CSE 344
SYSTEM PROGRAMMING

HOMEWORK 1
REPORT

OGUZ MUTLU
1801042624

1. PROBLEM DEFINITION

Writing a program that takes up to the three command-line arguments, and this file should open the specified filename , if it necessary create, and append number of bytes to the file bu using write() at a time. Program use O_APPEND flag if third command line argument is not supplied. Otherwise program omits that flag. Use lseek(fd, 0, SEEK_END) instead of O_APPEND.

“$appendMeMore f1 1000000 & appendMeMore f1 1000000” Program runs like that command-line if the third command-line argument is provided program should run like “$appendMeMore f2 1000000 x & appendMeMore f2 1000000 x”.

In second problem we have to implement the dup and dup2() with using fcntl() function. If old file descriptor is not valid, then function should return -1 and errno set to EBADF. Also check the open, close, fcntl syscall errors.

In third problem we have to show that duplicated file descriptors share a file offset value and open file.

2.PROBLEM SOLUTION APPROACH

Part1

1.1 First thing we had to think about was that the file to be opened was to be opened using which mode.

1.2 Second thing is we had to think about was that the file access permission arrangement.

1.3 Controlling the input correctness, by comparing the argument number and argument input validity etc...

1.4 After controlling the command-line argument is valid we have to turn the argument from string to the integer.

1.5 Consider the conditions with changing command-line argument.

Part2

2.1. Read manual about the fcntl, dup, dup2.

2.2 Controlling the file descriptor if file descriptor is exist then program should move on, otherwise program returns an error about this issue(EBADF).

2.3 Duplicate the existing file descriptor with the help of fcntl macros.

2.4 For dup2() 2.3 error check remains the same in addition we have to consider what if the new file descriptor what user wants is same as old file descriptor.

2.5 Test that dup and dup2 works.

Part3

3.1 Open a file with system calls and duplicate the file descriptor using my own implementation.

3.2 Write a some string inside that file, with using original and duplicated file descriptor, and compare with the offset of these file by comparing the current offset location.

3.3 Error check when using the functions that implemented and system calls.

3.IMPLEMENTATION

In this part I explained that how I made implemention on problem solution approach.
Part1
1.1 : I initialize the flags, for file I use;
O_CREAT : creating a file when it is necessary
O_APPEND : append mode when each write system call occurs, our file offset must be in end of the file
O_RDWR : read and writing to a file.

```
flags = O_CREAT | O_APPEND | O_RDWR;
```

1.2 After file flags are initialized I initialize file access permissions by using mode macros.

- **S_IRUSR** = user read
- **S_IWUSR** = user write
- **S_IRGRP** = group read
- **S_IWGRP** = group write
- **S_IROTH** = other read
- **S_IWOTH** = other write

```
mode = S_IRUSR | S_IWUSR |
       S_IRGRP | S_IWGRP |
       S_IROTH | S_IWOTH;
```

1.3 All initializations are done I checked the command-line arguments, if given argc = 3 then flags must include O_APPEND; if argc = 4 and argv[3][0] == ‘x’ then open flag must not include O_APPEND flag and I opened the file according to that check..

```
else if(4 == argc && 'x' == argv[3][0]){
    flags = O_CREAT | O_RDWR;
    fd = open(argv[1], flags, mode);
```

```
if(3 == argc){
    flags = O_CREAT | O_APPEND | O_RDWR;
    fd = open(argv[1], flags, mode);
```

1.4 After opening the file we have write argument size byte to the our created or existing file, Since command-line argument is a string I want to write my own string to long integer converter. For this first I find length of the string which is number of bytes, then I substract ‘0’ for each character

then I move that digit to power function. Finally I converted the command-line argument to integer.

```
size_t string_length(char* string_number);
size_t string_to_digit(char* number_of_bytes, size_t len);
size_t power_of(int base, int power);
```

```
while('\0'!= *number_of_bytes){
    len--;
    digit = *number_of_bytes++ - '0';
    i = i + (digit*power_of(10, len));
}
```

1.5 I finally write one byte at a time with help of for loop. Without ‘x’ argument I write to file ‘a’ char buffer size is just one character. With O_APPEND writes every byte without race conditions. With ‘x’ argument I use lseek(fileDescriptor, 0, SEED_END).

```
for(size_t i = 0; i < digit ; i++){
    wr = write(fd,"a", 1);
}
```

```
for(size_t i = 0; i < digit ; i++){
    current = lseek(fd, 0, SEEK_END);
    wr = write(fd,"a", 1);
}
```

Part2

2.1 dup() system call allocates a new file descriptor that refers to the same open file description as the descriptor, the dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor.
2.2 we use only old file descriptor, dup function check only the old file descriptor is valid or not and set errno to EBADF

```
int is_file_opened = fcntl(oldfd, F_GETFL);

if (-1 == is_file_opened) {
    errno = EBADF;
    return -1;
}

int new_file_descriptor = fcntl(oldfd, F_DUPFD);
```

2.3 with the help of fcntl duplication flags **F_DUPFD,** fcntl returns created duplicated file description number, if we use only old file descriptor dup function check only the old file descriptor is valid or not, in dup2 we check file descriptor is valid and fcntl system call.

```
int is_file_opened = fcntl(oldfd, F_GETFL);

if (-1 == is_file_opened) {
    errno = EBADF;
    return -1;
}

int new_file_descriptor = fcntl(oldfd, F_DUPFD);
```

```
}
else{
    if(-1 != fcntl(newfd, F_GETFD)){/*According to
        close(newfd);                /*errors durin
    }

    if(-1 == fcntl(oldfd, F_DUPFD, newfd)){
        return -1;
    }
```

2.4 If old descriptor and new descriptor is same then dup2 returns new descriptor.

```
if(oldfd == newfd){
    return newfd;
}
```

2.5 Testing shown in test cases.

Part3

3.1 I duplicated the opened file descriptor, with my own implementation
3.2 I write string into that created text file with original file descriptor and duplicated descriptor separately results shown in test cases and result section.

```
fd_duplicated = dup(fd_original);
if(-1 == fd_duplicated){
    perror("duplicate");
    return -1;
}
```

```
char* original_buffer = "Hello from original file descriptor.\n";
wr_orgnl = write(fd_original,original_buffer, strlen(original_buffer));

char* duplicated_buffer = "Hello from duplicated file descriptor.\n";
wr_dup = write(fd_duplicated,duplicated_buffer, strlen(duplicated_buffer));
```

4.TEST CASES AND RESULTS

Part1 Result:

```
-rw-rw-r-- 1 oguz oguz 2000000 Mar 30 18:17 f1
-rw-rw-r-- 1 oguz oguz 1940175 Mar 30 18:17 f2
-rwxrwxr-x 1 oguz oguz   16280 Mar 30 18:17 part1
```

Part2 Result:


```
fd_dup = dup(fd);
char* original_buffer = "Hello from original file descriptor.\n";
wr_orgnl = write(fd,original_buffer, strlen(original_buffer));

char* duplicated_buffer = "Hello from dup file descriptor.\n";
wr_dup = write(fd_dup,duplicated_buffer, strlen(duplicated_buffer));


fd_dup2 = dup2(fd, fd_dup);
char* dup2_buffer = "Hello from dup2 file descriptor.\n";
wr_dup2 = write(fd_dup2,dup2_buffer, strlen(dup2_buffer));
```

Part3 Result:

```
oguz@oguz-ubuntu:~/Desktop/System Programmin
original file descriptor offset: 76
duplicated file descriptor offset: 76
oguz@oguz-ubuntu:~/Desktop/System Programmin
```

```
1 Hello from original file descriptor.
2 Hello from duplicated file descriptor.
```

Test Case 1

Steps:

1. Write same file with/without ‘x’ command-line argument
2. Run with & in background.

Expectation:

- f1.txt created and file size must be 2 times number of bytes given in command-line argument
- x command-line argument bytes less than first result

Result:

-PASS

```
-rw-rw-r-- 1 oguz oguz 2000000 Mar 30 18:17 f1
-rw-rw-r-- 1 oguz oguz 1940175 Mar 30 18:17 f2
-rwxrwxr-x 1 oguz oguz   16280 Mar 30 18:17 part1
```

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw1$ ./part1 f1 1000000 & ./part1 f1 100
0000
[1] 29576
[1]+  Done                    ./part1 f1 1000000
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw1$ ./part1 f2 1000000 x & ./part1 f2 1
000000 x
[1] 29579
[1]+  Done                    ./part1 f2 1000000 x
```

Test Case 2

Steps:

1.Create duplication uninitialized file descriptor.

Expectation:

- dup2() function returns error and terminate.

Result:

- PASS

```
oguz@oguz-ubuntu:~/Des
fd_dup2 : -1
oguz@oguz-ubuntu:~/Des
```

```
fd_dup2 = dup2(15, 5);
printf("fd_dup2 : %d\n", fd_dup2);

fd = open("part2.txt", flags, mode);
```

Test Case 3

Steps:

1.Create file descriptor.
2. Create duplication of file descriptor. (with dup())
3. Create duplication of file descriptor (with dup2())
4. Send dup() file descriptor to the dup2() for testing
5.Write to a created file from using each file descriptors.

Expectation:

- In text file all file descriptor buffers writed succesfully

Result:

- PASS

part2.txt

```
1 Hello from original file descriptor.
2 Hello from dup file descriptor.
3 Hello from dup2 file descriptor.
```

```
fd_dup = dup(fd);
char* original_buffer = "Hello from original file descriptor.\n";
wr_orgnl = write(fd,original_buffer, strlen(original_buffer));

char* duplicated_buffer = "Hello from dup file descriptor.\n";
wr_dup = write(fd_dup,duplicated_buffer, strlen(duplicated_buffer));


fd_dup2 = dup2(fd, fd_dup);
char* dup2_buffer = "Hello from dup2 file descriptor.\n";
wr_dup2 = write(fd_dup2,dup2_buffer, strlen(dup2_buffer));
```

Test Case 4

Steps:

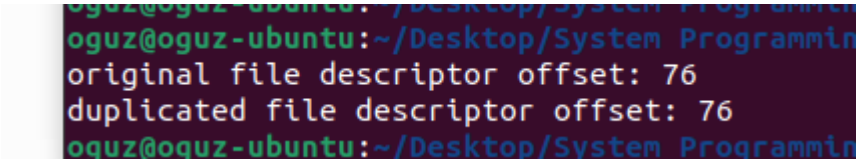
1.Create file descriptor.
2. Create duplication of file descriptor. (with dup())
3. Write to file using both file descriptors that created
4. Show the offset of files

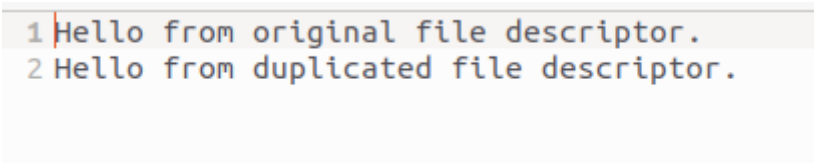
Expectation:

- The offset value of both descriptors are the same and each write opearation done successfully.

Result:

- PASS






```
char* original_buffer = "Hello from original file descriptor.\n";
wr_orgnl = write(fd_original,original_buffer, strlen(original_buffer));

char* duplicated_buffer = "Hello from duplicated file descriptor.\n";
wr_dup = write(fd_duplicated,duplicated_buffer, strlen(duplicated_buffer));

long int offset_fd = lseek(fd_original, 0, SEEK_CUR);
long int offset_fd_duplicated = lseek(fd_duplicated, 0, SEEK_CUR);

printf("original file descriptor offset: %ld\nduplicated file descriptor offset: %ld\n", offset_fd, offset_fd_duplicated);
```

5.CONCLUSION

In conclusion difference between part1’s differences between two different execution; If the first process executing the code is interrupted between the lseek() and write() calls by a second process doing the same thing, then both processes will set their file offset to the same location before writing, and when the first process is rescheduled, it will overwrite the data already written by the second process. This is a race condition.
With O_APPEND flag we avoid this problem, This problem requires that the seek to the next byte past the end of the file and the write operation happen atomically