Andreas Vasberg, Patrik Viitala

Lab 4 - Group 5

The beginning of the lab went on very smooth and unlike the first two labs we didn't have any problem with the setup. The understanding of the process were kinda easy except for some minor undeclared things. For example if buf_in would return the first value when it reads from the text-file or if it should return the first value the second time buf_in is called.

Task 1:

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

char* read_ptr;

char buf_in(char buffer[], int fd, int buffer_size)
{
    char next;
    if(*read_ptr == '\0')
    {
        read_ptr = buffer;
        int progress = read(fd, buffer, buffer_size);
        if(progress == 0){
            next = '\0';
            buffer[0] = '\0';
            return next;
        }
        buffer[progress] = '\0';
    }
    next = *read_ptr;
    read_ptr++;
    return next;
}

int main()
{
    int buffer_size = 16;
    char buffer[buffer_size + 1];
    buffer[0] = '\0';
    read_ptr = buffer;
    int fd = open("read.txt", O_RDONLY, 0);
    char c = '0';

    while(1)
    {
```

```
        c = buf_in(buffer, fd, buffer_size);
        if(c == '\0')
        {
            break;
        }
        printf("%c", c);
    }
    printf("\n");
    close(fd);

    return 0;
}
```

Buf_in takes a char array, a text file (int to keep track of progress in document) and buffer size as arguments and returns the next character of the buffer. If the buffer is empty, it will read from the text file and return the first character. The buffer is handled by a pointer. It is initially set to point to the first character of the buffer. On each call, the pointer is moved one step to the right until it reaches the end of the buffer. If that happens, the buffer will be refilled on the next call and the pointer is set to point at the start again.

Task 2:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

char* write_ptr;

void buf_out(char c, char buffer[], int fd, int buffer_size)
{
    if(&buffer[buffer_size] == write_ptr)
    {
        write_ptr = buffer;
        int progress = write(fd, buffer, buffer_size);
    }
    *write_ptr = c;
    write_ptr++;
    *write_ptr = '\0';
}

void buf_flush(char buffer[], int fd)
{
    if(buffer[0] != '\0')
    {
```

```
        write(fd, buffer, strlen(buffer));
        buffer[0] = '\0';
        write_ptr = buffer;
    }
}

int main()
{
    int buffer_size = 16;
    char buffer[buffer_size + 1];
    buffer[0] = '\0';
    write_ptr = buffer;
    int fd = open("write.txt", O_WRONLY, 0);
    char* c = "asidlfjasldkfjasflkjklhhkhjhjhjhjhjhjhjhlhjhlllhjk";
    int i = 0;

    while(i < strlen(c))
    {
        buf_out(c[i], buffer, fd, buffer_size);
        i++;
    }
    buf_flush(buffer, fd);

    close(fd);

    return 0;
}
```

Buf_out takes a char c, char array buffer, a text file and buffer size and does not return anything. It checks if the buffer is full and in that case write everything to the argument file, otherwise add the given character to the current address of the pointer.
Buf_flush takes char array buffer and a text file as arguments and writes every existing character from buffer to the argument file, regardless of the size.

Task 3:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

char* read_ptr;
char* write_ptr;

char buf_in(char buffer[], int fd, int buffer_size)
```

```
{
    char next;
    if(*read_ptr == '\0')
    {
        read_ptr = buffer;
        int progress = read(fd, buffer, buffer_size);
        if(progress == 0){
            next = '\0';
            buffer[0] = '\0';
            return next;
        }
        buffer[progress] = '\0';
    }
    next = *read_ptr;
    read_ptr++;
    return next;
}

void buf_out(char c, char buffer[], int fd, int buffer_size)
{
    if(&buffer[buffer_size] == write_ptr)
    {
        write_ptr = buffer;
        int progress = write(fd, buffer, buffer_size);
    }
    *write_ptr = c;
    write_ptr++;
    *write_ptr = '\0';
}

void buf_flush(char buffer[], int fd)
{
    if(buffer[0] != '\0')
    {
        write(fd, buffer, strlen(buffer));
        buffer[0] = '\0';
        write_ptr = buffer;
    }
}

int main(int argc, char* argv[])
{
    int buffer_size = 16;
    char read_buffer[buffer_size + 1];
    char write_buffer[buffer_size + 1];
    read_buffer[0] = '\0';
    write_buffer[0] = '\0';
    read_ptr = read_buffer;
```

```
    write_ptr = write_buffer;
    int fd1 = 0; int fd0 = 0;
    if(argc == 3)
    {
        fd0 = open(argv[1], O_RDONLY, 0);
        fd1 = open(argv[2], O_WRONLY, 0);
    }
    else
    {
        fd0 = open("read.txt", O_RDONLY, 0);
        fd1 = open("write.txt", O_WRONLY, 0);
    }
    char c = '0';

    while(1)
    {
        c = buf_in(read_buffer, fd0, buffer_size);
        if(c == '\0')
        {
            break;
        }
        buf_out(c, write_buffer, fd1, buffer_size);
    }
    buf_flush(write_buffer, fd1);

    close(fd1);
    close(fd0);

    return 0;
}
```

In this task, the code from the former tasks were fused together. The characters are retrieved with buf_in and transferred to buf_out. This time around, the filenames are passed through the command-line. (If no arguments are passed, it defaults to other names).

Diff:

```
patvih161@ltse01:~/datateknik/labb4$ diff read.txt write.txt
patvih161@ltse01:~/datateknik/labb4$
```

No difference was observed between the copy and the original file.

Buffered vs unbuffered:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <time.h>
```

```
char* p;

char buf_in(char buffer[], int fd, int buffer_size)
{
    char next;
    if(*p == '\0')
    {
        p = buffer;
        int progress = read(fd, buffer, buffer_size);
        if(progress == 0){
              next = '\0';
              buffer[0] = '\0';
              return next;
        }
    buffer[progress] = '\0';
    }
    next = *p;
    p++;
    return next;
}

int main()
{
    int buffer_size = 16;
    char buffer[buffer_size + 1];
    buffer[0] = '\0';
    p = buffer;
    int fd = open("read.txt", O_RDONLY, 0);
    char c = '0';

    clock_t begin = clock();
    while(1)
    {
        c = buf_in(buffer, fd, buffer_size);
        if(c == '\0')
        {
            break;
        }
        printf("%c\n", c);
    }

    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("%lf\n", time_spent);
```

```
    close(fd);

    return 0;
}
```

The code for measuring the buffered input.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

char* p;

void buf_out(char c, char buffer[], int fd, int buffer_size)
{
    if(&buffer[buffer_size] == p)
    {
        p = buffer;
        int progress = write(fd, buffer, buffer_size);
    }
    *p = c;
    p++;
    *p = '\0';
}

void buf_flush(char buffer[], int fd)
{
    if(buffer[0] != '\0')
    {
        write(fd, buffer, strlen(buffer));
        buffer[0] = '\0';
        p = buffer;
    }
}

int main()
{
    int buffer_size = 16;
    char buffer[buffer_size + 1];
    buffer[0] = '\0';
    p = buffer;
    int fd = open("write.txt", O_WRONLY, 0);
    char* c = "a";
```

```
    int i = 0;

    clock_t begin = clock();
    while(i < strlen(c))
    {
        buf_out(c[i], buffer, fd, buffer_size);
        i++;
    }
    buf_flush(buffer, fd);

    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("%lf\n", time_spent);

    close(fd);

    return 0;
}
```

The code for measuring the buffered output.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <time.h>

int main()
{
    int fd0 = open("read.txt", O_RDONLY, 0);
    int fd1 = open("write.txt", O_WRONLY, 0);
    char c = 'a';

    clock_t begin = clock();

    /*read(fd0, &c, 1);
    printf("%c\n", c)*/
    write(fd1, &c, 1);

    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("%lf\n", time_spent);

    close(fd1);
    close(fd0);
```

```
    return 0;
}
```

The code for measuring the unbuffered input and output. For measuring reading, the read line is used. For measuring writing, the write line is used.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_byte_buf
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000084
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000196
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000075
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000064
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000077
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000072
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000081
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000074
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000088
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_buf
a
0.000097
patvih161@ltse01:~/datateknik/labb4$
```

Average read with buffered input = 0.0000908 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc unbuffered.c -ansi -o read_byte_unbuf
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000154
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000076
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000593
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000293
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000090
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000316
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000088
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000079
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000085
patvih161@ltse01:~/datateknik/labb4$ ./read_byte_unbuf
a
0.000062
patvih161@ltse01:~/datateknik/labb4$
```

Average read with unbuffered input = 0.0001836 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task2a.c -ansi -o write_byte_buf
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000028
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000041
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000052
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000042
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000021
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000021
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000028
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000021
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000026
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_buf
0.000055
patvih161@ltse01:~/datateknik/labb4$
```

Average write with buffered output = 0.0000335 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc unbuffered.c -ansi -o write_byte_unbuf
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000029
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000074
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000039
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000044
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000038
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000036
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000027
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000071
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000039
patvih161@ltse01:~/datateknik/labb4$ ./write_byte_unbuf
0.000040
patvih161@ltse01:~/datateknik/labb4$
```

Average write with unbuffered output = 0.0000437 seconds.

The point of these results is to show that IO has the biggest impact on performance. All the extra operations that are required to handle the buffers have less importance than the read and write operations. The difference in execution time can depend on the OS.

Comparing buffer sizes:

Reading 16 bytes with 16, 32 and 64 byte buffers:

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_16
patvih161@ltse01:~/datateknik/labb4$ ./read_16
o
v
e
r
c
o
m
p
l
i
c
a
t
i
o
n
0.000132
```

Time to read 16 bytes with 16 byte buffer: 0.000132 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_32
patvih161@ltse01:~/datateknik/labb4$ ./read_32
o
v
e
r
c
o
m
p
l
i
c
a
t
i
o
n
0.000126
patvih161@ltse01:~/datateknik/labb4$
```

Time to read 16 bytes with 32 byte buffer: 0.000126 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_64
patvih161@ltse01:~/datateknik/labb4$ ./read_64
o
v
e
r
c
o
m
p
l
i
c
a
t
i
o
n
0.000122
```

Time to read 16 bytes with 64 byte buffer: 0.000122 seconds.

Reading 32 bytes with 16, 32 and 64 byte buffers:

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_16
patvih161@ltse01:~/datateknik/labb4$ ./read_16
Conjugationally-Hyperexcitement-
0.000146
```

Time to read 32 bytes with 16 byte buffer: 0.000146 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_32
patvih161@ltse01:~/datateknik/labb4$ ./read_32
Conjugationally-Hyperexcitement-
0.000080
```

Time to read 32 bytes with 32 byte buffer: 0.000080 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_64
patvih161@ltse01:~/datateknik/labb4$ ./read_64
Conjugationally-Hyperexcitement-
0.000069
```

Time to read 32 bytes with 64 byte buffer: 0.000069 seconds.

Reading 64 bytes with 16, 32 and 64 byte buffers:

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_16
patvih161@ltse01:~/datateknik/labb4$ ./read_16
Conjugationally-Hyperexcitement-Overdramatizing-Photochemically-
0.000146
```

Time to read 64 bytes with 16 byte buffer: 0.000146 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_32
patvih161@ltse01:~/datateknik/labb4$ ./read_32
Conjugationally-Hyperexcitement-Overdramatizing-Photochemically-
0.000052
```

Time to read 64 bytes with 32 byte buffer: 0.000052 seconds.

```
patvih161@ltse01:~/datateknik/labb4$ gcc task1a.c -ansi -o read_64
patvih161@ltse01:~/datateknik/labb4$ ./read_64
Conjugationally-Hyperexcitement-Overdramatizing-Photochemically-
0.000050
```

Time to read 64 bytes with 64 byte buffer: 0.000050 seconds.

The biggest jump is observed when moving from a 16 byte buffer to a 32 byte buffer. The results can vary a lot, but in theory the bigger buffer size should increase the speed. I/O is the slow part. With a bigger buffer size, the amount of I/O operations decrease, which results in faster execution overall.

100k copy with different buffer sizes:

```
patvih161@tlae02:~/datateknik/labb4$ gcc task3a.c -ansi -o buf_16_100k
patvih161@tlae02:~/datateknik/labb4$ time ./buf_16_100k big_file.txt big_copy.tx
t

real    0m0.028s
user    0m0.000s
sys     0m0.020s
```

16 byte buffer

```
patvih161@tlae02:~/datateknik/labb4$ gcc task3a.c -ansi -o buf_32_100k
patvih161@tlae02:~/datateknik/labb4$ time ./buf_32_100k big_file.txt big_copy.tx
t

real    0m0.016s
user    0m0.000s
sys     0m0.008s
```

Andreas Vasberg, Patrik Viitala

32 byte buffer

```
patvih161@tlae02:~/datateknik/labb4$ gcc task3a.c -ansi -o buf_64_100k
patvih161@tlae02:~/datateknik/labb4$ time ./buf_64_100k big_file.txt big_copy.tx
t

real    0m0.013s
user    0m0.008s
sys     0m0.000s
```

64 byte buffer

Shell command cp:

```
patvih161@tlae02:~/datateknik/labb4$ time cp big_file.txt big_copy.txt

real    0m0.008s
user    0m0.000s
sys     0m0.000s
```

The cp command beats our program.