시스템 프로그래밍 Lab6 Report

경영학과 2017-15108
박지상

1. 실행결과 – lsb_relase, uname

```
park@park-Standard-PC-Q35-ICH9-2009:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.4 LTS
Release:        20.04
Codename:       focal
park@park-Standard-PC-Q35-ICH9-2009:~$ uname -ar
Linux park-Standard-PC-Q35-ICH9-2009 5.13.0-44-generic #49~20.04.1-Ubuntu SMP Wed May 18 18:44:28 UTC 2022 x86_64 x86_64 x86_64
GNU/Linux
```

1. 실행결과 – 1) ptree

```
root@park-Standard-PC-Q35-ICH9-2009:/home/park/System-Programming/HW6/paddr# cd /sys/kernel/debug/ptree
root@park-Standard-PC-Q35-ICH9-2009:/sys/kernel/debug/ptree# ps
    PID TTY          TIME CMD
   2765 pts/0    00:00:00 sudo
   2777 pts/0    00:00:00 su
   2782 pts/0    00:00:00 bash
   6921 pts/0    00:00:00 ps
root@park-Standard-PC-Q35-ICH9-2009:/sys/kernel/debug/ptree# echo 2765 > input
root@park-Standard-PC-Q35-ICH9-2009:/sys/kernel/debug/ptree# cat ptree
systemd (1)
sh (1089)
node (1104)
node (1185)
bash (2746)
sudo (2765)
root@park-Standard-PC-Q35-ICH9-2009:/sys/kernel/debug/ptree# pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─acpid
        ├─avahi-daemon───avahi-daemon
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd───2*[dbus]
        ├─dbus-daemon
        ├─2*[kerneloops]
        ├─login───bash
        ├─networkd-dispat
        ├─polkitd───2*[{polkitd}]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─rtkit-daemon───2*[{rtkit-daemon}]
        ├─sh───node─┬─node─┬─bash───sudo───su───bash───pstree
        │           │      └─11*[{node}]
```

1. 실행결과 – 2) paddr

```
park@park-Standard-PC-Q35-ICH9-2009:~/System-Programming/HW6/paddr$ sudo ./app
[sudo] password for park:
[TEST CASE]    PASS
```

2. 구현 방법 : ptree – (1) dbfs_module_init()

```c
static const struct file_operations dbfs_fops = {
        .write = write_pid_to_input,
};

static int __init dbfs_module_init(void)
{
        // Implement init module code

        dir = debugfs_create_dir("ptree", NULL);

        if (!dir) {
                printk("Cannot create ptree dir\n");
                return -1;
        }
        // struct dentry* debugfs_create_file(const char *name, umode_t mode, struct dentry *parent, void *data, const struct file_operations *fops)
        // S_IWUSR: 쓰기 권한
        inputdir = debugfs_create_file("input", S_IWUSR, dir, NULL, &dbfs_fops);

        // Blob (Binary Large Object) 메모리 할당
        // kmalloc: kernel malloc
        blob = (struct debugfs_blob_wrapper *)kmalloc(sizeof(struct debugfs_blob_wrapper), GFP_KERNEL);

        // struct dentry* debufgs_create_blob(const char *name, umode_t mode, struct dentry *parent, struct debugfs_blob_wrapper *blob);
        // S_IRUSR: 읽기 권한
        ptreedir = debugfs_create_blob("ptree", S_IRUSR, dir, blob); // Find suitable debugfs API

        printk("dbfs_ptree module initialize done\n");

        return 0;
}
```

debugfs를 활용한 file 및 blob 생성

2. 구현 방법 : ptree – (2) write_pid_to_input()

```c
static ssize_t write_pid_to_input(struct file *fp,
                                  const char __user *user_buffer,
                                  size_t length,
                                  loff_t *position)
{
        pid_t input_pid;
        int size = 0;
        struct ptree_node *head;
        struct ptree_node *node;
        struct ptree_node *temp;

        // user_buffer에 담긴 pid를 읽어들이고 input_pid에 입력
        sscanf(user_buffer, "%u", &input_pid);

        // Find task_struct using input_pid. Hint: pid_task
        curr = pid_task(find_get_pid(input_pid), PIDTYPE_PID);

// Tracing process tree from input_pid to init(1) process
head = (struct ptree_node *) kmalloc(sizeof(struct ptree_node), GFP_KERNEL);

while(1){
        // 새로운 ptree_node를 생성해 head에 삽입
        node = (struct ptree_node *) kmalloc(sizeof(struct ptree_node), GFP_KERNEL);

        node -> process_command = curr -> comm;
        node -> process_id = curr -> pid;
        node -> next = head -> next;
        head -> next = node;

        // curr 업데이트
        if(curr -> pid == 1) break;
        curr = curr -> parent;
}
```

```c
struct ptree_node{
        char* process_command;
        int process_id;
        struct ptree_node *next;
};
```

- **sscanf로 input으로 들어온 pid를 읽음**
- **pid로 task_struct를 찾음**
- **task_struct의 linked list를 생성함**
  - 입력받은 프로세스에서 부모 프로세스를 반복해서 올라가면서 root process에 다다를 때까지 반복

2. 구현 방법 : ptree – (2) write_pid_to_input()

```
// Make Output Format string: process_command (process_id)
logs = kmalloc(sizeof(char) * 101010, GFP_KERNEL);

temp = head -> next;

while(temp != NULL){
        // snprintf: https://www.ibm.com/docs/ko/i/7.3?topic=functions-snprintf-print-formatted-data-buffer 참고
        size += snprintf(logs + size, 64, "%s (%d)\n", temp->process_command, temp->process_id);
        temp = temp -> next;
}
blob->data = logs;
blob->size = (unsigned long)strlen(logs);

free_list(head);

return length;
```

```
static void free_list(struct ptree_node* head){
        struct ptree_node *temp;
        while(head != NULL)
        {
                temp = head;
                head = head->next;
                kfree(temp);
        }
}
```

- Linked List를 순차적으로 순회하면서 process_command (process_id)와 같은 형태로 출력한다.
- Linked List를 free한다.

2. 구현 방법 : ptree – (3) dbfs_module_exit()

```
static void __exit dbfs_module_exit(void)
{
        // Implement exit module code

        // // Free kmalloc'd object
        kfree(logs);
        kfree(blob);

        // struct dentry* debufgs_remove_recursive(struct dentry *dentry)
        debugfs_remove_recursive(dir);

        printk("dbfs_ptree module exit\n");
}
```

- kalloc된 메모리를 kfree()한다
- debugfs_remove_recursive() 실행한다.

2. 구현 방법 : paddr – (1) dbfs_module_init()

```
static const struct file_operations dbfs_fops = {
        // Mapping file operations with your functions
        .read = read_output,
};

static int __init dbfs_module_init(void)
{
        // Implement init module

        dir = debugfs_create_dir("paddr", NULL);
        if (!dir) {
                printk("Cannot create paddr dir\n");
                return -1;
        }

        // Fill in the arguments below
        // struct dentry* debugfs_create_file(const char *name, umode_t mode, struct dentry *parent, void *data, const struct file_operations *fops)
        // S_IRUSR: 읽기 권한
        output = debugfs_create_file("output", S_IRUSR, dir , NULL, &dbfs_fops);

        printk("dbfs_paddr module initialize done\n");

        return 0;
}
```

debugfs를 활용한 file 생성

2. 구현 방법 : paddr – (2) read_output()

```c
static ssize_t read_output(struct file *fp,
                           char __user *user_buffer,
                           size_t length,
                           loff_t *position)
{
        // Implement read file operation
        struct packet *pckt;
        int ret;

        // Page Table
        pgd_t *pgd; // Top level page table entry
        p4d_t *p4d; // Next level page table entry
        pud_t *pud;
        pmd_t *pmd;
        pte_t *pte; // Pate Table Entry

        pckt = (struct packet*) user_buffer;

        // Get pid of app & virtual address
        task = pid_task(find_get_pid(pckt -> pid), PIDTYPE_PID);
```

```c
static struct dentry *dir, *output;
static struct task_struct *task;

struct packet {
        pid_t pid;
        unsigned long vaddr;
        unsigned long paddr;
};
```

- packet을 읽어들임
- packet은 app의 process id와 physical / virtual address를 담고 있다.
- process id를 기반으로 task_struct를 얻는다.

2. 구현 방법 : paddr – (2) read_output()

```
// Get PHysical Address
pgd = pgd_offset(task->mm, pckt->vaddr);
p4d = p4d_offset(pgd, pckt->vaddr);
pud = pud_offset(p4d, pckt->vaddr);
pmd = pmd_offset(pud, pckt->vaddr);
pte = pte_offset_kernel(pmd, pckt->vaddr);

// Offset: 12bit
pckt->paddr = (pte_val(*pte) & 0xFFFFFFFFF000) | (pckt->vaddr & 0x0000000000000FFF);
pte_unmap(pte);

// Returns Physical address
// int copy_to_user(void __user* to, const void* from, unsigned long n)
ret = copy_to_user(user_buffer, pckt, sizeof(struct packet));

return length;
```

5 Level Page Table을 따라가면서
최종적으로 Physical Address를 획득하고, copy_to_user를 통해 반환한다.

2. 구현 방법 : paddr – (3) dbfs_module_exit()

```
static void __exit dbfs_module_exit(void)
{
        // Implement exit module
        debugfs_remove_recursive(dir);

        printk("dbfs_paddr module exit\n");
}
```

debugfs_remove_recursive() 실행한다.

3. 어려웠던 점

- Kernel programming이 처음이라, 그에 따른 코드의 차이를 이해하는데 어려움을 겪었다. (kalloc, kfree 등)
- Debugfs Module을 직접 뜯어보면서 적합한 메소드를 탐색하고, 파라미터를 선정하는 데에 어려움을 겪었다.
- Page Walk API 내에서 필요한 메소드를 파악하고, type을 일치시키는데 어려움을 겪었다.

4. 새롭게 배운 점

- Kernel programming는 일반 user space programming과 다르다.
- Kernel 상에 compile한 insmod를 통해 module로 올려서 사용할 수 있다.
- Debugf를 통해 kernel programming을 하면서 user space와 소통할 수 있다.