System Programming

2. File IO (1): Standard I/O Library - 2

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng.

File Open

```
#include <stdio.h>
FILE * fopen ( const char *filename, const char *type);
            type : access mode
 return  a file stream pointer to the open file if succeed, or
            NULL (error: failed to open)
```

- File access modes

Access modes	*Description*
r	Read only
w	Truncate file to zero length or create a file for writing.
a	Append mode(EOF), write only. The file is created if it does not exist.
r+	Read/write
w+	Truncate file to zero length or create a file for reading & writing.
a+	Read/append mode(EOF). The file is created if it does not exist.

- The number of open files has a limitation (by system configuration)

File Reopen

```
#include <stdio.h>

FILE * freopen ( const char *filename, const char *type, FILE *stream);
        type : access mode
        stream : file pointer
 return  a file stream pointer to the open file
        NULL (error: failed to open)
```

- first, close a file linked to the input stream (3rd arg)
- and open a file with a given filename by reusing the old stream
- NOTE: the original file descriptor is also reused!

Example

- guess what will happen in this code!

```
freopen("myfile.txt", "w", stdout);
printf("This sentence is redirected to a file.");
fclose(stdout);
```

File Close

```
#include <stdio.h>
int fclose( FILE *stream);
  return 0 for normal, EOF for error
```

- When a process exits normally, all files are *automatically* closed.
- If a process is terminated without closing a file,
  - cannot check some errors that are reported by the fclose().
  - file data in the library buffer might be lost.

File I/O functions

Function Prototypes	*Input Arg.*	*Return*	
		normal	*error*
size_t ***fread*** *(void *ptr, size_t size, size_t nitems, FILE *stream)*	- *ptr*: destination buffer address - *size*: # of bytes of the object unit - *nitems*: # of objects - *stream*: file pointer	# of objects read	0
size_t ***fwrite*** *(void *ptr, size_t size, size_t nitems, FILE *stream)*	- *ptr*: source buffer address - *size*: # of bytes of the object unit - *nitems*: # of objects - *stream*: file pointer	# of objects written	0

Character Input

Function Prototypes	Description	Return	
		normal	*error*
int ***getc*** *(FILE *stream),* *int* ***fgetc*** *(FILE *stream)*	Get a character from file.	char in integer type	EOF
int ***getchar****(void)*	Get a character from *stdin.*	char in integer type	EOF
*char **fgets** (char *s, int size, FILE *stream)*	Get a NULL (“\0”) terminated string. Read until a newline or EOF. Max string size = size -1.	char string address	NULL
*char **gets** (char *s)*	Get a NULL (“\0”) terminated string from *stdin*. Read until a newline or EOF.	char string address	NULL
int ***ungetc*** *(int c, FILE *stream)*	Put the character *c* into the file to enable rereading.	c	EOF

Character Output

Function Prototypes	*Description*	*Return*	
		normal	*error*
*int **putc** (int c, FILE *stream),* *int **fputc** (int c, FILE *stream)*	Write a character to file.	char in integer type	EOF
*int **putchar** (int c)*	Write a character to *stdout.*	char in integer type	EOF
*int ***fputs** (const char *s, FILE *stream)*	Write a string without its trailing "\0".	# of chars	EOF
*char ***puts** (const char *s)*	Write a string and a trailing newline to *stdout.*	# of chars	EOF

File I/O example (1)

fileio-ex.c

```
$ ./fileio-ex firstFile secondFile
```

```
argc=3
argv[0]="./fileio-ex"
argv[1]="firstFile"
argv[2]="secondFile"
```

```
#include <stdio.h>

int main( int argc, char *argv[])
{
        int c;
        FILE *fpin, *fpout;

        if( argc != 3) {
                perror( argv[0]);
                exit(1);
        }
        if(( fpin = fopen( argv[1], "r")) == NULL) {
                perror( argv[1]);
                exit(2);
        }
```

File I/O example (2)

```
        if(( fpout = fopen( argv[2], "a")) == NULL) {
                perror( argv[2];
                exit(3);
        }
        setbuf( fpin, NULL);   // unbuffered I/O
        setbuf( fpout, NULL); // unbuffered I/O

        while(( c = getc( fpin)) != EOF)
        putc( c, fpout);

        fclose( fpin);
        fclose( fpout);
        exit(0);
}
```

File I/O example (3)

- Execution

```
$ cat test1.txt
Hello, world (1)

$ cat test2.txt
Hello, world (2)

$ ./a.out  test1.txt   test2.txt

$ cat test1.txt
Hello, world (1)

$ cat test2.txt
Hello, world (2)
Hello, world (1)

$
```

Line I/O example (1)

lineio-ex.c

```
#include <stdio.h>

#define BUFFER_SIZE 100

int main(int argc, char *argv[])
{
        char ubuf[BUFFER_SIZE], line[BUFFER_SIZE];
        FILE *fpin, *fpout;

        if(argc != 3) {
                perror(argv[0]);
                return 1;
        }
        if ((fpin = fopen(argv[1], "r")) == NULL) {
                perror(argv[1]);
                return 2;
        }
```

Line I/O example (2)

```
        if ((fpout = fopen(argv[2], "a")) == NULL) {
                perror(argv[2]);
                return 3;
        }
        if (setvbuf (fpin, ubuf, _IOLBF, BUFFER_SIZE) != 0) { // line buffering
                perror("setvbuf(fpin)");
                return 4;
        }
        if (setvbuf (fpout, ubuf, _IOLBF, BUFFER_SIZE) != 0) {
                perror("setvbuf(fpout) ");
                return 5;
        }

        while ( fgets (line, BUFFER_SIZE, fpin) != NULL)
                fputs (line, fpout);

        fclose(fpin);    fclose(fpout);
        return 0;
}
```

Array I/O example

```
#define ARRAY_SIZE 10

        ......
        int i;
        int sample_arry[ARRAY_SIZE];
        FLLE *stream;

        if ((stream = fopen(argv[1], "w")) == NULL) {
                perror(argv[1]);
                return 1;
        }

        if (fwrite (sample_array, sizeof(int), ARRAY_SIZE, stream) != ARRAY_SIZE) {
                perror("fwrite error");
                return 2;
        }
        ......
```

Struct I/O example

```
struct {
        short count;
        char sample;
        long total;
        float numeric[LENGTH];
} object;

FILE *stream;
…

if (fwrite (&object, sizeof(object), 1, stream) !=1)
        perror("fwrite error");
```

File copy with Full buffering (1)

filecopy.c

```
# include <stdio.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[])
{
        char ubuf[BUFFER_SIZE], fbuf[BUFFER_SIZE];
        int n;
        FILE *fpin, *fpout;

        if(argc != 3) {
                perror(argv[0]);
                return 1;
        }
        if((fpin = fopen(argv[1], "r")) == NULL) {
                perror(argv[1]);
                return 2;
        }
```

File copy with Full buffering (2)

```
        if((fpout = fopen(argv[2], "w")) == NULL) {
                perror(argv[2]);
                return 3;
        }
        if (setvbuf(fpin, ubuf, _IOFBF, BUFFER_SIZE) != 0) { // full buffering
                perror("setvbuf(fpin)");
                return 4;
        }
        if (setvbuf (fout, ubuf, _IOFBF, BUFFER_SIZE) != 0 {
                perror("setvbuf(fpout)");
                return 5;
        }

        while ( n= fread(fbuf, sizeof(char), BUFFER_SIZE, fpin ) > 0)
                fwrite (fbuf, sizeof(char), n, fpout);

        fclose(fpin);
        fclose(fpout);
        return 0;
}
```