CS-392 Systems Programming — Spring 2023

Assignment 4 · Standard I/O Library of Your Own

Lecturer: Shudong Hao — *Date: See Canvas*

For this assignment, you will be creating your own I/O library called `"stdiox.h"`. Our library should provide the following utilities:

- Write integers/strings to and read integers/strings from a file;
- Print integers/strings to terminal;
- Receive user input as integers/strings from keyboard.

1 Function Specification

1.1 `fprintfx()`

The first function we're going to write is `fprintfx()`, whose declaration is as follows:

```c
int fprintfx(char* filename, char format, void* data);
```

- **Parameters:**
  - `char* filename`:
    The name of the file we want to write to. If it's empty, print the data to the terminal. If the file exists, append data to the end of the file; otherwise, create a new file with `rw-r-----` permission, and write to that file. Note: we don't need to close the file in this function;
  - `char format`:
    This indicates if we want to output a string or an integer to the file. If it's `'s'`, we output a string; if it's `'d'`, we output an integer;
  - `void* data`:
    The pointer the data we want to write. Note this pointer either points to a single integer, or a character array. We don't need to consider the situation where we have an integer array. If the base address is `NULL`, return -1 and set `errno` to `EIO`.
- **Return Value:**

- 0 if the data has been successfully printed to the destination;
- -1 otherwise.

▶ **Errors:**

- `EIO` : for any type of errors, `errno` should be set ot `EIO` .[1]

1.2 `fscanfx()`

The second function `fscanfx()` is declared as follows:

```c
int fscanfx(char* filename, char format, void* dst);
```

▶ **Parameters:**

- `char* filename` :
  The name of the file we want to read from. If it's empty, receive data from keyboard; otherwise, read **one line** from it. If the file exists but not opened, open the file first, and read one line from it. Note: we don't need to close the file here;
- `char format` :
  This indicates if we want to receive a string or an integer from the file. If it's `'s'` , we receive a string; if it's `'d'` , we receive an integer;
- `void* dst` :
  The address where we store the data read from the file.

▶ **Return value:**

- 0 if everything goes well;
- -1 when there's an error.

▶ **Errors:**

- `ENOENT` : when the file doesn't exist;
- `EIO` : all other errors.

▶ **Hint:**

- When reading from an existing file, if the file is not opened yet, we will open the file, and read one line from it; if the file is opened and still readable, we read next line from it. If we reached the end of the file, return -1 but no need to set `errno`;

[1]See `https://man7.org/linux/man-pages/man3/errno.3.html`.

- Because there's no way to predict how many characters we're going to read in one line, we have to dynamically create a buffer. If the buffer is full, you need to re-allocate a new buffer with larger size. You can assume the buffer initially has 128 bytes, and each time we allocate **128 more** bytes.

1.3 Clean

Lastly, because we didn't close the files in the two functions above, we need to close them in `clean()` function:

```
int clean();
```

This function simply returns 0 if everything goes well; otherwise returns -1, and sets `errno` to `EIO`.

In this function, you'd need to check all the files opened by the program, and close them using `close()` function. However, do not close `stdin`, `stdout`, and `stderr`. Close only user files.

2 Implementation Hints

- ▶ All the file descriptors of opened file in the current process are stored as the names of the files in directory `/proc/self/fd/`;
- ▶ To get the full path of a file, use `realpath()` function;
- ▶ To see if a file is opened, first get all file descriptors from the current process. Then compare their inode numbers with the inode number of the file you want to open;
- ▶ In `fscanfx()` function, you don't want to read everything at once, and store them into `dst` (the third parameter);
- ▶ You can certainly use `lseek()` to decide if you reached the end of the file, but you need to be careful. Additionally, remember `lseek()` doesn't work on `stdin` and `stdout`;
- ▶ You can assume when format is `'d'`, all input and output can be converted safely to integers, *i.e.*, there's no non-digit characters. However, when format is `'s'`, the input and output can be anything;
- ▶ When testing your code, you can certainly use `stdio.h` to help you verify the results. However, remember to **remove** anything from `stdio.h` (even if it's inside a pair of macros). "Forgot to remove it but my code works even without it" is an unacceptable excuse for partial credit;

▶ Here's a starter code for `stdiox.h`. Do not change this file. For any helper function you wrote or any header file you included, put them in `stdiox.c`. You can include other libraries but not `stdio.h`:

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <limits.h>

int fprintfx(char*, char, void*);
int fscanfx(char*, char, void*);
int clean();
```

3 Test

You should write your implementations in `stdiox.c`, and include `stdiox.h` where the functions are declared and necessary utility libraries are included. Then you should write you own `main.c` where you test the functionalities of your implementation.

Below is an example `main.c`. To compile, use the following command:

```
gcc main.c std392.c
```

Note: Your code should be able to compile without any `gcc` flags. No exceptions.

```c
#include "stdiox.h"

int main(int argc, char const *argv[]) {

    int array[5] = {1,2,-9,12,-3};
    char* string = "Hello!";

    /* Print integers to stdout */
    for (size_t i = 0; i < 5; i++) {
        fprintfx("", 'd', &array[i]);
    }

    /* Print string to stdout */
    fprintfx("", 's', string);

    /* Error: unrecognized format */
    fprintfx("", 'i', string);

    /* Write integers to a text file */
    for (size_t i = 0; i < 5; i ++) {
        fprintfx("text", 'd', &array[i]);
    }

    /* Write string to a text file */
    fprintfx("text", 's', string);

    clean();

    char newstr[1024] = {0};
    int  num;
    /* Receive a string from stdin */
    fscanfx("", 's', newstr);
    /* Print out the string to stdout */
    fprintfx("", 's', newstr);

    /* Receive an integer from stdin */
    fscanfx("", 'd', &num);
    /* Print out the integer to stdout */
    fprintfx("", 'd', &num);

    /* Read a file */
    while (!fscanfx("text", 's', newstr)) {
        fprintfx("", 's', "Line: ");
        fprintfx("", 's', newstr);
    }

    /* Close opened files */
    clean();

    return 0;
}
```

After execution, the file `text` should look like this:

```
1
2
-9
12
```

```
-3
Hello!
```

If during execution `stdin` receives input `world` and `420`, the entire terminal should look like this:

```
1  1
2  2
3  -9
4  12
5  -3
6  Hello!
7  world
8  world
9  420
10 420
11 Line:
12 1
13 Line:
14 2
15 Line:
16 -9
17 Line:
18 12
19 Line:
20 -3
21 Line:
22 Hello!
```

Note line 7 and line 9 are from `stdin`.

4 Requirement

- Including `stdio.h` anywhere in your code is prohibited and is a guarantee to zero credit (even if you didn't use any function from it);
- Declaring global variables is not allowed;
- Do not change the prototype of the three functions and the starter code (`stdiox.h`) mentioned above;
- You code should have comments, be able to compile, no memory leak, no un-closed files in the end;

- Do not include any header file that you didn't use any function from;
- You code has to be able to compile without any `gcc` flags.

Deliverable

Two files: `stdiox.h`, and `stdiox.c`. **Do not zip them.**

Extra Credits

In addition to integers and strings, we'll offer 10 extra credits for implementing floating points. Any requirement will be the same, except the format character would be `'f'`. Please use `float` type instead of `double` in your implementation. Note:

- The extra credits are added to your homework 4 total, not grand total;
- There's no partial extra credits;
- You won't be graded for extra credits if you didn't get full points on the main task;
- If you did extra credits, please indicate it in the comments on Canvas.