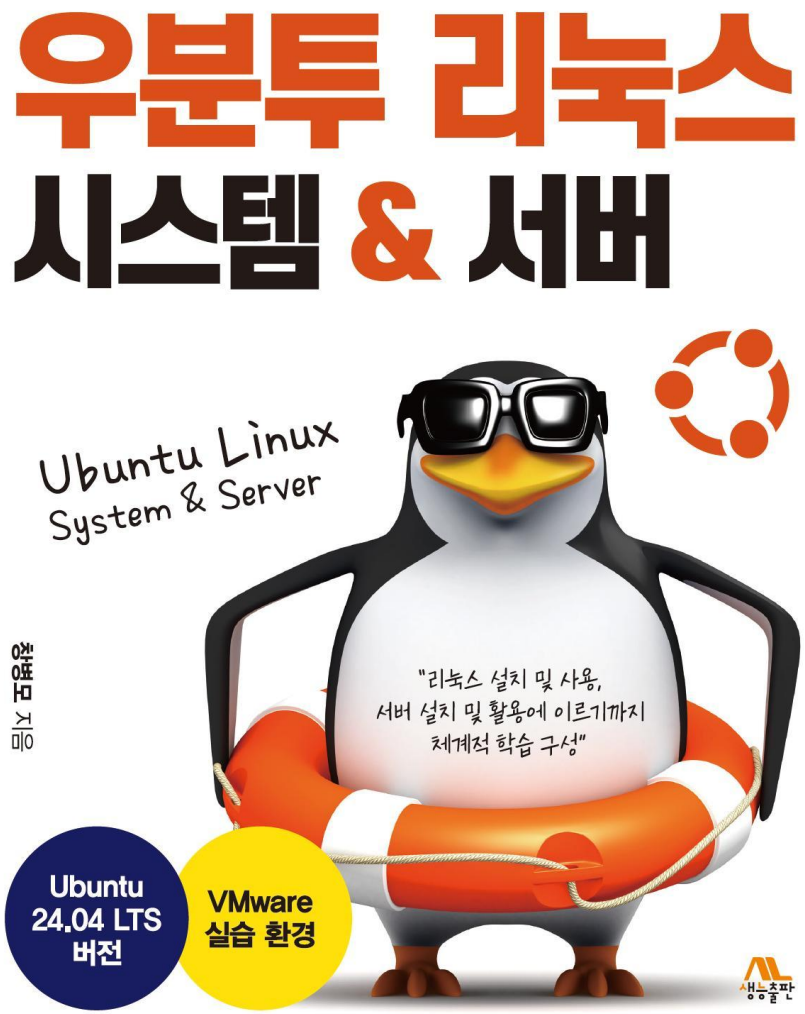
LINUX

Autumn 2025

Daejeon Univ.
Seongbok Baik
sbbaik@dju.ac.kr

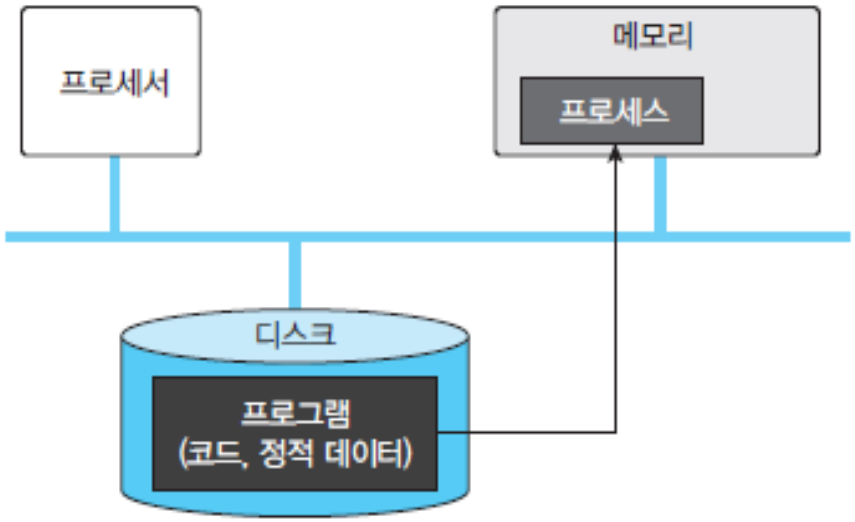
Text Book

교재명	우분투 리눅스 시스템 & 서버
저자	창병모
출판사	생능출판사
발행년	2024.07.12
ISBN	979-11-92932-72-9



프로세스(process)

- **프로세스**(process)는 실행중인 프로그램이다.
- 각 프로세스는 유일한 프로세스 번호 PID를 갖는다.
- 각 프로세스는 부모 프로세스에 의해 생성된다.



프로세스 상태 보기: ps(process staus)

- 사용법

> $ ps [-옵션]
>
> 현재 시스템 내에 존재하는 프로세스들의 실행 상태를 요약해서 출력한다.

- 사용 예

```
$ ps
PID   TTY      TIME     CMD
1519 pts/3    00:00:00 bash
1551 pts/3    00:00:00 ps

$ ps -f
UID     PID  PPID C STIME TTY TIME     CMD
chang 1519 1518 0 17:40 pts/3 00:00:00 /bin/bash
chang 1551 1519 0 17:40 pts/3 00:00:00 ps -f
```

ps –ef

```
$ ps -ef | more
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 4월04 ?        00:00:23 /sbin/init
root         2     0  0 4월04 ?        00:00:00 [kthreadd]
root         3     2  0 4월04 ?        00:00:00 [pool_workqueue_release]
root         4     2  0 4월04 ?        00:00:00 [kworker/R-rcu_g]
root         5     2  0 4월04 ?        00:00:00 [kworker/R-rcu_p]
root         6     2  0 4월04 ?        00:00:00 [kworker/R-slub_]
root         7     2  0 4월04 ?        00:00:00 [kworker/R-netns]
root        12     2  0 4월04 ?        00:00:00 [kworker/R-mm_pe]
root        13     2  0 4월04 ?        00:00:00 [rcu_tasks_kthread]
...
root        24     2  0 4월04 ?        00:00:00 [kauditd]
root        25     2  0 4월04 ?        00:00:00 [khungtaskd]
root        26     2  0 4월04 ?        00:00:00 [oom_reaper]
--More--
```

ps 출력 정보

항목	의미
UID	프로세스를 실행시킨 사용자 ID
PID	프로세스 번호
PPID	부모 프로세스 번호
C	프로세스의 우선순의
STIME	프로세스의 시작 시간
TTY	명령어가 시작된 터미널
TIME	프로세스에 사용된 CPU 시간
CMD	실행되고 있는 명령어(프로그램) 이름

특정 프로세스 리스트: pgrep

- 특정 프로세스만 리스트

```
$ ps -ef | grep -w sshd
```

- 사용법

```
$ pgrep [옵션] [패턴]
```

패턴에 해당하는 프로세스들만을 리스트 한다.

-l : PID와 함께 프로세스의 이름을 출력한다.

-f : 명령어의 경로도 출력한다.

-n : 패턴과 일치하는 프로세스들 중에서 가장 최근 프로세스만을 출력한다.

-x : 패턴과 정확하게 일치되는 프로세스만 출력한다.

특정 프로세스 리스트: pgrep

- 예

```
$ pgrep sshd
5032
```

- -l 옵션: 프로세스 번호와 프로세스 이름을 함께 출력

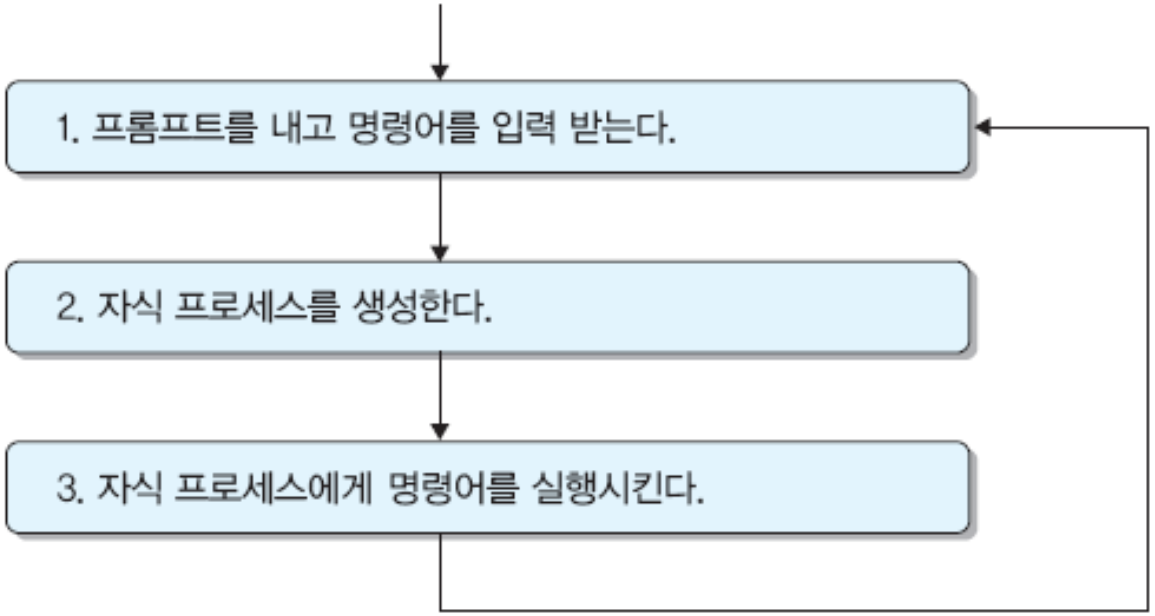
```
$ pgrep -l sshd
5032 sshd
```

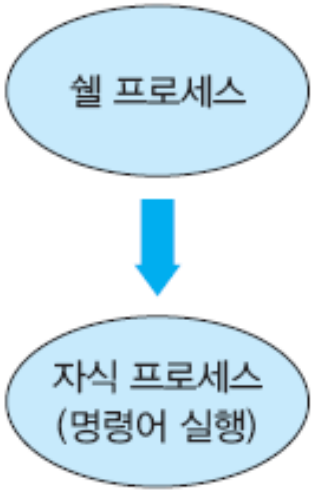
- -n 옵션: 가장 최근 프로세스만 출력한다.

```
$ pgrep -ln sshd
5032 sshd
```

6.2 작업 제어

쉘과 프로세스


1. 프롬프트를 내고 명령어를 입력 받는다.
2. 자식 프로세스를 생성한다.
3. 자식 프로세스에게 명령어를 실행시킨다.


쉘 프로세스
자식 프로세스
(명령어 실행)

후면 처리

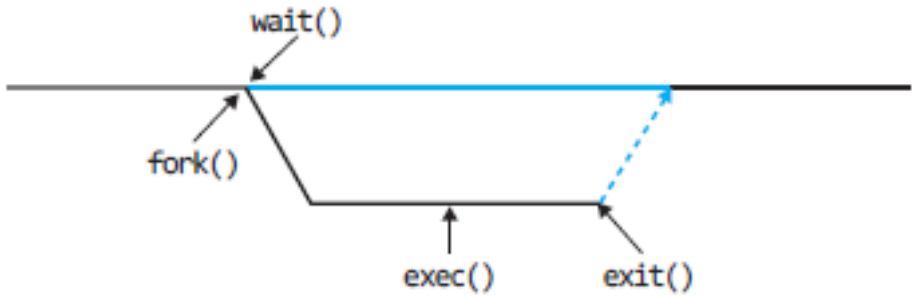
$ 명령어 &
[1] 프로세스번호

- 예

```
$ sleep 10 &
[1] 6530
$ ps
PID TTY TIME CMD
1519 pts/0 00:00:00 bash
6530 pts/0 00:00:00 sleep
6535 pts/0 00:00:00 ps
```

쉘의 명령어 처리 원리

- fork() 시스템 호출
  - 새로운 자식 프로세스를 생성한다.
- exec() 시스템 호출
  - 명령어(프로그램)을 실행시킨다.
- 쉘의 명령어 처리 원리
  - 쉘(부모) 프로세스는 fork() 호출로 자식 프로세스를 생성하고
  - 자식 프로세스는 exec() 호출로 명령어(프로그램)을 실행한다.
  - 자식 프로세스는 exec() 호출이 성공하면 exit() 하여 종료한다.
  - 부모 프로세스는 자식 프로세스가 끝나기를 기다린다.



쉘 재우기

- 사용법

```
$ sleep 초
```

명시된 시간만큼 프로세스 실행을 중지시킨다.

- 예

```
$ (echo 시작; sleep 5; echo 끝)
```

강제 종료

- 강제종료 Ctrl-C
  ```
  $ 명령어
  ^C
  ```

- 예
  ```
  $ (sleep 100; echo DONE)
  ^C
  $
  ```

- 실행 정지 Ctrl-Z
  ```
  $ 명령어
  ^Z
  [1]+ 멈춤 명령어
  ```

후면 작업의 전면 전환: fg(foreground)

$ fg

정지된 작업을 다시 전면에서 실행시킨다.

- 예

```
$ (sleep 100; echo done)
^Z
[1]+ 멈춤 ( sleep 100; echo DONE )
$ fg
( sleep 100; echo DONE )
```

후면 작업의 전면 전환: fg(foreground)

$ fg %작업번호

작업번호에 해당하는 후면 작업을 전면 작업으로 전환시킨다.

- 예

```
$ (sleep 100; echo DONE) &
[1] 10067
$ fg %1
( sleep 100; echo DONE )
```

전면 작업의 후면 전환: bg(background)

- 사용법
  - Ctrl-Z 키를 눌러 전면 실행중인 작업을 먼저 중지시킨 후
  - bg 명령어 사용하여 후면 작업으로 전환

> $ bg %작업번호
>
> 작업번호에 해당하는 중지된 작업을 후면 작업으로 전환하여 실행한다.

- 예

```
$ ( sleep 100; echo DONE )
^Z
[1]+ 멈춤     ( sleep 100; echo DONE )
$ bg %1
[1]+ ( sleep 100; echo DONE ) &
```

후면 작업의 입출력 제어

- 후면 작업의 출력

```
$ 명령어 > 출력파일 &
$ find . -name test.c -print > find.txt &
$ find . -name test.c -print | mail chang &
```

- 후면 작업의 입력

```
$ 명령어 < 입력파일 &
```

6.3 프로세스 제어

프로세스 끝내기: kill

- 프로세스 강제 종료

```
$ kill 프로세스번호
$ kill %작업번호
```

프로세스 번호(혹은 작업 번호)에 해당하는 프로세스를 강제로 종료시킨다.

- 예

```
$ (sleep 100; echo done) &
[1] 8320
$ kill 8320 혹은 $ kill %1
[1] 종료됨    ( sleep 100; echo done )
```

- exit 명령어

```
exit [종료코드]
```

프로세스 기다리기: wait

- 사용법

```
$ wait [프로세스번호]
```

프로세스 번호로 지정한 자식 프로세스가 종료될 때까지 기다린다.

지정하지 않으면 모든 자식 프로세스가 끝나기를 기다린다.

- 예

```
$ (sleep 10; echo 1번 끝) &
[1] 1231
$ echo 2번 끝; wait 1231; echo 3번 끝
2번 끝
1번 끝
3번 끝
```

로그아웃 후에도 프로세스 실행하기

$ nohup 명령어 [인수] &

로그아웃 후에도 명령어 실행이 계속되도록 실행한다.

- 출력 nohup.out에 저장

```
$ nuhup find / -name *.c -print &
```

- 출력 파일 지정

```
$ nuhup find / -name *.c -print > filename &
```

프로세스 기다리기: wait

- 예

```
$ (sleep 10; echo 1번 끝) &
$ (sleep 10; echo 2번 끝) &
$ echo 3번 끝; wait; echo 4번 끝
3번 끝
1번 끝
2번 끝
4번 끝
```

프로세스 우선순위

- 실행 우선순위 nice 값
  - 19(제일 낮음) ~ -20(제일 높음)
  - 보통 기본 우선순위 0으로 명령어를 실행

- nice 명령어

```
$ nice [-n 조정수치] 명령어 [인수들]
```

주어진 명령을 조정된 우선순위로 실행한다.

- 예

```
$ nice                  // 현재 우선순위 출력
0
$ nice -n 10 ps -ef         // 조정된 우선순위로 실행
```

프로세스 우선순위 조정

- 사용법

```
$ renice [-n] 우선순위 [-gpu] PID
```

이미 수행중인 프로세스의 우선순위를 명시된 우선순위로 변경한다.

-g : 해당 그룹명 소유로 된 프로세스를 의미한다.

-u : 지정한 사용자명의 소유로 된 프로세스를 의미한다.

-p : 해당 프로세스의 PID를 지정한다.

6.4 프로세스의 사용자 ID

프로세스의 사용자 ID

- 프로세스의 사용자 ID와 그룹 ID를 갖는다.
  - 그 프로세스를 실행시킨 사용자의 ID와 사용자의 그룹 ID
  - 프로세스가 수행할 수 있는 연산을 결정하는 데 사용된다.

- id 명령어

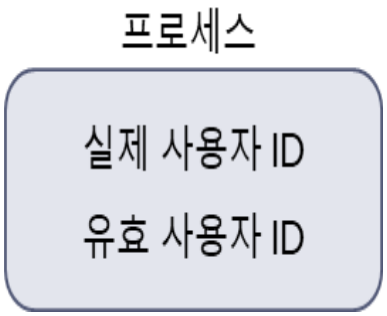
```
$ id [사용자명]
```

사용자의 실제 ID와 유효 사용자 ID, 그룹 ID 등을 보여준다.

```
$ id
uid=1000(chang) gid=1000(chang) groups=1000(chang),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),100(users),114(lpadmin)

$ echo $UID $EUID
1000 1000
```

프로세스의 사용자 ID

- 프로세스의 실제 사용자 ID(real user ID)
  - 그 프로세스를 실행시킨 사용자의 ID로 설정된다.
  - 예: chang 사용자 ID로 로그인하여 어떤 프로그램을 실행시키면 그 프로세스의 실제 사용자 ID는 chang이 된다.
- 프로세스의 유효 사용자 ID(effective user ID)
  - 현재 유효한 사용자 ID
  - 보통 유효 사용자 ID와 실제 사용자 ID는 같다.
  - 새로 파일을 만들 때나 파일의 접근권한을 검사할 때 주로 사용됨
  - 특별한 실행파일을 실행할 때 유효 사용자 ID는 달라진다.



set-user-id 실행파일

- set-user-id(set user ID upon execution) 실행권한
  - set-user-id가 설정된 실행파일을 실행하면
  - 이 프로세스의 유효 사용자 ID는 그 실행파일의 소유자로 바뀜.
  - 이 프로세스는 실행되는 동안 그 파일의 소유자 권한을 갖게 됨.

- 예

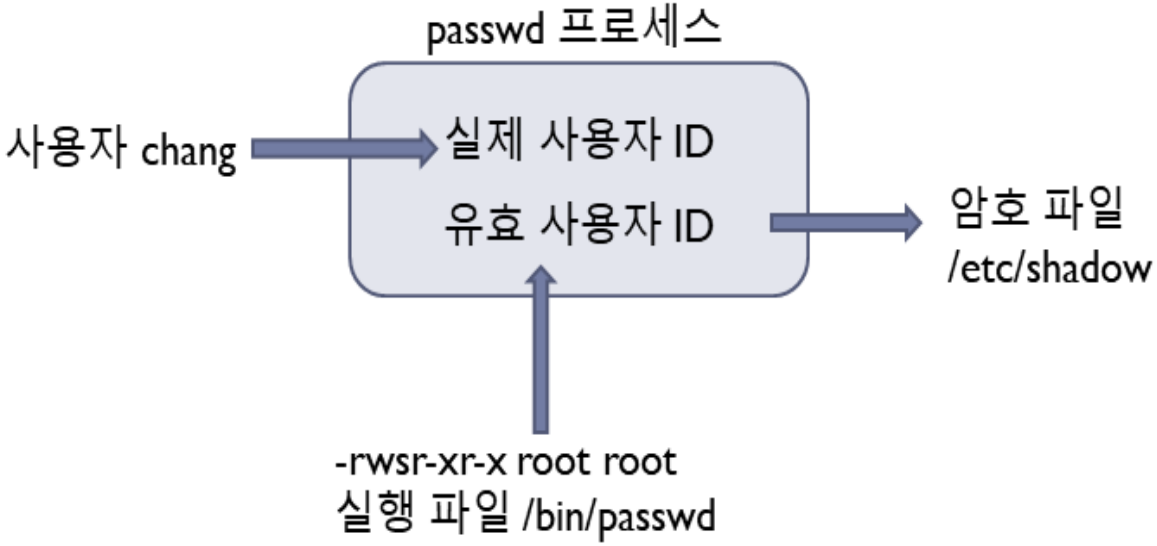
```
$ ls -l /bin/sudo
-rwsr-xr-x 1 root root 277936 4월 1 02:17 /bin/sudo

$ ls -l /bin/passwd
-rwsr-xr-x 1 root  root  59976  4월 14 14:57 /bin/passwd
```

set-user-id 실행파일을 실행하는 과정

(1) /etc/passwd 파일은 set-user-id 실행권한이 설정된 실행파일이며 소유자는 root

(2) 일반 사용자가 이 파일을 실행하면 이 프로세스의 유효 사용자 ID는 root가 됨.

(3) 유효 사용자 ID가 root이므로 root만 수정할 수 있는 암호 파일 /etc/shadow 파일을 접근하여 수정



set-group-id 실행파일

- set-group-id(set group ID upon execution) 실행권한
  - 실행되는 동안에 그 파일 소유자의 그룹이 프로세스의 유효 그룹 ID가 된다.
  - set-group-id 실행권한은 8진수 모드로는 2000으로 표현된다.

- set-group-id 실행파일 예

```
$ ls -l /bin/crontab
-rwxr-sr-x 1 root crontab 39664 3월 31 09:06 /bin/crontab
```

set-user-id/set-group-id 설정

- set-user-id 실행권한 설정

  $ chmod 4755 파일 혹은 $ chmod u+s 파일

- set-group-id 실행권한 설정

  $ chmod 2755 파일 혹은 $ chmod g+s 파일

6.5 시그널과 프로세스

시그널

- 시그널은 예기치 않은 사건이 발생할 때 이를 알리는 소프트웨어 인터럽트이다.
- 시그널 발생 예
  - SIGFPE 부동소수점 오류
  - SIGPWR 정전
  - SIGALRM 알람시계 울림
  - SIGCHLD 자식 프로세스 종료
  - SIGINT 키보드로부터 종료 요청 (Ctrl-C)
  - SIGTSTP 키보드로부터 정지 요청 (Ctrl-Z)

주요 시그널

시그널 이름	의미	기본 처리
SIGABRT	abort()에서 발생되는 종료 시그널	종료(코어 덤프)
SIGALRM	자명종 시계 alarm() 울림 때 발생하는 알람 시그널	종료
SIGCHLD	프로세스의 종료 혹은 정지를 부모에게 알리는 시그널	무시
SIGCONT	정지된 프로세스를 계속시키는 시그널	무시
SIGFPE	0으로 나누기와 같은 심각한 산술 오류	종료(코어 덤프)
SIGHUP	연결 끊김	종료
SIGILL	잘못된 하드웨어 명령어 수행	종료(코어 덤프)
SIGIO	비동기화 I/O 이벤트 알림	종료
SIGINT	터미널에서 Ctrl-C 할 때 발생하는 인터럽트 시그널	종료
SIGKILL	잡을 수 없는 프로세스 종료시키는 시그널	종료
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료
SIGPIPE	끊어진 파이프	종료

주요 시그널

SIGPWR	전원고장	종료
SIGSEGV	유효하지 않은 메모리 참조	종료(코어 덤프)
SIGSTOP	프로세스 정지 시그널	정지
SIGTSTP	터미널에서 Ctrl-Z 할 때 발생하는 정지 시그널	정지
SIGSYS	유효하지 않은 시스템 호출	종료(코어 덤프)
SIGTERM	잡을 수 있는 프로세스 종료 시그널	종료
SIGTTIN	후면 프로세스가 제어 터미널을 읽기	정지
SIGTTOU	후면 프로세스가 제어 터미널에 쓰기	정지
SIGUSR1	사용자 정의 시그널	종료
SIGUSR2	사용자 정의 시그널	종료

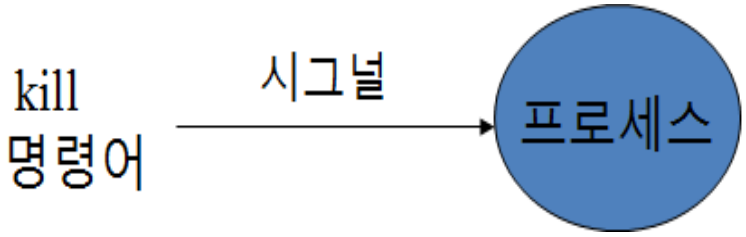
시그널 리스트

- $ 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) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
```

시그널 보내기: kill 명령어

- kill 명령어
  - 한 프로세스가 다른 프로세스를 제어하기 위해 특정 프로세스에 임의의 시그널을 강제적으로 보낸다.



- 사용법

```
$ kill [-시그널] 프로세스번호
$ kill [-시그널] %작업번호
```

프로세스 번호(혹은 작업 번호)로 지정된 프로세스에 원하는 시그널을 보낸다.

시그널을 명시하지 않으면 SIGTERM 시그널을 보내 해당 프로세스를 강제 종료

시그널 보내기: kill 명령어

- 종료 시그널 보내기

```
$ kill -9 프로세스번호
$ kill -KILL 프로세스번호
```

- 다른 시그널 보내기

```
$ 명령어 &
[1] 1234
$ kill -STOP 1234
[1]+ 멈춤    명령어
$ kill -CONT 1234
```

6.6 명령 스케줄링

주기적 실행 cron

- cron 시스템
  - 유닉스의 명령어 스케줄링 시스템으로
  - crontab 파일에 명시된 대로 주기적으로 명령을 수행한다.

- crontab 파일 등록법

```
$ crontab 파일
```

crontab 파일을 cron 시스템에 등록한다.

- crontab 파일
  - 7개의 필드로 구성
  - 분 시 일 월 요일 [사용자] 명령

주기적 실행 cron

- crontab 명령어

$ crontab -l [사용자]

사용자의 등록된 crontab 파일 리스트를 보여준다.

$ crontab -e [사용자]

사용자의 등록된 crontab 파일을 수정 혹은 생성한다.

$ crontab -r [사용자]

사용자의 등록된 crontab 파일을 삭제한다.

crontab 파일 예

- chang.cron

```
30 18 * * * rm /home/chang/tmp/*
```

- 사용예

```
$ crontab chang.cron
$ crontab -l
 30 18 * * * rm /home/chang/tmp/*
$ crontab -r
$ crontab -l
 no crontab for chang
```

crontab 파일 예

- crontab 파일 예1

  0 * * * * echo “뻐꾹” >> /tmp/x

  매 시간 정각에 “뻐꾹” 메시지를 /tmp/x 파일에 덧붙인다.

- crontab 파일 예2

  20 1 * * * root find /tmp -atime +3 -exec rm -f {} \;

  매일 새벽 1시 20분에 3일간 접근하지 않은 /tmp 내의 파일을 삭제

- crontab 파일 예3

  30 1 * 2,4,6,8,10,12 3-5 /usr/bin/wall /var/tmp/message

  2개월마다 수요일부터 금요일까지 1시 30분에 wall 명령을 사용해서 시스템의 모든 사용자에게 메시지를 전송

한번 실행: at

- at 명령어
  - 미래의 특정 시간에 지정한 명령어가 한 번 실행되도록 한다.
  - 실행할 명령은 표준입력을 통해서 받는다.

- 사용법

> $ at [-f 파일] 시간
>
> 지정된 시간에 명령이 실행되도록 등록한다. 실행할 명령은 표준입력으로 받는다.
>
> -f : 실행할 명령들을 파일로 작성해서 등록할 수도 있다.

- 예

```
$ at 1145 jan 31
at> sort infile > outfile
at> <EOT>
```

한번 실행: at

- atq 명령어
  - at 시스템의 큐에 등록되어 있는 at 작업을 볼 수 있다.

- 사용예

```
$ atq
Rank Execution Date Owner Job Queue Job Name
1st Jan 31, 2012 11:45 chang 1327977900.a a stdin
```

- at -r 옵션

> $ at -r 작업번호
>
> 지정된 작업번호에 해당하는 작업을 제거한다.

- 사용 예

```
$ at -r 1327977900.a
```

핵심 개념

- 프로세스는 실행중인 프로그램이다.
- 각 프로세스는 프로세스 ID를 갖는다. 각 프로세스는 부모 프로세스에 의해 생성된다.
- 쉘은 사용자와 운영체제 사이에 창구 역할을 하는 소프트웨어로 사용자로부터 명령어를 입력받아 이를 처리하는 명령어 처리기 역할을 한다.
- 전면 처리는 명령어가 전면에서 실행되므로 쉘이 명령어 실행이 끝나기를 기다리지만 후면 처리는 명령어가 후면에서 실행되므로 쉘이 명령어 실행이 끝나기를 기다리지 않는다.
- 각 프로세스는 실제 사용자 ID와 유효 사용자 ID를 갖는다.
- 시그널은 예기치 않은 사건이 발생할 때 이를 알리는 소프트웨어 인터럽트이다.
- kill 명령어를 이용하여 특정 프로세스에 원하는 시그널을 보낼 수 있다.