NAME

hostid — get host ID

SYNOPSIS

hostid

DESCRIPTION

Writes the supposedly-unique identifier of the host, as obtained via gethostid(3), formatted as a zero-padded eight-character hexadecimal string, followed by a newline, to the standard output stream.

SEE ALSO

gethostid(3)

STANDARDS

Compatible with SunOS 4 (Solaris).

CAVEATS

The gethostid(3) function has many inherent problems and varies wildly across implementations and usually not at all across systems, and is not required to mean anything (cf. **HISTORY**, **Standards**, below); you should use neither it, nor **hostid** (which is, additionally, sparsely available, cf. **HISTORY**, **Status quo**, below).

There are *some* useful ways of distinguishing hosts — hostnames, at a small scale, the UUID in /etc/machine-id (systemd machine-id(5)), the *kern.hostuuid* sysctl (FreeBSD sysctl(3)), or one generated on package installation. **hostid** isn't.

HISTORY

The BSD

Appeared in 4.2BSD (undocumented in 4.1cBSD, except gethostid(2), with a single difference, noted below) as

> hostid – set or print identifier of current host system

either %x-formatting gethostid(2), or, if specified, %x-parsing the first argument (*identifier*), and passing it to sethostid(2).

Both hostid(1) and gethostid(2) reiterate that this "32-bit identifier" is "intended to be unique among all UNIX systems in existence.", and is, per norm, a "DARPA Internet address" for the host. They also specify that the identifier is for the "processor", which is a fun thought to consider, given that uniprocessor systems have been extinct for decades now (indeed, contemporary 3B2 systems were multiprocessor ex definitione).

gethostid(2) has a new-in-4.2BSD **BUGS** sexion declaring the retrospectively as obvious as had appeared between April and August of 1983:

> 32 bits for the identifier is too small.

The syscalls do exactly nothing except either returning a global *long hostid;*, or assigning to it if the caller is root (EPERMing otherwise).

hostid(1) also specs that assignation is "usually" done in /etc/rc.local, but the distribution startup scripts do not specify it (which would be impossible without another dedicated program to parse /etc/hosts) or indicate that it should be inserted there. Similarly, the kernel doesn't use *hostid* at all (except for holding it), and the userland doesn't use [gs]ethostid(2) at all (except in **hostid**); although getpid(2) (since 4.1cBSD, but with a typo in the crossref) recommends it, proclaiming that it is most often "used with the host identifier gethostid(2) to generate uniquely-named temporary files.".

4.3BSD declares setting the host ID in /etc/rc.local common, rather than the norm, and allows hostnames (the first address from gethostbyname(3N), if any), then dot-notation addressses via inet_addr(3N) (if the identifier contains a dot ('**.**')), both in host (little) byte order, then a hexadecimal number with an optional **0x/0X** prefix; there's a usage string designating the argument "hexnum or internet address" if all the aforementioned methods fail, but the manual **SYNOPSIS** is not updated (the

DESCRIPTION now says that a number or the hostname may be passed). The no-argument output is **0x**-prefixed (`%#x`-formatted).

Thanks to this, the distribution `/etc/rc.local` runs

> **hostid `hostname`**

but the only other new user is the `hunt`(6) driver, using `gethostid`(2) as the player's machine if configured for playing in the `UNIX` domain (otherwise, in the `INTERNET` domain, it's the calling peer's address in host byte order), and the `getpid`(2) recommendation disappears. `[gs]ethostid`(2) also return/take *long*s — both are 4 bytes on the ILP32 VAX, the only potential reasoning for this would be overfitting to the ARPA network long (4 bytes), as opposed to the network short (2 bytes); it is, nonetheless, inexplicable.

4.3BSD–Tahoe propagates the synopsis change to the **SYNOPSIS**, and the `[gs]ethostid`(2) types to the formatting/parsing specifiers as `%#lx`/`%lx`. It also, unrelatedly, moves network initalisation, including the **hostid** call-site, to `/etc/netstart`.

Networking Release 2 (Net/2) omits **hostid**, with the only user of `gethostid`(2) being as part of the salt for random DES key generation in libdes (`des_crypt`(3)) (the only part of MIT Kerberos 4 in the distribution), and no `sethostid`(2) callers.

4.4BSD sees **hostid**, unchanged, in `/usr/old` (as opposed to `/bin` in previous releases), and `[gs]ethostid`(3) as Standard C Library (libc, –lc) shims in the `compat-43` subdirectory — deprecated in favour of, and implemented in terms of, the new-in-4.4BSD **sysctl** mechanism (documented as `sysctl`(2), but actually `sysctl`(3)) under the {`CTL_KERN`, `KERN_HOSTID`} ("*kern.hostid*" symbolic `sysctl`(8)) name, documented as "Get or set the host id.". The syscalls are likewise retained for compatibility (but only if `COMPAT_43` (**osethostid**()) or either `COMPAT_43` or `COMPAT_SUNOS`, as part of SunOS emulation (**ogethostid**()) is defined at build-time), but renamed to `SYS_o[gs]ethostid`. *hostid* is initialised (other than the **0** default) on the SPARC (Sun-4c) port, attempting compatibility with SunOS — from the ID PROM residing in the Mostek MK48T02 time-of-day clock device's MMIO area (or, from the FORTH PROM perspective, the "eeprom" device's, that also happens to contain the clock, MMIO area), with the highest byte being the machine type, the middle two bytes being the first two bytes of the 3-byte host ID field, and the lowest byte also being the first byte.

4.4BSD–Lite drops **hostid** entirely.

4.4BSD–Lite2 fixes *hostid* initialisation by assigning the final byte of the host ID PROM field to the lowest byte, finally achieving compatibility with SunOS and renames all `o`(ld)-prefixed syscalls to be `compat_43_`-prefixed instead.

This leaves the BSD at exactly two users, ever, being part of the distribution: `hunt`(6) and MIT Kerberos 4 libdes.

SunOS

The 4.2BSD-based SunOS carries its **hostid** and `gethostid`(2), but removes **sethostid**() — `hostid`(1) says simply that

> This numeric value is unique across all hosts.

and `gethostid`(2) is much less verbose (and **BUGS**-free), saying only that the 32-bit identifier is likewise "unique across all hosts".

SunOS 2 actually removes the `sethostid`(2) syscall and clarifies in `hostid`(1) that the value is unique "across all *Sun* hosts" (font change original), but in `gethostid`(2) that it just "should be" unique, and clarifies, that

> On the Sun, this number is taken from the CPU board's ID PROM.

Save for minor maturing formatting choices (including a non-italic "Sun"), the only real difference comes in 4.3BSD-derived SunOS 4 (Solaris) whose **hostid** formats `gethostid`(2) (now decelestialised to "a Sun workstation") as `%08x` (the same as this implementation!). Therein also lies the first user in the distribution: `snap`(1) via list of hosts to administer in `systems`(5) (although, respectively: only available on "Sun 386i systems" running SunOS 4.0, removed in SunOS 4.1; optional, empty by default), explicitly specifying the **hostid** output format.

Predictably, most binaries distributed in the **User_Diag** package use `gethostid`(2). OpenWindows uses it in a few places, most notably (as its inetd server `ttdbserverd`(8) (**`rpc.ttdbserverd`**) is in the **usr** package) for the ToolTalk database.

System V

AT&T System V Release 4 UNIX (x86), as part of the 4.3BSD merge, includes a new `sysinfo`(2) syscall for returning data as strings; some already available as part of `uname`(2) (`SI_SYSNAME`, `SI_HOSTNAME`, `SI_RELEASE`, `SI_VERSION`, `SI_MACHINE`), with the hostname now newly settable via `SI_SET_HOSTNAME` ("unpublished" despite being in the header) — the only official interface being **`sethostid`**() as part of libucb ("`ucblibc`") in the **compat** package (the "BSD compatibility package").

Likewise, therein resides **`gethostid`**(), parsing the result of `SI_HW_SERIAL` — the *`char hw_serial[];`* kernel variable, assigned from the `HW_SERIAL` macro **`"0"`** — as `%12x` (despite (a) the buffer size being `HOSTIDLEN` (**40**) and (b) **12** not matching any reasonable limit (**8** for `%x`, **10** for `%d`, **11** for `%o`)) and **`printf`**()s **`"name = %s"`** the result beforehand.

`/usr/ucb/`**`hostid`**, also relegated to the package, just **`printf`**()s the **`sysinfo`** (`SI_HW_SERIAL`) output (although with a 256-byte buffer).

The 3B2 port defines `HW_SERIAL` to **`"serial number"`** (and doesn't ship the **compat** package) — this is not a respected API on AT&T's part.

SunOS (again)

SunOS 5 (Solaris 2), now based on AT&T System V Release 4 UNIX, inherits its `sysinfo`(2); libucb ("`ucblibc`") **`gethostid`**() is fixed (by removing the **`printf`**(), parsing with **`strtoul`**() in explicit base-**10**, handling the parse error, if any (returning **-1**, later dubbed `HW_INVALID_HOSTID`), and likewise returning **-1** for an ID of **0**).

However, Standard C Library (libc, −lc) *also* carries a `gethostid`(3) implementation, which is both entirely unrelated and entirely identical (except that it uses a slightly bigger buffer, which doesn't matter, since *hw_serial* is **11** *`char`*s (2^{32} + NUL) long and the additional byte's space in the receiving buffer is not passed to `sysinfo`(2) and doesn't return an error for an ID of **0**).

`hostid` also notes an error ("bad hostid format") and exits **1** if `gethostid`(3) returns **-1**, i.e. if the `sysinfo`(2) call fails, which it can't, but is otherwise equivalent to SunOS 4 (Solaris)'.

For all SPARC (Sun-4, SPARC V9) platforms the scheme is the same as in 4.4BSD (unsurprisingly), although expressed much more succinctly as

```
idprom.id_machine << 24 + idprom.id_serial
```

because SPARC (Sun-4) is big-endian. The capitalisation du jour appears to be "IDprom". For Intel platforms (x86, IA64), *hw_serial* (among *hw_provider* (`SI_HW_PROVIDER`) &al.) is supposed to be initialised by the bootloader from `/etc/bootrc` — said code *does* exist for the hostname (by specifying `setprop si-machine` *`the-hostname`*) and *hw_provider* (`si-hw-provider`) — but the only way *hw_serial* is changed from its **`"0"`** initialiser on x86 is by loading, then immediately unloading, the `misc/sysinit` module which takes a lot of complicated-looking math and a manual **`sprintf`**(`%u`) implementation to also write the constant **`"0"`** via the *_hs1107* DDI-workaround symbol. Yes[1]. Reportedly[2], some SunOS distributors provide their own module. COMPAQ, as a vendor, is additionally autodetected on x86 by matching on **`"COMPAQ"`** at a magic address and setting *hw_provider* **`"COMPAQ"`**.

The only users of `gethostid`(3) in the distribution are **`in.ndpd`** (Neighbor Discovery for IPv6, RFC2461), if any non-loopback IPv6 interface exists, **`in.ripngd`** (Routing Information Protocol for IPv6, RFC2080), if the host is an IPv6 router, and **`in.rdisc`** (ICMP Router Discovery Protocol, RFC1256), if the host is an IPv4 router. All of them use it exactly once and exactly in the same way: by calling

```
srandom(gethostid());
```

Standards

X/Open Portability Guide Issue 4, Version 2 ("XPG4.2") includes **gethostid**() as an X/OPEN UNIX Extension; the migration guide describes it succinctly:

> The **gethostid**() function retrieves a 32-bit identifier for the current host. X/Open does not define the domain in which the return value is unique.

Version 2 of the Single UNIX Specification ("SUSv2") moves it to BASE, as an X/Open Systems Interfaces (XSI) extension to the C standards.

Status quo

Platform	Flags	API	Source	Notes
illumos/SPARC	**khx**	SunOS	SPARC ID PROM	Can be configured per-zone
illumos/x86	**kxp**	SunOS	[M1]	(likewise)
NetBSD/SPARC	**khs**	4.4BSD	SPARC ID PROM	
NetBSD/newsmips	**khs**	4.4BSD	ID ROM	4-byte serial field
NetBSD/news68k	**ks**	4.4BSD	Likewise (2-byte on the 1[4567]00), but not exported to *hostid*	
NetBSD/Amiga	**khs**	4.4BSD	Serial of DraCo workstation, stored on battery-backed RTC (`drbbc`(4))	
NetBSD	**ks**	4.4BSD	None; **0**	`SYS_compat_43_o[gs]ethostid` supported
OpenBSD/SPARC64	**khs**	4.4BSD	SPARC ID PROM	
OpenBSD	**ks**	4.4BSD	None; **0**	
FreeBSD (glibc/kFreeBSD)	**ksp**	4.4BSD	[M2]	Private per-jail; supports old syscalls (if configured with `COMPAT_43`), no `SYS_` macros
GNU	**spx**		`/etc/hostid`	[M3]
glibc/Linux	**spx**		[M4]	All userlands provide a compatible **hostid**
musl/Linux	**x**		None; **0**	(likewise)
uClibc[-ng]/Linux	**spx**		[M4]	(likewise); **sethostid**() only if `USE_BSD` (the default)
dietlibc/Linux			N/A	
Bionic/Linux			N/A	`in_posix_and_glibc_but_actually_dead`

Legend:
- **k** stored in the kernel
- **h** hardware-derived
- **s** can be set
- **p** persisted in userspace
- **x** has hostid(1)

Status quo — illumos/x86 [M1]

During kernel startup, `/etc/hostid` is opened; if that succeeds, it's lexed as a series of system config file (newline-delimited, whitespace-insignificant, **#**-commented, typed text, strings **""**-wrapped; `system`(5)) strings, each decoded from ROT47, parsed again as a system config file number (case-insensitive, **-**, **~** operators for negation and bitwise inversion, automatic base detection between **8**, **10**, and **16** from prefix), the final valid of which wins (with invalid numbers warned about). If no valid numbers exist (or the final one was the aforementioned `HW_INVALID_HOSTID` (all bits set), used as an initialiser) a warning is issued about the file being corrupt, and *hw_serial* is unchanged (**"0"**). Following that, another warning is issued about the inability to set the host ID.

If it doesn't, then the legacy `misc/sysinit` module is loaded, which, if succeeds, is followed by parsing *hw_serial*. If it fails, and the system has a non-disabled SMBIOS, then its UUID is used, but not if it's all-zeroes, all-ones, or another known-bad value (in which case this step is skipped with a warning to

"Contact BIOS manufacturer for repair." (as if!)). The host ID is initialised to **0**, then consecutive bytes of the UUID XORed with consecutive bytes thereof, starting with the least significant, wrapping around as necessary, thus distributing all of the UUID bits evenly across the ID. Actual hash algorithms are in modules unavailable during early boot — this is loudly lamented in a comment. The top bit is then discarded to prevent accidental false negatives by constructing `HW_INVALID_HOSTID` and potential sign extension (but this also means that the top bit of the 4th, 8th, 12th, and 16th bytes of the UUID don't contribute to the ID). If this fails (or is skipped), then the bottom 24 bits (except for bits 20 and 21, for no semantic reason) of the Time Stamp Counter (TSC, which increases monotonically when the CPU is running) are used as a "»random«" ID.

If either of these methods worked, the ID is decimalised into *hw_serial* (which, curiously, is `ddi_strtoul`(9F)ed on *every* use).

In the global zone, on non-SPARC, the host ID is saved to `/etc/hostid` (unless it already exists) by the **`hostid`** service, which encodes **`hostid`** output to the effect of

```
echo "\"$(echo "0x$(hostid)" | tr 'P-~!-O' '!-~')\""
```

prepended with a scary "`# DO NOT EDIT`" comment.

Status quo — FreeBSD [M2]

FreeBSD's {`CTL_KERN`, `KERN_HOSTID`} ("*kern.hostid*") actually exposes the full (*`unsigned`*) *`long`* range to userspace, so 64-bit userlands on 64-bit kernels may set it to any 64-bit number. Additionally, it grew {`CTL_KERN`, `KERN_HOSTUUID`} ("*kern.hostuuid*"), which is a **64**-byte (**63** characters + NUL) string.

These are initialised to **0** and **"00000000-0000-0000-0000-000000000000"**, respectively, but see below. Jails may inherit them (otherwise they get the same initial values) at any granularity, or they may be set via the *host.host*[*uu*]*id* variables.

If *hostid_enable* is set in `/etc/rc.conf` (the default), then `/etc/rc/hostid` and `/etc/rc/hostid_save` run on boot, but not in a jail. The former reads a line from *hostid_file* (`/etc/hostid` by default), validates it to make sure it's both in lower-case UUID format (additionaly rejecting broken UUIDs, like all-zero, last-12-bytes-all-one, **1-2-3-4-5**, &c.), and commits it to the sysctls — *kern.hostuuid* verbatim, but *kern.hostid* is the first four bytes of the MD5 hash of the UUID string (no newline) as a hexadecimal number.

If it fails validation, the same process is applied to the *smbios.system.uuid* kernel environment variable set by the bootloader. If that also fails validation, a warning about the situation is issued, boot halted for two seconds, and a fresh UUID is generated with `uuidgen`(1), and committed unconditionally.

The "extra" **`reset`** command behaves as if *hostid_file* failed validation, then writes the UUID to *hostid_file*.

The latter, dependency-ordered after the former, commits the current *kern.hostuuid* to *hostid_file* if it differs from its current contents.

Status quo — GNU [M3]

glibc comments Hurd **`gethostid`**() as

```
/* Return the current machine's Internet number.  */
```

the comment in the body reads

```
/* The hostid is just the contents of the file /etc/hostid,
   kept as text of hexadecimal digits.  */
/* XXX this is supposed to come from the hardware serial number */
```

Conversely, the **`sethostid`**() comment is

```
/* Set the current machine's Internet number to ID.
   This call is restricted to the super-user.  */
/* XXX [...] isn't hostid supposed to be hardwired and unchangeable? */
```

but the body comment

```
/* The hostid is kept in the file /etc/hostid,
   eight characters of upper-case hexadecimal.  */
```

In a classic glibc moment, this indicates: (a) confusion on the part of glibc authors with regards to what *hostid* is, (b) confusion on the part of glibc authors with regards to how they're storing it, (c) confusion on the part of glibc authors with regards to what *hostid* is supposed to be, (d) confusion on the part of glibc authors with regards to what the standard says about `[gs]ethostid`(3), (e) confusion on the part of glibc authors with regards to `[gs]ethostid`(3) prior art.

Status quo — glibc/Linux, uClibc[-ng]/Linux [M4]

Both do the same basic thing:

- Blit a 32-bit unsigned integer from `/etc/hostid`, or
- Get the IPv4 address corresponding to the current hostname (if any), do not decode it from network byte order, swap the upper and lower 2-byte chunks ("to make it not look exactly like the IP"), or
- **0**.

for reading and blit a 32-bit unsigned integer into `/etc/hostid` (erroring if they failed to write all 4 bytes).

This scheme (which glibc calls an "intelligent guess") has a multitude of problems: most hosts don't have an internet address, the address corresponding to the hostname is very rarely the internet address (anymore, at least, cf. **HISTORY**, **The BSD**, 4.3 BSD; in a staggering moment of self-awareness, uClibc[-ng] comments that, indeed, this is usually the loopback address), a staggering amount of hosts are IPv6-only. On a little-endian platform, unless you're running `zfs`(4), it's therefore highly likely that your system's host ID is in fact `007f0101`.

There is, as ever when dealing with glibc, a few minor devils in the minutiae; for **`sethostid`**():

- glibc `EPERM`s if it's in its usual "secure" mode (cf. `secure_getenv`(3) — it's SUID/SGID, has capabilities, or `AT_SECURE` was asserted in the auxiliary vector), uClibc[-ng] if either the real or effective UIDs aren't root;
- glibc `EOVERFLOW`s if the argument exceeds 32-bit limits;
- glibc opens `/etc/hostid` `O_TRUNC` in addition to `O_CREAT, 644` — even if the write fails, the ID is reset.

And for **`gethostid`**():

- glibc uses `/etc/hostid` if it read all 4 bytes, uClibc[-ng] even if it read just one;
- uClibc[-ng] uses `getaddrinfo`(3) with all-zero *`hints`* (so it may get results from any family), and casts all addresses to *`struct sockaddr_in`* — if an IPv6 address is returned, it reads the *`struct sockaddr_in6`* always-**0** *`sin6_flowinfo`* field, residing at the same offset as the expected *`sin_addr`* field (cf. `ip`(7), `ipv6`(7)),
- glibc, on the other hand, uses `gethostbyname_r`(3), querying only IPv4 addresses ex def., so if one wasn't found it returns a **0** in the error path.

So, in their equally byzantine ways, here at least they are exactly identical.

uClibc-ng since 1.0.42 also only quesries IPv4 addresses by specifying *`hints`* `= AF_INET`.

References

[1] **https://lfs.nabijaczleweli.xyz/0017-twitter-export#1528864887954616321**

[2] **https://twitter.com/gedamore/status/1524961429794631680**