

ps,libkvm,ls malloc/free

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2003-09/0482.html>

From: Allan Fields (bsd_at_afields.ca)

Date: 09/30/03

Date: Mon, 29 Sep 2003 22:28:05 -0400

To: freebsd-hackers@freebsd.org

Hi,

While trying to debug issues with libkvm access from an experimental tool (quick-hack) I've been working on, I've come across a segfault that occurs in `_kvm_malloc()`. This is a stock install of FreeBSD 5.1-RELEASE but the code invoking libkvm is originally taken from `bin/ps/ps.c` in the latest RELENG_5_1 source distribution.

The situation is related to a call of `kvm_openfiles()` subsequent to a previous call and `kvm_close()`. I've poked around in the most recent libkvm sources and so-far didn't see why `_kvm_malloc()` would fail in this case.

Further my backtrace doesn't show the same calling order one might expect:
Program terminated with signal 11, Segmentation fault.

#0 0x080ea74d in `_kvm_malloc ()`

#1 0x080eaa41 in `kvm_openfiles ()`

#2 0x08066e3a in `main (argc=1, argv=0x82f4000) at /usr/src-5_1/bin/ps/ps.c:324`

In the source it looks more like: `kvm_openfiles() => _kvm_open()` (and I can't find the `_kvm_malloc()` call.)

Does this have anything to do with changes for descriptors being flagged close-on-exec? Has something changed recently so that I have to update libkvm?

I solved a similar issue with `fts_open()` segfaulting `bin/ls`, on subsequent calls by cleaning up before `traverse()` returns. [Patch below.]

To be truthful: it is my non-standard usage of these routines that has turned up these problems, and I was expecting some issues like this to appear. This problem is appearing only because of the atypical usage of these calls multiple times with-in the same process which could potentially be avoided. But my question is: why doesn't `kvm_close()` do the job? Is `kvm_open*()` allowed to be called more than once by the same process?

I also noticed that certain binaries including ls and ps don't explicitly free memory and close file descriptors before return. Usually this is not a problem, however: is this the right approach to take from the standpoint of code style? Is it fair to assume a return from main() will always clean-up for us? I have made this assumption myself before. exit(3) usually does what we need.

I can see that it wouldn't be hard to go through and create #ifdef CLEANUP blocks to give the option of explicit clean-up. Better still would be avoiding situations where there are items remaining to be free(ed) on return by cleaning up directly after use.

Also of concern to me here is the number of globals in the base system commands, but so far this hasn't been a serious problem. I'm actually suprised I have had the sucess with the approach that I have.

-- Allan Fields

```
-----
diff -u ls.c.orig ls.c:
--- ls.c.orig Mon Sep 29 06:10:41 2003
+++ ls.c Mon Sep 29 06:39:28 2003
@@ -507,6 +507,12 @@
     }
     if (errno)
         err(1, "fts_read");
+
+ /* Clean-up: */
+ free(p);
+ if (fts_close(ftsp)) /* explicitly close descriptor */
+ err(1, "fts_close");
+
 }

 /*
-----
```

freebsd-hackers@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-hackers>

To unsubscribe, send any mail to "freebsd-hackers-unsubscribe@freebsd.org"