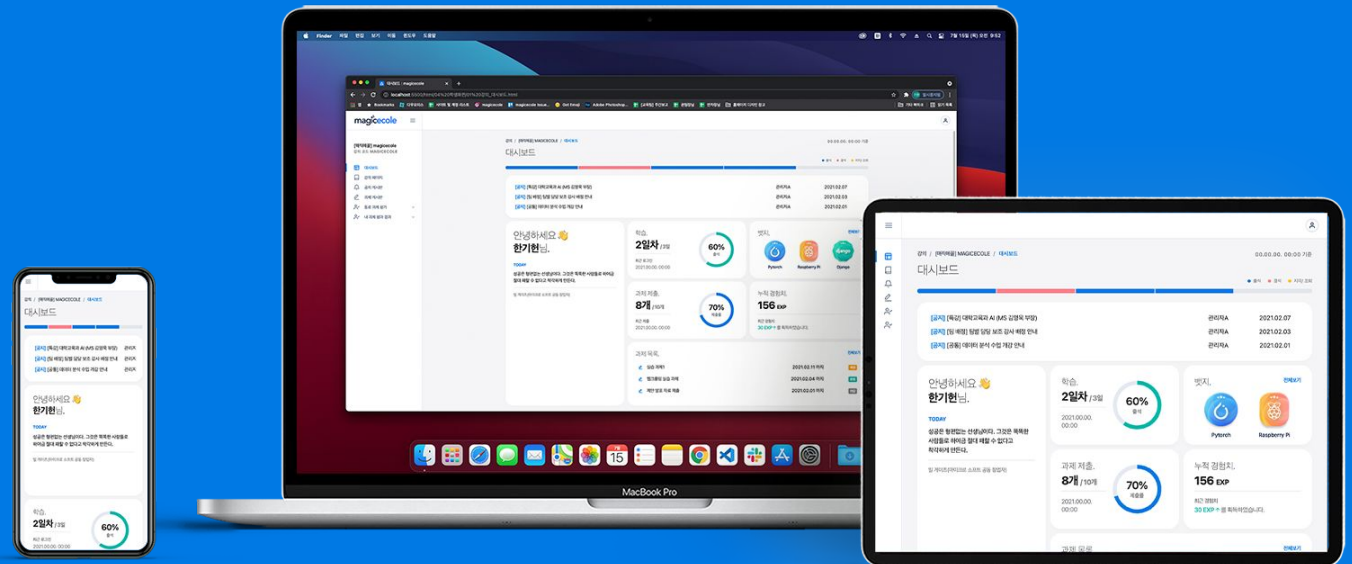


DevOps - Linux

# 프로세스와 서비스



# 다룰 내용

- 프로세스의 의미
- 리눅스의 프로세스
- 리눅스의 서비스



# 프로세스

# 프로세스

어떤 일의 **프로세스**?

절차? 흐름?



process<sup>1</sup> 미국식 [ˈpraɪsɪz; ˈpraʊsɪz]  영국식 [ˈprəʊsɪz] 

1. [명사] (특정 결과를 달성하기 위한) **과정[절차]** (→peace **process**)
2. [명사] (자연스러운 변화가 일어나는) 과정
3. [동사] (원자재식품 등을) 가공[처리]하다
4. [동사] (문서요청 사항 등을 공식적으로) **처리하다**

순서에 따라 **일** 하는 것



보편적으로 프로세스를 일의 과정이라고 할 때, 이것 컴퓨터 운영체제에서의 "프로세스"라는 단어와 어떻게 관련지을 수 있을까?



프로세스는 컴퓨터 운영체제에서 실행 중인 프로그램을 의미합니다. 컴퓨터에서 프로그램을 실행하면, 해당 프로그램은 운영체제로부터 할당 받은 자원(메모리, CPU 시간, 입출력 등)을 사용하여 작업을 수행합니다. 이때 프로그램은 운영체제에서 프로세스라고 불리는 실행 중인 작업으로 취급됩니다.

프로세스는 일의 과정과 유사한 면이 있습니다. 일의 과정에서는 시작과 끝이 존재하고, 여러 단계를 거쳐 작업이 수행되는 것처럼, 프로세스도 시작과 끝이 있으며, 프로세스가 실행되는 동안 여러 단계를 거쳐 작업이 수행됩니다.

또한, 일의 과정에서는 필요한 자원(도구, 인력 등)을 할당받아 작업을 수행하고, 작업이 완료되면 자원을 반환하는 것처럼, 프로세스도 운영체제로부터 필요한 자원을 할당받아 작업을 수행하고, 작업이 끝나면 할당받은 자원을 반환합니다.

따라서, 프로세스는 컴퓨터 운영체제에서 실행 중인 일의 과정으로 볼 수 있습니다.

# 리눅스의 프로세스





리눅스의 프로세스의 개념에 대해 알려줘



리눅스에서 프로세스란, 실행 중인 **프로그램의 인스턴스**입니다. 간단하게 말하면, 운영 체제가 실행 가능한 프로그램을 실행하여 프로세스를 만듭니다. 이 프로세스는 운영 체제에 의해 추적되며, 프로그램이 실행 중일 때 사용되는 **시스템 리소스(메모리, CPU 시간 등)**를 할당받습니다.

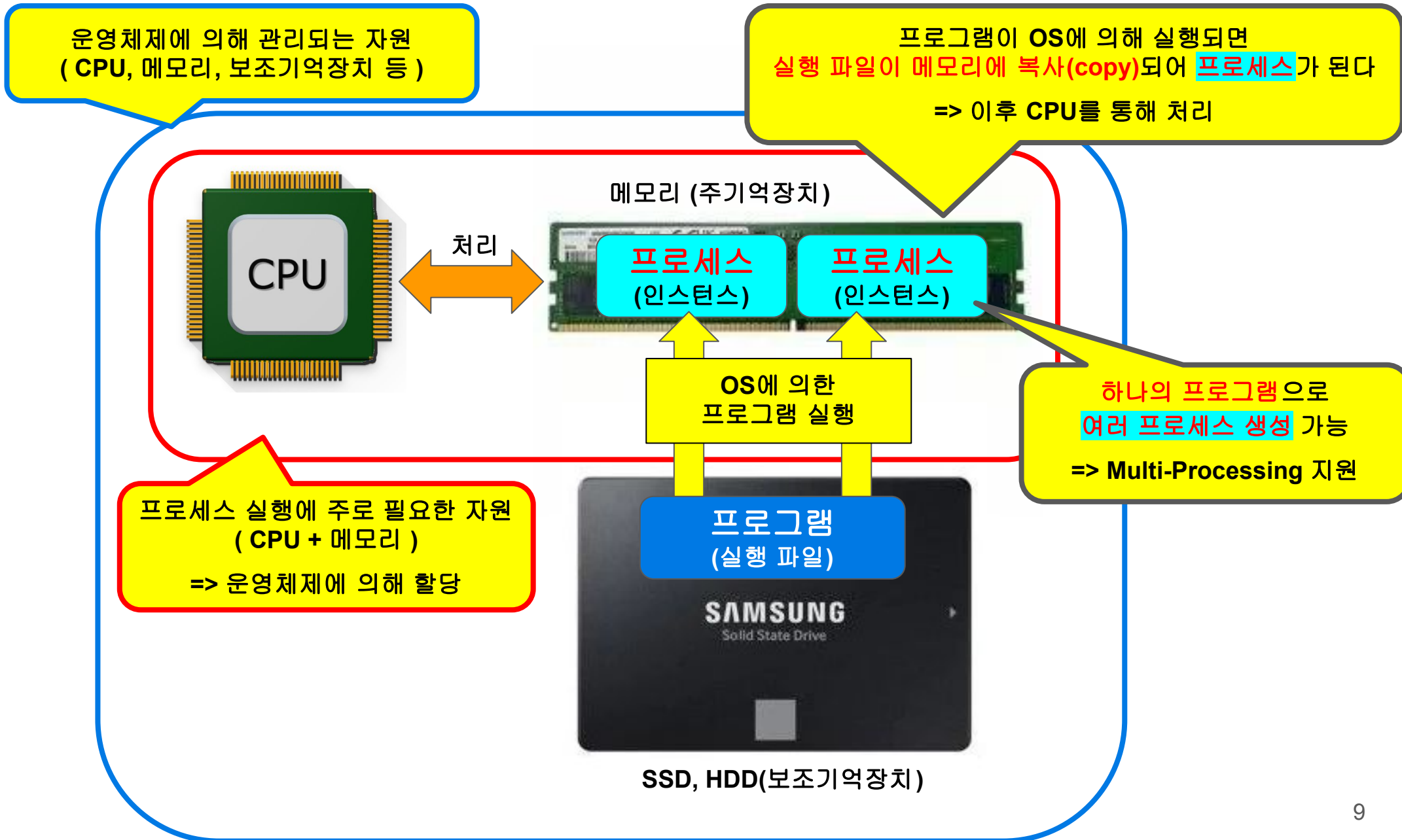
프로세스는 일반적으로 **부모-자식 계층 구조를 형성**합니다. 부모 프로세스는 자식 프로세스를 생성하고, 자식 프로세스는 다시 자식 프로세스를 생성할 수 있습니다. 이러한 구조는 일반적으로 운영 체제의 프로세스 관리를 용이하게 합니다.

리눅스에서 프로세스는 **고유한 ID 번호를 부여받습니다**. 이 ID를 사용하여 프로세스를 식별하고 추적할 수 있습니다. 또한, 각 프로세스는 독립적으로 실행되며, 다른 프로세스에 영향을 미치지 않습니다. 그러나 프로세스는 시스템 리소스를 공유할 수 있습니다.

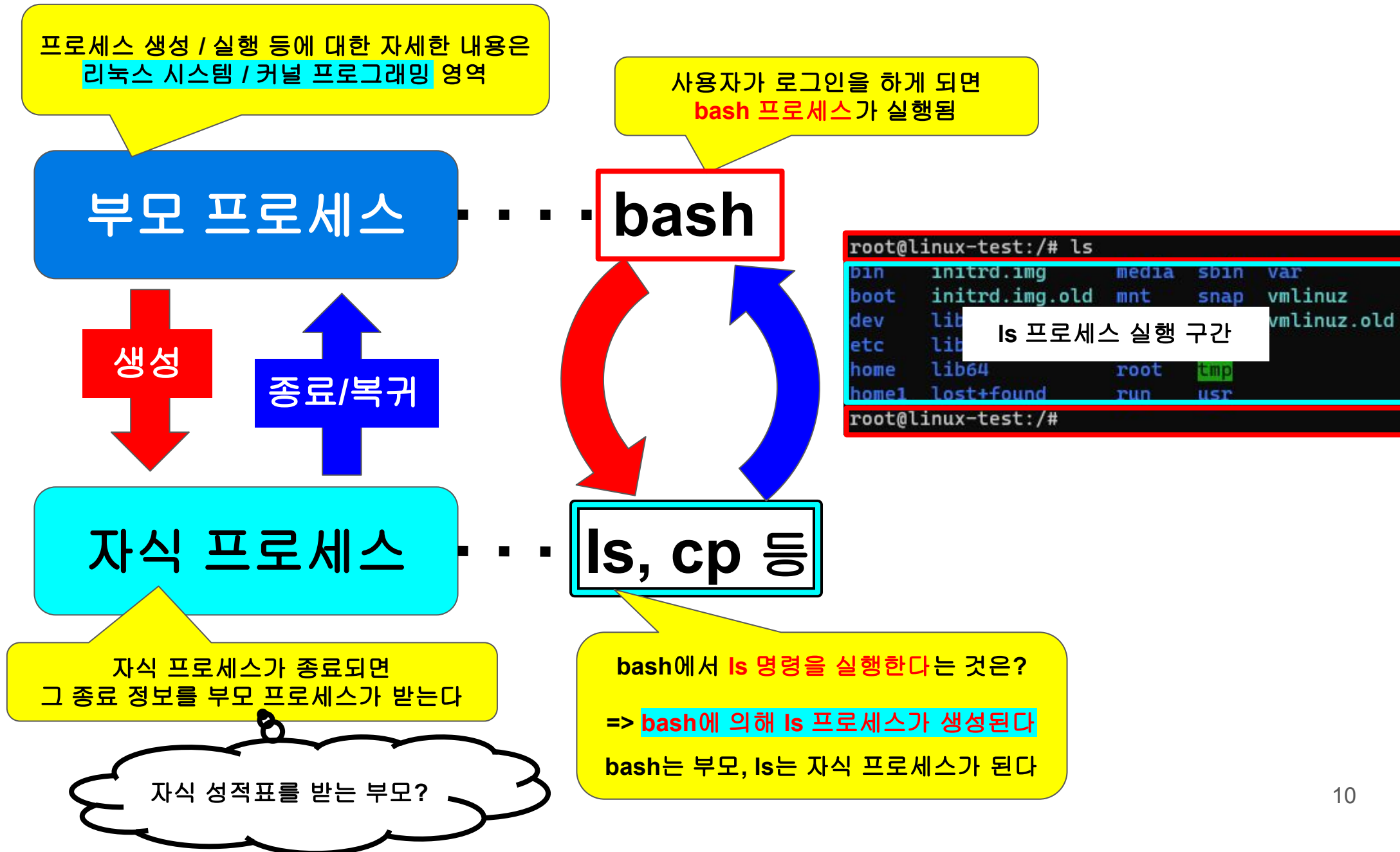
리눅스에서는 **다양한 도구를 사용하여 프로세스를 모니터링하고 관리**할 수 있습니다. 이러한 도구 중 일부는 "ps"와 "top"과 같은 명령 줄 도구이며, 다른 도구는 그래픽 사용자 인터페이스(GUI)를 제공하는 프로그램입니다.



# 프로그램과 프로세스



# 프로세스의 부모 / 자식 관계



# 프로세스 계층 구조

## init 프로세스(systemd)

=> 시스템 부팅 시 가장 처음 시작  
=> 시스템에서 동작하는 모든 프로세스의 부모 프로세스

init 프로세스를 시조(?)로 하여  
시스템 상의 프로세스 생성

```

systemd(1)-accounts-daemon(542)
            -agetty(851)
            -agetty(852)
            -atd(545)
            -chronyd(888)
            -cron(539)
            -dbus-daemon(528)
            -dhclient(724)
            -lvmetad(415)
            -lxcfs(540)
            -networkd-dispat(547)
            -noms_nsight(902)——noms_nsight(903)
            -polkitd(560)
            -rsyslogd(546)
            -sshd(867)——sshd(1599)——bash(1723)——pstree(3057)
            -systemd(1604)——(sd-pam)(1605)
            -systemd(868)——(sd-pam)(870)
            -systemd-journal(409)
            -systemd-logind(544)
            -systemd-network(459)
            -systemd-resolve(698)
            -systemd-udev(423)
            -unattended-upgr(814)
            -xe-daemon(1009)——logger(1021)
                               |
                               -logger(1023)
    
```

모든 프로세스는 ID 값을 갖는다  
(PID)

=> init 프로세스의 PID는 1

# 프로세스의 종류

프로세스가 실행되는 방법, 실행되는 영역, 역할, 상태에 따라 여러가지로 분류됨

주요 학습 범위

사용자(User) 프로세스

- bash와 같은 **셸** 프로세스
- **사용자가 직접 실행**하는 프로그램이나 명령어

데몬(Daemon) 프로세스

- **사용자(사람)와의 상호작용 없음 (모니터, 키보드 X)**
- 주로 시스템 부팅 시 **init(systemd)** 프로세스에 의해 실행
- 서버로서의 역할을 수행하는 서비스(웹 서버, 데이터베이스 등)

고아(Orphan) 프로세스

- **부모 프로세스가 먼저 종료**되고 남게 된 프로세스
- **init** 프로세스가 새로운 부모가 되어 종료할 수 있게 함

좀비(Zombie) 프로세스

- 종료되었지만 프로세스의 자원을 계속 차지하고 있는 경우
- 부모 프로세스의 종료 정보에 대한 처리가 잘 되지 않을 때 발생

시스템(System) 프로세스

- 시스템 초기화, 메모리 관리, 인터럽트 처리 등의 OS 핵심 기능 수행
- **init(systemd)** 등

커널 스레드 (Kernel Thread)

- OS 커널 레벨에서 관리

동작 중인 리눅스 시스템에서

1. 필요에 따라 프로세스를 생성하고 제거하는 것
2. 어떤 프로세스가 어떤 상태인지 아는 것
3. 실행 중인 프로세스에 대한 자원을 조정하는 것

왜 할까?

=> 한정된 시스템의 자원을 **효율적으로** 사용하기 위해서

# 프로세스 관리 명령

## ● ps

- **기능** : 현재 실행 중인 프로세스에 대한 정보 출력
- **형식** : **ps [옵션]**

1. 현재 쉘에서 실행 중인 프로세스 출력 : **ps**

```
root@linux-test:~# ps
  PID TTY          TIME CMD
 1673 pts/0    00:00:00 bash
 2661 pts/0    00:00:00 ps
```

System V라는 유닉스 계열의 운영체제에서 사용하는 옵션 형식 ('-' 문자로 시작)

2. 현재 실행 중인 **모든 프로세스에 대한 정보를 자세히** 출력 : **ps -ef**

```
root@linux-test:~# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  09:32 ?        00:00:01 /sbin/init
root           2        0  0  09:32 ?        00:00:00 [kthreadd]
root           4        2  0  09:32 ?        00:00:00 [kworker/0:0H]
root           6        2  0  09:32 ?        00:00:00 [mm_percpu_wq]
root           7        2  0  09:32 ?        00:00:00 [ksoftirqd/0]

(중략)
root          535        1  0  09:32 ?        00:00:00 /usr/sbin/cron -f
daemon        537        1  0  09:32 ?        00:00:00 /usr/sbin/atd -f
root          542        1  0  09:32 ?        00:00:00 /usr/lib/accountsservice/a
syslog        543        1  0  09:32 ?        00:00:00 /usr/sbin/rsyslogd -n
message+      544        1  0  09:32 ?        00:00:00 /usr/bin/dbus-daemon --sys
```

(하략)

UID	프로세스 소유자 (유효 사용자)
PID	프로세스 ID 번호
PPID	부모 프로세스의 PID
C	CPU 사용률
STIME	프로세스가 시작된 시간/날짜
TTY	프로세스와 연결된 터미널
TIME	총 CPU 사용 시간
CMD	프로세스의 실행 명령행

# 프로세스 관리 명령

## ● ps

BSD라는 유닉스 계열의 운영체제에서 사용하는 옵션 형식  
( ' - ' 문자로 시작하지 않음 )

3. 현재 실행중인 **모든 프로세스에 대한 정보를 자세히** 출력 : **ps aux**  
(CPU / 메모리 사용량, 프로세스 상태 추가)

```
root@linux-test:~# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.4	159500	8732	?	Ss	09:32	0:01	/sbin/init
root	2	0.0	0.0	0	0	?	S	09:32	0:00	[kthreadd]
root	4	0.0	0.0	0	0	?	I<	09:32	0:00	[smntd]
root	6	0.0	0.0	0	0	?	I<	09:32	0:00	[kworker/0:0]
root	7	0.0	0.0	0	0	?	S	09:32	0:00	[kworker/0:0]
root	8	0.0	0.0	0	0	?	I	09:32	0:00	[rcu_sched]
root	9	0.0	0.0	0	0	?	I	09:32	0:00	[rcu_bh]
root	10	0.0	0.0	0	0	?	S	09:32	0:00	[migration]
(중략)										
root	1556	0.0	0.3	76520	7416	?	Ss	09:37	0:00	/lib/systemd/systemd
root	1557	0.0	0.1	195352	2304	?	S	09:37	0:00	(sd-pam)
root	1673	0.0	0.2	22924	5556	pts/0	Ss	09:37	0:00	-bash
root	2620	0.0	0.0	0	0	?	I	11:20	0:00	[kworker/u128:1]
root	2669	0.0	0.0	0	0	?	I	11:27	0:00	[kworker/u128:0]
root	2872	0.0	0.0	0	0	?	I	11:35	0:00	[kworker/u128:2]
root	2874	0.0	0.0	0	0	?	I	11:35	0:00	[kworker/u128:3]
root	2875	0.0	0.3	109860	7044	?	Ss	11:35	0:00	sshd: root@
root	2964	0.0	0.2	22660	5008	pts/1	Ss+	11:35	0:00	-bash
root	3039	0.0	0.1	37376	3260	pts/0	R+	11:39	0:00	ps aux

%CPU	CPU 사용비율 추정치
%MEM	메모리 사용비율 추정치
VSZ	가상 메모리 사용량
RSS	물리 메모리 사용량
STAT	현재 프로세스의 상태코드

지금은 대충 이렇게  
있구나 하고 봅시다



# 프로세스 관리 명령

## ● pstree

- **기능** : 프로세스의 상태를 트리(Tree) 구조로 출력
- **형식** : **pstree** [옵션]

```
root@linux-test:~# pstree
systemd--accounts-daemon--2*[{accounts-daemon}]
--2*[agetty]
--atd
--chronyd
--cron
--dbus-daemon
--dhclient
--lvmetad
--lxcfs--2*[{lxcfs}]
--networkd-dispat--{networkd-dispat}
--noms_nsight--noms_nsight--8*[{noms_nsight}]
--polkitd--2*[{polkitd}]
--rsyslogd--3*[{rsyslogd}]
--sshd--sshd--bash
--sshd--sshd--bash--pstree
--2*[systemd--(sd-pam)]
--systemd-journal
--systemd-logind
--systemd-network
--systemd-resolve
--systemd-udev
--unattended-upgr--{unattended-upgr}
--xe-daemon--2*[logger]
--xe-daemon--6*[{xe-daemon}]
```

옵션	설명
-a	각 프로세스의 명령행 인자까지 출력
-h	현재 프로세스와 그 조상 프로세스를 하이라이트로 강조
-n	프로세스 이름 대신 PID값으로 정렬
-p	PID 값을 같이 출력

부모  자식

# 프로세스 관리 명령

## ● pgrep

- **기능** : 지정한 패턴과 일치하는 프로세스의 **PID**값을 출력
- **형식** : **pgrep** [옵션] [패턴]

ps -ef와 pgrep 비교 : sshd 정보 출력

```
root@linux-test:~# ps -ef | grep sshd
root      864      1  0 09:21 ?        00:00:00 /usr/sbin/sshd -D
root     1569     864  0 09:30 ?        00:00:00 sshd: root@pts/0
root     1780     864  0 09:36 ?        00:00:00 sshd: root@pts/1
root     2050    1863  0 10:02 pts/1    00:00:00 grep --color=auto sshd

root@linux-test:~#
root@linux-test:~# pgrep sshd
864
1569
1780
root@linux-test:~#
```

pgrep은 PID 정보만 출력  
=> 별도의 파싱이 필요한 경우 ps -ef 보다  
유리

# 프로세스 관리 명령

## • top

- **기능** : 동작 중인 프로세스들의 상태를 실시간으로 화면에 출력
- **형식** : top [옵션]

```
top - 10:26:12 up 1:04, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 92 total, 1 running, 55 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.3 st
KiB Mem : 2021448 total, 1261432 free, 129044 used, 630972 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 1741068 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	225024	8688	6612	S	0.0	0.4	0:01.54	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
7	root	20	0	0	0	0	S	0.0	0.0	0:00.06	ksoftirqd/0
8	root	20	0	0	0	0	I	0.0	0.0	0:00.05	rcu_sched
9	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.03	watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
14	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_kthre
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kauditd
17	root	20	0	0	0	0	S	0.0	0.0	0:00.01	xenbus
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	xenwatch
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	xenwatch
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	xenwatch
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	xenwatch
22	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	xenwatch
23	root	20	0	0	0	0	S	0.0	0.0	0:00.00	xenwatch
24	root	25	5	0	0	0	S	0.0	0.0	0:00.00	xenwatch
25	root	39	19	0	0	0	S	0.0	0.0	0:00.00	xenwatch

[ Up ], [ Down ] 키 : 위,아래 스크롤  
[ Space ] : 화면 갱신  
q : 종료

옵션	설명
-d	갱신 시간 설정(초)
-p	특정 PID 프로세스 모니터링
-n	top 명령 실행 횟수 지정

PR	우선순위
NI	Nice Vaule (-20 ~19) 작을수록 우선순위 높아짐
VIRT	가상 메모리 사용량
RES	물리적 메모리 사용량
SHR	공유 메모리 사용량
S	현재 프로세스 상태

# 프로세스 관리 명령

## ● kill

OS 커널에서 프로세스에게 전달하는 알림 메시지  
숫자값으로 정의됨

- **기능** : 지정한 **시그널**을 프로세스에게 보냄
- **형식** : **kill** **[옵션]** **[-시그널 번호]** **[PID]**

시그널 목록 출력 : **kill -l**

```
root@linux-test:~# kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL	5) SIGTRAP
6) SIGABRT	7) SIGBUS	8) SIGFPE	9) SIGKILL	10) SIGUSR1
11) SIGSEGV	12) SIGUSR2	13) SIGPIPE	14) SIGALRM	15) SIGTERM
16) SIGSTKFLT	17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGVIRT	28) SIGWINCH	29) SIGPOLL	30) SIGIO
31) SIGSYS	34) SIGRTMIN	35) SIGRTMAX-1		
38) SIGRTMIN+4	39) SIGRTMIN+5	40) SIGRTMIN+10		
43) SIGRTMIN+9	44) SIGRTMIN+10	45) SIGRTMIN+15		
48) SIGRTMIN+14	49) SIGRTMIN+15	50) SIGRTMAX-11		
53) SIGRTMAX-11	54) SIGRTMAX-10	55) SIGRTMAX-6		
58) SIGRTMAX-6	59) SIGRTMAX-5	60) SIGRTMAX		
63) SIGRTMAX-1	64) SIGRTMAX			

시그널을 받은 프로세스는  
기본 지정된 동작이나  
별도로 정의된 동작을 실행

시그널	번호	기본처리	의미
SIGHUP	1	종료	사용자 로그아웃, 터미널 연결 종료 시 발생 프로세스 재시작 요청으로도 사용
SIGINT	2	종료	키보드(Ctrl + c) 입력 시 발생 (프로세스 종료)
SIGQUIT	3	종료 및 코어덤프	키보드(Ctrl + \) 입력 시 발생 (프로세스 종료) 디버깅 목적으로 종료 - [덤프(저장)된 코어 파일 이용]
SIGKILL	9	종료	강제 종료 (이 시그널을 무시할 수 없음 - 최후의 수단)
SIGSEGV	11	종료 및 코어 덤프	잘못된 메모리 영역 침범 시 발생 - [덤프(저장)된 코어 파일 이용하여 디버깅]
SIGTERM	15	종료	프로세스 종료 요청 (kill 명령이 보내는 기본 시그널)

**kill 724**

: PID 724 프로세스에 SIGTERM 시그널 전송

**kill -9 756 757 758**

: 756, 757, 758 프로세스 강제 종료

# 프로세스 관리 명령

## ● killall

- **기능** : 프로세스명으로 지정된 모든 프로세스에 시그널을 전송한다
- **형식** : **killall** [옵션] [-시그널 번호] [프로세스명]

### 주요 옵션

옵션	설명
-l	시그널의 종류 출력
-w	시그널을 받은 프로세스들이 종료될 때까지 대기
-v	시그널 전송 결과를 출력
-u [사용자]	특정 사용자의 프로세스를 지정
-s [시그널명]	시그널의 이름을 지정

### 사용 예

**killall httpd**

: 실행 중인 모든 httpd 프로세스를 종료한다

**kill -v -9 httpd**

: 실행 중인 모든 httpd 프로세스에 SIGKILL(9) 시그널을 보낸다  
(강제 종료한다)

**kill -9 -u user1**

: user1 사용자의 모든 프로세스를 강제 종료한다

시그널 번호가 명시되지 않을 경우  
**SIGTERM(15)** 시그널을 전송한다

# 프로세스 관리 명령

## • uptime

- **기능** : 시스템이 가동된 후 얼마나 오래 되었는지 표시
- **형식** : uptime

```
root@linux-test:~# uptime
11:53:28 up 2:10 1 user load average: 0.00, 0.00, 0.00
```

현재 시각

시스템 활성화 후 누적 시간  
(2시간 10분)

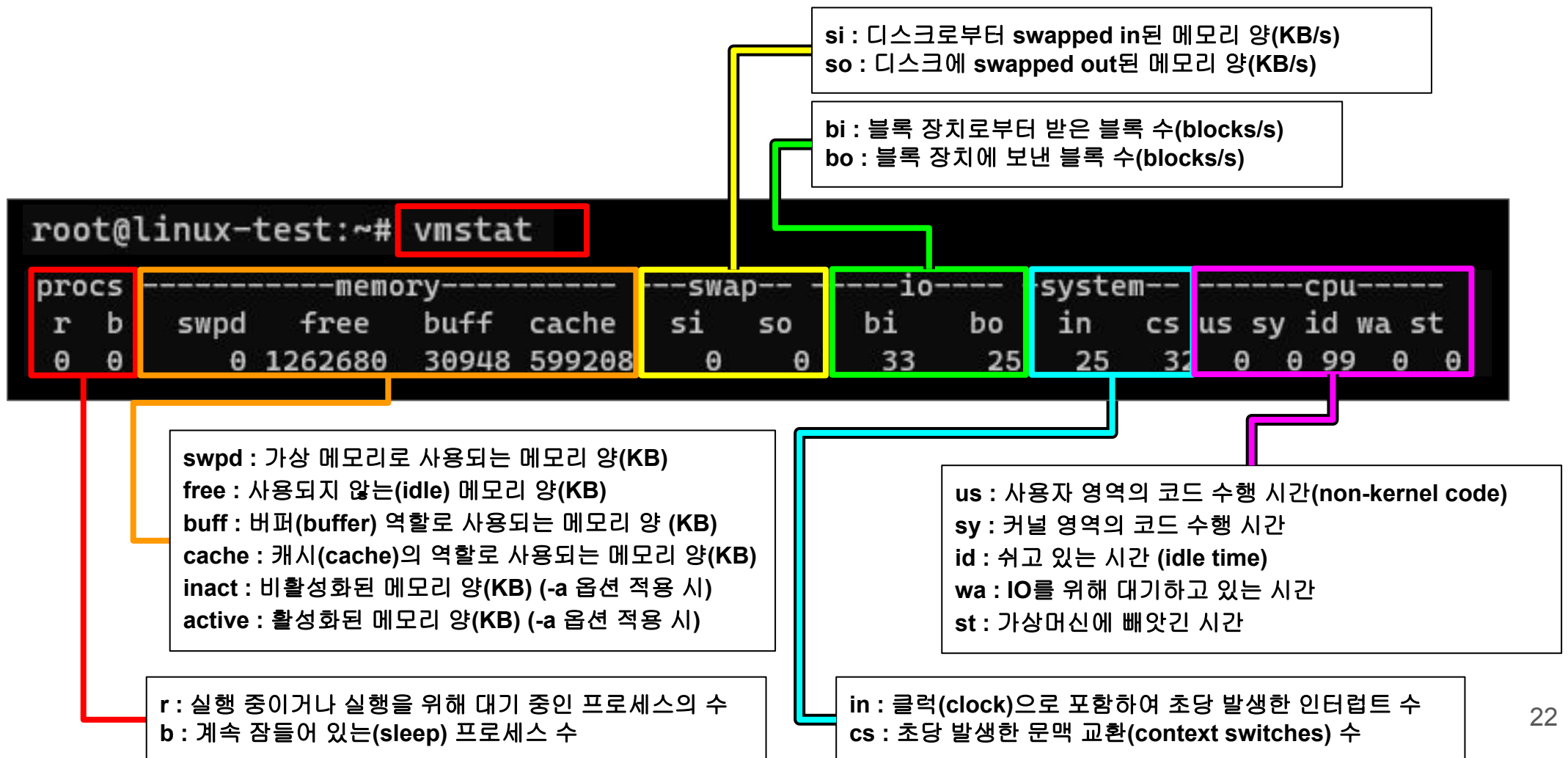
ssh 등으로 현재 접속 중인  
사용자 수

1분, 5분, 15분에 대한 시스템  
부하(load)의 평균(average)

‘부하(Load)’라는 건 뭘  
의미하는 걸까?

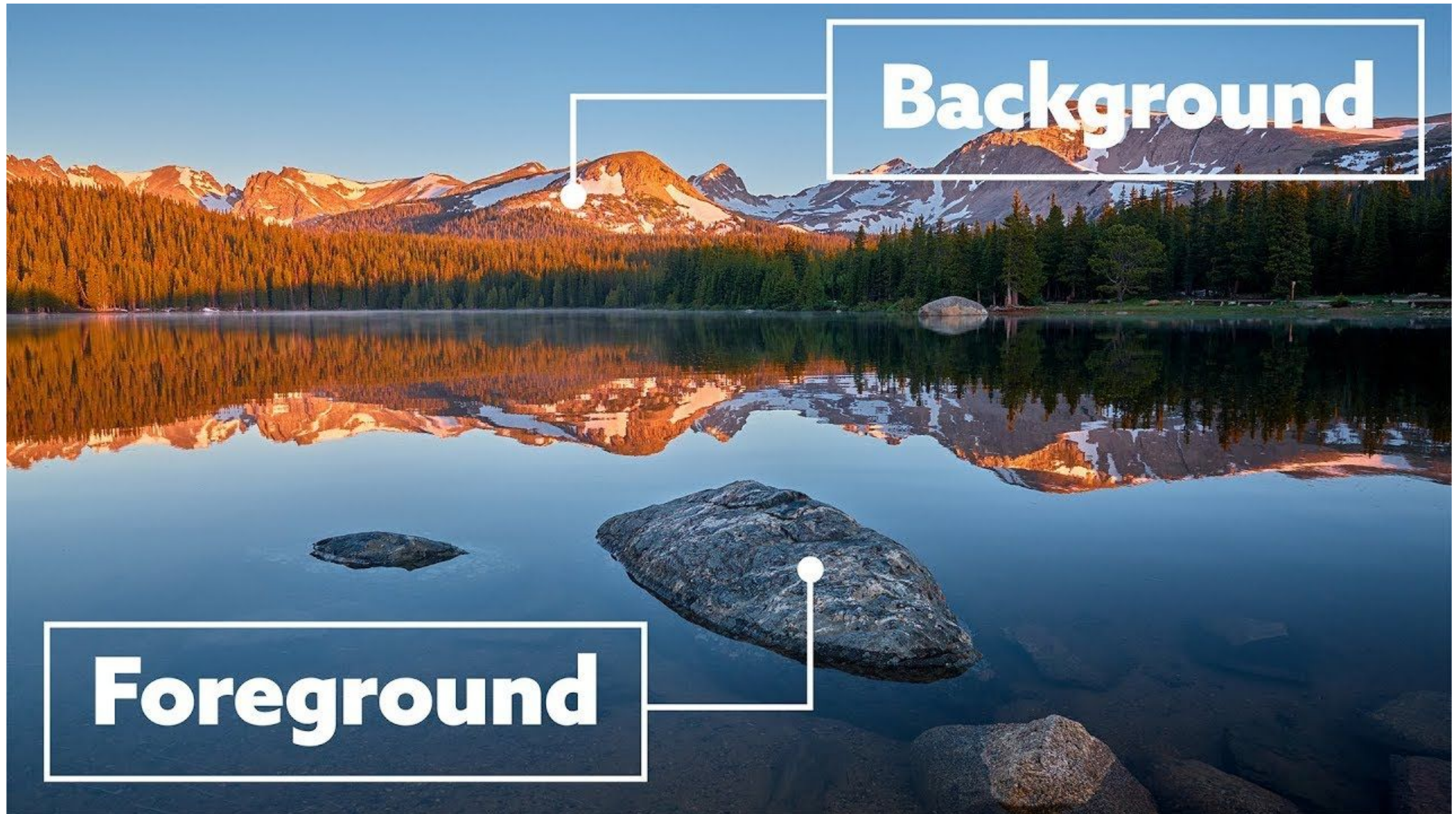
## ● vmstat

- **기능** : 프로세스, 메모리, 스왑, 블록 장치 IO, CPU 사용률 정보 등을 출력
- **형식** : **vmstat** [옵션]





# Foreground와 Background



# Foreground와 Background

## Foreground(전경) 프로세스

리눅스에서 일반적으로 사용자가 명령을 실행하는 방식

```
root@linux-test:/# ls
bin      initrd.img      media  sbin  var
boot     initrd.img.old  mnt     snap  vmlinuz
dev      lib             vmlinuz.old
etc      lib
home     lib64          root   tmp
home1    lost+found     run    usr
root@linux-test:/#
```

ls 프로세스 실행 구간

ls가 실행이 끝날 때까지 bash는 대기

ls가 **Foreground(전경)** 프로세스로 실행됨

지금 실행되는 프로세스가  
앞으로 부각됨

bash 프로세스

ls 프로세스

bash 프로세스  
대기 구간

# Foreground와 Background

## Background(배경) 프로세스

```
root@linux-test:~# find / -name "*.sh" > list_sh &
[1] 4831
root@linux-test:~# who am i
root      pts/0      2023-03-27 10:25 (221.146.63.213)
[1]+  Done                  find / -name "*.sh" > list_sh
root@linux-test:~#
```

Background로 프로세스 실행  
=> 명령행 끝에 '**&**' 추가

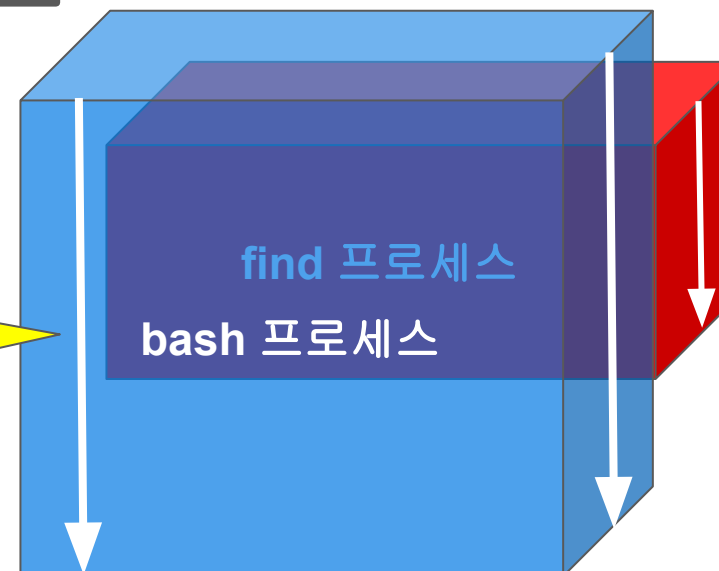
find 프로세스 시작

find 프로세스가 실행되는 중에도  
셸 입력이 동시에 가능

find 프로세스 종료

find 프로세스를 **Background(배경)**  
프로세스로 실행

**bash**는 대기없이 계속 실행  
=> 여러 프로세스를 동시 실행 가능



# Background 프로세스 확인

## • jobs

- **기능** : Background 프로세스(작업)을 조회
- **형식** : jobs [%작업 번호]

Background 프로세스 2개 실행

sleep 프로세스의  
PID는 5478, 작업 번호는 1

find 프로세스의  
PID는 5479, 작업 번호는 2

전체 background 작업 조회

1번 background 작업 조회

```

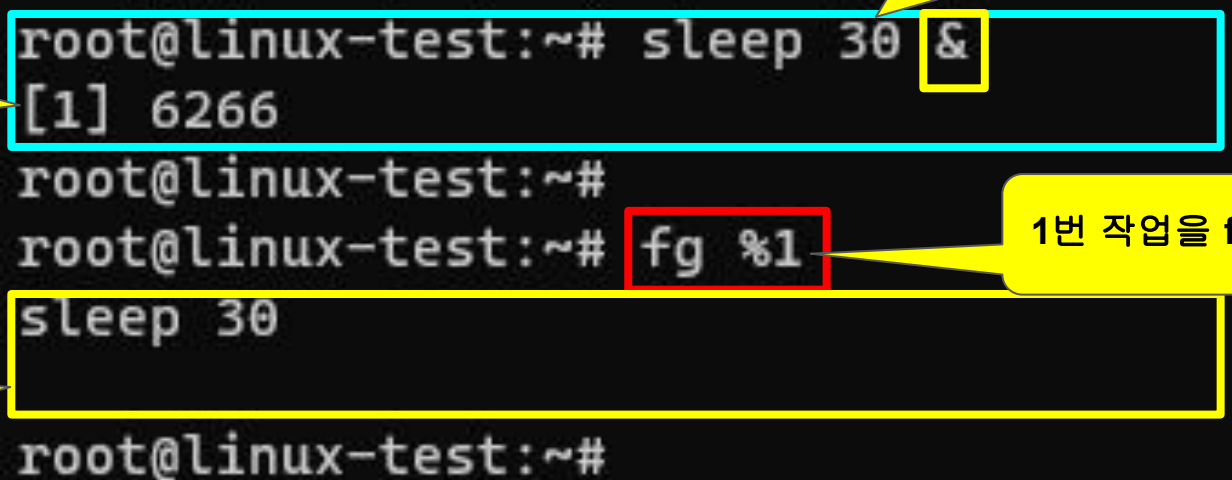
root@linux-test:~# sleep 100 &
[1] 5478
root@linux-test:~# find / -name "*.sh" > list_sh &
[2] 5479
root@linux-test:~# jobs
[1]-  Running (실행중)      sleep 100 &
[2]+  Done (완료)         find / -name "*.sh" > list_sh
root@linux-test:~#
root@linux-test:~# jobs %1
[1]+  Running              sleep 100 &
root@linux-test:~#
  
```



# 프로세스 동작 전환

## • fg

- **기능** : Background 작업을 Foreground로 전환
- **형식** : fg [%작업 번호]



The terminal screenshot shows the following sequence of commands and outputs:

```

root@linux-test:~# sleep 30 &
[1] 6266
root@linux-test:~#
root@linux-test:~# fg %1
sleep 30
root@linux-test:~#
  
```

**Callouts:**

- Background 작업 실행 => 30초 동안 sleep**: Points to the `sleep 30 &` command.
- sleep 프로세스의 PID는 6266, 작업 번호는 1**: Points to the output `[1] 6266`.
- 1번 작업을 foreground로 전환**: Points to the `fg %1` command.
- bash는 sleep이 종료할 때까지 대기**: Points to the `sleep 30` output line.

# 프로세스 동작 전환

## • bg

- **기능** : Foreground 작업을 Background로 전환
- **형식** : **bg [%작업 번호]**

키보드 **Ctrl + z** 입력  
=> Foreground 작업 잠시 중단

sleep 30을 foreground로 실행  
=> bash는 대기

```
root@linux-test:~# sleep 30
^Z
[1]+  Stopped (멈춤)                  sleep 30
root@linux-test:~# bg %1
[1]+  sleep 30 &
root@linux-test:~#
```

1번 작업을 background로 재개

# 프로세스 종료

## Foreground 프로세스

Foreground로 작업 실행

키보드 Ctrl + c로 종료  
=> sleep에 SIGINT(2) 시그널 전달

```
root@linux-test:~# sleep 100
^C
root@linux-test:~#
```

## Background 프로세스

Background로 작업 실행

kill 명령으로 종료 시그널 전달  
=> SIGINT(2), SIGKILL(9), SIGTERM(15)

```
root@linux-test:~# sleep 100 &
[1] 7632
root@linux-test:~#
root@linux-test:~# kill -9 7632
root@linux-test:~#
[1]+  Killed                  sleep 100
root@linux-test:~#
```



# 로그아웃 이후 Background 작업 계속 실행

## • nohup (NO Hang UP)

- **기능** : 로그아웃한 후에도 background 작업 계속 실행
- **형식** : nohup [명령] &

터미널 세션 종료 시, 해당 세션에서 실행된 background 작업이 같이 종료되는 것을 막음

nohup으로 background 작업 실행

Background 작업의 PID 확인

```
root@linux-test:~# nohup sleep 100 &
[1] 6539
root@linux-test:~# nohup: ignoring input and appending output to 'nohup.out'
```

출력을 nohup.out에 기록

```
root@linux-test:~# ps -ef | grep sleep
root      6539      4750  0 14:38 pts/0    00:00:00 sleep 100
root      6541      4750  0 14:38 pts/0    00:00:00 grep --color=auto
```

리눅스 터미널 로그아웃

```
root@linux-test:~# exit
logout
```

원도우에서 ssh 재접속

```
Goodbye root. Have a nice day!
Connection to 106.10.32.76 closed.
PS C:\Users\magicnotebook>
PS C:\Users\magicnotebook> ssh root@106.10.32.76 -p 1024
root@106.10.32.76's password:
root@106.10.32.76:~# cat /etc/os-release
PRETTY_NAME="Ubuntu 18.04.6 LTS"
NAME="Ubuntu"
VERSION="18.04.6 LTS (GNU/Linux 4.15.0-118-generic x86_64)"
```

sleep 100 여전히 실행 확인  
그런데 부모 프로세스의 PID가 1  
(고아 프로세스로 동작)

(중략)

```
root@linux-test:~# ps -ef | grep sleep
root      6539      1      0 14:38 ?        00:00:00 sleep 100
root      6667      6655  0 14:39 pts/0    00:00:00 grep --color=auto sleep
root@linux-test:~#
```

# 작업 예약

## • at

- **기능** : 정해진 시간에 명령을 **한번** 실행하도록 예약
- **형식** : **at** **[옵션]** **[시간]**

1. 18:00에 실행할 명령 예약 : **at 18:00**

```
root@linux-test:~# at 18:00
warning: commands will be executed using /bin/sh
at> echo "Time to go home"
at> echo "See You Tomorrow"
at> <EOT>
job 1 at Mon Mar 27 18:00:00 2023
root@linux-test:~#
```

Ctrl + d로 입력 완료

18시에 실행할 명령 예약

실행할 명령 입력

2. 현재 사용자의 예약된 작업 조회 : **at -l** 또는 **atq**

```
root@linux-test:~# at -l
1      Mon Mar 27 18:00:00 2023 a root
root@linux-test:~# atq
1      Mon Mar 27 18:00:00 2023 a root
```

3. 예약된 1번 작업 삭제 : **at -d 1** 또는 **atrm 1**

```
root@linux-test:~# at -d 1
root@linux-test:~#
root@linux-test:~# at -l
root@linux-test:~#
```

# 작업 예약

## ● crontab

- **기능** : 사용자의 crontab 파일을 관리한다  
(정해진 시간에 명령을 **반복하여** 실행하도록 예약)
- **형식** : crontab **[옵션]** **[파일명]**

**/var/spool/cron/crontabs** 디렉토리에 계정  
이름의 파일로 저장

1. crontab 파일의 목록(예약 작업) 확인 : **crontab -l**

```
root@linux-test:~# crontab -l
MAILTO=""
30 06 * * * /usr/sbin/nsight_updater > /tmp/.nu.log 2> /tmp/.nu_err.log
```

실행 명령

요일(0~7), \* : 모든 값

월(1~12), \* : 모든 값

일(1~30), \* : 모든 값

시 (0~23)

분 (0~59)

root 계정에 대해 예약된 작업

=> **매일 06:30에 명령 실행** (6시간 30분 간격이 아님에 주의)

6시간 30분 간격으로 실행하려면  
어떻게 할까?

## ● crontab

2. 매 3분마다 uptime 명령을 실행하고 /root/uptime.log에 출력 저장

```
root@linux-test:~# export EDITOR=nano
root@linux-test:~# crontab -e
```

기본 텍스트 에디터를 nano로 지정  
=> Default는 vi

현재 사용자(root)의 crontab 파일 편집

```
GNU nano 2.9.3 /tmp/crontab.31VBKc/crontab

MAILTO=""
30 06 * * * /usr/sbin/nsight_updater > /tmp/.nu.log 2> /tmp/.nu_err.log
*/3 * * * * /usr/bin/uptime >> /root/uptime.log

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell
```

**\*/3** (3분 마다 실행)

실습 후 다시 지우세요 꼭!!!!  
로그가 쌓입니다!

```
root@linux-test:~# cat uptime.log
17:24:01 up 7:58, 2 users, load average: 0.00, 0.00, 0.00
17:27:01 up 8:01, 2 users, load average: 0.05, 0.01, 0.00
17:30:01 up 8:04, 2 users, load average: 0.00, 0.00, 0.00
17:33:01 up 8:07, 2 users, load average: 0.00, 0.00, 0.00
17:36:01 up 8:10, 2 users, load average: 0.00, 0.00, 0.00
17:39:01 up 8:13, 2 users, load average: 0.00, 0.00, 0.00
root@linux-test:~#
```

uptime 실행 로그 저장 확인

# 리눅스의 서비스

# 리눅스의 서비스(service)

동작 형태에 따른 프로세스 구분



Background 프로세스의 범주에 속하면서  
사용자와 상호작용이 없는 등의 특징을 갖는 프로세스  
=> 부팅하고 자동으로 동작하는 프로세스

주요 running 서비스

```
accounts-daemon.service
atd.service
chrony.service
cron.service
dbus.service
getty@tty1.service
lvm2-lvmetad.service
lxcfs.service
networkd-dispatcher.service
polkit.service
rc-local.service
rsyslog.service
serial-getty@hvc0.service
ssh.service
systemd-journald.service
systemd-logind.service
systemd-networkd.service
systemd-resolved.service
systemd-udev.service
unattended-upgrades.service
user@0.service
user@1000.service
xe-linux-distribution.service
```

at로 예약된 동작 수행

crontab에 예약된 동작 수행

ssh 서버

# 서비스 실행 과정

리눅스 부팅 프로세스

이런 과정을 거치는구나~~  
정도로만 이해합시다

실행 레벨이 뭐지?

systemd가 수행

지정된 실행 레벨(runlevel)에 대한 서비스 실행

default.target

- accounts-daemon.service
- apport.service
- display-manager.service
- grub-common.service
- nms\_nstght.service
- systemd-update-utmp-runlevel.service
- ureadahead.service
- xe-linux-distribution.service
- multi-user.target
  - apport.service
  - atd.service
  - chrony.service
  - console-setup.service
  - cron.service
  - dbus.service
  - ebttables.service
  - grub-common.service
  - irqbalance.service
  - lxcfs.service
  - lxd-containers.service
  - networkd-dispatcher.service
  - networking.service
  - nms\_nstght.service
  - ondemand.service

(하락)

전원 인가(Power-On)

POST(Power On Self Test) 수행

BIOS가 수행

Bootloader (GRUB) 이미지 로드

Bootloader (GRUB)가  
수행

리눅스 커널 이미지 로드

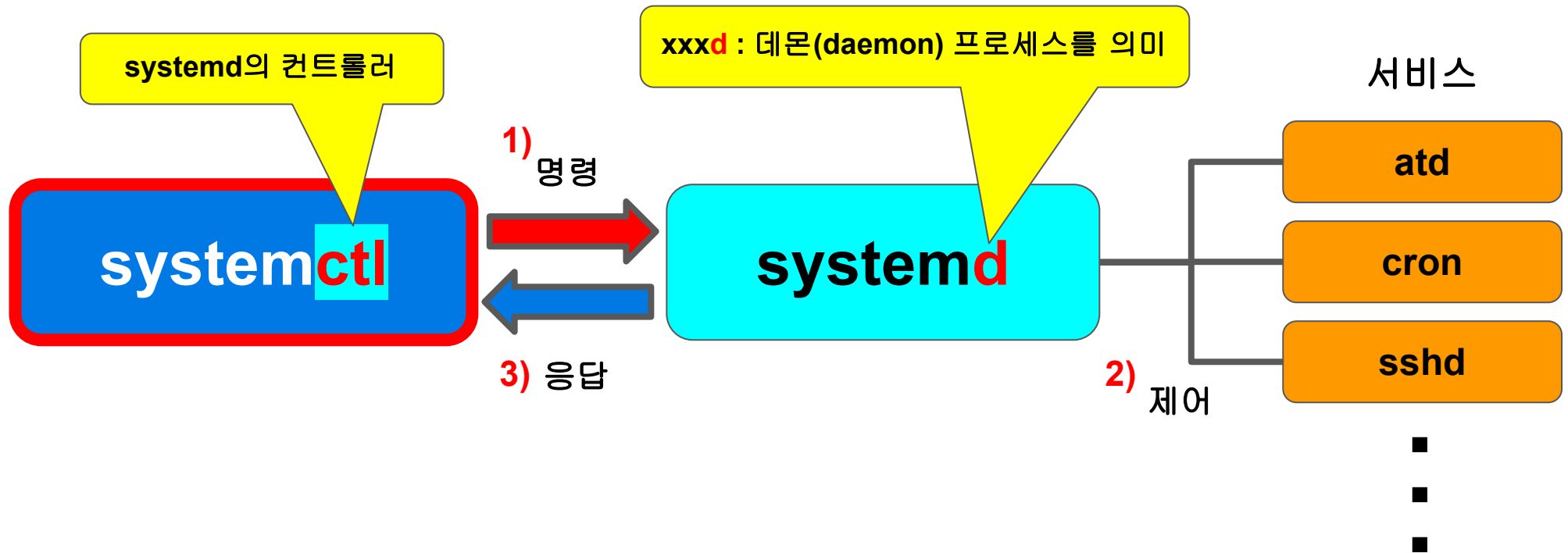
커널 초기화

리눅스 커널이 수행

init 프로세스 (systemd) 실행



# 서비스 제어



# 서비스 제어

## ● systemctl

- **기능** : systemd 기반의 시스템 및 서비스를 제어
- **형식** : **systemctl** [옵션] [명령] [유닛 이름]

### 주요 명령

명령	설명
<b>start</b>	유닛을 시작한다
<b>stop</b>	유닛을 정지한다
<b>reload</b>	유닛의 설정 파일을 다시 읽어온다
<b>daemon-reload</b>	유닛 설정 파일의 변경 사항 적용
<b>restart</b>	유닛을 재시작한다
<b>status</b>	유닛 상태를 출력한다
<b>enable</b>	부팅 시 유닛이 시작되도록 설정한다
<b>disable</b>	부팅 시 유닛이 시작하지 않도록 설정한다
<b>kill</b>	유닛에 시그널을 전송한다

### 유닛(unit) :

systemd가 시스템을 관리하는 단위 (구성요소)

service, target, mount, timer 등 여러가지 종류

### 주요 옵션

옵션	설명
<b>-t</b>	지정한 종류의 유닛만 출력
<b>--state</b>	유닛의 상태를 필터링하여 출력
<b>-a</b>	상태 관계없이 유닛 전체를 출력한다

# 서비스 제어

## • systemctl

1. 현재 실행 중(running) 상태인 서비스의 리스트 출력 : **systemctl -t service --state=running**

```

root@linux-test:~# systemctl -t service --state=running
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
accounts-daemon.service            loaded active running Accounts Service
atd.service                         loaded active running Deferred execution scheduler
chrony.service                     loaded active running chronyd
cron.service                       loaded active running Regular background program
dbus.service                       loaded active running D-Bus System
getty@tty1.service                 loaded active running Getty
lvm2-lvmetad.service               loaded active running LVM2 metadata daemon
lxcfs.service                      loaded active running FUSE filesystem for LXC
networkd-dispatcher.service         loaded active running Dispatcher daemon for systemd-networkd
polkit.service                     loaded active running Authorization Manager
rsyslog.service                    loaded active running System Log
serial-getty@hvc0.service           loaded active running Serial Getty
ssh.service                        loaded active running OpenSSH
systemd-journald.service             loaded active running Journal Service
systemd-logind.service              loaded active running Login Service
systemd-networkd.service            loaded active running Network Service
systemd-resolved.service            loaded active running Systemd-resolved
systemd-udev.service                loaded active running udev
unattended-upgrades.service          loaded active running Unattended Upgrades
user@0.service                     loaded active running User Service
user@1000.service                  loaded active running User Service
vmstat.service                     loaded active running Virtual Memory Statistics
xe-linux-distribution.service        loaded active running Linux Distribution

```

유닛의 상태를 LOAD, ACTIVE, SUB 3개의 범주로 나누어 구분

주요 유닛 상태 (service)	
loaded	유닛의 configuration file을 성공적으로 읽고 처리
active (exited)	일회성 구성을 성공적으로 실행, 실행 후 유닛은 활성 프로세스를 실행하거나 이벤트를 기다리지 않음
active (running)	일회성 구성을 성공적으로 실행, 실행 후 유닛은 하나 이상의 활성 프로세스를 실행중.
active (waiting)	일회성 구성을 성공적으로 실행, 실행 후 유닛은 이벤트를 대기중
inactive (dead)	일회성 구성이 실패했거나 아직 실행되지 않음

LOAD = Reflects whether the unit definition was properly loaded

ACTIVE = The high-level unit activation state, i.e. general state of the unit

SUB = The low-level unit activation state, values depend on the unit type

23 loaded units listed. Pass --all to see loaded but inactive units, too.

To show all installed unit files use 'systemctl list-unit-files'.

# 서비스 제어

## • systemctl

2. cron.service의 상태 출력 : **systemctl status cron.service**

```
root@linux-test:~# systemctl status cron.service
```

```
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-03-29 09:55:09 KST; 4h 18min ago
     Docs: man:cron(8)
  Main PID: 539 (cron)
    Tasks: 1 (limit: 2292)
   CGroup: /system.slice/cron.service
           └─539 /usr/sbin/cron -f

Mar 29 14:06:01 linux-test CRON[4370]: (CRON) info (No MTA installed, discarding output)
Mar 29 14:06:01 linux-test CRON[4370]: pam_unix(cron:session): session closed for user user1
Mar 29 14:09:01 linux-test CRON[4395]: pam_unix(cron:session): session opened for user user1 by (uid=0)
Mar 29 14:09:01 linux-test CRON[4396]: (user1) CMD (/usr/bin/uptime)
Mar 29 14:09:01 linux-test CRON[4395]: (CRON) info (No MTA installed, discarding output)
Mar 29 14:09:01 linux-test CRON[4395]: pam_unix(cron:session): session closed for user user1
Mar 29 14:12:01 linux-test CRON[4421]: pam_unix(cron:session): session opened for user user1 by (uid=0)
Mar 29 14:12:01 linux-test CRON[4422]: (user1) CMD (/usr/bin/uptime)
Mar 29 14:12:01 linux-test CRON[4421]: (CRON) info (No MTA installed, discarding output)
Mar 29 14:12:01 linux-test CRON[4421]: pam_unix(cron:session): session closed for user user1
root@linux-test:~#
```

서비스 유닛 상태

서비스의 실행 로그

# 서비스 제어

## • systemctl

3. cron.service를 시작/정지 : **systemctl start (stop) cron.service**

cron 서비스를 정지

서비스의 상태 확인  
=> inactive(dead)

```
root@linux-test:~# systemctl stop cron.service
root@linux-test:~#
root@linux-test:~# systemctl status cron.service | head -6
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Wed 2023-03-29 14:21:51 KST; 15s ago
     Docs: man:cron(8)
    Process: 4538 ExecStart=/usr/sbin/cron -f $EXTRA_OPTS (code=killed, signal=TERM)
   Main PID: 4538 (code=killed, signal=TERM)
```

cron 서비스를 시작

서비스의 상태 확인  
=> active(running)

```
root@linux-test:~# systemctl start cron.service
root@linux-test:~#
root@linux-test:~# systemctl status cron.service | head -6
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-03-29 14:22:20 KST; 3s ago
     Docs: man:cron(8)
    Main PID: 4598 (cron)
      Tasks: 1 (limit: 2292)
```



# 서비스 생성 및 등록

실습 : 3분마다 vmstat을 실행하고 로그를 기록하는 서비스를 생성하고 등록

1. 서비스에 대한 설정 파일 작성:

파일의 경로에 주의!

```
root@linux-test:~# nano /etc/systemd/system/vmstat.service
```

```
[Unit]
Description=Vmstat Service
```

**[Unit]**  
**Description** : 서비스의 이름이나 설명을 작성  
**After** : 이 서비스가 의존하는 다른 서비스가 먼저 실행되도록 지정

```
[Service]
ExecStart=/bin/bash /usr/local/bin/vmstat_script.sh
Restart=always
User=root
```

**[Service]**  
**ExecStart** : 서비스를 시작하는 명령어를 작성  
**Restart** : 서비스가 비정상적으로 종료될 경우 자동으로 재시작하도록 지정  
**User** : 서비스를 실행할 사용자를 지정  
**Group** : 서비스를 실행할 그룹을 지정  
**Environment** : 서비스 실행에 필요한 환경변수를 지정  
**WorkingDirectory** : 서비스가 실행될 디렉토리를 지정

```
[Install]
WantedBy=multi-user.target
```

**[Install]**  
**WantedBy**: 이 서비스가 어떤 타겟(시스템의 실행 레벨)에 의존하는지를 지정  
=> multi-user.target으로 지정하면 멀티 유저 모드에서 실행

# 서비스 생성 및 등록

실습 : 3분마다 vmstat을 실행하고 로그를 기록하는 서비스를 생성하고 등록

2. 서비스의 동작을 위한 쉘 스크립트 작성

파일의 경로에 주의!

```
root@linux-test:~# nano /usr/local/bin/vmstat_script.sh
```

```
GNU nano 2.9.3 /usr/local/bin/vmstat_script.sh

#!/bin/bash

while true
do
    echo "$(date +%Y-%m-%d %H:%M:%S)" >> /var/log/vmstat.log
    vmstat >> /var/log/vmstat.log

    echo "" >> /var/log/vmstat.log
    echo "" >> /var/log/vmstat.log

    sleep 180
done
```

이 코드의 의미를 이해하고  
설명할 수 있으면 Good!! ^^

파일의 경로에 주의!

^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify  
 ^X Exit        ^R Read File    ^\ Replace    ^U Uncut Text ^T To Linter



# 서비스 생성 및 등록

실습 : 3분마다 vmstat을 실행하고 로그를 기록하는 서비스를 생성하고 등록

## 3. 서비스 실행 및 로그 확인

```
root@linux-test:~# systemctl daemon-reload
root@linux-test:~# systemctl start vmstat.service
root@linux-test:~# systemctl status vmstat.service
● vmstat.service - Vmstat Service
   Loaded: loaded (/etc/systemd/system/vmstat.service; disabled; vendor preset: enabled)
   Active: active (running) since Thu 2023-03-30 10:14:37; 1min ago
     Main PID: 2192 (bash)
       Tasks: 2 (limit: 2292)
      CGroup: /system.slice/vmstat.service
              └─2192 /bin/bash /usr/local/bin/vmstat_script.sh
                 2203 sleep 180
```

**systemctl daemon-reload**

=> 서비스 유닛의 설정 파일 변경 사항 적용

**systemctl start vmstat.service**

=> vmstat.service 실행

**systemctl status vmstat.service**

=> vmstat.service 상태 확인

```
Mar 30 10:14:37 linux-test systemd[1]: Started Vmstat Service.
```

```
root@linux-test:~# cat /var/log/vmstat.log
```

저장된 로그 파일 확인

```
2023-03-29 10:27:21
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
2  0      0 1248716 19456 617268  0  0  251  211  42  70  1  0  97  0  1
```

```
2023-03-29 10:30:21
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
0  0      0 1248536 19608 617372  0  0  230  194  41  66  1  0  98  0  1
```

```
2023-03-29 10:33:21
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
0  0      0 1247744 19776 618120  0  0  212  179  39  63  1  0  98  0  1
```

# 서비스 생성 및 등록

실습 : 3분마다 vmstat을 실행하고 로그를 기록하는 서비스를 생성하고 등록

## 3. 부팅 시에 자동으로 실행되도록 서비스 등록

**systemctl enable vmstat.service**

=> vmstat.service를 부팅 시에 실행하도록 등록

먼저 작성한 vmstat.service 파일에 대한 **심볼릭 링크**를 생성  
=> 부팅 시에 systemd가 이 심볼릭 링크를 참조

```
root@linux-test:~# systemctl enable vmstat.service
Created symlink /etc/systemd/system/multi-user.target.wants/vmstat.service → /etc/systemd/system/vmstat.service.
root@linux-test:~#
root@linux-test:~# ls /etc/systemd/system/multi-user.target.wants/
atd.service          networkd-dispatcher.service  snapd.apparmor.service    systemd-networkd.service
chrony.service        networking.service          snapd.autoimport.service  systemd-resolved.service
console-setup.service ondemand.service           snapd.core-fixup.service  ufw.service
cron.service          open-vm-tools.service       snapd.recovery-chooser-trigger.service  unattended-upgrades.service
ebtables.service      pollinate.service          snapd.seeded.service      vmstat.service
irqbalance.service    remote-fs.target            snapd.service
lxcfs.service         rsync.service              ssh.service
lxd-containers.service rsyslog.service            sysstat.service
```

심볼릭 링크 파일 확인

꼭 하세요!!  
안그럼 로그 쌓입니다!!

## 4. vmstat 서비스 중지 및 등록 해제

```
root@linux-test:~# systemctl stop vmstat.service
root@linux-test:~# systemctl disable vmstat.service
Removed /etc/systemd/system/multi-user.target.wants/vmstat.service.
```

# 리눅스 시스템 종료

## ● shutdown

- **기능** : 리눅스 시스템을 종료
- **형식** : **shutdown** [옵션] [시간] [메시지]

모든 사용자에게 보낼 메시지

시스템을 종료할 시간(hh:mm, +m, now)

### 주요 옵션

옵션	설명
<b>-h</b>	시스템을 종료(power-off)
<b>-r</b>	시스템을 재시작(reboot)
<b>-k</b>	실제로 종료하거나 재부팅하지 않고, 단지 로그인 중인 사용자에게 메시지 전송
<b>-c</b>	이전에 지정한 shutdown 명령을 취소(cancel)
<b>-f</b>	시스템 종료 또는 재부팅 강제 실행

### 사용 예

**shutdown -h now**

: 바로 시스템 종료 절차 시작

**shutdown -f now**

: 바로 시스템 강제 종료

**shutdown -h +2 "shutdown in 2 min"**

: 2분 후에 시스템이 종료되도록 설정하고 모든 사용자에게 메시지 전송

**shutdown -r +3**

: 3분 후 시스템 재시작하도록 설정

**shutdown -c**

: shutdown 명령 취소

# Appendix – init 프로세스



리눅스의 init 프로세스는 뭐야?



리눅스에서 **init은 시스템 부팅 중에 가장 처음 시작되는 프로세스로, 시스템 초기화 및 다**  
**른 프로세스를 시작하는데 사용됩니다.** init은 모든 리눅스 배포판에서 사용되는 프로세스이  
며, 시스템의 초기화 단계에서 중요한 역할을 합니다.



init은 일반적으로 시스템 레벨을 제어하는데 사용되며, 시스템 레벨은 시스템의 상태와 기  
능을 결정하는데 중요한 역할을 합니다. 예를 들어, init은 시스템 부팅 후 자동으로 실행되  
는 프로세스를 결정하고, 시스템 레벨을 변경하여 특정 서비스를 시작 또는 중지할 수 있습  
니다.

init은 일반적으로 다른 시스템 프로세스를 시작하는데 사용되는 스크립트 파일을 실행합니  
다. 이러한 스크립트 파일은 일반적으로 `/etc/init.d` 디렉토리에 저장되며, 시스템 레벨에  
따라 시작되거나 중지됩니다. init은 또한 `/etc/inittab` 파일을 사용하여 초기화 및 부팅 시  
스템 레벨을 설정합니다.

**최신 리눅스 시스템에서는 systemd라는 init 시스템이 사용되기도 합니다.** systemd는 init의  
기능을 수행하면서 시스템의 초기화, 프로세스 관리, 로깅, 네트워크 설정 등의 작업을 처리  
하는 더욱 강력한 시스템입니다.



# Appendix - 백그라운드 프로세스와 데몬 프로세스 차이



백그라운드 프로세스와 데몬 프로세스의 차이에 대해 알려줘



백그라운드 프로세스와 데몬 프로세스는 둘 다 백그라운드에서 실행되는 프로세스입니다. 그러나 두 개념은 몇 가지 차이점이 있습니다.

- 백그라운드 프로세스: 백그라운드 프로세스는 일반적으로 사용자가 직접 실행하는 프로세스입니다. 예를 들어, 사용자가 콘솔에서 명령어를 입력하면 해당 명령어가 백그라운드에서 실행됩니다. 이러한 프로세스는 사용자와 상호작용할 수 있습니다. 즉, 사용자가 입력한 명령어의 결과를 출력하거나, 사용자가 입력한 신호를 받아들이거나, 사용자와의 통신 등이 가능합니다. 또한, 백그라운드 프로세스는 사용자가 로그아웃하면 함께 종료됩니다.
- 데몬 프로세스: 데몬 프로세스는 시스템에서 백그라운드에서 실행되는 프로세스로, 사용자와 직접적인 상호작용이 없습니다. 데몬 프로세스는 시스템의 특정 서비스를 제공하거나, 백그라운드에서 특정 작업을 수행하며, 주로 서버나 시스템 관리와 같은 작업에 사용됩니다. 데몬 프로세스는 일반적으로 부모 프로세스가 없으며, 시스템 시작 시 자동으로 시작됩니다. 또한, 데몬 프로세스는 사용자가 로그아웃해도 계속해서 실행됩니다.

따라서, 백그라운드 프로세스와 데몬 프로세스는 운영체제에 맞게 사용되는 것이 일반적입니다.

init 프로세스가 부모가 됨

# Appendix – 부하(Load)의 의미



리눅스 시스템에서 부하(Load)라는 건 뭘 의미하는 거야?



리눅스 시스템에서 부하(Load)는 시스템이 처리할 수 있는 작업의 양보다 더 많은 작업이 존재하여 대기 중인 작업의 수를 나타내는 지표입니다. 부하는 일반적으로 1분, 5분, 15분의 평균값으로 표시됩니다.

리눅스 시스템에서는 CPU, 디스크, 메모리 등 여러 자원들이 있으며 이러한 자원들은 작업을 처리하는데 한계가 있습니다. 따라서 시스템에 적재되어 있는 작업이 너무 많아지면 자원 부족 현상이 발생하게 되고, 이는 시스템 성능에 영향을 줍니다.

부하는 일반적으로 1, 5, 15분의 평균값으로 표시되는 작업량을 의미합니다. 부하 값이 1을 넘어가면 심각한 과부하 상태라고 볼 수 있습니다. 따라서 적절한 대응을 취해야 합니다.



리눅스의 uptime 명령에서 표시되는 load average는 어떻게 계산되는 거야?



리눅스에서의 load average란, 특정 시간 동안 시스템에 얼마나 많은 작업이 대기중인지를 나타내는 지표입니다. 이 지표는 'uptime' 명령을 통해 확인할 수 있습니다.

load average는 시스템에 대기 중인 작업 수를 세 가지 값으로 나타냅니다. 각각은 1분, 5분, 15분 간격의 평균 값입니다. 이러한 값은 시스템의 프로세스 큐에서 대기 중인 프로세스 수를 포함하고 있습니다.

load average는 다음과 같이 계산됩니다.

- 1분 평균: 최근 1분 동안 시스템에 대기 중인 작업 수
- 5분 평균: 최근 5분 동안 시스템에 대기 중인 작업 수의 평균
- 15분 평균: 최근 15분 동안 시스템에 대기 중인 작업 수의 평균

이 수치는 일반적으로 CPU의 코어 수보다 적은 수치를 유지하는 것이 좋습니다. 만약 load average가 CPU 코어 수를 초과하거나 매우 높게 유지된다면 시스템에 문제가 있을 가능성이 있으므로, 시스템의 상태를 파악하고 대응하는 것이 필요합니다.

# Appendix – 버퍼(Buffer)와 캐시(cache)의 차이

위키백과 참조

## 버퍼(Buffer)

보통 데이터를 받는 쪽의 처리 속도가 보내는 쪽보다 느릴 때, 받는 쪽에서 데이터가 유실되지 않도록 배려(?)하기 위한 임시 메모리 영역 (메모리 to HDD)

컴퓨팅에서 버퍼(buffer, **문화어**: 완충기억기)는 데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역이다. 버퍼링(buffering)이란 버퍼를 활용하는 방식 또는 버퍼를 채우는 동작을 말한다. 다른 말로 '큐(Queue)'라고도 표현한다.

버퍼는 컴퓨터 안의 프로세스 사이에서 데이터를 이동시킬 때 사용된다. 보통 데이터는 키보드와 같은 입력 장치로부터 받거나 프린터와 같은 출력 장치로 내보낼 때 버퍼 안에 저장된다. 이는 전자 통신의 버퍼와 비유할 수 있다. 버퍼는 하드웨어나 소프트웨어에 추가될 수 있지만 버퍼는 상당수가 소프트웨어에 추가된다. 버퍼는 보통 속도가 계속 바뀔 수 있으므로 데이터 수신, 처리 속도에 차이가 있다. (예: 프린터 스펀러)

버퍼는 네트워크 상에서 자료를 주고 받을 때나 스피커에 소리를 재생할 때, 또는 디스크 드라이브와 같은 하드웨어의 입출력을 결합하는 데에 자주 이용된다. 버퍼는 또한 순서대로 데이터를 출력하는 FIFO 방식에서 보통 사용된다.

## 캐시(Cache)

캐시(cache, **문화어**: 캐쉬, 고속완충기, 고속완충기억기)는 컴퓨터 과학에서 데이터나 값을 미리 복사해 놓는 임시 장소를 가리킨다. 캐시는 캐시의 접근 시간에 비해 원래 데이터를 접근하는 시간이 오래 걸리는 경우나 값을 다시 계산하는 시간을 절약하고 싶은 경우에 사용한다. 캐시에 데이터를 미리 복사해 놓으면 계산이나 접근 시간없이 더 빠른 속도로 데이터에 접근할 수 있다.

캐시는 시스템의 효율성을 위해 여러 분야에서 두루 쓰이고 있다.

더 빨리 데이터에 접근할 수 있도록 하는 메모리 영역



# Appendix - 리눅스의 실행 레벨(runlevel) - 정의



리눅스의 runlevel에 대해 systemd와 관련해서 설명해줘





리눅스에서 runlevel은 특정 시스템 상태에 대한 식별자로, 시스템이 부팅될 때 어떤 서비스와 프로세스가 시작될지 결정합니다. 예를 들어, runlevel 3은 텍스트 모드에서 네트워크 및 다른 기본 서비스가 실행되는 상태를 나타내며, runlevel 5는 그래픽 모드에서 X Window 시스템과 같은 그래픽 서비스가 추가로 실행되는 상태를 나타냅니다.

systemd는 초기화 및 서비스 관리 프로세스입니다. systemd는 runlevel 대신에 "target"을 사용합니다. target은 서비스와 관련된 여러 가지 유형 중 하나를 나타냅니다. 예를 들어, multi-user.target은 runlevel 3과 유사하며, 그래픽 모드에서 시작하는 것을 제외하고 기본 서비스가 시작됩니다.

systemd에서 runlevel 대신 target을 사용하는 이유는 runlevel이 일반적으로 고정된 서비스 목록을 나타내는 반면, target은 서비스 간의 종속성을 보다 유연하게 다룰 수 있기 때문입니다. 또한 systemd는 더 빠른 부팅과 자동 병렬 실행 등의 기능을 제공하며, 이전 SysV init 시스템과 비교하여 향상된 성능과 안정성을 제공합니다.

# Appendix - 리눅스의 실행 레벨(runlevel) - 종류

 그럼 runlevel에는 어떻게 있어?

 기본적으로, 리눅스 시스템에서 사용되는 runlevel은 0부터 6까지 총 7가지가 있습니다. 각각의 runlevel은 시스템의 상태를 다르게 설정합니다. 아래는 각각의 runlevel에 대한 설명입니다:

- 0: Power off
- 1: Single user mode
- 2: Multi-user mode without networking
- 3: Multi-user mode with networking
- 4: Undefined
- 5: Multi-user mode with graphical user interface (GUI)
- 6: Reboot

기본적으로, 일반 사용자들은 runlevel 3 또는 5를 사용합니다. 이 중 runlevel 3은 텍스트 모드로 실행되며, 네트워크 및 기본 서비스가 시작됩니다. runlevel 5는 그래픽 모드로 실행되며, X Window 시스템과 같은 그래픽 서비스가 시작됩니다.

runlevel은 일반적으로 `/etc/inittab` 파일에서 설정할 수 있지만, 최신 버전의 리눅스 배포판에서는 systemd가 사용되기 때문에 `/etc/inittab`은 더 이상 사용되지 않습니다. 대신, systemd의 target을 사용하여 각각의 runlevel과 유사한 상태를 설정합니다.

# Appendix - 리눅스의 실행 레벨(runlevel) - systemd



그럼 각각의 runlevel이 어떤 target과 매칭되는 거지?



리눅스에서 사용되는 runlevel과 systemd target 간의 대응은 다음과 같습니다.

Runlevel 0 : poweroff.target  
Runlevel 1 : rescue.target  
Runlevel 2, 3, 4 : multi-user.target  
Runlevel 5 : graphical.target  
Runlevel 6 : reboot.target

현재 실행하고 있는 ubuntu 18.04에서는  
[Runlevel 5 : graphical.target]이  
default.target으로 설정됨

위의 목록에서, 각각의 runlevel은 해당하는 시스템 상태를 나타내며, systemd target은 해당하는 서비스와 프로세스를 관리하는 단위를 나타냅니다.

예를 들어, runlevel 5는 GUI 모드로 실행되며, systemd에서는 graphical.target을 사용하여 이와 같은 모드를 설정합니다. 이 target은 그래픽 모드에서 실행되는 서비스를 시작합니다. 이러한 서비스는 주로 X Window System과 같은 그래픽 서비스이며, 이러한 서비스는 runlevel 3에서는 실행되지 않습니다.

따라서, systemd에서 runlevel 대신 target을 사용하여, 더욱 유연하고 개별적으로 서비스를 관리할 수 있습니다.

# Thank You