Alma Mater Studiorum – Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

MODIFICA DINAMICA DEL CLOCK DI VIRTUALBOX

Relatore: Chiar.mo
prof. VITTORIO GHINI

Presentata da:
GIOVANNI CATTANI

Sessione I

Anno accademico 2013-2014

Indice

Prefazione

Questa tesi focalizzerà l'attenzione sullo studio del funzionamento del clock di Virtualbox (noto software per l'utilizzo di macchine virtuali) e sui possibili approcci che permettono di modificare dinamicamente il clock in questione.
Dobbiamo qui precisare che il clock di cui si parla è esclusivamente di tipo virtuale; sarebbe infatti impensabile trattare questo tipo di problema riferendosi al clock reale di un hardware generico.
Poter modificare il clock in un sistema è, in generale, una tecnica che può rivelarsi utile per vari scopi: ad esempio, velocizzando il clock, si possono ridurre i tempi di esecuzione di programmi che normalmente ne richiederebbero molto di più; rallentando il clock, invece, è possibile aggirare problemi legati alla latenza della rete (nel caso di comunicazione tra programmi a distanza).

Il lavoro che segue sarà suddiviso in capitoli, ognuno dei quali tratterà un preciso argomento; si partirà dalle motivazioni che hanno portato allo studio di questa tesi, per poi proseguire con una breve panoramica sul clock (reale), spiegando cos'è esattamente, come'è strutturato e qual è il suo ruolo all'interno di un sistema generico.
In seguito si parlerà di Virtualbox, presentando tale software con una breve introduzione, illustrando i punti che ci hanno permesso di iniziare lo studio vero e proprio del problema.
Infine verranno mostrati i risultati ottenuti, un metodo concreto, alcuni problemi ed infine eventuali sviluppi futuri.

Prima di proseguire, mostriamo la rappresentazione grafica di un qualsiasi sistema operativo che esegue un secondo sistema tramite Virtualbox.

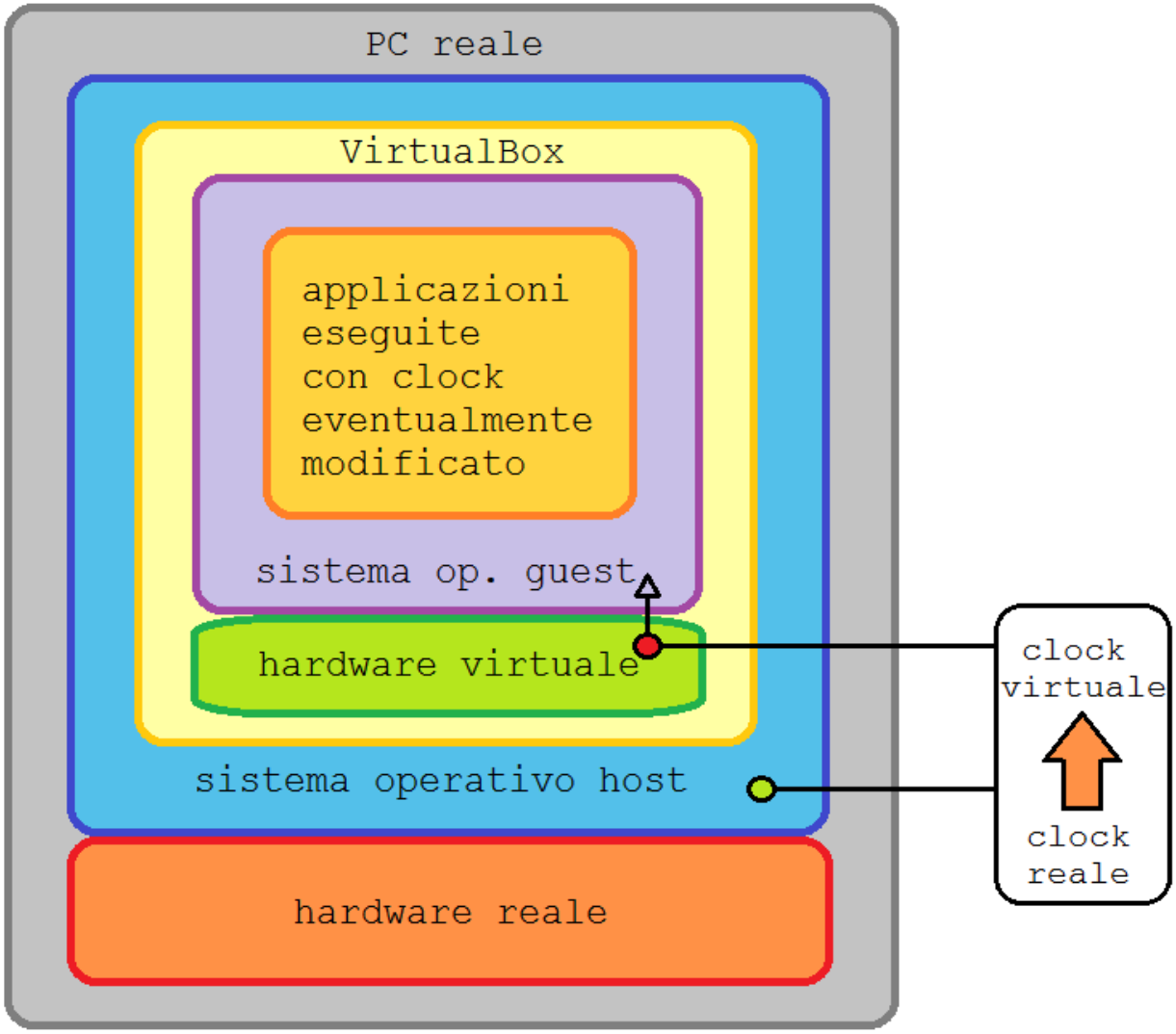

Figura introduttiva - Stratificazione dell'hardware-software

Come mostra la figura, il clock virtuale si basa su quello reale, ma viene mediato tramite software e proprio per questo motivo sarà possibile intervenire e scrivere le opportune modifiche del programma.

Capitolo 1

Motivazioni

Come abbiamo detto, poter modificare dinamicamente il clock può risultare utile in molteplici situazioni. Quella di nostro interesse si concentra sul caso in cui il tempo all'interno della macchina virtuale viene rallentato, in conseguenza di una riduzione della velocità del clock.
In particolare abbiamo pensato ad un ipotetico scenario, che vede protagonisti un'azienda di sviluppo software e la necessità di ambienti (le macchine virtuali) in cui poter **testare programmi che interagiscano a distanza, tramite una connessione di rete**. Quest'ultimo aspetto racchiude in sé il problema principale: la distanza.
Infatti una comunicazione tramite rete provoca necessariamente lag, ovvero un ritardo dovuto dal tempo di latenza (fattore che determina la velocità di risposta di un sistema). La quantità del ritardo è variabile e dipende da diversi fattori, tra cui ad esempio la distanza tra i due (o più) partecipanti alla comunicazione.

Precisiamo inoltre che **i programmi considerati non devono assolutamente essere modificati**.

1.1 Un esempio concreto

Vediamo ora un esempio concreto per capire meglio la situazione.
L'azienda sviluppa due programmi, A e B, che hanno la necessità di comunicare tra loro.

In particolare, durante l'esecuzione, ad un certo istante, A manda un segnale a B ed attende 2 secondi; allo scadere del timeout, A esegue determinate azioni basandosi sul tipo di risposta ricevuta da B ed è previsto anche il caso in cui B non risponda entro il tempo atteso. Il test dei software avviene in due sedi differenti dell'azienda, X e Y, dislocate a migliaia di chilometri di distanza. Avremo quindi il seguente scenario:

- nella sede X, tramite una macchina virtuale gestita da Virtualbox, è in esecuzione il programma A
- nella sede Y, tramite una macchina virtuale non specificata, è in esecuzione il programma B
- A e B comunicano tramite una connessione di rete

È chiaro che, in una situazione come quella presentata, si ha un problema persistente: quando A manda il segnale e scatta il timeout di attesa (2 secondi), il tempo effettivo di risposta di B dipenderà anche dalla latenza, che necessariamente interferirà, aumentando il tempo di risposta reale (inteso come il tempo di comunicazione tra A e B trascurando il ritardo di rete).
In uno scenario come questo può quindi verificarsi il seguente caso pessimo: B risponde in tempo, rispettando quindi i 2 secondi di attesa massima previsti da A, ma, a causa della latenza di rete, A non riceve in tempo la risposta compromettendo tutta l'esecuzione seguente dei due programmi.

1.2 Conclusioni

Per risolvere il problema sopra citato, prendendo sempre in considerazione il fatto **di non poter in alcun modo modificare i programmi da testare**, abbiamo pensato di agire ad un livello notevolmente esterno rispetto a tali programmi: il codice di Virtualbox.

In questo modo, non solo evitiamo di modificare i programmi stessi, ma lasciamo inalterato anche il sistema operativo eseguito dalla macchina virtuale.
Poter regolare dinamicamente il clock di Virtualbox ci permetterà di rallentare a piacere l'esecuzione dei programmi, ritardando quindi tutte le meccaniche legate alla temporizzazione, compresi i sistemi di timeout.
Negli esempi pratici da noi svolti e documentati nei capitoli successivi, abbiamo utilizzato la funzione `sleep` (noto meccanismo di timeout) proprio per lavorare in uno scenario simile a quello visto nella sezione precedente.

Capitolo 2

Panoramica sul clock

In questo capitolo verrà presentata una panoramica generale sul clock (reale) di un comune calcolatore.
Il clock è un meccanismo hardware, integrato nella scheda madre di qualsiasi elaboratore. Viene utilizzato per tenere traccia dello scorrere del tempo, ma anche per fornire meccanismi di sincronizzazione al sistema operativo.
Questo meccanismo lavora ad una data frequenza (espressa in MHz) e genera degli interrupts hardware che possono essere singoli (detti "one-shot") oppure periodici. Quest'ultimi sono quelli di nostro interesse in quanto vengono contati per stabilire il tempo trascorso. In genere sono dotati di un numero di comparatori programmabili (solitamente dal kernel del sistema operativo all'avvio) in modo da poter settare la frequenza di generazione degli interrupt.

2.1 Due tipi di orologio

Esiste uno stretto legame tra il clock e l'orologio di sistema: possiamo distinguere tra due tipi di orologi, comunemente chiamati "orologio hardware" ed "orologio software"[1].

Orologio hardware: misurato tramite il clock, questo orologio è sempre attivo, anche quando il calcolatore è spento o scollegato dall'alimentazione; ciò è possibile grazie ad una batteria, la quale alimenta il circuito e viene ricaricata ogni volta che

la macchina viene alimentata; è genericamente chiamato anche "orologio del BIOS". È possibile stabilire l'ora a cui fare riferimento scegliendo tra quella locale o quella universale (UTC). Solitamente, in ambiente Linux, si opta per la seconda scelta, in quanto la sincronizzazione con l'ora locale viene poi affidata al sistema operativo, in modo da non creare nessun tipo di conflitto.

Orologio software: come si può intuire dal nome questo orologio viene gestito via software. A meno di casi particolari l'unico legame con l'apparato hardware è quando il calcolatore viene acceso: infatti, durante la fase di inizializzazione del sistema operativo, il kernel legge l'ora dall'orologio hardware; da questo momento in poi, però, la misurazione avverrà interamente a livello software. Esistono comunque alcuni metodi per forzare la sincronizzazione con l'orologio hardware: in ambiente Linux è possibile ad esempio utilizzare il comando da terminale `hwclock -s`).

Il kernel tiene traccia dell'ora basandosi sull' "epoch", uno standard che stabilisce la misura del tempo calcolando il numero di secondi trascorsi dall'1 gennaio del 1970. In seguito per applicare la differenza data dall'ora locale viene utilizzato il file `etc/localtime`.

2.2 La funzione gettimeofday

```
#include <sys/time.h >
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Questa è una delle funzioni che servono per calcolare il tempo nei sistemi Linux[2]; il secondo parametro servirebbe per calcolare l'ora locale, ma è ormai considerato obsoleto, e viene quindi inserito NULL durante la chiamata. Il primo parametro invece è un puntatore ad una struttura definita nel seguente modo:

```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

come mostrato, `timeval` è composto da due campi, che servono per indicare rispettivamente i secondi ed il millisecondi.
Quando `gettimeofday` viene chiamata salva all'interno di `timeval` il tempo trascorso dalla mezzanotte (UTC) dell'1 gennaio 1970, la cosiddetta data "epoch".
La funzione restituisce 0 in caso di success, -1 in caso di error, settando opportunamente da variabile `errno`, come di consueto.

Una piccola curiosità riguarda tutti i calcolatori e i sistemi operativi a 32 bit: poiché `time_t` è di tipo intero con segno a 32 bit, il 19 gennaio 2038 saranno trascorsi sufficienti secondi da non poter più essere salvati in una variabile definita in questo modo.

2.3 Tipologie di clock

Esistono diversi tipi di clock, che si differenziano per la frequenza alla quale lavorano, per la quantità di comparatori settabili e per la precisione ottenibile.
Nei sistemi più datati veniva utilizzato il PIT (Programmable Interval Timer), che lavora ad una frequenza di 1.19 MHz; poi venne aggiunto il RTC (Real Time Clock), che lavora ad una frequenza di 32768 (ovvero 2^{15}) Hz e permette una misurazione del tempo molto più precisa.
Nel 2005 venne poi introdotto l'HPET (High Precision Event Timer), che, come suggerisce il nome, è dotato di un'alta precisione, nonché di un gran numero di comparatori.

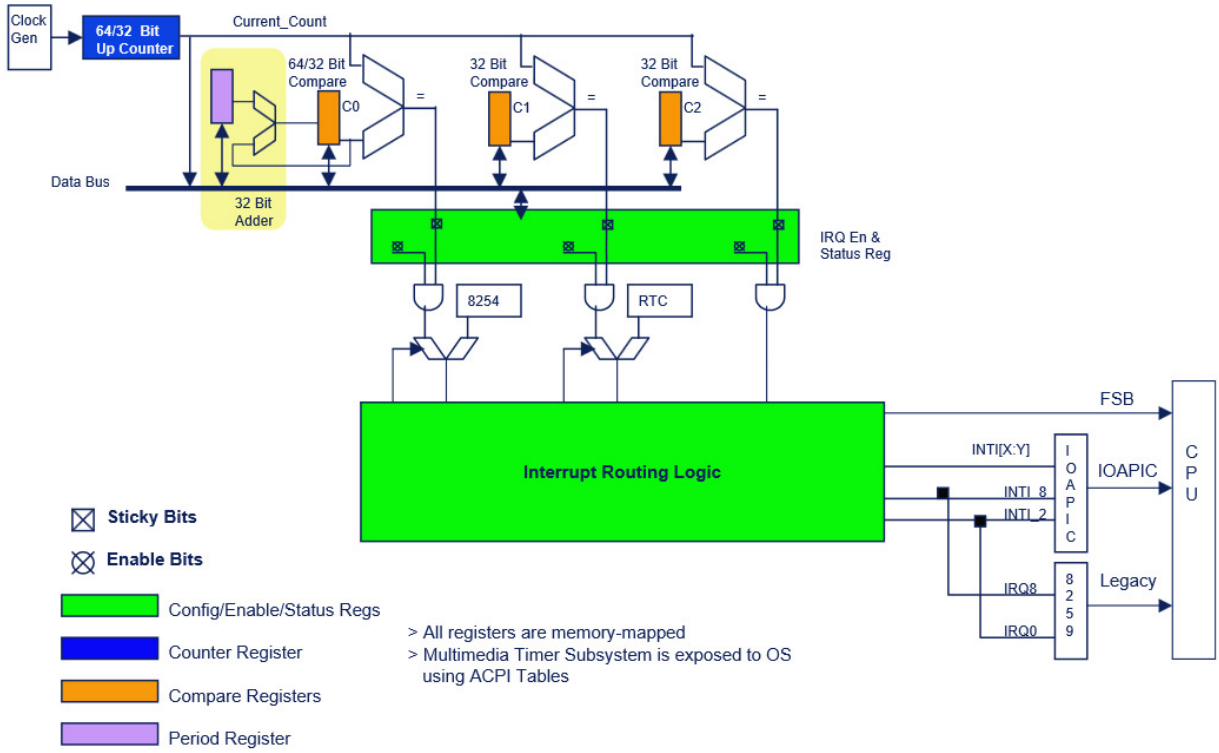

Figura 2.3 Struttura dell'HPET

2.4 Un primo approccio possibile

Già dopo queste breve panoramica abbiamo pensato ad un possibile approccio al problema riguardante la modifica del clock di un sistema e, benché tale metodo risulti poco versatile, se non quasi inutile, riteniamo valga comunque la pena spiegarlo brevemente.

Come abbiamo spiegato è il kernel che si occupa di settare questi comparatori che regolano il funzionamento del clock e, per farlo, utilizza un valore prestabilito (variabile in base alla tipologia e versione del kernel ed al tipo di meccanismo hardware supportato). Nel caso di codice open source, un possibile approccio potrebbe consistere nella modifica di tale valore: moltiplicandolo infatti per 10 potremo ottenere un sistema circa 10 volte

più lento. Ciò avviene perché il sistema aspetterà 10 volte il numero originario di interrupts prima di stabilire che è trascorso 1 secondo.

Questo metodo andrebbe testato con attenzione, ma non ci siamo soffermati sul problema in quanto risulterebbe in ogni caso poco pratico: infatti ogni volta che il suddetto valore viene modificato bisogna ricompilare l'intero kernel. È chiaro che una simile soluzione è assolutamente inefficiente, sia in termini di tempo impiegato per ogni nuova modifica, sia in termini di portabilità. Inoltre va sottolineato che, in questo modo, la modifica al clock è completamente di tipo statico.

L'unico vantaggio consisterebbe nel poter evitare l'utilizzo di una macchina virtuale, ma gli aspetti negativi di tale approccio sono in ogni caso talmente evidenti da rendere questo vantaggio insignificante.

Capitolo 3

Virtualbox e documentazione

In questo capitolo faremo una breve introduzione su Virtualbox, seguita poi dai primi passi dello svolgimento del progetto (basati sulla documentazione del suddetto software).
Come già annunciato, per il nostro lavoro abbiamo utilizzato Virtualbox; tale programma emula un hardware e quindi anche la componente clock. Questo fa proprio al caso nostro, in quanto il clock è virtuale e quindi modificabile. Esistono diversi software che svolgono le stesse funzioni di Virtualbox, ma la nostra scelta è ricaduta su quest'ultimo per diversi motivi: innanzitutto è un programma open source (caratteristica fondamentale per il nostro scopo); può essere installato su tutti i sistemi operativi più diffusi (il che lo rende uno dei software di virtualizzazione più utilizzati e supportati) ed è scritto in un linguaggio di programmazione molto potente. Per i dettagli, si rimanda al paragrafo successivo.

3.1 Virtualbox

Virtualbox è un software per l'esecuzione di macchine virtuali, sviluppato da Oracle Corporation, caratterizzato da una licenza di tipo open source e scritto con il linguaggio di programmazione C++.
Può essere installato ed utilizzato da tutti i sistemi operativi più diffusi: Microsoft Windows in tutte le sue versioni, i sistemi GNU/Linux e Mac OS X. In tale contesto questi sistemi vengono chiamati "host". Il sistema operativo che invece viene installato ed

eseguito da Virtualbox viene chiamato "guest", ed anche in questo caso la scelta possibile è davvero molto ampia: praticamente tutti i sistemi operativi più diffusi, tranne Mac OS X. Come già accennato Virtualbox emula parte dell'hardware sopra il quale viene eseguito il sistema "guest".

Riguardo all'organizzazione, uno dei concetti fondamentali di Virtualbox è la modularità: infatti, come illustra la seguente figura, il programma è suddiviso in diversi strati[3]:

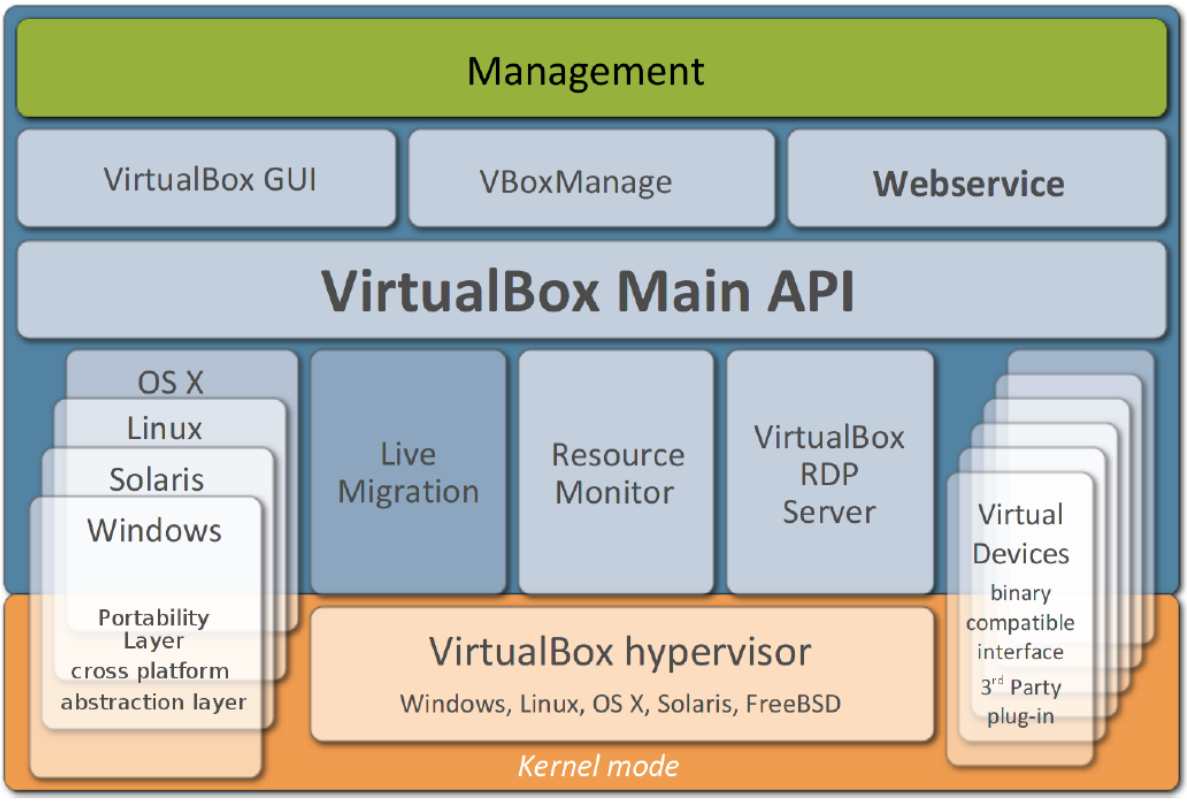

Figura 3.1 Suddivisione a strati di Virtualbox

Nella figura sovrastante, la parte arancione mostra il codice eseguito in kernel mode (chiamato anche Ring 0), mentre la parte in blu rappresenta il codice eseguito in user mode (chiamato anche Ring 3).

Partendo dal basso, il Virtualbox hypervisor rappresenta il nucleo del motore di virtualizzazione, che si occupa del controllo dell'esecuzione delle macchine virtuali, assicurandosi inoltre che non si verifichino conflitti di nessun genere, sia tra le macchine virtuali stesse, sia tra Virtualbox e il lavoro che sta svolgendo l'host. Proseguendo verso l'alto, le tre categorie centrali (Live Migration, Resource Monitor, RDP Server) rappresentano moduli per funzionalità extra, che non sono di interesse per quanto riguarda il nostro studio.
A sinistra ci sono i possibili sistemi host mentre a destra, come suggerisce il nome, ci sono i dispositivi virtuali, tra cui il clock.
In cima abbiamo la Main API con la quale si possono creare, configurare, avviare, stoppare ed eliminare macchine virtuali e molto altro; viene infatti fornito anche un SDK (Software Development Kit), la cui documentazione espone le varie interfacce che principalmente lavorano con la Main API, la quale può essere utilizzata tramite due servizi di comunicazione: un web service (mostrato nella figura, in alto a destra), che funziona come un server http, oppure utilizzando COM (Component Object model), un'interfaccia per componenti software introdotto da Microsoft. Per i sistemi host che non supportano COM esiste un'implementazione free, chiamata XPCOM, che funziona nello stesso modo ed è stata sviluppata da Mozilla.
Infine compaiono VBoxManage e VirtualBox GUI: il primo è uno strumento sempre di controllo, che funziona da riga di comando e come vedremo più avanti è lo strumento che useremo; il secondo, invece, come suggerisce la sigla lavora tramite interfaccia grafica.

3.2 Emulazione e simulazione

Come detto in precedenza Virtualbox emula l'hardware sottostante al sistema guest. Spesso si può fare confusione tra il concetto di emulazione e simulazione: abbiamo quindi deciso di spiegare, molto brevemente, la differenza tra i due, almeno per quanto riguarda il contesto informatico.

Emulatore: programma che permette l'esecuzione di software originariamente scritto per un ambiente (hardware o software) diverso da quello sul quale l'emulatore viene eseguito[4]. Un noto esempio è DOSBox, un programma che emula il vecchio sistema operativo MS DOS permettendo quindi di eseguire software che non potrebbe girare sui sistemi moderni.

Simulatore: programma che si prefigge di arrivare allo stesso risultato, riscrivendo però (in tutto o in parte) la routine del programma da simulare, in modo da renderlo comprensibile alla macchina su cui viene eseguito. Quindi la simulazione propone un modello della realtà. Un simulatore è più veloce di un emulatore, ma è meno accurato.

Alcuni esempio sono i simulatori di volo o di guida, ovvero programmi che ripropongono il comportamento di un veicolo reale, ma in modo totalmente virtuale.

3.3 Guest Additions

I Guest Additions sono sostanzialmente dei drivers che servono per il corretto e performante funzionamento del sistema operativo guest; vanno infatti installati all'interno del sistema guest e sono ottenibili tramite l'interfaccia esterna di Virtualbox. Le principali caratteristiche comprendono:

- integrazione del puntatore del mouse, nel passaggio tra host e guest
- condivisione di cartelle tra host e guest
- un miglior supporto per la scheda grafica virtuale
- condivisione della clipboard tra host e guest
- sincronizzazione del tempo

L'ultimo punto è di nostro grande interesse. Infatti la documentazione spiega che per

diversi motivi il tempo all'interno del sistema guest non sempre scorre (quindi viene calcolato) esattamente come nel sistema host. Per questo, tramite un meccanismo di sincronizzazione apposito, avviene un graduale aggiustamento del tempo, eseguendo frequentemente un confronto tra il tempo guest e quello host; quando la differenza tra i due è troppo grande (il caso più comune si verifica quando una macchina virtuale viene messa in pausa e ne viene poi ripristinato lo stato), il tempo viene immediatamente regolato, senza un aggiustamento graduale.
Più avanti vedremo come e perché questo servizio è legato al nostro problema.

3.4 Documentazione

Come già accennato in precedenza, uno dei primi passi si è basato sulla lettura della documentazione ufficiale di Virtualbox.
Abbiamo qui trovato un punto di partenza molto interessante: nel capitolo 9, la sottosezione intitolata “Accelerate or slow down the guest clock” spiega che è possibile modificare il clock (benché solo in modo statico, come vedremo tra poco).
Questo è possibile grazie ad uno strumento fornito nativamente con il codice sorgente: `VBoxManage`, strumento che avevamo introdotto nella sezione 3.1.
`VBoxManage` funziona tramite linea di comando e permette moltissime tipologie di modifica; quella di nostro interesse consiste nel seguente comando[5]:

```
VBoxManage setextradata "VM" "VBoxInternal/TM/WarpDrivePercentage" X
```

dove `"VM"` indica il nome di una macchina virtuale esistente ed `"X"` indica il valore del clock che si vuole utilizzare, dove 100 corrisponde alla velocità normale, 50 alla metà della velocità, 200 al doppio della velocità e così via. Il valore dev'essere compreso tra 2 e 20000.
La documentazione parla anche di un possibile problema di sincronizzazione quando il

valore del clock viene cambiato: come abbiamo accennato prima, Virtualbox è infatti configurato in modo che, quando i Guest Additions sono attivi (ovvero nella maggior parte dei casi), vi è un meccanismo di sincronizzazione che interroga spesso l'orologio dell'host per regolare quello guest.
Nel caso in cui si imposti un valore del clock diverso da quello di default (quindi diverso da 100), viene consigliato di disabilitare il meccanismo di cui sopra. Per farlo si può eseguire il seguente comando:

```
VBoxManage setextradata "VM" "VBoxInternal/Devices/VMMDev/0/Config/
GetHostTimeDisabled" 1
```

In questo modo si potranno evitare i problemi legati alla sincronizzazione tra host e guest.

Per ogni macchina virtuale installata Virtualbox crea una cartella specifica all'interno della quale viene generato un file di configurazione xml (nelle recenti versioni di Virtualbox l'estensione sarà .vbox e non più .xml, ma la struttura interna del file rimane invariata). Questo file è di nostro interesse poiché, quando viene lanciato il comando `setextradata`, il valore specificato con `X` viene salvato in questo file xml.

Tale valore viene poi letto da Virtualbox soltanto in due casi:

1. Quando viene avviata la macchina virtuale
2. Quando viene ripresa l'esecuzione di una macchina virtuale precedentemente sospesa tramite l'apposta funzione “Save the machine state” che, come suggerisce il nome, permette di salvare lo stato attuale senza dover spegnere il sistema operativo guest.

In entrambi i casi Virtualbox, dopo aver letto tale valore dall'xml, lo assegna ad una variabile chiamata `u32VirtualWarpDrivePercentage`. Da quel momento in poi, ogni volta che la macchina deve utilizzare il valore del clock, lo farà tramite questa variabile ed il file xml verrà quindi ignorato. Proprio per questo motivo il metodo di variazione del clock fornito nativamente da Virtualbox (tramite `setextradata`) è stato da noi definito come "statico": mentre una macchina è in esecuzione è possibile lanciare il comando `setextradata` senza riscontare alcun errore, ma il valore desiderato non verrà utilizzato dalla macchina fino al verificarsi di uno dei due casi di cui sopra. **Di fatto questo metodo non permette una modifica del clock a runtime**.
In ogni caso, tutto questo ci ha permesso di stabilire il vero e proprio punto di partenza per quanto riguardo l'analisi del codice sorgente: la variabile che si occupa di stabilire la velocità del clock, ovvero `u32VirtualWarpDrivePercentage`. Per i dettagli si rimanda al capitolo seguente.

Va comunque menzionato il file `src/Vbox/Main/xml/Settings.cpp` all'interno del quale vi è una funzione chiamata `readExtraData`: tale funzione, implementata alla riga 484, serve per leggere il file xml filtrando i campi extradata (nel caso di nostro interesse serve per leggere il valore `VirtualWarpPercentage` dal file .vbox dalla macchina virtuale).

3.5 Orologi in Virtualbox

Di seguito vengono elencati i quattro differenti orologi presenti in Virtualbox; le descrizioni qui riportate sono stare prese direttamente dai commenti ufficiali presenti nel file `/src/Vbox/VMM/VMMR3/TM.cpp` che, come vedremo, è uno dei principali moduli di nostro interesse.

- **Synchronous Virtual (guest)**: è allineato con l'orologio Virtual, ma tiene conto del ritardo causato dallo scheduling dell'host. Quando è troppo in ritardo viene velocizzato per raggiungere il valore dell'orologio Virtual. Tutti i dispositivi che implementano temporizzazione e che sono utilizzati dal sistema guest fanno riferimento a questo orologio (compresi utilizzi di timer, ecc).
  Questo è quindi l'orologio di nostro maggior interesse.

- **Virtual (guest)**: è implementato come un offset ad un wall-clock ad alta risoluzione (per wall-clock si intende un orologio normale, ovvero che calcola il tempo con la percezione umana).
  Usa la funzione `RTTimeNanoTS` (funzione che restituisce il corrente timestamp in nanosecondi) basata su GIP (Global Info Pages). Ciò che ne risulta è un orologio piuttosto preciso che funziona in tutti i contesti e su tutti gli host. L'orologio virtuale è in pausa quando la VM non è in stato di esecuzione.

- **CPU tick – TSC (guest)**: virtualizzato in funzione del Synchronous Clock, dove la frequenza di default è quella della frequenza della cpu dell'host. In questo modo è possibile regolare la frequenza. Un'altra opzione (non-standard) è quella di usare i valori non modificati del TSC dell'host.

- **Real clock (host)**: L'orologio dell'host, a bassa risoluzione. Non ci sono servizi che usano il real clock quando la VM non è in esecuzione.

Capitolo 4

Analisi del codice sorgente

In questo capitolo si svolgerà la vera e propria ricerca riguardo al clock di Virtualbox: verranno infatti analizzate diverse parti del codice sorgente che utilizzano o gestiscono il clock, per capire dove si potrebbe intervenire per ottenere una possibile modifica a runtime. Durante la ricerca sono state analizzate altre sezioni di codice che qui non verranno riportate, in quanto si sono dimostrate non essere propriamente inerenti allo scopo di questo lavoro.
Un primo dettaglio che va subito specificato è che Virtualbox, quando una macchina virtuale è in esecuzione, organizza tutta la sua configurazione con una struttura al albero.
Inoltre ci sono due strutture chiave che vanno assolutamente esposte:

1. `struct VM`:
   percorso: `/include/Vbox/vmm/vm.h`.

   si tratta della struttura principale, che contiene tutte le variabili e a sua volta altre strutture che compongono tutta la configurazione della macchina virtuale in esecuzione.

2. `struct TM`:
   percorso: `/src/Vbox/VMM/include/TMInternal.h`.

   è una delle strutture contenute in quella principale (la `VM`). La sigla `TM` significa

Time Manager ed infatti all'interno di questa struttura sono contenute tutte le variabili che si occupano della gestione del tempo: per questo è di nostro interesse.

Prima di proseguire, un'ultima nota: poiché non è possibile utilizzare lo standard output da Virtualbox mentre una macchina virtuale è in esecuzione, abbiamo utilizzato il file di log per verificare, ad esempio, le effettive chiamate di alcune funzioni o i valori assegnati a certe variabili in precisi istanti dell'esecuzione. Esistono infatti alcune funzioni fornite da Virtualbox che permettono di stampare stringhe e valori nel file di log (in particolare è stata utilizzata la funzione `LogRel`).

4.1 Le variabili del clock

Le variabili di nostro interesse sono tutte dichiarate come campi della struttura `TM`:

- **`u32VirtualWarpDrivePercentage`**: come già accennato tale variabile si occupa di stabilire la velocità del clock del sistema guest che viene eseguito su Virtualbox ed è di tipo unsigned int.

- **`fVirtualWarpDrive`**: di tipo booleano, viene utilizzata per indicare quando il clock ha un valore diverso da quello di default (ovvero diverso da 100). Come vedremo più avanti, infatti, ci sono delle funzioni per la gestione del tempo scritte appositamente per i casi in cui la variabile del clock è diversa dal valore di default.

- **`u64VirtualWarpDriveStart`**: variabile che viene utilizzata per memorizzare l'istante temporale in cui Virtualbox inizia a tenere traccia del tempo. Viene utilizzata ogni volta che dev'essere calcolato il tempo, quindi nell'ordine di migliaia di volte ogni secondo.

- **`u64VirtualOffset`**: come suggerisce il nome, viene utilizzato come offset

rispetto alla sorgente temporale. Nel caso di un clock diverso da 100 anche tale variabile viene utilizzata ogni volta che dev'essere calcolato il tempo trascorso.

4.2 File e funzioni legate al clock

In questo paragrafo ci occuperemo di analizzare i diversi files .cpp che utilizzano le variabili del clock (tramite funzioni). Prima di proseguire ci soffermiamo un momento per sottolineare un aspetto dell'organizzazione di tali files. Come abbiamo detto in precedenza, Virtualbox si divide principalmente in due differenti livelli: ring 0 (kernel mode) e ring 3 (user mode).
Alcuni dei files che analizzeremo si trovano nella directory `/src/Vbox/VMM/VMMR3/`, dove la sottostringa `"R3"` indica che tutti i file contenuti fanno parte del ring 3 e quindi sono modificabili più facilmente. I files che invece si trovano nella directory `/src/Vbox/VMM/VMMR0` fanno parte del ring 0, mentre i files all'interno del percorso `/src/Vbox/VMM/VMMAll` fanno parte di entrambi. Infatti in questo caso vengono compilati più volte, e, come vedremo, hanno le stesse limitazioni dei files appartenenti al ring 0.

4.2.1 TM.cpp

percorso:

`/src/Vbox/VMM/VMMR3/`

funzioni:

- **`TMR3Init:`** è la funzione che si occupa dell'inizializzazione di tutto il comparto che gestirà il tempo. Viene infatti chiamata una sola volta, quando viene avviata

una macchina virtuale. Uno dei compiti di nostro interesse, svolto da tale funzione, è l'assegnazione del valore del clock: alla riga 511 viene chiamata la seguente funzione :

- **`CFGMR3QueryU32:`** presi in input un nodo della macchina (come abbiamo già detto, tutto è organizzato con struttura ad albero), un valore da cercare ed una variabile in cui salvarlo, tramite una serie di altre funzioni esegue la ricerca e setta il valore desiderato. Nel nostro caso è questa la funzione che si occupa di assegnare il valore a `u32VirtualWarpDrivePercentage`. Come abbiamo detto nella sezione 3.4, il valore assegnato alla variabile del clock viene preso da un file di configurazione xml; entrando nel dettaglio, dobbiamo qui specificare che, quando viene letto il file xml, tale valore viene copiato in un file di configurazione interno al quale poi si accede tramite la struttura ad albero. Non vi è quindi un settaggio diretto dal file xml.

Una volta eseguita l'assegnazione e controllati eventuali errori, nel caso di un valore del clock diverso da zero, viene opportunamente cambiato il valore booleano di `fVirtualWarpDrive` da false a true.

- **`tmR3SetWarpDrive:`** come suggerisce il nome è una funzione che serve per fare un settaggio sul `WarpDrive`, ovvero il clock; viene però chiamata solo in due casi: ovvero tramite l'eseguibile `tstAnimate` oppure tramite `VBoxSDL` .Entrambi sono programmi che vengono utilizzati al di fuori dell'esecuzione base della macchina e, come viene ripetuto più volte nella documentazione, gli strumenti forniti da Virtualbox che eseguono modifiche alla configurazione della macchina richiedono una sospensione e quindi, di fatto, si tratterebbe nuovamente di modifiche statiche.

- **`tmR3InfoClocks`**: ultima funzione all'interno di `TM.cpp` che utilizza la variabile

del clock. In questo caso si tratta di una semplice stampa a video del valore corrente.

4.2.2 TMAllVirtual.cpp

percorso:

`/src/Vbox/VMM/VMMAll/`

funzioni:

- **`TMVirtualGet`**: usata per ottenere il valore corrente del Virtual Clock. Chiama direttamente **`tmVirtualGet`**, la quale fa un controllo su eventuali timers in uso e alla fine restituisce il timestamp servendosi della seguente funzione:

    - **`tmVirtualGetRaw`**: nel caso in cui il clock sia al 100% chiama direttamente la **`tmVirtualGetRawNanoTS`** (spiegata più in basso). Altrimenti chiama la seguente funzione:

        - **`tmVirtualGetRawNonNormal`**: chiamata solo quando il clock è stato settato con un valore diverso da 100, tale funzione calcola il tempo trascorso e restituisce il timestamp espresso in nanosecondi. Viene chiamata circa 9000 volte ogni secondo.
          Per fare i calcoli usa le variabili elencate nella sezione 4.1, ma il primo compito che svolge consiste nel chiamare la seguente funzione:

            - **`tmVirtualGetRawNanoTS`**: funzione che restituisce il timestamp del tempo reale, ovvero corrispondente a quando il clock lavora al 100%. Il valore di ritorno è espresso in

nanosecondi e non supera mai 4,300,000,000.
Una volta ottenuto il timetsamp del tempo reale calcola la variazione del tempo data dall'eventuale clock modificato e lo restituisce (sempre in nanosecondi).

- **`TMVirtualSyncGet`**: usata per ottenere il valore corrente del Synchronous Virtual Clock; chiama direttamente **`TMVirtualSyncGetEx`**.

- **`tmVirtualResumeLocked`**: questa funzione viene chiamata solamente quando una macchina virtuale viene avviata o ne viene ripresa l'esecuzione da una precedente sospensione. Il suo compito è quello di assegnare il valore iniziale a due delle variabili che gestiscono il calcolo del clock (introdotte nella sezione 4.1), ovvero `u64VirtualWarpDriveStart` e `u64VirtualOffSet`.

4.2.3 TMAll.cpp

percorso:

`/src/Vbox/VMM/VMMAll/`

funzioni:

- **`tmTimerPollReturnMiss`**: questa funzione calcola il tempo dei timers sulla base dell'aggiustamento dato dal settaggio del clock. Anch'essa viene chiamata migliaia di volte ogni secondo e per eseguire i vari calcoli utilizza le variabili che regolano il funzionamento del clock.

4.2.4 timesup.cpp

percorso:

`/src/Vbox/Runtime/common/time/`

funzioni:

- **`RTTimeNanoTS`**: funzione che restituisce il corrente timestamp in millisecondi. Richiama:
    - **`rTimeNanoTSInternal`**: richiama a sua volta la seguente funzione:

        - **`RTTimeSystemNanoTS`**: questa funzione è definita in diversi file e viene scelta in base al sistema installato. Nel nostro caso (Linux) la funzione richiamata è quella definita nel file `/src/Vbox/Runtime/r3/linux/time-linux.cpp`. Nel caso di Windows verrebbe chiamata la versione definita in `../r3/win/time-win.cpp` e così via. Ognuna di queste funzioni chiama infine la propria versione di:

            - **`rtTimeGetSystemNanoTS`**: di particolare importanza in quanto, tra i vari calcoli che esegue, utilizza la funzione di sistema **`gettimeofday`**, funzione della quale abbiamo già parlato nella sezione 2.2.

Capitolo 5

Lo scenario

In questo capitolo parleremo dell'obiettivo finale di questo studio.

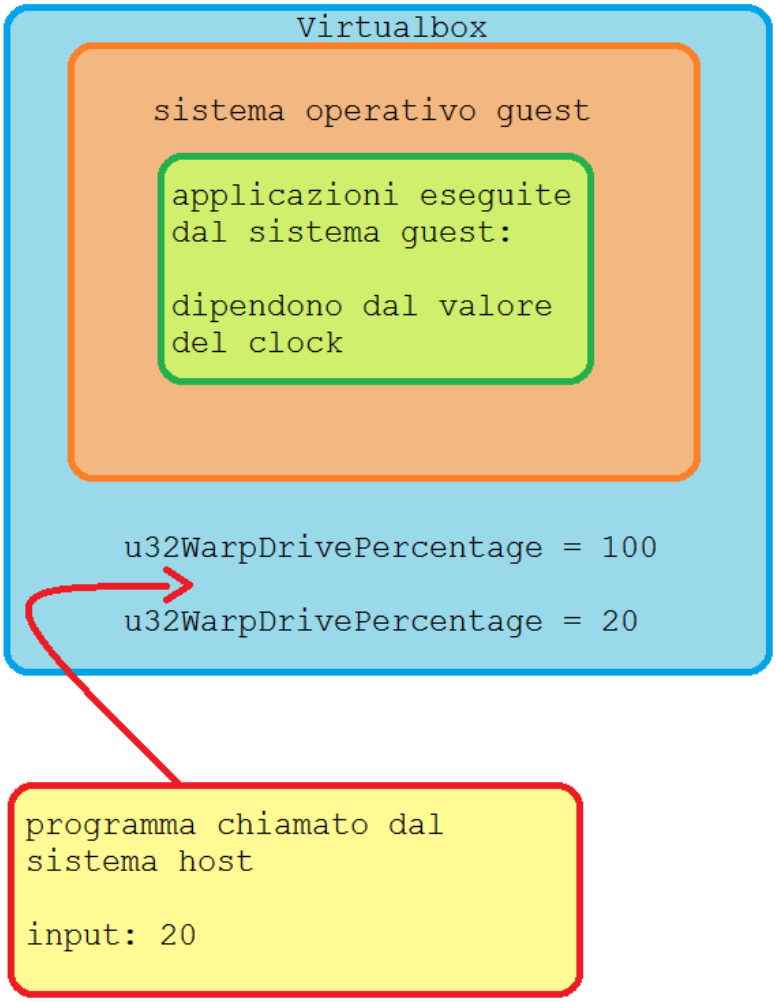

Figura 5 - Lo scenario

Analizziamo ora la precedente figura che mostra nel dettaglio il nostro scopo: come avevamo già anticipato, vogliamo trovare un modo per modificare il clock di Virtualbox dinamicamente.
La sezione azzurra rappresenta Virtualbox, che esegue una macchina virtuale con il proprio sistema operativo guest, indicato in arancione. Al suo interno, in verde, è rappresentata una qualsiasi applicazione eseguita dal sistema guest.
In basso, nella figura 5, la sezione in giallo rappresenta un nostro programma, che viene chiamato dal sistema host, quindi completamente all'esterno di Virtualbox. Tale eseguibile permette di inserire in input un valore, il quale verrà passato alla struttura interna di Virtualbox: tale valore verrà assegnato alla variabile che si occupa di gestire il clock, ovvero la già menzionata `u32VirtualWarpDrivePercentage`.
Nell'esempio rappresentato nella figura 5, la macchina virtuale è stata avviata con valore del clock iniziale standard, cioè 100. In un momento qualsiasi possiamo eseguire il nostro programma con input 20 e quest'ultimo valore andrà a influire sul tempo che scorre all'interno della macchina virtuale (in questo caso il tempo passerà a circa 1/5 della velocità del normale flusso).

5.1 Un primo approccio dinamico

Sin dall'inizio abbiamo considerato un possibile problema legato alla sincronizzazione del tempo che potrebbe verificarsi modificando il clock dinamicamente, in quanto Virtualbox non prevede questa possibilità e potrebbero esserci delle variabili che vanno modificate (oltre a quella di velocità del clock). Prima di analizzare questo problema abbiamo voluto fare un primo tentativo di modifica dinamica direttamente dall'interno del codice, cambiando solamente il valore associato a `u32VirtualWarpDrivePercentage` senza preoccuparci di altre questioni.

5.1.1 Modifica dinamica dall'interno

Tra le varie funzioni che utilizzano `u32VirtualWarpDrivePercentage` abbiamo scelto `tmTimerPollReturnMiss`, la quale viene costantemente richiamata durante tutta l'esecuzione della macchina virtuale, nell'ordine di centinaia di volte ogni secondo.
Modificando opportunamente il codice della funzione abbiamo fatto in modo che circa ogni 3 secondi la variabile del clock venisse decrementata di 1 in modo da realizzare una modifica dinamica senza necessità di stoppare/sospendere la macchina in esecuzione.

5.1.2 Test e risultati

Oltre a controllare lo scorrere dell'orologio all'interno del sistema guest (prestando particolare attenzione ai secondi) abbiamo scritto un semplice programma C che esegue, con un ciclo infinito, una `printf` seguita da una `sleep(1)`. Eseguendo tale programma abbiamo potuto verificare che la frequenza di stampa della `printf` diminuisce progressivamente, come ci aspettavamo. È stato poi eseguito anche un test opposto (velocizzando il clock).

5.2 Verso una soluzione concreta

Dopo aver mostrato in modo dettagliato lo scenario rappresentante il nostro obiettivo finale, concludiamo questo capitolo anticipando che lo scopo è stato raggiunto: come vedremo infatti nel capitolo 7 abbiamo modificato una parte già esistente di un front-end di Virtualbox, in modo da poter richiamare una funzione interna, passando in input il valore che andrà a modificare la velocità del clock.
Come abbiamo accennato, nel caso di una modifica dinamica potrebbero esserci dei problemi legati alla sincronizzazione del tempo. Il seguente capitolo analizzerà nel dettaglio questa potenziale complicazione.

Capitolo 6

Il problema del tempo: la sincronizzazione

In questa sezione faremo un'analisi di un possibile problema legato alla sincronizzazione del tempo: Virtualbox infatti non prevede la possibilità di cambiare la velocità del clock durante l'esecuzione di una macchina virtuale. Questo potrebbe risultare problematico in quanto ci sono alcuni valori che vengono settati solo una volta in fase di inizializzazione e vengono poi usati per calcolare l'avanzamento del tempo. Come vedremo potrebbe verificarsi il caso in cui uno (o più) degli orologi utilizzati da Virtualbox retroceda rispetto alla normale timeline nel caso in cui il clock venga rallentato.

Prendiamo in considerazione la seguente formula generale per calcolare un certo istante di tempo:

$$\mathrm{t} = t_0 + \frac{(t_x - t_0) * Clock}{100}$$

dove:

- t = istante di tempo da trovare
- t_0 = istante iniziale; tale valore rimane invariato
- t_x = istante che varia in base alla *t* da trovare
- *Clock* = corrisponde al valore di percentuale del clock

6.1 Caso di default

Questo è il caso in cui il clock non viene modificato. Può avere una valore diverso da 100 utilizzando il metodo statico fornito da Virtualbox (quello di cui abbiamo discusso nella sezione 3.4). L'importante è che il valore non venga cambiato a runtime. Questo è appunto il caso di default, che funziona perfettamente con qualsiasi valore iniziale del clock in quanto si tratta di un caso previsto da Virtualbox e quindi opportunamente gestito.

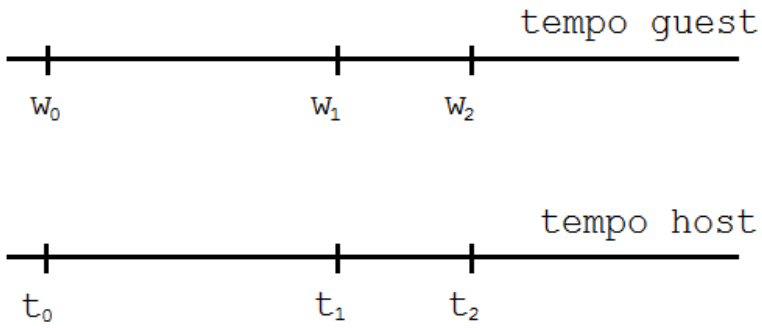

Figura 6.1 Timeline di default

La figura 6.1 illustra la situazione che si verifica nel caso di default; sono presenti due timeline, ovvero linee che rappresentano lo scorrere del tempo. Leggendo da sinistra a destra i vari istanti sono posti in ordine cronologico.
La linea sottostante rappresenta la timeline del tempo che scorre nel sistema host, mentre l'altra rappresenta il tempo nel sistema guest.
Per mostrare un esempio concreto assegniamo dei valori ai vari istanti temporali:

- *t0* = 400
- *t1* = 600
- *t2* = 700

Applichiamo poi la formula illustrata prima, per calcolare ad esempio *w2*:

$$w2 = 400 + \frac{(700 - 400) * 100}{100}$$

È facilmente calcolabile che w2 vale 700 e corrisponde quindi a t2 come ci si poteva aspettare analizzando la figura.
Nel caso in cui il valore del clock fosse stato 50 il calcolo sarebbe stato:

$$w2 = 400 + \frac{(700 - 400) * 50}{100}$$

ed il valore di *w2* sarebbe stato 550; chiaramente, rallentando il clock nel sistema guest, il tempo che ne viene percepito all'interno trascorre più lentamente rispetto al tempo reale, ovvero quello del sistema host.
L'importante è che ogni istante temporale *w(x)* segua sempre l'istante *w(x-1)*: in altre parole non deve mai accadere che il calcolo di un istante temporale dia come risultato un valore che sia minore dell'istante precedentemente calcolato. Qui sotto riportiamo il caso in cui la macchina guest sia stata avviata con un valore del clock inferiore a 100:

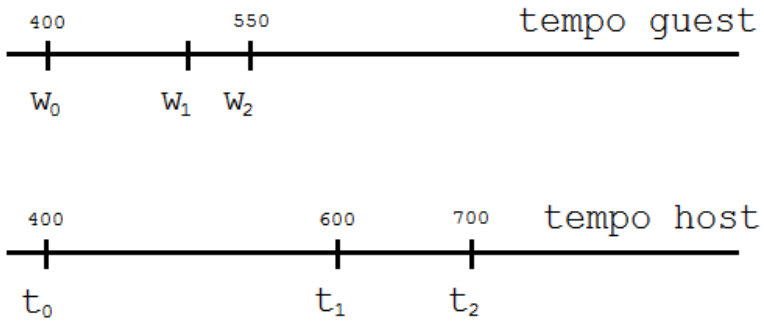

Figura 6.1-b Timeline con clock <100

Come illustra la figura 6.1-b il tempo nella timeline superiore passa più lentamente, ma ogni istante è il successivo di quello precedente, rispettando quindi le condizioni di correttezza.

6.2 Caso clock rallentato a runtime

Ora analizzeremo il caso più delicato, ovvero quello in cui la velocità del clock viene rallentata a runtime.
Per la timeline del sistema host utilizziamo sempre i valori di prima, ovvero:

- *t0* = 400
- *t1* = 600
- *t2* = 700

Poniamo di aver avviato la macchina guest con una velocità di clock di 80 e calcoliamo l'istante *w1* utilizzando la solita formula:

$$w1 = 400 + \frac{(600 - 400) * 80}{100}$$

In questo caso *w1* vale 560.
Ora, prima di calcolare *w2*, poniamo che il clock venga rallentato a 40; procedendo poi applicando la formula per *w2* ed il nuovo valore del clock, avremmo una situazione del genere:

$$w2 = 400 + \frac{(700 - 400) * 40}{100}$$

Il risultato è che w2 vale 520, un valore inferiore all'istante temporale precedente. Una situazione di questo tipo può rivelarsi dannosa, in quanto il tempo calcolato nella macchina guest potrebbe virtualmente indietreggiare per poi magari tornare a scorrere normalmente in conseguenza di un aumento opportuno della velocità del clock.
Come mostreremo nel capitolo 8, abbiamo fatto alcune prove pratiche direttamente su Virtualbox per vedere se questo tipo di situazioni comporta effettivamente dei problemi.
Qui sotto viene illustrata la situazione appena descritta:

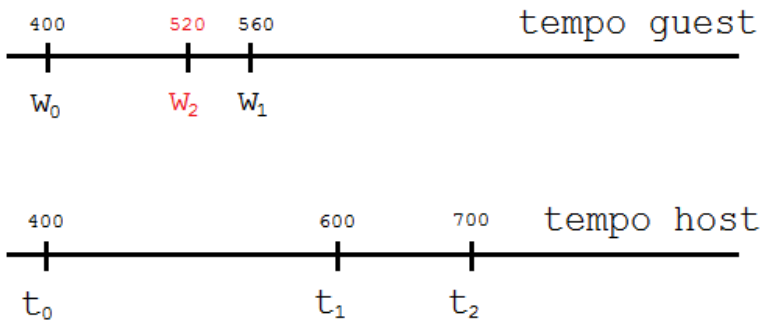

Figura 6.2 Timeline con clock modificato

6.3 Caso clock velocizzato a runtime

L'ultimo caso è quello in cui la velocità del clock viene aumentata rispetto a quella di partenza della macchina virtuale.
Questa volta non ci saranno problemi legati alla progressione degli istanti temporali in quando il tempo trascorrerà solamente più veloce lasciando intatta la corretta progressione del conteggio degli istanti temporali.

6.4 Conclusioni riguardo al tempo

Per quanto riguarda il livello teorico del conteggio del tempo, abbiamo mostrato i diversi casi possibili facendo alcune considerazioni e prestando particolare attenzione al caso potenzialmente più dannoso, ovvero al caso in cui il clock viene rallentato.

Dopo aver considerato ed esaminato tutti i problemi legati al tempo esposti in questo capitolo, abbiamo analizzato dove e come in Virtualbox si possano riscontrare queste problematiche (Capitolo 8). Come vedremo, però, la situazione è un po' diversa all'interno di Virtualbox in quanto esistono diversi orologi da considerare, ovvero quelli elencati e descritti nella sezione 3.5.

Capitolo 7

La modifica dinamica

Come già anticipato siamo riusciti a trovare un modo per cambiare la variabile del clock a runtime, specificando in input un parametro a scelta. Inoltre tale modifica potrà essere effettuata un numero infinito di volte sempre senza la necessità di stoppare la macchina virtuale. In questo capitolo spiegheremo nel dettaglio la progettazione di tale meccanismo e mostreremo come utilizzarlo.
Nella sezione 4.2 abbiamo esaminato le principali funzioni che utilizzano la variabile che si occupa di stabilire la velocità del clock e, come nel test effettuato nel paragrafo 5.1, abbiamo utilizzato anche la funzione `tmTimerPollReturnMiss` poiché viene chiamata molto frequentemente, e garantisce una modifica pressoché immediata, almeno per quanto riguarda la percezione umana.

7.1 Meccanismo di input/output

Dopo aver trovato un punto ideale per effettuare la modifica ci siamo posti il problema di come effettuare input/output, ovvero come leggere dal sistema host esterno un valore.
Una prima idea è stata quella di utilizzare la funzione di sistema `fopen`: tale funzione permette infatti di aprire un file e leggerne il contenuto. Sicuramente non sarebbe stato il modo migliore, poiché avremmo poi dovuto scrivere un secondo programma per modificare il file letto da `fopen`, a meno di non modificarlo ogni volta manualmente. Inoltre avremmo dovuto specificare manualmente il percorso del file da leggere, il che

avrebbe annullato un'eventuale portabilità di Virtualbox così modificato.
Un approccio di questo tipo sarebbe quindi stato corretto a livello funzionale, ma non molto adatto e sicuramente migliorabile.
Abbiamo tra l'altro scoperto che ci sono molti file, tra cui quello contenente la funzione `tmTimerPollReturnMiss (/src/Vbox/VMM/VMMAll/TMAll.cpp)` che non permettono di utilizzare librerie che non siano previste dal programma, come ad esempio la `<stdio.h>` (necessaria per il nostro tipo approccio).
Ci è stato infatti spiegato da uno dei membri dell'assistenza di Virtualbox, tramite il loro servizio di mailing list per gli sviluppatori, che alcuni file vengono compilati più volte ed una delle copie è destinata alla kernel mode, settore volutamente protetto. Ci è stato poi suggerito un metodo sicuramente più adatto allo scopo, ovvero utilizzare uno degli strumenti forniti nativamente da Virtualbox, in particolare quelli front-end, come il già menzionato `VBoxManage`.
Infine abbiamo concluso che per salvare eventuali valori di input possiamo sfruttare l'apposita struttura `VM`, visibile in qualsiasi punto del codice ed i cui valori interni possono essere facilmente letti.

7.2 Input tramite VBoXManage

Tra gli strumenti di front-end abbiamo analizzato `VBoXManage`, interfaccia da linea di comando che viene utilizzata anche per eseguire la modifica statica del clock (come visto nella sezione 3.4).
Questo eseguibile offre un gran numero di possibilità, tutte elencate nella documentazione ufficiale di Virtualbox ed inoltre per richiamare funzioni interne utilizza la tecnologia COM.
Proprio quest'ultima caratteristica ci ha permesso di sviluppare un metodo per passare un valore in input dall'host esterno, o per meglio dire, di aggiungere funzionalità ad un servizio già esistente. Sicuramente è possibile sviluppare ad hoc tutto un insieme di

funzioni nuove che utilizzano COM, ma per semplicità abbiamo intanto utilizzato quelle già esistenti.
Siamo partiti dal seguente comando:

```
VBoxManage debugvm "VM" info clock
```

Senza considerare momentaneamente `"VM"` che specifica quale macchina virtuale stiamo interrogando, il parametro `debugvm` seguito da `info` serve per richiamare una tra le funzioni interne specificata poi dall'ultimo parametro, ovvero nel nostro caso `clock`. Tramite quest'ultimo viene eseguita la funzione `tmR3InfoClock` che serve per visualizzare i valori attuali degli orologi di Virtualbox. Utilizzeremo questa opzione anche nel prossimo capitolo per fare un'analisi del tempo, ma quel che è di nostro interesse ora è che la funzione è predisposta per ricevere un parametro extra e facoltativo, che servirebbe per filtrare ulteriormente le informazioni.
Basandoci sulla struttura di questo sistema per richiamare funzioni interne tramite `VBoxManage`, abbiamo fatto alcune modifiche al codice sorgente ed infine aggiunto una nuova funzione, ottenendo un metodo per modificare il clock dinamicamente a tutti gli effetti. Nel seguente paragrafo verranno esposti tutti i dettagli a riguardo.

7.3 Il metodo finale

In questa sezione mostreremo nel dettaglio le modifiche effettuate al codice ed il funzionamento del metodo per cambiare la velocità del clock a runtime da noi sviluppato.
Modifiche al codice:

- all'interno della struttura `TM` in `/src/Vbox/VMM/include/TMInternal.h`. abbiamo aggiunto due variabili:

- `uint32_t` **`myCustom`**: variabile alla quale viene assegnato ogni volta il valore in input; tale valore viene poi assegnato alla variabile del clock.
- `bool` **`myCheckUpdate`**: di tipo booleano, serve per sapere quando è stato letto un nuovo valore in input e sia quindi necessario proseguire con la modifica del clock. In fase di inizializzazione è settata a `false`.

- all'interno del file `/src/Vbox/VMM/VMMR3/TM.cpp` abbiamo scritto una nuova funzione, `tmR3SetCustomClocks`, che si occupa di leggere il valore in input ed assegnarlo alla nuova variabile `myCustom` creata da noi. Inoltre gestisce i vari casi in cui l'utente inserisce un valore non valido.
  La funzione è dichiarata ed implementata nello stesso file, ed inoltre alla riga 769 abbiamo aggiunto un handler per il parametro `setclock` che viene poi utilizzato al posto di `clock` per dare l'input tramite `VBoxManage`.

```
3209 static uint32_t myReadValue;
3210
3211 static DECLCALLBACK(void) tmR3SetCustomClocks(PVM pVM, PCDBGFINFOHLP pHlp, const char *inputClock)
3212 {
3213
3214         /* se c'e' un input, interpreta la stringa come in intero */
3215         if (inputClock != NULL) {
3216
3217                 myReadValue = atoi(inputClock);
3218
3219                 if (myReadValue == 0) {
3220                         /* errore: é stata inserito un input di tipo non integer */
3221                         pHlp->pfnPrintf(pHlp, "Errore: é stato inserito un valore non intero\n");
3222                 }
3223                 else if ( myReadValue < 2 || myReadValue > 20000 ){
3224                         pHlp->pfnPrintf(pHlp, "Errore: input non valido\n");
3225                         pHlp->pfnPrintf(pHlp, "I valori ammessi sono compresi tra 2 e 20000\n");
3226                 }
3227                 else {
3228                         /* assegno il nuovo valore;poi verrá settato dalla tmTimerPollReturnMiss */
3229                         pVM->tm.s.myCustomWarp = myReadValue;
3230
3231                         /* true in modo che in tmTimerPollReturnMiss venga fatta l'assegnazione */
3232                         pVM->tm.s.myCheckUpdate = true;
3233                         pHlp->pfnPrintf(pHlp, "La nuova velocitá del clock é %u\n", myReadValue);
3234                 }
3235         }
3236         /* se non é stato inserito l'input*/
3237         else    {
3238                         pHlp->pfnPrintf(pHlp, "Input non inserito\n");
3239         }
3240 }
3241
```

Figura 7 - Screenshot della nostra funzione

- all'interno del file `/src/Vbox/VMM/VMMAll/TMAll.cpp` abbiamo aggiunto una parte di codice all'interno della funzione `tmTimerPollReturnMiss`:

```
647 DECLINLINE(uint64_t) tmTimerPollReturnMiss(PVM pVM, uint64_t u64Now, uint64_t u64Delta, uint6
648 {       /* LogRel(("joe: esegue tmTimerPoll\n")); */
649     Assert(!(u64Delta & RT_BIT_64(63)));
650
651
652 /*************************************************************************************************
653         joe: parte aggiunta da me, dove avviene l'effettiva modifica alla variabile del clock
654         in conseguenza in un cambiamente richiesto dall'esterno tramite
655         "VBoxManage debugvm VM_name info setclock valùe"¨
656 */
657
658         if (pVM->tm.s.myCheckUpdate) { /* controlla se bisogna eseguire una modifica */
659
660                 /* assegna il nuovo valore alla variabile del clock */
661                 pVM->tm.s.u32VirtualWarpDrivePercentage = pVM->tm.s.myCustomWarp;
662
663                 /* assegna il valore alla variabile booleana che indica se il clock é 100 */
664                 pVM->tm.s.fVirtualWarpDrive = pVM->tm.s.u32VirtualWarpDrivePercentage != 100;
665
666                 /* finito l'aggiornamento, rimette a false myCheckUpdate per evitare
667                         di eseguire codice inutile ogni volta che non serve*/
668                 pVM->tm.s.myCheckUpdate = false;
669         }
670
671 /*************************************************************************************************
672 */
673
```

Figura 7-b - Screenshot della nostra aggiunta a `tmTimerPollReturnMiss`

Dopo aver quindi mostrato nel dettaglio tutte le modifiche da noi effettuate, ricapitoliamo il funzionamento:

- aprire un terminale e posizionarsi nel percorso dove sono presenti gli eseguibili di Virtualbox, nel nostro caso: `./VBox4.3/out/linux.x86/release/bin` ed eseguire il seguente comando:

  ```
  ./VBoxManage debugvm "VM" info setclock "X"
  ```

  dove `"VM"` è il nome della macchina virtuale ed `"X"` è il valore che si vuole assegnare alla velocità del clock, ricordando che 100 è quella di default, 50 significa che il sistema guest lavorerà circa alla metà della velocità, 200 al doppio,

e così via. Ricordiamo inoltre che non sono ammessi valori inferiori a 2 o superiori a 20000 e che è consigliato disabilitare la sincronizzazione fornita dai Guest Additions (vedi sezione 3.4).
Una volta eseguito il primo passo, ecco cosa succede all'interno del codice:

- tramite l'eseguibile `VBoxManage` vengono richiamate una serie di funzioni che, in base ai parametri in input, seguono uno specifico percorso di handlers fino ad arrivare alla nostra funzione `tmR3SetCustomClocks` tramite il parametro `setclock`.

- `tmR3SetCustomClocks` legge l'ultimo parametro, ovvero `"X"`, e, nel caso in cui il valore sia legale, viene assegnato alla nostra variabile `myCustom`; infine l'altra nostra variabile, `myCheckUpdate`, viene settata a `true`.

- la prossima volta che `tmTimerPollReturnMiss` verrà richiamata dalla routine di esecuzione (come abbiamo già puntualizzato tale funzione viene chiamata migliaia di volte ogni secondo) il `true` assegnato a `myCheckUpdate` farà in modo che venga eseguito il codice aggiunto da noi (Figura 7-b):

    - a **`u32VirtualWarpDrivePercentage`** viene assegnato il nuovo valore di velocità del clock

    - **`fVirtualWarpDrive`** cambia valore di verità in base al nuovo valore

    - **`myCheckUpdate`** viene settata a `false` per evitare di eseguire il nostro codice fino a quando non verrà assegnato un nuovo valore al clock dall'esterno

Utilizzando questo metodo è possibile cambiare la velocità del clock a runtime e scegliere ogni volta il valore desiderato. Rimane ora solo da trattare il problema della sincronizzazione del tempo, già discusso in linea generale nel capitolo 6. Nel capitolo successivo vedremo quindi se, e come, questo problema si verifica all'interno di Virtualbox.

Capitolo 8

Il problema del tempo in Virtualbox

In questo capitolo analizzeremo il problema del tempo, protagonista del capitolo 6, e vedremo in che modo si manifesta in Virtualbox. In precedenza avevamo accennato al fatto che la presenza di più orologi in Virtualbox comporti sia svantaggi che vantaggi per quanto riguarda il nostro problema della sincronizzazione.

8.1 Conseguenze delle modifiche dinamiche

Dopo aver eseguito molteplici test abbiamo tratto la seguente conclusione: modificare dinamicamente, e più volte, la velocità del clock comporta le conseguenze qui elencate:

- cominciamo da quella più grave: il Virtual Clock perde completamente la temporizzazione e soprattutto viene colpito dal problema discusso nella sezione 6.2. Infatti, in caso di rallentamento dinamico, il Virtual Clock viene improvvisamente arretrato rispetto all'istante temporale precedente.

- un altro problema vede protagonisti sia il Virtual Clock sia il Synchronous Virtual Clock. Questi due orologi, infatti, sono gestiti in modo da essere sempre prossimi l'uno all'altro ed in particolare il Synchronous viene automaticamente velocizzato

quando è troppo arretrato rispetto al Virtual. Precisiamo che il valore dei due orologi non sarà mai esattamente uguale, ma si tratterà pur sempre di una differenza talmente minima da essere trascurabile.
Nel caso di una modifica dinamica al clock si perderà tale sincronizzazione e anche nel caso di accelerazione, quando il Virtual sarà molto più avanti del Synchronous, il meccanismo di "catch-up" per riavvicinare il Synchronous al Virtual non sarà sufficiente in quanto non è previsto un distacco simile.

- Il Synchronous, preso singolarmente, non subisce problemi di temporizzazione: semplicemente va più lento quando il clock viene rallentato e va invece più veloce quando il clock viene accelerato.

- Il Cpu Tick Timer, come il Synchronous, non subisce problemi di temporizzazione, se non, giustamente, il rallentamento o l'accelerazione per quanto riguarda la progressione del calcolo del tempo.

- Il Real Clock rimane completamente indifferente a qualsiasi cambiamento di velocità del clock .

Come spiegato nella sezione 3.5, tra questi orologi quello che in Virtualbox viene utilizzato per tutta le gestione del tempo, che comprende quindi l'utilizzo dei timer, il mantenimento dell'orologio di sistema, ecc, è proprio il Synchronous Clock.
Prima di mostrare il test pratico finale illustriamo di seguito l'analisi di quanto detto finora, utilizzando lo strumento `VBoxManage debugvm "VM" info clock`
che ci permette di visualizzare il valore corrente dei vari orologi.

8.2 Test: Cpu Tick e Real Clock

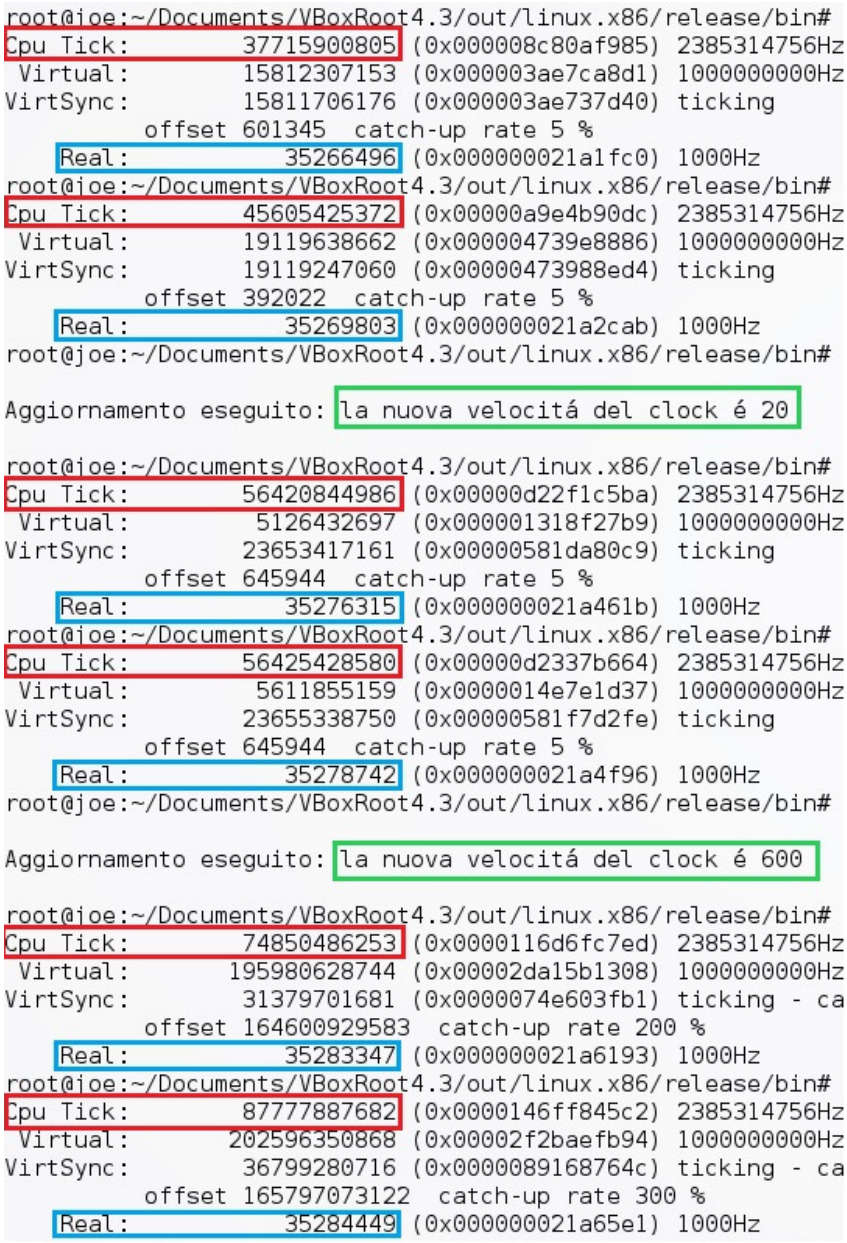

Figura 8 - Screenshot terminale per Cpu Tick e Real Clock

Facendo riferimento alla figura 8, abbiamo interrogato i diversi orologi più volte, anche in seguito ad un'improvvisa modifica al clock, prima rallentandolo, e poi velocizzandolo (i due distinti momenti sono evidenziati in verde). Come si può notare sia il Cpu Tick Clock

(in rosso) sia il Real Clock (in azzurro) non subiscono problemi di temporizzazione.

8.3 Test: Virtual e Synchronous Clock

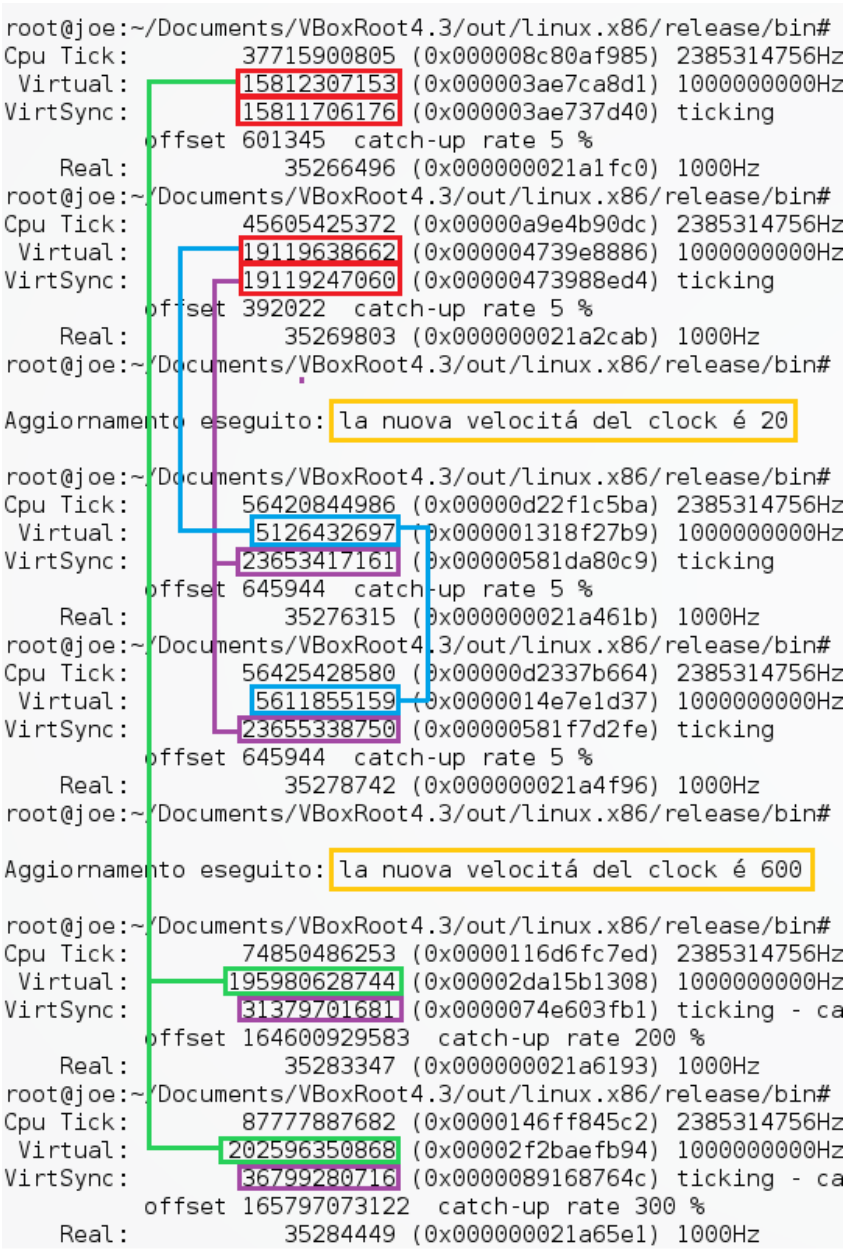

Figura 8-b - Screenshot terminale per Virtual e Synchronous clock

Ora analizziamo cosa succede al Virtual ed al Synchronous: innanzitutto va notato che all'inizio i valori di entrambi sono sempre simili (evidenziati in rosso); questa infatti è la situazione di default. Quando invece il clock viene improvvisamente rallentato (primo cambiamento evidenziata in giallo) si può vedere come il Synchronous continui ad avanzare (valori evidenziati in viola) mentre il Virtual è arretrato di molto (evidenziato in azzurro). Da notare che, come avevamo detto, il Synchronous avanza sempre, benché più lentamente (vedi i primi due valori in viola rispetto ai primi due in rosso riferiti al Synchronous). In questo caso si perde però la sincronizzazione tra Virtual e Synchronous, che non riesce ad essere corretta dal sistema di "catch up".
Analizziamo infine la situazione in cui improvvisamente il clock viene velocizzato (seconda parte evidenziata in giallo): il Virtual viene fortemente aumentato(valori in verde) tanto da superare il primo valore che avevamo ottenuto (il primo valore in rosso) ristabilendo almeno la temporizzazione. Confrontando però Virtual e Synchronous anche in questo caso non vi è alcuna sincronizzazione tra i due orologi.

8.4 Un test finale

Come conclusione del nostro lavoro abbiamo eseguito ulteriori prove. Come per il test della sezione 5.1, abbiamo scritto un piccolo programma all'interno del sistema guest, che esegue un ciclo infinto di `printf` seguite da una `sleep` opportunamente settata. Lo abbiamo avviato e nell'host, tramite l'interfaccia `VBoxManage` modificata da noi (vedi capitolo 8), abbiamo cambiato più volte la velocità del clock. Ad ogni modifica abbiamo ottenuto il risultato che ci aspettavamo: la sequenza di stampa rallentava/accelerava in base al valore da noi settato. Abbiamo inoltre tenuto sempre sott'occhio anche l'orologio del sistema guest per verificare che non arretrasse e possiamo affermare che **l'ora avanza sempre senza mai subire i problemi di temporizzazione.**

8.4.1 Utilizzo della CPU

Un'ultima nota riguarda l'utilizzo della CPU. Durante il test descritto nella sezione precedente abbiamo anche tenuto sotto controllo l'utilizzo delle risorse da parte del sistema host, ottenendo i seguenti risultati:

- velocità del clock = 100: il carico di lavoro della CPU è nella norma
- velocità del clock > 100: il carico di lavoro della CPU rimane inalterato rispetto al caso precedente, quindi nella norma
- velocità del clock < 100: il carico di lavoro della CPU aumenta direttamente al 100%, e resta tale per tutta la durata dell'esecuzione. Dopo molteplici prove differenti abbiamo inoltre constatato che questa situazione si verifica indipendentemente da quanto il clock viene rallentato. Infatti, abbassando la velocità del clock anche solo dell'1% (settando quindi un valore pari a 99), la CPU dell'host lavora al 100%, così come nel caso in cui venga settato uno degli altri 98 valori possibili.

Capitolo 9

Conclusioni e sviluppi futuri

Dopo questa analisi possiamo trarre le seguenti conclusioni: bisognerebbe provare a trovare una soluzione per i problemi che abbiamo evidenziato nel capitolo 8. Ciò richiede uno studio approfondito delle funzioni che gestiscono i diversi orologi, in particolare quella legata al Virtual Clock.
Da quello che abbiamo visto e studiato, la maggior parte delle funzioni di Virtualbox (quantomeno quelle che gestiscono il tempo) richiamano al loro interno altre funzioni, che a loro volta richiamano altre funzioni e così via.

Per quanto riguarda il nostro scopo iniziale, ovvero trovare un modo per modificare dinamicamente il clock in modo da poter velocizzare/rallentare l'esecuzione dei programmi all'interno del sistema guest, **possiamo concludere che il metodo è stato trovato ed è funzionante.**

Di seguito riportiamo un breve elenco di possibili sviluppi futuri:

- L'interfaccia da linea di comando `VBoxManage` potrebbe essere migliorata: si potrebbero infatti scrivere una serie di handlers completamente dedicati alla modifica dinamica del clock anziché appoggiarsi a quelli già esistenti senza quindi dover usare parametri di per sé inutili, come ad esempio `info`.

- Si potrebbe fare una ricerca approfondita sul Virtual Clock, cercando di capire

come funziona e in quali occasioni viene effettivamente utilizzato da Virtualbox.

- Legato al punto precedente, sarebbe opportuno trovare un modo per risolvere tutti i problemi legati alla temporizzazione e alla sincronizzazione tra il Virtual ed il Synchronous Clock. Si potrebbero utilizzare delle variabili di supporto per memorizzare gli istanti temporali prima che la velocità del clock venga cambiata in modo da impedire che il Virtual Clock possa arretrare.

Bibliografia

[1] http://en.wikipedia.org/wiki/Real-time_clock

[2] http://linux.die.net/man/ - Manuale di Linux

[3] http://download.virtualbox.org/virtualbox/SDKRef.pdf - Manuale SDK di Virtualbox

[4] http://en.wikipedia.org/wiki/Emulator

[5] http://download.virtualbox.org/virtualbox/UserManual.pdf - Manuale di Virtualbox