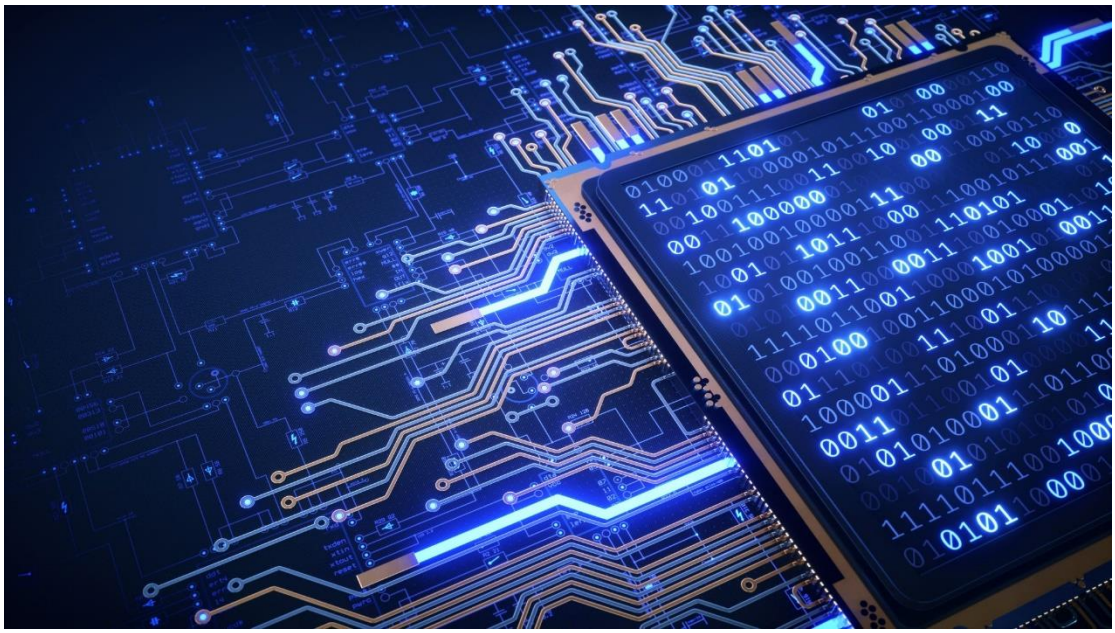
ADVANCED UNIX PROGRAMMING ASSIGNMENT REPORT

ASSIGNMENT 3



TEAM 9 — 林禾堃、馬毓昇、陳曦

Oct 2023

1. Code Implementation

1.1 fmemopen function

This is our structure for the cookies:

```
typedef struct {
        char    *buf; // buffer
        int     size; // buffer size in bytes
        int     len;  // data length in bytes
        int     off;  // current offset
}fmemopen_cookie_t;
```

And in the function, we first check the buffer pointer and initialize the cookie:

```
FILE *fmemopen(void *restrict buf, size_t size, const char *mode){
        // If buf is NULL, we should allocate a memory with size bytes of memory
        if(buf == NULL) buf = malloc(size * sizeof(char));

        // Initialize a cookie for stream
        fmemopen_cookie_t *c = malloc(sizeof(fmemopen_cookie_t));
        c->buf = buf;
        c->size = size;
        c->len = 0;
        c->off = 0;
```

And then, the read_flag and write_flag are set according to the mode chosen:

```
    // Handle modes
    // I think we don't need to implement x, b in this assignment
    if(mode == NULL) return NULL;
    int read_flag = 0;
    int write_flag = 0;
```

```
    switch(mode[0]){
            case 'r':
                    read_flag = 1;
                    c->len = c->size;
                    break;
            case 'w':
                    write_flag = 1;
                    break;
            case 'a':
                    write_flag = 1;
                    c->len = strnlen(c->buf, c->size);
                    c->off = strnlen(c->buf, c->size);
                    break;
            default:
                    return NULL;
    }
    if(strlen(mode) >= 2 && mode[1] == '+'){
            read_flag = write_flag = 1;
    }
```

During this session, we also set the data length for read. For append, we set the data length and offset by the buffer size. Also, for the plus signs, we set both flags to 1 (reading and writing). Finally, the function returns the file pointer through calling funopen and passing the cookie and four functions:

```
    return  funopen(c,
                    ((read_flag == 1) ? readfn : NULL),
                    ((write_flag == 1) ? writefn : NULL),
                    seekfn,
                    closefn);
```

1.2 readfn function

```
int readfn(void *cookie, char *buf, int nbytes){
        fmemopen_cookie_t *c = (fmemopen_cookie_t*)(cookie);
        nbytes = MIN(nbytes, c->len - c->off);
        if(nbytes <= 0) return 0;
        strncpy(buf, c->buf + c->off, nbytes);
        c->off += nbytes;
        return nbytes;
}
```

In the readfn function, we first check the bytes needed to be read, and then copy the required bytes of the file into buffer. Finally, adjust the offset accordingly and return the bytes read.

1.3 writefn function

```
int writefn(void *cookie, const char *buf, int nbytes){
        fmemopen_cookie_t *c = (fmemopen_cookie_t*)(cookie);
        nbytes = MIN(nbytes, c->size - c->off - 1); // -1 for ending '\0'
        if(nbytes <= 0) return 0;
        strncpy(c->buf + c->off, buf, nbytes);
        c->off += nbytes;
        c->len  = MAX(c->off, c->size - 1);
        c->buf[c->off] = '\0';
        return nbytes;
}
```

In the writefn function, we first check the bytes needed to be written (mind the ‘\0’), and then copy the required bytes of the file into buffer. Finally, adjust the offset and data length accordingly (mind the ‘\0’ also), put the ‘\0’ into the buffer and return the bytes written.

1.4 seekfn function

```
fpos_t seekfn(void *cookie, fpos_t offset, int whence){
        fmemopen_cookie_t *c = (fmemopen_cookie_t*)(cookie);
        switch(whence){
                case SEEK_SET:
                        if(offset >= c->size || offset < 0) return -1;
                        c->off = offset;
                        break;
                case SEEK_CUR:
                        if(c->off + offset >= c->size || c->off + offset < 0) return -1;
                        c->off += offset;
                        break;
                case SEEK_END:
                        if(c->size + offset < 0 || c->size + offset >= c->size) return -1;
                        c->off = c->size + offset;
                        break;
```

```
                case SEEK_HOLE:
                        if(offset >= c->size || offset < 0) return -1;
                        while(c->buf[offset] != 0 && offset < c->size) offset++;
                        if(offset >= c->size) return -1;
                        c->off = offset;
                        break;
                case SEEK_DATA:
                        if(offset >= c->size || offset < 0) return -1;
                        while(c->buf[offset] == 0 && offset < c->size) offset++;
                        if(offset >= c->size) return -1;
                        c->off = offset;
                        break;
                default:
                        return -1;
        }
        return c->off;
}
```

For the seekfn function, we check the validity first in each of the case, and set the offset and buffer data according to the whence argument.

1.5 closefn function

```
int closefn(void *cookie){
        fmemopen_cookie_t* c = (fmemopen_cookie_t*)(cookie);
        free(c); // we are not able to check whether free is success or not
        return 0;
}
```

To close the file, we free the cookie.

1.6 main function

In our main function, we complete each of the tasks in the given order after open the file with our fmemopen function:

1. Write “hello, world” in the file stream.

```
        // Task1: Write "hello, world" in the file stream
        char *buf = malloc(100);
        FILE *fp = fmemopen(buf, 100, "w+");
        fprintf(fp, "hello, world");
```

2. Seek the position of “world” in the file stream.

```c
// Task2: Seek the position of "world" in the file stream
fseek(fp, 7, SEEK_SET);
```

3. Read the word “world” from the file stream and print it. Then, print the whole sentence “hello, world”.

```c
// Task3: Read the word "world" from the file stream and print it. Then, print the whole sentence "hello, world".
buf = malloc(100);
fread(buf, sizeof(char), 5, fp);
printf("%s\n", buf);
fseek(fp, 0, SEEK_SET);
buf = malloc(100);
fread(buf, sizeof(char), 12, fp);
printf("%s\n", buf);
```

4. Close the file stream correctly.

```c
// Task4: Close the file stream correctly
fclose(fp);
```

2. Result

```
world
hello, world
```

As above, the output of the function is as expected.