HALLYM UNIVERSITY

CIC Convergence Information & Communications

(Operating System) Practice -4-

Make & File I/O

Index

I. Make

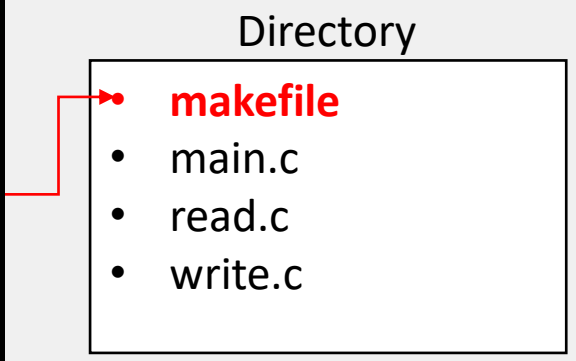
II. File I/O

Make

- **Make**
  - Compile 자동화 Tool
  - 여러 파일들간 의존성 관계, 필요한 명령어를 정의함
  - 여러 개의 소스코드 파일을 사용해서 최종 프로그램을 만드는 과정을 문법으로 표현
  - 최종 프로그램을 구성하는 소스파일 & makefile로 구성
    - ✓ makefile: 최종 프로그램 구성 방법을 정의하는 텍스트파일

Example

```
main: main.o read.o write.o
    gcc -W -Wall main.o read.o write.o -o main

main.o:main.c
    gcc -W -Wall -c main.c

read.o:read.c
    gcc -W -Wall -c read.c

write.o:write.c
    gcc -W -Wall -c write.c
```



Make

- **makefile 문법**
  - 구성

생성하고자 하는 파일의 이름

dependency 혹은 prerequisites (target 파일을 생성하기 위해 필요한 파일들, 즉 선행파일)

✓ target : [file1] [file2] ……..

TAB "target" 파일을 생성하기 위한 명령어

Example

makefile

```
test : test.c
	gcc test1.c  -o  test
```

test1.c

```
int main(){
	printf("temp make file test!\n");
	return 0;
}
```

실행결과

```
temp makefile test!
```

Make

- **Make 실습**

make_test1.c

```c
#include <stdio.h>

void func1();
void func2();

int main(){
    printf("test1.c has run.\n");
    func1();
    func2();
}
```

makefile

```
test : make_test1.o make_test2.o make_test3.o
    gcc make_test1.o make_test2.o make_test3.o -o make_test
make_test1.o : make_test1.c
    gcc -c make_test1.c
make_test2.o : make_test2.c
    gcc -c make_test2.c
make_test3.o : make_test3.c
    gcc -c make_test3.c
clean :
    rm make_test1.o make_test2.o make_test3.o
```

make_test2.c

```c
#include <stdio.h>

extern void func1(){
    printf("test2.c has run.\n");
}
```

make_test3.c

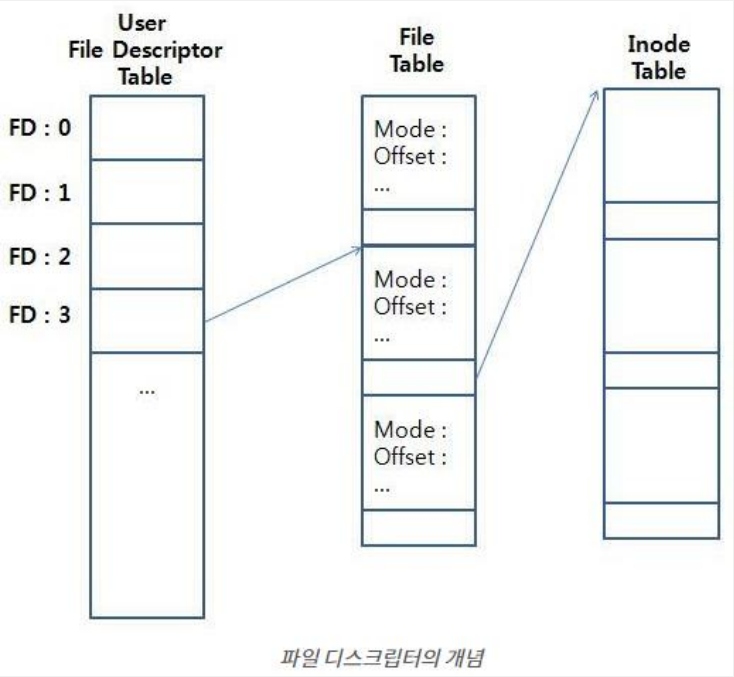
```c
#include <stdio.h>

extern void func2(){
    printf("test3.c has run.\n");
}
```

실행결과

```
ubuntu-pc@ubuntupc-VirtualBox:~/practice4$ make
gcc -c make_test1.c
gcc -c make_test2.c
gcc -c make_test3.c
gcc make_test1.o make_test2.o make_test3.o -o make_test
ubuntu-pc@ubuntupc-VirtualBox:~/practice4$ ./make_test
test1.c has run.
test2.c has run.
test3.c has run.
ubuntu-pc@ubuntupc-VirtualBox:~/practice4$ make clean
rm make_test1.o make_test2.o make_test3.o
ubuntu-pc@ubuntupc-VirtualBox:~/
```

File I/O

- **File Descriptor**
  - 시스템으로부터 할당 받은 파일을 대표하는 0이 아닌 정수 값
  - 프로세스는 File Descriptor(FD) Table 유지
  - FD Table의 각 요소는 File Table로의 포인터를 갖고 있으며, 이를 통해 파일에 접근 가능



File Descriptor Concept

Default FD

FD	목적	stdio 스트림
0	표준입력	stdin
1	표준출력	stdout
2	표준에러	stderr

File I/O

- **Open/Read/Write/Close**
  - Open
    - ✓ 동작: 파일을 열기
    - ✓ 형태: open(const char *FILENAME, int FALGS, mode_t mode)
    - ✓ 인수
      - const char *FILENAME: 열고자 하는 파일의 이름
      - int FALGS: 파일 열기에 대한 옵션
      - mode_t mode: FLAGS에 “O_CREAT” 옵션을 사용할 경우, 생성되는 파일에 접근권한 지정
    - ✓ 반환
      - Int: 정상작동 시 파일의 descriptor를 반환, 비정상 작동시 -1을 반환
    - ✓ <fcntl.h> 헤더에 포함되어 있음

File I/O

- **Open/Read/Write**
  - Open

Option	Description
O_RDONLY	읽기 전용으로 열기
O_WRONLY	쓰기 전용으로 열기
O_RDWR	읽기와 쓰기 모두 가능
O_CREAT	해당 파일이 없다면 생성 파일의 접근권한 설정을 위해 접근 권한 값 추가 필요 open("jwmx", O_WRONLY | O_CREAT, 0644)
O_EXCL	O_CREAT 사용 시, 함께 사용하여 파일이 이미 있을 경우 open()되지 않아 파일 보존 가능 fd = open("./test.txt", O_WRONLY | O_CREAT | O_EXCL, 0644)
O_TRUNC	기존 파일 내용 모두 삭제
O_APPEND	파일을 추가해 쓰기가 가능하도록 open() 후에 쓰기 포인터가 파일 끝에 위치
O_NOCITTY	열기 대상이 터미널일 경우, 프로그램의 제어 터미널에 할당하지 않음
O_NONBLOCK	읽을 내용이 없을 경우, 읽을 내용을 기다리지 않고 바로 복귀
O_SYNC	쓰기를 할 경우, 실제 쓰기가 완료될 때까지 대기 물리적으로 쓰기가 완료될 경우 복귀

File I/O

- **Open/Read/Write**
  - Read
    - ✓ 동작: 파일의 내용을 읽기
    - ✓ 형태: read(int fd, void *buf, size_t nbytes)
    - ✓ 인수
      - int fd: 파일 디스크립터
      - void *buf: 파일을 읽어 들일 버퍼
      - size_t nbytes : 버퍼의 크기
    - ✓ 반환
      - Int: 정상작동 시 읽어들인 바이트의 수를 반환, 비정상 작동시 -1을 반환
    - ✓ <unistd.h> 헤더에 포함되어 있음

File I/O

- **Open/Read/Write**
  - Write
    - ✓ 동작: 파일에 내용을 쓰기
    - ✓ 형태: write(int fd, const void * buf size_t n)
    - ✓ 인수
      - int fd: 파일 디스크립터
      - void *buf: 파일에 쓸 내용을 담은 버퍼
      - size_t n: 쓰기를 수행할 바이트 개수
    - ✓ 반환
      - Int: 정상작동 시 쓰기를 수행한 바이트의 수를 반환, 비정상 작동시 -1을 반환
    - ✓ <unistd.h> 헤더에 포함되어 있음

File I/O

- **Open/Read/Write/Close**
  - Close
    - ✓ 동작: Open() 함으로 열었던 파일의 사용을 중지
    - ✓ 형태: close(int fd)
    - ✓ 인수
      - int fd: 파일 디스크립터
    - ✓ 반환
      - Int: 정상작동 시 0을 반환, 비정상 작동시 -1을 반환
    - ✓ <unistd.h> 헤더에 포함되어 있음

File I/O

- **lseek**
  - lseek
    - ✓ 동작: 함수의 seek pointer (커서)를 조정
    - ✓ 형태: lseek(int fd, off_t offset, int whence)
    - ✓ 인수
      - int fd: 파일 디스크립터
      - off_t offset: 기준점으로 부터 이동할 거리
      - int whence: 기준점 → SEEK_SET 파일 맨 앞 / SEEK_CUR 현재 커서 위치 / SEEK_END 파일 맨 끝
    - ✓ 반환
      - Int: 정상작동 시 Seek point의 위치, 비정상 작동시 -1을 반환

Example

<Read를 수행 이전>
(seek_pointer)*[파일내용]*

<Read를 수행 이후>
*[파일내용]****(seek_pointer)***

File I/O

- **Example Code**

```c
#include <stdio.h> // shell input/output functions
#include <unistd.h> // symbols and constant values used in UNIX
#include <stdlib.h> // general functions for C
#include <fcntl.h> //includes open, write, read
#define BUF_SIZE 1024

int main(){
    int fd, n;
    char buf[BUF_SIZE];
    // OPEN FILE
    fd = open("test.txt", O_CREAT|O_RDWR|O_TRUNC, 0644);
    // O_CREAT: If there is no file, create a file
    // O_RDWR: read and write
    // O_TRUNC: remove the exsting contents in a file
    // 0644: set permission for reading and writing a file
    if(fd < 0){
        printf("filed to open file\n"); exit(1);
    }else{
        printf("Opend fd(%d) file.\n", fd);
    }
    // READ & WRITE
    n = read(0, buf, BUF_SIZE); // read from command
    n = write(fd, buf, n); // write the string into the file
    if(n < 0){
        printf("filed to write file\n"); exit(1);
    }
    lseek(fd, 0, SEEK_SET); // Set a cursor at the first

    n = read(fd, buf, BUF_SIZE); // read from the file
    if(n==0){
        printf("the file is empty\n"); exit(1);
    }
    printf("-----test.txt----\n");
    printf("%s\n", buf); // print the string in buf
    close(fd);
    return(0);
}
```