Virtual Private Systems for FreeBSD

Klaus P. Ohrhallinger

06. June 2010

Abstract

Virtual Private Systems for FreeBSD (VPS) is a novel virtualization implementation which is based on the operating system level. Its main design advantage is the multiplexing rather than isolation of global resources, as e.g. "Jail" does. This permits the virtual environments being much more similar to the non-virtual ones, reducing configuration and adaption effort. Additionally it supports live migration which can largely reduce maintenance time and therefore sysadmin hours and service downtimes.

1 Introduction

Virtual Private Systems for FreeBSD, short VPS, is a new operating system level virtualization implementation for FreeBSD.

Virtualization at OS level makes a very small performance overhead and great scalability, compared to other virtualization methods, possible.

It is assumed the reader has basic knowledge about UNIX system internals.

1.1 Common virtualization methods

Emulation of Hardware A complete set of hardware, including a CPU, is emulated in software. The emulator does not need special privileges on the physical host. Any architecture can be emulated. This approach is the slowest of all, but very useful for cross architecture development and debugging.

Example: Bochs.

Hypervisor One small program called "hypervisor" runs on top of the hardware. Its job is to dispatch access to memory, privileged instructions and hardware. One guest instance runs the actual hardware drivers and serves requests for all other guests. Different guest operating systems can be used. Unless the CPU supports virtualization, they all need some modifications for virtual memory mapping and hardware access. This approach is expected to be quite fast and secure but it doesn't scale very well for big numbers of guests.

Examples: XEN, VMware ESX.

OS level virtualization Only one operating system instance is used. The operating system has support for virtualizing itself, creating multiple virtual environments. This approach has a very small performance overhead and overcommitment of memory and diskspace works well.

Examples: AIX Workload Partitions, Solaris Containers, Virtuozzo/OpenVZ (Linux).

1.2 Definitions

FreeBSD The FreeBSD Operating System, on which the VPS virtualization implementation is based on [1].

VPS Short for "Virtual Private Systems for FreeBSD", the name of the implementation covered in this paper.

Jail / Prison A facility in FreeBSD for isolating processes or creating a sort of virtual system instance.

In the userland Jail is referred to as "jail", though inside the kernel as "prison".

Jail appeared in FreeBSD 4.0 and is widely used [2].

VPS instance A single virtual environment provided by the VPS implementation. It looks similar to a physical host running the FreeBSD kernel and is able to run a FreeBSD userland, including the init process.

Live migration The process of moving a VPS instance from the physical host it is currently running on, to a different physical host. "Live" means that after the migration process succeeded, all processes inside the VPS instance are still running, continuing exactly where they were suspended for migration. TCP connections remain valid and functional. Even a user logged into the VPS instance by SSH during the live migration process will only experience a short "freeze".

2 Overview

2.1 Features

Low overhead Since VPS is based on the operating system level, the cpu, memory and diskspace overhead for virtualizing is very small.

Similarity of virtual to non-virtual environments VPS aims to create virtual environments as similar to non-virtual environments as possible. When migrating a non-virtual installation into a VPS instance, only configuration files referring to hardware need modification (e.g. /etc/fstab).

Nested virtualization VPS works in a hierarchical way, allowing VPS instances to create and manage their own VPS child instances. If a resource quota is set for a VPS instance, it has to share its own quota with VPS child instances it creates.

Live migration VPS instances can be migrated from one physical host to another, preserving the running state. This includes all network configuration, running processes and established TCP connections.

Fine grained resource control VPS will support fine grained resource accounting and limiting in the future.

2.2 VPS versus Jail

Although Jail is a great, widely used feature in FreeBSD, its goal was and is increasing security by isolating and constraining processes with low overhead and impact on the FreeBSD code base, rather than virtualizing the system in a full featured way.

Multiplexing versus Isolating The implementation of Jail and VPS differ substantially.

For instance, while VPS keeps one process table/tree for each VPS instance, Jail uses only one global process table and decides for each entry, if it can be "seen" by another Jail instance. Therefore no PID can exist in two Jail instances at the same time, which means no (unmodified) /sbin/init and no live migration are possible.

Unvirtualized resources In FreeBSD Jail, many resources are not virtualized. For some of them, access can be disabled via sysctls, and some of them just remain shared between all virtual environments.

Jail was more meant to be a security feature than a full-featured virtualization facility. In contrast VPS virtualizes any resource necessary for providing feature complete virtual environments that look and act as real as possible.

3 Implementation

3.1 System prerequisites

Currently development is done on a FreeBSD 8.0-RELEASE source tree. When applying the patch to a CURRENT tree it will need some work.

The kernel configuration must include "option VIMAGE". VPS uses VIMAGE for providing virtual network stacks.

Currently only the i386 architecture is supported.

3.2 Virtualization basics

In order to provide virtual environments, all global resources the kernel keeps, need to be either multiplexed or isolated. Multiplexing means keeping a certain resource not only one time but n-times, one time for each virtual instance. Isolating means to restrict access to a certain resource to one certain virtual instance.

A good example for multiplexing is the table of processes. If each virtual instance gets its own process table, each virtual instance can use any PID number it wants, without having collisions between the virtual instances. Furthermore it makes access from one virtual instance to processes of a different virtual instance impossible.

A good example for isolating is access to real hardware like a harddisk drive. There is no point in trying to virtualize a physical harddisk. So only one virtual instance gets the right to use this harddisk drive, while all other virtual instances are isolated from it.

3.3 Major points of code integration

As much as possible of the VPS code resides in its own subdirectory in the kernel source tree. These are the major points where the VPS code gets involved with the FreeBSD code base:

References to global variables References to certain global variables are prefixed with "V_", which are newly declared preprocessor macros that resolve to VPS instance private variables. See 3.4 on the following page for an example.

fork1() and exit1() functions Some changes for dealing with processes without parents, i.e. the init process of a VPS instance.

Device Filesystem devfs Hiding entries based on the VPS instance reference, see 3.6.1 on page 6 for further explanation.

/dev/console device driver Only "vps0" (the "main" VPS instance) performs operations on the actual /dev/console device. Operations made from any other VPS instance context are silently discarded.

Pseudo tty (pts) code Keeping a VPS instance reference in pts devices and being able to allocate devices with given index numbers.

boot() function See 3.6.2 on page 6.

priv_check() interface Extended by a VPS instance dependent privilege check.

Syscall entry and return points Some additional information, required for resuming live migrated threads, is stored.

Kernel initialization Setting up the VPS subsystem.

VFS mounts Keeping a reference to the VPS instance in VFS (Virtual File System) mounts.

TCP input and output routines Delaying or discarding any input/output if the VPS instance is suspended or scheduled to be aborted.

3.4 Multiplexing global variables

A FreeBSD kernel without VPS maintains global variables like the process table, the hostname, number of currently existing processes, and much more.

In a VPS enabled kernel, global variables are replaced by variables private to a VPS instance. Therefore even if no VPS instance is explicitly created, the system knows the instance "vps0", which is the "main system". This instance is created very early at kernel boot and has all privileges.

VPS instances can be created in a hierarchical way, allowing one VPS instance to manage its child instances and pass on part of their resource quotas.

Each "struct ucred" keeps a pointer to the real VPS instance, and each "struct thread" keeps the pointer to the effective VPS instance. A "struct ucred" contains user credentials and is referenced by threads, processes, sockets, some devices as well as some other resources.

The following example (in reverse order) shows how the list of all processes on the system, is virtualized:

Original, non virtualized code:

```
int
fork1(td, flags, pages, procp)
{
   ...
   LIST_INSERT_HEAD(&allproc, p2, p_list);
   ...
}

struct proclist allproc;
```

Modified, virtualized code:

```
int
fork1(td, flags, pages, procp)
{
   ...
   LIST_INSERT_HEAD(&V_allproc, p2, p_list);
   ...
}

#define V_allproc VPSYM(allproc)

#define VPSYM(x) curthread->td_vps->_##x

struct thread {
   ...
   struct vps *td_vps;
   ...
}

struct vps {
   ...
   struct proclist _allproc;
   ...
}
```

This method was taken from the VIMAGE/VNET Network Stack Virtualization.

3.5 Runtime system configuration

A VPS enabled kernel must be able to allocate and free certain resources that would be otherwise only allocated at boot time and never freed.

3.6 Special virtual resources and facilities

3.6.1 Device Filesystem

If a device has a reference to a VPS instance (kept in "struct ucred"), devfs will only show it to the right VPS instance. Currently this is used for virtualizing pseudo ttys. The global registry of devices remains unchanged.

3.6.2 The "boot" system call

Calls to the boot() system call in "vps0" context get dispatched to the original functions. However calls made in any other VPS contexts lead to a VPS instance shutdown or reboot.

3.6.3 Virtual File System Operations

The VFS code is not virtualized. This makes it possible to share filesystems and access directories of VPS child instances. Critical VFS operations are forbidden by default for instances other than "vps0", but can be enabled if needed.

3.7 Snapshots, Restoration and Live Migration

3.7.1 Overview

VPS is capable of creating snapshots of a running VPS instance. A snapshot image includes network interfaces, routing tables, filesystem mounts, processes, process memory, open files, sockets, devices and much more.

A snapshot can be used to restore a VPS instance at a later point or on a different physical host.

The live migration process consists of several operations:
Synchronization of the filesystem to the remote host.
Suspending the local instance.
Second filesystem synchronization.
Creating the snapshot image.
Transferring the snapshot image to the remote host.
Issuing restore command to the remote host.
Aborting the local instance.
Resuming the remote instance.
If the live migration process fails the local instance is resumed.

On success the migrated VPS instance continues its work like nothing had happened. Even established TCP sessions remain functional.

3.7.2 Consistency

To make sure the snapshot content is consistent, the VPS instance has to be suspended. Every thread is removed from sleep queues and the scheduler's list. Threads in uninterruptible sleep (waiting for I/O operations to complete) have to be waited for. A special flag in the virtual network stack is set to keep it from sending or receiving any more data.

After the snapshot is created, or a snapshot was restored, the VPS instance may be resumed.

3.7.3 Dumping

All objects belonging to the VPS instance are dumped to memory. For some objects additional information has to be gathered and saved. Finally the userspace tool writes the snapshot image to a file or sends it over the network. All userspace memory pages are directly mapped into the userspace tool's memory, rather than being copied.

There is no way to determine in advance the amount of memory needed to hold the snapshot image. As soon as an object is unlocked, it may change or even

vanish. Therefore VPS reserves a quite big virtual continuous address space and maps in physical pages on demand. While holding locks that forbid sleeping, allocating pages may fail. All snapshot routines that hold non-sleepable locks while allocating pages are able to restart, in case memory allocation fails.

3.7.4 Restore

After a sanity check of the snapshot file, all objects are restored, eventually creating a full VPS instance.

Threads which were in a syscall when the snapshot was made, restart the syscall if possible or return to userspace with EINTR (Interrupted system call).

Some kind of objects have dependencies which make it impossible to restore one by one. For instance connected unix domain sockets keep references to each other. Therefore each restored object that might have dependencies like this, is put on a temporary list, including information like an ID number and the pointer it had when it was dumped. During the restore run, objects can be looked up on this list whenever a reference is to be resolved. Apart from rebuilding dependency trees, this list is also used to run some fixup operations at the end of the restore process.

3.8 Virtual Networking

The network part is done by including VIMAGE/VNET network stack virtualization. Each VPS instance keeps a pointer to one, private VNET instance.

For easy interconnecting of VPS instances the if_vps network device was written, which basically acts as a layer 3 switch.

IP packets are received on the if_vps interface that owns the destination address and is allowed to use it by means of VPS instance configuration. If no valid output interface is found, the packet is received on “vps0”, which is supposed to be owned by the “main” VPS instance.

In a typical setup, the physical host would be connected via ethernet and maintain published arp entries for its VPS instances. Instead of if_vps any other network solution can be used.

3.9 Privilege Checking

Unlike FreeBSD Jail, VPS does not have many points where privileges must be checked.

Most of the work is already done by having separate system globals. E.g. no VPS instance is able to access processes of another instance, because it simply has no access to process tables of others.

3.10 Management

Management of VPS is done using the ”vpsctl” userspace command, which in turn performs ioctl and mmap operations on /dev/vps.

3.11 Configuration

Configuration of instances is kept in config files, which are read by the "vpsctl" userspace command.

Typically a file would include a VPS instance name, where and how to mount the vfs root, number and type of network interfaces, allowed IP addresses and resource limits.

4 Future

4.1 Current status and focus in further development

Testing, improving stability, adding features Currently VPS has to be considered highly experimental. Loads of things are implemented and work quite well, but there are still unsupported system resources, missing privilege checks and bugs.

The focus in the next time will be on extended systematic testing, in order to get the code as stable as possible. On a more long term view, VPS will get more feature complete for typical uses, like running common server software (e.g. Apache, Postgres, some scripting languages) and being able to reliably live migrate them.

Resource accounting and limiting At the time of writing there is no resource accounting or limiting implemented.

However the goal is to have a fine grained resource limit configuration, allowing to keep VPS instances from affecting each other, but also making overcommitment possible.

This will include limiting and sharing CPU and I/O bandwidth according to configured constraints.

Specification of the Snapshot file format At the time of writing, VPS writes e.g. a "struct proc" structure to the snapshot file as it is. On the restore host this structure is read using the restore host's definition of "struct proc". Therefore, if the two hosts differ in the definition of "struct proc", data will be interpreted in a wrong way most likely leading to a kernel panic.

A snapshot file format specification, including a version number, will define intermediate structures. This will make migration between hosts running different FreeBSD versions and even between i386 and amd64 possible.

Support of architectures other than i386 VPS has only very little architecture dependent code. If development hosts, by means of physical or emulated hardware, become available, porting can be done very easily.

Feature completeness On a long term view VPS will support virtualization and live migration of any possible setup.

4.2 Potential use cases

Server consolidation The usual way of server configuration is to have one physical host per task or service. Often this results in many servers which are idling most of the time. On the other hand, putting too many tasks into one installation makes maintenance much more difficult and prone to security issues and misconfiguration.

The smart way is to setup a few VPS enabled servers and create one VPS instance for each task or service. The system load can be distributed easily by live migrating virtual servers as needed. Even hardware maintenance can be done without service outage. All the affected virtual servers can be live migrated to other physical hosts and moved back when the maintenance is finished. This way saves hardware, rackspace and power.

Mass hosting VPS allows customers to have their own servers that behave almost like physical hosts, but cost the hosting provider only very little resources. Due to the hierarchical nature of VPS, customers can even setup their own VPS child instances in their instances, if allowed.

Separation of services While having more (virtual) servers means more maintenance effort, security and manageability can be increased this way.

4.3 Call for participation

Testing VPS needs testing!

Patches and binaries as a drop-in replacement for FreeBSD releases are available at [3].

Development Any help in developing VPS is welcome. See [3] for more information.

Bug reports Bug reports are welcome at [4].

References

[1] http://www.freebsd.org/

[2] http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/jail.html

[3] http://www.7he.at/freebsd/vps/

[4] http://bugzilla.7he.at/