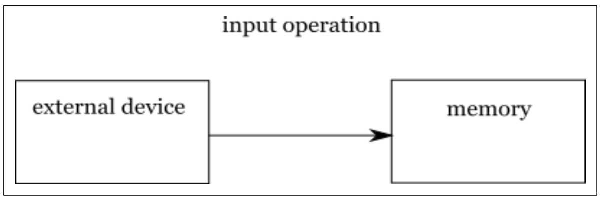
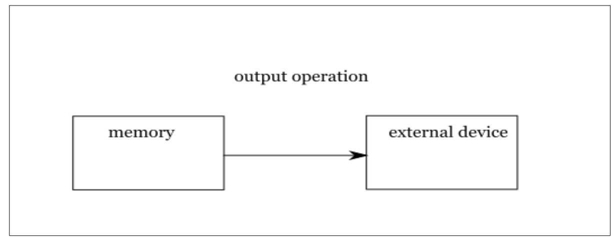
Buffered/Unbuffered IO

Input/Output (I/O)

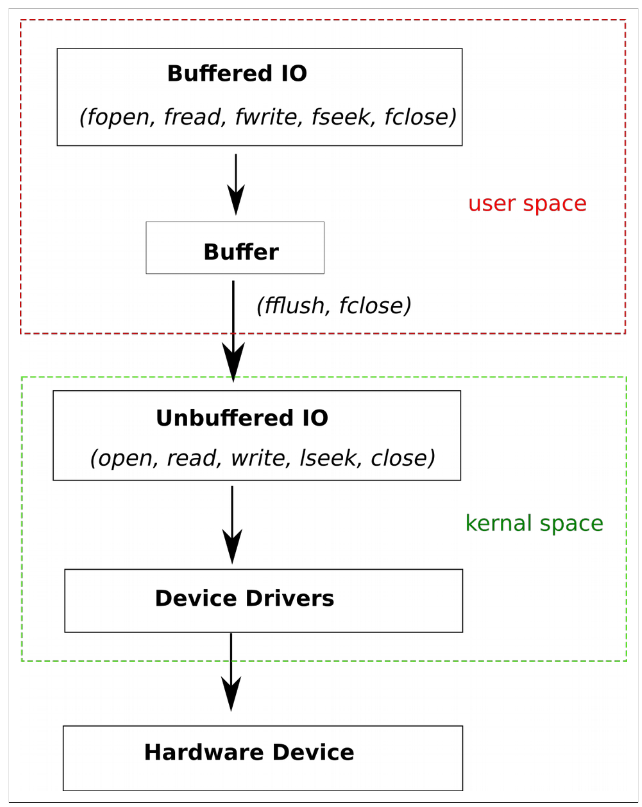
What is Input/Output

Input/Output (I/O) is the process of copying data between memory and external devices such as disk drivers, termials and networks.
Input operation copies data from an external device to memory.
Output operation copies data from memory to external device.





Buffered & Unbuffered IO



Unbuffered IO

Unbuffered IO is a part of POSIX, includes: open(), read(), write(), lseek(), close(), etc.

Unbuffered IO functions transfer data directly between memory and hardware device.

Buffered IO

Buffered IO is a part of ISO C, includes: fopen(), fread(), fwrite(), fseek(), fclose(), etc.

Buffered IO functions transfer data into a cache, that is called buffer.
The buffer will be written to hardware device in the following circumstances:
- The buffer end signal is reached.
- Function fclose() or fflush() is called.
- File is closed when process is terminated.

Buffered IO API is implemented by unbuffered IO API, in low level.

Buffered vs Unbuffered IO

Bubuffer IO functions are expensive

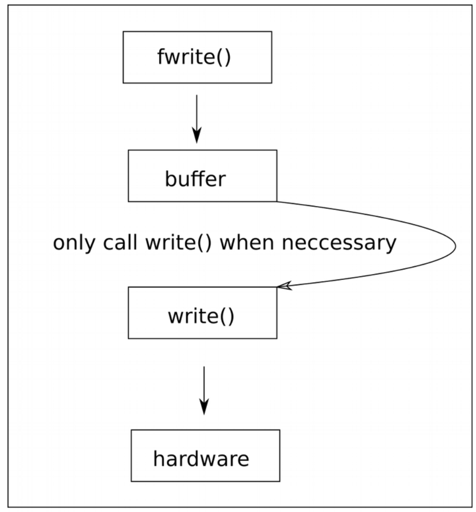
- Unbuffer IO functions like *open(), read(), write(), lseek(), close()* are also called [system call (syscalls)](). Each time a syscall is invoked, the transitions between user space and kernal space are performed, such as copying data from user space to kernal space and back again. So, syscall is more expensive than a function call.

- The hardware has limitations and impose restrictions on the size of data blocks that can be read/written at a time. For example, the size of data block is 512 bytes; although we attempt to write 32 bytes, the driver still performs on 512 bytes.

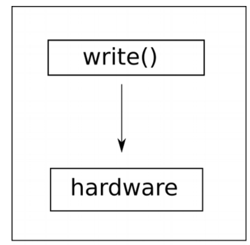
How Buffered IO API give us better performance?
The key is the buffer.

With Unbuffer IO, the driver performs write operation onto hardware, each time *write()* is called.

With Buffer IO function *fwrite()*, data is cached into buffer. Syscall and write operation is executed only when necessary.



buffered io



Unbuffer IO

Which to choose: Buffered vs Unbuffered IO?
When reading/writting data, Buffered IO gives better performance. Because, Unbuffered IO functions are expensive in each call. And Buffered IO is designed to reduce the call times.

But, sometime Unbuffer IO is the only option, such as accessing file metadata.

Buffer

What is Buffer?

Buffer is the cache where Buffer IO functions like fwrite() transfer data into, before data is written data to hardware device.

When is Buffer flushed?

Buffer is flushed when functions fflush(), fclose() is called or file is closed when process terminated.

Besides, standard I/O library also tries to flush buffer automatically through buffer type.

Macro	Buffer Type	When data is flushed
_IOFBF	Fully buffered	Buffer is full
_IOLBF	Line buffered	Newline character is encountered
_IONBF	Unbuffered	Buffer io functions is called

Default Buffer

When a file is opened, standard I/O library automatically create a buffer and associate it to the file, this is called default buffer.

Normally, regular files are fully buffered.

If file refers to a termial (stdin/stdout), it is line buffered.

Stderr is unbuffered.

Custom Buffer

Ater opening file, we can explicitly specify buffer by callilng setvbuf(), setbuf(), setbuffer() or setlinebuf().

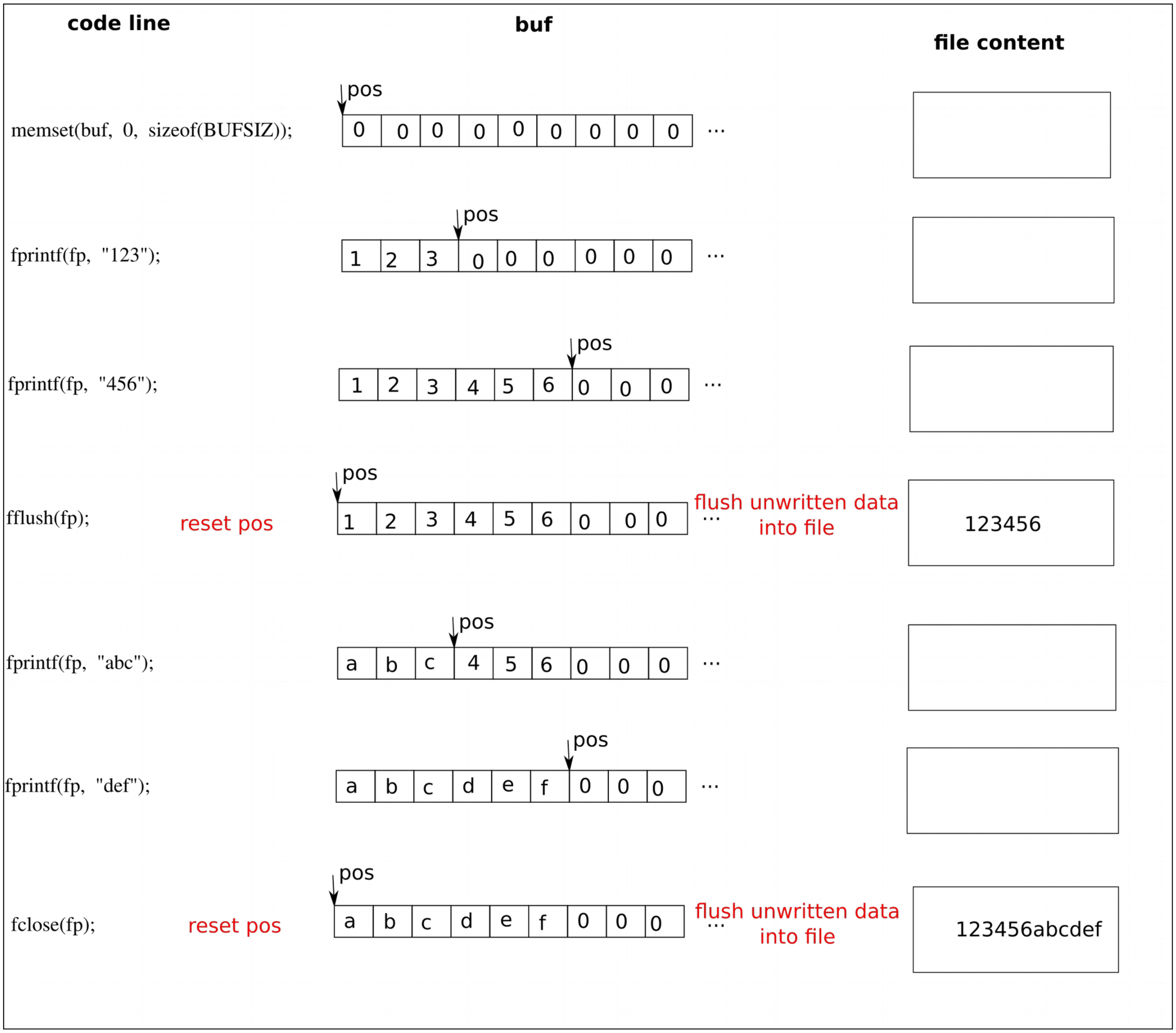
Practice

Excercise: custom and watch buffer value

Code: demo.c

```
#include <stdio.h>
#include <string.h>

int main() {
    char buf[BUFSIZ];
    memset(buf, 0, sizeof(BUFSIZ)); //buf: 000000000...

    FILE* fp = fopen("data.txt", "w");
    setvbuf(fp, buf, _IOFBF, BUFSIZ);

    fprintf(fp, "123"); //buf: 123000000... ,   pos: 3
    fprintf(fp, "456"); //buf: 123456000... ,   pos: 6
    fflush(fp);         //                  ,   pos: 0

    fprintf(fp, "abc"); //buf: abc456000... ,   pos: 3
    fprintf(fp, "def"); //buf: abcdef000... ,   pos: 6
    fclose(fp);

    return 0;
}
```

How it work



Compile

```
$gcc demo.c -o demo -g
```

Debug

```
gdb demo
b 11
r
p buf
n
p buf
n
p buf
n
p buf
n
c
```

```
Reading symbols from demo...done.
(gdb) b 11
Breakpoint 1 at 0x887: file demo.c, line 11.
(gdb) r
Starting program: /home/maxter/Exp/demo

Breakpoint 1, main () at demo.c:11
11          fprintf(fp, "123"); //buf: 123000000... ,   pos: 3
(gdb) p buf
$1 = '\000' <repeats 5097 times>...
(gdb) n
12          fprintf(fp, "456"); //buf: 123456000... ,   pos: 6
(gdb) p buf
$2 = "123", '\000' <repeats 5094 times>...
(gdb) n
13          fflush(fp);         //                  ,   pos: 0
(gdb) p buf
$3 = "123456", '\000' <repeats 5091 times>...
(gdb) n
15          fprintf(fp, "abc"); //buf: abc456000... ,   pos: 3
(gdb) p buf
$4 = "123456", '\000' <repeats 5091 times>...
(gdb) n
16          fprintf(fp, "def"); //buf: abcdef000... ,   pos: 6
(gdb) p buf
$5 = "abc456", '\000' <repeats 5091 times>...
(gdb) n
17          fclose(fp);
(gdb) n
19          return 0;
(gdb) c
Continuing.
[Inferior 1 (process 7581) exited normally]
```

//continue