Virtual Private Systems for FreeBSD

by Klaus P. Ohrhallinger
October 2010

What is VPS ?

- A novel virtualization implementation
- Based on the operating system level

Live demonstration

- Two hosts running FreeBSD 8.1
- Starting up a preconfigured VPS instance
- Live migrating it from host A to host B
- SSH session and running programs remain functional

Virtualization methods

- Emulation of Hardware
- Hypervisor
- OS level virtualization
- Other methods ...

Features

- Low virtualization overhead
- Similarity of virtual to non-virtual environments
- Nested virtualization
- Live migration
- Fine grained resource control

VPS versus Jail

- Jail:
  - Great feature
  - Was first meant to isolate and constraint processes and process groups, rather than being a OS virtualization implementation
- VPS:
  - Multiplexing globals and resources instead of isolating them
  - Providing any resource a non virtual system would have

OS level virtualization

- Basically any global resource needs to be multiplexed or isolated
- Multiplexing:
  - Resource exists n times rather than 1 time
  - Allocated and destroyed on demand
  - E.g. process table: each VPS instance has its own
- Isolating:
  - E.g.: a harddisk: only one VPS instance (typically vps0) is allowed access

Implementation

Multiplexing globals

- Example: process table
  - One table for each VPS instance
  - Each VPS instance needs PID '1' for /sbin/init
  - Live migration allocates certain PID numbers
  - VPS instance can only "see" its own processes
  - No "p_cansee()"-style check necessary

Multiplexing globals

- Original code:

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

- Multiplexing code:

```
Int
fork1(rd, 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;
        ...
}
```

Major integration points

- References to global variables
- fork1() and exit1() functions
- Device Filesystem devfs
- /dev/console device driver
- Pseudo-tty (pts) code
- reboot() function

Major integration points

- priv_check() interface
- Syscall entry and return points
- Kernel initialization
- VFS mount operations
- TCP input and output routines

Runtime system configuration

- Some objects are allocated on boot and never freed.
- VPS has to free destroyed instances entirely.
- Examples:
  - ...

Special virtual resources

- For some resources, special treatment is necessary:
  - Device filesystem devfs
  - The reboot() system call
  - Virtual File System (VFS) operations

Device filesystem devfs

- Hiding devfs entries by means of devfs rules
- Each devfs mount keeps VPS reference
- "User devices" like pseudo terminals only show up in right devfs mount
- Global registry of devices is unchanged

The reboot() syscall

- Any VPS instance can call reboot()
- Only "vps0" executes the actual reboot() call
- Other vps instances halt or reboot themselves

Virtual File System VFS

- Virtualizing VFS would be too tricky
- Possibility of sharing filesystems
- Accessing directories of child instances possible

Live migration

- Snapshot and restore functionality
- → Live migration:
  - First filesystem synchronization
  - Suspending VPS instance on local host
  - Second filesystem synchronization
  - Creating snapshot image
  - Transferring snapshot image

Live migration

- Issuing restore command to remote host
- On error, resuming local instance
- On success:
- Aborting local instance
- Resuming remote instance
- Announcing remote instance on network

- Didn't lose a single TCP connection

Consistency

- VPS instance has to be suspended
  - Removing every thread from sleep queues and scheduler
  - Waiting for threads in uninterruptible sleep
  - Setting a flag to keep network stack from receiving data.
  - For resuming the instance, everything has to be consistent again.

Dumping

- General VPS instance information
- VFS mounts
- Network stack:
  - Interfaces, state and addresses
  - Routing tables
  - More settings and counters
- SYSV IPC

Dumping

- Processes:
  - Process information
  - Threads
  - Virtual Memory (VM) space
  - Userspace pages mapped into vpsctl's vmspace
  - File descriptors
  - Sockets, socket buffers
  - Much much more

Dumping

- Size of snapshot can't be predicted
  - While dumping and holding non-sleepable locks no memory allocation possible
  - Snapshot shall be in continuous memory
  - Reserving huge continuous space for entire dump and mapping physical pages in as needed
  - Snapshot functions have to unlock and try again when memory is available

Restore

- Sanity check of snapshot dump
- Resource availability check
- Maintaining list of restored objects to resolve circular references
- Syscalls get prepared to be restarted or return EINTR
- VPS instance is in suspended state

Virtual Networking

- Uses the VNET/VIMAGE network stack virtualization
- Many different ways of interconnecting VPS instances
- if_vps
  - Layer 3 switch
  - Address has to be owned by VPS instance
  - Published ARP entries on physical ethernet

Privilege Checking

- Not many additional checks neccessary:

  multiplexing globals keeps instances separated
  - E.g. separate process tables instead of p_cansee() style function.
- priv_check() → vps_priv_check()
  - Any single PRIV_* privilege can be configured per VPS instance to be either "allowed", ENOSYS or EPERM.

Management

- vpsctl command
  - start, stop
  - suspend, resume
  - snapshot, restore
  - migrate
  - Configuration files
- /dev/vps
  - mmap()
  - ioctl()

Configuration

- One configuration file for each VPS instance
  - VPS instance name
  - Mount command for VFS root
  - Number and type of network interfaces
  - Allowed IP addresses
  - Resource limits
  - On migration, configuration file gets synced

Future

Current status and focus in further developmet

- Testing, improving stability, adding features
- Resource accounting and limiting
- Specification of the snapshot format
- Support for other architectures than i386
- Feature completeness

Testing, improving and new features

- Currently VPS is highly experimental
  - Unsupported resources
  - Missing privilege checks
  - Bugs
- Go for stability
- Feature completeness for typical use cases
- Being able to live migrate them reliably

Resource accounting

- Currently not implemented at all
- Soft and hard limits for any resource, allowing overcommitment
- CPU usage and I/O bandwidth configurable
- VPS aware scheduler
- A properly constrained VPS instance must not be able to affect the host system or other instances

Specification of snapshot format

- Currently kernel structures are dumped directly
- Incompatibilities between different kernel versions
- Define own intermediate structures
- Version number on snapshot format
- Userspace tool for converting between versions
- Interoperability between i386 and amd64

Support for other architectures

- Currently only i386 supported
- Very little architecture dependent code
- As development boxes become available porting can be done easily

Feature completeness

- Ability of copying physical host into VPS instance, only changing hardware related configuration like /etc/fstab
- Behavior exactly like on physical host
- No unsupported (not virtualized) resources

Potential use cases

- Server consolidation
- Mass hosting
- Separation of services
- Easier engineering/development

Server consolidation

- A few big physical hosts instead of many idling hosts
- Better utilization of the hardware
- Distributing load by moving VPS instances
- Hardware maintenance possible without shutdown of services and almost no outage
- Easier disaster recovery

Mass hosting

- With shadow filesystem, only a few MBs per instance needed
- VPS instances behave almost like physical host
- Excellent resource overcommitment possible
- Customers can setup and manage their own child instances if allowed

Separation of services

- One VPS instance per task or service
  - Increasing security
  - Host setups are simpler, therefore easier to maintain

Easier engineering/development

- "Staging" engineering
  - Setting up, configuring and testing host setups in VPS instances on development hardware
  - When ready, migrate VPS instances onto production hardware
  - Engineering in VPS instance and deploying to physical hardware
- Snapshots
  - Easy backups
  - Easy "rollback"

Management on a large scale

- Decentralized Management
  - No dedicated "management" hosts necessary
  - Not introducing Single Point of Failures.
- Daemon on each VPS server
  - Object orientated implemenation
  - Privilege separation by multiple process design.
- Communication Protocol in JSON
  - Easy to handle and human-readable

Management on a large scale

- Fancy GUI:
  - Connects up to multiple VPS servers
  - Is able to set up one-time authentication between VPS servers for migration
  - Portable thanks to wxWidgets
  - Drag'n'Drop

Management on a large scale

- Optional Web interface for customers
  - Provides maintenance and disaster recovery functionality like kill, restart, backup/restore, reinstall, ...
  - Runs unprivileged and connects up to the actual VPS server daemons.
- Integration into ISP's own infrastructure
  - Protocol is well defined
  - E.g. VPS instance fully automatically created, installed and started up by an online shop.

Participation

- Testing
- Submitting bug reports
- Submitting patches
- Any other help is welcome as well
- Further reading, bug tracker, source and binary sets are available at:

  http://www.7he.at/freebsd/vps/

Thanks !

http://www.7he.at/freebsd/vps/