운영체제_02 과제

제출일	2019.05.01.	전공	컴퓨터공학과
과제명	나만의 쉘(Shell) 만들기	학번	2015211365
담당교수	정준호	이름	김태윤

1. shell이란?

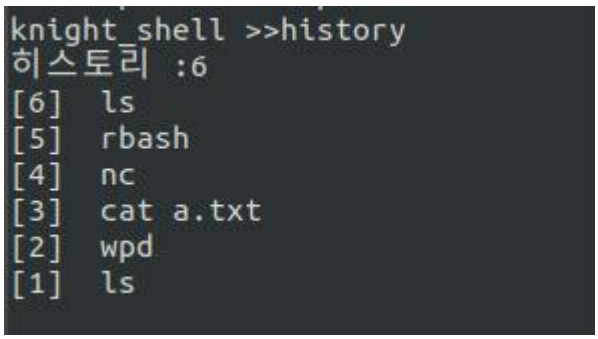
쉘(shell)은 운영체제의 커널과 사용자 사이를 이어주는 역할을 한다. 커널은 운영체제의 일부로 컴퓨터 메모리에 항상 작동되고 있는 하나의 프로그램이며 쉘은 사용자가 입력한 명령어를 운영체제가 알 수 있도록 지시해주는 역할을 한다.

따라서 이번 프로젝트에서 원하는 부분인 기본적인 쉘의 구성 및 history 기능 구현.
추가적인 기능으로는 파이프라인 기능을 구현하였습니다.

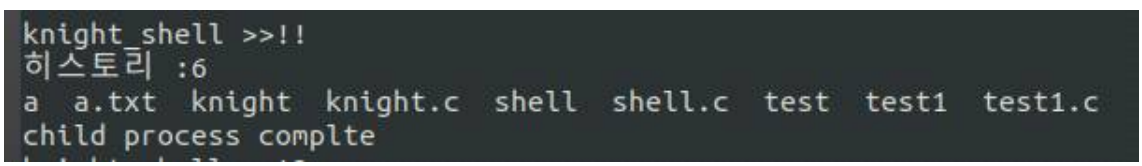
2. 구현한 knight shell의 구현 기능

1) execvp로 구현 가능한 함수들.

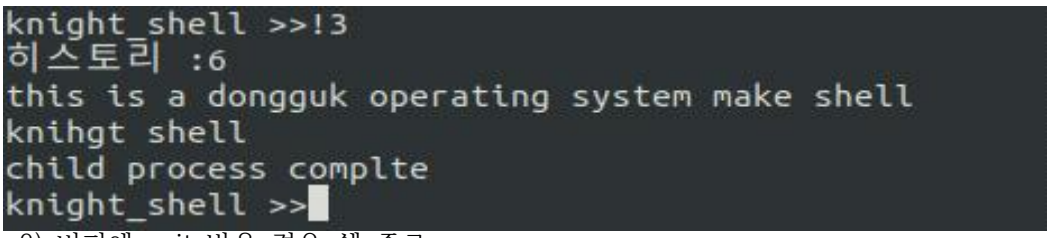
2)history 함수 구현.



(1) !!을 입력하였을 때

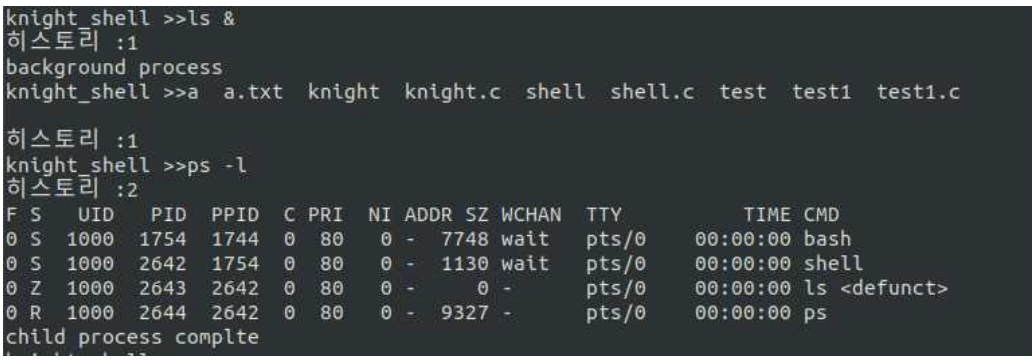


(2) !n(n=정수)을 입력하였을 때



3) 버퍼에 exit 받을 경우 쉘 종료.

4) 버퍼 마지막에 &가 입력되었을 때 백그라운드에서 프로세스 실행



문제점 : &를 마지막에 입력시키고 백그라운드에 실행 시켰을 경우 wait()가 없기 때문에 좀비 프로세스로 된 것을 볼 수 있다.

3. 구현한 knight shell의 구현 방식.
쉘이 시작이 되면 int main(void) 부분부터 시작이 되는데 args와 buffer의 사이즈를 정해주고 should_run을 1로 지정해줍니다. should_run은 while문의 반복을 시켜주는 중요한 역할이며 should_run이 0일 때 쉘이 종료되게 프로그래밍 하였습니다.

버퍼에 추가적으로 기본적으로 입력되는 함수는 아래 리스트와 같으며 다른 설정 없이 execvp 함수로 바로 실행 가능하다.

should_run이 0으로 변경되는 경우는 오로지 buffer에 exit를 적어 명령을 실행하였을 경우이며 이 명령을 실행 하고 나서는 쉘의 작동이 종료됩니다.

```
root@taeyoon:/bin# ls
bash            df              kmod        nisdomainname  rm                          tempfile
brltty          dir             less        ntfs-3g        rmdir                       touch
bsd-csh         dmesg           lessecho    ntfs-3g.probe  rnano                       true
bunzip2         dnsdomainname   lessfile    ntfscat        run-parts                   udevadm
busybox         domainname      lesskey     ntfscluster    sed                         ulockmgr_server
bzcat           dumpkeys        lesspipe    ntfscmp        setfacl                     umount
bzcmp           echo            ln          ntfsfallocate  setfont                     uname
bzdiff          ed              loadkeys    ntfsfix        setupcon                    uncompress
bzegrep         efibootdump     login       ntfsinfo       sh                          unicode_start
bzexe           efibootmgr      loginctl    ntfsls         sh.distrib                  vdir
bzfgrep         egrep           lowntfs-3g  ntfsmove       sleep                       wdctl
bzgrep          false           ls          ntfsrecover    ss                          which
bzip2           fgconsole       lsblk       ntfssecaudit   static-sh                   whiptail
bzip2recover    fgrep           lsmod       ntfstruncate   stty                        ypdomainname
bzless          findmnt         mkdir       ntfsusermap    su                          zcat
bzmore          fuser           mknod       ntfswipe       sync                        zcmp
cat             fusermount      mktemp      open           systemctl                   zdiff
chacl           getfacl         more        openvt         systemd                     zegrep
chgrp           grep            mount       pidof          systemd-ask-password        zfgrep
chmod           gunzip          mountpoint  ping           systemd-escape              zforce
chown           gzexe           mt          ping4          systemd-hwdb                zgrep
chvt            gzip            mt-gnu      ping6          systemd-inhibit             zless
cp              hciconfig       mv          plymouth       systemd-machine-id-setup    zmore
cpio            hostname        nano        ps             systemd-notify              znew
csh             ip              nc          pwd            systemd-sysusers
dash            journalctl      nc.openbsd  rbash          systemd-tmpfiles
date            kbd_mode        netcat      readlink       systemd-tty-ask-password-agent
dd              kill            networkctl  red            tar
root@taeyoon:/bin# exit
```

<history기능>
history기능은 index_of_history라는 변수를 선언하여 history[]라는 배열에 입력을 할 수 있도록 buffer를 파싱하여 history의 예외처리를 하였습니다. 처음 buffer[]에서 만약 !!이나 !n을 입력 하였을 경우 !!도 명령어의 히스토리에 들어갈 수 있으므로 historys()함수의 배열에 저장하지 않도록 예외처리를 하였으며 history명령도 예외처리를 하여 이전의 사용한 함수만 쉽게 볼 수 있도록 하였습니다.

```
^[[Ataeyoon@taeyoon:~/shell$ ./shell
knight_shell >>ls
히스토리 :1
a  a.txt  knight  knight.c  shell  shell.c  test  test1  test1.c
child process complte
knight_shell >>history
히스토리 :1
[1]  ls

knight_shell >>history
히스토리 :1
[1]  ls

knight_shell >>
```

```
knight_shell >>ls
히스토리 :3
a  a.txt  knight  knight.c  shell  shell.c  test  test1  test1.c
child process complte
knight_shell >>pwd
히스토리 :4
/home/taeyoon/shell
child process complte
knight_shell >>!3
히스토리 :4
a  a.txt  knight  knight.c  shell  shell.c  test  test1  test1.c
child process complte
knight_shell >>!!
히스토리 :4
/home/taeyoon/shell
child process complte
knight_shell >>
```

<백그라운드>

백그라운드 실행시 마지막에 &을 붙여주면 실행이 되나 wait0;가 구현이 되지 않아 좀비프로세스로 변하게 **되는 현상이 발생합니다.**

차후 개선사항으로 좀비프로세스를 발생 안하게 자식프로세스가 종료되면 좀비프로세스도 동일하게 종료 시켜주는 기능을 구현하면 해결이 될 것 이다.

```
knight_shell >>ls &
버퍼 : ls &
args : ls
히스토리 :4
knight_shell >>a  knight  knight.c  shell  shell.c  test  test.c  test1  test1.c
```

```
knight_shell >>ps -l
버퍼 : ps -l
args : ps
히스토리 :8
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2930  2920  0  80   0 -  7787 wait   pts/0    00:00:00 bash
0 S  1000  3151  2930  0  80   0 -  1130 wait   pts/0    00:00:00 shell
0 Z  1000  3169  3151  0  80   0 -     0 -      pts/0    00:00:00 ls <defunct>
0 R  1000  3175  3151  0  80   0 -  9327 -      pts/0    00:00:00 ps
```

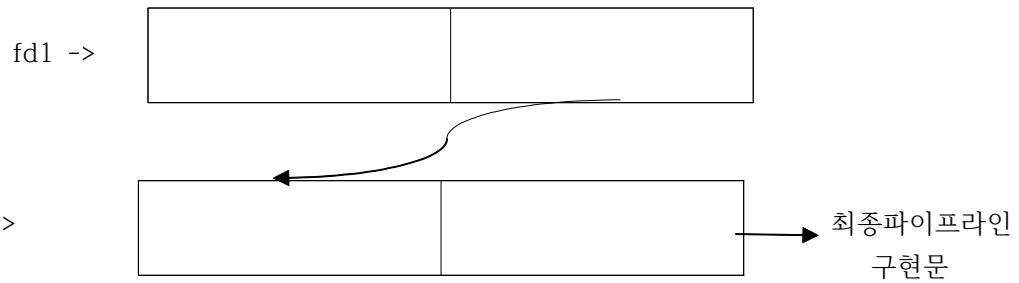
<파이프 문자 입력 처리.>

```
knight_shell >>ls -l | more
히스토리 :4
합계 112
drwxr-xr-x 2 taeyoon taeyoon  4096  4월 27 00:10 a
-rw-r--r-- 1 taeyoon taeyoon    59  4월 29 17:04 a.txt
-rwxr-xr-x 1 taeyoon taeyoon 13560  4월 11 11:33 knight
-rw-r--r-- 1 taeyoon taeyoon  6215  4월 26 22:19 knight.c
-rwxr-xr-x 1 taeyoon taeyoon 17936  4월 29 16:55 shell
-rw-r--r-- 1 taeyoon taeyoon  8313  4월 29 16:55 shell.c
-rwxr-xr-x 1 taeyoon taeyoon 18288  4월 27 13:20 test
-rwxr-xr-x 1 taeyoon taeyoon 18720  4월 26 21:38 test1
-rw-r--r-- 1 taeyoon taeyoon  5701  4월 26 21:38 test1.c

knight_shell >>history
히스토리 :4
[4]  ls -l | more
[3]  ls
[2]  ps -l
[1]  ls &

knight_shell >>!!
히스토리 :4
ls: '|'에 접근할 수 없습니다: 그런 파일이나 디렉터리가 없습니다
ls: 'more'에 접근할 수 없습니다: 그런 파일이나 디렉터리가 없습니다
child process complte
knight_shell >>
```

버퍼에서 입력 받은 명령이 '|'가 있을 경우 파이프라인 명령이므로 일단 명령을 따로 판별하게 구성하였습니다.
하지만 히스토리에서 명령 기록을 불러와서 사용할 경우 파이프라인을 분리하는 각 함수가 적용이 되지 않는 현상이 발생하였습니다.

1. 버퍼에서 입력을 받을 경우 '|'가 있을 경우를 pipe_exists라는 함수를 통해 판별함
2. 만약 pipe_exists에서 파이프라인이 있을 경우 pipe_exe 함수로 넘어감.
3. div_buffer 함수를 통해서 파이프라인 좌 우측의 명령을 분리시켜 따로 저장
4. dup2라는 함수를 사용하여 파이프 통신 시작 (0번이 stdin ,1번이 stdout, 2번이 stderr)
5. dup2(fd[WRITEONPIPE], STDOUT_FILENO);
   close(fd[READONPIPE]);
   close(fd[WRITEONPIPE]);

위와 같은 코드로 최종적으로 동작 방식은 아래 그림과 같습니다.



파이프라인의 단점은 2개이상의 파이프라인은 인식이 안되는 현상이 발생합니다.

< 느낀점 >

쉘이라는 것을 실제로 만들어 볼 생각은 하질 못했습니다. 그러나 과제가 막상 나오니 아무런 개념이 없기에 인터넷의 깃 허브나 여러 스택오버플로우를 통해 execvp에 대한 개념과 쉘의 동작원리에 대해 알게 되었습니다. cd 명령어가 path에 없다는 것을 보고 자주 썼던 명령어도 따로 설정한 명령어라는 것을 알게 되기도 했습니다. 쉘이 유저에게 더욱 편하게 사용하려면 여러 함수를 구현하여 최대한의 많은 명령어를 만들어야 되며 bash와 csh가 얼마나 사용자가 사용하기 편리한 쉘인지를 다시 한 번 느끼게 되었으며 차후 시간이 된다면 장기적으로 나만의 쉘을 완벽하게 구현하고 싶어졌습니다. 이런 과제가 할땐 너무 어려웠지만 학교의 선배들의 도움을 받게 되어 좀 더 좋은 코드를 짤수 있던 것도 저에게는 색다른 프로젝트였습니다.