설계과제 개요 : SoongSil Shell #2

Linux System Programming by Jiman Hong (jiman@ssu.ac.kr),
Sping 2017, School of CSE, Soongsil University

1. 개요

쉘(Shell)은 유닉스/리눅스 컴퓨팅 환경에서 기본적이고 중요한 부분으로, 명령 줄(Command line, cmd line)이라고도 부르며 사용자와 커널 사이의 인터페이스를 담당한다. 사용자의 명령을 해석하여 커널기능을 이용할 수 있게 해주고 자체적으로 프로그래밍 기능을 제공하여 반복적으로 수행해야 하는 명령어를 처리할 수 있다. 또, 각 사용자들의 환경을 저장하여 사용자에 맞는 환경을 제공한다.

proc 파일시스템은 유닉스 계열 운영체제에서 프로세스들과 시스템 정보를 계층적 파일 구조의 형식으로 보여주는 특별한 파일시스템으로서, 커널 메모리를 간접적으로 접근하는 기존의 방식보다 더 편리하고 표준적인 방식으로 프로세스 데이터에 접근한다. 일반적으로 이것은 부팅할 때, /proc이라는 이름으로 마운트 된다. Solaris, BSD, 리눅스, AIX등의 여러 유닉스 계열의 운영체제에서 지원한다.

2. 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 향상시킨다. 또한, proc 파일시스템을 이해함으로써 프로세스에 관한 대략적인 정보를 습득하고, 이해할 수 있는 능력을 향상시킨다.

- 아래의 쉘 명령어 구현

명령어	내용
ssu_lsproc	리눅스 시스템 상에 마운트 되어있는 proc 파일시스템을 참조해서 사용자가 원하는 정보를 출력해주는 프로그램

3. 팀 구성

개인별 프로젝트

4. 개발환경

가. OS : **Ubuntu 16.04**

나. Tools : vi(m), gcc, gdb

5. 보고서 제출 방법

1) 제출할 파일

- 보고서와 소스파일을 함께 압축하여 제출
  - 보고서 파일 : 한글로 작성
  - 소스코드 : ssu_lsproc.c, Makefile이 존재해야 함.
  - 압축파일명 : [P2]_학번_v0.0.zip

2) 제출할 곳

- http://oslab.ssu.ac.kr 접속 [Courses] ⇒ [리눅스 시스템 프로그래밍] ⇒ [과제제출]
- 게시물 제목은 파일이름과 동일함
- 압축된 파일을 첨부하여 과제 제출

3) 제출 기한

- 5월 2일(화) 오후 11시 59분 59초까지 과제 게시판으로 제출

6. 보고서 양식

보고서는 다음과 같은 양식으로 작성

1. 과제 개요
2. 설계
3. 구현
   - 각 함수별 기능
4. 테스트 및 결과
   - 테스트 프로그램의 실행 결과를 분석
5. 소스코드(주석 포함)

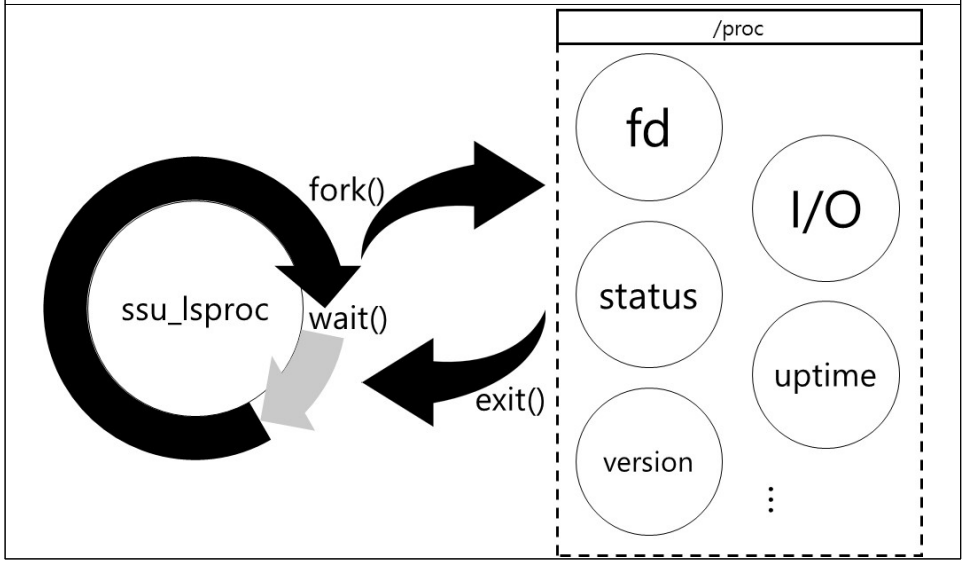
7. 설계 및 구현

1) ssu_lsproc

가) 프로그램 설명

- ssu_lsproc은 /proc에 마운트 된 proc 파일시스템의 정보를 읽어 들여서 사용자의 요청에 따라서 다른 출력을 보여주는 명령어
- ssu_lsproc 프로세스(부모 프로세스)는 옵션을 수행하는 자식 프로세스를 생성하고, 생성된 자식 프로세스들이 옵션을 수행
- 부모 프로세스가 fork()를 호출하면 자식 프로세스가 생성됨
- ssu_lsproc의 한 옵션이 실행되는 동안에 다른 옵션이 수행 될 수 없으며, 부모 프로세스는 wait()를 호출하며 잠시 수행을 중지함
- 자식 프로세스가 옵션에 대한 모든 수행을 마치면, exit()를 호출하여 프로세스를 종료함
- 부모 프로세스는 자식 프로세스가 종료함을 알게 되면 다시 수행을 재개하여 다른 옵션을 수행

[그림-1] 설계과제 2의 실행 흐름도



나) 프로그램 명세

- ssu_lsproc [OPTION]
  - 옵션들에 대해서 실행 예시 [0-1]과 같이 접근 권한을 검사
    - ✓ 파일이 존재 하지 않는 경우에는 "경로 doesn't exist" 출력
    - ✓ 읽기 권한이 없는 경우에는 "경로 can't be read."의 문자열을 출력하도록 구현

실행 예시 [0-1]

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 1 392
>: ssu_lsproc start. :<
([/proc/1/fd])
/proc/1/fd can't be read.
([/proc/392/fd])
/proc/392/fd doesn't exist.
>: ssu_lsproc terminated. :<
```

- ✓ 명령어 시작과 끝에는 실행 예시 [0-2]과 같은 문자열이 출력되도록 구현
  (이 문자열은 어떤 옵션의 영향이라도 받지 않음)

실행 예시 [0-2]

```
oslab@localhost:~/lsp$ ./ssu_lsproc
>: ssu_lsproc start. :<
>: ssu_lsproc terminated. :<
```

- [OPTION] : 실행할 프로그램의 옵션
  - **각 옵션은 종속 옵션을 제외하고는 독립적으로 수행함**
  - **-f [PID1] [PID2] ... [PID16]** : /proc/[PID]/fd에 존재하는 모든 파일 디스크립터의 내용을 출력

[그림-2] /proc/[PID]/fd 디렉터리의 내용

```
oslab@localhost:~/lsp$ ls /proc/18990/fd
0  1  2  3
oslab@localhost:~/lsp$ ls -al /proc/18990/fd
합계 0
dr-x------ 2 oslab oslab  0  3월 23 11:57 .
dr-xr-xr-x 9 oslab oslab  0  3월 23 11:57 ..
lrwx------ 1 oslab oslab 64  3월 23 11:57 0 -> /dev/pts/22
lrwx------ 1 oslab oslab 64  3월 23 11:57 1 -> /dev/pts/22
lrwx------ 1 oslab oslab 64  3월 23 11:57 2 -> /dev/pts/22
lr-x------ 1 oslab oslab 64  3월 23 11:57 3 -> /home/oslab/lsp/ssu_test.txt
```

- ✓ /proc/[PID]/fd에는 PID에 해당하는 프로세스가 개방하고 있는 파일들의 파일 디스크립터 정보가 존재
- ✓ 숫자인 파일 디스크립터로 나타난 symbolic link는 적절한 함수를 사용해서 숫자가 아닌 실제 파일 경로를 출력하도록 구현
- ✓ 사용자가 **PID를 주지 않았다**면 **실행한 프로세스 자신**에 대해서 이 작업을 수행
- ✓ 실행 예시 [1-1], [1-2], [1-3], [1-4]와 같은 결과로 출력되도록 구현

실행 예시 [1-1] - 프로세스 자신에 대해서 수행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f
>: ssu_lsproc start. :<
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /proc/14490/fd
>: ssu_lsproc terminated. :<
```

실행 예시 [1-2] - PID가 주어진 경우(loop_read의 PID는 18990)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 18990
>: ssu_lsproc start. :<
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /home/oslab/lsp/ssu_test.txt
>: ssu_lsproc terminated. :<
```


실행 예시 [1-3] - PID가 여럿 주어진 경우, 구별이 되도록 각 PID의 경로를 출력

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 18990 18874
>: ssu_lsproc start. :<
([/proc/18990/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /home/oslab/lsp/ssu_test.txt
([/proc/18874/fd])
File Descriptor number: 0, Opened File: /dev/null
File Descriptor number: 1, Opened File: socket:[1111976]
File Descriptor number: 2, Opened File: socket:[1111976]
File Descriptor number: 3, Opened File: socket:[1115397]
File Descriptor number: 4, Opened File: anon_inode:[eventfd]
File Descriptor number: 5, Opened File: anon_inode:[eventfd]
File Descriptor number: 6, Opened File: socket:[1115398]
File Descriptor number: 7, Opened File: anon_inode:[eventfd]
File Descriptor number: 8, Opened File: anon_inode:[eventfd]
File Descriptor number: 9, Opened File: socket:[1116288]
File Descriptor number: 10, Opened File: anon_inode:[eventfd]
File Descriptor number: 11, Opened File: anon_inode:inotify
>: ssu_lsproc terminated. :<
```

실행 예시 [1-4] - 접근 권한이 없는 PID에 접근 하는 경우 (1번 프로세스 : root 권한 필요)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 1
>: ssu_lsproc start. :<
/proc/1/fd can't be read.
>: ssu_lsproc terminated. :<
```

- **-t [PID1] [PID2] ... [PID16]** : /proc/[PID]/status의 파일의 일부 내용을 출력

[그림-3] /proc/[PID]/status 파일의 일부 내용

```
oslab@localhost:~/lsp$ cat /proc/21318/status
Name:   loop_read
State:  R (running)
Tgid:   21318
Ngid:   0
Pid:    21318
PPid:   18781
TracerPid:      0
Uid:    1002    1002    1002    1002
Gid:    1002    1002    1002    1002
FDSize: 256
Groups: 1002
```

- ✓ /proc/[PID]/status에는 프로세스의 이름, 상태, pid, ppid, uid등의 정보가 존재
- ✓ 파일의 내용 중 프로세스의 상태에 관련해서는 알파벳으로 나타나 있음. man proc을 참조해서 이 알파벳에 대한 내용을 확인 한 후, 알파벳 대신 문장으로 출력하도록 구현
  (단, man proc의 /proc/[PID]/status 파일의 state 설명 중 괄호안의 내용은 출력에 반영하지 않아도 됨)
- ✓ 사용자가 **PID를 주지 않았다**면 **실행한 프로세스 자신**에 대해서 수행
- ✓ 실행 예시 [2-1], [2-2], [2-3]에 알맞게 출력해야 함

실행 예시 [2-1] - 프로세스 자신에 대해서 수행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -t
>: ssu_lsproc start. :<
Name: ssu_lsproc
State: Running
Tgid: 14516
Ngid: 0
Pid: 14516
PPid: 14515
TracerPid: 0
Uid: 1002
Gid: 1002
>: ssu_lsproc terminated. :<
```

실행 예시 [2-2] - PID가 주어진 경우(loop_read의 PID는 21318이라고 가정)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -t 21318
>: ssu_lsproc start. :<
Name: loop_read
State: Running
Tgid: 21318
Ngid: 0
Pid: 21318
PPid: 18781
TracerPid: 0
Uid: 1002
Gid: 1002
>: ssu_lsproc terminated. :<
```

실행 예시 [2-3] - PID가 여럿 주어진 경우, 구별이 되도록 각각의 경로를 출력

```
oslab@localhost:~/lsp$ ./ssu_lsproc -t 21318 18874
>: ssu_lsproc start. :<
([/proc/21318/status])
Name: loop_read
State: Running
Tgid: 21318
Ngid: 0
Pid: 21318
PPid: 18781
TracerPid: 0
Uid: 1002
Gid: 1002
([/proc/18874/status])
Name: update-notifier
State: Sleeping in an interruptible wait
Tgid: 18874
Ngid: 0
Pid: 18874
PPid: 18298
TracerPid: 0
Uid: 1002
Gid: 1002
>: ssu_lsproc terminated. :<
```

- **-c [PID1] [PID2] ... [PID16]** : /proc/[PID]/cmdline의 파일 내용을 출력

[그림-4] /proc/[PID]/cmdline 파일의 내용

```
oslab@localhost:~/lsp$ cat /proc/21318/cmdline
./loop_readoslab@localhost:~/lsp$
```

✓ 사용자가 PID를 주지 않았다면 실행한 프로세스 자신에 대해서 수행

✓ **파일의 내용을 적절한 명령어를 이용해서 확인하고, 적절한 방법을 고안해서 읽어 들인 다음, 실행 예시 [3-1], [3-2], [3-3]과 같이 출력하도록 구현**

실행 예시 [3-1] - 프로세스 자신에 대해서 수행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -c
>: ssu_lsproc start. :<
argv[0] = ./ssu_lsproc
argv[1] = -c
>: ssu_lsproc terminated. :<
```

실행 예시 [3-2] - PID가 주어진 경우(loop_read의 PID는 21318이라고 가정)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -c 21318
>: ssu_lsproc start. :<
argv[0] = ./loop_read
>: ssu_lsproc terminated. :<
```

실행 예시 [3-3] - PID가 여럿 주어진 경우, 구별이 되도록 각각의 경로를 출력

```
oslab@localhost:~/lsp$ ./ssu_lsproc -c 21318 18874
>: ssu_lsproc start. :<
([/proc/21318/cmdline])
argv[0] = ./loop_read
([/proc/18874/cmdline])
argv[0] = update-notifier
>: ssu_lsproc terminated. :<
```


- **-n [PID1] [PID2] ... [PID16]** : /proc/[PID]/io의 파일 내용인 프로세스의 IO 관련 정보 출력

[그림-5] /proc/[PID]/io 파일의 내용

```
oslab@localhost:~$ cat /proc/21318/io
rchar: 1223
wchar: 0
syscr: 7
syscw: 0
read_bytes: 0
write_bytes: 0
cancelled_write_bytes: 0
```

✓ /proc/[PID]/io에는 한 프로세스에서 일어나는 read, write 및 관련 시스템 콜 호출 정보를 담고 있음

✓ 사용자가 **PID를 주지 않았다**면 **실행한 프로세스 자신**에 대해서 수행

✓ man proc의 /proc/[PID]/io 항목을 참조해서 기존의 내용을 매뉴얼 관련 내용으로 바꿔서 출력

실행 예시 [4-1] - 프로세스 자신에 대해서 수행한 결과

```
oslab@localhost:~/lsp$ ./ssu_lsproc -n
>: ssu_lsproc start. :<
Characters read : 0
Characters written : 0
Read syscalls : 0
Write syscalls : 0
Bytes read : 0
Bytes written : 0
Cancelled write bytes : 0
>: ssu_lsproc terminated. :<
```

실행 예시 [4-2] - PID가 주어진 경우의 수행 일부 내용(loop_read의 PID는 21318이라고 가정)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -n 21318
>: ssu_lsproc start. :<
Characters read : 1223
Characters written : 0
Read syscalls : 7
Write syscalls : 0
Bytes read : 0
Bytes written : 0
Cancelled write bytes : 0
>: ssu_lsproc terminated. :<
```

- **-m [PID1] [PID2] ... [PID16] [-k] [KEY1] [KEY2] ... [KEY16]** : /proc/[PID]/environ의 파일 내용인 환경 변수에 관한 내용을 출력

[그림-6] /proc/[PID]/environ 파일의 일부 내용

```
oslab@localhost:~/lsp$ cat /proc/21318/environ
XDG_VTNR=7XDG_SESSION_ID=c2CLUTTER_IM_MODULE=ximXDG_GREETER_DATA_DIR=/var/lib/lightdm-dat
a/oslabGPG_AGENT_INFO=/home/oslab/.gnupg/S.gpg-agent:0:1SHELL=/bin/bashTERM=xterm-256colo
rVTE_VERSION=4205QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1WINDOWID=73400330UPSTART_SESSION=unix:
abstract=/com/ubuntu/upstart-session/1002/17977GNOME_KEYRING_CONTROL=GTK_MODULES=gail:atk
-bridge:unity-gtk-moduleUSER=oslabLS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;3
5:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:
ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.
lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=
01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=
```

✓ /proc/[PID]/environ은 해당 프로세스의 환경 변수의 값을 가지고 있음

✓ 사용자가 **PID를 주지 않았다**면 **실행한 프로세스 자신**에 대해서 수행

✓ **getenv(), putenv(), setenv(), unsetenv()등의 환경변수 관련 함수 일체 사용 불가능**

✓ **적절한 커맨드 라인 명령어를 이용해서 /proc/[PID]/environ의 내용을 확인한 후, 실행 예시[5-1], [5-2]와 동일하게 출력되도록 구현**

실행 예시 [5-1] - 프로세스 자신에 대해서 수행한 일부 내용

```
oslab@localhost:~/lsp$ ./ssu_lsproc -m
>: ssu_lsproc start. :<
XDG_VTNR=7
XDG_SESSION_ID=c2
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/oslab
GPG_AGENT_INFO=/home/oslab/.gnupg/S.gpg-agent:0:1
SHELL=/bin/bash
TERM=xterm-256color
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=73400330
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1002/17977
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=oslab
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;3
:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;
```

실행 예시 [5-2] - PID가 주어진 경우의 수행 일부 내용(loop_read의 PID는 21318이라고 가정)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -m 21318
>: ssu_lsproc start. :<
...
LANGUAGE=ko
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=started starting
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=oslab
QT4_IM_MODULE=fcitx
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/s
napd/desktop
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-GtBpmYiVVN
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=Unity
UPSTART_JOB=unity-settings-daemon
XDG_RUNTIME_DIR=/run/user/1002
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=fcitx
```

- ✓ -k 옵션은 -m에 종속적인 옵션으로, 단독으로 쓰일 수 없는 옵션
- ✓ -k 옵션에 KEY가 주어지면 그 KEY에 해당하는 환경변수만 출력
- ✓ -k 옵션만 있고 KEY가 주어지지 않는다면 해당 프로세스의 모든 환경변수를 출력
- ✓ -k 옵션과 KEY가 하나라도 있는 경우에만 -r 옵션이 적용되도록 구현

실행 예시 [5-3] - '-k'옵션만 주고 ssu_lsproc을 실행한 결과

```
oslab@localhost:~/lsp$ ./ssu_lsproc -m 21318 -k
>: ssu_lsproc start. :<
XDG_VTNR=7
XDG_SESSION_ID=c2
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/oslab
GPG_AGENT_INFO=/home/oslab/.gnupg/S.gpg-agent:0:1
SHELL=/bin/bash
TERM=xterm-256color
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=73400330
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1002/17977
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=oslab
```

실행 예시 [5-4] - '-k'옵션과 여러 개의 KEY를 인자로 제공

```
oslab@localhost:~/lsp$ ./ssu_lsproc -m 21318 -k USER LANGUAGE LOGNAME INSTANCE
>: ssu_lsproc start. :<
USER=oslab
LANGUAGE=ko
LOGNAME=oslab
INSTANCE=Unity
>: ssu_lsproc terminated. :<
```

- **-w** : /proc/interrupts의 파일 내용인 인터럽트가 얼마나 발생하였는지, 어떤 주요 지표가 있는지를 출력

[그림-7] /proc/interrupts 파일의 일부 내용

```
oslab@localhost:~/lsp$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:         16          0          0          0   IO-APIC   2-edge      timer
  1:          1          1          0          0   IO-APIC   1-edge      i8042
  7:          7          0          0          0   IO-APIC   7-edge
  8:          1          0          0          0   IO-APIC   8-edge      rtc0
  9:          0          2          1          0   IO-APIC   9-fasteoi   acpi
 10:          0       1850       6174       2112   IO-APIC  10-edge      ite-cir
```

- ✓ CPU개수가 다른 환경에도 동작하도록 구현
- ✓ /proc/interrupts를 참조하여 CPU개수를 출력하도록 구현
- ✓ **(숫자): 로 시작하는 항목은 출력하지 않음**
- ✓ 실행 예시[6-1]의 결과와 같이 출력되도록 구현 하며, 출력 앞부분의 공백은 적절하게 설정
- ✓ 각 항목의 평균값은 소수점 아래 세자리까지 출력

실행 예시 [6-1] - CPU의 개수에 따른 실행결과 (좌 : 4개, 우 : 1개)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -w
>: ssu_lsproc start. :<
---Number of CPUs : 4---
      [Average] : [Description]
       2664.500 : <NMI> Non-maskable interrupts
   26239992.250 : <LOC> Local timer interrupts
          0.000 : <SPU> Spurious interrupts
       2664.500 : <PMI> Performance monitoring interrupts
          0.250 : <IWI> IRQ work interrupts
          0.750 : <RTR> APIC ICR read retries
     417300.500 : <RES> Rescheduling interrupts
       2637.750 : <CAL> Function call interrupts
        574.250 : <TLB> TLB shootdowns
          0.000 : <TRM> Thermal event interrupts
          0.000 : <THR> Threshold APIC interrupts
          0.000 : <DFR> Deferred Error APIC interrupts
          0.000 : <MCE> Machine check exceptions
        595.000 : <MCP> Machine check polls
          7.000 : <ERR>
          0.000 : <MIS>
          0.000 : <PIN> Posted-interrupt notification event
          0.000 : <PIW> Posted-interrupt wakeup event
>: ssu_lsproc terminated. :<
```

```
oslab@localhost:~$ ./ssu_lsproc -w
>: ssu_lsproc start. :<
---Number of CPUs : 1---
      [Average] : [Description]
          0.000 : <NMI> Non-maskable interrupts
      41468.000 : <LOC> Local timer interrupts
          0.000 : <SPU> Spurious interrupts
          0.000 : <PMI> Performance monitoring interrupts
          0.000 : <IWI> IRQ work interrupts
          0.000 : <RTR> APIC ICR read retries
          0.000 : <RES> Rescheduling interrupts
          0.000 : <CAL> Function call interrupts
          0.000 : <TLB> TLB shootdowns
          0.000 : <TRM> Thermal event interrupts
          0.000 : <THR> Threshold APIC interrupts
          0.000 : <DFR> Deferred Error APIC interrupts
          0.000 : <MCE> Machine check exceptions
          2.000 : <MCP> Machine check polls
          0.000 : <ERR>
          0.000 : <MIS>
          0.000 : <PIN> Posted-interrupt notification event
          0.000 : <PIW> Posted-interrupt wakeup event
>: ssu_lsproc terminated. :<
```

- **-e** : /proc/filesystem은 커널에서 지원하는 파일 시스템의 정보를 나타냄

[그림-8] /proc/filesystems 파일의 일부 내용

```
oslab@localhost:~/lsp$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
```

- ✓ nodev로 mark된 파일 시스템은 가상 파일시스템이나 네트워크 파일시스템을 나타낼 수 있음
- ✓ 본 옵션에서 nodev인 파일 시스템은 출력하지 않음
- ✓ 파일의 내용을 읽어서 실행 예시[7-1]과 동일하게 한 행에 5개의 파일 시스템이 들어가도록 구현

실행 예시 [7-1] - ssu_lsproc 명령어를 통한 커널에서 지원하는 파일 시스템 확인

```
oslab@localhost:~/lsp$ ./ssu_lsproc -e
>: ssu_lsproc start. :<
<<Supported Filesystems>>
ext3, ext2, ext4, squashfs, vfat
fuseblk
```

- **-l** : /proc/uptime은 시스템의 시작부터 수행되었던 프로세스들의 시간 총합, 유휴 프로세스들의 시간 총합을 가지고 있음
  - ✓ man proc을 참조해서 실행 예시 [8-1]과 동일하게 출력되도록 구현

실행 예시 [8-1] - ssu_lsproc 명령어를 통한 시간 관련 정보 확인

```
oslab@localhost:~/lsp$ ./ssu_lsproc -l
>: ssu_lsproc start. :<
Process worked time : 179447.49(sec)
Process idle time : 639829.39(sec)
```

- **-v** : /proc/version에 존재하는 리눅스 버전관련 정보를 출력
  - ✓ 파일의 내용을 읽어서 실행 예시 [9-1]의 출력 형식과 동일하게 출력되도록 구현(한 줄로 출력)

실행 예시 [9-1] - ssu_lsproc 명령어를 통한 시간 관련 정보 확인

```
oslab@localhost:~/lsp$ ./ssu_lsproc -v
>: ssu_lsproc start. :<
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4) )
 #1 SMP Tue Feb 21 13:59:09 KST 2017
```

- **-r** : [PID]나 [KEY]를 사용하는 옵션에 오름차순 정렬을 지원하도록 구현
  - ✓ **이 옵션에 대해서는 새로운 프로세스를 생성해서 수행하지 않아도 됨**

실행 예시 [10-1] - PID를 받는 옵션에 정렬을 적용

```
oslab@localhost:~/lsp$ ./ssu_lsproc -r -f 21318 18874
>: ssu_lsproc start. :<
([/proc/18874/fd])
File Descriptor number: 0, Opened File: /dev/null
File Descriptor number: 1, Opened File: socket:[1111976]
File Descriptor number: 2, Opened File: socket:[1111976]
File Descriptor number: 3, Opened File: socket:[1115397]
File Descriptor number: 4, Opened File: anon_inode:[eventfd]
File Descriptor number: 5, Opened File: anon_inode:[eventfd]
File Descriptor number: 6, Opened File: socket:[1115398]
File Descriptor number: 7, Opened File: anon_inode:[eventfd]
File Descriptor number: 8, Opened File: anon_inode:[eventfd]
File Descriptor number: 9, Opened File: socket:[1116288]
File Descriptor number: 10, Opened File: anon_inode:[eventfd]
File Descriptor number: 11, Opened File: anon_inode:inotify
([/proc/21318/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
```

실행 예시 [10-2] - KEY를 받는 옵션에 정렬을 적용

```
oslab@localhost:~/lsp$ ./ssu_lsproc -m 21318 -k USER LANGUAGE LOGNAME INSTANCE -r
>: ssu_lsproc start. :<
INSTANCE=Unity
LANGUAGE=ko
LOGNAME=oslab
USER=oslab
>: ssu_lsproc terminated. :<
```

- **-s [ATTRIBUTE_NAME]** : 현재 시스템 상에서 수행되고 있는 모든 사용자 프로세스를 대상으로 동일한 uid를 가진 프로세스에 대해서 ATTRIBUTE_NAME에 해당하는 내용을 출력
  - ✓ 이 프로그램(ssu_lsproc)을 실행한 사용자와 동일한 uid를 가지는 모든 프로세스에 대해서 출력
  - ✓ 현재 프로세스의 uid를 확인할 수 있는 함수를 사용하고, /proc/[PID]/ 디렉터리 하위에 있는 특정 파일을 참조해서 uid의 일치 여부를 확인 후, 출력하도록 구현
  - ✓ [ATTRIBUTE_NAME]에 사용할 수 있는 항목은 다음과 같음
    - ① FILEDES : 대상이 되는 모든 프로세스에 대해서 /proc/[pid]/fd의 파일 내용을 출력
    - ② CMDLINE : 대상이 되는 모든 프로세스에 대해서 /proc/[pid]/cmdline의 파일 내용을 출력
    - ③ IO : 대상이 되는 모든 프로세스에 대해서 /proc/[pid]/io의 파일 내용을 출력
    - ④ STAT : 대상이 되는 모든 프로세스에 대해서 /proc/[pid]/status의 파일 내용을 출력
    - ⑤ ENVIRON : 대상이 되는 모든 프로세스에 대해서 /proc/[pid]/environ의 파일 내용을 출력
  - ✓ 이 옵션에 대해서는 디렉터리 혹은 파일에 **접근 불가능 한 경우 출력하는 문자열을 출력하지 않아도 됨**
  - ✓ 이 옵션에서는 각 항목을 수행하기 위해서 자식 프로세스를 생성하지 않아도 됨
    (**단, -s 옵션을 수행하는 자식 프로세스는 생성 되어야 함**)
  - ✓ **출력 순서는 실행 예시 [11-2]를 참조하여, PID는 오름차순으로, 내부 항목은 'FILEDES - CMDLINE - STAT - ENVIRON' 순서대로 출력되도록 구현**

실행 예시 [11-1] - 하나의 항목에 대해서 출력한 내용의 일부

```
oslab@localhost:~/lsp$ ./ssu_lsproc -s CMDLINE
>: ssu_lsproc start. :<
## Attribute : CMDLINE, Target Process ID : 2621 ##
argv[0] = /lib/systemd/systemd
argv[1] = --user
## Attribute : CMDLINE, Target Process ID : 2622 ##
argv[0] = (sd-pam)
## Attribute : CMDLINE, Target Process ID : 2653 ##
argv[0] = sshd: oslab@pts/8
## Attribute : CMDLINE, Target Process ID : 2654 ##
argv[0] = -bash
## Attribute : CMDLINE, Target Process ID : 2707 ##
argv[0] = sshd: oslab@pts/9
## Attribute : CMDLINE, Target Process ID : 2708 ##
argv[0] = -bash
```

실행 예시 [11-2] - 여러 항목에 대해서 출력한 내용의 일부

```
oslab@localhost:~/lsp$ ./ssu_lsproc -s FILEDES CMDLINE
>: ssu_lsproc start. :<
## Attribute : FILEDES, Target Process ID : 2621 ##
File Descriptor number: 0, Opened File: /dev/null
File Descriptor number: 1, Opened File: socket:[51582]
File Descriptor number: 2, Opened File: socket:[51582]
File Descriptor number: 3, Opened File: socket:[53238]
File Descriptor number: 4, Opened File: anon_inode:[eventpoll]
File Descriptor number: 5, Opened File: anon_inode:[signalfd]
File Descriptor number: 6, Opened File: /sys/fs/cgroup/systemd/user.slice/user-1002.slice/user@1002.service
File Descriptor number: 7, Opened File: anon_inode:[timerfd]
File Descriptor number: 8, Opened File: socket:[51596]
File Descriptor number: 9, Opened File: anon_inode:[eventpoll]
File Descriptor number: 10, Opened File: /proc/2621/mountinfo
File Descriptor number: 11, Opened File: anon_inode:inotify
File Descriptor number: 12, Opened File: /proc/swaps
File Descriptor number: 13, Opened File: socket:[51601]
File Descriptor number: 14, Opened File: socket:[51603]
## Attribute : CMDLINE, Target Process ID : 2621 ##
argv[0] = /lib/systemd/systemd
argv[1] = --user
## Attribute : CMDLINE, Target Process ID : 2622 ##
argv[0] = (sd-pam)
```

- **-o [FILE_NAME]** : 인자로 받은 [FILE_NAME]에 출력의 결과를 담음. 이 옵션에 대해서는 새로운 프로세스를 생성해서 수행하지 않아도 됨
  - ✓ 출력 재지정을 수행 후, 실행 예시 [12-1]과 같은 성공적으로 재지정 되었음을 알리는 문장을 출력
  - ✓ **ssu_lsproc 명령어를 실행 할 때 출력되는 시작과 끝 문장은 정상적으로 출력 되어야 함**
  - ✓ **이 옵션을 구현할 때, write(), fwrite()와 같은 파일 입출력 함수의 사용 금지**

실행 예시 [12-1] - 다른 옵션 없이 실행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -o ssu_assign2.txt
>: ssu_lsproc start. :<
>: ssu_lsproc terminated. :<
oslab@localhost:~/lsp$ cat ssu_assign2.txt
!--Successfully Redirected : ssu_assign2.txt--!
```

실행 예시 [12-2] - 다른 옵션을 추가하고 실행 (loop_read의 PID는 21318이라고 가정)

```
oslab@localhost:~/lsp$ ./ssu_lsproc -o ssu_assign2.txt -f 21318
>: ssu_lsproc start. :<
>: ssu_lsproc terminated. :<
oslab@localhost:~/lsp$ cat ssu_assign2.txt
!--Successfully Redirected : ssu_assign2.txt--!
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /home/oslab/lsp/ssu_test.txt
```

다) 세부 기능 및 기능별 요구 조건

- 입력 요구조건
  - 옵션에 따른 추가적인 가변 인자(PID, KEY)들을 받아서 바람직한 수행을 보여야 함
  - 가변인자를 받는 옵션의 경우, 최대로 받을 수 있는 가변인자의 개수는 16개로 제한. 그 이상의 가변

인자를 받으면 실행 예시[13-1]과 같은 메시지를 출력하고 **수행에는 반영하지 않도록 구현**

- 가변인자의 개수는 옵션 다음부터 차례대로 셈을 시작하고, 순서대로 16개를 받도록 구현. **정렬 옵션이 같이 주어지더라도 차례대로 셈을 시작한 16개의 인자를 대상**으로만 정렬을 수행하도록 구현

실행 예시 [13-1] - 가변인자 최대 개수를 초과한 경우의 출력의 일부

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
>: ssu_lsproc start. :<
Maximum Number of Argument Exceeded. :: 17
Maximum Number of Argument Exceeded. :: 18
Maximum Number of Argument Exceeded. :: 19
([/proc/1/fd])
/proc/1/fd can't be read.
([/proc/2/fd])
/proc/2/fd can't be read.
([/proc/3/fd])
/proc/3/fd can't be read.
([/proc/4/fd])
/proc/4/fd doesn't exist.
([/proc/5/fd])
...
```

실행 예시 [13-2] - 정렬 옵션을 사용하며 가변인자 최대 개수를 초과한 경우의 출력의 일부

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 -r
>: ssu_lsproc start. :<
Maximum Number of Argument Exceeded. :: 3
Maximum Number of Argument Exceeded. :: 2
Maximum Number of Argument Exceeded. :: 1
([/proc/4/fd])
/proc/4/fd doesn't exist.
([/proc/5/fd])
/proc/5/fd can't be read.
([/proc/6/fd])
/proc/6/fd doesn't exist.
([/proc/7/fd])
/proc/7/fd can't be read.
([/proc/8/fd])
/proc/8/fd can't be read.
([/proc/9/fd])
```

- 출력 요구조건
  - 여러 옵션이 주어졌을 경우 옵션들의 출력 순서는 **'나) 프로그램 명세'의 옵션 순서대로** 출력

실행 예시 [14-1] - 여러 옵션을 주었을 때 명세의 출력 순서를 따른 결과

```
oslab@localhost:~/lsp$ ./ssu_lsproc -v -f
>: ssu_lsproc start. :<
([/proc/4664/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
([/proc/version])
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~1
6.04.4) ) #1 SMP Tue Feb 21 13:59:09 KST 2017
>: ssu_lsproc terminated. :<
```

- **각 옵션마다 새로운 프로세스를 생성하여 수행하도록 구현 (단, '-o' 옵션 및 '-r' 옵션은 제외)**
  - ✓ ssu_lsproc을 옵션과 같이 실행했을 때 옵션을 수행할 자식 프로세스를 생성하고, 부모 프로세스는 자식 프로세스를 기다림
  - ✓ 출력에서 **부모 프로세스와 자식 프로세스의 경쟁 조건**이 발생하지 않도록 구현
  - ✓ 옵션을 수행 완료한 자식 프로세스는 반드시 종료하도록 구현
- **<linux/proc_fs.h>에 존재하는 함수나 proc API 관련 함수의 사용은 일체 금지함**

라) 실행 예

실행 예시 [15-1] - 여러 옵션을 주고 실행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 21318 -v -l
>: ssu_lsproc start. :<
([/proc/21318/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /home/oslab/lsp/ssu_test.txt
([/proc/uptime])
Process worked time : 258315.72(sec)
Process idle time : 642712.32(sec)
([/proc/version])
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~1
6.04.4) ) #1 SMP Tue Feb 21 13:59:09 KST 2017
>: ssu_lsproc terminated. :<
```

실행 예시 [15-2] - 여러 옵션을 주고 출력 재지정 실행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f 21318 -v -l -o ssu_assign2.txt
>: ssu_lsproc start. :<
>: ssu_lsproc terminated. :<
oslab@localhost:~/lsp$ cat ssu_assign2.txt
!--Successfully Redirected : ssu_assign2.txt--!
([/proc/21318/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /dev/pts/22
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 3, Opened File: /home/oslab/lsp/ssu_test.txt
([/proc/uptime])
Process worked time : 258431.17(sec)
Process idle time : 642716.52(sec)
([/proc/version])
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~1
6.04.4) ) #1 SMP Tue Feb 21 13:59:09 KST 2017
```

실행 예시 [15-3] - 여러 옵션을 주고 실행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -s CMDLINE -v
>: ssu_lsproc start. :<
([/proc/version])
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4) ) #1 SMP Tue Feb
 21 13:59:09 KST 2017
([/proc/2621/cmdline])
## Attribute : CMDLINE, Target Process ID : 2621 ##
argv[0] = /lib/systemd/systemd
argv[1] = --user
([/proc/2622/cmdline])
## Attribute : CMDLINE, Target Process ID : 2622 ##
argv[0] = (sd-pam)
```

실행 예시 [15-4] - 여러 옵션을 주고 실행

```
oslab@localhost:~/lsp$ ./ssu_lsproc -f -s CMDLINE -v -l -o ssu_result.txt
>: ssu_lsproc start. :<
>: ssu_lsproc terminated. :<
oslab@localhost:~/lsp$ cat ssu_result.txt
!--Successfully Redirected : ssu_result.txt--!
([/proc/13955/fd])
File Descriptor number: 0, Opened File: /dev/pts/22
File Descriptor number: 1, Opened File: /home/oslab/lsp/ssu_result.txt
File Descriptor number: 2, Opened File: /dev/pts/22
File Descriptor number: 4, Opened File: /dev/pts/22
([/proc/uptime])
Process worked time : 599928.47(sec)
Process idle time : 656405.48(sec)
([/proc/version])
Linux version 4.4.3 (root@localhost) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4) ) #1 SMP Tue Feb
 21 13:59:09 KST 2017
([/proc/2621/cmdline])
## Attribute : CMDLINE, Target Process ID : 2621 ##
argv[0] = /lib/systemd/systemd
argv[1] = --user
([/proc/2622/cmdline])
## Attribute : CMDLINE, Target Process ID : 2622 ##
argv[0] = (sd-pam)
```

-o 옵션 / -f 옵션 / -l 옵션 / -v 옵션 / -s CMDLINE 옵션 / -s CMDLINE 옵션

※ 과제 구현에 필요한 함수들

1. fork(), getpid()

- 리눅스 시스템에서 새로운 프로세스를 생성할 때 사용하는 함수
- fork()에 의해 생성된 자식 프로세스는 PID를 부여받으며, 부모 프로세스의 PID를 PPID로 가짐
- 상속받는 항목은 PGID, SID 상속받으며, 파일 디스크립터 테이블 및 시그널을 상속받음
- 리눅스에서 fork()는 copy-on-write 매커니즘에 의해 구현되어 있음
- getpid()를 통해서 현재 실행 중인 프로세스의 PID를 얻을 수 있음

```
#include <unistd.h>
pid_t fork(void);
```

반환값	== 0	자식 프로세스에게 전달되는 값.
	> 0	부모 프로세스에게 전달되는 자식 프로세스의 PID 값.
	< 0	더 이상 프로세스를 만들 수 없을 경우 –1이 반환 되며 errno가 설정됨.

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
```

반환값	호출한 프로세스의 PID를 반환함.

fork_getpid_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    int pid;
    pid = fork();
    if (pid > 0) {
        printf("Parent : %d -> fork() -> : %d\n", getpid(), pid);
        sleep(1);
    }
    else if (pid == 0)
        printf("Child : %d\n", getpid());
    else if (pid == -1) {
        perror("fork error : ");
        exit(0);
    }

    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./fork_getpid_example
Parent : 10220 -> fork() -> : 10221
Child : 10221
```

2. wait계열 함수들

- 부모 프로세스가 자식 프로세스의 종료를 기다릴 경우 사용할 수 있는 함수

#include <sys/types.h> #include <sys/wait.h> pid_t wait(int *status);		
인자	*status	자식 프로세스가 종료된 상태를 저장할 주소
반환값	성공 시 종료된 자식 프로세스의 PID를 반환. 실패 시 –1이 반환 됨	
pid_t waitpid(pid_t pid, int *status, int options);		
인자	pid	기다릴 대상이 될 프로세스를 지정할 수 있음. man 2 wait 참조
	*status	자식 프로세스가 종료된 상태를 저장할 주소
	options	waitpid()가 반환 될 때 추가적인 행동을 수행하도록 설정 가능
반환값	성공 시 상태가 변화한 자식 프로세스의 PID를 반환. 실패 시 –1이 반환 됨	

fork_wait_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
    int pid, status;

    pid = fork();
    if (pid < 0) {
        fprintf(stderr,"FORK ERROR :");
        exit(0);
    }
    else if (pid == 0) {
        int i;
        for (i = 0; i < 5; i++) {
            printf("Child : %d\n", i);
            sleep(2);
        }
        exit(1);
    }
    else {
        printf("I wait Child(%d)\n", pid);
        wait(&status);
        printf("Child is exit (%d)\n", status);
    }
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./fork_wait_example
I wait Child(10739)
Child : 0
Child : 1
Child : 2
Child : 3
Child : 4
Child is exit (256)
```


3. access()

- 사용자를 기준으로 주어지는 경로에 해당하는 파일의 권한을 확인할 수 있는 함수

#include <unistd.h> int access(const char *pathname, int mode);		
인자	*pathname	접근 가능 여부를 확인할 path
	mode	읽기, 쓰기, 실행 가능 여부와 같은 접근성을 체크하기 위한 모드 설정 값
반환값	성공 시 0이 반환, 실패 시 –1이 반환 되며 errno가 설정 됨	

access_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

int main (int argc, char* argv[])
{
    char* path = argv[1];
    int rval;

    rval = access (path, F_OK);    // File existence checking.
    if (rval == 0)
        printf ("%s exists\n", path);
    else {
        if (errno == ENOENT)
            printf ("%s does not exist\n", path);
        else if (errno == EACCES)
            printf ("%s is not accessible\n", path);
        exit(0);
    }

    rval = access (path, R_OK);    // Read accessibility checking.
    if (rval == 0)
        printf ("%s is readable\n", path);
    else
        printf ("%s is not readable (access denied)\n", path);

    rval = access (path, W_OK);    // Write accessibility checking.
    if (rval == 0)
        printf ("%s is writable\n", path);
    else if (errno == EACCES)
        printf ("%s is not writable (access denied)\n", path);
    else if (errno == EROFS)
```

```
        printf ("%s is not writable (read-only filesystem)\n", path);
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./access_example /etc/passwd
/etc/passwd exists
/etc/passwd is readable
/etc/passwd is not writable (access denied)
```

4. getuid(), getpid()

- getuid() : 현재 프로세스를 수행하는 사용자의 uid를 반환하는 함수
- getpid() : 현재 프로세스의 PID를 반환하는 함수

```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void);
```

반환값	성공 시 호출하는 프로세스의 실제 UID를 반환

```
#include <unistd.h>
#include <sys/types.h>
pid_t getpid(void);
```

반환값	성공 시 호출하는 프로세스의 PID를 반환

getuid_pid_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (void)
{
    uid_t uid;

    if ((uid = getuid()) == -1)
        fprintf(stderr,"getuid() error\n");
    else
        printf("PID is %d, The real UID is : %d\n", getpid(), uid);
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./getuid_pid_example
PID is 9212, The real UID is : 1002
```


5. memset()

- 할당 받은 메모리 주소의 시작부터 특정 값으로 입력 받은 길이만큼 메모리를 초기화하는 함수

```
#include <string.h>
void *memset(void *buf, int val, size_t n);
```

인자	*buf	메모리의 값을 초기화할 포인터
	val	초기화 할 값
	n	초기화 길이
반환값	*buf에 대한 포인터를 반환, 실패 시 NULL을 반환	

memset_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char *ptr;

    ptr = (char *) malloc (10);
    ptr[9] = '\0';
    printf("%s\n", (char *)memset(ptr, 'f', 9));
    printf("%s\n", ptr);
    free(ptr);

    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./memset_example
fffffffff
fffffffff
```

※ 과제 수행 시 유용하게 사용할 수 있는 Tips

1. loop_read.c

- loop_read.c는 ssu_test.txt파일을 개방하고 무한 루프에 빠지는 프로그램
- background 프로세스로 실행시킬 때 출력되는 값인 PID를 이용해서 ssu_lsproc의 옵션 테스트에 사용

loop_read.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fd = open("ssu_test.txt",O_RDONLY);
    while (1);

    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./loop_read &
[1] 18917
```

2. od [PATHNAME]

- [PATHNAME]에 해당하는 파일의 내용을 8진수로 보여주는 명령어
- 추가적인 옵션 및 자세한 사항은 man od를 참조

실행결과

```
oslab@localhost:~$ cat ssu_test.txt
Soongsil University
Linux System Programming
Assignment #2
oslab@localhost:~$ od ssu_test.txt
0000000 067523 067157 071547 066151 052440 064556 062566 071562
0000020 072151 005171 064514 072556 020170 074523 072163 066545
0000040 050040 067562 071147 066541 064555 063556 040412 071563
0000060 063551 066556 067145 020164 031043 000012
0000073
oslab@localhost:~$ od -xc ssu_test.txt
0000000    6f53    6e6f    7367    6c69    5520    696e    6576    7372
          S   o   o   n   g   s   i   l       U   n   i   v   e   r   s
0000020    7469    0a79    694c    756e    2078    7953    7473    6d65
          i   t   y  \n   L   i   n   u   x       S   y   s   t   e   m
0000040    5020    6f72    7267    6d61    696d    676e    410a    7373
             P   r   o   g   r   a   m   m   i   n   g  \n   A   s   s
0000060    6769    6d6e    6e65    2074    3223    000a
          i   g   n   m   e   n   t       #   2  \n
0000073
```

3. 프로세스 관련 커맨드 라인 명령어

- ps
    - 현재 시스템에서 활성화 된 프로세스의 리스트이며 PID, TTY, 실행 시간등의 정보를 확인 할 수 있음
    - -ef 옵션과 같이 사용하면 현재 시스템에서 수행중인 모든 프로세스를 출력
    - 자세한 내용은 man ps를 참조
- kill
    - 프로세스에게 시그널을 전송하는 명령어
    - 인자로 아무것도 주지 않았다면 기본적으로 SIGTERM이 전송
    - 강제로 프로세스를 종료시키려면 -9(SIGKILL) 옵션을 사용

실행결과

```
oslab@localhost:~/lsp$ ps
  PID TTY          TIME CMD
 9745 pts/11   5-15:46:50 fd_read
10100 pts/11   00:00:00 bash
14190 pts/11   00:00:00 ps
oslab@localhost:~/lsp$ kill -9 9745
oslab@localhost:~/lsp$
[1]+  죽었음               ./fd_read
```


8. 설계 구성 요소

1) 목표 설정
    - 주어진 요구 조건을 이해하고 명확한 설계 목표를 설정한다.
    - 설계의 목표 및 요구조건을 문서화한다.
2) 분석
    - 목표 설정에서 명시한 요구 조건을 분석하고 해결을 위한 기본 전략을 수립한다.
    - 문제 해결을 위한 배경 지식을 이론 강의에서 듣고 개별적으로 학생들이 다양한 경로를 통해 자료를 찾아 분석한다.
3) 합성 (구조 설계)
    - 분석 결과를 토대로 적절한 구조를 도출한다.
    - 도출된 구조를 토대로 모듈을 작성한다.
4) 제작 (구현)
    - 각 모듈의 입출력을 명세한다.
    - 구조와 모듈을 C 언어로 구현하고 이를 컴파일하여 실행 파일을 만든다.
5) 시험 및 평가(성능 평가)
    - 모듈의 입출력 명세에 따라, 다양한 입력 데이타를 작성하고 모듈을 테스트한다.
    - 작성된 실행 파일이 안정적으로 수행되는지 다양하게 테스트하고 이를 평가한다.
6) 결과 도출
    - 안정적으로 수행되는 최종 결과(Output)를 캡쳐하여 최종 결과물은 보고서에 반영한다.

9. 평가 도구

1) 설계 보고서 평가
2) 실행 평가 (컴파일 및 실행)

10. 평가 준거(방법)

1) 평가 도구 1)에 대한 평가 준거
    - 소스 코드 분석 및 새로운 모듈의 설계가 제대로 이루어졌는가?
        - 설계 요구 사항을 제대로 분석하였는가?
        - 설계의 제약 조건을 제대로 반영하였는가?
        - 설계 방법이 적절한가?
    - 문서화
        - 소스 코드에 주석을 제대로 달았는가?
        - 설계 보고서가 잘 조직화되고 잘 쓰여졌는가?
2) 평가 도구 2)에 대한 평가 준거
    - 요구조건에 따라 올바르게 수행되는가?

11. 기타

1) 보고서 제출 마감은 제출일 자정까지

2) 지연 제출 시 감점

- 1일 지연 시 마다 30% 감점
- 3일 지연 후부터는 미제출 처리

3) 압축 오류, 파일 누락

- 50% 감점 처리 (추후 확인)

4) copy 발견시

- F 처리

12. 점수 배점

▶ ssu_lsproc 100
1. 각 옵션별로 새로운 프로세스 생성 6
2. 경쟁 조건이 일어나지 않은 결과 6
3. 수행을 마친 자식 프로세스 종료 5
4. 여러 옵션이 주어질 때 출력 되는 순서 4
5. -f 옵션 구현 9
6. -t 옵션 구현 8
7. -c 옵션 구현 7
8. -n 옵션 구현 7
9. -m 옵션 구현 5
10. -m 옵션에 종속되는 -k 옵션 구현 8
11. -w 옵션 구현 5
12. -e 옵션 구현 5
13. -l 옵션 구현 3
14. -v 옵션 구현 2
15. -r 옵션 구현 5
16. -s 옵션 구현 9
17. -o 옵션 구현 5
18. Makefile 사용 1

※ 필수적으로 구현해야 할 기능 : 1, 2, 3, 4, 5, 6, 9, 16, 17, 18