곽민석 - 20183380
2023년 9월 19일 화요일

운영체제

커널 설치와 ps 명령어 구현

1. 리눅스 커널(linux-5.15.120) 컴파일 및 설치

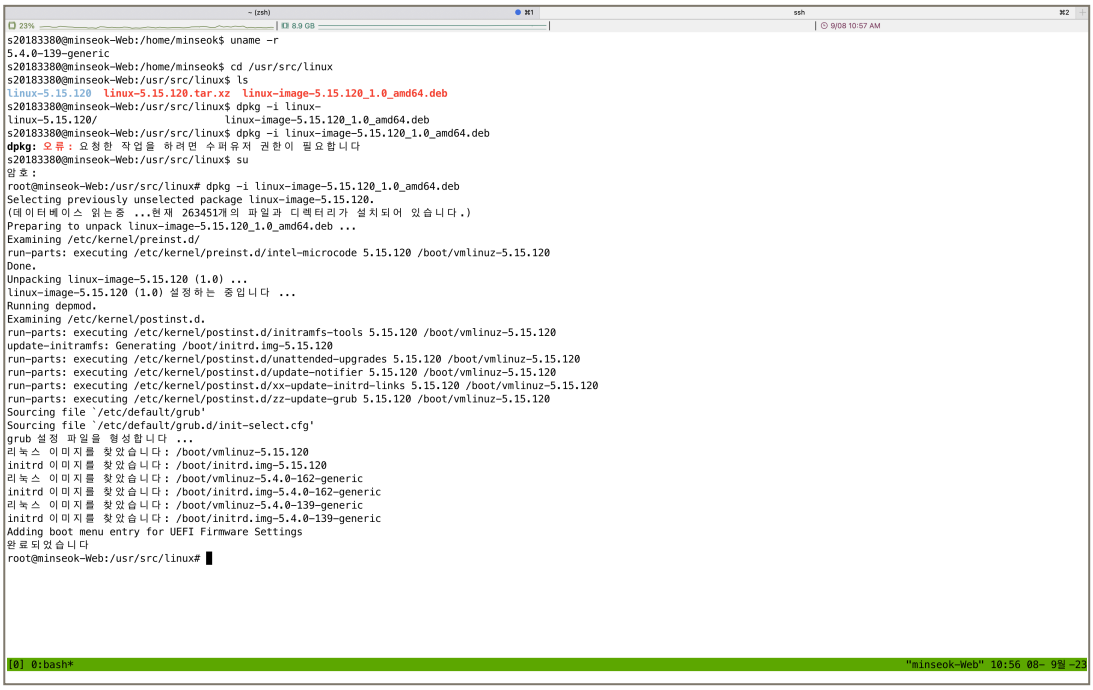

Fig 1. 초기 설치된 커널 버전과 컴파일된 커널 설치.

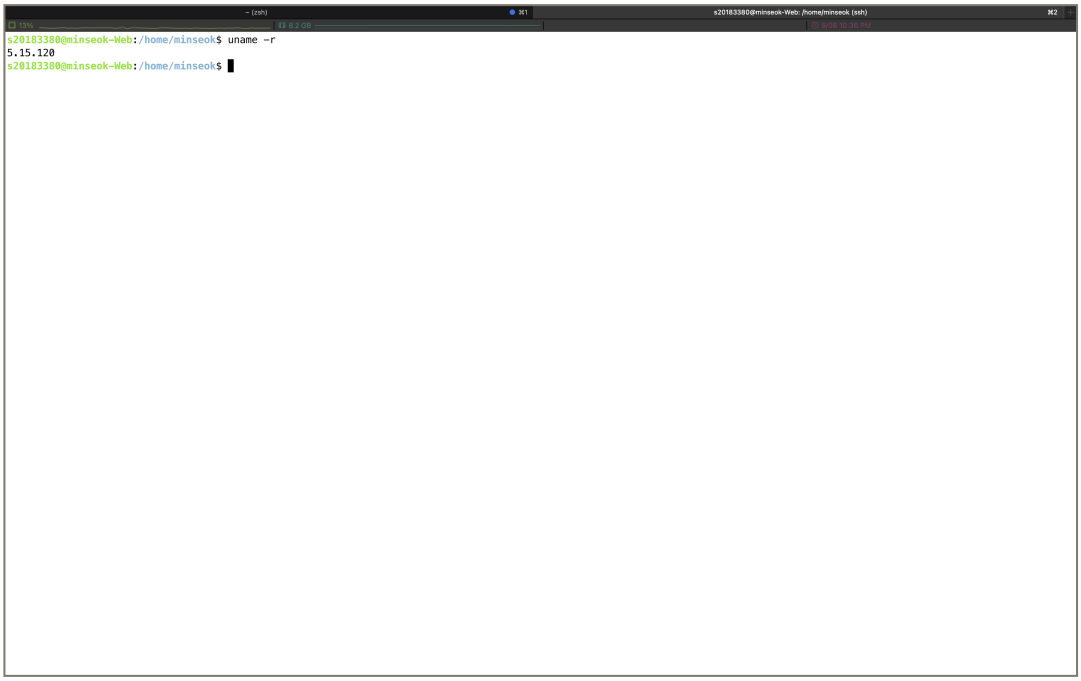

Fig 2. 커널 변경 및 재부팅 후 설치된 커널 확인.

커널 컴파일과 설치는 참고자료를 참조하여 진행하였다.

다만, “scripts/config” 명령어를 이용한 키 관련, 명령어를 통해 끄는것이 불가능하여, “.config” 파일에서 수동으로 옵션을 제거하였다. 또한 가상머신이 아닌 실기기에서 커널을 설치할 때 인증서 문제로 인해 BIOS에서 Secure Boot 옵션을 꺼주어야 정상 부팅이 가능했다.

2. newps 명령어 구현

A. newps 컴파일 및 사용 방법

1. 프로젝트 폴더에서 “make” 명령어를 이용하여 컴파일 실행.
2. 컴파일된 “newps”를 실행. 가능한 옵션은 다음과 같다.
   2.1.default(별도 입력 불필요): 현재 실행되어있는 TTY의 자식 프로세스만 출력.
   2.2.tty: 현재 TTY에서 실행한 프로세스만 출력.
   2.3.uid: 현재 UID와 동일한 UID를 갖는 프로세스만 출력.
   2.4.all: “/proc” 내의 모든 프로세스 출력.
3. 실행의 예는 다음과 같다.
   3.1. ./newps (tty | uid | all)

B. 소스 설명에 앞선 배경지식

A. /proc/PID/ 디렉토리 구조

리눅스에서는 프로세스에 대해 /proc/PID 폴더에 해당 프로세스의 정보를 저장해놓고 관리하게 된다. 각각의 파일마다 다른 정보를 가지고 있는데, 이번 프로젝트에서는 stat과 status 파일의 정보를 이용하였다. 또한 해당 PID의 프로세스가 어떤 TTY를 이용하고 있는지 알기 위해 입출력 /proc/PID/fd/0 파일을 이용하였다.

B. /proc/PID/stat 파일 구조

해당 파일에는 공백을 기준으로 각각의 프로세스 세부 정보를 기입해놓은 파일이다. 파일 내용은 다음과 같다.

```
123 (kstrp) I 2 0 0 0 -1 69238880 0 0 0 0 0 0 0 0 0 -20 1 0 89
0 0 18446744073709551615 0 0 0 0 0 0 0 2147483647 0 0 0 0 17 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```

File 1. stat 파일 샘플.

각 자리마다 정보가 다른데, 예를들어 첫 번째에 기입이 되어있는 정보는 PID로, 123번 PID의 프로세스 정보이다. 해당 프로젝트에서는 부모의 PID, TTY 정보 등을 가져오기 위해 사용하였다.

C. /proc/PID/status 파일 구조

해당 파일은 /proc/PID/stat과 /proc/PID/statm의 정보를 사람이 읽기 쉽게 만들어놓은 파일이다. 해당 내용 일부는 다음과 같이 나온다.

```
Name:  kstrp
Umask: 0000
State: I (idle)
Tgid:  123
Ngid:  0
Pid:   123
PPid:  2
TracerPid:  0
```

… 생략 …

File 2. status 파일 샘플.

해당 프로젝트에서는 특정 프로세스에 대해 UID 정보를 수집하기 위해 사용하였다.

D. /proc/PID/fd/0

해당 파일은 프로세스가 어떤 입출력에서 실행된것인지 저장해둔 symbolic link 파일이다. 이 파일이 참조하고 있는 입출력 링크를 참조해 어떤 TTY를 사용하고 있는지 알 수 있다.

C. 소스 설명

C.A.구조체

`typedef struct pid_info PID_STAT_INFO`

```c
typedef struct pid_info {
    char info[52][128];
} PID_STAT_INFO;
```

"/proc/PID/stat"의 정보를 받아 문자열 배열로 저장하여 이용가능하도록 만들어놓은 구조체이다. "stat" 파일 내에는 총 52개의 정보가 존재하여, 이를 모두 담을수 있는 문자열 배열 변수 "info"를 가지고 있다.

```
typedef struct options ARG_OPTIONS
```

```
typedef struct options {
    int is_checked;
    int is_uid;
    int is_tty;
    int is_all;
} ARG_OPTIONS;
```

사용자가 명령어 실행 시 옵션을 입력한 경우 정보를 가지고있기 위한 구조체이다. 옵션이 들어오게 되면 각 옵션에 맞는 변수는 1로 표기하고, 옵션이 1개 이상 입력됨을 나타내기 위해 “is_checked” 변수 또한 1로 표기한다.

C.B.함수

해당 프로젝트에서 제작된 함수 중 출력과 옵션을 맞추어주기 위한 함수 외의 실질적으로 동작을 위한 함수에 대해 설명을 하였다.

```
double tick2sec(long tick, char *time)
```

```
double tick2sec(long tick, char *time) {
    return (double)atoi(time) / tick;
}
```

프로세스가 사용하고 있는 틱단위의 시간을 사람이 알수있는 시간으로 변환하는 함수다. 이는 time을 받아 tick으로 나누어 계산한다.

```
char* getTTY(char *pid)
```

```
char* getTTY(char *pid) {
    char tty_path[128];
    snprintf(tty_path, sizeof(tty_path), "/proc/%s/fd/0", pid);

    static char tty_name[128];
    ssize_t tty_name_length = readlink(tty_path, tty_name, sizeof(tty_name) -
1);

    // 가져온 TTY의 정보가 없는 경우.
    if (tty_name_length == -1) {
        static char tty_err[] = "NONE";
        return tty_err;
    }
    tty_name[tty_name_length] = '\0';

    return tty_name;
}
```

프로세스의 TTY를 찾아내는 함수이다. 입력된 PID를 기준으로 “/proc/PID/fd/0” 파일의 링크를 읽어, 참조 지점의 이름을 알아낸다. 읽었을 때 아무 정보가 없는 경우 “NONE”으로 반환한다.

```
char* getPIDUID(char *pid)
```

```c
char* getPIDUID(char *pid) {
    static char uid[8] = "NONE";
    char status_path[64];
    snprintf(status_path, sizeof(status_path), "/proc/%s/status", pid);

    // /proc/PID/status 파일 열기.
    FILE *status_file = fopen(status_path, "r");
    if (status_file == NULL) {
        return uid;
    }

    char line[64];

    // 파일에서 UID 정보 찾기.
    while (fgets(line, sizeof(line), status_file) != NULL) {
        if (strncmp(line, "Uid:", 4) == 0) {
            sscanf(line, "Uid:\t%s", uid);
            break;
        }
    }

    fclose(status_file);

    return uid;
}
```

프로세스의 UID를 찾아내는 함수이다. 입력된 PID를 기준으로 “/proc/PID/status” 파일을 읽어드려 해당 파일에서 UID 정보를 읽어드린다. 이 정보는 “Uid:”로 시작되는 줄에 기입되어 있다. 만약, 파일 읽기가 실패하는 경우 NONE으로 반환한다.

C.C.main 함수

main 함수의 경우 작동별로 나누어 따로 작성해놓았다. 크게 변수 선언부, 정보 수집, 옵션 판별과 그에따른 기타 설정, 출력 부분이 있다.

```
int main(int argc, char const *argv[]) {
    // 현재 실행시킨 UID 가져오기.
    const uid_t CURRENT_UID     = getuid();
    // 현재 시스템의 초당 클럭 틱 가져오기.
    const long  CLK_TCK         = sysconf(_SC_CLK_TCK);
    // 현재 TTY 가져오기.
    char        *CURRENT_TTY    = ttyname(STDIN_FILENO);

    // /proc 폴더용 DIR 구조체.
    DIR *dir_proc;
    // /proc/PID/stat를 읽기 위한 FILE 구조체.
    FILE *file_stat;
    // /proc/PID/stat의 정보를 담기위한 PID_STAT_INFO 구조체.
    PID_STAT_INFO pid_stat_info;
    // 입력된 옵션을 가져오기 및 저장후 구조체 선언.
    ARG_OPTIONS options = menuSelector(argc, argv);

    // /proc 내의 파일 정보를 가져오기 위한 dirent 구조체.
    struct dirent *dir;

    // /proc/PID/stat의 경로를 가져오기 위한 문자열.
    char path_stat[1024] = "";
    // /proc/PID/stat 내용을 저장하기 위한 문자열.
    char stat_info[1024] = "";
    // TTY가 동일한 부모의 PID를 저장하기 위한 문자열 배열.
    char list_tty_parent_pid[512][8];
    // 부모 TTY 경로를 저장하기 위한 문자열 배열.
    char tty_parent_name[512]       = "";
    int cnt_tty_parent_pid          = 0;

    ...
```

먼저, 변수 선언 부분이다. 해당 부분에서는 newps가 작동하는 동안 지속적으로 사용되는 변수를 선언해놓았다. 각 변수에 대한 쓰임은 변수명 상단에 기록해두었다.

```c
int main(int argc, char const *argv[]) {
    ...
    // proc 경로 설정.
    dir_proc = opendir(PROC_PATH);

    // HEAD 출력.
    printf("    PID        TTY     TIME  CMD\n");

    // /proc 폴더가 열린 경우 작업 진행.
    if(dir_proc) {
        while ((dir = readdir(dir_proc)) != NULL) {

            if(isDigit(dir->d_name)) {
                // /proc/PID/stat 경로에서 stat 파일 읽어오기.
                sprintf(path_stat, PROC_STAT_PATH, dir->d_name);
                file_stat = fopen(path_stat, "r");

                // 파일이 읽히지 않은 경우 넘어가기.
                if (!file_stat) {
                    fclose(file_stat);
                    continue;
                }

                // 파일 내용 불러오기.
                fgets(stat_info, 1024, file_stat);
                fclose(file_stat);

                // 파일 내용을 구조체로 변환.
                pid_stat_info = getPIDInfo(stat_info);
    ...
```

다음 부분에서는 “/proc” 폴더에 있는 폴더를 읽어드린다. 그 후 표기될 정보의 컬럼을 출력한 후 “dir_proc”이 비어있을 때 까지 읽어드려 작업을 진행한다. 이때, PID는 숫자로만 이루어져 있기 때문에, 읽은 폴더명이 숫자만을 가지고있는 폴더에 대해서만 작업한다. 그 후 “/proc/PID/stat” 파일을 읽어드려 구조체에 저장한다.

```c
int main(int argc, char const *argv[]) {
    ...
    if(dir_proc) {
        while ((dir = readdir(dir_proc)) != NULL) {

    ...

                // 옵션이 all이 아닌 경우 다른 요소 고려.
                if(!options.is_all) {
                    // UID가 다른경우 넘어가기.
                    if((atoi(getPIDUID(pid_stat_info.info[0])) != CURRENT_UID)
                        && options.is_uid) continue;

                    // TTY가 다른경우 넘어가기.
                    if(strncmp(getTTY(pid_stat_info.info[0]), CURRENT_TTY, 128)
                        && options.is_tty) continue;

                    // 현재 TTY와 같은지 판별.
                    if(!strncmp(getTTY(pid_stat_info.info[0]), CURRENT_TTY, 128) &&
                    !options.is_checked) {
                        // 같으면 부모로 추가.
                        strcpy(list_tty_parent_pid[cnt_tty_parent_pid++], pid_stat_info.info[0]);

                        // 가장 부모 TTY 저장.
                        if(cnt_tty_parent_pid == 1)
                            strcpy(tty_parent_name, getTTY(pid_stat_info.info[0]));
                    }
                    else if(!options.is_checked) {
                        int is_child = 0;

                        for(int i = 0;i < cnt_tty_parent_pid;i++) {
                            // 부모가 있는경우 자식을 부모 리스트에 추가.
                            if(!strncmp(list_tty_parent_pid[i], pid_stat_info.info[5], 128)) {
                                is_child = 1;
                                strcpy(list_tty_parent_pid[++cnt_tty_parent_pid], pid_stat_info.info[0]);
                                break;
                            }
                        }

                        // 자식이 아닌경우 넘어가기.
                        if(!is_child) continue;
                    }
    ...
```

해당 부분에서는 입력된 옵션에 대해 확인하고, 필터링하게 된다. 아무 옵션이 없는 경우 해당 프로세스의 부모 PID가 현재 실행된 TTY에서 파생된 프로세스인지 확인하고, 맞는경우 출력하게 된다.

그 중 현재 TTY와 동일한 프로세스 중 가장 빨리 실행된 프로세스에 대해 TTY 경로를 저장해 둔다. 그 이후에 들어온 프로세스의 경우 부모 프로세스로 등록하고, 추후 자식 프로세스를 판별할때 사용된다.

만약 TTY가 동일하지 않는 경우 기존에 등록된 부모 PID 중 자신의 부모 PID가 동일한것이 있는지 확인된 경우, 이도 마찬가지로 부모 PID에 등록한 후 출력 단계에 들어간다.

```
int main(int argc, char const *argv[]) {
    ...
    if(dir_proc) {
        while ((dir = readdir(dir_proc)) != NULL) {

    ...

                // PID 출력.
                printf("%7s ", pid_stat_info.info[0]);

                // TTY 출력.
                if(!options.is_checked)
                    printf("%10s ", tty_parent_name);
                else
                    printf("%10s ", getTTY(pid_stat_info.info[0]));

                // utime과 stime을 합산한 TIME 결과 출력.
                double utime = tick2sec(CLK_TCK, pid_stat_info.info[13]);
                double stime = tick2sec(CLK_TCK, pid_stat_info.info[14]);
                printCPUTime(utime + stime);

                // 실행 명령어 출력.
                pid_stat_info.info[1][0] = ' ';
                pid_stat_info.info[1][strlen(pid_stat_info.info[1]) - 1]
                  = '\0';

                printf("%s\n", pid_stat_info.info[1]);
    ...
```

해당 부분에서는 옵션에 따라 가져와진 정보를 출력하게 된다.

“/proc/PID/stat”의 정보는 각 자릿수마다 접근하여 읽을수 있게 하여, 해당 인덱스를 참조하여 출력하도록 만들어놓았다.

TIME은 stat 파일에서 utime과 stime을 읽어드린 후 각각 시스템의 초당 클럭 틱으로 나눈 후 둘을 더해서 나온다. 실제 출력은 printCPUTime에서 시, 분, 초에 맞추어 출력하도록 구현해 두었다.

실행 명령어 부분은 처음과 마지막에 괄호가 있어, 이를 제거한 후 출력하도록 구현하였다.

```c
int main(int argc, char const *argv[]) {
    ...
    if(dir_proc) {
        while ((dir = readdir(dir_proc)) != NULL) {

    ...

        }
        closedir(dir_proc);
    }

    return 0;
}
```

마지막으로 읽어드린 “/proc” 폴더 객체를 닫고 프로그램은 종료된다.

D. 실행 결과

```
> make
gcc -c -o main.o main.c
gcc -o newps main.o
> ps
    PID TTY          TIME CMD
 611296 pts/0    00:00:40 zsh
 611301 pts/0    00:00:00 zsh
 611409 pts/0    00:00:00 zsh
 611410 pts/0    00:00:03 zsh
 611412 pts/0    00:00:14 gitstatusd-linu
1746609 pts/0    00:00:00 ps
> ./newps
    PID        TTY     TIME  CMD
 611296 /dev/pts/0 00:00:40  zsh
 611301 /dev/pts/0 00:00:00  zsh
 611409 /dev/pts/0 00:00:00  zsh
 611410 /dev/pts/0 00:00:03  zsh
 611412 /dev/pts/0 00:00:14  gitstatusd-linu
1746640 /dev/pts/0 00:00:00  newps

~/Pr/O/newps                                  ok base py system node at 08:04:34 AM
```

해당 프로젝트 파일을 컴파일 한 후 실행했을 때 다음과 같이 나오게 된다.