리눅스 명령어 top, ps, lscpu 구현보고서

2012609

김시온

I. 서론

리눅스 명령어인 top, ps, lscpu 명령어 구현

1) system 함수 or exec 계열 함수를 사용하여 해당 명령어를 호출하지 않고 직접 기능을 구현하기

2) mytop, myps 명령어는 리눅스 top, ps 명령어와 최대한 유사한 결과를 출력하게 구현

II. 본론

top, ps , lscpu 명령어가 보여주는 모든 정보는 컴퓨터 속에 내장되어있다.

그 내장 되어있는 파일의 위치를 파악하여 사용자가 쉽게 볼 수 있도록 출력하는 것이 주된 목표이다.

그 내용들은 리눅스 명령어 grep, find 명령어를 통하여 그 파일의 정보를 검색하였으며, 그럼에도 나타나지 않는 정보에 대해서는 구글검색을 통해 위치정보를 검색하였다.

1.mytop

총 세개의 파일로 구성되어있다.

1)mytop.c

-void print_mytop(void) : 화면에 top 명령어에서 출력되는 값을 출력하는 함수이다.

-void isGreater(myProc *a, myProc * b):

두 proc* 를 cpu와 pid 로 비교하여 무엇이 큰지 반환하는 함수이다.

-void sort_by_cpu(void) : procList 를 cpu순으로 정렬하는 함수이다.

-int main(int argc,char *argv[])

실제로 수행되는 함수이다.

2)util.c

mytop.c 와 myps.c 에서 동시에 사용하는 함수들을 모아놓은 파일이다.

-unsigned long kib_to kb : Kib 단위를 kb로 변환하는 함수이다.

/proc/meminfo 에서 물리 메모리 크기를 얻을 때 사용한다.

-long double round_double (long double src, int rdx) :my Proc 구조체에 값을 넣을 때 소숫점 아래 3자리에서 반올림 하기 위한 함수이다.

-void getTTY(char path(PATH_LEN],char tty[TTY_LEN]){}

-unsigned long get_uptime(void) : /proc/uptime 에서 OS 부팅 후 지난 시간 얻는 함수

-unsigned log get_mem_total (void) {}

: /proc/meminfo에서 전체 물리 메모리 크기 얻는 함수

-void add_proc_list(char path[PATH_LEN], bool isMyps, bool aOption, bool uOption, bool xOption, unsigned long cpuTimetable[PID_MAX])

:

-void search_proc(bool isMyps, bool aOption, bool uOption, bool xOption , unsigned long cpuTimeTable[PID_MAX])

-void erase_proc(myProc *proc) : proc의 내용을 지우는 함수

-void erase_proce_list(void) : procList 내용을 지우는 함수

erase_proc 함수를 이용해 모두 지운다.

3)header.h

1. 사용할 라이브러리 함수를 위한 헤더파일을 선언한다.
2. 참조할 파일의 절대경로를 define을 이용해 간단히 표현하였다.
3. print_mytop함수에서 출력시 행과 열을 mvprintw 함수를 이용해 출력하는데, 그를 위한 행과 열을 정의해 주었다.
4. 출력시 필요한 정보를 포함하는 myProc 구조체를 선언하였다.

이는 util.c 속의 add_proc_list 함수를 호출함으로써 값이 입력된다.

```c
//process를 추상화 한 myProc 구조체
typedef struct{
    unsigned long pid;
    unsigned long uid;          //USER 구하기 위한 uid
    char user[UNAME_LEN];       //user명
    long double cpu;            //cpu 사용률
    long double mem;            //메모리 사용률
    unsigned long vsz;          //가상 메모리 사용량
    unsigned long rss;          //실제 메모리 사용량
    unsigned long shr;          //공유 메모리 사용량
    int priority;               //우선순위
    int nice;                   //nice 값
    char tty[TTY_LEN];          //터미널
    char stat[STAT_LEN];        //상태
    char start[TIME_LEN];       //프로세스 시작 시각
    char time[TIME_LEN];        //총 cpu 사용 시간
    char cmd[CMD_LEN];          //option 없을 경우에만 출력되는 command (short)
    char command[CMD_LEN];      //option 있을 경우에 출력되는 command (long)

}myProc;
```

top 명령어는 여러 줄로 표현된다.

1_ `top - 05:13:52 up  8:10,  1 user,  load average: 0.00, 0.07, 0.07`

시스템의 현재시간, OS가 살아있는 시간, 유저의 세션 수, 그리고

1분, 5분 15분에 대한 CPU Load의 이동평균을 표시한다.

여기서 CPU Load란 CPU가 수행하는 작업의 양이다.

시스템의 시간은 tm 구조체를 사용하여 3초마다 갱신한다.

사용자에 대한 정보는 utmp구조체를 사용한다.

setutent() 함수와 getutent() 함수를 사용하여 유저의 정보를 읽는다.

load average 값은, 헤더에서 선언한 위치인 /proc/loadavg 경로에서 값을 읽은 후, 이를 모두 종합하여

top의 첫번째 줄에 출력한다.

2_ `Tasks: 283 total,   2 running, 281 sleeping,   0 stopped,   0 zombie`

각 상태를 갖는 프로세스의 수를 출력한다.

프로세스에 대한 정보는 main함수에서 search_proc 함수를 호출함으로써 읽어 드린다.

proc 디렉토리를 탐색하여 process에 대한 정보를 procList[].stat 에 입력한 후, 이에 따라 running, sleeping, stopped, zombie 프로세스를 구분한다.

3_ %Cpu(s): 4.0 us, 9.7 sy, 0.0 ni, 86.0 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st

최근에 갱신 (3초마다) 이후로부터 CPU 사용률을 나타낸다.

us : 프로세스의 유저 영역에서의 CPU 사용률

sy : 프로세스의 커널 영역에서의 CPU 사용률

ni : 프로세스의 우선순위(priority) 설정에 사용하는 CPU 사용률

id : 사용하고 있지 않는 비율

wa : IO가 완료될때까지 기다리고 있는 CPU 비율

hi : 하드웨어 인터럽트에 사용되는 CPU 사용률

si : 소프트웨어 인터럽트에 사용되는 CPU 사용률

st : CPU를 VM에서 사용하여 대기하는 CPU 비율 이며,

이에 대한 정보는 /proc/stat 1행에 순서대로 존재한다.

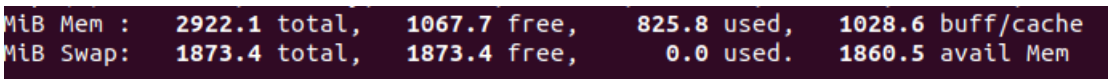
3초마다 갱신을 해서 출력을 해야하므로 main 함수에서

hertz = (unsigned int) sysconf(_SC_CLK_TCK) 함수를 통해

os의 hertz 값을 얻고, 이에 따라 tickCnt를 계산하여, 갱신한다.

hertz: 1초 간 일어나는 문맥교환 횟수

4~5행



메모리 사용량에 대한 정보를 출력하며,

/proc/meminfo 에 그에 대한 정보가 있다. 그 곳에 저장된 단위는 kb 이지만, 출력해야하는 단위는 kib이므로, 단위변환을 수행한다.

첫번째 줄은 RAM의 메모리 영역이고,

두번째 줄은 Swap의 메모리 영역이다.

일반적으로 Mem의 사용량이 거의 가득 찼을때 Swap 메모리 영역을 사용한다.

meminfo에 있는 값은 메모리의 Total 값과, Free값이므로,

사용되는 데이터 값은 Total - Free 식을 통해 계산된다.

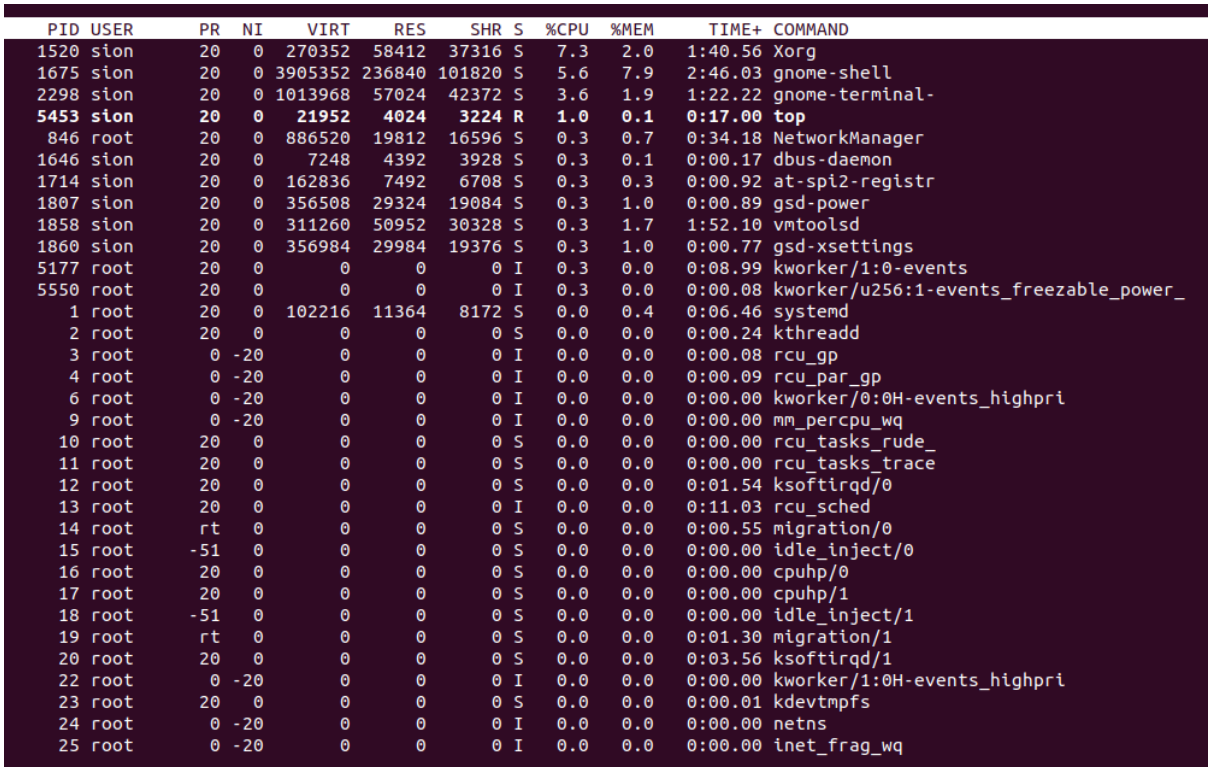
그리고 메모리는, 이 중에서도 추가로 버퍼에 필요한 메모리 값인 buff/cache값도 제외해야하므로, 식은 다음과 같다.

memUsed = memTotal - memFree - buffers- cached - sReclaimable;

(sReclaimable : Slab 영역 중 캐시 용도로 사용하는 메모리의 양)

swapUsed = swapTotal - swapFree

PID



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1520	sion	20	0	270352	58412	37316	S	7.3	2.0	1:40.56	Xorg
1675	sion	20	0	3905352	236840	101820	S	5.6	7.9	2:46.03	gnome-shell
2298	sion	20	0	1013968	57024	42372	S	3.6	1.9	1:22.22	gnome-terminal-
5453	**sion**	**20**	**0**	**21952**	**4024**	**3224**	**R**	**1.0**	**0.1**	**0:17.00**	**top**
846	root	20	0	886520	19812	16596	S	0.3	0.7	0:34.18	NetworkManager
1646	sion	20	0	7248	4392	3928	S	0.3	0.1	0:00.17	dbus-daemon
1714	sion	20	0	162836	7492	6708	S	0.3	0.3	0:00.92	at-spi2-registr
1807	sion	20	0	356508	29324	19084	S	0.3	1.0	0:00.89	gsd-power
1858	sion	20	0	311260	50952	30328	S	0.3	1.7	1:52.10	vmtoolsd
1860	sion	20	0	356984	29984	19376	S	0.3	1.0	0:00.77	gsd-xsettings
5177	root	20	0	0	0	0	I	0.3	0.0	0:08.99	kworker/1:0-events
5550	root	20	0	0	0	0	I	0.3	0.0	0:00.08	kworker/u256:1-events_freezable_power_
1	root	20	0	102216	11364	8172	S	0.0	0.4	0:06.46	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.24	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.08	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.09	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	0:01.54	ksoftirqd/0
13	root	20	0	0	0	0	I	0.0	0.0	0:11.03	rcu_sched
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.55	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	0:01.30	migration/1
20	root	20	0	0	0	0	S	0.0	0.0	0:03.56	ksoftirqd/1
22	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/1:0H-events_highpri
23	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kdevtmpfs
24	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	inet_frag_wq

각 열의 길이를 맞춰야 하므로 ,

int columnWidth[COLUMN_CNT] 배열을 통해 크기를 설정한다.

만약 주어진 내용이 정해진 배열의 너비보다 넓을 경우, 너비의 최댓값을 늘린다.

main 함수 속 search_proc 함수를 통해 입력받은 myProc값을

sort_by_cpu 함수를 통해, cpu 순으로 정렬한다.

그 값들은 sorted 배열로 저장되어 있는데, 차례대로

sorted[]->pid , sorted[i]-> user , sorted[]->priority , , sorted[]->nice , sorted[]->vsz , sorted[]->rss, sorted[]->shr, sorted[]->stat, sorted[]->cpu, sorted[]->mem, sorted[]->time,값을 출력한다.

방향키를 누른다면, column ,row 수를 변화시켜 출력을 보여준다.

q를 누르면 종료한다.

2. myps

util.c 와 header.h 는 동일하다.

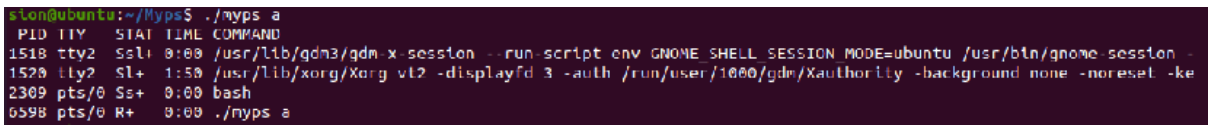
myps.c 는 두가지 함수가 존재한다.

1.print_myps(void){} : 실제 화면에 출력하는 함수.

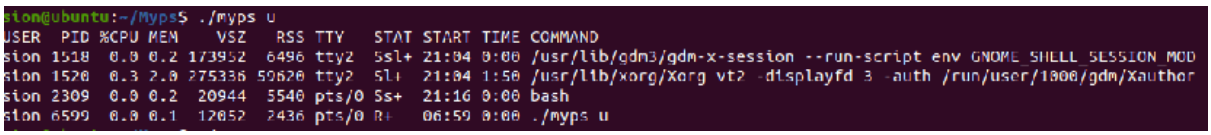
2. int main(int argc,char argv[]) : 옵션을 입력받을 수 있게 하는 메인함수이다.

옵션은 다음 세 가지를 의미한다

a : 모든사용자를 기준으로 출력한다

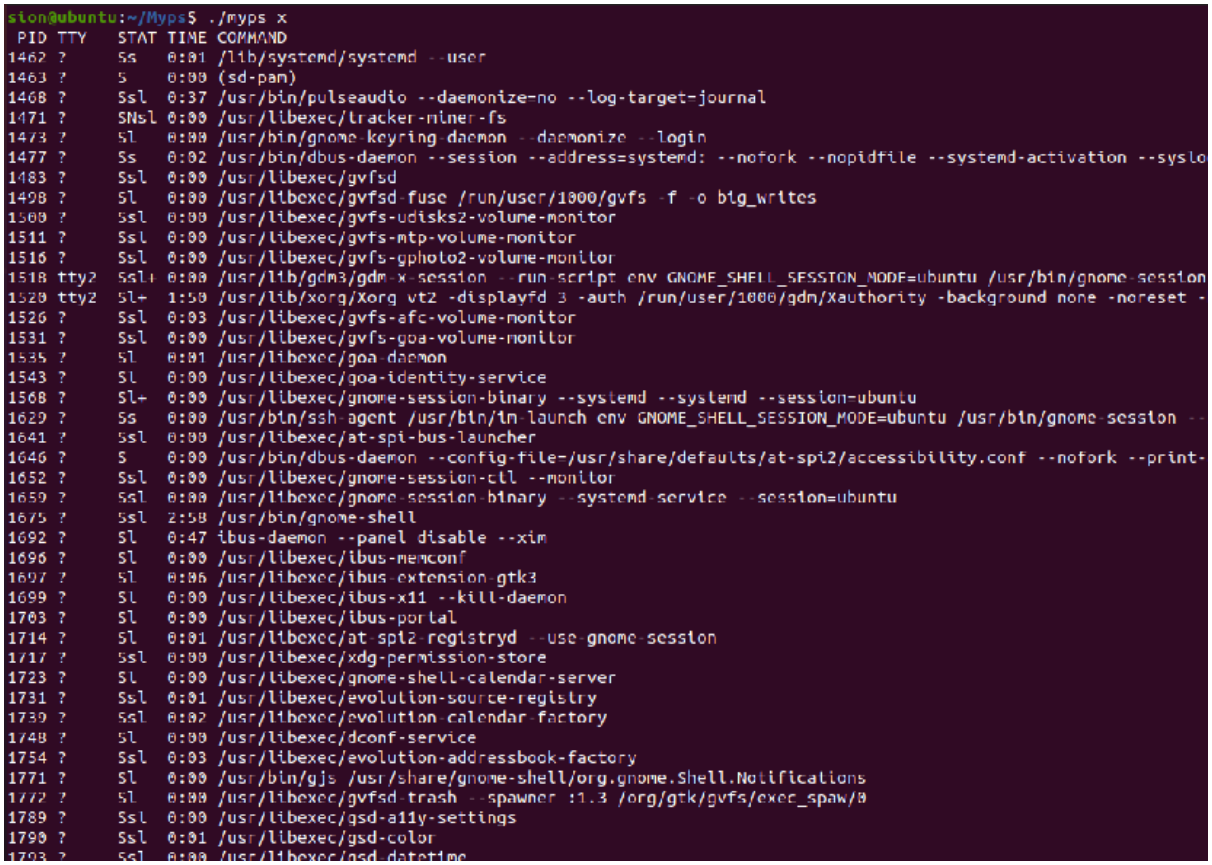


u : 프로세스의 소유자를 기준으로 출력한다.



x: 혼자 사용되면 사용자에 의해 소유한 모든 프로세스를 출력하고

a 옵션 과 함께 사용되어 모든 프로세스를 출력한다.



옵션에 따라 print_myps 함수에서,

```
if(!aOption && !uOption && !xOption){                    // myps 인 경우
                                                columnWidth[USER_IDX] = -1;       //USER 출력 X
                                                columnWidth[CPU_IDX] = -1;        //CPU 출력 X
                                                columnWidth[MEM_IDX] = -1;        //MEM 출력 X
                                                columnWidth[VSZ_IDX] = -1;        //VSZ 출력 X
                                                columnWidth[RSS_IDX] = -1;        //RSS 출력 X
                                                columnWidth[STAT_IDX] = -1;       //STAT 출력 X
                                                columnWidth[START_IDX] = -1;//START 출력 X
                                                columnWidth[COMMAND_IDX] = -1;//COMMAND 출
력 X
```

```
}
else if(!uOption){
// myps a / myps x / myps ax인 경우
        columnWidth[USER_IDX] = -1;       //USER 출력 X
        columnWidth[CPU_IDX] = -1;        //CPU 출력 X
        columnWidth[MEM_IDX] = -1;        //MEM 출력 X
        columnWidth[VSZ_IDX] = -1;        //VSZ 출력 X
        columnWidth[RSS_IDX] = -1;        //RSS 출력 X
        columnWidth[START_IDX] = -1;//START 출력 X
        columnWidth[CMD_IDX] = -1;        //CMD 출력 X
}
else
            // myps u / myps au / myps ux / myps aux인 경우
        columnWidth[CMD_IDX] = -1;        //CMD 출력 X
```

형식으로 옵션을 관리하였다.

그 외의 소스코드는 mytop과 유사하다.

3. Mylscpu

함수는 mylscpu.c 하나에 모두 있으며,

Vendor_id ~ Cpu MHz 는 /proc/cpuinfo 파일에 존재한다.

각각 2 , 3, 5, 8 번째 줄에 위치한다.

그리고, 캐쉬값을 의미하는

L1i,L1d,L2,L3 는

/sys/devices/system/cpu/cpu *0~ 코어-1* /cache / index * 0 ~3* 값에 존재한다.

cpu 코어의 수에 따라 cpu 폴더 수에 cpu0 ~ cpu(코어 -1) 까지 폴더가 존재하며,

각 폴더의 cache 폴더 속, index0~3 파일 안에는

index0 : L1i의 Kib,

index1 : L1d Kib

index2: L2 Kib

index3 : L3 Kib

이 저장되어있으며, lscpu 출력방식에 따라 L3Kib 값은 MiB 값으로 변환시켜준다.

```
sion@ubuntu:~/Mylscpu$ ./mylscpu
Vendor_id       : GenuineIntel
Cpu family      : 6
Model           : 142
Model name      : Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Cpu MHz         : 2712.000
L1i             : 64    KiB
L1d             : 64    KiB
L2              : 512   KiB
L3              : 6     MiB
```

III. 결론

위 과제를 수행하면서, 리눅스에는 정말 엄청나게 많은 파일들이 존재한다는 사실을 알았다. 그리고 그 필요한 정보들은 grep, find 등 여러 명령어를 사용하여 위치를 파악해야하고, 단위를 조정하는 작업이 필요함을 느꼈다.

각각의 정보의 위치가 어딘지 확실치 않아 인터넷을 찾아보았고, 이를 통해서 한번도 관심을 가지지 않았던 루트디렉토리에 있는 /proc, /sys 디렉토리가 무엇인지 파악할 수 있었다.