Operating Systems

HOCHSCHULE DER MEDIEN

Exercise D: File I/O

1. Universal I/O

In this exercise we are going to see how awesome a simple but powerful API can be.

- Create a file “universal.c”. Also create two textfiles “input.txt” and “output.txt” in the same folder. Copy some placeholder text (Lorem Ipsum) to the input.txt-file.
- Declare a variable that will serve as handle to an opened file. Check the manpage of open() for the correct type (you can open it with “man 2 open”).
- Open the “input.txt”-file with minimal permissions. We are going to read from it. Close the file at the end of your program.
- Create an input buffer. Read 1 KiByte (1024 Bytes) of data from the “input.txt”-file into the input buffer. Remember the actual number of read bytes for later.
- Write the data to the command line (STDOUT) using write(). How can you address STDOUT? Don’t write the entire buffer but only as many bytes as we actually read into the buffer.
- Now, instead of writing to the command line, write to the file “output.txt”. You will need to open the file (it should be writable). Also remember to close the file.
- Compare the two lines containing the write()-call. What did you change in order to write to another target?

Hints

These functions may be helpful to use. Use the manpage to get further information:

- open()
- close()
- read()
- write()

Includes

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

Operating Systems

2. Read and write it all

Now that we have explored the advantages of the universal I/O API, we are going to use it to implement a proper way to read and write large files. We will be needing this code later in the course of the lecture.

- Create a file “read_write_all.c”.
- Copy the string structure definition from the exercise C2 into your source file.
- Create a function with the following prototype. In the function, we are going to implement the logic for reading all the data from a file descriptor (e.g. an entire file).

```
string_t read_all(int fd, size_t max_size)
```

- In the function create a buffer for the data to be read. The buffer should store as much data as was specified in max_size. We want to return that buffer to the caller so a local array won't work as buffer.
- Read from the file as long as read() finds data or as long as there is space left in our buffer. Be aware of two things:
  - Tell read() to start writing at the end of the data that's already in the buffer.
  - Tell read() that it can write as many bytes as there is still space left in the buffer (space not filled by data already written).
- After you have read all possible data, make sure, the string in the buffer is \0-terminated.
- Open the “snake.txt”-file in your main function, use read_all() to read all the data from the file descriptor and close it at the end of the program. Print the data to the standard output. Also remember to free the data at the end of your program.

Hints

These functions may be helpful to use. Use the manpage to get further information:

- open()
- read()
- close()

Includes

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
```

Now that we are able to read entire files to memory, we should implement a write functionality.

- Add a function with the following prototype to your program.

  ```
  int write_all(int fd, string_t data)
  ```

  It should write all the data in “data” to the file descriptor “fd”. It should return the total number of bytes written or -1 on error.

- To implement write_all() use a loop similar to the read-loop from the previous function, but this time use the write()-call. Ensure that the entire buffer is written. We can achieve this by reversing some of the read-functionality. Remember also to properly tell write how many bytes are still left to write.
- If there are no more bytes to write you should stop writing.
- Use write_all() in your main function to write the data we've got from snake.txt to output.txt. You might want to use the O_CREAT and O_TRUNC flags for open(). Look into the man pages for more information.

Hints

These functions may be helpful to use. Use the manpage to get further information:

- write()

Includes

```
```