Computer Lab 4

Input-Output Operations and Buffers

Computer Architectures for MSc in Engineering

HT19, DT509G

Abshir Abdulrahman & Fria Khorshid

2019-10-17

Task1: Buffered input

The objective of this task was to create a function that opens a file in binary reading mode and read the contents into a 16 byte buffer. We created a sample .txt file which we used for reading and testing our function. Our function fills the buffer with 16 bytes from the text file the first time it is called. The subsequent calls only reads the next byte in the file.

In our function we first check if the array is empty or full. If it is empty we fill it up and if it is full we only print the next byte.

```c
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#define BUFFER_SIZE 16

/* This function reads a file into an array buffer of 16 bytes and then reads the next one byte of the files for each call   */
void buf_int(char *binary_array, int file_descriptor)
{
    int binary_size;
    if (!binary_array[0])  /* This checks if the buffer is full or not.  */
    {
        binary_size = read(file_descriptor, binary_array, BUFFER_SIZE);
    }
    else
    {
        binary_size = read(file_descriptor, binary_array, 1);
    }

    binary_array[binary_size] = '\0';
    printf("%s\n", binary_array);

}


int main()
{
    int i;
    int file_descriptor = open("bytefile.txt", O_RDONLY);
    char* binary_array = malloc(BUFFER_SIZE * sizeof(char));
    /* This loop calls the function multiple times for testing purposes  */
    for(i = 0; i < 32; i++)
    {
        buf_int(binary_array, file_descriptor);
    }

    return 0;
}
```

```
frikhh171@ltse01:~/datateknik/build$ ./labb4_task1
When they had sw
o
r
n

t
o

t
h
i
s

a
d
v
i
s
e
d

d
o
o
m
,



T
h
e
y
```

This is the result when running the program.

Task 2: Buffered output

The objective of this task was to create a program that uses the buffer from the previous task and writes the contents of the buffer into a new file. The program has two additional functions. Buf_out will only write out the contents if the buffer is full. Buf_flush will write out the contents of the buffer even if it isn't full.

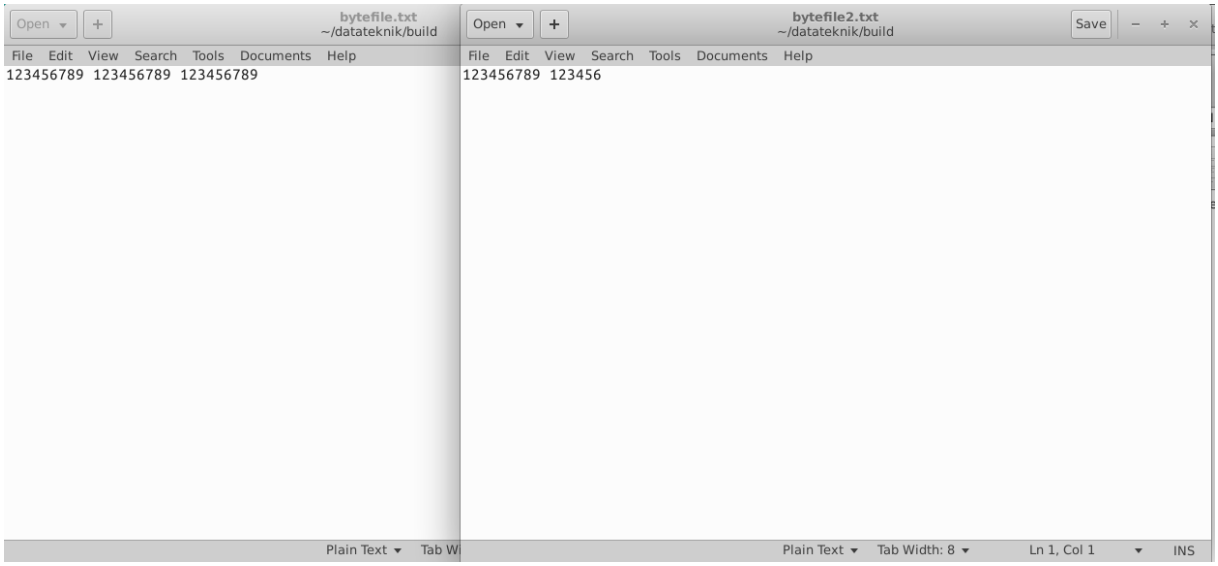
This is the change we made to our main function. This checks if the buffer is full or not. If it is full we use the buf_our function.

```
    if(binary_array[15])
    {
        buf_out(binary_array, file_descriptor2);
    }
    else
    {
        buf_flush(binary_array, file_descriptor2);
    }
    close(file_descriptor2);
    return 0;
}
```

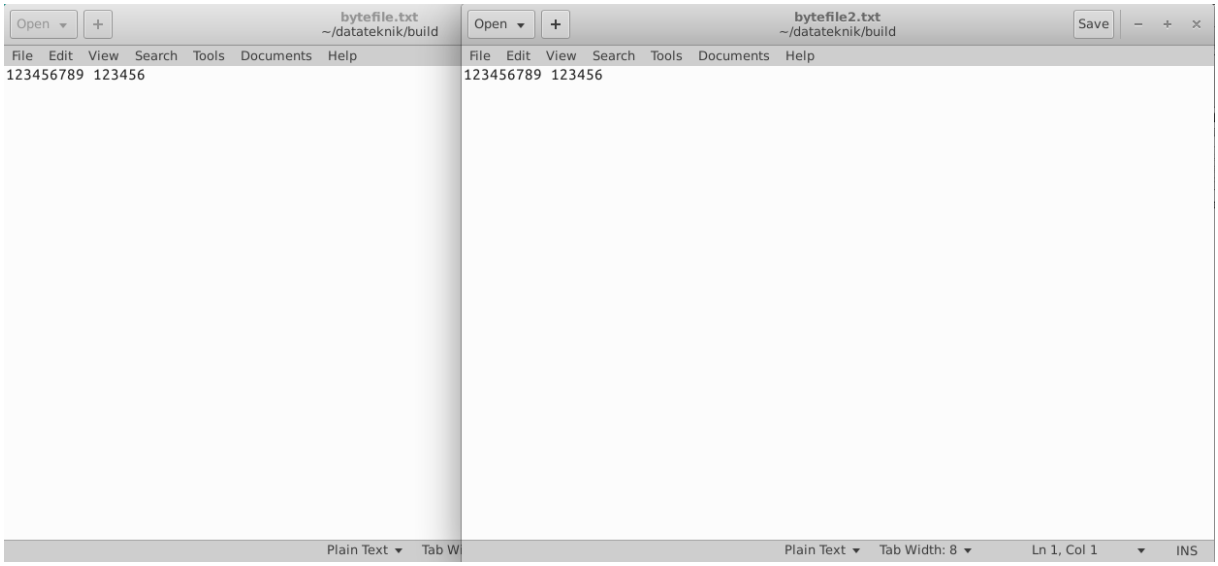
```
28  void buf_out(char *binary_array, int file_descriptor2)
29  {
30
31
32      if (binary_array[15])  /* This checks if the buffer is full or not.  */
33      {
34
35          write(file_descriptor2, binary_array, BUFFER_SIZE);
36      }
37      else
38      {
39          printf("The buffer is not full\n");
40      }
41
42  }
43
44  void buf_flush(char *binary_array, int file_descriptor3)
45  {
46      int i;
47      int size = 0;
48      for (i = 0; i < BUFFER_SIZE; i++)
49      {
50          if(binary_array[i])
51          {
52              size++;
53
54          }
55
56      }
57
58      write(file_descriptor3, binary_array, size );
59  }
```

These are the results we got when testing the functions.

First test is when the buffer is filled up to 16 bytes. Bytefile.txt is the file that is being read and Bytefile2.txt is the one our program creates.



Next test is when the buffer is exactly 16 bytes.



Last test is when the buffer has less than 16 bytes of content.