debianizzati

squeeze

GNU/Linux GNU/Hurd

1291399200

prodotto da debianizzati.org

n*5 dicembre 2010

Debianizzati

E-Zine

prodotto da debianizzati.org

Numero 5 **dicembre 2010**

Indice

Editoriale

DUCC-IT 2010

Introduzione

Nonostante qualche difficoltà passeggera — l'estate si è sempre mostrata un bel freno allo sviluppo del progetto e-zine — la rivista Debianizzati *della comunità* debianizzati.org *si appresta ad uscire: si tratta della sesta pubblicazione.*
Gli sforzi per sostenere e vivere il software libero sono sempre grandi e con un passo alla volta stiamo cercando di tracciare il nostro sentiero. Così, mentre all'interno della comunità ci si sta aggiornando (vi ricordo la migrazione di maggio verso una nuova engine *del portale), si guarda sempre un po' oltre e si cerca di intraprendere relazioni verso tutto il mondo là fuori: sia quello di mamma Debian, base sulla quale ha preso forma la nostra comunità, sia quello riguardante tutto il software libero.*
In un mondo dunque che sta cercando di sommare gli sforzi per un unico obiettivo comune, le comunità italiane di Debian e Ubuntu hanno riunito la Debian Community Conference *e l'*Ubuntu Meeting *per dare vita alla* Debian / Ubuntu Community Conference. *Il 18 e il 19 settembre 2010 è avvenuto dunque questo incontro, nome in codice* DUCC-IT 2010. *Fra i debianizzati hanno seguito l'incontro* MadameZou, *la quale ha anche sostenuto due conferenze, e* nex_necis.
A seguire, un riassunto dell'evento proprio da parte di nex_necis.

Buona lettura e a presto!

Brunitika

DUCC-IT 2010

Reduce dall'impegnativo fine settimana mi sembra doveroso scrivere qualche riga per donare una testimonianza a chi non ha potuto partecipare all'evento.
Il glorioso meeting ha inizio un bene augurante venerdì 17 del settembre 2010 presso "Projectz on island", un ridente *hacklab* e circolo nella periferia perugina. La prima sera viene dedicata alle presentazioni, ai saluti, a un'ottima cena e alle lunghe discussioni fra appassionati che, sfoderando ognuno il proprio portatile, si confrontavano sulle novità del mondo GNU/Linux e sugli argomenti più svariati. La serata è proseguita fino alle ore piccole generando in diversi di noi pesanti difficoltà per la sveglia della mattina successiva: tuttavia alle 8 la cucina del vicino ostello era già gremita di volenterosi che si preparavano per la marcia verso il centro.
Le 10:00 del 18 settembre non ci colgono impreparati: il Dipartimento di Matematica e Informatica vede confluire decine di persone verso lo stand di Ubuntu e la vicina aula già predisposta per l'avvio dei *talk*.
Dopo il benvenuto da parte dell'Università e degli organizzatori, Luca Bruno e Milo Casagrande hanno illustrato alla platea i molteplici modi in cui è possibile contribuire allo sviluppo e al miglioramento di Debian e Ubuntu, dando modo ai presenti di decidere quali discussioni seguire in base ai propri interessi e capacità. Successivamente giunge attesa l'ora dell'indecisione. Infatti la giornata si divide in due aule: la superiore in cui Milo Casagrande e Francesca Ciceri spiegano il funzionamento dei *team* di traduzione prima in via teorica e poi in via pratica, illustrando le modalità con cui questi lavorano; nell'aula inferiore Luca Bruno, Andrea Colangelo e Andrea Gasparini si sono occupati più di spiegare

dal lato tecnico il funzionamento e la struttura delle distribuzioni Debian e Ubuntu, gettando le basi per il pomeriggio.
Fra traduzioni e informazioni arriviamo affamati all'ora del pranzo alla mensa universitaria, dove un *overflow* alla cassa ci permette di scambiarci opinioni su quanto sentito, fra i borbottii degli stomaci.
I lavori proseguono nel pomeriggio con Paolo Sammicheli, impegnato nella teoria e pratica del testing delle immagini ISO nell'aula superiore; in quella inferiore invece si seguita con Andrea Colangelo e Andrea Gasparini il discorso della mattinata. Attraverso concetti semplici e chiari viene illustrato come creare un ambiente adatto alla pacchettizzazione e come si lavora in questo senso, dando ampio spazio agli standard usati nelle distribuzioni *debian-based* e ai *tools* utili per creare, aggiornare e adattare i pacchetti. Subito dopo è seguita la sessione pratica con la creazione in diretta di un *package* con aiuto in diretta ai volenterosi che si stavano cimentando.
Nello stesso momento nell'aula superiore Paolo Sammicheli e Luca Bruno spiegavano il *bug triaging*, la sua importanza, il peso che ha sulla mole di lavoro e le modalità con cui viene effettuato; facendo seguire alla teoria una sessione pratica di segnalazione *bug* e *triaging* (durante la quale si vocifera di un *bug* scoperto in loco).
Nell'aula inferiore si svolgeva intanto la relazione su *Marketing e Promozione del software libero* con Paolo Sammicheli e Italo Vignoli, molto partecipata da parte del pubblico e che ha sollevato molte interessanti osservazioni sulle modalità con cui questo opera. Il pomeriggio si è concluso con il *talk* del nostro DPL Stefano Zacchiroli, che ha esposto in maniera chiara la situazione attuale sul rapporto fra Debian, Ubuntu e le altre distribuzioni derivate, facendo forza sulla collaborazione che deve caratterizzarle per un comune sviluppo.
Oramai stremati dalle troppe informazioni ci concediamo una cena che solo l'Umbria può offrirci e ci tratteniamo nella stupenda Perugia per qualche ora approfondendo gli argomenti della giornata e lasciandoci andare ad argomenti più leggeri, sfoggiando battute e aneddoti di difficile comprensione a coloro che non facevano parte della comitiva.
La notte scorre tranquilla e il giorno successivo si torna al "Projectz on Island" per un'interessante incontro tenuto dalle rappresentanti di *Debian Women* e *Ubuntu Women* sul ruolo delle donne nell'*open source* e le problematiche che queste incontrano ogni giorno, anche questo svoltosi con numerosi interventi e un'appassionata partecipazione anche via IRC, grazie alla trasmissione in *streaming* dell'incontro.
Il terzo giorno si chiude con il pranzo, con i saluti e con la corsa ai treni.
Il bilancio finale è più che positivo, ottimi incontri, stupende persone e la sensazione di muoversi verso un unico obiettivo. Ci si può solo augurare che anche le prossime edizioni rispettino questo spirito e che l'affluenza sia ancora più numerosa.

nex_necis

Links

Comunicato stampa: http://www.fsugitalia.org/eventi/doku.php?id=duccit10:comunicato_stampa
Video DUCC-IT 2010: http://ftp.acc.umu.se/pub/debian-meetings/2010/ducc-it

News da debianizzati.org

1 Intervista a pmate

Nel numero 1 questa e-zine ha pubblicato un'intervista a MaXer, uno dei fondatori di debianizzati.org, nella quale si è parlato della nascita e dello scopo del portale.
Tanto tempo è passato, il web si è evoluto, anzi, si evolve in continuazione e debianizzati.org non è da meno. Recentemente il portale ha subito un profondo restyling, *sia nella grafica sia nei contenuti.*
Per comprendere meglio le trasformazioni e i nuovi servizi di debianizzati.org abbiamo fatto quattro chiacchere con uno degli amministratori, pmate.

Iniziamo con il conoscere il nostro interlocutore. Chi è pmate e qual è il suo ruolo all'interno di debianizzati.org?
Sono un tecnico informatico e un sistemista appassionato di free software da una decina d'anni.
Ho conosciuto debianizzati.org circa tre anni fa e me ne sono follemente innamorato… Oggi faccio del mio meglio per il portale contribuendo alla sua gestione insieme a MaXer, a tindal e a tutti gli altri membri dello staff.

Recentemente il portale ha subito profonde modifiche. Ci spieghi la struttura e gli obiettivi di debianizzati.org?
Questa è sicuramente una fase travagliata della vita di debianizzati.org. Come sapete il portale è costituito da vari "rami": forum, blog, wiki, chan IRC, e-zine e (recentemente) anche gruppo su identi.ca. Dal punto di vista della gestione "tecnica" il lavoro è tanto e si sta cercando (a partire dalla recente migrazione in poi) di intraprendere un percorso che possa portare allo sfruttamento di drupal come "contenitore" unico di tutte le attività.
Quello che però più mi preme sottolineare è lo sforzo che si sta facendo per "aprire" debianizzati.org all'utenza, per dare l'opportunità a chi ha voglia di contribuire di farlo in piena autonomia organizzativa.
Nel forum xtow e ferdybassi sono i nuovi moderatori delle sezioni Blog e Guide@Debianizzati.Org.
All'interno di queste sezioni sono stati creati due spazi "privati" dove gli appartenenti ai nuovi gruppi "blog member" e "wiki member" possono discutere della gestione di blog e wiki.
Proprio per quanto riguarda il blog, la scrittura degli articoli è stata aperta a tutti. Chi intendesse collaborare in questo senso non ha che da farsi avanti e da proporre un articolo.
Il wiki ha due nuovi amministratori: ferdybassi e s3v, che stanno completando il lavoro di indicizzazione e ricategorizzazione dei contenuti.
Come detto è nato un nuovo gruppo su identi.ca nel quale è possibile comunicare in maniera "diversa" e rimanere in contatto con altre realtà e persone legate al mondo di Debian e del "free software" in generale (ad esempio il dpl o Stallman).
Insomma, come hai ben detto, i cambiamenti sono tanti ma il filo conduttore è uno solo: debianizzati.org vuole incontrare i suoi utenti.

Visto che debianizzati.org richiama espressamente Debian, in che misura esso contribuisce allo sviluppo e alla diffusione di Debian, il Sistema Operativo universale?
Grazie della domanda: è un argomento a causa del quale, nel tempo, sono nati diversi malintesi. Spero di poter fare un po' di chiarezza.

debianizzati.org non è parte del progetto ufficiale Debian, questo deve'ssere chiaro. Il portale è frequentato da un gruppo di persone (ovviamente appassionate di questo meraviglioso sistema operativo) che si ritrovano insieme e si organizzano in totale autonomia e con totale indipendenza da Debian, secondo regole e consuetudini che non sono quelle di Debian. Il contributo, pertanto, è demandato alla volontà e all'intraprendenza dei singoli.
In senso stretto debianizzati.org non contribuisce ufficialmente al progetto Debian. Alcuni debianizzati adesso sono debian developers, altri sono entrati nel gruppo di traduzione, etc. ma tutti loro hanno seguito dei percorsi "autonomi" e non "istituzionalizzati" dal portale. Debian e debianizzati.org sono due realtà assolutamente indipendenti l'una dall'altra.
Esistono delle pagine della sezione Guide che spiegano come contribuire a Debian, se ne parla in diverse discussioni del forum o in alcuni post del blog, l'ezine ha intervistato due Debian Project Leader e i valori ai quali si ispirano le attività del portale richiamano espressamente quelle del contratto sociale di Debian, ma tutto questo non è indice di "legame ufficiale".
Personalmente mi auguro che tanti debianizzati possano prima o poi passare il guado e cominciare a contribuire attivamente al progetto Debian così come mi auguro di poter incontrare sempre più spesso debian developers o debian contributors nel canale IRC di debianizzati.org, di poter leggere qualche loro intervento nelle dicussioni del forum o un loro post nel blog.
Sarebbe un segnale bello e forte che finalmente utenti comuni e membri ufficiali del progetto vogliano "incontrarsi", vogliano cercare un terreno comune di contatto e di confronto. Al di fuori di ogni protocollo.
Se poi in Debian si dovesse valutare (finalmente) l'opportunità di creare un canale "ufficiale" di comunicazione con la base utenti (che non sia solo quello delle classiche mailing list o dei canali IRC visto che molti, a quanto sembra, non li trovano molto… confortevoli), beh, non potremmo che esserne felici e metterci a studiare tutti insieme una strategia comune di indirizzo alla contribuzione.

Ottimo, un'ultima domanda. Cosa occore per partecipare ed essere parte attiva in debianizzati.org?
Nessuna dote in particolare se non quella dell'entusiasmo e della voglia di fare e di contribuire. debianizzati.org è un portale aperto davvero a tutti. Non è assolutamente necessario dover partecipare a tutte le sue attività per poter contribuire. Basta solo farsi avanti per il settore che interessa, proporsi. Per sapere "come farlo" è sufficiente chiedere sul forum o contattare direttamente un membro dello staff o anche soltanto un utente più anziano.
Tutti saranno i benvenuti!

intervista by e-zine group

Storia e filosofia di Debian

2

Intervista ad Andrea Mennucci

Ho incontrato Andrea Mennucci alla Debian Ubuntu Community Conference (DUCCIT10) di quest'anno a Perugia: alloggiavamo nello stesso ostello e in quanto entrambi debianisti, oltre che entrambi toscani, è stato abbastanza naturale trovarci a chiacchierare.
Andrea, di professione ricercatore di Analisi Matematica alla Scuola Normale Superiore di Pisa, è diventato Debian Developer nel lontano febbraio 2001 e mantiene svariati pacchetti tra cui debdelta, freevo, Mplayer e Zope.
Ovviamente, non potevo lasciarmelo sfuggire e così gli ho strappato la promessa di rispondere a qualche domanda. Quindi ecco qui l'intervista che mi ha concesso.

MadameZou Ciao Andrea e innanzitutto grazie per aver accettato di farti fare qualche domanda. La mia prima curiosità è, se vuoi, piuttosto banale e tuttavia sorge spontanea ogni qualvolta incontro un altro debianista: come e con quali motivazioni ti sei avvicinato — da utente prima e da sviluppatore poi — a Debian?

Andrea A me era sempre piaciuto programmare, come hobby; nel 1985 mi fu regalato un Apple IIc, e come passatempo scrivevo programmi in Basic (e qualcosina in assembly). Negli anni '90 ho programmato per i primi Mac della Apple e per VAX/VMS. Poi ho scoperto il mondo Unix, che ho trovato molto più divertente: infatti con poca fatica, usando `apropos` e `man`, si poteva scoprire come funzionava il sistema operativo e come scrivere programmi interessanti. L'unico neo dei sistemi Unix era che… non ero sistemista e dunque non potevo fare tutte le modifiche che mi sarebbero piaciute. Quando nel 1995 ho sentito parlare di Linux ho iniziato a usare la distribuzione Slackware; ricordo ancora l'emozione, la prima volta che il mio piccolo PC ha eseguito il "boot" e si è trasformato nella mia "workstation personale". Nel 1997 sono passato a Debian (e ho inviato il mio primo bug report, numero 11468); ho apprezzato subito il carattere di comunità aperta, in cui si poteva collaborare; dopo pochi anni ho fatto domanda e sono diventato sviluppatore nel febbraio 2001.
Come vedi, il mio avvicinamento a Debian è in realtà la conclusione di un lungo percorso; alla fine mi sono fermato laddove ho trovato un ambiente che soddisfacesse ai miei desideri: avere pieno controllo sul sistema operativo e sul software che uso, divertirmi, e nel divertirmi fare qualcosa di utile.

MadameZou Vista la tua lunga, quasi decennale, militanza in Debian come sviluppatore è chiaro che la tua è una prospettiva privilegiata da cui guardare non solo l'evoluzione della struttura della distribuzione (ad esempio quindi le innovazioni sul piano meramente tecnico) ma anche dell'infrastruttura (le diverse modalità di gestire il lavoro degli sviluppatori, i nuovi ruoli) e del modo di proporsi verso l'esterno della distribuzione stessa. Qual è il cambiamento ai tuoi occhi più significativo avvenuto in Debian durante questo lasso di tempo?

Andrea Vorrei risponderti in modo paradossale: il miglior cambiamento di Debian è che Debian non è cambiata poi molto. Infatti la tua domanda si può leggere in varie maniere diverse.
Dal punto di vista numerico, Debian è cambiata moltissimo: tutti i numeri sono cresciuti esponenzialmente in questi anni (e ancora crescono), come il numero di pacchetti, le architetture supportate, il numero di bug e il numero degli sviluppatori. Per questo, il progetto si è molto modificato negli anni.

Nei primi tempi il progetto era più "anarchico" in quanto molte procedure erano regolate dalle consuetudini: c'erano persone che si occupavano delle faccende più delicate (come ad esempio i sysadmin dei server centrali), ma non era chiaro (almeno ai miei occhi) chi avesse deciso su chi si occupava di cosa. Questo era ragionevole in un progetto che era nato dal nulla e che all'inizio si era autoorganizzato. Adesso ci sono molti più sviluppatori, e ne è seguita un'organizzazione più regolata in cui vi sono precisi ruoli; esistono molti team, che si occupano di questioni specifiche, come si vede da http://www.debian.org/intro/organization, e il lavoro di questi team è più trasparente: i team inviano email informativi alle liste generali.
Dal punto di vista tecnico, Debian ha fatto grandi passi avanti. Solo per citare alcuni aspetti, ti posso dire che, oggigiorno, i nuovi pacchetti inviati nella "new queue" vengono automaticamente controllati usando lintian, ricompilati dai buildd per tutte le 14 architetture e quindi vengono installati e disinstallati automaticamente da piuparts; questo aiuta a tenere una qualità molto elevata.
Le cose più importanti, però, non sono cambiate!
In questi 17 anni da quanto Debian è nata, altre distribuzioni (sia commerciali sia di volontariato) sono nate, si sono trasformate, e poi alle volte sono morte. Debian invece è sempre qui, è sempre una comunità di volontari, guidata dal Contratto Sociale con gli utenti. Come dicevo sopra, ciò che conta per me è la possibilità di fare qualcosa di utile e divertente; e questo non è cambiato.

MadameZou Ho notato che hai mantenuto Mplayer, che è uno dei pacchetti che più di tutti soffre delle rigorose politiche Debian relative alle licenze dei programmi, dato che spesso sono necessari codec *non–free* per usarlo per vedere certi filmati e/o ascoltare musica in un determinato formato. Quindi spesso molti preferiscono appoggiarsi al repository (non ufficiale) multimedia e da lì prelevare sia Mplayer che i codec. Secondo te questo rappresenta un collo di bottiglia per questo pacchetto? È un problema che chi mantiene/co-mantiene il pacchetto si è posto? Come è possibile migliorare la situazione senza penalizzare al contempo le DFSG?

Andrea Sarebbe più giusto dire che "mantenevo MPlayer". Io me ne sono occupato per circa nove anni; i primi cinque li ho passati a cercare di far entrare MPlayer dentro Debian, risolvendo in qualche modo tutti i problemi legali; il pacchetto Mplayer entrò ufficialmente in Debian nell'ottobre 2006; però all'incirca dal marzo 2009 se ne occupa principalmente Reinhard Tartler, io mi sto dedicando più a debdelta e a freevo.
Vengo ora alla questione dei codec. Per quel che ne so io, la questione è cambiata parecchio negli ultimi tempi: le librerie FFMpeg, che sono usate anche da MPlayer, sono ora in grado di leggere tutti i formati più diffusi, senza bisogno di usare codec esterni; dunque la versione 1.0 rc3 di MPlayer che si trova in Debian/squeeze (cioè in testing) dovrebbe soddisfare tutti (o quasi). È venuta a cadere anche la necessità di usare debian-multimedia per avere Mencoder, in quanto Debian/squeeze contiene una versione "ragionevole" di Mencoder.
Viceversa, i pacchetti del repository Debian-multimedia sono diventati scomodi: se si installa Mplayer da quel repository in Debian/testing, si devono installare anche tutte le librerie FFMpeg dallo stesso repository; le versioni lì presenti sono però incompatibili con quelle distribuite dentro Debian. Questo danneggia molti altri programmi, come ad esempio Totem, che è il lettore video standard di Gnome.
Dunque penso che, quando Debian 6.0 sarà rilasciata, i problemi che hai elencato saranno pressoché tutti risolti e pochi troveranno necessario usare ancora Mplayer e Mencoder da debian-multimedia.

MadameZou A proposito dei tuoi pacchetti, a Perugia mi hai mostrato brevemente il funzionamento di un piccolo ma interessante programma scritto da te: debdelta. Ci spiegheresti più diffusamente come funziona?

Andrea Il pacchetto debdelta contiene quattro comandi fondamentali: `debdelta`, `debpatch`, `debdeltas` e `debdelta-upgrade`.
I comandi `debdelta` e `debpatch` fanno per i pacchetti debian binari quello che `diff` e `patch` fanno per i file di testo.
Il comando `debdelta` analizza due pacchetti binari e crea un delta; si usa come segue:

```
# debdelta foobar_2.deb foobar_3.deb foobar_2_3.debdelta
```

Il comando `debpatch` applica il delta, come ad esempio

```
# debdelta -A foobar_2_3.debdelta  foobar_2.deb  foobar_3_copia.deb
```

dopo questo comando, si otterrà che `foobar_3_copia.deb` è identico a `foobar_3.deb`.
Il comando `debdeltas` serve a generare molti `delta` in un colpo solo.
Il comando più utile per l'utente è `debdelta-upgrade`. Quando un utente vuole aggiornare il software, normalmente deve scaricare le nuove versioni dei pacchetti binari; questo può essere molto lungo (se la linea dati è lenta) e/o costoso (se la linea è a pagamento). `debdelta-upgrade` cerca di porre rimedio a questi problemi, scaricando opportuni `delta` invece dei pacchetti. L'uso è semplice, basta eseguire (come root) i comandi

```
#  apt-get update
#  debdelta-upgrade
#  apt-get upgrade
```

Maggiori informazioni si trovano in http://debdelta.debian.net.

MadameZou C'è un altro tema che mi piacerebbe discutere con te: l'uso del Software Libero in generale, e di Debian in particolare, all'interno delle Università. Tu lavori alla Scuola Normale e inoltre per motivi di lavoro hai contatti con professori e ricercatori di Università di tutto il mondo: viene usato — e se sì, come e quanto — il Software Libero nei contesti di ricerca? E all'interno delle semplici reti delle Università?

Andrea La Scuola Normale fa molto uso del Software Libero: i server centrali del centro di calcolo usano quasi tutti Debian; se un utente compra un computer desktop e chiede un installazione Linux, il centro di calcolo installa Ubuntu. Quasi tutti i miei colleghi del settore scientifico usano sistemi GNU/Linux. Anche se ho meno esperienza, mi pare di vedere che una simile situazione si trova anche al Dipartimento di Matematica; non saprei dire però cosa succede in generale all'Università.
Purtroppo però invece la penetrazione del software libero è molto limitata nella Scuola Normale nel settore dell'amministrazione: tutti i dipendenti amministrativi che conosco usano Microsoft Windows; e non mi pare che ci saranno cambiamenti in futuro.

MadameZou A Perugia (v. l'Editoriale) si è parlato molto di collaborazione tra *upstream* e *downstream* e più in generale collaborazione tra Debian e Ubuntu. Qual è la tua esperienza in questo senso?

Andrea La mia esperienza di rapporto con *upstream* è buona: io colloquio spesso con gli autori *upstream* di `Freevo`, a cui riporto i bug segnalati in Debian e mando le patch (se ho avuto modo e tempo di prepararle); e le mie patch sono in genere accettate. Per la mia esperienza personale, inoltre, anche i rapporti Debian/Ubuntu sono buoni: ho avuto alle volte contatto con i mantainer Ubuntu dei miei stessi pacchetti e ci siamo sempre aiutati, quando necessario. Inoltre da tempo le pagine QA di Debian segnalano se la versione Ubuntu dello stesso pacchetto ha delle patch extra, e/o dei bug significativi, come ad esempio nella pagina relativa a Freevo, http://packages.qa.debian.org/f/freevo.html.

MadameZou Un altro tema di cui si è discusso molto a Perugia è stata la comunicazione, in particolare la comunicazione tra Debian e le comunità di utenti. Al meeting si sono visti alcuni utenti, ma la maggior parte dei partecipanti erano pur sempre sviluppatori o persone che in vario modo già contribuivano a Debian o Ubuntu, mentre sono stati davvero pochi i semplici curiosi. È sempre stato così? Come è possibile — secondo te — colmare questo *gap* tra utenti e sviluppatori?

Andrea In parte questo è dovuto al problema di cui sopra, cioè che la penetrazione dell'Open Source è ancora troppo bassa e l'utilizzatore medio di computer non ha la più pallida idea di cosa sia l'Open Source. Il meeting di Perugia è stato molto ben organizzato, ma aveva un taglio un po' tecnico. Chi non ha mai usato un sistema OS può beneficiare di più di una delle giornate Linux Day di divulgazione, dove un utente inesperto può vedere dimostrazioni di come si può facilmente usare un sistema OS per tutte le attività comuni, come la posta elettronica, il browsing, la scrittura di documenti, etc etc. Il meeting di Perugia invece approfondiva tematiche, come la pacchettizzazione, o il bug reporting, che purtroppo non attraggono l'utente che non sa nulla del mondo OS.

Quest'ultimo punto mi porta a una riflessione (che è anche una digressione). Quando io (raramente) uso software non open-source, e mi capita di notare un bug, mi scontro col fatto che non vi è una maniera chiara di comunicarlo agli autori del software e di chiedere aiuto. A quanto ne so, le due case maggiori che producono sistemi operativi, dicasi Microsoft e Apple, non includono nel prezzo del software un servizio di assistenza ai clienti: l'utente che incontra un problema deve arrangiarsi spulciando per Internet, sui *knowledge-base*, o nei forum (dove poi la risposta più tipica è "reinstalla tutto"). Le case più piccole che producono software a pagamento offrono in genere un servizio assistenza clienti, ma anche lì la risposta che ho visto dare ai miei conoscenti è stata «reinstalla tutto». Debian invece ha un chiaro meccanismo per comunicare un bug: il *Bug Tracking System*. Pensandoci bene, questo è paradossale: chi usa software a pagamento, se trova un bug, spesso non ha maniera di chiedere aiuto; chi usa software gratuito invece sì. Forse una maniera per avvicinare gli utenti al mondo Open Source, e dunque agli sviluppatori, sarebbe proprio di pubblicizzare questo fatto: far sapere che nel mondo Open Source esiste una canale aperto fra sviluppatori e utenti.

MadameZou Grazie davvero di essere stato così disponibile, alla prossima!

Andrea Prego!

MadameZou

3 Debian GNU/Hurd: il debian-installer

3.1 Introduzione

Nell'ultimo numero vi avevamo parlato dell'ultima serie di CD (serie L, con il primo CD L1) del noto Philip Charles. Già allora correvano però le voci che un certo Jeremie Koenig, giovane studente d'informatica all'Università di Strasburgo, avrebbe partecipato al Google Summer of Code 2010 [1] per un progetto a favore di Debian, ossia il porting del debian-installer su Debian GNU/Hurd. Il lavoro di Phil, una sorta d'installer basato su GNU/Linux dal quale si poteva poi lanciare il processo d'installazione di Debian GNU/Hurd, sembrava dunque aver fatto la sua storia. Il 24 luglio 2010 un suo post nella mailing-list debian-hurd [2] ne diede la conferma:

> I have just resigned as a Debian Developer and I am withdrawing from Debian GNU/Hurd.
>
> At 72 I have found that my commitments are increasing. Phil, can you do this for us? …«You are on a pension?» …«Good, you are being paid!» So I need to review my commitments.
>
> There are now people working on the Debian GNU/Hurd installer and they are quite capable of taking over.
>
> Over a ten year period I have produced 39 sets of iso images of varying quality.
>
> Good luck for the future and I am looking forward to the first official release.
>
> Phil.

Tutto ciò lasciava presagire che nel futuro avremmo avuto dei CD d'installazione per Debian GNU/Hurd basati sul "classico" debian-installer, quello grazie al quale possiamo ora installare Debian GNU/Linux o Debian GNU/kfreeBSD.
Una settimana prima dell'annuncio di Phil, il 18 luglio 2010, Samuel Thibault, mentore di Jeremie per il GSoC, aveva infatti già annunciato [3] un'installazione riuscita del sistema utilizzando un'immagine disco del giorno prima, contenente il tanto atteso debian-installer e una versione minimale di Debian GNU/Hurd.
A estate terminata (e dunque a GSoC concluso) abbiamo voluto testare l'ultima immagine di Jeremie per installare Debian GNU/Hurd con il "classico" debian-installer.

3.2 Prerequisiti e preparativi

Qualche link di riferimento per incominciare:
a) la pagina del wiki di Debian:
http://wiki.debian.org/SummerOfCode2010/HurdDebianInstaller/JeremieKoenig
b) la pagina del wiki con la Roadmap:
http://wiki.debian.org/SummerOfCode2010/HurdDebianInstaller/JeremieKoenig/Roadmap
c) la pagina di Jeremie su GNU:
http://www.gnu.org/software/hurd/user/jkoenig.html

d) l'ultima immagine dell'installer:
http://jk.fr.eu.org/debian/hurd-installer/mini.iso (attualmente risalente al 17.08.2010),

Se si vorrà installare il sistema su una macchina virtuale quale qemu vi ricordo inoltre che sarà necessario modificare la scheda di rete di default a causa dei driver di supporto (v. Debianizzati, numero 4 [4]). Come descritto nell'articolo del numero precedente lanceremo dunque qemu con:

```
$ qemu -m 1024 -net user -net nic,model=pcnet -hda debian-hurd.img -cdrom mini.iso -boot d
```

Attenzione
Il disco virtuale creato (in questo esempio debian-hurd.img) non dovrà superare i 3GB, altrimenti il programma di installazione si bloccherà al momento di installare GRUB2. La natura di questo comportamento non mi è purtroppo nota, ma empiricamente lo abbiamo potuto constatare.

Seguendo il README di Jeremie [5] effettueremo le seguenti operazioni:

1. Scarichiamo l'immagine mini.iso:

   ```
   $ wget http://jk.fr.eu.org/debian/hurd-installer/mini.iso
   ```

2. creiamo un immagine disco virtuale per l'installazione:

   ```
   $ qemu-img create hurd-install.qemu 3G
   ```

3. lanciamo l'installer con qemu[1]:

   ```
   $ qemu -m 512 -net nic,model=ne2k_pci -net user -hda hurd-install.qemu \
       -cdrom mini.iso -boot d
   ```

Nota
La configurazione di Jeremie relativa alla scheda di rete è diversa da quella da noi proposta; in ogni caso le varianti sono da considerarsi equivalenti a livello pratico.
Il disco virtuale proposto da Jeremie ha il suffiso .qemu ancora in antitesi con il nostro .img proposto; anche in questo caso non vi sono differenze sostanziali a livello pratico.

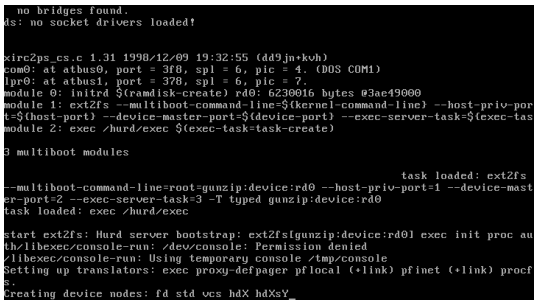
Infine, sempre a detta di Jeremie stesso, su una macchina reale saranno necessari almeno 200MB di RAM. Voglio infine ricordare che, allo stato attuale, il microkernel mach riconosce unicamente i dischi di tipo ATA. Per quelli SATA bisognerà ancora pazientare.

3.3 Installazione

Procurata l'immagine disco dell'installer (v. sopra) ci accingiamo ad avviare la macchina (reale o virtuale) impostando il BIOS (o chi per esso nel caso di una macchina virtuale) per avviare il sistema dall'immagine appena scaricata. La prima raffigurazione grafica è quella di un GRUB2 del 4 agosto con le invitanti opzioni:

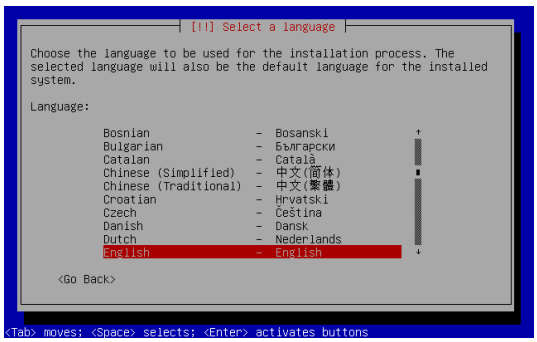
[1] Per i comandi il cui codice occupa più di una riga abbiamo adottato la soluzione valida per la shell: a capo preceduto dalla barra inversa, applicando inoltre un rientro per favorire la comprensione

Passeremo dunque all'ovvia scelta `Install Debian GNU/Hurd` facendo così partire l'installer vero e proprio. Schermata nera e prime scritte. Una sorta di *déjà vu* ci ricorderà l'avvio dei "classici" moduli del sistema, così come il caricamento del file system.
Ecco subito una piccola sorpresa.

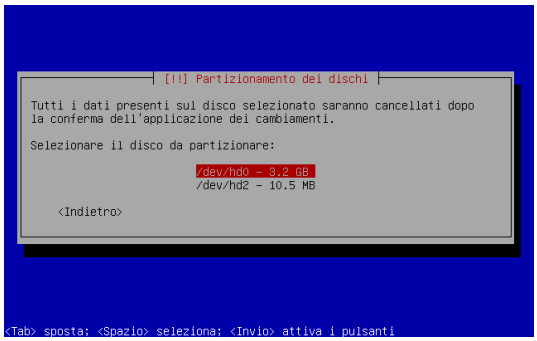


Quell'ultimo `Creating device nodes` ci offre la sensazione che qualcosa si stia "autocreando"… un respiro profondo… schermata nera:

Viene avviata una console, così come alcuni demoni di sistema. Quando però si avvia la console…

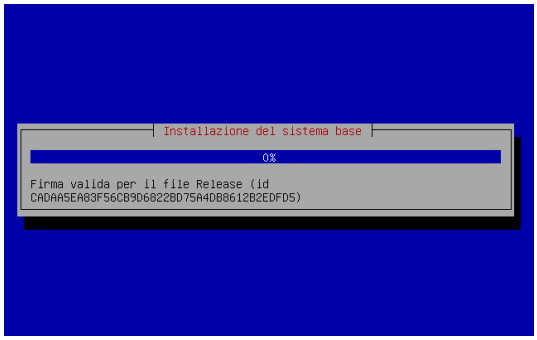


…arrivano le ncurses! Ed ecco il primo sguardo sul "classico" debian-installer! Per chi, come me, arriva da decine e decine d'installazioni fra abbozzi e inciampi, fra il linux-hurd-installer di Philip Charles o il crosshurd di Michael Bank, il momento è veramente entusiasmante (eh sì, mi accontento di poco…). Scegliamo dunque la lingua italiana e partiamo con l'installazione guidata. Alla selezione dei dischi, su un sistema virtuale con un disco da 3GB, l'installer riconosce le seguenti partizioni:

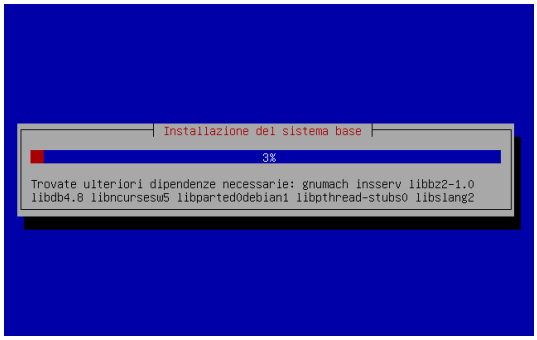


dove hd0 è riferito al disco virtuale e hd2 è riferito al CD dell'installer (come si può notare, al momento sono "solo" una decina di megabyte). Dopo aver ovviamente scelto hd0 quale disco da partizionare, lasciamo all'installer il doveroso compito di organizzare le partizioni. Otterremo il seguente risultato:

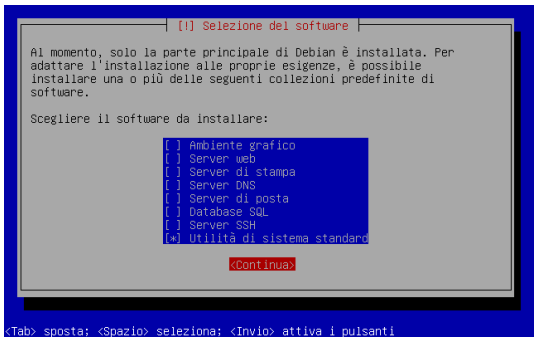
A questo punto viene verificata la firma dei pacchetti…



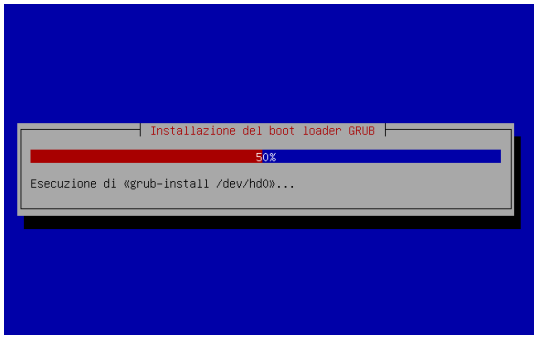
…e finalmente l'installazione può iniziare:



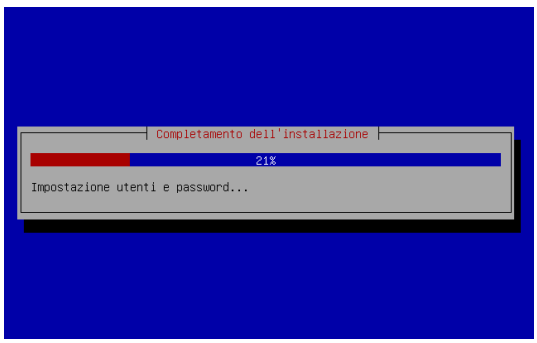
Terminata la prima fase ci verrà riproposto il "classico" menu con la scelta delle installazioni:



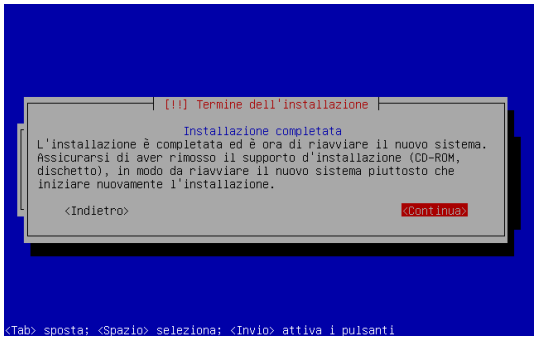
Nei nostri test abbiamo unicamente scelto il sistema di base; per installare altri servizi (quali ambiente grafico, server web, etc.) si consiglia di procedere in un secondo momento. Alla fine dell'installazione sarà la volta del bootloader GRUB2:



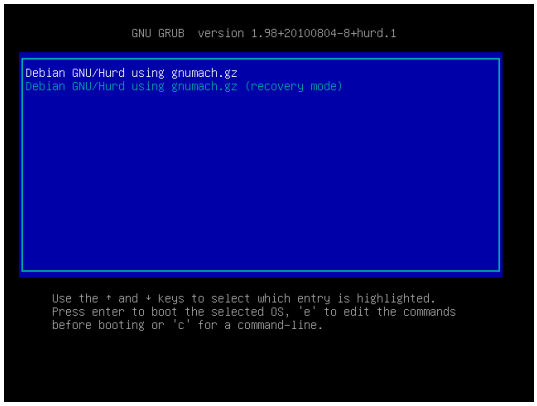
Seguirà l'impostazione degli utenti:



ed ecco l'installazione concludersi!



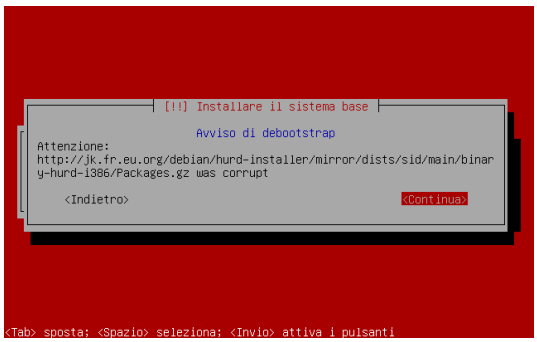
Al riavvio potremo lanciare il nuovo sistema dall'altrettanto fiammante GRUB2:



3.4 Problemi possibili e workaround

Durante la fase di studio dell'installer sono stati rivelati i seguenti problemi e sono stati adottatte le seguenti soluzioni.

3.4.1 Problema 1: archivio **Packages.gz** corrotto



Soluzione

Probabilmente relativo a un problema con i repository di Jeremie sul suo server. Anche con questo errore è stato comunque possibile continuare l'installazione. Durante gli ultimi test l'errore non si è più verificato.

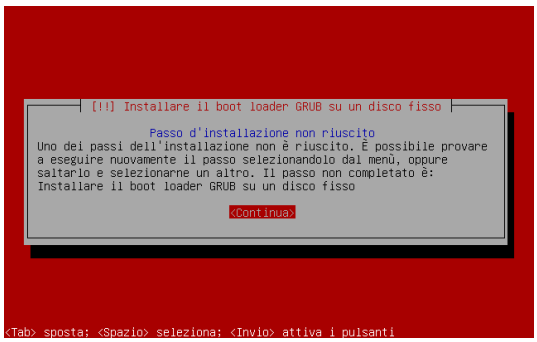
3.4.2 Problema 2a: impossibile installare GRUB

Soluzione

Come citato all'inizio di questo articolo (v. la sezione *Prerequisiti e preparativi*) questo errore è — almeno in qemu — direttamente legato alla grandezza del disco virtuale (che non deve essere più grande di 3GB). Se si volesse comunque risolverlo a installazione già avvenuta, in una macchina reale senza un GRUB(2) preinstallato o su una macchina virtuale, al riavvio del sistema (sempre con il CD d'installazione inserito) sembra sia sufficiente scegliere l'opzione `Boot from first hard disk` di GRUB; in ogni caso questo *workaround* non funziona se il sistema non è installato in modo corretto (v. Problema 2b). Bisognerà dunque utilizzare un GRUB "esterno" (su floppy o CD) per avviare il sistema (v. *Debianizzati*, numeri 0 [6], 1 [7] e 4 [8]).

3.4.3 Problema 2b: impossibile concludere l'installazione

Dal momento che risulta impossibile installare GRUB, anche scegliendo di continuare senza bootloader non è possibile concludere l'installazione:



Soluzione

Per uscire da questo *loop* non ci resta che spegnere la macchina.
Dal momento che l'installazione non viene terminata, fallisce in modo eguale la configurazione degli utenti e anche ripristinando GRUB (v. Problema 2) non è possibile "loggarsi" nella macchina (né come root né con l'utente creato durante l'installazione). Per risolvere si può utilizzare un hack che era stato integrato nell'ultimo installer di Phil: si monta la partizione di root su un altro sistema (ad esempio una live o una Debian GNU/Linux su di un'altra partizione oppure si monta la partizione in loopback se si tratta di un'immagine virtuale) e si vanno ad eliminare le `x` dal file /etc/passwd [9]: il mio piccolo contributo al progetto [10], denominato poi *Bruno's passwd hack* [11]. Eseguite le modifiche si potrà poi riavviare il sistema (previo `fsck` della partizione per riaggiustare i nodi dopo l'arresto "bruto" della macchina) e autenticarsi come root, per l'occasione senza password. Utilizzando `passwd` si potrà poi ricreare una password di proprio gradimento e tramite `adduser` si potranno creare tutti gli utenti desiderati.

3.5 Conclusioni

Come anticipato all'inizio dell'installazione, i test sono stati effettuati su una macchina virtuale (qemu). I test su una macchina reale sono purtroppo falliti in quanto il BIOS non ha riconosciuto il mini-CD come disco avviabile e dunque l'installazione non ha potuto neanche avere inizio. Probabilmente con un po' più di studio questo problema non dovrebbe rivelarsi troppo difficile da risolvere, sempre tenendo conto comunque che, al momento, l'installer è ancora in fase alpha.
Dopo tante battaglie per installare Debian GNU/Hurd, ecco infine arrivare il porting dell'installer "ufficiale". Nonostante il progetto sia allo stato attuale da considerarsi come sperimentale, la procedura d'installazione funziona egregiamente e si può ottenere una Debian GNU/Hurd perfettamente funzionante, sebbene ancora con mille limitazioni rispetto alle sorelle GNU/Linux o GNU/kfreebsd sicuramente, ad oggi, più complete. Si tratta comunque di un inizio.
Nonostante il fatto che una versione ufficiale del sistema sembra ancora essere lontana anni luce dalla realtà, con l'avvento dell'installer sarà forse possibile avvicinare più gente a GNU/Hurd e chissà che non possano esserci ulteriori sviluppi per questo sistema da alcuni considerato come il "vero" sistema GNU.
Al momento, non mi resta che incitarvi a provarlo e a darvi un arrivederci ai prossimi test.

Brunitika

3.6 Risorse di rete

[1]: http://socghop.appspot.com/gsoc/program/home/google/gsoc2010
[2]: http://lists.debian.org/debian-hurd/2010/07/msg00020.html
[3]: http://lists.debian.org/debian-hurd/2010/07/msg00016.html
[4]: http://e-zine.debianizzati.org/web-zine/numero_4/?page=45
[5]: http://jk.fr.eu.org/debian/hurd-installer/README.txt
[6]: http://e-zine.debianizzati.org/web-zine/numero_0/?page=56
[7]: http://e-zine.debianizzati.org/web-zine/numero_1/?page=22
[8]: http://e-zine.debianizzati.org/web-zine/numero_4/?page=42
[9]: http://lists.debian.org/debian-hurd/2009/09/msg00026.html
[10]: http://lists.debian.org/debian-hurd/2009/09/msg00065.html
[11]: http://lists.debian.org/debian-hurd/2009/09/msg00071.html

Hardware & Debian

4 Bye Bye nvidia proprietari

4.1 Introduzione

Chi ha sposato la filosofia che sta dietro al progetto Debian, chi si è preso le DFSG (*Debian Free Software Guidelines*, v. numero 3) e il progetto GNU a cuore, così come tutto quanto possa essere libero nel senso più ampio immaginabile del termine, si sarà più volte scontrato con l'enorme dubbio di cambiare hardware perché non supportato da strumenti liberi, di sfruttarlo a metà perché comunque una soluzione "a metà" esiste, o infine di accettare un compromesso con il software proprietario per poterlo utilizzare in modo esaustivo.
Fra i vari esempi che si potrebbero fare uno fra i più conosciuti riguarda le scheda grafiche nVidia [1]. Si potrà, infatti, decidere di non utilizzare questo tipo di hardware, utilizzarlo con i driver `vesa` [2] o `nv` [3] (anche se quest'ultimi non sono più sviluppati) ma rinunciando all'accelerazione 3D, oppure usufruire dei driver proprietari della nVidia stessa. Quest'ultima soluzione, seppur quella che consente di ottenere le migliori prestazioni, non è però sicuramente la più vicina alla filosofia che distingue il software libero.
Da alcuni anni si sta però lavorando alla creazione di un nuovo driver libero per schede grafiche nVidia sulla base del driver binario proprietario (*reverse engineering*). Il driver in questione è `nouveau` [4]; se già risulta essere utilizzabile senza problemi in un ambiente 2D, per il 3D bisognerà ancora aspettare un po' di tempo (gli obiettivi del progetto sono di portare a un driver open-source di qualità e che supporti proprio il 3D). In ogni caso, appoggiandosi sull'architettura `Gallium3D` [5] è già ora possibile sfruttare il driver `nouveau` con l'accelerazione grafica. Seppur ancora tutto sia abbastanza sperimentale, in questo articolo vi illustreremo come ottenere già ora un'installazione funzionante.

4.2 Bye Bye nvidia

Come prima cosa liberiamoci di tutto ciò che riguarda `nvidia` (a meno che stiamo utilizzando i driver `vesa` o `nv`); utilizzando `aptitude` possiamo fare ciò con un unico comando:

```
# aptitude purge ~nnvidia
```

eliminando dunque tutti quei pacchetti nel cui nome troviamo il termine «nvidia».

Nota
Se stessimo usando i driver `nvidia` proprietari in questione dovremo ricordarci che per rimuoverli non possiamo utilizzarli allo stesso tempo; sarà dunque necessario uscire dal server grafico e procedere in seguito alla rimozione dei driver.
Per uscire dal server grafico ci basterà un `logout` se non utilizziamo alcun display manager; altrimenti dovremo accedere a un'altra `tty` (`CTRL+ALT+Fx`, con *x* un numero da 1 a 6 a seconda della `tty` che vogliamo utilizzare), effettuare il login e spegnere da qui il display manager; ad esempio, per `gdm` (gnome):

```
# /etc/init.d/gdm stop
```

Notare che avremo bisogno di lanciare il comando da root. Se l'operazione dovesse fallire (ad esempio con i driver nouveau, almeno al momento ho delle difficoltà ad accedere ad altre tty con il server grafico avviato) possiamo sempre provare la strada più "bruta" premendo CTRL+ALT+BACKSPACE e forzando l'uscita dal server X.

4.3 Abilitiamo il driver nouveau

Prima di installare il driver sarà necessario avvalersi di un kernel con supporto a DRM (*Direct Rendering Manager* [6]). In Debian lo troviamo a partire dal kernel Linux 2.6.32-4 (patchato Debian, altrimenti il kernel Vanilla 2.6.33.1). Al momento attuale, su testing (squeeze) possiamo installare il kernel 2.6.32-5 con il nostro gestore dei pacchetti preferito. Su stable (lenny) dubito convenga "rovinare" l'installazione per esperimenti simili; in ogni caso, è comunque possibile installare il kernel in questione dai backports. Per installare poi il driver nouveau ci basterà come sempre un gestore dei pacchetti a caso (per me è comunque sempre aptitude):

```
# aptitude install xserver-xorg-video-nouveau
```

A questo punto siamo pronti a utilizzare il driver; se non avessimo ancora un kernel superiore alla versione 2.6.32-4 avviato dovremo riavviare il sistema con quest'ultimo kernel; fatto ciò, o se già stessimo utilizzando un kernel alla versione citata o superiore, dovremo solo dire a Xorg di utilizzare il driver nouveau all'avvio del server grafico. Per fare ciò andremo a modificare il "solito" /etc/X11/xorg.conf alla sezione Device:

```
Section "Device"
    Identifier     "nVidia Corporation G86 [GeForce 8400M GS]"
    Driver         "nouveau"
EndSection
```

Evidentemente alla voce `Identifier` va inserito il nome della scheda video (nell'esempio la mia) e alla voce `Driver`, se prima era utilizzato `nvidia` (o `vesa` o `nv`) andremo ora ad inserire il nostro nuovo `nouveau` (e scusate il gioco di parole).
Non ci resta ora che avviare il server grafico (`startx`) per vedere nouveau in azione.

Nota
Nonostante l'intenzione degli sviluppatori, al momento attuale il driver nouveau non supporta ancora il 3D.

Alla fine dell'articolo sono riportati una serie di benchmark con il seguente driver abilitato. Come promesso invece nell'introduzione vedremo ora come sfruttare l'accelerazione grafica di gallium per poterci avvalere del 3D.

4.4 Gallium

Gallium ci permette di utilizzare il driver `nouveau_dri.so`, il quale include supporto al 3D. Per installarlo bisognerà, però, innanzi tutto preparare un po' il nostro sistema.

4.4.1 Il kernel

Innanzi tutto occorre avere un kernel con supporto al DRM e in più con il codice del driver nouveau nei sorgenti. Per quanto riguarda il DRM, come descritto sopra, ci basterà dotarci del kernel Linux

2.6.32-4 o superiore. Per quanto riguarda il driver nouveau, questo è si presente nel kernel citato sotto forma di sorgente, ma la versione dei sorgenti inseriti nel kernel non è quella presente su http://www.freedesktop.org; per questo bisognerà "aggiornarli" dal git di freedesktop.org [7] e inserirli come modulo compilato del kernel. In alternativa si può utilizzare il kernel 2.6.34 da experimental che già contiene quanto richiesto.

Metodo manuale

Si potrà seguire la guida direttamente dal wiki di nouveau [8]; come prima cosa bisognerà inserire come anticipato sopra il DRM [9] ai sorgenti del kernel:

- Per scaricare via git i sorgenti del kernel:

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
```

- Per aggiungere il codice sorgente di nouveau:

```
$ git remote add nouveau git://anongit.freedesktop.org/nouveau/linux-2.6
```

- Infine si inserisce il codice di nouveau nel posto corretto nei sorgenti del kernel:

```
$ git checkout -b nouveau-master nouveau/master
```

A questo punto basterà configurare il kernel come si desidera (v. e-zine no. 1 *Compilazione del Kernel Linux*) prestando attenzione ad abilitare il modulo nouveau sotto `device drivers -> staging drivers -> nouveau`.

> **Attenzione**
> Per un suo funzionamento corretto il modulo nouveau dovrà essere compilato appunto come modulo (<M>) e non in modo statico (<*>) !

Compileremo e infine installeremo il kernel come di consueto.

Metodo Debian

Senza grandi storie, installeremo il kernel 2.6.34 da experimental dopo aver abilitato questo repository nel nostro /etc/apt/sources.list e aver dato un `update` della cache dei pacchetti. Nel mio caso:

```
# aptitude install -t experimental linux-image-2.6.34-1-amd64
```

L'architettura sarà evidentemente da adattare alla propria macchina.

4.4.2 Operazioni preliminari

Basandoci sull'esperienza del DD Sean "seanius" Finney [10] dovremo rimuovere altre librerie che saranno sostituite da versioni più aggiornate, onde evitare possibili conflitti tra le due versioni; installeremo, inoltre, qualche altro pacchetto che risulta fondamentale per l'installazione del driver.

```
# aptitude remove libdrm-dev
```

Quindi installeremo:

```
# aptitude install stow build-essential xorg-dev xutils-dev git-core libtool \
    mesa-common-dev libtalloc-dev automake autoconf
```

Oltre alle componenti di sviluppo, seanius ha avuto l'ottima idea di utilizzare `GNU stow` [11] per l'installazione delle componenti grafiche: `stow` è uno script perl che permette di installare una componente in `/usr/local/stow` creando poi dei symlink nelle parti del sistema dove essa è richiesta; in questo modo si avranno tutte le componenti installate tramite `stow` nelle directories di utilizzo del sistema (sotto forma di link appunto), ma facilmente localizzabili nella directory `/usr/local/stow`. Questa procedura, seppur "inutile" da un punto di vista funzionale, ci aiuterà moltissimo nel caso di rimozione e/o aggiornamento delle componenti stesse installate. Per ulteriori informazioni si legga il man page. Infine, per questioni di praticità creeremo una directory `nouveau` nella nostra $HOME per installare le componenti necessarie.

4.4.3 Installazione di dri2proto

Dopo esserci "piazzati" nella directory sopra creata (`cd nouveau`) incominceremo con scaricare i sorgenti dal git:

```
git clone git://anongit.freedesktop.org/xorg/proto/dri2proto
```

Questi ultimi verranno scaricati in una directory `dri2proto`. Ci sposteremo quindi in quest'ultima. Prima di passare alla configurazione dei sorgenti, utilizzando appunto stow per "ordinare" le componenti, dovremo creare una directory in `/usr/local/stow` per dri2proto, ad esempio con:

```
# mkdir -p /usr/local/stow/dri2proto-\$today
```

sostituendo la variabile `$today` con la data d'installazione, ad esempio 01092010. A questo punto, trovandoci nella directory dri2proto potremo passare alla configurazione vera e propria dei sorgenti:

```
$ ./autogen.sh --prefix=/usr/local/stow/dri2proto-01092010
```

L'opzione `--prefix` ci lascia appunto scegliere la directory d'installazione e dunque quella in stow. Seguirà la compilazione degli stessi con il noto:

```
$ make
```

In questo caso, ci verrà segnalato che il `make` è "inutile"; l'ho inserito comunque nei passaggi perché è comunque un bene ricordarsi la via di compilazione "classica" di un pacchetto. Infine passaremo all'installazione vera e propria con un "classico":

```
# make install
```

Infine diremo a stow di creare i symlink necessari per quanto riguarda l'installazione di dri2proto:

```
# stow -d /usr/local/stow dri2proto-01092010
```

L'opzione `-d` definisce solamente la posizione della componente (in stow appunto) dal momento che non ci troviamo in /usr/local/stow (da questa directory si potrebbe lanciare `stow` unicamente con: `stow dri2proto-01092010`).

4.4.4 Installazione di libdrm aggiornato

I passaggi sono praticamente identici. In breve, si scaricano i sorgenti:

```
$ git clone git://anongit.freedesktop.org/git/mesa/drm/
```

Ci si sposta nella directory creata e si prepara la directory in stow come sopra:

```
# mkdir -p /usr/local/stow/drm-\$today
```

Si configurano i sorgenti con:

```
$ ./autogen.sh --prefix=/usr/local/stow/drm-01092010 --disable-intel \
    --disable-radeon --enable-nouveau-experimental-api
```

Notare che sono definite le opzioni per indicare la directory d'installazione (`/usr/local/stow`), per disabilitare il supporto a intel e radeon, e per abilitare nouveau.
Compilazione:

```
$ make
```

installazione:

```
# make install
```

e stow:

```
# stow  -d /usr/local/stow  drm-01092010
```

4.4.5 Installazione di mesa

Ancora una volta, in modo analogo, si scaricano i sorgenti:

```
$ git clone git://anongit.freedesktop.org/git/mesa/mesa
```

Ci si sposta nella directory creata e si prepara la directory in stow come sopra:

```
# mkdir -p /usr/local/stow/mesa-\$today
```

Si configurano i sorgenti con:

```
$ ./autogen.sh --prefix=/usr/local/stow/mesa-01092010--enable-debug \
    --enable-glx-tls --disable-asm --with-dri-drivers= \
    --enable-gallium-nouveau --disable-gallium-intel \
    --disable-gallium-radeon --disable-gallium-svga \
    --with-state-trackers=glx,dri \
    --with-demos=xdemos,demos,trivial,tests
```

(le opzioni le ho ricavate direttamente dal lavoro di seanius: si abilita il gallium-nouveau con glx, si disabilita, nuovamente, il supporto a radeon e intel, infine è inserito qualche demo).
Compilazione:

```
$ make
```

Installazione:

```
# make install
```

e stow:

```
# stow -d /usr/local/stow mesa-01092010
```

4.4.6 Installazione del driver vero e proprio

Procederemo in ogni caso in modo analogo a quelli precedenti. Si scaricano i sorgenti:

```
$ git clone git://anongit.freedesktop.org/git/nouveau/xf86-video-nouveau/
```

Ci si sposta nella directory creata e si prepara la directory in stow come sopra:

```
# mkdir -p /usr/local/stow/xf86-video-nouveau-$today
```

Si configurano i sorgenti con:

```
$ ./autogen.sh --prefix=/usr/local/stow/xf86-video-nouveau-01092010
```

Compilazione:

```
$ make
```

Installazione:

```
# make install
```

e stow:

```
# stow -d /usr/local/stow xf86-video-nouveau-01092010
```

4.5 Configurazione

A questo punto siamo pronti a configurare il sistema per utilizzare il nuovo driver. Innanzi tutto dovremo dire a Xorg dove trovare i moduli appena installati e di abilitare il DRI. Per fare ciò aggiungeremo al nostro /etc/X11/xorg.conf le seguenti linee (dopo aver già evidentemente aggiunto il driver nouveau nella sezione Device; v. sopra, *Abilitiamo il driver nouveau*):

```
Section "Files"
    ModulePath "/usr/local/lib/xorg/modules,/usr/lib/xorg/modules"
EndSection

Section "ServerFlags"
    Option "GlxVisuals" "all"
Endsection
```

Dal momento che mesa non è in grado di vedere il driver nouveau_dri in /usr/local (quest'ultimo si trova in /usr/local/lib/dri/nouveau_dri.so) occorrerà creare un symlink di quest'ultimo in /usr/lib/dri con:

```
# ln -s /usr/local/lib/dri/nouveau_dri.so /usr/lib/dri/nouveau_dri.so
```

A questo punto possiamo riavviare la macchina in modo da caricare tutte le librerie del caso e gustarci ora il nostro server grafico viaggiare con il nuovo driver e l'accelerazione 3D!

4.6 Aggiornamenti

Per aggiornare il driver potremo sfruttare ora appieno l'organizzazione di stow. Innanzi tutto ci portiamo nella directory della componente che si vuole aggiornare; ad esempio dri2proto (nella directory nouveau creata all'inizio dell'installazione):

```
$ cd \$HOME/nouveau/dri2proto
```

poi si aggiornano i sorgenti dal git:

```
$ git clean -dfx && git pull
```

togliendo dapprima le "scorie" provocate dalla compilazione precedente. Si prepara poi una nuova directory in /usr/local/stow con la data dell'aggiornamento (ad esempio 02092010):

```
# mkdir -p /usr/local/stow/dri2proto-02092010
```

Si configurano i sorgenti come sopra:

```
$ ./autogen.sh --prefix=/usr/local/stow/dri2proto-02092010
```

Si compilano:

```
$ make
```

e si installa il tutto con:

```
# make install
```

Si eliminano infine i symlink dell'installazione precedente:

```
# stow -d /usr/local/stow -D dri2proto-01092010
```

e si creano infine per la nuova installazione:

```
# stow -d /usr/local/stow dri2proto-02092010
```

Nota
Grazie a stow il sistema rimane pulito, ma non solo: quando si utilizza l'opzione `-D` a essere cancellati sono solo i symlink; ciò significa che se l'installazione nuova non dovesse funzionare in modo corretto e si volesse ritornare a quella precedente, dal momento che la directory con i sorgenti compilati "vecchi" rimane intatta in /usr/local/stow, basterà ricreare i symlink con stow e tutto sarà come prima dell'aggiornamento (al di fuori chiaramente dei sorgenti).

4.7 Benchmark

Prima di terminare, per lasciarvi con qualche numero in mano, abbiamo effettuato qualche test di benchmark utilizzando la suite di phoronix [12]. I test sono stati effettuati sul seguente hardware:

- Processore: Intel Core 2 Duo CPU T5800 @ 2.00GHz (Total Cores: 2)
- Scheda madre: Dell 0U8042
- Chipset: Intel Mobile PM965/GM965/GL960 + ICH8M
- RAM: 3021MB
- Hard Disk: 250GB Western Digital WDC WD2500BEVT-7
- Scheda grafica: nVidia Corporation G86 [GeForce 8400M GS] 128MB (rev a1)
- Audio: SigmaTel STAC9228

Dati sul sistema operativo:

- Sistema: Debian testing/sid, Kernel Linux 2.6.34-1-amd64 da experimental
- File-System: ext3
- Desktop: GNOME 2.30.2
- Display Server: X.Org Server 1.7.7
- Display Driver
  - il "solo" driver nouveau: nouveau 0.0.15, OpenGL: 2.1 Mesa 7.7.1 (con Kernel Linux 2.6.32-5-amd64 da testing)
  - il nouveau/gallium: nouveau 0.0.16, OpenGL: 2.1 Mesa 7.9-devel
  - il driver nvidia: nvidia 195.36.24-4 con nvidia-glx 195.36.24-4

Incominciamo con una parentesi interessante: in rete si trovano spesso dei benchmark di grafica tramite la utility glxgears [13]; questo "strumento", seppur indicativo in grosso ordine, non è però l'ideale per dei benchmark significativi in quanto dipende molto dal processore e da altri fattori del sistema. In anteprima ecco i test con glxgears (media arrotondata):

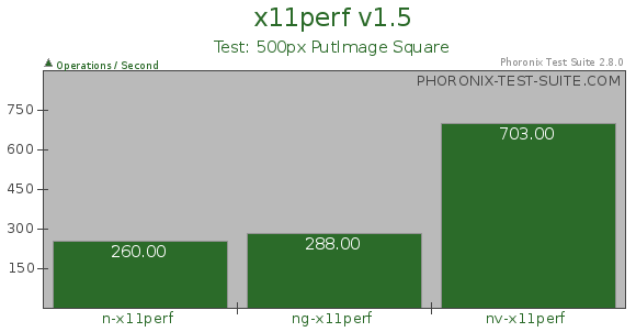
- Driver nouveau: 562 FPS
- Driver nouveau/gallium: 712 FPS
- Driver nvidia: 4381 FPS

Questo vorrebbe dire che il driver nouveau/gallium (d'ora in poi ng) è ca. 1.27 volte più veloce del driver nouveau (d'ora in poi n), mentre resta 6.15 volte più lento di quello proprietario nvidia (d'ora in poi nv); sarà effettivamente così? A seguire la risposta.

4.7.1 x11perf

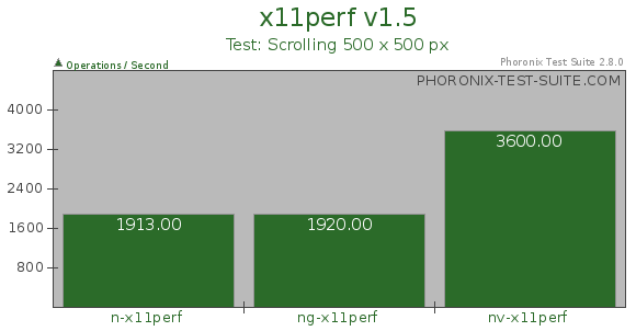
Questo test della suite di phoronix permette di effettuare una analisi del server grafico con prove di vario genere.

500px PutImage Square



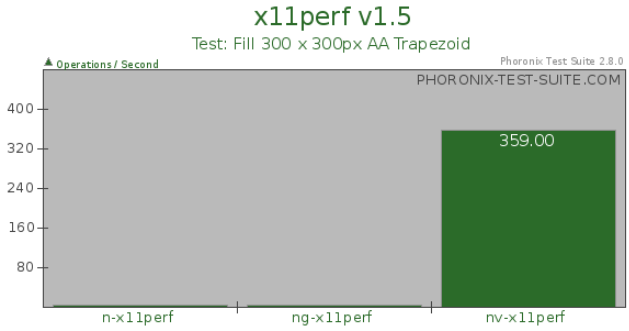
Driver ng 1.11 volte più veloce di n, 2.44 volte più lento di nv

Scrolling 500 x 500 px



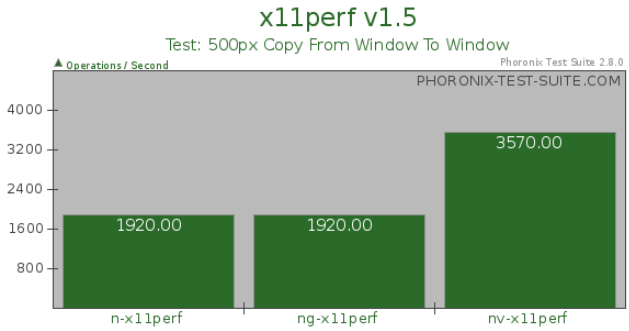
Driver ng e n praticamente equivalenti, più lenti di 1.87 volte rispetto a nv

Fill 300 x 300px AA Trapezoid



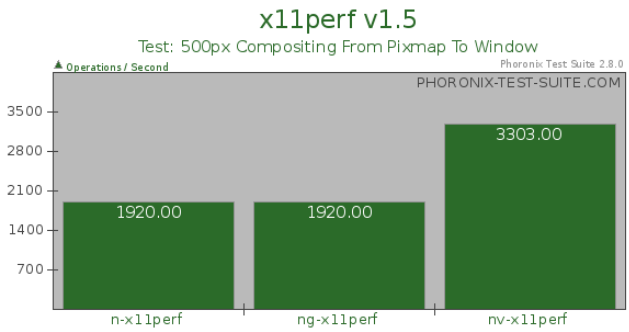
Guardando i singoli risultati (non visualizzabile nel diagramma) il driver nv risulterebbe 64 volte più veloce dei suoi compagni

500px-Copy-From-Window-To-Window



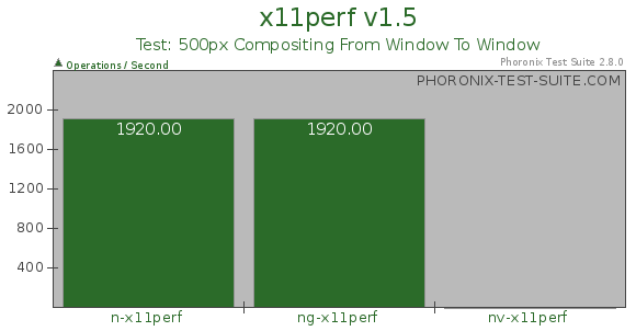
I driver ng e n sono equivalenti, entrambi 1.86 volte più lenti di nv

500px Compositing From Pixmap To Window



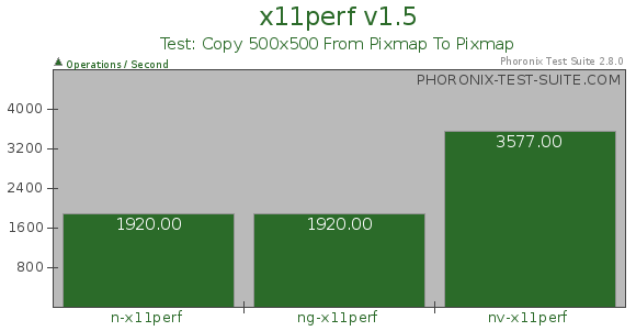
I driver ng e n sono equivalenti, entrambi 1.72 volte più lenti di nv

500px Compositing From Window To Window



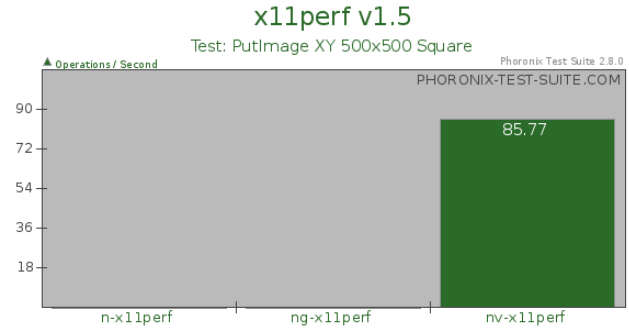
I driver ng e n sono equivalenti; in questo caso 1178 volte più veloci di nv (!)

Copy 500x500 From Pixmap To Pixmap



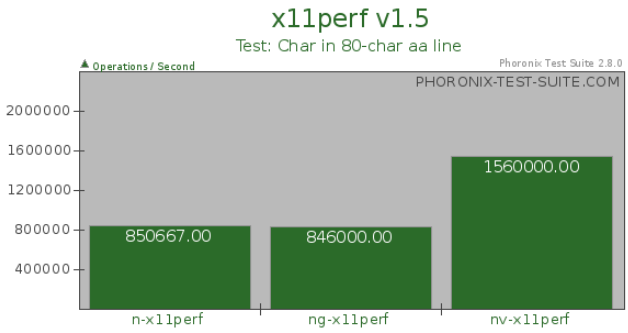
I driver ng e n sono equivalenti, entrambi 1.86 volte più lenti di nv

PutImage XY 500x500 Square



I driver ng e n sono equivalenti, entrambi 858 volte più lenti di nv (!)

Char in 80-char aa line

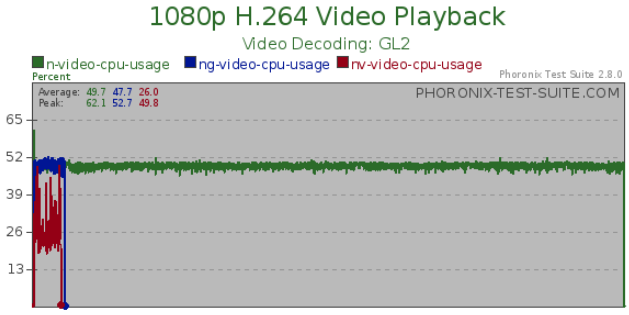


I driver ng e n sono praticamente equivalenti, entrambi 1.84 volte più lenti di nv

4.7.2 video-cpu-usage

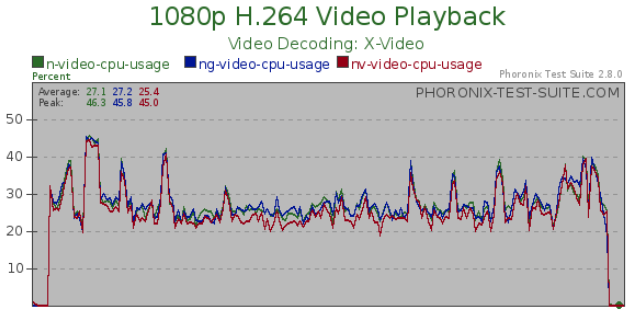
Questo test calcola il consumo della CPU per la visualizzazione di video con differenti codec; supportati dai tre driver in questione abbiamo testato GL2 e X-video.

GL2



Con una parentesi per il driver n, che come si può notare ha impiegato circa 12 volte di più di tempo per finire il test (il "collo della bottiglia" è in questo caso proprio il driver grafico e non la CPU come si può notare dal grafico), il driver nv "consuma" quasi la metà (1.84x) della CPU per visualizzare un filmato con il codec GL2.

X-video

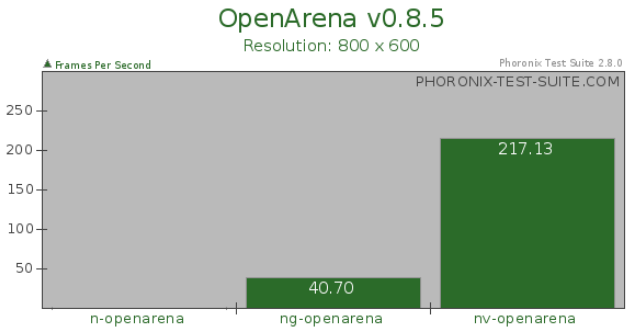


in questo caso, i tre driver sono quasi identici

4.7.3 openarena

Questo test calcola i FPS durante un'ipotetica partita al gioco openarena con differenti risoluzioni.

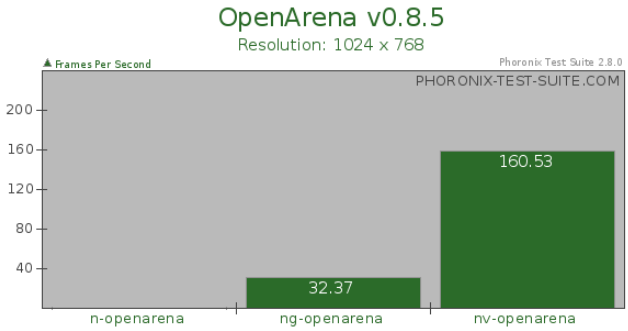
800x600



Il driver ng è di 25.44 volte più veloce di n, mentre resta 5.33 volte più lento di nv

1024 x 768

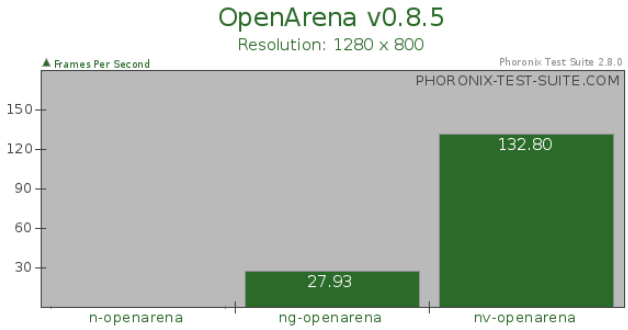
800x600



Il driver ng è di 32.37 volte più veloce di n, mentre resta 4.96 volte più lento di nv

1280 x 800

800x600



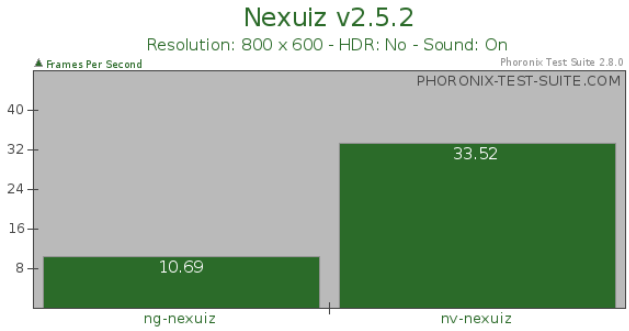
Il driver ng è di 34.91 volte più veloce di n, mentre resta 4.75 volte più lento di nv

4.7.4 Nexuiz

Ancora una volta una simulazione con il gioco Nexuiz; oltre alle differenti risoluzioni sono state fatte ulteriori misurazioni con HDR (*High Dynamic Range*) abilitato e senza. Viste le pessime prestazioni con il solo driver nouveau con openarena abbiamo rinunciato a testare questo driver con il ben più avido Nexuiz.

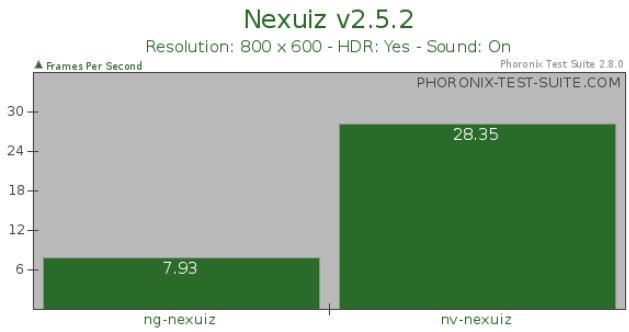
800 x 600 - HDR: No

800x600



Il driver ng si è mostrato 3.14 volte più lento di nv

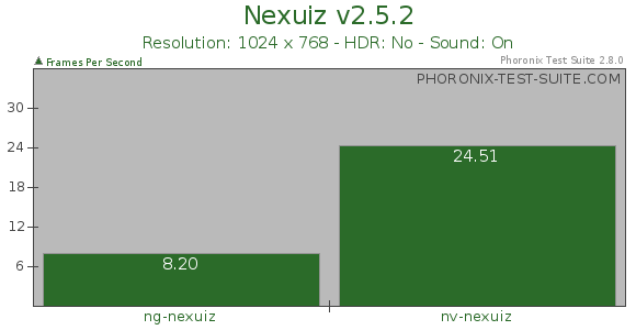
800 x 600 - HDR: Yes



Il driver ng si è mostrato 3.58 volte più lento di nv

1024 x 768 - HDR: No

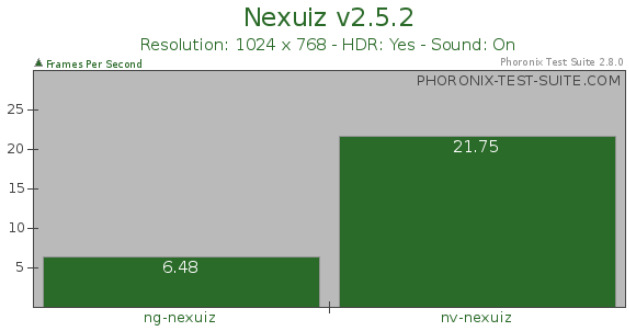
800x600



Il driver ng si è mostrato 2.99 volte più lento di nv

1024 x 768 - HDR: Yes

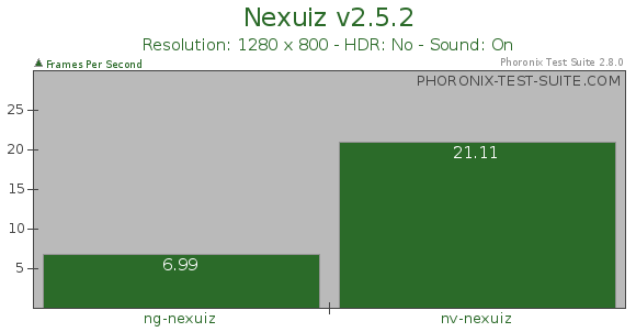
800x600



Il driver ng si è mostrato 3.36 volte più lento di nv

1280 x 800 - HDR: No

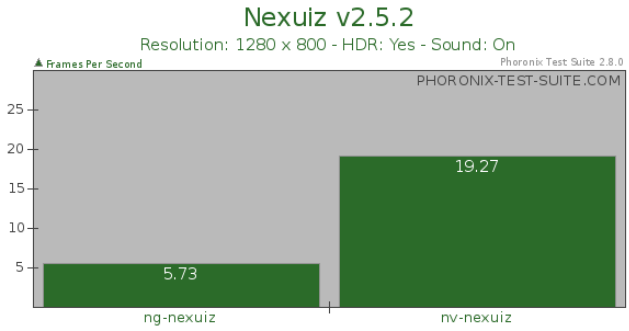
800x600



Il driver ng si è mostrato 3.02 volte più lento di nv

1280 x 800 - HDR: Yes

800x600



Il driver ng si è mostrato 3.36 volte più lento di nv

Conclusioni

Come possiamo notare dai risultati e aggiungendo da parte mia un qualche mese di sperimentazione, il driver nouveau/gallium incomincia a diventare maturo e forse in non molto tempo lo vedremo sempre più spesso inserito nelle distribuzioni più comuni (come del resto sta già succedendo con il “semplice”

driver nouveau). La situazione attuale lascia ben sperare che questo driver vada a sostituire quello proprietario per le schede grafiche nVidia.
Riprendendo ora l'ipotesi iniziale con glxgears, dove avevamo un driver nvidia proprietario più di sei volte più rapido del nouveau/gallium, dopo i risultati illustrati vediamo quanto glxgears non possa essere un riferimento sulle prestazioni di un driver, in quanto i valori calcolati tramite la suite di phoronix sono ben diversi da quelli ottenuti con quest'ultimo. Per sintetizzare diremo che:

- in un ambiente "desktop tradizionale" il driver nouveau/gallium e nouveau hanno praticamente prestazioni identiche. Il driver nvidia proprietario è superiore, ma "solo" in media di 1.9x volte superiore (sono stati tolti i risultati "sballati" del test dove la differenza era immensa). Ciò significa poco meno del doppio (sempre facendo riferimento alle 6 volte in più di glxgears).
- il "consumo" della CPU dei driver è praticamente uguale con il codec X-video; con il codec GL2, se nouveau e nouveau/gallium sono simili, il driver nvidia proprietario è molto meno avido e "consuma" quasi la metà. Interessante il fatto che nonostante i valori appena citati, i "picchi" del driver nvidia sono quasi equivalenti a quelli del driver nouveau/gallium; si potrebbe dunque presupporre un miglior *scaling* (utilizzare la prestazione massima della scheda solo quando serve, riducendo i consumi negli altri momenti) del driver nvidia rispetto ai suoi concorrenti. In relazione a ciò, è bene, però, ricordare che il driver nouveau (e nouveau/gallium) non supporta ancora alcun tipo di *power-management*.
- riguardo al driver nouveau, nonostante attualmente viene dichiarato un supporto unicamente al 2D, siamo riusciti a effettuare dei test anche con un gioco 3D complesso come openarena. Ciò nonostante le prestazioni sono pessime. Il driver nouveau/gallium invece fa una discreta figura, anche se resta quasi 5 volte più lento rispetto al cugino nvidia proprietario (in questo test è stata raggiunta la differenza massima fra i driver ng e nv). In ogni caso ulteriori segnali indicano un futuro prossimo soddisfacente. Il test con l'avidissimo Nexuiz (ci siamo risparmiati di soffrire con nouveau, il quale già con openarena arrivava fino ai grandiosi 0.8 FPS) è stato ai limiti già con nouveau/gallium, rendendo praticamente il gioco non utilizzabile. In ogni caso, in cifre assolute, il driver si è ancora mostrato "solo" poco più di 3 volte più lento rispetto al nvidia proprietario.

Non mi resta che lasciarvi provare il driver e per qualsiasi cosa non esitate a contattarmi o a segnalare qualsiasi osservazione sul forum di debianizzati.org.
Happy Hacking!

Brunitika

4.8 Risorse di rete

[1]: http://www.nvidia.it/page/home.html
[2]: http://www.x.org/wiki/vesa
[3]: http://www.x.org/wiki/nv
[4]: http://nouveau.freedesktop.org/wiki
[5]: http://www.freedesktop.org/wiki/Software/gallium
[6]: http://dri.freedesktop.org/wiki/DRM
[7]: git://anongit.freedesktop.org/nouveau/linux-2.6
[8]: http://nouveau.freedesktop.org/wiki/InstallNouveau
[9]: http://nouveau.freedesktop.org/wiki/InstallDRM
[10]: http://www.seanius.net/blog/2010/04/nouveau-gallium-and-compiz-on-debian
[11]: http://www.gnu.org/software/stow
[12]: http://www.phoronix-test-suite.com
[13]: http://www.xfree86.org/4.4.0/glxgears.1.html

Tips and Tricks

5 LVM mirrored

Piattaforma di riferimento Debian

A chi è rivolto a un "normale" medio utente desktop

Che cosa si dà per scontato che l'utente sappia installare Debian, conosca i principi base dell'LVM (*Local Volume Manager*) e abbia almeno una minima dimestichezza con un editor di testo e la linea di comando

A che cosa è finalizzato a ottenere con un sistema LVM i medesimi vantaggi di un sistema RAID (*Redundant Array of Independent Disks*). La guida illustra come eseguire semplici esperimenti su una macchina virtuale per testare le funzionalità di LVM, procedendo dapprima all'installazione su LVM, all'attivazione della modalità mirrored e quindi all'intervento in caso di guasto di un disco fisso mostrando come sostituire l'hard disk danneggiato e ripristinare la situazione iniziale di sicurezza

A che cosa può servire ad avere un sistema protetto da eventuali malfunzionamenti/guasti agli hard disk, così come ambiente di approfondimento e sperimentazione di soluzioni personalizzate.

5.1 Introduzione

Una problematica abbastanza comune, seppure spesso trascurata da molti utenti desktop, è l'affidabilità dei supporti (hard disk) e le garanzie contro la perdita dei dati. Purtroppo dell'importanza di quest'ultimi l'utente si accorge sovente troppo tardi, ovvero quando vengono irrimediabilmente persi. In particolare sarebbe opportuno preservare i dati sia contro erronee modifiche e cancellazioni da parte dell'utente stesso — ad esempio effettuando backup incrementali — sia contro il rischio di rottura dei supporti di memorizzazione di massa, ad esempio tramite un sistema di ridondanza dei dati.
Infatti, seppure sia un evento non così frequente, la rottura di un hard disk è sempre possibile e comporta la perdita inesorabile di tutti i dati in esso contenuti. Un sistema di ridondanza ci permette di distribuire i nostri preziosi dati (le foto del matrimonio, la tesi dell'università, le e-mail delle amanti…) su più supporti fisici in modo tale che alla rottura di uno di essi i dati siano pur sempre recuperabili dagli hard disk superstiti. In questo caso la soluzione più comunemente utilizzata e suggerita è il RAID, sia esso fisico oppure software.
Il RAID — *Redundant Array of Indipendent Disks* — permette di utilizzare più supporti di memorizzazione con la modalità *mirrored* (una partizione è copiata anche sugli altri dispositivi garantendo così la ridondanza), con la modalità *striped* (la partizione è creata dall'unione di più parti di disco differente) o con combinazione tra le due modalità a seconda del livello RAID prescelto.
Chi ha approfondito con la lettura di wikipedia [1] avrà notato che con la modalità *mirrored* si garantisce la sicurezza dei dati mentre con la modalità *striped*, oltre ad avere partizioni maggiori della capacità di un unico disco, si ottengono anche maggiori prestazioni di lettura e scrittura: i dati sono al contempo scritti su n dischi, quindi il tempo di accesso agli stessi si riduce di circa *n* volte. Purtroppo, per avere simultaneamente i vantaggi della sicurezza (*mirror*) e della velocità (*stripe*) è necessario almeno un RAID di livello 3 il quale richiede un minimo di tre dischi con tutti i maggiori costi e maggiori consumi

che ciò comporta. Inoltre a oggi, per limitazioni dell'architettura software stessa, non è ancora possibile nella maggior parte dei casi (eccezione fatta per alcuni particolari dispositivi fisici di RAID hardware) variare il livello di RAID o ampliare/ridurre il numero di hard disk presenti nel RAID stesso senza evitare la perdita dei dati.
Se però consideriamo una postazione desktop oppure un piccolo server domestico, difficilmente avremo necessità di alte prestazioni I/O di accesso ai dischi e difficilmente disporremo di più di un paio di hard disk. Proprio per queste particolari esigenze mi sono spinto a cercare una soluzione altrettanto efficace quanto il RAID, ma senza i problemi presenti in quest'ultimo. Ovvero una soluzione che permettesse di spostare a piacimento le partizioni da un hard disk all'altro, avere in modalità *mirrored* solo i dati importanti, di poter aggiungere o rimuovere dischi secondo necessità e di gestire dinamicamente la dimensione delle partizioni stesse, quasi come se fossero semplici cartelle in un filesystem. Ebbene, la soluzione è già esistente, si chiama LVM: *Logical Volume Manager* (Gestore di Volumi Logici).
Curiosità: a dire il vero inizialmente la «i» presente nell'acronimo RAID era inizialmente spesso interpretata anche come *Inexpensive* (economico): infatti, durante il periodo in cui il RAID fu ideato, questa soluzione permetteva tramite l'accorpamento di più dischi IDE di eguagliare e anche superare le prestazioni degli allora costosissimi SATA.

5.1.1 Che cosa sperimenteremo nell'articolo

L'obiettivo della guida è garantire la sicurezza dei nostri dati e del nostro sistema operativo avendo almeno la partizione di root e di home in ridondanza su due differenti hard disk. Tale ridondanza verrà effettuata utilizzando la modalità *mirrored* dell'LVM in modo tale che la partizione sul primo HD sia lo specchio di quella presente sul secondo e viceversa. Infine, seguendo la sperimentazione in ambiente virtuale proposta, vedremo come dalla rottura di un hard disk recuperare e avviare il sistema operativo nonché ripristinare la situazione iniziale.

5.1.2 Programmi utilizzati

La scelta d'uso di un ambiente virtuale per questo articolo è dovuta a semplicità operativa: non vi è bisogno di trafficare con cavi e cavetti, il ripristino della situazione iniziale è immediato e non sarà necessario rompere alcun hard disk. Inoltre il ricorso alla virtualizzazione permette a tutti i lettori di cimentarsi nella sperimentazione proposta e di replicarla a pari condizioni.
La scelta del sistema di virtualizzazione a cui affidarsi è ricaduta su KVM. Le ragioni di questa scelta sono in parte filosofiche, infatti si tratta di un progetto esclusivamente Free Software (Virtualbox invece si presenta nella versione OSE ed Enterprise), sia per la comodità della linea di comando disponibile, sia per le buone prestazioni grazie all'integrazione nativa nel kernel Linux.
Ovviamente oltre a una macchina virtuale sono necessari un po' di spazio sul disco (almeno 2/3GB per l'installazione solo testuale, minimo 10 GB con la presenza di un Desktop Enviroment), un installer Debian e tanta voglia di sperimentare.

5.2 Introduzione teorica al LVM

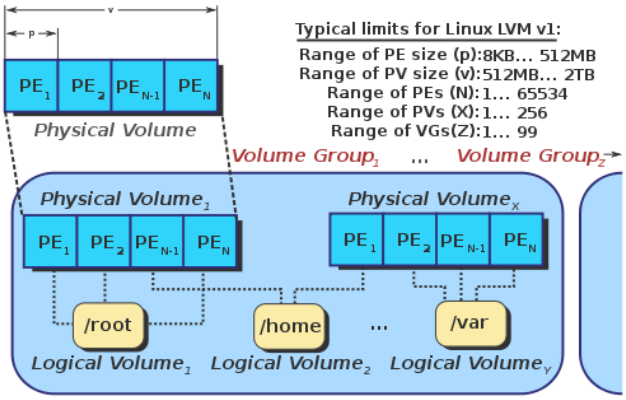
LVM è l'acronimo di *Logical Volume Manager*, ovvero di «Gestore dei volumi logici» di Linux. Ma quindi cosa sono questi volumi logici? L'idea di fondo è di sostituire al concetto di partizione un diverso contenitore (il *Logical Volume*) il quale non dipende più dalle limitate informazioni contenibili nell'MBR ma, tramite le gestione del kernel Linux, si può liberamente spostare/raggruppare/estendere da un settore all'altro di ogni disco, quasi come se fosse un file o una cartella all'interno del file system.
All'atto pratico la modalità di funzionamento dell'LVM può essere riassunta nel seguente modo. L'unità principale sulla quale si basa il sistema è il *Volume Group*. Detto VG verrà da noi dotato di una certa capacità di memoria pari alla capacità di tutti i supporti di massa che gli assegneremo. Tali supporti possono essere dischi SATA, PATA, USB, configurazioni RAID, SSD o quant'altro e mi riferirò a essi come *Physical Device*. I PD possono anche non avere alcuna partizione, seppure si consiglia di dotarli

almeno di una partizione primaria siccome altri sistemi operativi, qualora non riscontrino alcuna partizione, potrebbero inavvertitamente sovrascriverli. I PD, quando sono assegnati ad un VG, sono gestiti dall'LVM suddividendo lo spazio disponibile su ognuno di essi tramite una suddivisione in unità fisiche ridotte (di default di qualche MB) dette *Physical Extent*. Per questo motivo, prima di aggiungere un PD ad un VG sarà necessario inizializzarlo proprio per creare su di esso i vari PE. Quando creeremo un *Logical Volume* (ovvero lo spazio che useremo come "partizione") l'LVM gestirà lo spazio virtuale (logico) da noi richiesto suddividendolo in tanti *Logical Extent* tali da permettere di distribuire ogni LE, che altro non è che un pezzo del nostro LV, tra i PD. Questa allocazione avviene facendo corrispondere ad ogni LE un determinato PE secondo necessità.

Da tale modo di spezzettare i LV ed i PD nasce tutta la potenzialità dell'LVM. Infatti, quando all'interno del nostro VG creeremo un LV l'LVM non sarà obbligato a distribuire lo spazio da noi richiesto tra i PD ma tra le PE. Pertanto sarà possibile spostare/allargare/ridurre il LV semplicemente agendo sulla distribuzione delle LE tra i PE disponibili. Ad esempio, quando in questo articolo creeremo il nostro LV in modalità *mirrored*, l'LVM si limiterà semplicemente a sincronizzare alcune PE di un PD con le PE del nostro secondo PD. Pertanto a ogni coppia di PE sincronizzata corrisponderà il medesimo LE (e quindi il medesimo LV).

Per chi, perso tra queste sigle, volesse maggiormente approfondire segnalo alcuni utili HOWTO [2] [3].

Chi è riuscito a seguire fin qui il discorso, ecco un grafico esemplificativo del funzionamento. In questo caso i LE sono omessi in quanto corrispondono 1:1 ai PE; ovviamente se vi ci fosse un mirror la corrispondenza tra LE e PE sarebbe 1:2 (un LV fa riferimento a due diversi spazi sui PD). Infine si osservi il LV /home: quest'ultimo è in modalità *striped* ovvero è in parte sul primo PD ed in parte sul secondo PD.



5.3 Configurazione della macchina virtuale e del OS Guest

Nell'articolo è stato fatto esplicito riferimento a KVM. Ovviamente siete liberi di sperimentare la macchina virtuale che preferite. A riguardo dell'installazione di KVM, non essendo questo l'oggetto dell'articolo, vi rimando al wiki di www.debian.org [4] dove potrete trovare i link di riferimento ed i consigli rilevanti. Seppure nel presente testo abbia fatto riferimento esclusivamente ai comandi di shell è possibile, qualora preferiate una interfaccia grafica, utilizzare ad esempio *virt-manager*. Fatte le vostre scelte, iniziamo con la creazione dell'ambiente necessario alla simulazione.

Nota
La guida è stata testata anche su Virtualbox, occorre creare i dischi virtuali come statici e non dinamici

5.3.1 Considerazioni sugli hard disk

Seppure non ci siano limitazioni alle modalità di utilizzo degli hard disk è bene tenere a mente alcune cose, specialmente se si sta parlando di ambienti di produzione. Infatti il principale collo di bottiglia che limita la velocità di esecuzione di qualunque sistema operativo è la velocità di lettura/scrittura su hard disk. Pertanto qualora stessimo eseguendo la simulazione in ambiente virtuale consiglio se possibile di creare i dischi virtuali su hard disk differenti per incrementare le prestazioni di I/O. Infatti così sarà possibile scrivere contemporaneamente su entrambi, soprattutto quando entriamo in modalità *mirrored*.
In merito invece all'applicazione della guida su hardware vi consiglio particolare attenzione qualora stiate utilizzando un supporto IDE (ovvero il vecchio bus di comunicazione precedente al PATA ed al SATA). In questo caso prestate attenzione a far sì, se possibile, che i dischi siano su canali diversi. Infatti se saranno utilizzati uno in modalità master e uno in modalità slave allora si occuperanno la banda disponibile a vicenda (il canale è il medesimo) causando un ulteriore rallentamento del sistema. Con i SATA invece il problema non si pone essendo ogni canale unico.

5.3.2 Predisposizione dell'ambiente virtuale

Per prima cosa predisponiamo i due HD virtuali che utilizzeremo per la simulazione. Questi possono essere anche di dimensioni diverse ma maggiori di almeno 1GB se intendete procedere con la sola installazione testuale, di almeno 5 GB per l'installazione anche di Xserver e di un Desktop Enviroment. Se possibile, consiglio di stare un po' più larghi.

```
$ qemu-img create -f raw /dati_condivisi/hd1.img 8G
$ qemu-img create -f raw /dati_condivisi/hd2.img 8G
```

Avremmo potuto utilizzare anche il comando `kvm-img`, ma questo è comunque un alias di `qemu-img`. In dettaglio, abbiamo creato una immagine di disco in formato (opzione `-f`) raw, ovvero 1:1. Seppure in alternativa sia possibile creare anche immagini ad espansione specificando il formato qcow (crea una immagine del disco compressa), purtroppo tale modalità causa problemi di compatibilità con un sistema LVM.
Infine seppure non indispensabile, per velocizzare l'installazione e la simulazione, consiglio di utilizzare, qualora ne disponiate, due hard disk diversi su cui posizionare i file. In questo modo, potendo scrivere simultaneamente su entrambi i file, la simulazione dell'ambiente mirror avrà prestazioni maggiori.
A questo punto, dopo aver scaricato la iso o la net-install di Debian (net-install per amd64 A o per i386 B), si può procedere all'avvio della macchina virtuale:

```
$ kvm -m 1G -hda /dati_condivisi/hd1.img -hdb /dati_condivisi/hd2.img -cdrom \
    /home/risca/Working_in_progress/debian-506-i386-businesscard.iso -boot d \
    -ctrl-grab -icount 1
```

Mi raccomando di impostare la ram della macchina virtuale tramite l'opzione -m in base a quella disponibile sul proprio sistema. Un buon compromesso potrebbe essere il 50% di quella disponibile. Inoltre controllate la correttezza del path dei vostri hard disk (hda ed hdb) e della iso di installazione di Debian (cdrom).
L'opzione `-boot d` eseguirà all'avvio il boot da CDROM così da permetterci di iniziare l'installazione di Debian nel nostro ambiente virtuale. L'opzione `-ctrl-grab` indica che per liberare mouse e tastiera dall'ambiente virtuale è sufficiente premere il tasto `CTRL` di destra. L'opzione `-icount` permette invece di regolare la percentuale di CPU dedicata alla macchina virtuale. Con il valore `1` la CPU viene distribuita equamente tra Host e Guest.
Inoltre, qualora non abbiate installato un server grafico (Xorg) sul vostro host è possibile connettersi anche via vnc (aggiungere l'opzione `-vnc :0`). Infine la macchina virtuale dovrebbe essere connessa al web tramite l'interfaccia virtuale virtbr1. Verifichiamo:

```
ip addr
...
5: virbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
```

```
UNKNOWN
    link/ether 3a:ed:bb:83:7f:72 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.1/24 brd 10.0.1.255 scope global virbr1
```

In caso vi siano problemi di connessione rimando al seguente mini HOWTO [5].

> **Nota**
> È possibile seguire l'articolo anche in assenza di una connessione web. In questo caso sarà necessario eseguire l'installazione da un CD completo.

5.4 Alcuni comandi utili

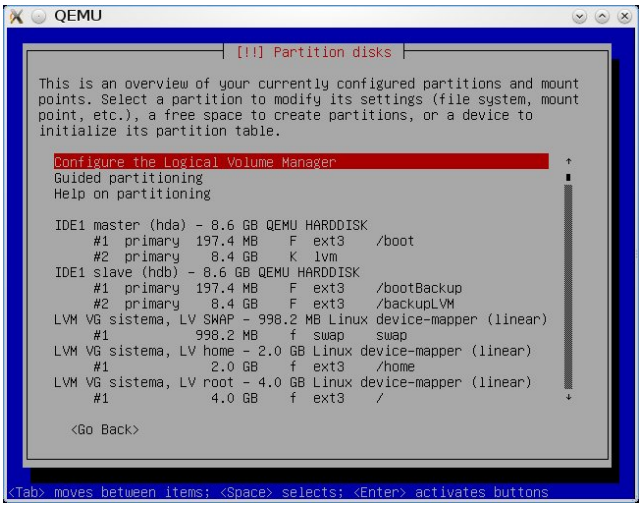
Con l'opzione `-ctrl-grab` fare attenzione alle seguenti combinazioni di tasti:
`ctrl_destro`: cattura/libera mouse dalla finestra di KVM
`ctrl_destro + F`: finestra a schermo intero
`ctrl_destro + $N`: vai alla shell $N
Infine ricordatevi sempre di fare di tanto in tanto una copia dei due dischi. Così, qualora nel corso delle sperimentazioni qualcosa vada storto, il ripristino della situazione iniziale risulterà immediato.

5.5 Installiamo Debian

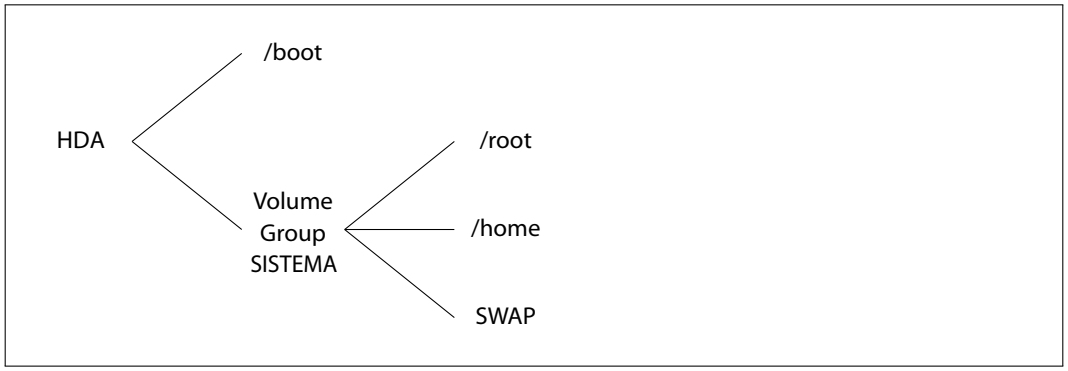
Iniziamo l'installazione di Debian alla solita maniera, indifferentemente se sia da CD o da netinstall, e soffermandoci soprattutto sulla parte riguardante la formattazione degli hard disk.

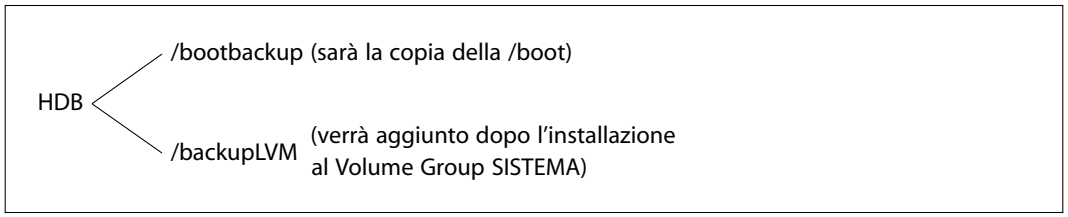
5.5.1 Impostiamo le partizioni

Dopo aver impartito le impostazioni iniziali sulla lingua, la localizzazione e quant'altro dovremmo finalmente arrivare al tool di partizionamento del disco. Entriamo in modalità partizione manuale…



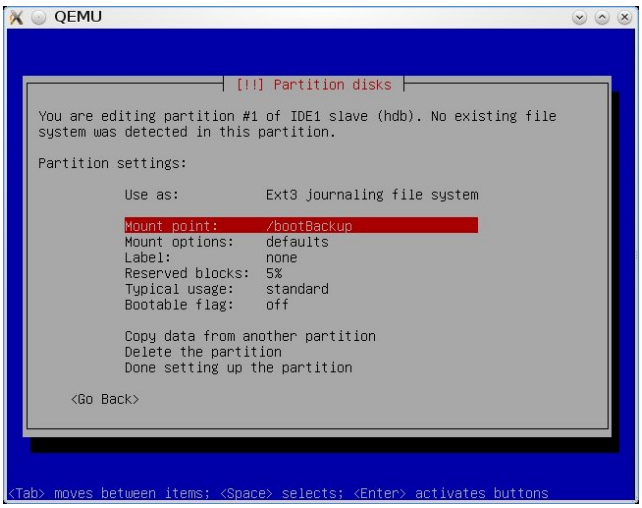
…e ci ritroveremo con i nostri due HD vergini appena creati da noi. Qui proviamo a selezionare un HD, probabilmente verrà richiesta la creazione della tabella di partizione essendo questi completamente formattati, anche nell'MBR. Procedete per entrambi i dischi senza paura.
Ora l'obiettivo sarà di creare la seguente configurazione di partizioni:





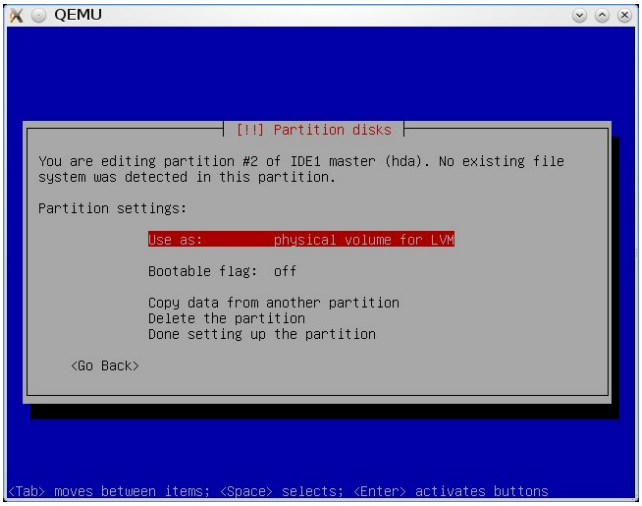
Per raggiungere il nostro scopo procediamo come indicato. Per prima cosa creiamo le partizioni presenti in hdb:

1. selezioniamo lo spazio libero su hdb;
2. selezioniamo create a new partition;
3. impostiamo una dimensione di ca. 200MB (digitare `200M`);
4. di tipo primario (primary);
5. e dall'inizio (beginning);
6. assegnamo un punto di mount manualmente, quale, ad esempio, /bootbackup e usciamo (done setting up the partition).

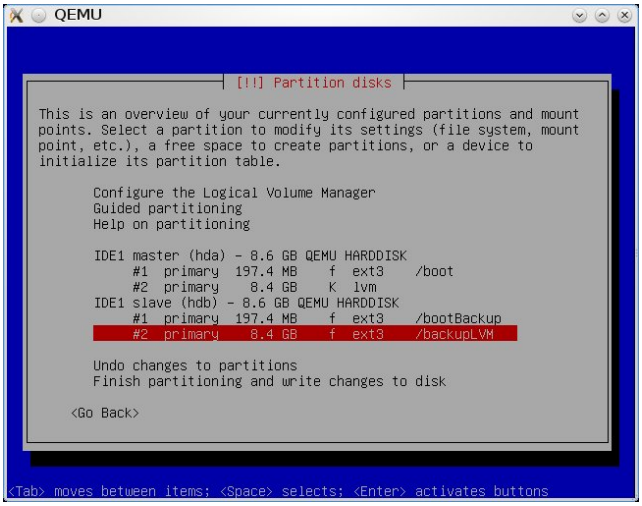


Ripetiamo ora gli stessi passaggi sempre su hdb ma su una nuova partizione assegnandole tutto lo spazio rimanente e nominandola /backupLVM.
Fatto ciò dedichiamoci alla formattazione del primo hard disk, ovvero hda. Per prima cosa, seguendo i passaggi appena sopra proposti, creiamo una prima partizione di 200 MB a cui assegneremo il punto

di mount /boot. A questo punto prestate attenzione: dobbiamo creare lo spazio per il nostro LVM. Pertanto procediamo creando una partizione unica su tutto lo spazio rimanente su hda e alla fine, piuttosto che assegnare un punto di mount, indichiamo di utilizzare la partizione come LVM (`Use as: physical volume for LVM`) e procediamo.



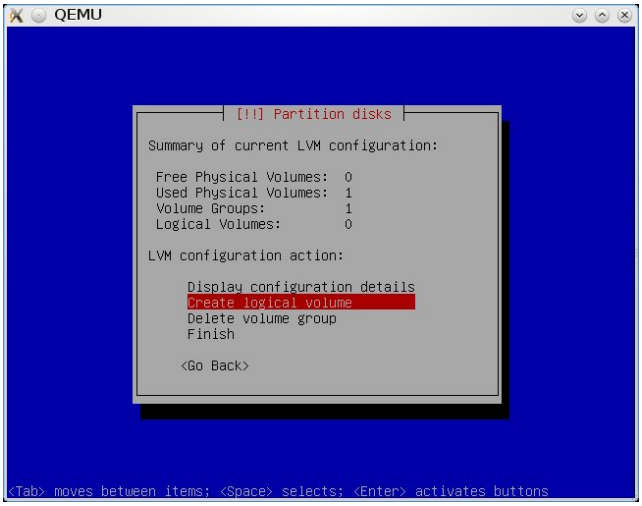
Arrivati fin qui la situazione dovrebbe essere come la seguente e non rimarrà altro che da creare i nostri primi LV all'interno della partizione di LVM.



5.5.2 Impostiamo i LV

Per creare i LV durante la fase di installazione dobbiamo accedere alla prima voce del menù `Configure the Logical Volume Manager` (voce visibile anche nell'immagine precedente), quindi rendere definitive le modiche fino a ora apportate.

Adesso, senza uscire dal menù, dobbiamo per prima cosa creare almeno un VG. Accedendo alla voce `create a Volume Group` creiamo un VG assegnandogli il nome `sistema` e il disco (in questo caso l'unico disponibile) /dev/hda2.



A questo punto possiamo creare i 3 LV di cui abbiamo ancora bisogno (root, home e swap). Sempre dal precedente menù selezioniamo `create logical volume`, indichiamo il VG sistema e, assegnatogli il nome root, diamogli una dimensione di ca. 4GB. Ripetiamo i passaggi appena eseguiti per creare anche un LV home di 2GB ed un LV SWAP di 1GB.
A questo punto, premendo `finish`, torniamo al menù generale di partizionamento e, come abbiamo fatto nel paragrafo precedente con le normali partizioni, andiamo sul LV home, indichiamo di utilizzarlo come ext3 e montiamolo in /home. Alla stessa maniera assegniamo, usando sempre ext3, il punto di mount della root al LV root e indichiamo di usare il LV SWAP come partizione di swap.
Il risultato finale dovrebbe essere il seguente:

```
Partizionamento completato
```

Infine completiamo la formattazione del disco scrivendo le modifiche sullo stesso e terminiamo la fase di installazione, scegliendo a nostra discrezione se installare o meno anche l'ambiente grafico (quest'ultimo richiede ovviamente più spazio e partizioni più capienti rispetto alla sola modalità testuale).

Nota
Si raccomanda di lasciare uno spazio libero di circa il 5% nel VG sistema per dare spazio ai log dei mirror. I log potranno essere mantenuti anche in memoria, ma in questo caso sarà necessaria la resincronizzazione di tutti i mirror a ogni riavvio.

5.6 Creazione dell'ambiente mirrored

5.6.1 Analisi dell'ambiente di lavoro

Se siamo arrivati fino a questo punto, adesso ci troveremo una Debian installata su un HD con un secondo HD libero a nostra disposizione. Pertanto possiamo tranquillamente avviare la macchina virtuale…

```
$ kvm -m 1G -no-acpi -hda /dati_condivisi/hd1.img -hdb /dati_condivisi/hd2.img -ctrl-grab
```

…e fare il punto della situazione. Per prima cosa analizziamo le partizioni disponibili. Tra i vari comandi disponibili (ad esempio `fdisk -l`) ho preferito utilizzare il comando `df` che, tramite una chiara tabella, evidenzia le informazioni per ogni filesystem presente nel sistema:

```
# df -H
```

```
Filesystem               Size   Used  Avail  Use  Mounted on
/dev/mapper/sistema-root 4.0G   3.3G   1.5G   70  /
tmpfs                    531M      0   531M    0  /lib/init/rw
udev                      11M   607k   9.9M    6  /dev
tmpfs                    531M      0   531M    0  /dev/shm
/dev/hdb2                8.3G   153M   7.7G    2  /LVMbackup
/dev/hda1                192M    23M   159M   13  /boot
/dev/hdb1                192M   5.8M   176M    4  /bootbackup
/dev/mapper/sistema-home 2.0G    52M   1.9G    3  /home
```

Si osservino i filesystem indicati. Esclusi quelli di sistema (`tmpfs` e `udev`) abbiamo indicate solo 3 delle 4 partizioni da noi precedentemente create. Infatti la partizione hda2, gestita tramite l'`LVM`, non è formattata con un filesystem dal momento è il PD che in fase di installazione abbiamo assegnato al VG sistema. Su di esso vi è invece un mosaico di filesystem divisi e sparpagliati in una moltitudine di PE. Però è proprio dall'assegnazione eseguita dall'LVM tra PE e LE che viene alloccato lo spazio dei LV da noi nominati root e sistema. Tutti i LV sono astratti dal kernel e raggiungibili tramite un path così composto: /dev/mapper/[Nome_VG-Nome_LV]. Questo path altro non è che un link a un rispettivo disco virtuale gestito dal kernel in /dev/dm-ID.
Inoltriamoci ora nell'analisi dell'LVM partendo dall'identificazione dei VG presenti:

```
# vgdisplay
  --- Volume group ---
  VG Name               sistema
  Format                lvm2
  VG Access             read/write
  VG Status             resizable
  Cur LV                3
  Open LV               3
  Cur PV                1
  VG Size               7.81 GB
  VG UUID               DLd0DF-Rr0p-LnMj-1IJH-v5jK-v5Hi-m5LTvl
```

L'output appena proposto (dal quale è stato estrapolato solo qualche rigo) indica la presenza di un unico VG nominato sistema, contenente 3 volumi logici e composto da un unico disco (il PD hda2). Per analizzare i LV possiamo utilizzare lvdisplay (in tutto e per tutto analogo a vgdisplay) oppure utilizzare il più sintetico lvs (*logical volume show*). Nell'esempio riportato è stato utilizzato lvs specificando di mostrare tutti i LV presenti (`-a` di `all`) con in più l'indicazione del dispositivo dove è allocato ogni LV (`-o +devices`).

```
# lvs -a -o +devices
  LV   VG      Attr   LSize   Origin Snap%  Move Log Copy%  Convert Devices
  home sistema -wi-ao   1.86G
/dev/hda2(1192)
  root sistema -wi-ao   3.72G
/dev/hda2(0)
  swap sistema -wi-ao 952.00M
/dev/hda2(1668)
```

5.6.2 Aggiungere un Physical Device al Volume Group

Per prima cosa aggiungiamo il *physical device* hdb2, da noi chiamato /LVMbackup, al Volume Group `sistema`. Per fare ciò dobbiamo prima di tutto predisporre la partizione desiderata /dev/hdb2 a diventare parte del sistema LVM, pertanto smontiamo la partizione e inizializziamola:

```
# umount /dev/hdb2
# pvcreate /dev/hdb2
  Physical volume "/dev/hdb2" successfully created
```

Il comando `pvcreate` semplicemente crea le PE sul disco indicato. A questo punto è possibile estendere il Volume Group `sistema` alla partizione hdb2:

```
# vgextend sistema /dev/hdb2
  Volume group "sistema" successfully extended
```

Fatto ciò il VG sistema ora dispone di due partizioni site su dischi differenti. Pertanto adesso possiamo procedere a creare i mirrors della home e dalla root sul PD appena aggiunto.

> **Nota**
> A questo punto è necessario modificare la *file system table* rimuovendo dal file /etc/fstab la riga di mount relativa alla partizione /hda2/backupLVM. Altrimenti al prossimo riavvio il sistema cercherà di montare la partizione, seppure ormai inesistente.

5.6.3 Convertire in modalità mirrored i Logical Volume

La modalità *mirrored* garantirà la sicurezza dei nostri dati da eventuali malfunzionamenti hardware. Però possiamo osservare fin da subito che non tutti i nostri dati sono indispensabili. Per esempio la partizione di swap potrebbe essere persa senza danni, pertanto non verrà duplicata permettendo così un risparmio di spazio.
Di solito, quando si desidera creare ex-novo un LV in modalità *mirrored*, è sufficiente impartire il comando lvcreate specificando la dimensione del disco (`-L`), il numero di mirror presenti (`-m 1`), il nome del LV (-n volume1), il VG utilizzato (`sistema`) e il path dei PD da utilizzare, come nel seguente modo:

```
# lvcreate -L 5G -m 1 -n volume1 sistema /dev/hdb2 /dev/hda2
```

Però, dal momento che i nostri LV sono già esistenti, ci limitiamo semplicemente a convertirli da LV lineari ad LV con associata copia in modalità mirror:

```
# lvconvert -m 1 --corelog /dev/sistema/home
  /dev/sistema/home: Converted: 3.4%
  /dev/sistema/home: Converted: 6.3%
  /dev/sistema/home: Converted: 9.9%
  ...
  ...
  /dev/sistema/home: Converted: 97.9%
  /dev/sistema/home: Converted: 100.0%
  Logical volume home converted.
```

Si osservi come appena impartito il comando l'LVM inizi la sincronizzazione tra i due LV.
Il comando appena impartito presenta l'opzione `-m 1` che indica all'LVM di creare una unica copia (sarebbe possibile, per i fanatici della sicurezza, averne anche di più!), invece l'opzione `corelog`, da non sottovalutare, è approfondita nel paragrafo successivo e consiglio di utilizzarla solo nel corso della nostra prova, non su macchine di produzione. Infine, qualora in presenza di più *physical device*, è possibile specificare su quale partizione effettuare la copia indicando, a seguito del comando posto negli esempi, il path indicante il *physical device*. Però nel nostro caso, essendovi due soli dischi, di default la copia è creata nell'altro disco disponibile.
Ovviamente, verifichiamo che quanto fino a ora fatto sia avvenuto a regola d'arte:

```
# pvs
  PV         VG      Fmt  Attr PSize PFree
  /dev/hda2  sistema lvm2 a-   7.81G 376.00M
  /dev/hdb2  sistema lvm2 a-   7.81G   5.95G
# lvs -a -o +devices
  LV              VG      Attr   LSize   Origin Snap%  Move Log Copy%  Convert
```

```
Devices
  SWAP            sistema -wi-ao 952.00M
/dev/hda2(1429)
  home            sistema mwi-ao   1.86G                       100.00
home_mimage_0(0),home_mimage_1(0)
  [home_mimage_0] sistema iwi-ao   1.86G
/dev/hda2(953)
  [home_mimage_1] sistema iwi-ao   1.86G
/dev/hdb2(0)
  root            sistema -wi-ao   3.72G
/dev/hda2(0)
```

Come potete osservare abbiamo ben due PD e la home in modalità *mirrored*. Analizzando l'ultimo log si nota come l'LVM faccia ora corrispondere alla partizione home due immagini differenti (i due mirror della partizione) di cui uno è sul disco hda2, l'altro su hdb2.
A questo punto non rimane che ripetere la medesima operazione con la partizione di root e il nostro LVM sarà settato a dovere:

```
# lvconvert -m 1 --corelog /dev/sistema/root
```

Per concludere ricordiamoci di aggiornare con le modifiche appena effettuate anche il file di sistema /etc/fstab in modo da non avere errori nel corso della fase di boot. In questo caso sarà sufficiente eliminare nella fstab il riferimento alla partizione /LVMbackup la quale è diventata parte del VG sistema. I mirror invece non sono da indicarsi, siccome fanno capo agli stessi LV di root e home già inseriti in automatico nel corso della fase di installazione. In ogni caso qualora ce ne fossimo dimenticati o siano presenti errori sarà possibile apportare modifiche durante la prossima fase di boot.

5.6.4 Osservazioni sull'opzione --corelog

Di default l'LVM, alla creazione di un LV in modalità *mirrored*, crea anche un file sui dischi stessi indicante lo stato di sincronizzazione tra i LV. In questo modo al prossimo riavvio della macchina l'LVM saprà già fino a che punto i dischi sono sincronizzati e si limiterà ad aggiornare solo le parti che non sono segnate come uguali. Questa impostazione risparmia molto lavoro al sistema, specialmente se i dischi in modalità *mirrored* sono molto larghi. Ovviamente il log indicante lo stato di sincronizzazione tra i mirrors occupa uno certo spazio che varia in base alla dimensione del LV ed alla dimensione dei LE. A grandi linee questo spazio può essere stimato in condizioni di default in meno del 5% del LV stesso.
Qualora lo spazio sul disco non sia sufficiente (o più semplicemente non sia importante avere una copia del log sul disco), è possibile tramite l'opzione `-corelog` mantenere solo in memoria lo stato di avanzamento della sincronizzazione dei mirror e quindi risparmiare qualche kilobyte sul disco. Ovviamente il conseguente svantaggio è che ad ogni nuovo avvio della macchina, perdendo i dati in memoria RAM, il computer dovrà eseguire daccapo la sincronizzazione del mirror confrontandolo integramente con l'originale. Considerato che la dimensione del log è contenuta, consiglio pertanto su LV di grandi dimensioni e su macchine desktop di uso quotidiano di utilizzare la cache (quindi di non impartire il comando `-corelog`). Qualora abbiate già creato il mirror con l'opzione `corelog` è possibile modificare successivamente lo status. È sufficiente utilizzare `lvconvert` con l'opzione `mirrorlog` e a seguire `disk` per avere una unica copia del log oppure `mirrored` per avere una copia del log su entrambi i dischi. Ad esempio:

```
# lvconvert -m 1 --mirrorlog disk VG/LV.
```

Purtroppo la questione del log sul disco non è così semplice. Infatti di default l'LVM richiede tre dispositivi: uno per il primo mirror, uno per il secondo ed un ultimo per i log. Ovviamente questa è una limitazione, considerato in special modo che il log non aumenta la nostra resilienza dei dati. È possibile risolvere il problema indicando di allocare i dischi ovunque, nel seguente modo:

```
# lvconvert -m 1 --alloc anywhere VG/LV
```

Nelle prove da me effettuate non sono stato in grado di trasformare direttamente un LV con l'opzione corelog ad avere il mirrorlog sul disco. Pertanto per effettuare la trasformazione mi è stato necessario dapprima trasformarlo in lineare, quindi ricreare il mirror. Si trasforma un LV in lineare con il seguente comando e si può indicare anche il disco sul quale si desidera mantenere il LV:

```
# lvconvert -m 0 /dev/sistema/home /dev/hdb2
```

A questo punto si può ricreare il mirror con l'opzione di default (`mirrorlog disk`) e `alloc anywhere`. Un ulteriore comando che potrebbe risultare utile è pvmove, permette di spostare un LV da un PD all'altro. Ad esempio con pvmove -n /dev/sistema/home /dev/hda2 /dev/hdb2 potrei spostare la home dal disco hda a hdb.

Nota
Seppure abbiate lo spazio a sufficienza per eseguire il mirror è possibile, qualora non indichiate l'opzione `corelog`, che si riceva errore nel caso in cui non vi sia spazio sufficiente per il log. In questo caso o riducete la dimensione del LV (sì, grazie all'LVM è possibile) oppure non potrete creare alcun log sul disco. Se invece volete creare un mirror con la cache sul disco utilizzando due soli PD, allora dovrete utilizzare anche l'opzione `alloc any` come spiegato nel presente paragrafo.

5.7 Ultimi aggiustamenti

5.7.1 Creazione della SWAP

Qualora la partizione di swap non l'aveste già creata nella fase di installazione di Debian oppure se sia essa da ripristinare, procediamo come segue. Se invece la swap è già sistemata passate pure al punto successivo. Per prima cosa creiamo un LV da utilizzare come swap. Pertanto usiamo il comando `lvcreate` assegnando 1 Giga di dimensione (-L) o quanto riteniate voi opportuno, nominiamolo `SWAP` per semplificarci la vita (`-name`) e creiamolo nel nostro VG sistema:

```
# lvcreate -L 1G -name SWAP sistema
```

Fatto ciò non resta che editare il file /etc/fstab in modo tale che il LV appena creato sia utilizzato appunto come swap:

```
/dev/mapper/sistema-SWAP      none     swap      sw       0       0
```

Con il comando `mount -a` possiamo iniziare a utilizzare il nuovo LV da subito, in ogni caso verrà montato dal kernel in automatico dal prossimo riavvio.

5.7.2 Copia della partizione di boot

Per garantire l'avvio indipendentemente da un disco all'altro è ancora necessario copiare la partizione /boot e installare GRUB anche sull'altro HD. Ricordatevi che in caso di aggiornamento del kernel, seppure evento raro, sarebbe opportuno effettuare nuovamente tutte le operazione descritte in questa sezione.
Iniziamo con l'effettuare la copia della boot tramite il semplice comando `cp`:

```
# cp -av /boot/* /bootbackup/
```

5.7.3 Installazione di GRUB

Come ultimo passaggio installiamo GRUB anche su hdb. Siccome i passaggi non sono così scontati consiglio vivamente, prima di iniziare a pasticciare con il bootloader, di eseguire una copia di ripristino

dei due hard disk. Infatti il problema è quello di installare GRUB sui MBR di entrambi i dischi facendogli riconoscere hda e hdb come identici ed intercambiabili.
Prendendo spunto da questa guida [6] sono riuscito a risolvere il problema per GRUB1 (Legacy) come segue. L'idea di base è far sì che entrambi i dischi (hda e hdb) siano per GRUB corrispondenti al medesimo dispositivo (hd0). Quindi per prima cosa editiamo il file device.map come da esempio:

```
# cat /boot/GRUB/device.map
(hd0)   /dev/hda
(hd0)   /dev/hdb
```

A questo punto, se proviamo ad installare GRUB, verremo avvertiti di un errore:

```
# GRUB-install /dev/hdb
The drive (hd0) is defined multiple times in the device map
/boot/GRUB/device.map
```

Giustamente il programma ci avvisa che ad (hd0) sono assegnati due dischi diversi e l'installazione va in palla. La soluzione è eseguire il seguente *work-around* tramite l'installazione direttamente da linea di comando:

```
# GRUB
GRUB> device (hd0) /dev/hdb
GRUB> root (hd0,0)
 Filesystem type is ext2fs, partition type 0xfd

GRUB> setup (hd0)
 Checking if "/boot/GRUB/stage1" exists... yes
 Checking if "/boot/GRUB/stage2" exists... yes
 Checking if "/boot/GRUB/e2fs_stage1_5" exists... yes
 Running "embed /boot/GRUB/e2fs_stage1_5 (hd0)"...  15 sectors are
embedded.
succeeded
 Running "install /boot/GRUB/stage1 (hd0) (hd0)1+15 p (hd0,0)/boot/GRUB/stage2
/boot/GRUB/menu.lst"... succeeded
Done.

GRUB> quit
```

Per GRUB2 invece la situazione è ancora in fermento e a oggi non è ancora possibile eseguire una installazione su entrambi i dischi che permetta il boot da entrambi i supporti. Pertanto, in attesa che il problema sia risolto, consiglio di eseguire il *downgrade* a GRUB Legacy oppure di impratichirsi con la shell rescue di GRUB2 [7]. Infatti tramite essa è possibile avviare qualsiasi sistema presente sul computer.
A questo punto i dischi saranno del tutto identici e intercambiabili!

5.8 Simulazione guasto hard disk e recupero situazione iniziale

Lo svantaggio principale tra un sistema RAID e un sistema LVM è che mentre per il primo, in caso di rottura di un HD, il sistema dovrebbe essere ancora bootabile in automatico, per il secondo è necessario qualche rapido intervento da shell. Il guasto che andremo a simulare sarà la rottura totale di uno solo dei due hard disk, infatti al momento non ho ancora trovato soluzioni di recupero per la rottura di tutti gli HD presenti nel sistema…
Prima di iniziare creiamoci una nuova immagine disco (uno dei due si è rotto, no?). Per semplicità possiamo crearlo come abbiamo fatto all'inizio dell'articolo nel seguente modo:

```
# qemu-img create -f raw /dati_condivisi/hd-nuovo.img 8G
```

Seppure la procedura di ripristino del sistema sia pressoché la medesima, è necessario fare dei distinguo a seconda del disco di cui simuleremo la rottura e del modo di procedere. A proposito valgono le osservazioni dei seguenti paragrafi.

5.8.1 Rottura hda

Questa è una situazione a cui dobbiamo prestare un po' più di attenzione. Il motivo principale è che GRUB si trova solo sul disco superstite, inoltre la fstab ha, quale partizione di boot, quella presente sul disco hda. Per risolvere tali problematiche possiamo intervenire in due differenti maniere.

Sostituire hda con hdb, mettere un hard disk vergine al posto di hdb

In questo caso basta avviare la macchina virtuale nel seguente modo:

```
# kvm -m 1G -no-acpi -hda /dati_condivisi/hd2.img \
    -hdb /dati_condivisi/hd-nuovo.img -ctrl-grab
```

Nella realtà dobbiamo aprire il *case* e cambiare l'ordine dei cavetti in modo che il disco hdb si trovi ora sul primo canale. In alcuni casi è possibile anche invertire l'ordine dal BIOS.
Se questa è la strada intrapresa allora non ci resta che avviare la macchina e procedere al (v. sotto) **Recupero del sistema**.

Installare un nuovo disco vergine al posto di hda

In questo caso dobbiamo, se siamo su una macchina reale, tramite il BIOS indicare di eseguire il boot del secondo hard disk. Nel caso di KVM possiamo utilizzare il seguente comando:

```
# kvm -m 1G -no-acpi --boot menu=on -hda \
    /dati_condivisi/hd-nuovo.img -hdb /dati_condivisi/hd2.img -ctrl-grab
```

In questa maniera, appena si avvia la macchina virtuale, premendo F12 entriamo nel menù di avvio dal quale possiamo selezionare il secondo hard disk. Inoltre, una volta avviata la fase di boot, dovremo entrare nel sistema con la password di root ed editare il file /etc/fstab: adesso la partizione di boot non sarà più /dev/hda1 ma /dev/hdb1. Per il resto possiamo procedere come indicato in (v. sotto) **Recupero del sistema**.

5.8.2 Rottura hdb

Questa è la soluzione più semplice, infatti in automatico il BIOS ricercherà il bootloader sul primo disco. Inoltre la partizione di /boot indicata nella fstab è quella del primo disco (/dev/hda1). Per eseguire la prova con la macchina virtuale avviamo KVM nel seguente modo sostituendo hdb con un nuovo disco vergine:

```
# kvm -m 1G -no-acpi -hda /dati_condivisi/hd1.img \
    -hdb /dati_condivisi/hd-nuovo.img -ctrl-grab
```

A questo punto possiamo procedere come indicato nel punto successivo.

5.8.3 Recupero del sistema

Continuiamo la simulazione, con la macchina virtuale appena avviata come sopra, analizzando ora come recuperare il sistema a iniziare dall'accesso ai LVM. A parte quanto già detto nei due paragrafi precedenti, poco cambia da quale sia il disco superstite. Un ulteriore differenza è l'assenza su hdb della partizione swap, fatto del tutto ininfluente dal momento che essa non è strettamente necessaria all'avvio dell'OS, seppure su macchine datate che presentano meno di 64 Mb di RAM potrebbe rivelarsi un problema.

Appena compare la schermata di GRUB procediamo come se nulla fosse all'avvio del sistema. Quasi subito dovremmo ricevere un messaggio d'errore, infatti il kernel non è in grado di identificare i LV root e home sul nostro VG sistema. Il messaggio d'errore sarà come il seguente, ripetuto un paio di volte:

```
Loading, please wait...
   Couldn't find device with uuid 'vAI01-asd2-...-...'
   Couldn't find all physical volumes for volume group sistema.
   ...
   Volume group "sistema" not found
   ...
```

Qui il computer si prenderà una lunga – molto lunga – pausa di riflessione (ma non disperate, basta avere pazienza e aspettare), fino a quando desisterà e finalmente ci cederà il prompt dei comandi:

```
(initramsf) _
```

Finalmente possiamo intervenire! Il problema verificatosi è che, essendo il VG sistema incompleto di uno dei suoi PD, il sistema non riesce a leggere il nostro VG e, dopo una estenuante ricerca del PD scomparso, chiede il nostro aiuto. Allora entriamo nel prompt di gestione di LVM e risolviamo questo primo inconveniente:

```
(initramsf) lvm
lvm> _
```

e indichiamo all'LVM di montare tutti i LV disponibili (`-ay`) seppure il nostro VG sia incompleto (ovvero sia `-Partial`):

```
lvm> vgchange -ay -P
```

Ovviamente ci verrà ancora restituito un qualche errore (un HD è pur sempre mancante), però verremo avvisati che almeno alcuni LV sono attivi. Per risolvere ora tutti i rimanenti problemi riduciamo il VG sistema al solo PD superstite eliminando dal VG i PD mancanti (ovvero `missing`):

```
lvm> vgreduce --removemissing sistema
```

… et voilà!
Usciamo e lasciamo procedere con l'avvio:

```
lvm> quit
(initramsf) exit
```

A questo punto a volte, nel corso di alcune simulazioni, mi è capitato un *kernel panic*. La causa non è chiara dal log, però riavviando il sistema con le stesse modalità di prima ho sempre risolto il problema. Potrebbe essere che in alcuni casi il bootloader, dopo aver reso accessibile l'LVM non sia in grado di caricare subito dopo la partizione di root (ma queste sono al momento solo ipotesi).
Aspettate! Debian chiede ancora il nostro aiuto. Infatti seppure adesso l'LVM sia propriamente settato, sarà ancora necessario correggere la *File System Tab* siccome adesso sono assenti le partizioni di uno dei due dischi.

```
Give root password for maintenance
(ot type Control-D to continue):
```

Inseriamo la password di root per accedere alla console. Quindi eseguiamo per precauzione una copia della fstab e poi editiamola:

```
# cp /etc/fstab /etc/fstab.backup
# vim /etc/fstab
```

Dal file fstab dobbiamo eliminare, o anche solo commentare, tutte le partizioni ora non più presenti. Quindi nel caso in cui l'hard disk superstite sia:

- HDA: eliminiamo il riferimento a /dev/hdb1
- HDB: commentiamo la linea di riferimento della swap (questa era nell'LVM sul PD hda) e
  - nel caso hdb non sia stato invertito di ordine: commentiamo hda1 e assegnamo a hdb1 il punto di mount /boot
  - nel caso hdb sia stato inserito al posto di hda: commentiamo anche il riferimento a hdb1 (la nostra prima partizione di hdb è ora interpretata dal sistema come hda)

Questa fase merita più attenzione se si utilizza la nomenclatura con uuid (Debian Squeeze e successive), sarà infatti da indicare il riferimento all'uuid corretto e non al dispositivo. Se necessario sarà possibile aiutarsi del comando `blkid` seppure in questa fase troviamo già tutti i riferimenti di cui abbiamo bisogno all'interno della fstab.
A questo punto terminiamo con `exit` la shell e continuiamo il boot, il sistema sarà adesso pienamente utilizzabile.

5.8.4 Ripristino della situazione iniziale

Adesso il sistema è di nuovo pienamente utilizzabile, però abbiamo uno dei due dischi completamente formattato e non c'è più il sistema di ridondanza con LVM. Quindi i nostri dati adesso non sono più a prova di guasto di hard disk! Pertanto procediamo al più presto al ripristino della situazione iniziale. Le utility che fanno al caso nostro sono `fdisk` per la partizione del PD e mkfs per la creazione del filesystem. Procediamo.

5.8.5 Ripristino partizioni

A seconda di quale disco abbiamo deciso di "rompere" e in base a come abbiamo proceduto potremmo trovarci ad avere l'hard disk contenente i nostri dati sia in hda che in hdb. Nulla cambia nel procedimento da eseguire, siate solo accordi a formattare il supporto giusto, ovvero quello vuoto! Attenzione che eventuali errori in questa fase non perdonano e possono comportare la perdita certa di tutti i nostri dati. Siate attenti a quale disco formattate! Per prima cosa verifichiamo che tutti e due i PD siano correttamente visualizzati e verifichiamo l'effettiva posizione del nuovo hard disk. Da root eseguiamo una rapida verifica con `fdisk`:

```
# fdisk -l
Disk /dev/hda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00000000

Disk /dev/hda doesn't contain a valid partition table

Disk /dev/hdb: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000364da

   Device Boot      Start         End      Blocks   Id  System
/dev/hdb1               1          24      192748+  83  Linux
/dev/hdb2              25        1044     8193150   83  Linux

Disk /dev/dm-0: 3997 MB, 3997171712 bytes...
```

Dal precedente comando abbiamo verificato che, in questo caso, le partizioni salvate si trovano su hdb mentre l'hard disk vergine (senza partizioni) è hda. Ripristiniamo su quest'ultimo le due partizioni iniziali. Per fare ciò dobbiamo:

1. avviare l'utility `fdisk` su /dev/hda

2. aggiungere una prima partizione di 200M
3. aggiungere una seconda partizione primaria su tutto lo spazio disponibile
4. assegnare alla seconda la descrizione di tipo di partizione LVM

```
# fdisk /dev/hda
The number of cylinders for this disk is set to 1044.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/hda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x9f775ea0

   Device Boot      Start         End      Blocks   Id  System

Command (m for help): o
Building a new DOS disklabel with disk identifier 0x77fb1f68.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

The number of cylinders for this disk is set to 1044.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1044, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1044, default 1044): +200M

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (26-1044, default 26):
Using default value 26
Last cylinder or +size or +sizeM or +sizeK (26-1044, default 1044):
Using default value 1044

Command (m for help): t
Partition number (1-4): 2
```

```
Hex code (type L to list codes): 8e
Changed system type of partition 2 to 8e (Linux LVM)
Command (m for help): p

Disk /dev/hda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x77fb1f68

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1               1          25      200781   83  Linux
/dev/hda2              26        1044     8185117+  8e  Linux LVM

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Adesso possiamo creare il file system ext3 sulla prima nuova partizione (sulla seconda non è necessario siccome verrà inglobata nell'LVM):

```
# mkfs -t ext3 /dev/hda1
```

A questo punto siamo tornati nella stessa situazione in cui eravamo subito dopo l'installazione di Debian. Pertanto seguiamo nuovamente i passaggi indicati nel punto 4 - **Creazione dell'ambiente mirrored** in modo tale da ripristinare la situazione di sicurezza e ridondanza iniziale.

5.9 Avvertenze su GRUB e LVM

Nell'articolo proposto forse qualcuno avrà osservato che ho preferito non inserire la partizione di boot all'interno dell' LVM. Il motivo principale è di garantire un più facile ripristino del sistema in caso di eventuali malfunzionamenti, in particolar riferimento ad eventuali errori del boot loader. Infatti a oggi il bootloader GRUB è perfettamente in grado di gestire senza problemi le partizioni su LVM, /boot compresa. Però potrebbe capitare di trovarsi in una situazione in cui, per qualsiasi motivo, GRUB non riesca a caricare il modulo che gestisce l'LVM. Allora in questo caso, se si ha la /boot separata, sarà comunque possibile tramite la rescue shell accedere a essa, caricare il modulo LVM e procedere con l'avvio del sistema.

5.10 Considerazioni finali

Con questo articolo ho avuto il piacere di illustrare i vantaggi della virtualizzazione per eseguire test nonché il piacere di introdurvi al *mirroring* con l'LVM. Ma l'LVM è molto più della semplice modalità mirror. Infatti i maggiori vantaggi, i quali mi hanno portato ad adottare l'LVM su tutti i miei computer, derivano proprio dalla libertà con cui è possibile gestire la dimensione delle partizioni: è possibile aumentare on-line lo spazio di un LV così come ridurlo da off-line. Inoltre è possibile spostare i LV o i VG da un PD ad un altro con estrema semplicità, non si è più vincolati al rigido limite imposto dal tradizionale partizionamento!
Un'ulteriore comodità è la possibilità di effettuare *snapshot*. Queste fotografie di un LV possono essere usate benissimo come punto di ripristino nel caso dobbiate, ad esempio, effettuare modifiche rischiose alla vostra Debian. Oppure con lo *snapshot* è possibile eseguire il backup di un server web con MySQL garantendo l'integrità del database.
Ricordate inoltre che un modo utile per risparmiare spazio è mettere in modalità mirrored solo lo stretto necessario; ad esempio la swap e la /tmp posso essere su un semplice LV lineare.

Infine ricordate che per lavorare sulla partizione di /root o di /home in modalità offline si può ricorrere a una live.
L'articolo termina qui, ma vi invito a proseguire per conto vostro le sperimentazioni sull'uso del Logical Volume Manager. Ulteriore materiale informativo lo potrete trovare anche sul sito di debianizzati.org oppure tramite il forum o il canale IRC.

5.11 Ringraziamenti

Concludo ringraziando tutto lo staff dell'e-zine con il quale è stato un piacere collaborare, tra cui il debianizzato mm-barabba per l'attenzione posta alla revisione dell'articolo e ai consigli forniti. *Last but not least* tutta la comunità Debianizzati e i nostri lettori.
Arrivederci al prossimo numero!

Risca

5.12 Risorse di rete

Software

[A] http://cdimage.debian.org/debian-cd/5.0.6/amd64/iso-cd/debian-506-amd64-businesscard.iso
[B] http://cdimage.debian.org/debian-cd/5.0.6/i386/iso-cd/debian-506-i386-businesscard.iso

Sitografia

[1] http://it.wikipedia.org/wiki/RAID
[2] http://en.wikipedia.org/wiki/Logical_volume_management
[3] http://tldp.org/HOWTO/LVM-HOWTO/index.html
[4] http://wiki.debian.org/KVM
[5] http://www.linux-kvm.org/page/Networking
[6] http://GRUB.enbug.org/MirroringRAID
[7] https://help.ubuntu.com/community/GRUB2/#Command20Line%20and%20Rescue%20Mode

6 Introduzione ai kernel: parte seconda

6.1 Introduzione

Nel numero precedente si è sollevata la questione riguardante la definizione di "sistema operativo". Data la difficoltà ad affrontare un argomento così vasto e dispersivo, si è cercato di dare una risposta a tale questione non tramite l'analisi, bensì adoperando una simulazione di progetto. L'importanza di un simile approccio risiede nell'apprendere come, dato un problema, esso possa essere risolto in un numero elevato di modi; in base a come lo si osserva, a quali aspetti si da più importanza, alle risorse disponibili e a molti altri motivi otterremo, per il medesimo problema, soluzioni differenti non solo nell'implementazione, ma anche nell'efficienza.
Visti i presupposti, in questo articolo analizzeremo i moderni approcci alla progettazione di un kernel, cercando di capire le motivazioni che spingono un progettista a fare alcune scelte a discapito di altre. Se il primo articolo ha cercato di mantenere una forma semplice e discorsiva, spesso sacrificando i dettagli, in questo capitolo si dovrà rinunciare alla semplicità; verrà pertanto fatto un uso elevato di esempi in modo da aiutare la comprensione; conoscere qualche rudimento di programmazione può essere d'aiuto, ma non è una condizione del tutto necessaria.

6.2 Il kernel

6.2.1 Premessa

Di derivazione inglese, tardo '500, il termine «kernel» significa «nocciolo» nel senso figurativo di «parte centrale di qualcosa».
In ambito informatico il kernel fa riferimento al nucleo di un sistema operativo, ovvero a quella porzione di software la cui presenza è essenziale per il suo funzionamento; questo è il motivo per cui spesso i due termini acquisiscono lo stesso significato e di conseguenza le stesse ambiguità.
Come risulta difficile definire i limiti entro i quali si parla di sistema operativo o di software opzionale, così anche per il kernel vi sono scuole di pensiero differenti; per rifarci all'infrastruttura di esempio citata nell'articolo precedente, potremmo comprendere gli strati dal primo al quarto, oppure limitarci ai primi due includendo qualche caratteristica addizionale. Vedremo in seguito quali sono le soluzioni maggiormente adottate, per ora limitiamoci a studiare quali sono i requisiti minimi di un kernel.

6.2.2 Meccanismi di base

Vediamo per cominciare quali sono gli strumenti essenziali che accomunano i kernel moderni: se l'intenzione fosse quella di creare un supporto minimale dovremmo identificare un numero esiguo di meccanismi da includere in esso.
Gli obiettivi principali di un sistema operativo generico sono due: amministrare le risorse del calcolatore e fornire una piattaforma di supporto per i compiti che l'utente intende svolgere. Tutto ciò si traduce in un primitivo meccanismo che permetta l'esistenza di una "istanza di lavoro" e in uno strumento che permetta la corretta gestione dei dispositivi presenti all'interno del calcolatore.

Il processo

Ciascun moderno sistema operativo esporta un concetto fondamentale: il processo. Tramite tale nozione si implementano caratteristiche molto importanti, prima fra tutte la multiprogrammazione, ovvero la possibilità di eseguire più compiti in parallelo, dando l'illusione all'utente che il tutto stia avvenendo contemporaneamente pur avendo a disposizione un solo processore.
Il processo è la trasposizione del concetto di lavoro o *task* in ambito informatico; esso rappresenta un compito da portare a termine nella sua interezza. Tale entità sarà perciò composta dal suo codice (l'insieme di istruzioni da eseguire), dai suoi dati e dalle informazioni aggiuntive utili al suo impiego; esse riguardano principalmente i file aperti (ovvero i dispositivi in uso dal programma) e le indicazioni relative alla schedulazione del processo stesso.
Sarà compito del kernel fornire il supporto più completo possibile al processo, le implicazioni principali sono le seguenti:
per la CPU

- Creazione e terminazione
- Cambio di contesto (cambio del processo in esecuzione sulla CPU)
- Schedulazione
- Comunicazione interprocesso (tramite IPC o memoria condivisa)

per la memoria

- Allocazione (creazione) degli spazi necessari ai processi
- Deallocazione degli spazi relativi ai processi esauriti
- Gestione degli spazi associati e liberi

Unix rappresenta il processo come una struttura, definita PCB (*Process Control Block*), realizzata tramite il linguaggio C; in essa sono contenuti i puntatori agli spazi di memoria in cui alloggiano tutte le informazioni appartenenti al processo stesso. In questo modo ciascun processo appare come una entità autonoma e completa, una scatola chiusa nel cui interno vi è tutto il necessario.
Ecco come appariva il PCB nel primo Kernel di Linux (linux-0.01) [1]:

```
struct task_struct    {
    long state;    /* Stato del processo, in principio poteva essere:
     - 1 non eseguibile; 0 eseguibile; > 0 fermo    */
    long counter;    /* Contatore interno al processo    */
    long priority;   /* Priorita associata al processo    */
    long signal;     /* Campo adibito alla gestione dei segnali
    (comunicazione interprocesso IPC)    */
    fn_ptr sig_restorer;    /* Campo adibito alla gestione dei segnali
(comunicazione interprocesso IPC)   */
    fn_ptr sig_fn[32];    /* Campo adibito alla gestione dei segnali
(comunicazione interprocesso IPC)   */
    int exit_code;    /* Stato di uscita del processo (0 success, != 0
failure)    */
    unsigned long end_code, end_data, brk, start_stack;    /* Puntatori
agli estremi delle locazioni in memoria di codice e dati    */
    long pid, father, pgrp, session, leader;    /* Fra i vari campi si
identificano il pid (process id) e il pid del padre    */
    unsigned short uid, euid, suid;    /* uid (user id) reale, effettivo e
quello salvato al momento dell'avvio del processo    */
    unsigned short gid, egid, sgid;    /* gid (group id) reale, effettivo e
quello salvato al momento dell'avvio del processo   */
    long alarm;    /* In questo campo è possibile impostare il timer per il
segnale SIGALRM    */
    long utime, stime, cutime, cstime, start_time;    /* Statistiche
temporali del processo    */
```

```
    unsigned short used_math; /* file system info */
    int tty;       /* Nome del terminale sul quale il processo è in
esecuzione; - 1 se non appartiene a nessun terminale    */
    unsigned short umask;    /*  Maschera del processo    */
    struct m_inode * pwd;    /*  Percorso della directory in cui il processo
opera    */
    struct m_inode * root;   /* Percorso della radice del filesystem    */
    unsigned long close_on_exec;
    struct file * filp[NR_OPEN];    /* Elenco dei file aperti */
    /* ldt for this task 0 - zero 1 - cs 2 - ds&ss */
    struct desc_struct ldt[3];
   /* tss for this task */
    struct tss_struct tss;    /* Struttura in cui sono memorizzati i
contenuti dei registri della CPU    */
};
```

Agendo sulle singole variabili interne è possibile modificare radicalmente le proprietà del processo in esecuzione; per esempio intervenendo sul valore del campo `priority`, attraverso la `system call nice()` (vedremo più avanti cosa sono le system call), si cambia la priorità associata al processo, mutando di conseguenza il comportamento dello schedulatore nei confronti dello stesso.

Le interrupt

Nel numero precedente abbiamo visto come le interrupt siano il mezzo con cui il sistema operativo è in grado di avvertire il presentarsi di un evento.

Occorre perciò fornire una infrastruttura che permetta il funzionamento di tale meccanismo; ciò non significa implementare completamente la gestione delle interrupt, non sarebbe possibile prevedere tutti i segnali offerti dalle architetture oggi in commercio, stiamo progettando un sistema operativo per una macchina generica, non per un dispositivo specifico. Saranno i programmatori dei livelli superiori a definire le politiche e i meccanismi relativi ai vari segnali che si intendono realizzare.

A questo livello si definiscono semplicemente i meccanismi di ricezione e riconoscimento dei segnali e si dedica un'area di memoria per poterli ospitare; dopo di che si restituiscono ai programmatori delle funzioni, dette «primitive», che a ogni avvio della macchina (la sua configurazione potrebbe cambiare fra un avvio e un altro) popoleranno correttamente tale memoria. Osservando la parte iniziale dei log del kernel si può notare che fra le prime operazioni eseguite all'avvio vi è proprio il caricamento delle interrupt.

6.3 Kernel mode

Con la gestione di processore, memoria e interrupt si è in grado di coprire le componenti elementari di un calcolatore generico, naturalmente è possibile scorporare ulteriormente alcuni strumenti, viene però da chiedersi quanto ciò sia utile ai fini della progettazione.

Poiché abbiamo identificato questi meccanismi come essenziali per il nostro sistema operativo, sfruttando il bit di modo messo a disposizione dalle moderne architetture, attribuiremo ad essi i privilegi di sistema; le *routine* (alias «funzioni», v. la sezione seguente) incluse in questo livello opereranno in modalità kernel con il flag di modo settato a `0`.

Una ulteriore caratteristica da includere in tale spazio operativo è l'inibizione dell'ascolto delle interrupt per evitare di compromettere il funzionamento di procedure così delicate.

Tale scelta progettuale implica conseguenze significative: ogni volta che il sistema risiederà in modalità kernel sarà del tutto "sordo" agli eventi esterni, questo aspetto va a influire sui tempi di risposta del sistema operativo agli stimoli dell'utente, perciò entrare troppo spesso in tale spazio o restarci troppo a lungo influirebbe notevolmente sull'usabilità dell'intero sistema.

6.4 Interfaccia del kernel, le system call

6.4.1 Premessa

Abbiamo analizzato una struttura generica nonché minimizzata di un kernel, l'ultima questione che rimane da approfondire è l'interfaccia con cui usufruire dei servizi offerti da quest'ultimo. Ricordiamo infatti che, secondo il principio di astrazione, il kernel deve apparire agli strati superiori come un pacchetto chiuso, con il quale sarà possibile interagire attraverso un insieme di direttive semplici, ma complete.
L'interfaccia di un kernel costituisce parte dei comandi più vicini alla macchina fra quelli messi a disposizione, essi ovviamente faranno riferimento all'insieme degli strumenti proposti dal kernel stesso, perciò il loro numero e tipologia varia di progetto in progetto.
L'informatica rappresenta un insieme di procedure tramite funzioni; generalmente le funzioni utili alla comunicazione con un kernel sono definite «funzioni primitive». Invocare una funzione significa inizializzare e avviare le istruzioni per il calcolatore in essa contenute; si parla perciò di «chiamate a funzioni primitive di sistema», più brevemente, «chiamate a sistema» (in inglese: *system call*, spesso abbreviato in *syscall*). Le syscall sono l'unico modo che il programmatore ha a disposizione per interagire con il kernel.
Vediamo in conclusione un paio di esempi di system call messe a disposizione da Unix.

6.4.2 fork() e exec()

La funzione in C `fork()` è l'unico metodo disponibile per generare un nuovo processo, essa duplica gli spazi di memoria relativi al programma in esecuzione; al termine della chiamata ci troveremo due processi, uno il duplicato dell'altro e con un legame di parentela, perciò sono definiti nel linguaggio comune processo «padre» e processo «figlio».
La `exec()` permette di sostituire il codice residente nel processo (sovrascrive il contenuto relativo al codice eseguibile all'interno del PCB) con altro codice arbitrario; in questo modo si può lanciare un programma all'interno del proprio processo cambiandone radicalmente il contenuto.
Vediamo un esempio classico sull'utilizzo di queste funzioni.
All'apertura di un terminale ci si ritrova con una linea di comando in attesa di una vostra disposizione, nella maggior parte dei casi tale comando risulta in realtà essere un programma da lanciare con i relativi parametri da associare. Supponiamo di lanciare il comando:

```
$  mkdir miacartella
```

Ecco, a grandi linee, cosa accade all'interno del nostro sistema operativo:

- Invocazione della fork(), una system call
- Passaggio alla modalità kernel, il bit di modo va a `0`. Da questo momento il sistema non reagisce ad eventuali interrupt
- Interruzione del processo in corso sulla CPU e caricamento del codice relativo alla fork() (cambio di contesto)
- Duplicazione della memoria del processo padre (il terminale)
- Il processo figlio avrà a disposizione tutte le informazioni del padre, non condivise, duplicate in un'area di memoria distinta.
- Uscita dalla modalità kernel, bit di modo a `1`. Si è nuovamente sensibili alle interrupt
- Restituzione del controllo del processore al programma.

A questo punto abbiamo due processi detti «concorrenti» poiché entrambi concorreranno all'uso del processore (sarà lo scheduler a decidere l'ordine di accesso). Il programma prende due strade differenti: il processo padre (il terminale) semplicemente si mette in attesa del termine del figlio (invocando la syscall apposita `wait()`); quando ciò si verificherà esso raccoglierà lo stato di terminazione e seguirà le istruzioni del programmatore; vediamo invece come prosegue il cammino del figlio:

- Lettura dell'area di memoria in cui è localizzato il codice relativo al programma `mkdir`
- Invocazione della system call `exec()`.
- Passaggio a modalità kernel e cambio di contesto
- La exec() rimpiazza il codice all'interno del processo, che ricordiamo essere ancora il codice relativo al terminale, col codice del nuovo programma richiesto dall'utente (`mkdir`)
- Allocazione della memoria relativa al nuovo programma e lettura dei parametri passati da linea di comando (in questo caso la stringa miacartella)
- Uscita dalla modalità kernel, il controllo è restituito al programma

All'inizializzazione di un nuovo programma da eseguire, quindi, verrà generato un nuovo PCB identico a quello del processo che ha invocato la `fork()`. Sarà la `exec()` a trasformare il nuovo PCB, operando sui campi in esso contenuti, in modo tale da renderlo adeguato al programma che si intende lanciare. Questi sono alcuni dei meccanismi che stanno dietro alla creazione e manipolazione dei processi in Unix.
Ogni qual volta voi aprite un nuovo programma, sia che lo lanciate da terminale, sia che utilizziate l'ausilio di una interfaccia grafica, queste sono le direttive che il kernel Linux riceve ed esegue.
In conclusione il kernel è uno strumento che esporta un insieme chiuso di servizi accessibili attraverso un'interfaccia composta da funzioni specifiche. I programmatori che decidono di appoggiarsi direttamente a tale strumento per sviluppare i propri software, devono conoscere bene il comportamento delle system call; ciò non implica la conoscenza dell'implementazione delle stesse, il punto di forza del principio di astrazione consiste proprio nello svincolare il programmatore da tali problematiche.

NoxDaFox

6.5 Risorse di rete

[1]: http://e-zine.debianizzati.org/source/linux-0.01.tar.bz2

7 Principali tipologie di kernel

7.1 Principali tipologie di kernel

Se si volesse prendere in analisi esclusivamente il kernel di un sistema operativo si noterebbe una distinzione piuttosto netta fra le varie categorie attualmente impiegate. Infatti, rimanendo esclusivamente nell'ambito dei kernel, ciò che differenzia una tipologia dall'altra è semplicemente il grado di astrazione che essi operano sulla macchina.
Un kernel monolitico, come si evince dal nome stesso, tenderà a includere l'intera gestione dell'hardware al suo interno, mentre un microkernel si limiterà a contenere i servizi minimali utili al funzionamento di un sistema operativo.
È importante comprendere come l'impiego di un determinato kernel all'interno di un progetto influenzi radicalmente le scelte successive. Le differenze più sensibili sono concentrate nello sviluppo dei servizi sovrastanti il kernel e non nello stesso.
Andiamo quindi ad analizzare le diverse classi di kernel sotto questo aspetto, valutando soprattutto le conseguenze che comporta la scelta di una soluzione rispetto ad un altra.

7.1.1 Kernel monolitici

Come già accennato, l'approccio "monolitico" alla progettazione di un kernel prevede l'inclusione della totalità dei moduli all'interno dello stesso. Si fa riferimento in particolare, oltre ai meccanismi spiegati in precedenza, alla gestione di:

- filesystem
- *device driver* di ogni categoria: dalla scheda video alla stampante USB
- *networking*: praticamente l'intera pila protocollare ISO/OSI
- sicurezza dei dati e crittografia
- servizi opzionali quali emulatori e supporto alla virtualizzazione

Il risultato è un file binario (ovvero un file eseguibile) di elevate proporzioni che viene interamente caricato in memoria all'avvio della macchina. Lo spazio di memoria dedicato è uno solo, compatto e accessibile solamente dal kernel stesso. Le risorse interne sono completamente condivise, qualsiasi procedura può avere accesso all'intero segmento di memoria dedicato al kernel; per esempio, il modulo relativo alla gestione del filesystem può accedere direttamente alle risorse contenute nel modulo di gestione degli hard disk senza dover far uso di procedure che lo mettano in comunicazione con esso. Allo sviluppatore non rimane che studiare bene l'interfaccia (costituita dalle system call proposte) per poter sviluppare il proprio software applicativo; si è soliti procedere alla stesura di un insieme di librerie che, sfruttando i servizi offerti dal kernel, semplifichino ulteriormente l'architettura, il tutto nel pieno rispetto del principio di astrazione. Il cammino è semplice: dal sottosistema X.Org, passando per i *toolkit* basati su di esso (Qt e Gtk, per citarne alcuni) per giungere infine ai vari Desktop Environment con le relative applicazioni utilizzate dagli utenti nel quotidiano.

Il modello presentato (ovviamente semplificato) costituisce il classico sistema operativo a cui gli utenti di debianizzati.org sono abituati. Alla domanda: «Che cosa è il kernel?» è ora facile rispondere: «Il software che gestisce interamente la macchina, sul quale, attraverso astrazioni successive, si appoggiano tutti i programmi utilizzati dall'utente».
Vediamo le conseguenze che presenta la scelta di adottare una simile soluzione all'interno di un sistema operativo.
La totale condivisione delle risorse permette innanzitutto di ottimizzare le procedure interne, ottenendo prestazioni superiori a tutte le altre soluzioni poiché l'accesso all'hardware avviene in maniera diretta, utilizzando sempre istruzioni privilegiate.
L'architettura di un kernel monolitico, seppur confusionale, è più semplice; gli sviluppatori possono utilizzare qualsiasi risorsa con estrema libertà, non hanno bisogno di introdurre procedure ausiliarie; se occorre, per esempio, accedere a un campo del PCB possono farlo direttamente. Inoltre i programmatori di alto livello si ritrovano ad essere del tutto svincolati dalla gestione della macchina. Questi sono forse gli aspetti che hanno giocato il ruolo di maggior rilievo nel successo dell'approccio "monolitico" alla progettazione di un kernel; il costo di progetto è sempre uno dei principali discriminanti nello sviluppo di una tecnologia.
La scelta di raggruppare l'intera gestione della macchina in un unico file eseguibile porta diversi svantaggi, primo fra tutti l'impossibilità di modificarne la struttura durante l'esecuzione. Qualora risultasse necessario introdurre o cambiare dei moduli, poiché, per esempio, si è sostituito un dispositivo all'interno del calcolatore, non sarebbe possibile farlo se non ricompilando la totalità dei sorgenti. Un software una volta compilato non è più modificabile: occorre ricompilare i sorgenti *ex-novo*.
In secondo luogo, al crescere dei dispositivi supportati e delle tecnologie incluse, consegue l'aumento della dimensione del file binario ottenuto; è opportuno quindi operare un bilanciamento fra i servizi che si intendono proporre e il contenimento del peso in Byte.
Un kernel monolitico pertanto è fortemente statico e tenderà sempre a crescere di dimensione all'avanzare del suo sviluppo.
Altro aspetto di grande importanza è rappresentato dalla stretta condivisione delle risorse: qualora sia presente codice viziato da errori di programmazione, essendo utilizzato e utilizzando lui stesso risorse condivise, sarà in grado di propagare errori all'interno dell'architettura, portando con elevata probabilità al blocco dell'intero sistema. È importante che il codice rilasciato sia altamente stabile e privo di errori, poiché è sufficiente un piccolo difetto all'interno di un modulo per generare un *kernel panic* compromettendo il funzionamento del calcolatore.
In ultima istanza occorre osservare come un sistema operativo con una simile tecnologia passi molto tempo in modalità kernel; ogni aspetto della macchina viene gestito all'interno dello spazio privilegiato (con il Bit di modo della CPU a 0); durante queste routine il sistema non è in grado di ricevere e gestire le *interrupt*. Si ottiene perciò un'elevata latenza nella risposta agli stimoli esterni rendendo il kernel monolitico poco indicato per le soluzioni Real Time.

7.1.2 Microkernel

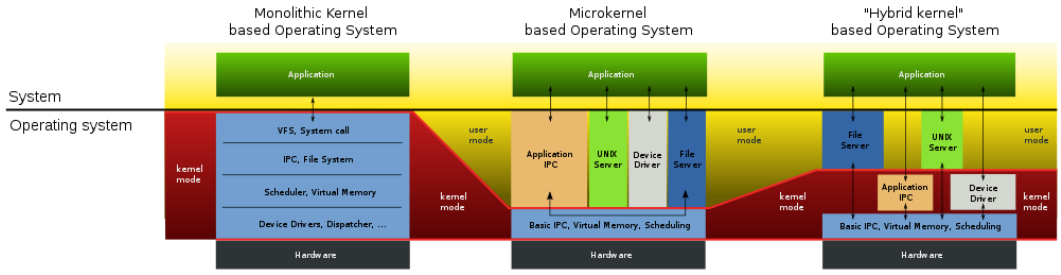
Jochen Liedtke nel principio di minimalità asserisce che un concetto è tollerabile all'interno di un microkernel solamente se la sua esclusione dallo stesso impedisce l'implementazione delle funzionalità di sistema richieste.
Altro importante principio alla base del design di un microkernel consiste nella separazione delle politiche dai meccanismi. Nelle scienze informatiche le politiche (o criteri) rappresentano le scelte che un programmatore intraprende, mentre i meccanismi specificano come tali scelte vanno realizzate; la separararazione dei due aspetti permette di ottenere un sistema maggiormente flessibile.
Per fare un esempio, si immagini una ipotetica gestione della temporizzazione dei processi all'interno di una CPU, ovvero la quantità della risorsa "tempo processore" che viene assegnata a ciascuna unità di lavoro; il meccanismo è la realizzazione dello strumento vero e proprio (lo scheduler), mentre il criterio è la quantità di tempo da assegnare; l'obiettivo è quello di permettere in modo arbitrario la scelta della risorsa (spesso definita *time slice*, porzione di tempo), indipendentemente dal meccanismo di assegnazione implementato.

Un microkernel si limita a fornire un'infrastruttura minimale con un'astrazione dell'hardware piuttosto semplice.
Generalmente i servizi offerti sono la gestione della memoria, il supporto alla multiprogrammazione e la comunicazione interprocesso (IPC). L'insieme di servizi più avanzati viene suddiviso in unità distinte e autonome definite *server*.
I *server* sono progettati per essere eseguiti all'interno dello userspace, lo spazio riservato al codice non privilegiato, possiedono aree di memoria strettamente dedicate e non possono intervenire al di fuori di esse. Solitamente, a differenza dei programmi applicativi, hanno la possibilità di accedere ad alcuni spazi di indirizzamento fisico della memoria: questo per permettere ad alcuni tipi di *server* (ad esempio i *device driver*) di poter scrivere all'interno di eventuali buffer e registri specifici. Al di fuori di questa peculiarità (sempre concessa dal kernel) ciascun *server* viene eseguito esattamente come un'applicazione a livello utente.
La difficoltà principale che si incontra nell'adozione di una simile architettura risiede nella realizzazione della comunicazione interprocesso; se in un kernel monolitico ciascun modulo può accedere direttamente a tutte le risorse disponibili, in un microkernel la situazione risulta essere l'esatto opposto. Ciascun *server* presenta una struttura strettamente dedicata, sviluppata in maniera autonoma; ciò tuttavia non implica una sua indipendenza dagli altri moduli: il *server* adibito alla gestione del filesystem avrà sempre bisogno dei servizi offerti dal *device driver* del disco fisso.
Poichè ciascun modulo risiede all'interno di uno spazio di memoria dedicato, sarà quindi necessario studiare un meccanismo in grado di permettere uno scambio piuttosto intenso di messaggi fra i vari *server*; il numero e la tipologia dei messaggi scambiati è molto difficile da prevedere in fase di progetto, occorre perciò fornire uno strumento svincolato da questa prospettiva (nel pieno rispetto del principio di separazione dei meccanismi dai criteri).
Le difficoltà progettuali che comporta lo sviluppo di un adeguato sistema di IPC e le scarse prestazioni ottenute a causa dell'*overhead* introdotto dalle stesse sono sempre state un freno nello sviluppo delle architetture a microkernel.
Tuttavia i vantaggi offerti da questa soluzione hanno spinto diversi sviluppatori a proseguire su questa strada: il kernel, una volta ben strutturato, non sarà soggetto alla crescita in dimensioni tipica della soluzione esaminata in precedenza; il pieno rispetto del principio di separazione fra criteri e meccanismi permette di ottenere un sistema operativo altamente flessibile, la sua espansione sarà legata alla sola introduzione dei nuovi *server* senza la necessità di modificare in alcun modo le strutture preesistenti.
I *server* vengono eseguiti al di fuori dello spazio riservato al kernel: ciò permette una manutenzione decisamente più semplice. Ciascun *server* può essere arrestato in qualsiasi momento senza che il sistema entri in crisi; è possibile attuare qualsiasi intervento necessario inoltre, qualora si verifichi un crash, sarà sufficiente riavviare il *server* coinvolto e gli applicativi che ne stavano facendo uso.
La differenza di questo approccio rispetto a quello monolitico si fa decisamente marcata se si osserva questo aspetto dal punto di vista progettuale: poiché le risorse in un sistema a microkernel non sono condivise, un eventuale errore nel codice rimarrebbe circoscritto al *server* ospite, rendendo il sistema decisamente più tollerante ai guasti. D'altro canto, come già spiegato, la separazione delle risorse costringe a dover sviluppare un'infrastruttura *ad hoc* che permetta una condivisione di tipo indiretto (le IPC), introducendo serie difficoltà di progetto.
L'ultima prospettiva da considerare riguarda la dimensione del microkernel: l'insieme delle istruzioni privilegiate (ovvero quelle istruzioni sulle quali non si può avere diritto di prelazione) in esso contenute è decisamente minore rispetto a qualsiasi altra soluzione. Ne consegue perciò che il sistema operativo non solo sarà più sicuro, poiché saranno pochissime le *routine* in grado di causare errori irreversibili della macchina, ma sarà anche molto più reattivo. Le latenze in risposta agli stimoli esterni sono decisamente più moderate in un sistema basato su architettura a microkernel, rendendolo di fatto molto più adatto alle applicazioni Real Time.

7.1.3 Kernel ibridi

Una volta comprese le architetture monolitiche e a microkernel è piuttosto facile immaginare in cosa consistano i kernel ibridi. Storicamente sono molti i sistemi che, nati in un ottica di impiego di un microkernel, dopo essersi trovati di fronte a difficoltà di progetto che superavano le risorse a

disposizione, hanno preferito optare per una via di mezzo, evitando così di scartare del tutto il lavoro già portato a termine.
Un kernel ibrido solitamente parte da un'architettura a microkernel, andando però a includere un numero sempre maggiore di *server* all'interno del kernel stesso. Le motivazioni sono dovute principalmente al bisogno di semplificare il progetto: spesso gli sviluppatori si ritrovano con sistemi troppo poco efficienti e, poiché l'ottimizzazione della struttura è una delle soluzioni più costose in termini di risorse e di tempo, preferiscono includere alcuni servizi nello spazio delle istruzioni privilegiate andando ad aumentare semplicità ed efficienza del sistema operativo, senza influire eccessivamente sui costi.
Il risultato è una via di mezzo fra un sistema basato su kernel monolitico e uno basato su microkernel, un buon compromesso per ridurre l'*overhead* introdotto dall'elevato scambio di comunicazioni interprocesso e, nel contempo, per scorporare alcuni elementi dal kernel ritenuti non necessari.
L'immagine sottostante (gentilmente concessa da Wikipedia) è utile a ricapitolare le strutture dei tre kernel presi in esame.



Notare come nelle immagini relative a microkernel e kernel ibrido sia presente un *server* denominato «Unix Server», l'architettura a microkernel presenta diverse similitudini con gli hypervisor delle macchine virtuali, questo permette in via teorica di eseguire l'intero kernel Unix all'interno di un *server* appositamente strutturato; per esempio il Microkernel Fiasco [1] derivato dalla famiglia L4 è in grado di eseguire il kernel di Linux in uno spazio di memoria dedicato.

7.1.4 Exokernel

Il pensiero alla base della concezione di un exokernel consiste nel fatto che solo il programmatore sa quali siano effettivamente le risorse necessarie al funzionamento del proprio software. Obiettivo primario nella progettazione di una simile soluzione è l'eliminazione, per quanto possibile, dell'astrazione dell'hardware; questa idea, in forte contrasto con i princìpi classici dello sviluppo dei sistemi operativi, è molto recente e a oggi sono pochi i progetti che presentano un grado di maturità accettabile anche solo a livello sperimentale.
Un exokernel si limita a fornire la multiplazione (accesso multiplo da parte di più attori) e la minima protezione dell'hardware; le sue dimensioni risulteranno essere decisamente ridotte. Ciascun programmatore avrà libero accesso alle risorse del calcolatore (tempo processore, spazi di indirizzamento della memoria e così via), mentre il kernel si limiterà a garantire che tali risorse non siano impegnate al momento della richiesta. Questa soluzione permette agli sviluppatori di implementare liberamente le proprie soluzioni, applicando algoritmi e costrutti propri senza essere vincolati dalla struttura del software sottostante.
Nell'articolo precedente si erano evidenziati i pregi e i difetti introdotti dall'astrazione di un sistema. La scelta di impiegare un exokernel introduce la possibilità di ottenere prestazioni decisamente superiori (si rinuncia a un *overhead* notevole) a discapito della semplicità di progettazione. L'accesso libero all'hardware comporta il rischio di commettere errori di programmazione non indifferenti, errori in grado di compromettere il funzionamento dell'intero calcolatore; occorre conoscere approfonditamente l'architettura (a livello logico, non elettronico) sulla quale si va a sviluppare il proprio software.
La parte restante del sistema operativo è costituita da librerie di sistema (definite libOS), facenti uso diretto delle risorse rese disponibili dall'hardware della macchina ospite. È possibile sviluppare quante

librerie si ritengano necessarie: un programmatore può scegliere se fare uso di quelle già disponibili o se implementarne di proprie; esse verranno affiancate a quelle esistenti ampliando le possibilità del sistema operativo. Ad esempio si potrebbero sviluppare librerie in grado di esportare le funzioni classiche dei sistemi Unix e affiancarle a quelle tipiche di Windows, permettendo di eseguire contemporaneamente software di entrambe le piattaforme sullo stesso sistema.

Data la giovane età di questo approccio non esistono sistemi stabili, le poche versioni sviluppate sono altamente sperimentali e disponibili solamente all'interno di Università e centri di ricerca. Questa soluzione è sicuramente quella che lascia più spazio alla creatività permettendo di sperimentare algoritmi e strutture dati innovative, tenendo però in considerazione l'elevata probabilità di commettere errori piuttosto disastrosi.

7.2 Conclusione

L'idea dell'articolo nasce dal bisogno di rispondere al classico interrogativo riguardante la natura dei kernel e la loro differenziazione, domanda alla quale si spera di aver dato una risposta soddisfacente.

Una delle principali lacune delle scienze informatiche consiste proprio nella carenza di spiegazioni chiare e semplici riguardo ai principali strumenti di cui fanno utilizzo; solitamente si può distinguere fra i tutorial in cui si impara passo a passo a padroneggiarle e i manuali di riferimento, in cui le architetture sono interamente sviluppate ed esposte in maniera fredda e sintetica. Questo approccio scoraggia le persone ad avvicinarsi alle *Information Technologies* al di fuori del mero utilizzo, rendendo progettisti e sviluppatori dei veri e propri "guru" di una scienza complicata e misteriosa; è evidente come il porre le nozioni in modo diverso possa rendere qualcosa di oscuro e incomprensibile un semplice caso di *problem solving*.

Un sistema operativo è il semplice risultato dell'applicazione dei meccanismi classici dell'ingegneria: studiata approfonditamente l'architettura della macchina ospite, si procede alla risoluzione di un problema che, in questo caso, risulta essere la creazione di un supporto in grado di permettere all'utente di eseguire un elevato numero di mansioni. La fase successiva consiste nella progettazione di un sistema in grado di soddisfare gli obiettivi posti. Per far ciò, data la vastità dei problemi contemplati all'interno della soluzione, si applicano strategie atte a ridurre la difficoltà del processo di creazione; se si procedesse a caso sarebbe difficile ottenere soluzioni decenti in tempi accettabili.

Il kernel è il risultato della trasformazione di un insieme eterogeneo di strumenti e dispositivi, residenti all'interno di un calcolatore, in un pacchetto di servizi software in grado di fornire le basi per lo sviluppo di applicazioni più evolute. L'impiego di una soluzione comporta conseguenze importanti all'interno dello sviluppo di un sistema operativo, la scelta dell'architettura influenza tutto il lavoro successivo andando a incidere in maniera importante sulla percezione che l'utente avrà della piattaforma in uso.

La progettazione di un sistema operativo è un lavoro complesso: esso prevede lo sviluppo di diverse soluzioni e costringe gli sviluppatori a operare diverse scelte. Occorre valutare bene le risorse a disposizione e gli obiettivi preposti prima di procedere con il lavoro, perché la scorretta implementazione di un servizio o una decisione sbagliata possono portare alla morte prematura del progetto.

Come esempio basta pensare al software GNU/Hurd [2], nato ufficialmente nel 1990 è tutt'ora in via di sviluppo. Esso doveva rappresentare la base per il sistema operativo della Free Software Foundation [3] di Richard Stallman, basato su Microkernel Mach [4], ma fu soppiantato dal kernel monolitico Linux di Linus Torvalds per motivi di praticità. A oggi, gli utenti dell'Open Source conoscono il sistema GNU/Linux mentre GNU/Hurd è un sistema utilizzato da una ristretta nicchia di utenti, per lo più sviluppatori.

7.3 Risorse

Risorse bibliografiche

Abraham Silberschatz – Peter Baer Galvin – Greg Gagne, *Sistemi operativi. Concetti ed esempi*, Pearson Education Italia, Milano 2006[7].
Andrew Stuart Tanenbaum, *I moderni sistemi operativi*, Pearson Education Italia, Milano 2009[3].
Carl Hamacher – Zvonko Vranesic – Safwat Zaky, *Introduzione all'architettura dei calcolatori*, McGraw–Hill, Milano 2007.
Un ringraziamento speciale ai ragazzi di Wikipedia.

Risorse in rete

[1]: http://os.inf.tu-dresden.de/fiasco
[2]: http://www.gnu.org/software/hurd/index.html
[3]: http://www.fsf.org
[4]: http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

Impressum

Redazione

brunitika, mm-barabba, pmate, xtow, Simone, bel.gio, ferdybassi, fr4nc3sco, NoxDaFox, risca, samiel

Redattori articoli

MadameZou Intervista ad Andrea Mennucci

brunitika Debian GNU/Hurd: il debian-installer

brunitika Bye bye nvidia proprietari

risca LVM mirrored

NoxDaFox Introduzione ai kernel: seconda parte

NoxDaFox Principali tipologie di kernel

Copertina

mm-barabba

Impaginazione

samiel (versione stampa), **brunitika** (web-zine)

Ringraziamenti

Oltre ai membri di redazione, un particolare grazie a MadameZou per l'intervista ad Andrea Mennucci, ad Aki per la revisione dell'articolo sull'installer di Debian GNU/Hurd e a nex_necis per il resoconto sul DUCCIT10 proposto nell'editoriale.

Contatti

Tutti i membri del progetto sono reperibili sul forum del portale http://www.debianizzati.org, dove è possibile trovarli cercando l'utente relativo nel forum.
I sorgenti LaTeX di quesa versione e delle precedenti sono disponibili all'indirizzo:

http://e-zine.debianizzati.org/source/

Happy Debian, Happy Hacking!