INGENIERÍA DE SERVIDORES (2016-2017)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Comparativa entre systemd e init.

18 de diciembre de 2016

Resumen

En el mundo de *UNIX* donde el kernel está público y cada vez es más grande la comunidad que lo desarrolla, aparecen nuevas contribuciones para intentar mejorar el sistema. En nuestro caso vamos a analizar una nueva implementación del "proceso 1" que rápidamente ha sido adaptada por las principales distribuciones de *UNIX*, `systemd`, sustituyendo al proceso `init`. Explicaremos en que consiste tanto `systemd` como `init` con sus ventajas e inconvenientes, además examinaremos la implementación de ambos para saber un poco más sobre como funcionan. Por último haremos una conclusión final dando una valoración sobre estos procesos aplicados en el mundo de los servidores.

1. Introducción

Como se indica en la historia de The Open Group [8], los sistemas de tipo *UNIX*, concedidos a principios de la década de los 70, se idearon de forma gratuita y libre, fomentando la idea de software libre y de código abierto. Creando así la comunidad tan grande y amplia que conocemos hoy en día. Gracias a esta comunidad se ha seguido desarrollando y avanzando el sistema *UNIX* hasta el kernel de *Linux* que conocemos y utilizamos a diario. Constantemente se están desarrollando nuevas tecnologías y procesos para mejorar este sistema.

Nosotros nos vamos a centrar en comparar una nueva forma de uno de los procesos principales del sistema con su predecesor que curiosamente se ha mantenido casi intacto desde la década de los noventa, es decir, desde hace más de 20 años no ha sido modificado, hecho sorprendente pensando en la velocidad a la que avanza siempre la tecnología. El proceso en cuestión es el encargado del inicio del sistema, es el proceso con PID 1 (Proccess ID). El arranque del sistema funciona con un proceso que va ejecutando todos los demás procesos, como los drivers del adaptador de red o del ratón, los controladores de pantalla, etc.

Este proceso era `init` que era utilizado por todas las distribuciones basadas en Linux. Pero Lennart Poettering, como redacta en su articulo [2], y Kay Sievers decidieron desarrollar un nuevo proceso de arranque del sistema, desarrollando así `systemd`, el encargado de iniciar prácticamente todas las distribuciones de Linux actuales.

Aunque ambos sistemas son daemons (demonios) hay multitud de diferencias entre ellos, que se explicarán mas adelante y finalmente se hará una conclusión de ambos sistemas de arranque para el mundo de los servidores.

2. Proceso init

El proceso `init` se caracteriza por su simpleza y facilidad de uso. El funcionamiento de `init` consiste en ir iniciando los procesos listados en un archivo de configuración, es decir, inicia el primer proceso del listado y cuando éste se ha iniciado inicia el siguiente y así sucesivamente. De esta forma tan simple es como `init` inicia el sistema. Además utiliza la filosofía del software libre de ser transparente para el usuario y permitirle poder modificar por completo su implementación, ya que basta con modificar el fichero de configuración. De esta forma el proceso `init` siempre es el padre o antecesor de todo proceso existente en el sistema y además adopta cualquier proceso que pudiera quedarse sin un padre por el motivo que fuera.

Sin embargo presenta unos claros inconvenientes derivados de su propia naturaleza simple y sencilla. El primer inconveniente es el control de las dependencias, ya que deben controlarse por parte del programador y tiene que tener conocimientos de las dependencias que tienen los procesos y listar primero los procesos que no tienen dependencias o cuyas dependencias ya se han resuelto, por ejemplo si en el listado de procesos aparece un proceso apache que depende de la configuración de

red este proceso se iniciará con errores o no se iniciará puesto que el controlador de red todavía no se ha iniciado, aún cuando tanto apache como el controlador de red estén perfectamente configurados. El otro inconveniente principal, sobre todo en servidores, es la sobrecarga al inicio del sistema, ya que se inician los procesos de uno en uno y hasta que uno no acabe de arrancar no se inicia el siguiente.

2.1. System V

Tal como se dice en el artículo de Jonas Gorauskas en "Linux Jornal" [13], `System V` fue una versión mejorada del `init` original y es la que ha estado presente todo este tiempo en las distintas distribuciones de `Linux`. Aunque esta versión de `init` sigue utilizando su estructura monolítica y por tanto mantiene el inconveniente del control de dependencia, añadía un sistema de llamadas por niveles de prioridad para la ejecución de los procesos del sistema. Este proceso de llamadas se realiza mediante un esquema de directorios con niveles de ejecución que contienen los scripts de arranque de los servicios. Dependiendo de la distribución de `Linux` se utilizan más o menos niveles de prioridad de ejecución.

2.1.1. Problemas de System V

System V presenta una serie de problemas debido a como fue concebido, ya que se pensó para equipos estáticos y que mantenían su hardware, por ello se desarrollo de forma estática y síncrona para el arranque y apagado del sistema, dando lugar a bloqueos de tareas futuras hasta que las actuales no fueran completadas. Esto deja al sistema sin poder reaccionar ante algunos eventos que no estaban programados para el inicio o apagado del sistema. Además no había un control sobre los demonios una vez ejecutados, si estos tenían cualquier problema durante su ejecución no eran tratados por nadie, como mucho si se quedaban huérfanos los recogía el proceso `init`, pero si se paraban no volvía a lanzarlos aunque fueran procesos importantes o vitales para el sistema.

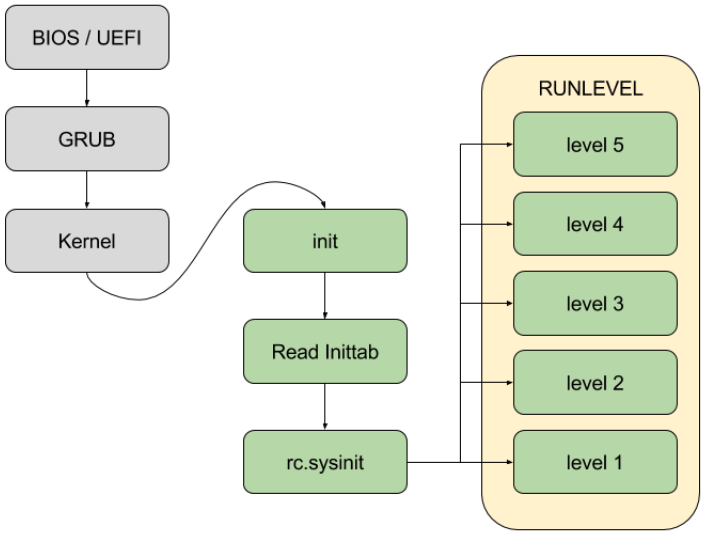

Figura 2.1: System V - boot process

Como se puede observar en la figura 2.1 el proceso de arranque del sistema empieza en las BIOS, continuando por la llamada al GRUB que seleccionando un sistema operativo carga el Kernel. En el caso de sistemas Linux con `System V init` primero leería el fichero de configuración inittab, en el cual se indica el esquema que debe seguir los niveles de ejecución del proceso. En este caso hemos utilizado de ejemplo un sistema con 5 niveles.

3. Proceso systemd

Como ya hemos dicho anteriormente `systemd` fue desarrollado por Lennart Poettering y Kay Sievers, ambos empleados de Red Hat. El proceso `systemd` tal como lo definen los autores en la web oficial de `systemd` [6] es una suite o conjunto de herramientas diseñadas para ofrecer una funcionalidad específica, en este caso para facilitar y mejorar el arranque del sistema operativo *Linux*. Específicamente es un conjunto de demonios de *Linux*, un demonio [4] es un proceso que funciona en segundo plano en el sistema a la espera de eventos o llamadas que se producen en el sistema y cuando son despertados realizan una tarea concreta. Systemd no solo ha sustituido al proceso init, sino a toda la gestión que era necesaria para el correcto funcionamiento del inicio del sistema.

Lennart y Kay diseñaron `systemd` buscando una mayor eficiencia en el inicio de los sistemas `Linux`, para ello crearon un `framework`, un `framework` en software es una infraestructura estandarizada en resolver un problema específico donde poder expresar todas las dependencias y así poder realizar una mejor gestión de las dependencias. Para conseguir una mayor eficiencia se centraron en el paralelismo frente la ejecución de procesos de forma secuencial que utilizaba su predecesor. Ahora `systemd` realiza todas las llamadas para el inicio del sistema por lo que se ha eliminado la utilización de una terminal como intermediario, ahorrando tiempo en el inicio del sistema. El `framework` de dependencias es mucho más permisivo ya que es capaz de controlarlas a un nivel más bajo consiguiendo de esta forma ser mucho más agresivo en la paralelización del inicio del sistema. Además permite al administrador del sistema un mayor control sobre el orden en que se inician los servicios.

En el artículo de ZDNet [12] podemos ver un esquema de los componentes de systemd como se muestra en la figura 3.1. Se pueden observar los cinco grande módulos en los que se divide la suite `systemd`: Libraries, Core, Targets, Daemons y Utilities.

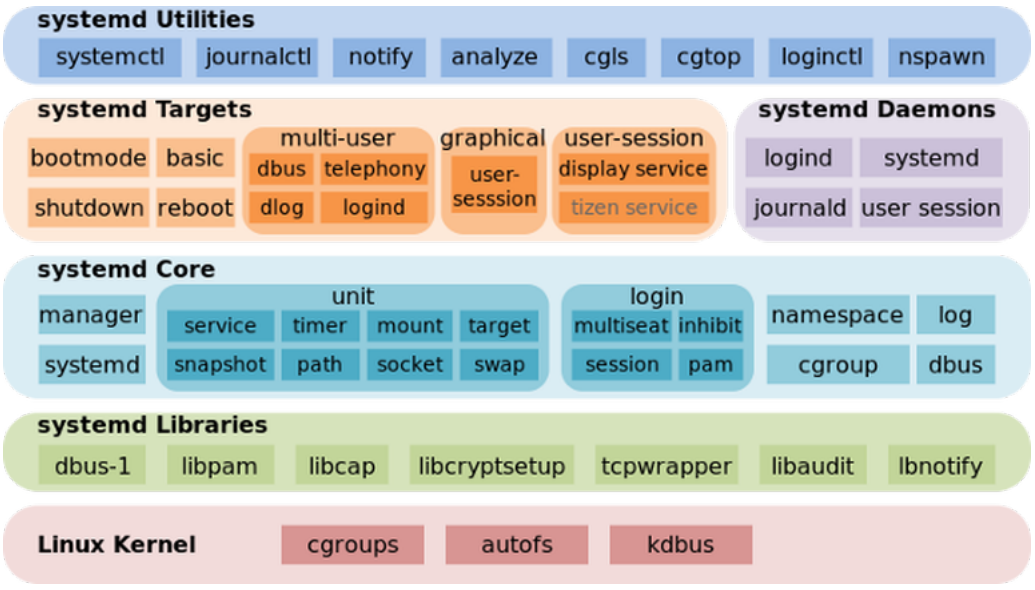

Figura 3.1: Componentes de systemd

Para la resolución de dependencias se fijaron en su problema más fundamental, la utilización de servicios y comunicaciones con otros procesos. Para las comunicaciones se sirven normalmente de los sockets, un socket no es más que una estructura de datos del sistema de archivos para el intercambio de información entre procesos de un sistema Linux, como indica el manual (man) de Linux [3]. De esta forma se pueden iniciar todos los demás procesos de forma paralela teniendo en cuenta menos dependencias puesto que al estar los sockets ya creados pueden empezar la comunicación aunque alguno de los dos procesos todavía no este iniciado. Por ejemplo los procesos que necesitan escribir en los "logs" del sistema dependen del socket "/dev/log", entonces systemd crea este socket de los primeros, eliminando así múltiples dependencias y bloqueos a otros servicios por consecuencia de las dependencias. De esta forma se consigue que aunque el proceso no se haya iniciado, el socket donde tiene que leer ya esté cargado con la información necesaria y así cuando el proceso se inicie cargarla directamente.

Las dependencias que se mantienen después de las mejoras previamente descritas se resuelven con un sistema de "target", si en System V teníamos un esquema de jerarquía con niveles de ejecución en systemd tenemos algo parecido que sirve al mismo propósito. Como se dice en ArchLinux Wiki [5]: "*Cada target se nomina, en lugar de numerarse, y está destinado a servir a un propósito específico con la posibilidad de realizar más de una acción al mismo tiempo. Algunos targets son activados heredando todos los servicios de otro target e implementando servicios adicionales. Como hay targets de systemd que imitan los runlevels de SystemV init, es, por tanto, posible pasar de un target a otro...*".

La suite de systemd como comentamos anteriormente trae una serie de herramientas orientadas a mejorar el sistema y las opciones que puede realizar el usuario final, tal como se indican en la wiki de ArchLinux [5] y en la documentación de "SUSE" [7] incluye los demonios para el control de sesiones (logind), redes (networkd) y los mensajes del sistema (journald). También cuenta con utilidades como un analizador de estado del sistema, que nos muestra de forma cómoda las unidades que han tenido algún tipo de problema o simplemente las unidades instaladas en el sistema. Otra utilidad es la gestión de energía instalada en el paquete de logind que, por ejemplo, permite un apagado o reinicio del sistema.

Vamos a comentar los principales demonios de systemd:

3.1. Demonio journald

Para conocer un poco más en profundidad sobre este demonio vamos a consultar un articulo en el blog de Lennart [1] en el cual se explica lo siguiente. Journald es un demonio incluido en la suite de systemd para la gestión de los mensajes del sistema que cuenta con un nuevo esquema provisto de una estructura de índices. Con esta nueva versión de demonio para los logs se mantiene la compatibilidad con los demonios anteriores y además es más eficiente que su antecesor ya que utiliza la salida estándar junto con las llamadas al sistema de forma nativa en el propio demonio.

3.2. Demonio logind

Este demonio, tal y como se dice en la documentación oficial de systemd [10], es el encargado de administrar los inicios de sesión de los usuarios del sistema. Por lo tanto realiza un control de los usuarios y sus sesiones así como de sus procesos. La configuración de este servicio se guarda en logind.conf.

3.3. Demonio user session

Este demonio es el más sencillo, su función es permitir o bloquear los inicios de sesión de usuario dependiendo del estado del sistema [11].

3.4. Implementación

Podemos encontrar la implementación completa de systemd en su GitHub [9]. Por ejemplo en el archivo `src/systemd/sd-daemon.h` podemos ver la definición de los niveles de "logs" que utiliza el proceso `systemd`.

```
/*
  Log levels for usage on stderr:
        fprintf(stderr, SD_NOTICE "Hello World!\n");
  This is similar to printk() usage in the kernel.
*/
#define SD_EMERG   "<0>"  /* system is unusable */
#define SD_ALERT   "<1>"  /* action must be taken immediately */
#define SD_CRIT    "<2>"  /* critical conditions */
#define SD_ERR     "<3>"  /* error conditions */
#define SD_WARNING "<4>"  /* warning conditions */
#define SD_NOTICE  "<5>"  /* normal but significant condition */
#define SD_INFO    "<6>"  /* informational */
#define SD_DEBUG   "<7>"  /* debug-level messages */
```

Se puede observar que se definen ocho niveles de "logs", de más prioritario a menos, siendo 0 el más prioritario.

4. Ventajas e inconvenientes de systemd frente init (system V)

4.1. Ventajas

4.2. Inconvenientes

5. Conclusiones

Referencias

[1] http://0pointer.de/blog/projects/systemctl-journal.html, consultado el 14 de Diciembre de 2016. Blog de Lennart.

[2] http://0pointer.de/blog/projects/systemd.html#faqs, consultado el 14 de Diciembre de 2016. Blog de Lennart.

[3] http://man7.org/linux/man-pages/man7/unix.7.html, consultado el 14 de Diciembre de 2016. Man de Linux.

[4] https://wiki.archlinux.org/index.php/daemons, consultado el 14 de Diciembre de 2016. ArchWiki, apartado sobre daemons.

[5] https://wiki.archlinux.org/index.php/systemd_(Espa%C3%B1ol), consultado el 14 de Diciembre de 2016. Archlinux Wiki.

[6] https://www.freedesktop.org/wiki/Software/systemd/, consultado el 14 de Diciembre de 2016. Web oficial de systemd.

[7] https://www.suse.com/docrep/documents/huz0a6bf9a/systemd_in_suse_linux_enterprise_12_white_paper.pdf, consultado el 14 de Diciembre de 2016. Documentaciión de SUSE enterprise.

[8] http://www.unix.org/what_is_unix/history_timeline.html, consultado el 14 de Diciembre de 2016. The Open Group, UNIX.

[9] https://github.com/systemd/systemd, consultado el 18 de Diciembre de 2016. Implementación de systemd.

[10] https://www.freedesktop.org/software/systemd/man/systemd-logind.html, consultado el 18 de Diciembre de 2016. man de systemd logind.

[11] https://www.freedesktop.org/software/systemd/man/systemd-user-sessions.service.html, consultado el 18 de Diciembre de 2016. man de systemd user-sessions.

[12] http://www.zdnet.com/article/after-linux-civil-war-ubuntu-to-adopt-systemd/, consultado el 18 de Diciembre de 2016. Artículo sobre Systemd de zdnet.

[13] Jonas Gorauskas. http://www.linuxjournal.com/content/initializing-and-managing-services-linux-past-present-and-future?page=0,0, consultado el 14 de Diciembre de 2016. Articulo de “Linux Journal”.