NAME

BSD – Tcl interface to various BSD UNIX functions

SYNOPSIS

::bsd::getloadavg
::bsd::rlimit *get|set soft|hard limitName ?val?*
::bsd::rusage
::bsd::setproctitle *string*
::bsd::statfs *path*
::bsd::getfsstat *?-wait|-nowait?*
::bsd::syslog open *ident logopt facility*
::bsd::syslog close
::bsd::syslog logmask *priority*
::bsd::syslog log *priority message*
::bsd::abort
::bsd::cptime *arrayName*
::bsd::getkey *?timeout?*
::bsd::uptime

DESCRIPTION

The BSD extension provides Tcl programs with new Tcl commands to interface to various system calls and library routines present in BSD UNIX.

::bsd::getloadavg returns the number of processes in the system run queue as a list, representing averages over the last 1, 5, and 15 minutes.

::bsd::rlimit lets you get and set the current soft and hard settings for resource limits. You can specify a 64-bit integer value for any of the limits or **unlimited** to indicate that there is or should be no limit.

::bsd::rlimit limits can be set (or gotten) for the maximum amount of virtual memory the process is allowed to map, in bytes, using the *virtual* parameter. Likewise the largest size core file that can be created, in bytes, can be set (or read) using the *core* parameter.

The maximum amount of CPU time that can be used by the process can be set with the *cpu* parameter, and the maximum size of the data segment of the process can be set with the *data* argument. The largest size file that can be created, in bytes, can be set with the *fsize* parameter, while the maximum size in bytes that a process can lock into memory using the mlock system call can be set with the *memlock* argument.

The maximum number of files that can be opened by the process can be set with the *nofile* argument, while the maximum number of simultaneous processes for this user ID can be set with the *nproc* argument. The maximum size that a process's resident set size may grow (a limit on the amount of physical memory to be given to the process) can be set, in bytes, using the *rss* argument, while the maximum size of socket buffer usage can be controlled using the *sockbuf* argument.

The maximum stack size, in bytes, can be set with the *stack* argument, while the maximum amount of spawp space that can be reserved or used by all of this user's processes can be examined or controlled using the Iswap parameter, while the maximum number of pseudo-terminals created by the user can be set or examined using the *ptys* argument.

Resource limits can be specified as *soft* or *hard*. If a soft limit is exceeded the process may receive a signal but will be allowed to continue until it reaches the hard limit.

::bsd::rlimit set hard data [expr {256 * 1024 * 1024}], for example, sets a hard limit on the data segment size of the process to 256 megs of memory.

See the Berkeley manpage for *getrlimit* and *setrlimit* for further details on the options available and what they do.

::bsd::rusage returns the cumulative resource usage for the current process, as a list of key-value pairs, suitable for loading into an array using *array set*. Resources listed include user CPU, system CPU, max resident set size, shared text Kbyte seconds, unshared data Kbyte seconds, statck Kbyte seconds, page reclaims, page faults, number of times swapped, number of file system inputs and outputs, number of

interprocess communications (IPC) messages sent and received, number of signals delivered, number of voluntary context switches, and number of involuntary context switches. For more information see the BSD *getursage* manpage.

::bsd::setproctitle sets the process title that appears on the *ps* command. The title is set from the executable's name, followed by the specified string. If the string argument begins with a dash characters, the executable's name is skipped. If no string is specified, the process title is restored. For more information see the BSD *setproctitle* manpage.

::bsd::statfs returns information about a mounted filesystem. The *path* argument is the path name of any file within the mounted filesystem. Results are returned as a list of key-value pairs suitable for loading into an array using *array set*. Values returned include the fundamental filesystem block size (*fundamentalFileSystemBlockSize*), the optimal transfer block size (*optimalTransferBlockSize*), the total number of data blocks (*totalDataBlocks*), the total number of free blocks (*freeBlocks*), and the total number of available free blocks (*availableFreeBlocks*). Also included are the total number of filesystem nodes (*totalFileNodes*) and the free file nodes (*freeFileNodes*), the file system type (*fileSystemType*), mount point (*mountPoint), and mounted filesystem name (mountedFileSystem*). Finally a list of flags is provided, which can include **readOnly**, **synchronous**, **noExec**, **noSUID**, **noDev**, **union**, **asynchronous**, **SUIDdir**, **softUpdates**, **noFollowSymlinks**, **noAtime**, **noClusterRead**, **noClusterWrite**, **exportReadOnly**, **exported**, **worldExported**, **anonUidMapping**, **kerberosUidMapping**, **webNFS**, **filesystemStoredLocally**, **quotasEnabled**, **rootFilesystem**, **userMounted**, **multiLabel**, **gjournal** and **aclsEnabled**.

::bsd::getfsstat functions like **statfs**, except that it returns a list of lists with information about all mounted filesystems. If *-nowait* or no option is specified, **getfsstat** will directly return the filesystem information retained in the kernel to avoid delays caused by waiting for updated information from a filesystem that is perhaps temporarily unable to respond, at the cost of the data perhaps not reflecting the absolutely current status of the filesystem. As some of the information returned may be out of date, if *-wait* is specified, **getfsstat** will request updated information for each mounted filesystem prior to returning.

::bsd::syslog provides a direct interface to the syslog library to provide a way to write messages into the system log. **::bsd::syslog open** takes an identifier, a list of zero or more *log options*, and a facility name. The facility name can be one of **auth**, **authpriv**, **console**, **cron**, **daemon**, **ftp**, **kern**, **lpr**, **mail**, **news**, **ntp**, **security**, **syslog**, **user**, **uucp**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, or **local7**.

Log options can include **console**, which causes the message to be logged on the console if there are errors in sending it, **no_delay**, which theoretically says not to delay opening the syslog connection until the first syslog call is made, but check your documentation on syslog(3) to see if that's really true, **pid**, which says to log the process ID along with whatever else, and **perror**, which says to log the message to stderr as well as to the system log.

::bsd::syslog log takes a priority and a message. Priority can be one of **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, or **debug, in descending order of priority. Check your documentation on your syslog library, high priority messages do stuff besides logging, like blasting to all logged in users, while ::bsd::syslog close** closes the connection to the system log. Finally, .fB::bsd::syslog logmask sets the log priority mask and masks all priorities up to and including the specified one. For instance, if you invoked the logmask option with the **info** priority, both *info* and *debug* log messages would be suppressed.

::bsd::abort causes abnormal program termination to occur by issuing an abort signal and will work unless the SIGABRT signal is being caught and the signal handler does not return.

::bsd::cptime takes an array name as an argument and will fill it with key-value pairs representing system, user, nice, interrupt, idle, and total clock counts.

Upon a second invocation with the same array as an argument, it will calculate the cpu time of the various elements as a percentage over the range of elapsed time since the prior call, returning the percentages as a list of key-value pairs and updating the array with the latest sampled values.

::bsd::cptime time; after 1000; puts [::bsd::cptime time]

user 12 nice 0 sys 0 intr 0 idle 87

This is a more instantaneous way to judge CPU utilization than load average, which is a weighted average

that changes slowly over time, so the load average can be 100 but the CPUs may be 95% because really the load average isn't 100, it's just slowly falling back from when it was at or even far above that.

::bsd::getkey reads a single key from the user's terminal and returns it as an integer value. For example, invoking the function and then pressing **a** on the terminal returns **97**.

The *timeout* value, if specified, is the number of seconds to wait for a key to be pressed. It has accuracy to about a tenth of a second and must be a value between 0.0 and 25.5. If no key is pressed within the timeout interval, an empty string is returned. If the timeout is not specified, getkey will wait for an unlimited amount of time for a key to be pressed.

::bsd::uptime returns the number of CPU seconds that the system has been up, as a double-precision floating point number.