Datateknik

Lab 4: Input-Output Operations and Buffers

Alfred Elfving
Fahad Rejabo

Materials

• ThinLinc client on windows PC
• ThinLinc server running Ubuntu

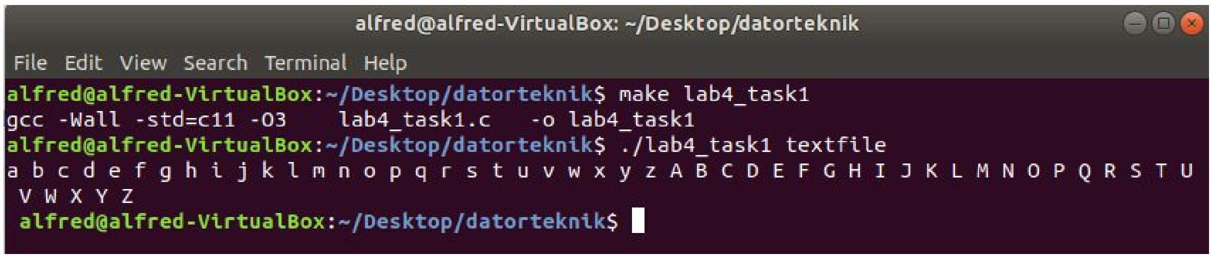
Task 1: Buffered Input

The task was to create a C program which uses C API, system calls to write to a buffer in binary mode. We created an external text file which contained random data, we then called the the C program to read the file and extract 16 bytes at a time from the file to our buffer. i.e. **Buf_in** should read 16 bytes from the input file and save them in the buffer and next calls it should return the next byte in the buffer. Code below shows how buf_in are implemented. Buf_in checks if the buffer is full by comparing the current position to the buffer length. When the position is equal to the buffer length the position is reset to zero and after in the loop import data into the buffer.

The code

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#define BUFFSIZE 16
int position = 0;

char buf_in(int file, char buff[BUFFSIZE]){
        if (position >= BUFFSIZE-1){
                position = 0;
        }
        if(read(file, buff, (ssize_t) 1)){
                position++;
                return 0;
        }

                return EOF;
}

int main(int argc, char* argv[]){
        if (argc == 2){
                char buffer[BUFFSIZE];
                int input_file;

                input_file = open(argv[1],O_RDONLY);
                while (buf_in(input_file, buffer) != EOF){
                        /*write the latest char that we just read in*/
                        if (write(1, &buffer, sizeof(char))){
                                /*write a space after each read character*/
                                char space = ' ';
                                if (write(1, &space, sizeof(char))){
                                        //
                                }

                        }
                }

                close(input_file);
        }else{
                printf("You need to enter the file as argument!\n");
        }

        return 0;
}
```

The program execution ('textfile' contained alphabet a-z and A-Z, program prints each byte + a space)


```
alfred@alfred-VirtualBox:~/Desktop/datorteknik$ make lab4_task1
gcc -Wall -std=c11 -O3    lab4_task1.c    -o lab4_task1
alfred@alfred-VirtualBox:~/Desktop/datorteknik$ ./lab4_task1 textfile
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U
 V W X Y Z
 alfred@alfred-VirtualBox:~/Desktop/datorteknik$
```

The file size is 32 bytes and **buf_in** is called the same number of times (from a-Z=32 times).

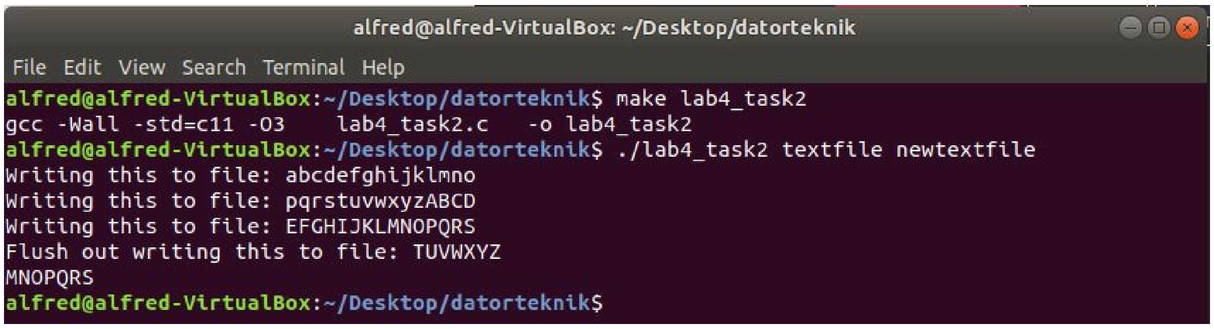
Task 2: Buffered Output

The task was an extension of previous task, we were assigned to do a buffered output as well as doing the buffered input. First we had to read an external file as previously, and then we were to write the contents of the buffer to another new file. The buffered output also uses 16 bytes at the time. When the buffer is full a function called **buf_out** should be called, this function writes all the contents which are stored in the buffer to an output text file. Another function **buf_flush** is also implemented in case the buffer is not full and to force output the actual data in the buffer to a file. Code below shows how buf_in, buf_out and flush_out are implemented:

The code

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#define BUFFSIZE 16
int position = 0;
char buffer[BUFFSIZE];



char buf_in(int file, char buff[1]){
	if (position >= BUFFSIZE-1){
		position = 0;
		return 0;
	}
	if(read(file, buff, 1)){
		position++;
		return 0;
	}
		return EOF;
}

int buf_out(int file){
	if (position >= BUFFSIZE-1){ //ggwgw
		if (write(file, &buffer, BUFFSIZE-1)){
			printf("Writing this to file: %s\n", buffer);
			return 0;
		}
	}
	return -1;
}

void flush_out(int file, char buff[BUFFSIZE]){
	if (position > 0){
		if (write(file, buff, position-1) > 0){
			printf("Flush out writing this to file: %s\n", buffer);
		}
	}
}

int main(int argc, char* argv[]){
	if (argc == 3){
		//char buffer[BUFFSIZE];
		int input_file, output_file;
		input_file = open(argv[1],O_RDONLY);
		output_file = open(argv[2],O_WRONLY | O_CREAT, 0644);

		while (buf_in(input_file, &(buffer[position])) != EOF){
			buf_out(output_file);
		}
		flush_out(output_file, buffer);

		close(input_file);
	}else{
		printf("You need to enter input file and output file as arguments!\n");
	}

	return 0;
}
```

The execution (same ‘textfile’ as before, now creating a new file to output ‘newtextfile’)


alfred@alfred-VirtualBox: ~/Desktop/datorteknik
File Edit View Search Terminal Help

```
alfred@alfred-VirtualBox:~/Desktop/datorteknik$ make lab4_task2
gcc -Wall -std=c11 -O3    lab4_task2.c   -o lab4_task2
alfred@alfred-VirtualBox:~/Desktop/datorteknik$ ./lab4_task2 textfile newtextfile
Writing this to file: abcdefghijklmno
Writing this to file: pqrstuvwxyzABCD
Writing this to file: EFGHIJKLMNOPQRS
Flush out writing this to file: TUVWXYZ
MNOPQRS
alfred@alfred-VirtualBox:~/Desktop/datorteknik$
```