IT@COOKBOOK

시스템 프로그래밍

리눅스&유닉스

Chapter 02 디렉터리 다루기

목차

학습목표

- 리눅스 디렉터리와 파일의 특징을 이해한다.
- 함수를 사용해 디렉터리를 생성하고 삭제할 수 있다.
- 함수를 사용해 디렉터리를 관리할 수 있다.
- 함수를 사용해 디렉터리의 내용을 읽을 수 있다.

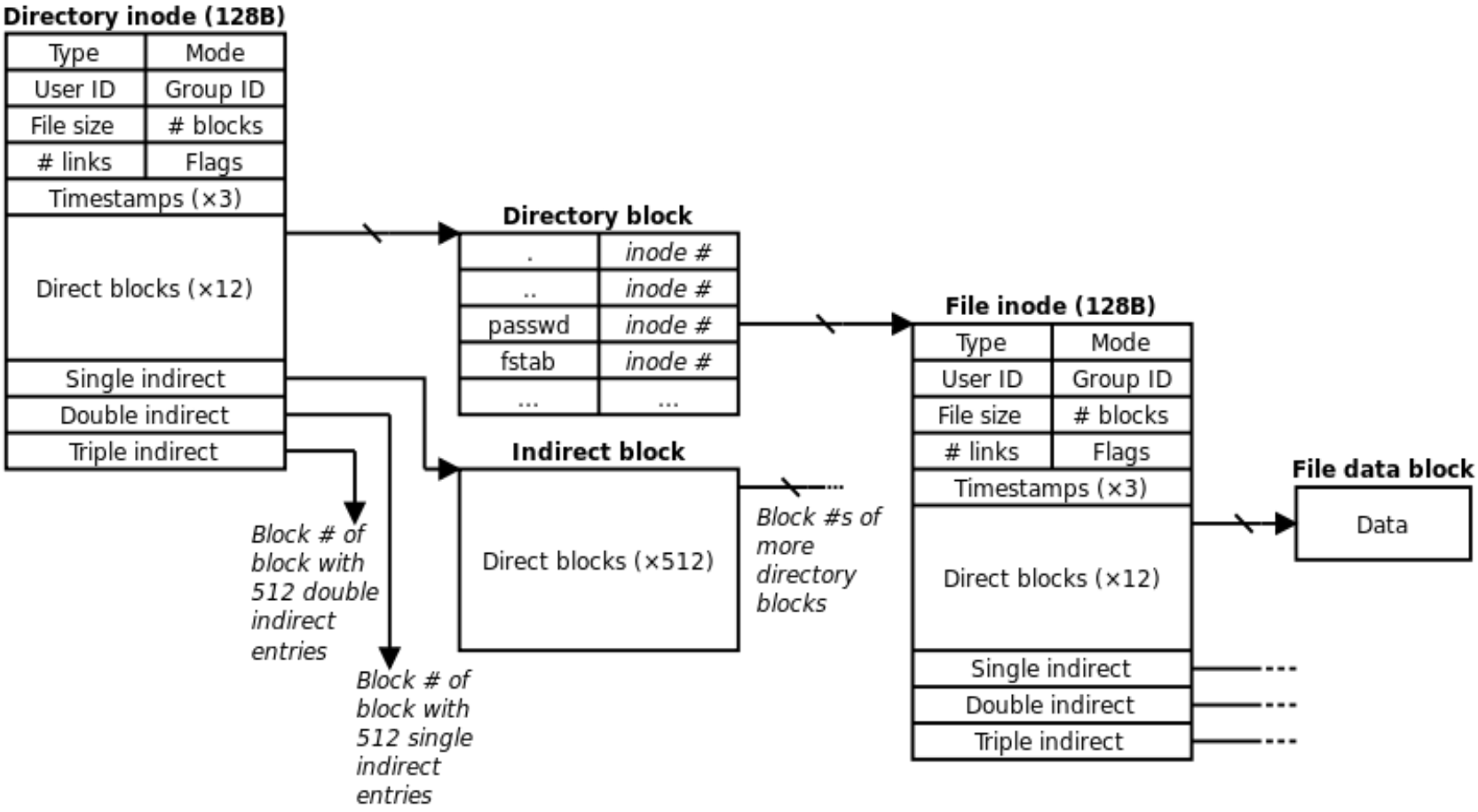
01. 개요

■ 리눅스와 디렉터리

- 리눅스의 파일 구분
  - 리눅스에서는 파일을 일반 파일과 특수 파일, 디렉터리로 구분
  - 디렉터리는 해당 디렉터리에 속한 파일을 관리하는 특별한 파일

- 리눅스의 파일 구성
  - 파일명: 사용자가 파일에 접근할 때 사용
  - Inode: 파일의 소유자나 크기 등의 정보와 실제 데이터를 저장하고 있는 데이터 블록의 위치를 나타내는 주소들이 저장
  - 데이터 블록: 실제로 데이터가 저장되는 하드디스크의 공간

Inode 구조


Directory inode (128B)
Type
Mode
User ID
Group ID
File size
blocks
links
Flags
Timestamps (×3)
Direct blocks (×12)
Single indirect
Double indirect
Triple indirect
Directory block
.
inode #
..
inode #
passwd
inode #
fstab
inode #
...
...
Indirect block
Direct blocks (×512)
Block #s of more directory blocks
Block # of block with 512 double indirect entries
Block # of block with 512 single indirect entries
File inode (128B)
Type
Mode
User ID
Group ID
File size
blocks
links
Flags
Timestamps (×3)
Direct blocks (×12)
Single indirect
Double indirect
Triple indirect
File data block
Data

01. 개요

■ 리눅스와 디렉터리

- 리눅스의 함수

표 2-1 디렉터리 생성과 삭제 함수

기능	함수
디렉터리 생성	int mkdir(const char *pathname, mode_t mode);
디렉터리 삭제	int rmdir(const char *pathname);

표 2-2 디렉터리 관리 함수

기능	함수
현재 위치 확인	char *getcwd(char *buf, size_t size);
	char *get_current_dir_name(void);
디렉터리명 변경	int rename(const char *oldpath, const char *newpath);
디렉터리 이동	int chdir(const char *path);
	int fchdir(int fd);

표 2-3 디렉터리 읽기 함수

기능	함수
디렉터리 열기	DIR *opendir(const char *name);
디렉터리 닫기	int closedir(DIR *dirp);
디렉터리 내용 읽기	struct dirent *readdir(DIR *dirp);
디렉터리 오프셋	long telldir(DIR *dirp);
	void seekdir(DIR *dirp, long loc);
	void rewinddir(DIR *dirp);

02. 리눅스 파일의 특징

■ 파일의 종류

- 파일의 종류
  - 크게 일반 파일, 특수 파일, 디렉터리로 구분

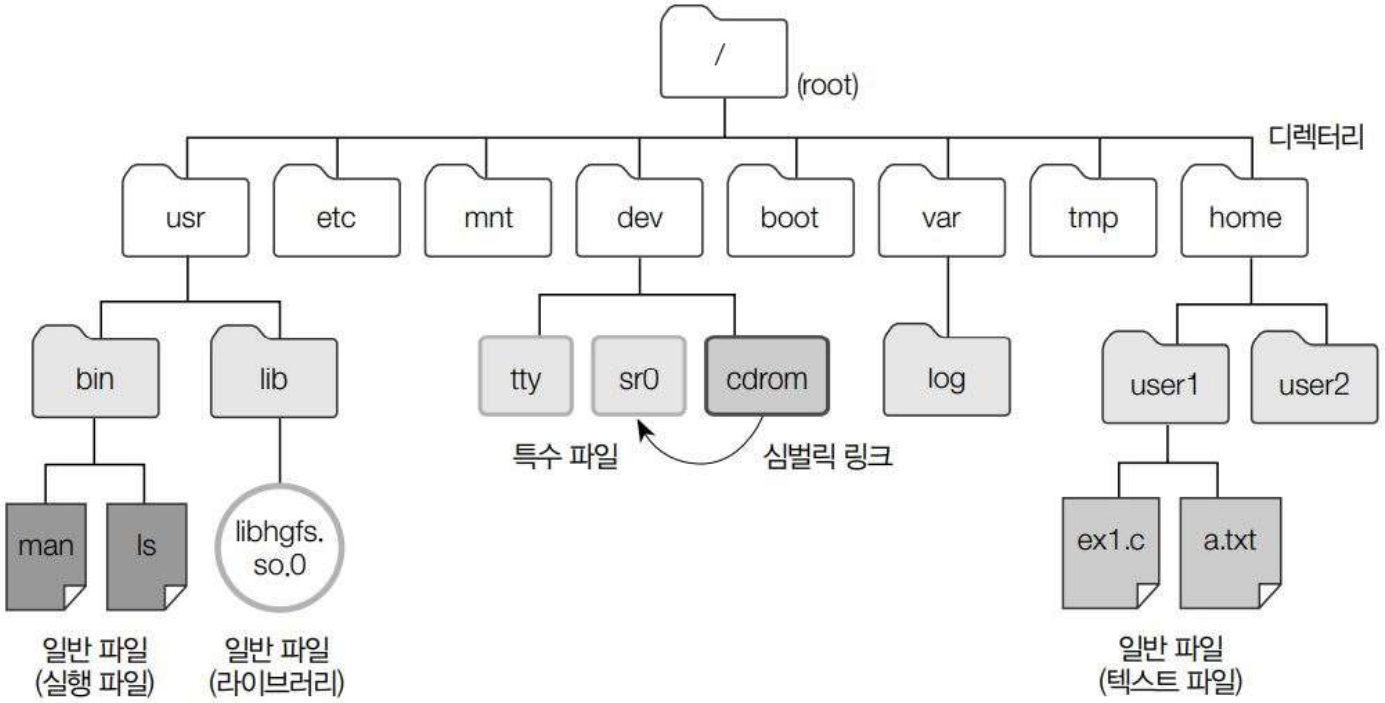


그림 2-1 리눅스 파일의 종류

02. 리눅스 파일의 특징

■ 파일의 종류

- 일반 파일
  - 텍스트 파일, 실행 파일, 라이브러리, 이미지 등 리눅스에서 사용하는 대부분의 파일이 일반 파일에 해당
  - 데이터 블록에 텍스트나 바이너리 형태의 데이터를 저장하고 있음
  - vi 같은 편집기를 사용해 만들기도 하고 컴파일러나 다른 응용 프로그램에서 생성할 수 도 있음

```
$ ls -l /usr/bin
합계 177456
lrwxrwxrwx 1 root root        11 1월 27 16:12 GET -> lwp-request
lrwxrwxrwx 1 root root        11 1월 27 16:12 HEAD -> lwp-request
lrwxrwxrwx 1 root root        11 1월 27 16:12 POST -> lwp-request
-rwxr-xr-x 1 root root    141856 6월 22 2020 VGAuthService
lrwxrwxrwx 1 root root         4 1월 17 18:13 X -> Xorg
lrwxrwxrwx 1 root root         1 1월 27 16:12 X11 -> .
-rwxr-xr-x 1 root root   2434568 1월 17 18:13 Xephyr
-rwxr-xr-x 1 root root       274 1월 17 18:13 Xorg
-rwxr-xr-x 1 root root   2324456 1월 17 18:13 Xwayland
(생략)
```

02. 리눅스 파일의 특징

■ 파일의 종류

- 특수 파일
  - 리눅스에서 통신을 하거나 터미널 또는 디스크 등의 장치를 사용할 때 연관된 특수 파일을 이용
  - 장치 사용하지 않는 대신 장치의 종류를 나타내는 장치 번호를 inode에 저장
  - 장치 관련 특수 파일을 다른 파일과 구분해 장치 파일이라고도 함
  - 데이터 블록을 사용하지 않으며, 블록 장치 파일과 문자 장치 파일이 있음
    - 블록 장치 파일은 블록 단위로 데이터를 읽고 씀
    - 문자 장치 파일은 하드 디스크인 경우 섹터 단위로 읽고 씀

```
$ ls -l /dev
합계 0
crw------- 1 root root      10, 175     2월 14 20:19 agpgart
crw-r--r-- 1 root root      10,235     2월 14 20:19 autofs
drwxr-xr-x 2 root root         360     2월 14 20:19 block
drwxr-xr-x 2 root root          80     2월 14 20:19 bsg
crw------- 1 root root      10, 234     2월 14 20:19 btrfs-control
drwxr-xr-x 3 root root          60     2월 14 20:19 bus
lrwxrwxrwx 1 root root           3     2월 14 20:19 cdrom -> sr0
lrwxrwxrwx 1 root root           3     2월 14 20:19 cdrw -> sr0
drwxr-xr-x 2 root root        3720     2월 14 20:19 char
crw--w---- 1 root tty         5, 1     2월 14 20:20 console
(생략)
```

02. 리눅스 파일의 특징

■ 파일의 종류

- 디렉터리
  - 리눅스에서는 디렉터리도 파일로 취급
  - 디렉터리와 연관된 데이터 블록은 해당 디렉터 리에 속한 파일의 목록과 inode를 저장
  - 디렉터리를 생성하려면 mkdir, 삭제하려면 rmdir 또는 rm –r, 복사하려면 cp -r 명령을 사용

```
$ ls -l /
합계 1190428
lrwxrwxrwx   1  root  root          7 1월 27 16:12 bin -> usr/bin
drwxr-xr-x   4  root  root       4096 2월 11 17:02 boot
drwxrwxr-x   2  root  root       4096 1월 27 16:15 cdrom
drwxr-xr-x  19  root  root       4180 2월 14 20:19 dev
drwxr-xr-x 130  root  root      12288 2월 20 09:26 etc
drwxr-xr-x   3  root  root       4096 1월 27 16:18 home
(생략)
```

02. 리눅스 파일의 특징

■ 파일의 종류

- 파일의 종류 구분
  - ls -l 명령을 사용하면 파일의 종류를 알 수 있음

```
$ ls -l /usr/bin/cp
-rwxr-xr-x 1 root root 153976 9월 5 2019 /usr/bin/cp
```

  - 명령의 결과 중 파일의 권한을 표시하는 부분인 -rwxr-x-x에서 맨 앞의 하이픈(-)이 파일의 종류를 나타냄

표 2-4 파일의 종류 식별 문자

문자	파일의 종류
-	일반 파일
d	디렉터리
b	블록 장치 특수 파일
c	문자 장치 특수 파일
l	심벌릭 링크

```
$ ls -l /dev
합계 0
(생략)
drwxr-xr-x   2 root  root         60  2월 14 20:19 lightnvm
lrwxrwxrwx   1 root  root         28  2월 14 20:19 log -> /run/systemd/journal/dev-log
crw-rw----   1 root  disk   10, 237  2월 14 20:19 loop-control
brw-rw----   1 root  disk      7, 0  2월 14 20:19 loop0
(생략)
```

02. 리눅스 파일의 특징

■ 파일의 구성 요소

- 파일명
  - 사용자가 파일에 접근할 때 사용하며 파일명과 관련된 inode가 반드시 있어야 함
  - 유닉스에서는 예전에는 시스템 파일명으로 최대 14자까지 사용할 수 있었지만, 현재는 255바이트까지 사용할 수 있음
  - 리눅스에서는 255바이트보다 긴 파 일명도 사용할 수 있음
  - 파일명이나 디렉터리명은 /와 null 문자를 제외하고 아무 문자나 사용할 수 있음
  - 그러나 출력이 가능 한 문자를 사용하고 혼동을 줄 수 있는 특수문자는 사용을 자제하는 것이 관례

  - 파일명 지정 주의 사항
    - 파일명과 디렉터리명에 사용하는 알파벳은 대소문자를 구분
    - 파일명과 디렉터리명이 '.'으로 시작하면 숨김 파일로 간주

02. 리눅스 파일의 특징

■ 파일의 구성 요소

- inode
  - 파일에 대한 정보를 저장하고 있는 객체로, 실제로 디스크에 저장되어 있음
  - 리눅스 커널의 입장에서는 파일의 정보를 관리하는 자료 구조로 사용
  - inode는 외부적으로는 번호로 표현하며, 내부적으로는 두 부분으로 나누어 정보를 저장

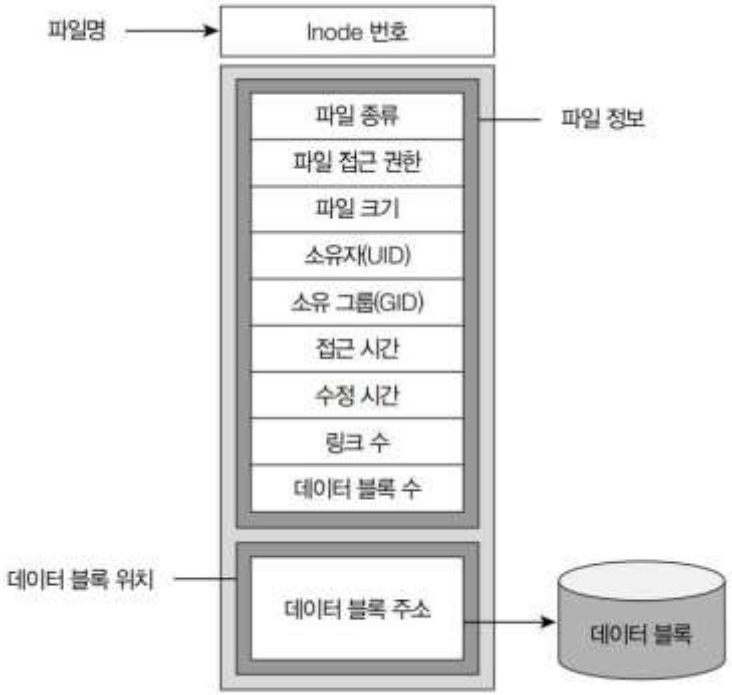


그림 2-2 inode의 구조

02. 리눅스 파일의 특징

■ 파일의 구성 요소

- inode
  - inode의 첫 번째 부분에는 파일에 관한 정보가 저장
    - (파일 종류, 파일 접근 권한, 파일 크기, 소유자, 소유 그룹, 파일 변경 시각, 하드 링크 수, 데이터 블록 수 등)
  - ls –l 명령은 inode의 정보를 읽어서 출력
  - 두 번째 부분에는 파일의 실제 데이터가 저장되어 있는 데이터 블록의 위치를 나타내는 주소들이 저장
  - 파일의 inode 번호는 ls –i 명령으로 알 수 있음

```
$ ls -i
1048595 src    1073157 다운로드    1073156 바탕화면    1073162 사진    1073158 템플릿
1073159 공개   1073160 문서        1073163 비디오      1073161 음악
```

- 데이터 블록
  - 실제로 데이터가 저장되는 하드 디스크의 공간
  - 일반 파일이나 디렉터리, 심벌릭 링크는 데이터 블록에 관련 내용을 직접 저장
  - 장치 파일은 데이터 블록을 사용하지 않고 장치에 관한 정보를 inode에 저장

03. 디렉터리 생성과 삭제

■ 디렉터리 생성 : mkdir(2)

```
#include <sys/stat.h>                                    [함수 원형]
#include <sys/types.h>

int mkdir(const char *pathname, mode_t mode);
```

- pathname : 디렉터리가 포함된 경로
- mode : 접근 권한

- mkdir()의 특징
  - mkdir() 함수는 생성하려는 디렉터리명을 포함한 경로를 받고, 생성하는 디렉터리의 기본 접근 권한을 지정
  - mkdir() 함수는 수행에 성공하면 0을, 실패하면 –1을 리턴

03. 디렉터리 생성과 삭제

■ [예제 2-1] 디렉터리 생성하기

```
01 #include <sys/stat.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     if (mkdir("han", 0755) == -1) {
07         perror("han");
08         exit(1);
09     }
10 }
```

실행

```
$ ch2_1.out
jw@jw09:~/src/ch2$ ls
ch2_1.c  ch2_1.out  han
```

- **06행** 접근 권한을 755로 지정해 han 디렉터리를 생성
- **실행 결과** han 디렉터리가 생성

03. 디렉터리 생성과 삭제

■ [예제 2-2] 디렉터리 삭제하기

```
01 #include <unistd.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     if (rmdir("han") == -1) {
07         perror("han");
08         exit(1);
09     }
10 }
```

실행

```
$ ls
ch2_1.c  ch2_2.c  ch2_2.out  han
$ ch2_2.out
$ ls
ch2_1.c  ch2_2.c  ch2_2.out
```

- **06행** 현재 디렉터리에서 han 디렉터리를 찾아 삭제
- **실행 결과** han 디렉터리가 삭제

03. 디렉터리 생성과 삭제

■ 현재 작업 디렉터리의 위치 검색 1 : getcwd(3)

```
#include <sys/stat.h>                                    [함수 원형]

char *getcwd(char *buf, size_t size);
```

- buf : 현재 디렉터리의 절대 경로를 저장할 버퍼 주소
- size : 버퍼의 크기

- getcwd()의 특징
  - getcwd() 함수는 경로를 저장할 버퍼의 주소와 버퍼의 크기를 인자로 받음
  - 인자를 지정하는 방법
    ① buf에 경로를 저장할 만큼 충분한 메모리를 할당하고 그 크기를 size에 지정
    ② buf에 NULL을 지정하고 할당이 필요한 메모리 크기를 size에 지정
    ③ buf에 NULL을 지정하고 size는 0으로 지정

04. 디렉터리 관리

■ [예제 2-3] 현재 디렉터리의 위치 검색하기 1

```
01 #include <unistd.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     char *cwd;
07     char wd1[BUFSIZ];
08     char wd2[10];
09
10     getcwd(wd1, BUFSIZ);
11     printf("wd1 = %s₩n", wd1);
12
13     cwd = getcwd(NULL, BUFSIZ);
14     printf("cwd1 = %s₩n", cwd);
15     free(cwd);
16
17     cwd = getcwd(NULL, 0);
18     printf("cwd2 = %s₩n", cwd);
19     free(cwd);
20
21     if(getcwd(wd2, 10) == NULL) {
22         perror("getcwd");
23         exit(1);
24     }
25 }
```

실행

```
$ ch2_3.out
wd1 = /home/jw/src/ch2
cwd1 = /home/jw/src/ch2
cwd2 = /home/jw/src/ch2
getcwd: Numerical result out of range
```

04. 디렉터리 관리

■ [예제 2-3] 현재 디렉터리의 위치 검색하기 1 (test2.c)

```
01 #include <unistd.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     char *cwd;
07     char wd1[BUFSIZ];
08     char wd2[10];
09
10     getcwd(wd1, BUFSIZ);
11     printf("wd1 = %s₩n", wd1);
12
13     cwd = getcwd(NULL, BUFSIZ);
14     printf("cwd1 = %s₩n", cwd);
15     free(cwd);
16
17     cwd = getcwd(NULL, 0);
18     printf("cwd2 = %s₩n", cwd);
19     free(cwd);
20
21     if(getcwd(wd2, 10) == NULL) {
22         perror("getcwd");
23         exit(1);
24     }
25 }
```

- **06행** 시스템이 할당하는 버퍼의 주소를 저장하기 위한 포인터 변수
- **07행** 경로를 저장하기 위한 배열로, 크기는 BUFSIZ
  BUFSIZ는 stdio.h 파일에 8192로 정의되어 있음
  따라서 wd1은 8KB 크기의 배열
- **08행** 경로를 저장하기 위한 배열로, 크기는 10바이트
- **10~11행** wd1 배열에 현재 경로를 저장하고 이를 출력
- **13~15행** cwd 포인터에 BUFSIZ만큼 메모리를 할당하고 이 메모리에 경로를 저장
  15행에서는 free() 함수를 사용해 메모리를 해제
- **17~19행** getcwd() 함수의 인자로 NULL과 0을 지정
  이 경우 시스템이 자동으로 경로에 필요한 메모리를 할당하고 주소를 리턴
  19행에서 사용이 끝난 메모리를 해제
- **21~24행** getcwd() 함수의 인자로 저장할 경로보다 크기가 작은 버퍼를 지정
  getcwd()함수 처리에서 오류가 발생하면 NULL을 리턴
- **실행 결과** 11행, 14행, 18행에서는 경로를 정상적으로 출력
  22행에서 오류 메시지를 출력
  오류 메시지는 결과가 메모리의 범위를 벗어났다는 뜻

04. 디렉터리 관리

■ 현재 작업 디렉터리 위치 검색 2 : get_current_dir_name(3)

```
#include <unistd.h>                                    [함수 원형]

char *getcwd(char *buf, size_t size);
```

- void : 함수로 전달할 인자가 없다는 뜻

- get_current_dir_name() 의 특징
  - 현제 디렉터리의 절대 경로를 리턴
  - 인자로 아무것도 전달하지 않으며, 시스템이 메모리를 자동으로 할당해 경로를 저장하고 리턴

04. 디렉터리 관리

■ [예제 2-4] 현재 디렉터리의 위치 검색하기 2 (test3.c)

```
01 #define _GNU_SOURCE
02 #include <unistd.h>
03 #include <stdlib.h>
04 #include <stdio.h>
05
06
07 int main() {
08     char *cwd;
09
10     cwd = get_current_dir_name();
11     printf("cwd = %s₩n", cwd);
12     free(cwd);
13 }
```

실행

```
$ ch2_4.out
cwd = /home/jw/src/ch2
```

- **01행** #define문을 사용해 _GNU_SOURCE를 정의
- **10행** get_current_dir_name() 함수를 사용해 경로를 검색
  시스템이 메모리를 자동으로 할당하고 경로를 저장해 리턴한다.
- **11행** 10행에서 리턴한 경로를 출력한다.
- **12행** 메모리 사용이 끝났으므로 free() 함수로 메모리를 해제

04. 디렉터리 관리

■ 디렉터리명 변경 : rename(2)

```
#include <stdio.h>                                    [함수 원형]

int rename(const char *oldpath, const char *newpath);
```

- oldpath : 변경할 파일/디렉터리명
- newpath : 새 파일/디렉터리명

- rename()의 특징
  - 만약 두 번째 인자로 지정한 이름이 이미 있으면 해당 디렉터리를 삭제
  - 실행 도중 오류가 발생하면 원본과 새로운 디렉터리명이 모두 남음
  - rename() 함수는 수행에 성공하면 0을, 실패하면 -1을 리턴함
  - rename() 함수는 파일명을 변경하는 데도 사용할 수 있음
  - rename() 함수의 매뉴얼을 보려면 man –s 2 rename을 이용해야 함

04. 디렉터리 관리

■ [예제 2-5] 디렉터리명 변경하기

```
01 #include <sys/stat.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     if (rename("han", "bit") == -1) {
07         perror("rename");
08         exit(1);
09     }
10 }
```

실행

```
$ ls
ch2_5.c   ch2_5.out   han
$ ch2_5.out
$ ls
bit   ch2_5.c   ch2_5.out
```

- **06행** 디렉터리명을 han에서 bit로 변경
- **실행 결과** 디렉터리명이 han에서 bit로 바뀐 것을 알 수 있음

04. 디렉터리 관리

■ 디렉터리 이동 1 : chdir(2)

```
#include <unistd.h>                                    [함수 원형]

int chdir(const char *path);
```

- path: 이동하려는 디렉터리 경로

- chdir()의 특징
  - 이동하려는 디렉터리의 경로를 인자로 받으며, 절대 경로와 상대 경로 모두 사용할 수 있음
  - chdir() 함수는 수행에 성공하면 0을, 실패하면 -1을 리턴

04. 디렉터리 관리

■ [예제 2-6] 디렉터리 이동하기 1 (test4.c)

```
01 #include <unistd.h>
02 #include <stdio.h>
03 #include <stdlib.h>
04
05 int main() {
06     char *cwd;
07
08     cwd = getcwd(NULL, BUFSIZ);
09     printf("1.Current Directory: %s₩n", cwd);
10
11     chdir("bit");
12
13     cwd = getcwd(NULL, BUFSIZ);
14     printf("2.Current Directory: %s₩n", cwd);
15
16     free(cwd);
17 }
```

실행

```
$ ch2_6.out
1.Current Directory: /home/jw/src/ch2
2.Current Directory: /home/jw/src/ch2/bit
$ pwd
/home/jw/src/ch2
```

- **08~09행** 현재 디렉터리의 경로를 getcwd() 함수로 읽어 출력
- **11행** bit 디렉터리로 이동
- **13~14행** 현재 디렉터리의 경로를 getcwd() 함수로 읽어 출력
- **실행 결과** 디렉터리가 이동, 그런데 디렉터리 이동은 프로그램 내부에서만 진행된 것으로 pwd 명령으로 확인하면 현재 디렉터리가 바뀐 것은 아님을 알 수 있음

04. 디렉터리 관리

■ 디렉터리 이동 2 : fchdir(2)

```
#include <unistd.h>                                    [함수 원형]

int fchdir(int fd);
```

- fd : 이동하려는 디렉터리의 파일 디스크립터

- chdir()의 특징
  - 파일 디스크립터를 인자로 받음 파일
    - 파일 디스크립터: open()함수로 디렉터리를 열고 돌려받는 것
  - fchdir() 함수를 사용하려면 open() 함수로 해당디렉터리를 먼저 열어야 함
  - fchdir() 함수는 수행에 성공하면 0을, 실패하면 –1을 리턴

04. 디렉터리 관리

■ [예제 2-7] 디렉터리 이동하기 2 (test5.c)

```
01 #include <fcntl.h>
02 #include <unistd.h>
03 #include <stdio.h>
04 #include <stdlib.h>
05
06 int main() {
07     char *cwd;
08     int fd;
09
10     cwd = getcwd(NULL, BUFSIZ);
11     printf("1.Current Directory: %s₩n", cwd);
12
13     fd = open("bit", O_RDONLY);
14     fchdir(fd);
15
16     cwd = getcwd(NULL, BUFSIZ);
17     printf("2.Current Directory: %s₩n", cwd);
18
19     close(fd);
20     free(cwd);
21 }
```

실행

```
$ ch2_9.out
Start Position : 0
Read : ch2_9_1.c -> Cur Position : 216790469681228906
Read : ch2_9.out -> Cur Position : 924654860703974355
Read : ch2_2.c -> Cur Position : 3034506938176788278
(생략)
** Directory Position Rewind **
Cur Position : 0
** Move Directory Pointer **
Cur Position : 216790469681228906
Read : ch2_9.out
```

- **01행** O_RDONLY는 fcntl.h 파일에 정의되어 있음
- **10~11행** 현재 디렉터리의 경로를 getcwd() 함수로 읽어 출력
- **13행** open() 함수로 bit 디렉터리를 실행
- **14행** fchdir() 함수를 사용해 디렉터리를 이동
  open() 함수가 리턴한 파일 디스크립터를 인자로 지정
- **16~17행** 현재 디렉터리의 경로를 getcwd() 함수로 읽어 출력
- **19~20행** close() 함수로 열린 디렉터리를 닫고 free() 함수로 메모리를 해제
- **실행 결과** 디렉터리가 이동, pwd 명령으로 확인해보면 프로그램 내부에서만 디렉터리가 바뀐 것을 알 수 있음

05. 디렉토리 내용 읽기

■ 디렉토리 열기 : opendir(3)

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(Const char *name);
```

■ 디렉토리 닫기 : closedir(3)

```
#include <sys/types.h>
#include <dirent.h>
Int closedir(DIR *dirp);
```

■ 디렉토리 내용 읽기 : readdir(3)

```
#include <dirent.h>

Struct dirent *readdir(DIR *dirp)
```

```
typedef struct dirent {
    ino_t           d_ino;
    off_t           d_off;
    unsigned short  d_reclen;
    unsigned char   d_type;
    char            d_name[256];
} dirent_t;
```

05. 디렉토리 내용 읽기

- **test6.c**
- **디렉토리의 내용을 읽는 위치 변경하기 : telldir/seekdir/rewinddir(3) (test7.c)**

```
#include <sys/types.h>
#include <dirent.h>

long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
void rewinddir(DIR *dirp);
```

시스템 프로그래밍

리눅스&유닉스

감사합니다.