```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);
```

The read() function attempts to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf.
If nbyte is 0, read() will return 0 and have no other results.

On files that support seeking (for example, a regular file), the read() starts at a position in the file given by the file offset associated with fildes. The file offset is incremented by the number of bytes actually read.

Files that do not support seeking, for example, terminals, always read from the current position. The value of a file offset associated with such a file is undefined.

No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent read() requests is implementation-dependent.

If the value of nbyte is greater than {SSIZE_MAX}, the result is implementation-dependent.

When attempting to read from an empty pipe or FIFO:

If no process has the pipe open for writing, read() will return 0 to indicate end-of-file.
If some process has the pipe open for writing and O_NONBLOCK is set, read() will return -1 and set errno to [EAGAIN].
If some process has the pipe open for writing and O_NONBLOCK is clear, read() will block the calling thread until some data is written or the pipe is closed by all processes that had the pipe open for writing.
When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and has no data currently available:

If O_NONBLOCK is set, read() will return a -1 and set errno to [EAGAIN].
If O_NONBLOCK is clear, read() will block the calling thread until some data becomes available.
The use of the O_NONBLOCK flag has no effect if there is some data available.

```
#include <unistd.h>

ssize_t write(int fildes, const void *buf, size_t nbyte);
```

The write() function attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fildes.
If nbyte is 0, write() will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with fildes. Before successful return from write(), the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.

On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.

If a write() requests that more bytes be written than there is room for (for example, the ulimit or the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes will give a failure return (except as noted below) and the implementation will generate a SIGXFSZ signal for the thread.

If write() is interrupted by a signal before it writes any data, it will return -1 with errno set to [EINTR].

If write() is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of nbyte is greater than {SSIZE_MAX}, the result is implementation-dependent.

After a write() to a regular file has successfully returned:

Any successful read() from each byte position in the file that was modified by that write will return the data specified by the write() for that position until such byte positions are again modified.
Any subsequent successful write() to the same byte position in the file will overwrite that file data.
Write requests to a pipe or FIFO will be handled the same as a regular file with the following exceptions:

There is no file offset associated with a pipe, hence each write request will append to the end of the pipe.
Write requests of {PIPE_BUF} bytes or less will not be interleaved with data from other processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag of the file status flags is set.
If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it will return nbyte.
If the O_NONBLOCK flag is set, write() requests will be handled differently, in the following ways:
The write() function will not block the thread.
A write request for {PIPE_BUF} or fewer bytes will have the following effect:
If there is sufficient space available in the pipe, write() will transfer all the data and return the number of bytes requested. Otherwise, write() will transfer no data and return -1 with errno set to [EAGAIN].
A write request for more than {PIPE_BUF} bytes will case one of the following:
When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe is read, it will transfer at least {PIPE_BUF} bytes.
When no data can be written, transfer no data and return -1 with errno set to [EAGAIN].
When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

If the O_NONBLOCK flag is clear, write() will block the calling thread until the data can be accepted.
If the O_NONBLOCK flag is set, write() will not block the process. If some data can be written without blocking the process, write() will write what it can and return the number of bytes written. Otherwise, it will return -1 and errno will be set to [EAGAIN].