설계과제 3 개요 : Soongsil Monitoring

Linux System Programming, School of CSE, Soongsil University, Spring 2023

○ 개요

- 리눅스 시스템 상에서 사용자가 원하는 디렉토리를 디몬 프로세스로 모니터링하고, 로그로 관리하는 프로그램

○ 목표

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

○ 팀 구성

- 개인별 프로젝트

○ 보고서 제출 방법

- 설계과제는 "#P설계과제번호_학번.zip"(예. #P3_20190000_V1.zip) 형태로 압축하여 classroom.google.com에 제출해야 함
- 구현보고서인 "#P설계과제번호.hwp"에는 1. 과제개요(명세에서 주어진 개요를 그대로 쓰면 안됨. 자기가 구현한 내용 요약) 2. 기능(구현한 기능 요약), 3. 상세설계(함수 및 모듈 구성, 순서도, 구현한 함수 프로토타입 등), 4. 실행결과 (구현한 모든 기능 및 실행 결과 캡쳐)를 반드시 포함시켜야 함.
- 과제를 기한 내 새로 제출할 경우 기존 것은 삭제하지 않고 #P설계과제번호_학번_V1.0.zip 형태로 버전 이름만 붙이면 됨. 버전 이름은 대문자 V와 함께 integer를 1부터 incremental 증가시키면서 부여하면 됨. (예. V1, V2, V3, ...) 단, 처음 제출 시는 버전 번호를 붙이지 않아도 되며 두 번째부터 V1를 붙여 제출하면 됨. 단, #P설계과제번호_학번_V1.0.zip 압축파일 내 구현보고서에는 버전 이름을 붙이지 않아도 됨.
- 압축파일 내 "보고서" 디렉토리와 "소스코드" 디렉토리 2개 만들어 제출해야 함
- 제출한 압축 파일을 풀었을 때 해당 디렉토리에서 컴파일 및 실행이 되어야 함(특정한 디렉토리에서 실행해야 할 경우는 제외). 해당 디렉토리에서 컴파일이나 실행되지 않을 경우, 기본과제 및 설계과제 제출 방법(파일명, 디렉토리면, 컴파일 시에 포함되어야 할 파일 등)을 따르지 않으면 무조건 해당 과제 배점의 50% 감점
  - ✓ 설계과제의 기준 및 각 과제별 "필수기능요건"은 각 설계 과제 명세서에 별도 설명. 설계과제명세서와 강의계획서 상 배점 기준이 다를 경우 해당 설계과제명세서의 배점 기준이 우선 적용
  - ✓ 보고서 #P설계과제번호.hwp (15점) : 개요1?점, 기능 1점, 상세설계 10점, 실행 결과 3점
  - ✓ 소스코드 (85점) : 소스코드 주석 5점, 실행 여부 80점 (설계 요구에 따르지 않고 설계된 경우 소스코드 주석 및 실행 여부는 0점 부여. 설계 요구에 따라 설계된 경우 기능 미구현 부분을 설계명세서의 100점 기준에서 해당 기능 감점 후 이를 80점으로 환산)
- 기타 내용은 강의계획서 참고

○ 제출 기한

- 5월 31일(수) 오후 11시 59분 59초

○ 보고서 및 소스코드 배점

- 보고서는 다음과 같은 양식으로 작성(강의계획서 FAQ 참고)

1. 과제 개요 (1점) // 명세에 주어진 개요를 더 상세하게 작성
2. 구현 기능 (1점) // 함수 프로토타입 반드시 포함
3. 상세 설계 (10점) // 함수 기능별 흐름도(순서도) 반드시 포함
4. 실행결과 (3점) // 테스트 프로그램의 실행결과 캡쳐 및 분석

- 소스코드 및 실행 여부 (85점) // 주석 (5점), 실행 여부 (80점)

○ ssu_monitor 프로그램 기본 사항

- ssu_monitor 프로그램은 모니터링 대상 디렉토리를 지정하고, 지정한 디렉토리 내의 모든 정규 파일의 변경 상태를 모니터링
- ssu_monitor은 대상 디렉토리 하위의 모든 정규 파일을 모니터링하는 디몬 프로세스 생성
- 디몬 프로세스는 파일이 생성, 삭제, 수정된 경우 모니터링 대상 디렉토리 밑 "log.txt" 파일에 변경 사항 기록
- log.txt 파일의 구조는 아래와 같음

예. log.txt 형태

```
% pwd
/home/oslab/P3/testdir
% cat log.txt
[2023-05-03 10:00:00][modify][/home/oslab/P3/testdir/a.txt]
[2023-05-03 10:00:15][create][/home/oslab/P3/testdir/b.txt]
[2023-05-03 10:00:30][create][/home/oslab/P3/testdir/a/a.txt]
[2023-05-03 10:01:05][modify][/home/oslab/P3/testdir/c.txt]
% rm d.txt
% touch a/b/a.txt
% cat log.txt
[2023-05-03 10:00:00][modify][/home/oslab/P3/testdir/a.txt]
[2023-05-03 10:00:15][create][/home/oslab/P3/testdir/b.txt]
[2023-05-03 10:00:30][create][/home/oslab/P3/testdir/a/a.txt]
[2023-05-03 10:01:05][modify][/home/oslab/P3/testdir/c.txt]
[2023-05-03 10:04:01][remove][/home/oslab/P3/testdir/d.txt]
[2023-05-03 10:04:19][create][/home/oslab/P3/testdir/a/b/a.txt]
```

  - ✓ [생성, 삭제, 수정 시간] [수행 내용] [파일의 절대경로] 형태로 작성
  - ✓ 파일 변경 사항은 "log.txt" 파일의 끝에 추가
  - ✓ 파일 이름은 한글을 지원하지 않아도 됨
- 모니터링이 종료되면 종료 메시지 출력 후 종료

○ 설계 및 구현

1. ssu_monitor

1) Usage : ssu_monitor
- 실행 시 프롬프트 출력 : "학번〉" ex)20190000〉
- 프롬프트는 add, delete, tree, exit, help 명령어만 수행

2) 실행 결과
- "학번〉"이 표준 출력되고 내장명령어(add, delete, tree, help, exit) 입력 대기. 학번이 20190000일 경우 "20190000〉" 형태로 출력
- 프롬프트 상에서 엔터만 입력 시 프롬프트 재출력

예. ssu_monitor 실행결과

```
% ./ssu_monitor
20190000〉
```

3) 예외처리 (스스로 판단하여 추가 구현)
- 프롬프트 상에서 지정한 내장명령어 외 기타 명령어 입력 시 help 명령어 실행 후 프롬프트 재출력

2. 내장명령어 1. add

1) Usage : add 〈DIRPATH〉 [OPTION]

2) 인자 설명

- 첫 번째 인자 〈DIRPATH〉는 디렉토리의 상대경로와 절대경로 모두 입력 가능해야 함
- 두 번째 인자 [OPTION]은 '-t'만 있으며 생략 가능(-t 옵션은 아래 설명 확인)

3) 실행결과

- 디렉토리 경로〈DIRPATH〉를 입력받아 디몬 프로세스를 생성하여 〈DIRPATH〉의 모니터링 시작
- 디몬 프로세스는 〈DIRPATH〉 하위의 모든 정규 파일을 모니터링하며 전체 모니터링 완료 1초 후 재시작
- 현재 작업 디렉토리(pwd)의 "monitor_list.txt"에 〈DIRPATH〉의 절대경로와 생성된 디몬 프로세스의 pid 정보 저장
  - ✓ "monitor_list.txt"는 ssu_monitor가 종료된 이후에도 남아있으며, ssu_monitor의 내장명령어 "add와 delete"로만 관리
  - ✓ "monitor_list.txt"에 존재하는 디몬 프로세스를 외부에서 kill 등으로 종료하는 경우는 고려하지 않음

예. monitor_list.txt 형태

```
% cat monitor_list.txt
/home/oslab/P3/testdir 5230
/home/oslab/P3/testdir2 5539
/home/oslab/P3/testdir3 5545
```

예. add 명령어 실행 결과

```
20190000> add testdir
monitoring started (/home/oslab/P3/testdir)
20190000>
```

- '-t 〈TIME〉' 옵션 : 입력 시 디몬 프로세스는 〈TIME〉 간격을 두고 모니터링 재시작

4) 예외 처리 (스스로 판단하여 추가 구현)

- 첫 번째 인자 입력이 없거나 잘못되었을 때 Usage 출력 후 프롬프트 재출력
- 첫 번째 인자로 입력받은 경로가 존재하지 않거나 디렉토리가 아닐 때 Usage 출력 후 프롬프트 재출력
- 첫 번째 인자로 입력받은 경로가 이미 모니터링 중인 디렉토리의 경로와 같거나/포함하거나/포함될 때 에러 처리
- -t 옵션의 사용이 올바르지 않은 경우 Usage 출력 후 프롬프트 재출력

3. 내장명령어 2. delete

1) Usage : delete 〈DAEMON_PID〉

2) 인자 설명

- 첫 번째 인자 〈DAEMON_PID〉는 "monitor_list.txt"의 디몬 프로세스 중 하나

3) 실행 결과

- 〈DAEMON_PID〉에 SIGUSR1 시그널을 보내 디몬 프로세스를 종료함
- "monitor_list.txt"에서 〈DAEMON_PID〉에 해당하는 정보 삭제

예. delete 명령어 실행 결과

```
20190000> delete 5539
monitoring ended (/home/oslab/P3/testdir2)
20190000> exit
% cat monitor_list.txt
/home/oslab/P3/testdir 5230
/home/oslab/P3/testdir3 5545
```

4) 예외 처리 (스스로 판단하여 추가 구현)

- 첫 번째 인자 〈DAEMON_PID〉가 "monitor_list.txt"에 존재하지 않는 경우 에러 출력

4. 내장명령어 3. tree

1) Usage : tree 〈DIRPATH〉

2) 인자 설명

- 첫 번째 인자 〈DIRPATH〉는 "monitor_list.txt"의 모니터링 디렉토리 경로 중 하나

✓ 절대경로, 상대경로 모두 입력 가능

3) 실행 결과

- 〈DIRPATH〉 디렉토리 구조를 tree 형태로 출력 (꼭 아래 예제 형태를 따르지 않아도 됨)
- system(), exec 계열 함수를 사용한 경우 0점 처리

예. add 내장명령어의 -d 옵션

```
20190000> tree testdir
testdir
├── a
│   ├── a.txt
│   └── b
│       └── a.txt
├── a.txt
├── b.txt
└── c.txt
20190000>
```

4) 예외 처리(스스로 판단하여 추가 구현)

- 첫 번째 인자 〈DIRPATH〉가 "monitor_list.txt"에 존재하지 않는 경우 에러 출력

5. 내장명령어 4. help

1) Usage : help

2) 실행 결과(스스로 판단하여 프로그램 개요가 한눈에 보이게 구현)

6. 내장명령어 5. exit

1) Usage : exit

2) 실행 결과

- 프로그램 종료

○ 과제 구현에 필요한 함수 (필수 아님)

- 1. 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
```

- 2. scandir : 디렉토리에 존재하는 파일 및 디렉토리 전체 목록 조회하는 라이브러리 함수

```
#include <dirent.h>
int scandir(const char *dirp, struct dirent ***namelist, int (*filter)(const struct dirent *), int
(*compar)(const struct dirent **, const struct dirent **));

-1 : 오류가 발생, 상세한 오류 내용은 errno에 설정
0 이상 : 정상적으로 처리, namelist에 저장된 struct dirent *의 개수가 return
```

- 3. realpath : 상대경로를 절대경로로 변환하는 라이브러리 함수

```
#include <stdlib.h>
char *realpath(const char *path, char *resolved_path);

NULL : 오류가 발생, 상세한 오류 내용은 errno 전역변수에 설정
NULL이 아닌 경우 : resolved_path가 NULL이 아니면, resolved_path를 return,
resolved_path가 NULL이면, malloc(3)으로 할당하여 real path를 저장한 후에 return
```

○ make와 Makefile

- make : 프로젝트 관리 유틸리티
  - ✓ 파일에 대한 반복 명령어를 자동화하고 수정된 소스 파일만 체크하여 재컴파일 후 종속된 부분만 재링크함
  - ✓ Makefile(규칙을 기술한 파일)에 기술된 대로 컴파일 명령 또는 쉘 명령을 순차적으로 실행함
- Makefile의 구성
  - ✓ Macro(매크로) : 자주 사용되는 문자열 또는 변수 정의 (컴파일러, 링크 옵션, 플래그 등)
  - ✓ Target(타겟) : 생성할 파일
  - ✓ Dependency(종속 항목) : 타겟을 만들기 위해 필요한 파일의 목록
  - ✓ Command(명령) : 타겟을 만들기 위해 필요한 명령(shell)

```
Macro

Target : Dependency1 Dependency2 ...
<-Tab->Command 1
<-Tab->Command 2
<-Tab->...
```

- Makefile의 동작 순서
  - ✓ make 사용 시 타겟을 지정하지 않으면 제일 처음의 타겟을 수행
  - ✓ 타겟과 종속 항목들은 관습적으로 파일명을 명시
  - ✓ 명령 항목들이 충족되었을 때 타겟을 생성하기 위해 명령 (command 라인의 맨 위부터 순차적으로 수행)
  - ✓ 종속 항목의 마지막 수정 시간(st_mtime)을 비교 후 수행
- Makefile 작성 시 참고사항
  - ✓ 명령의 시작은 반드시 Tab으로 시작해야함
  - ✓ 한 줄 주석은 #, 여러 줄 주석은 ₩
  - ✓ 자동 매크로 : 현재 타겟의 이름이나 종속 파일을 표현하는 매크로

매크로	설명
$?	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서 사용 불가능)
$^	현재 타겟의 종속 항목 (확장자 규칙에서 사용 불가능)
$<	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서만 사용 가능)
$*	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서만 사용 가능)
$@	현재 타겟의 이름

- Makefile 작성 예시

(예 1). Makefile

```
test : test.o add.o sub.o
    gcc test.o add.o sub.o -o
test

test.o: test.c
    gcc -c test.c

add.o: add.c
    gcc -c add.c

sub.o: sub.c
    gcc -c sub.c

clean :
    rm test.o
    rm add.o
    rm sub.o
    rm test
```

(예 2). 매크로를 사용한 경우

```
OBJECTS = test.o add.o sub.o
TARGET = test
CC = gcc

$(TARGET) : $(OBJECTS)
    $(CC) -o $(TARGET) $(OBJECTS)

test.o: test.c
     $(CC) -c test.c

add.o: add.c
     $(CC) -c add.c

sub.o: sub.c
     $(CC) -c sub.c
```

(예 3). 자동 매크로를 사용한 경우

```
OBJECTS = test.o add.o sub.o
TARGET = test
CC = gcc

$(TARGET) : $(OBJECTS)
    $(CC) -o $@ $^

test.o: test.c
     $(CC) -c $^

add.o: add.c
     $(CC) -c $^

sub.o: sub.c
     $(CC) -c $^
```

- (예 4). Makefile 수행 예시

```
oslab@a-VirtualBox:~$ make
gcc -c test.c
gcc -c add.c
gcc -c sub.c
gcc test.o add.o sub.o -o test
oslab@a-VirtualBox:~$
```

```
oslab@a-VirtualBox:~$ make clean
rm test.o
rm add.o
rm sub.o
rm test
oslab@a-VirtualBox:~$
```

○ 보고서 제출 시 유의 사항

- 보고서 제출 마감은 제출일 11:59PM까지 (구글 서버시간)
- 지연 제출 시 감점 : 1일 지연 시 30% 감점, 2일 이후 미제출 처리
- 압축 오류, 파일 누락 관련 감점 syllabus 참고

- 필수구현 : 1. ssu_monitor, 2. 내장명령어 add, delete, help, exit
- 부분별 배점(100점 만점. 실행 여부 배점 80점으로 최종 환산)

1. ssu_monitor – 50점
   - 프롬프트 : 5점
   - 디몬 프로세스 : 45점
2. 내장명령어 1. add – 10점
3. 내장명령어 2. delete – 10점
4. 내장명령어 3. tree – 25점
5. 내장명령어 4. help – 3점
6. 내장명령어 5. exit – 2점