

## Re: Format of Pointers in Unix

**Source:** <http://coding.derkeiler.com/Archive/C/ CPP/comp.lang.c/2004-05/1888.html>

---

**From:** Daniel Rudy (*dcrudy\_at\_invalid.pacbell.nospam.net.0123456789*)

**Date:** 05/17/04

Date: Mon, 17 May 2004 06:44:48 GMT

And somewhere around the time of 05/16/2004 17:28, the world stopped and listened as Chris Torek contributed the following to humanity:

>>And Daniel Rudy said...

>>

>>>Besides, I \*DID\* look in the compiler manual for cc and it doesn't say.

>>> Which is why I'm asking in the first place. Also, does Unix use the

>>>segmented or flat memory model. I'm asking because I don't know and the

>>>docs on my system don't really give a straight answer either way.

>

>

> This helps illustrate why cross-posting is often a bad idea. :-)

>

My appologies about that, but because of the nature of my question, I believe that it would be best to invite individuals from both groups as my question deals with C programming on the Unix platform, specifically FreeBSD on IA32 hardware.

> In article <news:pan.2004.05.16.23.08.46.586902@Utel.no>

> Nils O. Sel?dal <NOS@Utel.no> writes:

>

>>All unixes I've seen use a flat memory model for user space applications.

>

>

> Unix (or more specifically POSIX, although there are a number of

> standards one can use to define "Unix" as well) imposes some

> constraints that make life extremely difficult for anyone who wants

> to use a non-uniform / "non-flat" per-process memory layout.

> Specifically, the mmap() interface and functions like dlsym() even

> manage to rule out all but a weakened form of Harvard architectures

> ("separate I&D").

>

> Older (V6 and/or V7) Unixes did in fact run on PDP-11/70s with

> truly separate I&D spaces, so that:

>

> char \*p;

> void (\*q)();

```
> p = (void *)0x1234;
> q = (void (*)())p;
>
> had "p" and "q" pointing to different *physical* memory, even though
> the two addresses were clearly identical. (In effect, addresses
> were 17, not 16, bits long, with the "topmost" bit being "0 for
> read/write-access, 1 for execute-access". Hence *p referred to
> what one might call "address 0x01234", while (*q)() referred to
> "0x11234".) The idea that one can mmap() an executable file and
> then call into it, however, pretty much rules this right out.
>
> The C language is much less restrictive than a typical Unix-like
> system, however, so C can run on all kinds of machines that Unix
> can never use. This is both good and bad: a less-restrictive system
> can run on more hardware, but a more-restrictive system offers much
> more "concreteness" to the programmer and is thus much easier to
> write code for.
```

Now here's an interesting peice of code that I wrote earlier today complete with the compile and run:

```
strata:/home/dcrudy/c 1027 $$$ -->cat ptr_test1.c
/*
```

Pointer Test Code. Not in Book

```
*/
#include <stdio.h>

int thing_var; /* This is the actual thing */
int *thing_ptr; /* This is a pointer to thing */
int **thing_ptr_2; /* This is a pointer to pointer thing_ptr */

main()
{
    /* Initial value of thing_var */
    thing_var = 4;
    printf("Value of thing_var is %d\n\n", thing_var);

    /* Load pointer with address of variable thing_var */
    thing_ptr = &thing_var;
    printf("Address of thing_var is %x\n", &thing_var);
    printf("Value of pointer is %x\n\n", thing_ptr);

    /* Assign value to thing using pointer */
    *thing_ptr = 5;
    printf("New value of thing_var is %d\n\n", thing_var);

    /* Load pointer with address of pointer thing_ptr */
    thing_ptr_2 = &thing_ptr;
```

## comp.lang.c: Re: Format of Pointers in Unix

```
printf("Address of thing_ptr is %x\n", &thing_ptr);
printf("Contents of thing_ptr_2 %x\n\n", thing_ptr_2);

/* Change thing_var by referencing thing_ptr_2 */
**thing_ptr_2 = 6;
printf("Value of thing_var is %d\n\n", thing_var);

/* Lets do a pointer programming error... */
thing_ptr_2 = 7;
printf("Address of thing_ptr is %x\n", &thing_ptr);
printf("Contents of thing_ptr_2 %x\n\n", thing_ptr_2);

/* This will probably seg_fault the program. */
printf("Contents of thing via our blown ptr %d\n", **thing_ptr_2);

}
```

```
strata:/home/dcrudy/c 1031 $$$ ->cc -g -o ptr_test1 ptr_test1.c
ptr_test1.c: In function `main':
ptr_test1.c:38: warning: assignment makes pointer from integer without a
cast
strata:/home/dcrudy/c 1032 $$$ ->./ptr_test1
Value of thing_var is 4
```

```
Address of thing_var is 8049834
Value of pointer is 8049834
```

```
New value of thing_var is 5
```

```
Address of thing_ptr is 8049830
Contents of thing_ptr_2 8049830
```

```
Value of thing_var is 6
```

```
Address of thing_ptr is 8049830
Contents of thing_ptr_2 7
```

```
Memory fault (core dumped)
```

```
strata:/home/dcrudy/c 1033 $$$ ->gdb -se ptr_test1 -c ptr_test1.core
```

```
GNU gdb 4.18 (FreeBSD)
```

```
Copyright 1998 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain
conditions.
```

```
Type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB. Type "show warranty" for details.
```

```
This GDB was configured as "i386-unknown-freebsd"...Deprecated bfd_read
called at /usr/src/gnu/usr.bin/binutils/
```

```
gdb/../../../../contrib/gdb/gdb/dbxread.c line 2627 in
```

```
elfstab_build_psymtabs
```

```
Deprecated bfd_read called at
```

comp.lang.c: Re: Format of Pointers in Unix

```
/usr/src/gnu/usr.bin/binutils/gdb/../../../../contrib/gdb/gdb/dbxread.c  
line 933 i  
n fill_symbuf
```

```
Core was generated by `ptr_test1'.  
Program terminated with signal 11, Segmentation fault.  
Reading symbols from /usr/lib/libc.so.4...done.  
Reading symbols from /usr/libexec/ld-elf.so.1...done.  
#0 0x80485b1 in main () at ptr_test1.c:43  
43 printf("Contents of thing via our blown ptr %d\n",  
**thing_ptr_2);  
(gdb) quit  
strata:/home/dcrudy/c 1034 $$$ ->
```

\*\*\*

As you can see, the program cored because it did not like the address that I assigned to thing\_ptr\_2, which would suggest a segmented memory model? I am having a problem with this because if it was a true flat memory model, then I would be able to read data from other process which I have nothing to do with. There was a mention in the kernel options about allowing a program to modify it's own LDT, which implies that FreeBSD does use a segmented memory model of some sort.

```
--  
Daniel Rudy  
Remove nospam, invalid, and 0123456789 to reply.
```