CSE 344 – 2023 SYSTEM PROGRAMMING HOMEWORK 1 - REPORT

MUHAMMED SİNAN PEHLİVANOĞLU

1901042664

PART 1

NOTE: SPECIFIC X ARGUMENT IS “x”.

The default flags are O_RDONLY , O_WRONLY , O_APPEND and O_CREAT to create file.

If the ‘x’ flag is entered , then the O_APPEND is omitted from write_flags.

```
// check x flag
if (argc == 4 && strcmp(argv[3], "x") == 0) {
    append_flag = true;
    write_flags &=  ~O_APPEND;
}
```

Then the file is opened according to write_flags and permission mask.

Permission mask is 0777. Means that user , group and others can read-write-execute the file.

After the open file, writing is performed. If the append_flag is set to the true, then lseek operation is called before each write call .

```
int appendBytes(int fd, size_t count, bool append_flag){

    int totalbytes = 0;
    //struct flock lock;
    int byteswritten = 0;
    char byte = 'x';
    /*memset(&lock, 0, sizeof(lock));
    lock.l_type = F_WRLCK;
    fcntl(fd, F_SETLKW, &lock);*/

    while (totalbytes < count) {

        if (append_flag) {
            while ((byteswritten = write(fd, &byte, 1)) == -1 && errno == EINTR);
        }
        else{
            do{
                lseek(fd, 0, SEEK_END);
            }while((byteswritten = write(fd, &byte, 1)) == -1 && errno == EINTR);

        }
        if (byteswritten == -1) {
            perror("Failed to write to file");
            totalbytes = -1;
            break;
        }
        totalbytes += byteswritten;
    }
    /*lock.l_type = F_UNLCK;
    fcntl(fd, F_SETLKW, &lock);
    */
    return totalbytes;

}
```

RESULTS

```
-rwxr-xr-x 1 root root 2000000 Mar 30 16:46 f1
-rwxr-xr-x 1 root root 1309297 Mar 30 16:46 f2
```

Without “x” flag ， Size of the f1 is 2 million bytes , on the other hand using “x” flag that call lseek before each write call writed 1.3 million bytes.

PART2 – PART3 (COMBINED)

- **Dup implementation:**

```c
extern errno;

int my_dup(int oldfd){
    int retval;
    if(fcntl(oldfd,F_GETFD) < 0)
    {
        errno = EBADF;
        retval = -1;
    }
    else{
        retval = fcntl(oldfd, F_DUPFD, 0);
    }
    return retval;
}
```

Firstly, the oldfd is checked to find whether is a bad file descriptor or not.

If not, then fcntl is called using F_DUPPFD flag to create duplication of oldfd.

- **Dup2 implementation:**

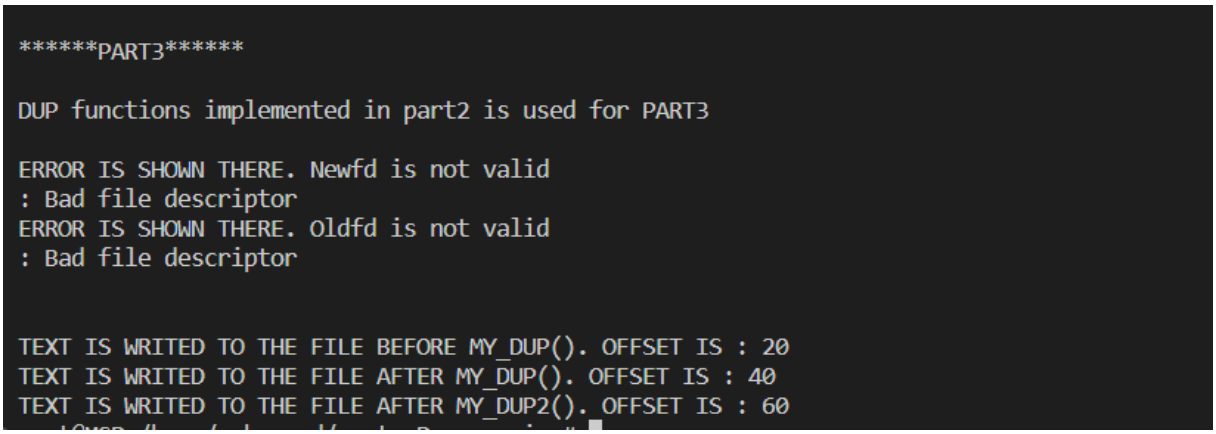
```c
int my_dup2(int oldfd, int newfd){
    int retval;

    //printf("%d", fcntl(oldfd, F_GETFD));

    if(fcntl(oldfd, F_GETFD) < 0){
        retval = -1;
        errno = EBADF;

    }
    else if(oldfd == newfd){

        retval =  newfd;
    }
    else{

        if(fcntl(newfd, F_GETFD) >= 0){
            if(close(newfd) == -1){
                fprintf(stderr, "Failed to close file\n");
                exit(EXIT_FAILURE);
            }

            retval = fcntl(oldfd, F_DUPFD, newfd);
        }
        else {

            retval = -1;
            errno = EBADF;
        }
    }
    return retval;
}
```

Firstly, the oldfd is checked to find whether is a bad file descriptor or not.

Also if oldfd is not a EBADF and newfd and oldfd is same , then oldfd value is returned.

Another possible situation is that if the newfd is not bad file descriptor then newfd is closed and Fcntl is called with F_DUPDFD flag with oldfd and newfd parameters.

PART3 RESULT



Firstly, the file is opened then returned value is assigned to the oldfd value.

- **First case is newfd is bad file descriptor. There is a error.**
- **Second case is oldfd is bad file descriptor. There is a error.**
- **Third case is writing sample text into file descripted by oldfd.**

Offset of file became 20.

- **Fourt case is writing sample text into file descripted by newfd created by my_Dup function that creates duplication of oldfd.**

Offset of file became 40.

- **Fifth case is writing sample text into file descripted by newfd2 created by my_Dup2 function that creates duplication of oldfd.**

Offset of file became 60.

SO each file descriptor shares same offset.