설계과제 3 개요 : ssu_crontab, ssu_rsync

Linux System Programming, School of CSE, Soongsil University, Spring 2020

○ 개요
- 리눅스 시스템 상에서 사용자가 동기화를 원하는 파일이나 디렉토리를 동기화하는 프로그램인 ssu_rsync와 리눅스 시스템 상에서 사용자가 주기적으로 실행하고자 하는 명령어를 등록하고 실행시켜주는 ssu_crontab을 구현한다.

○ 목표
- 새로운 명령어를 시스템 함수를 사용하여 구현함으로써 쉘의 원리를 이해하고, 유닉스/리눅스 시스템에서 제공하는 여러 시스템 자료구조를 이용하여 프로그램을 작성함으로써 시스템 프로그래밍 설계 및 응용 능력을 향상

○ 팀 구성
- 개인별 프로젝트

○ 보고서 제출 방법
- 설계과제는 " 보고서.hwp "(개요, 상세설계, 구현방법, 결과 및 소스코드와 실행결과가 함께 있는 워드(hwp 또는 MS-Word) 파일))와 " 소스코드 "(makefile, obj, *.c, *.h 등 컴파일하고 실행하기 위한 모든 파일)를 제출해야 함
- 모든 설계과제 결과물은 "#P설계과제번호_학번_버전.zip" (예. #P3_20190000_v1.0.zip)형태로 파일 이름을 명명하고, zip프로그램으로 압축하여 제출해야 함.
- 압축파일 내 "보고서" 디렉토리와 "소스코드" 디렉토리 2개 만들어 제출해야 함
- 제출한 압축 파일을 풀었을 때 해당 디렉토리에서 컴파일 및 실행이 되어야 함. 해당 디렉토리에서 컴파일이나 실행되지 않을 경우, 기본과제 및 설계과제 제출 방법을 따르지 않는 경우 감점 20% 외 추가 20% 감점
- 기타 내용은 syllabus 참고

○ 제출 기한
- 6월 10일(수) 오후 11시 59분 59초 (서버 시간이 30분 정도 빠를 수 있기 때문에 1시간 지연 허용)

○ 보고서 양식
- 보고서는 다음과 같은 양식으로 작성

1. 과제 개요 // 위 개요를 더 상세하게 작성
2. 설계 // 함수 기능별 흐름도(순서도) 반드시 포함
3. 구현 // 함수 프로토타입 반드시 포함
4. 테스트 및 결과 // 테스트 프로그램의 실행 결과 캡쳐 및 분석
5. 소스코드 // 주석

○ ssu_crontab 프로그램 기본 사항
- 사용자가 주기적으로 실행하는 명령어를 "ssu_crontab_file"에 저장 및 삭제하는 프로그램
- 주기적으로 "ssu_crontab_file"에 저장된 명령어를 실행시킬 "ssu_crond" 디몬 프로그램
  - ✓ "ssu_crond"는 운영체제 시작 시 함께 실행되어 "ssu_crontab_file"에 저장된 명령어를 주기적으로 실행시킴
- "ssu_crontab_file"에 저장된 명령어가 정상적으로 수행된 경우만 "ssu_crontab_log"로그파일에 다음 사항을 기록

예. 로그 파일 작성 예시

```
oslab@oslab-VirtualBox:~/lsp$ cat ssu_crontab_log
[Mon May 11 11:57:23 2020] add * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 11:57:34 2020] add 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
[Mon May 11 11:57:51 2020] add 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt
[Mon May 11 11:58:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 11:59:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:00:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:00:00 2020] run 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt
[Mon May 11 12:00:46 2020] add * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:01:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:01:00 2020] run * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:02:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:02:00 2020] run * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:03:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:03:00 2020] run * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:04:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:04:00 2020] run * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:05:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:05:00 2020] run * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:05:13 2020] remove * * * * * echo "add command" > echomsg.txt
[Mon May 11 12:06:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
[Mon May 11 12:07:00 2020] run * * * * * echo "echo every minute" > echomsg.txt
```

✓ 출력형태

- [수행시간] 프롬프트명령어 실행주기 명령어 명령어 옵션
- [수행시간] : 요일 월 날짜 명령어 실행시작 시간
- 프롬프트 명령어 : add, remove, run 중 하나표시
  - add : ssu_crontab을 통해 명령어가 저장된 경우
  - remove : ssu_crontab을 통해 명령어가 삭제된 경우
  - run : ssu_crond를 통해 명령어가 수행된 경우
- 실행주기 : 분 시 일 월 요일
  - 각 항목의 값으로 들어가는 내용은 설계 및 구현에서 설명
- 명령어와 명령어 옵션

✓ “ssu_crontab_file”에 명령어가 추가, 제거 될 때마다 “ssu_crontab_log”로그파일에 기록

✓ “ssu_crond”가 “ssu_crontab_file”에 저장된 명령어를 주기적으로 실행할 때마다 “ssu_crontab_log”로그파일에 기록

○ ssu_rsync 프로그램 기본 사항

- 명령어 형태 : ssu_rsync src(소스파일 또는 디렉토리) dst(목적 디렉토리)
- 인자로 주어진 src 파일 혹은 디렉토리를 dst 디렉토리에 동기화
- 동기화 시 src 파일이 dst 디렉토리 내 동일한 파일(파일 이름과 파일 크기, 수정시간이 같은 경우)로 존재할 경우 해당 파일은 동기화 하지 않음
- 예1) src가 일반 파일인 경우 src와 동일한 이름의 dst 디렉토리 내에 일반파일로 존재할 경우 해당 파일은 동기화 하지 않음
- 예2) src가 디렉토리 파일인 경우 src 디렉토리 내 모든 파일과 dst 디렉토리 내 모든 파일을 비교하여 동일한 일반 파일이 존재할 경우 해당 파일은 동기화하지 않음
- 동기화 작업 도중 SIGINT가 발생하면 동기화 작업이 취소되고 dst 디렉토리 내 파일들이 동기화하지 않은 상태로 유지되어야 함
- 단 동기화 중에는 사용자가 동기화 중인 파일을 open 하는 것은 허용하지 않음
- src 파일 dst 디렉토리 내 파일의 동기화가 모두 완료되면 “ssu_rsync_log”로그파일에 기록

✓ 출력형태

- [동기화 완료시간] 명령어 및 명령어 옵션
  파일 이름 파일 크기
- 동기화 완료시간 : 요일 월 날짜 명령어 동기화 완료시간
- 명령어 및 명령어 옵션

- 파일 이름 : 동기화한 파일의 이름(src 디렉토리 기준 상대경로)
- 파일 크기 : 동기화한 파일의 크기, 단. 파일이 삭제된 경우 delete를 기록

예. 로그 파일 작성 예시

```
oslab@oslab-VirtualBox:~/lsp$ cat ssu_rsync_log
[Mon May 11 20:00:46 2020] ssu_rsync src dst
        b.c 110bytes
        a.txt 8bytes
[Mon May 11 20:08:13 2020] ssu_rsync -r src dst
        b.c 110bytes
        a.txt 8bytes
        dir/d.c 64bytes
        dir/c.txt 5bytes
[Mon May 11 20:17:23 2020] ssu_rsync -t src dst
        totalSize 10240bytes
        b.c
        a.txt
[Mon May 11 20:22:34 2020] ssu_rsync -m src dst
        b.c 110bytes
        a.txt 8bytes
        e.txt delete
```

○ 설계 및 구현

- 가) ssu_crontab

✓ ssu_crontab 실행 후 다음과 같은 프롬프트 출력

- 프롬프트 모양 : “ssu_crontab_file”에 저장된 모든 명령어 출력 및 개행 후, 공백 없이 학번, ‘>’ 문자 출력.

실행 예. prompt

```
0. * * * * * echo "echo every minute" > echomsg.txt
1. 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
2. 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt

20200000>
```

- 프롬프트에서 실행 가능한 명령어 : add, remove, exit
- 이외 명령어 입력 시 에러 처리 후 프롬프트로 제어가 넘어감
- 엔터만 입력 시 프롬프트 재출력
- exit 명령이 입력될 때까지 위에서 지정한 실행 가능 명령어를 입력받아 실행

✓ add <실행주기> <명령어>

- “ssu_crontab_file”에 실행주기와 명령어가 기록되어야 하며 “ssu_crontab_file”파일이 없을 경우 생성함
- 실행주기
  - 실행주기의 각 항목은 분 시 일 월 요일 의 5가지 항목으로 구성됨
  - 각 항목은 분(0-59), 시(0-23), 일(1-31), 월(1-12), 요일(0-6, 0은 일요일 1~6순으로 월요일에서 토요일을 의미)의 범위를 가짐
  - 각 항목의 값은 ‘*’, ‘-’, ‘,’, ‘/’ 기호를 사용하여 설정할 수 있음
    - ‘*’ : 해당 필드의 모든 값을 의미함
    - ‘-’ : ‘-’ 으로 연결된 값 사이의 모든 값을 의미함(범위 지정)
    - ‘,’ : ‘,’ 로 연결된 값들 모두를 의미함(목록)
    - ‘/’ : 앞에 나온 주기의 범위를 뒤에 나온 숫자만큼 건너뛰는 것을 의미함
  - 예) 0 0,12 */2 * * -> 2일마다 0시 0분, 12시 0분에 작업수행
- 명령어 인자에는 주기적으로 수행할 명령어 및 해당 명령어의 옵션까지 모두 입력

- 실행주기의 입력이 잘못 된 경우 에러 처리 후 프롬프트로 제어가 넘어감
- add 프롬프트 명령어를 통해 명령어가 “ssu_crontab_file” 파일에 저장되면 “ssu_crontab_log” 로그파일에 로그를 남김

실행 예. add

```
0. * * * * * echo "echo every minute" > echomsg.txt
1. 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
2. 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt

20200000> add * * * * * echo "add command" > echomsg.txt
0. * * * * * echo "echo every minute" > echomsg.txt
1. 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
2. 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt
3. * * * * * echo "add command" > echomsg.txt

20200000>
```

✓ remove <COMMAND_NUMBER>
- 옵션으로 입력한 번호의 명령어를 제거
- 잘못된 COMMAND_NUMBER가 입력된 경우 에러 처리 후 프롬프트로 제어가 넘어감
- COMMAND_NUMBER를 입력하지 않은 경우 에러 처리 후 프롬프트로 제어가 넘어감
- remove 프롬프트 명령어를 통해 명령어가 “ssu_crontab_file” 파일에서 삭제되면 “ssu_crontab_log” 로그파일에 로그를 남김

실행 예. remove

```
0. * * * * * echo "echo every minute" > echomsg.txt
1. 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
2. 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt
3. * * * * * echo "add command" > echomsg.txt

20200000> remove 3
0. * * * * * echo "echo every minute" > echomsg.txt
1. 0 5 * * 0 echo "echo every sunday at 5am" > echomsg.txt
2. 0 0,12 */2 * * "echo every two days at 0am, 12am" > echomsg.txt

20200000>
```

✓ exit
- 프로그램 종료

- 나) ssu_crond
✓ 운영체제 시작 시 함께 실행되어 “ssu_crontab_file” 에 저장된 명령어를 주기적으로 실행시키는 디몬 프로그램
✓ “ssu_crond” 는 “ssu_crontab_file” 을 읽어 주기적으로 명령어를 실행해야 함
✓ “ssu_crontab_file” 에 명령어가 저장 및 삭제되면 이를 반영 해야함
✓ “ssu_crond” 를 통해 명령어가 수행되면 “ssu_crontab_log” 로그파일에 로그를 남김

- 다) ssu_rsync [option] <src> <dst>
✓ “ssu_rsync” 는 인자로 주어진 src 파일 혹은 디렉토리를 dst 디렉토리에 동기화함
✓ 동기화 시 src 파일이 dst 디렉토리 내 동일한 파일(파일 이름과 파일 크기, 수정시간이 같은 경우)이 존재하지 않는 경우 src 파일을 dst 디렉토리 내에 복사함
✓ 동기화 시 dst 디렉토리 내에 파일 이름이 같은 다른 파일(파일 이름은 같으나 파일 크기 혹은

수정시간이 다른 경우)이 존재할 경우 dst 디렉토리 내의 파일을 src 파일로 대체함

✓ dst 디렉토리 내의 동기화된 파일은 “ssu_rsync” 프로그램에서 src 디렉토리의 파일과 같은 파일로 인식할 수 있어야함

✓ src 인자는 파일 및 디렉토리 모두 허용

- 상대경로와 절대경로 모두 입력 가능
- 인자로 입력받은 파일 혹은 디렉토리를 찾을 수 없으면 usage 출력 후 프로그램 종료
- 인자로 입력받은 파일 혹은 디렉토리의 접근권한이 없는 경우 usage 출력 후 프로그램 종료

✓ dst 인자는 디렉토리만 허용

- 상대경로와 절대경로 모두 입력 가능
- 인자로 입력받은 디렉토리를 찾을 수 없으면 usage 출력 후 프로그램 종료
- 인자로 입력받은 디렉토리가 디렉토리 파일이 아니라면 usage 출력 후 프로그램 종료
- 인자로 입력받은 디렉토리의 접근권한이 없는 경우 usage 출력 후 프로그램 종료

✓ ‘-r’ 옵션을 설정하지 않은 경우 인자로 지정한 src의 서브디렉토리는 동기화하지 않음

✓ ‘-m’ 옵션을 설정하지 않은 경우 src 디렉토리에 존재하지 않는 파일 및 디렉토리가 dst 디렉토리에 존재할 수 있음

✓ 동기화 작업 도중 SIGINT가 발생하면 동기화 작업이 취소되고 dst 디렉토리 내 파일들이 동기화하지 않은 상태로 유지되어야 함

- 라) -r

✓ -r 옵션이 설정되면 src의 서브디렉토리 내의 파일 또한 동기화함

✓ dst 디렉토리에 동기화할 서브디렉토리가 없는 경우 디렉토리를 생성해 주어야함

- 마) -t

✓ -t 옵션이 설정되면 동기화가 필요한 대상들을 묶어 한번에 동기화 작업을 수행

✓ tar를 활용하여 묶음

✓ tar파일 전송 완료 후 묶음 해제하여 정확한 위치에 동기화

✓ -t 옵션 사용 시 로그기록 방식이 달라짐

✓ 로그기록 방식

- [동기화 완료시간] 명령어 및 명령어 옵션
  totalSize 전송된 tar 파일의 크기
  파일이름
- 동기화 완료시간 : 요일 월 날짜 명령어 동기화 완료시간
- 명령어 및 명령어 옵션
- totalSize : 로그 출력양식용 텍스트
- 전송된 tar 파일의 크기 : 동기화가 필요한 대상들을 묶은 tar 파일의 크기
- 파일 이름 : 동기화한 파일의 이름(src 디렉토리 기준 상대경로)

- 바) -m

✓ -m 옵션이 설정되면 dst 디렉토리에 src 디렉토리에 없는 파일 및 디렉토리가 존재할 경우 dst 디렉토리에서 해당하는 파일 및 디렉토리를 삭제함

✓ src 디렉토리에 없는 파일 및 디렉토리가 삭제되면 “ssu_rsync_log” 로그파일에 로그를 남김

<참고> 과제 구현에 필요한 함수(쓰레드 관련 함수 외 나머지 함수는 필수 사용 아님)

1. pthread_create() : p.354

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr,
                   void *(*start_rtn)(void *), void *restrict arg);
                                                    리턴값: 성공 시 0, 실패 시 오류 번호
```

2. pthread_exit() : p.362

```
#include <pthread.h>
void pthread_exit(void *rval_ptr);
```

3. pthread_join(), pthread_detach() : p.365

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **rval_ptr);
int pthread_detach(pthread_t tid);
                                                    리턴값: 성공 시 0, 실패 시 오류 번호
```

4. pthread_mutex_init(), pthread_mutex_destroy() : p.374

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexatt_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
                                                    리턴값: 성공 시 0, 실패 시 오류 번호
```

5. pthread_cond_init(), pthread_cond_destroy() : p.378

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict condition, pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cond_t *condition);
                                                    리턴값: 성공 시 0, 실패 시 오류 번호
```

6. getopt() : 프로그램 실행 시 입력한 인자를 처리하는 라이브러리 함수

```
#include <unistd.h>
int getopt(int argc, char * const argv[], const char *optstring); //_POSIX_C_SOURCE

#include <getopt.h>
int getopt_long(int argc, char * const argv[], const char *optstring, const struct option
*longopts, int *longindex); //_GNU_SOURCE
```

7. system() : p.270

```
#include <stdlib.h>
int system(const char* string);
                                                    리턴값: 성공시 0이 아닌 값, 실패시 0
```

8. localtime() : p.522

```
#include <time.h>
struct tm *localtime(const time_t *calptr);
                                        리턴 값: 성공 시 분할된 시간을 가리키는 포인터, 에러 시 NULL
```

9. ctime() : p.524

```
#include <time.h>
char *ctime(const time_t *calptr);
                                        리턴 값: 성공 시 문자열 포인터, 에러 시 NULL
```

○ 보고서 제출 시 유의 사항
- 보고서 제출 마감은 제출일 자정까지 (1시간 지연 허용)
- 지연 제출 시 감점 : 1일 지연 시 마다 30% 감점, 3일 지연 후부터는 미제출 처리
- 압축 오류, 파일 누락 관련 감점 syllabus 참고

○ 구현 점수
가. ssu_crontab 30
나. ssu_crand 30
다. ssu_rsync 30
라. -r 3
마. -t 4
바. -m 3
사. gettimeofday()를 사용하여 프로그램의 수행 시간 측정 - 모든 과제 동일 (미구현시 -5)

○ 필수 구현 사항 : 가, 나, 다, 사