컴퓨터학부 20162448 김병준

1. **개요**

리눅스에는 다양한 시스템 명령어가 존재한다. 시스템 명령어는 운영체제의 이용에 있어서 복잡한 명령들을 단순화 하여 한 줄로 처리할 수 있는 기능을 제공한다. 이러한 명령어들 중에는 운영체제의 최적한 운영 혹은 변경사항에 대한 기록을 위해서 특정한 작업에 대해서 주기적으로 실행하도록 할 수 있다. 대표적인 예시로 메모리 Garbage collection같은 작업이 있다. 리눅스에서는 사용자가 이러한 주기적으로 실행시킬 작업을 등록하는 시스템 명령어인 crontab과 이를 실행시킬 데몬 프로세스 crond가 존재한다. 이 외에도 두개의 사이트(혹은 작업영역)간의 파일과 디렉토리의 동기화 기능을 제공해주는 rsync가 존재한다. 원격 사이트 미러링 및 데이터 백업등의 용도로 활용을 한다. 이번 설계 과제는 이러한 시스템 명령어인 crontab과 rsync, 그리고 crontab에 등록한 명령어들을 처리할 프로세스 crond를 구현하고자 한다.

1. **설계**

이번 설계 프로젝트의 구성은 사용자가 주기적으로 실행하는 명령어를 목록(ssu\_crontab\_file)에 저장 및 삭제하는 프로그램(ssu\_crontab), 목록을 읽어들여 실행 주기에 맞게 명령어를 실행하는 프로그램(ssu\_crond) 그리고 인자로 주어진 src파일 혹은 디렉토리를 dst디렉토리에 동기화하는 프로그램(ssu\_rsync)로 구성된다. 모든 프로그램에 공통적으로 들어가는 헤더, 함수 및 매크로는 common.\*에 정의되어있다. 마찬가지로 cron관련 공통 사용 헤더, 함수, 매크로는 cron\_support.\*, 그리고 이외의 각 프로그램별 고유 사용 헤더, 함수 및 매크로는 ssu\_crontab.\*, ssu\_crond.\*, ssu\_rsync.\*에 정의되어 있다.

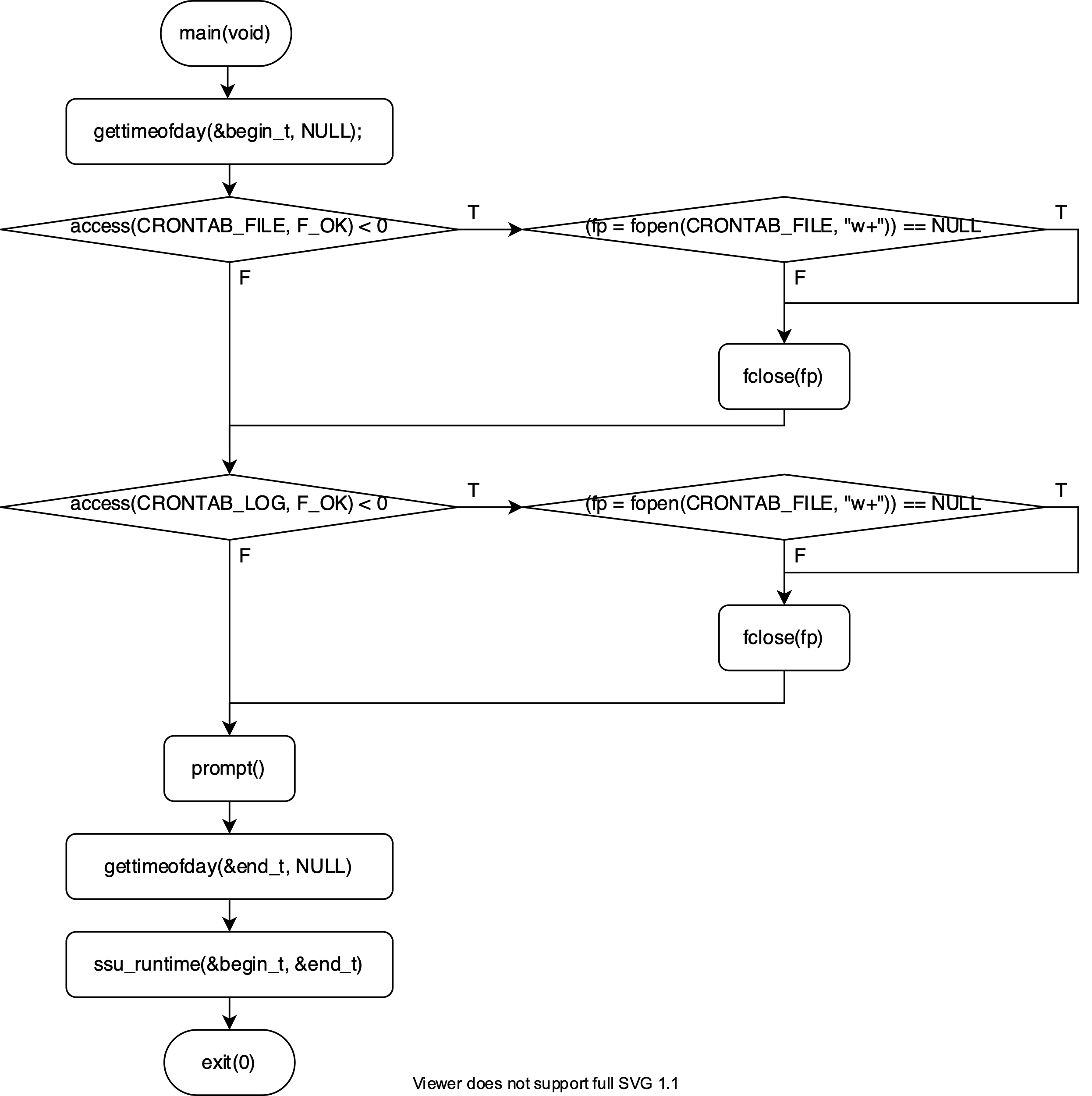
**2-1. Makefile**

Makefile은 기본 컴파일러로 gcc를 사용하며, 각 프로그램에 필요한 오브젝트 파일 생성 시 옵션으로 CFLAGS를 포함한다. 그리고 make명령어를 단독으로 입력했을 때, 모든 프로그램이 컴파일 및 실행 프로그램이 생성되도록 작성하였다. make all을 입력하여도 동일한 결과를 실행한다. 프로그램별 의존성 파일 생성은 make dependency, 재생성은 make new, 생성 파일 삭제는 make clean이다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | .SUFFIXES: .c .o | | .PHONY: dep all new clean | |  | | # Compiler | | CC = gcc | | # Compile option | | # -c: Generate object file | | # -W, -Wall: Print warning about all ambigous grammer | | # -Wextra: Print warning out of -W, -Wall | | # -O2: Optimization | | # -g: Debugging, PLEASE DELETE AFTER PROJECT COMPLETE! | | CFLAGS = -c -W -Wall -Wextra -g $(INC) | |  | | # Execute program file | | CRONTAB = ssu\_crontab | | CROND = ssu\_crond | | RSYNC = ssu\_rsync | | # Source file | | COMMON\_SRCS = common.c | | CRONTAB\_SRCS = ssu\_crontab.c | | CROND\_SRCS = ssu\_crond.c | | RSYNC\_SRCS = ssu\_rsync.c | | CRON\_SUPPORT\_SRCS = cron\_support.c | | SRCS = $(COMMON\_SRCS) $(CRONTAB\_SRCS) $(CROND\_SRCS) $(RSYNC\_SRCS) $(CRON\_SUPPORT\_SRCS) | | # Object file | | COMMON\_OBJS = $(COMMON\_SRCS:.c=.o) | | CRONTAB\_OBJS = $(CRONTAB\_SRCS:.c=.o) | | CROND\_OBJS = $(CROND\_SRCS:.c=.o) | | RSYNC\_OBJS = $(RSYNC\_SRCS:.c=.o) | | CRON\_SUPPORT\_OBJS = $(CRON\_SUPPORT\_SRCS:.c=.o) | | OBJS = $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CROND\_OBJS) $(RSYNC\_OBJS) $(CRON\_SUPPORT\_OBJS) | | # Library file | | LIBS = | | # Include path | | INC = | |  | | # Execute file grneration | | # $@ = TARGET | | # $^ = DEPENDENCY | | # make all: Make all execute file | | all : $(OBJS) | | $(CC) -o $(CRONTAB) $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CRON\_SUPPORT\_OBJS) $(LIBS) | | $(CC) -o $(CROND) $(CROND\_OBJS) $(CRON\_SUPPORT\_OBJS) $(LIBS) -lpthread | | $(CC) -o $(RSYNC) $(COMMON\_OBJS) $(RSYNC\_OBJS) $(LIBS) | | $(CRONTAB) : $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CRON\_SUPPORT\_OBJS) | | $(CC) -o $@ $^ $(LIBS) | | $(CROND) : $(CROND\_OBJS) $(CRON\_SUPPORT\_OBJS) | | $(CC) -o $@ $^ $(LIBS) -lpthread | | $(RSYNC) : $(COMMON\_OBJS) $(RSYNC\_OBJS) | | $(CC) -o $@ $^ $(LIBS) | |  | | # Object file generation | | $(OBJS): | | $(CC) $(CFLAGS) $(SRCS) | |  | | # make dep: Make dependency information file | | dep: | | $(CC) -M $(INC) $(SRCS) > .dependency | |  | | # make new: Re-generation | | new: | | $(MAKE) clean | | $(MAKE) all | |  | | # make clean: Remove all generated file | | clean: | | rm -rf $(OBJS) $(CRONTAB) $(CROND) $(RSYNC) | |

**2-2. SSU\_CRONTAB**

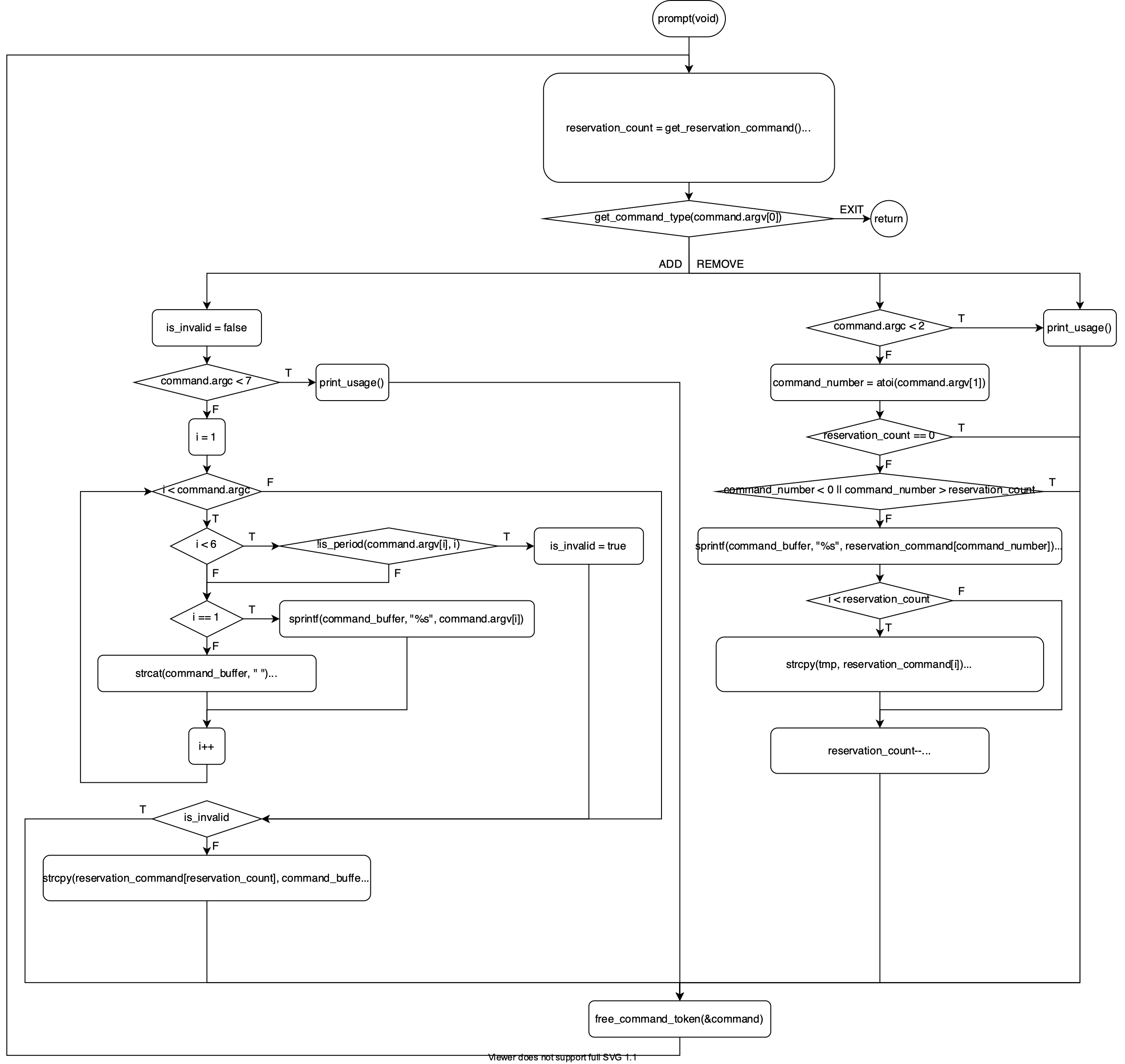
이 프로그램은 사용자가 주기적으로 실행하는 명령어를 목록(ssu\_crontab\_file)을 생성, 저장 및 삭제하는 프로그램이다. 명령어를 저장하기 위해 사용자로부터 명령어를 입력받으며, 이를 위해 프롬프트를 사용자에게 제공한다. 사용자는 프롬프트에서 저장(ADD), 삭제(REMOVE), 종료(EXIT) 명령어를 입력 할 수 있다. 또한 프로그램의 실행 시간을 측정하여 출력한다. ssu\_crontab의 흐름도는 다음과 같다.



[그림 1] ssu\_crontab 프로그램 흐름도

**2-2-1. void prompt(void)**

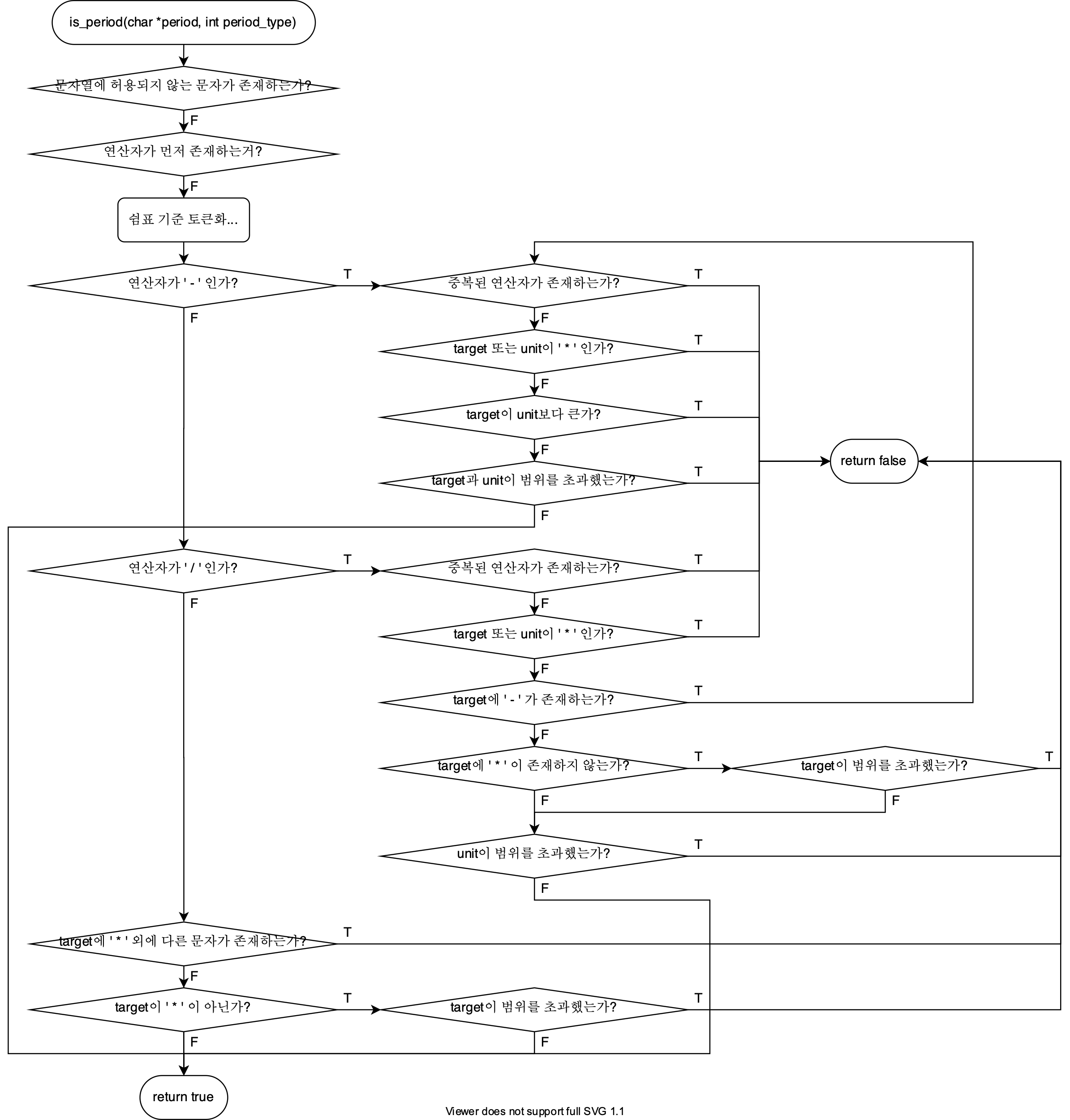
이 함수는 ssu\_crontab에서 제공하는 프롬프트는 사용자가 주기적으로 실행하는 명령어를 저장 혹은 삭제한다. 프롬프트는 시작하면 ssu\_crontab\_file에 저장된 예약 명령어를 읽어들인 후 출력하고, 이후 프롬프트를 출력한다. 그리고 사용자가 입력한 모든 명령행 문자열에 대하여 토큰으로 분리한다. 이후 명령어에 해당하는 작업을 수행한다. 앞의 과정은 프롬프트가 종료되기 전까지 무한적으로 반복한다. 이 함수의 흐름도는 다음과 같다.



[그림 2] prompt 흐름도

**2-2-2. bool is\_period(char \*period, int period\_type)**

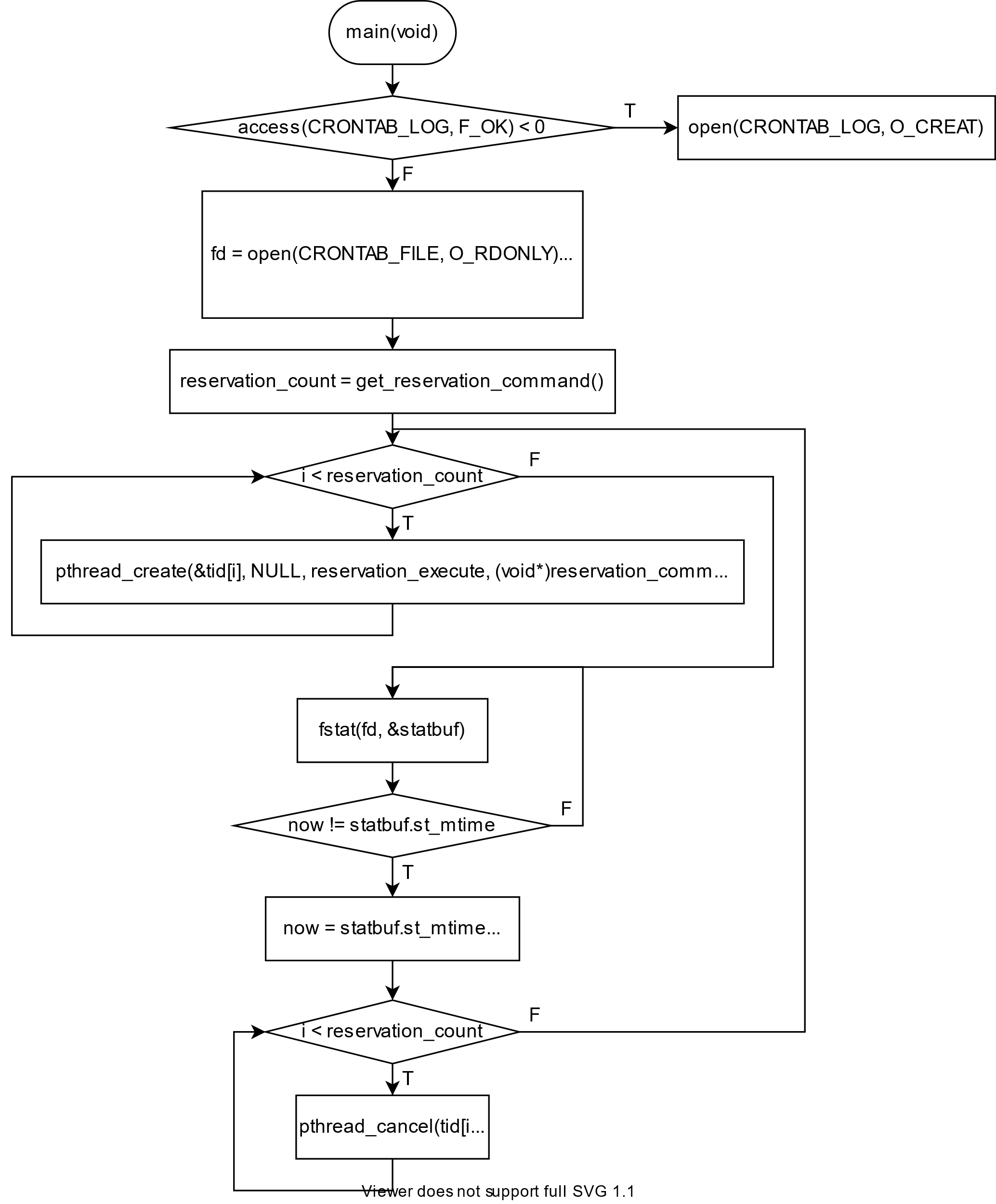
사용자에게 주기적으로 실행할 명령을 입력받으면, 주기가 올바른지 검사하는 함수이다. 정상일 경우 true, 오류를 발견했을 경우 false를 리턴한다. 이 함수를 호출할 때, 각 시간 타입에 해당하는 명령행 문자열 토큰과 시간 타입이 인자로 주어진다. 이후 함수가 호출되면, 기본적으로 문자열 내부에 존재해선 안되는 문자가 존재하는지 판단한다. 이후 쉼표(,) 기준으로 주기를 토큰화 시킨다. 만약 쉼표 앞뒤로 토큰이 비어있다면 오류 처리한다. 쉼표 분리가 완료되었다면, 슬래시(/)와 범위(-)를 기준으로 분리한다. 만약 슬래시와 범위가 동시에 존재한다면 슬래시를 기준으로 분리를 한다. 만약 둘 중에 하나만 존재한다면 존재하는 것을 기준으로 분리한다. 분리가 된 후에는 연산자를 기준으로 앞에 존재하는 문자열은 target, 뒤에 존재하는 문자열은 unit이다. 슬래시를 우선으로 분리하므로, 만약 슬래시가 존재하지 않는다면 operator는 범위가 된다. 이후 unit에 중복된 operator가 존재할 경우, 시작과 끝이 전체(\*)로 끝날 경우, target이 unit보다 클 경우, 시간 타입에 따른 허용 범위를 초과했을 경우를 판별한다. 반대로 만약 슬래시가 존재한다면, 중복된 연산자가 존재할 경우, target이 전체(\*)만 존재하지 않을 경우, unit이 전체(\*)일 경우를 판별한다. 또한 target이 범위일 경우, 범위(-)를 기준으로 다시 한번 분리를 한다. 이후 과정은 앞서 operator가 범위일 경우와 동일하다. 마지막으로 operator가 존재하지 않을 경우 target이 허용 범위를 초과했을 경우를 판별한다. 이 함수의 흐름도는 다음과 같다.



[그림 3] is\_period 흐름도

**2-3. SSU\_CROND**

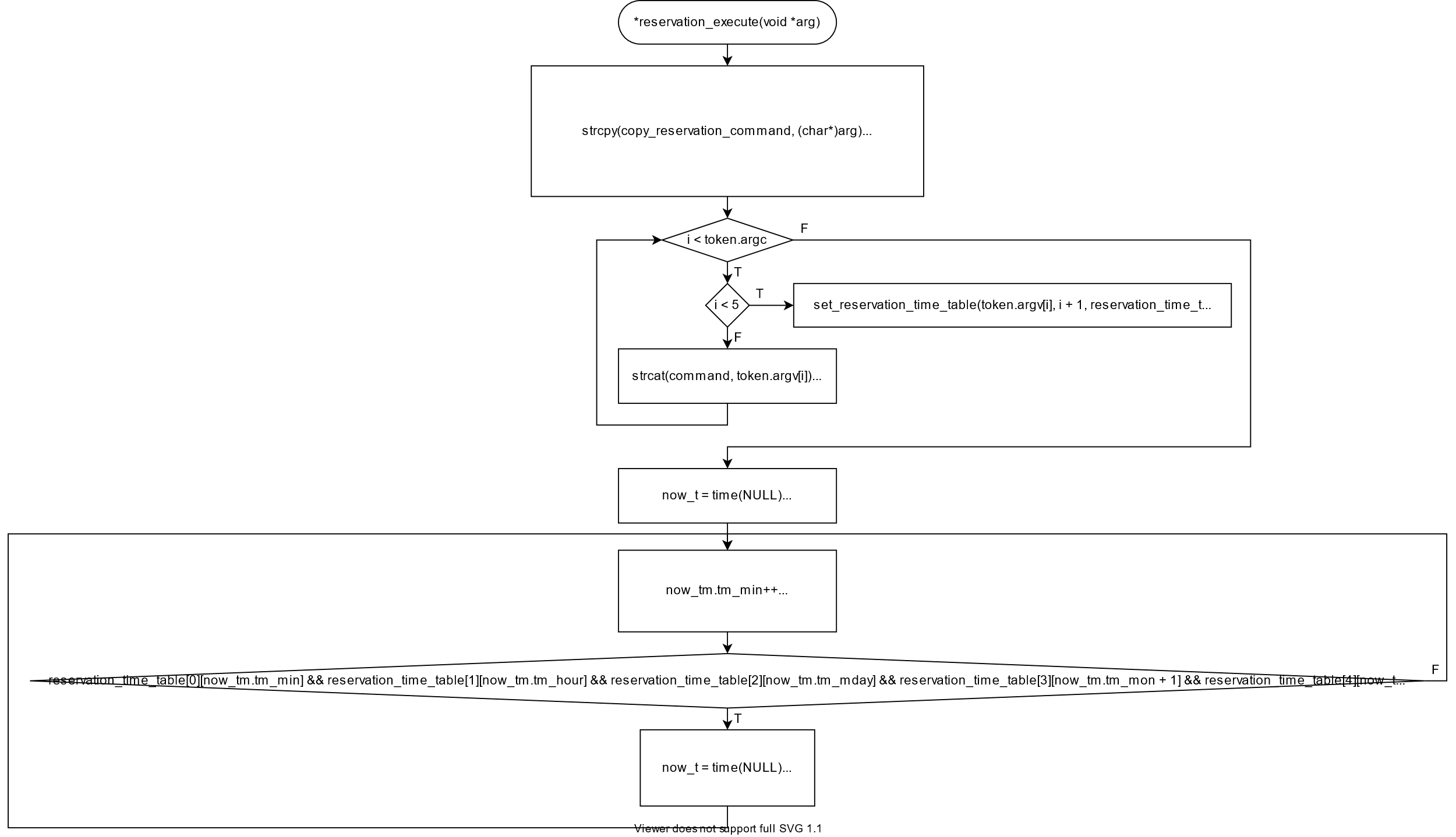
이 프로그램은 사용자가 주기적으로 실행하는 명령어를 저장한 목록(ssu\_crontab\_file)을 읽어들여 스레드로 실행한다. 프로그램이 실행되면, 로그파일을 확인 및 생성하고, ssu\_crontab\_file을 읽어와 명령어마다 스레드를 생성하여 해당하는 주기에 맞추어 실행한다. 각 명령어별 스레드 생성 이후 프로세스에서는 ssu\_crontab\_file의 상태정보를 반복적으로 읽어들여 정보가 수정되었을 경우, 스레드를 모두 중단시키고 변경된 목록에 맞추어 스레드를 재생성한다. 이 과정을 프로세스가 종료되기 전까지 반복한다. 이 프로그램의 흐름도는 다음과 같다.



[그림 4] ssu\_crond 프로그램 흐름도

**2-3-1. void \*reservation\_execute(void \*arg)**

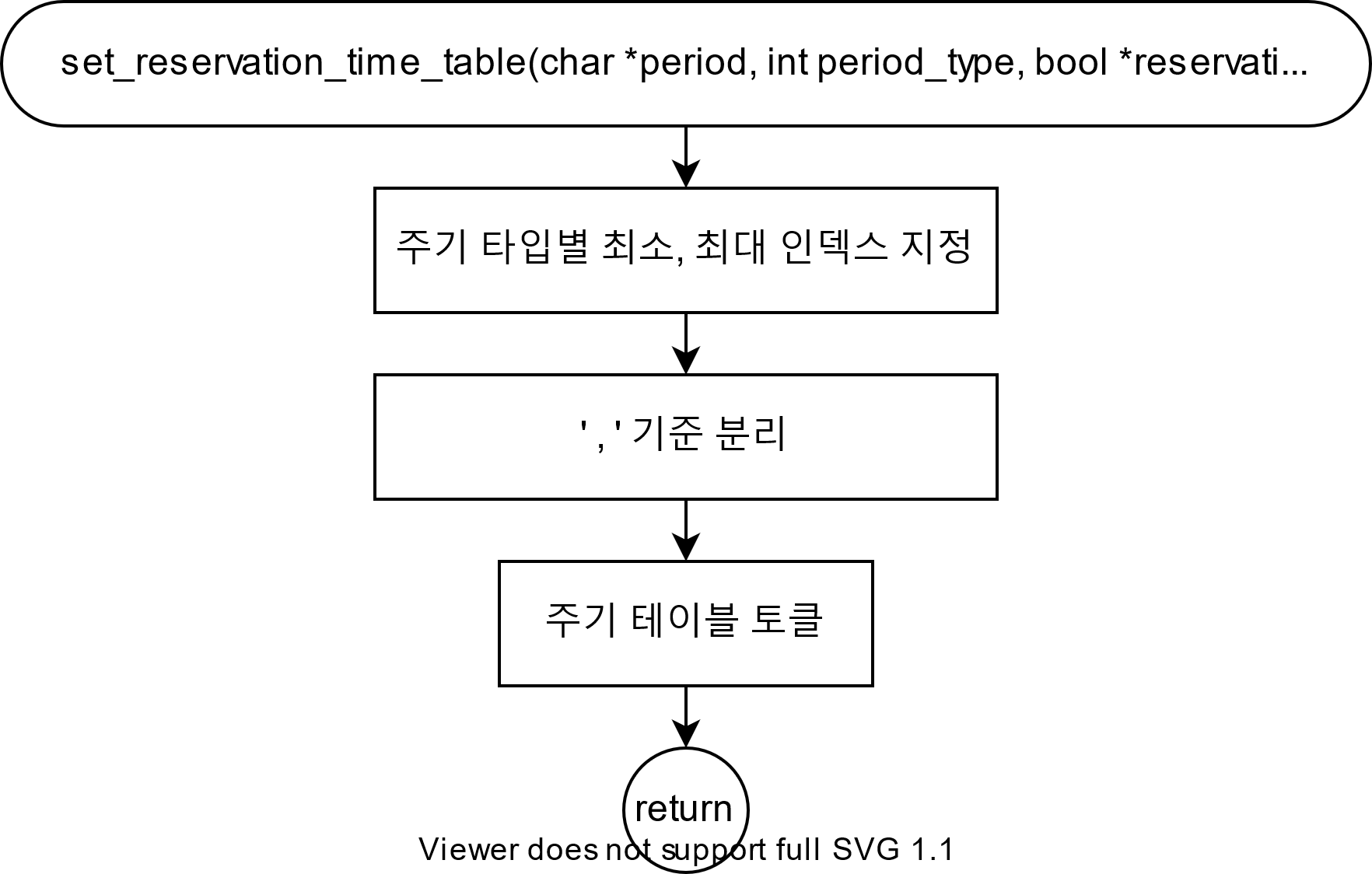
명령어를 주기적으로 실행하기 위해 스레드를 생성할 때, 인자로 ssu\_crontab\_file에서 읽어들인 행을 인자로 넘겨준다. 이 함수에서는 받은 인자를 통해 토큰화하고, 주기 테이블을 생성한다. 그리고 주기 테이블에서 가장 가까운 주기를 now\_tm변수의 tm\_min을 증가시키고 AND연산으로 참이 된 주기에 맞추어서 현재 시간과 차이를 비교한다. 이후 그 차이만큼 sleep()을 통해 대기한다. 대기 시간이 끝나면 지정된 시간이 되었을 때, system()를 이용하여 명령어를 실행한다. 이 함수의 흐름도는 다음과 같다.



[그림 5] reservation\_execute 흐름도

**2-3-2. void set\_reservation\_time\_table(char \*period, int period\_type, bool \*reservation\_table)**

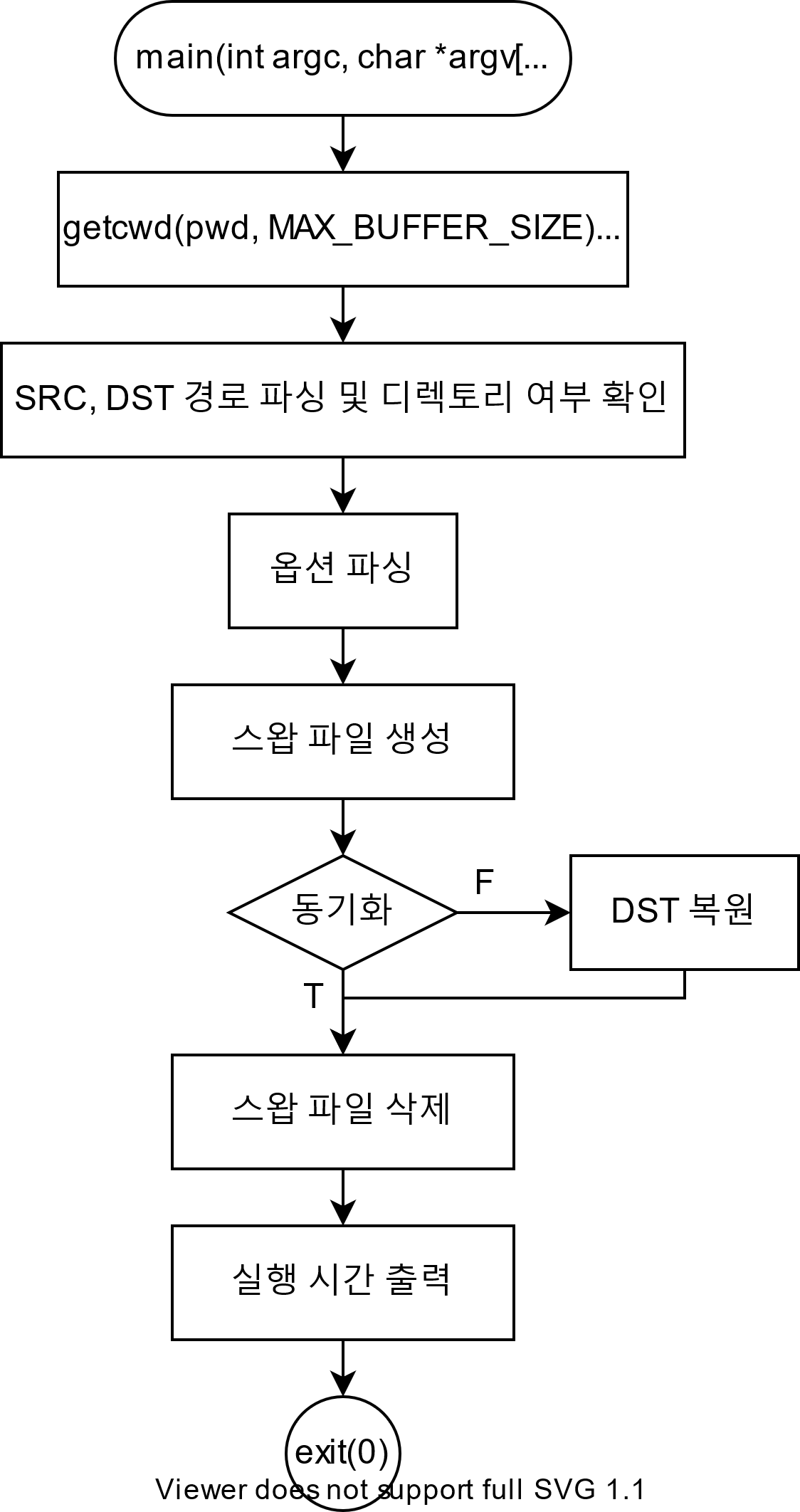
주기 테이블은 각 주기 타입별로 크기가 다르다. MINUTE은 60, HOUR은 24, DAY는 32, MONTH는 13, DAY\_OF\_WEEK는 8의 크기를 가진 bool타입의 배열이다. 이 함수에서는 해당하는 주기의 배열 인덱스를 토글한다. 주기를 분리하는 방법은 2-2-2. Is\_period를 참고한다. 이 함수의 흐름도는 다음과 같다.



[그림 6] set\_reservation\_time\_table 흐름도

**2-4. SSU\_RSYNC**

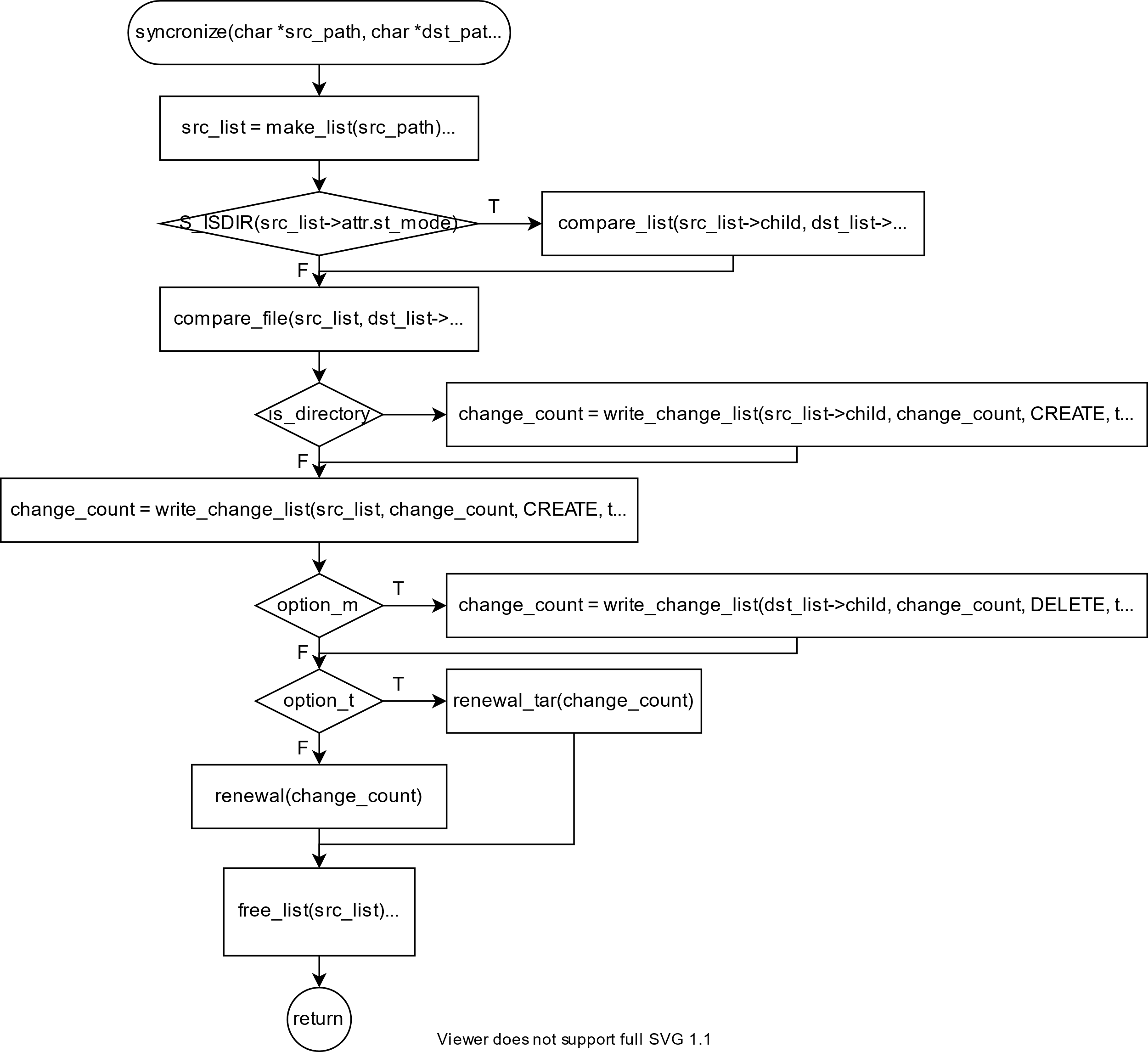
이 프로그램은 사용자가 지정한 SRC(파일 또는 디렉토리)를 DST(디렉토리)에 동기화한다. 동기화를 실행하기 전, SRC와 DST의 접근 가능 여부를 확인한다. 접근이 가능하다면 절대경로로 변환한다. 그리고 상태 정보를 획득하여 디렉토리 여부를 판단한다. 이후 옵션을 파싱하여 옵션 확인 변수를 토글한다. 이후 DST의 정보를 가진 스왑 파일을 생성한다. 이후 동기화를 진행하며, 동기화가 성공적으로 이루어졌다면 스왑파일을 제거한다. 만약 중간에 SIGINT시그널이 수신되었다면, 동기화되고 있던 DST를 삭제하고, 스왑 파일을 복원한다. 이 프로그램의 흐름도는 다음과 같다.



[그림 7] ssu\_rsync 프로그램 흐름도

**2-4-1. void synchronize(char \*src\_path, char \*dst\_path)**

이 함수는 동기화기능의 메인 함수이다. 동기화는 SRC와 DST의 파일 비교로 이루어진다. 비교는 SRC와 DST의 하위 파일들을 트리로 만들어서 비교한다. 기본적으로 동일한 파일의 기준은 파일 이름과 파일 크기, 수정시간이 같은 경우이다. 같은 이름의 파일이 존재할 때, 후술된 조건 중 하나라도 다르다면 수정되었음으로 판단한다. 만약 같은 이름의 파일이 존재하지 않는다면, 생성되었음으로 판단한다. 수정 혹은 생성되었을 경우 SRC의 트리에서 해당하는 파일의 상태를 MODIFY 혹은 CREATE로 지정한다. 또한SRC에는 존재하지 않으나 DST에서 존재하는 파일들에 대해서는 DST의 트리에서 해당하는 파일의 상태를 DELETE로 지정한다. 이후 변경 사항 목록(change\_list)에 기록하며, 변경 사항 목록에 대하여 동기화를 진행한다. 이 함수의 흐름도는 다음과 같다.



[그림 8] synchronize 흐름도

**3. 구현**

**3-1. 프로그램 공통**

**3-1-1.** void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

실행 시간 출력

**3-2. SSU\_CRONTAB, SSU\_CROND 공통**

**3-2-1.** void make\_command\_token(CommandToken \*command, char \*command\_buffer)

입력한 명령행을 토큰 구조체로 변환

**3-2-2.** void free\_command\_token(CommandToken \*command)

토큰 구조체 메모리 해제

**3-2-3.** int get\_reservation\_command(void)

예약 명령 목록 가져오기, 반환: 등록된 명령어 개수

**3-2-4.** void write\_log(int command\_type, char \*command)

로그 파일에 이력 기록

**3-3. SSU\_CRONTAB**

**3-3-1.** void prompt(void)

프롬프트 메인 함수

**3-3-2.** int get\_command\_type(char \*command)

명령 타입 확인 및 타입 번호 반환, 반환:명령 타입 번호

**3-3-3.** char \*rtrim(char \*\_str)

문자열 오른쪽 공백 제거, 반환:공백 제거된 문자열

**3-3-4.** char \*ltrim(char \*\_str)

문자열 왼쪽 공백 제거, 반환:공백 제거된 문자열

**3-3-5.** void to\_lower\_case(char \*str)

문자열 소문자 변환

**3-3-6.** void print\_reservation\_list(int count)

예약 명령 목록 출력

**3-3-7.** bool is\_period(char \*period, int period\_type)

주기 인자 검사, 문제없음:true, 주기오류:false

**3-3-8.** bool is\_period\_character(char c)

주기 문자 검사, 문제없음:true, 주기오류:false

**3-3-9.** void write\_reservation\_file(int count)

예약 명령 목록 파일 기록

**3-3-10.** void print\_usage(void)

사용법 출력

**3-4. SSU\_CROND**

**3-4-1.** void \*reservation\_execute(void \*arg)

예약 명령 실행 스레드

**3-4-2.** void set\_reservation\_time\_table(char \*period, int period\_type, bool \*reservation\_table)

예약 시간 테이블 설정

**3-5. SSU\_RSYNC**

**3-5-1.** void copy\_argument(int argc, char \*argv[])

명령행 인자 백업

**3-5-2.** void syncronize(char \*src\_path, char \*dst\_path)

동기화 메인 함수

**3-5-3.** file\_node \*make\_node(void)

노드 생성, 반환:새로운 노드

**3-5-4.** file\_node \*make\_list(char \*path)

디렉토리 내부 파일 목록 트리화, 반환:트리 루트 노드

**3-5-5.** int count\_size(file\_node \*head)

디렉토리 크기 반환, 반환:디렉토리 크기

**3-5-6.** void compare\_list(file\_node \*src\_list, file\_node \*dst\_list)

파일 목록 트리 비교

**3-5-7.** bool compare\_file(file\_node \*src\_file, file\_node \*dst\_file)

파일 정보 비교, 비교성공:true, 비교실패:false

**3-5-8.** int write\_change\_list(file\_node \*head, int idx, int status)

변경 사항 목록 작성, 반환:변경 사항 개수

**3-5-9.** void write\_log(int count, unsigned long long totalsize)

로그 파일 기록

**3-5-10.** void free\_list(file\_node \*head)

파일 목록 구조체 메모리 할당 해제

**3-5-11.** void renewal(int count)

변경 사항 목록 적용

**3-5-12.** void renewal\_tar(int count)

변경 사항 목록 tar명령어 적용

**3-5-13.** void recovery(int signo)

DST 디렉토리 복원

**3-5-14.** void remove\_directory(const char \*path)

디렉토리 삭제

**3-5-15.** char \*get\_file\_name(char \*path)

파일명 추출, 반환:파일명 시작 메모리주소

**3-5-16.** void print\_usage(char \*execute\_file)

사용법 출력

**4. 테스트 및 결과**

**4-1. SSU\_CRONTAB**

**4-1-1. 프롬프트 출력**

**전화이(가) 표시된 사진

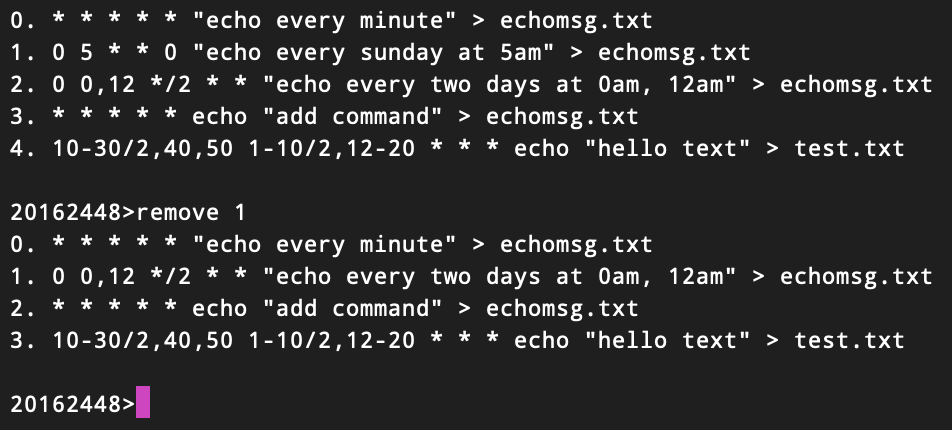
자동 생성된 설명**

**4-1-2. 추가(ADD)**

**텍스트, 표지판이(가) 표시된 사진

자동 생성된 설명**

**4-1-3. 삭제(REMOVE)**

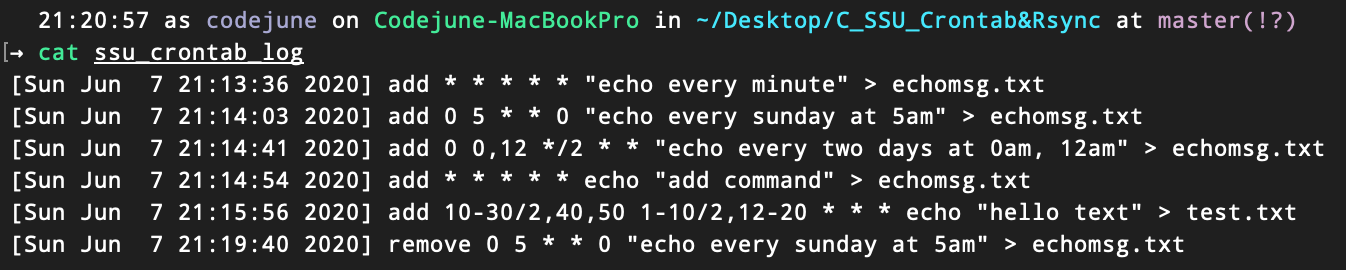
****

**4-1-4. ssu\_crontab\_file 생성 및 기록**

**모니터, 텔레비전, 화면, 앉아있는이(가) 표시된 사진

자동 생성된 설명**

**4-1-5. ssu\_crontab\_log 생성 및 기록**

****

**4-1-6. 종료(EXIT)**

**모니터, 텔레비전, 화면, 앉아있는이(가) 표시된 사진

자동 생성된 설명**

**4-2. SSU\_CROND**

**4-2-1. 예약 명령 실행 및 로그 기록**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**4-3. SSU\_RSYNC**

**4-3-1. SRC가 파일인 경우**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**4-3-2. SRC가 디렉토리인 경우**

**스크린샷이(가) 표시된 사진

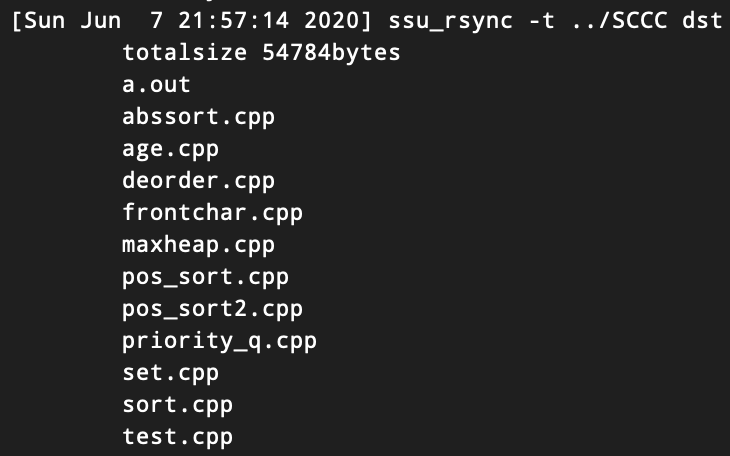
자동 생성된 설명**

**4-3-3. R옵션**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**4-3-4. T옵션(상:로그, 하:동기화결과)**

****

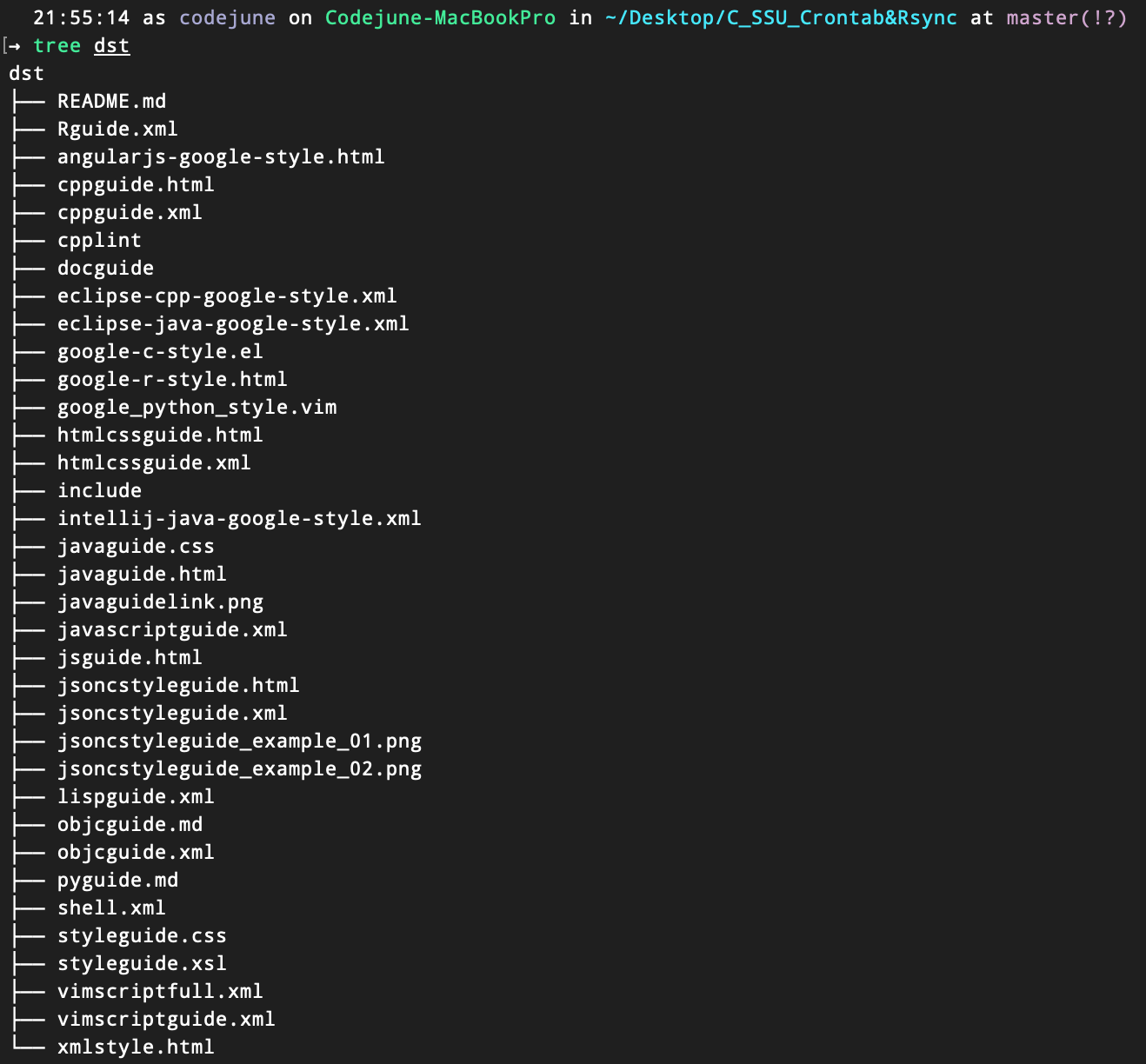
**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**4-3-5. M옵션(상:로그, 하:동기화결과)**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

****

**4-3-6. 중복 옵션 실행**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**5. 소스코드와 주석**

**5-1. Makefile**

|  |
| --- |
|  |
| .SUFFIXES: .c .o |
| .PHONY: dep all new clean | |
|  | |
| # Compiler | |
| CC = gcc | |
| # Compile option | |
| # -c: Generate object file | |
| # -W, -Wall: Print warning about all ambigous grammer | |
| # -Wextra: Print warning out of -W, -Wall | |
| # -O2: Optimization | |
| # -g: Debugging, PLEASE DELETE AFTER PROJECT COMPLETE! | |
| CFLAGS = -c -W -Wall -Wextra -g $(INC) | |
|  | |
| # Execute program file | |
| CRONTAB = ssu\_crontab | |
| CROND = ssu\_crond | |
| RSYNC = ssu\_rsync | |
| # Source file | |
| COMMON\_SRCS = common.c | |
| CRONTAB\_SRCS = ssu\_crontab.c | |
| CROND\_SRCS = ssu\_crond.c | |
| RSYNC\_SRCS = ssu\_rsync.c | |
| CRON\_SUPPORT\_SRCS = cron\_support.c | |
| SRCS = $(COMMON\_SRCS) $(CRONTAB\_SRCS) $(CROND\_SRCS) $(RSYNC\_SRCS) $(CRON\_SUPPORT\_SRCS) | |
| # Object file | |
| COMMON\_OBJS = $(COMMON\_SRCS:.c=.o) | |
| CRONTAB\_OBJS = $(CRONTAB\_SRCS:.c=.o) | |
| CROND\_OBJS = $(CROND\_SRCS:.c=.o) | |
| RSYNC\_OBJS = $(RSYNC\_SRCS:.c=.o) | |
| CRON\_SUPPORT\_OBJS = $(CRON\_SUPPORT\_SRCS:.c=.o) | |
| OBJS = $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CROND\_OBJS) $(RSYNC\_OBJS) $(CRON\_SUPPORT\_OBJS) | |
| # Library file | |
| LIBS = | |
| # Include path | |
| INC = | |
|  | |
| # Execute file grneration | |
| # $@ = TARGET | |
| # $^ = DEPENDENCY | |
| # make all: Make all execute file | |
| all : $(OBJS) | |
| $(CC) -o $(CRONTAB) $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CRON\_SUPPORT\_OBJS) $(LIBS) | |
| $(CC) -o $(CROND) $(CROND\_OBJS) $(CRON\_SUPPORT\_OBJS) $(LIBS) -lpthread | |
| $(CC) -o $(RSYNC) $(COMMON\_OBJS) $(RSYNC\_OBJS) $(LIBS) | |
| $(CRONTAB) : $(COMMON\_OBJS) $(CRONTAB\_OBJS) $(CRON\_SUPPORT\_OBJS) | |
| $(CC) -o $@ $^ $(LIBS) | |
| $(CROND) : $(CROND\_OBJS) $(CRON\_SUPPORT\_OBJS) | |
| $(CC) -o $@ $^ $(LIBS) -lpthread | |
| $(RSYNC) : $(COMMON\_OBJS) $(RSYNC\_OBJS) | |
| $(CC) -o $@ $^ $(LIBS) | |
|  | |
| # Object file generation | |
| $(OBJS): | |
| $(CC) $(CFLAGS) $(SRCS) | |
|  | |
| # make dep: Make dependency information file | |
| dep: | |
| $(CC) -M $(INC) $(SRCS) > .dependency | |
|  | |
| # make new: Re-generation | |
| new: | |
| $(MAKE) clean | |
| $(MAKE) all | |
|  | |
| # make clean: Remove all generated file | |
| clean: | |
| rm -rf $(OBJS) $(CRONTAB) $(CROND) $(RSYNC) | |
|  |  | |

**5-2. common.h**

|  |
| --- |
| /\*\* |
| \* @file common.h |
| \* @brief 프로젝트에서 공통적으로 사용되는 라이브러리, 매크로 정의 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #ifndef COMMON\_H |
| #define COMMON\_H |
|  |
| //#define DEBUG |
|  |
| #include <stdio.h> |
| #include <stdlib.h> |
| #include <unistd.h> |
| #include <stdbool.h> |
| #include <string.h> |
| #include <time.h> |
| #include <sys/time.h> |
|  |
| // 버퍼 크기 |
| #define BUFFER\_SIZE 256 |
| #define MAX\_BUFFER\_SIZE 1024 |
|  |
| // 시간 크기 |
| #define SECOND\_TO\_MICRO 100000 |
|  |
| void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t); // 실행시간 출력 |
|  |
| #endif // COMMON\_H |

**5-3. common.c**

|  |
| --- |
| /\*\* |
| \* @file common.c |
| \* @brief 프로젝트에서 공통적으로 사용되는 함수 정의 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #include "common.h" |
|  |
| /\*\* |
| \* @brief 실행 시간 출력 |
| \* @param begin\_t 시작 시간 |
| \* @param end\_t 종료 시간 |
| \*/ |
| void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t) // 실행 시간 출력 |
| { |
| end\_t->tv\_sec -= begin\_t->tv\_sec; |
|  |
| 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; |
| printf("Runtime: %ld:%ld(sec:usec)\n", end\_t->tv\_sec, end\_t->tv\_usec); |
| } |

**5-4. cron\_support.h**

|  |
| --- |
| /\*\* |
| \* @file cron\_support.h |
| \* @brief ssu\_crontab과 ssu\_crond에서 공통적으로 사용되는 라이브러리, 매크로 정의 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #ifndef CRON\_SUPPORT\_H |
| #define CRON\_SUPPORT\_H |
|  |
| #include <pthread.h> |
| #include "common.h" |
|  |
| /\*\* |
| \* @brief 명령어 타입 번호 |
| \*/ |
| #define ADD 1 |
| #define REMOVE 2 |
| #define RUN 3 |
| #define EXIT 4 |
| #define UNKNOWN 5 |
|  |
| /\*\* |
| \* @brief 주기 타입 번호 |
| \*/ |
| #define MINUTE 1 |
| #define HOUR 2 |
| #define DAY 3 |
| #define MONTH 4 |
| #define DAY\_OF\_WEEK 5 |
|  |
| /\*\* |
| \* @brief 파일 이름 |
| \*/ |
| #define CRONTAB\_FILE "ssu\_crontab\_file" |
| #define CRONTAB\_LOG "ssu\_crontab\_log" |
|  |
| /\*\* |
| \* @brief 프롬프트 명령행 토큰 구조체 |
| \* @param argc 인자 개수 |
| \* @param argv 인자 토큰 |
| \*/ |
| typedef struct CommandTokenStruct |
| { |
| int argc; // 인자 개수 |
| char \*\*argv; // 인자 토큰 |
| } CommandToken; |
|  |
| void make\_command\_token(CommandToken \*command, char \*command\_buffer); // 입력한 명령행을 토큰 구조체로 변환 |
| void free\_command\_token(CommandToken \*command); // 토큰 구조체 메모리 해제 |
| int get\_reservation\_command(void); // 예약 명령 목록 가져오기 |
| void write\_log(int command\_type, char \*command); // 로그 파일에 이력 기록 |
|  |
| #endif // CRON\_SUPPORT\_H |

**5-5. cron\_support.c**

|  |
| --- |
|  |
| /\*\* |
| \* @file cron\_support.c | |
| \* @brief ssu\_crontab, ssu\_crond에서 공통적으로 사용되는 함수 | |
| \* @author 김병준 (kbj9704@gmail.com) | |
| \*/ | |
| #include "cron\_support.h" | |
|  | |
| FILE \*fp; | |
| pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER; // 뮤텍스 객체 선언 | |
| extern char reservation\_command[BUFFER\_SIZE][MAX\_BUFFER\_SIZE]; // 예약 명령 목록 | |
|  | |
| /\*\* | |
| \* @brief 입력한 명령행을 토큰 구조체로 변환 | |
| \* @param command 명령행 토큰 구조체 | |
| \* @param command\_buffer 명령 문자열 | |
| \*/ | |
| void make\_command\_token(CommandToken \*command, char \*command\_buffer) // 입력한 명령행을 토큰 구조체로 변환 | |
| { | |
| char \*tmp; | |
| char \*last; | |
| #ifdef DEBUG | |
| printf("make\_command\_token(): command\_buffer = %s\n", command\_buffer); | |
| #endif | |
| command->argv = (char\*\*)calloc(BUFFER\_SIZE, sizeof(char\*)); | |
| command->argc = 0; | |
|  | |
| if ((tmp = strtok\_r(command\_buffer, " ", &last)) == NULL) | |
| return; | |
| #ifdef DEBUG | |
| printf("make\_command\_token(): command->argv[%d] = %s\n", command->argc, tmp); | |
| #endif | |
|  | |
| command->argv[command->argc] = (char \*)calloc(BUFFER\_SIZE, sizeof(char)); | |
| strcpy(command->argv[command->argc++], tmp); | |
|  | |
| while ((tmp = strtok\_r(NULL, " ", &last)) != NULL) { | |
| #ifdef DEBUG | |
| printf("make\_command\_token(): command->argv[%d] = %s\n", command->argc, tmp); | |
| #endif | |
| command->argv[command->argc] = (char \*)calloc(BUFFER\_SIZE, sizeof(char)); | |
| strcpy(command->argv[command->argc++], tmp); | |
| } | |
| } | |
|  | |
| /\*\* | |
| \* @brief 명령행 구조체 초기화 | |
| \* @param command 명령행 구조체 | |
| \*/ | |
| void free\_command\_token(CommandToken \*command) // 명령행 구조체 초기화 | |
| { | |
| int i; | |
|  | |
| for (i = 0; i < command->argc; i++) | |
| free(command->argv[i]); | |
| free(command->argv); | |
| } | |
|  | |
| /\*\* | |
| \* @brief 예약 명령 목록 가져오기 | |
| \*/ | |
| int get\_reservation\_command(void) // 예약 명령 목록 가져오 | |
| { | |
| FILE \*fp; | |
|  | |
| int count = 0; | |
|  | |
| if ((fp = fopen(CRONTAB\_FILE, "r+")) == NULL) { | |
| fprintf(stderr, "get\_reservation\_command: fopen error for %s\n", CRONTAB\_FILE); | |
| return count; | |
| } | |
|  | |
| while(fscanf(fp, "%[^\n]\n", reservation\_command[count]) > 0) | |
| count++; | |
| #ifdef DEBUG | |
| printf("get\_reservation\_command: reservation\_count = %d\n", count); | |
| #endif | |
|  | |
| fclose(fp); | |
| return count; | |
| } | |
|  | |
| /\*\* | |
| \* @brief 로그 파일에 이력 기록 | |
| \* @param command\_type 명령 타입 번호 | |
| \* @param command 명령 문자열 | |
| \*/ | |
| void write\_log(int command\_type, char \*command) // 로그 파일에 이력 기록 | |
| { | |
| time\_t now\_t; | |
| struct tm \*now\_tm; | |
|  | |
| pthread\_mutex\_lock(&mutex); | |
|  | |
| if ((fp = fopen(CRONTAB\_LOG, "r+")) == NULL) | |
| fp = fopen(CRONTAB\_LOG, "w"); | |
|  | |
| fseek(fp, 0, SEEK\_END); | |
|  | |
| time(&now\_t); | |
| now\_tm = localtime(&now\_t); | |
|  | |
| switch (command\_type) { | |
| case ADD: | |
| //sprintf(temp, "echo \"[%.24s] %s %s\" >> %s", asctime(now\_tm), "add", command, CRONTAB\_LOG); | |
| fprintf(fp, "[%.24s] %s %s\n", asctime(now\_tm), "add", command); | |
| break; | |
| case REMOVE: | |
| //sprintf(temp, "echo \"[%.24s] %s %s\" >> %s", asctime(now\_tm), "remove", command, CRONTAB\_LOG); | |
| fprintf(fp, "[%.24s] %s %s\n", asctime(now\_tm), "remove", command); | |
| break; | |
| case RUN: | |
| //sprintf(temp, "echo \"[%.24s] %s %s\" >> %s", asctime(now\_tm), "run", command, CRONTAB\_LOG); | |
| fprintf(fp, "[%.24s] %s %s\n", asctime(now\_tm), "run", command); | |
| break; | |
| } | |
| fclose(fp); | |
| pthread\_mutex\_unlock(&mutex); | |
| } | |
|  |  | |

**5-6. ssu\_crontab.h**

|  |
| --- |
| /\*\* |
| \* @file ssu\_crontab.h |
| \* @brief ssu\_crontab.c에서 사용되는 라이브러리, 매크로, 프로토타입 선언 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #ifndef SSU\_CRONTAB\_H // Define Guard |
| #define SSU\_CRONTAB\_H |
|  |
| #include <string.h> |
| #include <ctype.h> |
| #include <time.h> |
| #include "common.h" |
| #include "cron\_support.h" |
|  |
| void prompt(void); // 프롬프트 메인 |
| int get\_command\_type(char \*command); // 명령 타입 확인 및 번호 변환 |
| char \*rtrim(char \*\_str); // 문자열 오른쪽 공백 제거 |
| char \*ltrim(char \*\_str); // 문자열 왼쪽 공백 제거 |
| void to\_lower\_case(char \*str); // 문자열 소문자 변환 |
| void print\_reservation\_list(int count); // 예약 명령 목록 출력 |
| bool is\_period(char \*period, int period\_type); // 주기 인자 검사 |
| bool is\_period\_character(char c); // 주기 문자 검사 |
| void write\_reservation\_file(int count); // 예약 명령 목록 파일 기록 |
| void print\_usage(void); // 사용법 출력 |
|  |
| #endif // SSU\_CRONTAB\_H |

**5-7. ssu\_crontab.c**

|  |
| --- |
| /\* |
| \* @file ssu\_crontab.c |
| \* @brief ssu\_crontab 프로그램 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #include "ssu\_crontab.h" |
|  |
| char reservation\_command[BUFFER\_SIZE][MAX\_BUFFER\_SIZE]; |
|  |
| /\*\* |
| \* @brief ssu\_crontab 메인 함수 |
| \*/ |
| int main(void) |
| { |
| FILE \*fp; |
| struct timeval begin\_t, end\_t; |
|  |
| gettimeofday(&begin\_t, NULL); |
| if (access(CRONTAB\_FILE, F\_OK) < 0) { // 예약 명령 목록 파일 확인 |
| if ((fp = fopen(CRONTAB\_FILE, "w+")) == NULL) // 존재하지 않을 경우 생성 |
| fprintf(stderr, "main: fopen error for %s\n", CRONTAB\_FILE); |
| fclose(fp); |
| } |
|  |
| if (access(CRONTAB\_LOG, F\_OK) < 0) { // 로그 파일 확인 |
| if ((fp = fopen(CRONTAB\_LOG, "w+")) == NULL) // 존재하지 않을 경우 생성 |
| fprintf(stderr, "main: fopen error for %s\n", CRONTAB\_LOG); |
| fclose(fp); |
| } |
|  |
| prompt(); |
| gettimeofday(&end\_t, NULL); |
| ssu\_runtime(&begin\_t, &end\_t); |
| exit(0); // 정상 종료 |
| } |
|  |
| /\*\* |
| \* @brief 프롬프트 메인 |
| \*/ |
| void prompt(void) // 프롬프트 메인 |
| { |
| // PROMPT |
| char command\_buffer[MAX\_BUFFER\_SIZE]; // 명령행 버퍼 |
| CommandToken command; // 명령행 토큰 구조체 |
|  |
| // COMMON |
| int reservation\_count; |
| bool is\_invalid; |
|  |
| // REMOVE |
| int command\_number; |
| char tmp[MAX\_BUFFER\_SIZE]; |
|  |
| while (true) { |
| reservation\_count = get\_reservation\_command(); |
| print\_reservation\_list(reservation\_count); |
| fputs("20162448>", stdout); // 프롬프트 출력 |
| fgets(command\_buffer, MAX\_BUFFER\_SIZE, stdin); // 명령행 입력 |
| strcpy(command\_buffer, ltrim(rtrim(command\_buffer))); // 명령행 좌우 공백 제거 |
| make\_command\_token(&command, command\_buffer); // 명령행 토큰화 |
| switch (get\_command\_type(command.argv[0])) { |
|  |
| case ADD: |
|  |
| is\_invalid = false; |
| if(command.argc < 7) { // 인자 개수가 부족할 경우 |
| print\_usage(); |
| break; |
| } |
|  |
| for (int i = 1; i < command.argc; i++) { // 예약 명령 문자열 생성 |
|  |
| // PERIOD 오류 검사 |
| if (i < 6) |
| if (!is\_period(command.argv[i], i)) { |
| is\_invalid = true; |
| break; |
| } |
|  |
| if (i == 1) { |
| sprintf(command\_buffer, "%s", command.argv[i]); |
| continue; |
| } |
| strcat(command\_buffer, " "); |
| strcat(command\_buffer, command.argv[i]); |
| } |
|  |
| if (is\_invalid) { |
| fprintf(stderr, "prompt.ADD: invalid PERIOD\n"); |
| break; |
| } |
| #ifdef DEBUG |
| printf("prompt().ADD: command\_buffer = %s\n", command\_buffer); |
| #endif |
| strcpy(reservation\_command[reservation\_count], command\_buffer); // 예약 목록 배열에 추가 |
| write\_reservation\_file(reservation\_count); // 예약 목록 파일 작성 |
| write\_log(ADD, command\_buffer); |
| break; |
|  |
| case REMOVE: |
|  |
| if (command.argc < 2) { |
| fprintf(stderr, "prompt: COMMAND\_NUMBER doesn't exist\n"); |
| break; |
| } |
|  |
| command\_number = atoi(command.argv[1]); |
|  |
| if (reservation\_count == 0) { |
| fprintf(stderr, "prompt: no reservation command in ssu\_crontab\_file\n"); |
| break; |
| } |
|  |
| if (command\_number < 0 || command\_number > reservation\_count) { |
| fprintf(stderr, "prompt: invalid COMMAND\_NUMBER\n"); |
| break; |
| } |
|  |
| sprintf(command\_buffer, "%s", reservation\_command[command\_number]); |
| #ifdef DEBUG |
| printf("prompt().REMOVE: command\_buffer = %s\n", command\_buffer); |
| #endif |
|  |
| for (int i = command\_number; i < reservation\_count; i++) { // 삭제할 예약 명령을 맨 끝으로 이동 |
| strcpy(tmp, reservation\_command[i]); |
| strcpy(reservation\_command[i], reservation\_command[i + 1]); |
| strcpy(reservation\_command[i + 1], tmp); |
| } |
|  |
| reservation\_count--; |
| write\_reservation\_file(reservation\_count); |
| write\_log(REMOVE, command\_buffer); |
| break; |
|  |
| case EXIT: |
| return; |
|  |
| case UNKNOWN: |
| print\_usage(); |
| break; |
| } |
| free\_command\_token(&command); |
| } |
| } |
|  |
| /\*\* |
| \* @brief 명령 타입 확인 및 번호 변환 |
| \* @param command 명령 문자열 |
| \* @return 명령 타입 번호 |
| \*/ |
| int get\_command\_type(char \*command) // 명령 타입 확인 및 번호 변환 |
| { |
| #ifdef DEBUG |
| printf("get\_command\_type(): command = %s\n", command); |
| #endif |
| if (command == NULL) |
| return false; |
| else if (!strcmp(command, "add")) |
| return ADD; |
| else if (!strcmp(command, "remove")) |
| return REMOVE; |
| else if (!strcmp(command, "exit")) |
| return EXIT; |
| else |
| return UNKNOWN; |
| } |
|  |
| /\*\* |
| \* @brief 문자열 오른쪽 공백 제거 |
| \* @param \_str 대상 문자열 |
| \* @return 공백이 제거된 문자열 |
| \*/ |
| char \*rtrim(char \*\_str) // 문자열 오른쪽 공백 제거 |
| { |
| char tmp[MAX\_BUFFER\_SIZE]; |
| char \*end; |
|  |
| strcpy(tmp, \_str); |
| end = tmp + strlen(tmp) - 1; |
| while (end != \_str && isspace(\*end)) |
| --end; |
|  |
| \*(end + 1) = '\0'; |
| \_str = tmp; |
| return \_str; |
| } |
|  |
| /\*\* |
| \* @brief 문자열 왼쪽 공백 제거 |
| \* @param \_str 대상 문자열 |
| \* @return 공백이 제거된 문자열 |
| \*/ |
| char \*ltrim(char \*\_str) // 문자열 왼쪽 공백 제거 |
| { |
| char \*start = \_str; |
|  |
| while (\*start != '\0' && isspace(\*start)) |
| ++start; |
| \_str = start; |
| return \_str; |
| } |
|  |
| void to\_lower\_case(char \*str) // 문자열 소문자 변환 |
| { |
| int i = 0; |
|  |
| while (str[i]) { |
| if (str[i] >= 'A' && str[i] <= 'Z'){ |
| str[i] = str[i]+32; |
| } |
| i++; |
| } |
| } |
|  |
| /\*\* |
| \* @brief 예약 명령 목록 출력 |
| \*/ |
| void print\_reservation\_list(int count) // 예약 명령 목록 출력 |
| { |
| for(int i = 0; i < count; i++) |
| printf("%d. %s\n", i, reservation\_command[i]); |
| printf("\n"); |
| } |
|  |
| /\*\* |
| \* @brief 주기 인자 검사 |
| \* @param period 주기 문자열 |
| \* @param period\_type 주기 타입 번호 |
| \*/ |
| bool is\_period(char \*period, int period\_type) // 주기 인자 검사 |
| { |
|  |
| int length = strlen(period); |
| char period\_buffer[MAX\_BUFFER\_SIZE]; |
| char period\_token[BUFFER\_SIZE][BUFFER\_SIZE]; |
| int period\_token\_count = 0; |
| char \*tmp; |
| char target[BUFFER\_SIZE]; |
| char unit[BUFFER\_SIZE]; |
| char operator; |
|  |
| // 복사본 문자열 생성 |
| strcpy(period\_buffer, period); |
|  |
| // 주기 문자 검사 |
| for(int i = 0; i < length; i++) { |
| if (!is\_period\_character(period\_buffer[i])) |
| return false; |
| } |
|  |
| // 1-1. 쉼표(,) 분리 |
| if ((tmp = strtok(period\_buffer, ",")) == NULL) // 쉼표 앞에 아무것도 존재하지 않을 경우 |
| return false; |
|  |
| // 1-2. 맨 앞에 기호만 나왔을 경우 |
| if (tmp[0] == '-' || tmp[0] == '/') |
| return false; |
| #ifdef DEBUG |
| printf("is\_period(): type = %d, period\_token[%d] = %s\n", period\_type, period\_token\_count, tmp); |
| #endif |
| strcpy(period\_token[period\_token\_count++], tmp); |
| while ((tmp = strtok(NULL, ",")) != NULL) |
| #ifdef DEBUG |
| { |
| printf("is\_period(): type = %d, period\_token[%d] = %s\n", period\_type, period\_token\_count, tmp); |
| strcpy(period\_token[period\_token\_count++], tmp); |
| } |
| #else |
| strcpy(period\_token[period\_token\_count++], tmp); |
| #endif |
|  |
| // 2. 슬래쉬(/), 바(-) 분리 |
| for(int i = 0; i < period\_token\_count; i++) { |
| operator = 0; |
| memset(target, 0, BUFFER\_SIZE); |
| memset(unit, 0, BUFFER\_SIZE); |
| if (strchr(period\_token[i], '-') && strchr(period\_token[i], '/')) // 슬래시와 범위가 둘다 포함되는 경우 |
| sscanf(period\_token[i], "%[^/]%c%s", target, &operator, unit); |
| else // 둘중 하나만 포함되는 경우 |
| sscanf(period\_token[i], "%[^-/]%c%s", target, &operator, unit); |
| #ifdef DEBUG |
| printf("is\_period(): target = %s, operator = %c, unit = %s\n", target, operator, unit); |
| #endif |
| } |
|  |
| // 3. 슬래시(/)가 존재하지 않을 경우 범위 확인 |
| if (operator == '-') { |
|  |
| // 3-1. 중복된 바(-)가 존재할 경우 |
| if (strchr(unit, '-') != NULL) |
| return false; |
|  |
| // 3-2. 시작, 끝범위가 전체(\*)로 끝날 경우 |
| if (strchr(target, '\*') != NULL || strchr(unit, '\*') != NULL) |
| return false; |
|  |
| // 3-3. 앞의 수가 뒤의 수보다 클 경우 |
| if (atoi(target) > atoi(unit)) |
| return false; |
|  |
| // 3-4 허용 범위를 초과했을 경우 |
| switch (period\_type) { |
| case MINUTE: |
|  |
| if (atoi(target) < 0 || atoi(target) > 59) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(unit) < 0 || atoi(unit) > 59) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case HOUR: |
|  |
| if (atoi(target) < 0 || atoi(target) > 23) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(unit) < 0 || atoi(unit) > 23) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY: |
|  |
| if (atoi(target) < 1 || atoi(unit) > 31) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(unit) < 1 || atoi(unit) > 31) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case MONTH: |
|  |
| if (atoi(target) < 1 || atoi(target) > 12) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(unit) < 1 || atoi(unit) > 12) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY\_OF\_WEEK: |
|  |
| if (atoi(target) < 0 || atoi(target) > 6) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(unit) < 0 || atoi(unit) > 6) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
| } |
|  |
| } else if (operator == '/') { // 4. 슬래시(/)가 존재할 경우 범위 확인 |
|  |
| // 4-1. 중복된 슬래시(/) 또는 바(-)가 존재할 경우 |
| if (strchr(unit, '/') != NULL || strchr(unit, '-') != NULL) |
| return false; |
|  |
| // 4-2. 끝의 범위가 전체(\*)로 끝날 경우 |
| if (strchr(unit, '\*') != NULL) |
| return false; |
|  |
| // 4-3. 앞 토큰이 범위일 경우 |
| if (strchr(target, '-') != NULL) { |
| char front\_target[BUFFER\_SIZE] = { 0 }; |
| char front\_unit[BUFFER\_SIZE] = { 0 }; |
| char front\_operator = 0; |
|  |
| sscanf(target, "%[^-]%c%s", front\_target, &front\_operator, front\_unit); |
| #ifdef DEBUG |
| printf("is\_period(): front\_target = %s, front\_operator = %c, front\_unit = %s\n", front\_target, front\_operator, front\_unit); |
| #endif |
|  |
| // 4-4. 중복된 바(-)가 존재할 경우 |
| if (strchr(front\_unit, '-') != NULL) |
| return false; |
|  |
| // 4-5. 시작, 끝범위가 전체(\*)로 끝날 경우 |
| if (strchr(front\_target, '\*') != NULL || strchr(front\_unit, '\*') != NULL) |
| return false; |
|  |
| // 4-6. 앞의 수가 뒤의 수보다 작을 경우 |
| if (atoi(front\_target) > atoi(front\_unit)) |
| return false; |
|  |
| // 4-7 허용 범위를 초과했을 경우 |
| switch (period\_type) { |
| case MINUTE: |
|  |
| if (atoi(front\_target) < 0 || atoi(front\_target) > 59) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(front\_unit) < 0 || atoi(front\_unit) > 59) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case HOUR: |
|  |
| if (atoi(front\_target) < 0 || atoi(front\_target) > 23) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(front\_unit) < 0 || atoi(front\_unit) > 23) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY: |
|  |
| if (atoi(front\_target) < 1 || atoi(front\_target) > 31) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(front\_unit) < 1 || atoi(front\_unit) > 31) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case MONTH: |
|  |
| if (atoi(front\_target) < 1 || atoi(front\_target) > 12) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(front\_unit) < 1 || atoi(front\_unit) > 12) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY\_OF\_WEEK: |
|  |
| if (atoi(front\_target) < 0 || atoi(front\_target) > 6) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| else if (atoi(front\_unit) < 0 || atoi(front\_unit) > 6) // 뒤의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| } |
| } else { // 4-8. 앞 토큰이 범위가 아닐 경우 |
|  |
| // 4-9. 시작 범위가 숫자일 때 허용 범위 초과했을 경우 |
| if (strchr(target, '\*') == NULL) |
| switch (period\_type) { |
| case MINUTE: |
|  |
| if (atoi(target) < 0 || atoi(target) > 59) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case HOUR: |
|  |
| if (atoi(target) < 0 || atoi(target) > 23) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY: |
|  |
| if (atoi(target) < 1 || atoi(target) > 31) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case MONTH: |
|  |
| if (atoi(target) < 1 || atoi(target) > 12) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY\_OF\_WEEK: |
|  |
| if (atoi(target) < 0 || atoi(target) > 6) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
| } |
|  |
| // 4-10. 끝 범위가 숫자일 때 허용 범위 초과했을 경우 |
| switch (period\_type) { |
| case MINUTE: |
|  |
| if (atoi(unit) < 0 || atoi(unit) > 59) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case HOUR: |
|  |
| if (atoi(unit) < 0 || atoi(unit) > 23) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY: |
|  |
| if (atoi(unit) < 1 || atoi(unit) > 31) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case MONTH: |
|  |
| if (atoi(unit) < 1 || atoi(unit) > 12) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY\_OF\_WEEK: |
|  |
| if (atoi(unit) < 0 || atoi(unit) > 6) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
| } |
| } |
| } else { |
| if(strchr(target, '\*') && strlen(target) != 1) |
| return false; |
|  |
| if(strchr(target, '\*') == NULL) |
| switch (period\_type) { |
| case MINUTE: |
|  |
| if (atoi(target) < 0 || atoi(target) > 59) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case HOUR: |
|  |
| if (atoi(target) < 0 || atoi(target) > 23) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY: |
|  |
| if (atoi(target) < 1 || atoi(target) > 31) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case MONTH: |
|  |
| if (atoi(target) < 1 || atoi(target) > 12) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
|  |
| case DAY\_OF\_WEEK: |
|  |
| if (atoi(target) < 0 || atoi(target) > 6) // 앞의 수가 범위를 초과할 경우 |
| return false; |
| break; |
| } |
| } |
| return true; |
| } |
|  |
| /\*\* |
| \* @brief 주기 문자 검사 |
| \* @param c 주기 문자 |
| \*/ |
| bool is\_period\_character(char c) // 주기 문자 검사 |
| { |
| if (c == '\*') |
| return true; |
| else if (c == '-') |
| return true; |
| else if (c == '/') |
| return true; |
| else if (c == ',') |
| return true; |
| else if (isdigit(c)) |
| return true; |
| else |
| return false; |
| } |
|  |
| /\*\* |
| \* @brief 예약 명령 목록 파일 기록 |
| \*/ |
| void write\_reservation\_file(int count) // 예약 명령 목록 파일 기록 |
| { |
| FILE \*fp; |
|  |
| if ((fp = fopen(CRONTAB\_FILE, "w+")) == NULL) { // 예약 명령 목록 파일 열기 |
| fprintf(stderr, "prompt: fopen error for %s\n", CRONTAB\_FILE); |
| return; |
| } |
| fseek(fp, 0, SEEK\_SET); |
|  |
| for(int i = 0; i <= count; i++) |
| #ifdef DEBUG |
| { |
| printf("write\_reservation\_file: reservation\_command[%d] = %s\n", i, reservation\_command[i]); |
| fprintf(fp, "%s\n", reservation\_command[i]); |
| } |
| #else |
| fprintf(fp, "%s\n", reservation\_command[i]); |
| #endif |
|  |
| fclose(fp); |
| } |
|  |
| /\*\* |
| \* @brief 사용법 출력 |
| \*/ |
| void print\_usage(void) // 사용법 출력 |
| { |
| printf("Usage: [COMMAND]\n"); |
| printf("Command : \n"); |
| printf(" add <PERIOD> <EXECUTE COMMAND> add commands to be excuted periodically in the reservation list\n"); |
| printf(" remove <COMMAND\_NUMBER> remove the execution reservation of COMMAND\_NUMBER from the reservation list\n"); |
| } |

**5-8. ssu\_crond.h**

|  |
| --- |
| /\* |
| \* @file ssu\_crond.h |
| \* @brief ssu\_crond.c에서 사용되는 라이브러리, 매크로, 프로토타입 선언 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #ifndef SSU\_CROND\_H |
| #define SSU\_CROND\_H |
|  |
| #include <fcntl.h> |
| #include <signal.h> |
| #include <sys/types.h> |
| #include <sys/stat.h> |
|  |
| #include "common.h" |
| #include "cron\_support.h" |
|  |
| #define SSU\_CROND\_SERVICE ssu\_crond.service |
|  |
| void \*reservation\_execute(void \*arg); // 예약 명령 실행 스레드 |
| void set\_reservation\_time\_table(char \*period, int period\_type, bool \*reservation\_table); // 예약 시간 테이블 설정 |
| #endif // SSU\_CROND\_H |

**5-9. ssu\_crond.c**

|  |
| --- |
| /\* |
| \* @file ssu\_crond.c |
| \* @brief ssu\_crontab 프로그램 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #include "ssu\_crond.h" |
|  |
| char pwd[BUFFER\_SIZE]; |
| char reservation\_command[BUFFER\_SIZE][MAX\_BUFFER\_SIZE]; |
|  |
| /\*\* |
| \* @brief ssu\_crond 메인 함수 |
| \* @param argc 최초 실행시에 제공되는 인자들의 개수 |
| \* @param argv 최초 실행시에 제공되는 파일들의 경로 |
| \*/ |
| int main(void) |
| { |
| pthread\_t tid[BUFFER\_SIZE]; // 스레드 ID 배열 |
| int reservation\_count; // 예약 명령 개수 |
| int fd; // 예약 명령 파일 디스크럽터 |
| struct stat statbuf; |
| time\_t now; |
|  |
| if (access(CRONTAB\_LOG, F\_OK) < 0) |
| open(CRONTAB\_LOG, O\_CREAT); |
|  |
| // 마지막 수정 시간 획득 |
| fd = open(CRONTAB\_FILE, O\_RDONLY); |
| fstat(fd, &statbuf); |
| now = statbuf.st\_mtime; |
|  |
| getcwd(pwd, BUFFER\_SIZE); |
| #ifdef DEBUG |
| printf("ssu\_crond(): pwd = %s\n", pwd); |
| #endif |
|  |
| while (true) { |
|  |
| reservation\_count = get\_reservation\_command(); // 예약 명령 획득 |
|  |
| for (int i = 0; i < reservation\_count; i++) // 획득한 명령 개수만큼의 스레드 시작 |
| #ifdef DEBUG |
| { |
| printf("ssu\_crond(): reservation\_command[%d] = %s\n", i, reservation\_command[i]); |
| #endif |
| pthread\_create(&tid[i], NULL, reservation\_execute, (void\*)reservation\_command[i]); // 스레드 생성, 인덱스 전달 |
| #ifdef DEBUG |
| printf("ssu\_crond(): tid[%d] = %ld\n", i, tid[i]); |
| } |
| #endif |
|  |
| while (true) { // 예약 명령 파일 수정 확인 |
| fstat(fd, &statbuf); // 파일 상태정보 획득 |
| if (now != statbuf.st\_mtime) { // 파일의 수정시간이 변경되었을 경우 |
| #ifdef DEBUG |
| printf("ssu\_crond(): %s is modified!\n", CRONTAB\_FILE); |
| #endif |
| now = statbuf.st\_mtime; // 최종 수정시간 갱신 |
| for(int i = 0; i < reservation\_count; i++) |
| pthread\_cancel(tid[i]); // 기존 스레드 모두 정지 |
| break; |
| } |
| } |
| } |
| } |
|  |
| /\*\* |
| \* @brief 예약 명령 실행 스레드 |
| \* @param arg 예약 명령 문자열 |
| \*/ |
| void \*reservation\_execute(void \*arg) // 예약 명령 실행 스레드 |
| { |
| CommandToken token; |
| char copy\_reservation\_command[MAX\_BUFFER\_SIZE]; |
| char command[MAX\_BUFFER\_SIZE] = { 0 }; |
| time\_t now\_t; |
| time\_t execute\_t; |
| struct tm now\_tm; |
| bool \*reservation\_time\_table[5]; // 예약 시간 테이블 |
|  |
| strcpy(copy\_reservation\_command, (char\*)arg); |
| #ifdef DEBUG |
| printf("reservation\_excute(): copy\_reservation\_command = %s\n", copy\_reservation\_command); |
| #endif |
|  |
| // 예약 시간 테이블 생성 및 초기화 |
| reservation\_time\_table[0] = calloc(60, sizeof(bool)); // MINUTE |
| reservation\_time\_table[1] = calloc(24, sizeof(bool)); // HOUR |
| reservation\_time\_table[2] = calloc(32, sizeof(bool)); // DAY |
| reservation\_time\_table[3] = calloc(13, sizeof(bool)); // MONTH |
| reservation\_time\_table[4] = calloc(8, sizeof(bool)); // DAY\_OF\_WEEK |
| make\_command\_token(&token, copy\_reservation\_command); // 예약 명령 토큰화 |
| for (int i = 0; i < token.argc; i++) { |
| if (i < 5) // 예약 시간 테이블 설정 |
| set\_reservation\_time\_table(token.argv[i], i + 1, reservation\_time\_table[i]); |
| else { // 실행 명령어 생성 |
| strcat(command, token.argv[i]); |
| strcat(command, " "); |
| } |
| } |
| //free\_command\_token(&token); |
| #ifdef DEBUG |
| printf("reservation\_execute(): command = %s\n", command); |
| #endif |
|  |
| // 현재 시간 추출 |
| now\_t = time(NULL); |
| now\_tm = \*localtime(&now\_t); |
| now\_tm.tm\_sec = 0; |
|  |
| while (true) { |
|  |
| now\_tm.tm\_min++; |
| execute\_t = mktime(&now\_tm); |
| now\_tm = \*localtime(&execute\_t); |
| now\_tm.tm\_sec = 0; |
|  |
| if (reservation\_time\_table[0][now\_tm.tm\_min] && |
| reservation\_time\_table[1][now\_tm.tm\_hour] && |
| reservation\_time\_table[2][now\_tm.tm\_mday] && |
| reservation\_time\_table[3][now\_tm.tm\_mon + 1] && |
| reservation\_time\_table[4][now\_tm.tm\_wday]) { // 실행 시간이 되었을 경우 |
|  |
| now\_t = time(NULL); |
| #ifdef DEBUG |
| printf("reservation\_execute(): %s, execute waitting for %ld second\n", (char\*)arg, execute\_t - now\_t); |
| #endif |
| sleep(execute\_t - now\_t); // 명령어 실행 대기 |
| #ifdef DEBUG |
| printf("reservation\_execute(): %s, running at %s", (char\*)arg, asctime(&now\_tm)); |
| #endif |
| system(command); // 명령어 실행 |
| write\_log(RUN, (char\*)arg); |
| } |
| } |
| } |
|  |
| /\*\* |
| \* @brief 예약 시간 테이블 설정 |
| \* @param period 주기 문자열 |
| \* @param period\_type 주기 타입 번호 |
| \* @param reservation\_table 예약 시간 테이블 |
| \*/ |
| void set\_reservation\_time\_table(char \*period, int period\_type, bool \*reservation\_table) // 예약 시간 테이블 설정 |
| { |
| char period\_token[BUFFER\_SIZE][BUFFER\_SIZE]; |
| char target[BUFFER\_SIZE]; |
| char unit[BUFFER\_SIZE]; |
| char operator; |
| char \*tmp; |
| char \*last; |
| int period\_token\_count = 0; |
| int begin; // 예약 시간 테이블 시작 |
| int end; // 예약 시간 테이블 끝 |
|  |
| #ifdef DEBUG |
| printf("set\_reservation\_time\_table: period = %s, period\_type = %d\n", period, period\_type); |
| #endif |
| switch (period\_type) { |
| case MINUTE: |
| begin = 0; |
| end = 59; |
| break; |
| case HOUR: |
| begin = 0; |
| end = 23; |
| break; |
| case DAY: |
| begin = 1; |
| end = 31; |
| break; |
| case MONTH: |
| begin = 1; |
| end = 12; |
| break; |
| case DAY\_OF\_WEEK: |
| begin = 0; |
| end = 6; |
| break; |
| } |
|  |
| // 1. 쉼표(,) 기준 분리 |
| tmp = strtok\_r(period, ",", &last); |
| strcpy(period\_token[period\_token\_count++], tmp); |
| while ((tmp = strtok\_r(NULL, ",", &last)) != NULL) |
| strcpy(period\_token[period\_token\_count++], tmp); |
|  |
| // 2. 토큰에 해당하는 시간을 설정 |
| for (int i = 0; i < period\_token\_count; i++) { |
|  |
| // 임시 변수 초기화 |
| operator = 0; |
| memset(target, 0, BUFFER\_SIZE); |
| memset(unit, 0, BUFFER\_SIZE); |
|  |
| if (strchr(period\_token[i], '-') && strchr(period\_token[i], '/')) // 슬래시와 범위가 둘다 포함되는 경우 |
| sscanf(period\_token[i], "%[^/]%c%s", target, &operator, unit); |
| else // 둘중 하나만 포함되는 경우 |
| sscanf(period\_token[i], "%[^-/]%c%s", target, &operator, unit); |
| #ifdef DEBUG |
| printf("set\_reservation\_time\_table: target = %s, operator = %c, unit = %s\n", target, operator, unit); |
| #endif |
| // 3. 슬래시(/)가 존재하지 않을 경우: 범위 |
| if (operator == '-') { |
|  |
| for (int j = atoi(target); j <= atoi(unit); j++) |
| reservation\_table[j] = true; |
|  |
| } else if (operator == '/') { // 4. 슬래시(/)가 존재할 경우 |
|  |
| int count = atoi(unit); |
|  |
| // 4-1. 앞 토큰이 범위일 경우 |
| if (strchr(target, '-') != NULL) { |
| char front\_target[BUFFER\_SIZE] = { 0 }; |
| char front\_unit[BUFFER\_SIZE] = { 0 }; |
| char front\_operator = 0; |
| sscanf(target, "%[^-]%c%s", front\_target, &front\_operator, front\_unit); |
| #ifdef DEBUG |
| printf("is\_period(): front\_target = %s, front\_operator = %c, front\_unit = %s\n", front\_target, front\_operator, front\_unit); |
| #endif |
| for (int j = atoi(front\_target); j <= atoi(front\_unit); j++) |
| if(count++ % atoi(unit) == 0) |
| reservation\_table[j] = true; |
|  |
| } else { // 4-2. 앞 토큰이 범위가 아닐 경우 |
|  |
| for (int j = begin; j <= end; j++) |
| if(count++ % atoi(unit) == 0) |
| reservation\_table[j] = true; |
| } |
| } else { // 5. 숫자 혹은 전체(\*)일 경우 |
|  |
| if (strchr(target, '\*') == NULL) // 숫자일 경우 |
| reservation\_table[atoi(target)] = true; |
| else // 전체(\*)일 경우 |
| for (int j = begin; j <= end; j++) |
| reservation\_table[j] = true; |
| } |
| } |
|  |
| } |

**5-10. ssu\_rsync.h**

|  |
| --- |
| /\*\* |
| \* @file ssu\_rsync.h |
| \* @brief ssu\_rsync에서 사용되는 라이브러리, 매크로, 프로토타입 선언 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #ifndef SSU\_RSYNC\_H |
| #define SSU\_RSYNC\_H |
|  |
| #include <fcntl.h> |
| #include <dirent.h> |
| #include <signal.h> |
| #include <utime.h> |
| #include <sys/stat.h> |
| #include <sys/types.h> |
| #include "common.h" |
|  |
| /\*\* |
| \* @brief diff 상태 |
| \*/ |
| #define UNCHCK -1 |
| #define CHCKED 0 |
| #define DELETE 1 |
| #define CREATE 2 |
| #define MODIFY 3 |
|  |
| /\*\* |
| \* @brief 파일 이름 |
| \*/ |
| #define RSYNC\_LOG "ssu\_rsync\_log" |
|  |
| typedef struct ssu\_fileNode{ // 파일 목록 구조체 |
| char name[BUFFER\_SIZE]; // 파일 이름 |
| struct stat attr; // 파일 상태 정보 |
| struct dirent \*\*namelist; // 디렉토리 경우 하위 파일 목록 |
| struct ssu\_fileNode \*next; // 하위 디렉토리 파일 포인터 |
| struct ssu\_fileNode \*child; // 같은 레벨의 다음 파일 포인터 |
| int size; // 파일 크기 |
| int status; // 확인 상태 |
| } file\_node; |
|  |
| void copy\_argument(int argc, char \*argv[]); // 명령행 인자 백업 |
| void syncronize(char \*src\_path, char \*dst\_path); // 동기화 함수 |
| file\_node \*make\_node(void); // 노드 생성 |
| file\_node \*make\_list(char \*path); // 디렉토리 파일 목록 트리화 |
| int count\_size(file\_node \*head); // 디렉토리 크기 반환 |
| void compare\_list(file\_node \*src\_list, file\_node \*dst\_list); // 파일 목록 트리 비교 |
| bool compare\_file(file\_node \*src\_file, file\_node \*dst\_file); // 파일 정보 비교 |
| int write\_change\_list(file\_node \*head, int idx, int status); // 변경사항 목록 작성 |
| void write\_log(int count, unsigned long long totalsize); // 로그 파일 작성 |
| void free\_list(file\_node \*head); // 파일 목록 메모리 할당 해제 |
| void renewal(int count); // 파일 동기화 |
| void renewal\_tar(int count); // tar 동기화 |
| void recovery(int signo); // SIGINT 시그널 처리 |
| void remove\_directory(const char \*path); // 디렉토리 삭제 |
| char \*get\_file\_name(char \*path); // 파일명 추출 |
| void print\_usage(char \*execute\_file); // 사용법 출력 |
| #endif // SSU\_RSYNC\_H |

**5-11. ssu\_rsync.c**

|  |
| --- |
| /\*\* |
| \* @file ssu\_rsync.c |
| \* @brief ssu\_rsync 프로그램 |
| \* @author 김병준 (kbj9704@gmail.com) |
| \*/ |
| #include "ssu\_rsync.h" |
|  |
| // 경로 |
| char pwd[MAX\_BUFFER\_SIZE] = { 0 }; // 프로그램 실행 경로 |
| char src\_path[MAX\_BUFFER\_SIZE] = { 0 }; // 타겟 경로 |
| char dst\_path[MAX\_BUFFER\_SIZE] = { 0 }; // 동기화 경로 |
| char swap\_path[MAX\_BUFFER\_SIZE] = { 0 }; // 스왑 파일 |
|  |
| // 옵션 |
| bool option\_r = false; // R 옵션 |
| bool option\_t = false; // T 옵션 |
| bool option\_m = false; // M 옵션 |
| bool is\_complete = false; // 동기화 완료 확인 |
|  |
| file\_node change\_list[MAX\_BUFFER\_SIZE]; // 변경 목록 |
| char \*\*saved\_argv; |
| int saved\_argc; |
| bool src\_is\_dir = false; |
|  |
|  |
| /\*\* |
| \* @brief ssu\_rsync 메인 함수 |
| \* @param argc 인자 개수 |
| \* @param argv 인자 문자열 |
| \*/ |
| int main(int argc, char \*argv[]) |
| { |
| // 실행 시간 측정 |
| struct timeval begin\_t, end\_t; |
|  |
| // swap 생성 |
| char swap\_path[BUFFER\_SIZE]; |
| char command[MAX\_BUFFER\_SIZE]; |
|  |
| // 유효 검사 |
| char opt; |
| struct stat statbuf; |
| bool is\_invalid = false; |
| bool is\_src = false; |
| bool is\_dst = false; |
|  |
| gettimeofday(&begin\_t, NULL); // 측정 시작 |
|  |
| if (argc < 3) { // 인자 개수가 부족할 경우 |
| print\_usage(argv[0]); |
| exit(1); |
| } |
|  |
| getcwd(pwd, MAX\_BUFFER\_SIZE); |
| copy\_argument(argc, argv); |
|  |
| for (int i = 1; i < argc; i++) { |
| #ifdef DEBUG |
| printf("ssu\_rsync(): argv[%d] = %s\n", i, argv[i]); |
| #endif |
| // 옵션 생략 |
| if (argv[i][0] == '-') |
| continue; |
|  |
| // 타겟 경로 파싱 |
| if (!is\_src) { |
| if (access(argv[i], F\_OK) < 0) { |
| #ifdef DEBUG |
| fprintf(stderr, "ssu\_rsync(): access error for %s\n", argv[i]); |
| #endif |
| is\_invalid = true; |
| break; |
| } |
|  |
| realpath(argv[i], src\_path); // 절대 경로로 변환 |
| #ifdef DEBUG |
| printf("ssu\_rsync(): src\_path = %s\n", src\_path); |
| #endif |
| lstat(src\_path, &statbuf); |
| if (S\_ISDIR(statbuf.st\_mode)) |
| src\_is\_dir = true; |
| #ifdef DEBUG |
| fprintf(stderr, "ssu\_rsync(): dst\_path doesn't directory\n"); |
| #endif |
| is\_src = true; |
| continue; |
| } |
|  |
| // 동기화 경로 파싱 |
| if (!is\_dst) { |
| if (access(argv[i], F\_OK) < 0) { |
| #ifdef DEBUG |
| fprintf(stderr, "ssu\_rsync(): access error for %s\n", argv[i]); |
| #endif |
| is\_invalid = true; |
| break; |
| } |
|  |
| realpath(argv[i], dst\_path); // 동기화 디렉토리 절대 경로 변환 |
| #ifdef DEBUG |
| printf("ssu\_rsync(): dst\_path = %s\n", dst\_path); |
| #endif |
| lstat(dst\_path, &statbuf); // 동기화 디렉토리 상태 정보 획득 |
| if (!S\_ISDIR(statbuf.st\_mode)) { // 동기화 경로가 디렉토리가 아닐 경우 |
| #ifdef DEBUG |
| fprintf(stderr, "ssu\_rsync(): dst\_path doesn't directory\n"); |
| #endif |
| is\_invalid = true; |
| break; |
| } |
| is\_dst = true; |
| continue; |
| } |
| } |
|  |
| // 옵션 파싱 |
| while ((opt = getopt(argc, argv, "rtm")) != -1) { |
| switch (opt) { |
| case 'r': |
| #ifdef DEBUG |
| printf("ssu\_rsync(): R option found\n"); |
| #endif |
| option\_r = true; |
| break; |
| case 't': |
| #ifdef DEBUG |
| printf("ssu\_rsync(): T option found\n"); |
| #endif |
| option\_t = true; |
| break; |
| case 'm': |
| #ifdef DEBUG |
| printf("ssu\_rsync(): M option found\n"); |
| #endif |
| option\_m = true; |
| break; |
| default: |
| #ifdef DEBUG |
| printf("ssu\_rsync(): invalid option found\n"); |
| #endif |
| is\_invalid = true; |
| break; |
| } |
| } |
|  |
| // 파싱 중 에러 발견 |
| if (is\_invalid || !is\_src || !is\_dst) { |
| print\_usage(argv[0]); |
| exit(1); |
| } |
|  |
| strncpy(swap\_path, dst\_path, get\_file\_name(dst\_path) - dst\_path); |
| #ifdef DEBUG |
| printf("ssu\_rsync(): cd %s\n", swap\_path); |
| #endif |
|  |
| chdir(swap\_path); |
|  |
| #ifdef DEBUG |
| sprintf(command, "tar -cvf %s.swp %s", get\_file\_name(dst\_path), get\_file\_name(dst\_path)); // 명령어 생성 |
| printf("ssu\_rsync(): command = %s\n", command); |
| #else |
| sprintf(command, "tar -cf %s.swp %s", get\_file\_name(dst\_path), get\_file\_name(dst\_path)); // 명령어 생성 |
| #endif |
|  |
| system(command); // 명령어 실행(압축) |
| chdir(pwd); // 실행 경로로 복귀 |
| signal(SIGINT, recovery); // SIGINT 시그널 처리 |
| syncronize(src\_path, dst\_path); // 동기화 |
| sprintf(swap\_path, "%.\*s.swp", (int)strlen(dst\_path), dst\_path); |
| #ifdef DEBUG |
| printf("ssu\_rsync(): swap\_path = %s\n", swap\_path); |
| #endif |
| unlink(swap\_path); // swap 파일 삭제 |
|  |
| gettimeofday(&end\_t, NULL); // 측정 종료 |
| ssu\_runtime(&begin\_t, &end\_t); // 실행 시간 출력 |
| exit(0); |
| } |
|  |
| /\*\* |
| \* @brief 명령행 인자 백업 |
| \* @param argc 인자 개수 |
| \* @param argv 인자 문자열 배열 |
| \*/ |
| void copy\_argument(int argc, char \*argv[]) // 명령행 인자 백업 |
| { |
| saved\_argc = argc; |
| saved\_argv = calloc(argc, sizeof(char\*)); |
| for(int i = 0; i < saved\_argc; i++) { |
| saved\_argv[i] = calloc(MAX\_BUFFER\_SIZE, sizeof(char)); |
| strcpy(saved\_argv[i], argv[i]); |
| } |
| } |
|  |
| /\*\* |
| \* @brief 동기화 함수 |
| \* @param src\_path 타겟 경로 |
| \* @param dst\_path 동기화 경로 |
| \*/ |
| void syncronize(char \*src\_path, char \*dst\_path) // 동기화 함수 |
| { |
| file\_node \*src\_list; // 타겟 경로 파일 목록 |
| file\_node \*dst\_list; // 동기화 경로 파일 목록 |
| bool is\_directory = false; |
| int change\_count = 0; |
|  |
| src\_list = make\_list(src\_path); |
| dst\_list = make\_list(dst\_path); |
|  |
| if (S\_ISDIR(src\_list->attr.st\_mode)) { // 타겟이 디렉토리일 경우 |
| compare\_list(src\_list->child, dst\_list->child); |
| is\_directory = true; |
| } else // 타겟이 파일일 경우 |
| compare\_file(src\_list, dst\_list->child); |
|  |
| if (is\_directory) |
| change\_count = write\_change\_list(src\_list->child, change\_count, CREATE); // 생성 혹은 수정된 파일 확인 |
| else |
| change\_count = write\_change\_list(src\_list, change\_count, CREATE); // 생성 혹은 수정된 파일 확인 |
|  |
| if (option\_m) |
| change\_count = write\_change\_list(dst\_list->child, change\_count, DELETE); // 삭제 혹은 수정된 파일 확인 |
|  |
| if (option\_t) |
| renewal\_tar(change\_count); |
| else |
| renewal(change\_count); |
|  |
| free\_list(src\_list); |
| free\_list(dst\_list); |
| is\_complete = true; |
| } |
|  |
| /\*\* |
| \* @brief 노드 생성 |
| \* @return 새로운 노드 |
| \*/ |
| file\_node \*make\_node(void) // 노드 생성 |
| { |
| file\_node \*tmp = calloc(true, sizeof(file\_node)); |
|  |
| memset(tmp->name, 0, BUFFER\_SIZE); |
| tmp->next = NULL; |
| tmp->child = NULL; |
| tmp->namelist = NULL; |
| tmp->size = 0; |
| tmp->status = UNCHCK; |
|  |
| return tmp; |
| } |
|  |
| /\*\* |
| \* @brief 디렉토리 파일 목록 트리화 |
| \* @param path 디렉토리 경로 |
| \* @return 트리 루트 노드 |
| \*/ |
| file\_node \*make\_list(char \*path) // 디렉토리 파일 목록 트리화 |
| { |
| file\_node \*head, \*now; |
| // 파일 : 노드 생성(절대경로/이름, 상태정보) |
| // 디렉토리: 트리 생성 |
| int file\_count; |
| int is\_dirattr = true; |
| int i; |
|  |
| // 부모: 현재 경로, 디렉토리 상태정보, 파일 목록 정보 |
| // 자식: 절대경로 파일이름, 파일 상태정보, 다음 파일 포인터 및 디렉토리 하위 부모노드 포인터 |
| head = make\_node(); |
| now = head; |
|  |
| strcpy(head->name, path); // 현재 경로 저장 |
| stat(head->name, &(head->attr)); // 상태 정보 저장 |
|  |
| file\_count = scandir(head->name, &(head->namelist), NULL, alphasort); // 현재 경로의 모든 파일 탐색 및 개수 저장 |
| for(i = 0; i < file\_count; i++) { |
|  |
| if(!strcmp(head->namelist[i]->d\_name, ".") || !strcmp(head->namelist[i]->d\_name, "..")) // 현재, 상위 디렉토리 접근 지정자 생략 |
| continue; |
|  |
| file\_node \*new = make\_node(); // 새로운 노드 생성 |
|  |
| sprintf(new->name, "%s/%s", head->name, head->namelist[i]->d\_name); // 파일 이름 저장 |
| stat(new->name, &(new->attr)); |
|  |
| if(S\_ISDIR(new->attr.st\_mode)) // 현재 경로의 파일 목록 중 탐색한 파일이 디렉토리일 경우 |
| new = make\_list(new->name); // 해당 디렉토리 파일 목록 트리화 |
| else |
| new->size = new->attr.st\_size; |
|  |
| if(is\_dirattr) { // 현재 노드가 현재 경로의 부모노드일 경우 |
| now->child = new; |
| now = now->child; |
| is\_dirattr = false; |
| } else { // 아닐 경우 형제로 연결 |
| now->next = new; |
| now = now->next; |
| } |
| } |
| head->size = count\_size(head); |
| return head; |
| } |
|  |
| /\*\* |
| \* @brief 디렉토리 크기 반환 |
| \* @param 디렉토리 노드 |
| \* @return 디렉토리 크기 |
| \*/ |
| int count\_size(file\_node \*head) // 디렉토리 크기 반환 |
| { |
| file\_node \*now; |
| int size; |
|  |
| size = false; |
|  |
| if(S\_ISDIR(head->attr.st\_mode)) |
| now = head->child; |
| else |
| return head->attr.st\_size; |
|  |
| while(now != NULL) { |
| size += now->size; |
| now = now->next; |
| } |
|  |
| return size; |
| } |
|  |
| /\*\* |
| \* @brief 파일 목록 트리 비교 |
| \* @param src\_list 타겟 파일 목록 |
| \* @param dst\_list 동기화 디렉토리 파일 목록 |
| \*/ |
| void compare\_list(file\_node \*src\_list, file\_node \*dst\_list) // 파일 목록 트리 비교 |
| { |
| file\_node \*now; |
|  |
| if (src\_list == NULL || dst\_list == NULL) { // 둘중 하나라도 비교 대상이 존재하지 않을 경우 |
| #ifdef DEBUG |
| printf("compare\_list(): there is NULL node exist\n"); |
| #endif |
| return; |
| } |
|  |
| now = src\_list; |
|  |
| while (now != NULL) { // 타겟 파일 탐색 |
|  |
| compare\_file(now, dst\_list); |
|  |
| if (now->child != NULL) |
| compare\_list(now->child, dst\_list); |
|  |
| now = now->next; |
| } |
| } |
|  |
| /\*\* |
| \* @brief 파일 정보 비교 |
| \* @param src\_file 타겟 파일 노드 |
| \* @param dst\_file 동기화 디렉토리 파일 노드 |
| \* @return 비교 성공 유무 |
| \*/ |
| bool compare\_file(file\_node \*src\_file, file\_node \*dst\_file) // 파일 정보 비교 |
| { |
| file\_node \*now; |
|  |
| now = dst\_file; |
|  |
| while (now != NULL) { |
|  |
| #ifdef DEBUG |
| if (src\_is\_dir) |
| printf("compare\_file(): src\_file->name = %s, dst\_file->name = %s\n", src\_file->name + strlen(src\_path) + 1, now->name + strlen(dst\_path) + 1); |
| else |
| printf("compare\_file(): src\_file->name = %s, dst\_file->name = %s\n", get\_file\_name(src\_file->name), now->name + strlen(dst\_path) + 1); |
| #endif |
| if ((src\_is\_dir && !strcmp(src\_file->name + strlen(src\_path) + 1, now->name + strlen(dst\_path) + 1)) |
| || (!src\_is\_dir && !strcmp(get\_file\_name(src\_file->name), now->name + strlen(dst\_path) + 1))) { // 파일 이름이 같은 경우 |
|  |
| #ifdef DEBUG |
| printf("compare\_file(): file found\n"); |
| #endif |
| src\_file->status = CHCKED; |
|  |
| if (src\_file->attr.st\_mode != now->attr.st\_mode) { // 1. 파일 형식이 다를 경우 |
| #ifdef DEBUG |
| printf("compare\_file(): type different\n"); |
| #endif |
| src\_file->status = MODIFY; |
| } else if (src\_file->attr.st\_mtime != now->attr.st\_mtime) { // 2. 수정시간이 다를 경우 |
| #ifdef DEBUG |
| printf("compare\_file(): mtime different\n"); |
| #endif |
| src\_file->status = MODIFY; |
| } else if (src\_file->size != now->size) { // 3. 크기가 다를 경우 |
|  |
| #ifdef DEBUG |
| printf("compare\_file(): size different\n"); |
| #endif |
| src\_file->status = MODIFY; |
| } |
|  |
| now->status = CHCKED; |
| #ifdef DEBUG |
| printf("compare\_file(): src\_file->status = %d, dst\_file->status = %d\n", src\_file->status, now->status); |
| #endif |
| return true; |
| } |
|  |
| if(now->child != NULL) // 디렉토리 안에 파일이 존재할 경우 |
| if(compare\_file(src\_file, now->child)) |
| break; |
|  |
| now = now->next; |
| } |
|  |
| return false; |
| } |
|  |
| /\*\* |
| \* @brief 변경사항 목록 작성 |
| \* @param head 트리 루트 노드 |
| \* @param idx 변경사항 목록 시작 인덱스 |
| \* @param status 변경 사항 타입 번호 |
| \* @param is\_first 첫번째 레벨 확인 변수 |
| \*/ |
| int write\_change\_list(file\_node \*head, int idx, int status) // 변경사항 목록 작성 |
| { |
| file\_node \*now; |
|  |
| now = head; |
|  |
| while (now != NULL) { |
|  |
| switch (now->status) { |
|  |
| case UNCHCK: |
|  |
| switch (status) { |
|  |
| case CREATE: |
| strcpy(change\_list[idx].name, now->name); |
| change\_list[idx].status = CREATE; |
| break; |
|  |
| case DELETE: |
| strcpy(change\_list[idx].name, now->name); |
| change\_list[idx].status = DELETE; |
| break; |
|  |
| } |
| change\_list[idx++].size = now->size; |
| #ifdef DEBUG |
| printf("write\_change\_list(): change\_list[%d] = %s(%dbyte), status = %d\n", idx - 1, change\_list[idx - 1].name, change\_list[idx - 1].size, change\_list[idx - 1].status); |
| #endif |
| break; |
|  |
| case MODIFY: // 수정됨 |
| strcpy(change\_list[idx].name, now->name); |
| change\_list[idx].status = MODIFY; |
| change\_list[idx++].size = now->size; |
| #ifdef DEBUG |
| printf("write\_change\_list(): change\_list[%d] = %s(%dbyte), status = %d\n", idx - 1, change\_list[idx - 1].name, change\_list[idx - 1].size, change\_list[idx - 1].status); |
| #endif |
| break; |
| } |
|  |
| if (option\_r && now->child != NULL) // R옵션이 주어진 경우 |
| idx = write\_change\_list(now->child, idx, status); |
|  |
| now = now->next; |
| } |
|  |
| return idx; |
| } |
|  |
| /\*\* |
| \* @brief 파일 동기화 |
| \* @param count 변경 사항 개수 |
| \*/ |
| void renewal(int count) // 파일 동기화 |
| { |
| int src\_fd, dst\_fd; |
| char path[MAX\_BUFFER\_SIZE]; |
| char buf[MAX\_BUFFER\_SIZE]; |
| struct stat statbuf; |
| struct utimbuf attr; |
| size\_t length; |
|  |
|  |
| for (int i = 0; i < count; i++) { |
|  |
| switch (change\_list[i].status) { |
| case DELETE: |
|  |
| lstat(change\_list[i].name, &statbuf); |
| if (S\_ISDIR(statbuf.st\_mode)) |
| remove\_directory(change\_list[i].name); |
| else |
| remove(change\_list[i].name); |
| break; |
|  |
| case CREATE: |
| case MODIFY: |
|  |
| memset(path, 0, MAX\_BUFFER\_SIZE); |
|  |
| lstat(change\_list[i].name, &statbuf); |
| if (src\_is\_dir) |
| sprintf(path, "%s/%s", dst\_path, change\_list[i].name + strlen(src\_path) + 1); // 동기화된 파일 경로 생성 |
| else |
| sprintf(path, "%s/%s", dst\_path, change\_list[i].name + strlen(src\_path) - strlen(get\_file\_name(src\_path))); // 동기화된 파일 경로 생성 |
| #ifdef DEBUG |
| printf("renewal: path = %s\n", path); |
| #endif |
|  |
| if (S\_ISDIR(statbuf.st\_mode)) |
| mkdir(path, 0755); |
| else { |
| if ((src\_fd = open(change\_list[i].name, O\_RDONLY)) < 0) { // 타겟 읽기 전용 열기 |
| fprintf(stderr, "renewal(): open error for %s\n", change\_list[i].name); |
| break; |
| } |
|  |
| if ((dst\_fd = open(path, O\_WRONLY | O\_CREAT | O\_TRUNC, statbuf.st\_mode)) < 0) { // 동기화 파일 열기 |
| fprintf(stderr, "renewal(): open error for %s\n", path); |
| break; |
| } |
|  |
| while ((length = read(src\_fd, buf, MAX\_BUFFER\_SIZE)) > 0) // 동기화 파일 쓰기 (새로쓰기 혹은 덮어쓰기) |
| write(dst\_fd, buf, length); |
|  |
| close(src\_fd); |
| close(dst\_fd); |
| } |
|  |
| // 동기화 파일 속성 및 권한 복원 |
| attr.actime = statbuf.st\_atime; |
| attr.modtime = statbuf.st\_mtime; |
| utime(path, &attr); |
| chmod(path, statbuf.st\_mode); |
| break; |
| } |
| } |
| write\_log(count, 0); |
| } |
|  |
| /\*\* |
| \* @brief tar 동기화 |
| \* @param count 변경 사항 개수 |
| \*/ |
| void renewal\_tar(int count) |
| { |
| char file\_name[BUFFER\_SIZE] = { 0 }; |
| char path[MAX\_BUFFER\_SIZE] = { 0 }; |
| char command[MAX\_BUFFER\_SIZE] = { 0 }; |
| struct stat statbuf; |
| unsigned long long size; |
|  |
| if (count == 0) { |
| fprintf(stderr, "write\_log(): already up to date in %s\n", dst\_path); |
| return; |
| } |
|  |
| // 1. 타겟 디렉토리 tar 생성 |
| sprintf(file\_name, "%s.tar", get\_file\_name(dst\_path)); // tar 파일 이름 생성 |
| #ifdef DEBUG |
| printf("renewal\_tar(): cd %s\n", path); |
| #endif |
| if (src\_is\_dir) // SRC가 디렉토리일 경우 SRC디렉토리 내부로 이동 |
| chdir(src\_path); |
| else { // SRC가 파일일 경우 SRC가 존재하는 경로로 이동 |
| strncpy(path, src\_path, get\_file\_name(src\_path) - src\_path - 1); |
| chdir(path); |
| } |
|  |
| // 2. 압축 실행 |
| for(int i = 0; i < count; i++) { |
| switch (change\_list[i].status) { |
| case DELETE: |
|  |
| lstat(change\_list[i].name, &statbuf); |
| if (S\_ISDIR(statbuf.st\_mode)) |
| remove\_directory(change\_list[i].name); |
| else |
| remove(change\_list[i].name); |
| break; |
|  |
| case CREATE: |
| case MODIFY: |
|  |
| // 압축 파일에 파일 추가 |
| if (src\_is\_dir) |
| #ifdef DEBUG |
| sprintf(command, "tar -rvf %s %s", file\_name, change\_list[i].name + strlen(src\_path) + 1); |
| #else |
| sprintf(command, "tar -rf %s %s", file\_name, change\_list[i].name + strlen(src\_path) + 1); |
| #endif |
| else |
| #ifdef DEBUG |
| sprintf(command, "tar -rvf %s %s", file\_name, get\_file\_name(src\_path)); |
| #else |
| sprintf(command, "tar -rf %s %s", file\_name, get\_file\_name(src\_path)); |
| #endif |
| #ifdef DEBUG |
| printf("renewal\_tar(): command = %s\n", command); |
| #endif |
| system(command); // 압축 실행 |
| break; |
| } |
| } |
|  |
| lstat(file\_name, &statbuf); // 상태 정보 획득 |
| size = statbuf.st\_size; // 압축 파일 크기 획득 |
|  |
| // DST 내부로 tar 파일 이동 |
| sprintf(path, "%s/%.\*s", dst\_path, (int)strlen(file\_name), file\_name); // 이동 시킬 경로 생성 |
| #ifdef DEBUG |
| printf("renewal\_tar(): mv %s %s\n", file\_name, path); |
| #endif |
| rename(file\_name, path); // 압축 파일 이동 |
|  |
| // DST 내부로 이동 |
| #ifdef DEBUG |
| printf("renewal\_tar(): cd %s\n", dst\_path); |
| #endif |
| chdir(dst\_path); |
|  |
| #ifdef DEBUG |
| sprintf(command, "tar -xvf %s", file\_name); |
| printf("renewal\_tar(): command = %s\n", command); |
| #else |
| sprintf(command, "tar -xf %s", file\_name); |
| #endif |
| system(command); |
| unlink(file\_name); |
| chdir(pwd); |
|  |
| write\_log(count, size); |
| } |
|  |
| /\*\* |
| \* @brief 로그 파일 작성 |
| \* @param count 변경 사항 개수 |
| \*/ |
| void write\_log(int count, unsigned long long totalsize) // 로그 파일 작성 |
| { |
| FILE \*fp; |
| time\_t now\_t; |
| struct tm \*now\_tm; |
| char command[MAX\_BUFFER\_SIZE]; |
|  |
| if (count == 0) { |
| fprintf(stderr, "write\_log(): already up to date in %s\n", dst\_path); |
| return; |
| } |
|  |
| if ((fp = fopen(RSYNC\_LOG, "r+")) == NULL) // 로그파일 열기 |
| fp = fopen(RSYNC\_LOG, "w"); // 존재하지 않을 경우 생성 |
|  |
| fseek(fp, 0, SEEK\_END); |
|  |
| // 헤더 명령행 문자열 생성 |
| strcpy(command, get\_file\_name(saved\_argv[0])); |
| for (int i = 1; i < saved\_argc; i++) { |
| strcat(command, " "); |
| strcat(command, saved\_argv[i]); |
| } |
|  |
| #ifdef DEBUG |
| printf("write\_log(): command = %s\n", command); |
| #endif |
|  |
| // 헤더 시간 생성 |
| time(&now\_t); |
| now\_tm = localtime(&now\_t); |
|  |
| fprintf(fp, "[%.24s] %s\n", asctime(now\_tm), command); // 헤더 라인 쓰기 |
|  |
| if (option\_t) |
| fprintf(fp, " totalsize %llubytes\n", totalsize); |
|  |
| for (int i = 0; i < count; i++) // 변경 사항 쓰기 |
| switch (change\_list[i].status) { |
| case DELETE: |
| fprintf(fp, " %s delete\n", change\_list[i].name + strlen(dst\_path) + 1); |
| break; |
| case CREATE: |
| case MODIFY: |
| if (src\_is\_dir) |
| if (option\_t) |
| fprintf(fp, " %s\n", change\_list[i].name + strlen(src\_path) + 1); |
| else |
| fprintf(fp, " %s %dbytes\n", change\_list[i].name + strlen(src\_path) + 1, change\_list[i].size); |
| else |
| if (option\_t) |
| fprintf(fp, " %s\n", change\_list[i].name + strlen(src\_path) - strlen(get\_file\_name(src\_path))); |
| else |
| fprintf(fp, " %s %dbytes\n", change\_list[i].name + strlen(src\_path) - strlen(get\_file\_name(src\_path)), change\_list[i].size); |
| break; |
| } |
| } |
|  |
| /\*\* |
| \* @brief 모니터링 파일 목록 메모리 할당 해제 |
| \* @param head 트리의 루트 노드 |
| \*/ |
| void free\_list(file\_node \*head) // 모니터링 파일 목록 메모리 할당 해제 |
| { |
| // 모든 노드들을 찾아서 메모리 할당을 해제한다. |
| if(head->child != NULL) // 자식 탐색 |
| free\_list(head->child); |
|  |
| if(head->next != NULL) // 형제 탐색 |
| free\_list(head->next); |
|  |
| free(head->namelist); |
| free(head); // 메모리 엑세스 허용 |
| } |
|  |
| /\*\* |
| \* @brief SIGINT 시그널 처리 |
| \* @param signo 시그널 |
| \*/ |
| void recovery(int signo) // SIGINT 시그널 처리 |
| { |
| char file\_name[BUFFER\_SIZE] = { 0 }; |
| char command[MAX\_BUFFER\_SIZE] = { 0 }; |
| char path[MAX\_BUFFER\_SIZE] = { 0 }; |
|  |
| if(signo == SIGINT) { // SIGINT 시그널 획득 시 |
| #ifdef DEBUG |
| printf("recovery(): SIGINT signal is arrived\n"); |
| #endif |
| if(is\_complete) // 동기화가 완료되었을 경우 |
| return; |
|  |
| strncpy(path, dst\_path, get\_file\_name(dst\_path) - dst\_path); |
| #ifdef DEBUG |
| printf("recovery(): cd %s\n", path); |
| #endif |
| chdir(path); |
|  |
| sprintf(file\_name, "%s.swp", get\_file\_name(dst\_path)); |
| #ifdef DEBUG |
| sprintf(command, "tar -xvf %s", file\_name); // 복원 명령어 생성(압축 해제) |
| printf("recovery(): command = %s\n", command); |
| #else |
| sprintf(command, "tar -xf %s", file\_name); // 복원 명령어 생성(압축 해제) |
| #endif |
| remove\_directory(dst\_path); // 기존 동기화 디렉토리 삭제 |
| system(command); // 복원 명령어 실행 |
| unlink(file\_name); // swap 파일 삭제 |
|  |
| } |
| exit(1); |
| } |
|  |
| /\*\* |
| \* @brief 디렉토리 삭제 |
| \* @param path 삭제할 디렉토리 경로 |
| \*/ |
| void remove\_directory(const char \*path) // 디렉토리 삭제 |
| { |
| struct dirent \*dirp; |
| struct stat statbuf; |
| DIR \*dp; |
| char tmp[MAX\_BUFFER\_SIZE]; |
|  |
| if ((dp = opendir(path)) == NULL) |
| return; |
|  |
| while ((dirp = readdir(dp)) != NULL) { // path에 존재하는 디렉토리 안에 파일들 전부 삭제 |
| if (!strcmp(dirp->d\_name, ".") || !strcmp(dirp->d\_name, "..")) |
| continue; |
|  |
| sprintf(tmp, "%s/%s", path, dirp->d\_name); // tmp = 디렉토리 내부 파일 |
|  |
| if (lstat(tmp, &statbuf) == -1) // 파일 상태 정보 추출 |
| continue; |
|  |
| if (S\_ISDIR(statbuf.st\_mode)) // 디렉토리일 경우 재귀적으로 제거 |
| remove\_directory(tmp); |
| else |
| unlink(tmp); |
| } |
| closedir(dp); |
| rmdir(path); |
| } |
|  |
| /\*\* |
| \* @brief 파일명 반환 |
| \* @param path 경로 |
| \* @return 파일명 시작 메모리 주소 |
| \*/ |
| char \*get\_file\_name(char \*path) // 파일명 추출 |
| { |
| char \*tmp = path; |
| int length = strlen(path); |
|  |
| for (int i = 0; i < length; i++) |
| if(path[i] == '/') |
| tmp = path + i; |
|  |
| return tmp + 1; |
| } |
|  |
| /\*\* |
| \* @brief 사용법 출력 |
| \* @param 실행파일 문자열 |
| \*/ |
| void print\_usage(char \*execute\_file) // 사용법 출력 |
| { |
| fprintf(stderr, "ssu\_rsync(): Usage: %s [OPTION] <SOURCE> <DESTINATION>\n", execute\_file); |
| } |