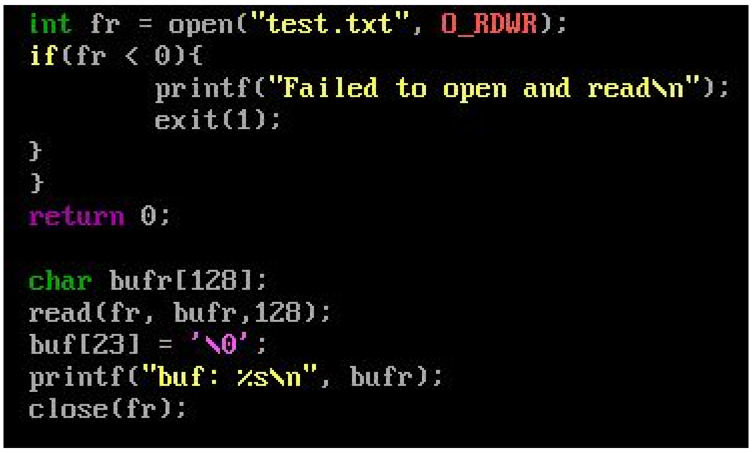
Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya

Task 1: Buffered input

For buffered input we are supposed to write a buffer for reading the text file that we have made.
From scratch, we start with creating a file; nano test.txt and entering an input. To be in alignment with the task we make it a size 32 bytes which is 32 characters of type char. We can read the contents of the file by using flags O_RDWR, O_RDONLY and print it out. Like this in the main function:



```
int fr = open("test.txt", O_RDWR);
if(fr < 0){
        printf("Failed to open and read\n");
        exit(1);
}
}
return 0;

char bufr[128];
read(fr, bufr,128);
buf[23] = '\0';
printf("buf: %s\n", bufr);
close(fr);
```

Figure 1.

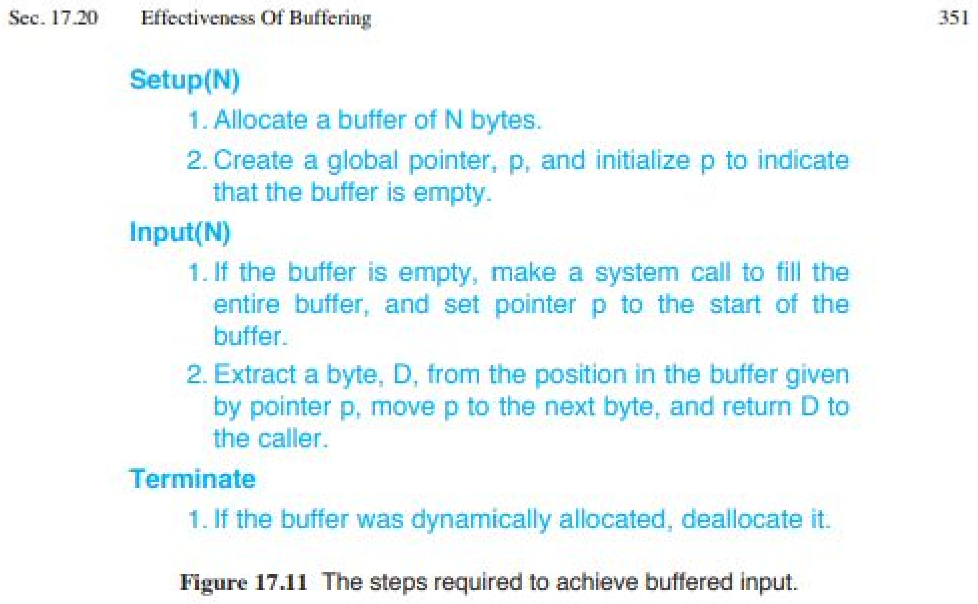
Note: For flag (O_RDWR, O_CREATE, O_RDONLY etc.. syntax we need to include the following libraries:
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
Main form is: int(open(const char *pathname,int flags, (eventually mode_t mode)):

This will return **buf: <our input>.** But this is a call for getting the entire input of test.txt
What we want to create is a buffer which will go through the contents of the file up to a defined size. As the task specifies, we are to create a function buf_in of size 16 bytes (stores 16 chars) which will store the 16th first letters of the text-file we have done. This is implemented to have a faster return of the input and is the concept of buffering.

Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya

In the course text-book, the instructions are specified as:



Sec. 17.20 Effectiveness Of Buffering 351

Setup(N)

1. Allocate a buffer of N bytes.
2. Create a global pointer, p, and initialize p to indicate that the buffer is empty.

Input(N)

1. If the buffer is empty, make a system call to fill the entire buffer, and set pointer p to the start of the buffer.
2. Extract a byte, D, from the position in the buffer given by pointer p, move p to the next byte, and return D to the caller.

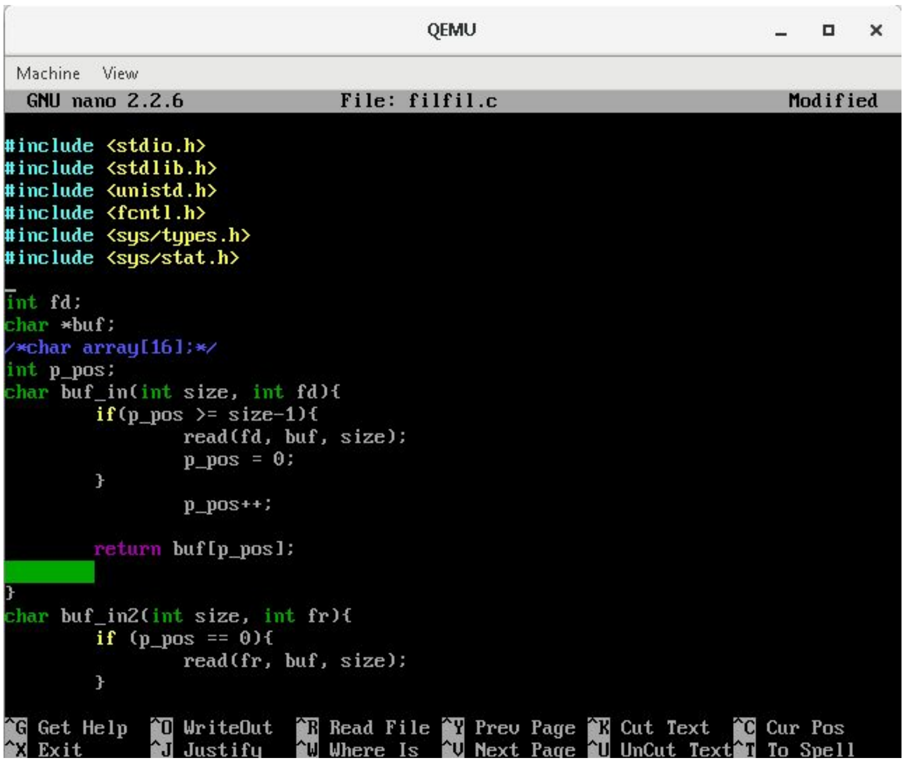
Terminate

1. If the buffer was dynamically allocated, deallocate it.

Figure 17.11 The steps required to achieve buffered input.

Figure 2. - textbook steps of assignment

So the first thing we have done is allocating a char type buffer globally for our function, then for input if the buffer is empty we use the read operation for each character in the initial test file, which is called upon using basic flag notation O_RDWR; and read function. Since the buf_in function is called many times and our incrementation variable p_pos is increased each time, upon reading it may not start at p_pos=0 which is why we set it to 0 if its 15 or 16 (size-1 or size).



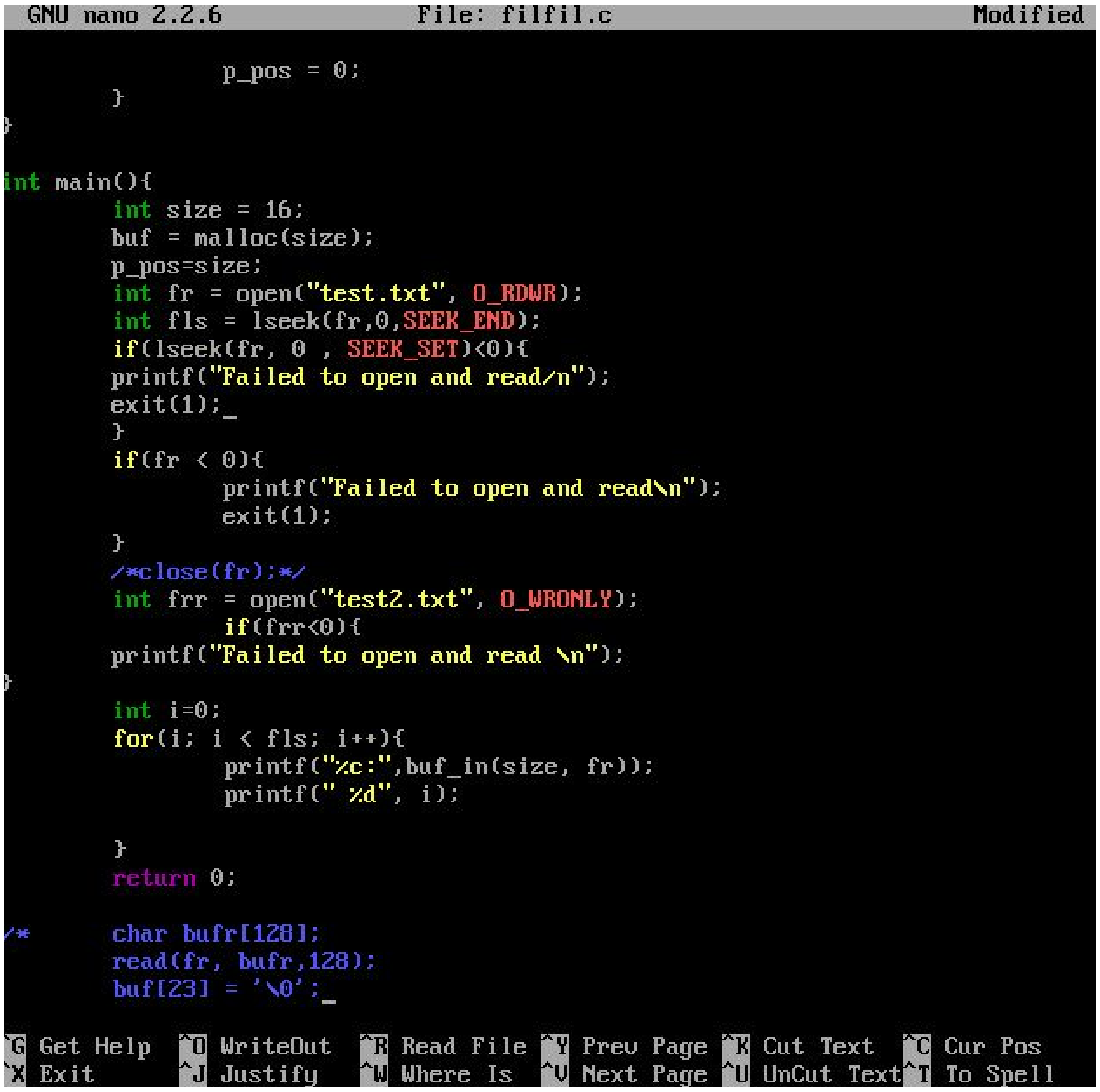
```
  GNU nano 2.2.6              File: filfil.c                        Modified

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int fd;
char *buf;
/*char array[16];*/
int p_pos;
char buf_in(int size, int fd){
        if(p_pos >= size-1){
                read(fd, buf, size);
                p_pos = 0;
        }
                p_pos++;

        return buf[p_pos];

}
char buf_in2(int size, int fr){
        if (p_pos == 0){
                read(fr, buf, size);
        }

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 3. - buf in function

Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya

Main function



```
  GNU nano 2.2.6                File: filfil.c                          Modified

                p_pos = 0;
        }
}

int main(){
        int size = 16;
        buf = malloc(size);
        p_pos=size;
        int fr = open("test.txt", O_RDWR);
        int fls = lseek(fr,0,SEEK_END);
        if(lseek(fr, 0 , SEEK_SET)<0){
        printf("Failed to open and read/n");
        exit(1);_
        }
        if(fr < 0){
                printf("Failed to open and read\n");
                exit(1);
        }
        /*close(fr);*/
        int frr = open("test2.txt", O_WRONLY);
                if(frr<0){
        printf("Failed to open and read \n");
}
        int i=0;
        for(i; i < fls; i++){
                printf("%c:",buf_in(size, fr));
                printf(" %d", i);

        }
        return 0;

/*      char bufr[128];
        read(fr, bufr,128);
        buf[23] = '\0';_

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 5. - main function of our task 1.

For reading the character (letters of alphabet) in our file we need to know how large the file is and for this we use lseek function. it is written in syntax lseek(const char *filename, int offset, flag).

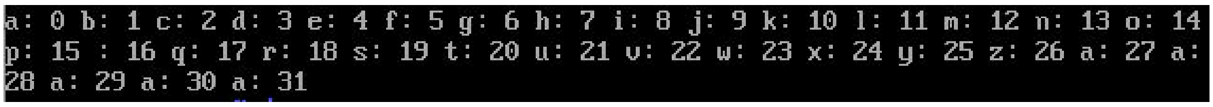
We set the offset and end of file to seek. Because we will always want to go through the whole file for this we need only use the seek to end and offset 0. If we would like to maybe count from the nth character in our file to end of file we could have something like

Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya

lseek(filename,0,seek end) - lseek(filename, n, seek set). Then we chose to print out both the index (position number in our array) and the actual character.

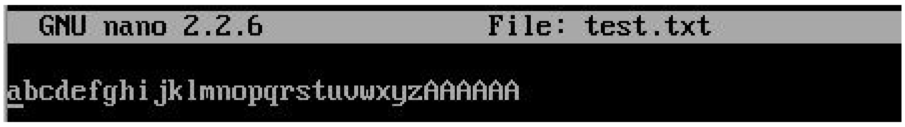
For input we had “abcdefghijklmnopqrstuwxyzaaaaaa” in text file, the small letters alphabet which is 26 bytes in our storage and 6 additional a:s. Output returned is:



```
a: 0 b: 1 c: 2 d: 3 e: 4 f: 5 g: 6 h: 7 i: 8 j: 9 k: 10 l: 11 m: 12 n: 13 o: 14
p: 15 : 16 q: 17 r: 18 s: 19 t: 20 u: 21 v: 22 w: 23 x: 24 y: 25 z: 26 a: 27 a:
28 a: 29 a: 30 a: 31
```

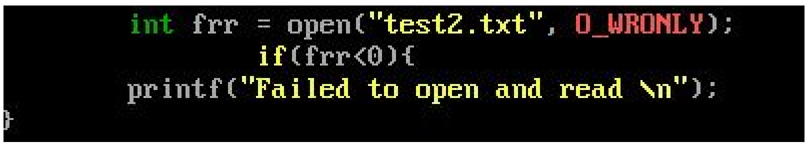
Figure 6. - Output of buf in.

Task 2: Buffered output
For buffered output we will use the same concept of buffering and it will be also 16 bytes of data from our 32-byte test.txt file. It will again contain the alphabet in our test.txt file which is read in our main -> “abcdefghijklmnopqrstuwxyzAAAAAA”.



```
  GNU nano 2.2.6              File: test.txt

abcdefghijklmnopqrstuvwxyzAAAAAA
```

We are going to use the input buffer from task one to read characters and make it for output buffer. For this we will start by creating a file with Write-only flag in our main. In the first task we wanted only read and this task we want to write.



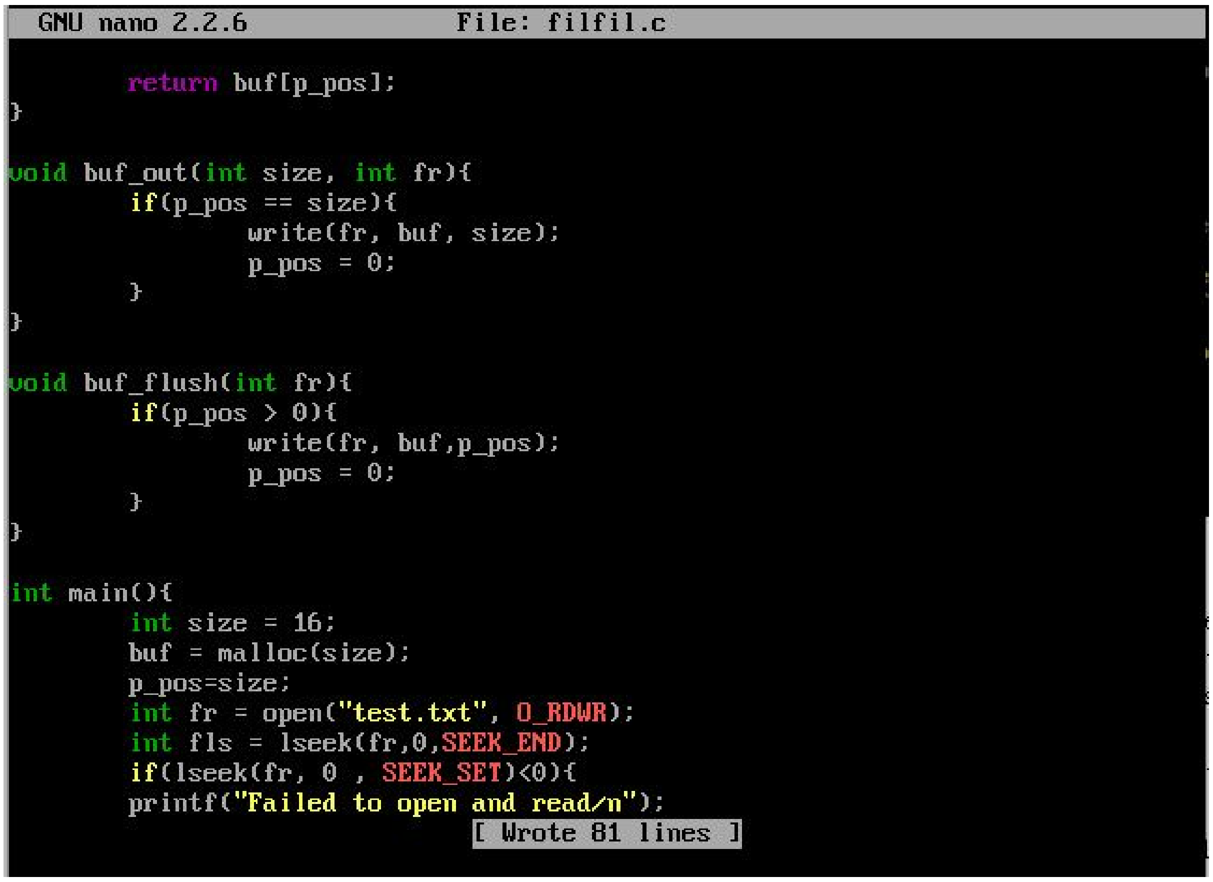
```
        int frr = open("test2.txt", O_WRONLY);
                if(frr<0){
        printf("Failed to open and read \n");
}
```

This text file will be empty for it will be written into and output from our test.txt file.

```
  GNU nano 2.2.6              File: test2.txt                          Modified
```

Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya



```
  GNU nano 2.2.6              File: filfil.c

        return buf[p_pos];
}

void buf_out(int size, int fr){
        if(p_pos == size){
                write(fr, buf, size);
                p_pos = 0;
        }
}

void buf_flush(int fr){
        if(p_pos > 0){
                write(fr, buf,p_pos);
                p_pos = 0;
        }
}

int main(){
        int size = 16;
        buf = malloc(size);
        p_pos=size;
        int fr = open("test.txt", O_RDWR);
        int fls = lseek(fr,0,SEEK_END);
        if(lseek(fr, 0 , SEEK_SET)<0){
        printf("Failed to open and read/n");
                              [ Wrote 81 lines ]
```

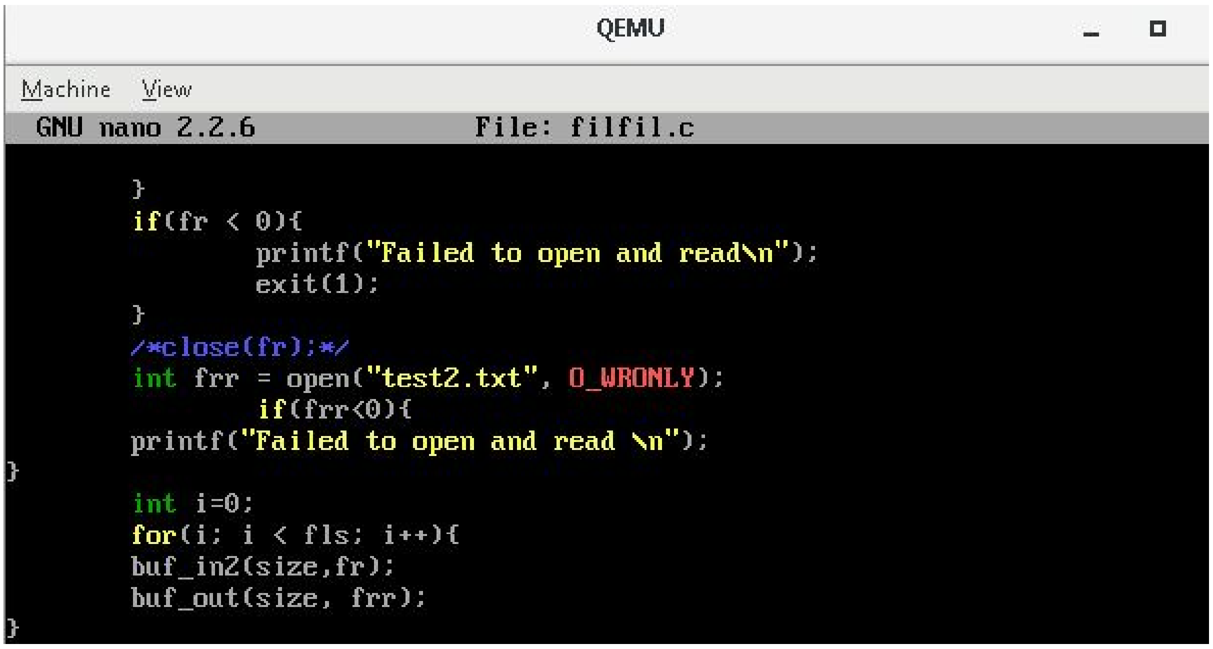
Figure 7. - Task 2 functions buf out and flush out

Our buf out function is based on the buf in function by using same arguments and method of write(fr,buf,size)... but replaced access to file - instead of read we use write for the output to be written. Also as the instructions of task ask, if the buffer is full we are to write the contents and after this we set the p_pos variable to zero for the next implementation.

• If the buffer is full, buf_out should write the contents to the file.

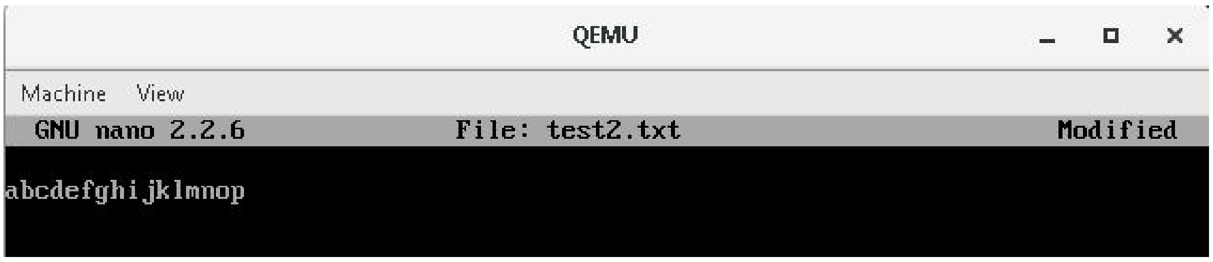
• Implement an additional function buf_flush which forces the contents of the write buffer to be written into the file.

The buf_flush function will write the contents of the write buffer into our test2.txt file built on same method and arguments but this will not only print if our buffer is full rather as long as it is not empty, so there can be any number of chars from >0 to size which is 16. When we call upon this function in our main we will iterate our counter i which will be index of each character in our buffer. This can be seen in our for loop below. Again we use the file size with function lseek as in previous task.



```
  GNU nano 2.2.6              File: filfil.c

        }
        if(fr < 0){
                printf("Failed to open and read\n");
                exit(1);
        }
        /*close(fr);*/
        int frr = open("test2.txt", O_WRONLY);
                if(frr<0){
        printf("Failed to open and read \n");
}
        int i=0;
        for(i; i < fls; i++){
        buf_in2(size,fr);
        buf_out(size, frr);
}
```

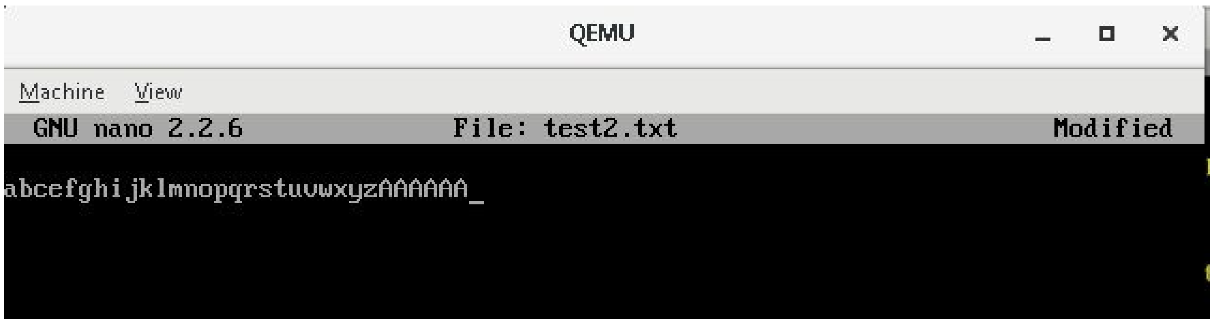
Frr is our open("test2.txt, O_WRONLY) as can be seen above. So the output should be stored into this buffer. Upon calling this function we receive output:



```
  GNU nano 2.2.6             File: test2.txt                       Modified

abcdefghijklmnop
```

It has written the indexed letters 0 to 16 (size) from our test.txt file into this.

Our buf_flush will output::



```
  GNU nano 2.2.6             File: test2.txt                       Modified

abcefghijklmnopqrstuvwxyzAAAAAA_
```

Datorteknik Lab 4: Input-Output Operations and Buffers

Alen Gazibegovic
Thanadon Suriya