Datorteknik, HT18, DT509G
Lab 4: Input-Output Operations and Buffers

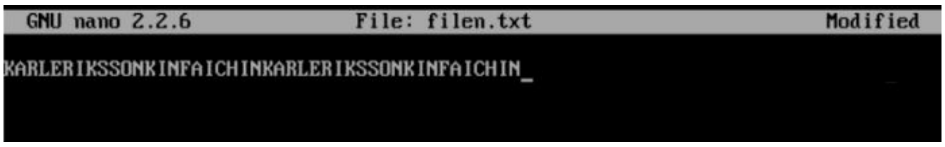
Kinfai Chin & Karl Eriksson

2018-10-23

Task 1: Buffered input (7 points)

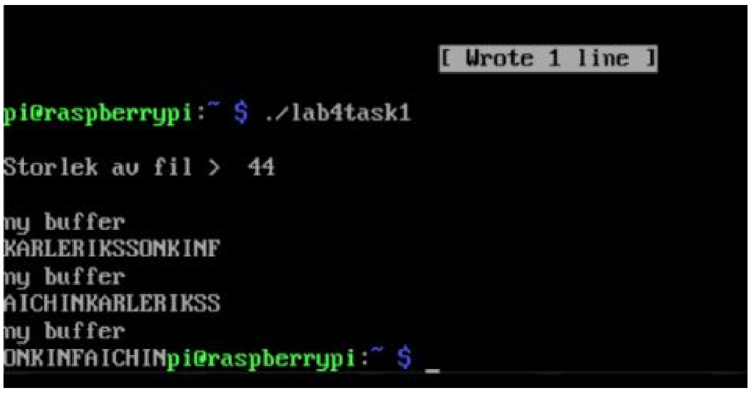
The tasks was to implement read and write buffers. Instead of just using a single string, a struct was created with essential components. (Codeline 9)

The buf_in function takes as arguments: a pointer to read buffer element and filename to reading destination. The way the function works is that the buffer struct have an integer that keeps count of the position in the buffer array(starts at 0). Every time the function runs the position value is incremented and returns the next element in the buffer. If the buffer is empty or the position is at the end of the array the function reads from the file and inserts new elements to the buffer.(codeline 23)



```
 GNU nano 2.2.6              File: filen.txt                    Modified
KARLERIKSSONKINFAICHINKARLERIKSSONKINFAICHIN_
```

A text file of size 44 was used to test the buf_in function.

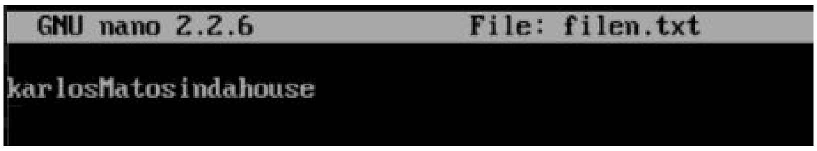


```
                                         [ Wrote 1 line ]
pi@raspberrypi:~ $ ./lab4task1
Storlek av fil >  44
ny buffer
KARLERIKSSONKINF
ny buffer
AICHINKARLERIKSS
ny buffer
ONKINFAICHINpi@raspberrypi:~ $ _
```

The output from the test of buf_in. buf_in filled the buffer 3 times as indicated by the prints “ny buffer”.

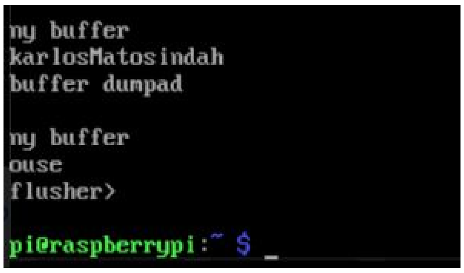
Task 2: Buffered output (8 points)

Another buffer was created for the buf_out and buf_flush functions. The arguments for buf_out is: The name of the file to write to, pointer to write buffer and a character to store in the buffer. Each time the buf_out function runs, it adds a character to the buffer and when it is full the function writes to the file.(codeline 34) In cases where the buffer can not be filled to the maximum the function buf_flush comes to use. The arguments for buf_flush is: filename to write to, the size of the file and a pointer to the write buffer. Filesize modulo buffersize gives the amount of elements in the last non-filled buffer and then it writes that amount from the buffer to the file.(codeline 44)



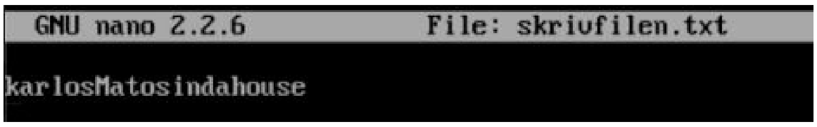
```
GNU nano 2.2.6              File: filen.txt

karlosMatosindahouse
```

The file to fill the write buffer from.



```
ny buffer
karlosMatosindah
buffer dumpad

ny buffer
ouse
flusher>

pi@raspberrypi:~ $ _
```

This output is given by running a read buffer that gives elements to the write buffer and then flush at the end.

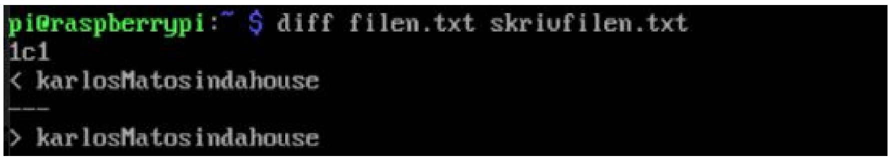


```
GNU nano 2.2.6              File: skrivfilen.txt

karlosMatosindahouse
```

The resulting text file given by reading and writing.

Task 3: Performance evaluation (10 points)

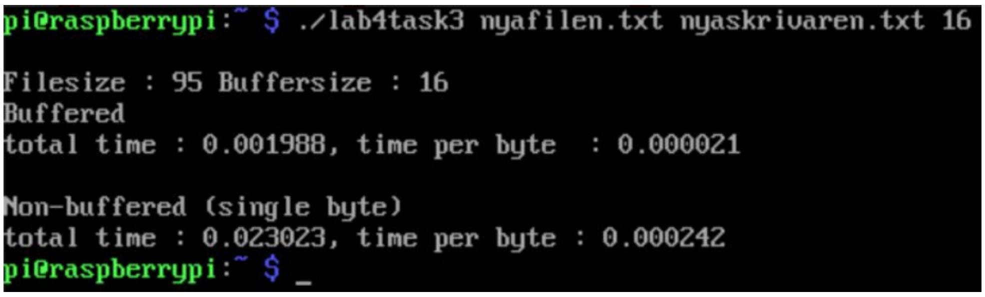
At codeline 75-79 the copying of a file is made with a simple while loop (iteration through all the elements of the file).



```
pi@raspberrypi:~ $ diff filen.txt skrivfilen.txt
1c1
< karlosMatosindahouse
---
> karlosMatosindahouse
```

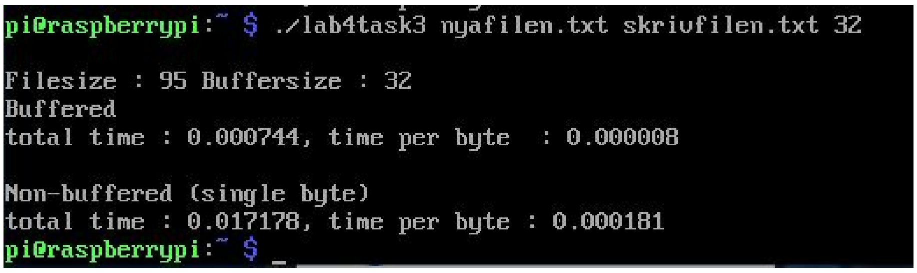
The shell command **diff** on read and write text files.

A non buffered part of the copying was made at codeline 91-100. The time of that copying was timed and divided by the file size. The result is the time it takes to read and write a single byte. The time of the buffered copying was timed as well.(codeline 74-81)



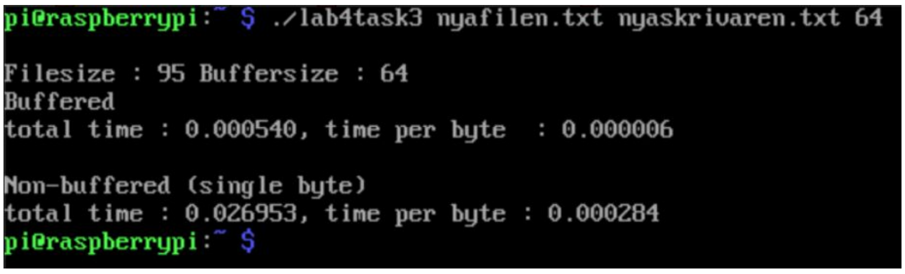
```
pi@raspberrypi:~ $ ./lab4task3 nyafilen.txt nyaskrivaren.txt 16

Filesize : 95 Buffersize : 16
Buffered
total time : 0.001988, time per byte  : 0.000021

Non-buffered (single byte)
total time : 0.023023, time per byte : 0.000242
pi@raspberrypi:~ $ _
```

The results of timing with buffer size of 16 bytes.



```
pi@raspberrypi:~ $ ./lab4task3 nyafilen.txt skrivfilen.txt 32

Filesize : 95 Buffersize : 32
Buffered
total time : 0.000744, time per byte  : 0.000008

Non-buffered (single byte)
total time : 0.017178, time per byte : 0.000181
pi@raspberrypi:~ $ _
```

The results of timing with buffer size of 32 bytes

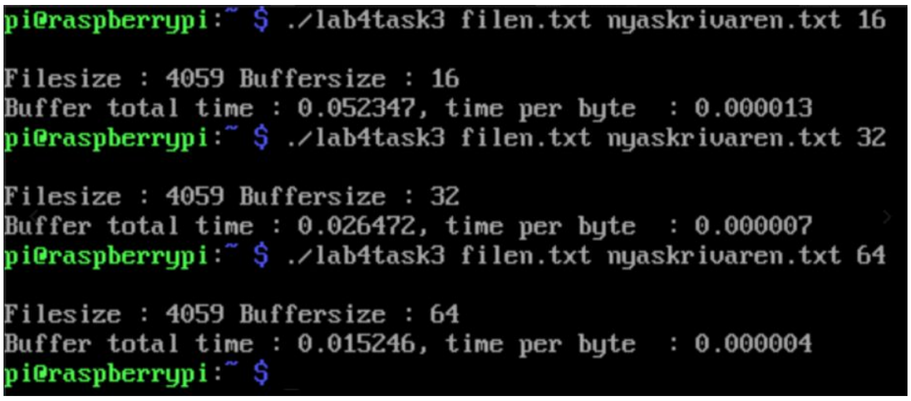


```
pi@raspberrypi:~ $ ./lab4task3 nyafilen.txt nyaskrivaren.txt 64

Filesize : 95 Buffersize : 64
Buffered
total time : 0.000540, time per byte  : 0.000006

Non-buffered (single byte)
total time : 0.026953, time per byte : 0.000284
pi@raspberrypi:~ $
```

.

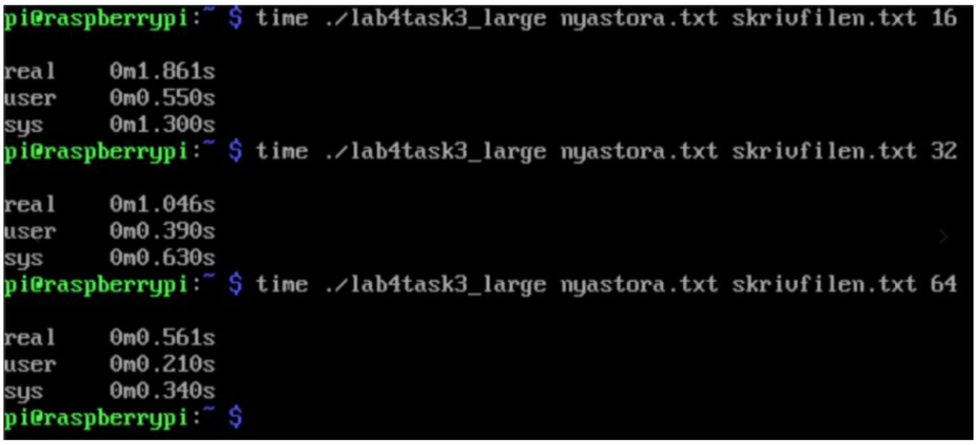
The results of timing with buffer size of 64 bytes.

The copying with buffers was also made on a larger file of size 4059 bytes.



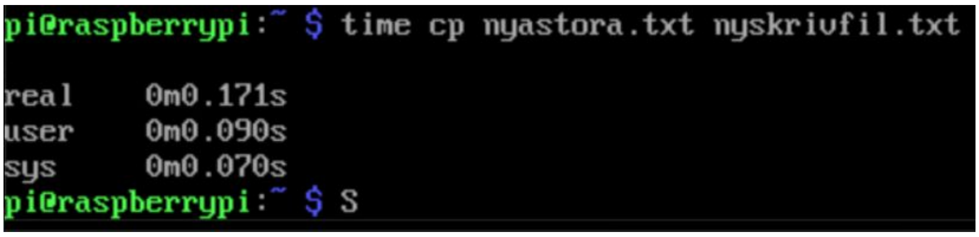
```
pi@raspberrypi:~ $ ./lab4task3 filen.txt nyaskrivaren.txt 16

Filesize : 4059 Buffersize : 16
Buffer total time : 0.052347, time per byte  : 0.000013
pi@raspberrypi:~ $ ./lab4task3 filen.txt nyaskrivaren.txt 32

Filesize : 4059 Buffersize : 32
Buffer total time : 0.026472, time per byte  : 0.000007
pi@raspberrypi:~ $ ./lab4task3 filen.txt nyaskrivaren.txt 64

Filesize : 4059 Buffersize : 64
Buffer total time : 0.015246, time per byte  : 0.000004
pi@raspberrypi:~ $
```

Resulting times of copying a larger file.



```
pi@raspberrypi:~ $ time ./lab4task3_large nyastora.txt skrivfilen.txt 16

real    0m1.861s
user    0m0.550s
sys     0m1.300s
pi@raspberrypi:~ $ time ./lab4task3_large nyastora.txt skrivfilen.txt 32

real    0m1.046s
user    0m0.390s
sys     0m0.630s
pi@raspberrypi:~ $ time ./lab4task3_large nyastora.txt skrivfilen.txt 64

real    0m0.561s
user    0m0.210s
sys     0m0.340s
pi@raspberrypi:~ $
```

Result of using the shell command **time** when copying a large file(100Kb) with different buffer sizes.



```
pi@raspberrypi:~ $ time cp nyastora.txt nyskriufil.txt

real	0m0.171s
user	0m0.090s
sys	0m0.070s
pi@raspberrypi:~ $ S
```

Times from using the shell command **cp** on the large file(100Kb).

Code:

```c
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>


typedef struct{
    int size;
    int pos;
    char *buffer_array;
}buffer;

buffer * buffer_init(int size){
    buffer * buff = malloc(sizeof(buffer));
    buff->buffer_array = (char*)malloc(sizeof(char)*size);
    buff->size=size;
    buff->pos=0;
    return buff;
}

char buf_in(int file,buffer *buff){
    if((buff->pos % buff->size) == 0){
            read(file,buff->buffer_array,buff->size);
            buff->pos = 0;
    }
    char returnvalue = buff->buffer_array[buff->pos];
    buff->pos = buff->pos + 1;

    return returnvalue;
}

void buf_out(int output, buffer * buff, char value){
    buff->buffer_array[buff->pos] = value;
    buff->pos = buff->pos + 1;
    if(buff->pos == buff->size){
            write(output,buff->buffer_array,buff->size);
            buff->pos = 0;
    }
    return;
}

void buf_flush(int output, int filesize, buffer *buff){
    int elements = (filesize) %  buff->size;

    write(output,buff->buffer_array,elements);
    return;
}

int main(int argc, char * argv[]){
    int file;
    int output;
    output = open(argv[2], O_CREAT | O_RDWR | O_TRUNC);
    file = open(argv[1],O_CREAT | O_RDWR);
    int fileSize = lseek(file,0,SEEK_END) - 1;
    lseek(file,0,SEEK_SET);
    lseek(output,0,SEEK_SET);
    if(file < 0){
            printf("\nthe filer is empty\n");
            return 0;
    }
    printf("\nFilesize : %d Buffersize : %d",fileSize, atoi(argv[3]));
    buffer * buff_in = buffer_init(atoi(argv[3]));
    buffer * buff_out = buffer_init(atoi(argv[3]));
    int loop = 0;
    double totaltime = 0;
    while(loop <=100){
    lseek(file,0,SEEK_SET);
    lseek(output,0,SEEK_SET);
    int k = 0;
    clock_t start;
    clock_t end;
    start = clock();
    while (k < fileSize){
            buf_out(output,buff_out, buf_in(file,buff_in));
            k++;
    }
    buf_flush(output,fileSize,buff_out);
    end = clock();
    totaltime = totaltime + ((double)(end - start))/CLOCKS_PER_SEC;
    loop++;
    }
    printf("\nBuffered\ntotal time : %f, time per byte  : %f\n", totaltime/100, totaltime/(100*fileSize));
    int looper2 = 0;
    totaltime = 0;
    while(looper2<=10){
    char byte[1];
    lseek(file,0,SEEK_SET);
    lseek(output,0,SEEK_SET);
    clock_t start = clock();
    for(loop = 0;loop < fileSize;loop++){
            read(file,byte,1);
            write(output,byte,1);
    }
    clock_t end = clock();
    totaltime = totaltime + ((double)(end - start))/CLOCKS_PER_SEC;
    looper2++;
    }
    printf("\nNon-buffered (single byte)\ntotal time : %f, time per byte : %f\n",totaltime/10, totaltime/(fileSize*10));
    close(file);
    close(output);
    return;
}

```