

System-Level I/O: *Standard I/O*

Standard I/O Functions

- The C standard library (`libc.so`) contains a collection of higher-level *standard I/O* functions
 - Documented in Appendix B of K&R
- Examples of standard I/O functions:
 - Opening and closing files (`fopen` and `fclose`)
 - Reading and writing bytes (`fread` and `fwrite`)
 - Reading and writing text lines (`fgets` and `fputs`)
 - Formatted reading and writing (`fscanf` and `fprintf`)

Standard I/O Streams

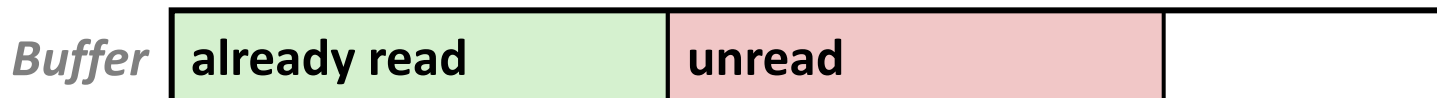
- Standard I/O models open files as *streams*
 - Abstraction for a file descriptor and a buffer in memory
- C programs begin life with three open streams (defined in `stdio.h`)
 - `stdin` (standard input)
 - `stdout` (standard output)
 - `stderr` (standard error)

```
#include <stdio.h>
extern FILE *stdin; /* standard input (descriptor 0) */
extern FILE *stdout; /* standard output (descriptor 1) */
extern FILE *stderr; /* standard error (descriptor 2) */

int main() {
    fprintf(stdout, "Hello, world\n");
}
```

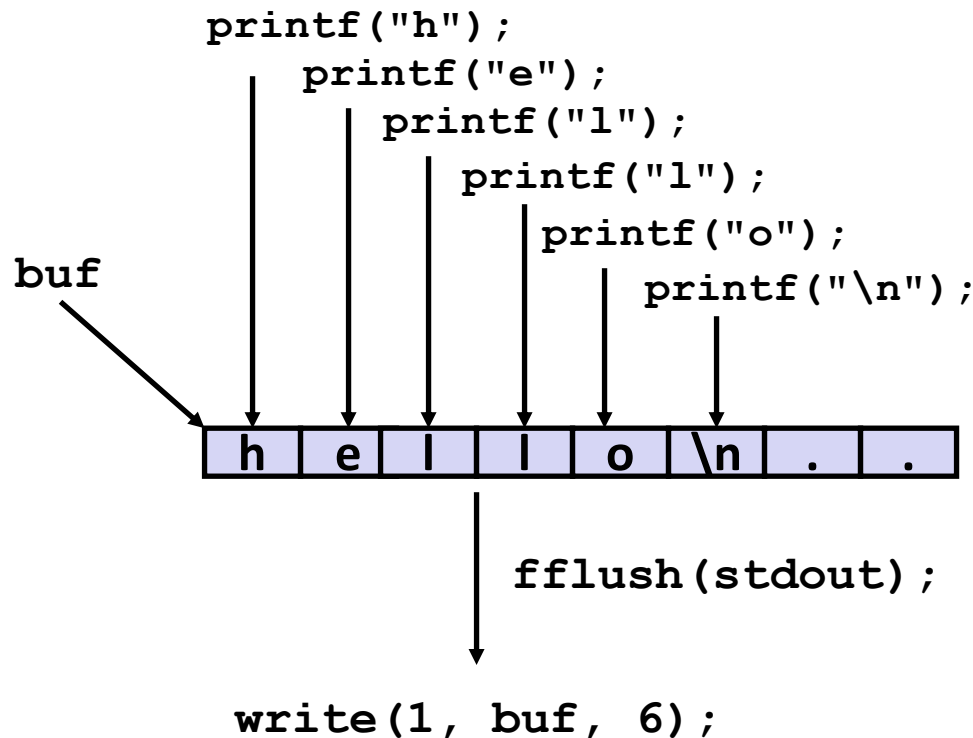
Buffered I/O: Motivation

- **Applications often read/write one character at a time**
 - `getc`, `putc`, `ungetc`
 - `gets`, `fgets`
 - Read line of text one character at a time, stopping at newline
- **Implementing as Unix I/O calls expensive**
 - `read` and `write` require Unix kernel calls
 - > 10,000 clock cycles
- **Solution: Buffered read**
 - Use Unix `read` to grab block of bytes
 - User input functions take one byte at a time from buffer
 - Refill buffer when empty



Buffering in Standard I/O

- Standard I/O functions use buffered I/O



- Buffer flushed to output fd on “\n”, call to `fflush` or `exit`, or return from `main`.

Standard I/O Buffering in Action

- You can see this buffering in action for yourself, using the always fascinating Linux `strace` program:

```
#include <stdio.h>

int main()
{
    printf("h");
    printf("e");
    printf("l");
    printf("l");
    printf("o");
    printf("\n");
    fflush(stdout);
    exit(0);
}
```

Standard I/O Buffering in Action

- You can see this buffering in action for yourself, using the always fascinating Linux `strace` program:

```
#include <stdio.h>

int main()
{
    printf("h");
    printf("e");
    printf("l");
    printf("l");
    printf("o");
    printf("\n");
    fflush(stdout);
    exit(0);
}
```

```
linux> strace ./hello
execve("./hello", ["hello"], [/* ... */]).
...
write(1, "hello\n", 6)                = 6
...
exit_group(0)                         = ?
```