

## Re: serial: flushing device buffer – am I close?

**Source:** <http://unix.derkeiler.com/Newsgroups/comp.unix.programmer/2004-06/0582.html>

---

**From:** Joerg Schmitz-Linneweber ([jsl\\_at\\_sth.ruhr-uni-bochum.de](mailto:jsl_at_sth.ruhr-uni-bochum.de))

**Date:** 06/24/04

Date: Thu, 24 Jun 2004 10:59:43 +0200

Hi TJ,

I don't really see the point...

You get what you asked (the library) for. But let's have a look:

TJ wrote:

- > *I'm writing a program in C that communicates with a serial device. I have*
- > *a conceptual question to clarify my interpretation of the HOWTOs...*
- >
- > *The specs of my device, are that it responds with a fixed number of bytes*
- > *per issued command (typically 6 bytes). Sometimes, after an abend and*
- > *restart, the device's response will only be two bytes, and then my program*
- > *hangs while waiting for the remaining 4 bytes. My thought is that those*
- > *two bytes in the device's buffer are residual from the last run of the*
- > *program, assuming abnormal termination.*

OK. If you don't have fixed input record length', you'll need some kind of "frame syncing" however this will be done (start character, timeout, ...)

- > *I changed my approach of "always wait for 6 bytes" to "read bytes until*
- > *the*
- > *buffer is empty" – certainly a better approach. I also decided to flush*
- > *the buffer on the device by performing a read before any commands are*
- > *issued.*

Which buffer do you flush? (input or output)

But you know that the buffer might be empty although you don't have an actual record yet? (Your reads may be faster than the device delivers data...)

- > *But now that I am reading the buffer until empty, it always appears to be*
- > *empty... perhaps because data has not arrived yet?? I'm sure there is a*
- > *simple technique, used millions of times, and I'm not quite finding it.*
- > *My loop will wait until data arrives, but the read() always returns zero,*
- > *so*
- > *there never appears to be data. If I ignore the loop, and just ask for 6*
- > *bytes, sometimes I get all of them, sometimes I don't. Tricky.*

Not so tricky... Of course, most of the time the buffer will be empty since your CPU is much faster than the device delivers data

comp.unix.programmer: Re: serial: flushing device buffer – am I close?

> *From the manpage on read():*  
> *On success, the number of bytes read is returned (zero indicates end of*  
> *file), and the file position is advanced by this number. It is not an*  
> *error if this number is smaller than the number of bytes requested; this*  
> *may happen for example because fewer bytes are actually available right*  
> *now (maybe because we were close to end-of-file, or because we are reading*  
> *from a pipe, or from a terminal), or because read() was interrupted by a*  
> *signal. On error, -1 is returned, and errno is set appropriately. In this*  
> *case it is left unspecified whether the file position (if any) changes.*  
Right.

> *So possibly its my configuration/code:*  
> *Linux 2.6 , GCC 3.3.3 (SuSE 9.1)*  
>  
> *Notes:*  
> *1. I set a fixed size for 128 since my device never exceeds 72 bytes in*  
> *or out.*  
> *2. The result bytes do not always come at once, they have to be*  
> *concatenated as they arrive.*  
>  
> *dpDevWIDGET = open(strDevWIDGET, O\_RDWR | O\_NOCTTY | O\_NONBLOCK);*

Why do you use NON\_BLOCKING operation? Is this only pseudocode or are you performing other things while your reads return zero?

Why not try to read one byte in blocking mode? The read will return you one byte if it is available (or it may return 0 if there was a signal (interrupt) or even bail out with < 0)

```
> if (dpDevWIDGET == -1){  
> intUTILS_PrintMsg(7, "WIDGET", "Port not opened");  
> }else{  
> intLRC = tcgetattr(dpDevWIDGET, &tattr);  
> intLRC = cfsetispeed(&tattr, B1200);  
> intLRC = cfsetospeed(&tattr, B1200);  
> intLRC = tcsetattr(dpDevWIDGET, TCSAFLUSH, &tattr);  
> tattr.c_lflag &= ~(ICANON | ECHO | ECHOE | ECHOK | ISIG);  
> /* reset the control characters */  
> for (i=0; i<NCCS; i++) {  
> tattr.c_cc[i] = _POSIX_VDISABLE;  
> }  
> }  
> }
```

Can't say much about the settings of the line discipline... (anyone?)

```
> // expect 6 bytes, but they dont always arrive at once, so concatenate  
> them  
> as they arrive from multiple reads. Use the same routine to flush the  
> device buffer before any commands are issued to the device that get a  
> response...  
>  
> while(blnEOF == FALSE){  
> intSize = read(dpDevWIDGET, inbuffer, 128);  
> intTmpBufferTotalSize += intSize;
```

Re: serial: flushing device buffer – am I close?

comp.unix.programmer: Re: serial: flushing device buffer – am I close?

```
> if (intSize < 0){
> perror("WIDGET");
> blnEOF = TRUE;
> return(FAILURE);
> break;
> }else if (intSize == 0){
> printf("WIDGET:\tBuffer empty\n");
> blnEOF = TRUE;
> break;
> }else if (intSize > 0){
> for (intInBufferPos = 0; intInBufferPos < intSize;
> intInBufferPos++){
> tmpbuffer[intTmpBufferPos++] =
> inbuffer[intInBufferPos];
> }
> }
> }
```

OK. See above for using blocking reads eventually. But be aware that a possible scenario for your reads may be: (You are expecting to get 6 bytes) [Assuming that at time 'step00' you are in-sync (i.e. at start-of-frame)]

```
step00: read () returns 0
step01: read () returns 0
step02: read () returns 0
step03: read () returns 2
step04: read () returns 0
step05: read () returns 1
step06: read () returns 0
step07: read () returns 0
step08: read () returns 4
step09: read () returns 6
step10: read () returns 0
step11: read () returns 0
step12: read () returns 0
step13: read () returns 0
```

Let' see: after step03 you've got data but not enough. Don't discard it!  
After step05 the same. After step08 there is too much data! So now you've to deal with "one and a half" frame... And so on. But most of the time there is no data waiting!

HTH. Salut, Jörg

--

gpg/pgp key # 0xe40a9d7a  
fingerprint d4f8 b448 835b 7bcf 4161 ce35 7e8b ab47 e40a 9d7a

Re: serial: flushing device buffer – am I close?