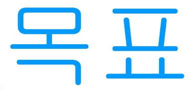
제5장 C 표준 라이브러리



목표

- C 표준 라이브러리의 깊이 있는 이해
- 시스템 호출과 C 표준 라이브러리 관계

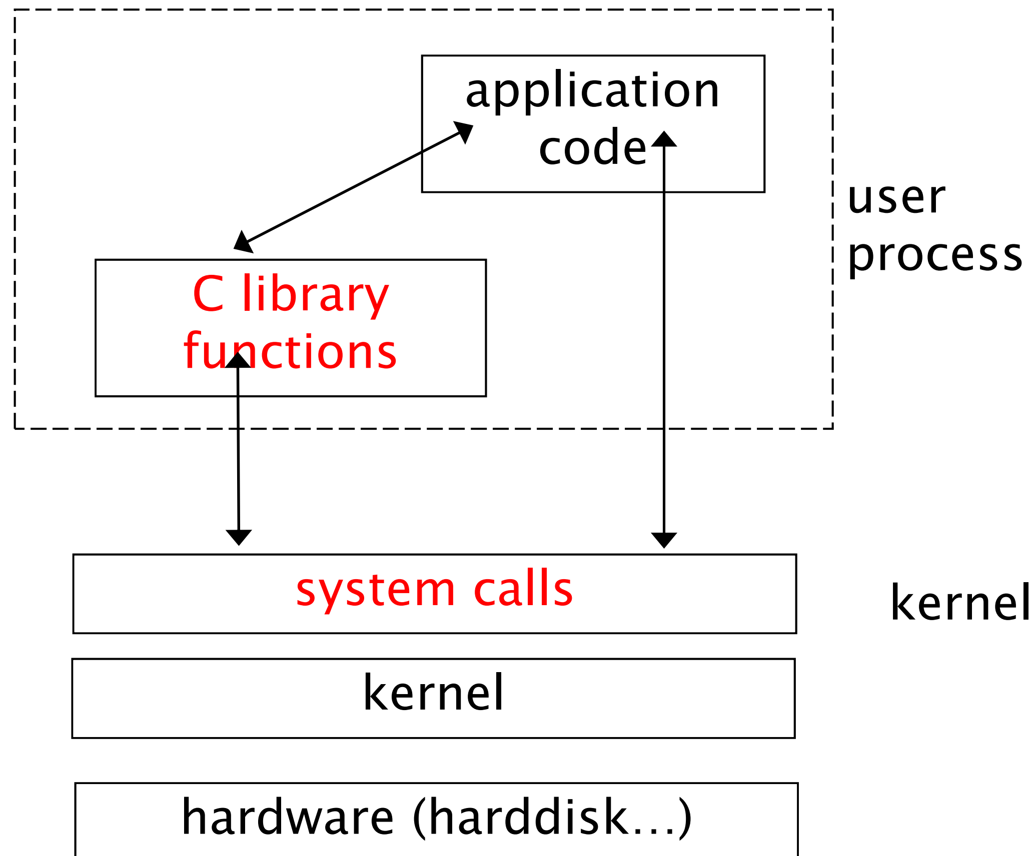
C 입출력 라이브러리 함수

시스템 호출과 라이브러리 함수

- System Calls
  - well defined entry points directly into the kernel
  - documented in section 2 of the UNIX man pages
  - look like C functions which can be called from a user's program – just need to include the appropriate header

- C Library Functions
  - the library function is often more elaborate than the system call, and usually invokes the system call

시스템 호출과 라이브러리 함수



©숙대 창병모

- Written by Dennis Ritchie in 1975
- Implemented on many OS
- ANSI C Standard Library
- Buffer allocation
  - 최적의 크기 단위로 I/O를 수행
  - 디스크 I/O 횟수 최소화
- Streams
  - 문자의 흐름으로 파일 입출력을 다룬다

파일 열기

- **파일 입출력 과정**
  - 파일 열기, 읽기/쓰기, 파일 닫기

- **파일 열기 fopen()**

```
#include <stdio.h>
 FILE    *fp;
 fp=fopen("파일 이름", "입출력방식");
```

fopen 모드

모드	내 용
"r"	읽기 전용으로 연다
"w"	쓰기 전용으로 연다. 파일이 없으면 새로 생성하고 이미 존재하면 그 파일 내용을 삭제
"a"	추가용으로 연다. 파일이 없으면 새로 생성한다.
"r+"	이미 존재하는 파일을 읽기쓰기(갱신)용으로 연다.
"w+"	파일을 생성하고 갱신용으로 연다.
"a+"	파일을 추가용, 갱신용으로 연다. 파일이 없으면 새로 생성한다.
"rb"	이진 파일을 읽기용으로 연다.
"wb"	이진 파일을 쓰기용으로 연다.
"ab"	이진파일을 추가용으로 연다. 파일이 없으면 새로 생성한다.
"rb+"	이미 존재하는 이진파일을 갱신용으로 연다.
"wb+"	이진파일을 생성하고 갱신용으로 연다.
"ab+"	이진 파일을 추가용, 갱신용으로 연다. 파일이 없으면 새로 생성한다.

FILE 구조체

- 파일 관련 시스템 호출 함수
  - 파일 디스크립터 (file descriptor)
- 표준 입출력 함수
  - FILE 구조체에 대한 포인터
- 하나의 스트림을 다루기 위한 정보를 포함하는 구조체
  - 버퍼에 대한 포인터, 버퍼 크기 …
  - 에러 플래그 등
  - 파일 디스크립터 (File descriptor)
- #include <stdio.h>

FILE 구조체

- 열린 파일의 상태를 저장하기 위한 구조체
  - <stdio.h>에 정의되어 있음

```
typedef struct
{
  int           _cnt;          // 버퍼에 남아 있는 문자의 수
  unsigned char *_ptr;         // 버퍼 내에 다음  쓸(읽을) 위치  포인터

  unsigned char *_base;        // 버퍼 시작 주소
  unsigned char  _flag;        //스트림의 현재 상태
                                     _IOFBF, _IOLBF, _IONBUF
                                     _IOEOF, _IOERR _IOREAD, _IOWRT
  unsigned char  _file;        // 파일 디스크립터
} FILE ;
```

FILE 구조체

- FILE * (FILE 구조체에 대한 포인터)
  - 스트림을 열면 (fopen 함수) 리턴됨
  - 열린 파일을 가리키는 FILE 포인터
  - 표준 I/O 함수들의 매개변수로 전달해야 함
  - 프로그래머는 FILE 구조체의 내부를 알 필요 없음

표준 입력/출력/에러

- 표준 I/O 스트림 (stream)
  - 프로그램이 시작되면 자동으로 open되는 스트림
  - stdin, stdout, stderr
  - FILE*
  - #include <stdio.h>

표준 입출력 포인터	설명	가리키는 장치
stdin	표준 입력에 대한 FILE 포인터	키보드
stdout	표준 출력에 대한 FILE 포인터	모니터
stderr	표준 오류에 대한 FILE 포인터	모니터

표준 입출력: 예

```c
1   //*******************************************************************
2   // copy1.c
3   //
4   // 키보드에서 문자를 입력 받아 모니터에 출력
5   //*******************************************************************
6
7   #include <stdio.h>
8
9   int main()
10  {
11      int c;
12
13      c = fgetc(stdin);    // 키보드로부터 입력 받은 문자의 ASCII 코드
14      while (c != EOF) {  // file 끝이 아니면.
15          fputc(c, stdout);    // fp가 가리키는 파일에 문자 c 출력
16          c = fgetc(stdin);    // 키보드로부터 문자를 읽어 c에 저장
17      }
18
19      return 0;
20  }
```

fclose()

```
#include <stdio.h>

int fclose ( FILE *fp );
```

- 스트림을 닫는다
- 리턴 값: 성공하면 0, 실패하면 EOF (-1)
- 출력 버퍼에 있는 모든 자료는 파일에 저장되고, 입력 버퍼에 있는 모든 자료는 버려진다.
- 프로세스가 정상적으로 종료한 경우에는 모든 열려진 스트림이 저절로 닫힌다

표준/파일 입출력 함수

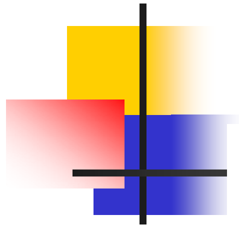
표준 입출력함수	표준 파일 입출력 함수	기능
getchar()	fgetc(), getc()	문자단위로 입력하는 함수
putchar()	fputc(), putc()	문자단위로 출력하는 함수
gets()	fgets()	문자열을 입력하는 함수
puts()	fputs()	문자열을 출력하는 함수
scanf()	fscanf()	자료형에 따라 자료를 입력하는 함수
printf()	fprintf()	자료형에 따라 자료를 출력하는 함수

파일 입출력 함수

- 파일 입력 함수
  - fscanf(), fgets(), fgetc()
  - 열린 파일에서 내용을 읽어 들이는 함수
- 파일 출력 함수
  - fprintf(), fputs(), fputc()
  - 열린 파일에 내용을 기록하는 함수
- 파일 닫기
  - fclose()

파일입출력: 예

```
1   //************************************************************************
2   // copy2.c
3   //
4   // 키보드에서 문자를 입력 받아 파일에 저장
5   //************************************************************************
6
7   #include <stdio.h>
8
9   int main(int argc, char *argv[])
10  {
11      FILE *fp;
12      int c;
13
14      fp = fopen(argv[1],"w");    // 쓰기 전용으로
15      c = getc(stdin);            // 키보드로부터 입력받은 문자의 ASCII코드
16      while (c != EOF) {          // file의 끝이 아니면.
17          putc(c, fp);            // fp가 가리키는 파일에 문자 c저장
18          c = getc(stdin);        // 키보드로부터 문자를 읽어 c에 저장
19      }
20      fclose(fp);
21      printf("%s 파일에 저장 완료.\n“, argv[1]);
22
23      return 0;
24  }
```



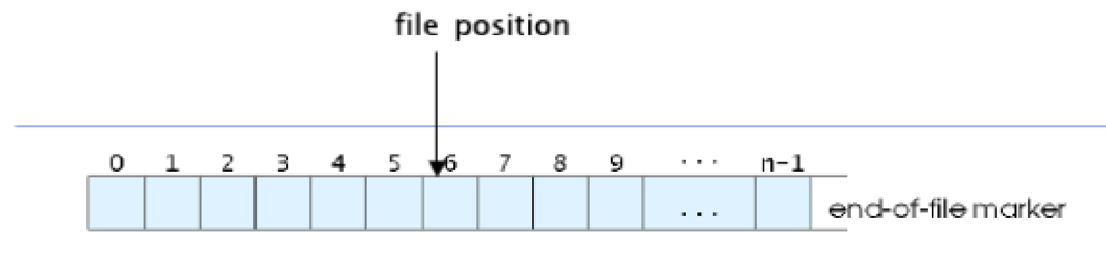
파일 복사 1: 예

```
1   //*********************************************************************
2   // fileCopy1.c
3   //
4   // 파일 복사 프로그램
5   //*********************************************************************
6
7   #include <stdio.h>
8
9   int main(int argc, char *argv[])
10  {
11      char c;
12      FILE *fp1, *fp2;
13
14      fp1 = fopen(argv[1], "r");
15      if (fp1 == NULL) {
16          printf("파일 %s 열기 오류!\n", argv[1]);
17          exit(1);
18      }
19
20      fp2 = fopen(argv[2], "w");
21      while ((c = fgetc(fp1)) != EOF)
22          fputc(c, fp2);
23
24      fclose(fp1);
25      fclose(fp2);
26
27      return 0;
28  }
```

파일 복사 2: 예

```
#include <stdio.h>
int main(argc, argv)
int argc; char* argv[];
{
        FILE *fp1, *fp2;
        char buffer[100];//데이터를 임시로 저장하기위한 배열

        fp1 =fopen(argv[1], “r");
        fp2 =fopen(argv[2], "w");

        if(fp1==NULL)
           printf("file not found");

        /*최대 길이가 100인 문자열을 fp가 가리키는 파일에서 읽어서
          buffer에 저장 후  fp_write가 가리키는 파일에 buffer 내용 기록*/

        while(fgets(buffer, 100, fp1)!=NULL)
            fputs(buffer, fp2);

        fclose(fp1); fclose(fp2);
}
```

파일 위치(file position)



- fseek(FILE *fp, long offset, int mode)
  FILE 포인터 fp가 가리키고 파일의 파일 위치를 모드(mode)가 나타내는 기준점을 기준으로 오프셋(offset)만큼 옮긴다.

- rewind(FILE *fp)
  파일 위치를 파일 시작점에 위치시킴으로써 처음부터 다시 읽을 수 있게 한다.

- ftell(FILE *fp)
  FILE 포인터 fp가 가리키고 있는 파일의 현재 파일 위치를 나타내는 파일 위치 지정자 값을 반환한다.

기호	값	의미
SEEK_SET	0	파일 시작
SEEK_CUR	1	현재 위치
SEEK_END	2	파일 끝

- int fread(void *buf, int size, int n, FILE *fp);

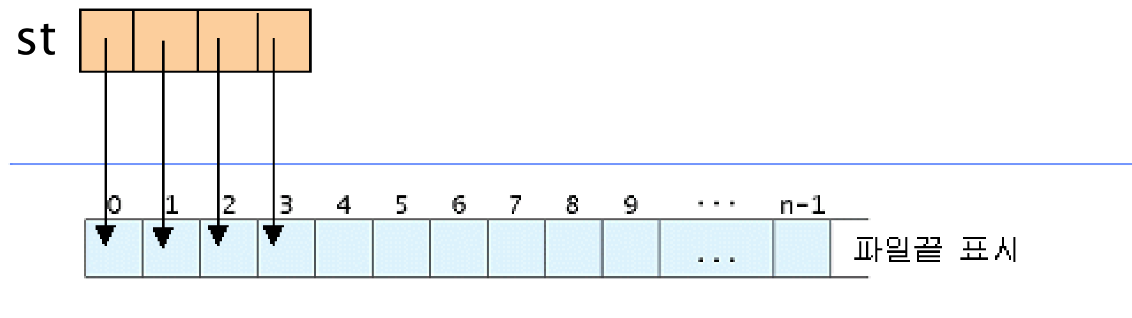
  fp가 가리키는 파일에서 size 크기의 블록을 n개 읽어서 버퍼 포인터 buf가 가리키는 곳에 저장한다. 읽은 블록의 수를 반환한다.

- int fwrite(const void *buf, int size, int n, FILE *fp);

  파일 포인터 fp가 지정한 파일에 버퍼 buf에 저장되어 있는 size 크기의 블록(연속된 바이트)을 n개 기록한다.

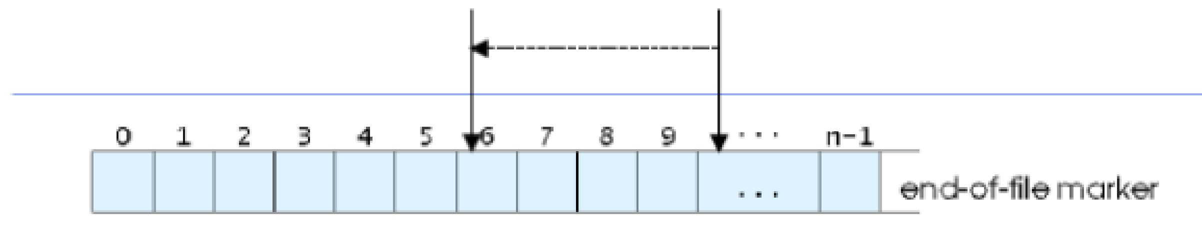
블록 출력: 예

```
struct student st;
FILE *fp = fopen("st_file", "w");

...  /* 입력된 학생 정보를 st에 저장 */

fwrite(&st, sizeof(struct student), 1, fp)
```



블록 수정: 예

```
fread(&st, sizeof(struct student), 1, fp);

        … // 학생 정보 수정

fseek(fp, -sizeof(struct student), SEEK_CUR);
fwrite(&st, sizeof(struct student), 1, fp);
```



파일 복사 3: 예

```
#include <stdio.h>
#define  MAXSIZE 256
int  main(argc, argv)
int argc; char *argv[];
{
    FILE *fpin, *fpout;
    char  buffer[MAXSIZE];
    int   num;

    fpin = fopen(argv[1], "r");
    fpout = fopen(argv[2], "w");
    if(fpin == NULL || fpout == NULL) {
       fprintf(stderr, "파일 열기 오류\n");
       exit(1);
   }
    while((num = fread(buffer, 1, MAXSIZE, fpin)) > 0)  {
       fwrite(buffer, num, 1, fpout);
    }
    fclose(fpin);      fclose(fpout);
    printf("파일 복사 완료\n");
}
```

C library buffer

C library buffer

- C library buffer 사용 목적
  - 디스크 I/O 수행의 최소화
    - read (), write () 함수 호출의 최소화
  - 최적의 크기 단위로 I/O 수행
  - 시스템 성능 향상

- C library buffer 방식
  - fully buffered
  - line buffered
  - unbuffered

C library buffer 방식

- Fully Buffered
  - 버퍼가 꽉 찼을 때 실제 I/O 수행
  - 디스크 파일 입출력
- Line Buffered
  - 줄 바꿈 문자(newline)에서 실제 I/O 수행
  - 터미널 입출력 (stdin, stdout)
- Unbuffered
  - 버퍼를 사용하지 않는다.
  - 표준 에러 (stderr)

©숙대 창병모

```
#include <stdio.h>

void  setbuf (FILE *fp, char *buf);
int   setvbuf (FILE *fp, char *buf, int mode, size_t size);
```

- 버퍼의 관리 방법을 변경한다

- 호출 시기
  - 스트림이 오픈된 후,
  - 입출력 연산 수행 전에 호출되어야 함

setbuf()

void setbuf (FILE **fp*, char **buf*);

- 버퍼 사용을 on/off 할 수 있다.
- *buf* 가 NULL 이면 unbuffered
- *buf* 가 BUFSIZ 크기의 공간을 가리키면 fully/line buffered
  - 터미널 장치면 line buffered
  - 그렇지 않으면 fully buffered

setvbuf()

int setvbuf (FILE **fp*, char **buf*, int *mode*, size_t *size*);

- 버퍼 사용 방법을 변경
- 리턴 값: 성공하면 0, 실패하면 nonzero
- mode
  - _IOFBF : fully buffered
  - _IOLBF : line buffered
  - _IONBF : unbuffered

setvbuf()

- mode == _IONBF
  - *buf* 와 *size* 는 무시됨

- mode == _IOLBF or _IOFBF
  - buf 가 NULL이 아니면
    - buf 에서 *size* 만큼의 공간 사용

  - NULL이면 라이브러리가 알아서 적당한 크기 할당 사용
    - stat 구조체의 st_blksize 크기 할당 (disk files)
    - st_blksize 값을 알 수 없으면 BUFSIZ 크기 할당 (pipes)

예제:setbuf

```
#include <stdio.h>   /* buffer.c */
main()
{
  printf("Hello, "); sleep(1);
  printf("UNIX!"); sleep(1);
  printf("\n"); sleep(1);

  setbuf(stdout, NULL);
  printf("How "); sleep(1);
  printf("are "); sleep(1);
  printf("you?"); sleep(1);
  printf("\n"); sleep(1);
}
```

예제: Print buffering

```
int main( )
{
    FILE *fp;
    fputs(“enter any character\n”, stdout);
    if (getchar() == EOF)
        perror(“getchar error”);
    fputs(“one line to standard error\n”, stderr);

    pr_stdio(“stdin”, stdin);
    pr_stdio(“stdout”, stdout);
    pr_stdio(“stderr”, stderr);

    if ( ( fp = fopen(“/etc/motd”, “r”)) == NULL)
        perror(“fopen error”);
    if (getc(fp) == EOF)
        perror(“getc error”);
    pr_stdio(“/etc/motd”, fp);
    exit(0);
}
```

예제: Print buffering

```
#include <stdio.h>
void pr_stdio(const char *name, FILE *fp)
{
    print("stream = %s ", name);
    if (fp->_flag & _IONBF) printf("unbuffered\n");
    else if (fp->_flag & _IOLBF) printf("line buffered\n");
    else printf("fully buffered\n");
    printf(", buffer size =%d\n", fp->_bufsize);

}
```

©숙대 창병모

fflush()

```
#include <stdio.h>

int  fflush (FILE *fp);
```

- *fp* 스트림의 출력 버퍼에 남아있는 내용을 write() 시스템 호출을 통하여 커널에 전달한다
- 리턴 값 : 성공하면 0, 실패하면 EOF (-1)
- *fp* 가 NULL이면, 모든 출력 스트림의 출력 버퍼에 남아있는 내용을 커널에 전달한다