

## Re: fgets() equivalent?

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2007-12/msg00011.htm>

---

- *From:* Pops <[notreal@xxxxxxxxxxxx](mailto:notreal@xxxxxxxxxxxx)>
  - *Date:* Mon, 03 Dec 2007 03:29:35 -0500
- 

Tim Roberts wrote:

Pops <[dude@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:dude@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)> wrote:

I would like to see an official definition of that. By your definition, TELNET and all telecommunication programs in the annals of computers is both RAW and COOKED. So by this basis, I can see what you are saying, but it is completely foreign in my software designs of telnet server, telnet clients, ANSI/VT10x emulators, command line based client/server systems, etc, etc, etc.

Most terminal emulators on Unix use raw mode, so they can get the keystrokes immediately. Any program that uses curses, for example, switches the terminal into raw mode. Vi uses raw mode, so that I can see the effect of my keys immediately.

The default state, however, is cooked mode. In that state, the program doesn't see any characters at all until I press Enter. Let's say I have a loop like this:

```
while( 1 )  
    putchar( getch() );
```

In raw mode with echo enabled, if I press A then B then C then enter, I will see:  
AABBCC

because the keys will be sent to the app immediately. In cooked mode (the default), if I do the same thing I'll see:

```
ABC  
ABC
```

The first line is the echo of my keystrokes, the second line is from the putchar function, because getch was blocked until I pressed Enter.

And what I am saying is that you are forgetting about the hardware devices and/or any client side

## Re: fgets() equivalent?

considerations. More below.

COOK vs RAW is what it is, COOK is a translation of some kind being done, RAW means its no translation.

Again, that's a very convenient definition, and you are free to use that if you want, but that is NOT how the terms are used with Unix terminals. "Cooked" and "raw" have a well-defined and well-established meaning in Unix, and it is not the meaning you are attempting to assign.

Please keep in mind this is what we do. We write online and automated hosting server and clients side applications that has dealt with this issues for the past 25+ years and deal with customers across the board. So please, there is no justice in trying to nit pick at generalizations in a cyber space message world. Its a waste of your time and my time to get into this nit picking mode. But I will try to outline for the general reader who are keen to the concepts:

Regardless of the OS, in telecommunications, you have these five basic layers:

- client application
- sender device
- transmission
- receiver device
- server application

The control codes (0 to 31) are interpreted and used in many different ways. They can be escaped or not escaped, in some disciplines, "cooked" or "not cooked", it could be software or hardware flow control, they can be rendering, transformations, etc, I mean, the sky is the limit.

In a more complex WAN world, the transmission can be practically anything, but for the sake of this note, I just to point out the possibility of having PADs involved in a packet switching networks:

– transmission  
Sender PAD  
network (switches and routers)  
Receiver PAD

So keep that in mind that there are possible issues here too. However, in today's world, Smart PADs auto negotiate very reliably, so it isn't a problem like it use to be in the past in a mix world of Smart vs Non-Smart PADs connections, especially in growth area of online telecommunications, file transfers and terminal services where the concept of TEXT vs BINARY was important.

In a pure DUMB terminal scenario, you have:

- sender device (I mean really DUMB terminal)
- transmission
- receiver device
- server application

## Re: fgets() equivalent?

In this case, the user (sender) has a dumb terminal wired to some network, it is typically a RS232 port to some network RS232 port.

Now, the terminal device typically has 2–4 important setup options that one has to recognize in this application:

ECHO EACH CHARACTER  
APPEND LINE FEED  
COMMAND LINE MODE (optional, depend on terminal)  
XON/XOFF (software flow control)

Not all early terminals offer command line because that assume some level of intelligence with having a BUFFER to hold the characters. But we can say for the most part, when we moved into the Smart Terminal era, where it did have some embedded programming and the introduction of FUNCTION KEYS where one can stack characters plus ^M, for the most part, lets assume we are not dealing with a command line mode terminal type.

In additional, with XON/XOFF, lets assume that this is OFF. In the past, it use to be ON by default when the era was mostly console and text driven. But not today, XON/XOFF is typically escaped. So lets assume this is off. We also assume the transmission also escapes XON/XOFF and the the receiver device escapes XON/XOFF. This reduces the layout to this:

- sender device (I mean really DUMB terminal)
- server application

ECHO and LINEFEED terminal options are often matched with the server characteristics. Does the server echo the characters, does the server take into account the EOL translation?

So using your example, if the terminal had echo on and the server has echo on, if you typed the letter A, you will see two:

AA <--- one echo by the terminal, one echo by the server.

So as you can imagine, one can say to in order to save bandwidth, the terminal will normally have ECHO ON and the server has ECHO OFF.

But that depends.

In most hosting systems, the SERVER will echo it. Maybe it wants to do special processing, like a online PASSWORD input where it wants to ECHO the asterisk as the user types the password:

Enter your password: \*\*\*\*\*

So in most cases, the terminal has ECHO off, the Server does the echoing.

That is not typically referred to as a "Cooking vs Raw" concept. If by change the terminal itself was doing a "translation" to an asterisk (somehow it knew it was password input mode), then maybe one might say that is "cooking" idea. But I never head of such a concept in a dumb terminal – definitely in smarter terminals where we begin to have SOFTWARE on the terminal client side.

Now, when it comes to the linefeed, that also depends.

Re: fgets() equivalent?

## Re: fgets() equivalent?

In a Unix world, it is probably better to have the dumb terminal set to NOT issue the LineFeed. In a MS-DOS world, it is probably better to have the dumb terminal set to issue the LineFeed.

But it really depends on the server. The better servers, it is prudent that it handles any kind of EOL format because it doesn't know what type of client and/or terminal is connected. So it has to handle both possible styles. Some servers expect it <CR><LF>, others do not. It depends. In some servers the user can change his server options to match his terminal options, etc.

Finally, if we are talking strictly FILE STORAGE, then we also have a "COOK" concept or translation concept, especially for Unix.

In DOS, if I was piping con device to foo:

```
type con > foo
```

the MS-DOS console process is doing the echoing and <CR> --> <CR><LF> translation. My keyboard is not doing it.

I forget my Unix, so you can confirm this, if you do the equivalent over a dumb terminal with LINEFEED enabled (effectively the same as the keyboard doing it), will it capture all characters sent, include <cr><lf> and it waits for a ^Z (eof) to end the piping?

If not, and it indeed stores only <LF> then Unix is cooking it!

--

HLS

.