1, 개요

리눅스 환경에서 fork와 exec계열 함수를 사용하여 shell 프로그램을 만들어보고 프로세서에 대한 이해를 높이고 동시에 리눅스 명령어를 알아보는 시간을 가집니다.

2. 프로그램 구조 설명

2.1 함수에 대한 설명

strcmp("str1”, “str2”) : 문자열을 비교하는 함수입니다. Str1이 str2 보다 사전순으로 앞서면 음수, 뒤에있으면 양수, 둘 다 같으면 0을 리턴합니다. (<https://chogahui05.blog.me/221308825045>)

. gets(문자열, 문자열의 크기) : 문자열을 입력받는 함수입니다.

fork() : 자식 프로세스를 생성합니다.

wait() : 자식 프로세스가 종료할 때 까지 기다립니다.

int execv(const char \*path, char \*const argv[]) : fork로 자식 프로세스를 만든 후 그 프로세스를 새로운 독립적인 프로세스로 만들어 주는 역할을 합니다. 따라서 그것은 우리가 명령프롬프트에 실행파일을 실행하는 것과 비슷한 이치입니다.([https://pmsdobak.tistory.com/entry/execl함수와-execv함수의-사용법에-대해서](https://pmsdobak.tistory.com/entry/execl%ED%95%A8%EC%88%98%EC%99%80-execv%ED%95%A8%EC%88%98%EC%9D%98-%EC%82%AC%EC%9A%A9%EB%B2%95%EC%97%90-%EB%8C%80%ED%95%B4%EC%84%9C) )

int pwd() : 쉘에서 리눅스명령어 pwd를 구현한것으로, 프로그램의 현재 폴더 위치를 알려줍니다..

int ls() : 쉘에서 리눅스명령어 ls를 구현한것으로, 기능을 합니다.

int ls\_a() : 쉘에서 리눅스명령어 ls -a를 구현한것으로, 숨겨진 파일이나 디렉토리도 출력해주는 기능을 합니다.

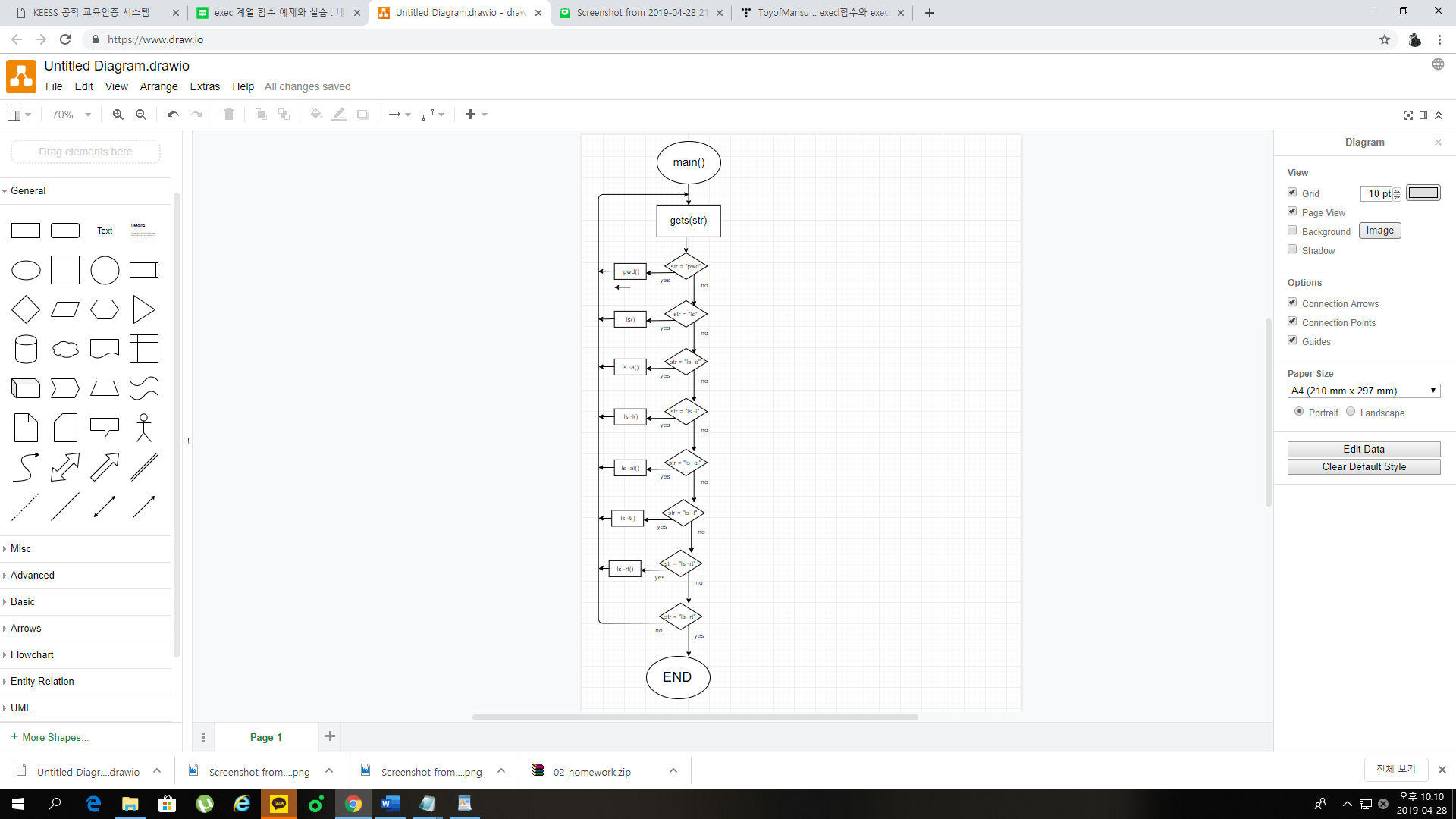
int ls\_l() : 쉘에서 리눅스명령어 ls -l을 구현한것으로, 파일이나 디렉토리 정보를 한 줄씩 하나씩 길게 보여줍니다. 출력되는 정보는 총 7개의 필드로 파일의 접근허용, 링크의 수 (파일) 또는 이동 가능한 디렉토리 개수, 소유자, 그룹, 파일크기, 최종 수정 날짜 및 시간, 파일 또는 디렉토리 명으로 구성됩니다..

int ls\_al() : 쉘에서 리눅스명령어 ls -al을 구현한것으로, 숨겨진 파일이나 디렉토리도 ls -l 합니다.

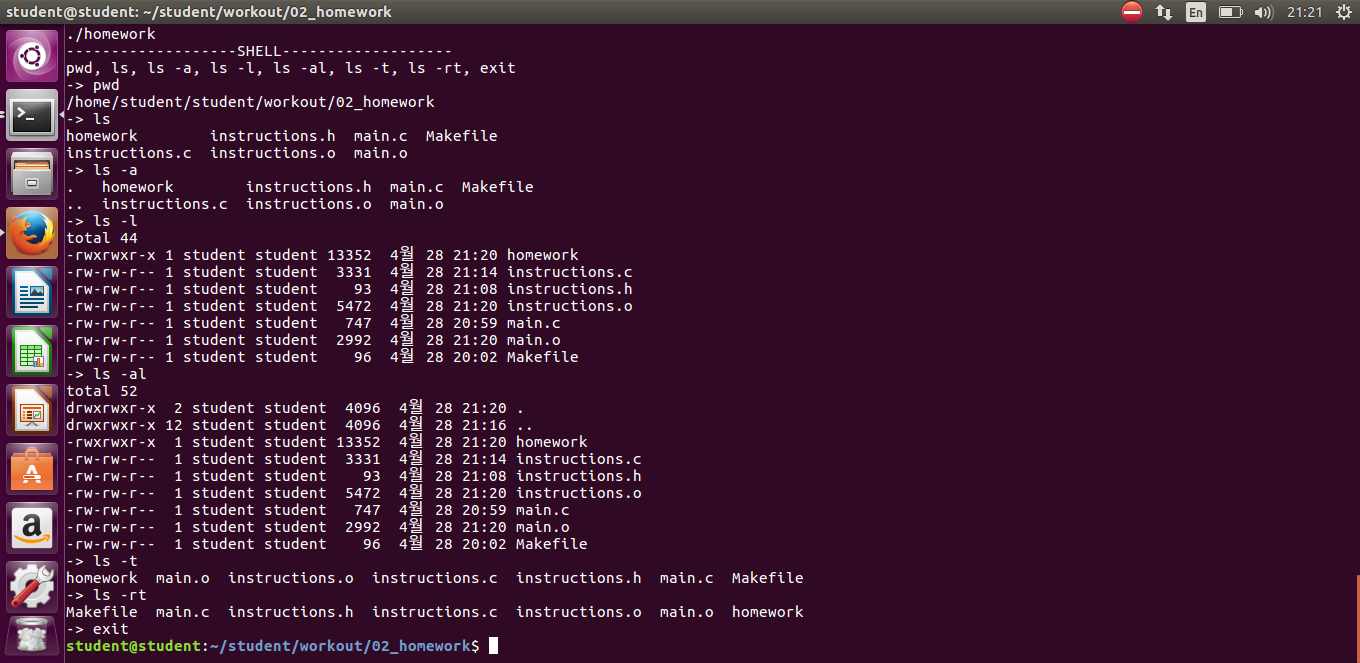
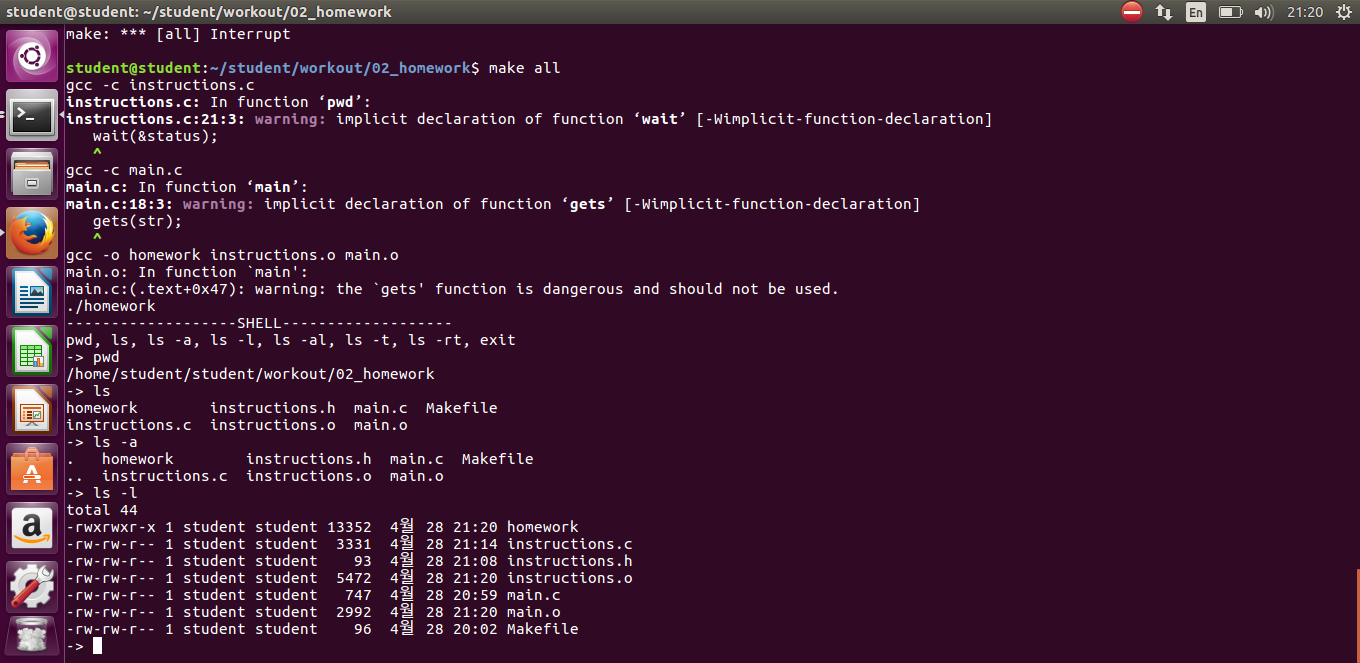
int ls\_t() : 쉘에서 리눅스명령어 ls -t를 구현한것으로, 출력되는 정보가 알파벳 순이 아니고 최근수정된 파일부터 시간순으로 출력됩니다.

int ls\_rt() : 쉘에서 리눅스명령어 ls -rt를 구현한것으로, 최신순이 아닌 오래된 순으로 출력합니다.

2.2 다이어그램



3. 실행 결과



4. 고찰

수업시간에 fork() 함수에 대해서 처음 배웠을 때, 부모 프로세스와 자식 프로세스가 종료되는 순서가 운영체제마다 다를 수 있다고 해서 당황스러웠는데, 실제 프로그래밍에서 wait() 함수를 통해 자식 프로세스가 먼저 종료될 때 까지 대기해 좀 더 확실하게 프로그래밍 할 수 있게 돼었고, 이번 fork() 함수를 사용하는 과제의 백미라고 느껴지는 exec 계열의 함수를 사용해보며 프로세스가 어떤 원리로 실행되는지 조금이나마 이해해보는 기회를 가졌습니다.. 인자가 여러 개 일 때 어떻게 수행해야하는지 알게 되었고, 아쉬운 부분이 있다면 지난번 과제인 strtok 함수를 사용해 좀 더 깔끔하게 코딩 할 수 있었을 것 같은데 생각처럼 잘 수행이 되지 않았다는 점입니다. 추상적으로 이해했던 쉘의 개념을 간단하게나마 직접 만들어보며 구체적으로 이해할 수 있었습니다.

5. 프로그램 소스 파일

|  |  |
| --- | --- |
| Instructions.c | 해석 |
| int pwd() {  int pid, status;  char\* argv[] = { "pwd", 0 };  pid = fork();  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/pwd", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls() {  int pid, status;  char\* argv[] = { "ls",0 };  pid = fork();  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls\_a() {  int pid, status;  char\* argv[] = { "ls","-a",0 };  pid = fork(); //프로세스 생성  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls\_l() {  int pid, status;  char\* argv[] = { "ls","-l",0 };  pid = fork(); //프로세스 생성  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls\_al() {  int pid, status;  char\* argv[] = { "ls", "-al",0 };    pid = fork();  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls\_t() {  int pid, status;  char\* argv[] = { "ls","-t",0 };  pid = fork();  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  }  int ls\_rt() {  int pid, status;  char\* argv[] = { "ls","-rt",0 };  pid = fork();  switch (pid) {  case -1:  printf("fork error \n");  exit(1);  case 0:  execv("/bin/ls", argv);  printf("exec error \n");  exit(1);  default:  wait(&status);  }  return 0;  } | pwd 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, 인자 끝 표시 배열  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다.  ls 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, 인자 끝 표시 배열  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다.  ls -a 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, “인자”, 인자 끝 표시 배열 <- 인자가 두 개 이상인 경우에 배열의 길이를 늘림.  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다.  ls -l 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, “인자”, 인자 끝 표시 배열 <- 인자가 두 개 이상인 경우에 배열의 길이를 늘림.  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다  ls -al 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, “인자”, 인자 끝 표시 배열 <- 인자가 두 개 이상인 경우에 배열의 길이를 늘림.  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다  ls -t 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, “인자”, 인자 끝 표시 배열 <- 인자가 두 개 이상인 경우에 배열의 길이를 늘림.  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다  ls -rt 기능을 구현  pid 와 status값을 받을 변수 선언.  “파일명”, “인자”, 인자 끝 표시 배열 <- 인자가 두 개 이상인 경우에 배열의 길이를 늘림.  프로세서 생성  pid를 통해 자식 프로세스 생성됐는지 확인  생성되지 않았을 경우  exit()함수로 분기  자식 프로세스 생성 된 경우  execv (“path”,인자);로 구성된 함수로 각각 경로(path)에서 실행할 인자와 execv 함수의 1번째 매개변수인 인자를 실행할 경로(path)이다.  생성되지 않았을경우 오류 메시지와 함께 exit() 함수로 분기  자식 프로세스가 종료 될 때 까지 기다린다 |
| main.c | 주석 |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <stdbool.h>  #include "instructions.h"  #define size 10  int main()  {  char str[size];  printf("-------------------SHELL-------------------\n");  printf("pwd, ls, ls -a, ls -l, ls -al, ls -t, ls -rt, exit\n");    while (true)  {  printf("-> ");  gets(str);    if (!strcmp("pwd", str))  pwd();      if (!strcmp("ls", str))  ls();  if (!strcmp("ls -a", str))  ls\_a();    if (!strcmp("ls -l", str))  ls\_l();    if (!strcmp("ls -al",str))  ls\_al();    if (!strcmp("ls -t", str))  ls\_t();    if (!strcmp("ls -rt", str))  ls\_rt();    if (!strcmp("exit", str))  break;    }  return 0;  } | ;반복문을 통해 지속적으로 값을 받습니다  값을 strcmp 함수를 이용해 비교하고 pwd 와 같은 문자열을 받는다면 pwd 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls 와 같은 문자열을 받는다면 ls 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls -a 와 같은 문자열을 받는다면 ls -a 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls -l 와 같은 문자열을 받는다면 ls -l 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls -al 와 같은 문자열을 받는다면 ls -al 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls -t 와 같은 문자열을 받는다면 ls -t 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 ls -rt 와 같은 문자열을 받는다면 ls -rt 함수를 실행합니다.  값을 strcmp 함수를 이용해 비교하고 exit 와 같은 문자열을 받는다면 break를 사용해 while 반복문에서 벗어납니다. |

6. 자료 출처

int execv(const char \*path, char \*const argv[])[https://pmsdobak.tistory.com/entry/execl함수와-execv함수의-사용법에-대해서](https://pmsdobak.tistory.com/entry/execl%ED%95%A8%EC%88%98%EC%99%80-execv%ED%95%A8%EC%88%98%EC%9D%98-%EC%82%AC%EC%9A%A9%EB%B2%95%EC%97%90-%EB%8C%80%ED%95%B4%EC%84%9C) )

strcmp("str1”, “str2”) : 문자열을 비교하는 함수입니다. Str1이 str2 보다 사전순으로 앞서면 음수, 뒤에있으면 양수, 둘 다 같으면 0을 리턴합니다. (<https://chogahui05.blog.me/221308825045>)