OS

> *Stability: 2 - Stable*

The `os` module provides operating system-related utility methods and properties. It can be accessed using:

```
const os = require('os');
```

`os.EOL`

- {string}

The operating system-specific end-of-line marker.

- `\n` on POSIX
- `\r\n` on Windows

`os.arch()`

- Returns: {string}

Returns the operating system CPU architecture for which the Node.js binary was compiled. Possible values are `'arm'`, `'arm64'`, `'ia32'`, `'mips'`, `'mipsel'`, `'ppc'`, `'ppc64'`, `'s390'`, `'s390x'`, and `'x64'`.

The return value is equivalent to `process.arch`.

`os.constants`

- {Object}

Contains commonly used operating system-specific constants for error codes, process signals, and so on. The specific constants defined are described in OS constants.

`os.cpus()`

- Returns: {Object[]}

Returns an array of objects containing information about each logical CPU core.

The properties included on each object include:

- `model` {string}
- `speed` {number} (in MHz)
- `times` {Object}
  - `user` {number} The number of milliseconds the CPU has spent in user mode.
  - `nice` {number} The number of milliseconds the CPU has spent in nice mode.
  - `sys` {number} The number of milliseconds the CPU has spent in sys mode.
  - `idle` {number} The number of milliseconds the CPU has spent in idle mode.
  - `irq` {number} The number of milliseconds the CPU has spent in irq mode.

```
[
  {
    model: 'Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz',
    speed: 2926,
    times: {
      user: 252020,
      nice: 0,
      sys: 30340,
      idle: 1070356870,
      irq: 0
    }
  },
  {
    model: 'Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz',
    speed: 2926,
    times: {
      user: 306960,
      nice: 0,
      sys: 26980,
      idle: 1071569080,
      irq: 0
    }
  },
  {
    model: 'Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz',
    speed: 2926,
    times: {
      user: 248450,
      nice: 0,
      sys: 21750,
      idle: 1070919370,
      irq: 0
    }
  },
  {
    model: 'Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz',
    speed: 2926,
    times: {
      user: 256880,
      nice: 0,
      sys: 19430,
      idle: 1070905480,
      irq: 20
    }
  },
]
```

`nice` values are POSIX-only. On Windows, the `nice` values of all processors are always 0.

`os.devNull`

- {string}

The platform-specific file path of the null device.

- `\\.\nul` on Windows
- `/dev/null` on POSIX

`os.endianness()`

- Returns: {string}

Returns a string identifying the endianness of the CPU for which the Node.js binary was compiled.

Possible values are `'BE'` for big endian and `'LE'` for little endian.

`os.freemem()`

- Returns: {integer}

Returns the amount of free system memory in bytes as an integer.

`os.getPriority([pid])`

- `pid` {integer} The process ID to retrieve scheduling priority for. **Default:** `0`.
- Returns: {integer}

Returns the scheduling priority for the process specified by `pid`. If `pid` is not provided or is `0`, the priority of the current process is returned.

`os.homedir()`

- Returns: {string}

Returns the string path of the current user's home directory.

On POSIX, it uses the `$HOME` environment variable if defined. Otherwise it uses the effective UID to look up the user's home directory.

On Windows, it uses the `USERPROFILE` environment variable if defined. Otherwise it uses the path to the profile directory of the current user.

`os.hostname()`

- Returns: {string}

Returns the host name of the operating system as a string.

`os.loadavg()`

- Returns: {number[]}

Returns an array containing the 1, 5, and 15 minute load averages.

The load average is a measure of system activity calculated by the operating system and expressed as a fractional number.

The load average is a Unix-specific concept. On Windows, the return value is always `[0, 0, 0]`.

`os.networkInterfaces()`

- Returns: {Object}

Returns an object containing network interfaces that have been assigned a network address.

Each key on the returned object identifies a network interface. The associated value is an array of objects that each describe an assigned network address.

The properties available on the assigned network address object include:

- `address` {string} The assigned IPv4 or IPv6 address
- `netmask` {string} The IPv4 or IPv6 network mask
- `family` {string} Either `IPv4` or `IPv6`
- `mac` {string} The MAC address of the network interface
- `internal` {boolean} `true` if the network interface is a loopback or similar interface that is not remotely accessible; otherwise `false`
- `scopeid` {number} The numeric IPv6 scope ID (only specified when `family` is `IPv6`)
- `cidr` {string} The assigned IPv4 or IPv6 address with the routing prefix in CIDR notation. If the `netmask` is invalid, this property is set to `null`.

```
{
  lo: [
    {
      address: '127.0.0.1',
      netmask: '255.0.0.0',
      family: 'IPv4',
      mac: '00:00:00:00:00:00',
      internal: true,
      cidr: '127.0.0.1/8'
    },
    {
      address: '::1',
      netmask: 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff',
      family: 'IPv6',
      mac: '00:00:00:00:00:00',
      scopeid: 0,
      internal: true,
      cidr: '::1/128'
    }
  ],
  eth0: [
    {
      address: '192.168.1.108',
      netmask: '255.255.255.0',
      family: 'IPv4',
      mac: '01:02:03:0a:0b:0c',
      internal: false,
      cidr: '192.168.1.108/24'
    },
    {
      address: 'fe80::a00:27ff:fe4e:66a1',
      netmask: 'ffff:ffff:ffff:ffff::',
      family: 'IPv6',
      mac: '01:02:03:0a:0b:0c',
      scopeid: 1,
      internal: false,
      cidr: 'fe80::a00:27ff:fe4e:66a1/64'
    }
  ]
}
```

`os.platform()`

- Returns: {string}

Returns a string identifying the operating system platform for which the Node.js binary was compiled. The value is set at compile time. Possible values are `'aix'`, `'darwin'`, `'freebsd'`, `'linux'`, `'openbsd'`, `'sunos'`, and `'win32'`.

The return value is equivalent to `process.platform`.

The value `'android'` may also be returned if Node.js is built on the Android operating system. Android support is experimental.

`os.release()`

- Returns: {string}

Returns the operating system as a string.

On POSIX systems, the operating system release is determined by calling `uname(3)`. On Windows, `GetVersionExW()` is used. See https://en.wikipedia.org/wiki/Uname#Examples for more information.

`os.setPriority([pid, ]priority)`

- `pid` {integer} The process ID to set scheduling priority for. **Default:** `0`.
- `priority` {integer} The scheduling priority to assign to the process.

Attempts to set the scheduling priority for the process specified by `pid`. If `pid` is not provided or is `0`, the process ID of the current process is used.

The `priority` input must be an integer between `-20` (high priority) and `19` (low priority). Due to differences between Unix priority levels and Windows priority classes, `priority` is mapped to one of six priority constants in `os.constants.priority`. When retrieving a process priority level, this range mapping may cause the return value to be slightly different on Windows. To avoid confusion, set `priority` to one of the priority constants.

On Windows, setting priority to `PRIORITY_HIGHEST` requires elevated user privileges. Otherwise the set priority will be silently reduced to `PRIORITY_HIGH`.

`os.tmpdir()`

- Returns: {string}

Returns the operating system's default directory for temporary files as a string.

`os.totalmem()`

- Returns: {integer}

Returns the total amount of system memory in bytes as an integer.

`os.type()`

- Returns: {string}

Returns the operating system name as returned by `uname(3)`. For example, it returns `'Linux'` on Linux, `'Darwin'` on macOS, and `'Windows_NT'` on Windows.

See https://en.wikipedia.org/wiki/Uname#Examples for additional information about the output of running `uname(3)` on various operating systems.

`os.uptime()`

- Returns: {integer}

Returns the system uptime in number of seconds.

`os.userInfo([options])`

- `options` {Object}
  - `encoding` {string} Character encoding used to interpret resulting strings. If `encoding` is set to `'buffer'`, the `username`, `shell`, and `homedir` values will be `Buffer` instances. **Default:** `'utf8'`.
- Returns: {Object}

Returns information about the currently effective user. On POSIX platforms, this is typically a subset of the password file. The returned object includes the `username`, `uid`, `gid`, `shell`, and `homedir`. On Windows, the `uid` and `gid` fields are `-1`, and `shell` is `null`.

The value of `homedir` returned by `os.userInfo()` is provided by the operating system. This differs from the result of `os.homedir()`, which queries environment variables for the home directory before falling back to the operating system response.

Throws a `SystemError` if a user has no `username` or `homedir`.

`os.version()`

- Returns {string}

Returns a string identifying the kernel version.

On POSIX systems, the operating system release is determined by calling `uname(3)`. On Windows, `RtlGetVersion()` is used, and if it is not available, `GetVersionExW()` will be used. See https://en.wikipedia.org/wiki/Uname#Examples for more information.

OS constants

The following constants are exported by `os.constants`.

Not all constants will be available on every operating system.

Signal constants

The following signal constants are exported by `os.constants.signals`.

Constant	Description
`SIGHUP`	Sent to indicate when a controlling terminal is closed or a parent process exits.
`SIGINT`	Sent to indicate when a user wishes to interrupt a process (`Ctrl`+`C`).
`SIGQUIT`	Sent to indicate when a user wishes to terminate a process and perform a core dump.
`SIGILL`	Sent to a process to notify that it has attempted to perform an illegal, malformed, unknown, or privileged instruction.
`SIGTRAP`	Sent to a process when an exception has occurred.
`SIGABRT`	Sent to a process to request that it abort.
`SIGIOT`	Synonym for `SIGABRT`
`SIGBUS`	Sent to a process to notify that it has caused a bus error.
`SIGFPE`	Sent to a process to notify that it has performed an illegal arithmetic operation.
`SIGKILL`	Sent to a process to terminate it immediately.
`SIGUSR1` `SIGUSR2`	Sent to a process to identify user-defined conditions.
`SIGSEGV`	Sent to a process to notify of a segmentation fault.
`SIGPIPE`	Sent to a process when it has attempted to write to a disconnected pipe.
`SIGALRM`	Sent to a process when a system timer elapses.
`SIGTERM`	Sent to a process to request termination.
`SIGCHLD`	Sent to a process when a child process terminates.
`SIGSTKFLT`	Sent to a process to indicate a stack fault on a coprocessor.
`SIGCONT`	Sent to instruct the operating system to continue a paused process.
`SIGSTOP`	Sent to instruct the operating system to halt a process.
`SIGTSTP`	Sent to a process to request it to stop.

`SIGBREAK`	Sent to indicate when a user wishes to interrupt a process.
`SIGTTIN`	Sent to a process when it reads from the TTY while in the background.
`SIGTTOU`	Sent to a process when it writes to the TTY while in the background.
`SIGURG`	Sent to a process when a socket has urgent data to read.
`SIGXCPU`	Sent to a process when it has exceeded its limit on CPU usage.
`SIGXFSZ`	Sent to a process when it grows a file larger than the maximum allowed.
`SIGVTALRM`	Sent to a process when a virtual timer has elapsed.
`SIGPROF`	Sent to a process when a system timer has elapsed.
`SIGWINCH`	Sent to a process when the controlling terminal has changed its size.
`SIGIO`	Sent to a process when I/O is available.
`SIGPOLL`	Synonym for `SIGIO`
`SIGLOST`	Sent to a process when a file lock has been lost.
`SIGPWR`	Sent to a process to notify of a power failure.
`SIGINFO`	Synonym for `SIGPWR`
`SIGSYS`	Sent to a process to notify of a bad argument.
`SIGUNUSED`	Synonym for `SIGSYS`

Error constants

The following error constants are exported by `os.constants.errno`.

POSIX error constants

Constant	Description
`E2BIG`	Indicates that the list of arguments is longer than expected.
`EACCES`	Indicates that the operation did not have sufficient permissions.
`EADDRINUSE`	Indicates that the network address is already in use.
`EADDRNOTAVAIL`	Indicates that the network address is currently unavailable for use.
`EAFNOSUPPORT`	Indicates that the network address family is not supported.
`EAGAIN`	Indicates that there is no data available and to try the operation again later.
`EALREADY`	Indicates that the socket already has a pending connection in progress.
`EBADF`	Indicates that a file descriptor is not valid.
`EBADMSG`	Indicates an invalid data message.
`EBUSY`	Indicates that a device or resource is busy.

`ECANCELED`	Indicates that an operation was canceled.
`ECHILD`	Indicates that there are no child processes.
`ECONNABORTED`	Indicates that the network connection has been aborted.
`ECONNREFUSED`	Indicates that the network connection has been refused.
`ECONNRESET`	Indicates that the network connection has been reset.
`EDEADLK`	Indicates that a resource deadlock has been avoided.
`EDESTADDRREQ`	Indicates that a destination address is required.
`EDOM`	Indicates that an argument is out of the domain of the function.
`EDQUOT`	Indicates that the disk quota has been exceeded.
`EEXIST`	Indicates that the file already exists.
`EFAULT`	Indicates an invalid pointer address.
`EFBIG`	Indicates that the file is too large.
`EHOSTUNREACH`	Indicates that the host is unreachable.
`EIDRM`	Indicates that the identifier has been removed.
`EILSEQ`	Indicates an illegal byte sequence.
`EINPROGRESS`	Indicates that an operation is already in progress.
`EINTR`	Indicates that a function call was interrupted.
`EINVAL`	Indicates that an invalid argument was provided.
`EIO`	Indicates an otherwise unspecified I/O error.
`EISCONN`	Indicates that the socket is connected.
`EISDIR`	Indicates that the path is a directory.
`ELOOP`	Indicates too many levels of symbolic links in a path.
`EMFILE`	Indicates that there are too many open files.
`EMLINK`	Indicates that there are too many hard links to a file.
`EMSGSIZE`	Indicates that the provided message is too long.
`EMULTIHOP`	Indicates that a multihop was attempted.
`ENAMETOOLONG`	Indicates that the filename is too long.
`ENETDOWN`	Indicates that the network is down.
`ENETRESET`	Indicates that the connection has been aborted by the network.
`ENETUNREACH`	Indicates that the network is unreachable.
`ENFILE`	Indicates too many open files in the system.

`ENOBUFS`	Indicates that no buffer space is available.
`ENODATA`	Indicates that no message is available on the stream head read queue.
`ENODEV`	Indicates that there is no such device.
`ENOENT`	Indicates that there is no such file or directory.
`ENOEXEC`	Indicates an exec format error.
`ENOLCK`	Indicates that there are no locks available.
`ENOLINK`	Indications that a link has been severed.
`ENOMEM`	Indicates that there is not enough space.
`ENOMSG`	Indicates that there is no message of the desired type.
`ENOPROTOOPT`	Indicates that a given protocol is not available.
`ENOSPC`	Indicates that there is no space available on the device.
`ENOSR`	Indicates that there are no stream resources available.
`ENOSTR`	Indicates that a given resource is not a stream.
`ENOSYS`	Indicates that a function has not been implemented.
`ENOTCONN`	Indicates that the socket is not connected.
`ENOTDIR`	Indicates that the path is not a directory.
`ENOTEMPTY`	Indicates that the directory is not empty.
`ENOTSOCK`	Indicates that the given item is not a socket.
`ENOTSUP`	Indicates that a given operation is not supported.
`ENOTTY`	Indicates an inappropriate I/O control operation.
`ENXIO`	Indicates no such device or address.
`EOPNOTSUPP`	Indicates that an operation is not supported on the socket. Although `ENOTSUP` and `EOPNOTSUPP` have the same value on Linux, according to POSIX.1 these error values should be distinct.)
`EOVERFLOW`	Indicates that a value is too large to be stored in a given data type.
`EPERM`	Indicates that the operation is not permitted.
`EPIPE`	Indicates a broken pipe.
`EPROTO`	Indicates a protocol error.
`EPROTONOSUPPORT`	Indicates that a protocol is not supported.
`EPROTOTYPE`	Indicates the wrong type of protocol for a socket.
`ERANGE`	Indicates that the results are too large.

`EROFS`	Indicates that the file system is read only.
`ESPIPE`	Indicates an invalid seek operation.
`ESRCH`	Indicates that there is no such process.
`ESTALE`	Indicates that the file handle is stale.
`ETIME`	Indicates an expired timer.
`ETIMEDOUT`	Indicates that the connection timed out.
`ETXTBSY`	Indicates that a text file is busy.
`EWOULDBLOCK`	Indicates that the operation would block.
`EXDEV`	Indicates an improper link.

Windows-specific error constants

The following error codes are specific to the Windows operating system.

Constant	Description
`WSAEINTR`	Indicates an interrupted function call.
`WSAEBADF`	Indicates an invalid file handle.
`WSAEACCES`	Indicates insufficient permissions to complete the operation.
`WSAEFAULT`	Indicates an invalid pointer address.
`WSAEINVAL`	Indicates that an invalid argument was passed.
`WSAEMFILE`	Indicates that there are too many open files.
`WSAEWOULDBLOCK`	Indicates that a resource is temporarily unavailable.
`WSAEINPROGRESS`	Indicates that an operation is currently in progress.
`WSAEALREADY`	Indicates that an operation is already in progress.
`WSAENOTSOCK`	Indicates that the resource is not a socket.
`WSAEDESTADDRREQ`	Indicates that a destination address is required.
`WSAEMSGSIZE`	Indicates that the message size is too long.
`WSAEPROTOTYPE`	Indicates the wrong protocol type for the socket.
`WSAENOPROTOOPT`	Indicates a bad protocol option.
`WSAEPROTONOSUPPORT`	Indicates that the protocol is not supported.
`WSAESOCKTNOSUPPORT`	Indicates that the socket type is not supported.
`WSAEOPNOTSUPP`	Indicates that the operation is not supported.
`WSAEPFNOSUPPORT`	Indicates that the protocol family is not supported.

`WSAEAFNOSUPPORT`	Indicates that the address family is not supported.
`WSAEADDRINUSE`	Indicates that the network address is already in use.
`WSAEADDRNOTAVAIL`	Indicates that the network address is not available.
`WSAENETDOWN`	Indicates that the network is down.
`WSAENETUNREACH`	Indicates that the network is unreachable.
`WSAENETRESET`	Indicates that the network connection has been reset.
`WSAECONNABORTED`	Indicates that the connection has been aborted.
`WSAECONNRESET`	Indicates that the connection has been reset by the peer.
`WSAENOBUFS`	Indicates that there is no buffer space available.
`WSAEISCONN`	Indicates that the socket is already connected.
`WSAENOTCONN`	Indicates that the socket is not connected.
`WSAESHUTDOWN`	Indicates that data cannot be sent after the socket has been shutdown.
`WSAETOOMANYREFS`	Indicates that there are too many references.
`WSAETIMEDOUT`	Indicates that the connection has timed out.
`WSAECONNREFUSED`	Indicates that the connection has been refused.
`WSAELOOP`	Indicates that a name cannot be translated.
`WSAENAMETOOLONG`	Indicates that a name was too long.
`WSAEHOSTDOWN`	Indicates that a network host is down.
`WSAEHOSTUNREACH`	Indicates that there is no route to a network host.
`WSAENOTEMPTY`	Indicates that the directory is not empty.
`WSAEPROCLIM`	Indicates that there are too many processes.
`WSAEUSERS`	Indicates that the user quota has been exceeded.
`WSAEDQUOT`	Indicates that the disk quota has been exceeded.
`WSAESTALE`	Indicates a stale file handle reference.
`WSAEREMOTE`	Indicates that the item is remote.
`WSASYSNOTREADY`	Indicates that the network subsystem is not ready.
`WSAVERNOTSUPPORTED`	Indicates that the `winsock.dll` version is out of range.
`WSANOTINITIALISED`	Indicates that successful WSAStartup has not yet been performed.
`WSAEDISCON`	Indicates that a graceful shutdown is in progress.
`WSAENOMORE`	Indicates that there are no more results.
`WSAECANCELLED`	Indicates that an operation has been canceled.

`WSAEINVALIDPROCTABLE`	Indicates that the procedure call table is invalid.
`WSAEINVALIDPROVIDER`	Indicates an invalid service provider.
`WSAEPROVIDERFAILEDINIT`	Indicates that the service provider failed to initialized.
`WSASYSCALLFAILURE`	Indicates a system call failure.
`WSASERVICE_NOT_FOUND`	Indicates that a service was not found.
`WSATYPE_NOT_FOUND`	Indicates that a class type was not found.
`WSA_E_NO_MORE`	Indicates that there are no more results.
`WSA_E_CANCELLED`	Indicates that the call was canceled.
`WSAEREFUSED`	Indicates that a database query was refused.

dlopen constants

If available on the operating system, the following constants are exported in `os.constants.dlopen`. See dlopen(3) for detailed information.

Constant	Description
`RTLD_LAZY`	Perform lazy binding. Node.js sets this flag by default.
`RTLD_NOW`	Resolve all undefined symbols in the library before dlopen(3) returns.
`RTLD_GLOBAL`	Symbols defined by the library will be made available for symbol resolution of subsequently loaded libraries.
`RTLD_LOCAL`	The converse of `RTLD_GLOBAL`. This is the default behavior if neither flag is specified.
`RTLD_DEEPBIND`	Make a self-contained library use its own symbols in preference to symbols from previously loaded libraries.

Priority constants

The following process scheduling constants are exported by `os.constants.priority`.

Constant	Description
`PRIORITY_LOW`	The lowest process scheduling priority. This corresponds to `IDLE_PRIORITY_CLASS` on Windows, and a nice value of `19` on all other platforms.
`PRIORITY_BELOW_NORMAL`	The process scheduling priority above `PRIORITY_LOW` and below `PRIORITY_NORMAL`. This corresponds to `BELOW_NORMAL_PRIORITY_CLASS` on Windows, and a nice value of `10` on all other platforms.
`PRIORITY_NORMAL`	The default process scheduling priority. This corresponds to `NORMAL_PRIORITY_CLASS` on Windows, and a nice value of `0` on all other platforms.
`PRIORITY_ABOVE_NORMAL`	The process scheduling priority above `PRIORITY_NORMAL` and below `PRIORITY_HIGH`. This corresponds to `ABOVE_NORMAL_PRIORITY_CLASS` on Windows, and a nice value of `-7` on all other platforms.

`PRIORITY_HIGH`	The process scheduling priority above `PRIORITY_ABOVE_NORMAL` and below `PRIORITY_HIGHEST`. This corresponds to `HIGH_PRIORITY_CLASS` on Windows, and a nice value of `-14` on all other platforms.
`PRIORITY_HIGHEST`	The highest process scheduling priority. This corresponds to `REALTIME_PRIORITY_CLASS` on Windows, and a nice value of `-20` on all other platforms.

libuv constants

Constant	Description
`UV_UDP_REUSEADDR`	