PARTE A

INTRODUZIONE A USER MODE LINUX

Modulo 1:
User Mode Linux

User Mode Linux

- **Esegue il Kernel di Linux come se fosse un normale programma Linux**

- **Crea un ambiente**
  - Protetto
  - Facilmente ispezionabile
  - Facilmente controllabile

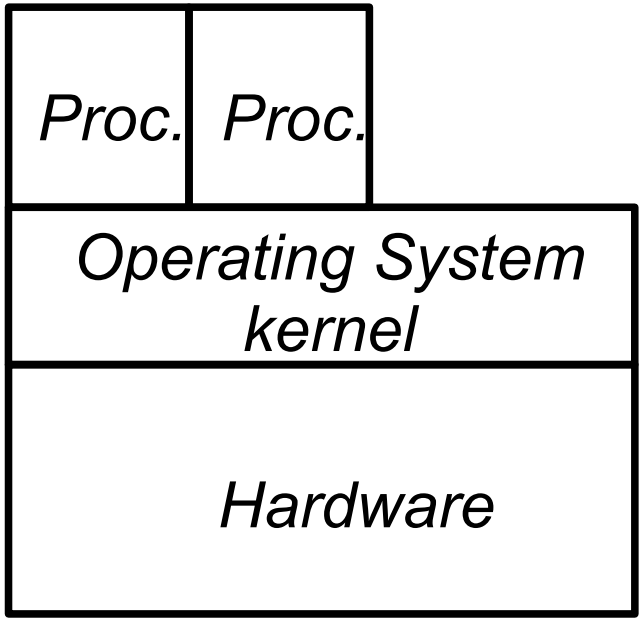
  - **Homepage**
    - http://user-mode-linux.sourceforge.net/

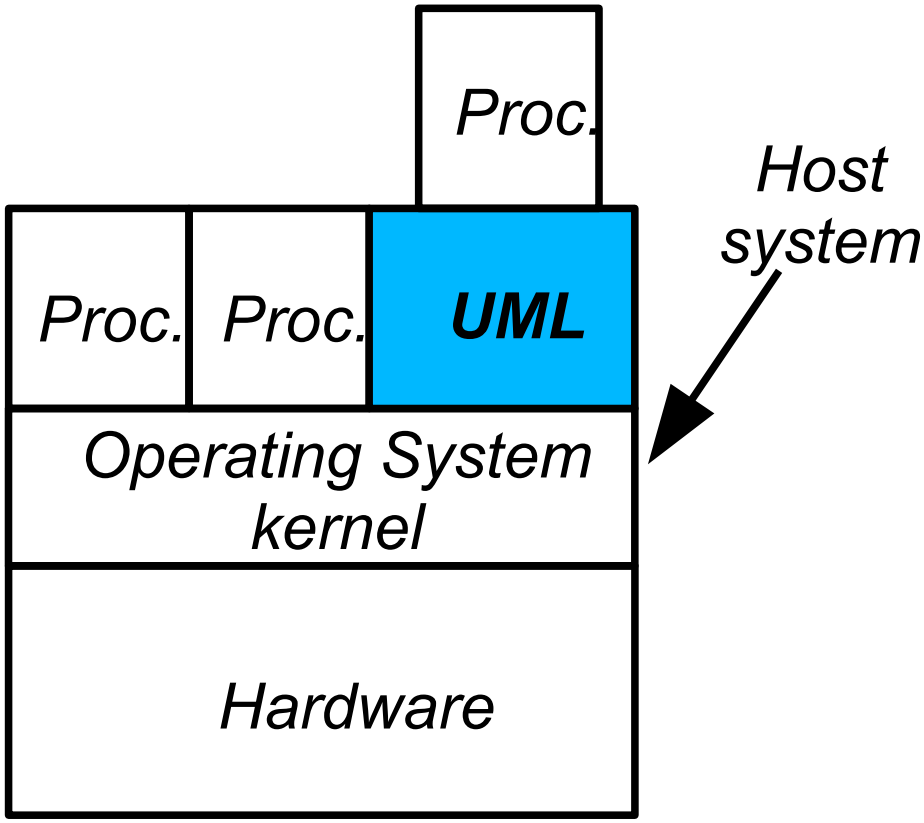
Storia di User Model Linux

- **Febbraio 1999:**
  - Jeff Dike inizia a lavorare a User Mode Linux
  - La prima versione si presenta come una patch per il kernel versione 2.0
- **1999-2002:**
  - Lo sviluppo di User Mode Linux procede in parallelo a quello del kernel
  - User Mode Linux è sempre una patch esterna per i kernel versioni 2.0, 2.2, 2.4
- **Settembre 2002:**
  - User Mode Linux è inserito nel kernel 2.5.34
- **Oggi:**
  - User Mode Linux è parte del kernel dalla versione 2.6

Come funziona (1)

- **Di norma i processi si appoggiano sul kernel del sistema operativo che incapsula l'hardware**
- **User Mode Linux aggiunge un livello software**





Come funziona (2)

- **User Mode Linux aggiunge uno strato software**
  - Per il kernel “Host” è un processo normale
  - Per i processi che girano sulla macchina virtuale è un kernel
  - Offre risorse virtuali (rete, dischi, CPU, …)

- **User mode Linux è un kernel full featured**
  - Nei sorgenti del kernel è una nuova architettura hardware (um)
  - Modifiche di alcuni aspetti architecture dependent nel kernel (directory /arch/um nei sorgenti)
  - **Tutto il resto del kernel rimane invariato**

A cosa serve User Mode Linux

- **Debugging del kernel**
  - Se il kernel UML si pianta si può analizzare cosa è andato storto
- **Sicurezza**
  - Un problema (o una compromissione) su UML non interessa il resto della macchina
  - Jail system (bind, sendmail), Honeypot
- **Testing dei sistemi**
  - Consente di creare reti virtuali
  - Consente di provare diverse ditribuzioni
- **Didattica**
  - Se gli studenti fanno qualche danno non compromettono le macchine “vere”

Come si presenta UML

- **Eseguibili**
  - `linux` (il kernel)
  - **`uml_moo`, `uml_mkcow`** (utility per la gestione dei dischi in modalità Copy On Write)
  - **`uml_switch`** (gestione della rete virtuale)
  - **`uml_mconsole`** (gestione delle macchine dall'esterno)
  - altre utility...

- **Richiede un'immagine di un filesystem funzionante (un file organizzato come se fosse un disco fisso)**
- **Altri software e componenti sviluppati successivamente estendo le funzionalità di UML (e.g., vde - virtual distributed ethernet)**
- **NOTA: in questo corso ci si concentra su un sistema Debian GNU/Linux, per la creazione di un kernel o di un'immagine alternativa consultare le guide allegate**

Nota di laboratorio

- **Per chi lavora sulle macchine del linfa:**
  - Scaricare l'immagine del kernel e del filesystem
- **Per chi lavora sulle macchine virtuali**
  - Kernel e filesystem si trovano sotto */user*/share/marionnet/kernels */user*/share/marionnet/filesystems
  - Per comodità creeremo link nella home

Lanciare il kernel UML

Sintassi comando:

```
linux [opzioni-kernel] [opzioni-init]
```

In questo corso impiegheremo diverse opzioni del kernel, mentre raramente impiegheremo opzioni di init.

- **Le opzioni possono modificare alcune delle caratteristiche dei sistemi guest**, e **definiscono i dispositivi e le interfacce di cui vogliamo dotare I sistemi guest. Nel corso impiegheremo opzioni per:**
  - **montare dischi aggiuntivi**
  - **usare interfacce di rete ethernet**
  - **fare comunicare i sistemi guest con quello host, e con internet**

Opzioni del kernel UML: Montare Dischi

Sintassi dell'opzione per aggiungere un disco:

```
ubd<N>=[<cow_file>,]<image_file>
```

- **ubd<N> identifica che si vuole aggiungere un disco virtuale (solitamente <N>=0 indica il filesystem montato come root)**
- **<image_file> è il nome del file con l'immagine del filesystem**
- **<cow_file> è il nome del file di appoggio per la modalità copy on write**

NB: è obbligatorio indicare almeno un disco fra le opzioni passate al kernel, che viene montato come root filesystem della macchina virtuale UML

Esempio (creare l'host h1 con filesystem root rootfs.ext4):

```
linux ubd0=h1.cow,rootfs.ext4
```

Opzioni del kernel: Interfacce di Rete

Sintassi dell'opzione per aggiungere un'interfaccia di rete:

```
eth<N>=daemon,,unix,<socket.ctl>
```

oppure

```
eth<N>=vde,<socket-dir>[,[<mac_addr>]
[,<port_number>]]
```

- **<N>** indica l'identificativo dell'interfaccia di rete
- **daemon indica che vogliamo utilizzare il protocollo per comunicare con *uml_switch***
- **vde** indica che utilizziamo lo stack ***vde*** (per comunicare con *vde_switch*)

Esistono altre modalità di gestione della rete ma richiedono privilegi particolari per poter essere utilizzate (e.g. tun/tap).

Esempio (collegare l'host h1 alla porta 5 dello switch vde in /tmp/s1):

```
linux ubd0=h1.cow,rootfs.ext eth0=vde,/tmp/s1,,5
```

Lanciare il kernel: Altre opzioni

`-mem=<MB>`

- quantità di memoria centrale assegnata alla macchina UML

`-root=<nome_device>`

- **specifica la root partition, di solito non è necessaria**

`-devfs=mount`

- **abilita l'utilizzo del dev filesystem**
- **usata con UML versione 2.4**

Errori comuni

- **Mettere lo spazio tra i sottoparametri del kernel**
  - NO: `ubd0=file.cow, file.ext3`
  - SI: `ubd0=file.cow,file.ext3`
- **Sbagliare a scrivere i nomi dei parametri**
  - NO: `udb`
  - SI: `ubd`
  - NO: `demon o damon`
  - SI: `daemon`
- **Terminare in modo sporco i processi UML:**

  MAI chiudere le finestre di UML, dare il comando shutdown (o equivalente) dalla linea di comando del sistema UML

Esecuzione di vde_switch

```
vde_switch -s <socket-dir>
```

Con uml, connettersi allo switch tramite con lo stack ***vde:***

```
eth0=vde,<path>,<mac_addr>,<port_number>
```

esempi: `vde_switch -s /tmp/switch1`

(1) `eth0=vde,/tmp/switch1` ← mac e porta auto

(2) `eth0=vde,/tmp/switch1,,1` ← mac auto, porta 1

In seguito all'esecuzione si ha accesso a una console per configurare lo switch

Parametri opzionali di vde_switch

Vedere **`vde_switch --help`** per l'elenco completo:

`-hub` : disattiva l'engine dello switch e si comporta da hub (esteso di `-x`)

`-M <path>` : crea una socket per connettersi da un terminale **`vdeterm`**

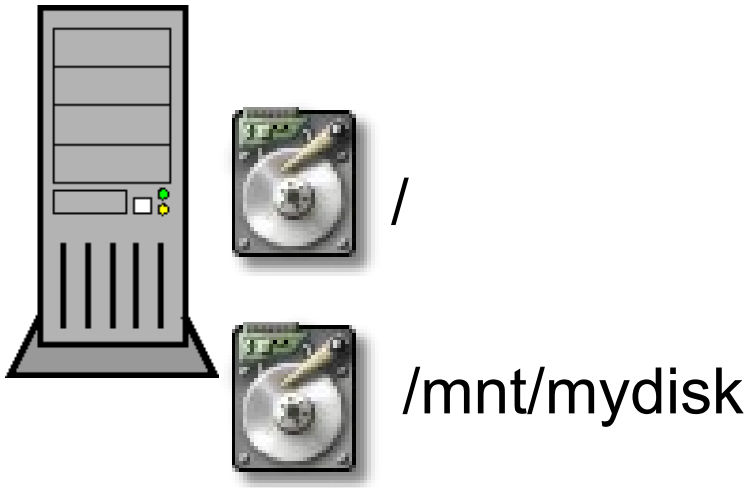
`-f <path>` : specifica un file di configurazione (alias di --rcfile)

`-d`: esegue lo switch in background (utile da utilizzare insieme a **`-M`**)

Prima esperienza: collegare due dischi a una macchina UML

Creare un nodo con due dischi virtuali:

- **il primo è l'immagine rootfs.ext4, montato da UML come root filesystem**
- **il secondo è un'immagine di un filesystem ext4 vuoto, creato appositamente, da montare in /mnt/mydisk**



- **Obiettivo**: far funzionare il nodo e vedere se funziona a dovere

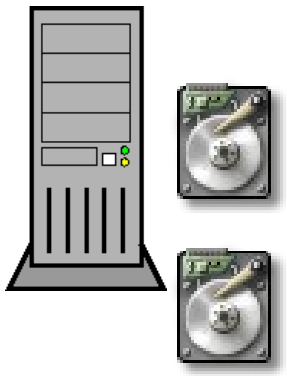
Prima esperienza: scaletta

Creare un nodo con due dischi virtuali:

- **il primo è l'immagine rootfs.ext4, montato da UML come root filesystem**
- **il secondo è un'immagine di un filesystem ext4 vuoto, creato appositamente, da montare in /mnt/mydisk**

Scaletta:

1) creare il filesystem
2) eseguire il comando linux con le opzioni corrette
3) montare il secondo disco in /mnt/mydisk



1. Creare un'immagine di un filesystem usando dd e mkfs

dd : comando ***data dump***, legge e scrive dati in modo *raw*

```
$ dd if=<input-file> of=<output-file>                \
     bs=<block-size> count=<blocks-number>           \
     seek=<offset-size>
```

Esempio sparse file:

```
$ dd if=/dev/zero of=myfs.ext4 bs=1 count=1 seek=300M
```

Leggo da /dev/zero 1 Byte e lo scrivo su myfs.ext a 300MB rispetto all'inizio del file. La dimensione del file è di ~300MB, ma i dati realmente scritti sul disco sono solo di 1Byte (usare ls -s o du -h).

Per creare il filesystem (senza tabella delle partizioni):

```
$ mkfs.ext4 myfs.ext4
```

2. Eseguire il kernel UML

(per utilizzare più facilmente kernel e rootfs di Marionnet) Creazione link "comodi" del Kernel e dell'immagine già presenti sui computer del laboratorio:

```
$ ln -s /usr/share/marionnet/kernels/linux-par-1 linux
$ ln -s /usr/share/marionnet/filesystems/machine-par-1 \
        rootfs.ext4
```

Eseguire il kernel UML

```
$ ./linux ubd0=root1.cow,rootfs.ext4 \
          ubd1=usr1.cow,myfs.ext4
```

NB: per spegnere la macchina UML utilizzare uno dei comandi classici utilizzati per spegnere i sistemi GNU/Linux, ad esempio **`halt`**.

3. Montare il disco sulla macchine guest

1. Identificare il path del device in `/dev` con i comandi oppurtuni (e.g., `blkid`, `lsblk`):

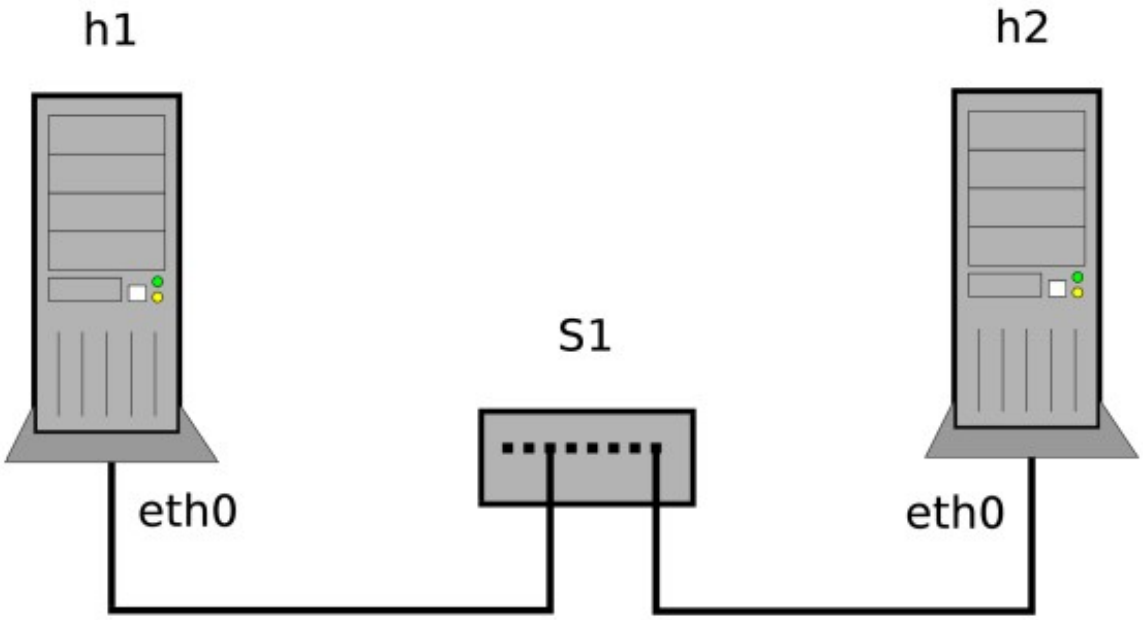
```
# lsblk
  NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
   1.ubda  98:0    0  1.5G  0 disk /
   2.ubdb  98:16   0  300M  0 disk
# blkid
 /dev/ubda: UUID="ed0cc056-705f-45e7-8dd4-2069584becae"
            TYPE="ext4"
 /dev/ubdb: UUID="1fb53b75-f82c-49f3-87a1-c76a06852a1d"
            TYPE="ext4"
```

2. Montare i dischi (modificare il file `/etc/fstab` per montare i dischi all'avvio)

```
# mkdir /mnt/myfs
# mount <dev> /mnt/myfs
```

Seconda esperienza: Configurazione di una rete con UML e vde_switch

- **Collegare in rete due nodi tramite uno switch.**



Obiettivi:

- eseguire correttamente lo switch vde e le macchine UML
- verificare il collegamento

Soluzione: eseguire vde_switch e le macchine UML

- Creare una directory apposita in cui sono disponibili il kernel UML e il filesystem rootfs.ext4 (vedi esercitazione precedente) e utilizzare 3 terminali separati per s1, h1 e h2:

```
(s1)      $ vde_switch -s s1 -M `pwd`/s1_term -d
(s1)      $ vdeterm s1_term

(h1)      $ ./linux ubd0=h1.cow,rootfs.ext4 eth0=vde,s1

(h2)      $ ./linux ubd0=h2.cow,rootfs.ext4 eth0=vde,s1
```

Soluzione: test del collegamento su vde_term

```
   (s1-vde) $ vde[s1_term]: port/print

Port 0001 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
 Current User: luca Access Control: (User: NONE - Group: NONE)
 IN:  pkts          0          bytes                       0
 OUT: pkts          0          bytes                       0
  -- endpoint ID 0007 module unix prog   : UML vde_transport
user=luca PID=32298
Port 0002 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
 Current User: luca Access Control: (User: NONE - Group: NONE)
 IN:  pkts          0          bytes                       0
 OUT: pkts          0          bytes                       0
  -- endpoint ID 0009 module unix prog   : UML vde_transport
user=luca PID=32636
   Success
```

Soluzione: testing connettività con tool di rete

Per testare il collegamento è necessario utilizzare dei comandi che verranno spiegati meglio nelle prossime lezioni.

Qui si propone un test di connettività “minimalista”, che non richiede la configurazione delle interfacce di rete:

```
(h1-uml) # ifconfig eth0 up
(h1-uml) # arping -0Bi eth0

(h2-uml) # ifconfig eth0 up
(h2-uml) # tcpdump -ni eth0
```

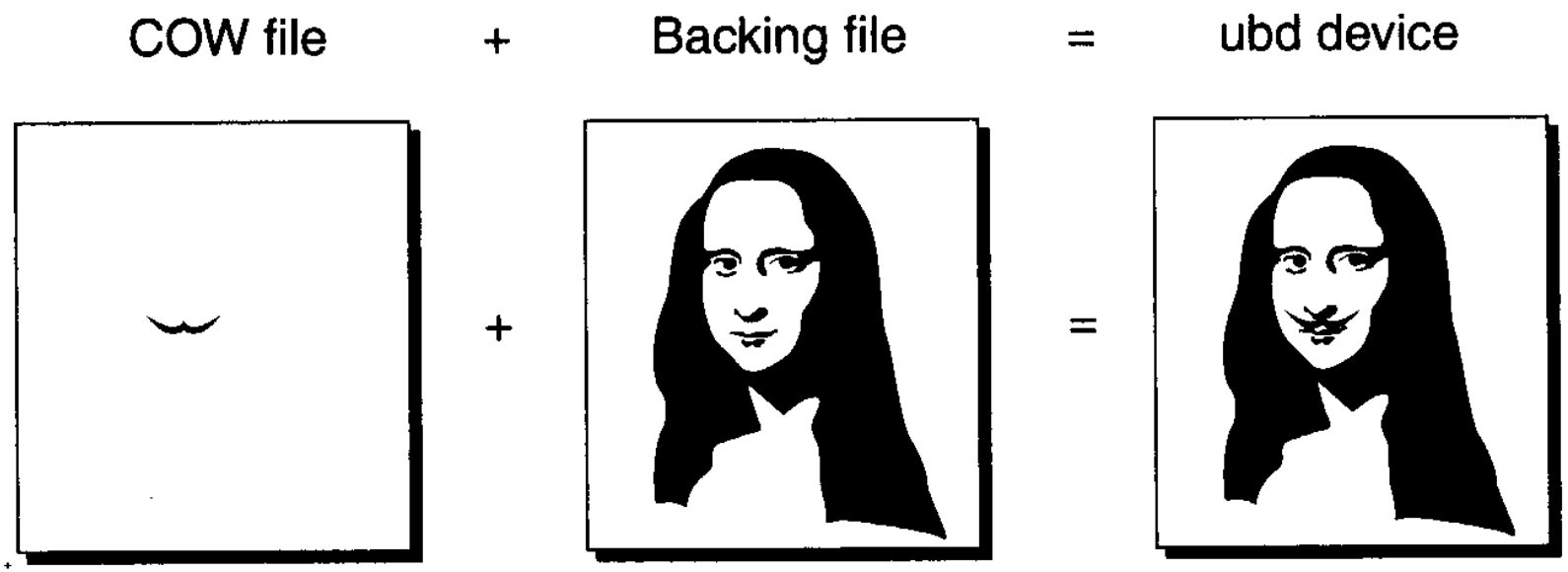
Osservare se h2 “vede” il traffico generato da h1

Tutti i comandi e i protocolli utilizzati verranno spiegati nelle prossime lezioni

Modulo 2:
File copy on write (COW)

File copy on write (1)

- **COW = copy on write**
- **Dalle parole di Jeff Dyke:**
  - Come fare a disegnare i baffi alla gioconda senza andare in prigione?
  - Semplice: mettiamo un telo di plastica davanti al dipinto

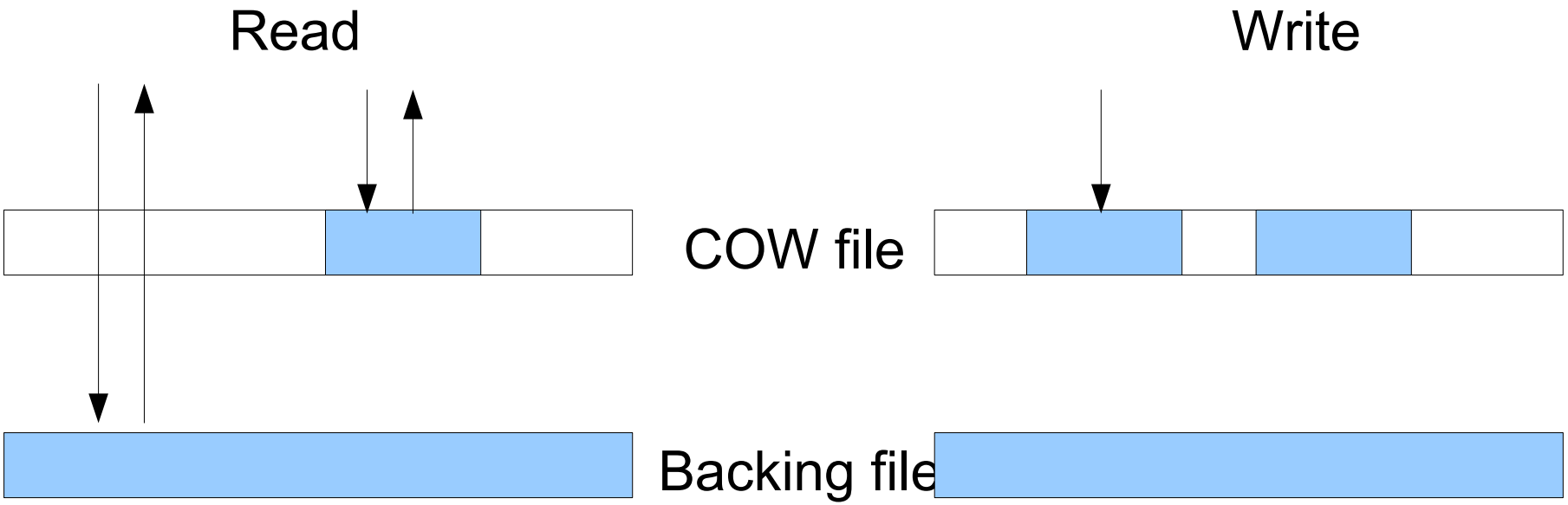


File copy on write (2)

- **Consente di non sovrascrivere le immagini di dischi originali**
  - L'immagine originale (backing file) viene usata in modalità read-only

- **il file .cow contiene le differenze con il file originale**
  - Una stessa immagine originale può essere condivisa tra tanti sistemi virtuali
  - Ciascuno di questi sistemi scriverà le sue differenze sul suo file COW
  - Questo consente di **risparmiare spazio perchè i file .cow sono scattered file**
  - Attenzione al timestamp dei file: **il file originale DEVE essere più vecchio del file .cow**

File copy on write (3)



Utility per gestire file Copy On Write

Le immagini cow sono gestite direttamente dal kernel

`uml_mkcow`

- **Crea un'immagine cow**
- ***Questa operazione viene eseguita in modo trasparente dal kernel UML***

`uml_moo`

- **Da un'immagine cow e dalla corrispondente immagine originale crea una nuova immagine**
- **Integra e sincronizza le differenze**
- **Uso: uml_moo <file.cow> <nuova_immainge.ext3>**
- **Non serve immagine di partenza: l'informazione sta nel file .cow**

Modulo 3:
Copia e Backup
di dischi virtuali

Come si scompatta un archivio

Si utilizza il comando: tar (TApe aRchive)

- **Opzioni**
  - -v (verbose operation)
  - -f file
  - comando
    - -x scompatta
    - -t mostra contenuto
  - algoritmo di compressione
    - -z gzip (archivi .tar.gz)
    - -j bzip2 (archivi .tar.bz2)
- **Esempio:**

```
tar -xzvf archivio.tar.gz
tar -xzvf archivio.tgz
```

Salvare il lavoro fatto (1)

- **Alla fine di una sessione di lavoro può far comodo salvare le proprie immagini UML**
- **Il meccanismo Copy on Write è molto pignolo sulla consistenza**
  file cow ↔ ***file ext3***
  - L'età dei file deve corrispondere
  - In caso contrario UML si rifiuterà all'avvio successivo di montare i dischi
- **Occorre preservare le date di accesso**
  - Opzione **-p** (preserve) o **-a** (archive) del comando ***cp***
  - Opzione **--atime-preserve** del comando ***tar***
- Ed è meglio sfruttare le caratteristiche dei file sparse
  - Opzione **--sparse=always** del comando ***cp***
  - Opzione **-S** (S maiuscola) del comando ***tar***

Salvare il lavoro fatto (2)

Esempi:

- Creare un archivio contenente tutti I file cow e le immagini disco originali

```
tar -cSzvf mie_immagini.tgz *.cow *.ext3 \
    --atime-preserve
```

- Copiare tutti i file cow nella sottocartella *backup*

```
cp -p --sparse=always *.cow ./backup/
cp -a --sparse=always *.cow ./backup/
```

NB: operare sui file sempre a macchine (guest) spente!

Fondere i file cow con i file originali

Per fondere i file cow con le immagini disco originali, UML mette a disposizione il comando uml_moo

Sintassi

`./uml_moo file.cow new-file.ext3`

Esempio:

```
$ ls -lsh
231M -rw-r--r-- 1 root root 350M Oct  8 15:24 file.ext3
1.1M -rw-r--r-- 1 root root 351M Oct 11 15:47 file.cow

$ ./uml_moo file.cow new-file.ext3
$ ls -lsh
231M -rw-r--r-- 1 root root 350M Oct  8 15:24 file.ext3
1.1M -rw-r--r-- 1 root root 351M Oct 11 15:47 file.cow
231M -rw-r--r-- 1 root root 350M Oct 11 15:48 new-file.ext3
```