Contents
I/O and File Descriptor – Exercises ................................................................. 2
  Exercise 1: ........................................................................................................ 2
  Exercise 1 – Solutions .................................................................................... 2
  Exercise 2: ....................................................................................................... 4
  Exercise 2 – Solutions .................................................................................... 5
  Example 1 ....................................................................................................... 6
  Example 2 ....................................................................................................... 7
  Exercise 3: ....................................................................................................... 8
  Exercise 3 – Solutions .................................................................................... 9
  Example 3 ....................................................................................................... 10
  Exercise 4 ....................................................................................................... 12
Exercise 1:

```c
int EXE1()
{
    int fd2 = open("logout.txt", O_WRONLY | O_CREAT);
    int fd3 = open("logerr.txt", O_WRONLY | O_CREAT);
    printf("the value of fd2 = %d \n", fd2);
    printf("the value of fd3 = %d \n", fd3);
    return 0;
}
```

1. What is the output of the function EXE1()?
2. Draw diagrams as above for the process fd’s table (user side) and the devices/file (kernel side)!

Exercise 1 – Solutions:

1. The output of the function is:
   - the value of fd2 = 3
   - the value of fd3 = 4
   - the default 0, 1, 2 are allocated to keyboard and monitor, then 3 and 4 are the lowest available free numbers. (the behavior of the kernel)

2. We allocated another 2 fd’s 3, 4 then the process fd’s table will have 5 entries.
**Important Notes:**

1- As we see 5 entries for Process1.
2- The direction of the arrow is tell if its read or write, fd equal to zero we read from the keyboard in other hand the fd 1 to 4 are writes to devices or files.
Exercise 2:

```c
int EXE2()
{
    close(1);
    int output_fd = open("logout.txt", O_WRONLY | O_CREAT);
    close(2);
    int outerr_fd = open("logerr.txt", O_WRONLY | O_CREAT);
    printf("the value of output_fd = %d \n", output_fd);
    printf("the value of outerr_fd = %d \n", outerr_fd);

    fprintf(stdout,
            "It does not matter how slowly you go as long as you do not stop, Confucius");
    fprintf(stderr, "Hello World full of errors on line1 \n");
    return 0;
}
```

1. We run the EXE2(), we get nothing in output-console ,Why?
2. What are the values of `output_fd` and `outerr_fd`?
3. Draw diagrams as above for the process fd’s table (user side) and the devices/file (kernel side) !?
Exercise 2 – Solutions :

1. The output of the function `printf()` is written to file descriptor equal to 1 or stdout (standard output), `fprintf()` is write with given file descriptor.
   - stdout mapped from monitor to logout.txt
   - stderr mapped from monitor to logerr.txt

2. The values of `output_fd` and `outerr_fd`
   The value of `output_fd = 1 // our standard output`
   The value of `outerr_fd = 2 // our standard error output`

3. The Diagram

![Diagram]

Important Notes:
1. stdout, stderr are Macros defined equal to 1 and 2 respectively.
2. `fprintf()` are used to simulate and error output with `fd = stderr`!
**Read & Write**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(fd,buf,n)</code></td>
<td><code>fd : the file descriptor for device/file</code>&lt;br&gt;<code>n : number of bytes to read (at most)</code>&lt;br&gt;<code>buf : buffer for bytes to write to it</code></td>
<td>Number of bytes the actually read, and zero for the end of file.</td>
</tr>
<tr>
<td><code>write(fd,buf,n)</code></td>
<td><code>fd : the file descriptor for device/file</code>&lt;br&gt;<code>n : number of bytes to write (at most)</code>&lt;br&gt;<code>buf : buffer for bytes to read from it</code></td>
<td>Number of bytes the actually written, and zero for the end of file.</td>
</tr>
</tbody>
</table>

**Example 1**

```c
int EXP1()
{
    char buf_read[512];
    int n;
    for (;;)
    {
        n = read(0, buf_read, sizeof(buf_read)); // return number of bytes
        if (n == 0) break; // if it the end of file
        if (n < 0) // error in read
            {
            fprintf(stderr, "read error
");
            exit(0);
        }
        if (write(1, buf_read, n) != n) // if the written bytes not equal to n, error !!
            {
            fprintf(stderr, "write error
");
        }
    }
    return 0;
}
```

היתהיית קוראת מהמקלדת – stdin על ידי פונקציית read ומדפיסה על המסך – stdout, במקורה Shields יש ייקרא read stdin. stderr – stderr, מדפיס בآخر המסר.
File descriptors and Fork&Exec

 samtgal shel Fd -im meshakluta casar gunsha () , lair ker gm casar meritzim tokhna teshu'a vayotz othilir teshu'a.}

משפה() הספרה עם

Example 2

```c
int EXP2()
{
    int pid, n;
    char buf[64];
    close(1);
    int fd2 = open("logout.txt", O_WRONLY | O_CREAT);
    close(2);
    int fd3 = open("logerr.txt", O_WRONLY | O_CREAT);
    if ((pid = fork()) == 0))
    {
        sprintf(buf, "/proc/%d/fd/", getpid());
        char *args[] =
            { "ls", "-la", buf, (char *) 0 };
        execvp(args[0], args);
    }
    else
    {
        sprintf(buf, "/proc/%d/fd/", getpid());
        char *args[] =
            { "ls", "-la", buf, (char *) 0 };
        execvp(args[0], args);
    }
    return 0;
}
```

logout.txt

```
total 0
dr-x----- 2 borak borak 0 Oct 9 11:34 .
dr-xr-xr-x 9 borak borak 0 Oct 9 11:34 ..
lrwx------ 1 borak borak 64 Oct 9 11:34 0 -> /dev/pts/3
-rwx------ 1 borak borak 64 Oct 9 11:34 1 -> /home/borak/workspace/Chapter0/logout.txt
-rwx------ 1 borak borak 64 Oct 9 11:34 2 -> /home/borak/workspace/Chapter0/logerr.txt
lr-x------ 1 borak borak 64 Oct 9 11:34 3 -> /proc/5070/fd
```

total 0
```
```
Important Notes:

1. `/proc` is directory where all the processes details are exist. like fd’s
2. The child and the father are pointed to the same files with same fd’s number!
3. `Sprintf()` function that fill the buffer with required string.
4. `Fork()` duplicate the fd’s table, `exec()` preserve the file table !!!

Exercise 3:

```c
int EXE3()
{
    close(1);
    int fd1 = open("logout.txt", O_WRONLY | O_CREAT);
    int fd2 = open("logerr.txt", O_WRONLY | O_CREAT);
    int fd3 = dup(fd1);
    int fd4 = fd1;

    printf("the value of fd’s (1) = %d , (2) = %d , (3) = %d , (4) = %d ", fd1,
            fd2, fd3, fd4);

    fprintf(stdout,
            "It does not matter how slowly you go as long as you do not stop, Confucius");
    fprintf(stderr, "Hello World full of errors on line! \n");

    return 0;
}
```

1. If we run the code which sentence we will see in monitor and which not?
2. What are the values of fds?
3. Draw diagrams as above for the process fd’s table (user side) and the devices/file (kernel side) !?
Exercise 3 – Solutions:

1. The fprintf() with stderr as file descriptor will shown in the monitor, other printing functions will print in the file logout.txt.

2. The values of fds are:
   The value of fd's (1) = 1, (2) = 3, (3) = 4, (4) = 1

3. The Diagram

   dup() & close() & open() :
   The fd1 and fd3 are allocate to same file (logout.txt) and have the value 1, 4 respectively.

   fd1 is the stdout – standard out put – redirect the stdout from the monitor to logout.txt file by close and open.

   file descriptor equal to 2 are still point to the monitor (stderr – standard error output)

   open() :
   fd2 have the value 3 and pointed to logerr.txt – regular file
fd4 is just copy of value not file descriptor, the difference between dup() and copy; The idea of duplication is to make a new file descriptor, a new entry in file descriptor table. if we delete fd1 we still have access to logout.txt! Through fd3. In other hand the copy will fail by delete fd1 to have access to logout.txt

Example 3

```c
int EXP3()
{
    int p[2];
    char* argv[2];
    argv[0] = "wc";
    argv[1] = 0;
    pipe(p);
    if (fork() == 0) // 1**
    {
        close(0);
        dup(p[0]);
        close(p[0]);
        close(p[1]);
        execvp("wc", argv);
    }
    else
    {
        close(p[0]);
        write(p[1], "hello world\n", 12);
        close(p[1]);
    }
    return 0; // 2**
}
```

The diagram: without mention the monitor fd (1&2) – for simplicity

Part1- after fork() finished create the child process

![Diagram](image)
Part2 – before return from the function.

The child read from pipe as standard input – stdin, then after write “hello world”, the child get/read the sentence with execute “wc”. The output will be:

1 2 12

1 line and 2 words and 12 character.

*Important Notes:*

1- wc blocked till give a input (stdin),in our example wc read from pipe.
Exercise 4

Build a communication system between 2 processes.

How can we get a communication system like this?