

# Group\_7 Assignment\_3 report

---

## Specifications

BSD-based systems provide a function called `funopen` that allows us to intercept read, write, seek, and close calls on a stream. Use this function to implement `fmemopen` for FreeBSD.

- Below is the description of our implementation of `fmemopen`

```
typedef struct {  
    char *data;  
    size_t size;  
    size_t position;  
} mem_file_t;
```

- First, we declare a struct called `mem_file_t`, which is the memory structure of file stream

```

FILE *fmemopen(void *buf, size_t size, const char *mode){
    mem_file_t *mem_file = malloc(sizeof(mem_file_t));
    mem_file->data = buf;
    mem_file->size = size;
    mem_file->position = 0;

    if (mode == NULL) {
        free(mem_file);
        return NULL;
    }

    // checking for the mode
    if (mode[1] == '+') {
        if (mode[0] == 'a') {
            while (mem_file->position < mem_file->size){
                mem_file->position++;
            }
            return funopen(mem_file, mem_read, mem_write, mem_seek, mem_close);
        } else if (mode[0] == 'w' || mode[0] == 'r'){
            return funopen(mem_file, mem_read, mem_write, mem_seek, mem_close);
        } else {
            return NULL;
        }
    } else {
        switch (mode[0]){
            case 'r':
                return funopen(mem_file, mem_read, NULL, mem_seek, mem_close);
                break;
            case 'w':
                return funopen(mem_file, NULL, mem_write, mem_seek, mem_close);
                break;
            case 'a':
                while (mem_file->position < mem_file->size){
                    mem_file->position++;
                }
                return funopen(mem_file, NULL, mem_write, mem_seek, mem_close);
                break;
            default:
                free(mem_file);
                return NULL;
                break;
        }
    }
    return funopen(mem_file, mem_read, mem_write, mem_seek, mem_close);
}

```

- After the declaration, we implement the func. `fmemopen`
  - We first use `malloc` to create the memory file metadata
  - Next, we check the argument of the mode, and according to the mode, we will decide the argument in the func. `funopen` and return.
    - if the `mode[1]` has a “+” sign, the `funopen` will have four arguments.
    - On the other hand, if the `mode[1]` was not “+” sign, we will replace the `readfn` or `writfn` to `NULL` according to the `mode[0]`

In your `fmemopen`, you should create a memory structure to initialize a file stream, and use `funopen` to deal with it with the following functions:

- (i) Read
- (ii) Write
- (iii) Seek
- (iv) Close

- below are four func. we implement for the Read, Write, Seek and Close
- Read

```
// Read function for funopen
int mem_read(void *cookie, char *buf, int size) {
    mem_file_t *mem_file = (mem_file_t *)cookie;

    // If the size of the read is bigger than the size of the file, read the rest of the whole file
    if (mem_file->position + size > mem_file->size) {
        size = mem_file->size - mem_file->position;
    }

    for (int i = 0; i < size; i++){
        buf[i] = mem_file->data[mem_file->position + i];
    }
    mem_file->position += size;
    return size;
}
```

- for the read func., first we will check if the size we want to read is bigger than the rest of the file stream

- if it is bigger than the stream, we will read the rest of the file.
    - if it is not bigger than the rest of the stream, we will read the required size
  - Last, we will return the size we have read.
- Write

```
// Write function for funopen
int mem_write(void *cookie, const char *buf, int size) {
    mem_file_t *mem_file = (mem_file_t *)cookie;

    // If there is no space left for writing -> return -1
    if (mem_file->position + size > mem_file->size) {
        printf("%d, %d\n", (int)mem_file->position + size, (int)mem_file->size);
        return -1;
    }

    for (int i = 0; i < size; i++){
        mem_file->data[mem_file->position + i] = buf[i];
    }
    mem_file->position += size;

    return size;
}
```

- for the write func., first we check if the rest space of the stream is big enough for the writing.
    - If it is big enough, we will start writing the data into the stream and return the size we have write
    - If it is too small, we will print out the size problem and return -1
- Seek

```
// Seek function for funopen
fpos_t mem_seek(void *cookie, fpos_t offset, int whence){
    mem_file_t *mem_file = (mem_file_t *)cookie;

    switch (whence) {
        case SEEK_SET:
            mem_file->position = offset;
            break;
        case SEEK_CUR:
            mem_file->position += offset;
            break;
        case SEEK_END:
            mem_file->position = mem_file->size + offset;
            break;
        default:
            return -1;
    }

    return mem_file->position;
}
```

- For the seek func., we use `switch-case` to execute function.
  - If the `whence` is `SEEK_SET`, then the cursor position will move to the `offset` argument position.
  - If the `whence` is `SEEK_CUR`, then the cursor position will become original position plus `offset` position.
  - If the `whence` is `SEEK_END`, then the cursor will be moved to the end of the file.
- In the end, we will return where the current cursor position is.
- Close

```
// Close function for funopen
int mem_close(void *cookie){
    if (cookie == NULL)
        return -1;

    free(cookie);

    return 0;
}
```

- For the close func., we first check if the `cookie` is `NULL`
  - If it is `NULL`, then there is nothing to close, the func. will return -1
  - On the other hand, the func. will free the `cookie` and return 0

In the main function, you should use your `fmemopen` function to:

- (1 pt) **Write** “hello, world” in the file stream.
  - (1 pt) **Seek** the position of “world” in the file stream.
  - (1 pt) **Read** the word “world” from the file stream and **print** it. Then, **print** the whole sentence “hello, world”.
  - (1 pt) **Close** the file stream correctly.
- (Notice that the order of the 4 tasks above is not fixed. You should figure out which to do first to complete all of them correctly.)

- For the required `main` func., below is the implementation we have done

```
int main() {
    // expected execution : Write -> Seek -> Read -> ( Seek -> Read ) -> Close
    char *buffer = malloc(sizeof(char) * BUFF_SIZE);
    char output_buffer[BUFF_SIZE] = {0};

    FILE *mem_stream = fmemopen(buffer, BUFF_SIZE, "w+"); // in this case, the mode is not really working, we didn't use it
    /*-

    int out = fwrite("hello, world\n", 1, 13, mem_stream);

    fseek(mem_stream, 7, SEEK_SET);

    fread(output_buffer, 1, 5, mem_stream);
    printf("%s\n", output_buffer);

    fseek(mem_stream, 0, SEEK_SET);

    fread(output_buffer, 1, 13, mem_stream);
    printf("%s", output_buffer);

    fclose(mem_stream);
    return 0;
}
```

- First, we call the `fmemopen` to create the memory structure and initialize the file stream.
- Next, we use `fwrite` to write the “hello, world” into memory stream.
- Then, we use `fseek` to move the cursor to the head of “world”, and `fread` to read the “world” to the buffer and print it out.

- Again, we use the `fseek` to move the cursor to the head of the memory stream, and `fread` to read out the “hello, world” to the buffer and print it out.
- In the end, we use the `fclose` to delete the memory stream.