Stefano Busnelli (IlGuru)

USER MODE LINUX

5 June 2018

UML è un sistema di virtualizzazione di linux comparso come patch del kernel dalle versioni 2.2 del sistema operativo, che è stato poi integrato nello sviluppo del kernel di linux dalla versione 2.6. Permette di eseguire un sistema operativo completo in user space come una macchina virtuale a cui assegnare varie risorse (drive, schede di rete, console, ecc...) ed è utile per virtualizzare macchine linux complete oppure per testare diverse versioni di kernel, driver, programmi ecc... all'interno di un sistema isolato che non può intaccare l'host che esegue uml. E' possibile eseguire varie macchine uml contemporaneamente simulando varie topologie di rete, implementare honetpots, testare la sicurezza di reti ecc... Il sistema da eseguire è tutto contenuto in un root file system che viene montato da uml alla partenza. Può essere contenuto in un uno o più file separati il cui contenuto può essere popolato e modificato tramite script prima dell'esecuzione del binario uml in modo da automatizzare la pubblicazione di macchine virtuali. Insomma le possibilità sono davvero numerevoli.

Nota: Non funziona su raspberry, perchè il processore arm non è dotato delle caratteristiche hardware necessarie per la virtualizzazione, funziona di sicuro su architetture x86 e x86_64.

1 - Installazione

1.1 - Via semplice

Installiamo il pacchetto uml della distribuzione (debian o ubuntu):

```
# apt install user-mode-linux
```

L'eseguibile del kernel viene installato in /usr/bin/linux

1.2 - Via meno semplice

Abbiamo bisogno di alcuni strumenti per la compilazione del kernel che se non presenti vanno installati:

- gcc
- make
- bc

Per prima cosa scarichiamo da www.kernel.org l'archivio contenente la versione del kernel in esecuzione sull'host:

```
# uname -a
Linux debian-00 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64 GNU/Lin
# wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.9.tar.gz
# tar -xzf linux-4.9.tar.gz
# cd linux-4.9
```

A meno di issue note, è possibile eseguire qualunque kernel guest all'interno di un kernel host, ma per non incorrere inutilmente in problemi si fa prima ad utilizzare la stessa versione.

Ora configuriamo e compiliamo il kernel. E' indispensabile specificare sempre l'architettura del kernel che compiliamo inserendo ARCH=um

```
# make defconfig ARCH=um
```

Possiamo modificare le impostazioni di default del kernel, ma non è necessario. In alternativa a config si può usare menuconfig per una configurazione tramite menù in console oppure xconfig se siamo in una console grafica.

```
# make config ARCH=um
```

Eseguiamo la compilazione.

```
# make all ARCH=um
```

Terminata la compilazione abbiamo a disposizione l'eseguibile ./linux che è il nostro kernel da eseguire da riga di comando.

```
# cp linux /usr/bin/linux
# chmod 755 /usr/bin/linux
# linux --version

4.9.0
```

2 - Installazione delle utilities

Sia che abbiamo installato l'eseguibile uml, sia che ce lo siamo compilato, ci serviranno le uml utilities, in insieme di programmi di contorno che vengono richiesti per l'esecuzione delle virtual machines.

2.1 - Pacchetto deb precompilato

```
$ sudo apt install uml-utilities
```

2.2 - Compilazione da codice sorgente

```
$ wget http://user-mode-linux.sourceforge.net/uml_utilities_20070815.tar.bz2
$ tar -xjvf uml_utilities_20070815.tar.bz2
$ cd tools-20070815/
$ make
$ sudo make install
```

2.3 - Note

Potrebbe esserci un problema con il pacchetto uml_utilities per cui alla fine del boot di una VM non viene trovato il programma port-helper compromettendo l'avvio degli xterms.

port-helper è presente in /usr/lib64/uml ma questa directory non è presente nella variabile PATH. Per ovviare al problema si può aggiungere la directory nella variabile PATH per tutti gli utenti:

```
$ echo "export PATH=\$PATH:/usr/lib64/uml" > /etc/profile.d/uml-path.sh
$ source /etc/profile
```

In alternativa si può creare un link simbolico che punta a quel file in /usr/local/bin:

```
# ln -s /usr/lib64/uml/port-helper /usr/local/bin/port-helper
```

3 - Creazione di un root filesystem

Da qui possiamo eseguire uml come un normale utente, per la creazione del root filesystem utilizzeremo un file vuoto che popoleremo con debootstrap che va installato:

```
$ sudo apt install debootstrap
```

Il root filesystem per un sistema debian minimale occupa qualche centinaio di mega, in questo esempio ne creeremo uno da 300M:

```
$ cd ~
$ mkdir uml
$ cd uml
$ dd if=/dev/zero of=uml-00-root bs=1024K count=300
$ loop0=$(sudo losetup -f --show uml-00-root)
/dev/loop0
$ sudo mkfs.ext4 $loop0
$ mkdir umlroot
$ sudo mount $loop0 umlroot/
$ sudo debootstrap stable umlroot/ http://deb.debian.org/debian/
```

Non appena debootstrap termina di installare i pacchetti della debian stable, nella cartella ./umlroot potremo vedere un sistema linux completo.

```
$ du -sh umlroot/
252M    umlroot/
```

Per prima cosa dobbiamo impostare la password di root di questo sistema, cosa che può essere fatta facilmente utilizzando passwd montando la cartella umlroot in charoot:

```
$ sudo chroot umlroot/ /bin/bash
# passwd
```

Ora dobbiamo impostare il nome della macchina uml, ad esempio uml-00:

```
# echo "uml-00" > /etc/hostname
```

Impostiamo anche fstab in modo da non avere / montato read-only

```
# echo "/dev/ubda / ext4 defaults,noatime 0 1" >> /etc/fstab
```

Timezone, locales ...

```
# apt install locales
# ln -fs /usr/share/zoneinfo/Europe/Rome /etc/localtime
# dpkg-reconfigure locales
```

Usciamo da chroot e smontiamo il root filesistem:

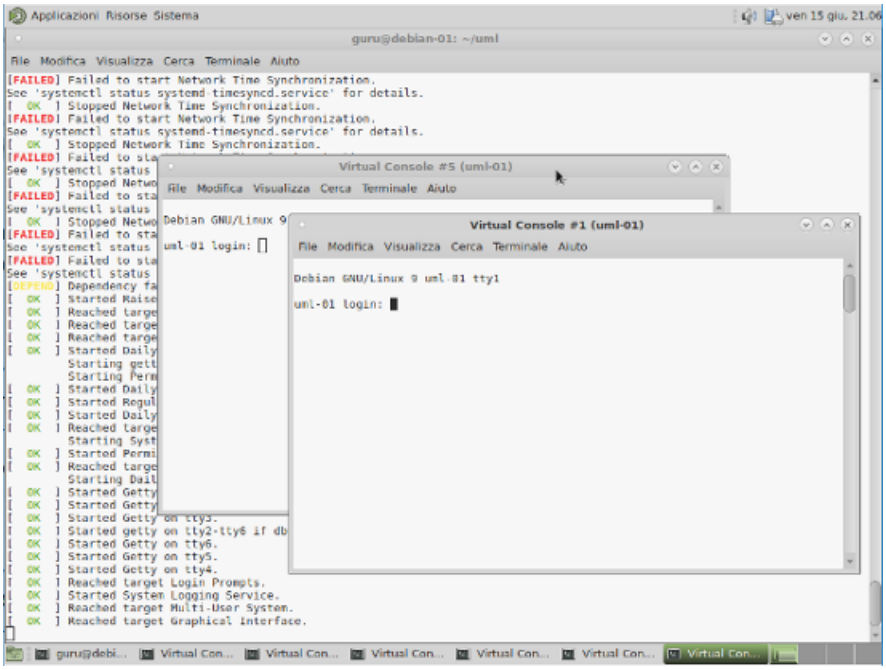
```
# exit
$ sudo umount $loop0
$ sudo losetup -d $loop0
```

4 - Esecuzione del sistema uml

4.1 - Avvio di una VM in ambiente grafico

Se ci troviamo in un ambiente grafico possiamo eseguire la VM senza specificare nessuna console ed alla fine del processo di boot verranno aperti alcuni xterm collegati alle virtual console tty mostrate nel log di boot.

```
$/usr/bin/linux ubda=uml-00-root mem=512M
```



4.2 - Avvio di una VM in console

Se non siamo in un ambiente grafico invece, dovremo specificare su quali console, linee seriali ecc vogliamo collegarci per fare login. Eseguiamo il sistema appena configurato assegnandogli 512M di memoria ram e mappando tutte le console con gli pseudo terminali dell' host:

```
$/usr/bin/linux ubda=uml-00-root mem=512M con0=fd:0,fd:1 con=pts
```

5 - Login

Normalmente se siamo in un ambiente grafico ogni virtual console viene presentata in un xterm. E' possibile configurare delle linee seriali, delle console oppure configurare interfacce di rete per accedere alla macchina uml. In questo esempio abbiamo avviato la VM da una semplice console configurando delle interfacce con che vengono mappate con i dispositivi /dev/pts/* disponibili nell' host.

Appena la macchina uml finisce di fare il boot, gli ultimi messaggi ci comunicano come vengono mappati i device pts:

```
Virtual console 6 assigned device '/dev/pts/2'
Virtual console 4 assigned device '/dev/pts/3'
Virtual console 1 assigned device '/dev/pts/4'
Virtual console 5 assigned device '/dev/pts/5'
Virtual console 3 assigned device '/dev/pts/6'
Virtual console 2 assigned device '/dev/pts/7'
```

Aprendo un'altra console nell' host possiamo accedere agli pseudo terminali tramite screen, minicom ecc:

```
$ screen /dev/pts/4
```

oppure

```
$ minicom -p /dev/pts/4
```

```
Debian GNU/Linux 9 uml-00 tty1
uml-00 login:
```

Possiamo effettuare il login e finire di configurare la macchina uml, tutte le modifiche rimarranno salvate nel file uml-00-root

6 - Utilizzare più virtual block devices

Le partizioni vengono viste all'interno di uml come device chiamati /dev/udb* dove il primo è /dev/udba. Quando eseguiamo uml montiamo il root filesystem su questo block device tramite questa opzione:

```
$ linux ... udba=./root_file_system ...
```

Avendo a disposizione altri file possiamo far si che il sistema li veda in altre partizioni (ubdb, ubdc, ...) e montarle correttamente all'avvio inserendo le voci corrispondenti nel file /etc/fstab.

Costruiamo ad esempio un sistema con due file, uml-01-root da 256 M che conterrà / ed uml-01-var da 128 M che conterrà la cartella /var Li monteremo nella cartella ./umlroot prima di lanciare debootstrap in modo che questo inserisca tutti i file nella cartella /var del rootfilesystem, dopodichè editeremo il file /etc/fstab. Molti passaggi sono identici all'esempio precedente:

```
$ dd if=/dev/zero of=uml-01-root bs=1024K count=256
$ dd if=/dev/zero of=uml-01-var bs=1024K count=128
$ sudo mkfs.ext4 ./uml-01-root
$ sudo mkfs.ext4 ./uml-01-var
$ loop0=$(sudo losetup -f --show ./uml-01-root)
/dev/loop0
$ loop1=$(sudo losetup -f --show ./uml-01-var)
/dev/loop1
$ sudo mount $loop0 ./umlroot/
$ sudo mkdir ./umlroot/var
$ sudo mount $loop1 ./umlroot/var/
$ sudo debootstrap stable ./umlroot/ http://deb.debian.org/debian/
$ sudo chroot umlroot/ /bin/bash
# passwd
```

```
# echo "uml-01" > /etc/hostname
# ln -fs /usr/share/zoneinfo/Europe/Rome /etc/localtime
# dpkg-reconfigure locales
```

All' interno di chroot dopo aver impostato password ed hostname, definiamo come montare le due partizioni inserendo le due righe in /etc/fstab

```
# echo "/dev/ubda /    ext4 defaults,noatime 0 1" >> /etc/fstab
# echo "/dev/ubdb /var ext4 defaults,noatime 0 1" >> /etc/fstab
```

Smontiamo i due file

```
$ sudo umount $loop1
$ sudo umount $loop0
$ sudo losetup -D
```

```
$ linux ubda=uml-01-root ubdb=uml-01-var mem=512M con0=fd:0,fd:1 con=pts
```

Eseguiamo il login come nell'esempio precedente:

```
# mount
/dev/ubda on / type ext4 (rw,noatime,data=ordered)
.
.
.
/dev/ubdb on /var type ext4 (rw,noatime,data=ordered)
```

7 - Networking

UML mette a disposizione diversi tipi di transport con cui configurare un'interfaccia di rete:

- ethertap
- tuntap
- slip
- slirp
- pcap
- daemon
- multicast

tuntap, ethertap, slip e slirp si usano per realizzare scampi di pacchetti ip tra la VM e l'host, che può essere configurato come un router per ridirezionarli verso l'esterno o verso altre VM. pcap è un transport di solo ascolto, è utile per realizzare sniffer e analizzatori di traffico. multicast e daemon sono utili per realizzare reti virtuali tra VM, la rete rimane completamente scollegata dalle altre interfacce a meno che una delle VM faccia da gateway per la rete.

Nel sistema host impostiamo un'interfaccia virtuale tuntap per avere un collegamento point-to-point tra l'host e la virtual machine assegnandole un indirizzo:

```
# ip tuntap add tap0 mode tap
# ifconfig tap0 10.0.0.254 up
# ip addr show
6: tap0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group
    link/ether de:af:40:72:07:9e brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.254/8 brd 10.255.255.255 scope global tap0
       valid_lft forever preferred_lft forever
```

Abilitiamo anche l'inoltro dei pacchetti tra questa interfaccia (tap0) e quella collegata ad internet (nel mio host enp0s3) altrimenti la comunicazione sarebbe limitata unicamente alla rete formata da host e macchina virtuale:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
# iptables -A FORWARD -i tap0 -o enp0s3 -j ACCEPT
# iptables -A FORWARD -i enp0s3 -o tap0 \
           -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Avviamo la macchina virtuale impostando l'interfaccia eth0 impostando passando i paramentri tuntap come transport, tap0 e l'indirizzo di tap0:

```
$ linux ubda=uml-00-root \
        mem=512M \
        con0=fd:0,fd:1 con=pts \
        eth0=tuntap,tap0,10.0.0.254
```

Una volta avviata la vm ed effettuato il login possiamo configurare l'interfaccia eth0. Verifichiamo innanzi tutto la presenza del device:

```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether c6:71:91:58:37:2e brd ff:ff:ff:ff:ff:ff
```

Ora possiamo configurare eth0 assegnando indirizzo e routing table:

```
# ip addr add 10.0.0.1 dev eth0
# ip link set eth0 up
```

```
# ip route add 10.0.0.0/8 dev eth0
# ip route add default via 10.0.0.254 dev eth0
```

Verifica:

```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN grou
    link/ether c6:71:91:58:37:2e brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/8 brd 10.255.255.255 scope global eth0
       valid_lft forever preferred_lft forever

# ip route list
default via 10.0.0.254 dev eth0
10.0.0.0/8 dev eth0 scope link
```

La VM comunica con se stessa, con l'host e con internet:

```
# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.023 m

# ping 10.0.0.254
PING 10.0.0.254 (10.0.0.254) 56(84) bytes of data.
64 bytes from 10.0.0.254: icmp_seq=1 ttl=64 time=0.116 ms
64 bytes from 10.0.0.254: icmp_seq=2 ttl=64 time=0.064 ms

# ping www.google.it
PING www.google.it (216.58.205.163) 56(84) bytes of data.
64 bytes from mil04s28-in-f3.1e100.net (216.58.205.163): icmp_seq=1 ttl=52 time=11.9 ms
64 bytes from mil04s28-in-f3.1e100.net (216.58.205.163): icmp_seq=2 ttl=52 time=10.0 ms
```

Siamo in grado di aggiornare il sistema ed installare pacchetti con apt, avviare serivizi accessibili dalla rete in cui è collegata la VM ecc..

Per rendere permanente la configurazione di rete dell'interfaccia etho inseriamo le informazioni nel file /etc/network/interfaces:

```
# cat >> /etc/network/interfaces << "EOF"
allow-hotplug eth0
iface eth0 inet static
```

```
 address 10.0.0.1
 netmask 255.0.0.0
 gateway 10.0.0.254
EOF
```

8 - Swap

Assegnamo dello spazio di swap alla VM:

```
$ dd if=/dev/zero of=uml-00-swap bs=1024K count=512
$ sudo mkswap uml-00-swap
$ linux ubda=uml-00-root \
        ubdb=uml-00-swap \
        umid=uml-00 \
        mem=256M \
        con0=fd:0,fd:1 con=pts \
        eth0=tuntap,tap0,10.0.0.254
```

Effettuiamo login e verifichiamo la presenza di /dev/ubdb:

```
# ls -l /dev/ubdb
brw-rw---- 1 root disk 98, 16 giu 15 10:40 /dev/ubdb
```

Attiviamo lo spap:

```
# swapon -v /dev/ubdb
swapon: /dev/ubdb: found signature [pagesize=4096, signature=swap]
swapon: /dev/ubdb: pagesize=4096, swapsize=536870912, devsize=536870912
```

Si può montare a partizione di swap al boot della VM, ricordarsi poi di passare sempre il parametro ubdb=... quando la si avvia altrimenti init inizierà a cercare un device che non esiste.

```
# echo "/dev/ubdb swap swap defaults 0 0" >> /etc/fstab
```

9 - Condividere filesystem tra VM

Per risparmiare spazio nell' host, UML mette a disposizione un sistema per creare i virtual block device (ubda, ubdb, ...) tramite due file, uno in cui scrivere le modifiche (il cowfile) ed uno condiviso che viene montato read-only (sharedfile che può essere il root file system). La sintassi è la seguente:

```
linux ... ubda=cowfile,sharedfile ...
```

Utilizzando la VM uml-01, creaiamo 4 cow files (2 per la root e 2 per var):

```
$ uml_mkcow uml-01a-root uml-01-root
$ uml_mkcow uml-01b-root uml-01-root
$ uml_mkcow uml-01a-var uml-01-var
$ uml_mkcow uml-01b-var uml-01-var
```

I file appena creati sono molto più piccoli del root file system a cui fanno riferimento e si può essere tratti in inganno facendo un semplice ls -lh:

```
$ ls -lh uml-01?-root
-rw-r--r-- 1 guru guru 257M giu 15 12:02 uml-01a-root
-rw-r--r-- 1 guru guru 257M giu 15 12:02 uml-01b-root
```

Sembra che occupino 257M ma una essendo sparse file, la loro dimensione è molto minore come mostra una lettura più attenta:

```
$ ls -lsh uml-01?-root
12K -rw-r--r-- 1 guru guru 257M giu 15 12:02 uml-01a-root
12K -rw-r--r-- 1 guru guru 257M giu 15 12:02 uml-01b-root
```

In realtà, non avendo delta da memorizzare, occupano solo 12K. In due diverse console, avviamo le VM in cui modificheremo gli hostname:

```
$ linux ubda=uml-01a-root,uml-01-root \
        ubdb=uml-01a-var,uml-01-var \
        umid=uml-01a \
        mem=256M con0=fd:0,fd:1 con=pts

$ linux ubda=uml-01b-root,uml-01-root \
        ubdb=uml-01b-var,uml-01-var \
        umid=uml-01b \
        mem=256M con0=fd:0,fd:1 con=pts
```

Effettuiamo login in altre 2 console (utilizzando i /dev/pts/* mostrati nei messaggi di boot delle due VM) e modifichiamo l'hostname:

```
# echo "uml-01a" > /etc/hostname
# shutdown now

# echo "uml-01b" > /etc/hostname
# shutdown now
```

Nell' host possiamo verificare che i cowfile sono stati modificati passando da 12K a 208K e 168K:

```
$ ls -lsh uml-01?-root
208K -rw-r--r-- 1 guru guru 257M giu 15 12:17 uml-01a-root
168K -rw-r--r-- 1 guru guru 257M giu 15 12:16 uml-01b-root
```

Se la avviamo di nuovo le due VM con i due cowfile tra i messaggi di boot vediamo che gli hostname sono cambiati:

```
systemd[1]: Set hostname to <uml-01a>.

systemd[1]: Set hostname to <uml-01b>.
```

Cowfiles e Sharedfiles sono strettamente legati, se ora avviamo la VM come nei primi esempi senza cowfile, corromperemo questa relazione. Avviamo la VM senza cowfile, facciamo login e shutdown:

```
$ linux ubda=uml-01-root \
        ubdb=uml-01-var \
        umid=uml-01 \
        mem=256M con0=fd:0,fd:1 con=pts
```

Tra i messaggi di boot notiamo che l'hostname è ancora quello originale:

```
systemd[1]: Set hostname to <uml-01>.

# shutdown now
```

Ora avviamo con i cowfiles:

```
$ linux ubda=uml-01b-root,uml-01-root \
        ubdb=uml-01b-var,uml-01-var \
        umid=uml-01b \
        mem=256M con0=fd:0,fd:1 con=pts
```

Notiamo che la VM non si avvia mostrando diversi errori di questo tipo:

```
mtime mismatch (1528999971 vs 1529058969) of COW header vs backing file
Failed to open 'uml-01b-root', errno = 22
.
.
.

Annullato
```

Non possiamo più utilizzare i vecchi cowfiles ed abbiamo perso le modifiche che contengono, quindi attenzione :=)

10 - Accesso alle risorse host

UML mette a disposizione il filesystem della macchina host attraverso il mount type hostfs, cosa molto utile per scambiare files. I permessi vengono ereditati dall'utente che ha avviato la VM quindi a meno che non stiamo utilizzando la VM come root, non è possibile fare danni nell' host.

Dentro la VM:

```
# mkdir host
mount -t hostfs none host/
```

```
# ls -1 host/
bin
boot
.
.
.
tmp
usr
var
vmlinuz
vmlinuz.old
```

```
# touch host/home/guru/prova.txt
# ls -lh host/home/guru/prova.txt
-rw-r--r-- 1 1001 1001 0 Jun 15 10:52 host/home/guru/prova.txt
```

```
# touch host/root/prova.txt
touch: cannot touch 'host/root/prova.txt': Permission denied
```

11 - Aumentare lo spazio disco

E' possibile aumentare lo spazio in un virtual block device aggiungendo spazio nel file utilizzato. Ad esempio nell' host abbiamo il root filesystem si 256M:

```
$ ls -lh uml-01-root
-rw-r--r-- 1 guru guru 256M giu 17 15:57 uml-01-root
```

Aggiungiamo 0 alla fine del file, partendo dalla file (seek=256 con blocchi da 1024K) per aòtrettanti 256M (count=256):

```
$ dd if=/dev/zero of=uml-01-root bs=1024K count=256 seek=256
256+0 record dentro
```

```
256+0 record fuori
268435456 bytes (268 MB, 256 MiB) copied, 0,548811 s, 489 MB/s
```

Ora avviamo la VM, facciamo login e ridimensioniamo il file system del device /dev/ubda:

```
# df -h /
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       240M  192M   32M  86% /

# resize2fs /dev/ubda
resize2fs 1.43.4 (31-Jan-2017)
Filesystem at /dev/ubda is mounted on /; on-line resizing required
old_desc_blocks = 2, new_desc_blocks = 4
The filesystem on /dev/ubda is now 524288 (1k) blocks long.


# df -h /
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       488M  192M  269M  42% /
```

Riferimenti

- The User-mode Linux Kernel Home Page
- Wiki User-mode Linux

Estratto da "https://www.electroyou.it/mediawiki/index.php?title=UsersPages:Ilguru:user-mode-linux"