NAME

vps — Virtual Private Systems internals

DESCRIPTION

This manual page is supposed to give an overview of how Virtual Private Systems (short VPS) works and how it is implemented.

IMPLEMENTATION NOTES

As much as possible (which is almost all) of the code resides in the *sys/vps* directory.

It can be roughly split up into several components:

core All that absolutely has to be statically linked into the kernel. Very early in the boot process the kernel has to be able to allocate the *vps0* instance, which is the instance reflecting the physical host.

The smaller this part is, the better, as it is even there when VPS is not used.

Files: *vps/vps_core.c*, *vps/vps_priv.c*, *vps/vps_console.c*, *vps/vps_pager.c*, *vps/vps_devfsruleset.h*, *vps/vps_int.h*, *vps/vps.h*

device Provides the */dev/vps* interface and all functions for managing VPS from userspace.

Files: *vps/vps_dev.c*, *vps/vps_user.c*, *vps/vps_user.h*

network interface

The *vpsN* network device. Acts like a virtual layer 3 switch, is the easiest way of connecting up VPS instances to a physical network.

Files: *vps/if_vps.c*

suspend

Suspending and resuming a vps instance.

Files: *vps/vps_suspend.c*

libdump

Common routines for the snapshot and restore modules and userspace programs. Provides functions for reading and manipulating snapshot files and definitions of all involved data structures.

Files: *vps/vps_libdump.c*, *vps/vps_libdump.h*

snapshot

All the snapshot functionality.

Files: *vps/vps_snapst.c*, *vps/vps_snapst.h*

restore

All the restore functionality.

Files: *vps/vps_restore.c*

accounting

All the resource accounting and limiting functionality.

Files: *vps/vps_account.c*, *vps/vps_account.h*

debug Debugging routines, *DDB* integration.

Files: *vps/vps_ddb.c*

Overview how things work:

Taking snapshots

This procedure is quite simple. First all threads have to be suspended, which happens at the end of the *syscall()* function. A flag in the respective vnet instance is set, causing *tcp_input()* and *tcp_output()* to drop incoming data and not sending outgoing data. This is important for live migration.

Then general information about the vps instance is dumped, after that each mount that belongs in the vps context. Next is networking related information like interfaces, routing tables.

Each process gets dumped, including proc structure, user credentials, signal handlers, vmspace, fdset, and threads. Note that the userspace pages (vmspace) are not copied but wired and directly mapped in the vmspace of vpsctl.

When the /dev/vps filehandle on which snapshot was requested is closed again, the snapshot information is deleted and the vps instance remains suspended and can be resumed again.

The resulting snapshot or dump is of a well-defined format. This format is defined in vps/vps_libdump.h and has a version number.

This allows live migration between different kernel versions and kernels compiled with different options that result in different data structures.

Restoring snapshots

This applies for restoring from a file and in live migration as well.

First sanity checks on the restore file have to be performed. First of all a magic pattern in the header, a checksum and the snapshot format version are compared. The dumped objects are serialized using length encoding, so it is necessary to check nothing runs out of bounds.

If the snapshot file is found to be valid, or the user forced it, the actual restore process is started. A new vps instance is allocated, general vps information, mounts, network interfaces, including their flags and addresses, and routing tables are restored. Then sessions (for process groups) are restored, and then each process including all its information. Threads which were interrupted in certain syscalls get fixed up. For instance nanosleep is restarted for sleeping the remaining time. Afterwards the process tree (child/parent relationship, process groups, ...) is fixed up. At the end leftovers are cleaned up and the restored vps instance remains in state suspended, ready for being resumed.

Virtualization of globals

Currently for storing and accessing virtualized global variables *vnet* is used. This can be changed easily by replacing a few macros.

The **process tree**, i.e. the proctree, allproc and zombproc list and locks, are private to each vps instance. This allows each vps instance to run its init task as pid 1, and using the right pids when restoring a vps instance.

Devfs keeps a reference in each mount to the respective vps instance, and can therefore provide virtualized namespace for devices like pseudo ttys. The pts code already uses ucred references, so the only extension is support for restoring pts instances with certain unit numbers. The unit number allocator (kern/subr_unit.c) was extended by unr_alloc_unit().

Privilege checking / Security

By default a reasonable set of priv(9) privileges is given to a vps instance, but it is possible to give any privilege to a vps instance by configuration.

`Resource accounting and limiting`

Currently might not always work as expected.

SEE ALSO

`vps`(4), `vpsctl`(8), `vps.conf`(5), `mount_vpsfs`(8), *http://www.7he.at/freebsd/vps/*

HISTORY

Work on VPS was started in February 2009.

AUTHORS

Virtual Private Systems for FreeBSD and this manual page as well, were written by Klaus P. Ohrhallinger.

Development of this software was partly funded by:

TransIP.nl <http://www.transip.nl/>

BUGS

VPS is in an early stage of development and has to be considered as experimental. This means many bugs have to be expected.

Please submit bug reports to *freebsd-vps@7he.at*.

VERSION

$Id: vps.9 120 2012-08-30 11:14:36Z klaus $