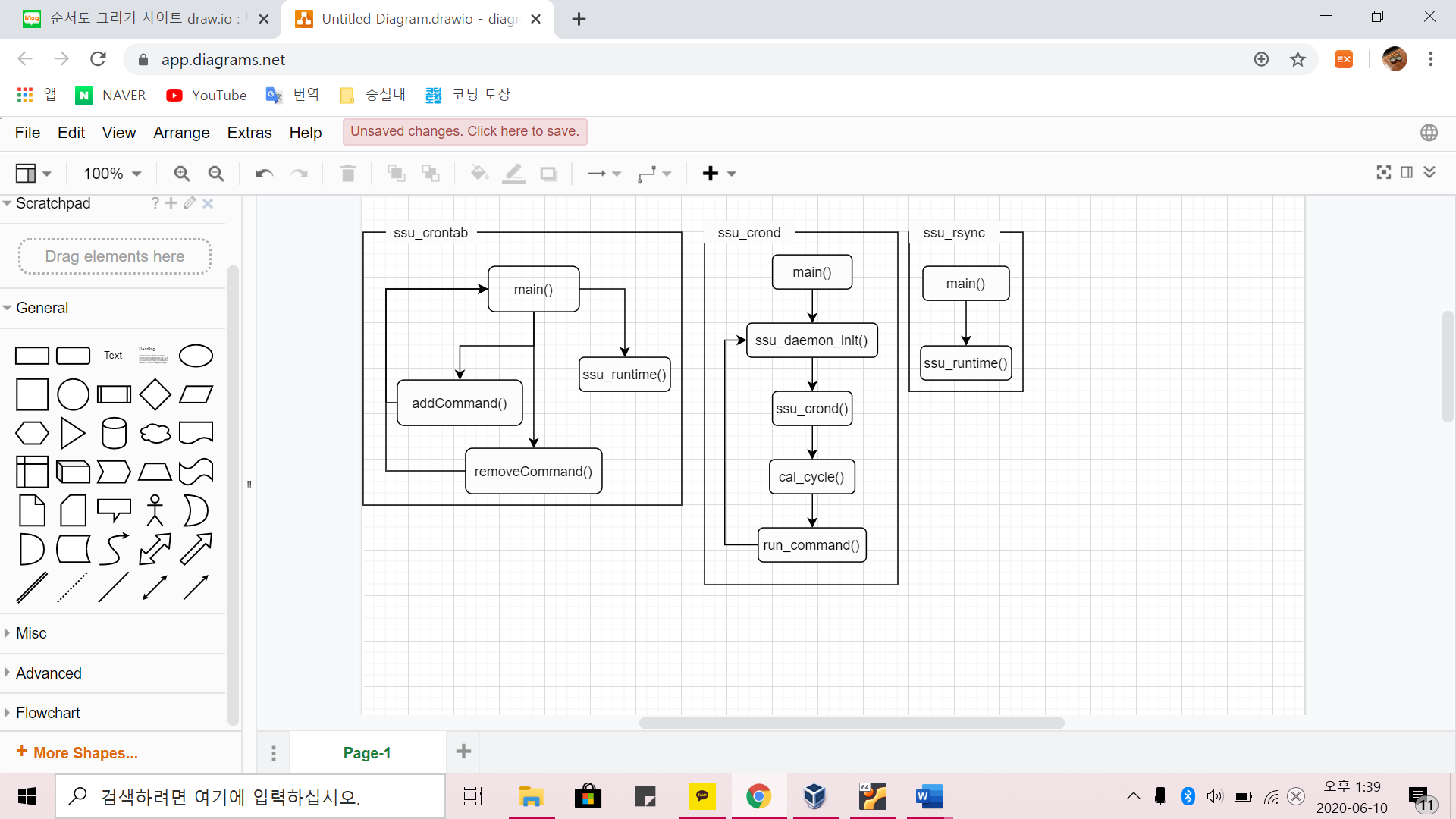
컴퓨터학부 20172626 박예진

139813

1. 개요

ssu\_crontab 프로그램은 사용자가 주기적으로 실행하고자하는 명령어를 추가 삭제하는 프로그램이며 ssu\_crond는 디몬프로그램으로 백그라운드에서 돌아가며 ssu\_crontab프로그램을 통해 등록한 명령어를 주기적으로 실행하고 log를 기록하는 함수이다. 또한 ssu\_rsync는 사용자가 동기화를 원하는 파일이나 디렉토리를 목적 디렉토리에 동기화 시켜주고 log를 기록하는 프로그램이다.

2. 설계 (함수 흐름도)



ssu\_crontab : addCommand()를 통해 ssu\_crontab\_file에 명령어를 등록하고 removeCommand()를 통해 ssu\_crontab\_file에 등록된 명령어를 삭제한다. ssu\_runtime()을 통해 프로그램의 수행시간을 측정하고 출력한다.

ssu\_crond : ssu\_daemon\_init()을 통해 디몬 규칙을 따르는 프로세스를 생성하고 ssu\_crond를 무한 반복 호출한다. 프로그램은 백그라운드에서 지속수행되며 ssu\_crontab\_file에서 등록된 명령어와 명령어의 주기를 불러와 cal\_cycle()을 통해 주기를 계산하고 지정된 시간에 run\_command()를 통해 명령어를 수행한다. 그 후 ssu\_crontab\_log에 명령어가 수행되었음을 기록한다.

ssu\_rsync : 사용자가 지정한 src 파일이나 디렉토리를 목적디렉토리 dst에 동기화 한다. ssu\_runtime()을 통해 프로그램의 수행시간을 측정하고 출력한다.

3. 구현

**<ssu\_crontab.c>**

**int main(void);**

프롬프트를 출력하고 add, remove, exit 명령어를 수행한다.

add 명령어와 함께 등록할 명령어 입력시 ssu\_crontab\_file에 명령어를 등록하고 remove명령어와 함께 삭제할 명령 어의 번호를 입력시 ssu\_crontab\_file에 등록된 명령어가 삭제된다. exit명령어 입력시 프로그램이 종료된다.

**void addCommand(char \* inputstr, FILE \*fp);**

입력된 명령어를 ssu\_crontab\_file에 등록하고 ssu\_crontab\_log에 명령어가 추가되었음을 기록한다.

**void removeCommand(char \* inputstr, FILE \*fp);**

ssu\_crontab\_file에서 명령어를 삭제하고, ssu\_crontab\_log에 명령어가 삭제되었음을 기록한다.

**void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t);**

프로그램의 수행시간 측정

**<ssu\_crond.c>**

**int cti(char c);** 문자를 정수로 변환해주는 함수 (int)c-48을 리턴한다.

**int ssu\_daemon\_init(void);**

디몬규칙에 따라 프로세스를 생성하는 함수

무한 반복으로 ssu\_crond()를 호출한다.

**void ssu\_crond(void);**

등록된 명령어의 개수만큼 cal\_cycle()을 호출한다.

**void cal\_cycle(char \* line);**

명령어의 실행 주기를 계산하는 함수

**void run\_command(char \* passline, char \* line);**

명령어의 실행 주기와 현재 시간이 일치하면 명령어를 수행하고 ssu\_crontab\_log에 명령어가 실행되었음을 기록

**<ssu\_rsync.c>**

**void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t);**

프로그램의 수행시간을 측정하는 함수

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

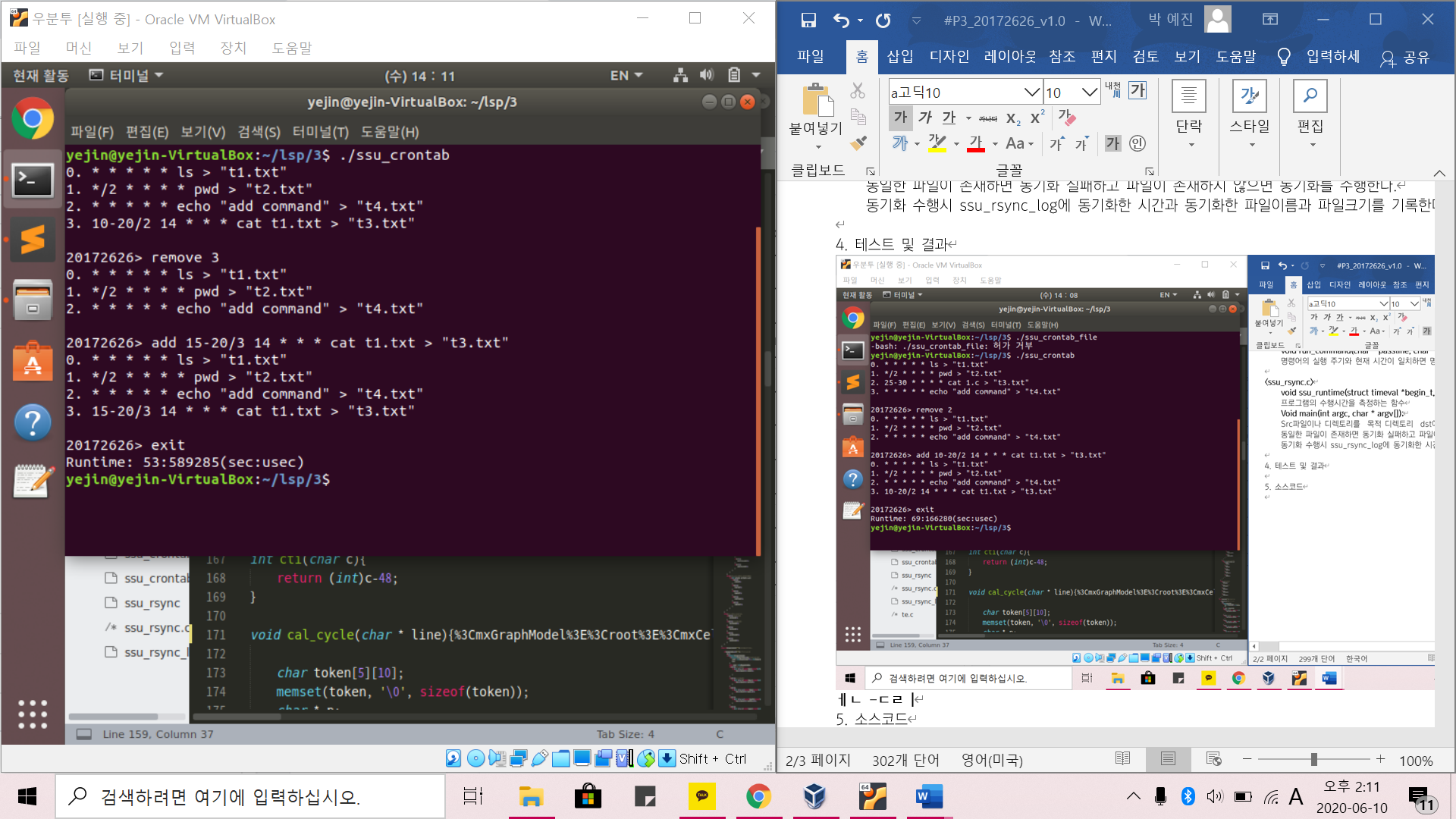
Src파일이나 디렉토리를 목적 디렉토리 dst에 동기화하는 함수,

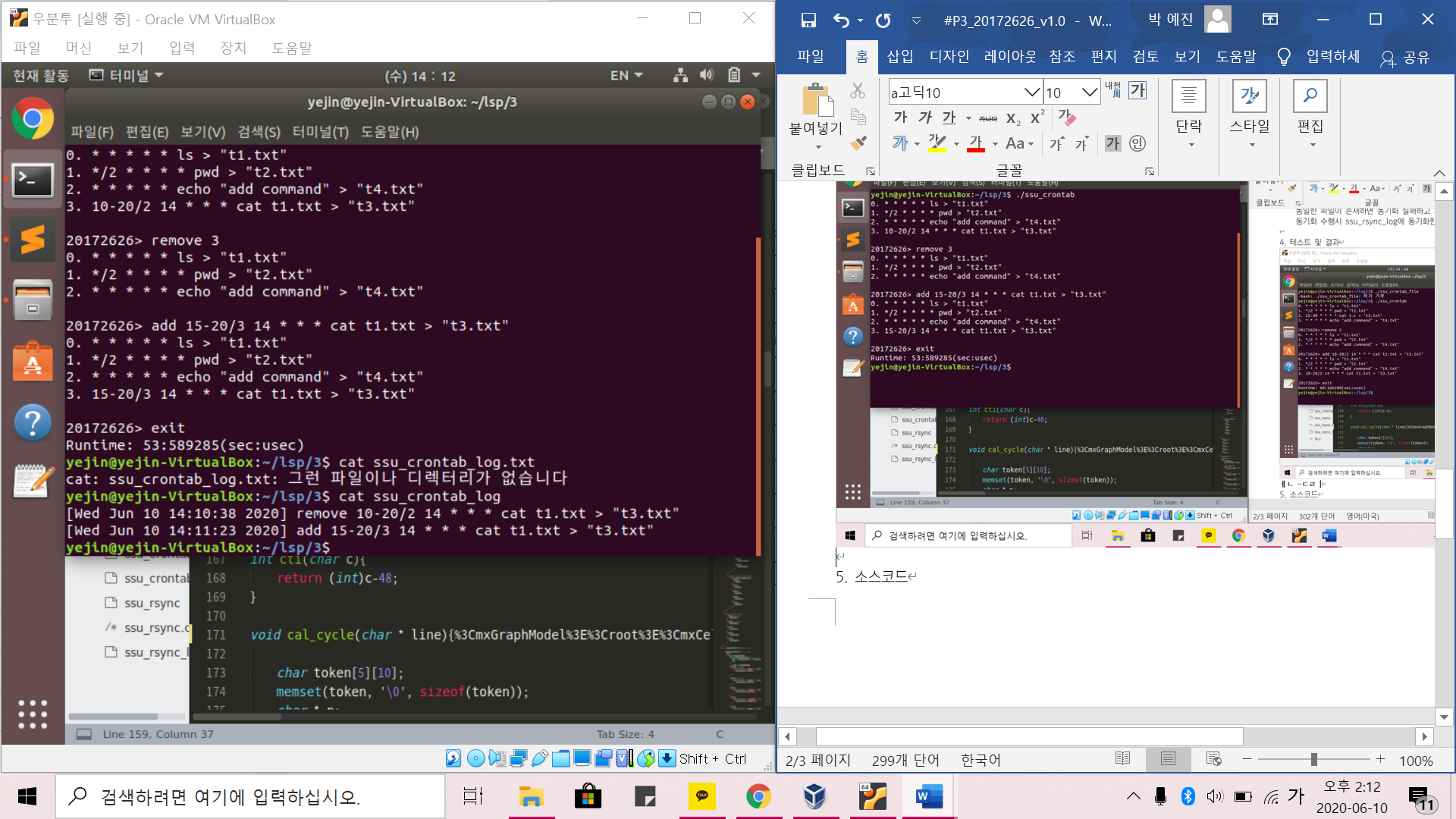
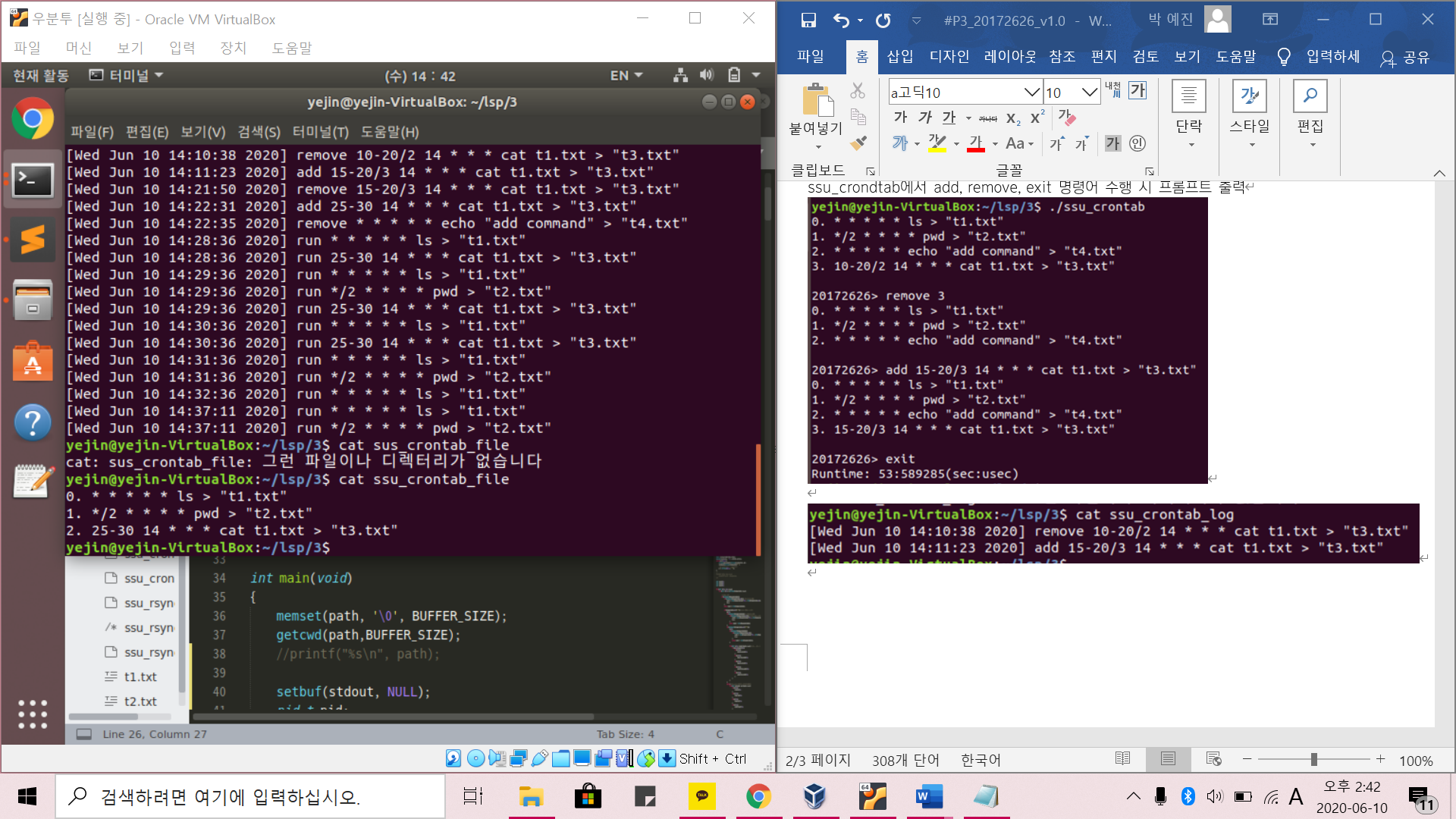
동일한 파일이 존재하면 동기화 실패하고 파일이 존재하지 않으면 동기화를 수행한다.

동기화 수행시 ssu\_rsync\_log에 동기화한 시간과 동기화한 파일이름과 파일크기를 기록한다.

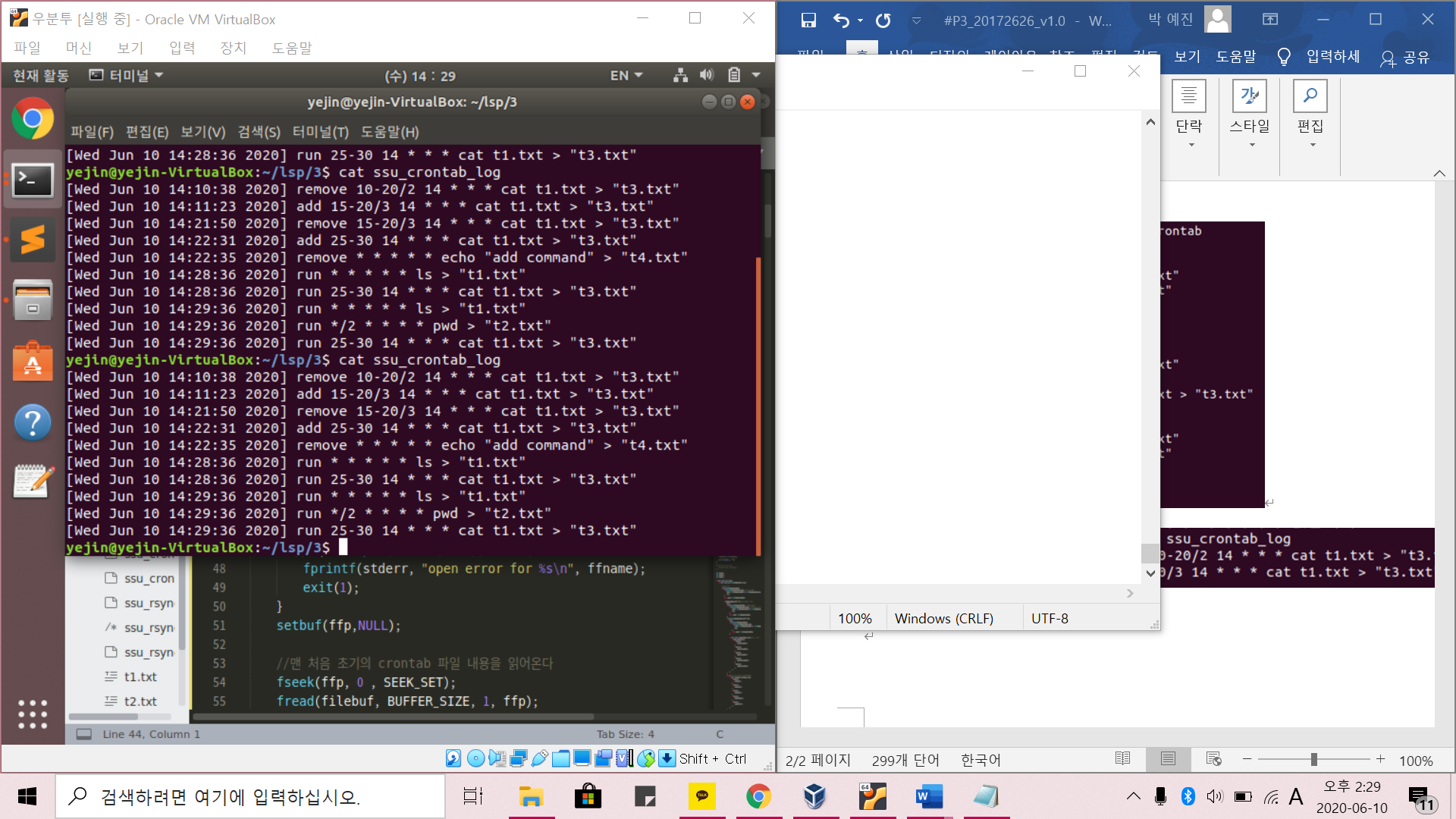
4. 테스트 및 결과

ssu\_crondtab에서 add, remove, exit 명령어 수행 시 프롬프트 출력

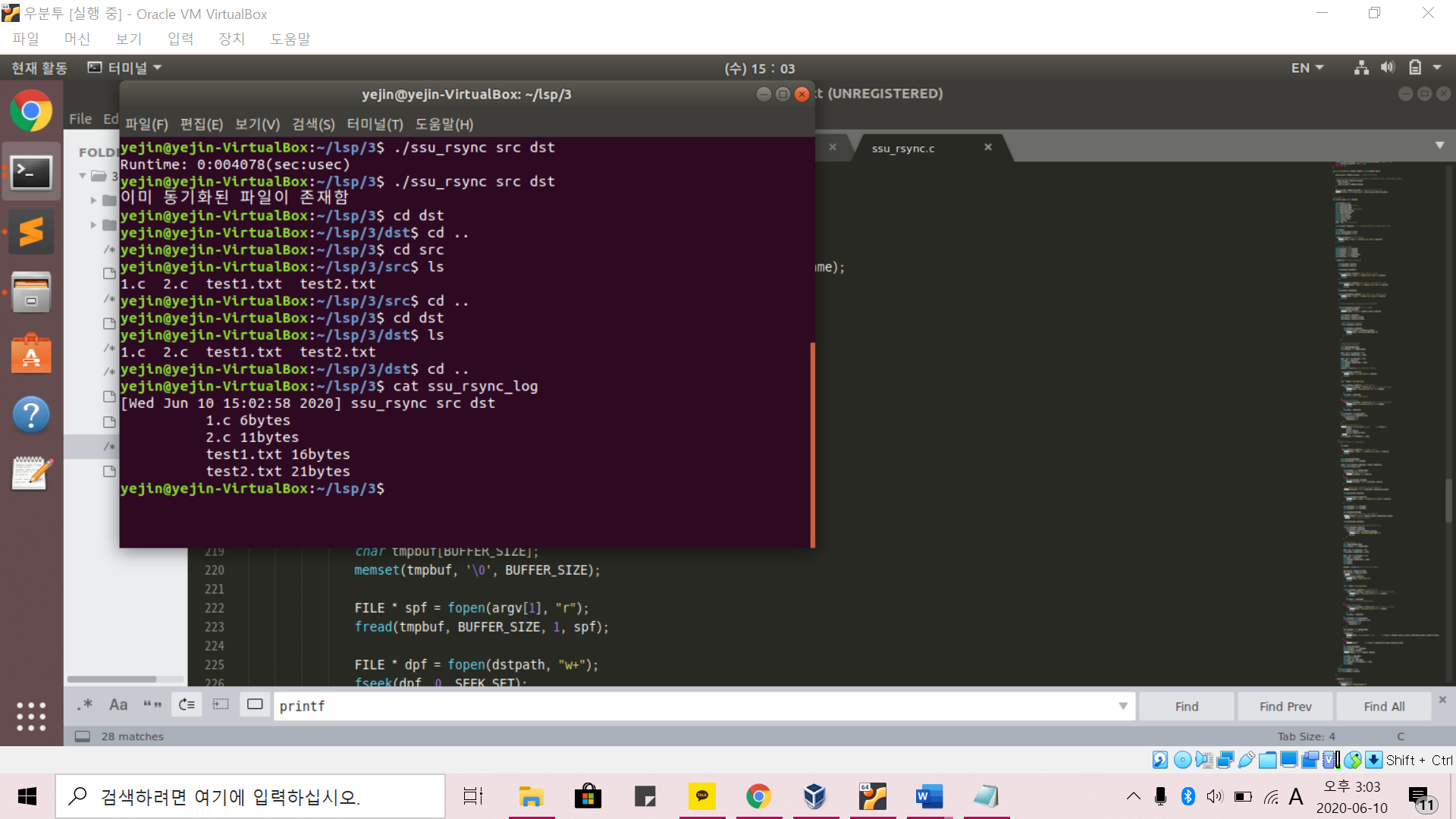


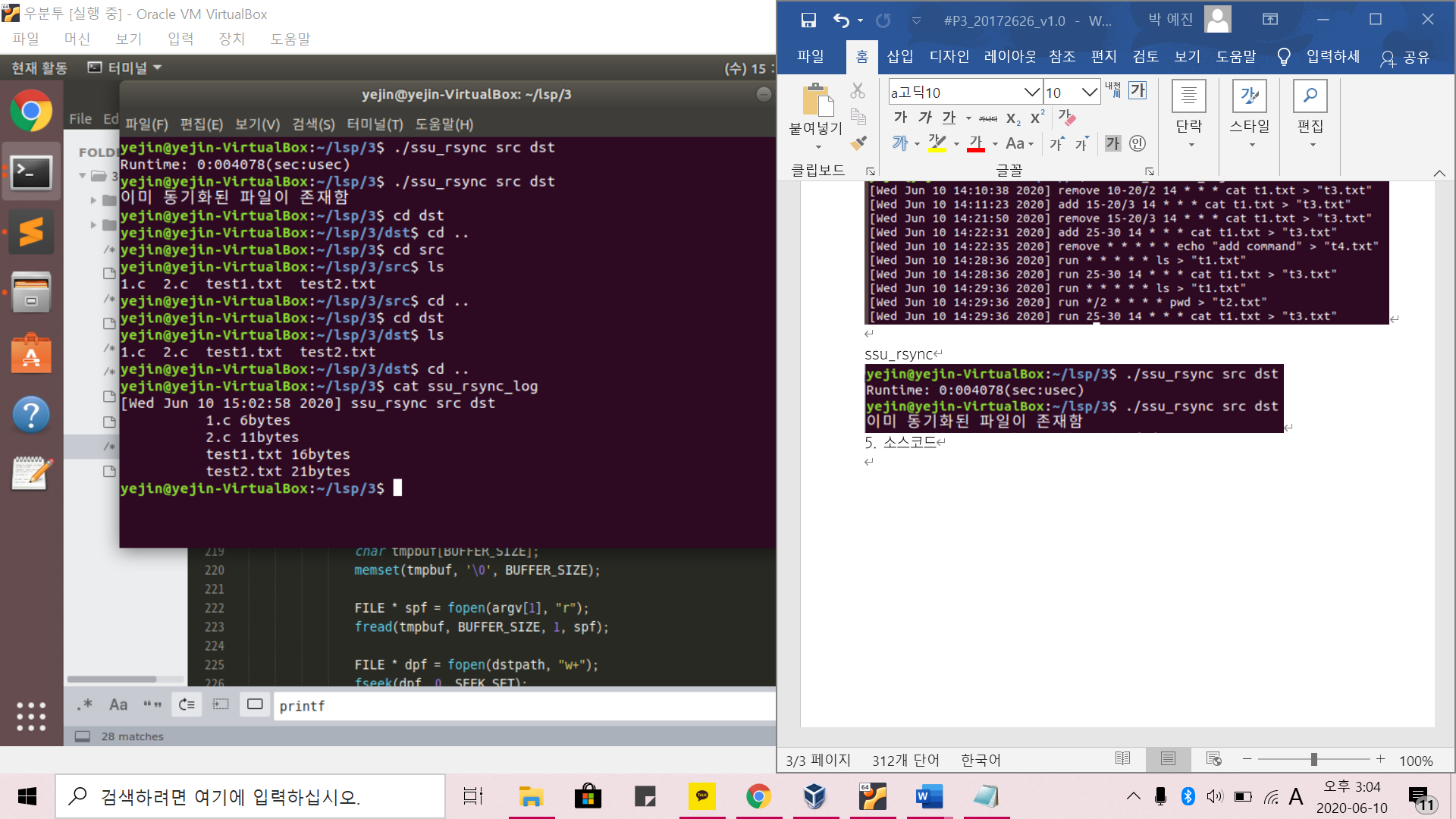


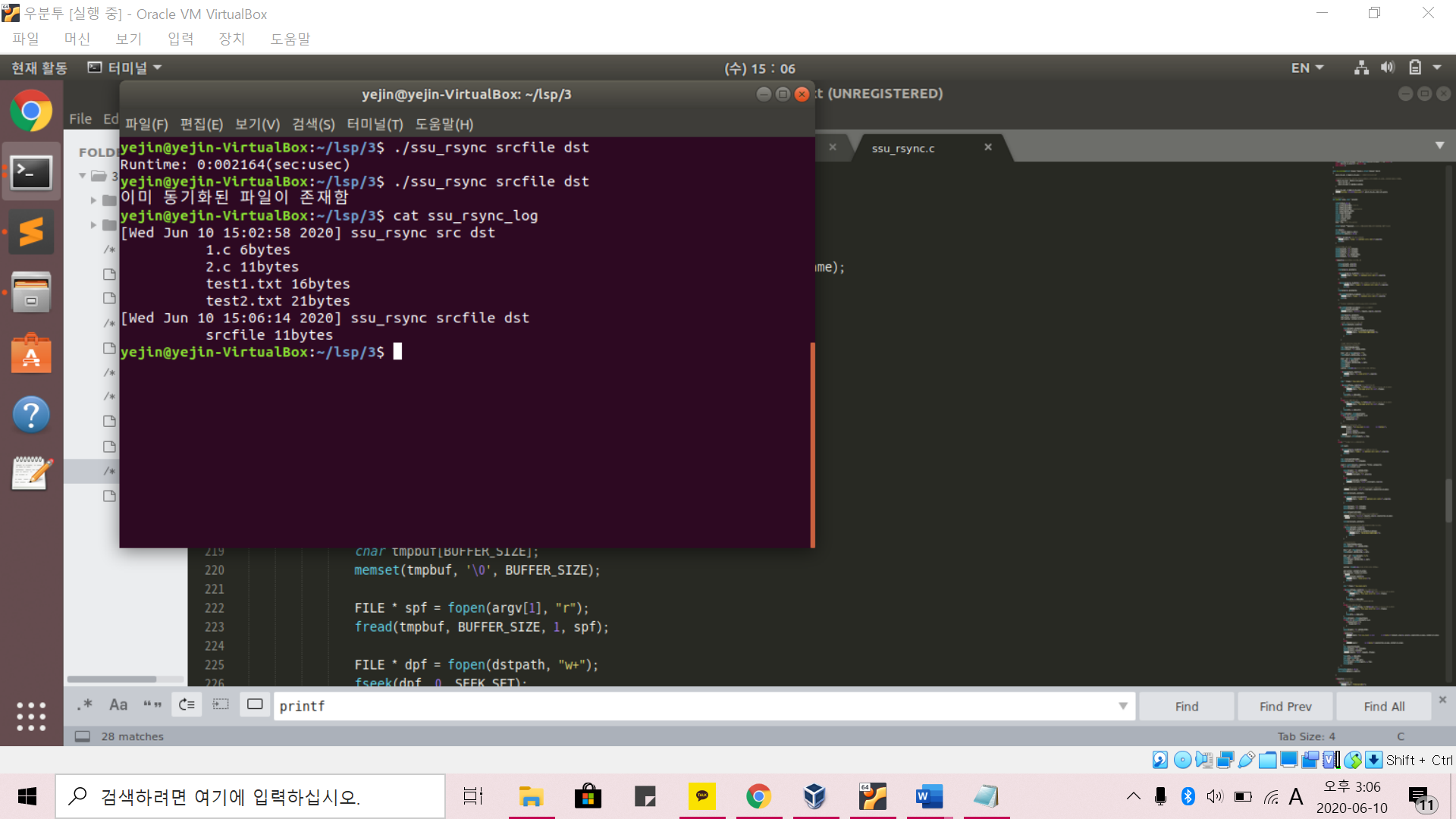
ssu\_crond 실행시 log기록



ssu\_rsync







5. 소스코드

**<ssu\_crontab.h>**

#define BUFFER\_SIZE 1024

#define MAXLEN 500 //명령어 저장 최대 길이

#define LINEMAX 100 //명령어 최대 갯수

void addCommand(char \* inputstr, FILE \*fp);

void removeCommand(char \* inputstr, FILE \*fp);

**<ssu\_crontab.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <ctype.h>

#include <sys/time.h>

#include <time.h>

#include "ssu\_crontab.h"

#define SECOND\_TO\_MICRO 1000000

#define PATHLEN 1024

FILE \* lfp;

/\*ssu\_crontab.c\*/

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

{

end\_t->tv\_sec -= begin\_t->tv\_sec; // 수행시간이 몇 초인지 계산

//수행시간의 마이크로초보다 수행 시작 시각의 마이크로초가 크면 수행시간을 1초 줄이고, 1000000마이크로초 더해준다.

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

}

end\_t->tv\_usec -= begin\_t->tv\_usec; //수행시간이 몇 마이크로초인지 계산

printf("Runtime: %ld:%06ld(sec:usec)\n", end\_t->tv\_sec, end\_t->tv\_usec);

}

int main(void){

struct timeval begin\_t, end\_t;

FILE \*fp;

char \* fname = "ssu\_crontab\_file";

char filebuf[BUFFER\_SIZE];

char inputstr[MAXLEN];

gettimeofday(&begin\_t, NULL);

memset(filebuf, '\0', BUFFER\_SIZE);

//ssu\_crontab\_file이 존재하면

if(!access(fname, F\_OK)){

//파일 내용을 읽어와 출력

if((fp = fopen(fname, "r+"))==NULL){//파일을 "r+"모드로 오픈 (O\_RDWR)

fprintf(stderr, "file open error for %s\n", fname);

exit(1);

}

setbuf(fp, NULL);

fread(filebuf, BUFFER\_SIZE, 1 , fp);//파일에서 내용을 읽어와 버퍼에 저장하고

if(strlen(filebuf)!=0){

printf("%s\n", filebuf);//화면에 출력한다

}

}

char \*lfname = "ssu\_crontab\_log";//로그 파일 이름

if(access(lfname,F\_OK)!=0){//로그파일이 존재하지 않으므로 파일 생성

if((lfp=fopen(lfname,"w+"))==NULL){

fprintf(stderr, "1open error for %s\n", lfname);

exit(1);

}

}

else{//로그 파일이 이미 존재할 때

if((lfp = fopen(lfname, "a+"))==NULL){

fprintf(stderr, "2open error for %s\n", lfname);

exit(1);

}

}

setbuf(lfp,NULL);

while(1){

memset(inputstr, '\0', MAXLEN);//입력받은 문자열을 저장할 배열 초기화

printf("20172626> ");

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

getchar();

//엔터만 입력시 프롬프트 재출력

if(inputstr[0]=='\n'){

continue;

}

if(strncmp(inputstr, "add", 3)==0){

if(access(fname,F\_OK)==-1){//ssu\_crontab\_file이 존재하지 않으면 생성

if((fp=fopen(fname, "w+"))==NULL){

fprintf(stderr, "file open error for %s\n", fname);

exit(1);

}

setbuf(fp,NULL);

}

addCommand(inputstr, fp);

continue;

}

if(strncmp(inputstr, "remove", 6)==0){

removeCommand(inputstr, fp);

continue;

}

if(strncmp(inputstr, "exit", 4)==0){

fclose(fp);

gettimeofday(&end\_t, NULL);

ssu\_runtime(&begin\_t, &end\_t);

exit(0);

}

else{

fprintf(stderr, "잘못된 명령어 입니다.\n");

continue;

}

}

}

void addCommand(char \*inputstr, FILE \* fp){

time\_t atime;

char filebuf[BUFFER\_SIZE];

char addbuf[MAXLEN];

int lcount=0;//명령어의 개수를 세는 변수

char checkcycle[5][10]; //주기 저장 배열

char \*ptr;

//배열 초기화

memset(filebuf, '\0', BUFFER\_SIZE);

memset(addbuf, '\0', MAXLEN);

fseek(fp, 0, SEEK\_SET); //오프셋을 파일 앞으로 이동시킴

fread(filebuf, BUFFER\_SIZE, 1, fp); //파일의 내용을 읽어옴

if(strlen(filebuf)==0){

lcount=0;

}

else{

//lcount+=1;

for(int i=0; i<strlen(filebuf); i++){ //파일의 개행 수를 계산해 lcount변수에 저장

if(filebuf[i]=='\n'){

lcount++;

}

}

}

char linebuf[BUFFER\_SIZE];

memset(linebuf, '\0', BUFFER\_SIZE);

strcpy(linebuf, inputstr+4);

//기록 형식에 따라 문자열 제작 (포인터 문자열 배열을 먼저 저장을 해놓음) // 나중에 해놓으면 왜 안되지...?

sprintf(addbuf, "%d. %s", lcount, inputstr +4);

//printf("%s\n", addbuf);

//입력받은 주기를 자른다

char \* str = inputstr +4;

ptr = strtok(str, " ");

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

strcpy(checkcycle[i], ptr);

ptr = strtok(NULL, " ");

}

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

printf("%s\n", checkcycle[i]);

}\*/

int n;

//입력받은 주기가 올바른지 검사

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

for(int j=0; j<strlen(checkcycle[i]); j++){

if(isdigit(checkcycle[i][j])) {

if(isdigit(checkcycle[i][j+1])){

n = ((int)checkcycle[i][j]-48)\*10 + (int)checkcycle[i][j+1]-48;

//printf("%d %d\n",((int)checkcycle[i][j]-48)\*10, (int)checkcycle[i][j+1]-48);

j++;

}

else{

n = (int)checkcycle[i][j]-48;

}

if(i==0){

if(n<0||n>59) {

fprintf(stderr, "분 실행 주기가 잘못 입력되었습니다.\n");

return;

}

else continue;

}

else if(i==1){

if(n<0||n>23) {

fprintf(stderr, "시 실행 주기가 잘못 입력되었습니다.\n");

return;

}

else continue;

}

else if(i==2){

if(n<1||n>31) {

fprintf(stderr, "일 실행 주기가 잘못 입력되었습니다.\n");

return;

}

else continue;

}

else if(i==3){

if(n<1||n>12) {

fprintf(stderr, "월 실행 주기가 잘못 입력되었습니다.\n");

return;

}

else continue;

}

else if(i==4){ //0은 일요일 1-6 순으로 월요일에서 토요일

if(n<0||n>6) {

fprintf(stderr, "요일 실행 주기가 잘못 입력되었습니다.\n");

return;

}

else continue;

}

else continue;

}

else if(checkcycle[i][j]=='\*') continue;

else if(checkcycle[i][j]=='-') continue;

else if(checkcycle[i][j]==',') continue;

else if(checkcycle[i][j]=='/') continue;

else{

fprintf(stderr, "실행 주기가 잘못 입력되었습니다.\n");

return;

}

}

}

if((strlen(filebuf)>0)&&(filebuf[strlen(filebuf)-1]!='\n')){

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

}

//파일에 실행주기와 명령어 기록

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

memcpy(&filebuf[strlen(filebuf)], addbuf, strlen(addbuf));

fseek(fp,0, SEEK\_SET);

atime= time(NULL);

char timebuf[BUFFER\_SIZE];

memset(timebuf, '\0', BUFFER\_SIZE);

strcpy(timebuf, ctime(&atime));

timebuf[strlen(timebuf)-1]=0;

fwrite(filebuf, BUFFER\_SIZE, 1, fp);//file.txt에 명령어 작성

char logbuf[BUFFER\_SIZE];

memset(logbuf, '\0', BUFFER\_SIZE);

sprintf(logbuf, "[%s] add %s\n", timebuf, linebuf);

fseek(lfp, 0, SEEK\_END);

fwrite(logbuf, strlen(logbuf), 1, lfp);

printf("%s\n", filebuf);//화면에 출력한다

}

void removeCommand(char \* inputstr, FILE \*fp){

time\_t rtime;

int num;

char filebuf[BUFFER\_SIZE];

char linebuf[LINEMAX][MAXLEN];

char tmpbuf[BUFFER\_SIZE];

int lcount=0;

char \* ptr;

if(isdigit(\*&inputstr[7])){

if(isdigit(\*&inputstr[8])){

num = (char)((\*&inputstr[7])\*10)-48 + \*&inputstr[8]-48;

}

else{

num = (char)(\*&inputstr[7]-48);

}

}

else{

fprintf(stderr, "잘못된 COMMAND\_NUMBER\n");

return;

}

memset(filebuf, '\0', BUFFER\_SIZE);

memset(tmpbuf, '\0', BUFFER\_SIZE);

fseek(fp, 0, SEEK\_SET);

fread(filebuf, BUFFER\_SIZE, 1, fp);

//파일의 개행 개수를 센다

if(strlen(filebuf)==0){

lcount=0;

}

else{

for(int i=0; i<strlen(filebuf); i++){ //파일의 개행 수를 계산해 lcount변수에 저장

if(filebuf[i]=='\n'){

lcount++;

}

}

}

lcount-=1;//파일 끝 개행 처리

//filebuf에 저장된 파일 내용을 개행을 나누어 linebuf에 저장

int N=0;

ptr = strtok(filebuf, "\n");

while(1){

if(N>lcount) break;

strcpy(linebuf[N], ptr);

ptr = strtok(NULL, "\n");

N++;

}

lcount+=1;

if(num > lcount){//존재하지 않는 번호이면

fprintf(stderr, "잘못된 COMMAND\_NUMBER\n");

return;

}

int l=0;

int offset=0;

char deletebuf[BUFFER\_SIZE];

memset(deletebuf, '\0', BUFFER\_SIZE);

int d=0;

while(l<lcount){

if(l==num){

memcpy(deletebuf, linebuf[l], strlen(linebuf[l]));

l++;

continue;

}

else if(l>num){

linebuf[l][0]=(char)l+47;

if(offset!=0){

tmpbuf[offset]='\n';

offset+=1;

}

memcpy(&tmpbuf[offset], linebuf[l], strlen(linebuf[l]));

offset += strlen(linebuf[l]);

l++;

}

else {

if(offset!=0){

tmpbuf[offset]='\n';

offset+=1;

}

memcpy(&tmpbuf[offset], linebuf[l], strlen(linebuf[l]));

offset += strlen(linebuf[l]);

l++;

}

}

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

char logbuf[BUFFER\_SIZE];

memset(logbuf, '\0', BUFFER\_SIZE);

rtime = time(NULL);

char timebuf[BUFFER\_SIZE];

memset(timebuf, '\0', BUFFER\_SIZE);

strcpy(timebuf, ctime(&rtime));

timebuf[strlen(timebuf)-1]=0;

for(int i=0; i<strlen(deletebuf); i++){

if(deletebuf[i]==' '){

char tbuf[BUFFER\_SIZE];

memset(tbuf, '\0', BUFFER\_SIZE);

strcpy(tbuf, deletebuf);

memset(deletebuf, '\0', BUFFER\_SIZE);

strcpy(deletebuf,&tbuf[i+1]);

break;

}

}

sprintf(logbuf, "[%s] remove %s\n", timebuf, deletebuf);

fseek(lfp, 0, SEEK\_END);

fwrite(logbuf, strlen(logbuf), 1, lfp);

if(num==0){

fseek(fp, 0, SEEK\_SET);

memset(tmpbuf, '\0', BUFFER\_SIZE);

fwrite(tmpbuf, BUFFER\_SIZE, 1, fp);

}

else{

fseek(fp, 0, SEEK\_SET);

fwrite(tmpbuf, BUFFER\_SIZE, 1, fp);

printf("%s\n", tmpbuf);//화면에 출력한다

}

}

**<ssu\_crond.c>**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/time.h>

#include <syslog.h>

#include <time.h>

#include <ctype.h>

#define BUFFER\_SIZE 1024

int min[60];

int hour[60];

int day[32];

int month[13];

int week[7];

/\*ssu\_crond.c\*/

FILE \* ffp;//ssu\_crondtab\_file

FILE \* lfp;//ssu\_crondtab\_log

char path[BUFFER\_SIZE];

char filebuf[BUFFER\_SIZE];

int cti(char c);

int ssu\_daemon\_init(void);

void ssu\_crond(void);

void cal\_cycle(char \* line);

void run\_command(char \* passline, char \* line);

int main(void)

{

memset(path, '\0', BUFFER\_SIZE);

getcwd(path,BUFFER\_SIZE);

//printf("%s\n", path);

setbuf(stdout, NULL);

pid\_t pid;

pid = getpid();

char \* ffname ="ssu\_crontab\_file";

if((ffp = fopen(ffname, "r"))==NULL){

fprintf(stderr, "open error for %s\n", ffname);

exit(1);

}

setbuf(ffp,NULL);

//맨 처음 초기의 crontab 파일 내용을 읽어온다

fseek(ffp, 0 , SEEK\_SET);

fread(filebuf, BUFFER\_SIZE, 1, ffp);

//daemon process 생성

if(ssu\_daemon\_init() < 0) {

fprintf(stderr, "ssu\_daemon\_init failed\n");

exit(1);

}

exit(0);

}

int ssu\_daemon\_init(void){

int fd, maxfd;

pid\_t pid;

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

fprintf(stderr, "fork error\n");

exit(1);

}

else if(pid !=0)

exit(0);

pid = getpid();

setsid();

signal(SIGTTIN, SIG\_IGN);

signal(SIGTTOU, SIG\_IGN);

signal(SIGTSTP, SIG\_IGN);

maxfd = getdtablesize();

for(fd = 0; fd < maxfd; fd++)

close(fd);

umask(0);

chdir("/");

fd = open("/dev/null", O\_RDWR);

dup(0);

dup(0);

chdir(path);

while(1){

char \* lfname = "ssu\_crontab\_log";

if(access(lfname,F\_OK)!=0){//로그파일이 존재하지 않으므로 파일 생성

if((lfp=fopen(lfname,"w+"))==NULL){

fprintf(stderr, "open error for %s\n", lfname);

exit(1);

}

}

else{//로그 파일이 이미 존재할 때

if((lfp = fopen(lfname, "a+"))==NULL){

fprintf(stderr, "open error for %s\n", lfname);

exit(1);

}

}

setbuf(lfp,NULL);

ssu\_crond();

sleep(60);

fclose(lfp);

}

return 0;

}

void ssu\_crond(void){

int count=0;

for (int i=0; i<strlen(filebuf); i++){

if(filebuf[i]=='\n') count++;

}

int l=0;

int j=0;

char linebuf[count][BUFFER\_SIZE];

memset(linebuf, '\0', sizeof(linebuf));

for(int i=0; i<strlen(filebuf); i++){

if(filebuf[i]=='\n'){

l++;

j=0;

}

else{

linebuf[l][j]= filebuf[i];

j++;

}

}

char tbuf[BUFFER\_SIZE];

//문자열에서 앞에 숫자부분을 빼주는 코드

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

memset(tbuf, '\0', BUFFER\_SIZE);

strcpy(tbuf, linebuf[i]);

//printf("%s\n", tbuf);

for(int j=0; j<strlen(linebuf[i]); j++){

if(linebuf[i][j]==' '){

memset(linebuf[i], '\0', strlen(linebuf[i]));

strcpy(linebuf[i], &tbuf[j+1]);

break;

}

}

}

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

memset(min, 0, sizeof(min));

memset(hour, 0, sizeof(hour));

memset(day, 0, sizeof(day));

memset(month, 0, sizeof(month));

memset(week, 0, sizeof(week));

cal\_cycle(linebuf[i]);

//printf("%d\n", i);

}

}

//문자를 정수로 바꿔주는 함수

int cti(char c){

return (int)c-48;

}

void cal\_cycle(char \* line){

char token[5][10];

memset(token, '\0', sizeof(token));

char \* p;

char linearr[BUFFER\_SIZE];

memset(linearr, '\0', BUFFER\_SIZE);

char passline[BUFFER\_SIZE];

memset(passline, '\0', BUFFER\_SIZE);

strcpy(passline, line);

strcpy(linearr, line);

p = strtok(linearr, " ");

//주기 토큰 자르기

int w=0;

strcpy(token[w],p);

p=strtok(NULL, " ");

while(w<4){

w++;

strcpy(token[w], p);

p = strtok(NULL, " ");

}

char command[BUFFER\_SIZE];

memset(command, '\0', BUFFER\_SIZE);

while(p!=NULL){

strcpy(&command[strlen(command)], p);

command[strlen(command)]=' ';

//printf("%s\n", command);

p = strtok(NULL, " ");

}

/\*

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

printf("%s\n", token[i]);

}\*/

int num1=1;

int num2=0;

int num3=1;

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

for(int j=0; j<strlen(token[i]); j++){

//숫자로 시작

if(isdigit(token[i][j])){

if(isdigit(token[i][j+1])){//두자리 숫자이면

num1 = cti(token[i][j])\*10 + cti(token[i][j+1]);

j++;

}

else num1 = cti(token[i][j]);

if(token[i][j+1]=='-'){//숫자 뒤에 -가 오면

j+=2;

if(isdigit(token[i][j])){//-뒤에 숫자이면

if(isdigit(token[i][j+1])){//두자리 숫자가 오면

num2 = cti(token[i][j])\*10 + cti(token[i][j+1]);

j++;

}

else num2 = cti(token[i][j]);

}

if(isdigit(token[i][j+1]=='/')){

j+=2;

if(isdigit(token[i][j])){

if(isdigit(token[i][j+1])){

num3 = cti(token[i][j])\*10 + cti(token[i][j+1]);

j++;

}

else num3 = cti(token[i][j]);

}

for(int a=0; num1<=num2; a++){//숫자 - 숫자 / 숫자

if((a+1)%num3==0){

if(i==0){

min[i+num1]=1;

}

if(i==1){

hour[i+num1]=1;

}

if(i==2){

day[i+num1]=1;

}

if(i==3){

month[i+num1]=1;

}

if(i==4){

week[i+num1]=1;

}

}

}

}

else{//숫자-숫자 뒤에 /가 안오면 (범위)

for(int a=num1; a<=num2; a++){

if(i==0){

min[a]=1;

}

if(i==1){

hour[a]=1;

}

if(i==2){

day[a]==1;

}

if(i==3){

month[a]==1;

}

if(i==4){

week[a]==1;

}

}

}

}

else{//-가 안오고 숫자만 온다

if(i==0){//분필드일때

min[num1]=1;

}

if(i==1){//시 필드일때

hour[num1]=1;

}

if(i==2){

day[num1]=1;

}

if(i==3){

month[num1]=1;

}

if(i==4){

week[num1]=1;

}

}

}

else if(token[i][j]=='\*'){//맨 처음에 \*이 온다

if(token[i][j+1]=='/'){//\*뒤 /가 올 때

j+=2;

if(isdigit(token[i][j])){

if(isdigit(token[i][j+1])){//두자리 숫자이면

num1 = cti(token[i][j])\*10 + cti(token[i][j+1]);

j++;

}

else num1 = cti(token[i][j]);

if(i==0){//분필드일때=

for(int k=0; k<=sizeof(min)/sizeof(int); k++){

if((k+1)%num1==0) min[k]=1;

}

}

if(i==1){//시 필드일때

for(int k=0; k<=sizeof(hour)/sizeof(int); k++){

if((k+1)%num1==0) hour[k]=1;

}

}

if(i==2){

for(int k=0; k+1<=sizeof(day)/sizeof(int); k++){

if((k+1)%num1==0) day[k+1]=1;

}

}

if(i==3){

for(int k=0; k+1<=sizeof(month)/sizeof(int); k++){

if((k+1)%num1==0) month[k+1]=1;

}

}

if(i==4){

for(int k=0; k<=sizeof(week)/sizeof(int); k++){

if((k+1)%num1==0) week[k]=1;

}

}

}

}

else{//\*만 올때

if(i==0){//분필드일때

for(int k=0; k<=sizeof(min)/sizeof(int); k++){

min[k]=1;

}

}

if(i==1){//시 필드일때

for(int k=0; k<=sizeof(hour)/sizeof(int); k++){

hour[k]=1;

}

}

if(i==2){

for(int k=0; k+1<=sizeof(day)/sizeof(int); k++){

day[k]=1;

}

}

if(i==3){

for(int k=0; k+1<=sizeof(month)/sizeof(int); k++){

month[k]=1;

}

}

if(i==4){

for(int k=0; k<=sizeof(week)/sizeof(int); k++){

week[k]=1;

}

}

}

}

}

}

run\_command(passline, command);

}

void run\_command(char \* passline, char \* line){

char command[BUFFER\_SIZE];

memset(command, '\0', BUFFER\_SIZE);

struct tm t;

time\_t curtime = time(NULL);

localtime\_r(&curtime, &t);

strcpy(command, line);

//printf("command : %s\n" , command);

if(min[t.tm\_min]&&hour[t.tm\_hour]&&day[t.tm\_mday]&&month[t.tm\_mon+1]&&week[t.tm\_wday]){

//printf("hello\n");

system(command);

char passbuf[BUFFER\_SIZE];

memset(passbuf, '\0', BUFFER\_SIZE);

strcpy(passbuf, passline);

char timebuf[BUFFER\_SIZE];

memset(timebuf, '\0', BUFFER\_SIZE);

strcpy(timebuf, ctime(&curtime));

for(int i=0; i<strlen(timebuf); i++){

if(timebuf[i]=='\n'){

timebuf[i]='\0';

break;

}

}

char logbuf[BUFFER\_SIZE];

memset(logbuf, '\0', BUFFER\_SIZE);

sprintf(logbuf, "[%s] run %s\n", timebuf, passbuf);

fseek(lfp, 0, SEEK\_END);

fwrite(logbuf, strlen(logbuf), 1, lfp);

}

}

**<ssu\_rsync.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <sys/stat.h>

#include <utime.h>

#include <time.h>

#include <sys/time.h>

#include <dirent.h>

#define PATHLEN 1024

#define BUFFER\_SIZE 1024

#define SECOND\_TO\_MICRO 1000000

static int filter(const struct dirent \*dirent){

if(!(strcmp(dirent->d\_name, ".")) || !(strcmp(dirent->d\_name, ".."))) return 0;

else if(dirent->d\_name[0]=='.'){return 0;}

else return 1;

}

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

{

end\_t->tv\_sec -= begin\_t->tv\_sec; // 수행시간이 몇 초인지 계산

//수행시간의 마이크로초보다 수행 시작 시각의 마이크로초가 크면 수행시간을 1초 줄이고, 1000000마이크로초 더해준다.

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

}

end\_t->tv\_usec -= begin\_t->tv\_usec; //수행시간이 몇 마이크로초인지 계산

printf("Runtime: %ld:%06ld(sec:usec)\n", end\_t->tv\_sec, end\_t->tv\_usec);

}

/\*ssu\_rsync.c\*/

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

setbuf(stdout,NULL);

char srcpath[PATHLEN]; //source

char tmppath[PATHLEN];

char dstpath[PATHLEN]; //destination

char logbuf[BUFFER\_SIZE];

char timebuf[PATHLEN];

struct stat statbuf;

struct stat statbuf2;

struct utimbuf ubuf;

time\_t syntime;//

FILE \* lfp; //log file pointer

struct dirent \*\*namelist; //src가 디렉토리일때 디렉토리 안의 파일정보를 저장할 구조체

int check=0;

struct timeval begin\_t, end\_t;

gettimeofday(&begin\_t, NULL);

if(argc>4 || argc<3){//인자 부족시 에러처리

fprintf(stderr, "usage : %s [option] <src> <dst>\n",argv[0]);

exit(1);

}

//사용할 버퍼를 모두 초기화

memset(srcpath, '\0', PATHLEN);

memset(tmppath, '\0', PATHLEN);

memset(dstpath, '\0', PATHLEN);

memset(logbuf, '\0', BUFFER\_SIZE);

memset(timebuf, '\0', PATHLEN);

if(argc==3){//옵션 입력을 하지 않았을 때

strcpy(srcpath, argv[1]);

strcpy(dstpath, argv[2]);

lstat(argv[1], &statbuf);

if(access(argv[1], F\_OK)!=0){//동기화할 파일을 찾을 수 없으면

fprintf(stderr, "usage : %s [option] <src> <dst>\n", argv[0]);

exit(1);

}

if(access(argv[2], F\_OK)!=0){//인자로 입력받은 dst디렉토리를 찾을 수 없으면

fprintf(stderr, "usage : %s [option] <src> <dst>\n", argv[0]);

exit(1);

}

lstat(argv[2], &statbuf2);

if(!S\_ISDIR(statbuf2.st\_mode)){//인자로 입력받은 dst가 디렉토리가 아닐 때

fprintf(stderr, "usage : %s [option] <src> <dst>\n", argv[0]);

exit(1);

}

//\*\*\*동기화할 파일에 접근할수 없으면 usage 출력 후 프로그램 종료

if(!S\_ISDIR(statbuf.st\_mode)){//src가 소스파일이면

getcwd(tmppath,PATHLEN);

sprintf(dstpath, "%s/%s/%s", tmppath, argv[2], argv[1]);

lstat(argv[1], &statbuf);

ubuf.actime = statbuf.st\_atime;

ubuf.modtime = statbuf.st\_mtime;

//동일한 파일이 있을 경우 동기화 안함

if(access(dstpath, F\_OK)==0){

lstat(dstpath, &statbuf2);

if(statbuf.st\_mtime == statbuf2.st\_mtime){

fprintf(stderr, "이미 동기화된 파일이 존재함\n");

exit(0);

}

}

//동일한 파일이 없을 경우(동기화)

//link(argv[1], dstpath);

char tmpbuf[BUFFER\_SIZE];

memset(tmpbuf, '\0', BUFFER\_SIZE);

FILE \* spf = fopen(argv[1], "r");

fread(tmpbuf, BUFFER\_SIZE, 1, spf);

FILE \* dpf = fopen(dstpath, "w+");

fseek(dpf, 0, SEEK\_SET);

fwrite(tmpbuf, BUFFER\_SIZE, 1, dpf);

fclose(spf);

fclose(dpf);

syntime = time(NULL);//동기화 할때의 시간을 저장해놓음

if(utime(dstpath, &ubuf)<0){

fprintf(stderr, "%s: utime error\n", argv[1]);

exit(1);

}

char \* lfname = "ssu\_rsync\_log";

if(access(lfname, F\_OK)==0){//log 파일이 있을 경우

if((lfp = fopen(lfname, "r+"))==NULL){//파일을 "r+"모드로 오픈 (O\_RDWR)

fprintf(stderr, "file open error for %s\n", lfname);

exit(1);

}

fseek(lfp, 0, SEEK\_END);

//fputs('\n',lfp); //파일에 개행 입력

}

else{//log 파일이 없을 경우

if((lfp = fopen(lfname, "w+"))==NULL){//파일을 "w+"모드로 오픈 (O\_RDWR)

fprintf(stderr, "file open error for %s\n", lfname);

exit(1);

}

fseek(lfp, 0, SEEK\_SET);

}

strcpy(timebuf, ctime(&syntime));

for(int i=0; i<strlen(timebuf); i++){

if(timebuf[i]=='\n'){

timebuf[i]='\0';

}

}

//localtime\_r(&syntime, &t);

sprintf(logbuf, "[%s] ssu\_rsync %s %s\n %s %ldbytes\n",

timebuf,

argv[1], argv[2],

argv[1], statbuf.st\_size);

//printf("%s\n", logbuf);

fwrite(logbuf, strlen(logbuf), 1, lfp);

}

else{//\*\*\*\*동기화할 src 가 디렉토리인 경우

int count;

if(access(argv[1], F\_OK)!=0){//src 디렉토리가 없을 때

fprintf(stderr, "usage : %s [option] <src> <dst>\n", argv[0]);

exit(1);

}

char srcdirpath[PATHLEN];

memset(srcdirpath, '\0', PATHLEN);

count = scandir(argv[1], &namelist, \*filter, alphasort);

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

memset(logbuf, '\0', BUFFER\_SIZE);

if(argv[1][0]=='/'){//절대경로 입력시

sprintf(srcdirpath, "%s", argv[1]);

}

else{

getcwd(srcdirpath, PATHLEN);

sprintf(srcdirpath, "%s/%s", srcdirpath, argv[1]);

}

//src 디렉토리 안에있는 파일의 경로를 srcdirpath 배열에 저장

sprintf(srcdirpath, "%s/%s", srcdirpath, namelist[i]->d\_name);

lstat(srcdirpath, &statbuf);

if(S\_ISDIR(statbuf.st\_mode)!=0){

fprintf(stderr, "usage : %s [option] <src> <dst>\n", argv[0]);

exit(1);

}

memset(dstpath, '\0', PATHLEN);

memset(tmppath, '\0', PATHLEN);

getcwd(tmppath,PATHLEN);

//dst 디렉토리 안에있는 파일의 경로를 dstpath배열에 저장

sprintf(dstpath, "%s/%s/%s", tmppath, argv[2], namelist[i]->d\_name);

//printf("%s %s\n", tmppath, dstpath);

lstat(srcdirpath, &statbuf);

//디렉토리 안에 이미 동기화된 파일이 존재하면 동기화를 하지 않음

if(access(dstpath, F\_OK)==0){

lstat(dstpath, &statbuf2);

if(statbuf.st\_mtime == statbuf2.st\_mtime){

fprintf(stderr, "이미 동기화된 파일이 존재함\n");

exit(1);

}

}

//fopen(dstpath)

char tmpbuf[BUFFER\_SIZE];

memset(tmpbuf, '\0', BUFFER\_SIZE);

FILE \* spf = fopen(argv[1], "r");

fread(tmpbuf, BUFFER\_SIZE, 1, spf);

FILE \* dpf = fopen(dstpath, "w+");

fseek(dpf, 0, SEEK\_SET);

fwrite(tmpbuf, BUFFER\_SIZE, 1, dpf);

fclose(spf);

fclose(dpf);

syntime = time(NULL);//동기화 할때의 시간을 저장해놓음

ubuf.actime = statbuf.st\_atime;

ubuf.modtime = statbuf.st\_mtime;

//printf("%s\n", dstpath);

if(utime(dstpath, &ubuf)<0){

fprintf(stderr, "utime error\n");

exit(1);

}

char \* lfname = "ssu\_rsync\_log";

if(access(lfname, F\_OK)==0){//log 파일이 있을 경우

if((lfp = fopen(lfname, "a+"))==NULL){//파일을 "r+"모드로 오픈 (O\_RDWR)

fprintf(stderr, "file open error for %s\n", lfname);

exit(1);

}

fseek(lfp, 0, SEEK\_END);

//fputs('\n',lfp); //파일에 개행 입력

}

else{//log 파일이 없을 경우

if((lfp = fopen(lfname, "w+"))==NULL){//파일을 "w+"모드로 오픈 (O\_RDWR)

fprintf(stderr, "file open error for %s\n", lfname);

exit(1);

}

fseek(lfp, 0, SEEK\_SET);

}

strcpy(timebuf, ctime(&syntime));

for(int i=0; i<strlen(timebuf); i++){

if(timebuf[i]=='\n'){

timebuf[i]='\0';

}

}

memset(logbuf, '\0', BUFFER\_SIZE);

//localtime\_r(&syntime, &t);

if(check==0){

sprintf(logbuf, "[%s] ssu\_rsync %s %s\n %s %ldbytes\n",timebuf, argv[1], argv[2], namelist[i]->d\_name, statbuf.st\_size);

check=1;

}

else{

sprintf(logbuf," %s %ldbytes\n",namelist[i]->d\_name, statbuf.st\_size);

}

char logpath[PATHLEN];

memset(logpath, '\0', PATHLEN);

getcwd(logpath, PATHLEN);

sprintf(logpath, "%s/%s", logpath, lfname);

fseek(lfp, 0, SEEK\_END);

long size=ftell(lfp);

fseek(lfp, size, SEEK\_SET);

fwrite(logbuf, strlen(logbuf), 1, lfp);

fclose(lfp);

}

}

gettimeofday(&end\_t, NULL);

ssu\_runtime(&begin\_t, &end\_t);

}

if(argc==4){//옵션 입력 시

//argv[1]은 옵션 인자

if(argv[1][0]!='-'){

fprintf(stderr, "잘못된 옵션 입력\n");

exit(1);

}

strcpy(srcpath, argv[2]);

strcpy(dstpath, argv[3]);

}

}