수행하고 세 번째 터미널 창을 열어서 pgrep -x yes 명령으로 실행한 다음 yes 프로세스가 종료되었는지를 확인합니다.

```
# pkill yes          → 첫 번째 터미널 창에서 입력
# pkill yes          → 두 번째 터미널 창에서 입력
# pgrep -x yes       → 세 번째 터미널 창에서 입력
```



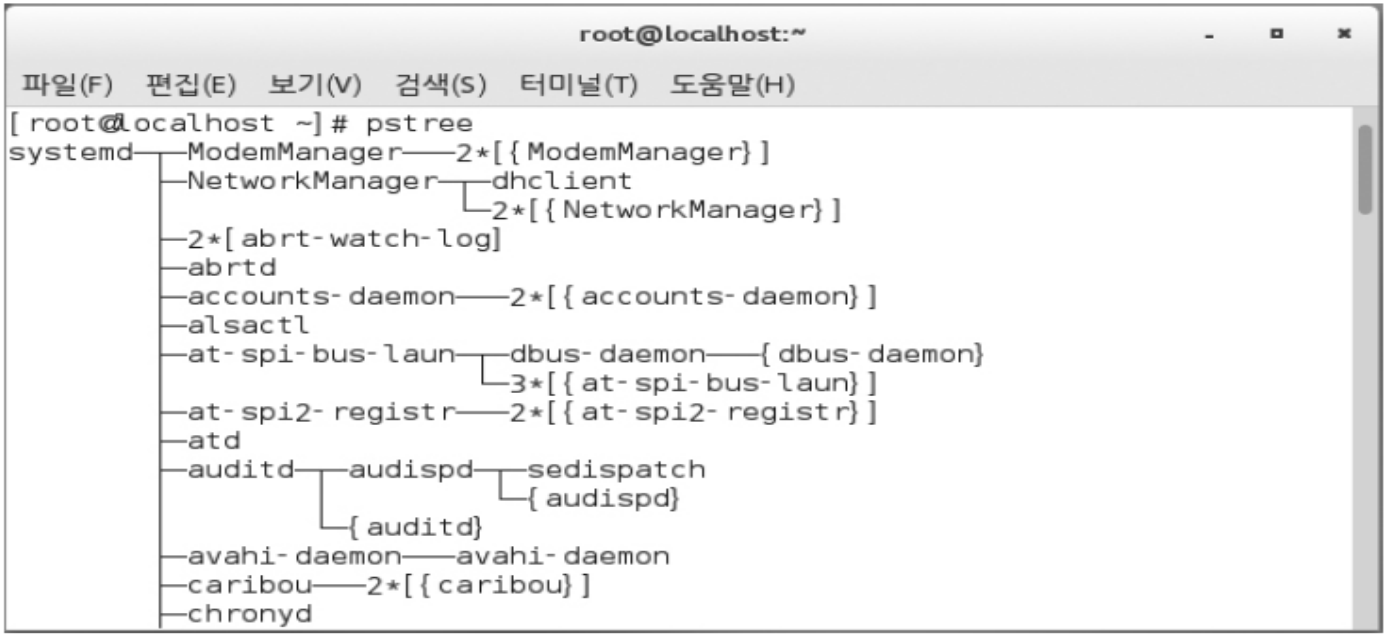
[그림 10-9] pkill 명령으로 프로세스의 명령이름이 yes인 프로세스를 찾아서 종료

프로세스의 트리 구조

| 예제 10-7 |

프로세스의 관계를 트리 형태로 출력하기 위해 터미널 창에서 pstree 명령을 수행합니다.

```
# pstree
```



[그림 10-10] 프로세스의 관계를 트리형태로 출력

서비스와 소켓

서비스

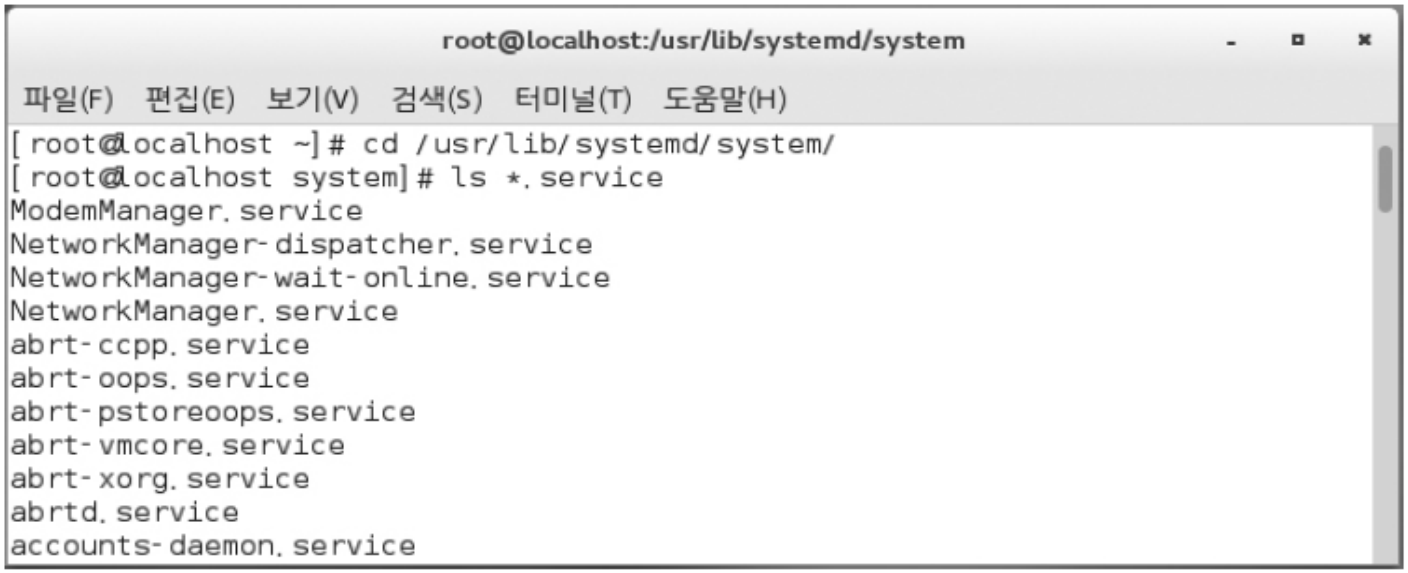
서비스Service는 데몬Daemon이라고도 하며 서버 프로세스를 의미합니다. 서버 프로세스라 함은 네임 서버, 웹 서버, DB 서버 등의 프로세스 또는 네임 서버 데몬, 웹 서버 데몬, DB 서버 데몬 등으로 표현하기도 합니다. 서비스는 눈에 보이지 않는 무형의 형태로 현재 시스템에서 동작 중인 프로세스이므로 백그라운드 프로세스의 일종이라고도 할 수 있습니다.

이와 같이 서비스는 시스템과 상관없이 독자적으로 구동되어 제공되는 프로세스를 의미하며 실행과 종료는 'systemctl start/stop/restart 서비스 이름' 형식으로 사용됩니다. 서비스의 실행 스크립트 파일은 /usr/lib/systemd/system/ 디렉터리에 존재하며 시스템에서 제공하는 서비스 목록을 확인할 수 있습니다. 다음 예제를 통해 살펴보도록 하겠습니다.

예제 10-8

리눅스 시스템에서 제공하고 있는 서비스의 스크립트 종류를 살펴보기 위해 다음과 같이 명령을 수행합니다.

```
# cd /usr/lib/systemd/system/
# ls *.service
```



[그림 10-11] 리눅스 시스템에서 제공하는 서비스의 스크립트

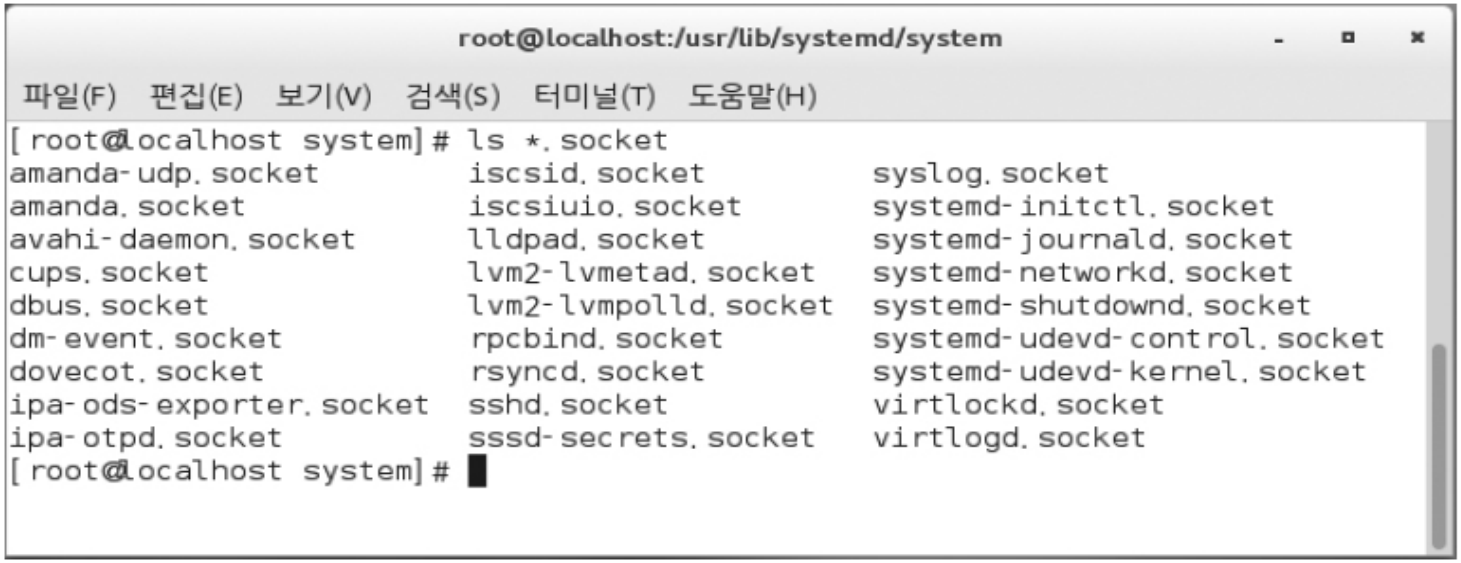
소켓

서비스 또는 데몬은 필요여부를 떠나 항상 가동하고 있는 서버 프로세스를 의미하는 반면 소켓Socket은 필요할 때만 작동하는 서버 프로세스를 의미합니다. 소켓은 외부에서 특정 서비스를 요청할 경우 systemd 서비스가 구동시키며 요청이 끝나면 소켓도 종료됩니다. 소켓과 관련된 파일은 /usr/lib/systemd/system/ 디렉터리에 '소켓이름.socket'이라는 이름으로 존재합니다. 다음 예제를 통해 살펴보도록 하겠습니다.

예제 10-9

리눅스 시스템에서 제공하고 있는 소켓의 스크립트 파일 종류를 살펴보기 위해 다음과 같이 명령을 수행합니다.

```
# cd /usr/lib/systemd/system/
# ls *.socket
```



[그림 10-12] 리눅스 시스템에서 제공하는 소켓 스크립트 파일

systemd 서비스 매니저

서비스와 소켓은 systemd라고 부르는 서비스 매니저 프로그램으로 작동시키거나 관리합니다. systemd 서비스는 리눅스의 시스템과 서비스 관리자로서 기존 유닉스의 init 프로세스가 수행하던 작업을 대신 수행합니다.

그리고 systemd 서비스는 다양한 서비스 데몬을 시작하고 프로세스들의 상태를 유지해 주며 시스템의 상태를 관리해 줍니다. systemd는 전체 시스템을 시작하고 관리하는데 유닛Units이라고 부르는 구성요소를 사용합니다. systemd 서비스 관리대상의 이름을 '서비스.유닛종류'의 형태로 관리합니다. 유닛의 종류는 [표 10-3]과 같이 정리하였습니다.

[표 10-3] systemd 서비스 유닛의 종류

유닛 종류	의미
service	가장 확실한 유닛으로 데몬을 시작/종료/재시작/로딩 수행
socket	소켓 관리
device	리눅스 장치 트리에 있는 장치 관리
mount	디렉터리 계층 구조의 마운트 포인트 관리
automount	디렉터리 계층 구조에서 자동 마운트 포인트 관리
path	경로 관리
snapshot	다른 유닛을 참조
swap	스왑 장치 관리
target	유닛들을 그루핑
timer	타이머 관련 기능 관리

systemd 서비스는 기존 유닉스에서 제공하던 init 방식에 비해 다음과 같은 특징을 가지고 있습니다.

- 셸과 독립적으로 부팅 가능
- fsck와 마운트 제어 가능
- 소켓기반 동작으로 inetd와의 호환성 유지
- 시스템 상태에 대한 스냅숏 유지
- 서비스에 시그널 전달
- 프로그램 자동 종료 전에 사용자 세션의 안전한 종료

systemd 관련 명령

 # systemctl

기능 systemd 서비스를 제어

형식 systemctl [옵션] [명령] [유닛이름] Enter↵

옵션 -a : 상태 불문하고 유닛 전체 출력

-t 유닛종류 : 지정한 종류의 유닛만 출력

명령 start : 유닛 시작

stop : 유닛 정지

reload : 유닛 설정파일 다시 읽어옴

restart : 유닛 재시작

status : 유닛 상태 출력

enable : 부팅 시 유닛 자동시작 설정

disable : 부팅 시 유닛 자동 시작하지 않도록 설정

is-active : 유닛 동작 유무 확인

is-enable : 유닛 시작 유무 확인

isolate : 지정한 유닛과 관련된 유닛만 시작하고 나머지는 정지

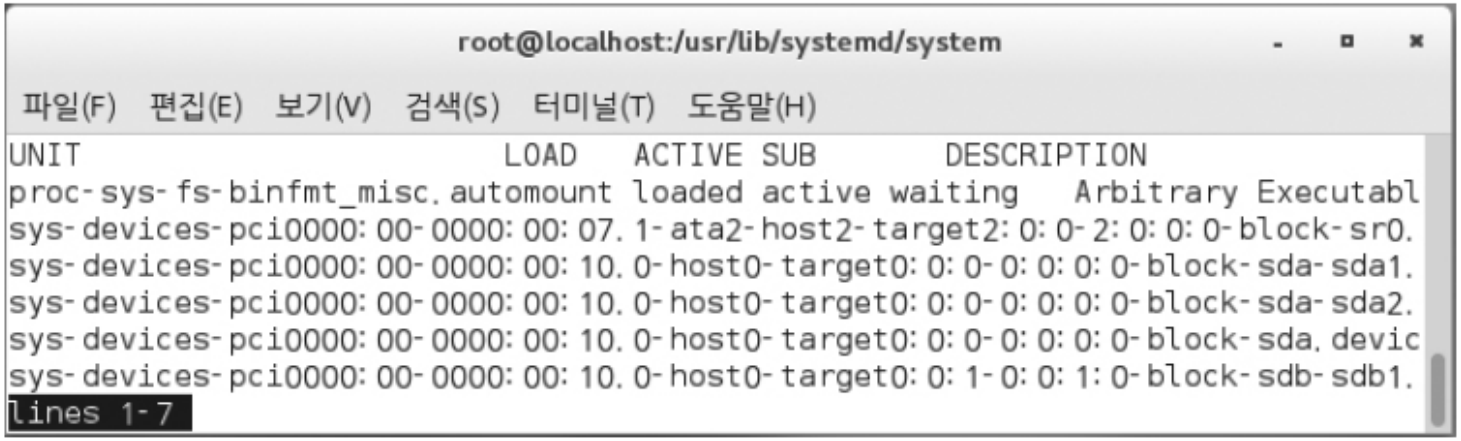
kill : 유닛에 시그널 전송

동작 중인 유닛 출력 : systemctl

예제 10-10

리눅스 시스템에서 현재 동작 중인 유닛을 출력하기 위해 다음 명령을 수행합니다.

```
# systemctl    → 종료는 q를 입력
```



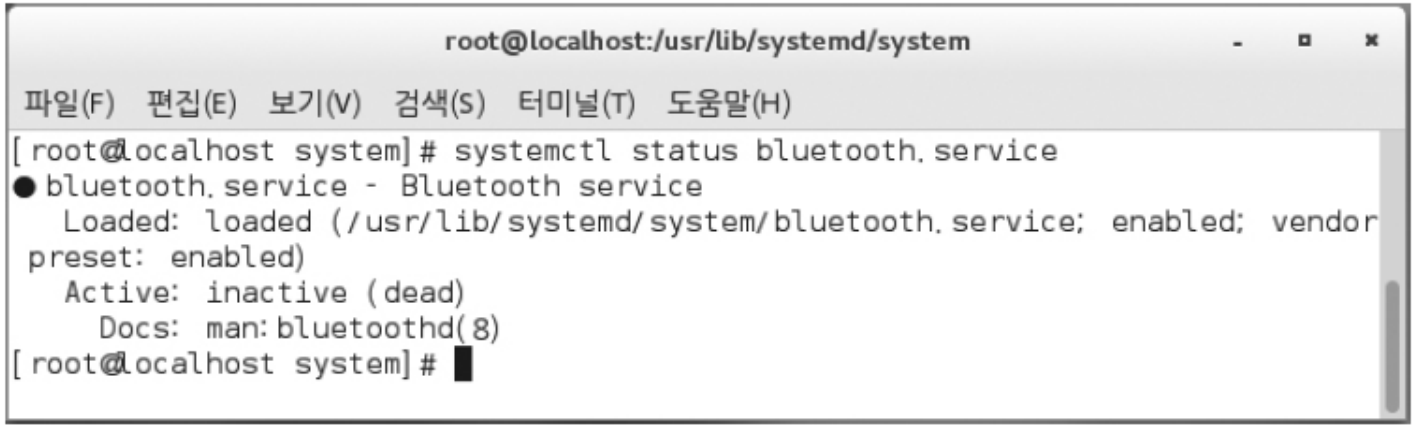
[그림 10-13] 리눅스 시스템에서 현재 동작 중인 유닛 목록

특정 유닛의 상태 확인 : statu와 stop

예제 10-11

- **Step 01** | 리눅스 시스템에서 현재 동작 중인 블루투스 관련 유닛의 상태를 출력하기 위해 다음 명령을 수행합니다.

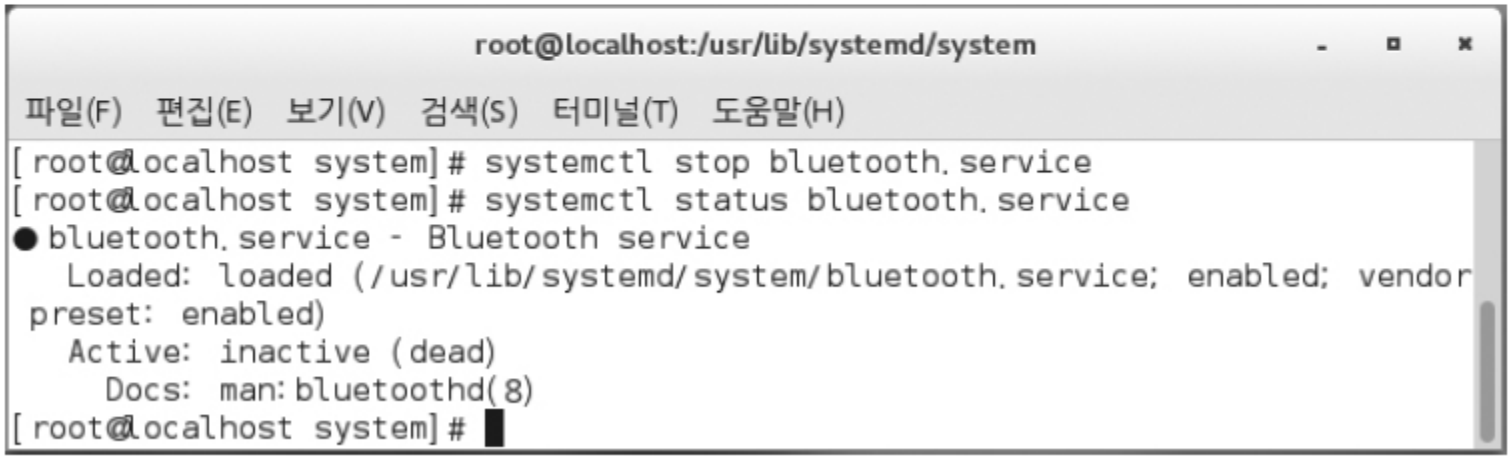
```
# systemctl status bluetooth.service
```



[그림 10-14] 블루투스 유닛 상태 확인

- **Step 02** | 리눅스 시스템에서 현재 동작 중인 블루투스 관련 유닛을 stop명령으로 정지한 다음 정지된 블루투스의 유닛 상태를 확인하기 위해 다음 명령을 수행합니다.

```
# systemctl stop bluetooth.service
# systemctl status bluetooth.service
```



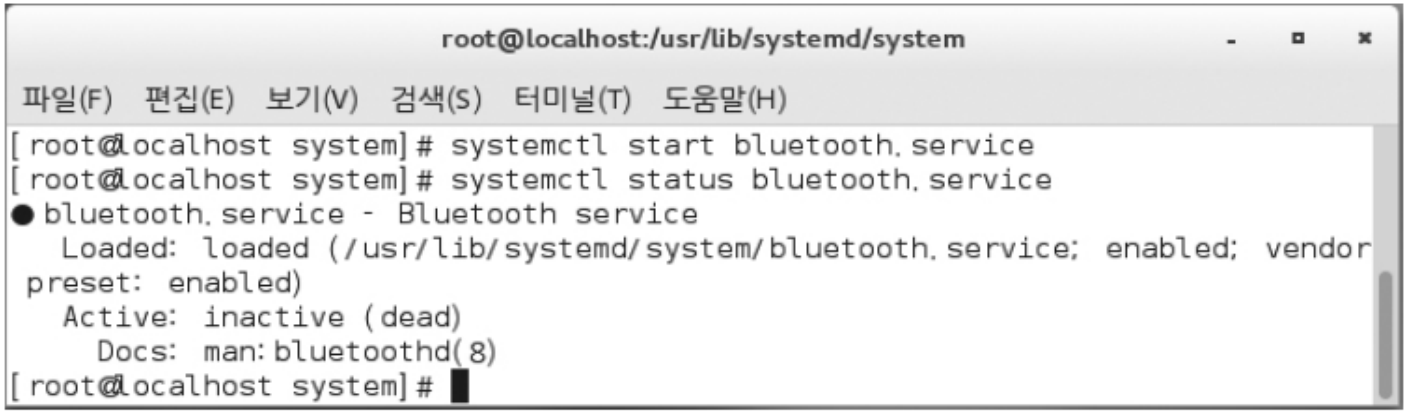
[그림 10-15] 블루투스 유닛 정지 후 상태 확인

특정 유닛의 서비스 시작 : start

| 예제 10-12 |

블루투스 관련 유닛을 정지한 상태에서 start 명령으로 유닛 서비스를 시작한 다음 블루투스 유닛의 상태를 확인하기 위해 다음 명령을 수행합니다.

```
# systemctl start bluetooth.service
# systemctl status bluetooth.service
```



[그림 10-16] 정지된 블루투스 유닛 시작 후 상태 확인

실습 10-1 다음 항목에서 주어진 지시사항을 수행하시오.

1. 서비스 스크립트가 존재하는 디렉터리로 이동하기
2. 리눅스 시스템에서 제공하고 있는 서비스의 스크립트 정보 확인하기
3. usbmuxd.service 유닛의 상태를 status 명령으로 확인하기
4. usbmuxd.service 유닛 정지 후 유닛 상태를 status 명령으로 확인하기
5. usbmuxd.service 유닛을 다시 시작하기
6. usbmuxd.service 유닛 실행 상태에서 PID와 PPID 확인하기

```
# cd /usr/lib/systemd/system/
# ls *.service
# systemctl status usbmuxd.service
# systemctl stop usbmuxd.service
# systemctl status usbmuxd.service
# systemctl start usbmuxd.service
# ps -f
```

응급 복구

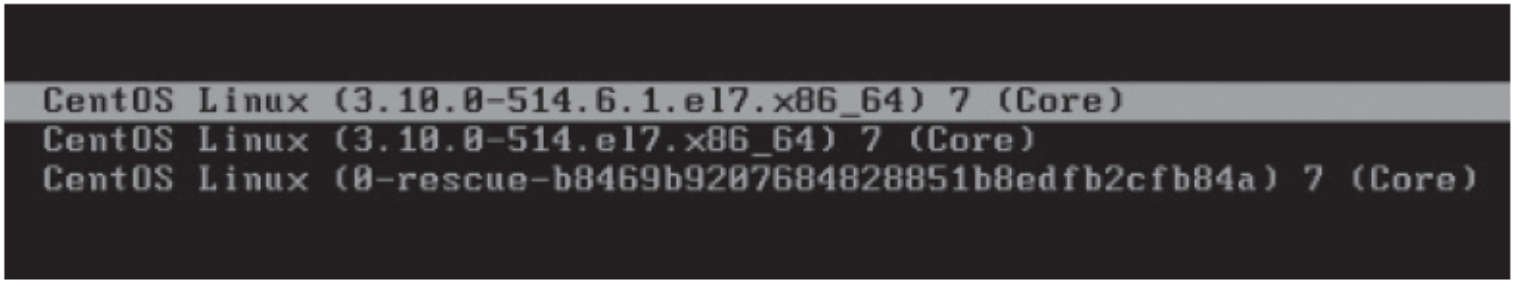
- 예기치 못한 상황에 대처하기 위한 응급 복구 이해하기

1. 응급 복구에 대한 개념을 이해합니다.
2. root 암호 재설정 응급 복구에 대해 실습합니다.
3. 암호 재설정 후 마운트를 다시 설정하는 사용방법에 대해 살펴봅니다.

root 암호 재설정 응급 복구

예제 10-13

- **Step 01** | 리눅스 시스템을 root 계정으로 접속하려고 할 때 암호가 기억나지 않는다는 전제하에 실습을 진행하도록 하겠습니다. 부팅 시 처음에 나타나는 GRUB 화면에서 Edit를 의미하는 알파벳 E를 누릅니다.



[그림 10-17] GRUB 메뉴에서 E를 누름

- **Step 02** | 키보드의 ↓ 방향키를 이용하여 아래쪽 있는 문장 중에서 linux16 /vmlinuz-3.10.0-514.6.1.el7.x86~ 행 앞에 커서를 놓고 Esc를 눌러 행의 끝으로 커서를 이동한 다음 rhgb quiet LANG=kor_KR.UTF-8을 지웁니다. 지운 위치에서 몇 칸을 띄운 다음 init=bin/sh를 입력한 다음 Ctrl+X를 눌러 부팅합니다.

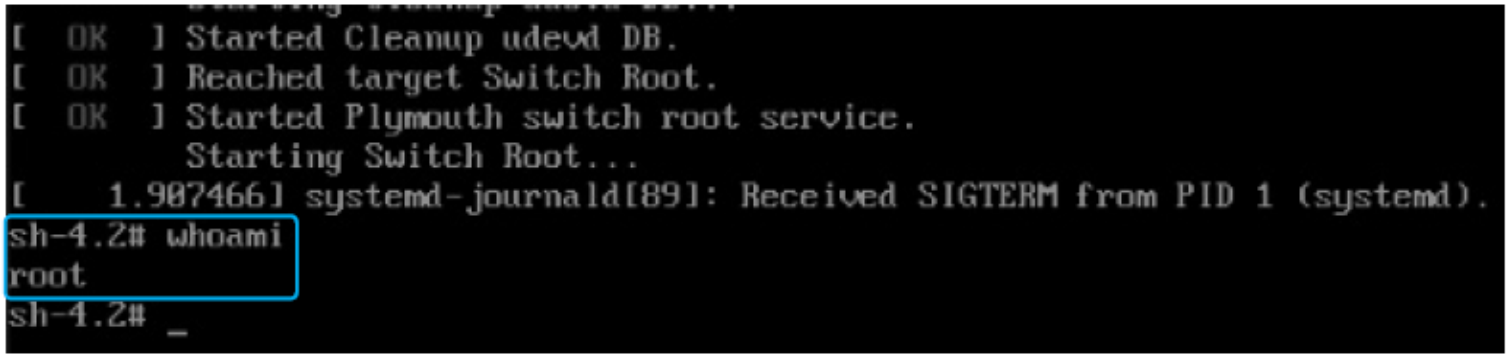
```
        insmod xfs                                                      ↑
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hin\
t-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1'  7d7300b0-c\
470-4fa6-be38-f09f47fa8673
        else
          search --no-floppy --fs-uuid --set=root 7d7300b0-c470-4fa6-be38-f09f\
47fa8673
        fi
        linux16 /vmlinuz-3.10.0-514.6.1.el7.x86_64 root=/dev/mapper/cl-root ro\
 crashkernel=auto rd.lvm.lv=cl/root rd.lvm.lv=cl/swap rhgb quiet LANG=ko_KR.UT\
F-8
        initrd16 /initramfs-3.10.0-514.6.1.el7.x86_64.img


      Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
      discard edits and return to the menu. Pressing Tab lists
      possible completions.
```

[그림 10-18] rhgb quiet LANG=ko_KR.UTF-8을 지우고 init=/bin/sh 입력

- **Step 03** | 부팅이 되면서 sh4-2#이라는 프롬프트가 나타납니다. 현재 로그인된 사용자를 확인하기 위해 다음 명령을 수행하면 로그인 사용자가 root 임을 보여줍니다.

```
sh4-2# whoami
```



[그림 10-19] 로그인 사용자 계정 확인

- **Step 04** | root 사용자의 암호를 변경하기 위해 passwd 명령어를 입력하고 새로운 암호를 8자 이상으로 설정합니다.

```
sh4-2# passwd
New password : 123456
Retype new password : 123456
```

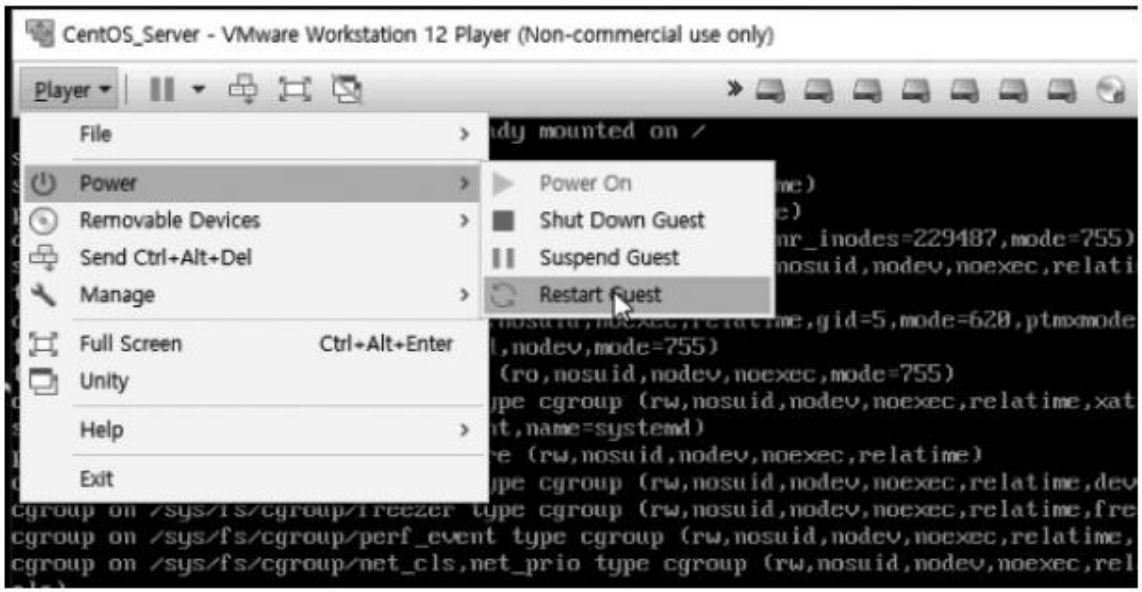
```
root
sh-4.2# passwd
Changing password for user root.
New password:
Retype new password:
passwd: Authentication token manipulation error
sh-4.2#
```

[그림 10-20] root 사용자의 암호 변경 설정

- **Step 05** | 비밀번호를 123456으로 설정하였더니 에러 메시지가 출력되었습니다. 그 이유는 현재 root 파티션이 읽기 전용으로 마운트가 설정되어 있기 때문입니다. root 파티션을 읽기/쓰기(rw) 모드로 마운트를 다시 설정한 다음 마운트를 실행합니다.

```
sh4-2# mount -o remount.rw /
New password : 123456
Retype new password : 123456
sh4-2# mount
```

- **Step 06** | VMware 메뉴에서 시스템을 강제로 재부팅하면 변경한 root 계정의 암호로 접속할 수 있습니다.



[그림 10-21] 시스템 강제 재부팅

이와 같은 방법으로 root 계정의 암호를 변경할 경우 관리자 이외에 타인이 임의로 암호를 변경할 수도 있는 부작용이 발생하게 되므로 시스템 보호를 위해 반드시 처음 부팅할 때 나오는 GRUB 자체를 편집할 수 없도록 설정해야 합니다. 이 부분을 다음 섹션에서 다루기로 하겠습니다.

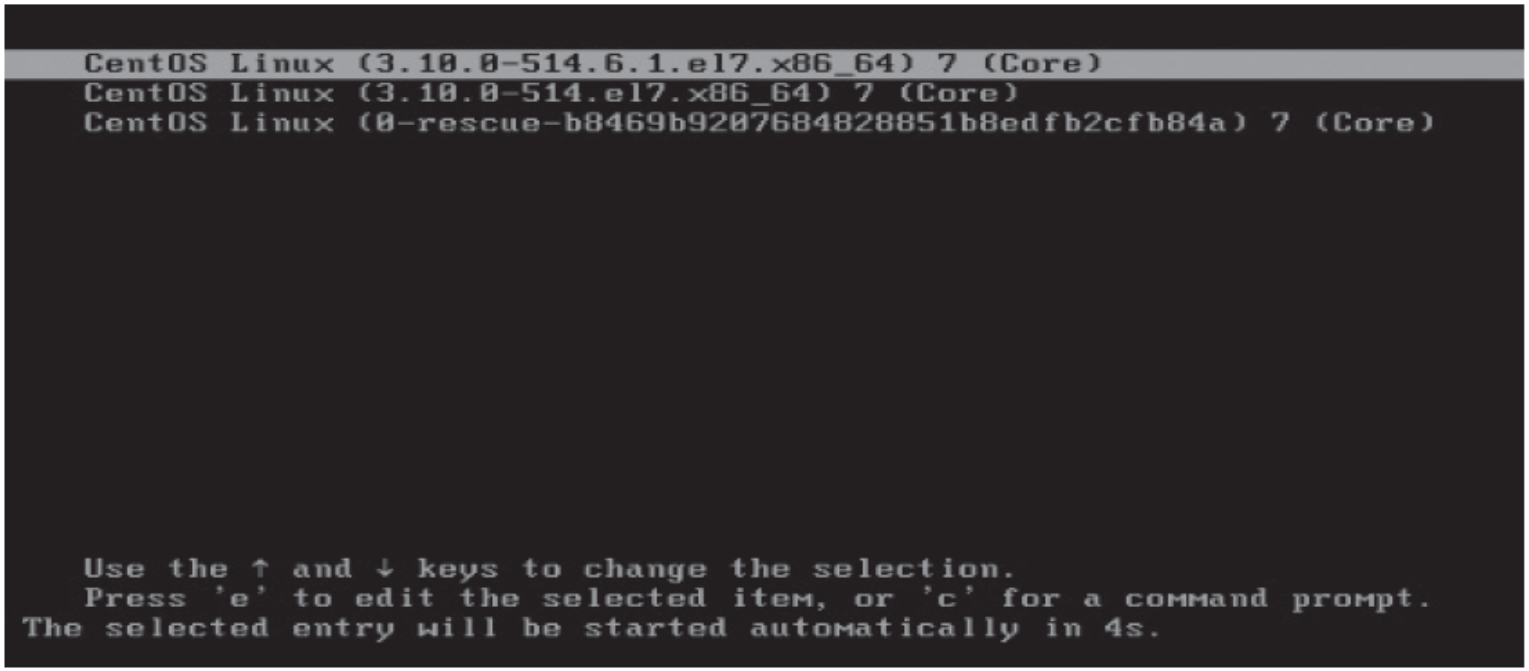
GRUB 부트로더

- GRUB 부트로더 이해하기

1. GRUB의 개념과 기능에 대해 이해합니다.
2. GRUB 부트로더의 이해와 사용방법에 대해 살펴봅니다.
3. GRUB 부트로더 전용 사용자와 암호 설정방법에 대해 실습합니다.

GRUB 개념

GRUB[GRand Unified Bootloader]는 리눅스 계열에서 가장 많이 사용하는 부트로더입니다. 부트로더[Boot loader]란 컴퓨터가 부팅되면서 처음으로 실행되는 프로그램으로 운영체제를 불러오는 역할을 수행합니다. 최근의 CentOS에서는 기본 부트로더를 이전의 GRUB보다 더 향상된 GRUB2 버전을 사용합니다. GRUE2 부트로더 화면은 [그림 10-22]와 같이 시스템을 부팅할 때 처음으로 나오는 선택화면을 의미합니다.



[그림 10-22] GRUB2 부트로더 화면

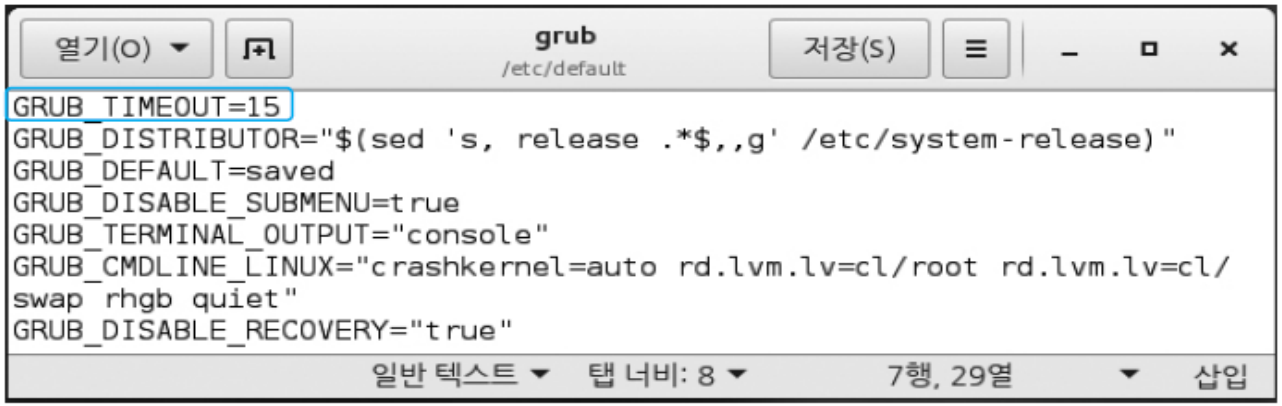
GRUB 기능

부팅 대기시간 설정

| 예제 10-14 |

- **Step 01** | root 계정으로 접속하여 터미널 창에서 gedit로 /etc/default/grub 파일을 열어서 부팅 시간을 15초로 변경하고 저장한 다음 gedit 창을 닫습니다.

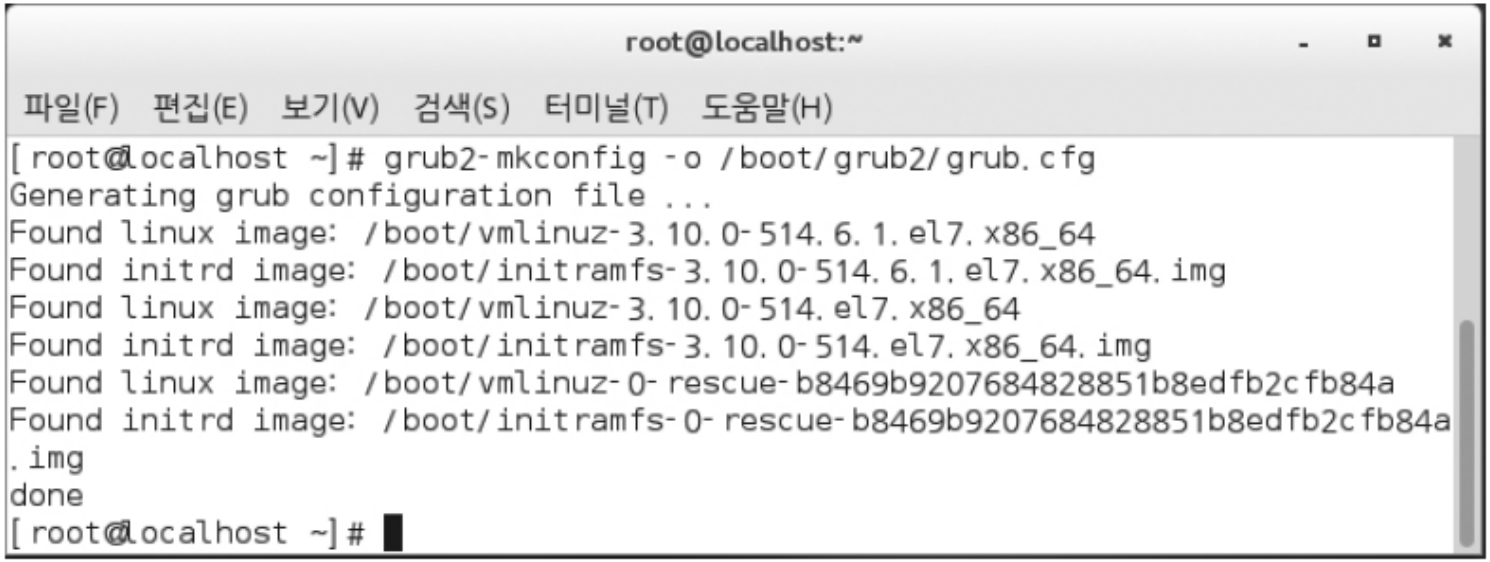
```
# gedit /etc/default/grub
```



[그림 10-23] 부팅시간 설정 변경

- **Step 02** | 부팅시간을 변경 한 다음에는 변경된 내용을 적용하기 위해 다음과 같이 명령을 수행합니다.

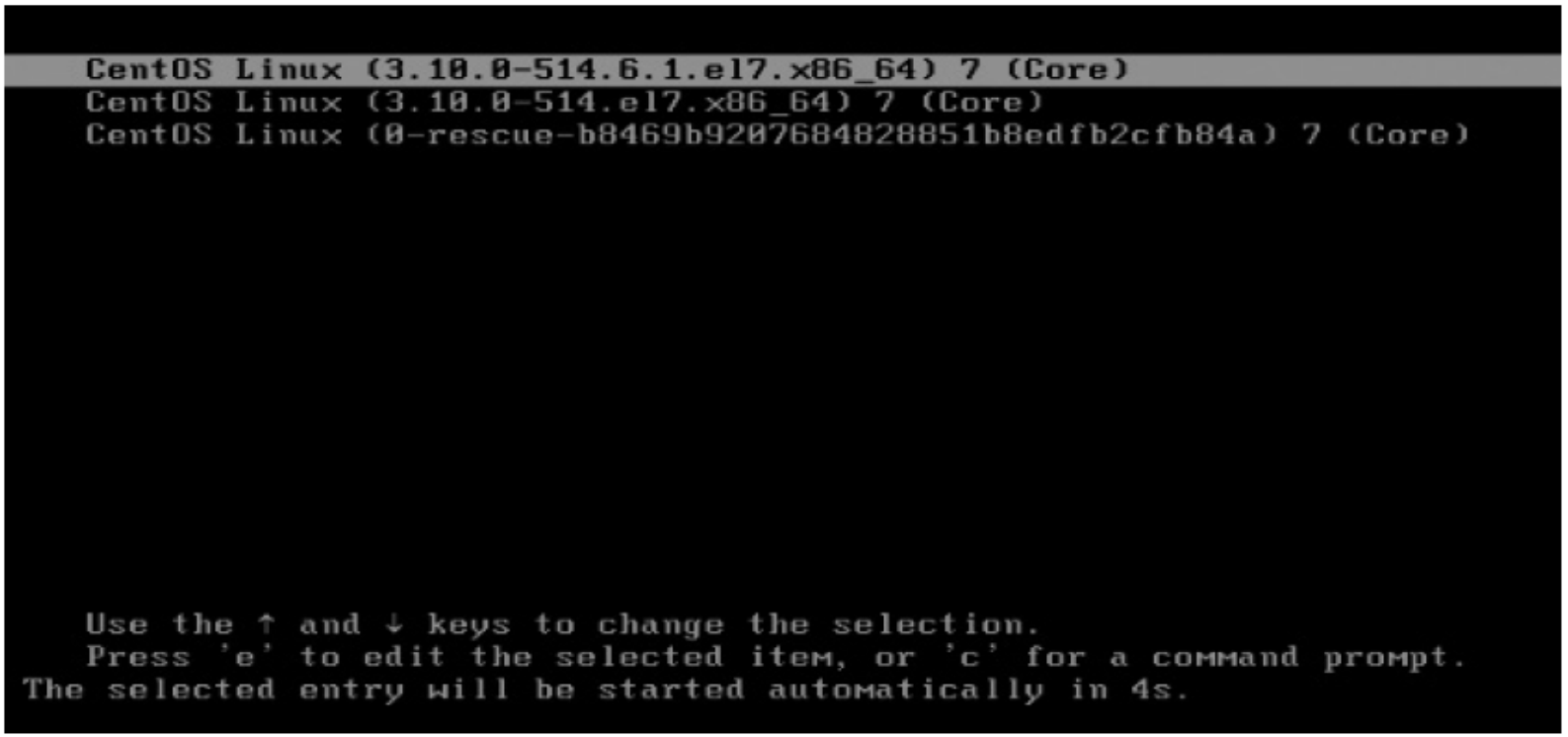
```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```



[그림 10-24] 변경된 GRUB 내용 적용

- **Step 03** | 터미널 창에서 reboot 명령으로 시스템을 재부팅하면 GRUB의 초기화면이 나타나며 15초 동안 대기하게 됩니다.

```
# reboot
```



[그림 10-25] 변경된 GRUB 부팅 화면

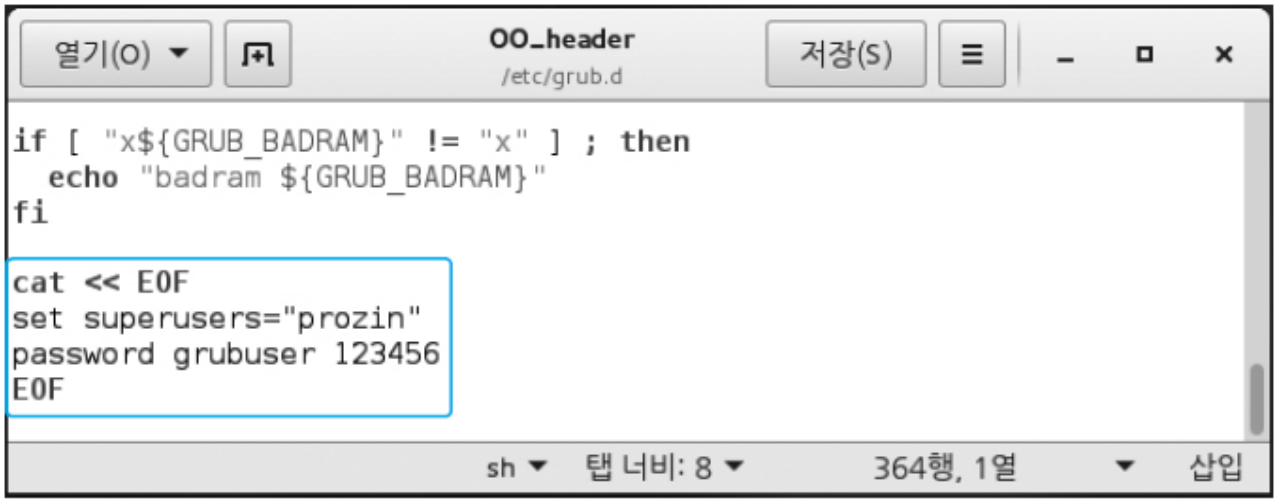
GRUB 전용 사용자와 암호 설정

예제 10-15

- **Step 01** | 터미널 창에서 gedit로 /etc/grub.d/00_header 파일을 열어서 맨 아래쪽에 4개의 행을 추가하여 저장합니다. 여기에서 지정하는 GRUB 사용자는 기존 리눅스 사용자와는 관련이 없으므로 새로 지정하면 됩니다.

```
# gedit /etc/grub.d/00_hearder

cat << EOF
set superusers="prozin"
password grubuser 123456
EOF
```



[그림 10-26] /etc/grub/00_header 파일 편집

- **Step 02** | 변경된 내용을 적용하기 위해 다음 명령을 수행하고 시스템을 재부팅합니다.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
# reboot
```

- **Step 03** | GRUB 부트로더를 편집하려면 반드시 앞에서 설정한 전용 사용자 prozin과 암호 123456을 입력해야만 편집할 수 있습니다. E를 누르면 GRUB 부트로더를 편집으로 들어가기에 앞서 전용 사용자와 암호를 입력하라는 창이 나타납니다.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
# reboot
```

E

```
Enter username :
prozin
Enter password :
123456
```

Ctrl + X

[그림 10-27] GRUB 전용 사용자와 암호 임력

BRUB 부트로더 편집 창에서 전용 사용자와 암호를 입력한 다음에는 Ctrl+X를 눌러 부팅되도록 합니다.

실습 10-2 다음 항목에서 주어진 지시사항을 수행하시오.

1. GRUB 부트로더 부팅 대기시간을 30초로 변경하기
2. GRUB 전용 사용자와 암호 설정하기
   - 전용 사용자 : space
   - 암호 : 123456
3. 설정된 GRUB 전용 사용자와 암호 입력하여 편집모드로 들어가기
4. GRUB 편집 창 닫고 부팅하기

# gedit /etc/default/grub
GRUB_TIMEOUT=30 → 값 수정
gedit 저장 후 종료
grub2-mkconfig –o /boot/grub2/grub.cfg
reboot
gedit /etc/grub.d/00_hearder
cat << EOF
set superusers=“prozin”
password grubuser 123456
EOF
gedit 저장 후 종료
grub2-mkconfig –o /boot/grub2/grub.cfg
reboot
E
[Ctrl] + [X]

Chapter 10

최상의 노력에 따른 인고의 가치는 반드시 증명될 수 있습니다!

Thank You