NAME

umlwatch − User interface for the UMLMON monitor

SYNOPSIS

umlwatch [*OPTIONS*] *vmname*[*@host*]

DESCRIPTION

This command allows the user to send commands to the UMLMON monitor daemon, and to retrieve status information from it. Effectively, umlwatch is a user frontend for the RPC interface controlling the daemon.

If no options are given, umlwatch enters an interactive mode where users can enter commands. These commands are the same that can be specified as options on the command line without the preceding hyphen, e.g. one can type **info** to get the same status page that is also printed when the **−info** option is given.

When options are given, these are interpreted as commands that are executed one after the other.

One can contact locally running monitors as well as monitors running on remote hosts when these hosts enable the TCP sockets. Use the "@" syntax to contact remote hosts.

Local connections are authorized by the file permissions of the corresponding Unix Domain socket. Either one is allowed to contact the daemon or not.

TCP connections are authenticated. In this case, umlwatch asks for a password. If the user knows the password, he is also considered as authorized.

DISK INTERFACE

A few words about the disk interface. The current version of the daemon enforces some restrictions where disk files can be accessed and how they must be named. Note that these restrictions are implemented by the daemon and not by umlwatch.

The disk files are looked up relative to the root of the chroot jail. The user only sees the file system relative to this root. There are only two directories where disks can be stored:

/disks: This directory is intended for disks owned by the VM (i.e. these disks can only be accessed by the current VM, and are out of reach by other VMs). One can create, delete, and manipulate disk files using the RPC interface only when they are in this directory. Files are usually owned by the user that runs the VM.

/shared/disks:
This directory is intended for disks shared by the VMs. All VMs can access the same files under this path name. Although it is possible to use these files for ubd devices directly when the file permissions permit it, one cannot create, delete, or modify them using the RPC interface. The host administrator should prepare these files, and should give them appropriate file permissions.

The types of the disk files are distinguished by the file name suffix. The extension .dsk means that the file is a flat image, i.e. an array of blocks. The extension .cow means that the file is a copy−on−write image. These images implicitly refer to a so−called backing file whose absolute name is stored in the header of the copy−on−write image. The extension .dev is used for block devices. Following the above philosophy, .dev files should only be created in /shared/disks. (Actually, one can set the **device** parameters of the global section of the /etc/umlmon configuration file to let the umlmon daemon create the block devices.)

Note that it might be difficult to import an existing copy−on−write image into UMLMON, because the absolute name of the backing file usually changes. However, once this problem is solved UMLMON’s naming scheme is quite useful, for example, it is no problem to copy a pair of backing file and copy−on−write image from one host to another, even if the directory location changes, because only the chroot−relative name counts, and this name does not change.

OPTIONS

−info: Outputs the status page. This includes the current configuration and the current run−time state. If the VM is running, the configuration is always the effective one that is actually used.

−tail *file*[,*n*]:
Outputs the tail of a log file. Use the name "monitor" to see the log file of the umlmon daemon. Use the name of a virtual console or serial line to see the logged output for the channel, e.g. "con0" to see the output of the UML kernel.

−connect (con*n*|ssl*n*):
Interactively connects with the virtual console or virtual serial line. You can disconnect with the special key CTRL−].

−force−connect (con*n*|ssl*n*):
Same as **−connect**, but it is possible to take over the session of another user.

−start: Starts the UML kernel with default arguments.

−start−with *arg*:
Starts the UML kernel, and passes *arg* as additional argument. This can be, for example, a runlevel specification, e.g. **single** to eneter single user mode.

−suspend:
Suspends the UML kernel.

−continue:
Resume a suspended UML kernel.

−halt: Immediately halts the UML kernel. This should only be used as last resort to stop the kernel running.

−ctrlaltdel:
Sends a CTRL−ALT−DEL event to the UML kernel. This is the recommended way to shut down the virtual system.

−kill: Immediately kills the UML kernel. This should only be used as last resort to stop the kernel running.

−sysrq *letter*:
Sends a system request to the UML kernel. The letter indicates the request. Use −sysrq h to see a list in the kernel log file (view with −tail con0).

−rotate:
Reopens all log files. This is necessary after log file rotation.

−wait *n*:
Waits until the UML kernel is halted, or until *n* seconds are over, what ever comes first.

−port: Outputs the TCP port number where the monitor can be contacted.

−set *param*=*value*:
Sets or adds a configuration parameter (user change). This is only possible for certain parameters, see umlmon(5) for a list. The change is in effect the next time the UML kernel is started.

−del *param*:
Deletes a configuration parameter (user change). This is only possible for certain parameters, see umlmon(5) for a list. The change is in effect the next time the UML kernel is started.

−disk−create *name*:*size*[:**sparse**]:
Creates a new virtual disk. The *name* must be given relative to the jail root. Actually, only names of the form /disks/*base*.dsk are accepted. The *size* is the size in megabytes. The **sparse** keyword arranges that the file is created sparsely, otherwise the blocks of the file are immediately allocated.

−disk−create−cow *name*:*bfile*:
Creates a new copy−on−write disk for a given backing file. Both *name* and *bfile* must be given relative to the jail root. Actually, only names of the form /disks/*base*.cow are accepted, and the backing file must be either /disks/*base*.dsk or /shared/disks/*base*.dsk. The copy−on−write disk has always the same size as the backing file, and it is always sparse.

−disk−delete *name*:
Deletes an existing disk. The *name* must be given relative to the jail root.

−disk−copy *src*:*dest*:
Copies a disk. The names given in *src* and *dest* must be given relative to the jail root. It is only possible to copy a .dsk file to a .dsk file, and a .cow file to a .cow file. The destination file must be in /disks.

−disk−resize *name*:*size*[:**sparse**]:
Resizes an existing disk of type .dsk. The *name* must be given relative to the jail root. In *size* pass the new size in megabytes. If the **sparse** keyword is given and the disk size grows, the new blocks are sparse, and not immediately allocated. This command changes only the file, but not the file system that might be stored in the file. When the disk grows, the file system must be resized after this command is executed. When the disk shrinks, the file system must already be resized before this command is executed.

−change−password:
Interactively change the password of the VM.

−site *arg*:
Executes the site script with *arg* as argument.

−version:
Outpus the version number and exits.

FILES

/var/lib/umlmon/*vmname*/ctrl:
This is the Unix Domain socket connecting to the RPC interface of the daemon for the VM *vmname*.

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlmon(5), umlmon(7), umlmon(8), umldir(8), umladmin(8)

NAME

umlmon − confi guration fi le of the UMLMON monitor

SYNOPSIS

/etc/umlmon

DESCRIPTION

UMLMON is a monitor that starts and communicates with virtual machines (VMs) basing on the User Mode Linux technology. This confi guration fi le sets some aspects common of all virtual machines, specifi es which virtual machines exist, and also sets the initial properties of the VMs. Note that many properties of the VMs can be changed by the user, and that these changes are stored in another fi le (see below). In this respect, /etc/umlmon only contains the default settings.

The confi guration fi le uses the well−known **ini** format. This fi le is structured into sections, and every section is started by a header in brackets, e.g.

[*section name*]

The sections contain parameter settings in the form

parameter name = *parameter value*

For every VM one needs a section. The name of the section is also the name of the VM. For example, this section describes the properties of the VM **foo**:

```
[foo]
vmuser = foo_user
logfi le = monitor
start = manual
kernel = /shared/kernels/linux
mem = 64
ubd0 = /disks/root.dsk
con0 = pty:con0
con1 = pty
```

In addition to this, there is also a global section covering the properties of the whole UMLMON system. This section has the reserved name **GLOBAL**.

THE GLOBAL SECTION

The global section may have the following parameters:

vardir = *directory*

Specifi es the location of the data directory where the VMs are stored. The directory must be an absolute path. Defaults to /var/lib/umlmon.

shared = *directory*

Specifi es the location of the shared data directory. This directory can be accessed from all VMs. The directory must be given by an absolute path. Defaults to **vardir**/shared.

tcp = *bool*

Sets whether the UMLMON monitors listen on TCP ports. Every monitor will listen on its own anonymous port which is announced in the UMLMON directory service UMLDIR. The boolean value must be either **true** or **false**. Defaults to false. Note that the monitors always listen on Unix Domain sockets. Note also that TCP connections are password−protected.

tcp_redir = *bool*

Sets whether the UMLMON directory service UMLDIR enables its TCP redirection service. This is another way of making the monitors accessible over the network that works independently of the **tcp** setting. UMLDIR listens on a single, known port, and forwards the connection to the monitor controlling the selected VM.

device = *device specification*

This directive makes a block device of the host system available for the VMs. This directive can be used several times to deal with several block devices. The device specification has the form

device_file(name=*name*,target_file=*file*,group=*group*,identity=*idaction*)

Here, the name is the name under which the VMs see the device. The name must have the suffix .dev. The monitor will create a block special file in the directory for shared disks with this name. The target file is the block special file identifying the device on the host. All users of the specified group can access the device read/write. Example:

device_file(name=data1.dev,target_file=/dev/sdc5,group=uml)

This directive makes the host device /dev/sdc5 available as shared disk data1.dev, and it is accessible by the users of the group "uml".

The optional parameter **identity** allows one to check whether the (already existing) device *name* and the target *file* still refer to the same device (devices have become quite dynamic in Linux). The *idaction* can be one of **ignore**, **check** (the default if omitted), or **update**. For **check**, the monitor refuses to start up when an identity change is detected. For **update**, the identity change is propagated from the target file to the local device node. This may be useful for e.g. LVM devices, dm devices, and USB devices.

pool = *pool specification*

Sets the properties of the pool of MAC addresses. UMLMON can manage a pool of such addresses, and every time a network interface is set up, it is possible to fetch a MAC address from the pool. Note that you must also configure the network interface such that it uses the pool. The pool specification has the following form:

mac_pool(first_mac_address=XX:XX:XX:XX:XX:XX,number=*n*)

Here, the X must be hex digits. The pool contains the address range starting with the explicitly mentioned address, and containing n consecutive addresses. If the pool parameter is omitted, the pool defaults to the range starting at 0A:12:BC:AF:32:00 and containing 256 addresses.

Note that MAC addresses are visible in the whole network segment. If you have several UMLMON servers in the segment and if you use pools, it is very advisable to assign every server a distinct MAC range.

password = *password*

Sets the master password that allows unlimited access to the monitors over TCP connections. By default, no such password exists. The password can be given in clear text, or as MD5 sum written as 32 hex digits. Note that connections over Unix Domain sockets are not affected by this setting.

sitecmd = *program*

The site command is typically a user−written script that can perform certain administrative actions that are specific to the site installation. The site command can be invoked over the RPC interface. The program must be given as absolute path to an executable.

VM SECTIONS

The sections describing the VMs can have the following parameters. A parameter is said to be protected if the user cannot change it, so only the administrator has the right to set it.

vmuser = *username*
Sets the user as which the VM will run. This parameter is required and protected.

vmgroup = *groupname*
Sets the main group as which the VM will run. The VM is always member of the groups the user is member of. The main group has a special meaning as new files are created with this group owner. By default, the main group is the main group of the user. This parameter is protected.

scheduler = *scheduler specification*
This directive may set the scheduling class and certain scheduling options of the VM process. By default, nothing special is set up regarding the scheduler. The specification may have one of these forms:

sched_default(prio=*n*)

Uses the standard scheduler with priority n. This is a number in the range from −20 to 19, smaller numbers mean higher priority. Priority 0 is the default process priority, so negative numbers give an increased and positive numbers a decreased priority compared to most other processes. Values −5 to −10 give already very noticeable boosts.

sched_rr(prio=*n*)

Uses the round−robin realtime scheduler with priority n. This is currently not recommended because the system may freeze if the VM eats up all available CPU time.

This parameter is protected.

jaildir = *directory*
Sets the directory where the chroot jail is set up. Defaults to **vardir**/*vmname*/jail. This parameter is protected.

logdir = *directory*
Sets the directory where the log files are created. Defaults to **vardir**/*vmname*/log. This parameter is protected.

logfile = *file*
Sets the log file for the UMLMON log. The file name should not contain a directory portion.

kernel = *file*
Sets the kernel executable. The file name must be given from the perspective of the chroot jail. It has usually the form /shared/kernels/*name*.

kernelarg = *arg*
Passes an additional arbitrary kernel argument.

mem = *n*
Sets how much memory the VM gets, given in megabytes. UMLMON automatically creates a tmpfs filesystem of this size and arranges that the memory backing file is created in it. Additionally, the kernel is informed about the memory size. This parameter is required.

mem_limit = *n*
Sets the administrative limit for the **mem** parameter. Users can change the amount of memory until the limit is reached. This parameter is protected.

con*n* = *terminal specification*

Specifies the console channel n. The UMLMON monitor can log messages sent to VM consoles, and it is even possible to set up bidirectional connections to consoles. Console con0 is always used for kernel messages and messages emitted when runlevels are changed. Consoles con1 and following are usually interactive consoles where users can log in. **It is strongly recommended to configure con0 and a small number of further consoles!**

The specification can have one of the forms:

pty

A pseudo terminal driver is used to manage the console connection.

pty:*logfile*

In addition to the pty driver, a logfile is written. The logfile should not contain a directory portion.

ssl*n* = *terminal specification*

Specifies the serial channel n (from the VM it can be accessed as /dev/ttyS*n*). The specification has the same format as for console channels.

ubd*n* = *file*

Sets the file used for the emulation of the virtual disk *n*. The file name is given from the perspective of the chroot jail, and must have the form /disks/*name* or /shared/disks/*name*. The first form refers to a file in the local directory of the VM, and the second form refers to a file in the shared directory used by all VMs. The name of the file must have a certain extension: .dsk for flat images, .cow for copy−on−write images, and .dev for block devices. Note that it is not necessary to specify the backing file of copy−on−write images as these are determined automatically.

ubd0 is usually the root disk of the VM.

sync_disks = *disk* ... | **all**

Determines which disks are forced to be opened in synchronous mode. For current UML kernels, this is recommended for disks containing sensitive data, and thus the default is **all**. However, you can also list here the ubd devices individually.

eth*n* = *NIC specification*

Configures the virtual NIC number n of the VM. The specification can be arbitrary text which is passed to the UML kernel, or one of the following special forms. This parameter is protected.

host_to_host(proxy_if=*nicname*,
 proxy_addr=*ipaddr*,
 proxy_hwaddr=*mac*,
 guest_addr=*ipaddr*,
 guest_hwaddr=*mac*,
 host_if=*nicname*)

Configures a host−to−host connection that works only for IP version 4, and that needs two IP addresses. The parameters proxy_hwaddr, guest_hwaddr and host_if are optional.

On the host side a proxy interface is established using the TUN/TAP driver. The name of the proxy_if can be chosen arbitrarily. The proxy interface needs its own IP address which becomes another IP address of the host system. The VM can reach the host under this address.

On the VM side the network interface must use the IP address announced here (guest_addr). A route is set such that the host can reach the VM under this address.

Optionally, one can set the MAC addresses of the proxy and the VM interfaces. Use either the format XX:XX:XX:XX:XX:XX where the Xs are hex digits, or the special keyword **pool**. The addresses are then taken from the MAC pool UMLMON manages.

Optionally, one can announce the guest IP address in another LAN segment (proxy ARP). This is typically the segment connecting the host system with the Internet. Use host_if with the name of the interface of this segment.

bridged(bridge_if=*nicname*,
 proxy_if=*nicname*,
 proxy_hwaddr=*mac*,
 guest_hwaddr=*mac*)

Configures bridged networking. It is necessary that the bridge interface already exists, i.e. create it with brctl before starting UMLMON. The rest of the setup is done automatically.

The bridge_if is the name of the bridge to connect to. The proxy_if is the name of the TUN/TAP interface the monitor creates. This can be an arbitrary name. The proxy interface is automatically added to the bridge.

In bridged networking, the VM can use an arbitrary IP address. Bridges are not restricted to IP version 4; all protocols available for Ethernet transport are possible. No routing is required to reach other hosts of the LAN segment.

Optionally, one can set the MAC addresses of the proxy and the VM interfaces. Use either the format XX:XX:XX:XX:XX:XX where the Xs are hex digits, or the special keyword **pool**. The addresses are then taken from the MAC pool UMLMON manages.

password = *password*

Sets the password of this VM that is used to authenticate TCP connections. The password can be given in clear text, or as MD5 sum written as 32 hex digits.

start = *type*

Sets the start type of the monitor and the VM: **manual** means that the monitor is started on boot but not the VM, **boot** means that the VM is booted, and **off** means that neither the monitor nor the VM is started.

USER CHANGES

The user can change unprotected parameters of the VM section. These changes are not stored in /etc/umlmon but in a second file **vardir**/*vmname*/config. The changes are usually performed by commands of the RPC interface.

In order to see the effective parameters of a VM, call the command

umladmin config *vmname*

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlwatch(1), umlmon(7), umlmon(8), umldir(8), umladmin(8)

NAME

umlmon − Concepts and Architecture

DESCRIPTION

UMLMON is an advanced monitor for User Mode Linux that controls the virtual machines (VMs). In this page we will give an overview over the UMLMON concept and explain its architecture.

If you read this text for the first time, it is recommended to read the section GETTING STARTED first (near the end).

OVERVIEW

UMLMON is a complete run−time environment for User Mode Linux. This environment implies a certain directory structure where disk files and other data file are stored. As the VMs are run in chroot jails it is quite important to understand this structure, because the VMs cannot access files outside the jail.

There is a monitor daemon for every VM. This daemon creates the run−time environment and starts the VM by executing the UML kernel. The daemon has an RPC interface over which commands can be sent (e.g. the command to start the VM). The daemon also determines certain kernel arguments that are passed to the UML kernel. Although one can simply pass any argument, the daemon has special support to set up arguments for memory size, virtual disks, virtual network interfaces, and console channels in a convenient way.

The monitor daemon includes also routines to do certain administration tasks like the creation of disks.

One of the superb parts of the monitor daemon is the ability to establish bidirectional connections to virtual consoles and virtual serial lines. This allows the user to have interactive sessions on the VMs.

The UMLMON directory service is another daemon that can be used to figure out which VMs are available on the system (or better, which monitor daemons are running).

DIRECTORY STRUCTURE

UMLMON defines a certain directory structure, so the VM can run in a chroot jail. Unlike other such jails, UML does not need many support files in it, so the jail has a very simple layout. The few needed files are automatically created by the monitor.

The basic structure is:

/var/lib/umlmon/*vmname*:
: This directory contains all data for the VM called *vmname*.

/var/lib/umlmon/*vmname*/jail:
: This is the chroot jail, i.e. the new root directory.

/var/lib/umlmon/shared:
: This is a directory for shared files. These files are accessible from the jails of all VMs.

/var/lib/umlmon/*vmname*/jail/shared:
: This is where the shared directory is mounted inside the jail.

The VM cannot escape the jail (unless it runs as privileged user). When the VM accesses files, this is done relative to the root directory of the jail.

Some directories have defined purposes, and should be used in this way. The following paths are relative to the root of the jail, because they must usually be given in this form, and not in the fully absolute form where the jail root is prepended.

/disks: This directory may contain disk images and block devices. The suffix of the file name determines the file type:
.dsk files are flat images, .cow files are copy−on−write images, and .dev files are block devices.

/tmp: This directory is actually a tmpfs filesystem used to represent virtual memory.

/shared/disks:
These are disks shared by several VMs. For example, one can place here backing files for copy−on−write disks that are shared by several VMs. To some extent, the monitor tracks whether the files are accessed sanely, and not corrupted by uncoordinated parallel accesses.

/shared/kernels:
This directory should contain the UML kernel executables.

PROCESS MODEL

The monitor daemon is a program with the name "umlmon". For every VM, a separate instance of umlmon is started. (This is done by the rc script.) Once started, umlmon puts itself into the background, and wait for commands on the RPC interface (see below). The umlmon daemon runs with full root privileges, so it can prepare the environment for the VMs (among other things, the daemon must be able to mount file systems, to create block devices, and to perform network setups).

When the VM is started, umlmon forks once, and the child process establishes the chroot environment and drops all the privileges. The parent and the child processes communicate over a private pipe. The child process starts the UML kernel, and watches its activity. Usually, the child process is also the controlling terminal for the UML kernel, so it is possible to kill UML by killing the child.

By some trickery with file descriptors, the UML kernel gets also access to the pseudo terminals it needs for the console and serial channels.

THE RPC INTERFACE

The umlmon daemon has an RPC interface that can be used to control it. This is classic SunRPC. Note, however, that the portmapper is not used, and that a private directory service ("umldir") is established.

There is a Unix Domain socket over which one can send RPC requests:

/var/lib/umlmon/*vmname*/ctrl

By default, any user can connect to this socket. The protocol itself does not enforce authentication when used over this socket. The only way to restrict access is to set the privileges of the parent directory.

It is also possible to enable TCP for the RPC interface. An anonymous port is used. Unlike the Unix Domain socket, the TCP socket has built−in authentication. As the standard SunRPC mechanisms are either insecure or overly complex, a simple HMAC−based authentication scheme has been added. The first RPC call must perform authentication, or the connection is immediately closed.

One can set a separate password for every VM, and a master password that enables access to all VMs. (The latter may be useful for the Web interface.) These passwords have nothing to do with system accounts, and can be simply set in the configuration file.

In order to find out the TCP port of the RPC interface of a certain VM, one can query the directory service. This service is provided by the daemon umldir. Actually, umldir provides a second type of RPC interface (with very few procedures). The key procedure is get_port allowing to find out the TCP port by VM name. Unlike umlmon, umldir listens on a defined TCP port. The port must be defined in /etc/services for the name "umldir".

Because this port scheme is sometimes unhandy, umldir provides a second, multiplexing TCP socket. This is the so−called TCP redirector. This socket has also a defined name ("umlredir" in /etc/services). When connecting to it, one can announce to which VM one would like to talk, and umldir arranges that all further data over this connection are forwarded to the right monitor daemon. This way, one needs only two defined TCP ports to control all VMs on the system. This scheme is advantegous when there are firewalls or tunnels between client and server.

In order to send commands to the RPC interface, one can use the command−line tool umlwatch. It is called as

umlwatch *vmname*[*@host*]

Without "@" or with "@localhost", umlwatch uses the Unix Domain socket to establish the control connection. Otherwise, it connects to the "umldir" port on the given host, and finds out how to create the control connection.

The RPC interface is fully documented in the file umlmonctrl.x (with classic RPC IDL syntax).

VIRTUAL MEMORY

The umlmon daemon prepares the allocation of virtual memory by creating a tmpfs filesystem of the appropriate size. The filesystem is mounted at the directory

/var/lib/umlmon/*vmname*/jail/tmp

just before the UML kernel is executed. The kernel will create the backing file for its VM memory in this directory. (This file is immediately deleted, so you cannot see it.) Furthermore, this directory also contains the "mconsole socket" over which certain control commands can be sent to the running kernel.

VIRTUAL CONSOLES

The umlmon daemon can establish channels that connect the virtual consoles and virtual serial lines with the outer world. The RPC interface provides procedures allowing to read data from these channels or write data into them. The umlwatch program is a simple front−end for these procedures, so one can interactively connect with the consoles, and have a user session. The data flow is usually:

termemu <−[pty]−> umlwatch <−[RPC]−> umlmon <−[pty]−> UML kernel

The termemu on the left side is where the user currently is logged in on the host system (e.g. an xterm). umlwatch (with −connect argument) passes all interactive I/O over the RPC interface to umlmon, which finally drives the consoles or serial lines in the UML kernel.

The umlmon daemon allocates pseudo terminals (pty) to communicate with the running VM kernel. A pseudo terminal is a kernel channel with two endpoints, the master device and the slave device. The file descriptor accessing the slave device is passed to the UML kernel, and the master device can be opened by umlmon itself. When one calls the mentioned RPC procedures, the master device is opened, and the data exchange is performed.

It is possible to log all data flowing from the VM to the host.

In order to support the web interface, the umlmon daemon even contains its own implementation of a terminal emulator. The command set of a normal Linux console is supported.

VIRTUAL DISKS

The UML kernel supports two kinds of virtual disks: Flat images and copy−on−write images. Flat images can be regular files containing the blocks of the disk to emulate in ascending order, or they can be real partitions of real disks. Copy−on−write images are files that are backed by flat images, and only the differences to the backing file are stored.

In order to simplify the handling of these formats, UMLMON implements a number of features. First, the umlmon daemon has administrative commands to create, delete, copy and resize images. These commands can be executed over the RPC interface (e.g. use umlwatch to perform them). Second, it is defined in which directories the disk files may be contained (see above). Third, the disk files have defined extensions: the suffix .dsk is used for flat images in regular files, .dev is used for block devices, and .cow is used for copy−on−write images. Fourth, the daemon determines the kernel arguments.

The daemon tracks which disk files are open for reading and which are open for writing. Furthermore, the daemon knows which backing files and which COW files refer to each other. The daemon prevents operations that would destroy data, e.g. because the same image is open r/w by two VMs.

VIRTUAL NETWORKS

The VM can create virtual network interfaces. It is, however, quite difficult to connect these interface with the real network. UMLMON tries to simplify this by providing special support for two kinds of setup.

The first setup is "host to host" networking. On the host system, a proxy network interface is created that has a point−to−point connection with the virtual interface existing within the VM. Every packet sent from the proxy interface appears at the virtual interface and vice versa. Both interfaces need IP addresses. This setup is simply configured with a single line:

```
eth0 = host_to_host(proxy_if=proxy1, proxy_addr=192.168.2.1, guest_addr=192.168.2.2)
```

The monitor performs the complete setup at the host side. The user only has to configure networking within the VM in the right way (by setting the IP address and the gateway).

Optionally, one can announce the guest address in the real network (by ARP proxying, just add host_if=eth0 to this configuration).

The disadvantage of this setup is that two addresses are needed and that the connection is restricted to IP version 4. A more powerful setup is "bridged" networking.

Here, the host administrator must already have set up a bridge interface. The bridge can remain a purely virtual LAN segment, or it can be connected with the real network. This part of the setup is outside of the scope of UMLMON. The daemon includes the necessary knowledge to attach a proxy interface to the bridge (which is again magically connected with the virtual interface within the VM). The configuration looks like:

```
eth0 = bridged(bridge_if=br0, proxy_if=proxy1)
```

For both setups the monitor manages a pool of MAC addresses. One problem of existing UML is that the MAC addresses of the proxy interfaces are randomly selected, and that the MAC addresses of the virtual interfaces are derived from the IP addresses − which might not work for some configurations. The pool contains a range of reserved MAC addresses, and when the mentioned interfaces are configured, the MAC addresses may be fetched from the pool.

GETTING STARTED

User Mode Linux (UML) is a special port of the Linux kernel that does not run directly on a certain hardware platform but as a user−space process on a host system (which must be, at the time of this writing, also a Linux system). Of course, you need a UML kernel binary which is normally a statically linked executable called **linux**.

Furthermore, you need a disk image, i.e. the file system that will be used for the virtual system. This should be a flat image (i.e. a sequence of disk blocks) without partition table and MBR. The image has the right format when you can loop−mount it. In the following, we assume that the image has the file name **root.dsk** (note that UMLMON needs certain file name suffixes like .dsk to identify file types). You can use any type of filesystem (ext2, ext3, reiserfs) as long as the UML kernel binary supports it.

The disk image must have a few special properties:

There should be block devices for the ubd driver in the /dev directory. You can create them using

```
for n in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15; do
  mknod <mnt>/dev/ubd$n b 98 $((n*16));
done
```

Here, <mnt> is the path where the disk image is mounted.

You may need to adjust <mnt>/etc/fstab to use these block devices.

The CTRL−ALT−DEL event should halt the system instead of rebooting it. Edit in <mnt>/etc/inittab the line where the string "ctrlaltdel" occurs. It should look like

```
ca:12345:ctrlaltdel:/sbin/shutdown −t1 −a −h now
```

The point is that the −h switch will halt the system.

If the system on disk image uses TLS and NPTL, you should disable that, because UML does not support these features yet (when this text was written). Often, this can be achieved my moving the library <mnt>/lib/tls away,e.g.

```
mv <mnt>/lib/tls <mnt>/lib/tls.disabled
```

This assumes, however, that the system has another version of the multi−threading library in <mnt>/lib without TLS support (i.e. "LinuxThreads"). This is the case for all major Linux distributions.

Now you need a configuration file for UMLMON. Create the file /etc/umlmon with the contents:

```
[GLOBAL]

[vm1]
vmuser = nobody
logfile = monitor
start = manual
kernel = /shared/kernels/linux
mem = 64
ubd0 = /disks/root.dsk
con0 = pty:con0
con1 = pty
con2 = pty
con3 = pty
con4 = pty
```

"vm1" is the name of the virtual machine. As vmuser you can use any valid user of the host system, instead of "nobody" this could be simply your personal user account, or a special user created for VMs. Do not use "root", however, because this is insecure.

You may wonder why there are the strange paths /shared/kernels/linux and /disks/root.dsk that do not exist. Well, they do not yet exist, but we arrange that now.

Now we start UMLMON. There is a script in /etc/init.d that performs this:

/etc/init.d/umlmon start

You get a message that a service called umldir is started. This is the directory service, i.e. a special daemon that tracks which VMs are defined on the host and that can be used to connect to these VMs over the network.

Furthermore, this script says it has started a monitor for vm1. You can see this monitor in the process listing (ps −ef) as "umlmon" process. There is one such process for every inactive VM, and two such processes for every running VM.

The VM will run in a chroot jail. The monitor prepared a directory for this purpose: /var/lib/umlmon/vm1/jail. The strange paths you observed above are interpreted relative to this directory, i.e. /disks/root.dsk is /var/lib/umlmon/vm1/jail/disks/root.dsk in reality.

Put now the files root.dsk and linux in its places:

```
mv root.dsk /var/lib/umlmon/vm1/jail/disks/root.dsk
mv linux /var/lib/umlmon/vm1/jail/shared/kernels/linux
```

If you are a very good observer, you will notice that /var/lib/umlmon/vm1/jail/shared is physically the same directory as /var/lib/umlmon/shared. This is a kind of symbolic link that isn't a link (it is a so−called bind mount). The "shared" directory is linked into the chroot jails of all virtual machines, so files common of all VMs can be placed there. The UML kernel is an example.

Now you can start the virtual machine. The program umlwatch connects to the monitor and passes commands to it:

```
umlwatch vm1 −info
```

This outputs the current configuration, and the current runtime state of the VM.

```
umlwatch vm1 −start
```

The output of the UML kernel is logged in /var/lib/umlmon/vm1/log/con0. If this looks good, you can login:

```
umlwatch vm1 −connect con1
```

This connects to the first console, and you can start an interactive session in the VM. Use the key CTRL−] to suspend the session.

Note that it is not necessary to configure networking in order to have a login session on the VM.

To shut down the VM send the command

```
umlwatch vm1 −ctrlaltdel
```

This simulates a CTRL−ALT−DEL event. As the VM should be configured such that CTRL−ALT−DEL shuts the system down (see above), this command will initiate the shutdown.

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlwatch(1), umlmon(5), umlmon(8), umldir(8), umladmin(8)

NAME

umladmin − Administrative interface to UMLMON

SYNOPSIS

umladmin [*OPTIONS*] (**list**|**start**|**stop**|**update**|**password**|**config**) [*vmname ...*]

DESCRIPTION

This command provides a collection of administrative interfaces to UMLMON. In particular, one can list the configured virtual machines (VMs), one can start and stop them, and one can do some manipulations of the configuration file.

The first non−optional keyword selects the mode of operation:

list: The VMs configured in /etc/umlmon are listed together with the current runtime status. Note that VMs whose monitor is running but that have been removed from /etc/umlmon are not shown.

If names of VMs are passed on the command line, the statuses of these VMs are determined and printed.

start: The VMs configured in /etc/umlmon are started, if possible. In particular, it is checked how far the VM should be started up by inspecting the **start** parameter of the configuration file. If this parameter is **off**, the VM is not started at all. If the parameter is **manual**, the umlmon daemon is started, but the virtual system is not booted. If the parameter is **boot**, the daemon is started, and the UML kernel is booted.

If there is already a running daemon, its state is not changed at all, i.e. this command affects only VMs that are completely down.

Note that the boot process is performed in the background, i.e. the command does not wait until the started systems are ready.

If names of VMs are passed on the command line, these VMs are started instead of the configured ones.

stop: The VMs configured in /etc/umlmon are shut down. If there is a running UML kernel, it is tried to shut it down properly by sending a CTRL−ALT−DEL event. If the kernel is not down after a certain period of time (300 seconds by default), more drastic measures are taken. In particular, it is then tried to sync and umount the virtual disks, before the kernel is immediately halted. As a last resort, the UML kernel is killed.

After the kernel is shut down, the umlmon monitor is terminated, too.

The command initiates the shutdown of all affected VMs in parallel, and waits until all VMs are down.

If names of VMs are passed on the command line, these VMs are shut down instead of the configured ones.

update:

After VMs have been added to the configuration file or have been removed from it, this command can be used to synchronize the run−time state with the configuration. In particular, newly added VMs are started (as for the **start** command), and removed VMs are shut down (as for the **stop** command). VMs whose **start** parameter is set to **off** are shut down, too.

password:

Reads a password from stdin, and prints the corresponding MD5 hash on stdout.

config: The name of a single VM must be passed on the command line. The effective configuration of this VM is printed to stdout, i.e. the configuration after merging /etc/umlmon with the user−supplied changes of the configuration.

There are options to modify the printed configuration. These options do not modify the stored configuration, however.

OPTIONS

−active:
The *list* command lists only VMs with running UML kernel.

−inactive:
The *list* command lists only VMs with inactive UML kernel and running umlmon daemon.

−off: The *list* command lists only VMs without umlmon daemon.

−timeout *n*:
Changes the timeout for the shutdown of a virtual system from 300 seconds to *n* seconds. This options has only an effect for the **stop** and **update** commands.

−clear *param*:
The **config** command deletes this parameter from the printed configuration.

−set *param=value*
The **config** command sets this parameter in the printed configuration.

−rename *vmname*:
The **config** command renames the VM in the printed configuration to *vmname*.

−version:
Outputs the version number and exits.

FILES

These are the default locations of the files. One can change the locations in /etc/umlmon.

/etc/umlmon:
The main configuration file, see umlmon(5) for a description.

/var/run/umlmon/*vmname*.pid:
The process ID of the umlmon daemon for *vmname*.

/var/lib/umlmon/*vmname*/ctrl:
This is the Unix Domain socket connecting to the RPC interface of the umlmon daemon for *vmname*.

/var/lib/umlmon/*vmname*/config:
These are the user changes of the configuration for the VM *vmname*. These settings override the settings in /etc/umlmon.

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlwatch(1), umlmon(5), umlmon(7), umlmon (8), umldir(8)

NAME

umldir − UMLMON directory service daemon

SYNOPSIS

umlmon [*OPTION*]

DESCRIPTION

The directory service provides an interface to find out which umlmon daemons are started on the system, and to get the TCP ports where the RPC interface can be contacted (if enabled).

The directory service is required when the TCP sockets are enabled and when the web interface is used.

The directory service can be contacted over a local Unix Domain socket or over a TCP socket (when enabled). It provides its own RPC interface. Queries are always unauthenticated.

OPTIONS

−fg The daemon remains in the foreground. (This may be useful to debug problems.)

−version
Outputs the version number and exits.

FILES

These are the default locations of the files. One can change the locations in /etc/umlmon.

/etc/umlmon:
The main configuration file, see umlmon(5) for a description. The directory service processes only the global section of the file.

/var/run/umlmon/directory.pid:
The process ID of the daemon.

/var/lib/umlmon/directory:
This directory hierarchy contains the data and runtime files of the directory service.

/var/lib/umlmon/directory/ctrl:
This is the Unix Domain socket connecting to the RPC interface of the daemon.

SIGNALS

SIGTERM
Shuts the daemon regularly down.

SIGINT
If running in the foreground, this signal terminates the daemon process immediately.

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlwatch(1), umlmon(5), umlmon(7), umlmon (8), umladmin(8)

NAME

umlmon − Monitor daemon for User Mode Linux

SYNOPSIS

umlmon [*OPTION*] *vmname*

DESCRIPTION

The umlmon daemon controls the environment and the execution of a single User Mode Linux (UML) kernel.

The virtual machine must be configured in the configuration file(s). After execution, the umlmon daemon installs the RPC interface, puts itself into the background, and wait for commands arriving on the RPC interface. It does not start the UML kernel automatically.

OPTIONS

−fg The daemon remains in the foreground. (This may be useful to debug problems.)

−version
Outputs the version number and exits.

FILES

These are the default locations of the files. One can change the locations in /etc/umlmon.

/etc/umlmon:
The main configuration file, see umlmon(5) for a description.

/var/run/umlmon/*vmname*.pid:
The process ID of the daemon.

/var/lib/umlmon/*vmname*:
This directory hierarchy contains the data and runtime files of the virtual machine.

/var/lib/umlmon/*vmname*/ctrl:
This is the Unix Domain socket connecting to the RPC interface of the daemon.

/var/lib/umlmon/*vmname*/config:
These are the user changes of the configuration. These settings override the settings in /etc/umlmon.

/var/lib/umlmon/*vmname*/jail:
This directory is prepared as chroot jail for the UML kernel.

/var/lib/umlmon/shared:
This directory is the shared part of all UML jails. It contains several administrative files (locks and macpool) that must not be changed by the user.

MOUNT POINTS

The daemon mounts a number of file systems:

/var/lib/umlmon/*vmname*/jail/shared:
This is a bind mount of /var/lib/umlmon/shared. This mount is performed immediately after the daemon starts.

/var/lib/umlmon/*vmname*/jail/tmp:
This is a tmpfs filesystem of the appropriate size that will contain the backing file for the virtual memory of the UML kernel. This file system is created just before the UML kernel is executed.

/var/lib/umlmon/*vmname*/jail/proc/mm:
If the host system supports SKAS3, this file is a bind mount of /proc/mm so the UML kernel can access the mm interface. This mount is established just before the UML kernel is executed.

SIGNALS

SIGTERM
: Shuts the daemon regularly down. If the UML kernel is running, it is usually killed (because the controlling terminal vanishes).

SIGINT
: If running in the foreground, this signal terminates the daemon process immediately.

AUTHOR

UMLMON was written by Gerd Stolpmann.

REPORTING BUGS

Report bugs to gerd@gerd−stolpmann.de

GETTING SUPPORT

You can get commercial support for UMLMON. Please ask Gerd Stolpmann <gerd@gerd−stolpmann.de>.

COPYRIGHT

SEE ALSO

umlwatch(1), umlmon(5), umlmon(7), umldir(8), umladmin(8)