OS

> OS

- `[Doc]` TTY
- `[Doc]` OS ( )
- `[Doc]`
- `[Basic]`
- `[Point]` CheckList
- `[Basic]`

TTY

"tty" "teletype" , "pty" "pseudo-teletype `/dev/tty*` Unix , , (terminal).

`w` , tty.

```
$ w
 11:49:43 up 482 days, 19:38,  3 users,  load average: 0.03, 0.08, 0.07
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
dev      pts/0    10.0.128.252     10:44    1:01m  0.09s  0.07s -bash
dev      pts/2    10.0.128.252     11:08    2:07   0.17s  0.14s top
root     pts/3    10.0.240.2       11:43    7.00s  0.04s  0.00s w
```

ps tty :

```
$ ps -x
  PID TTY      STAT   TIME COMMAND
 5530 ?        S      0:00 sshd: dev@pts/3
 5531 pts/3    Ss+    0:00 -bash
11296 ?        S      0:00 sshd: dev@pts/4
11297 pts/4    Ss     0:00 -bash
13318 pts/4    R+     0:00 ps -x
23733 ?        Ssl    2:53 PM2 v1.1.2: God Daemon
```

`?` .TTY ,

Node.js stdio isTTY TTY ( ) .

```
$ node -p -e "Boolean(process.stdout.isTTY)"
true
$ node -p -e "Boolean(process.stdout.isTTY)" | cat
false
```

OS

OS .

os.EOL	`End Of Line`
os.arch()	`'x86'`,`'x64'`
os.constants	
os.cpus()	CPU
os.endianness()	CPU`BE`, `LE`
os.freemem()	,
os.homedir()	
os.hostname()	
os.loadavg()	
os.networkInterfaces()	`ifconfig`)
os.platform()	`win32`,`linux`, `process.platform()`
os.release()	
os.tmpdir()	
os.totalmem()	( )
os.type()	`[uname](https://en.wikipedia.org/wiki/Uname#Examples)`
os.uptime()	
os.userInfo([options])	

> “ (EOL) ?

end of line (EOL) newline, line ending, line break.

line feed (LF, `\n`) carriage return (CR, `\r`) . :

LF	Unix Unix (GNU/Linux, AIX, Xenix, Mac OS X, ...) BeOS Amiga RISC OS
CR+LF	MS-DOS (Microsoft Windows) Unix
CR	Apple II , Mac OS 9

EOL , / .

OS

- (Signal Constant `SIGHUP`, `SIGKILL` .
- POSIX (POSIX Error Constant `EACCES`, `EADDRINUSE` .
- Windows (Windows Specific Error Constant `WSAEACCES`, `WSAEBADF` .
- libuv (libuv Constants) `UV_UDP_REUSEADDR`.

Path

Node.js path . , .

Windows vs. POSIX

POSIX		Windows	
path.posix.sep	`'/'`	path.win32.sep	`'\\'`
path.posix.normalize('/foo/bar//baz/asdf/..')	`'/foo/bar/baz/asdf'`	path.win32.normalize('C:\temp\\foo\bar\..\')	`'C:\\temp\\foo\\'`
path.posix.basename('/tmp/myfile.html')	`'myfile.html'`	path.win32.basename('C:\temp\myfile.html')	`'myfile.html'`
path.posix.join('/asdf', '/test.html')	`'/asdf/test.html'`	path.win32.join('/asdf', '/test.html')	`'\\asdf\\test.html'`
path.posix.relative('/root/a', '/root/b')	`'../b'`	path.win32.relative('C:\a', 'c:\b')	`'..\\b'`
path.posix.isAbsolute('/baz/..')	`true`	path.win32.isAbsolute('C:\foo\..')	`true`
path.posix.delimiter	`':'`	path.win32.delimiter	`';'`
process.env.PATH	`'/usr/bin:/bin'`	process.env.PATH	`C:\Windows\system32;C:\Program Files\node\'`
PATH.split(path.posix.delimiter)	`['/usr/bin', '/bin']`	PATH.split(path.win32.delimiter)	`['C:\\Windows\\system32', 'C:\\Program Files\\node\\']`

`path` , mac, :

```
const path = require('path');
console.log(path.basename === path.posix.basename); // true
```

, , .

path

on POSIX:

```
path.parse('/home/user/dir/file.txt')
// Returns:
// {
//    root : "/",
//    dir : "/home/user/dir",
//    base : "file.txt",
//    ext : ".txt",
//    name : "file"
// }
```

```
┌─────────────────────┬────────────┐
│          dir        │    base    │
├──────┐              ├──────┬─────┤
│ root │              │ name │ ext │
"   /    home/user/dir / file  .txt "
└──────┴──────────────┴──────┴─────┘
```

on Windows:

```
path.parse('C:\\path\\dir\\file.txt')
// Returns:
// {
//    root : "C:\\",
//    dir : "C:\\path\\dir",
//    base : "file.txt",
//    ext : ".txt",
//    name : "file"
// }
```

```
┌─────────────────────┬────────────┐
│          dir        │    base    │
```

```
├─────┬──────────────┼──────┬──────┤
│ root │              │ name │  ext │
" C: \      path\dir   \ file  .txt "
└─────┴──────────────┴──────┴──────┘
```

path.extname(path)

case	return
path.extname('index.html')	`'.html'`
path.extname('index.coffee.md')	`'.md'`
path.extname('index.')	`'.'`
path.extname('index')	`''`
path.extname('.index')	`''`

(Command Line Options), CLI . CLI 4 :

- node [options] [v8 options] [script.js | -e "script"] [arguments]
- node debug [script.js | -e "script" | :] ...
- node --v8-options
- REPL

Options

-v, --version	node
-h, --help	
-e, --eval "script"	
-p, --print "script"	`-e`
-c, --check	
-i, --interactive	stdin REPL
-r, --require module	`require`
--no-deprecation	
--trace-deprecation	
--throw-deprecation	
--no-warnings	
--trace-warnings	stack

--trace-sync-io	I/O Event loop stack trace
--zero-fill-buffers	(zero-fill) **Buffer SlowBuffer**
--preserve-symlinks	
--track-heap-objects	
--prof-process	v8 `--prof` Profilling
--v8-options	v8
--tls-cipher-list=list	TLS
--enable-fips	FIPS-compliant crypto
--force-fips	FIPS-compliant
--openssl-config=file	OpenSSL
--icu-data-dir=file	ICU

`NODE_DEBUG=module[,…]`	
`NODE_PATH=path[:…]`	
`NODE_DISABLE_COLORS=1`	REPL
`NODE_ICU_DATA=file`	ICU (Intl object)
`NODE_REPL_HISTORY=file`	REPL
`NODE_TTY_UNSAFE_ASYNC=1`	1 , stdio (console.log)
`NODE_EXTRA_CA_CERTS=file`	CA (VeriSign)

. , , , crash.

Understanding Linux CPU Load CPU .

`uptime`, `top` , Node.js `os.loadavg()` :

```
load average: 0.09, 0.05, 0.01
```

1 , 5 , 15 CPU . CPU 1, CPU .

Node.js pidusage .

CPU , ( ) , .

CheckList

> “ “ ” “ KEY”, “ ?” “ ”

`top` ,`top` ,

, USE checklist

The USE Method provides a strategy for performing a complete check of system health, identifying common bottlenecks and errors. For each system resource, metrics for utilization, saturation and errors are identified and checked. Any issues discovered are then investigated using further strategies.

This is an example USE-based metric list for Linux operating systems (eg, Ubuntu, CentOS, Fedora). This is primarily intended for system administrators of the physical systems, who are using command line tools. Some of these metrics can be found in remote monitoring tools.

Physical Resources

component	type	metric
CPU	utilization	system-wide: `vmstat 1`, "us" + "sy" + "st"; `sar -u`, sum fields except "%idle" and "%iowait"; `dstat -c`, sum fields except "idl" and "wai"; per-cpu: `mpstat -P ALL 1`, sum fields except "%idle" and "%iowait"; `sar -P ALL`, same as `mpstat`; per-process: `top`, "%CPU"; `htop`, "CPU%"; `ps -o pcpu`; `pidstat 1`, "%CPU"; per-kernel-thread: `top`/`htop` ("K" to toggle), where VIRT == 0 (heuristic). [1]
CPU	saturation	system-wide: `vmstat 1`, "r" > CPU count [2]; `sar -q`, "runq-sz" > CPU count; `dstat -p`, "run" > CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); `perf sched latency` (shows "Average" and "Maximum" delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp "queued(us)" [3]

CPU	errors	`perf` (LPE) if processor specific error events (CPC) are available; eg, AMD64's "04Ah Single-bit ECC Errors Recorded by Scrubber" [4]
Memory capacity	utilization	system-wide: `free -m`, "Mem:" (main memory), "Swap:" (virtual memory); `vmstat 1`, "free" (main memory), "swap" (virtual memory); `sar -r`, "%memused"; `dstat -m`, "free"; `slabtop -s c` for kmem slab usage; per-process: `top/htop`, "RES" (resident main memory), "VIRT" (virtual memory), "Mem" for system-wide summary
Memory capacity	saturation	system-wide: `vmstat 1`, "si"/"so" (swapping); `sar -B`, "pgscank" + "pgscand" (scanning); `sar -W`; per-process: 10th field (min_flt) from /proc/PID/stat for minor-fault rate, or dynamic tracing [5]; OOM killer: `dmesg | grep killed`
Memory capacity	errors	`dmesg` for physical failures; dynamic tracing, eg, SystemTap uprobes for failed malloc()s
Network Interfaces	utilization	`sar -n DEV 1`, "rxKB/s"/max "txKB/s"/max; `ip -s link`, RX/TX tput / max bandwidth; /proc/net/dev, "bytes" RX/TX tput/max; nicstat "%Util" [6]
Network Interfaces	saturation	`ifconfig`, "overruns", "dropped"; `netstat -s`, "segments retransmited"; `sar -n EDEV`, *drop and *fifo metrics; /proc/net/dev, RX/TX "drop"; nicstat "Sat" [6]; dynamic tracing for other TCP/IP stack queueing [7]
Network Interfaces	errors	`ifconfig`, "errors", "dropped"; `netstat -i`, "RX-ERR"/"TX-ERR"; `ip -s link`, "errors"; `sar -n EDEV`, "rxerr/s" "txerr/s"; /proc/net/dev, "errs", "drop"; extra counters may be under /sys/class/net/...; dynamic tracing of driver function returns 76]
Storage device I/O	utilization	system-wide: `iostat -xz 1`, "%util"; `sar -d`, "%util"; per-process: iotop; `pidstat -d`; /proc/PID/sched "se.statistics.iowait_sum"

Storage device I/O	saturation	`iostat -xnz 1`, "avgqu-sz" > 1, or high "await"; `sar -d` same; LPE block probes for queue length/latency; dynamic/static tracing of I/O subsystem (incl. LPE block probes)
Storage device I/O	errors	/sys/devices/.../ioerr_cnt; `smartctl`; dynamic/static tracing of I/O subsystem response codes [8]
Storage capacity	utilization	swap: `swapon -s`; `free`; /proc/meminfo "SwapFree"/"SwapTotal"; file systems: "df -h"
Storage capacity	saturation	not sure this one makes sense - once it's full, ENOSPC
Storage capacity	errors	`strace` for ENOSPC; dynamic tracing for ENOSPC; /var/log/messages errs, depending on FS
Storage controller	utilization	`iostat -xz 1`, sum devices and compare to known IOPS/tput limits per-card
Storage controller	saturation	see storage device saturation, ...
Storage controller	errors	see storage device errors, ...
Network controller	utilization	infer from `ip -s link` (or /proc/net/dev) and known controller max tput for its interfaces
Network controller	saturation	see network interface saturation, ...
Network controller	errors	see network interface errors, ...
CPU interconnect	utilization	LPE (CPC) for CPU interconnect ports, tput / max
CPU interconnect	saturation	LPE (CPC) for stall cycles
CPU interconnect	errors	LPE (CPC) for whatever is available
Memory interconnect	utilization	LPE (CPC) for memory busses, tput / max; or CPI greater than, say, 5; CPC may also have local vs remote counters
Memory interconnect	saturation	LPE (CPC) for stall cycles
Memory interconnect	errors	LPE (CPC) for whatever is available
I/O interconnect	utilization	LPE (CPC) for tput / max if available; inference via known tput from iostat/ip/...
I/O interconnect	saturation	LPE (CPC) for stall cycles

I/O interconnect	errors	LPE (CPC) for whatever is available

Software Resources

component	type	metric
Kernel mutex	utilization	With CONFIG_LOCK_STATS=y, /proc/lock_stat "holdtime-totat" / "acquisitions" (also see "holdtime-min", "holdtime-max") [8]; dynamic tracing of lock functions or instructions (maybe)
Kernel mutex	saturation	With CONFIG_LOCK_STATS=y, /proc/lock_stat "waittime-total" / "contentions" (also see "waittime-min", "waittime-max"); dynamic tracing of lock functions or instructions (maybe); spinning shows up with profiling (`perf record -a -g -F 997 ...`, `oprofile`, dynamic tracing)
Kernel mutex	errors	dynamic tracing (eg, recusive mutex enter); other errors can cause kernel lockup/panic, debug with kdump/`crash`
User mutex	utilization	`valgrind --tool=drd --exclusive-threshold=...` (held time); dynamic tracing of lock to unlock function time
User mutex	saturation	`valgrind --tool=drd` to infer contention from held time; dynamic tracing of synchronization functions for wait time; profiling (oprofile, PEL, ...) user stacks for spins
User mutex	errors	`valgrind --tool=drd` various errors; dynamic tracing of pthread_mutex_lock() for EAGAIN, EINVAL, EPERM, EDEADLK, ENOMEM, EOWNERDEAD, ...
Task capacity	utilization	`top`/`htop`, "Tasks" (current); `sysctl kernel.threads-max`, /proc/sys/kernel/threads-max (max)
Task capacity	saturation	threads blocking on memory allocation; at this point the page scanner should be running (sar -B "pgscan*"), else examine using dynamic tracing

Task capacity	errors	"can't fork()" errors; user-level threads: pthread_create() failures with EAGAIN, EINVAL, ...; kernel: dynamic tracing of kernel_thread() ENOMEM
File descriptors	utilization	system-wide: `sar -v`, "file-nr" vs /proc/sys/fs/file-max; `dstat --fs`, "files"; or just /proc/sys/fs/file-nr; per-process: `ls /proc/PID/fd | wc -l` vs `ulimit -n`
File descriptors	saturation	does this make sense? I don't think there is any queueing or blocking, other than on memory allocation.
File descriptors	errors	`strace` errno == EMFILE on syscalls returning fds (eg, open(), accept(), ...).

ulimit

ulimit .

```
-a
-c      core      ,
-d   <     >         ,    KB
-f   <   > shell         ,
-H          ,
-m   <   >         ,     KB
-n   <      >           fd
-p   <    >         ,   512
-s   <   >       ,     KB
-S
-t    CPU      ,
-u   <   >
-v   <     >          ,     KB
```

:

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 127988
```

```
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 655360
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 4096
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

, open socket `ulimit -n` , socket .

Revision #1
Created 19 July 2021 15:19:22 by
Updated 19 July 2021 15:20:08 by