The way read() behaves depends on what is being read. For regular files, if you ask for N characters, you get N characters if they are available, less than N if end of file intervenes.

If read() is reading from a terminal in canonical/cooked mode, the tty driver provides data a line at a time. So if you tell read() to get 3 characters or 300, read will hang until the tty driver has seen a newline or the terminal's defined EOF key, and then read() will return with either the number of characters in the line or the number of characters you requested, whichever is smaller.

If read() is reading from a terminal in non-canonical/raw mode, read will have access to keypresses immediately. If you ask read() to get 3 characters it might return with anywhere from 0 to 3 characters depending on input timing and how the terminal was configured.

read() will behave differently in the face of signals, returning with less than the requesting number of characters, or -1 with errno set to EINTR if a signal interrupted the read before any characters arrived.

read() will behave differently if the descriptor has been configured for non-blocking I/O. read() will return -1 with errno set to EAGAIN or EWOULDBLOCK if no input was immediately available. This applies to sockets.

So as you can see, you should be ready for surprises when you call read(). You won't always get the number of characters you requested, and you might get non-fatal errors like EINTR, which means you should retry the read().