How to get printk format specifiers right

Author: Randy Dunlap <rdunlap@infradead.org>
Author: Andrew Murray <amurray@mpc-data.co.uk>

Integer types

If variable is of Type,	use printk format specifier:
char	%d or %x
unsigned char	%u or %x
short int	%d or %x
unsigned short int	%u or %x
int	%d or %x
unsigned int	%u or %x
long	%ld or %lx
unsigned long	%lu or %lx
long long	%lld or %llx
unsigned long long	%llu or %llx
size_t	%zu or %zx
ssize_t	%zd or %zx
s8	%d or %x
u8	%u or %x
s16	%d or %x
u16	%u or %x
s32	%d or %x
u32	%u or %x
s64	%lld or %llx
u64	%llu or %llx

If <type> is architecture-dependent for its size (e.g., cycles_t, tcflag_t) or is dependent on a config option for its size (e.g., blk_status_t), use a format specifier of its largest possible type and explicitly cast to it.

Example:

```
printk("test: latency: %llu cycles\n", (unsigned long long)time);
```

Reminder: sizeof() returns type size_t.

The kernel's printf does not support %n. Floating point formats (%e, %f, %g, %a) are also not recognized, for obvious reasons. Use of any unsupported specifier or length qualifier results in a WARN and early return from vsnprintf().

Pointer types

A raw pointer value may be printed with %p which will hash the address before printing. The kernel also supports extended specifiers for printing pointers of different types.

Some of the extended specifiers print the data on the given address instead of printing the address itself. In this case, the following error messages might be printed instead of the unreachable information:

```
(null)   data on plain NULL address
(efault) data on invalid address
(einval) invalid data on a valid address
```

Plain Pointers

```
%p      abcdef12 or 00000000abcdef12
```

Pointers printed without a specifier extension (i.e unadorned %p) are hashed to prevent leaking information about the kernel memory layout. This has the added benefit of providing a unique identifier. On 64-bit machines the first 32 bits are zeroed. The kernel will print `(ptrval)` until it gathers enough entropy.

When possible, use specialised modifiers such as %pS or %pB (described below) to avoid the need of providing an unhashed address that has to be interpreted post-hoc. If not possible, and the aim of printing the address is to provide more information for debugging, use %p and boot the kernel with the `no_hash_pointers` parameter during debugging, which will print all %p addresses unmodified. If you *really* always want the unmodified address, see %px below.

If (and only if) you are printing addresses as a content of a virtual file in e.g. procfs or sysfs (using e.g. seq_printf(), not printk()) read by a userspace process, use the %pK modifier described below instead of %p or %px.

Error Pointers

```
%pe     -ENOSPC
```

For printing error pointers (i.e. a pointer for which IS_ERR() is true) as a symbolic error name. Error values for which no symbolic name is known are printed in decimal, while a non-ERR_PTR passed as the argument to %pe gets treated as ordinary %p.

Symbols/Function Pointers

```
%pS     versatile_init+0x0/0x110
%ps     versatile_init
%pSR    versatile_init+0x9/0x110
        (with __builtin_extract_return_addr() translation)
%pB     prev_fn_of_versatile_init+0x88/0x88
```

The `S` and `s` specifiers are used for printing a pointer in symbolic format. They result in the symbol name with (S) or without (s) offsets. If KALLSYMS are disabled then the symbol address is printed instead.

The `B` specifier results in the symbol name with offsets and should be used when printing stack backtraces. The specifier takes into consideration the effect of compiler optimisations which may occur when tail-calls are used and marked with the noreturn GCC attribute.

If the pointer is within a module, the module name and optionally build ID is printed after the symbol name with an extra `b` appended to the end of the specifier.

```
%pS     versatile_init+0x0/0x110 [module_name]
%pSb    versatile_init+0x0/0x110 [module_name ed5019fdf5e53be37cb1ba7899292d7e143b259e]
%pSRb   versatile_init+0x9/0x110 [module_name ed5019fdf5e53be37cb1ba7899292d7e143b259e]
        (with __builtin_extract_return_addr() translation)
%pBb    prev_fn_of_versatile_init+0x88/0x88 [module_name ed5019fdf5e53be37cb1ba7899292d7e143b259e]
```

Probed Pointers from BPF / tracing

```
%pks    kernel string
%pus    user string
```

The `k` and `u` specifiers are used for printing prior probed memory from either kernel memory (k) or user memory (u). The subsequent `s` specifier results in printing a string. For direct use in regular vsnprintf() the (k) and (u) annotation is ignored, however, when used out of BPF's bpf_trace_printk(), for example, it reads the memory it is pointing to without faulting.

Kernel Pointers

```
%pK     01234567 or 0123456789abcdef
```

For printing kernel pointers which should be hidden from unprivileged users. The behaviour of %pK depends on the kptr_restrict sysctl - see Documentation/admin-guide/sysctl/kernel.rst for more details.

This modifier is *only* intended when producing content of a file read by userspace from e.g. procfs or sysfs, not for dmesg. Please refer to the section about %p above for discussion about how to manage hashing pointers in printk().

Unmodified Addresses

```
%px     01234567 or 0123456789abcdef
```

For printing pointers when you *really* want to print the address. Please consider whether or not you are leaking sensitive information about the kernel memory layout before printing pointers with %px. %px is functionally equivalent to %lx (or %lu). %px is preferred because it is more uniquely grep'able. If in the future we need to modify the way the kernel handles printing pointers we will be better equipped to find the call sites.

Before using %px, consider if using %p is sufficient together with enabling the `no_hash_pointers` kernel parameter during debugging sessions (see the %p description above). One valid scenario for %px might be printing information immediately before a panic, which prevents any sensitive information to be exploited anyway, and with %px there would be no need to reproduce the panic with no_hash_pointers.

Pointer Differences

```
%td     2560
%tx     a00
```

For printing the pointer differences, use the %t modifier for ptrdiff_t.

Example:

```
printk("test: difference between pointers: %td\n", ptr2 - ptr1);
```

Struct Resources

```
%pr     [mem 0x60000000-0x6fffffff flags 0x2200] or
        [mem 0x0000000060000000-0x000000006fffffff flags 0x2200]
%pR     [mem 0x60000000-0x6fffffff pref] or
```

```
[mem 0x0000000060000000-0x000000006fffffff pref]
```

For printing struct resources. The `R` and `r` specifiers result in a printed resource with (R) or without (r) a decoded flags member.

Passed by reference.

Physical address types phys_addr_t

```
%pa[p]  0x01234567 or 0x0123456789abcdef
```

For printing a phys_addr_t type (and its derivatives, such as resource_size_t) which can vary based on build options, regardless of the width of the CPU data path.

Passed by reference.

DMA address types dma_addr_t

```
%pad    0x01234567 or 0x0123456789abcdef
```

For printing a dma_addr_t type which can vary based on build options, regardless of the width of the CPU data path.

Passed by reference.

Raw buffer as an escaped string

```
%*pE[achnops]
```

For printing raw buffer as an escaped string. For the following buffer:

```
1b 62 20 5c 43 07 22 90 0d 5d
```

A few examples show how the conversion would be done (excluding surrounding quotes):

```
%*pE            "\eb \C\a"\220\r]"
%*pEhp          "\x1bb \C\x07"\x90\x0d]"
%*pEa           "\e\142\040\\\103\a\042\220\r\135"
```

The conversion rules are applied according to an optional combination of flags (see :c:func:`string_escape_mem` kernel documentation for the details):

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)printk-formats.rst, line 263); *backlink***
>
> Unknown interpreted text role "c:func".

- a - ESCAPE_ANY
- c - ESCAPE_SPECIAL
- h - ESCAPE_HEX
- n - ESCAPE_NULL
- o - ESCAPE_OCTAL
- p - ESCAPE_NP
- s - ESCAPE_SPACE

By default ESCAPE_ANY_NP is used.

ESCAPE_ANY_NP is the sane choice for many cases, in particularly for printing SSIDs.

If field width is omitted then 1 byte only will be escaped.

Raw buffer as a hex string

```
%*ph    00 01 02  ...  3f
%*phC   00:01:02: ... :3f
%*phD   00-01-02- ... -3f
%*phN   000102 ... 3f
```

For printing small buffers (up to 64 bytes long) as a hex string with a certain separator. For larger buffers consider using :c:func:`print_hex_dump`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)printk-formats.rst, line 292); *backlink***
>
> Unknown interpreted text role "c:func".

MAC/FDDI addresses

```
%pM     00:01:02:03:04:05
%pMR    05:04:03:02:01:00
%pMF    00-01-02-03-04-05
%pm     000102030405
%pmR    050403020100
```

For printing 6-byte MAC/FDDI addresses in hex notation. The `M` and `m` specifiers result in a printed address with (M) or without (m) byte separators. The default byte separator is the colon (:).

Where FDDI addresses are concerned the `F` specifier can be used after the `M` specifier to use dash (-) separators instead of the default separator.

For Bluetooth addresses the `R` specifier shall be used after the `M` specifier to use reversed byte order suitable for visual interpretation of Bluetooth addresses which are in the little endian order.

Passed by reference.

IPv4 addresses

```
%pI4    1.2.3.4
%pi4    001.002.003.004
%p[Ii]4[hnbl]
```

For printing IPv4 dot-separated decimal addresses. The `I4` and `i4` specifiers result in a printed address with (i4) or without (I4) leading zeros.

The additional `h`, `n`, `b`, and `l` specifiers are used to specify host, network, big or little endian order addresses respectively. Where no specifier is provided the default network/big endian order is used.

Passed by reference.

IPv6 addresses

```
%pI6    0001:0002:0003:0004:0005:0006:0007:0008
%pi6    00010002000300040005000600070008
%pI6c   1:2:3:4:5:6:7:8
```

For printing IPv6 network-order 16-bit hex addresses. The `I6` and `i6` specifiers result in a printed address with (I6) or without (i6) colon-separators. Leading zeros are always used.

The additional `c` specifier can be used with the `I` specifier to print a compressed IPv6 address as described by https://tools.ietf.org/html/rfc5952

Passed by reference.

IPv4/IPv6 addresses (generic, with port, flowinfo, scope)

```
%pIS    1.2.3.4         or 0001:0002:0003:0004:0005:0006:0007:0008
%piS    001.002.003.004 or 00010002000300040005000600070008
%pISc   1.2.3.4         or 1:2:3:4:5:6:7:8
%pISpc  1.2.3.4:12345   or [1:2:3:4:5:6:7:8]:12345
%p[Ii]S[pfschnbl]
```

For printing an IP address without the need to distinguish whether it's of type AF_INET or AF_INET6. A pointer to a valid struct sockaddr, specified through `IS` or `iS`, can be passed to this format specifier.

The additional `p`, `f`, and `s` specifiers are used to specify port (IPv4, IPv6), flowinfo (IPv6) and scope (IPv6). Ports have a `:` prefix, flowinfo a `/` and scope a `%`, each followed by the actual value.

In case of an IPv6 address the compressed IPv6 address as described by https://tools.ietf.org/html/rfc5952 is being used if the additional specifier `c` is given. The IPv6 address is surrounded by `[`, `]` in case of additional specifiers `p`, `f` or `s` as suggested by https://tools.ietf.org/html/draft-ietf-6man-text-addr-representation-07

In case of IPv4 addresses, the additional `h`, `n`, `b`, and `l` specifiers can be used as well and are ignored in case of an IPv6 address.

Passed by reference.

Further examples:

```
%pISfc          1.2.3.4         or [1:2:3:4:5:6:7:8]/123456789
%pISsc          1.2.3.4         or [1:2:3:4:5:6:7:8]%1234567890
%pISpfc         1.2.3.4:12345   or [1:2:3:4:5:6:7:8]:12345/123456789
```

UUID/GUID addresses

```
%pUb    00010203-0405-0607-0809-0a0b0c0d0e0f
%pUB    00010203-0405-0607-0809-0A0B0C0D0E0F
```

```
%pUl    03020100-0504-0706-0809-0a0b0c0e0e0f
%pUL    03020100-0504-0706-0809-0A0B0C0E0E0F
```

For printing 16-byte UUID/GUIDs addresses. The additional `l`, `L`, `b` and `B` specifiers are used to specify a little endian order in lower (l) or upper case (L) hex notation - and big endian order in lower (b) or upper case (B) hex notation.

Where no additional specifiers are used the default big endian order with lower case hex notation will be printed.

Passed by reference.

dentry names

```
%pd{,2,3,4}
%pD{,2,3,4}
```

For printing dentry name; if we race with :c:func:`d_move`, the name might be a mix of old and new ones, but it won't oops. %pd dentry is a safer equivalent of %s dentry->d_name.name we used to use, %pd<n> prints `n` last components. %pD does the same thing for struct file.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)printk-formats.rst`, line 424); *backlink***
>
> Unknown interpreted text role "c:func".

Passed by reference.

block_device names

```
%pg     sda, sda1 or loop0p1
```

For printing name of block_device pointers.

struct va_format

```
%pV
```

For printing struct va_format structures. These contain a format string and va_list as follows:

```
struct va_format {
        const char *fmt;
        va_list *va;
};
```

Implements a "recursive vsnprintf".

Do not use this feature without some mechanism to verify the correctness of the format string and va_list arguments.

Passed by reference.

Device tree nodes

```
%pOF[fnpPcCF]
```

For printing device tree node structures. Default behaviour is equivalent to %pOFf.

- f - device node full_name
- n - device node name
- p - device node phandle
- P - device node path spec (name + @unit)
- F - device node flags
- c - major compatible string
- C - full compatible string

The separator when using multiple arguments is ':'

Examples:

```
%pOF    /foo/bar@0                      - Node full name
%pOFf   /foo/bar@0                      - Same as above
%pOFfp  /foo/bar@0:10                   - Node full name + phandle
%pOFfcF /foo/bar@0:foo,device:--P-      - Node full name +
                                          major compatible string +
                                          node flags
                                                D - dynamic
                                                d - detached
```

```
P - Populated
B - Populated bus
```

Passed by reference.

Fwnode handles

```
%pfw[fP]
```

For printing information on fwnode handles. The default is to print the full node name, including the path. The modifiers are functionally equivalent to %pOF above.

- f - full name of the node, including the path
- P - the name of the node including an address (if there is one)

Examples (ACPI):

```
%pfwf   \_SB.PCI0.CIO2.port@1.endpoint@0        - Full node name
%pfwP   endpoint@0                              - Node name
```

Examples (OF):

```
%pfwf   /ocp@68000000/i2c@48072000/camera@10/port/endpoint - Full name
%pfwP   endpoint                                - Node name
```

Time and date

```
%pt[RT]                 YYYY-mm-ddTHH:MM:SS
%pt[RT]s                YYYY-mm-dd HH:MM:SS
%pt[RT]d                YYYY-mm-dd
%pt[RT]t                HH:MM:SS
%pt[RT][dt][r][s]
```

For printing date and time as represented by:

```
R  struct rtc_time structure
T  time64_t type
```

in human readable format.

By default year will be incremented by 1900 and month by 1. Use %pt[RT]r (raw) to suppress this behaviour.

The %pt[RT]s (space) will override ISO 8601 separator by using ' ' (space) instead of 'T' (Capital T) between date and time. It won't have any effect when date or time is omitted.

Passed by reference.

struct clk

```
%pC     pll1
%pCn    pll1
```

For printing struct clk structures. %pC and %pCn print the name of the clock (Common Clock Framework) or a unique 32-bit ID (legacy clock framework).

Passed by reference.

bitmap and its derivatives such as cpumask and nodemask

```
%*pb    0779
%*pbl   0,3-6,8-10
```

For printing bitmap and its derivatives such as cpumask and nodemask, %*pb outputs the bitmap with field width as the number of bits and %*pbl output the bitmap as range list with field width as the number of bits.

The field width is passed by value, the bitmap is passed by reference. Helper macros cpumask_pr_args() and nodemask_pr_args() are available to ease printing cpumask and nodemask.

Flags bitfields such as page flags, gfp_flags

```
%pGp    0x17ffffc0002036(referenced|uptodate|lru|active|private|node=0|zone=2|lastcpupid=0x1fffff)
%pGg    GFP_USER|GFP_DMA32|GFP_NOWARN
%pGv    read|exec|mayread|maywrite|mayexec|denywrite
```

For printing flags bitfields as a collection of symbolic constants that would construct the value. The type of flags is given by the third character. Currently supported are [p]age flags, [v]ma_flags (both expect `unsigned long *`) and [g]fp_flags (expects `gfp_t *`). The flag names and print order depends on the particular type.

Note that this format should not be used directly in the :c:func:`TP_printk()` part of a tracepoint. Instead, use the show_*_flags() functions from <trace/events/mmflags.h>.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)printk-formats.rst, line 593); *backlink***
>
> Unknown interpreted text role "c:func".

Passed by reference.

Network device features

```
%pNF    0x000000000000c000
```

For printing netdev_features_t.

Passed by reference.

V4L2 and DRM FourCC code (pixel format)

```
%p4cc
```

Print a FourCC code used by V4L2 or DRM, including format endianness and its numerical value as hexadecimal.

Passed by reference.

Examples:

```
%p4cc   BG12 little-endian (0x32314742)
%p4cc   Y10  little-endian (0x20303159)
%p4cc   NV12 big-endian (0xb231564e)
```

Thanks

If you add other %p extensions, please extend <lib/test_printf.c> with one or more test cases, if at all feasible.

Thank you for your cooperation and attention.