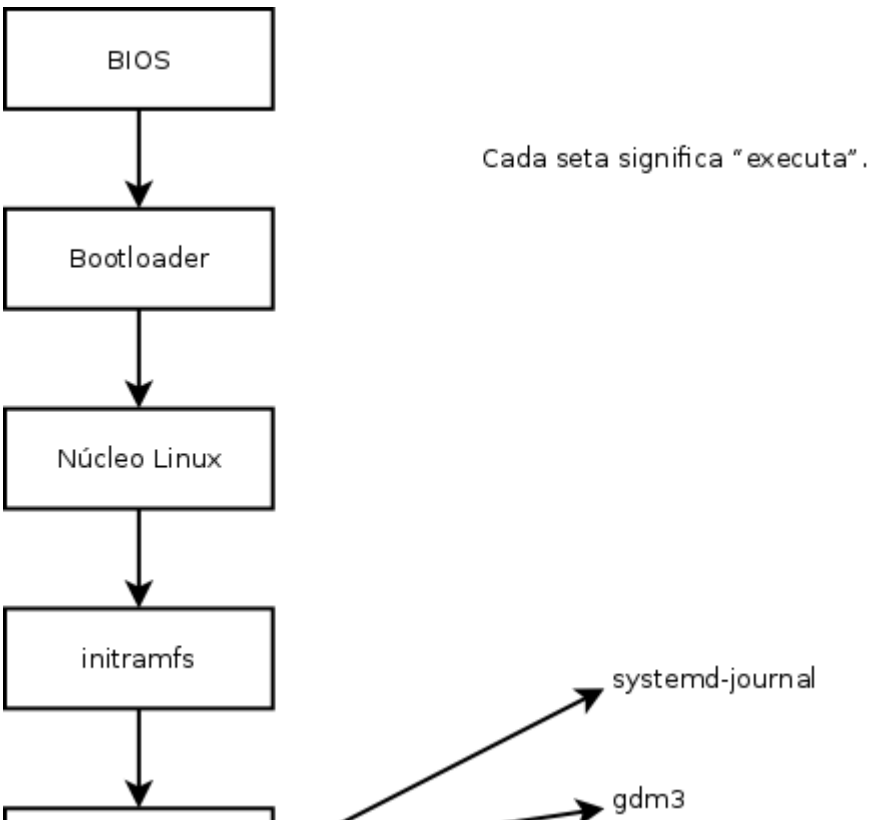
Capítulo 9. Serviços Unix

Este capítulo abrange uma série de serviços básicos que são comuns a muitos sistemas Unix. Todos os administradores devem estar familiarizados com eles.
From: <https://debian-handbook.info/browse/pt-BR/stable/unix-services.html#sect.system-boot>

9.1. Inicialização do Sistema

Quando você inicializar o computador, algumas mensagens rolarão pelo console automaticamente inicializando e as configurações são automaticamente executadas. Algumas vezes você pode desejar alterar como este estágio funciona, de forma que possa entender isto muito bem. Este é o propósito desta seção.

Primeiro, a BIOS pega o controle sobre o computador, detectando discos, carregando a *Master Boot Record*, e executa o carregador de inicialização. O carregador de inicialização assume, localiza o kernel no disco, carrega e o executa. O kernel é então inicializado e começa a pesquisa pela partição e monta a partição contendo o sistema raiz e finalmente o primeiro programa — **`init`**. Frequentemente, esta "partição raiz" e este **`init`** são, de fato, localizado em um sistema de arquivos virtual que só existe na RAM (daí o seu nome, "initramfs", anteriormente chamado de "initrd" para "initialization RAM disk"). Este sistema de arquivos é carregado na memoria pelo carregador de inicialização, muitas vezes a partir de um arquivo em um disco rígido ou da rede. Ele contém o mínimo exigido pelo kernel para carregar o sistema de arquivos raiz "verdadeiro". Este pode ser módulos de driver para o disco rígido ou outros dispositivos sem o qual o sistema pode não inicializar, ou, mais freqüentemente, scripts de inicialização e módulos para a montagem de arrays RAID, abrindo partições criptografadas, ativando volumes LVM, etc. Uma vez que a partição raiz é montada, o initramfs libera o controle para o init real, e a máquina voltará para o processo de inicialização padrão.



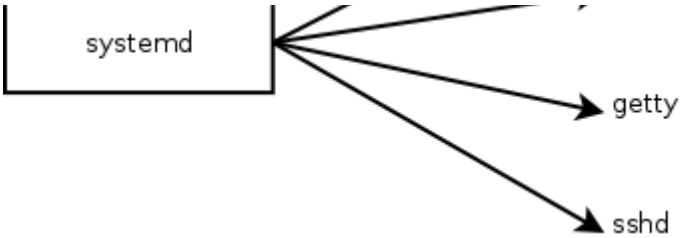


Figura 9.1. Seqüência de inicialização de um computador rodando Linux com systemd

9.1.1. O sistema init systemd

O “init real” é atualmente fornecido pelo systemd e essa seção documenta esse sistema init.

CULTURA Antes do `systemd`

O **`systemd`** é um "sistema init" relativamente recente, e embora ele já estivesse disponível, até certo ponto, no Wheezy, ele só se tornou o padrão no Debian Jessie. Lançamentos anteriores faziam uso, por padrão, do “System V init” (no pacote sysv-rc), um sistema muito mais tradicional. Nós descreveremos o "System V init" mais tarde.

ALTERNATIVA Outros sistemas de inicializações

Este livro descreve o sistema de inicialização usado por padrão no Debian Jessie (como implementado pelo pacote systemd), assim como o previamente padrão sysvinit, que é derivado e herdado dos sistemas Unix *System V*, mas existem outros.

file-rc é um sistema de inicialização com um processo muito simples. Ele mantém a ideia de runlevels (níveis de execução), mas substitui os diretórios e links simbólicos com um arquivo de configuração, que diz ao **`init`** os processos que devem ser iniciados e sua ordem de inicialização.

O recém-chegado sistema **`upstart`** ainda não está perfeitamente testado no Debian. Ele é baseado em eventos: scripts de inicialização não são mais executados em uma ordem sequencial, mas em resposta a eventos como a conclusão de um outro script do qual eles são dependentes. Este sistema, iniciado pelo Ubuntu, está presente no Debian Jessie, mas não é o padrão; ele vem, de fato, como um substituto para o sysvinit, e uma das funções iniciadas pelo **`upstart`** é iniciar os scripts escritos para sistemas tradicionais, especialmente aqueles do pacote sysv-rc.

Existem também outros sistemas e outros modos operacionais, tais como **`runit`** ou **`minit`**, mas eles são relativamente especializados e não generalizados.

SPECIFIC CASE Inicializando pela rede

Em algumas configurações, o BIOS pode ser configurado para não executar o MBR, mas buscar o seu equivalente na rede, tornando possível a construção de computadores sem disco rígido, ou que são completamente reinstalado a cada boot. Esta opção não está disponível em todos os hardwares e geralmente requer uma combinação adequada de BIOS e placa de rede.

A inicialização através da rede pode ser usada para iniciar o **`debian-installer`** ou FAI (ver Seção 4.1, “Métodos de Instalação”).

BACK TO BASICS Um processo, uma instância do programa

Um processo é a representação em memória de um programa em execução. Isto inclui todas as informações necessárias para a execução adequada do software (o código própriamente dito, mas também os dados que tem na

memória, a lista de arquivos que ele abriu, as conexões de rede que estabeleceu, etc.). Um único programa pode ser instanciado em muitos processos, não necessáriamente rodando sob diferentes IDs de usuários.

SECURITY Usando Shell como `init` para ganhar privilégios de root

Por convenção, o primeiro processo que é carregado é o programa **`init`** (que é uma ligação simbólica para **`/lib/systemd/systemd`**, por padrão). Contudo, é possível passar uma opção **`init`** para o kernel indicando um programa diferente.

Qualquer pessoa que é capaz de acessar o computador e poder pressionar o botão **`Reset`**, e, portanto reinicia-lo . Então, no prompt do inicializador do sistema, é possível passar opção **`init=/bin/sh`** para o kernel e ganhar acesso root sem no saber a senha de administrador.

Para evitar isso, você pode proteger o bootloader com uma senha. Você também pode pensar em proteger o acesso ao BIOS (um mecanismo de proteção por senha geralmente é disponível), sem que um intruso mal-intencionado ainda possa iniciar a máquina por uma mídia removível que contém o seu próprio sistema Linux, que pode então usar para acessar dados sobre discos rígidos do computador.

Finalmente, esteja ciente que a maioria das BIOS tem uma senha genérica disponível. Inicialmente destinado a solução de problemas para aqueles que esqueceram sua senha, essas senhas são agora público e disponível na Internet (veja por si mesmo procurando por "senhas genéricas" do BIOS em um motor de busca). Todas estas proteções deverá impedir o acesso não autorizado para a máquina sem ser capaz de impedir completamente. Não há nenhuma maneira confiável para proteger um computador se o atacante pode acessar fisicamente ela, pois eles podem desmontar os discos rígidos para conectá-los a um computador sob seu próprio controle de qualquer maneira, ou até mesmo roubar a máquina inteira, ou apagar a memória do BIOS para redefinir a senha…

O systemd executa vários processos, se encarregando de configurar o sistema: teclados, drivers, sistemas de arquivos, rede, serviços. Ele faz isso enquanto mantem uma visão global do sistema como um todo, e os requerimentos dos componentes. Cada componente é descrito por um "arquivo unit" (às vezes mais); a sintaxe geral é derivada do amplamente usado "arquivos *.ini", com os pares ***`chave = valor`*** agrupados entre cabeçalhos **`[`*`seção ("section")`*`]`**. Arquivos unit são armazenados em **`/lib/systemd/system/`** e **`/etc/systemd/system/`**; eles vem em vários sabores, mas nós iremos focar nos "services" e "targets" aqui.

Um "arquivo service" do systemd descreve um processo gerenciado pelo systemd. Ele contém, grosseiramente, a mesma informação dos antigos scripts init, mas com expressão de maneira declaratória (e muito mais concisa). O systemd maneja a massa de tarefas repetitivas (iniciar e para processo, checar seu status, "logging", descarte de privilégios e muito mais), e o arquivo service apenas precisa preencher as especificações do processo. Por exemplo, aqui está um arquivo service para o SSH:

```
[Unit]
Description=OpenBSD Secure Shell server
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

```
Alias=sshd.service
```

Como você pode ver, existe muito pouco código nele, apenas declarações. O systemd cuida da exibição dos relatórios de progresso, mantendo os rastros dos processos, e até mesmo reiniciando-os quando necessário.

O “arquivo target” do systemd descreve o estado do sistema, aonde um conjunto de serviços são conhecidos como estando operacionais. Ele pode ser pensado como um equivalente ao runlevel no estilo antigo. Um dos alvos ("targets") é **`local-fs.target`**; quando ele é alcançado, o resto do sistema pode assumir que todos os sistemas de arquivos locais estão montados e acessíveis. Outros alvos ("targets") incluem **`network-online.target`** e **`sound.target`**. As dependências de um alvo ("target") podem ser listadas tanto dentro de um arquivo target (na linha **`Requires=`**), quanto usando uma ligação simbólica para um arquivo service do diretório **`/lib/systemd/system/`*`targetname`*`.target.wants/`**. Por exemplo, **`/etc/systemd/system/printer.target.wants/`** contém uma ligação para **`/lib/systemd/system/cups.service`**; o systemd irá então garantir que o CUPS está rodando a fim de alcançar o **`printer.target`**.

Como arquivos unit são declarativos ao invéz de scripts ou programas, eles não podem ser rodados diretamente, e eles só são interpretados pelo systemd; vários utilitários, entretanto, permitem que o administrador interaja com o systemd e controle o estado do sistema e de cada componente.

O primeiro de tais utilitários é o **`systemctl`**. Quando rodado sem argumentos, ele lista todos os arquivos unit conhecidos pelo systemd (exceto aqueles que tenham sido desabilitados), assim como seus status. O **`systemctl status`** retorna uma visão melhor dos serviços, assim como os processos relacionados. Se o nome do serviço for informado (como em **`systemctl status ntp.service`**), ele retorna ainda mais detalhes, assim como as últimas linhas de registro ("log") relacionadas ao serviço (mais sobre isso mais tarde).

Iniciar um serviço a mão é uma simples questão de rodar **`systemctl start `*`nomedoserviço`*`.service`**. Como se pode imaginar, para o serviço é feito com **`systemctl stop `*`nomedoserviço`*`.service`**; outros subcomandos incluem **`reload`** e **`restart`**.

Para controlar se um serviço está ativo (ou seja, se ele será iniciado automaticamente na inicialização), use **`systemctl enable `*`nomedoserviço`*`.service`** (ou **`disable`**). **`is-enabled`** permite checar o status do serviço.

Um recurso interessante do systemd é que ele inclui um componente de "logging" de nome **`journald`**. Ele vem como um complemento para sistemas de "logging" mais tradicionais, tal como o **`syslogd`**, mas ele adiciona recursos interessantes tal como uma ligação formal entre um serviço e as mensagens que ele gera, e a habilidade de capturar mensagens de erro geradas pela sua sequência de inicialização. As mensagens podem ser exibidas mais tarde, com um pequena ajuda do comando **`journalctl`**. Sem qualquer argumento, ele simplismente derrama todas as mensagens de "log" que ocorreram desde a inicialização do sistema; ele raramente será usado de tal maneira. Na maior parte do tempo, ele será usado com um identificador de serviço:

```
# journalctl -u ssh.service
-- Logs begin at Tue 2015-03-31 10:08:49 CEST, end at Tue 2015-03-31 17:06:02 CEST. --
Mar 31 10:08:55 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.
Mar 31 10:08:55 mirtuel sshd[430]: Server listening on :: port 22.
Mar 31 10:09:00 mirtuel sshd[430]: Received SIGHUP; restarting.
Mar 31 10:09:00 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.
```

```
Mar 31 10:09:00 mirtuel sshd[430]: Server listening on :: port 22.
Mar 31 10:09:32 mirtuel sshd[1151]: Accepted password for roland from 192.168.1.129
port 53394 ssh2
Mar 31 10:09:32 mirtuel sshd[1151]: pam_unix(sshd:session): session opened for user
roland by (uid=0)
```

Outra opção de linha de comando útil é a **`-f`**, que instrui o **`journalctl`** a manter a exibição de novas mensagens assim que elas são emitidas (similar ao **`tail -f` *`arquivo`***).

Se um serviço não parece estar trabalhando como o esperado, o primeiro passo para resolver o problema é checar se o serviço está realmente rodando com **`systemctl status`**; se ele não está, e as mensagens obtidas pelo primeiro comando não são suficientes para diagnosticar o problema, confira os registros ("logs") coletados pelo journald sobre esse serviço. Por exemplo, suponha que o servidor SSH não esteja funcionando:

```
# systemctl status ssh.service
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
   Active: failed (Result: start-limit) since Tue 2015-03-31 17:30:36 CEST; 1s ago
  Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
  Process: 1188 ExecStart=/usr/sbin/sshd -D $SSHD_OPTS (code=exited, status=255)
 Main PID: 1188 (code=exited, status=255)

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
Mar 31 17:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly,
refusing to start.
Mar 31 17:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.
Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
# journalctl -u ssh.service
-- Logs begin at Tue 2015-03-31 17:29:27 CEST, end at Tue 2015-03-31 17:30:36 CEST. --
Mar 31 17:29:27 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.
Mar 31 17:29:27 mirtuel sshd[424]: Server listening on :: port 22.
Mar 31 17:29:29 mirtuel sshd[424]: Received SIGHUP; restarting.
Mar 31 17:29:29 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.
Mar 31 17:29:29 mirtuel sshd[424]: Server listening on :: port 22.
Mar 31 17:30:10 mirtuel sshd[1147]: Accepted password for roland from 192.168.1.129
port 38742 ssh2
Mar 31 17:30:10 mirtuel sshd[1147]: pam_unix(sshd:session): session opened for user
roland by (uid=0)
Mar 31 17:30:35 mirtuel sshd[1180]: /etc/ssh/sshd_config line 28: unsupported option
"yess".
Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
Mar 31 17:30:35 mirtuel sshd[1182]: /etc/ssh/sshd_config line 28: unsupported option
"yess".
Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
Mar 31 17:30:35 mirtuel sshd[1184]: /etc/ssh/sshd_config line 28: unsupported option
"yess".
Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
```

```
Mar 31 17:30:36 mirtuel sshd[1186]: /etc/ssh/sshd_config line 28: unsupported option
"yess".
Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
Mar 31 17:30:36 mirtuel sshd[1188]: /etc/ssh/sshd_config line 28: unsupported option
"yess".
Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
status=255/n/a
Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
Mar 31 17:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly,
refusing to start.
Mar 31 17:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.
Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
# vi /etc/ssh/sshd_config
# systemctl start ssh.service
# systemctl status ssh.service
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
   Active: active (running) since Tue 2015-03-31 17:31:09 CEST; 2s ago
  Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
 Main PID: 1222 (sshd)
   CGroup: /system.slice/ssh.service
           └─1222 /usr/sbin/sshd -D
#
```

Após checar o status do serviço (failed), nós fomos checar os registros ("logs"); eles indicam um erro no arquivo de configuração. Após editar o arquivo de configuração e consertar o erro, nós reiniciamos o serviço, e então verificamos se ele está realmente rodando.

INDO ALÉM Outros tipos de arquivos unit

Nesta seção, nós apenas descrevemos as mais básicas capacidades do systemd. Ele oferece muitos outros recursos interessantes; nós iremos listar apenas alguns aqui:

- ativação de "socket": um arquivo unit “socket” pode ser usado para descrever um "socket" de rede ou Unix gerenciado pelo systemd; isso significa que o "socket" será criado pelo systemd, e o real serviço pode ser iniciado por requisição ("on demand") quando uma real tentativa de conexão vier. Isso, grosseiramente, replica o recurso configurado pelo **`inetd`**. Veja systemd.socket(5).
- timers: um arquivo unit “timer” descreve eventos que ocorrem com uma frequência fixa ou em horários específicos; quando um serviço é ligado a tal timer, a tarefa correspondente será executada sempre que o timer for acionado. Isso permite replicar parte dos recursos do **`cron`**. Veja systemd.timer(5).
- network: um arquivo unit "network" descreve uma interface de rede, que permite a configuração de tais interfaces assim como expressar que um serviço dependa de uma interface em particular levantada.

9.1.2. O sistema init System V

O sistema init System V (que nós iremos chamar init para abreviar) executa vários processos, seguindo instruções a partir do arquivo **`/etc/inittab`**. O primeiro programa que é executado (o que corresponde ao passo *sysinit*) é **`/etc/init.d/rcS`** , um script que executa todos os programas que eestão dentro do diretório **`/etc/rcS.d /`**.

Entre estes,você encontrará sucessivamente programas responsáveis pela:

- configurar o teclado do console;
- carregando drivers: a maioria dos módulos do kernel serão carregados por si assim que o hardware seja detectado; drivers extra então são carregado automaticamente quando o modulo correspondente seja listado em **/etc/modules**;
- checar a integridade do sistema de arquivos;
- montar partições locais;
- configuração da rede;
- mountando sistemas de arquivos em rede (NFS).

DE VOLTA AO BÁSICO Kernel modules and options

Os módulos do kernel também tem opções que podem ser configuradas colocando alguns arquivos em **/etc/modprobe.d/** . Essas opções são definidas com as diretivas como esta: **opções*nome do módulo nome da opção=valor da opção*** . Várias opções podem ser especificadas com uma única diretiva, se necessário.

Estes arquivos de configuração são destinados para o **modprobe** - o programa que carrega um módulo do kernel com suas dependências (módulos podem realmente chamar outros módulos). Este programa é fornecido pelo pacote kmod.

Após este estágio, o **init** assume o controle e inicializa os programas habilitados no nível de execução padrão (que geralmente é no nível de execução 2). Ele executa o **/etc/init.d/rc 2**, um script que inicia todos os serviços que estão listados em **/etc/rc2.d/** e que os nomes começam com a letra "S". O número de duas casas que se segue tinha sido historicamente utilizado para definir a ordem em que os serviços devem de ser iniciados. Atualmente, o sistema de inicialização padrão usa **insserv**, o qual agenda automaticamente tudo, baseado nas dependências dos scripts. Desta forma, cada script de inicialização declara as condições que devem ser cumpridas para iniciar ou parar um serviço (por exemplo, se ele deve começar antes ou depois de outro serviço); o **init** em seguida, lança-os na ordem que satisfaça estas condições. A numeração estática dos scripts, portanto, não é mais levada em consideração (mas eles sempre devem ter um nome começando por "S" seguido por dois digitos e o nome atual do script usado por suas dependências). Geralmente, serviços base (tal como registros com o **rsyslog**, ou numeração de portas com **portmap**) são inicializados primeiro, seguidos por serviços padrões e a interface gráfica (**gdm3**).

Este sistema de inicialização baseado em dependência torna possível automatizar a numeração, que poderia ser um pouco entediante se tivesse que ser feito manualmente, e limita os riscos de erro humano, já que o agendamento é realizado de acordo com os parâmetros indicados. Outro benefício é que os serviços podem ser iniciados em paralelo quando são independentes um do outro, que pode acelerar o processo de inicialização.

init distingue vários runlevels, então para que ele possa alternar de um para outro com o comando **telinit*new-level***. Imediatamente,**init** executa **/etc/init.d/rc** novamente com novo runlevel. Este script irá, em seguida, iniciar os serviços ausentes e interromper aqueles que não são mais desejado. Para fazer isso, ele se dirige ao conteúdo do **/etc/rc*X*.d**(onde *X* representa o novo runlevel).Scripts começando com "S" (como em "Start") são serviços iniciados; aqueles que iniciam com "K" (como em "Kill") são os serviços interrompidos. O script não inicia qualquer serviço que já estava ativo em runlevel anterior.

Por padrão, o init System V no Debian usa quatro runlevels diferentes:

- Nível 0 é usada apenas temporariamente, enquanto o computador está desligando. Como tal, ele só contém muitos scripts de "K".
- Nível 1, também conhecido como modo de usuário único, corresponde ao sistema em modo degradado; inclui apenas os serviços básicos e destina-se para operações de manutenção onde interações com usuários comuns não são desejadas.
- Nível 2 é o funcionamento normal, o que inclui serviços de rede, uma interface gráfica, logons de usuário, etc.
- Nível 6 é semelhante ao nível 0, exceto que é utilizada durante a fase de desligamento que precede uma reinicialização.

Existem outros níveis, especialmente de 3 a 5. Por padrão, eles são configurados para operar da mesma maneira como nível 2, mas o administrador pode modificá-los (adicionando ou excluindo os scripts nos diretórios correspondentes **`/etc/rcX.d`**) para adaptá-los às necessidades específicas.

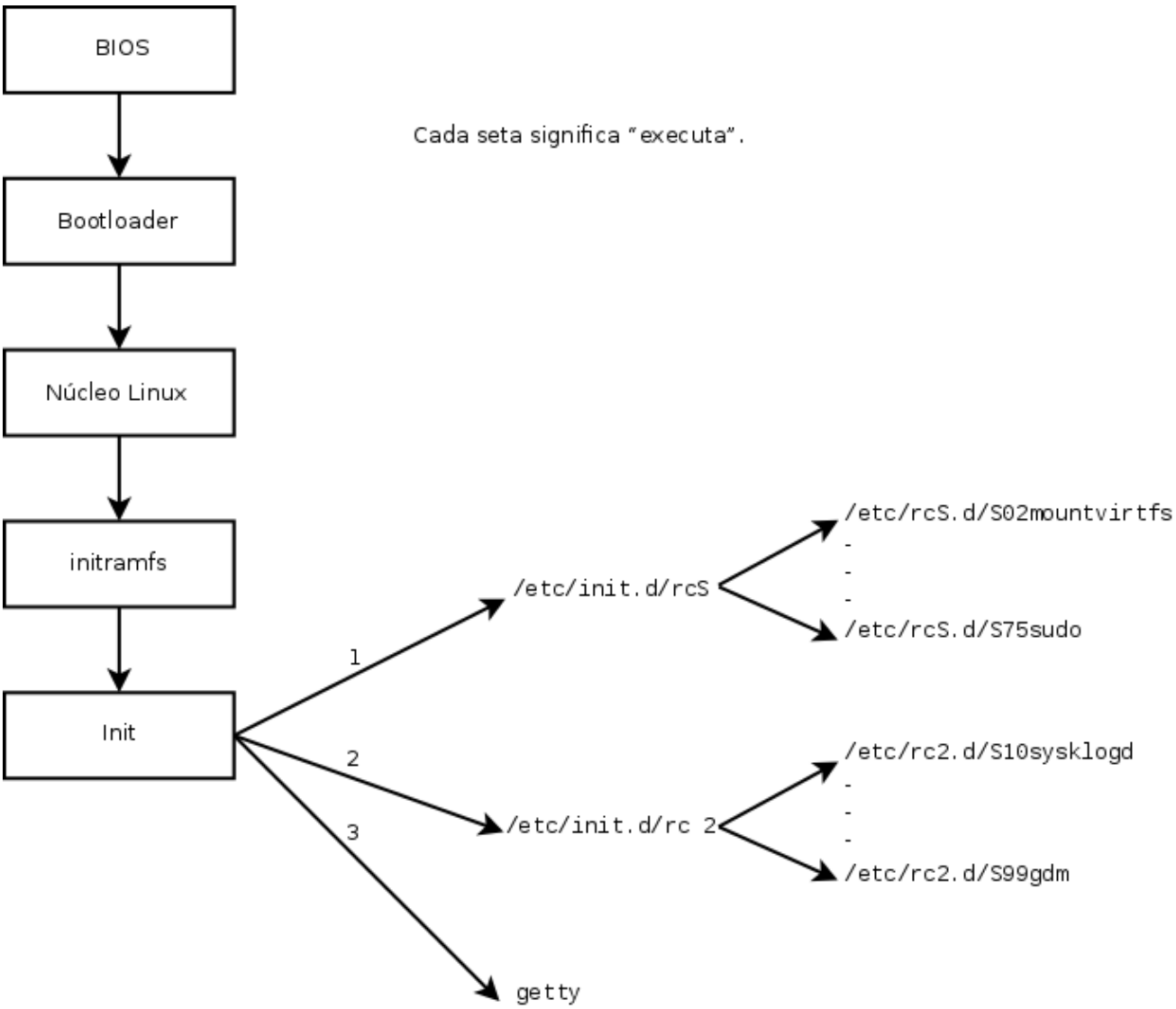


Figura 9.2. Seqüência de inicialização de um computador rodando Linux com o init System V

Todos os scripts contidos nos vários diretórios **`/etc/rcX.d`** são na verdade apenas links simbólicos — criados durante a instalação de pacotes pelo programa **`update-rc.d`** — apontando

para os scripts atuais que são armazenados no **/etc/init.d/**. O administrador pode ajustar os serviços disponíveis em cada nível de execução reexecutando o **update-rc.d** com parâmetros de ajuste. A página de manual do update-rc.d(1) descreve a sintaxe em detalhes. Note que remover todos os links simbólicos (com o parâmetro **remove**) não é um bom método para desabilitar um serviço. Ao invés disto você deve apenas configurar ele para não iniciar no nível de execução desejado (enquanto preserva as chamadas correspondentes para parar ele no caso do serviço iniciar num nível de execução anterior). Uma vez que o **update-rc.d** tem uma interface de certa forma "convoluted", você pode preferir usar **rcconf** (do pacote rcconf) que fornece uma interface de usuários mais amigável.

POLÍTICA DEBIAN Reiniciando serviços

Os scripts de manutenção para os pacotes Debian algumas vezes irão reiniciar alguns serviços para garantir a sua disponibilidade ou levá-los a tomar certas opções em conta. O comando que controla um serviço -- **service *serviço operação*** - não leva em consideração o nível de execução ("runlevel"), assume (erroneamente) que o serviço está sendo usado, e pode, assim, iniciar operações incorretas (começando um serviço que estava deliberadamente interrompido ou interromper um serviço que já esta parado, etc.) Portanto o Debian introduziu o programa **invoke-rc.d**: este programa deve ser usado por scripts de manutenção para executar serviços de scripts de inicialização e isso só irão executar os comandos necessários. Observe que, ao contrário do uso comum, o sufixo **.d** é usado aqui em um nome de programa, e não em um diretório.

Finalmente, **init** começa a controla programas para vários consoles virtuais (**getty**). Ele exibe um prompt, esperando por um nome de usuário, em seguida, executa o usuário **login *user*** para iniciar uma sessão.

VOCABULÁRIO Console e terminal

Os primeiros computadores eram geralmente separados em diversas, peças muito grandes: o compartimento de armazenamento e unidade central de processamento foram separados dos dispositivos periféricos usados pelos operadores para controlá-los. Estes eram parte de uma mobília separada, o "console". Este termo foi mantido, mas seu significado foi alterado. Tornou-se mais ou menos sinônimo de "terminal", sendo um teclado e uma tela.

Com o desenvolvimento de computadores, sistemas operacionais tem oferecido vários consoles virtuais para permitir várias sessões independentes ao mesmo tempo, mesmo se houver apenas um teclado e tela. A maioria dos sistemas GNU/Linux oferecem seis consoles virtuais (modo texto), acessíveis, digitando as combinações de teclas **Control**+**Alt**+**F1** through **Control**+**Alt**+**F6**.

Por extensão, os termos "console" e "terminal" também pode se referir a um emulador de terminal em uma sessão X11 gráfica (como **xterm**, **gnome-terminal** ou **konsole**).