systemd, il sistema di init della Fedora 15

systemd è il nuovo sistema di init di Fedora a partire da Fedora 15 (Lovelock), in procinto di essere adottato da un certo numero di distribuzioni maggiori, come OpenSUSE, e Mandriva mentre Debian, Gentoo, ArchLinux si stanno adoperando per l'adozione. Dopo essere stato in technologic preview in Fedora 14 ora è pronto per il debutto ufficiale. Vediamo le caratteristiche principali.

Per capire l' importanza del progetto ricordiamo che il suo ideatore, Lennart Poettering ha richiesto che Gnome 3.2 introduca come dipendenza esterna systemd richiedendo di fatto che tutte le distribuzioni che utilizzano Gnome 3 passino a systemd o che per lo meno lo includano come opzione.
Diamo quindi uno sguardo da più vicino a questa nuova parte del nostro sistema operativo.

Seguendo la definizione ufficiale pubblicata su freedesktop.org, systemd è un gestore di sistema e di servizi per Linux, compatibile con gli script di init di SysV e LSB. Inoltre systemd fornisce:

- Funzionalità aggressive di parallelizzazione,
- Usa socket e l'attivazione D-Bus per l'avviamento dei servizi,
- Offerte lo start su richiesta dei demoni,
- Tiene traccia di processi che utilizzano i cgroups Linux,
- Supporta e assiste nello snapshot e nel ripristino dello stato del sistema,
- Mantiene punti di mount e automount,
- Implementa e elabora una logica di controllo transazionale basata sulle dipendenze tra servizi

Proviamo ora a dare una visione di questi nuovi concetti introdotti da systemd.

Ripensare PID 1

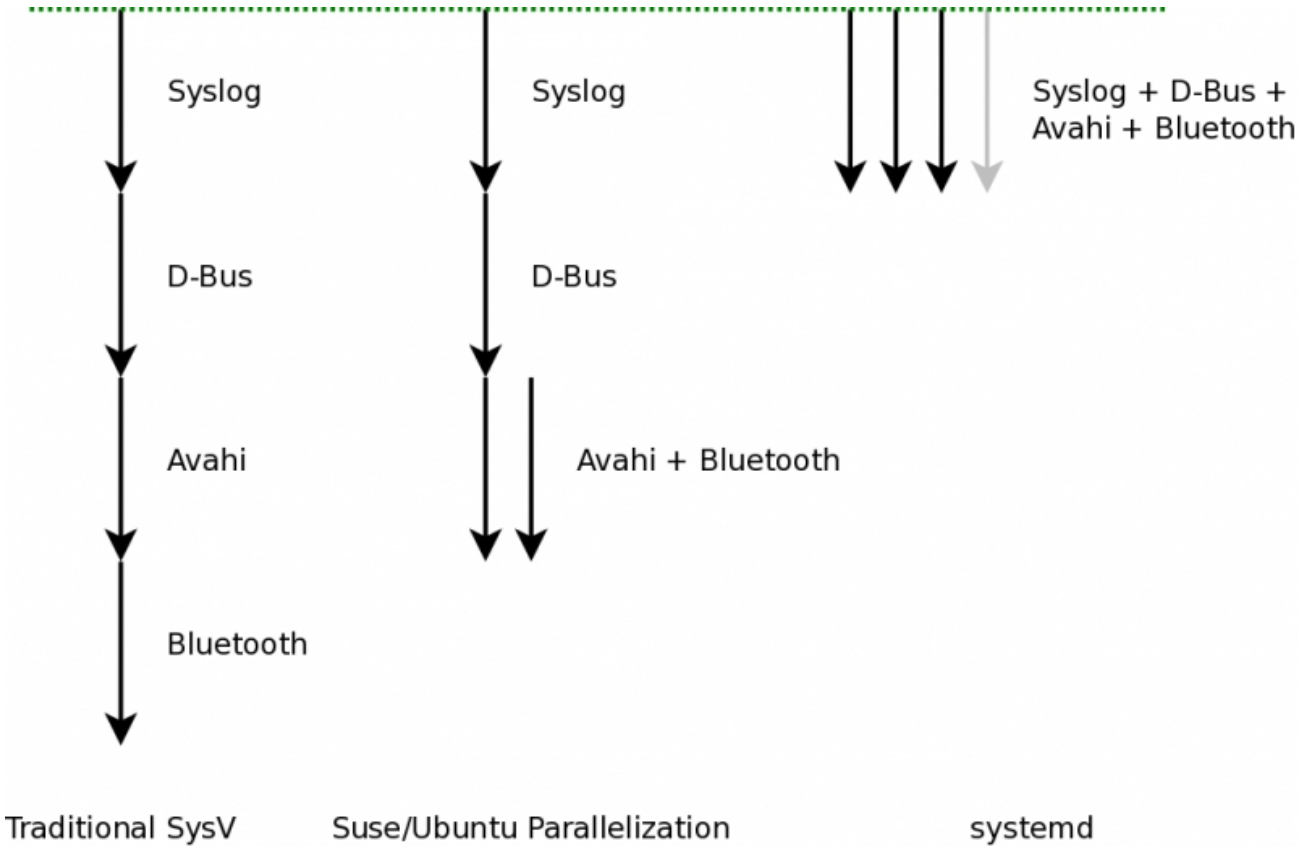
In ogni sistema Unix c'è un processo speciale con un identificativo di processo (detto PID) pari a 1. E 'avviato dal kernel prima di tutti gli altri processi, ed è il padre per tutti quei processi che non hanno più un processo padre di cui essere figlio. E' anche responsabile di alcune cose che ad altri processi non competono, come inizzializzare e mantenere lo userspace durante l'avvio. Molti sono stati i tentativi di sostituzione del Sysvinit, forse quello che più di tutti ci è riuscito è il progetto Upstart che si può trovare in molte distribuzioni attuali.
Tradizionalmente la principale responsabilità di un sistema di init consta nell' inizializzare lo userspace al boot e di farlo nella maniera più veloce possibile. Purtroppo SysV non è ne particolarmente veloce. Ad un efficiente sistema di boot si richiedono due cose fondamentali:

1. Far partire meno servizi possibile
2. Far partire più servizi in parallelo

Per capire meglio l'idea dietro systemd facciamo un esempio:

Ci sono alcuni servizi sotto Linux che sappiamo dovranno essere avviati prima o poi, (D-Bus, syslog, ecc), ma molti altri posso essere avviati solo all'occorrenza. Ad esempio, bluetoothd non ha bisogno di essere in esecuzione a meno che un dongle Bluetooth sia effettivamente collegato o ad esempio una applicazione vuole parlare con l' interfaccia D-Bus del bluetooth. Stessa cosa vale per un sistema di stampa: a meno che la macchina non sia fisicamente collegata a una stampante, o una applicazione richiede di stampare qualcosa, non è necessario eseguire un demone di stampa come CUPS. Nel caso di Avahi ad esempio se la macchina non è collegata a una rete non c'è bisogno di eseguirlo, a meno che qualche applicazione vuole utilizzare le sue API. E anche per SSH, fintanto che nessuno vuole contattare la vostra macchina non vi è alcuna necessità di tenerlo in esecuzione. Tipicamente queste richieste non potevano essere accolte dal sistema di init SysV mentre diventano caratteristiche importanti e nuove in systemd. Ma queste esposte non sono ancora le caratteristiche migliori introdotte da systemd.

Paralelizzazione e Socket Activation



Come è possibile notare nella figura SysV lancia i processi ad uno ad uno attendendo che ognuno di essi termini la sua esecuzione per lanciare il successivo.
Questo permette di rispettare le precedenze tra i processi e di ottenere un boot molto lineare

nel tempo. systemd per accelerare l'esecuzione in fare di boot utilizza una nuova tecnologia chiamata socket activation. Tutti i processi vengono lanciati da systemd contemporaneamente ed è il sistema a gestirne quindi lo stato. Nell' esempio in figura Upstart lancia Avahi e bluethootd in parallelo dato che non vi sono dipendeze tra loro ma entrambi solo dopo che è stato lancianto D-Bus.

Qual'è precisamente il motivo di questa scelta?

Semplicemente analizzando dettagliatamente il codice di Avahi e Bluetooth si scopre che l'unica dipendenza reale da D-Bus è il socket /run/dbus/system_bus_socket (in precedenza sotto /var/run/dbus/system_bus_socket), stessa cosa accade tra D-Bus e Syslog; l'unica dipendenza che D-Bus deve rispettare è che sia presente il socket ovvero /dev/log.

Come è possibile aumentare la velocità di boot in questo caso?

La soluzione che Lennart Poettering ha ideato è quella di portare fuori dai demoni il binding ai vari socket demandando questo lavoro al sistema di init che a questo punto non è solo un sistema di lancio dei processi ma si incarica di inizializzare e attivare i socket principali per il funzionamento del sistema operativo in un unico iniziale step prima del lancio dei vari processi. La creazione di tutti i socket in un unico iniziale punto permette di parallelizzare il successivo lancio di tutti i processi ottenendo il meglio dalle cpu coinvolte e del I/O disponibile senza dover prevedere delle dipendenze reciproche nel lancio dei processi.
Altro vantaggio si ha nel restart di un processo a seguito di un crash. Ad esempio un crash di syslog fino ad ora portava alla perdita dei messaggi che le applicazioni inviavano al socket fintanto che il sistema di init non si accorgeva del crash e rilanciava di nuovo il processo.
Con systemd questo non accade più dato che syslog non deve inizializzare di nuovo il suo socket ed inoltre nel frattempo i messaggi arrivati non andranno persi perchè il socket non è mai stato chiuso. Per syslog è necessario solo agganciare di nuovo il socket per ritrovare i messaggi che non aveva ancora ricevuto. Stessa cosa nel caso di aggiornamento di un demone rendendo il servizio continuamente responsivo anche durante di upgrade. systemd inoltre estende la socket activation non solo ai socket unix ma anche a quelli di rete come TCP/IP e UDP/IP sia IPv4 che IPv6 a named pipe e ai socket udev.

Tutti questi vantaggi hanno un costo infatti portare fuori da un demone l'attivazione di un socket per demandarla a systemd richiede che ogni demone debba essere modificato (creata una patch), anche se minimamente, oppure in maniera più semplice si può sfruttare l’integrazione di systemd con inetd e se il demone interessato supporta inetd allora funzionerà anche con systemd.

systemd tiene traccia di processi che utilizzano i cgroups Linux

In systemd ogni processo che parte viene generato in un gruppo di controllo che prende il nome del suo servizio. I gruppi di controllo (o cgroups) nella loro implementazione più elementare sono semplicemente gruppi di processi che possono essere organizzati in una gerarchia ed etichettati singolarmente. Quando i processi generano altri processi (ovvero eseguono un fork), questi processi figli sono resi automaticamente membri del cgroup genitore. Lasciare un cgroup non è possibile per i

processi non privilegiati. Cosi, cgroups può essere usato come un modo efficace di etichettare i processi dopo il servizio di appartenenza ed essere sicuri che il servizio non può sfuggire all'etichetta, indipendentemente da quanto spesso venga eseguito un fork o quante volte rinomina se stesso. Inoltre questo sistema può essere usato per eseguire il kill in modo sicuro un servizio e tutti i processi che ha creato, ancora una volta senza possibilità di fuga.
Per ottenere una lista dei processi e dei cgroups a cui appartengono è possibile usare il comando:

```
# ps xawf -eo pid,user,cgroup,args
```

Di seguito una immagine che illustra l'output di questo comando.

Nella terza colonna vediamo il cgroup systemd assegnato ad ogni processo. Scopriamo cosi che il processo udev si trova sotto il cgroup di nome /system/udev.service che è dove systemd ripone tutti i processi avviati al boot. Ricordo che è possibile impostare un alias a questo comando per ridurne la digitazione.
systemd fornisce un comando con un output più conciso per ottenere queste informazioni:

```
# systemd-cgls
```

```
├   2 [kthreadd]
└ system
  ├ 1 /sbin/init
  ├ lldpad.service
  │ └ 934 /usr/sbin/lldpad -d
  ├ cups.service
  │ └ 947 cupsd -C /etc/cups/cupsd.conf
  ├ sendmail.service
  │ ├  979 sendmail: accepting connections
  │ └ 1042 sendmail: Queue runner@01:00:00 for /var/spool/clientmqueue
  ├ abrt-oops.service
  │ └ 935 abrt-dump-oops -d /var/spool/abrt -rwx /var/log/messages
  ├ auditd.service
  │ ├ 833 auditd
  │ ├ 835 /sbin/audispd
  │ └ 836 /usr/sbin/sedispatch
  ├ dbus.service
  │ ├  674 /bin/dbus-daemon --system --address=systemd: --nofork --systemd-...
  │ ├  700 /usr/libexec/polkit-1/polkitd
  │ ├  705 /usr/sbin/modem-manager
  │ ├ 1316 /usr/libexec/udisks-daemon
  │ ├ 1317 udisks-daemon: polling /dev/sr0
  │ └ 1419 /usr/libexec/upowerd
  ├ system-setup-keyboard.service
  │ └ 662 /usr/bin/system-setup-keyboard
  ├ crond.service
  │ └ 660 /usr/sbin/crond -n
  ├ irqbalance.service
  ├ avahi-daemon.service
  │ ├ 658 avahi-daemon: running [linux.local]
  │ └ 682 avahi-daemon: chroot helper
  ├ abrtd.service
  │ └ 657 /usr/sbin/abrtd -d -s
  ├ rsyslog.service
  │ └ 667 /sbin/rsyslogd -n -c 5
  ├ dev-mqueue.mount
  ├ dev-hugepages.mount
  ├ udev.service
  │ ├  411 /sbin/udevd
  │ ├  998 /sbin/udevd
  │ └ 1245 /sbin/udevd
  ├ systemd-logger.service
  │ └ 405 /lib/systemd/systemd-logger
  └ media.mount
```