1 Opening and reading files

Simple assembly program:

```
section .data
        pathname DD "/home/priyanuj/temp/test.txt"
section .text
        global main

        main:
                MOV eax, 5
                MOV ebx, pathname
                MOV ecx, 0
                INT 80h

                MOV eax, 1
                INT 80h
```

We are opening a text file for reading in the program above. 5 is the code that will be stored in **eax**, **ebx** will store the pathname and **ecx** will store the mode in which the file will be opened. Here 0 indicates that file is opened in read-only mode.

1.1 Executing using GDB

We now, execute the program using GDB. First, we assemble and compile the program using **nasm** and **gcc**.

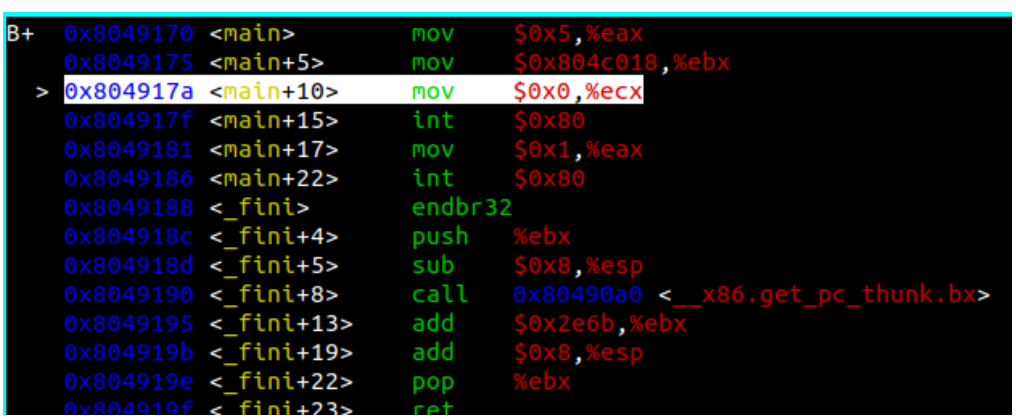

Figure 1: Code and address of pathname moved into **eax** and **ebx** respectively

We can see the values in **eax** and **ebx** as follows:

```
(gdb) info registers eax
eax            0x5                 5
(gdb) info registers ebx
ebx            0x804c018           134529048
```

Figure 2: Values in **eax** and **ebx**

ebx stores the address of the **pathname**. We can view how the pathname is stored as:

```
(gdb) x/10x 0x804c018
```

This will give us the following output:

```
(gdb) x/10x 0x804c018
0x804c018:      0x6d6f682f      0x72702f65      0x6e617969      0x742f6a75
0x804c028:      0x2f706d65      0x74736574      0x7478742e      0x00000000
0x804c038:      0x00000000      0x00000000
```

Figure 3: String stored in addresses

We can also view the actual string as:

```
(gdb) x/10s 0x804c018
```

We see the following output:

```
(gdb) x/10s 0x804c018
0x804c018:      "/home/priyanuj/temp/test.txt"
0x804c035:      ""
0x804c036:      ""
0x804c037:      ""
0x804c038:      ""
0x804c039:      ""
0x804c03a:      ""
0x804c03b:      ""
0x804c03c:      ""
0x804c03d:      ""
(gdb)
```

Figure 4: Pathname stored in memory

Now, after executing the interrupt **INT 80h**, we can see that the value of **eax** has changed:

```
(gdb) info registers eax
eax            0x3                 3
(gdb)
```

Figure 5: **eax** changed

Now, after the performing the interrupt, a file descriptor(an integer) is returned in **eax**. Here the file descriptor is 3.

2 Reading contents from the file

We can use this file descriptor to read data from a file.

```
section .data
        pathname DD "/home/priyanuj/temp/test.txt"

section .bss
        buffer RESB 1024

section .text
        MOV eax, 5
        MOV ebx, pathname
        MOV ecx, 0
        INT 80h

        MOV ebx, eax
        MOV eax, 3
        MOV ecx, buffer
        MOV edx, 1024
        INT 80h

        MOV eax, 1
        INT 80h
```

In order to save the read data, we need a buffer, so we declared an uninitialized buffer of 1024 bytes. Now, the file descriptor is stored in **eax**, so we need to move it to **ebx** first so that we can store the code 3 into **eax** which is the code for **sys_read** so that we can read from the file.

The image below shows the execution of the program:

```
B+  0x8049170 <main>        mov    $0x5,%eax
    0x8049175 <main+5>      mov    $0x804c018,%ebx
    0x804917a <main+10>     mov    $0x0,%ecx
    0x804917f <main+15>     int    $0x80
    0x8049181 <main+17>     mov    %eax,%ebx
    0x8049183 <main+19>     mov    $0x3,%eax
    0x8049188 <main+24>     mov    $0x804c038,%ecx
    0x804918d <main+29>     mov    $0x400,%edx
    0x8049192 <main+34>     int    $0x80
  > 0x8049194 <main+36>     mov    $0x1,%eax
    0x8049199 <main+41>     int    $0x80
    0x804919b               add    %dh,%bl
    0x804919d <_fini+1>     nop    %ebx
    0x80491a0 <_fini+4>     push   %ebx
    0x80491a1 <_fini+5>     sub    $0x8,%esp
    0x80491a4 <_fini+8>     call   0x80490a0 <__x86.get_pc_thunk.bx>
    0x80491a9 <_fini+13>    add    $0x2e57,%ebx
    0x80491af <_fini+19>    add    $0x8,%esp
    0x80491b2 <_fini+22>    pop    %ebx
    0x80491b3 <_fini+23>    ret
    0x80491b4               add    %al,(%eax)
    0x80491b6               add    %al,(%eax)
    0x80491b8               add    %al,(%eax)
    0x80491ba               add    %al,(%eax)
    0x80491bc               add    %al,(%eax)
```

Figure 6: Executing

We can see an address **0x804c038** which is being moved into **ecx**. That is actually the address of the buffer **buffer**.

```
(gdb) info registers eax
eax            0x2c                44
(gdb) x/x 0x804c038
0x804c038:      0x73696854
(gdb) x/s 0x804c038
0x804c038:      "This is a text file. It contains only text.\n"
(gdb)
```

Figure 7: Value in **eax** and the contents of **0x804c038**

.

So, the we can see that **eax** stores the number 44. The number actually represents the number of characters read from the file. So, it read 44 characters from the file because there were 44 characters in the file.

The string is stored in the memory slot starting from the address **0x804c038** which we can see from the image above.