오픈소스소프트웨어

Open-Source Software

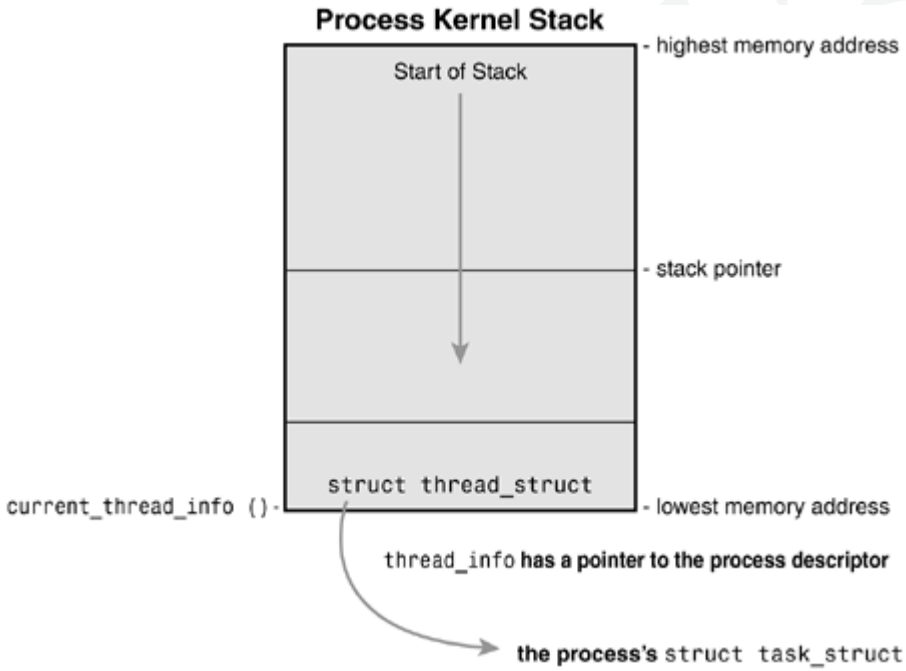
ICT융합학부 조용우

프로세스

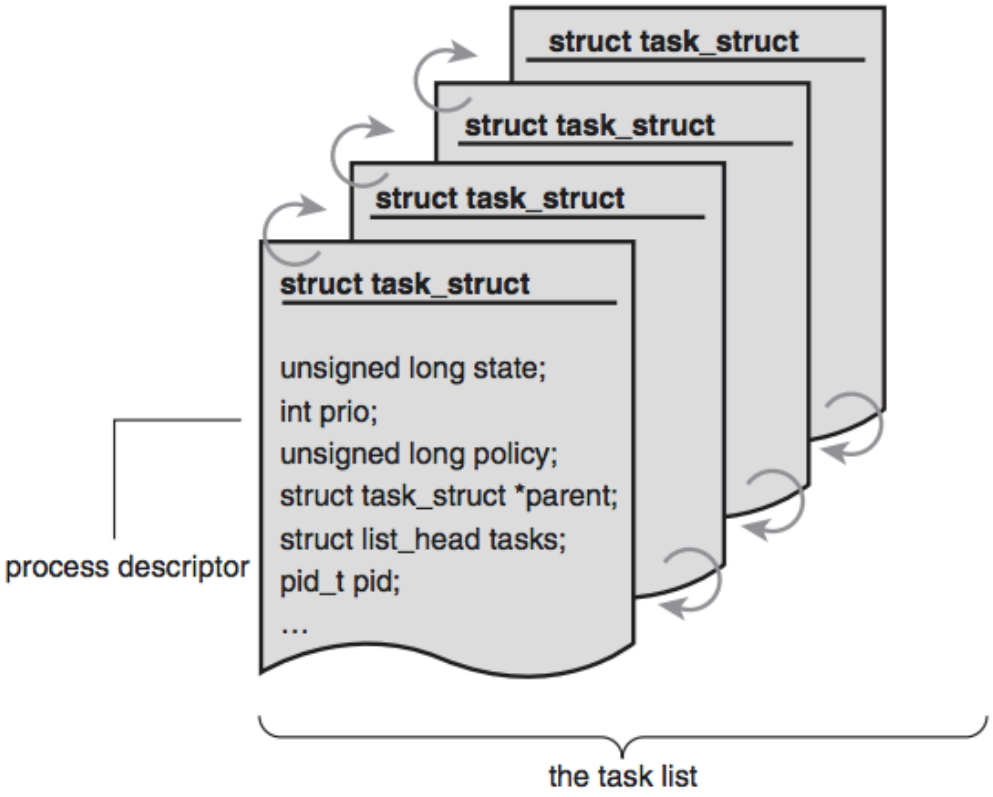
프로세스(process)

- 실행중인 프로그램을 **프로세스**(process)라고 부른다.
- 각 프로세스는 유일한 프로세스 번호 PID를 갖는다.
- 각 프로세스는 부모 프로세스에 의해 생성된다.

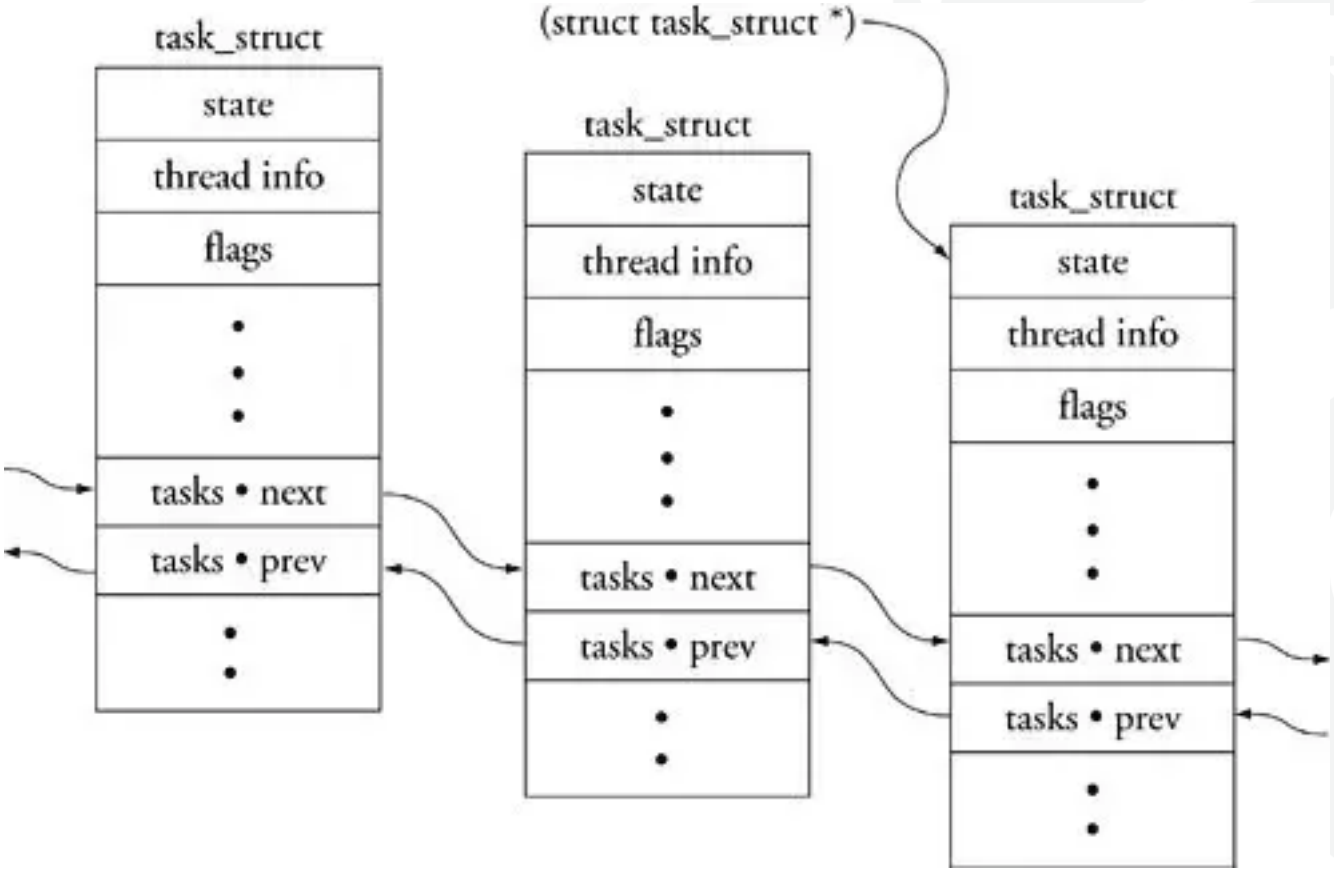
프로세스(process)



태스크(task)


struct task_struct
struct task_struct
struct task_struct
struct task_struct
unsigned long state;
int prio;
unsigned long policy;
struct task_struct *parent;
struct list_head tasks;
pid_t pid;
…
process descriptor
the task list

Task switching



프로세스의 식별

- 프로세스 ID (PID)
  - 각각의 프로세스를 구별하기 위한 고유한 값
  - 커널 내의 process table 의 index
  - 주요 커널 프로세스
    - 0 - swapper
    - 1 - init
    - 2 - pagedaemon

참조: 프로세스와 관련된 ID

- 프로세스 ID
  - 시스템에서 프로세스를 관리하기 위해 부여하는 ID
    - 프로세스 ID (PID)
    - 부모 프로세스 ID (PPID)
    - 프로세스 그룹 ID (PGID)
    - 세션 ID (SID)
- 유저 ID
  - 프로세스를 수행시키는 유저/그룹 ID 또는 프로세스의 권한 ID
    - Real user(group) ID (UID/GID)
    - Effective user(group) ID (EUID/EGID)
    - Saved-set-user(group) ID

fork - 자식 프로세스 생성

- Synopsis
  - #include <sys/types.h>
    #include <unistd.h>

    pid_t fork(void);
- Description
  - fork 함수를 호출한 부모(parent) 프로세스와 PID, PPID 만 다른 자식(child) 프로세스를 생성
- Return value
  - 성공 - PID 또는 0
    - 부모 프로세스에는 자식 프로세스의 PID 반환
    - 자식 프로세스에는 0 반환
  - 실패 - -1 (errno 설정)
    - 자식 프로세스 생성되지 않음

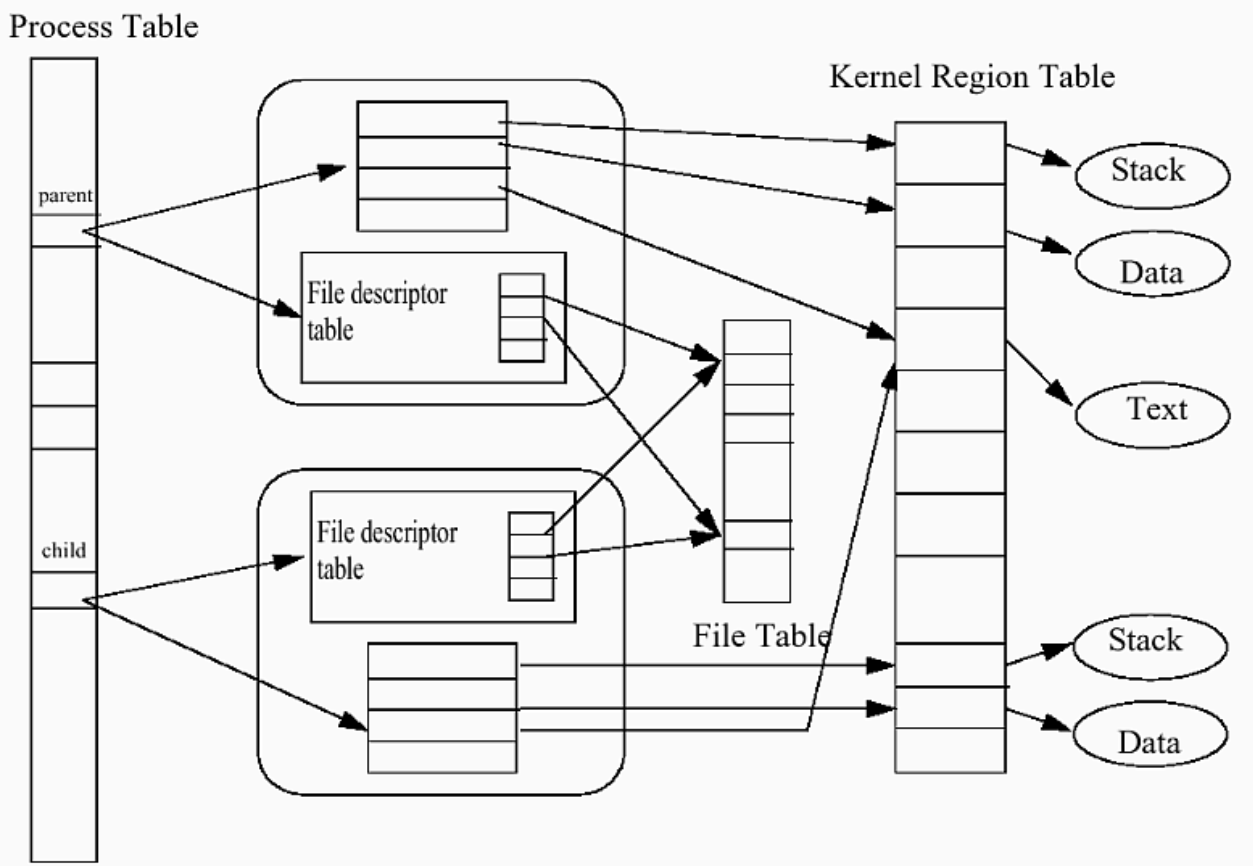
fork - 자식 프로세스 생성

- fork 이후 동작
  - 부모 프로세스
    - fork 가 child 의 PID 를 return 하고 그 시점부터 계속 수행
  - 자식 프로세스
    - 새로운 프로세스로서 스케쥴링에 들어감
    - 수행이 시작되는 지점은 fork 의 return 지점
    - fork 는 0 을 반환

fork - 자식 프로세스 생성

- 부모(parent) 프로세스
  - 변화하지 않음
- 자식 (child) 프로세스
  - 새로운 process table 항목을 생성하고 parent 의 region (code, data, stack, heap) 복사
  - 프로세스 정보
    - parent 로부터 정보 상속
      - » open files, file mode creation mask
      - » UID(GID), EUID(EGID), saved-set UID(GID)
      - » PGID, SID, controlling terminal
      - » current working directory, current root directory, environment
    - parent 와는 다른 정보
      - » fork 의 return value, PID, PPID

부모 프로세스와 자식 프로세스의 자료 구조



fork - 자식 프로세스 생성

- fork 용도
  - 동일한 프로그램의 실행
    - 같은 프로그램을 복제하여 부모와 자식이 각각 다른 코드 부분을 동시에 실행
    - 예
      - » 네트워크 서버 (network servers)
  - 새로운 프로그램의 실행
    - 자식 프로세스는 fork 후에 exec 함수를 호출하여 새로운 프로그램을 실행
    - 예
      - » 쉘 (shells)

fork.c

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

int main(int argc, char **argv) {
        pid_t pidP;

        printf("fork calling process : PID = %d\n", getpid());

        pidP = fork();

        if (pidP == 0) {        /* child process */
                printf("child process : PID = %d, PPID = %d, fork = %d\n",
              getpid(), getppid(), pidP);
        }
        else {                  /* parent process */
                printf("parent process : PID = %d, PPID = %d, fork = %d\n",
              getpid(), getppid(), pidP);
        }

        printf("both child & parent process :  PID = %d\n", getpid());
                                /* both child & parent process */

        return 0;
}
```

fork.c 실행결과

```
$ ./fork
fork calling process : PID = 4102
parent process : PID = 4102, PPID = 24196, fork = 4103
child process : PID = 4103, PPID = 4102, fork = 0
both child & parent process :  PID = 4103
both child & parent process :  PID = 4102
$ ./fork
fork calling process : PID = 4100
child process : PID = 4101, PPID = 4100, fork = 0
both child & parent process :  PID = 4101
parent process : PID = 4100, PPID = 24196, fork = 4101
both child & parent process :  PID = 4100
```

프로세스 상태 보기 : ps(process status)

$ ps [-옵션]

현재 시스템 내에 존재하는 프로세스들의 실행 상태를 요약해서 출력한다.

```
$ ps
PID TTY TIME CMD
8695 pts/3 00:00:00 bash
8720 pts/3 00:00:00 ps

$ ps -u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
ywcho 8695 0.0 0.0 5252 1728 pts/3 Ss 11:12 0:00 bash
ywcho 8793 0.0 0.0 4252  940 pts/3 R+ 11:15 0:00 ps -u
```

프로세스 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

$ 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

```
$ pgrep sshd
1720
1723
5032
```

- -l 옵션: 프로세스 번호와 프로세스 이름을 함께 출력

```
$ pgrep -l sshd
1720 sshd
1723 sshd
5032 sshd
```

- -n 옵션: 가장 최근 프로세스만 출력한다.

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

참고: 정규식

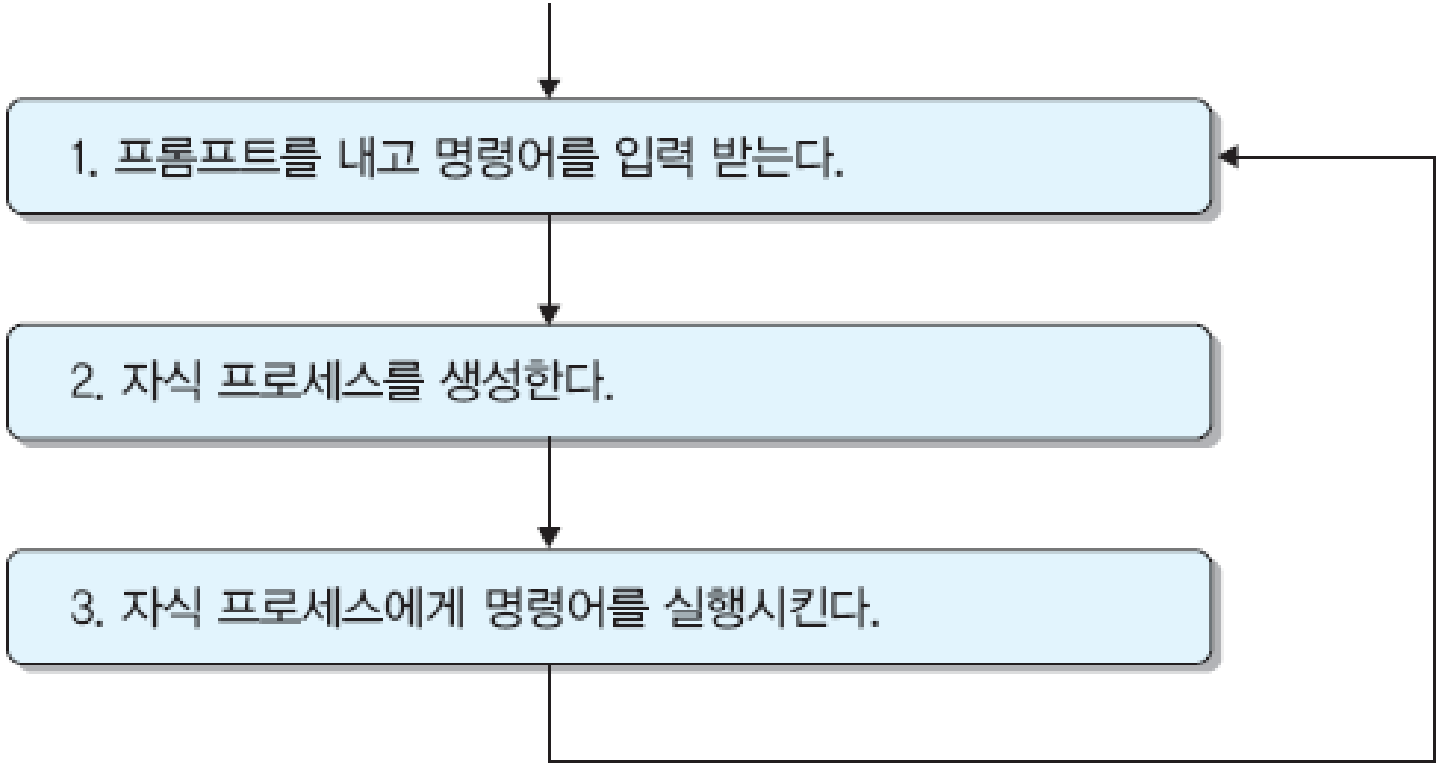
- 정규식(Regular Expression)
  - 유닉스 시스템에서 검색을 목적으로 활용
  - vi, grep, ex, sed, awk, emacs, more, less
- 메타문자
  - . : 아무 문자 하나와 매치
  - * : 앞의 문자가 0번 이상 나올 수 있음
  - + : 앞의 문자가 1번 이상 나올 수 있음
  - ^ : 라인의 시작
  - $: 라인의 끝
  - [ ] : 스퀘어 브래킷 안의 한 문자와 매치
  - ₩char : 메타 문자 자체를 나타냄

참고: 정규식

- 정규식의 예
  - bag : bag
  - ^bag : 라인의 시작이 bag
  - bag$: 라인의 끝이 bag
  - [Bb]ag : Bag 또는 bag
  - b[aeiou]g : bag, beg, big, bog, bug
  - b.g : b와 g사이에 아무 문자나 하나 있음
  - ... : 세글자로 된 단어
  - bugs* : bag, bags, bagss, bagsss ...
  - bugs.* : bugs로 시작되는 모든 단어
  - [A-Za-z] : 알파벳 한 문자
  - [a-z]+ : 소문자로 구성된 단어

작업제어 & 프로세스 제어

쉘과 프로세스



쉘 재우기

$ sleep 초

명시된 시간만큼 프로세스 실행을 중지시킨다.

```
$ (echo 시작; sleep 5; echo 끝)
```

강제 종료

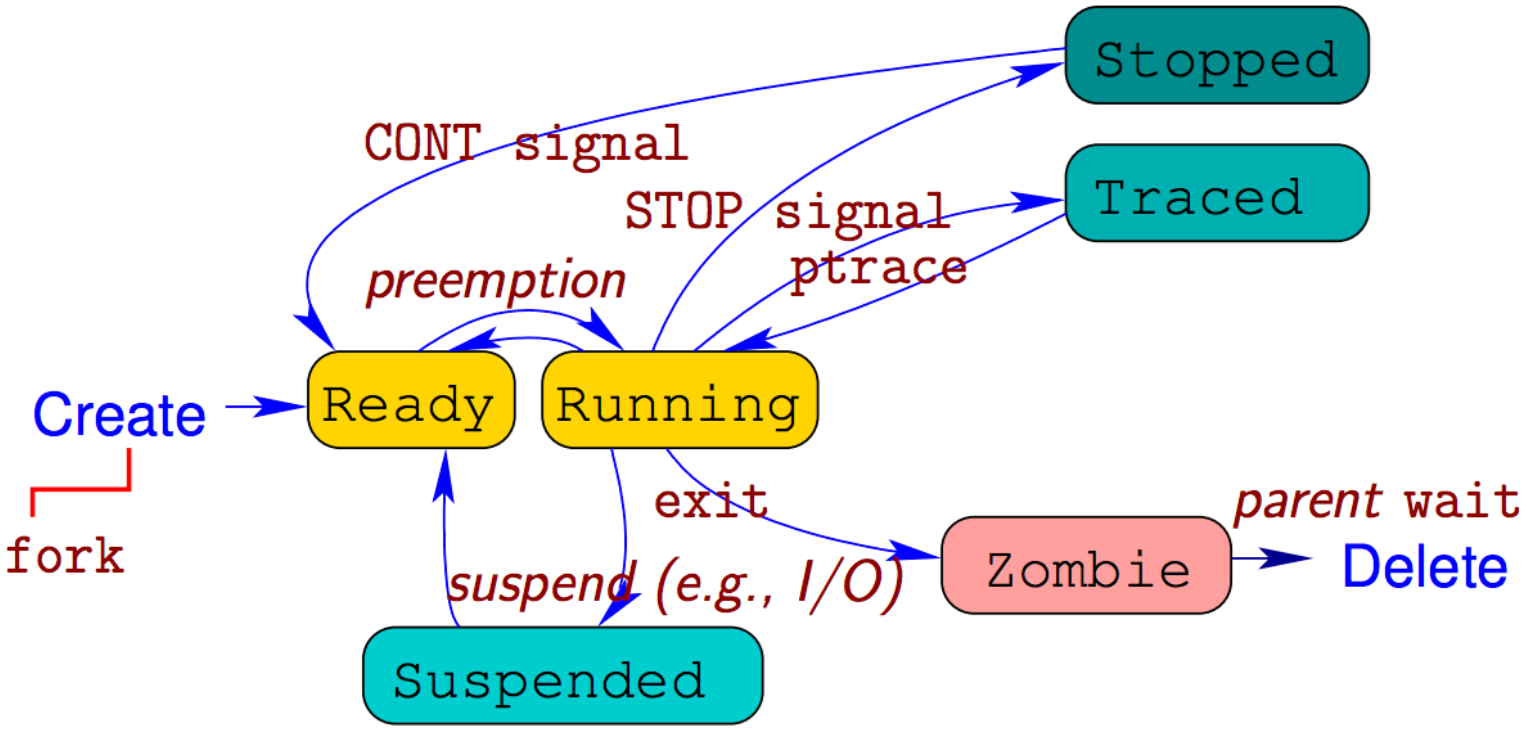
- 강제종료 Ctrl-C

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

- 실행 중지 Ctrl-Z

```
$ 명령어
^Z
[1]+ Stopped 명령어
```

프로세스의 상태


Stopped
CONT signal
STOP signal
Traced
preemption
ptrace
Create
Ready
Running
fork
exit
parent wait
Zombie
Delete
suspend (e.g., I/O)
Suspended

프로세스 끝내기 : kill

$ kill 프로세스 번호

$ kill %작업번호

프로세스 번호(혹은 작업 번호)에 해당하는 프로세스를 강제로 종료 시킨다.

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

프로세스 기다리기 : wait

$ wait [프로세스번호]

프로세스 번호로 지정한 자식 프로세스가 종료될 때까지 기다린다. 지정하지 않으면 모든 자식 프로세스가 끝나기를 기다린다.

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

프로세스 우선순위

$ nice [-n 조정수치] 명령어 [인수들]

주어진 명령을 조정된 우선순위로 실행한다.

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

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

프로세스 우선순위 조정

$ 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.

프로세스의 사용자 ID

프로세스의 사용자 ID

$ id [사용자명]

사용자의 실제 ID와 유효 사용자 ID, 그룹 ID등을 보여준다.

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

```
$ id
uid=1000(ywcho) gid=1002(cs) groups=1002(cs)
context=system_u:unconfined_r:unconfined_t:s0
```

프로세스의 사용자 ID

- 프로세스의 실제 사용자 ID(real user ID)
  - 그 프로세스를 실행시킨 사용자의 ID로 설정된다.
  - 예: ywcho 사용자 ID로 로그인하여 어떤 프로그램을 실행시키면 그 프로세스의 실제 사용자 ID는 ywcho이 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.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 27000 2010-08-22 12:00 /usr/bin/passwd
```

  - set-user-id 실행권한이 설정된 실행파일이며 소유자는 root
  - 일반 사용자가 이 파일을 실행하게 되면 이 프로세스의 유효 사용자 ID는 root가 됨.
  - /etc/passwd처럼 root만 수정할 수 있는 파일의 접근 및 수정 가능

set-group-id 실행파일

- set-group-id(set group ID upon execution) 실행권한
  - 실행되는 동안에 그 파일 소유자의 그룹을 프로세스의 유효 그룹 ID로 갖게 된다.
  - set-group-id 실행권한은 8진수 모드로 2000으로 표현된다.

- set-group-id 실행파일 예

```
$ ls -l /usr/bin/wall
-r-xr-sr-x. 1 root tty 15344 6월 10 2014 /usr/bin/wall
```

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

시그널

시그널(signal)

- 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)
SIGSTP	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 중지를 부모에게 알리는 시그널	무시
SIGCONT	중지된 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

주요 시그널

시그널 이름	의미	기본 처리
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료
SIGPWR	전원고장	종료
SIGSEGV	유효하지 않은 메모리 참조	종료(코어 덤프)
SIGSTOP	프로세스 중지 시그널	중지
SIGSTP	터미널에서 Ctrl-Z 할 때 발생하는 중지 시그널	중지
SIGSYS	유효하지 않은 시스템 호출	종료(코어 덤프)
SIGTERM	잡을 수 있는 프로세스 종료 시그널	종료
SIGTTIN	후면 프로세스가 제어 터미널을 읽기	중지
SIGTTOU	후면 프로세스가 제어 터미널에 쓰기	중지
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

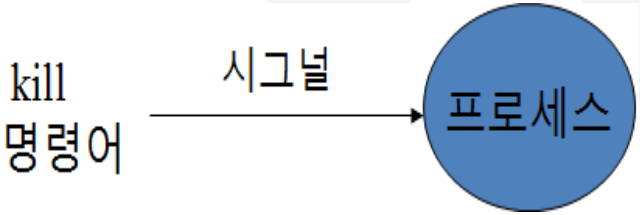
시그널 보내기 : kill 명령어

$ kill [-시그널] 프로세스번호

$ kill [-시그널] %작업번호

프로세스 번호(혹은 작업 번호)로 지정된 프로세스에 원하는 시그널을 보낸다. 시그널을 명시하지 않으면 SIGTERM 시그널을 보내 해당 프로세스를 강제종료

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



시그널 보내기 : kill 명령어

- 종료 시그널 보내기

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

- 다른 시그널 보내기

```
$ 명령어
[1] 1234
$ kill -STOP 1234
[1] + Suspended (signal) 명령어
$ kill -CONT 1234
```