운영체제(가) 설계과제

숭실대학교 컴퓨터학부 20182612 박민호

**<개요>**

🡺리눅스 내장 명령어를 실행하는 간단한 쉘(ssu\_shell) 구현

-> 입력 (대화식 또는 배치식 모드)을 읽고 입력을 토큰화

-> ./ssu\_shell : 대화식 모드

-> ./ssu\_shell commands.txt : 배치식 모드

- ssu\_shell은 사용자 입력 -> fork(2)를 사용하여 하나 이상의 자식 프로세스를 생성

-> 자식 프로세스로부터 exec()를 호출하여 사용자 명령을 실행

-> wait(2)를 이용하여 죽은 자식을 거둠

- ssu\_shell은 ls, cat, echo 및 sleep과 같은 모든 간단한 리눅스 내장 명령어 실행이 가능해야함.

- 리눅스 내장 명령어 말고 추가로 직접 구현할 명령어 : pps, ttop

- ssu\_shell의 프롬프트 = $

- 대화식 모드에서 사용자가 셀을 종료하기 위해 Ctrl + C를 누를 때까지 무기한 실행을 계속

- 배치식 모드에서 쉘은 파일 끝에 도달하면 종료 ( 한 줄씩 명령어를 읽음 )

- 실행할 명령어와 해당 인자의 입력에서 하나 이상의 공백으로 구분되어 있다고 가정하고 공백을 구분 기호로 사용

-> 입력 스트림을 토큰화 해야함.

- 입력된 명령어는 1024자 이하, 토큰은 64개 이하

- I/O 리디렉션 등의 특수 실행모드는 구현하지 않아도 되나 파이프는 구현되어야 함. ( 특수문자 구문 분석 불필요 )

- ssu\_shell은 오류를 정상적으로 처리해야 함

-> 빈 명령어 (엔터)은 단순히 쉘이 오류 메시지 없이 $ 프롬프트를 다시 표시

-> 잘못된 명령어, 기타 잘못된 입력에 대해 ssu\_shell은 오류 메시지 출력 후, 프롬프트 재출력

-> 모든 명령어에 대해 ssu\_shell은 자식 프로세스를 종료하고 회수해야 함.

🡺 쉘 명령어 ttop, pps 구현

- ttop와 pps 는 별도로 구현하고, 구현한 명령어가 ssu\_shell에서 실행 되어야함

- (1) ttop ( 미구현 )

- (2) pps

- pps 명령어 입력 시, 기본 출력은 ps와 유사한 기능을 가지고 있음

- 명령어 결과 출력 시 현재 터미널 크기에 맞게 글자 수가 한 행을 넘어가지 않도록 구현

-> libcurses5-dev 패키지 사용 필요

🡺 사용법을 숙지하지 못해 ioctl 함수를 이용해 pps 실행 시의 터미널 크기를 구했습니다.

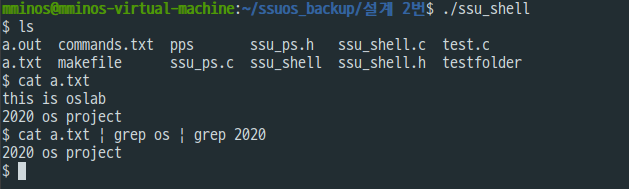
- 옵션 a, u, x 구현

-> 각각의 옵션에 대한 출력은 ps의 a, u, x 옵션과 동일

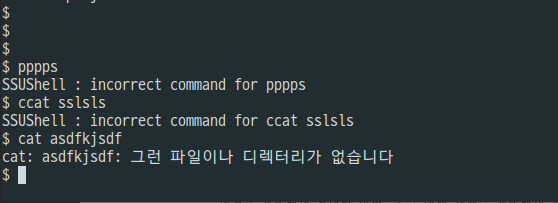
-> 위 세가지 옵션은 함께 사용 가능해야 함.

**<실행 결과>**

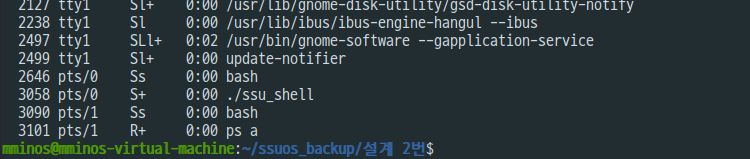
**<ssu\_shell>**



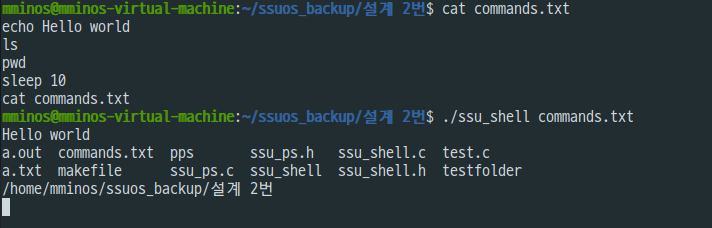
(ssu\_shell 실행 및 멀티 파이프 실행 예제)

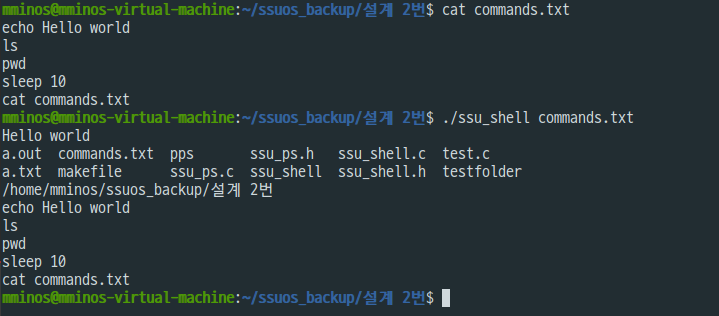


(빈 명령어 (엔터) 입력 시 프롬프트 재 출력과, 잘못된 명령어 입력 시 에러 메시지 출력)



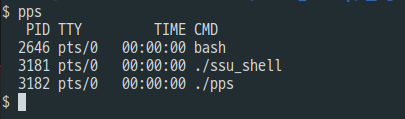
(다른 터미널에서 ps a 입력 시 출력 화면 => 자식 프로세스들이 좀비가 되지 않고 잘 처리됨)



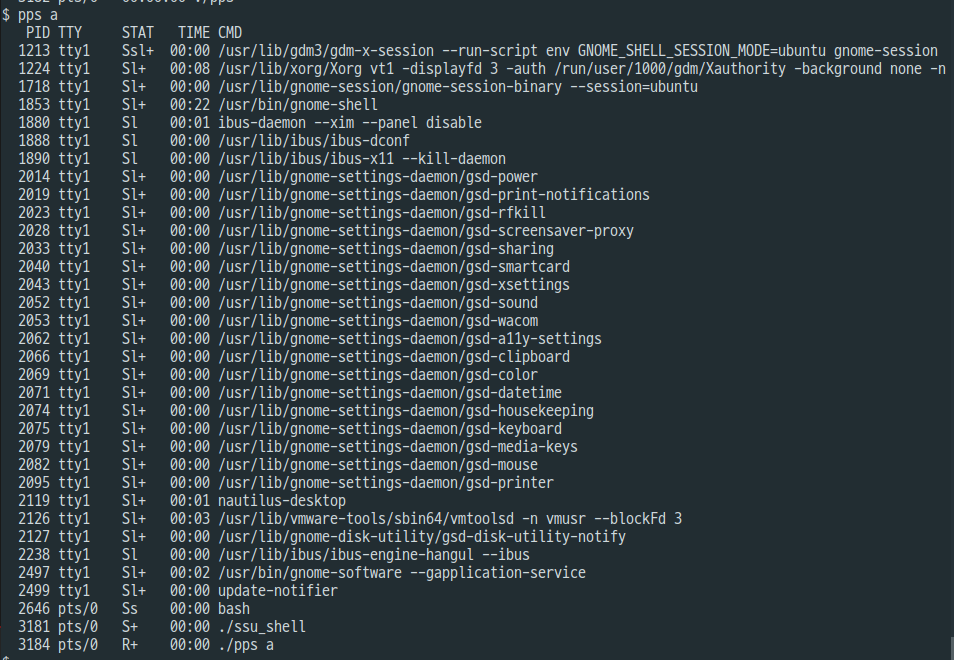


(배치식 모드)

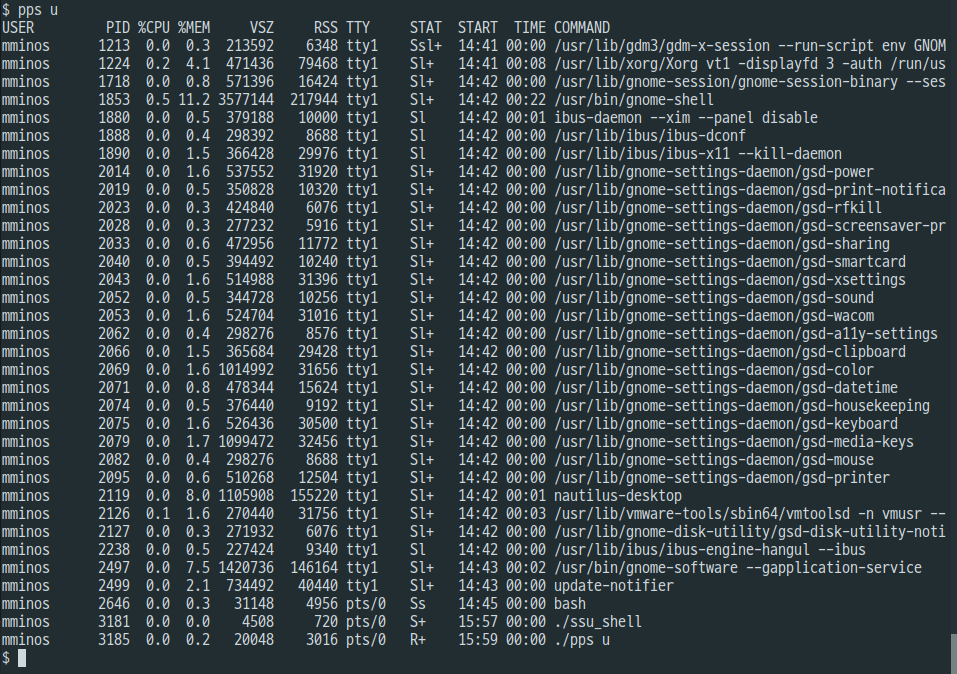
**<pps>**



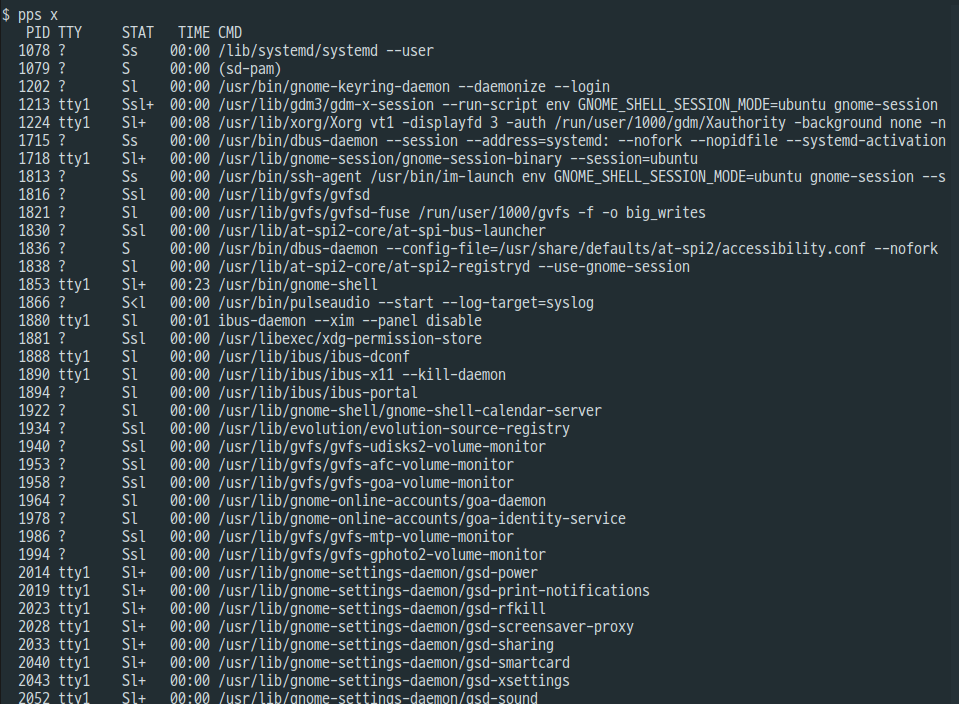
(pps 입력 시 ./pps 로 변환해서 명령어 실행했습니다.)



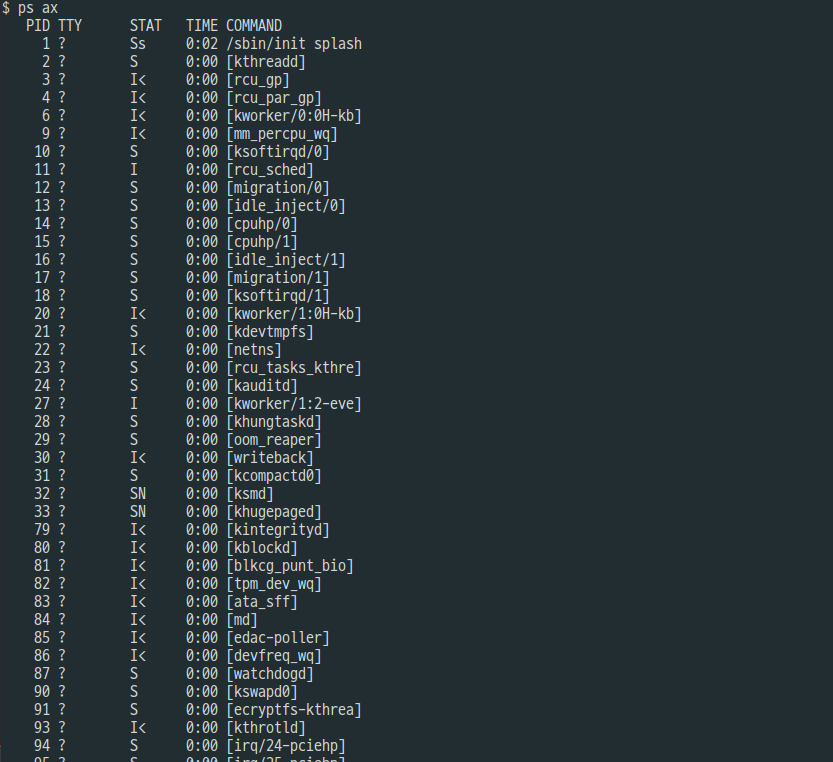
( 'a' 옵션 )



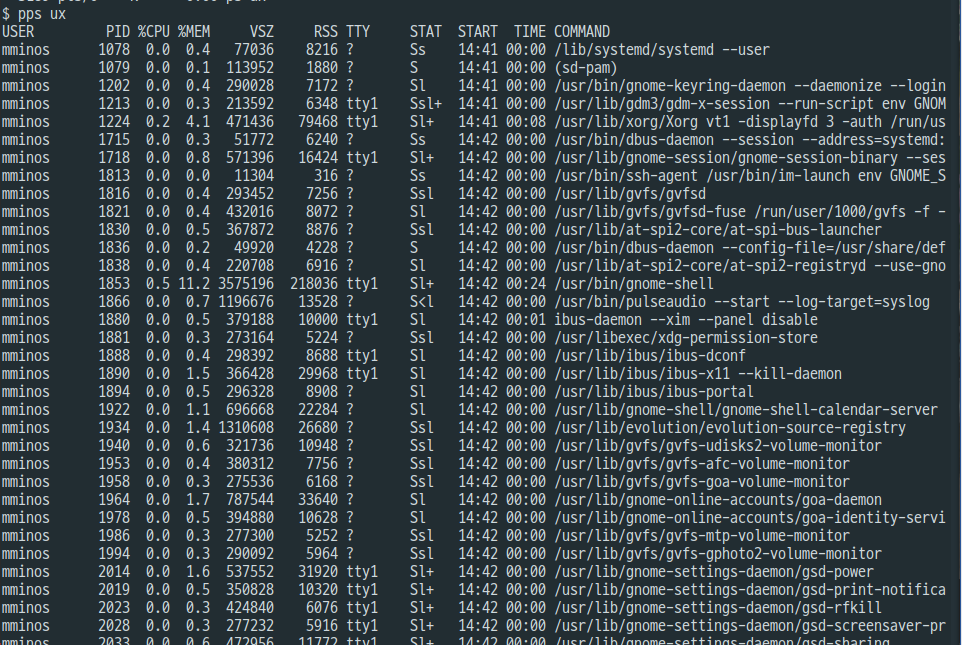
( 'u' 옵션 )



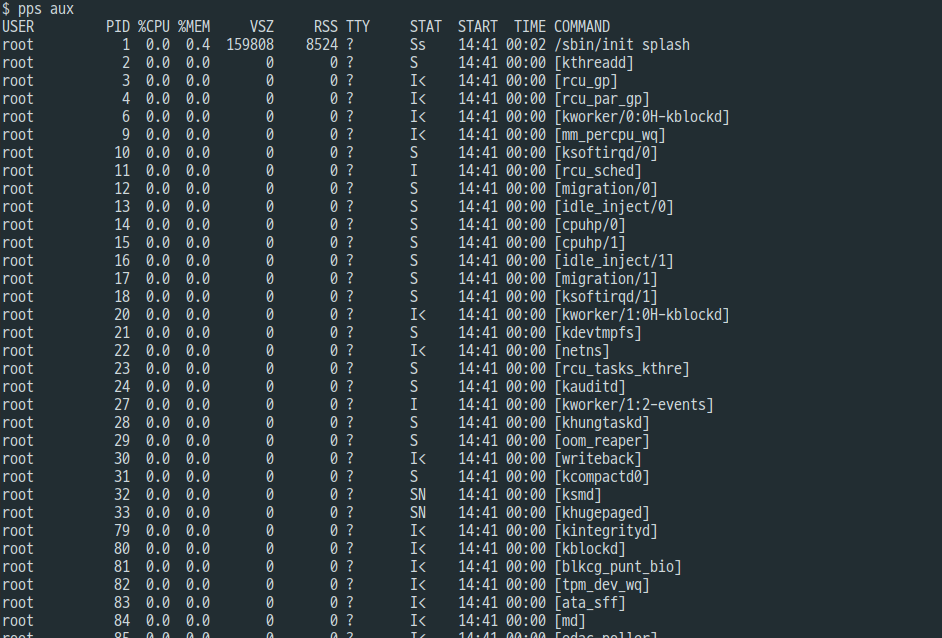
( 'x' 옵션 )

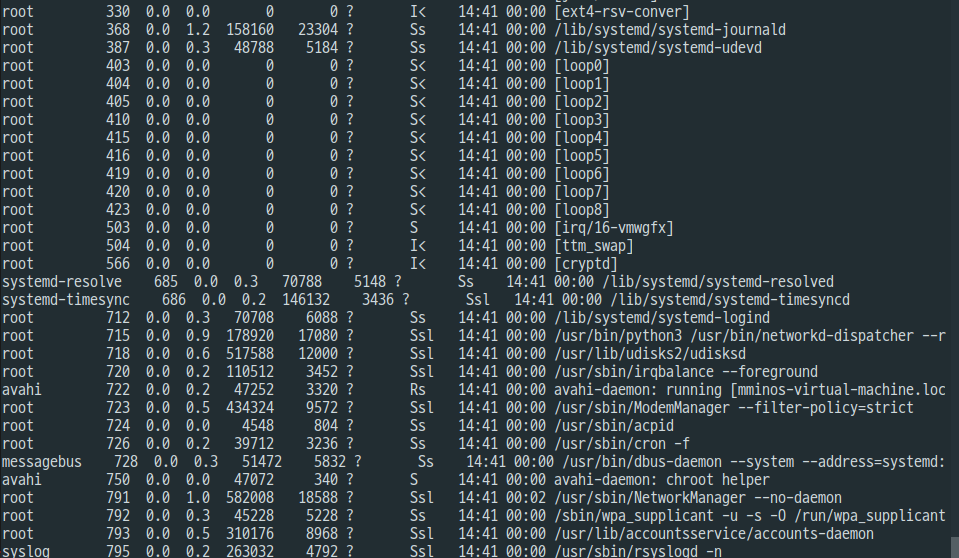


( 'ax' 옵션 - 전체 프로세스 출력 )



( 'ux' 옵션 )





( 'aux' 옵션 - 전체 프로세스 출력 )

**<ttop>**

- 미 구현

**<소스코드>**

**<ssu\_shell.h>**

#ifndef \_SSU\_SHELL\_

#define \_SSU\_SHELL\_

//headers

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <signal.h>

#include <errno.h>

//define

#define MAX\_INPUT\_SIZE 1024

#define MAX\_TOKEN\_SIZE 64

#define MAX\_TOKEN\_NUM 64

#define MAX\_COMMAND\_NUM 64

//functions

char \*\*tokenize(char \*line);

char \*\*merge\_to\_command(char \*\*tokens);

void execute\_commands(char \*\*commands, int commandNo);

void execute\_command(char \*command);

#endif

**<ssu\_shell.c>**

#include "ssu\_shell.h"

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

{

pid\_t pid;

char line[MAX\_INPUT\_SIZE];

char \*\*tokens;

char \*\*commands;

int i, commandNo = 0;

FILE\* fp;

if (argc == 2) {

fp = fopen(argv[1],"r");

if (fp == NULL) {

printf("File doesn't exists.");

return -1;

}

}

while(1) {

int pipe\_num = 0;

int status;

commandNo = 0;

memset(line, 0, sizeof(line));

if(argc == 2) {

if(fgets(line, sizeof(line), fp) == NULL) {

break;

}

line[strlen(line) - 1] = '\0';

} else {

printf("$ ");

scanf("%[^\n]", line);

getchar();

}

// printf("Command entered: %s (remove this debug output later)\n", line);

line[strlen(line)] = '\n';

tokens = tokenize(line);

for (i = 0; tokens[i] != NULL; i++) {

if (!strcmp(tokens[i], "pps")) {

tokens[i] = realloc(tokens[i], 6);

sprintf(tokens[i], "%s", "./pps");

}

else if (!strcmp(tokens[i], "ttop")) {

tokens[i] = realloc(tokens[i], 7);

sprintf(tokens[i], "%s", "./ttop");

}

// printf("found token %s (remove this debug output later)\n", tokens[i]);

}

commands = merge\_to\_command(tokens);

for (i = 0; commands[i] != NULL; i++, commandNo++) {

// printf("commands : %s (remove this depug output later)\n", commands[i]);

}

if ((pid = fork()) < 0) {

fprintf(stderr, "in function main: fork() error\n");

return -1;

}

else if (pid == 0) {

execute\_commands(commands, commandNo);

}

else {

wait(&status);

if (!WIFEXITED(status) && tokens[0] != NULL) {

fprintf(stderr, "child process doesn't exit normally.. check process\n");

continue;

}

}

for (i = 0; tokens[i] != NULL; i++){

free(tokens[i]);

}

free(tokens);

for (i = 0; commands[i] != NULL; i++) {

free(commands[i]);

}

free(commands);

}

return 0;

}

char \*\*tokenize(char \*line)

{

char \*\*tokens = (char \*\*)malloc(MAX\_TOKEN\_NUM \* sizeof(char \*));

char \*token = (char \*)malloc(MAX\_TOKEN\_SIZE \* sizeof(char));

int i, tokenIndex = 0, tokenNo = 0;

for (i = 0; i < strlen(line); i++) {

char readChar = line[i];

if (readChar == ' ' || readChar == '\n' || readChar == '\t') {

token[tokenIndex] = '\0';

if (tokenIndex != 0) {

tokens[tokenNo] = (char\*)malloc(MAX\_TOKEN\_SIZE\*sizeof(char));

strcpy(tokens[tokenNo++], token);

tokenIndex = 0;

}

}

else {

token[tokenIndex++] = readChar;

}

}

free(token);

tokens[tokenNo] = NULL ;

return tokens;

}

char \*\*merge\_to\_command(char \*\*tokens)

{

char \*\*commands = (char \*\*)malloc(MAX\_COMMAND\_NUM \* sizeof(char \*));

char command[1024];

int i, commandNo = 0;

memset(command, 0, 1024);

for (i = 0; tokens[i] != NULL; i++) {

if (!strcmp(tokens[i], "|")) {

commands[commandNo] = (char \*)malloc(1024 \* sizeof(char));

strcpy(commands[commandNo++], command);

memset(command, 0, 1024);

}

else {

strcat(command, tokens[i]);

strcat(command, " ");

}

}

commands[commandNo] = (char \*)malloc(1024 \* sizeof(char));

strcpy(commands[commandNo++], command);

commands[commandNo] = NULL;

return commands;

}

void execute\_commands(char \*\*commands, int commandNo)

{

int i;

int pipe\_fd[2];

pid\_t pid;

for (i = 0; i < commandNo - 1; i++) {

pipe(pipe\_fd);

if ((pid = fork()) < 0) {

fprintf(stderr, "in function execute\_commands: fork() error\n");

exit(1);

}

else if (pid == 0) {

close(pipe\_fd[0]);

dup2(pipe\_fd[1], STDOUT\_FILENO);

execute\_command(commands[i]);

}

else {

close(pipe\_fd[1]);

dup2(pipe\_fd[0], STDIN\_FILENO);

}

}

execute\_command(commands[i]);

}

void execute\_command(char \*command)

{

char \*\*command\_tok;

command\_tok = tokenize(command);

execvp(command\_tok[0], command\_tok);

fprintf(stderr, "SSUShell : incorrect command for %s\n", command);

exit(1);

}

**<ssu\_ps.h>**

#ifndef \_SSU\_PS\_

#define \_SSU\_PS\_

// header

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>

#include <errno.h>

#include <ctype.h>

#include <pwd.h>

#include <time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/ioctl.h>

#include <linux/kdev\_t.h>

//defines

#define true 1

#define false 0

#define PATHLEN 1024

#define NAMELEN 512

#define LINEMAX 512

typedef struct proc\_info {

int pid; // /proc/pid/stat

int ppid; // /porc/pid/stat

int pgrp, sid, tpgid; // /proc/pid/stat

int tty\_nr; // /proc/pid/stat

char state; // /proc/pid/stat

char cmdline[NAMELEN]; // /porc/pid/cmdline

char name[64]; // getpwuid..

unsigned long utime, stime; // /proc/pid/stat

unsigned long vsize; // /proc/pid/stat

long rss, nice, num\_threads; // /proc/pid/stat

unsigned long long starttime; // /proc/pid/stat

struct proc\_info \*next;

} proc\_list;

//functions

int check\_option(char \*\*argv);

void scan\_proc(void);

void print\_ps(void);

void print\_default(void);

void print\_without\_x(void);

void print\_without\_a(void);

void print\_all(void);

proc\_list \*create\_node(proc\_list \*stat);

void add\_node(proc\_list \*node);

void print\_list(void);

void free\_list(proc\_list \*tmp);

time\_t get\_boot\_time(void);

long get\_total\_memory(void);

double get\_uptime(void);

void get\_tty(int tty\_nr, char \*tty);

void get\_state(proc\_list \*tmp, char \*state);

int scandir\_filter(const struct dirent \*file);

int \_isdigit(char \*str);

#endif

**<ssu\_ps.c>**

#include "ssu\_ps.h"

struct winsize term; //터미널 크기

proc\_list \*head;

long mem\_tot; //총 메모리 kb

time\_t boot\_time; //오에스 부팅 타임

double uptime;

int a\_flag, u\_flag, x\_flag;

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

{

ioctl(STDOUT\_FILENO, TIOCGWINSZ, &term);

// printf("lines : %d\n", term.ws\_row);

// printf("columns %d\n", term.ws\_col);

if (argc > 2) {

fprintf(stderr, "usage : ./pps <option>\n");

exit(1);

}

if (argc == 2 && check\_option(argv) < 0) {

fprintf(stderr, "option error in pps\n");

exit(1);

}

boot\_time = get\_boot\_time();

mem\_tot = get\_total\_memory();

head = create\_node(NULL);

scan\_proc();

print\_ps();

free\_list(head);

exit(0);

}

int check\_option(char \*\*argv)

{

int len = strlen(argv[1]);

int i;

for (i = 0; i < len; i++) {

switch (argv[1][i]) {

case 'a' :

a\_flag = true;

break;

case 'u' :

u\_flag = true;

break;

case 'x' :

x\_flag = true;

break;

default :

return -1;

}

}

return 0;

}

void scan\_proc(void)

{

int count;

struct dirent \*\*dirlist;

struct passwd userpwd;

struct stat status;

char path[PATHLEN];

char line[LINEMAX];

char tmp[NAMELEN];

proc\_list pstat;

proc\_list \*node;

if ((count = scandir("/proc", &dirlist, scandir\_filter, NULL)) < 0) {

fprintf(stderr, "scandir error in /proc\n");

exit(1);

}

for (int i = 0; i < count && \_isdigit(dirlist[i]->d\_name); i++) {

memset(path, 0, PATHLEN);

sprintf(path, "%s/%s/%s", "/proc", dirlist[i]->d\_name, "stat");

FILE \*fp = fopen(path, "r");

memset(&pstat, 0, sizeof(proc\_list));

//stat파일에서 정보 읽어오기

fscanf(fp, "%d %s %c %d %d %d %d %d %\*u %\*u %\*u %\*u %\*u %lu %lu"

"%\*d %\*d %\*d %ld %ld %\*d %llu %lu %ld",

&pstat.pid, pstat.cmdline, &pstat.state, &pstat.ppid, &pstat.pgrp,

&pstat.sid, &pstat.tty\_nr, &pstat.tpgid, &pstat.utime, &pstat.stime,

&pstat.nice, &pstat.num\_threads, &pstat.starttime,

&pstat.vsize, &pstat.rss);

fclose(fp);

//uid에서 사용자 이름 구하기

memset(path, 0, PATHLEN);

sprintf(path, "%s/%s", "/proc", dirlist[i]->d\_name);

stat(path, &status);

userpwd = \*getpwuid(status.st\_uid);

strcpy(pstat.name, userpwd.pw\_name);

//cmdline 구하기

memset(path, 0, PATHLEN);

memset(tmp, 0, PATHLEN);

memset(line, 0, LINEMAX);

sprintf(path, "%s/%s/%s", "/proc", dirlist[i]->d\_name, "cmdline");

fp = fopen(path, "r");

fread(line, LINEMAX, 1, fp);

for (int i = 0; i < NAMELEN; i++) {

if (line[i] == 0) {

if (i == LINEMAX - 1)

break;

else if (line[i + 1] == 0)

break;

else

line[i] = ' ';

}

}

if (strlen(line) == 0) {

pstat.cmdline[0] = '[';

pstat.cmdline[strlen(pstat.cmdline) - 1] = ']';

}

else {

memset(pstat.cmdline, 0, NAMELEN);

strcpy(pstat.cmdline, line);

}

// printf("pid : %d ppid : %d tty\_nr : %d state : %c cmdline : %s\n"

// "name : %s utime : %lu stime : %lu vsize : %lu\n"

// "rss : %ld starttime : %llu\n", pstat.pid, pstat.ppid,

// pstat.tty\_nr, pstat.state, pstat.cmdline, pstat.name,

// pstat.utime, pstat.stime, pstat.vsize, pstat.rss, pstat.starttime);

//리스트에 저장하기

node = create\_node(&pstat);

add\_node(node);

}

for (int i = 0; i < count; i++)

free(dirlist[i]);

free(dirlist);

}

void print\_ps(void)

{

if (!a\_flag && !u\_flag && !x\_flag)

print\_default(); //no option

else if (!x\_flag)

print\_without\_x(); //option a, (a)u

else if (!a\_flag)

print\_without\_a(); //option x, ux

else

print\_all(); //option ax, aux

}

void print\_default(void)

{

proc\_list \*tmp = head->next;

int bash\_pid = getppid();

int bash\_tty = -1;

char bash\_name[64];

char headline[LINEMAX];

char tmp\_line[LINEMAX];

char line[LINEMAX];

char tty[8];

struct tm t;

time\_t runtime;

memset(headline, 0, LINEMAX);

memset(line, 0, LINEMAX);

sprintf(headline, "%6s %-7s %8s %s", "PID", "TTY", "TIME", "CMD");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

if (tmp->pid == bash\_pid) {

memset(bash\_name, 0, 64);

strcpy(bash\_name, tmp->name);

bash\_tty = tmp->tty\_nr;

}

tmp = tmp->next;

}

get\_tty(bash\_tty, tty);

tmp = head->next;

while (tmp != NULL) {

if (tmp->tty\_nr == bash\_tty && !strcmp(tmp->name, bash\_name)) {

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%6d %-7s %02d:%02d:%02d ", tmp->pid, tty,

t.tm\_hour, t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

}

tmp = tmp->next;

}

}

void print\_without\_x(void)

{

proc\_list \*tmp = head->next;

int bash\_pid = getppid();

char bash\_name[64];

char headline[LINEMAX];

char tmp\_line[LINEMAX];

char line[LINEMAX];

char tty[8];

char state[6];

double cpu\_per, mem\_per;

struct tm t, t2;

time\_t runtime;

time\_t starttime;

while (tmp != NULL) {

if (tmp->pid == bash\_pid) {

memset(bash\_name, 0, 64);

strcpy(bash\_name, tmp->name);

}

tmp = tmp->next;

}

tmp = head->next;

if (u\_flag) {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%-9s %6s %4s %4s %7s %7s %-7s %-5s %5s %5s %s",

"USER", "PID", "%CPU", "%MEM", "VSZ", "RSS", "TTY",

"STAT", "START", "TIME", "COMMAND");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

get\_tty(tmp->tty\_nr, tty);

if (strcmp(tty, "?") && !strcmp(tmp->name, bash\_name)) {

uptime = get\_uptime();

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

cpu\_per = runtime / uptime \* 100;

mem\_per = (double)tmp->rss \* 4 / mem\_tot \* 100;

starttime = tmp->starttime / sysconf(\_SC\_CLK\_TCK) + boot\_time;

t2 = \*localtime(&starttime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%-9s %6d %4.1lf %4.1lf %7lu %7ld "

"%-7s %-5s %02d:%02d %02d:%02d ",

tmp->name, tmp->pid, cpu\_per, mem\_per, tmp->vsize / 1024,

tmp->rss \* 4, tty, state, t2.tm\_hour, t2.tm\_min,

t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

}

tmp = tmp->next;

}

}

else {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%6s %-7s %-5s %5s %s", "PID", "TTY", "STAT", "TIME", "CMD");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

get\_tty(tmp->tty\_nr, tty);

if (strcmp(tty, "?") && !strcmp(tmp->name, bash\_name)) {

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%6d %-7s %-5s %02d:%02d ", tmp->pid, tty,

state, t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

}

tmp = tmp->next;

}

}

}

void print\_without\_a(void)

{

proc\_list \*tmp = head->next;

int bash\_pid = getppid();

char bash\_name[64];

char headline[LINEMAX];

char tmp\_line[LINEMAX];

char line[LINEMAX];

char tty[8];

char state[6];

double cpu\_per, mem\_per;

struct tm t, t2;

time\_t runtime, starttime;

while (tmp != NULL) {

if (tmp->pid == bash\_pid) {

memset(bash\_name, 0, 64);

strcpy(bash\_name, tmp->name);

}

tmp = tmp->next;

}

tmp = head->next;

if (u\_flag) {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%-9s %6s %4s %4s %7s %7s %-7s %-5s %5s %5s %s",

"USER", "PID", "%CPU", "%MEM", "VSZ", "RSS", "TTY",

"STAT", "START", "TIME", "COMMAND");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

if (!strcmp(tmp->name, bash\_name)) {

get\_tty(tmp->tty\_nr, tty);

uptime = get\_uptime();

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

cpu\_per = runtime / uptime \* 100;

mem\_per = (double)tmp->rss \* 4 / mem\_tot \* 100;

starttime = tmp->starttime / sysconf(\_SC\_CLK\_TCK) + boot\_time;

t2 = \*localtime(&starttime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%-9s %6d %4.1lf %4.1lf %7lu %7ld "

"%-7s %-5s %02d:%02d %02d:%02d ",

tmp->name, tmp->pid, cpu\_per, mem\_per, tmp->vsize / 1024,

tmp->rss \* 4, tty, state, t2.tm\_hour, t2.tm\_min,

t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

}

tmp = tmp->next;

}

}

else {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%6s %-7s %-5s %5s %s", "PID", "TTY", "STAT", "TIME", "CMD");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

if (!strcmp(tmp->name, bash\_name)) {

get\_tty(tmp->tty\_nr, tty);

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%6d %-7s %-5s %02d:%02d ", tmp->pid, tty,

state, t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

}

tmp = tmp->next;

}

}

}

void print\_all(void)

{

proc\_list \*tmp = head->next;

char headline[LINEMAX];

char tmp\_line[LINEMAX];

char line[LINEMAX];

char tty[8];

char state[6];

double cpu\_per, mem\_per;

struct tm t, t2;

time\_t runtime, starttime;

if (u\_flag) {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%-9s %6s %4s %4s %7s %7s %-7s %-5s %5s %5s %s",

"USER", "PID", "%CPU", "%MEM", "VSZ", "RSS", "TTY",

"STAT", "START", "TIME", "COMMAND");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

get\_tty(tmp->tty\_nr, tty);

uptime = get\_uptime();

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

cpu\_per = runtime / uptime \* 100;

mem\_per = (double)tmp->rss \* 4 / mem\_tot \* 100;

starttime = tmp->starttime / sysconf(\_SC\_CLK\_TCK) + boot\_time;

t2 = \*localtime(&starttime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%-9s %6d %4.1lf %4.1lf %7lu %7ld "

"%-7s %-5s %02d:%02d %02d:%02d ",

tmp->name, tmp->pid, cpu\_per, mem\_per, tmp->vsize / 1024,

tmp->rss \* 4, tty, state, t2.tm\_hour, t2.tm\_min,

t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

tmp = tmp->next;

}

}

else {

memset(line, 0, LINEMAX);

memset(headline, 0, LINEMAX);

sprintf(headline, "%6s %-7s %-5s %5s %s", "PID", "TTY", "STAT", "TIME", "CMD");

strncpy(line, headline, term.ws\_col);

printf("%s\n", line);

while (tmp != NULL) {

get\_tty(tmp->tty\_nr, tty);

runtime = tmp->stime + tmp->utime;

runtime /= sysconf(\_SC\_CLK\_TCK);

t = \*gmtime(&runtime);

get\_state(tmp, state);

memset(line, 0, LINEMAX);

memset(tmp\_line, 0, LINEMAX);

sprintf(tmp\_line, "%6d %-7s %-5s %02d:%02d ", tmp->pid, tty,

state, t.tm\_min, t.tm\_sec);

strcat(tmp\_line, tmp->cmdline);

strncpy(line, tmp\_line, term.ws\_col);

printf("%s\n", line);

tmp = tmp->next;

}

}

}

proc\_list \*create\_node(proc\_list\* stat)

{

proc\_list \*new;

new = (proc\_list \*)malloc(sizeof(proc\_list));

if (stat == NULL)

memset(new, 0, sizeof(proc\_list));

else

memcpy(new, stat, sizeof(proc\_list));

new->next = NULL;

return new;

}

void add\_node(proc\_list \*node)

{

proc\_list \*tmp = head;

while (tmp->next != NULL)

tmp = tmp->next;

tmp->next = node;

}

void print\_list(void) //리스트 확인

{

proc\_list \*pstat = head->next;

while (pstat != NULL) {

printf("pid : %d ppid : %d tty\_nr : %d state : %c cmdline : %s "

"name : %s utime : %lu stime : %lu vsize : %lu "

"rss : %ld starttime : %llu\n", pstat->pid, pstat->ppid,

pstat->tty\_nr, pstat->state, pstat->cmdline, pstat->name,

pstat->utime, pstat->stime, pstat->vsize, pstat->rss, pstat->starttime);

pstat = pstat->next;

}

}

void free\_list(proc\_list \*tmp)

{

if (tmp == NULL) return;

else free\_list(tmp->next);

free(tmp);

}

time\_t get\_boot\_time()

{

FILE \*fp = fopen("/proc/stat", "r");

time\_t res;

for (int i = 0; i < 5; i++) {

fscanf(fp, "%\*[^\n]\n");

}

fscanf(fp, "%\*s %lu", &res);

fclose(fp);

return res;

}

long get\_total\_memory(void)

{

FILE \*fp = fopen("/proc/meminfo", "r");

long res;

fscanf(fp, "%\*s %ld", &res);

fclose(fp);

return res;

}

double get\_uptime(void)

{

FILE \*fp = fopen("/proc/uptime", "r");

double res;

fscanf(fp, "%lf", &res);

fclose(fp);

return res;

}

void get\_tty(int tty\_nr, char \*tty)

{

int major, minor;

major = MAJOR(tty\_nr);

minor = MINOR(tty\_nr);

memset(tty, 0, 8);

if (major == 4) {

if (minor < 64)

sprintf(tty, "%s%d", "tty", minor);

else

sprintf(tty, "%sS%d", "tty", minor - 64);

}

else if (major >= 136 && major <= 143) {

sprintf(tty, "%s/%d", "pts", minor);

}

else

sprintf(tty, "%s", "?");

}

void get\_state(proc\_list \*tmp, char \*state) // L state(check locked page)의 경우 smaps에서 확인할 수 있다는

{ // 것 까지는 확인하였으나 확인하려면 CONFIG\_PROC\_PAGE\_MONITOR를

int idx = 0; // enable 시켜야 된다는데, 방법을 모르겠어서 구현을 못했습니다..

memset(state, 0, 6);

state[idx++] = tmp->state;

if (tmp->nice < -4) //priority low, high 기준을 정확히 못찾겠어서 임의로 정했습니다.

state[idx++] = '<'; //5 이상 low, -5 이하 high로 나타나는걸로 보였습니다 (제 pc 기준)

else if (tmp->nice > 4)

state[idx++] = 'N';

if (tmp->pid == tmp->sid)

state[idx++] = 's';

if (tmp->num\_threads > 1)

state[idx++] = 'l';

if (tmp->pgrp == tmp->tpgid)

state[idx++] = '+';

}

int scandir\_filter(const struct dirent \*file)

{

char name[NAMELEN] = {0, };

strcpy(name, file->d\_name);

if (\_isdigit(name)) return true;

else return false;

}

int \_isdigit(char \*str)

{

int len = strlen(str);

int i;

for (i = 0; i < len; i++)

if (!isdigit(str[i]))

return false;

return true;

}

<ssu\_top.h>

&

<ssu\_top.c> 🡺 미 구현, 구현 시 추가 예정입니다..