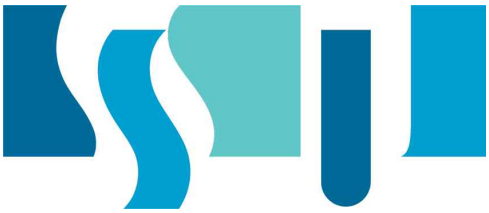

운영체제 과제 2

과목명	운영체제
교수명	김철홍 교수님
학 과	IT대학 컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.09.28.

< 목차 >

1. 개요

운영체제는 한정된 하드웨어 자원을 효율적으로 관리하여 여러 프로세스가 동시에 동작할 수 있도록 하며, 컴퓨터 사용자와 컴퓨터 하드웨어 사이에서 인터페이스 역할을 하여 사용자에게 편리한 컴퓨터 시스템을 제공한다. 운영체제의 궁극적인 목표는 프로세스 관리, 메모리 관리, 스토리지 관리 등을 통한 가상화, 병행성, 지속성의 제공이다.

top, ps, lscpu는 리눅스 시스템에서 사용자가 컴퓨터 자원을 파악할 때 간단히 알아볼 수 있는 명령어들이다. 대부분 proc file system 내의 정보를 바탕으로 사용자에게 제공되며, 이를 직접 구현함으로써 운영체제가 자원을 관리하는 방식에 대해 탐구할 수 있다.

2. 리눅스 명령어 구현

1) mytop

- top은 실시간으로 현재 시스템의 상태를 보여주는 명령어이다. 시스템의 시간, 동작 시간, 접속중인 유저 수, 동작 중인 프로세스의 수와 상태, CPU 사용률, memory 사용률, 프로세스들의 상세 정보를 제공한다.

- mytop은 proc 디렉토리 내의 uptime, loadavg, status, stat 등의 파일을 참조하여 top 명령어가 지원하는 대부분의 기능을 구현하였다. 다만 프로세스 상세 정보에서 메모리와 관련된 VIRT, RES, SHR가 어느 파일로부터 참조되는지 찾지 못해 해당 정보는 제공되지 않는다.

- 소스코드는 다음과 같다.

mytop.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>

typedef struct{
        char PID[20];
        int UID;
        int PR;
        int NI;
        int VIRT;
```

```
        int RES;
        int SHR;
        char S;
        float CPU;
        float MEM;
        int TIME;
        char COMMAND[101];
}myproc;


void printsystem(); //요약 출력
void printtask(); //task 출력
void printcpu(); //CPU usage 출력
void printmemory(); //memory usage 출력
void printdetail(); //process 상태 출력
int main()
{
        system("clear");
        while(1)
        {
                printsystem();
                printtask();
                printcpu();
                printmemory();
                printdetail();
                sleep(3); //top과 동일하게 매 3초마다 새로 출력
                system("clear");
        }
}
void printsystem() //uptime, loadavg
{//uptime 파일에서 시간을, loadavg파일에서 에서 1, 5, 15분간 load average 데이터 파
싱 및 출력
        time_t t=time(NULL);
        struct tm tm= *localtime(&t);
        FILE *fp;
        float uptime, load1, load5, load15;

        printf("top - %02d:%02d:%02d ", tm.tm_hour, tm.tm_min, tm.tm_sec);

        fp=fopen("/proc/uptime", "r");
```

```
        fscanf(fp, "%f", &uptime);
        printf("up %02d:%02d, ", (int)uptime/60/60, (int)uptime/60%60);
        fclose(fp);

        printf("%2d user,  ", 1); ////

        fp=fopen("/proc/loadavg", "r");
        fscanf(fp, "%f", &load1);
        fscanf(fp, "%f", &load5);
        fscanf(fp, "%f", &load15);
        printf("load average: %.2f, %.2f, %.2f", load1, load5, load15);
        fclose(fp);

        printf("\n");
}
void printtask()
{ //동작중인 프로세스들의 state 종합 및 출력
        FILE* fp;
        char path[100]="/proc/";
        char status[8]="/status";
        char temp[101];
        char c;
        DIR* dp=opendir("/proc"); //proc 디렉토리
        struct dirent* dentry; //proc 디렉토리용 엔트리
        int cnt=0; //proc 디렉토리 내 프로세스폴더 카운트
        int R=0, S=0, T=0, Z=0;

        while((dentry=readdir(dp)) != NULL)
                if(atoi(dentry->d_name)!=0) //proc 디렉토리 내 프로세스 개수 카운트
                        cnt++;
        rewinddir(dp);

        while((dentry=readdir(dp)) != NULL)
        {
                if(atoi(dentry->d_name)!=0)
                {
                        strcat(path, dentry->d_name);
                        strcat(path, status);
                        fp=fopen(path, "r");
                        fgets(temp, 100, fp);
```

```
                        fgets(temp, 100, fp);
                        fscanf(fp, "State:%c", &c);
                        switch (c)
                        {
                                case 'T':T++;break;
                                case 'R':R++;break;
                                case 'S':S++;break;
                                case 'I':S++;break;
                                case 'Z':Z++;break;
                        }
                        path[6]='\0';
                        fclose(fp);
                }
        }
        printf("Tasks: %3d total, %3d running, %3d sleeping, %3d stopped, %3d
zombie", cnt, R, S, T, Z);
        printf("\n");

        closedir(dp);
}
void printcpu()
{ //stat 파일을 참조하여 CPU usage 출력
        int us, sy, ni, id, wa, hi, si, st;
        float sum;
        FILE* fp=fopen("/proc/stat", "r");
        fscanf(fp, "cpu  %d", &us);
        fscanf(fp, " %d", &ni);
        fscanf(fp, " %d", &sy);
        fscanf(fp, " %d", &id);
        fscanf(fp, " %d", &wa);
        fscanf(fp, " %d", &hi);
        fscanf(fp, " %d", &si);
        fscanf(fp, " %d", &st);
        sum=0.0+us+sy+ni+id+wa+hi+si+st;

        ////
        printf("%%Cpu(s): %4.1f us, %4.1f sy, %4.1f ni, %4.1f id, %4.1f wa, %4.1f
hi, %4.1f si, %4.1f st",
                        us*100/sum, sy*100/sum, ni*100/sum, id*100/sum,
wa*100/sum, hi*100/sum, si*100/sum, st*100/sum);
```

```
	fclose(fp);
	printf("\n");
}
void printmemory()
{ //meminfo 파일을 참조하여 memory usage 출력
	FILE* fp=fopen("/proc/meminfo", "r");
	char temp[100];
	int memtotal, memfree, memavailable, buffers, cached, swaptotal,
swapfree, sreclaimable;
	fscanf(fp, "MemTotal: %d kB\n", &memtotal);
	fscanf(fp, "MemFree: %d kB\n", &memfree);
	fscanf(fp, "MemAvailable: %d kB\n", &memavailable);
	fscanf(fp, "Buffers: %d kB\n", &buffers);
	fscanf(fp, "Cached: %d kB\n", &cached);
	for(int i=0; i<9; i++)
		fgets(temp, 99, fp);
	fscanf(fp, "SwapTotal: %d kB\n", &swaptotal);
	fscanf(fp, "SwapFree: %d kB\n", &swapfree);
	for(int i=0; i<7; i++)
		fgets(temp, 99, fp);
	fscanf(fp, "SReclaimable: %d kB\n", &sreclaimable);
	printf("MiB Mem : %8.1f total, %8.1f free, %8.1f used, %8.1f buff/cache",
////
			memtotal/1000.0f,			memfree/1000.0f,
(memtotal-memfree-buffers-cached-sreclaimable)/1000.0f,
(buffers+cached+sreclaimable)/1000.0f);
	printf("\n");
	printf("MiB Swap: %8.1f total, %8.1f free, %8.1f used, %8.1f avail Mem",
////
			swaptotal/1000.0f,			swapfree/1000.0f,
(swaptotal-swapfree)/1000.0f, memavailable/1000.0f);

	fclose(fp);
	printf("\n\n");
}
void printdetail()
{ //모든 프로세스의 자세한 정보 출력
	DIR* dp=opendir("/proc"); //proc 디렉토리
	struct dirent* dentry; //proc 디렉토리용 엔트리
	int cnt=0; //proc 디렉토리 내 프로세스폴더 카운트
```

```
int col=0;
while((dentry=readdir(dp)) != NULL)
        if(atoi(dentry->d_name)!=0) //proc 디렉토리 내 프로세스 개수 카운트
                cnt++;

myproc* mp=(myproc*)malloc(sizeof(myproc)*cnt);

FILE* fp;
char path[100]="/proc/";
char temp[101];
char* c;

rewinddir(dp);
while((dentry=readdir(dp)) != NULL) // 모든 프로세스에서 데이터 가져와 저장
{
        if(col==cnt) break;
        if(atoi(dentry->d_name)!=0)
        {
                //char[] PID
                strcpy(mp[col].PID, dentry->d_name);

                //char[] UID
                strcat(path, dentry->d_name);
                strcat(path, "/status");
                fp=fopen(path, "r");
                for(int i=0; i<8; i++)
                        fgets(temp, 100, fp);
                fscanf(fp, "Uid: %d", &mp[col].UID);
                path[6]='\0';
                fclose(fp);

                //int PR //stat 18
                strcat(path, dentry->d_name);
                strcat(path, "/stat");
                fp=fopen(path, "r");
                fgets(temp, 100, fp);
                c=strtok(temp, " ");
                for(int i=0; i<17; i++)
                        c=strtok(NULL, " ");
                mp[col].PR=atoi(c);
```

```
path[6]='\0';
fclose(fp);

//int NI //stat 19
strcat(path, dentry->d_name);
strcat(path, "/stat");
fp=fopen(path, "r");
fgets(temp, 100, fp);
c=strtok(temp, " ");
for(int i=0; i<18; i++)
        c=strtok(NULL, " ");
mp[col].NI=atoi(c);
path[6]='\0';
fclose(fp);

//int VIRT

//int RES

//int SHR

//char S
strcat(path, dentry->d_name);
strcat(path, "/status");
fp=fopen(path, "r");
fgets(temp, 100, fp);
fgets(temp, 100, fp);
fscanf(fp, "State:%c", &(mp[col].S));
path[6]='\0';
fclose(fp);
//float CPU
//float MEM
//int TIME //stat 14, 15, 16, 17
strcat(path, dentry->d_name);
strcat(path, "/stat");
fp=fopen(path, "r");
fgets(temp, 100, fp);
c=strtok(temp, " ");
for(int i=0; i<13; i++)
        c=strtok(NULL, " ");
```

```
                        mp[col].TIME=atoi(c); //14
                        c=strtok(NULL, " ");
                        mp[col].TIME+=atoi(c); //15
                        c=strtok(NULL, " ");
                        mp[col].TIME+=atoi(c); //16
                        c=strtok(NULL, " ");
                        mp[col].TIME+=atoi(c); //17
                        //mp[col].TIME/=100;
                        path[6]='\0';
                        fclose(fp);

                        //char[] COMMAND
                        strcat(path, dentry->d_name);
                        strcat(path, "/comm");
                        fp=fopen(path, "r");
                        fgets(temp, 100, fp);
                        strcpy(mp[col].COMMAND, temp);
                        path[6]='\0';
                        fclose(fp);

                        col++;
                }
        }

        printf("    PID UID        PR  NI    VIRT    RES    SHR S  %%CPU
%%MEM     TIME+ COMMAND\n");
        for(int i=0; i<cnt; i++)
                printf("%7s %-8d %4d %3d %7d %6d %6d %c %5.1f %5.1f
%02d:%02d.%02d %s",
                                mp[i].PID, mp[i].UID, mp[i].PR, mp[i].NI, mp[i].VIRT,
mp[i].RES, mp[i].SHR,
                                mp[i].S, mp[i].CPU, mp[i].MEM, mp[i].TIME/6000,
mp[i].TIME%6000/100, mp[i].TIME%60, mp[i].COMMAND);

        closedir(dp);
        free(mp);
}
```

- top과의 비교 실행 결과는 다음과 같다.

```
san@linux:~/OS$ top

top - 23:07:23 up 22:14,  1 user,  load average: 0.05, 0.06, 0.08
Tasks: 303 total,   1 running, 284 sleeping,  18 stopped,   0 zombie
%Cpu(s):  3.2 us,  0.0 sy,  0.0 ni, 96.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3893.3 total,    383.5 free,   1321.1 used,   2188.8 buff/cache
MiB Swap:    947.2 total,    946.5 free,      0.8 used.   2280.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1694 san       20   0 4110248 261716 110516 S   6.2   6.6  15:24.45 gnome-shell
  41195 san       20   0 1050344  65336  47512 S   6.2   1.6   0:17.99 gnome-terminal-
      1 root      20   0  168892  12916   8316 S   0.0   0.3   0:07.85 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.03 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace
     12 root      20   0       0      0      0 S   0.0   0.0   0:01.99 ksoftirqd/0
     13 root      20   0       0      0      0 I   0.0   0.0   0:12.72 rcu_sched
     14 root      rt   0       0      0      0 S   0.0   0.0   0:00.28 migration/0
     15 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     16 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     17 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     18 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     19 root      rt   0       0      0      0 S   0.0   0.0   0:00.42 migration/1
     20 root      20   0       0      0      0 S   0.0   0.0   0:01.00 ksoftirqd/1
     22 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-events_highpri
```

```
top - 23:06:46 up 22:13,  1 user,  load average: 0.09, 0.06, 0.09
Tasks: 301 total,   1 running, 283 sleeping,  17 stopped,   0 zombie
%Cpu(s):  1.1 us,  1.1 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :   3986.8 total,    438.4 free,   1307.1 used,   2241.3 buff/cache
MiB Swap:    970.0 total,    969.2 free,      0.8 used.   2380.5 avail Mem

    PID UID         PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 0           20   0       0      0      0 S   0.0   0.0  04:10.49 systemd
      2 0           20   0       0      0      0 S   0.0   0.0  00:00.03 kthreadd
      3 0            0 -20       0      0      0 I   0.0   0.0  00:00.00 rcu_gp
      4 0            0 -20       0      0      0 I   0.0   0.0  00:00.00 rcu_par_gp
      6 0            0 -20       0      0      0 I   0.0   0.0  00:00.00 kworker/0:0H-events_highpri
      9 0            0 -20       0      0      0 I   0.0   0.0  00:00.00 mm_percpu_wq
     10 0           20   0       0      0      0 S   0.0   0.0  00:00.00 rcu_tasks_rude_
     11 0           20   0       0      0      0 S   0.0   0.0  00:00.00 rcu_tasks_trace
     12 0           20   0       0      0      0 S   0.0   0.0  00:01.19 ksoftirqd/0
     13 0           20   0       0      0      0 I   0.0   0.0  00:12.11 rcu_sched
     14 0         -100   0       0      0      0 S   0.0   0.0  00:00.28 migration/0
     15 0          -51   0       0      0      0 S   0.0   0.0  00:00.00 idle_inject/0
     16 0           20   0       0      0      0 S   0.0   0.0  00:00.00 cpuhp/0
     17 0           20   0       0      0      0 S   0.0   0.0  00:00.00 cpuhp/1
     18 0          -51   0       0      0      0 S   0.0   0.0  00:00.00 idle_inject/1
     19 0         -100   0       0      0      0 S   0.0   0.0  00:00.42 migration/1
     20 0           20   0       0      0      0 S   0.0   0.0  00:01.40 ksoftirqd/1
     22 0            0 -20       0      0      0 I   0.0   0.0  00:00.00 kworker/1:0H-events_highpri
     23 0           20   0       0      0      0 S   0.0   0.0  00:00.00 kdevtmpfs
```

2) myps

- myps는 현재 시스템에서 동작중인 프로세스들의 정보를 보여주는 명령어이다. 프로세스는 고유의 번호인 PID가 부여되고, 프로세스에 관련된 정보는 /proc 폴더 내에 주어진 PID를 이름으로 한 폴더 내에 존재한다.

- ps명령어 입력 시 보여주는 정보인 PID, TTY, TIME, CMD를 모두 표현하도록 구현하였고, 추가로 프로세스의 사용자 이름(USER)이 나타난다.

```
san@linux:~/OS$ ./myps
USER                 PID TTY          TIME CMD
```

- ps명령어 입력 시 현재 사용 중인 사용자, TTY가 동일한 프로세스만 나타나지만, 더 자세한 정보를 나타내기 위해 모든 프로세스가 나타나도록 구현하였다. 이는 -A 옵션을 준 ps와 유사하다.

- PID, UID의 경우, status 파일을 열람하여 정보를 얻어냈으며, USER명은 /etc/passwd 파일과 앞서 구한 UID를 매칭하면서 구하였다.

- TTY, TIME의 경우, 해당 프로세스 폴더 내의 stat 파일을 열람하여, 구해내었다. stat 파일의 7번째 멤버가 TTY를 의미하는 tty_nr 값이고, 14, 15, 16, 17번째 멤버를 합산하여 Jiffies를 구하였고, 적절한 연산을 통해 시, 분, 초 단위의 TIME을 표현하였다.

- CMD의 경우, comm파일로부터 정보를 얻어내었다.

- 소스코드는 다음과 같다.

myps.c

```
//20172655 LEE KANG SAN
//myps

#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <ctype.h>
#include <string.h>

typedef struct _proc{ //프로세스 정보를 담을 구조체
        char UID[20];
        char USER[20];
        char PID[20];
        int TTY; //
        int TIME;// jiffies time / 100 => seconds
        char CMD[50];
}MYPROC;



DIR* dp; //proc 디렉토리
struct dirent* dentry; //proc 디렉토리용 엔트리
int cnt=0; //proc 디렉토리 내 프로세스폴더 카운트
int col=0; //프로세스 정보 저장된 배열의 인덱스로 사용

int cnt2=0; //passwd파일 내 계정 개수 카운트

int main()
```

```
{
        FILE* fp; //to open proc file systems
        char path[100]="/proc/";
        const char status[8]="/status"; //status 파일 경로
        const char comm[6]="/comm"; //comm 파일 경로
        const char stat[6]="/stat"; //stat 파일 경로
        char temp[200];
        char* c; //

        FILE* fppw; //to open passwd
        char temp2[200]; //내용 파싱 위한 임시 배열
        char** pwarr; //etc/passwd 파일 내용 저장 위한 배열
        fppw=fopen("/etc/passwd", "r");
        for(int i=0; fgets(temp2, 199, fppw)!=NULL; i++) //passwd내 계정 개수 파악
                cnt2++; //count num of line in passwd file(=num of user)

        fseek(fppw, 0, SEEK_SET);

        pwarr=(char**)malloc(sizeof(char*)*cnt2);
        for(int i=0; i<cnt2; i++)
                pwarr[i]=(char*)malloc(sizeof(char)*101);

        for(int i=0; i<cnt2; i++) //passwd 내 모든 계정 정보 저장
                fgets(pwarr[i], 100, fppw);
        //get passwd file for UID



        dp=opendir("/proc");
        while((dentry=readdir(dp)) != NULL)
                if(atoi(dentry->d_name)!=0) //proc 디렉토리 내 숫자로만 이루어진 엔
트리 카운트(프로세스 개수)
                        cnt++;
        rewinddir(dp);
        MYPROC*  p=(MYPROC*)malloc(sizeof(MYPROC)*cnt);  //p[] : for saving
process data

        while((dentry=readdir(dp)) != NULL)
        {
                if(col==cnt) break; //cnt : proc 디렉토리에서 파악한 프로세스 개수
                if(atoi(dentry->d_name)!=0) //number name dir == process
```

```
{

        //open status : PID, UID
        strcat(path, dentry->d_name);
        strcat(path, status);
        fp=fopen(path, "r");
        for(int i=1; fgets(temp, 199, fp)!=NULL; i++)
        {
                if(i==6) //status 파일의 6번째 정보가 Pid
                {
                        c=strtok(temp, "Pid:        ");
                        strcpy(p[col].PID, c);
                        c=strchr(p[col].PID, '\n');
                        c[0]='\0';
                }
                else if(i==9) //status 파일의 9번째 정보가 Uid
                {
                        c=strtok(temp, "Uid:        ");
                        strcpy(p[col].UID, c);
                }
        }
        fclose(fp);
        path[6]='\0';

        //open comm : CMD
        strcat(path, dentry->d_name);
        strcat(path, comm);
        fp=fopen(path, "r");
        if(fgets(temp, 199, fp)!=NULL)
        {
                strcpy(p[col].CMD, temp);
                c=strchr(p[col].CMD, '\n');
                c[0]='\0';
        }
        fclose(fp);
        path[6]='\0';


        //open stat : TTY, TIME
        p[col].TIME=0;
```

```
                        strcat(path, dentry->d_name);
                        strcat(path, stat);
                        fp=fopen(path, "r");
                        fgets(temp, 199, fp);

                        c=strtok(temp, " ");
                        for(int i=1; i<18; i++)
                        {
                                if(i==7) //stat 파일의 7번째 필드가 tty_nr(TTY정보)
                                        p[col].TTY=atoi(c);

                                else if(14<=i&&i<=17) //stat 파일의 14~17번째 정보가 cpu time 관련 정보(jiffies)
                                        p[col].TIME+=atoi(c);

                                c=strtok(NULL, " ");
                        }
                        p[col].TIME/=100; //jiffies를 시분초 단위로 바꾸기 위한 적절한 연산
                        fclose(fp);

                        path[6]='\0';

                        col++; //col번째 프로세스 정보 p[col]에 저장 완료 후 반복
                }
        }

        char* d; //UID를 passwd 파일의 정보와 매칭하여 User 정보 파악 위해 사용
        for(int i=0; i<cnt; i++) //set USER by UID
        {          //passwd의 첫번째 필드 : user, 세번째 필드 : UID
                for(int j=0; j<cnt2; j++)
                {
                        strcpy(temp2, pwarr[j]);
                        c=strtok(temp2, ":");
                        d=c;                            //d = USER
                        c=strtok(NULL, ":");
                        c=strtok(NULL, ":");    //c = UID
                        if(strcmp(c, p[i].UID)==0)
                        {
                                strcpy(p[i].USER, d);
```

```
                                break;
                        }
                }
        }
        printf("USER                    PID TTY          TIME CMD\n"); //출력
        for(int i=0; i<cnt; i++)
        {
                printf("%-18s%5s    %-9d%02d:%02d:%02d    %-20s\n",    p[i].USER,
p[i].PID, p[i].TTY, p[i].TIME/3600, p[i].TIME/60, p[i].TIME%60, p[i].CMD);
        }
}
```

- ps와의 비교 실행 결과는 다음과 같다.

```
san@linux:~/OS$ ps -A
    PID TTY          TIME CMD
      1 ?        00:00:07 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-events_highpri
      9 ?        00:00:00 mm_percpu_wq
     10 ?        00:00:00 rcu_tasks_rude_
     11 ?        00:00:00 rcu_tasks_trace
     12 ?        00:00:01 ksoftirqd/0
     13 ?        00:00:08 rcu_sched
     14 ?        00:00:00 migration/0
     15 ?        00:00:00 idle_inject/0
     16 ?        00:00:00 cpuhp/0
     17 ?        00:00:00 cpuhp/1
     18 ?        00:00:00 idle_inject/1
     19 ?        00:00:00 migration/1
     20 ?        00:00:00 ksoftirqd/1
```

```
san@linux:~/OS$ ./myps
USER                PID TTY          TIME CMD
root                  1 0        00:04:07 systemd
root                  2 0        00:00:00 kthreadd
root                  3 0        00:00:00 rcu_gp
root                  4 0        00:00:00 rcu_par_gp
root                  6 0        00:00:00 kworker/0:0H-events_highpri
root                  9 0        00:00:00 mm_percpu_wq
root                 10 0        00:00:00 rcu_tasks_rude_
root                 11 0        00:00:00 rcu_tasks_trace
root                 12 0        00:00:01 ksoftirqd/0
root                 13 0        00:00:08 rcu_sched
root                 14 0        00:00:00 migration/0
root                 15 0        00:00:00 idle_inject/0
root                 16 0        00:00:00 cpuhp/0
root                 17 0        00:00:00 cpuhp/1
root                 18 0        00:00:00 idle_inject/1
root                 19 0        00:00:00 migration/1
root                 20 0        00:00:00 ksoftirqd/1
root                 22 0        00:00:00 kworker/1:0H-events_highpri
```

- ps -A와 비슷한 정보를 나타내도록 구현하였다.

3) mylscpu

- lscpu는 cpu의 정보를 알려주는 명령어이다. 대부분의 정보를 /proc/cpuinfo 파일에서 확인할 수 있지만, cache 관련 정보는 /sys/devices/system/cpu 폴더 내의 각 코어단위(/cpu0, 1)로 cache 계층 폴더(/index0, 1, 2, 3)가 존재한다.
- 주어진 필수 구현 항목인 CPU Vendor ID, CPU 모델명, CPU 속도, 캐시 크기(L1i, L1d, L2)를 모두 출력하며, CPU의 코어 개수, L3캐시의 크기 또한 출력하도록 구현하였다.
- 캐시의 정보를 상세히 알려주는 –cache 옵션이 구현되었다.
- 소스코드는 다음과 같다.

mylscpu.c

```
//20172655 LEE KANG SAN
//mylscpu

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct{ //정보를 담을 구조체
        char* vendor_id;   //0
        char* model_name; //1
        int num_of_core;   //
        char* cpu_speed;   //2
        int cache_l1d;
        int cache_l1i;
        int cache_l2;
        int cache_l3;
}mycpu; //cpu 정보 담을 구조체 변수

typedef struct _cache{ //캐시 관련 정보를 담을 구조체
        char level[20]; //1, 2, 3
        char size[20];
        char type[20]; //Data, Instruction, Unified
        char ways[20];
}CACHE;

CACHE ca[4]; //캐시 정보 담을 구조체 변수



void setmycpu(char**); //proc/cpuinfo로부터 받아온 정보를 구조체에 저장한다
```

```
void setmember(int, char* ,char**); //실제 저장 연산이 이루어지는 함수
void mylscpu(); //내용 출력
void mylscpucache(); //--cache 내용 출력

int cnt=0; // num of line
int main(int argc, char** argv)
{
	const char* path="/proc/cpuinfo";
	FILE* fp;
	char temp[1000];
	char** cpuinfo; //cpuinfo 파일로부터 파싱한 데이터 전부 저장될 배열
	if((fp=fopen(path, "r"))==NULL)
		exit(1);
	while(!feof(fp)) //count num of line
	{
		cnt++;
		fgets(temp, 1000, fp);
	}
	fseek(fp, 0, SEEK_SET);
	cpuinfo=(char**)malloc(sizeof(char*)*cnt);
	for(int i=0; i<cnt; i++)
		cpuinfo[i]=(char*)malloc(sizeof(char)*1000);

	for(int i=0; i<cnt; i++) //get cpuinfo
		fgets(cpuinfo[i], 1000, fp); //cpuinfo 데이터

	setmycpu(cpuinfo); //cpuinfo 배열 정보를 바탕으로 mycpu 구조체에 데이터 저
장

	if(argc==1) //옵션 없이 호출 시
		mylscpu();
	else if(argc==2 && strcmp(argv[1], "--cache")==0) //--cache 옵션으로 호출
시
		mylscpucache();

	fclose(fp);
	return 0;
}

void setmycpu(char** cpuinfo) //proc/cpuinfo로부터 받아온 정보를 구조체에 저장한다
```

```
{
        for(int j=0, i=0; i<cnt; i++) //cpuinfo는 코어 단위로 정보를 보여주므로 processor 개수를 세어 코어 수 계산
        {
                if(strstr(cpuinfo[i], "processor")!=NULL) j++;
                mycpu.num_of_core=j;
        }

        //get info from proc/cpuinfo
        setmember(0, "vendor_id", cpuinfo);
        setmember(1, "model name", cpuinfo);
        setmember(2, "cpu MHz", cpuinfo);

        //get info of caches
        char p[4][100]={"/sys/devices/system/cpu/cpu0/cache/index0/",
                "/sys/devices/system/cpu/cpu0/cache/index1/",
                "/sys/devices/system/cpu/cpu0/cache/index2/",
                "/sys/devices/system/cpu/cpu0/cache/index3/"};
        char* s[4]={"level", "size", "type", "ways_of_associativity"};
        FILE* fp;
        for(int i=0; i<4; i++) //하나의 코어에서 L1d, L1i, L2, L3의 정보를 각각 추출
        {
                for(int j=0; j<4; j++)
                {
                        strcat(p[j], s[i]);
                        fp=fopen(p[j], "r");
                        if(i==0) {
                                fgets(ca[j].level, 20, fp);
                                for(int a=0; a<20; a++)
                                        if(ca[j].level[a]=='\n') {
                                                ca[j].level[a]='\0';
                                                break;
                                        }
                        }
                        else if(i==1) {
                                fgets(ca[j].size, 20, fp);
                                for(int a=0; a<20; a++)
                                        if(ca[j].size[a]=='\n') {
                                                ca[j].size[a]='\0';
                                                break;
```

```
                                        }
                                }
                                else if(i==2) {
                                        fgets(ca[j].type, 20, fp);
                                        for(int a=0; a<20; a++)
                                                if(ca[j].type[a]=='\n') {
                                                        ca[j].type[a]='\0';
                                                        break;
                                                }
                                }
                                else if(i==3) {
                                        fgets(ca[j].ways, 20, fp);
                                        for(int a=0; a<20; a++)
                                                if(ca[j].ways[a]=='\n') {
                                                        ca[j].ways[a]='\0';
                                                        break;
                                                }
                                }
                                fclose(fp);
                                p[j][42]='\0';
                        }
                }
                //set caches
                for(int i=0; i<4; i++) //추출된 캐시 정보를 바탕으로 mycpu 구조체 내의 캐시 크기 정보를 저장
                {
                        if(strstr(ca[i].level, "1")!=NULL)
                        {
                                if(strstr(ca[i].type, "Data")!=NULL)

mycpu.cache_l1d=atoi(ca[i].size)*mycpu.num_of_core; //L1, Data
                                else

mycpu.cache_l1i=atoi(ca[i].size)*mycpu.num_of_core; //L1, Instruction

                        }
                        else if(strstr(ca[i].level, "2")!=NULL)
                                        mycpu.cache_l2=atoi(ca[i].size)*mycpu.num_of_core; //L2
                        else if(strstr(ca[i].level, "3")!=NULL)
```

```
                                        mycpu.cache_l3=atoi(ca[i].size)*mycpu.num_of_core;
//L3
        }
}

void setmember(int mem, char* str,char** cpuinfo) //mem은 구조체 내의 멤버 접근(switch)위해 사용
{
        char* ptr;
        for(int i=0; i<cnt; i++)
        {
                if((ptr=strstr(cpuinfo[i], str))!=NULL)
                {
                        ptr=strstr(cpuinfo[i], ":")+2;
                        switch (mem)
                        {
                                case 0:mycpu.vendor_id=ptr; break;
                                case 1:mycpu.model_name=ptr; break;
                                case 2:mycpu.cpu_speed=ptr; break;
                        }
                        break;
                }
        }
}

void mylscpu() //cpu 정보 출력
{
        printf("%-20s : %s", "Vendor ID", mycpu.vendor_id);
        printf("%-20s : %s", "Model name", mycpu.model_name);
        printf("%-20s : %d\n", "CPU(s)", mycpu.num_of_core);
        printf("%-20s : %s", "CPU MHz", mycpu.cpu_speed);
        printf("%-20s : %d KiB\n", "L1d cache", mycpu.cache_l1d);
        printf("%-20s : %d KiB\n", "L1i cache", mycpu.cache_l1i);
        printf("%-20s : %d KiB\n", "L2 cache", mycpu.cache_l2);
        printf("%-20s : %d MiB\n", "L3 cache", mycpu.cache_l3/1000);
}

void mylscpucache() //cache 정보 출력
{
        printf("NAME ONE-SIZE ALL-SIZE WAYS TYPE         LEVEL\n");
```

```
        for(int i=0; i<4; i++){
                if(strstr(ca[i].level, "1")!=NULL){
                        if(strstr(ca[i].type, "Data")!=NULL)
                                printf("L1d  %9s%8dK%5s  %-12s%5s\n",  ca[i].size,
mycpu.cache_l1d, ca[i].ways, ca[i].type, ca[i].level);
        }}
        for(int i=0; i<4; i++){
                if(strstr(ca[i].level, "1")!=NULL){
                        if(strstr(ca[i].type, "Instruction")!=NULL)
                                printf("L1i  %9s%8dK%5s  %-12s%5s\n",  ca[i].size,
mycpu.cache_l1i, ca[i].ways, ca[i].type, ca[i].level);
        }}
        for(int i=0; i<4; i++)
                if(strstr(ca[i].level, "2")!=NULL)
                                printf("L2    %9s%8dK%5s  %-12s%5s\n",  ca[i].size,
mycpu.cache_l2, ca[i].ways, ca[i].type, ca[i].level);
        for(int i=0; i<4; i++)
                if(strstr(ca[i].level, "3")!=NULL)
                                printf("L3    %9s%8dK%5s  %-12s%5s\n",  ca[i].size,
mycpu.cache_l3, ca[i].ways, ca[i].type, ca[i].level);
}
```

- lscpu와의 비교 실행 결과는 다음과 같다.

1) 옵션 없이 실행한 모습

```
san@linux:~/OS$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   45 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       2
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
Stepping:                        9
CPU MHz:                         2904.003
BogoMIPS:                        5808.00
Hypervisor vendor:               VMware
Virtualization type:             full
L1d cache:                       64 KiB
L1i cache:                       64 KiB
L2 cache:                        512 KiB
L3 cache:                        8 MiB
```

```
san@linux:~/OS$ ./mylscpu
Vendor ID             : GenuineIntel
Model name            : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
CPU(s)                : 2
CPU MHz               : 2904.003
L1d cache             : 64 KiB
L1i cache             : 64 KiB
L2 cache              : 512 KiB
L3 cache              : 8 MiB
san@linux:~/OS$
```

2) --cache 옵션을 주고 실행한 모습

```
san@linux:~/OS$ lscpu --cache
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K       64K    8 Data            1
L1i       32K       64K    8 Instruction     1
L2       256K      512K    4 Unified         2
L3         4M        8M   16 Unified         3
san@linux:~/OS$
```

```
san@linux:~/OS$ ./mylscpu --cache
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K       64K    8 Data            1
L1i       32K       64K    8 Instruction     1
L2       256K      512K    4 Unified         2
L3      4096K     8192K   16 Unified         3
san@linux:~/OS$
```

3. 과제를 통해 느낀 점

proc 디렉토리 내의 파일들을 직접 탐구하며 시스템에서 관리하는 다양한 정보들에 대해 알게 되었다. mytop, myps, mylscpu를 직접 구현해 봄으로써, 리눅스 OS가 자원을 관리하는 방식에 대해 탐구해 볼 수 있었다.