

운영체제론 실습 5주차

CPS LAB

프로세스 목록 나타내는 모듈 프로그래밍

목차

1. task struct

- ① 프로세스 정보를 나타내는 구조체 설명

2. 프로세스 목록 출력 프로젝트 설명

- ① 프로젝트 설명
- ② 프로젝트에 필요한 함수
- ③ 프로젝트 결과화면

ps 명령어

- 실행 중인 프로세스들의 정보를 출력하는 명령어

```
os@os-virtual-machine:~$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.3	0.2	225660	9184	?	Ss	22:40	0:14	/sbin/init splash
root	2	0.0	0.0	0	0	?	S	22:40	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	22:40	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	22:40	0:00	[rcu_par_gp]
root	6	0.0	0.0	0	0	?	I<	22:40	0:00	[kworker/0:0H-ev]
root	9	0.0	0.0	0	0	?	I<	22:40	0:00	[mm_percpu_wq]
root	10	0.0	0.0	0	0	?	S	22:40	0:00	[ksoftirqd/0]
root	11	0.0	0.0	0	0	?	I	22:40	0:00	[rcu_sched]
root	12	0.0	0.0	0	0	?	S	22:40	0:00	[migration/0]
root	13	0.0	0.0	0	0	?	S	22:40	0:00	[idle_inject/0]
root	14	0.0	0.0	0	0	?	S	22:40	0:00	[cpuhp/0]
root	15	0.0	0.0	0	0	?	S	22:40	0:00	[cpuhp/1]
root	16	0.0	0.0	0	0	?	S	22:40	0:00	[idle_inject/1]
root	17	0.0	0.0	0	0	?	S	22:40	0:00	[migration/1]
root	18	0.0	0.0	0	0	?	S	22:40	0:00	[ksoftirqd/1]
root	20	0.0	0.0	0	0	?	I<	22:40	0:00	[kworker/1:0H-kb]
root	21	0.0	0.0	0	0	?	S	22:40	0:00	[cpuhp/2]
root	22	0.0	0.0	0	0	?	S	22:40	0:00	[idle_inject/2]
root	23	0.0	0.0	0	0	?	S	22:40	0:00	[migration/2]
root	24	0.0	0.0	0	0	?	S	22:40	0:00	[ksoftirqd/2]
root	26	0.0	0.0	0	0	?	I<	22:40	0:00	[kworker/2:0H-ev]
root	27	0.0	0.0	0	0	?	S	22:40	0:00	[cpuhp/3]
root	28	0.0	0.0	0	0	?	S	22:40	0:00	[idle_inject/3]
root	29	0.0	0.0	0	0	?	S	22:40	0:00	[migration/3]
root	30	0.0	0.0	0	0	?	S	22:40	0:00	[ksoftirqd/3]
root	32	0.0	0.0	0	0	?	I<	22:40	0:00	[kworker/3:0H-ev]
root	33	0.0	0.0	0	0	?	S	22:40	0:00	[kdevtmpfs]
root	34	0.0	0.0	0	0	?	I<	22:40	0:00	[netns]
root	35	0.0	0.0	0	0	?	S	22:40	0:00	[rcu_tasks_rude_]
root	36	0.0	0.0	0	0	?	S	22:40	0:00	[rcu_tasks_trace]
root	37	0.0	0.0	0	0	?	S	22:40	0:00	[kauditd]
root	39	0.0	0.0	0	0	?	S	22:40	0:00	[khungtaskd]
root	40	0.0	0.0	0	0	?	S	22:40	0:00	[oom_reaper]
root	41	0.0	0.0	0	0	?	I<	22:40	0:00	[writeback]
root	42	0.0	0.0	0	0	?	S	22:40	0:00	[kcompactd0]

CHAPTER 1.

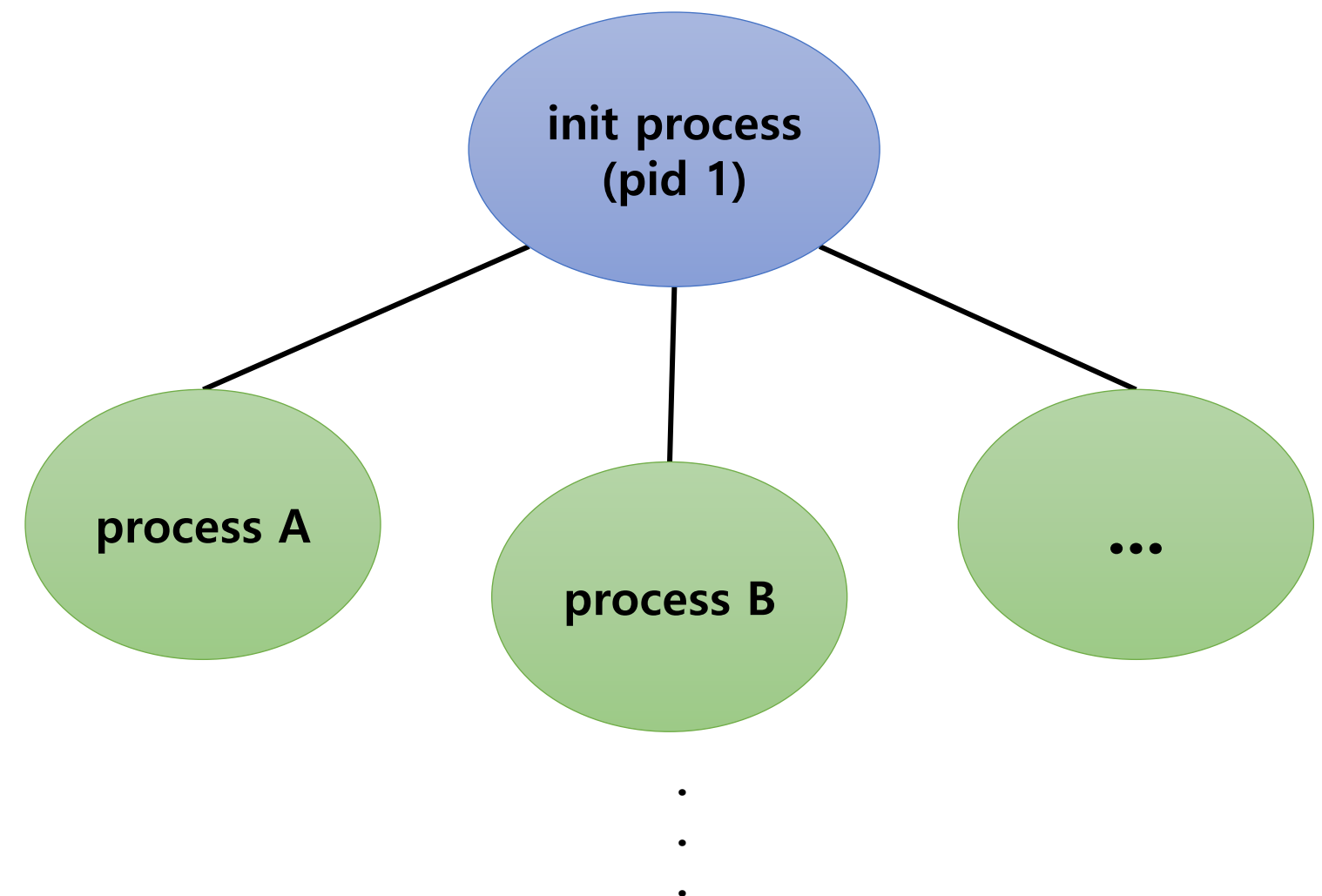
task struct

- 프로세스 정보를 나타내는 구조체 설명

task_struct (1)

- 프로세스가 생기면서 이를 관리하기 위해 같이 만들어지는 구조체
- 프로세스에 관한 모든 정보를 보관하는 프로세스 서술자
 - 사용중인 파일, 프로세스의 주소 공간, 프로세스 상태 등
- Linux가 fork를 통해서 모든 프로세스를 생성하기 때문에 가장 처음 생기는 프로세스인 init process에서 모든 프로세스들이 만들어짐

- init process와 process A는 부모-자식 관계
- process A와 process B는 형제자매(sibling) 관계



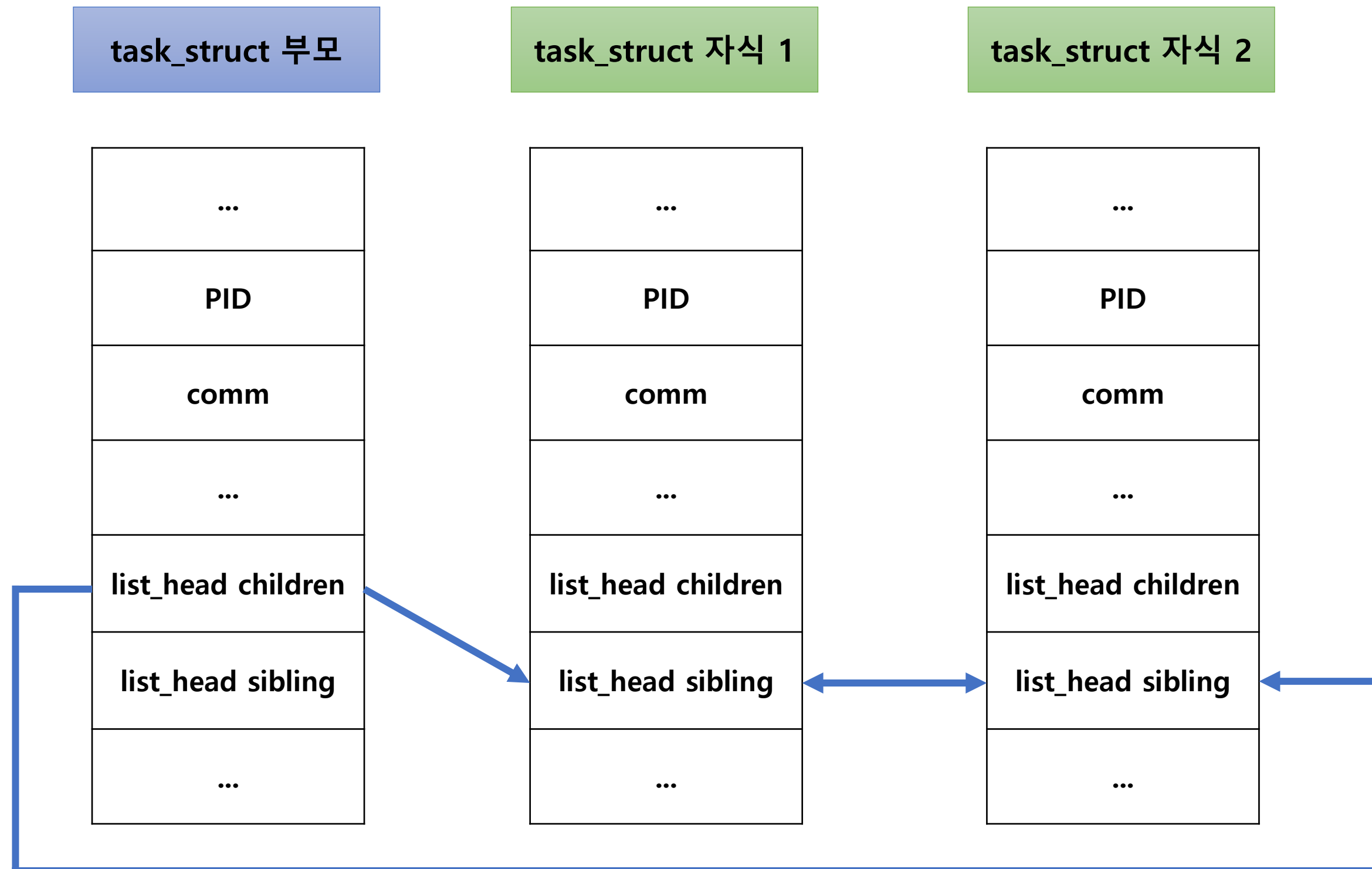
task_struct (2)

- <https://elixir.bootlin.com/linux/v5.10.8/source/include/linux/sched.h#L640>

- 본 실습에 필요한 속성 :

```
struct task_struct {  
  
    /* process id */  
    1 pid_t      pid;  
  
    /* executable name, excluding path. */  
    2 char      comm[TASK_COMM_LEN];  
  
    /* -1 unrunnable, 0 runnable, >0 stopped: */  
    3 volatile long    state;  
  
    /* Children/sibling form the list of natural children */  
    4 struct list_head  children;  
    struct list_head  sibling;  
    struct task_struct *group_leader;  
  
}
```

task_struct (3)



for_each_process

- <https://elixir.bootlin.com/linux/v5.10.8/source/include/linux/sched/signal.h#L589>

```
/ include / linux / sched / signal.h
```

```
541
```

```
542
```

```
#define for_each_process(p) \
```

```
543
```

```
for (p = &init_task ; (p = next_task(p)) != &init_task ; )
```

- init process를 시작으로 프로세스 리스트를 탐색

예제 1 : for_each_process

week5/1_for_each_process/all_pid_with_macro.c

- for_each_process 매크로 함수를 사용해
init process부터 실행된 모든 프로세스들의 command와 pid, 그리고 state 출력

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>

int module_start(void) {
    struct task_struct *task;

    printk(KERN_INFO "Init Module....");

    // Init Process 부터 실행된 모든 프로세스 정보 출력
    for_each_process(task) {
        printk("%s[%d] : %ld\n", task->comm, task->pid, task->state);
    }
    return 0;
}

void module_end(void) {
    printk("Module removing...");
}

module_init(module_start);
module_exit(module_end);
```

예제 1 : for_each_process 실행 결과

week5/1_for_each_process/all_pid_with_macro.c

- 모듈 컴파일 후, 설치하고 커널 메시지 확인

```
$ make
$ sudo insmod all_pid_with_macro.ko
$ dmesg
```

```
[ 7656.178611] Init Module....
[ 7656.178617] systemd[1] : 1
[ 7656.178620] kthreadd[2] : 1
[ 7656.178621] rcu_gp[3] : 1026
[ 7656.178623] rcu_par_gp[4] : 1026
[ 7656.178625] kworker/0:0H[6] : 1026
[ 7656.178626] kworker/0:1[7] : 1026
[ 7656.178628] mm_percpu_wq[9] : 1026
[ 7656.178629] ksoftirqd/0[10] : 1
[ 7656.178631] rcu_sched[11] : 1026
[ 7656.178633] migration/0[12] : 1
[ 7656.178634] idle_inject/0[13] : 1
[ 7656.178636] cpuhp/0[14] : 1
[ 7656.178638] cpuhp/1[15] : 1
[ 7656.178639] idle_inject/1[16] : 1
[ 7656.178641] migration/1[17] : 1
[ 7656.178642] ksoftirqd/1[18] : 1
[ 7656.178644] kworker/1:0H[20] : 1026
[ 7656.178646] cpuhp/2[21] : 1
```

...

```
[ 7656.179765] cups-browsed[3830] : 1
[ 7656.179767] kworker/2:0[4537] : 1026
[ 7656.179768] kworker/3:1[4614] : 1026
[ 7656.179770] kworker/u256:0[4660] : 1026
[ 7656.179771] kworker/u256:2[4685] : 1026
[ 7656.179773] kworker/2:2[4693] : 1026
[ 7656.179774] dhclient[4913] : 1
[ 7656.179775] kworker/3:2[4930] : 1026
[ 7656.179778] gnome-terminal-[5009] : 1
[ 7656.179779] bash[5020] : 1
[ 7656.179780] kworker/0:0[5308] : 1026
[ 7656.179782] kworker/1:8[5313] : 1026
[ 7656.179783] kworker/1:9[5314] : 1026
[ 7656.179784] kworker/3:0[6417] : 1026
[ 7656.179785] kworker/u256:1[6420] : 1026
[ 7656.179787] sudo[6433] : 1
[ 7656.179788] insmod[6434] : 0
```

예제 2 : process의 부모-자식 관계 파악해보기

week5/2_parent_child_relationship/parent_child.c

- 어떤 프로세스가 어떤 자식 프로세스를 생성했는지 출력
(부모 및 자식 프로세스의 pid, command 출력)

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>

int module_start(void) {
    struct task_struct *task;
    struct task_struct *child;
    struct list_head *list;

    printk("INSTALL: parent_child");
    // 1) init process로 시작해서 모든 프로세스를 탐색한다.
    for_each_process(task) {
        // 2) 해당 프로세스의 task 구조체 정보를 출력한다.
        printk("\n %4d task %s\n children: ", task->pid, task->comm);
        // 3) 해당 task의 자식 head를 가져온다.
        list_for_each(list, &task->children) {
            // 4) 자식 head를 기준으로 형제 관계(sibling)에 있는 모든 자식들을 불러온다.
            child = list_entry(list, struct task_struct, sibling);
            // 5) 자식의 정보를 출력한다.
            printk(" %4d %s", child->pid, child->comm);

            /* 결과적으로 task의 모든 자식들이 출력된다. */
        }
    }
    return 0;
}

void module_end(void) {
    printk("REMOVE: parent_child");
}

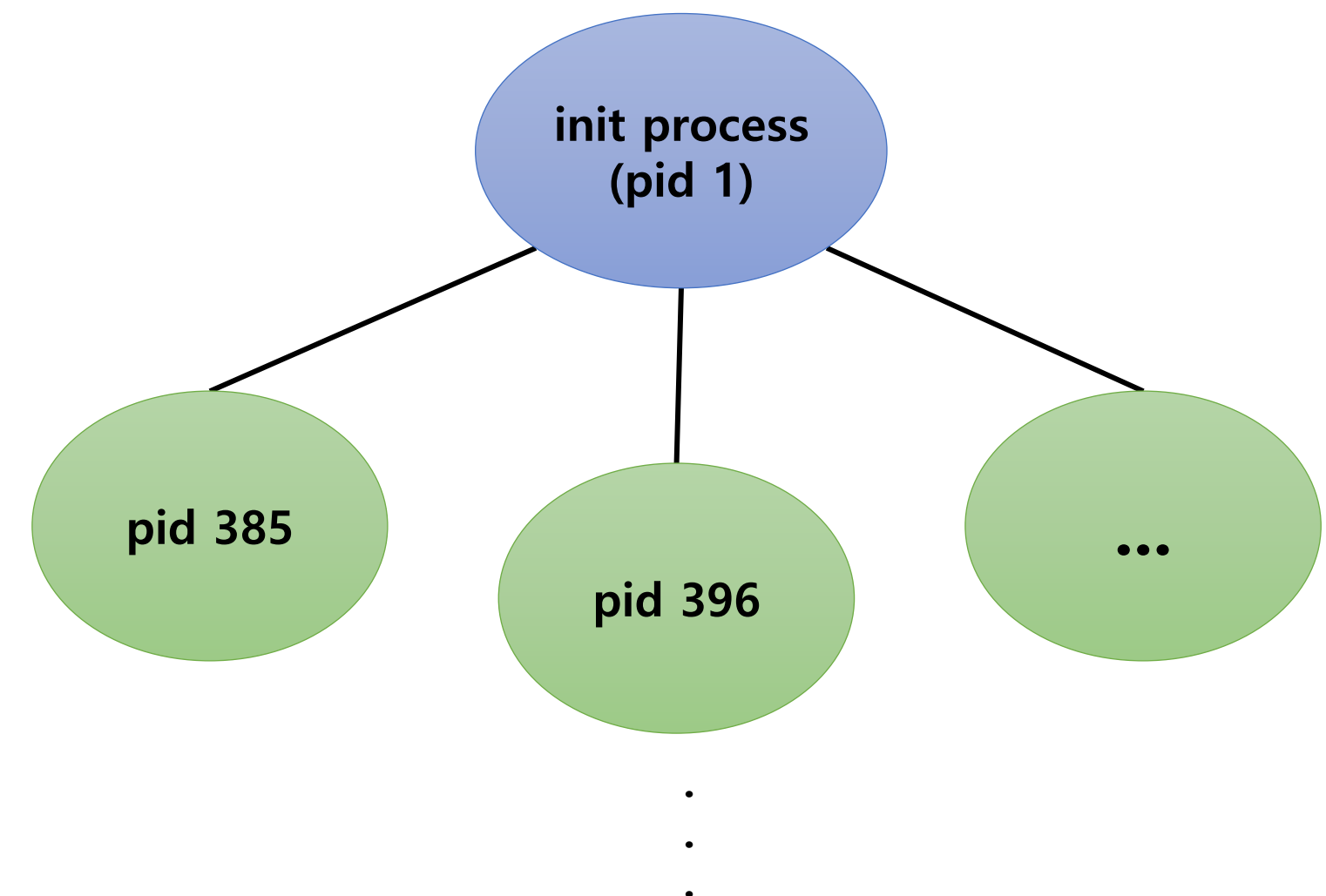
module_init(module_start);
module_exit(module_end);
```

예제 2 : process의 부모-자식 관계 파악해보기

week5/2_parent_child_relationship/parent_child.c

- 어떤 프로세스에 의해 어떤 자식 프로세스가 생성되었는지 확인
- e.g) systemd에 의해 init process (pid 1)이 실행되었고,
pid 385, 396, ... 등 많은 프로세스들이 init process의 자식 프로세스로 생성됨을 확인

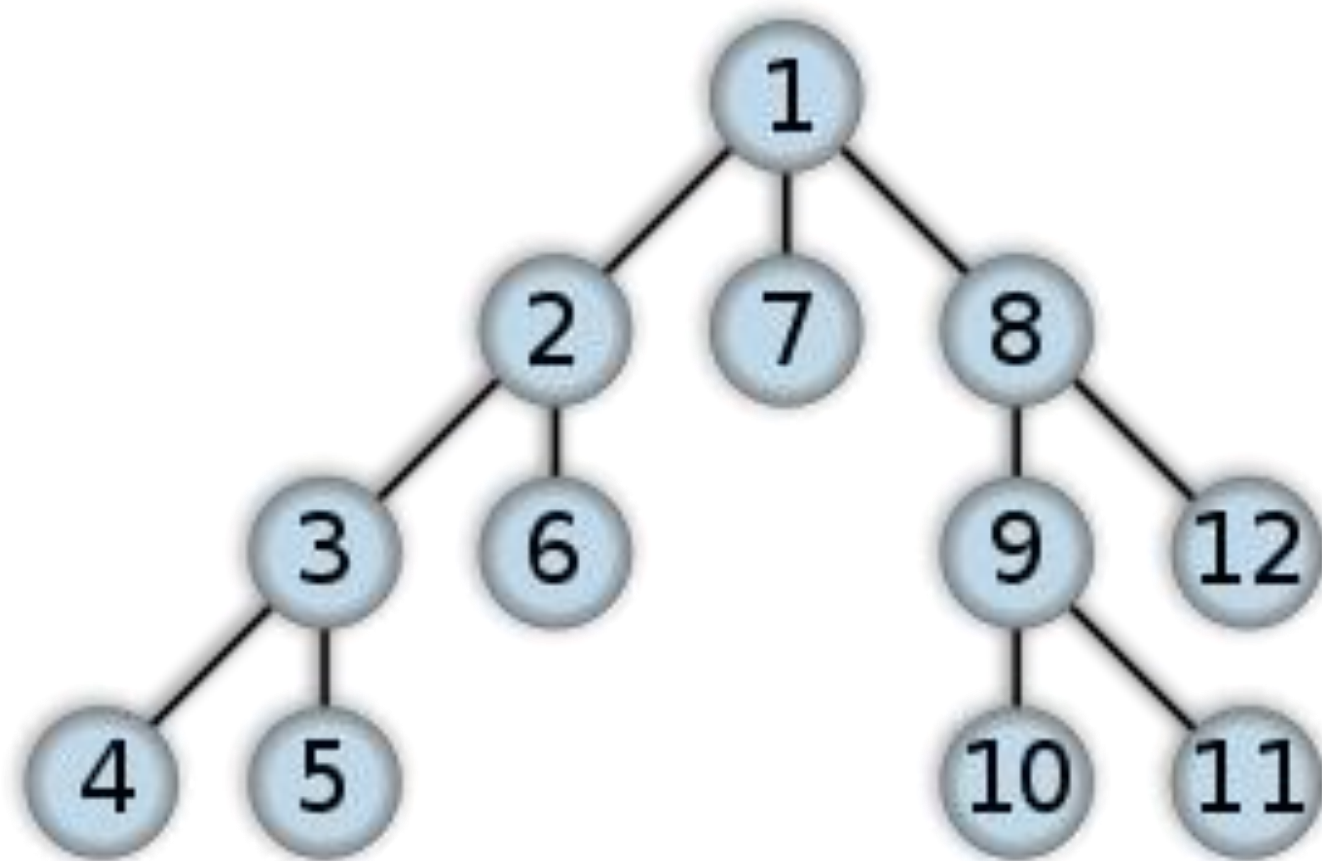
```
[ 2378.542729] INSTALL: parent_child
[ 2378.542732]
[ 2378.542733]     1 task systemd
[ 2378.542734]     children:
[ 2378.542735]     385 systemd-journal
[ 2378.542736]     396 systemd-udevd
[ 2378.542737]     645 systemd-resolve
[ 2378.542738]     651 systemd-timesyn
[ 2378.542739]     776 udiskd
[ 2378.542740]     784 irqbalance
[ 2378.542741]     786 dbus-daemon
[ 2378.542742]     815 rsyslogd
[ 2378.542743]     821 NetworkManager
[ 2378.542744]     822 cron
[ 2378.542745]     824 acpid
[ 2378.542746]     827 accounts-daemon
[ 2378.542747]     829 systemd-logind
[ 2378.542748]     830 ModemManager
[ 2378.542749]     832 avahi-daemon
[ 2378.542750]     838 networkd-dispat
[ 2378.542751]     839 wpa_supplicant
[ 2378.542752]     873 polkitd
[ 2378.542753]     893 unattended-upgr
[ 2378.542754]     909 gdm3
[ 2378.542755]     947 systemd
[ 2378.542756]     961 gnome-keyring-d
```



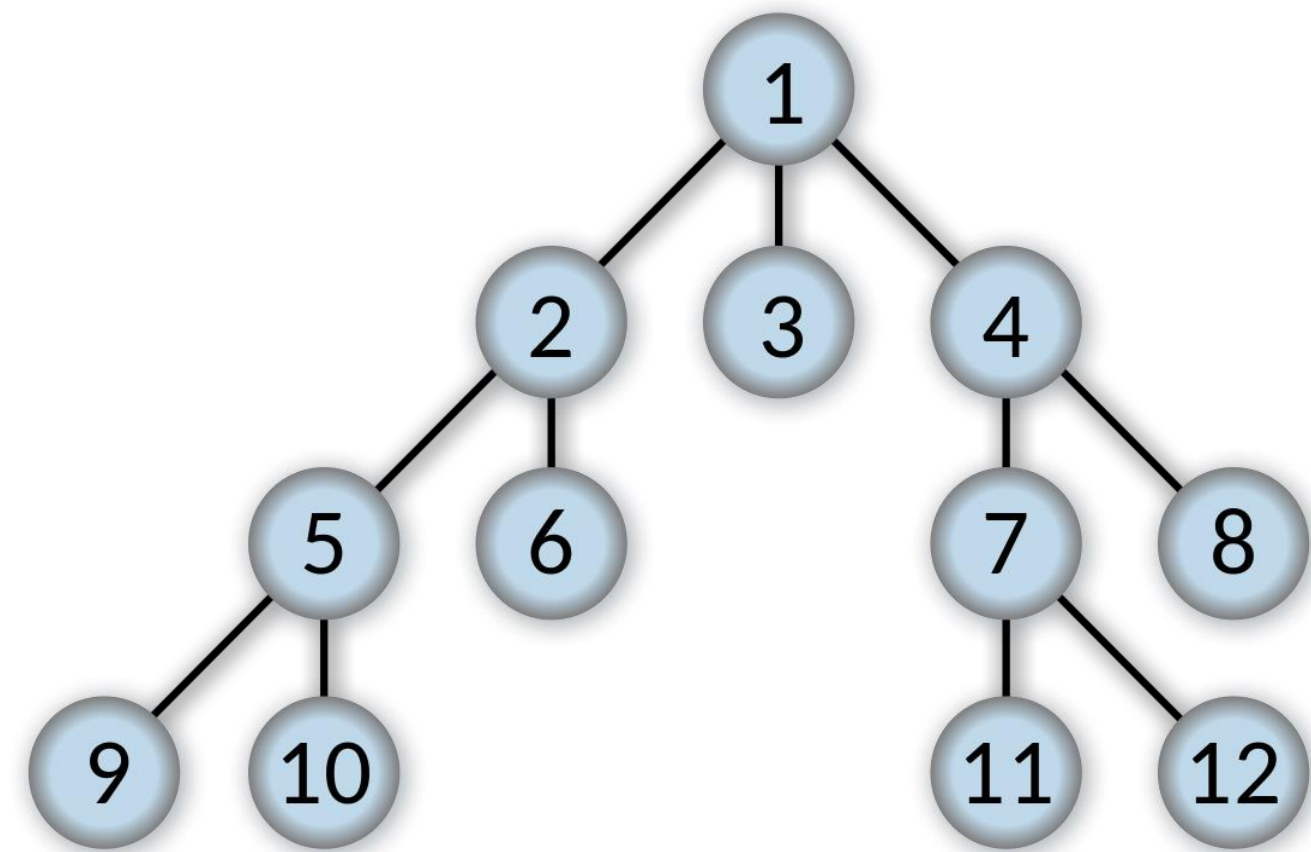
Depth-First Search

- 깊이를 우선 탐색하는 알고리즘
- 비교 대상으로는 너비 우선 탐색이 있음

Depth-First Search



Breadth-First Search



Depth-First Search

- PSEUDO CODE

```
dfs(i) {  
    print information of i  
    for each  $s \in \text{siblings}$  of  $\text{child}$  of i  
        dfs(s);  
}  
  
init() {  
    initial task t  
    dfs(t)  
}
```

- 구현 시 필요한 함수와 속성

- list_for_each
or
- list_for_each_entry
- task_struct -> children
- task_struct -> sibling

PSEUDO CODE 는 PSEUDO CODE 일 뿐,
각 함수에서 요구하는 타입은 이해를 바탕으로
구현해야 함

CHAPTER 2.

프로세스의
pid 데이터를
출력하는 모듈

- 프로젝트 설명
- 프로젝트에 필요한 함수
- 프로젝트 결과화면

프로젝트 : ps lite

week5/ps_lite/list_tasks_dfs.c

- 목표 : 기존 ps 명령의 lite 버전 구현
 - init process부터 모든 프로세스의 pid 데이터를 출력하는 모듈 구현

```
/* SOLUTION 1 : list_for_each 사용할 경우 */
list_for_each(/* TODO 1. 매개변수 채우기 */) {
    /* TODO 2. 빈칸 채우기 */
    dfs(child);
}

/* SOLUTION 2 : list_for_each_entry 사용할 경우 */
/* TODO 1. task의 children 주소 값을 list 포인터에 할당 */
list_for_each_entry(/* TODO 2. 매개변수 채우기 */) {
    dfs(child);
}
}

static int __init list_task_init(void) {

    struct task_struct *init_task;

    /* SOLUTION 1 : pid를 통해 프로세스를 불러오기 */

    /* TODO 1. init 프로세스의 pid 구조체 가져오기 (find_get_pid 사용) */

    /* TODO 2. 가져온 pid를 통해 해당 프로세스의
    task_struct 구조체 가져오기 (pid_task 사용) */

    printk(KERN_INFO "INSTALL: list_tasks_dfs\n");

    /* SOLUTION 2 : 현재 수행중인 프로세스의 조상을 찾아가기 */

    /* TODO 1. sched.h로부터 init_task로 현재 실행중인 프로세스 가져오기 */

    // pid가 1이 될 때 빠져나오는 조건이다.
    while (init_task->pid != 1){
        /* TODO 2. init 프로세스(pid = 1)를 가리킬때까지 부모 프로세스 탐색하기 */
    }

    dfs(init_task); // 깊이 우선 탐색을 통해 init 프로세스를 기점으로 모든
```


프로젝트 : ps lite 결과 예시

```
os@os-virtual-machine:~/os-week/week5/ps_lite_answer$ dmesg
[ 8032.498197] INSTALL: list_tasks_dfs
[ 8032.498212] COMM: systemd                STATE: 1      PID: 1
[ 8032.498216] COMM: systemd-journal         STATE: 1      PID: 383
[ 8032.498218] COMM: systemd-udevd          STATE: 1      PID: 397
[ 8032.498219] COMM: systemd-resolve        STATE: 1      PID: 658
[ 8032.498221] COMM: systemd-timesyn       STATE: 1      PID: 659
[ 8032.498224] COMM: rsyslogd              STATE: 1      PID: 800
[ 8032.498225] COMM: networkd-dispat       STATE: 1      PID: 803
[ 8032.498227] COMM: udisksd              STATE: 1      PID: 806
[ 8032.498229] COMM: bluetoothd           STATE: 1      PID: 808
[ 8032.498231] COMM: ModemManager          STATE: 1      PID: 811
[ 8032.498233] COMM: systemd-logind        STATE: 1      PID: 812
[ 8032.498234] COMM: cron                  STATE: 1      PID: 816
[ 8032.498236] COMM: avahi-daemon           STATE: 1      PID: 826
[ 8032.498238] COMM: avahi-daemon          STATE: 1      PID: 836
[ 8032.498240] COMM: acpid                 STATE: 1      PID: 830
[ 8032.498242] COMM: irqbalance            STATE: 1      PID: 831
[ 8032.498244] COMM: accounts-daemon       STATE: 1      PID: 845
[ 8032.498247] COMM: dbus-daemon           STATE: 1      PID: 862
[ 8032.498249] COMM: wpa_supplicant         STATE: 1      PID: 875
[ 8032.498250] COMM: NetworkManager        STATE: 1      PID: 876
[ 8032.498254] COMM: dhclient              STATE: 1      PID: 4913
[ 8032.498255] COMM: snapd                 STATE: 1      PID: 878
[ 8032.498258] COMM: polkitd               STATE: 1      PID: 893
[ 8032.498260] COMM: unattended-upgr       STATE: 1      PID: 911
[ 8032.498262] COMM: gdm3                  STATE: 1      PID: 920
[ 8032.498264] COMM: gdm-session-wor       STATE: 1      PID: 956
[ 8032.498265] COMM: gdm-x-session         STATE: 1      PID: 985
[ 8032.498267] COMM: Xorg                  STATE: 1      PID: 987
[ 8032.498269] COMM: gnome-session-b       STATE: 1      PID: 1157
[ 8032.498271] COMM: ssh-agent              STATE: 1      PID: 1252
[ 8032.498273] COMM: gnome-shell           STATE: 1      PID: 1328
[ 8032.498277] COMM: ibus-daemon           STATE: 1      PID: 1631
[ 8032.498278] COMM: ibus-dconf            STATE: 1      PID: 1635
```

감사합니다.

CPS LAB

Lim Jiseoup

jseoup@hanyang.ac.kr