

웹프로그래밍 응용 (깃 허브 및 명령어 정리)

<https://github.com/GUBBIB/2025SystemPrograming>

2021963057

장문용

1주차 실습과제

History for 2025SystemPrograming / 1Week0307 on `main`

Commits on Mar 11, 2025

Update README.md



 GUBBIB committed on Mar 11

2025SystemPrograming / 1Week0307 / 

Add file ▾

 GUBBIB Update README.md

7d87764 · 3 months ago  History

Name	Last commit message	Last commit date
 ..		
 README.md	Update README.md	3 months ago

README.md  

1주차 수업내용

Windows에 Ubuntu 설치 방법

1. WSL , WSL2 사용
2. VM(Virtual Machine) 사용
3. 디스크 파티션 분할 후 듀얼 부팅

2주차 실습과제

History for 2025SystemPrograming / 2Week0314 on `main`

2025SystemPrograming / 2Week0314 /

Add file ▾

...

Commits on Mar 16, 2025

Upload

GUBBIB committed on Mar 16

Update README.md

GUBBIB committed on Mar 16

GUBBIB Upload

ae3b58b · 3 months ago

History

Name

Last commit message

Last commit date



..



README.md

Upload

3 months ago

README.md



2주차 수업내용

Tree

우분투 에서 **폴더 구조** 를 **깔끔하게 파일트리** 형식으로 볼 수 있게 해주는 패키지이다.

`sudo apt install tree` 명령어로 설치가 가능하며 `tree` 명령어로 폴더 구조를 볼 수 있다.

mkdir 명령어

`mkdir` : 디렉토리를 만들 때 사용하는 명령어

3주차 실습과제

History for 2025SystemProgramming / 3Week0321 on main

Commits on Mar 27, 2025

Upload

GUBBIB committed on Mar 27

스크린 샷 두 개 내용은 아래와 같습니다

```
stud@DESKTOP-4LST220:~$ ls
total 8
drwxr-xr-x 2 stud stud 4096 Mar 21 10:42 .
drwxr-xr-x 5 stud stud 4096 Mar 21 10:43 ..
-rw-r--r-- 1 stud stud  0 Mar 21 10:42 hello.c
stud@DESKTOP-4LST220:~/0321$ ls
hello.c
stud@DESKTOP-4LST220:~/0321$ ll
total 8
drwxr-xr-x 2 stud stud 4096 Mar 21 10:42 .
drwxr-xr-x 5 stud stud 4096 Mar 21 10:43 ..
-rw-r--r-- 1 stud stud  0 Mar 21 10:42 hello.c
stud@DESKTOP-4LST220:~/0321$ cat > hello.c
#include<stdio.h>
int main()
{
    printf("Hello C World!!");
    return 0;
}
stud@DESKTOP-4LST220:~/0321$ cat hello.c
#include<stdio.h>
int main()
{
    printf("Hello C World!!");
    return 0;
}
stud@DESKTOP-4LST220:~/0321$ ll
total 12
drwxr-xr-x 2 stud stud 4096 Mar 21 10:42 .
drwxr-xr-x 5 stud stud 4096 Mar 21 10:43 ..
-rw-r--r-- 1 stud stud 72 Mar 21 10:45 hello.c
stud@DESKTOP-4LST220:~/0321$ ls
```

```
stud@DESKTOP-4LST220:~/0321$ ls
a.out hello.c
stud@DESKTOP-4LST220:~/0321$ ./a.out
JangNoonfang
2021963937
JangNoonfang 2021963937stud@DESKTOP-4LST220:~/0321$
```

2025SystemProgramming / 3Week0321 /

Add file

GUBBIB Upload

0c3a282 · 3 months ago

Name	Last commit message	Last commit
..		
0321	Upload	3 months
README.md	Upload	3 months
스크린샷 2025-03-21 104708.png	Upload	3 months
스크린샷 2025-03-21 113115.png	Upload	3 months

README.md

3주차 수업내용

Ubuntu

우분투 명령어 및 디렉토리 표기법

- . : 현재 디렉토리, 현재 위치를 의미한다.
- .. : 상위 디렉토리를 의미한다.
- / : root
- ~ : 현재사용자의 홈 디렉토리를 의미한다. /home/사용자이름

4주차 실습과제

History for 2025SystemPrograming / 4Week0328 on `main`

Commits on Apr 2, 2025

Upload

 GUBBIB committed on Apr 2

2025SystemPrograming / 4Week0328 / 




Add file ▾

...

 GUBBIB Upload

689c7ce · 2 months ago

 History

Name	Last commit message	Last commit date
 ..		
 README.md	Upload	2 months ago
 ex1.c	Upload	3 months ago

README.md

4주차 수업내용

1. 파일 복사 (cp)

파일을 복사할 때 `cp` 명령어를 사용한다.

```
$ cp 파일1 파일2
```



5주차 실습과제

History for 2025SystemPrograming / 5Week0404 on `main`

Commits on Apr 9, 2025

Upload

 GUBBIB committed on Apr 9

2025SystemPrograming / 5Week0404 / 



Add file ▾

...

 GUBBIB Upload

576ad40 · 2 months ago

 History

Name	Last commit message	Last commit date
 ..		
 README.md	Upload	2 months ago
 ex1.c	Upload	2 months ago

README.md

5주차 수업내용

Ubuntu에서 시작하는 Prompt 개념

1. Ubuntu에서의 Prompt란?

Ubuntu 같은 리눅스 시스템에서 **Prompt**는 사용자가 명령어를 입력할 수 있도록 표시해주는 문자열이다.

예시:

```
user@ubuntu:~$
```



6주차 실습과제

History for 2025SystemPrograming / 6Week0411 on `main`

Commits on Apr 16, 2025

Upload

GUBBIB committed on Apr 16

2025SystemPrograming / 6Week0411 /

Add file

...

GUBBIB Upload

417db78 · 2 months ago

History

Name	Last commit message	Last commit date
..		
0411	Upload	2 months ago
README.md	Upload	2 months ago

README.md

✎ ☰

6주차 수업내용

셸(Shell) 개념 정리

1. 셸이란?

- 셸(Shell): 사용자와 운영체제 사이의 창구 역할을 하는 소프트웨어
- 명령어 처리기(Command Processor): 사용자의 명령어를 입력받아 해석하고 실행

7주차 실습과제

README 파일을 수정했습니다

History for 2025SystemProgramming / 7Week0418 on main

Commit 9740b8b

Upload



GUBBIB committed last month

9740b8b



GUBBIB committed last month

Upload



main

Upload



GUBBIB committed on Apr 18

6d29129

Upload



GUBBIB committed on Apr 18

d1f030a

Upload



GUBBIB committed on Apr 18

d9c80ff

temProgramming / 7Week0418 /

Add file



IBBIB Upload

9740b8b - last month

History

Last commit message

Last commit date

18

Upload

2 months ago

ADME.md

Upload

last month

ME.md



3 files changed +543 -0 lines changed

> 10Week0509/README.md



> 7Week0418/README.md



> 9Week0502/README.md



닉스 시스템 프로그래밍 요약 (숙명여대 창병모)

프로세스

프로세스

실행 중인 프로그램을 **프로세스(process)**라고 함

- 각 프로세스는 고유한 PID를 가짐
- 부모 프로세스에 의해 생성

8주차 중간 시험

9주차 실습과제

README 파일을 수정했습니다

History for 2025SystemProgramming / 9Week0502 on

Commit 9740b8b

Commits on May 15, 2025

Upload

GUBBIB committed last month

9740b8b

Upload

main

Commits on May 2, 2025

Upload

GUBBIB committed on May 2

4401089

Upload

GUBBIB committed on May 2

c73aeed

3 files changed +543 -0 lines changed

> 10Week0509/README.md

> 7Week0418/README.md

> 9Week0502/README.md

2025SystemProgramming / 9Week0502 /

Add file ...

GUBBIB Upload

9740b8b · last month History

Name	Last commit message	Last commit date
..		
0502	Upload	last month
README.md	Upload	last month

README.md

9주차 수업내용

유닉스 시스템 프로그래밍 요약 (종합)

1. 시스템 프로그래밍 개요

- **커널(Kernel)**은 하드웨어 자원을 관리하여 주요 서비스를 제공:

10주차 실습과제

README 파일을 수정했습니다

History for 2025SystemPrograming / 10Week0509 on main

2025SystemPrograming / 10Week0509 /

Add file ...

Commits on May 15, 2025

Commit 9740b8b

Upload



GUBBIB committed last month

9740b8b



Commits on May 9, 2025

Upload



GUBBIB committed on May 9

e269cdf



Upload



GUBBIB committed on May 9

a64c4e5



GUBBIB committed last month

Upload



main

3 files changed +543 -0 lines changed

> 10Week0509/README.md

> 7Week0418/README.md

> 9Week0502/README.md

GUBBIB Upload

9740b8b - last month

History

Name	Last commit message	Last commit date
..		
0509	Upload	last month
README.md	Upload	last month

README.md



유닉스 시스템 프로그래밍 요약 (종합)

1. 시스템 프로그래밍 개요

- **커널(Kernel)**은 하드웨어 자원을 관리하며 주요 서비스를 제공:
 - 파일 관리
 - 프로세스 관리
 - 메모리 관리
 - 통신 관리
 - 주변장치 관리
- 시스템 호출(System Call): 커널에 서비스를 요청하는 프로그래밍 인터페이스

11주차 실습과제

History for 2025SystemPrograming / 11Week0516 on `main`

Commits on May 22, 2025

Readme.md Upload

 GUBBIB committed 3 weeks ago

2025SystemPrograming / 11Week0516 / 




Add file ▾

...

 GUBBIB Readme.md Upload

91e7071 · 3 weeks ago

 History

Name	Last commit message	Last commit date
 ..		
 0516	upload	last month
 README.md	Readme.md Upload	3 weeks ago

README.md

11주차 수업내용

Unix 시스템 프로그래밍 정리 (chap6 ~ chap9)

★ 리눅스 파일 시스템 구조

리눅스의 ext4 파일 시스템은 다음과 같은 네 가지 주요 영역으로 구성되어 있다:

12주차 실습과제

History for 2025SystemPrograming / 12Week0523 on `main`

Commits on May 29, 2025

12주차 수업 내용 업로드

 GUBBIB committed 2 weeks ago

2025SystemPrograming / 12Week0523 / 

Add file ▾

...

 GUBBIB 12주차 수업 내용 업로드

ca675b6 · 2 weeks ago

 History

Name	Last commit message	Last commit date
..		
0523	Upload	3 weeks ago
README.md	12주차 수업 내용 업로드	2 weeks ago

README.md

12주차 수업내용

9장 프로세스 제어 (Process Control)

9.1 프로세스 생성

fork() 시스템 호출

- 부모 프로세스가 자신을 복제하여 자식 프로세스를 생성함.
- 복제는 코드, 데이터, 힙, 스택 모두 동일한 상태로 이루어짐.

13주차 실습과제

2025SystemPrograming / 13Week0530 / 

Add file ▾

...

 GUBBIB Upload

db0a4ba · last week  History

Name

Last commit message

Last commit date



..



README.md

Upload

last week

README.md

History for 2025SystemPrograming / 13Week0530 on **main** 

13주차 수업내용

발표 및 조기 종강

 Commits on Jun 5, 2025

Upload

 GUBBIB committed last week

깃허브 점수 : 15

- 기간에 맞춰서 잘 제출했습니다.
- 빼먹은 과제가 없습니다.

우분투 명령어

1. pwd

명령어

- pwd: 현재 경로를 추려하는 명령어

```
#include <stdio.h>
#include <unistd.h> // getcwd 함수
#include <limits.h> // PATH_MAX 상수

int main() {
    char cwd[PATH_MAX]; // 현재 작업 디렉토리를 저장할 버퍼

    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("%s\n", cwd); // 현재 디렉토리 경로 출력
    } else {
        perror("getcwd 오류"); // 오류 메시지 출력
        return 1;
    }

    return 0;
}
```

구현 설명

- `getcwd(char *buf, size_t size)` 함수를 통해서 현재 위치를 저장할 수 있다.
- `PATH_MAX`는 `limits.h`에 정의된 상수이다. ※ 크기는 4096

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi pwd_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o pwd_c pwd_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./pwd_c
/home/ubuntu/c_File
ubuntu@ip-172-31-41-56:~/c_File$
```

2. echo

명령어

- echo: 문자열 출력

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    // 인자가 1개 이상 있으면 (argv[0]은 실행 파일 이름)
    for (int i = 1; i < argc; i++) {
        printf("%s", argv[i]);
        if (i < argc - 1) {
            printf(" "); // 단어 사이 공백
        }
    }
    printf("\n"); // 마지막에 줄바꿈
    return 0;
}
```

설명

- 명령줄 인자(argc, argv) 를 활용하여 구현했습니다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi echo.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o echo_c echo.c
ubuntu@ip-172-31-41-56:~/c_File$ ./echo_c c로 만든 구분투 명령어
c로 만든 구분투 명령어
```

3. clear

명령어

- clear: 화면 지우기

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

```
int main() {  
    // ANSI 이스케이프 시퀀스로 화면 지우고, 커서 맨 위로 이동  
    printf("\033[2J"); // 화면 전체 지우기  
    printf("\033[H"); // 커서를 좌측 상단으로 이동  
    fflush(stdout); // 출력 즉시 반영  
  
    return 0;  
}
```

설명

- printf("\033[2J") 와 printf("\033[H") 를 이용해 직접 ANSI escape 코드 를 출력하여 화면을 지우고 커서를 초기 위치로 이동시켰습니다.

컴파일 및 실행화면

- 실행 전

```
ubuntu@ip-172-31-41-56:~/c_File$ vi clear.c.c  
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o clear_c clear_c  
clear_c clear_c.c  
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o clear_c clear_c.c  
ubuntu@ip-172-31-41-56:~/c_File$ ./clear_c
```

- 실행 후

```
ubuntu@ip-172-31-41-56:~/c_File$
```

4. whoami

명령어

- whoami: 현재 사용자 출력

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

int main() {
    struct passwd *pw = getpwuid(getuid());
    if (pw != NULL) {
        printf("%s\n", pw->pw_name);
    } else {
        perror("getpwuid 오류");
        return 1;
    }
    return 0;
}
```

설명

- `getuid()`: 현재 사용자의 UID(User ID)를 반환
- `getpwuid(uid)`: `getuid` 로 얻은 `uid` 를 통해서 사용자 정보를 담은 구조체(struct) 를 반환
- `passwd 구조체`: `getpwuid` 를 통해 얻은 구조체에서 `pw_name` 을 출력

```
#include <pwd.h>

struct passwd {
    char    *pw_name;        // 사용자 이름 ← 우리가 원하는 정보
    char    *pw_passwd;      // 암호 (대개 x 또는 *)
    uid_t   pw_uid;          // 사용자 ID
    gid_t   pw_gid;          // 그룹 ID
    char    *pw_gecos;       // 사용자 전체 이름
    char    *pw_dir;         // 홈 디렉토리
    char    *pw_shell;       // 로그인 쉘
};
```

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi whoami.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o whoami_c whoami.c
ubuntu@ip-172-31-41-56:~/c_File$ ./whoami_c
ubuntu
ubuntu@ip-172-31-41-56:~/c_File$
```

5. date

명령어

- date: 현재 시간 출력

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    const char *weekdays[] = { "일", "월", "화", "수", "목", "금", "토" };

    if (t != NULL) {
        printf("%04d년 %02d월 %02d일 (%s) %02d:%02d:%02d\n",
            t->tm_year + 1900,
            t->tm_mon + 1,
            t->tm_mday,
            weekdays[t->tm_wday], // 여기만 수정
            t->tm_hour,
            t->tm_min,
            t->tm_sec);
    } else {
        perror("localtime 오류");
        return 1;
    }

    return 0;
}
```



설명

- time(): 현재 시각을 1970년 1월 1일 기준으로 초 단위로 반환
- localtime(): time_t 형식의 시간을 struct tm 구조체로 변환하여 년, 월, 일, 시, 분, 초 등의 정보로 나눠줍니다.
- tm 구조체: localtime 을 통해서 얻은 구조체에서 시간, 날짜, 요일 등을 출력

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi date.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o date_c date.c
ubuntu@ip-172-31-41-56:~/c_File$ ./date_c
2025년 05월 23일 (금) 09:00:50
```

6.hostname -i

명령어

- hostname -i: 명령어 관련 파일(실행파일, man 파일 등) 위치 모두 확인

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ifaddrs.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>

int main() {
    struct ifaddrs *ifaaddr, *ifa;
    char ip[INET_ADDRSTRLEN];

    if (getifaddrs(&ifaaddr) == -1) {
        perror("getifaddrs 오류");
        return 1;
    }

    for (ifa = ifaaddr; ifa != NULL; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr == NULL)
            continue;

        if (ifa->ifa_addr->sa_family == AF_INET) { // IPv4
            void *addr = &((struct sockaddr_in *)ifa->ifa_addr->sin_addr);
            inet_ntop(AF_INET, addr, ip, sizeof(ip));

            // 루프백(127.0.0.1)은 제외
            if (strcmp(ip, "127.", 4) != 0) {
                printf("%s ", ip);
            }
        }
    }

    printf("\n");
    freeifaddrs(ifaaddr);
    return 0;
}
```

설명

- getifaddrs() 함수로 현재 네트워크 인터페이스들의 리스트를 얻는다.
- 각 인터페이스에 대해 IPv4 주소(AF_INET)인 경우 inet_ntop() 으로 문자열 IP 주소로 변환한다.
- 루프백 주소(127.0.0.1)는 제외하고 출력한다.
- 여러 개의 IP가 있을 수 있으므로 공백으로 구분해 모두 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi hostname_I.c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o hostname_I_c hostname_I.c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./hostname_I_c
172.31.41.56
ubuntu@ip-172-31-41-56:~/c_File$
```

7. uname

명령어

- `uname`: 커널 이름 출력

```
#include <stdio.h>
#include <sys/utsname.h>

int main() {
    struct utsname buffer;

    if (uname(&buffer) == 0) {
        printf("%s\n", buffer.sysname); // 커널 이름만 출력 (예: Linux)
    } else {
        perror("uname 오류");
        return 1;
    }

    return 0;
}
```



설명

- `sys/utsname.h` 에 정의된 `uname()` 함수로 커널 이름을 들고온다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi uname_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o uname_c uname_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./uname_c
Linux
ubuntu@ip-172-31-41-56:~/c_File$
```

8. id

명령어

- id: 사용자 정보 출력

```
#include <stdio.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

int main() {
    uid_t uid = getuid();           // 사용자 UID
    gid_t gid = getgid();           // 그룹 GID

    struct passwd *pw = getpwuid(uid); // 사용자 정보
    struct group *gr = getgrgid(gid);   // 그룹 정보

    if (pw == NULL || gr == NULL) {
        perror("사용자 또는 그룹 정보를 가져올 수 없습니다");
        return 1;
    }

    printf("uid=%d(%s) gid=%d(%s)\n", uid, pw->pw_name, gid, gr->gr_name);

    return 0;
}
```

설명

- getuid(): 사용자 UID(User ID) 반환
- getgid(): 그룹 ID 반환
- getpwuid(): UID 를 통해서 구조체 반환
- getgrgid(): GID 를 통해서 구조체 반환
- 구조체: UID , GID 구조체를 통해서 각 name을 출력

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi id_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o id_c id_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./id_c
uid=1000(ubuntu) gid=1000(ubuntu)
ubuntu@ip-172-31-41-56:~/c_File$
```


9. id -u

명령어

- `id -u`: 현재 사용자의 UID(User ID) 만 출력하는 명령어

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

```
int main() {
    uid_t uid = getuid(); // 현재 사용자 UID 가져오기
    printf("%d\n", uid);  // UID 출력
    return 0;
}
```

설명

- `unistd.h` 에 정의된 `getuid()` 함수를 사용하여 현재 사용자 UID(User ID)를 가져온다.
- `printf()` 로 UID를 출력한다.
- `id -u` 명령어와 같은 방식으로 현재 사용자의 UID만 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi id_u.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o id_u_c id_u.c
ubuntu@ip-172-31-41-56:~/c_File$ ./id_u_c
1000
ubuntu@ip-172-31-41-56:~/c_File$
```

10. id -g

명령어

- id -g: 현재 사용자의 GID(기본 그룹 ID) 만 출력하는 명령어

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

```
int main() {
    gid_t gid = getgid(); // 현재 사용자 GID 가져오기
    printf("%d\n", gid);  // GID 출력
    return 0;
}
```

설명

- unistd.h 에 정의된 getgid() 함수를 사용하여 현재 사용자의 GID(Group ID)를 가져온다.
- printf() 로 GID를 출력한다.
- id -g 명령어와 같은 방식으로 현재 사용자의 기본 그룹 ID만 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi id_g.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o id_g_c id_g.c
ubuntu@ip-172-31-41-56:~/c_File$ ./id_g_c
1000
ubuntu@ip-172-31-41-56:~/c_File$
```

11. printenv

명령어

- printenv: 환경변수 값 출력

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

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [환경변수 이름]\n", argv[0]);
        return 1;
    }
}
```

```
char *value = getenv(argv[1]);
```

```
if (value != NULL) {
    printf("%s\n", value);
}
```

```
// value가 NULL이면 아무것도 출력하지 않음 (기존 printenv와 동일)
```

```
return 0;
```

```
}
```

설명

- 명령줄 인자(argc, argv) 를 활용하여 구현했습니다.
- getenv(): argv[1] 에 들은 환경변수의 경로를 가져온다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi printenv.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o printenv_c printenv.c
ubuntu@ip-172-31-41-56:~/c_File$ ./printenv_c PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu@ip-172-31-41-56:~/c_File$ ./printenv_c HOME
/home/ubuntu
ubuntu@ip-172-31-41-56:~/c_File$ ./printenv_c USER
ubuntu
ubuntu@ip-172-31-41-56:~/c_File$
```

12. exit

명령어

- exit: 프로그램 종료 명령어

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

```
int main() {  
    exit(0); // 0은 정상 종료를 의미  
}
```

설명

- **exit()**을 이용해서 구현했습니다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi exit_c.c  
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o exit_c exit_c.c  
ubuntu@ip-172-31-41-56:~/c_File$ ./exit_c  
ubuntu@ip-172-31-41-56:~/c_File$
```

13. ls

명령어

- ls: 현재 디렉토리의 파일/디렉토리 이름 출력

```
#include <stdio.h>
#include <dirent.h>

int main() {
    DIR *dir;
    struct dirent *entry;

    dir = opendir("."); // 현재 디렉토리 열기
    if (dir == NULL) {
        perror("디렉토리 열기 실패");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        // 현재 디렉토리에서 항목 하나씩 출력
        if (entry->d_name[0] != '.') {
            printf("%s\n", entry->d_name);
        }
    }

    closedir(dir);
    return 0;
}
```

설명

- opendir(): dirent.h 라이브러리를 통해서 DIR 타입의 포인터 변수를 만들고 opendir(".") 을 통해서 현재 디렉토리의 정보를 dir 변수에 넣는다.
- if(entry->d_name[0] != '.'): 숨김 파일은 출력하지 못하게 하기 위해 구현
- entry = readdir(dir) != NULL: 현재 디렉토리의 항목을 하나씩 읽고 NULL이 아니면 출력, NULL이면 while문을 나가고 closedir() 을 통해서 디렉토리를 닫는다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ls.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ls_c ls.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ls_c
hostname_c
pwd_c.c
uname_c.c
whoami_c
echo_c
echo_c.c
pwd_c
uname_c
exit_c
clear_c
exit_c.c
ls_c.c
ls_c
printenv_c.c
hostname_c.c
date_c
id_c.c
clear_c.c
whoami_c.c
id_c
printenv_c
date_c.c
ubuntu@ip-172-31-41-56:~/c_File$
```

14. ls -a

명령어

- ls -a: 현재 디렉토리의 파일/디렉토리, 숨김처리 파일/디렉토리 이름 출력

```
#include <stdio.h>
#include <dirent.h>
```

```
int main() {
    DIR *dir;
    struct dirent *entry;

    dir = opendir("."); // 현재 디렉토리 열기
    if (dir == NULL) {
        perror("디렉토리 열기 실패");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        // 현재 디렉토리에서 항목 하나씩 출력
        printf("%s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```

설명

- opendir(): dirent.h 라이브러리를 통해서 DIR 타입의 포인터 변수를 만들고 opendir(".") 을 통해서 현재 디렉토리의 정보를 dir 변수에 넣는다.
- entry = readdir(dir) != NULL: 현재 디렉토리의 항목을 하나씩 읽고 NULL이 아니면 출력, NULL이면 while문을 나가고 closedir() 을 통해서 디렉토리를 닫는다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ls_a.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ls_a_c ls_a.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ls_a_c
nostname_c
ls_a_c
pwd_c.c
uname_c.c
whoami_c
echo_c
echo_c.c
pwd_c
..
uname_c
exit_c
clear_c
exit_c.c
ls_c.c
ls_c
ls_a_c.c
prntenv_c.c
nostname_c.c
date_c
id_c.c
clear_c.c
whoami_c.c
.
id_c
prntenv_c
date_c.c
ubuntu@ip-172-31-41-56:~/c_File$
```

15. ls -l

설명

- dirent.h 라이브러리로 현재 디렉토리 열기, dirent 구조체 선언, readdir() 함수를 통해서 파일/디렉토리 항목 하나씩 읽기 등 구현
- sys/stat.h 라이브러리에 정의된 파일 권한 관련 상수들로 파일의 권한을 구분 및 stat 구조체 사용
- stat, dirent 구조체들로 필요한 정보를 출력

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ls_l.c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ls_l.c ls_l.c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ls_l.c
-rwxrwxr-x 1 ubuntu ubuntu 16104 May 23 09:23 hostname_c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:31 ls_a_c
-rw-rw-r-- 1 ubuntu ubuntu 416 May 23 08:19 pwd_c.c
-rw-rw-r-- 1 ubuntu ubuntu 285 May 23 09:32 uname_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:40 ls_l.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:48 whoami_c
-rwxrwxr-x 1 ubuntu ubuntu 16008 May 23 08:30 echo_c
-rw-rw-r-- 1 ubuntu ubuntu 350 May 23 08:30 echo_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:19 pwd_c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:32 uname_c
-rwxrwxr-x 1 ubuntu ubuntu 15960 May 23 09:58 exit_c
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear_c
-rw-rw-r-- 1 ubuntu ubuntu 83 May 23 09:57 exit_c.c
-rw-rw-r-- 1 ubuntu ubuntu 476 May 23 10:27 ls_g.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:27 ls_c
-rw-rw-r-- 1 ubuntu ubuntu 430 May 23 10:31 ls_a_c.c
-rw-rw-r-- 1 ubuntu ubuntu 401 May 23 09:49 printenv_c.c
-rw-rw-r-- 1 ubuntu ubuntu 371 May 23 09:23 hostname_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16128 May 23 09:00 date_c
-rw-rw-r-- 1 ubuntu ubuntu 559 May 23 09:40 id_c.c
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear_c.c
-rw-rw-r-- 1 ubuntu ubuntu 262 May 23 08:48 whoami_c.c
-rw-rw-r-- 1 ubuntu ubuntu 1586 May 23 10:40 ls_l_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16184 May 23 09:40 id_c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:50 printenv_c
-rw-rw-r-- 1 ubuntu ubuntu 605 May 23 09:00 date_c.c
ubuntu@ip-172-31-41-56:~/c_File$
```

명령어

- ls -l 현재 디렉토리의 파일/디렉토리 상세 정보 출력

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <string.h>

void print_mode(mode_t mode) {
    printf("%04o", mode & 0777);
    printf((mode & S_IRUSR) ? "r" : "-");
    printf((mode & S_IWUSR) ? "w" : "-");
    printf((mode & S_IXUSR) ? "x" : "-");
    printf((mode & S_IRGRP) ? "r" : "-");
    printf((mode & S_IWGRP) ? "w" : "-");
    printf((mode & S_IXGRP) ? "x" : "-");
    printf((mode & S_IROTH) ? "r" : "-");
    printf((mode & S_IWOTH) ? "w" : "-");
    printf((mode & S_IXOTH) ? "x" : "-");
}

int main() {
    DIR *dir;
    struct dirent *entry;
    struct stat st;

    dir = opendir(".");
    if (dir == NULL) {
        perror("디렉토리 열기 실패");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        // 숨김 파일 제외
        if (entry->d_name[0] == '.')
            continue;

        if (stat(entry->d_name, &st) == -1) {
            perror("stat 실패");
            continue;
        }

        print_mode(st.st_mode);
        printf(" %2d", st.st_nlink);

        struct passwd *pw = getpwuid(st.st_uid);
        struct group *gr = getgrgid(st.st_gid);
        printf(" %s", pw ? pw->pw_name : "unknown");
        printf(" %s", gr ? gr->gr_name : "unknown");

        printf(" %5d", st.st_size);

        char timebuf[64];
        strftime(timebuf, sizeof(timebuf), "%b %d %H:%M", localtime(&st.st_mtime));
        printf(" %s", timebuf);

        printf(" %s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```

16. ls -al

설명

- `dirent.h` 라이브러리로 현재 디렉토리 열기, `dirent` 구조체 선언, `readdir()` 함수를 통해서 파일/디렉토리 항목 하나씩 읽기 등 구현
- `sys/stat.h` 라이브러리에 정의된 파일 권한 관련 상수들로 파일의 권한을 구분 및 `stat` 구조체 사용
- `stat`, `dirent` 구조체들로 필요한 정보들 출력

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c File$ vi ls_al.c
ubuntu@ip-172-31-41-56:~/c File$ gcc -o ls_al_c ls_al.c
ubuntu@ip-172-31-41-56:~/c File$ ./ls_al_c
-rwxrwxr-x 1 ubuntu ubuntu 16104 May 23 09:23 hostname_c
-rw-rw-r-- 1 ubuntu ubuntu 1493 May 23 10:55 ls_al_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:31 ls_a_c
-rw-rw-r-- 1 ubuntu ubuntu 416 May 23 08:19 pwd_c.c
-rw-rw-r-- 1 ubuntu ubuntu 205 May 23 09:32 uname_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:40 ls_l.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:48 whoami_c
-rwxrwxr-x 1 ubuntu ubuntu 16008 May 23 08:30 echo_c
-rw-rw-r-- 1 ubuntu ubuntu 350 May 23 08:30 echo_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:19 pwd_c
drwxr-xr-x 9 ubuntu ubuntu 4096 May 23 10:55 .
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:55 ls_al_c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:32 uname_c
-rwxrwxr-x 1 ubuntu ubuntu 15960 May 23 09:58 exit_c
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear_c
-rw-rw-r-- 1 ubuntu ubuntu 83 May 23 09:57 exit_c.c
-rw-rw-r-- 1 ubuntu ubuntu 476 May 23 10:27 ls_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:27 ls_c
-rw-rw-r-- 1 ubuntu ubuntu 430 May 23 10:31 ls_a_c.c
-rw-rw-r-- 1 ubuntu ubuntu 401 May 23 09:49 printenv_c.c
-rw-rw-r-- 1 ubuntu ubuntu 371 May 23 09:23 hostname_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16128 May 23 09:00 date_c
-rw-rw-r-- 1 ubuntu ubuntu 559 May 23 09:40 id_c.c
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear_c.c
-rw-rw-r-- 1 ubuntu ubuntu 262 May 23 08:48 whoami_c.c
drwxrwxr-x 2 ubuntu ubuntu 4096 May 23 10:55 .
-rw-rw-r-- 1 ubuntu ubuntu 1586 May 23 10:40 ls_l_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16184 May 23 09:40 id_c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:50 printenv_c
-rw-rw-r-- 1 ubuntu ubuntu 605 May 23 09:00 date_c.c
ubuntu@ip-172-31-41-56:~/c File$
```

명령어

- `ls -al`: 현재 디렉토리의 파일/디렉토리 및 숨김처리 파일/디렉토리 상세 정보 출력

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <string.h>

void print_mode(mode_t mode) {
    printf(S_ISDIR(mode) ? "d" : "-");
    printf((mode & S_IRUSR) ? "r" : "-");
    printf((mode & S_IWUSR) ? "w" : "-");
    printf((mode & S_IXUSR) ? "x" : "-");
    printf((mode & S_IRGRP) ? "r" : "-");
    printf((mode & S_IWGRP) ? "w" : "-");
    printf((mode & S_IXGRP) ? "x" : "-");
    printf((mode & S_IROTH) ? "r" : "-");
    printf((mode & S_IWOTH) ? "w" : "-");
    printf((mode & S_IXOTH) ? "x" : "-");
}

int main() {
    DIR *dir;
    struct dirent *entry;
    struct stat st;

    dir = opendir(".");
    if (dir == NULL) {
        perror("디렉토리 열기 실패");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &st) == -1) {
            perror("stat 실패");
            continue;
        }

        print_mode(st.st_mode);
        printf(" %ld", st.st_nlink);

        struct passwd *pw = getpwuid(st.st_uid);
        struct group *gr = getgrgid(st.st_gid);
        printf(" %s", pw ? pw->pw_name : "unknown");
        printf(" %s", gr ? gr->gr_name : "unknown");

        printf(" %ld", st.st_size);

        char timebuf[64];
        strtime(timebuf, sizeof(timebuf), "No %d %M", localtime(&st.st_mtime));
        printf(" %s", timebuf);

        printf(" %s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```


17. mkdir

설명

- 명령줄 인자(argc, argv) 를 활용하여 구현했습니다.
- sys/stat.h 헤더에 있는 mkdir() 함수를 통해서 디렉토리를 생성하며, 권한으로 **0755(사용자 rwx, 그룹 r-x, 기타 r-x)**를 준다.

컴파일 및 실행화면

명령어

- mkdir: 디렉토리 생성 명령어

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [디렉토리 이름]\n", argv[0]);
        return 1;
    }

    const char *dirname = argv[1];

    // 0755 권한: rwxr-xr-x
    if (mkdir(dirname, 0755) == -1) {
        perror("디렉토리 생성 실패");
        return 1;
    }

    printf("디렉토리 '%s' 생성 완료\n", dirname);
    return 0;
}
```

```
ubuntu@ip-172-31-41-56:~/c_File$ vi mkdir.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o mkdir_c mkdir.c
ubuntu@ip-172-31-41-56:~/c_File$ ./mkdir_c mkdir_with_c
디렉토리 'mkdir_with_c' 생성 완료
ubuntu@ip-172-31-41-56:~/c_File$ ll
total 320
drwxrwxr-x 3 ubuntu ubuntu 4096 May 23 10:58 ./
drwxr-xr-x 9 ubuntu ubuntu 4096 May 23 10:58 ../
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear_c*
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16128 May 23 09:00 date_c*
-rw-rw-r-- 1 ubuntu ubuntu 605 May 23 09:00 date_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16008 May 23 08:30 echo_c*
-rw-rw-r-- 1 ubuntu ubuntu 350 May 23 08:30 echo_c.c
-rwxrwxr-x 1 ubuntu ubuntu 15960 May 23 09:58 exit_c*
-rw-rw-r-- 1 ubuntu ubuntu 83 May 23 09:57 exit_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16104 May 23 09:23 hostname_c*
-rw-rw-r-- 1 ubuntu ubuntu 371 May 23 09:23 hostname_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16184 May 23 09:40 id_c*
-rw-rw-r-- 1 ubuntu ubuntu 559 May 23 09:40 id_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:31 ls_a_c*
-rw-rw-r-- 1 ubuntu ubuntu 430 May 23 10:31 ls_a.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:55 ls_al_c*
-rw-rw-r-- 1 ubuntu ubuntu 1493 May 23 10:55 ls_al_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:27 ls_c*
-rw-rw-r-- 1 ubuntu ubuntu 476 May 23 10:27 ls_c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:40 ls_l_c*
-rw-rw-r-- 1 ubuntu ubuntu 1586 May 23 10:40 ls_l.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16120 May 23 10:58 mkdir_c*
-rw-rw-r-- 1 ubuntu ubuntu 471 May 23 10:57 mkdir_c.c
drwxr-xr-x 2 ubuntu ubuntu 4096 May 23 10:58 mkdir_with_c/
```

18. mkdir -p

명령어

- `mkdir -p`: 디렉토리 생성 명령어, 상위 디렉토리가 없으면 자동 생성 옵션



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>

int mkdir_p(const char *path) {
    char tmp[1024];
    char *p = NULL;
    size_t len;

    snprintf(tmp, sizeof(tmp), "%s", path);
    len = strlen(tmp);

    if (tmp[len - 1] == '/')
        tmp[len - 1] = '\0'; // 끝에 '/' 있으면 제거

    for (p = tmp + 1; *p; p++) {
        if (*p == '/') {
            *p = '\0';
            mkdir(tmp, 0755); // 중간 경로 만들기 (실패해도 무시)
            *p = '/';
        }
    }

    return mkdir(tmp, 0755); // 마지막 경로 만들기
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [경로]\n", argv[0]);
        return 1;
    }

    if (mkdir_p(argv[1]) == -1 && errno != EEXIST) {
        perror("디렉토리 생성 실패");
        return 1;
    }

    printf("디렉토리 '%s' 생성 완료\n", argv[1]);
    return 0;
}
```

설명

- 명령줄 인자(`argc`, `argv`)를 활용하여 구현했습니다.
- `mkdir_p()` 함수는 `/`로 구분된 경로를 단계별로 생성하여, 상위 디렉토리가 없는 경우에도 자동으로 만들어준다.
- `sys/stat.h` 헤더에 있는 `mkdir()` 함수를 통해서 디렉토리를 생성하며, 권한으로 `**0755`(사용자 `rw`x, 그룹 `r-x`, 기타 `r-x`)*를 준다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi mkdir_p.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o mkdir_p_c mkdir_p.c
ubuntu@ip-172-31-41-56:~/c_File$ ./mkdir_p_c first/second/third
디렉토리 'first/second/third' 생성 완료
ubuntu@ip-172-31-41-56:~/c_File$ cd first/
ubuntu@ip-172-31-41-56:~/c_File/first$ cd second/
ubuntu@ip-172-31-41-56:~/c_File/first/second$ cd third/
ubuntu@ip-172-31-41-56:~/c_File/first/second/third$
```

19. rmdir

명령어

- rmdir: 디렉토리 삭제 명령어

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

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [디렉토리 이름]\n", argv[0]);
        return 1;
    }

    const char *dirname = argv[1];

    if (rmdir(dirname) == -1) {
        perror("디렉토리 삭제 실패");
        return 1;
    }

    printf("디렉토리 '%s' 삭제 완료\n", dirname);
    return 0;
}
```

설명

- 명령줄 인자(argc, argv) 를 활용하여 구현했습니다.
- unistd.h 헤더에 있는 rmdir() 함수를 통해서 디렉토리를 삭제한다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi rmdir.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o rmdir_c rmdir.c
ubuntu@ip-172-31-41-56:~/c_File$ ll
total 364
drwxrwxr-x 4 ubuntu ubuntu 4096 May 24 06:54 ./
drwxr-xr-x 9 ubuntu ubuntu 4096 May 24 06:54 ../
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear.c*
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16128 May 23 09:00 date.c*
-rw-rw-r-- 1 ubuntu ubuntu 605 May 23 09:00 date.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16008 May 23 08:30 echo.c*
-rw-rw-r-- 1 ubuntu ubuntu 350 May 23 08:30 echo.c.c
-rwxrwxr-x 1 ubuntu ubuntu 15960 May 23 09:58 exit.c*
-rw-rw-r-- 1 ubuntu ubuntu 83 May 23 09:57 exit.c.c
drwxr-xr-x 3 ubuntu ubuntu 4096 May 23 11:07 first/
-rwxrwxr-x 1 ubuntu ubuntu 16104 May 23 09:23 hostname.c*
-rw-rw-r-- 1 ubuntu ubuntu 371 May 23 09:23 hostname.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16184 May 23 09:40 id.c*
-rw-rw-r-- 1 ubuntu ubuntu 559 May 23 09:40 id.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:31 ls_a.c*
-rw-rw-r-- 1 ubuntu ubuntu 430 May 23 10:31 ls_a.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:55 ls_al.c*
-rw-rw-r-- 1 ubuntu ubuntu 1493 May 23 10:55 ls_al.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16136 May 23 10:27 ls.c*
-rw-rw-r-- 1 ubuntu ubuntu 476 May 23 10:27 ls.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16448 May 23 10:40 ls_l.c*
-rw-rw-r-- 1 ubuntu ubuntu 1586 May 23 10:40 ls_l.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16120 May 23 10:58 mkdir.c*
-rw-rw-r-- 1 ubuntu ubuntu 471 May 23 10:57 mkdir.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16344 May 23 11:06 mkdir_p.c*
-rw-rw-r-- 1 ubuntu ubuntu 977 May 23 11:06 mkdir_p.c.c
drwxr-xr-x 2 ubuntu ubuntu 4096 May 23 10:58 mkdir_with_c/
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:50 printenv.c*
-rw-rw-r-- 1 ubuntu ubuntu 401 May 23 09:49 printenv.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:19 pwd.c*
-rw-rw-r-- 1 ubuntu ubuntu 416 May 23 08:19 pwd.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16120 May 24 06:54 rmdir.c*
-rw-rw-r-- 1 ubuntu ubuntu 430 May 24 06:54 rmdir.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 09:32 uname.c*
-rw-rw-r-- 1 ubuntu ubuntu 285 May 23 09:32 uname.c.c
-rwxrwxr-x 1 ubuntu ubuntu 16096 May 23 08:48 whoami.c*
-rw-rw-r-- 1 ubuntu ubuntu 262 May 23 08:48 whoami.c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./rmdir_c mkdir_with_c/
디렉토리 'mkdir_with_c/' 삭제 완료
ubuntu@ip-172-31-41-56:~/c_File$
```

20. touch

명령어

- touch: 파일이 없으면 생성, 있으면 마지막 수정 시간 갱신 명령어

```
#include <stdio.h>
#include <fcntl.h>
#include <utime.h>
#include <unistd.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];
    int fd;

    // 파일 열기 (없으면 생성), O_WRONLY 안 써도 시간 갱신 가능
    fd = open(filename, O_CREAT | O_WRONLY, 0644);
    if (fd == -1) {
        perror("파일 열기 실패");
        return 1;
    }
    close(fd); // 파일만 열고 닫기

    // 시간 정보 갱신
    if (utime(filename, NULL) == -1) {
        perror("시간 갱신 실패");
        return 1;
    }

    return 0;
}
```



설명

- 명령줄 인자(argc, argv) 를 활용하여 구현했습니다.
- open() 에서 fcntl.h 에 정의된 **상수(O_CREAT, O_WRONLY)**들을 사용하여 파일이 존재하면 열고, 없으면 **0644권한(사용자 rw-, 그룹 r--, 기타 r--)**의 파일을 생성한 뒤, 즉시 닫는다.
- utime.h 에 정의된 utime() 함수로 파일의 접근시간과 수정시간을 변경한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi touch_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o touch_c touch_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./touch_c a
ubuntu@ip-172-31-41-56:~/c_File$ ll
total 380
drwxrwxr-x 3 ubuntu ubuntu 4096 May 24 07:02 ./
drwxr-xr-x 9 ubuntu ubuntu 4096 May 24 07:02 ../
-rw-r--r-- 1 ubuntu ubuntu 0 May 24 07:02 a
```

21. rm

명령어

- rm: 파일 삭제 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];

    if (unlink(filename) == -1) {
        perror("파일 삭제 실패");
        return 1;
    }

    printf("파일 '%s' 삭제 완료\n", filename);
    return 0;
}
```

설명

- unistd.h 에 정의된 unlink() 함수로 파일을 삭제한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi rm_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o rm_c rm_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./rm_c a
파일 'a' 삭제 완료
ubuntu@ip-172-31-41-56:~/c_File$ ll
total 400
drwxrwxr-x 3 ubuntu ubuntu 4096 May 24 07:15 ./
drwxr-xr-x 9 ubuntu ubuntu 4096 May 24 07:15 ../
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear_c*
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear_c.c
```

22. rm -f

명령어

- rm -f: 파일 조용한 삭제 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        // 사용법 출력 (rm -f 형식이므로 인자는 1개만 받음)
        return 0; // 조용히 종료
    }

    const char *filename = argv[1];

    // 삭제 시도 (실패해도 아무 메시지 없이 무시)
    unlink(filename);

    // 에러 메시지 없음 → rm -f는 조용히 실패를 무시함
    return 0;
}
```

설명

- unistd.h 에 정의된 unlink() 함수로 파일을 삭제한다.
- 에러가 나도 무시하기때문에 별도의 if문 은 없다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi rm_f_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o rm_f_c rm_f_c.c
ubuntu@ip-172-31-41-56:~/c_File$ touch a
ubuntu@ip-172-31-41-56:~/c_File$ ./rm_f_c a
ubuntu@ip-172-31-41-56:~/c_File$ ll
total 420
drwxrwxr-x 3 ubuntu ubuntu 4096 May 24 07:20 ./
drwxr-xr-x 9 ubuntu ubuntu 4096 May 24 07:20 ../
-rwxrwxr-x 1 ubuntu ubuntu 16048 May 23 08:41 clear_c*
-rw-rw-r-- 1 ubuntu ubuntu 301 May 23 08:41 clear_c.c
```

23. rm -r

설명

- `sys/stat.h` 헤더에 정의된 `lstat()` 함수를 사용하여, 입력된 경로가 **파일인지 디렉토리인지** 판별한다.
- 경로가 **파일일** 경우, `unlink()` 함수를 이용해 삭제한다.
- 경로가 **디렉토리일** 경우, `dirent.h` 의 `readdir()` 을 사용해 내부 항목을 하나씩 확인하고, 각 항목에 대해 ****재귀적으로 `remove_recursive()` ****를 호출한다.
- 내부 항목이 파일이면 `unlink()` 로 삭제하고, 디렉토리면 다시 `remove_recursive()` 를 통해 같은 과정을 반복한다.
- 디렉토리 내부가 모두 비워진 뒤에는 `rmdir()` 함수를 사용하여 **디렉토리 자체를 삭제**한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_file$ vi rm_r.c
ubuntu@ip-172-31-41-56:~/c_file$ gcc -o rm_r.c rm_r.c
ubuntu@ip-172-31-41-56:~/c_file$ ./rm_r.c
clear_c      echo_c      hostname_c.c  ls_al_c      ls_l_c      printenv_c    rm_c.c      rmdir_c      uname_c.c
clear_c.c    exit_c      id_c          ls_al_c.c    mkdir_c      printenv_c.c  rm_f_c      rmdir_c.c    whoami_c
date_c       exit_c.c    id_c.c        ls_c          mkdir_c.c     pwd_c          rm_f.c.c    touch_c      whoami_c.c
date_c.c     first/      ls_a_c        ls_c.c        mkdir_p_c     pwd_c.c        rm_r_c      touch_c.c
echo_c       hostname_c  ls_a_c.c      ls_l_c        mkdir_p_c.c   rm_c           rm_r_c.c    uname_c
ubuntu@ip-172-31-41-56:~/c_file$ ./rm_r.c first
'first' 삭제 완료
```

명령어

- `rm -r`: 디렉토리 및 내부 파일/디렉토리 재귀적 삭제

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>

int remove_recursive(const char *path) {
    struct stat st;

    if (lstat(path, &st) == -1) {
        perror("lstat 실패");
        return -1;
    }

    // 파일 또는 심볼릭 링크만 unlink()
    if (S_ISREG(st.st_mode)) {
        return unlink(path);
    }

    // 디렉토리인 경우
    DIR *dir = opendir(path);
    if (!dir) {
        perror("C디렉토리 열기 실패");
        return -1;
    }

    struct dirent *entry;
    char fullpath[1024];

    while ((entry = readdir(dir)) != NULL) {
        // "." ".." 무시
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;

        snprintf(fullpath, sizeof(fullpath), "%s/%s", path, entry->d_name);

        // 재귀적으로 삭제
        if (remove_recursive(fullpath) == -1) {
            closedir(dir);
            return -1;
        }
    }

    closedir(dir);

    // 디렉토리 자체 삭제
    return rmdir(path);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [디렉토리 또는 파일]\n", argv[0]);
        return 1;
    }

    if (remove_recursive(argv[1]) == -1) {
        perror("실패");
        return 1;
    }

    printf("'%' 삭제 완료\n", argv[1]);
    return 0;
}
```

24. cat

명령어

- cat: 파일 내용을 출력하는 명령어

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("파일 열기 실패");
        return 1;
    }

    int c;
    while ((c = fgetc(fp)) != EOF) {
        putchar(c);
    }

    fclose(fp);
    return 0;
}
```

설명

- stdio.h 의 fopen() 함수를 사용하여 입력된 텍스트 파일을 **읽기 전용 모드("r")**로 연다.
- fgetc() 를 통해 파일에서 **한 문자씩 읽어오고**, putchar() 로 화면에 출력한다.
- 파일의 끝(EOF)까지 반복하며 출력한 뒤, fclose() 로 파일을 닫는다.
- 텍스트 파일 하나의 전체 내용을 터미널에 출력하는 cat 명령어와 동일한 방식으로 동작한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi cat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o cat_c cat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./cat_c clear_c.c
#include <stdio.h>

int main() {
    // ANSI 이스케이프 시퀀스로 화면 지우고, 커서 맨 위로 이동
    printf("\033[2J"); // 화면 전체 지우기
    printf("\033[H");  // 커서를 좌측 상단으로 이동
    fflush(stdout);    // 출력 즉시 반영

    return 0;
}

ubuntu@ip-172-31-41-56:~/c_File$
```


25. cat -n

설명

- fopen() 함수로 파일을 읽기 전용 모드 "r"로 연다.
- fgetc()로 파일을 한 문자씩 읽어오고, putchar()로 출력한다.
- 줄 바꿈 문자 \n이 나오면 줄 번호를 증가시키고 다음 줄에도 번호를 붙인다.
- cat -n 명령어처럼 각 줄 앞에 줄 번호를 출력한다.
- 출력 포맷은 printf("%6d ", line++)처럼 줄 번호 6자리 정렬.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi cat_n.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o cat_n_c cat_n.c
ubuntu@ip-172-31-41-56:~/c_File$ ./cat_n_c cat_n.c

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[]) {
5     if (argc != 2) {
6         fprintf(stderr, "사용법: %s [파일명]\n", argv[0]);
7         return 1;
8     }
9
10    FILE *file = fopen(argv[1], "r");
11    if (!file) {
12        perror("파일 열기 실패");
13        return 1;
14    }
15
16    int c;
17    int line = 1;
18    int new_line = 1;
19
20    while ((c = fgetc(file)) != EOF) {
21        if (new_line) {
22            printf("%6d ", line++);
23            new_line = 0;
24        }
25
26        putchar(c);
27
28        if (c == '\n') {
29            new_line = 1;
30        }
31    }
32
33    fclose(file);
34    return 0;
35 }
36
ubuntu@ip-172-31-41-56:~/c_File$
```

명령어

- cat -n: 텍스트 파일의 각 줄 앞에 줄 번호를 붙여 출력하는 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일명]\n", argv[0]);
        return 1;
    }

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        perror("파일 열기 실패");
        return 1;
    }

    int c;
    int line = 1;
    int new_line = 1;

    while ((c = fgetc(file)) != EOF) {
        if (new_line) {
            printf("%6d ", line++);
            new_line = 0;
        }

        putchar(c);

        if (c == '\n') {
            new_line = 1;
        }
    }

    fclose(file);
    return 0;
}
```

26. head

명령어

- head: 파일의 처음 몇 줄을 출력하는 명령어 [기본값:10줄]

```
#include <stdio.h>

#define MAX_LINE 1024
#define DEFAULT_LINE_COUNT 10

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("파일 열기 실패");
        return 1;
    }

    char line[MAX_LINE];
    int count = 0;

    while (fgets(line, sizeof(line), fp) != NULL && count < DEFAULT_LINE_COUNT) {
        printf("%s", line);
        count++;
    }

    fclose(fp);
    return 0;
}
```

설명

- `stdio.h` 의 `fopen()` 함수를 사용하여 입력된 텍스트 파일을 **읽기 전용 모드("r")**로 연다.
- `fgets()` 를 통해 파일에서 **한 줄씩 읽고**, `printf()` 를 사용하여 화면에 출력한다.
- 출력된 줄의 수가 10줄에 도달하면 반복을 종료한다.
- 파일을 모두 출력한 뒤에는 `fclose()` 를 사용하여 파일을 닫는다.
- 이는 `head` 명령어와 동일하게, 텍스트 파일의 처음 10줄만 출력하는 동작을 수행한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi cat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o cat_c cat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./cat_c clear_c.c
#include <stdio.h>

int main() {
    // ANSI 이스케이프 시퀀스로 화면 지우고, 커서 맨 위로 이동
    printf("\033[2J"); // 화면 전체 지우기
    printf("\033[H"); // 커서를 좌측 상단으로 이동
    fflush(stdout); // 출력 즉시 반영

    return 0;
}

ubuntu@ip-172-31-41-56:~/c_File$ vi head_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o head_c head_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./head_c rm_r.c.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>

int remove_recursive(const char *path) {
    struct stat st;

ubuntu@ip-172-31-41-56:~/c_File$
```

27. head -n

명령어

- **head -n**: 파일의 처음 몇 줄을 출력하는 명령어

```
#include <stdio.h>

#define MAX_LINE 1024
#define DEFAULT_LINE_COUNT 10
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("파일 열기 실패");
        return 1;
    }
}
```

```
char line[MAX_LINE];
int count = 0;
```

```
while (fgets(line, sizeof(line), fp) != NULL && count < DEFAULT_LINE_COUNT) {
    printf("%s", line);
    count++;
}
```

```
fclose(fp);
return 0;
}
```

설명

- 프로그램 실행 시 줄 수와 파일 이름을 인자로 전달받아, `execvp()` 함수를 통해 시스템의 `head -n [줄 수] [파일 이름]` 명령어를 실행한다.
- `char *args[]` 배열에 명령어 인자들을 구성하고, `execvp("head", args)` 를 호출하여 현재 프로세스를 `head` 로 대체한다.
- 이로써 외부 명령어 `head` 가 직접 실행되어 지정된 줄 수만큼 파일 내용을 출력하게 된다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi head_n.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o head_n.c head_n.c
ubuntu@ip-172-31-41-56:~/c_File$ ./head_n.c 4
cat_c      echo_c      head_n      id_c      ls_c      mkdir_p_c   rm_c      rmdir_c      whoami_c
cat_c.c    echo_c.c    head_n.c    id_c.c     ls_c.c     mkdir_p.c.c rm_c.c     rmdir_c.c    whoami_c.c
clear_c     exit_c     head_n.c    ls_a.c     ls_l.c     printenv_c  rm_f_c     touch_c
clear_c.c  exit_c.c   head_n.c.c ls_a.c.c   ls_l.c.c   printenv.c.c rm_f.c.c   touch_c.c
date_c     head_c     hostname_c  ls_al.c    mkdir_c    pwd_c       rm_r_c     uname_c
date_c.c   head_c.c   hostname_c.c ls_al.c.c  mkdir_c.c  pwd_c.c     rm_r.c.c   uname_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./head_n.c 4 clear_c.c
#include <stdio.h>

int main() {
    // ANSI 이스케이프 시퀀스로 화면 지우고, 커서 맨 위로 이동
ubuntu@ip-172-31-41-56:~/c_File$
```

28. tail

명령어

- tail: 파일의 마지막 몇 줄을 출력하는 명령어 [기간값: 10줄]



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

#define MAX_LINES 1024
#define MAX_LINE_LENGTH 1024
#define DEFAULT_TAIL_LINES 10

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        perror("파일 열기 실패");
        return 1;
    }

    char *lines[MAX_LINES];
    int count = 0;

    // 한 줄씩 읽고 배열에 저장
    while (!feof(fp)) {
        char buffer[MAX_LINE_LENGTH];
        if (fgets(buffer, sizeof(buffer), fp)) {
            lines[count % MAX_LINES] = strdup(buffer);
            count++;
        }
    }
    fclose(fp);

    // 출력 시작 인덱스 계산
    int start = count > DEFAULT_TAIL_LINES ? count - DEFAULT_TAIL_LINES : 0;

    // 출력
    for (int i = start; i < count; i++) {
        printf("%s", lines[i % MAX_LINES]);
        free(lines[i % MAX_LINES]); // 메모리 해제
    }

    return 0;
}
```

설명

- fopen() 함수로 텍스트 파일을 읽기 전용 모드로 연다.
- fgets() 를 사용해 한 줄씩 읽고, strdup() 으로 문자열을 동적으로 저장한다.
- 최대 1024줄까지 순환 버퍼 형태로 저장하여, 최근 줄만 유지하도록 한다.
- 읽은 줄의 총 개수가 10줄보다 많으면, count - 10 부터 출력한다.
- 출력 후 free() 함수를 통해 메모리를 해제한다.
- tail 명령어와 동일하게, 파일의 마지막 10줄만 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi tail_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o tail_c tail_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./tail_c rm_r.c.c

if (remove_recursive(argv[1]) == -1) {
    perror("삭제 실패");
    return 1;
}

printf("%s' 삭제 완료\n", argv[1]);
return 0;
}
ubuntu@ip-172-31-41-56:~/c_File$
```

29. tail -n

명령어

- tail -n: 파일의 마지막 n개 줄을 출력하는 명령어



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

#define MAX_LINES 1024
#define MAX_LINE_LENGTH 1024
#define DEFAULT_TAIL_LINES 10

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일 이름]\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        perror("파일 열기 실패");
        return 1;
    }

    char *lines[MAX_LINES];
    int count = 0;

    // 한 줄씩 읽고 배열에 저장
    while (!feof(fp)) {
        char buffer[MAX_LINE_LENGTH];
        if (fgets(buffer, sizeof(buffer), fp)) {
            lines[count % MAX_LINES] = strdup(buffer);
            count++;
        }
    }
    fclose(fp);

    // 출력 시작 인덱스 계산
    int start = count > DEFAULT_TAIL_LINES ? count - DEFAULT_TAIL_LINES : 0;

    // 출력
    for (int i = start; i < count; i++) {
        printf("%s", lines[i % MAX_LINES]);
        free(lines[i % MAX_LINES]); // 메모리 해제
    }

    return 0;
}
```

설명

- argv[1] 에서 출력할 줄 수를 정수로 파싱하고, argv[2] 의 파일을 fopen() 으로 읽기 전용으로 연다.
- fgets() 로 줄 단위로 읽고, strdup() 으로 복사하여 순환 배열에 저장한다.
- 총 줄 개수에서 출력할 줄 수를 뺀 인덱스부터 출력하고, free() 로 메모리를 해제한다.
- tail -n [줄 수] [파일명] 명령어와 같은 방식으로 동작한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi tail_n.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o tail_n tail_n.c
ubuntu@ip-172-31-41-56:~/c_File$ ./tail_n 7 date_c
ote.gnu.propertyubuntu@ip-172-31-41-56:~/c_File$ ./tail_n 7 head_c.c
count++;
}

fclose(fp);
return 0;
}

ubuntu@ip-172-31-41-56:~/c_File$
```

30. env

명령어

- env: 현재 환경 변수 전체 목록을 출력하는 명령어

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

```
extern char **environ;
```

```
int main() {
    for (char **env = environ; *env != NULL; env++) {
        printf("%s\n", *env);
    }
    return 0;
}
```

설명

- 전역 변수 `environ` 은 모든 환경 변수들을 문자열 배열 형태로 담고 있다.
- `for` 반복문으로 `NULL` 이 나올 때까지 환경 변수 문자열을 하나씩 출력한다.
- `env` 명령어처럼 현재 실행 환경의 모든 환경 변수를 출력한다.

컴파일 및 실행화면

[illegible]

31. file

명령어

- file: 파일의 **종류(type)**를 출력하는 명령어

```
#include <stdio.h>
#include <sys/stat.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일명]\n", argv[0]);
        return 1;
    }

    struct stat st;
    if (stat(argv[1], &st) == -1) {
        perror("stat 실패");
        return 1;
    }

    printf("%s: ", argv[1]);

    if (S_ISREG(st.st_mode)) {
        printf("regular file\n");
    } else if (S_ISDIR(st.st_mode)) {
        printf("directory\n");
    } else if (S_ISLNK(st.st_mode)) {
        printf("symbolic link\n");
    } else if (S_ISCHR(st.st_mode)) {
        printf("character device\n");
    } else if (S_ISBLK(st.st_mode)) {
        printf("block device\n");
    } else if (S_ISFIFO(st.st_mode)) {
        printf("FIFO/pipe\n");
    } else if (S_ISSOCK(st.st_mode)) {
        printf("socket\n");
    } else {
        printf("unknown type\n");
    }

    return 0;
}
```



설명

- sys/stat.h 의 stat() 함수로 파일 메타데이터를 구조체로 받아온다.
- st_mode 의 비트 값을 S_ISREG(), S_ISDIR() 등 매크로로 검사하여 파일의 타입을 판별한다.
- file 명령어처럼 해당 파일이 일반 파일인지, 디렉토리인지, 심볼릭 링크인지 등을 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi file_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o file_c file_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./file_c id_g.c.c
id_g_c.c: regular file
ubuntu@ip-172-31-41-56:~/c_File$ ./file_c id_g_c
id_g_c: regular file
ubuntu@ip-172-31-41-56:~/c_File$ ./file_c /etc
/etc: directory
ubuntu@ip-172-31-41-56:~/c_File$
```

32. who

명령어

- **who**: 현재 실행 중인 프로세스 목록을 출력하는 명령어

```
#include <stdio.h>
#include <utmp.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>

int main() {
    struct utmp entry;

    // utmp 파일 열기
    int fd = open(_PATH_UTMP, O_RDONLY);
    if (fd == -1) {
        perror("utmp 열기 실패");
        return 1;
    }

    printf("USER      TTY      DATE      TIME\n");

    while (read(fd, &entry, sizeof(entry)) == sizeof(entry)) {
        if (entry.ut_type == USER_PROCESS) {
            char timebuf[32];
            struct tm *lt = localtime((time_t *) &entry.ut_tv.tv_sec);
            strftime(timebuf, sizeof(timebuf), "%Y-%m-%d %H:%M", lt);

            printf("%-8s %-8s %s\n", entry.ut_user, entry.ut_line, timebuf);
        }
    }

    close(fd);
    return 0;
}
```



설명

- `utmp.h` 의 `_PATH_UTMP` 경로(`/var/run/utmp`)에서 로그인 세션 정보를 읽는다.
- `utmp` 구조체 배열을 순회하면서 `ut_type == USER_PROCESS` 인 항목만 출력한다.
- 사용자 이름(`ut_user`), 터미널 이름(`ut_line`), 로그인 시간(`ut_tv`)을 출력한다.
- `strftime()` 과 `localtime()` 을 사용하여 시간을 사람이 읽기 좋은 형식으로 변환한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi who_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o who_c who_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./who_c
USER      TTY      DATE      TIME
ubuntu pts/0  92843933-10-13 20:24
ubuntu@ip-172-31-41-56:~/c_File$
```


33. who -u

설명

- utmp.h 의 _PATH_UTMP 를 열고, ut_type == USER_PROCESS 인 사용자 항목만 필터링한다.
- ut_pid 필드를 통해 해당 사용자의 프로세스 ID를 출력한다.
- ut_line 을 기반으로 /dev/ttyX 경로를 생성하고, stat() 의 st_atime 을 통해 idle 시간(입력 없던 시간)을 계산한다.
- idle 시간은 분 단위로 계산하여 출력하며, 0분이면 . 을, 음수거나 오류면 > 로 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi who_u.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o who_u_c who_u.c
ubuntu@ip-172-31-41-56:~/c_File$ ./who_u_c
USER      TTY      DATE      TIME      IDLE      PID
ubuntu    pts/0    92843933-10-13 20:24    .         111063
ubuntu@ip-172-31-41-56:~/c_File$
```

명령어

- who -u 현재 실행 중인 프로세스 목록을 구체적으로 출력하는 명령어

```
#include <stdio.h>
#include <utmp.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>

#define IDLE_THRESHOLD 60 // 60초 단위로 idle 시간 계산

int main() {
    struct utmp entry;
    int fd = open(_PATH_UTMP, O_RDONLY);
    if (fd == -1) {
        perror("utmp 열기 실패");
        return 1;
    }

    printf("USER      TTY      DATE      TIME      IDLE      PID\n");

    while (read(fd, &entry, sizeof(entry)) == sizeof(entry)) {
        if (entry.ut_type == USER_PROCESS) {
            char timebuf[32];
            struct tm *lt = localtime((time_t *) &entry.ut_tv.tv_sec);
            strftime(timebuf, sizeof(timebuf), "%Y-%m-%d %H:%M", lt);

            // idle 시간 계산
            char tty_path[64];
            snprintf(tty_path, sizeof(tty_path), "/dev/%s", entry.ut_line);
            struct stat st;
            int idle_minutes = -1;

            if (stat(tty_path, &st) == 0) {
                time_t now = time(NULL);
                idle_minutes = (now - st.st_atime) / 60;
            }

            // idle 시간 형식
            char idle_buf[16];
            if (idle_minutes < 0) {
                snprintf(idle_buf, sizeof(idle_buf), "?");
            } else if (idle_minutes == 0) {
                snprintf(idle_buf, sizeof(idle_buf), ".");
            } else {
                snprintf(idle_buf, sizeof(idle_buf), "%02d%02d", idle_minutes / 60, idle_minutes % 60);
            }

            printf("%-8s %-8s %s %s %s %s\n",
                    entry.ut_user,
                    entry.ut_line,
                    timebuf,
                    idle_buf,
                    entry.ut_pid);
        }
    }

    close(fd);
    return 0;
}
```

34. uptime

명령어

- **uptime**: 얼마나 오래 켜져 있었는지(시작 이후 경과 시간)를 출력하는 명령어

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

int main() {
    FILE *fp = fopen("/proc/uptime", "r");
    if (fp == NULL) {
        perror("/proc/uptime 열기 실패");
        return 1;
    }

    double uptime_seconds;
    if (fscanf(fp, "%lf", &uptime_seconds) != 1) {
        perror("uptime 읽기 실패");
        fclose(fp);
        return 1;
    }

    fclose(fp);

    int days = (int)(uptime_seconds / 86400);
    int hours = ((int)uptime_seconds % 86400) / 3600;
    int minutes = ((int)uptime_seconds % 3600) / 60;

    printf("시스템 가동 시간: ");
    if (days > 0) {
        printf("%d일 ", days);
    }
    if (hours > 0 || days > 0) {
        printf("%d시간 ", hours);
    }
    printf("%d분\n", minutes);

    return 0;
}
```



설명

- `/proc/uptime` 파일은 시스템이 부팅된 이후 지난 초(second)를 담고 있다.
- 첫 번째 값(예: 10231.48)만 `fscanf()` 로 읽어온다.
- 초 단위를 읽은 후 일(day), 시간(hour), 분(minute) 단위로 변환하여 출력한다.
- 텍스트 기반의 간단한 `uptime` 기능을 구현하였다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi uptime_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o uptime_c uptime_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./uptime_c
시스템 가동 시간: 11일 5시간 11분
ubuntu@ip-172-31-41-56:~/c_File$
```

35. df

명령어

- df: 디스크의 파일 시스템 사용량(전체, 사용 중, 사용 가능 공간 등) 을 출력하는 명령어

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/statvfs.h>
```

```
int main() {
    struct statvfs fs;

    if (statvfs("/", &fs) != 0) {
        perror("statvfs 실패");
        return 1;
    }
}
```

```
unsigned long total = fs.f_blocks * fs.f_frsize;
unsigned long free = fs.f_bfree * fs.f_frsize;
unsigned long available = fs.f_bavail * fs.f_frsize;
unsigned long used = total - free;
```

```
printf("파일시스템\t총용량\t사용중\t사용가능\t마운트지점\n");
printf("/dev/root\t%lu\t%lu\t%lu\t/\n",
    total,
    used,
    available);

return 0;
}
```



설명

- sys/statvfs.h 헤더의 statvfs() 함수는 특정 경로에 대한 파일시스템 정보를 구조체에 채운다.
- f_blocks × f_frsize 로 전체 블록 크기를 계산한다.
- f_bfree : 전체 블록 중 사용 가능한 총 블록 (루트 포함)
- f_bavail : 일반 사용자에게 사용 가능한 블록
- 사용량 = 전체 - f_bfree
- 출력은 루트("/") 기준이며, 실제 df 명령어처럼 마운트된 위치의 디스크 사용량 정보를 보여준다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi df.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o df_c df.c
ubuntu@ip-172-31-41-56:~/c_File$ ./df_c
파일 시스템\t총용량\t사용중\t사용가능\t마운트 지점
/dev/root\t7203201024\t6693937152\t492486656\t/
ubuntu@ip-172-31-41-56:~/c_File$
```

36. df -h

설명

- statvfs() 함수로 루트 디렉토리(/)의 파일시스템 정보를 불러온다.
- 용량은 바이트 단위로 계산하고, 사람이 보기 쉬운 단위(B, K, M, G, T)로 변환하여 출력한다.
- print_size() 함수는 자동으로 적절한 단위를 선택해 2자리 소수까지 포맷하여 출력한다.
- df -h 명령어와 동일한 형식으로 동작한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi df_h.c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o df_h_c df_h.c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./df_h_c
파일 시스템      총용량    사용중    사용가능    마운트지점
/dev/root         6.71G    6.23G    469.65M    /
ubuntu@ip-172-31-41-56:~/c_File$
```

명령어

- df -h: 디스크의 파일 시스템 사용량(전체, 사용 중, 사용 가능 공간 등)을 사람이 읽기 좋게 출력하는 명령어

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/statvfs.h>

void print_size(unsigned long bytes) {
    const char *units[] = {"B", "K", "M", "G", "T"};
    int i = 0;
    double size = bytes;

    while (size >= 1024 && i < 4) {
        size /= 1024;
        i++;
    }

    printf("%.2f%s", size, units[i]);
}

int main() {
    struct statvfs fs;

    if (statvfs("/", &fs) != 0) {
        perror("statvfs 실패");
        return 1;
    }

    unsigned long total = fs.f_blocks * fs.f_frsize;
    unsigned long free = fs.f_bfree * fs.f_frsize;
    unsigned long available = fs.f_bavail * fs.f_frsize;
    unsigned long used = total - free;

    printf("파일시스템      총용량    사용중    사용가능    마운트지점\n");
    printf("/dev/root          ");
    print_size(total);
    printf(" ");
    print_size(used);
    printf(" ");
    print_size(available);
    printf("  /\n");

    return 0;
}
```

37. df -T

명령어

- ping -c: ping -c [횟수] [호스트명] 은 지정한 횟수만큼 ICMP ping 패킷을 보내는 명령어

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
unsigned short checksum(void *b, int len){
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;
```

```
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    if (len == 1)
        sum += *(unsigned char*)buf;
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
```

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "사용법: %s [횟수] [호스트명]\n", argv[0]);
        return 1;
    }

    int count = atoi(argv[1]);
    char *host = argv[2];

    struct hostent *h;
    struct sockaddr_in addr;
    int sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

    if (sockfd < 0) {
        perror("소켓 생성 실패 (루트 권한 필요)");
        return 1;
    }

    h = gethostbyname(host);
    if (!h) {
        fprintf(stderr, "호스트 이름 해석 실패\n");
        return 1;
    }

    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    memcpy(&addr.sin_addr, h->h_addr, h->h_length);

    for (int i = 0; i < count; i++) {
        char packet[64];
        struct icmp *icmp_hdr = (struct icmp *) packet;
        memset(packet, 0, sizeof(packet));
        icmp_hdr->icmp_type = ICMP_ECHO;
        icmp_hdr->icmp_code = 0;
        icmp_hdr->icmp_id = getpid();
        icmp_hdr->icmp_seq = 1 + i;
        icmp_hdr->icmp_cksum = checksum(icmp_hdr, sizeof(packet));

        if (sendto(sockfd, packet, sizeof(packet), 0, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
            perror("패킷 전송 실패");
        } else {
            printf("ping %s에 %s 전송 완료\n", i + 1, inet_ntoa(addr.sin_addr));
        }

        sleep(1);
    }

    close(sockfd);
    return 0;
}
```

설명

- socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)로 ICMP용 소켓 생성
- gethostbyname()으로 도메인 이름을 IP로 변환
- ICMP_ECHO 패킷을 생성하고, 지정된 횟수만큼 sendto()로 전송
- 실제 응답 받는 부분(recvfrom, 응답 시간 측정)은 생략됨 (간단 버전)

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ping_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ping_c ping_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ping_c 2 google.com
소켓 생성 실패 (루트 권한 필요): Operation not permitted
ubuntu@ip-172-31-41-56:~/c_File$ sudo ./ping_c google.com
사용법: ./ping_c [횟수] [호스트명]
ubuntu@ip-172-31-41-56:~/c_File$ sudo ./ping_c 2 google.com
ping #1 -> 142.250.74.110 전송 완료
ping #2 -> 142.250.74.110 전송 완료
ubuntu@ip-172-31-41-56:~/c_File$
```

38. which

명령어

- which: 디명령어 이름에 해당하는 실행 파일의 경로를 출력하는 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [명령어 이름]\n", argv[0]);
        return 1;
    }

    char *path_env = getenv("PATH");
    if (path_env == NULL) {
        fprintf(stderr, "PATH 환경변수를 찾을 수 없습니다.\n");
        return 1;
    }

    char *path_copy = strdup(path_env);
    char *dir = strtok(path_copy, ":");

    while (dir != NULL) {
        char full_path[1024];
        snprintf(full_path, sizeof(full_path), "%s/%s", dir, argv[1]);

        if (access(full_path, X_OK) == 0) {
            printf("%s\n", full_path);
            free(path_copy);
            return 0;
        }

        dir = strtok(NULL, ":");
    }

    printf("%s: 명령어를 찾을 수 없습니다.\n", argv[1]);
    free(path_copy);
    return 1;
}
```

설명

- `getenv("PATH")` 를 통해 `PATH` 환경변수를 가져온다.
- `strtok()` 으로 : 구분자를 기준으로 디렉토리를 하나씩 분리한다.
- 각 디렉토리에 대해 명령어 이름을 붙여 전체 경로를 만든 뒤, `access(path, X_OK)` 를 통해 실행 가능 여부를 확인한다.
- 찾으면 경로를 출력하고, 못 찾으면 오류 메시지를 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi which_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o which_c which_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./which_c ls
/usr/bin/ls
ubuntu@ip-172-31-41-56:~/c_File$
```

39. whereis

명령어

- whereis: 명령어 관련 파일(실행파일, man 파일 등) 위치 모두 확인

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

```
const char *paths[] = {
    "/bin",
    "/usr/bin",
    "/sbin",
    "/usr/sbin",
    "/usr/local/bin",
    "/usr/share/man/man1"
};
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [명령어 이름]\n", argv[0]);
        return 1;
    }
```

```
    const char *cmd = argv[1];
    char fullpath[1024];
```

```
    for (int i = 0; i < sizeof(paths) / sizeof(paths[0]); i++) {
        snprintf(fullpath, sizeof(fullpath), "%s/%s", paths[i], cmd);
```

```
        // 실행 파일 또는 매뉴얼 존재 여부 확인
```

```
        if (access(fullpath, F_OK) == 0) {
            printf("%s\n", fullpath);
```

```
        } else {
```

```
            // man 파일은 압축된 경우도 있음
```

```
            snprintf(fullpath, sizeof(fullpath), "%s/%s.1.gz", paths[i], cmd);
```

```
            if (access(fullpath, F_OK) == 0) {
                printf("%s\n", fullpath);
```

```
            }
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

설명

- whereis 명령어는 실행파일과 man 페이지의 경로를 알려준다.
- access() 함수를 통해 미리 지정한 경로들에서 해당 파일이 존재하는지 확인한다.
- 실행 파일(bin 등)과 man 파일(.1.gz) 디렉토리(/usr/share/man/man1) 등을 순회하며 출력한다.
- 옵션 없이 whereis [명령어이름] 만 지원한다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi whereis_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o whereis_c whereis_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./whereis_c ls
/bin/ls
/usr/bin/ls
/usr/share/man/man1/ls.1.gz
ubuntu@ip-172-31-41-56:~/c_File$
```

40. ping

명령어

- ping: 무한히 ICMP ping 패킷을 보내는 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [호스트명 또는 IP 주소]\n", argv[0]);
        return 1;
    }

    char command[256];
    // -c 4 제거 → 무한 ping
    snprintf(command, sizeof(command), "ping %s", argv[1]);

    int result = system(command);

    if (result == -1) {
        perror("ping 실행 실패");
        return 1;
    }

    return 0;
}
```

설명

- argv[1]에서 호스트명 또는 IP 주소를 입력받는다.
- 문자열 결합 함수인 snprintf()로 ping [주소] 명령어 문자열을 구성한다.
- system() 함수를 사용하여 ping 명령어를 실행한다.
- -c 옵션을 사용하지 않았기 때문에, ICMP 패킷을 무한 반복 전송하며, Ctrl+C로 수동 종료해야 한다.
- 외부 명령어를 호출하므로, 로컬에 ping 명령어가 설치되어 있어야 한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ping.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ping_c ping.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ping_c google.com
PING google.com (216.58.207.206) 56(84) bytes of data.
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=1 ttl=118 time=3.15 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=2 ttl=118 time=3.19 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=3 ttl=118 time=3.24 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=4 ttl=118 time=3.19 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=5 ttl=118 time=3.20 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=6 ttl=118 time=3.21 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=7 ttl=118 time=3.20 ms
64 bytes from arn1s04-in-f14.1e100.net (216.58.207.206): icmp_seq=8 ttl=118 time=3.18 ms
^C
--- google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7009ms
rtt min/avg/max/mdev = 3.152/3.195/3.235/0.022 ms
ubuntu@ip-172-31-41-56:~/c_File$
```


41. ping -c

명령어

- ping -c: ping -c [횟수] [호스트명] 은 지정한 횟수만큼 ICMP ping 패킷을 보내는 명령어

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;
```

```
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    if (len == 1)
        sum += *(unsigned char*)buf;
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "사용법: %s [횟수] [호스트명]\n", argv[0]);
        return 1;
    }

    int count = atoi(argv[1]);
    char *host = argv[2];

    struct hostent *h;
    struct sockaddr_in addr;
    int sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

    if (sockfd < 0) {
        perror("소켓 생성 실패 (루트 권한 필요)");
        return 1;
    }

    h = gethostbyname(host);
    if (!h) {
        fprintf(stderr, "호스트 이름 확인 실패\n");
        return 1;
    }

    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    memcpy(&addr.sin_addr, h->h_addr, h->h_length);

    for (int i = 0; i < count; i++) {
        char packet[64];
        struct icmp_hdr = (struct icmp *) packet;
        memset(packet, 0, sizeof(packet));
        icmp_hdr->icmp_type = ICMP_ECHO;
        icmp_hdr->icmp_code = 0;
        icmp_hdr->icmp_id = getpid();
        icmp_hdr->icmp_seq = i + 1;
        icmp_hdr->icmp_cksum = checksum(icmp_hdr, sizeof(packet));

        if (sendto(sockfd, packet, sizeof(packet), 0, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
            perror("패킷 전송 실패");
        } else {
            printf("ping %d -> %s 전송 완료\n", i + 1, inet_ntoa(addr.sin_addr));
        }

        sleep(1);
    }

    close(sockfd);
    return 0;
}
```

설명

- socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)로 ICMP용 소켓 생성
- gethostbyname()으로 도메인 이름을 IP로 변환
- ICMP_ECHO 패킷을 생성하고, 지정한 횟수만큼 sendto()로 전송
- 실제 응답 받는 부분(recvfrom, 응답 시간 측정)은 생략됨 (간단 버전)

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ping_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ping_c_c ping_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ping_c_c 2 google.com
소켓 생성 실패 (루트 권한 필요): Operation not permitted
ubuntu@ip-172-31-41-56:~/c_File$ sudo ./ping_c_c google.com
사용법: ./ping_c_c [횟수] [호스트명]
ubuntu@ip-172-31-41-56:~/c_File$ sudo ./ping_c_c 2 google.com
ping #1 -> 142.250.74.110 전송 완료
ping #2 -> 142.250.74.110 전송 완료
ubuntu@ip-172-31-41-56:~/c_File$
```

42. curl

명령어

- curl HTTP 요청을 보내고 응답을 출력하는 명령어

```
#include <stdio.h>
#include <curl/curl.h>

size_t write_callback(void *ptr, size_t size, size_t nmemb, void *userdata) {
    size_t total_size = size * nmemb;
    fwrite(ptr, size, nmemb, stdout); // 받은 데이터를 stdout에 출력
    return total_size;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [URL]\n", argv[0]);
        return 1;
    }

    CURL *curl = curl_easy_init();
    if (!curl) {
        fprintf(stderr, "libcurl 초기화 실패\n");
        return 1;
    }

    curl_easy_setopt(curl, CURLOPT_URL, argv[1]); // 요청할 URL
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback); // 응답 처리 콜백
    curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L); // 리디렉션 따라가기

    CURLcode res = curl_easy_perform(curl); // 요청 실행
    if (res != CURLE_OK) {
        fprintf(stderr, "요청 실패: %s\n", curl_easy_strerror(res));
    }

    curl_easy_cleanup(curl); // 리소스 해제
    return 0;
}
```

설명

- libcurl은 HTTP 요청을 보낼 수 있는 C용 라이브러리
- curl_easy_setopt()으로 요청 설정
- curl_easy_perform()으로 실행
- write_callback()에서 받은 응답을 터미널에 출력

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi curl.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o curl.o curl.c -lcurl
ubuntu@ip-172-31-41-56:~/c_File$ ./curl.o https://example.com
<!doctype html>
<html>
<head>
  <title>Example Domain</title>

  <meta charset="utf-8" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style type="text/css">
    body {
      background-color: #f0f0f2;
      margin: 0;
      padding: 0;
      font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
    }
    div {
      width: 600px;
      margin: 50px auto;
      padding: 20px;
      background-color: #fdd;
      border-radius: 0.5em;
      border: 1px solid #fdd;
      box-shadow: 2px 2px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
      color: #38488f;
      text-decoration: none;
    }
    @media (max-width: 700px) {
      div {
        margin: 0 auto;
        width: auto;
      }
    }
  </style>
</head>
<body>
<div>
  <h1>Example Domain</h1>
  <p>This domain is for use in illustrative examples in documents. You may use this
  domain in literature without prior coordination or asking for permission.</p>
  <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
ubuntu@ip-172-31-41-56:~/c_File$
```

43. basename

명령어

- **basename**: 경로 문자열에서 파일 이름만 추출하는 명령어

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

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [경로]\n", argv[0]);
        return 1;
    }
}
```

```
char path[1024];
snprintf(path, sizeof(path), "%s", argv[1]);

printf("%s\n", basename(path)); // 경로에서 파일 이름 추출
return 0;
```

설명

- `libgen.h`에 정의된 `basename()` 함수는 문자열에서 가장 마지막 / 이후의 부분을 반환한다.
- 원본 문자열을 직접 수정하므로, `snprintf()`로 복사한 후 사용한다.
- 경로를 인수로 받아서 파일명만 추출한 뒤 출력한다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi basename.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o basename_c basename.c
ubuntu@ip-172-31-41-56:~/c_File$ ./basename_c
사용법: ./basename_c [경로]
ubuntu@ip-172-31-41-56:~/c_File$ ./basename_c
basename_c      df_h.c.c      head_n      id_u.c      mkdir_p.c      rm_c      touch_c      who_c
basename_c.c    df_h.c.c      head_n.c    id_u.c.c    mkdir_p.c.c    rm_c.c    touch_c.c    who_c.c
cat_c           echo_c        head_n.c    ls_a.c      ping_c         rm_f.c    uname_a.c    who_u.c
cat_c.c         echo_c.c      head_n.c.c  ls_a.c.c    ping_c.c       rm_f.c.c  uname_a.c.c  who_u.c.c
clear_c         env_c         hostname_I.c ls_al.c     ping_c.c       rm_r.c    uname_c      whoami_c
clear_c.c       env_c.c       hostname_I.c.c ls_al.c.c   ping_c.c.c     rm_r.c.c  uname_c.c    whoami_c.c
curl_c          exit_c        hostname_c   ls_c        printenv_c     rmdir_c   uptime_c
curl_c.c        exit_c.c      hostname_c.c ls_c.c      printenv_c.c   rmdir_c.c uptime_c.c
date_c          file_c        id_c        ls_l.c      ps_c           tail_c    whereis_c
date_c.c        file_c.c      id_c.c      ls_l.c.c    ps_c.c         tail_c.c  whereis_c.c
df_c            head_c        id_g.c      mkdir_c     pwd_c          tail_n    which_c
df_c.c          head_c.c      id_g.c.c    mkdir_c.c   pwd_c.c        tail_n.c  which_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./basename_c env_c
env_c
ubuntu@ip-172-31-41-56:~/c_File$
```

44. dirname

명령어

- `dirname`: 경로에서 디렉토리 이름만 추출하는 명령어

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

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [경로]\n", argv[0]);
        return 1;
    }
}
```

```
char path[1024];
snprintf(path, sizeof(path), "%s", argv[1]);

printf("%s\n", dirname(path)); // 경로에서 디렉토리 경로 추출
return 0;
```

```
}
```

설명

- `libgen.h`에 정의된 `dirname()` 함수는 문자열에서 마지막 '/' 앞부분을 반환한다.
- `dirname()`은 원본 문자열을 수정하므로, `snprintf()`로 복사한 후 사용한다.
- 경로가 포함된 파일이나 디렉토리 경로를 받아 디렉토리 부분만 출력한다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi dirname.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o dirname_c dirname.c
ubuntu@ip-172-31-41-56:~/c_File$ ./dirname_c /home/ubuntu/
.bash_history      .cache/            .local/            .ssh/              WebProgrammingTeamProject/
.bash_logout       .config/           .npm/              .sudo_as_admin_successful c_File/
.bashrc            .lessshat         .profile           .viminfo
ubuntu@ip-172-31-41-56:~/c_File$ ./dirname_c /home/ubuntu/
.bash_history      .cache/            .local/            .ssh/              WebProgrammingTeamProject/
.bash_logout       .config/           .npm/              .sudo_as_admin_successful c_File/
.bashrc            .lessshat         .profile           .viminfo
ubuntu@ip-172-31-41-56:~/c_File$ ./dirname_c /home/ubuntu/.bash_logout
/home/ubuntu
ubuntu@ip-172-31-41-56:~/c_File$
```

45. sleep

명령어

- **sleep**: 주어진 초(seconds)만큼 프로그램 실행을 일시 중지하는 명령어

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [초 단위 시간]\n", argv[0]);
        return 1;
    }

    int seconds = atoi(argv[1]); // 문자열 → 정수 변환
    if (seconds < 0) {
        fprintf(stderr, "양의 정수를 입력하세요.\n");
        return 1;
    }

    sleep(seconds); // 실행 일시 정지

    return 0;
}
```

설명

- unistd.h에 정의된 sleep() 함수는 초 단위로 프로그램을 일시 중지시킨다.
- atoi() 함수로 입력 문자열을 정수로 바꾼 뒤, 음수면 오류 처리한다.
- ./sleep_c 5라고 실행하면 5초 동안 멈췄다가 종료된다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi sleep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o sleep_c sleep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./sleep_c 3
ubuntu@ip-172-31-41-56:~/c_File$
```

46. stat

명령어

- stat: 파일의 정보(크기, 권한, 마지막 수정 시간 등)를 출력하는 명령어

설명

- sys/stat.h의 stat() 함수로 파일 정보를 가져온다.
- struct stat 구조체에는 파일의 크기, 권한, 소유자 등의 정보가 들어 있다.
- getpwuid(), getgrgid()로 사용자 이름과 그룹 이름을 얻는다.
- ctime() 함수로 마지막 수정 시간을 사람이 읽을 수 있는 형식으로 출력한다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi stat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o stat_c stat_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./stat_c sleep_c
파일 : sleep_c
크기 : 16136 바이트
권한 : 775
소유자 : ubuntu
그룹 : ubuntu
마지막 수정 시간 : Wed Jun  4 19:33:06 2025
ubuntu@ip-172-31-41-56:~/c_File$
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "사용법: %s [파일명]\n", argv[0]);
        return 1;
    }

    struct stat st;

    if (stat(argv[1], &st) == -1) {
        perror("stat 실패");
        return 1;
    }

    struct passwd *pw = getpwuid(st.st_uid);
    struct group *gr = getgrgid(st.st_gid);

    printf("파일: %s\n", argv[1]);
    printf("크기: %ld 바이트\n", st.st_size);
    printf("권한: %o\n", st.st_mode & 0777);
    printf("소유자: %s\n", pw ? pw->pw_name : "알 수 없음");
    printf("그룹: %s\n", gr ? gr->gr_name : "알 수 없음");
    printf("마지막 수정 시간: %s", ctime(&st.st_mtime)); // 개행 포함

    return 0;
}
```

47. grep

명령어

- df -T: 주어진 파일에서 특정 문자열이 포함된 줄을 출력하는 명령어

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

#define MAX_LINE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "사용법: %s [검색할 문자열] [파일명]\n", argv[0]);
        return 1;
    }

    const char *keyword = argv[1];
    const char *filename = argv[2];

    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("파일 열기 실패");
        return 1;
    }

    char line[MAX_LINE];
    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, keyword) != NULL) {
            printf("%s", line);
        }
    }

    fclose(file);
    return 0;
}
```

설명

- fopen()으로 파일을 읽기 모드로 연다.
- fgets()로 한 줄씩 읽는다.
- strstr()로 해당 줄에 검색어가 포함되어 있는지 확인한다.
- 포함되어 있으면 printf()로 해당 줄을 출력한다.
- 파일을 모두 읽고 나면 fclose()로 닫는다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi grep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o grep_c grep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./grep_c
사용법: ./grep_c [검색할 문자열] [파일명]
ubuntu@ip-172-31-41-56:~/c_File$ ./grep_c print grep_c.c
        fprintf(stderr, "사용법: %s [검색할 문자열] [파일명]\n", argv[0]);
        printf("%s", line);
ubuntu@ip-172-31-41-56:~/c_File$
```

48. wc

명령어

- df -T: 주어진 파일에서 특정 문자열이 포함된 줄을 출력하는 명령어

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

#define MAX_LINE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "사용법: %s [검색할 문자열] [파일명]\n", argv[0]);
        return 1;
    }

    const char *keyword = argv[1];
    const char *filename = argv[2];

    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("파일 열기 실패");
        return 1;
    }

    char line[MAX_LINE];
    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, keyword) != NULL) {
            printf("%s", line);
        }
    }

    fclose(file);
    return 0;
}
```

설명

- fopen()으로 파일을 읽기 모드로 연다.
- fgets()로 한 줄씩 읽는다.
- strstr()로 해당 줄에 검색어가 포함되어 있는지 확인한다.
- 포함되어 있으면 printf()로 해당 줄을 출력한다.
- 파일을 모두 읽고 나면 fclose()로 닫는다.

컴파일 및 실행하면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi grep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o grep_c grep_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./grep_c
사용법: ./grep_c [검색할 문자열] [파일명]
ubuntu@ip-172-31-41-56:~/c_File$ ./grep_c print grep_c.c
        fprintf(stderr, "사용법: %s [검색할 문자열] [파일명]\n", argv[0]);
        printf("%s", line);
ubuntu@ip-172-31-41-56:~/c_File$
```


49. ps

명령어

- ps: 현재 실행 중인 프로세스 목록을 출력하는 명령어

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

int is_number(const char *str) {
    while (*str) {
        if (!isdigit(*str)) return 0;
        str++;
    }
    return 1;
}

int main() {
    DIR *proc = opendir("/proc");
    struct dirent *entry;

    if (!proc) {
        perror("/proc 열기 실패");
        return 1;
    }

    printf(" PID CMD\n");

    while ((entry = readdir(proc)) != NULL) {
        if (is_number(entry->d_name)) {
            char path[512]; // 버퍼하에 놀림
            char cmdline[256];
            FILE *fp;

            // PID 길이를 제한
            if (strlen(entry->d_name) > 20) continue;

            snprintf(path, sizeof(path), "/proc/%s/cmd", entry->d_name);

            fp = fopen(path, "r");
            if (fp) {
                if (fgets(cmdline, sizeof(cmdline), fp)) {
                    cmdline[strcspn(cmdline, "\n")] = '\0';
                    printf("%5s %s\n", entry->d_name, cmdline);
                }
                fclose(fp);
            }
        }
    }

    closedir(proc);
    return 0;
}
```

설명

- /proc 디렉토리에서 숫자로 된 PID 디렉토리만 필터링한다.
- 각 /proc/[PID]/cmd 파일을 열어 프로세스 이름을 읽어온다.
- fgets() 로 이름을 읽고, PID와 함께 출력한다.
- ps 명령어처럼 현재 실행 중인 프로세스 목록을 확인할 수 있다.

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi ps.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o ps ps.c
ubuntu@ip-172-31-41-56:~/c_File$ ./ps
PID CMD
1 systemd
2 kthreadd
3 pool_workqueue_release
4 kworker/R-rcu_g
5 kworker/R-rcu_p
6 kworker/R-slub
7 kworker/R-netns
9 kworker/0:0H-events_highpri
12 kworker/R-mm_pg
13 rcu_tasks_rude_kthread
14 rcu_tasks_trace_kthread
15 ksoftirqd/0
16 rcu_sched
17 migration/0
18 idle_inject/0
19 cpuhp/0
20 cpuhp/1
21 idle_inject/1
22 migration/1
23 ksoftirqd/1
25 kworker/1:0H-events_highpri
26 kdevtmpfs
27 kworker/R-lsm_
29 kauditd
31 kthundtaskd
32 oom_reaper
34 kworker/R-write
35 kcompactd0
36 ksmd
37 khugepaged
38 kworker/R-blkio
39 kworker/R-blkio
40 kworker/R-blkcg
41 irq/9-acpi
42 kworker/R-tpm_d
43 kworker/R-ata_s
44 kworker/R-md
45 kworker/R-md_b1
46 kworker/R-edac-
47 kworker/R-devic
48 watchdogd
49 kworker/1:1H-kblockd
50 kswapd0
51 ecryptfs-kthread
52 kworker/R-kthre
53 kworker/R-acpi_
54 kworker/R-nvme-
55 kworker/R-nvme-
56 kworker/R-nvme-
57 kworker/R-nvme-
59 kworker/R-mld
60 kworker/R-ipw6
67 kworker/R-katrp
69 kworker/u5:0
82 kworker/R-charg
```

50. hostname

명령어

- hostname: 호스트 이름 출력

```
#include <stdio.h>
#include <unistd.h>    // gethostname
#include <limits.h>    // HOST_NAME_MAX
```

```
int main() {
    char hostname[HOST_NAME_MAX + 1]; // 널 문자 포함 공간 확보

    if (gethostname(hostname, sizeof(hostname)) == 0) {
        printf("%s\n", hostname);
    } else {
        perror("gethostname 오류");
        return 1;
    }

    return 0;
}
```

설명

- HOST_NAME_MAX: limits.h 헤더에 정의된 상수이며 1을 더하는 이유는 문자열의 끝에 있는 널 문자(\0) 때문이다.
※크기는 리눅스에서는 64
- gethostname(char *name, size_t len): hostname을 반환하는 함수

컴파일 및 실행화면

```
ubuntu@ip-172-31-41-56:~/c_File$ vi hostname_c.c
ubuntu@ip-172-31-41-56:~/c_File$ gcc -o hostname_c hostname_c.c
ubuntu@ip-172-31-41-56:~/c_File$ ./hostname_c
ip-172-31-41-56
ubuntu@ip-172-31-41-56:~/c_File$
```

※ hostname이 ip주소가 뜨는이유는 AWS를 사용하면 hostname이 기본적으로 ip주소로 만들어지기 때문이다.

명령어 점수 : 15

- 기간에 맞춰서 잘 제출했습니다.
- 50개 명령어를 모두 c언어로 작성했습니다.