Linux Virtualization with Lguest

John Quigley
www.jquigley.com
jquigley@jquigley.com

Chicago Linux
January 2008

Definitions

initrd: initial ram disk

page tables: mapping between virtual and physical addresses

virtual address: unique to the accessing process

physical address: unique to the CPU

Definitions

initrd: initial ram disk

page tables: mapping between virtual and physical addresses

virtual address: unique to the accessing process

physical address: unique to the CPU

Definitions

initrd: initial ram disk

page tables: mapping between virtual and physical addresses

virtual address: unique to the accessing process

physical address: unique to the CPU

Definitions

initrd: initial ram disk

page tables: mapping between virtual and physical addresses

virtual address: unique to the accessing process

physical address: unique to the CPU

Definitions

initrd: initial ram disk

page tables: mapping between virtual and physical addresses

virtual address: unique to the accessing process

physical address: unique to the CPU

Definitions

supervisor: priviledged mode of operation

hypervisor: virtual machine monitor or governor

paravirtualization: virtual software interface to the machine

hierarchical protection domains: layers of software priviledge

Definitions

supervisor: priviledged mode of operation

hypervisor: virtual machine monitor or governor

paravirtualization: virtual software interface to the machine

hierarchical protection domains: layers of software priviledge

Definitions

supervisor: priviledged mode of operation

hypervisor: virtual machine monitor or governor

paravirtualization: virtual software interface to the machine

hierarchical protection domains: layers of software priviledge

Definitions

supervisor: priviledged mode of operation

hypervisor: virtual machine monitor or governor

paravirtualization: virtual software interface to the machine

hierarchical protection domains: layers of software priviledge

Definitions

supervisor: priviledged mode of operation

hypervisor: virtual machine monitor or governor

paravirtualization: virtual software interface to the machine

hierarchical protection domains: layers of software priviledge

Types of Virtualization

Hypervisors come in two general flavors.

Type 1, runs native on bare-metal (eg. Xen, KVM, ESX)

Type 2, hosted (eg. Workstation, UML, QEMU/Bochs)

Types of Virtualization

Hypervisors come in two general flavors.

Type 1, runs native on bare-metal (eg. Xen, KVM, ESX)

Type 2, hosted (eg. Workstation, UML, QEMU/Bochs)

Types of Virtualization

Hypervisors come in two general flavors.

Type 1, runs native on bare-metal (eg. Xen, KVM, ESX)

Type 2, hosted (eg. Workstation, UML, QEMU/Bochs)

Types of Virtualization

The interface by which virtualization is exposed is also binary:

Paravirtualization, guest OS redirects to hypervisor

Pure Virtualization, guest OS traps to hypervisor

Native virtualization requires hardware support:

- Intel: VT
- AMD: AMD-V

Types of Virtualization

The interface by which virtualization is exposed is also binary:

Paravirtualization, guest OS redirects to hypervisor

Pure Virtualization, guest OS traps to hypervisor

Native virtualization requires hardware support:

1. **Intel**: IVT
2. **AMD**: AMD-V

Types of Virtualization

The interface by which virtualization is exposed is also binary:

Paravirtualization, guest OS redirects to hypervisor

Pure Virtualization, guest OS traps to hypervisor

Native virtualization requires hardware support:

1. **Intel**: IVT
2. **AMD**: AMD-V

Types of Virtualization

The interface by which virtualization is exposed is also binary:

Paravirtualization, guest OS redirects to hypervisor

Pure Virtualization, guest OS traps to hypervisor

Native virtualization requires hardware support:

1. **Intel**: IVT
2. **AMD**: AMD-V

Types of Virtualization

The interface by which virtualization is exposed is also binary:

Paravirtualization, guest OS redirects to hypervisor

Pure Virtualization, guest OS traps to hypervisor

Native virtualization requires hardware support:

1. **Intel**: IVT
2. **AMD**: AMD-V

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

Linux Virtualization

Some popular solutions for Linux, and the type of virtualization they employ, are enumerated below.

1. **KVM**: type 1, pure/para
2. **Xen**: type 1, pure/para
3. **Lguest**: type 1, para
4. **UML**: type 2, para
5. **QEMU**: type 2, pure
6. **Bochs**: type 2, pure

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

- no Xen-style live migration
- no UML-style copy-on-write disk devices
- no resource management beyond what the kernel already provides
- no PAE or SMP support whatsoever
- is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

- no Xen-style live migration
- no UML-style copy-on-write disk devices
- no resource management beyond what the kernel already provides
- no PAE or SMP support whatsoever
- is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

- no xen-style live migration
- no UML style copy-on-write disk devices
- no resource management beyond what the kernel already provides
- no PAE or SMP support whatsoever
- is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

In development since 2005, previously known as lhype

Went upstream into Linus' kernel in a merge for 2.6.23

Provides very limited virtualization implementation:

1. no xen-style live migration
2. no UML-style copy-on-write disk devices
3. no resource management beyond what the kernel already provides
4. no PAE or SMP support whatsoever
5. is x86-only

About Lguest

Overall, it's a very clean and elegant design.

The simplicity of the code is one of it's attractive qualities.

Written with a hat tip to Donald Knuth, by employing literate programming style.

About Lguest

Overall, it's a very clean and elegant design.

The simplicity of the code is one of it's attractive qualities.

Written with a hat tip to Donald Knuth, by employing literate programming style.

About Lguest

Overall, it's a very clean and elegant design.

The simplicity of the code is one of it's attractive qualities.

Written with a hat tip to Donald Knuth, by employing literate programming style.

About Lguest

Overall, it's a very clean and elegant design.

The simplicity of the code is one of it's attractive qualities.

Written with a hat tip to Donald Knuth, by employing literate programming style.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

- maps guest kernel image into host's memory
- opens /dev/lguest and writes config info about guest
- hypervisor uses this to initialize and launch guest
- open pipe/file used for ongoing control/console I/O
- DMA-like I/O virtualment

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open pipe/file used for ongoing control/console I/O
5. DMA-like I/O virtualization

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Launcher

Launcher is an ELF userspace app that launches and monitors guests

It operates by performing the following tasks:

1. maps guest kernel imagine into host's memory
2. opens /dev/lguest and writes config info about guest
3. hypervisor uses this to initialize and launch guest
4. open procfile used for ongoing control, console I/O, DMA-like I/O via shmem

NOTE: Physical memory in the Guest is the Host's virtual memory.

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The hypervisor is simply a loadable kernel module ... cool!

Being able to insert a module and start new guest provides "low commitment" path to virtualization

Provides the /dev/lguest interface, whereby userspace launcher controls and communicates with guest

First write tells us memory size, pagetable, entry point, kernel offset

A read will run guest until pending signal (-EINTR), or guest does DMA out to launcher

Lguest Hypervisor

The Lguest hypervisor is unique in several ways.

Host domain is simply a normal kernel in ring-0

Hypervisor is loaded into top of kernel memory

Lguest Hypervisor

The Lguest hypervisor is unique in several ways.

Host domain is simply a normal kernel in ring-0

Hypervisor is loaded into top of kernel memory

Lguest Hypervisor

The Lguest hypervisor is unique in several ways.

Host domain is simply a normal kernel in ring-0

Hypervisor is loaded into top of kernel memory

Lguest Hypervisor

The Lguest hypervisor is unique in several ways.

Host domain is simply a normal kernel in ring-0

Hypervisor is loaded into top of kernel memory

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Lguest Hypervisor

Hypervisor contains only core facilities:

1. domain switching code
2. interrupt handlers
3. few low-level object that need to be virtualized
4. array of structs to maintain info for each guest domain

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

- Paravirtualized guest support = Y
- -> Lguest guest support = Y
- High Memory support = Off (or 4GB)
- Alignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

- Paravirtualized guest support = Y
- -> Lguest guest support = Y
- High Memory Support = Off (or 4GB)
- Alignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

1. Paravirtualized guest support = Y
2. -> Lguest guest support = Y
3. High Memory Support = Off (or 4GB)
4. Allignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

1. Paravirtualized guest support = Y
2. -> Lguest guest support = Y
3. High Memory Support = Off (or 4GB)
4. Allignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

1. Paravirtualized guest support = Y
2. -> Lguest guest support = Y
3. High Memory Support = Off (or 4GB)
4. Allignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

1. Paravirtualized guest support = Y
2. -> Lguest guest support = Y
3. High Memory Support = Off (or 4GB)
4. Allignment value of kernel = 0x100000

Kernel Preparations

You'll probably have to custom-configure your kernel.

Processor type and features ->

1. Paravirtualized guest support = Y
2. -> Lguest guest support = Y
3. High Memory Support = Off (or 4GB)
4. Allignment value of kernel = 0x100000

Kernel Preparations

Device Drivers ->

-> Network Device Support -> TUN/TAP = M/Y

-> Block Devices -> Virtio block driver = Y

Virtualization ->

Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Kernel Preparations

Device Drivers ->

1. -> Network Device Support -> TUN/TAP = M/Y
2. -> Block Devices -> Virtio block driver = Y

Virtualization ->

- [ ] Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Kernel Preparations

Device Drivers ->

1. -> Network Device Support -> TUN/TAP = M/Y
2. -> Block Devices -> Virtio block driver = Y

Virtualization ->

3. Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Kernel Preparations

Device Drivers ->

1. -> Network Device Support -> TUN/TAP = M/Y
2. -> Block Devices -> Virtio block driver = Y

Virtualization ->

1. Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Kernel Preparations

Device Drivers ->

1. -> Network Device Support -> TUN/TAP = M/Y
2. -> Block Devices -> Virtio block driver = Y

Virtualization ->

1. Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Kernel Preparations

Device Drivers ->

1. -> Network Device Support -> TUN/TAP = M/Y
2. -> Block Devices -> Virtio block driver = Y

Virtualization ->

1. Linux hypervisor example code = M

TIP: Hypervisor as module only, disable graphics facilities like AGP, DRI, fb

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the 'lg' module in kernel

Configure host masquerading:
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the 'lg' module in kernel

Configure host masquerading:
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the 'lg' module in kernel

Configure host masquerading:
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the 'lg' module in kernel

Configure host masquerading:
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the `lg' module in kernel

Configure host masquerading:

```
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Launcher Preparations

Next task is to build the user-space launcher.

Compile client driver with a standard call to make.

You'll also need an initial root disk image.

Let's get ready to run ... load the `lg' module in kernel

Configure host masquerading:
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
&&
echo 1 > /proc/sys/net/ipv4/ip_forward

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

- 64 – amount of memory to use, in megabytes
- vmlinux – kernel image, or standard bzImage
- –tunnet – configures a 'tap' device for networking
- –block – initial ram disk to employ
- root=/dev/vda ... – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

- 64 – amount of memory to use, in megabytes
- vmlinux – kernel image, or standard bzImage
- –tunnet – configures a "tap" device for networking
- –block – initial ram disk to employ
- root=/dev/vda ... – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

Run Guest

$ Documentation/lguest/lguest –tunnet=129.168.0.1 –block=initrd 64 vmlinux root=/dev/vda

A description of the options:

1. **64** – amount of memory to use, in megabytes
2. **vmlinux** – kernel image, or standard bzImage
3. **–tunnet** – configures a `tap' device for networking
4. **–block** – initial ram disk to employ
5. **root=/dev/bda ...** – kernel boot parameters

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceeds to try all registered paravirt_probe functions
5. end up in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Question: how does the kernel know it's an lguest guest?

1. first code x86 kernel runs are in startup_32 in head.S
2. tests if paging is already enabled
3. if it is, we know we're under some kind of hypervisor
4. proceede to try all registered paravirt_probe functions
5. end of in drivers/lguest/lguest.c

The Guest

Guests know that they can't do priviledged operations such as disable interrupts

... they have to ask the host to do such things, via hypercalls

Lguest includes replacements for low-level native hardware ops

... this is accomplished by implementing the paravirt_ops interface

The Guest

Guests know that they can't do priviledged operations such as disable interrupts

... they have to ask the host to do such things, via hypercalls

Lguest includes replacements for low-level native hardware ops

... this is accomplished by implementing the paravirt_ops interface

The Guest

Guests know that they can't do priviledged operations such as disable interrupts

... they have to ask the host to do such things, via hypercalls

Lguest includes replacements for low-level native hardware ops

... this is accomplished by implementing the paravirt_ops interface

The Guest

Guests know that they can't do priviledged operations such as disable interrupts

... they have to ask the host to do such things, via hypercalls

Lguest includes replacements for low-level native hardware ops

... this is accomplished by implementing the paravirt_ops interface

The Guest

Guests know that they can't do priviledged operations such as disable interrupts

... they have to ask the host to do such things, via hypercalls

Lguest includes replacements for low-level native hardware ops

... this is accomplished by implementing the paravirt_ops interface

The Guest

Lguest supports basic console, block and net device drivers, also.

All devices are implemented within the virtio infrastructure, and aren't Lguest-specific.

The device bus on which these virtio devices lives is trivial.

It's just an array of device descriptors contained just above the top of normal memory.

The Guest

Lguest supports basic console, block and net device drivers, also.

All devices are implemented within the virtio infrastructure, and aren't Lguest-specific.

The device bus on which these virtio devices lives is trivial.

It's just an array of device descriptors contained just above the top of normal memory.

The Guest

Lguest supports basic console, block and net device drivers, also.

All devices are implemented within the virtio infrastructure, and aren't Lguest-specific.

The device bus on which these virtio devices lives is trivial.

It's just an array of device descriptors contained just above the top of normal memory.

The Guest

Lguest supports basic console, block and net device drivers, also.

All devices are implemented within the virtio infrastructure, and aren't Lguest-specific.

The device bus on which these virtio devices lives is trivial.

It's just an array of device descriptors contained just above the top of normal memory.

The Guest

Lguest supports basic console, block and net device drivers, also.

All devices are implemented within the virtio infrastructure, and aren't Lguest-specific.

The device bus on which these virtio devices lives is trivial.

It's just an array of device descriptors contained just above the top of normal memory.

The Guest

Lowest-level fo lguest I/O subsystem deserves a mention.

A "DMA" mechanism supports higher-layer facilities like virtio.

This layer really just copies memory between buffers.

The Guest

Lowest-level fo lguest I/O subsystem deserves a mention.

A "DMA" mechanism supports higher-layer facilities like virtio.

This layer really just copies memory between buffers.

The Guest

Lowest-level fo lguest I/O subsystem deserves a mention.

A "DMA" mechanism supports higher-layer facilities like virtio.

This layer really just copies memory between buffers.

The Guest

Lowest-level fo lguest I/O subsystem deserves a mention.

A "DMA" mechanism supports higher-layer facilities like virtio.

This layer really just copies memory between buffers.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Guest

A DMA buffer can be bound to a given address.

Any attempt to perform DMA to that address then copies memory into buffer.

DMA area can be in memory which is shared between guests, in which case data will be copied from one guest to another, and reciever will get an interrupt ...

incidentally, this is how inter-guest networking is implemented.

If no shared DMA area is found, DMA transfers are instead referred to hypervisor.

The Switcher/Host Module

Switcher itself is part of lg.ko module.

The code sites at 0xFFC00000 to do low-level guest-host switching.

It's as simple as can be, but very much tied to x86 architecture.

The Switcher/Host Module

Switcher itself is part of lg.ko module.

The code sites at 0xFFC00000 to do low-level guest-host switching.

It's as simple as can be, but very much tied to x86 architecture.

The Switcher/Host Module

Switcher itself is part of lg.ko module.

The code sites at 0xFFC00000 to do low-level guest-host switching.

It's as simple as can be, but very much tied to x86 architecture.

The Switcher/Host Module

Switcher itself is part of lg.ko module.

The code sites at 0xFFC00000 to do low-level guest-host switching.

It's as simple as can be, but very much tied to x86 architecture.

The Switcher/Host Module

Core of Lguest is 'lg' loadable module.

Module allocates chunk of memory and maps it in kernel address space.

Small hypervisor loaded into this area.

Switching involves fiddling with page tables and register contents.

The Switcher/Host Module

Core of Lguest is `lg' loadable module.

Module allocates chunk of memory and maps it in kernel address space.

Small hypervisor loaded into this area.

Switching involves fiddling with page tables and register contents.

The Switcher/Host Module

Core of Lguest is `lg' loadable module.

Module allocates chunk of memory and maps it in kernel address space.

Small hypervisor loaded into this area.

Switching involves fiddling with page tables and register contents.

The Switcher/Host Module

Core of Lguest is `lg' loadable module.

Module allocates chunk of memory and maps it in kernel address space.

Small hypervisor loaded into this area.

Switching involves fiddling with page tables and register contents.

The Switcher/Host Module

Core of Lguest is `lg' loadable module.

Module allocates chunk of memory and maps it in kernel address space.

Small hypervisor loaded into this area.

Switching involves fiddling with page tables and register contents.

Security Considerations

Hypervisor runs in ring-0, with guest domains running as host domain tasks in ring-1.

This differs from Xen when the hypvervisor is in ring-0 and host kernel in ring-1.

This is really a moot point.

Security Considerations

Hypervisor runs in ring-0, with guest domains running as host domain tasks in ring-1.

This differs from Xen when the hypvervisor is in ring-0 and host kernel in ring-1.

This is really a moot point.

Security Considerations

Hypervisor runs in ring-0, with guest domains running as host domain tasks in ring-1.

This differs from Xen when the hypvervisor is in ring-0 and host kernel in ring-1.

This is really a moot point.

Security Considerations

Hypervisor runs in ring-0, with guest domains running as host domain tasks in ring-1.

This differs from Xen when the hypvervisor is in ring-0 and host kernel in ring-1.

This is really a moot point.

Security Considerations

From isolation point of view, host kernel is effectively part of hypervisor, because the share same hardware priviledge level.

Also, the Host has so much privileged access to hypervisor; not really meaningful to run them in separate rings.

Security Considerations

From isolation point of view, host kernel is effectively part of hypervisor, because the share same hardware priviledge level.

Also, the Host has so much privileged access to hypervisor; not really meaningful to run them in separate rings.

Security Considerations

From isolation point of view, host kernel is effectively part of hypervisor, because the share same hardware priviledge level.

Also, the Host has so much privileged access to hypervisor; not really meaningful to run them in separate rings.

Security Considerations

Hypervisor is also present in guest systems' virtual address space.

Since guest kernel runs in ring-1, normal x86 page protection won't keep it from messing with hypervisor.

Instead, memory segmentation mechanism used to keep that code out of reach.

Security Considerations

Hypervisor is also present in guest systems' virtual address space.

Since guest kernel runs in ring-1, normal x86 page protection won't keep it from messing with hypervisor.

Instead, memory segmentation mechanism used to keep that code out of reach.

Security Considerations

Hypervisor is also present in guest systems' virtual address space.

Since guest kernel runs in ring-1, normal x86 page protection won't keep it from messing with hypervisor.

Instead, memory segmentation mechanism used to keep that code out of reach.

Security Considerations

Hypervisor is also present in guest systems' virtual address space.

Since guest kernel runs in ring-1, normal x86 page protection won't keep it from messing with hypervisor.

Instead, memory segmentation mechanism used to keep that code out of reach.

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

- When guest userspace process hits unmapped page, control switches to Host
- Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
- Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

- When guest userspace process hits unmapped page, control switches to Host
- Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
- Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

1. when guest userspace process hits unmapped page, control switches to Host
2. Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
3. Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

1. when guest userspace process hits unmapped page, control switches to Host
2. Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
3. Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

1. when guest userspace process hits unmapped page, control switches to Host
2. Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
3. Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

1. when guest userspace process hits unmapped page, control switches to Host
2. Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
3. Guest makes hypercall to set page table entry, then returns to userspace

Performance Issues

Lguest is slower than other hypervisors, though not always noticeably so.

Main place where performance suffers is in page faulting.

This is because faulting requires two round trips:

1. when guest userspace process hits unmapped page, control switches to Host
2. Host walks page tables, finds it's not mapped, switches back to Guest page fault handler
3. Guest makes hypercall to set page table entry, then returns to userspace