Linux System Programming #4-1

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

표준 입출력 라이브러리

- **표준 입출력 라이브러리**
- **고수준 입출력 함수**
- **스트림형 구조체**
  - 바이트 단위로 입출력을 하는 것이 아니라 버퍼를 통해 파일 읽고 쓰기 작업
    - 문자 단위, 행 단위, 버퍼 단위 입출력 가능
    - 포맷(형식) 지정 가능

fopen(f), freopen(3), fdopen(3)

```
#include <stdio.h>
FILE *fopen(const char *pathname, const char *mode);
FILE *fdopen(int filedes, const char *mode);
FILE *freopen(const char *pathname, const char *mode, FILE *fp);
```
리턴 값: 성공 시 파일 포인터, 에러 시 NULL을 리턴하고 errno 설정

- **파일과 연관된 표준 입출력 스트림을 오픈하는 라이브러리**
  - fopen() 류 함수 호출이 성공 -> FILE 형 구조체 변수를 생성하고 그 구조체의 포인터(파일 스트림형 포인터)를 리턴
    - pathname과 연관된 FILE 구조체 영역과 파일 입출력 버퍼 확보 -> FILE 구조체 영역의 시작 주소 리턴
    - FILE형 구조체 영역의 멤버 변수는 오픈한 파일을 처리하는데 필요한 각종 정보가 자동 기록
    - 사용자가 FILE형 구조체의 각 멤버 변수를 직접 사용 불가
  - 에러 : 파일이 존재하지 않거나 접근 권한이 없는 경우, 혹은 하나의 프로세스가 오픈할 수 있는 최대 파일 개수를 넘어선 경우
  - fdopen() : 파일 디스크립터를 이용하여 파일 스트림 포인터를 리턴 => 저수준 함수인 open(), creat(), dup(), dup2(), pipe() 등의 호출을 통해 리턴된 파일 디스크립터를 스트림과 연관
  - freopen(): fp 인자로 지정된 파일의 파일 스트림 포인터로 재오픈.
    - freopen()가 호출되면 이미 오픈한 fp를 fclose()를 호출한 것처럼 닫고, pathname 인자로 지정한 파일을 fopen()를 호출한 것처럼 mode 인자에 지정되어 있는 접근 모드로 오픈

fopen(f), freopen(3), fdopen(3) 에서 사용하는 접근 권한

종류	설명
"r" "rb"	파일을 읽기전용으로 오픈. 이 모드로 오픈된 파일에 쓰기를 시도하면 에러가 발생. 만일 지정된 파일이 존재하지 않으면 fopen() 호출 시 NULL을 리턴. open()의 O_RDONLY와 동일한 의미
"w" "wb"	파일을 쓰기전용으로 오픈. 지정한 이름의 파일이 존재하지 않으면 지정한 이름의 파일을 생성. 해당 파일이 이미 존재하면 파일의 길이는 0이 되고, 파일의 처음부터 쓰게 되므로 원래 자료들을 모두 잃게 됨. open()의 O_WRONLY | O_CREAT | O_TRUNC와 동일한 의미
"a" "ab"	파일에 데이터를 추가하기 위해서 오픈. 지정된 이름의 파일이 존재하지 않으면 지정된 이름의 파일을 생성하며 만일 파일이 이미 존재한다면 해당 파일의 포인터가 파일의 맨 끝으로 이동하여 원래 자료들을 그대로 유지하면서 새로운 데이터들을 파일의 마지막에 추가. open()의 O_WRONLY | O_APPEND | O_CREAT와 동일한 의미
"r+" "rb+" "r+b"	파일을 읽기와 쓰기가 가능하도록 오픈. 만일 지정된 파일이 존재하지 않으면 함수의 호출은 실패하고 NULL을 리턴. 해당 파일이 이미 존재해도 원래의 자료는 그대로 유지됨. <<반드시 존재>> open()의 O_RDWR과 동일한 의미
"w+" "wb+" "w+b"	파일을 **읽기와 쓰기가** 가능하도록 오픈. 만일 지정된 파일이 존재하지 않으면 지정된 이름의 파일을 생성하며 만일 파일이 이미 존재한다면 해당 파일의 길이는 0이 되면서 원래의 자료는 모두 잃게 됨. open()의 O_RDWR | O_CREAT | O_TRUNC 와 동일한 의미
"a+" "ab+" "a+b"	파일을 읽기와 추가기능으로 파일을 오픈. 지정된 이름의 파일이 존재하지 않으면 지정된 이름의 파일을 생성하고, 이미 파일이 존재한다면 해당 파일의 포인터가 파일의 맨 끝으로 이동하여 원래 자료들을 그대로 유지하면서 새로운 데이터들을 파일의 마지막에 추가. open()의 O_RDWR | O_APPEND | O_CREAT과 동일한 의미

fopen() 예제

<ssu_test.txt>
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania

<ssu_fopen.c>

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   char *fname = "ssu_test.txt";
   char *mode = "r";
   if (fopen(fname, mode) == NULL) {
      fprintf(stderr, "fopen error for %s\n", fname);
      exit(1);
   }
   else
      printf("Success!\nFilename: <%s>, mode: <%s>\n", fname,
mode);

   exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_fopen

Success!

Filename: <ssu_test.txt>, mode: <r>
```

freopen() 예제

<ssu_freopen.c>

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

int main(void)
{
   char *fname = "ssu_test.txt";
   int fd;

   printf("First printf : Hello, OSLAB!!\n");

  if ((fd = open(fname, O_RDONLY)) < 0) {
      fprintf(stderr, "open error for %s\n", fname);
      exit(1);
   }
   if (freopen(fname, "w", stdout) != NULL)
      printf("Second printf : Hello, OSLAB!!\n");

   exit(0);

}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_freopen

First printf : Hello, OSLAB!!

root@localhost:/home/oslab# cat ssu_test.txt

Second printf : Hello, OSLAB!!
```

fclose(3), fcloseall(3)

```
#include <stdio.h>
int fclose(FILE *fp);
리턴 값: 성공 시 0, 에러 시 EOF를 리턴하고 errno 설정

#define _GNU_SOURCE
#include <stdio.h>
int fcloseall(void);
리턴 값: 성공 시 0, 에러 시 EOF
```

- **오픈된 파일 스트림을 닫는 라이브러리 함수**
- **만약 닫고자 하는 파일 스트림이 출력을 위해 오픈된 스트림**
  - fflush()를 이용해서 버퍼에 남아 있는 데이터를 써야 함
  - 프로세스가 정상적으로 종료되는 경우 입출력 스트림의 버퍼에 있는 데이터들은 모두 방출하고 입출력 스트림들을 모두 다음
- **fcloseall()**
  - stdin, stdout, stderr 모두 닫음
  - 닫기 전에 버퍼에 남아 있는 내용을 모두 스트림에 쓰고, 0을 리턴

fclose() 예제

<ssu_fclose.c>

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

int main(void)
{
   char *fname = "ssu_test.txt";
   FILE *fp;

   if ((fp = fopen(fname, "r")) == NULL) {
      fprintf(stderr, "fopen error for %s\n", fname);
      exit(1);
   }
   else {
      printf("Success!\n");
      printf("Opening \"%s\" in \"r\" mode!\n", fname);
   }

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

실행결과

```
root@localhost:/home/oslab# ./ssu_fclose

Success!

Opening "ssu_test.txt" in "r" mode!
```

setbuf(3), setvbuf(3)

```
#include <stdio.h>
void setbuf(FILE *fp, char *buf);
int setvbuf(FILE *fp, char *buf, int mode, size_t size);
```
리턴 값: 성공 시 0, 에러 시 0이 아닌 값을 리턴하고 errno 설정

- **표준 입출력 라이브러리가 제공하는 버퍼링의 종류를 설정하는 라이브러리 함수**
  - read() 시스템호출과 write() 시스템호출의 횟수 최소화
- **setbuf()**
  - fopen()/ freopen()에 의해 오픈된 파일에서 데이타 입출력 작업이 수행 전에 호출 => 버퍼 사용 방법 설정
  - buf == NULL :버퍼 사용하지 않음
  - Buf <> NULL : 사용자가 정의한 버퍼의 시작주소로 버퍼를 BUFSIZ 만큼의 char 배열을 버퍼로 사용.
    - 시스템이 미리 할당했던 버퍼는 해제
    - fp 인자가 터미널이면 사용자가 buf 인자를 지정하더라도 줄 단위 버퍼.
- **setvbuf()**
  - setbuf()와 기본적으로 동일
  - 차이점) 버퍼링의 타입과 크기 지정 가능
  - buf == NULL :시스템이 할당한 BUFSIZ 크기의 라이브러리 버퍼가 사용
  - Buf <> NULL : 사용자가 정의한 버퍼의 시작주소로 버퍼를 설정하고, size는 buf의 크기로 설정

setbuf(3), setvbuf(3)

- **표준 입출력 라이브러리가 제공하는 버퍼링의 종류를 설정하는 라이브러리 함수**
  - read() 시스템호출과 write() 시스템호출의 횟수를 최소화
- **표준 입출력 라이브러리가 지원하는 버퍼링**
  - full-buffered
    - 표준 입출력 버퍼가 꽉 찼을 때 실제 입출력 발생
    - 주로 디스크에 있는 파일에 full-buffered가 적용
    - full-buffered에 사용되는 버퍼는 주어진 한 스트림에 대하여 입출력 함수가 처음 수행 될 시 malloc()를 호출해서 할당
  - line-buffered
    - 입력이나 출력에서 개행(new line) 문자를 만났을 때 입출력을 수행
    - 터미널을 가리키는 스트림용 - > stdin, stdout
  - non-buffered
    - 문자들을 버퍼링 하지 않으며, 즉시 출력
    - stderr

setbuf(3), setvbuf(3) – mode <stdio.h>

mode	설명
_IOFBF	이 모드로 함수를 호출하면 입출력이 버퍼 단위로 이루어짐. 단말기와 관련되지 않은 모든 스트림의 기본 모드임
_IOLBF	이 모드로 함수를 호출하면 출력이 라인 단위로 버퍼링를 이용하게 되어 개행 문자를 만날 때마다 버퍼의 내용 출력. 이 외에도 버퍼를 비우게 되는 경우에는 버퍼가 다 차거나 입력이 요구되어지는 경우가 있음. 입력이 요구되어 진다라고 하는 것은, 개행을 포함하지 않는 printf()나 fputs()를 사용하는 경우에도 이들 함수 바로 뒤에 scanf()나 fgets() 등의 입력 함수가 오면 버퍼는 강제로 비워져 즉시 출력 하게 된다는 의미임
_IONBF	이 모드로 함수를 호출하면 입출력이 버퍼를 사용하지 않게 되며 buf와 size는 무시됨. stderr의 경우 이 모드가 기본 모드로 사용됨

setbuf(3), setvbuf(3) 요약

	mode	buf	버퍼의 크기	버퍼링 종류
ㄴetbuf()		NULL 아님	사용자가 제공한 buf, 길이는 BUFSIZ	full-buffered 또는 line-buffered
		setbufNULL	버퍼 없음	non-buffered
ㄴetvbuf()	_IOFBF	NULL 아님	사용자가 제공한 buf, 길이는 size	full-buffered
		setvbuf_IOFBFNULL	적절한 크기의 시스템 버퍼	
	_IOLBF	NULL 아님	사용자가 제공한 buf, 길이는 size	line-buffered
		setvbuf_IOLBFNULL	적절한 크기의 시스템 버퍼	
	_IONBF	무시됨	버퍼 없음	non-buffered

setbuf() 예제 1

<ssu_setbuf_1.c>

```
#include <stdio.h>
#include <stdlib.h>#include <unistd.h>#define BUFFER_SIZE 1024
int main(void)
{
   char buf[BUFFER_SIZE];
   setbuf(stdout, buf);
   printf("Hello, ");
   sleep(1);
   printf("OSLAB!!");
   sleep(1);
   printf("\n");
   sleep(1);
   setbuf(stdout, NULL);
   printf("How");
   sleep(1);
   printf(" are");
   sleep(1);
   printf(" you?");
   sleep(1);
   printf("\n");    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_setbuf_2

Hello, OSLAB!!

How are you?
```

setbuf() 예제 2

<ssu_setbuf_3.c>

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

#define BUFFER_SIZE 1024
int main(void)
{
   char buf[BUFFER_SIZE];
   int a, b;
   int i;
   setbuf(stdin, buf);
   scanf("%d %d", &a, &b);
   for (i = 0; buf[i] != '\n'; i++)
      putchar(buf[i]);

   putchar('\n');
   exit(0);
}
```

출력결과

```
root@localhost:/home/oslab# ./ssu_setbuf_3
1 2
1 2
```

setvbuf() 예제 1

```
<ssu_setvbuf.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define BUFFER_SIZE 1024

void ssu_setbuf(FILE *fp, char *buf);
int main(void)
{
   char buf[BUFFER_SIZE];    char *fname = "/dev/pts/19";
   FILE *fp;

   if ((fp = fopen(fname, "w")) == NULL) {
            fprintf(stderr, "fopen error for %s", fname);
            exit(1);
    }

   ssu_setbuf(fp, buf);
   fprintf(fp, "Hello, ");   sleep(1);
   fprintf(fp, "UNIX!!");   sleep(1);
   fprintf(fp, "\n");          sleep(1);
   ssu_setbuf(fp, NULL);
   fprintf(fp, "HOW");     sleep(1);
   fprintf(fp, " ARE");      sleep(1);
   fprintf(fp, " YOU?");    sleep(1);
   fprintf(fp, "\n");          sleep(1);
   exit(0);
}

void ssu_setbuf(FILE *fp, char *buf) {
   size_t size;   int fd;     int mode;
   fd = fileno(fp);
   if (isatty(fd))
      mode = _IOLBF;
   else
      mode = _IOFBF;
   if (buf == NULL) {
      mode = _IONBF;
      size = 0;
   }
   else
      size = BUFFER_SIZE;
   setvbuf(fp, buf, mode, size);
}
```

실행결과

```
oslab@localhost:~$ tty
/dev/pts/17

oslab@localhost:~$ ./ssu_setvbuf
Hello, UNIX!!
HOW ARE YOU?
```

fflush(3)

```
#include <stdio.h>
int fflush(FILE *fp);
```
리턴 값: 성공 시 0, 에러 시 EOF 리턴하고 errno 설정

- **스트림에 대해 아직 기록되지 않은 자료를 커널에 전달하는 라이브러리 함수로 버퍼의 내용을 해당 파일에 즉시씀**
  - fflush()가 호출되면 해당 파일의 버퍼가 완전히 차 있는지 아닌지에 관계없이 출력 버퍼를 강제로 비움
  - fp 인자로 전달된 스트림은 fflush() 호출 후에도 오픈 되어 있는 상태 유지
  - 프롬프트(prompt)
    - 개행 문자가 없는 문자열을 즉시 출력 시 사용