

Lab 4: Input-Output Operations and Buffers

This is a modified version of our previous report we sent in. During the lab Todor told us to use C standard library but a week after he told us it was wrong. Task 3 is modified to use system calls because we are measuring the time. Task 1 and task 2 still uses the C library because the principal is the same and we are not measuring time. In other words it doesn't matter that the C library implements a buffer also for those two tasks.

Link for the source files:

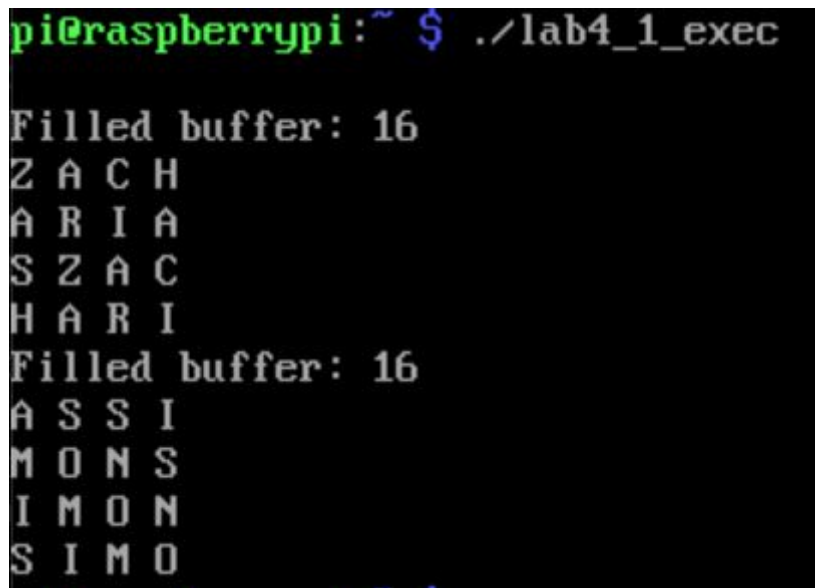
https://drive.google.com/open?id=1h2DDMnTob_lkmOHVffOTnwFwxkV0jeWi

Task 1: Buffered input (7 points)

1. We are opening the file in binary form on line 32.
2. The implementation of the function **buf_in** is at line 19. The functions reads and returns byte by byte but fills the buffer with 16 bytes everytime the buffer is empty.

Result:

As seen in the picture below it reads 32 bytes and refills the buffer one time. In total its two system calls.



```
pi@raspberrypi:~ $ ./lab4_1_exec
Filled buffer: 16
Z A C H
A R I A
S Z A C
H A R I
Filled buffer: 16
A S S I
M O N S
I M O N
S I M O
```

Implementation:

```
1 . #include <stdio.h>
2 . #include <stdlib.h>
3 . #define bufferSize 16
4 . typedef struct{
5 .     char* m_buffer;
6 .     char position;
7 .     char length;
8 . }buffer;
```

```
9 .
10 . buffer* setupBuffer(char size){
11 .     buffer* new_buffer;
12 .     new_buffer = malloc(sizeof(buffer));
13 .     new_buffer->m_buffer = malloc(size);
14 .     new_buffer->position = bufferSize;
15 .     new_buffer->length = bufferSize;
16 .     return new_buffer;
17 . }
18 .
19 . char buf_in(buffer* b, FILE* byteFile){
20 .     if((*b).position >= (*b).length){
21 .         size_t t = fread((void*) b->m_buffer, 1, b->length, byteFile);
22 .         (*b).position = 0;
23 .         printf("Filled buffer: %d\n", t);
24 .     }
25 .
26 .     char byte = *(b->m_buffer + b->position);
27 .     b->position++;
28 .     return byte;
29 . }
30 . int main(){
31 .     buffer* b = setupBuffer(bufferSize);
32 .     FILE* byteFile = fopen("file.txt", "a+b");
33 .     int i;
34 .     for(i = 0; i < 32; i++){
35 .         if(i%4 == 0){
36 .             printf("\n");
37 .         }
38 .         printf("%c ", buf_in(b,byteFile));
39 .     }
40 .     printf("\n");
41 .     fclose(byteFile);
42 .     return 0;
43 . }
44 .
```

Task 2: Buffered output (8 points)

1. The function **buf_out** works in the opposite way of **buf_in**, it writes byte by byte to the buffer and when the buffer is full it writes it's content to output file. This can be found at line 35.
2. The flush function, **buf_flush** writes the content of the buffer to the output file even if the buffer isn't full. If the buffer is empty, nothing will be written.

Result:

As shown below the **buf_flush** function writes two times to the output file (when it writes 5 bytes). The other times it's **buf_out** that calls **buf_flush** to write to file.

```
pi@raspberrypi:~ $ ./lab4_2_exec
```

```
Filled buffer: 16
```

```
Z A C H
```

```
A R I A
```

```
S Z A C
```

```
H A R I Buffer written : 16
```

```
Filled buffer: 16
```

```
A S S I
```

```
M O N S
```

```
I M O N
```

```
S I M O Buffer written : 16
```

```
Buffer written : 5
```

```
Buffer written : 16
```

```
Buffer written : 5
```

Implementation:

```
....
30. void buf_flush(buffer* b, FILE* byteFile){
31.     size_t t = fwrite((void*) b->m_buffer, 1, b->position, byteFile);
32.     (*b).position = 0;
33.     printf("Buffer written : %d\n", t);
34. }
35. void buf_out(buffer* b, FILE* byteFile, char byte){
36.     *(b->m_buffer + b->position) = byte;
37.     b->position++;
38.     if((*b).position >= (*b).length){
39.         buf_flush(b,byteFile);
40.     }
41. }
42.
43. int main(){
44.     buffer* readBuffer = setupBuffer(bufferSize);
45.     FILE* readFile = fopen("file.txt", "rb");
46.     FILE* writeFile = fopen("output.txt", "wb");
47.     buffer* writeBuffer = setupBuffer(bufferSize);
48.     writeBuffer->position = 0;
49.
50.     int i;
51.     for(i = 0; i < 32; i++){
52.         if(i%4 == 0){
53.             printf("\n");
54.         }
55.         char byte = buf_in(readBuffer,readFile);
56.         printf("%c ", byte);
57.         buf_out(writeBuffer, writeFile,byte);
58.     }
59.     buf_out(writeBuffer, writeFile,'L');
60.     buf_out(writeBuffer, writeFile,'E');
61.     buf_out(writeBuffer, writeFile,'S');
62.     buf_out(writeBuffer, writeFile,'S');
63.     buf_out(writeBuffer, writeFile,' ');
64.
65.     buf_flush(writeBuffer, writeFile);
66.
67.     buf_out(writeBuffer, writeFile,'M');
68.     buf_out(writeBuffer, writeFile,'O');
69.     buf_out(writeBuffer, writeFile,'R');
70.     buf_out(writeBuffer, writeFile,'E');
71.     buf_out(writeBuffer, writeFile,' ');
72.     buf_out(writeBuffer, writeFile,'T');
73.     buf_out(writeBuffer, writeFile,'H');
74.     buf_out(writeBuffer, writeFile,'A');
75.     buf_out(writeBuffer, writeFile,'N');
76.     buf_out(writeBuffer, writeFile,' ');
77.     buf_out(writeBuffer, writeFile,'1');
78.     buf_out(writeBuffer, writeFile,'6');
79.     buf_out(writeBuffer, writeFile,' ');
80.     buf_out(writeBuffer, writeFile,'B');
81.     buf_out(writeBuffer, writeFile,'Y');
82.     buf_out(writeBuffer, writeFile,'T');
83.     buf_out(writeBuffer, writeFile,'E');
84.     buf_out(writeBuffer, writeFile,'S');
85.     buf_out(writeBuffer, writeFile,'!');
86.     buf_out(writeBuffer, writeFile,'!');
87.     buf_out(writeBuffer, writeFile,'!');
```

```
88.     buf_flush(writeBuffer, writeFile);
89.
90.     printf("\n");
91.     free(readBuffer->m_buffer);
92.     free(writeBuffer->m_buffer);
93.     fclose(readFile);
94.     fclose(writeFile);
95.     return 0;
96. }
```

Task 3: Performance evaluation (10 points)

1. In the **main** body we open both files, the one to copy from and too. The average time is calculated with `time.h` that shows number of clock ticks. The test is done with different buffer sizes and without a buffer (Implementation 1).
2. We also removed all debug output and then timed the copy with the buffer size of 64 bytes and compared it to the time of the shell `cp` function. The copied file was the 100KB (Implementation 2).

Result:

```
pi@raspberrypi:~ $ gcc -ansi lab4_3.c -o lab4_3_exec
pi@raspberrypi:~ $ ./lab4_3_exec file.txt output.txt
File size in bytes: 1584. BufferSize: 16
```

```
Read/Write buffer time: 35998, avrage: 22.726009
```

```
Read/Write no buffer time: 483922, avrage: 305.506317
```

```
pi@raspberrypi:~ $ gcc -ansi lab4_3.c -o lab4_3_exec
pi@raspberrypi:~ $ ./lab4_3_exec file.txt output.txt
File size in bytes: 1584. BufferSize: 32
```

```
Read/Write buffer time: 6172, avrage: 3.896465
```

```
Read/Write no buffer time: 493701, avrage: 311.679932
```

```
pi@raspberrypi:~ $ gcc -ansi lab4_3.c -o lab4_3_exec
pi@raspberrypi:~ $ ./lab4_3_exec file.txt output.txt
File size in bytes: 1584. BufferSize: 64
```

```
Read/Write buffer time: 2182, avrage: 1.377525
```

```
Read/Write no buffer time: 487572, avrage: 307.810608
```

```
pi@raspberrypi:~ $ time ./lab4_3_large_exec largeFile output.txt
```

```
real    0m0.132s
user    0m0.080s
sys     0m0.050s
```

```
pi@raspberrypi:~ $ diff largeFile output.txt
```

```
pi@raspberrypi:~ $ time cp largeFile output.txt
```

```
real    0m0.141s
user    0m0.050s
sys     0m0.090s
```

Implementation 1 for task 3:

```
....  
3 . #include <time.h>  
....  
43. int main(int argc, char* argv[]){  
44.     FILE* readFile;  
45.     FILE* writeFile;  
46.  
47.     if(argc < 3){  
48.         printf("Using default files!\n");  
49.         readFile = fopen("file.txt", "rb");  
50.         writeFile = fopen("output.txt", "wb");  
51.     }else {  
52.         readFile = fopen(argv[1], "rb");  
53.         writeFile = fopen(argv[2], "wb");  
54.     }  
55.  
56.     buffer* readBuffer = setupBuffer(bufferSize);  
57.     buffer* writeBuffer = setupBuffer(bufferSize);  
58.     writeBuffer->position = 0;  
59.     fseek(readFile, 0, SEEK_END);  
60.     int size = ftell(readFile);  
61.     fseek(readFile, 0, SEEK_SET);  
62.     printf("File size in bytes: %d. BufferSize: %d\n", size, bufferSize);  
63.     int i;  
64.     clock_t start = clock();  
65.     for(i = 0; i < size; i++){  
66.         buf_out(writeBuffer, writeFile, buf_in(readBuffer, readFile));  
67.     }  
68.     buf_flush(writeBuffer, writeFile);  
69.     clock_t elapsed = clock() - start;  
70.  
71.  
72.     fseek(readFile, 0, SEEK_SET);  
73.     fseek(writeFile, 0, SEEK_SET);  
74.     start = clock();  
75.     char byte[1];  
76.     for(i = 0; i < size; i++){  
77.         fread(byte, 1, 1, readFile);  
78.         fflush(readFile);  
79.         fwrite(byte, 1, 1, writeFile);  
80.         fflush(writeFile);  
81.     }  
82.     clock_t elapsed2 = clock() - start;  
83.  
84.     printf("\nRead/Write buffer time: %d, avrage: %f\n", elapsed, elapsed/(float)size);  
85.     printf("\nRead/Write no buffer time: %d, avrage: %f\n", elapsed2, elapsed2/(float)size);  
86.  
87.     free(readBuffer->m_buffer);  
88.     free(writeBuffer->m_buffer);  
89.     fclose(readFile);  
90.     fclose(writeFile);  
91.     return 0;  
92. }  
93.
```

Implementation 2 for task 3(large file):

```
....
43. int main(int argc, char* argv[]){
44.     FILE* readFile;
45.     FILE* writeFile;
46.
47.     if(argc < 3){
48.         readFile = fopen("file.txt", "rb");
49.         writeFile = fopen("output.txt", "wb");
50.     }else {
51.         readFile = fopen(argv[1], "rb");
52.         writeFile = fopen(argv[2], "wb");
53.     }
54.
55.     buffer* readBuffer = setupBuffer(bufferSize);
56.     buffer* writeBuffer = setupBuffer(bufferSize);
57.     writeBuffer->position = 0;
58.     fseek(readFile, 0, SEEK_END);
59.     int size = ftell(readFile);
60.     fseek(readFile, 0, SEEK_SET);
61.     int i;
62.     clock_t start = clock();
63.     for(i = 0; i < size; i++){
64.         buf_out(writeBuffer, writeFile, buf_in(readBuffer,readFile));
65.     }
66.     buf_flush(writeBuffer, writeFile);
67.
68.     free(readBuffer->m_buffer);
69.     free(writeBuffer->m_buffer);
70.     fclose(readFile);
71.     fclose(writeFile);
72.     return 0;
73. }
74.
```


Task 3 (Modified: using system calls instead of C library):

The output is now in seconds instead of clock cycles. Similar result as before that larger buffer size gives faster copy time.

```
pi@raspberrypi:~$ ./lab4_3_MOD_exec file.txt output.txt 16
File size in bytes: 1584. BufferSize: 16

Read/Write buffer time: 4.147300e-02, avrage: 2.618245e-05

Read/Write no buffer time: 3.289230e-01, avrage: 2.076534e-04
pi@raspberrypi:~$ ./lab4_3_MOD_exec file.txt output.txt 32
File size in bytes: 1584. BufferSize: 32

Read/Write buffer time: 1.291600e-02, avrage: 8.154040e-06

Read/Write no buffer time: 3.548730e-01, avrage: 2.240360e-04
pi@raspberrypi:~$ ./lab4_3_MOD_exec file.txt output.txt 64
File size in bytes: 1584. BufferSize: 64

Read/Write buffer time: 6.255000e-03, avrage: 3.948864e-06

Read/Write no buffer time: 3.660270e-01, avrage: 2.310777e-04
```

As seen below we have a new result. Compared to the result with the C standard library it is now remarkably slower.

```
pi@raspberrypi:~$ time ./lab4_3_MOD_large_exec largeFile output.txt
real    0m0.496s
user    0m0.200s
sys     0m0.290s
pi@raspberrypi:~$ diff largeFile output.txt
pi@raspberrypi:~$ time cp largeFile output.txt
real    0m0.132s
user    0m0.040s
sys     0m0.090s
```

New source using system call instead of C standard library.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <unistd.h>
8  .
9  .typedef struct{
10 .    char* m_buffer;
```

```
11 .   char position;
12 .   char length;
13 .}buffer;
14 .
15 .buffer* setupBuffer(char size){
16 .   buffer* new_buffer;
17 .   new_buffer = malloc(sizeof(buffer));
18 .   new_buffer->m_buffer = malloc(size);
19 .   new_buffer->position = size;
20 .   new_buffer->length = size;
21 .   return new_buffer;
22 .}
23 .
24 .char buf_in(buffer* b, int byteFile){
25 .   if((*b).position >= (*b).length){
26 .       ssize_t t = read(byteFile, b->m_buffer, b->length);
27 .       (*b).position = 0;
28 .   }
29 .
30 .   char byte = *(b->m_buffer + b->position);
31 .   b->position++;
32 .   return byte;
33 .}
34 .
35 .void buf_flush(buffer *b, int byteFile){
36 .   ssize_t t = write(byteFile, b->m_buffer, b->position);
37 .   (*b).position = 0;
38 .}
39 .
40 .void buf_out(buffer* b, int byteFile, char byte){
41 .   *(b->m_buffer + b->position) = byte;
42 .   b->position++;
43 .   if((*b).position >= (*b).length){
44 .       buf_flush(b,byteFile);
45 .   }
46 .}
47 .
48 .int main(int argc, char* argv[]){
49 .   int readFile;
50 .   int writeFile;
51 .   int bufferSize = 16;
52 .   if(argc < 4){
53 .       readFile = open("file.txt", O_RDWR | O_CREAT);
54 .       writeFile = open("output.txt", O_RDWR | O_CREAT);
55 .   }else {
56 .       readFile = open(argv[1], O_RDWR | O_CREAT);
57 .       writeFile = open(argv[2], O_RDWR | O_CREAT);
58 .       bufferSize = atoi(argv[3]);
59 .   }
60 .
61 .   buffer* readBuffer = setupBuffer(bufferSize);
62 .   buffer* writeBuffer = setupBuffer(bufferSize);
63 .   writeBuffer->position = 0;
64 .   int size = lseek(readFile, 0, SEEK_END);
65 .   lseek(readFile, 0, SEEK_SET);
66 .   printf("File size in bytes: %d. BufferSize: %d\n", size, bufferSize);
67 .   int i;
68 .   clock_t start = clock();
69 .   for(i = 0; i < size; i++){
70 .       buf_out(writeBuffer, writeFile, buf_in(readBuffer,readFile));
```

```
71 .     }
72 .     buf_flush(writeBuffer, writeFile);
73 .     clock_t elapsed = clock() - start;
74 .
75 .
76 .     lseek(readFile, 0, SEEK_SET);
77 .     lseek(writeFile, 0, SEEK_SET);
78 .     start = clock();
79 .     char byte[1];
80 .     for(i = 0; i < size; i++){
81 .         read(readFile, byte, 1);
82 .         write(writeFile, byte, 1);
83 .     }
84 .     clock_t elapsed2 = clock() - start;
85 .
86 .     printf("\nRead/Write buffer time: %e, avrage: %e\n", elapsed/((double)CLOCKS_PER_SEC,
elapsed/((double)size/CLOCKS_PER_SEC);
87 .     printf("\nRead/Write no buffer time: %e, avrage: %e\n", elapsed2/((double)CLOCKS_PER_SEC,
elapsed2/((double)size/CLOCKS_PER_SEC);
88 .
89 .     free(readBuffer->m_buffer);
90 .     free(writeBuffer->m_buffer);
91 .     close(readFile);
92 .     close(writeFile);
93 .     return 0;
94 .}
95 .
```

For large files:

```
1 . #include <stdio.h>
2 . #include <stdlib.h>
3 . #include <time.h>
4 . #include <sys/types.h>
5 . #include <sys/stat.h>
6 . #include <fcntl.h>
7 . #include <unistd.h>
8 . #define bufferSize 64
9 .
10 . typedef struct{
11 .     char* m_buffer;
12 .     char position;
13 .     char length;
14 . }buffer;
15 .
16 . buffer* setupBuffer(char size){
17 .     buffer* new_buffer;
18 .     new_buffer = malloc(sizeof(buffer));
19 .     new_buffer->m_buffer = malloc(size);
20 .     new_buffer->position = bufferSize;
21 .     new_buffer->length = bufferSize;
22 .     return new_buffer;
23 . }
24 .
25 . char buf_in(buffer* b, int byteFile){
26 .     if((*b).position >= (*b).length){
27 .         ssize_t t = read(byteFile, b->m_buffer, b->length);
28 .         (*b).position = 0;
```

```
29 .     }
30 .
31 .     char byte = *(b->m_buffer + b->position);
32 .     b->position++;
33 .     return byte;
34 . }
35 . void buf_flush(buffer *b, int byteFile){
36 .     ssize_t t = write(byteFile, b->m_buffer, b->position);
37 .     (*b).position = 0;
38 . }
39 .
40 . void buf_out(buffer* b, int byteFile, char byte){
41 .     *(b->m_buffer + b->position) = byte;
42 .     b->position++;
43 .     if((*b).position >= (*b).length){
44 .         buf_flush(b,byteFile);
45 .     }
46 . }
47 .
48 . int main(int argc, char* argv[]){
49 .     int readFile;
50 .     int writeFile;
51 .
52 .     if(argc < 3){
53 .         readFile = open("file.txt", O_RDWR | O_CREAT);
54 .         writeFile = open("output.txt", O_RDWR | O_CREAT);
55 .     }else {
56 .         readFile = open(argv[1], O_RDWR | O_CREAT);
57 .         writeFile = open(argv[2], O_RDWR | O_CREAT);
58 .     }
59 .
60 .     buffer* readBuffer = setupBuffer(bufferSize);
61 .     buffer* writeBuffer = setupBuffer(bufferSize);
62 .     writeBuffer->position = 0;
63 .     int size = lseek(readFile, 0, SEEK_END);
64 .     lseek(readFile, 0, SEEK_SET);
65 .     int i;
66 .     clock_t start = clock();
67 .     for(i = 0; i < size; i++){
68 .         buf_out(writeBuffer, writeFile, buf_in(readBuffer,readFile));
69 .     }
70 .     buf_flush(writeBuffer, writeFile);
71 .
72 .     free(readBuffer->m_buffer);
73 .     free(writeBuffer->m_buffer);
74 .     close(readFile);
75 .     close(writeFile);
76 .     return 0;
77 . }
78 .
```