리눅스 부트 프로세스

김영휘

lachesis (at) kldp.org
http://lachesis.pe.kr

리눅스 부트 프로세스

지은이 김영휘

이 문서는 전원을 켰을 때 부터 로그인 프롬프트가 화면에 나타날 때까지 리눅스에서 무슨 일이 벌어지는지에 대한 문서입니다. 되도록 자세하게 적으려고 노력하였으며, 부팅 과정에서 사용되는 스크립트에 대해서도 분석(?)하도록 노력하였습니다. 레드햇 기반의 배포판을 기준으로 잡아 설명했습니다. 다른 배포판(debian)의 경우는 필요할 경우 부연설명을 하겠습니다.

문서의 버그패치 환영합니다. 제발 버그패치해 주세요. ^^

이 문서의 최신버전은 http://lachesis.pe.kr/etc/bootprocess/ (http://lachesis.pe.kr/etc/bootprocess/) 에서 구할 수 있습니다.

한 페이지에 다 보기를 원한다면, http://lachesis.pe.kr/etc/bootprocess/bootprocess.html (http://lachesis.pe.kr/etc/bootprocess/bootprocess.html) 을보세요. 출력할 때 편리합니다.

sgml(docbook) 소스입니다 : http://lachesis.pe.kr/etc/bootprocess/bootprocess.sgml (http://lachesis.pe.kr/etc/bootprocess/bootprocess.sgml)

고친 과정

고침 Version 0.1.12000. 8. 22 고친이 라키시스
최초로 문서를 완성했습니다.
고침 Version 0.2.02001. 7. 4 고친이 라키시스
docbook 으로 문서 포맷을 수정했습니다. 몇가지 버그를 수정했습니다. Debian GNU/Linux 의 경우를 _약간_ 고려했습니다.
고침 Version 0.2.12001. 8. 16 고친이 라키시스
/ file system 의 마운트 문제와 관련한 모호한 점에 대해 손을 좀 보았습니다.
고침 Version 0.2.22001. 10. 18고친이 라키시스
init 이후의 과정에 대해 약간의 설명을 더 추가했습니다.

차례

1장. 전원을 넣는 순간으로부터 init 프로세스가 시작될 때까지

1.1. lilo 로부터 커널 이미지 vmlinuz 의 로딩

우리의 PC 의 전원을 켜면 PC 는 BIOS 에 저장되어 있는 초기화 프로그램을 실행합니다. 초기화 프로그램에 의해 메모리 체크 등이 수행되고 필요한 초기화가 완료되고 나면, 하드 디스크 혹은 다른 부팅 매체 (플로피 디스크나 CDROM 등)의 0 번 섹터의 부트 프로그램을 읽습니다. 보통, 0번 섹터를 MBR (Master Boot Record, 하드디스크의 경우. 플로피디스크의 경우는 부트섹터라고 합니다.) 이라고 합니다. 여기는 리눅스를 위한 lilo 나 grub 등이 들어 있을 수도 있고, NT 나 OS/2 등 다른 운영체제의 부트 로더가 들어 있을 수도 있습니다. lilo 가 MBR 에 있다고 가정하고 계속 진행하겠습니다. lilo 는 커널을 실행하기 위해서 사용자의 입력을 기다립니다. 여러분이 컴퓨터를 켰을때 나오는

```
lilo:
```

라는 프롬프트가 그것입니다. 이때 우리는 lilo 라는 프롬프트 뒤에 커널에 주고싶은 옵션이나, 부팅하고자 하는 커널의 이미지를 지정해 줄 수 있습니다. 이에 관한 자세한 내용은 lilo HOWTO 문서를 참조하십시오.

이제, 사용자는 lilo 에 실행시키기를 원하는 커널의 이미지를 커널에 넘겨주고자 하는 옵션값과 함께 알려줍니다. 그리고, lilo 는 해당하는 커널의 이미지를 로딩해서 실행하게 됩니다. 기본 설정으로는 /vmlinuz 또는 /boot/vmlinuz 이미지가 메모리에 로드되고, 실행을 시작합니다.

vmlinuz 는 리눅스 커널의 압축 이미지입니다. 여기서 swapper 라고도 불리우는 프로세스 id 0 인 프로세스가 실행됩니다. 이 프로세스는 운영체제 그 자체라고도 할 수 있는 프로세스로써, 메모리 관리, 디스크 관리, 프로세스 관리 등을 수행합니다. 이 프로세스는 프로세스 id 1 인 init 라는 프로세스를 실행시키고는 본연의 기능인 swapper 로써의 기능을 수행하기 시작합니다. [1]

1.2. process 0 : swapper

여기서 swapper 가 무슨 일을 하는지 간단하게 이야기해 보겠습니다. [2]

유닉스 시스템에서 실행되는 모든 프로세스들은 "일생(lifetime)(?)"을 가지고 있습니다. 프로세스 생성에서부트 종료시까지... 그동안 cpu 를 점유하면서 실행되는 시간도 있을 것이고, 할 일 없이 사용자로부터의 입력을 기다린다든지의 이유로 "잠들어" 있는 시간도 있습니다. swapper 프로세스에 대해 설명하면서 이 이야기를 하는 이유는 swapper 가 하는 일이 바로, "잠들어" 있는 프로세스를 메모리에서 내려서 디스크 공간에 잠시 "스왑"시켰다가, 그 프로세

스가 깨어나야만 할 시기가 오면, 디스크의 프로세스를 다시 메모리로 적재해 주는 등의 일을 하기 때문입니다.

예를 들어서, 지금 시스템에 너무 많은 프로세스가 실행되고 있어서 그넘들이 다 들어갈 만큼 메모리가 크지 않다고 가정합시다. 그럴때, 시간이 좀 많이 걸리는 I/O 요청을 한 프로세스라든지, 사용자의 입력을 기다리는 프로세스 라든지.. 지금 sleep 상태로 있는 프로세스 (예를 들면, httpd 같은건 대부분의 시간을.. - 그리 접속이 많지 않은 웹 서버라면 - port 에서 요청이 들어오길 기다리면서 잠들어 있는 (asleep) 상태이겠죠?) 와 같은 넘들은 지금 당장 메모리에 있을 필요가 없는 것입니다. 그러면, 지금 메모리가 모자라니까 디스크로 "스왑" 을 해서 당장 실행되어야 하는 프로세스를 위한 메모리 공간을 늘리면 되겠죠? 이때 swapper 가 작동을 해서 메모리에 있는 프로세스를 디스크로 잠시 옮겨 두는 (swap out) 것입니다. 또, swapper 는 반대로 디스크에 스왑되어 있는 프로세스가 메모리로 적재되어서 실행되어야 할 필요가 있을 때에도 스왑된 프로세스를 메모리로 다시 올리는(swap in) 일을 하기도 합니다.

리눅스의 모든 프로세스는 모두 "부모" 프로세스를 가지고 있습니다. 즉, 그 프로세스를 생성시킨 넘이 존재한다는 얘기입니다. 그러나, 단 하나. pid 가 0 인 swapper 프로세스만은(이건 프로세스라기 보다는 운영체제 자체라고 해도 좋을 거 같습니다.) 부모가 존재하지 않고, lilo 등에 의해서 "수동"으로 실행이 됩니다. 나머지 프로세스는 모두 fork() 시스템 콜과 exec() 시스템 콜을 이용해서 생성이 됩니다.

또 여기서, fork() 시스템 콜과 exec() 시스템 콜에 대해 간단히 이야기하고 넘어가겠습니다. UNIX 시스템 프로그래밍을 공부해 보신 분들을 다 잘 아시는 내용일 것입니다.

fork() 나 exec() 모두 한 프로세스가 다른 프로세스를 실행시키기 위해 사용하는 시스템 호출입니다. 두 함수의 차이점을 위주로 설명하면, 우선 fork() 시스템 호출은 새로운 프로세스를 위한 메모리를 할당합니다. 그리고, fork() 를 호출한 프로세스를 새로운 공간으로 싸그리 복사합니다. 그리고 나서 원래 프로세스는 원래 프로세스대로 실행되고, fork() 를 이용해서 생성된 프로세스도 그 나름대로 fork() 시스템 콜이 수행된 라인의 다음 라인부터 실행이 됩니다. (새로 생성된 프로세스는 원래의 프로세스랑 똑같은 코드를가지고 있습니다.) 반면, exec() 시스템 콜은 fork() 처럼 새로운 프로세스를 위한 메모리를 할당하지 않고, exec() 를 호출한 프로세스의 메모리에 새로운 프로세스의 코드를 덮어씌워 버립니다. 따라서 exec() 를 호출한 프로세스가 아닌 exec() 에 의해 호출된 프로세스만 메모리에 남게 됩니다.

다시 간단하게 말하면, fork() 의 결과는 프로세스가 하나 더 생기는 것입니다. 물론, 프로세스 id (pid) 도 완전히 다른 또 하나의 프로세스가 생기는 것이죠. 반면, exec() 실행의 결과로 생성되는 새로운 프로세스는 없습니다. exec() 를 호출한 프로세스의 pid 가 그대로 새로운 프로세스에 적용이 되며, exec() 를 호출한 프로세스는 새로운 프로세스에 의해 덮어쓰여지게 됩니다.

주석

1. The design of the UNIX Operating System 에 의하면, 이 설명이 맞습니다. 그러나, 리눅스 시스템에서 프로세스 id 0 인 프로세스와 id 1 인 프로세스의 역할은 좀 다릅니다. 물론, 역사적인 이유로 0번 프로세스가 swapper 라고 불리우기는 하지만, 리눅스에서는 0 번 프로세스는 아무것도 하지 않는, idle 프로세스입니다. 즉, cpu 가 할 일이 없을 때 실행되는 프로세스입니다. 그래서 **ps aux** 를 해 보아도 PID 0 인 프로세스는 없습니다. 단, 모든 프로세

스들의 '조상' 임에는 틀림없죠.

ps aux 를 실행했을 때, 나오는 [kswapd], [bdflush] 등의 프로세스들은 커널 스레드(**kernel thread**) 라고 부르는 것들입니다. 자세한 내용은 이 문서의 성격에 맞지 않으므로 생략하겠습니다. 그런데, **ps aux** 를 실행했을 때, 1 번 프로세스는 [init] 라고 나오지 않고 그냥 init 라고 나오죠? (커널 2.4.4 기준) 그 이유는, 1번 커널 스레드인 (편의상 init 프로세스라고 부릅니다.) init 프로세스가 자신의 할 일을 마치고, `/sbin/init` 프로그램으로 자신을 대체해 버리기 때문입니다. 내용이 본문의 취지에서 너무 벗어난거 같네요 :-) 이만 줄이겠습니다.

2. 실제 리눅스 시스템에서는 커널 스레드들 중, kswapd, bdflush, kupdated, kreclaimd 등이 본문에서 설명하는 기능을 수행합니다.

2장. init 프로세스와 inittab 파일

2.1. init 프로세스와 inittab 파일

프로세스 id 1 번인 init 프로세스는 사용자들을 위해서 시스템을 설정하게 됩니다. pid 0 인 프로세스가 초기화(?)한 커널의 바탕에서 나머지 작업을 수행하는 것이죠.

init 프로세스(/sbin/init) 가 하는일의 대충은 다음과 같습니다 :

- 파일시스템의 구조 검사
- 파일시스템의 마운트[1]
- 서버 데몬을 띄우고,
- 사용자 로그인을 기다림
- 사용자가 로그인 하면, 사용자를 위한 쉘을 띄움

init 가 처음 시작해서 수행해야 할 작업들을 설정한 파일은 /etc/inittab 파일 입니다. init 는 새로운 실행레벨에서 실행할 프로세스를 결정하기 위해서 이 파일을 참조합니다. 다시 말하면, inittab 파일은 시스템의 상태에 따라서 해당하는 런레벨에서 init 프로세스가 수행해야 할 일들을 서술해 놓은 파일입니다. init 프로그램은 inittab 파일을 참조하여서 모든 새로운 런레벨에서 실행할 수 없는 프로세스가 만약 지금 실행중이면, 그 프로세스를 죽이고, 새로운 런레벨 에서 실행해야만 하는 프로세스 중 현재 실행되고 있지 않은 프로세스는 새로이 실행을 시킵니다.

2.2. /etc/inittab 파일의 예

아래에 예시한 것은 제 컴퓨터에 있는 inittab 파일입니다. (좀 허접하기 때문에 이해해 주시기 바랍니다. 단지 예시용으로 인용하였습니다. -_-;)

```
----------------------------------------------------------------------
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, >miquels@drinkel.nl.mugnet.org<
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
```

```
# 2 -
Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Things to run in every runlevel.
ud::once:/sbin/update

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -
h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -
c "Power Restored; Shutdown Cancelled"


# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
# xdm is now a separate service
x:5:respawn:/etc/X11/prefdm -nodaemon
```

2.3. runlevel

그럼 이제, inittab 파일을 한줄한줄 분석해 보겠습니다. # 으로 시작하는 줄은 주석이라는거다 알고 계실 겁니다. inittab 파일의 용도와 저자가 나오고, 각 런레벨(ren level) 에 대한 간략한 정의(?)가 나옵니다. [2]

```
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
```

렌레벨 **0**

시스템을 종료할 때 사용되는 런레벨입니다.

렌레벨 **1**

싱글 유저 모드에서 사용되는 레벨입니다. 여러분이 lilo: 프롬프트 에서 linux single 이라고 입력하면 런레벨 1 에서 리눅스가 시작하게 됩니다. [3] 이 때는 디폴트로 root 로 로그인되며, 대다수의 중요한 데몬들은 실행되지 않게 설정되어 있습니다.

렌레벨 **2**

NFS 를 지원하지 않는 다중 사용자 모드를 정의합니다. [4]

렌레벨 **3**

네트워킹을 지원하는 다중 사용자 모드입니다. (디폴트 런레벨로 주로 지정됩니다.)

렌레벨 **4**

여러분이 나름대로 정의해서 쓸 수 있는 런레벨입니다. 필요에 따라서 실행시키기 원하는 데몬이나 서비스를 /etc/rc.d/rc4.d 디렉토리 밑에 설정함으로써 여러분만의 특성을 갖는 고유의 런레벨을 정의할 수 있습니다.

렌레벨 **5**

X 를 실행시키기 위한 런레벨로 약속되어 있습니다.

렌레벨 **6**

시스템을 재부팅시키도록 정의된 런레벨입니다.

만약, inittab 파일의 처음에 나오는

```
id:3:initdefault:
```

의 라인을

```
id:6:initdefault:
```

혹은

```
id:0:initdefault:
```

등으로 지정해 버리면, 여러분의 리눅스 시스템은 부팅하자마자 종료되어 버리거나, 재시작을 거듭 반복하게 되어 버리므로 주의하시기 바랍니다. 그럴때에는 부팅시 lilo 에 옵션을 주어서 0이나 6 이외의 런레벨로 부팅해서 고쳐 주시면 됩니다.

잠깐! 런레벨이란 무엇인가... 하고 생각하시는 분들이 있을 겁니다. 아마, 여기까지 읽으셨다면, 대충 짐작은 하고 계시리라 생각합니다.

init 맨페이지의 정의 :

> 런레벨은 선택된 그룹의 프로세스만 시스템에서 실행되도록 허락 하기 위해 만든 소프트웨어 설정이다. [5]

시스템을 관리하다 보면, 시스템의 점검 혹은 디버깅을 위해서 보통때의 설정 과는 좀 다르게 몇가지 서비스를 시작하지 않고 부팅하고 싶은 경우가 있을 것입니다. 예를 들어서, 네트워크 지원 기능을 off 하고서 부팅하고 싶다든지, 다중 사용자 모드가 아니라 싱글 사용자 모드로 부팅하고 싶다든지 하는 경우가 생길 수 있습니다.

이처럼 서로 다른 설정으로 부팅하기 위해서 각각의 설정을 "런레벨" 이라는 이름으로 나누어 놓은 것입니다. 그리고, "런레벨" 에 대한 약속으로 0, 1, 2, 3, 5, 6 번 런레벨 [6] 은 미리 예약되어서 "시스템 종료, 재부팅, 싱글유저모드, 다중사용자 모드, 네트워크가 지원되지 않는 다중 사용자 모드 등으로 하자" 라고 약속을 해 둔 것입니다. (사실 예약된 런레벨은 0, 1, 6 밖에 없습니다. 하지만 나머지 2, 3, 5 번은 "관례상" 각각의 해당 용도로 사용하고 있는 것 같습니다. 런레벨 2, 3 은 SunOS 등 다른 System V 계열의 init 프로세스를 쓰는 운영체제에서도 리눅스와 동일하게 사용하고 있습니다.) 그 외에 7, 8, 9번 런레벨도 사용자가 정의해서 쓸 수 있지만, 관례상 그렇게 하지 않는다고 합니다. 그리고, S 와 s 런레벨도 특수한 런레벨로써 존재합니다.

만약 여러분이 시스템 종료시에 디폴트로 제공되는 일에 더해서 실행시키고 싶은 것이 있다면, 런레벨 0 에 추가를 하시면 되는 것입니다.

이렇게 각각의 런레벨에 원하는 작업을 추가하기 위해서는 `/etc/rc.d/rcN.d` [7] 디렉토리 아래의 심볼릭 링크들을 조정하시면 됩니다. 자세한 내용은 이 문서 4절에 적어 두었습니다.

2.4. man inittab

이제 inittab 파일의 형식을 살펴 보도록 하겠습니다. 자세히 inittab 파일을 보신 분이라면 모든 줄이 다음과 같은 형식으로 되어 있다는 것을 아셨을 겁니다.

```
id:run-levels:action:process
```

제일 처음에 나오는 id 는 해당 state를 구분하기 위한 레이블이라고 보시면 됩니다. 그리고, 그 다음의 rul-levels 는 그 줄의 내용을 적용하기 위한 런 레벨의 목록입니다. 그리고, action 은 그 줄(엔트리라고 하겠습니다.), 엔트리에 의해 실행되는 프로세스를 어떻게 할 것인가에 대한 설명입니다. 마지막으로 나오는 process 는 프로세스의 실행파일의 경로와 프로세스에 넘겨줄 인수입니다. (쉘에서 실행시키는 명령어와 같은 형식이라고 생각하세요, 즉, 해당 엔트리를 실행할 때 process 부분에 나오는 명령어로 실행하라는 뜻입니다. -_-;)

action 부분에 올 수 있는 키워드는 다음과 같습니다. (맨페이지에서 참조한 내용입니다.)

```
wait

  프로세스를 실행하고, 다음 줄의 엔트리로 가지 말고, 실행한 프로세스가
  종료하기까지 기다리라는 뜻입니다.


respawn

  프로세스를 실행하고, 그 프로세스가 죽게 되면, 다시 실행시키라는 의
  미입니다. 주로 getty 등의 프로세스입니다.


initdefault

  디폴트 런레벨을 지정하겠다는 뜻입니다. 위에 예시한 inittab 파일
  의 첫줄에 이 키워드가 나오는데, 해석하면, 런레벨 3 을 디폴트 런
  레벨로 지정한다는 의미입니다. 즉, initdefault 엔트리는 시스템의
  부트 프로세스가 종료된 후에 진입할 런레벨을 가리키는 엔트리입니
  다. process 필드는 아무런 의미가 없게 됩니다.


off : 아무것도 하지 말라는 뜻입니다.


once

  이미 실행되고 있는 프로세스라면 실행하지 말고, 실행되고 있지 않으면
  단지 한번만 실행시키라는 뜻입니다. 단, wait 처럼 기다리거나 하지는 않
  도록 지정합니다.


boot

  시스템 부팅시에 실행되어야 할 프로세스를 가리킵니다. 런레벨 필드는
  아무런 의미가 없게 됩니다. (무시됩니다.)
```

bootwait

프로세스가 시스템 부팅시에 실행되도록 지정합니다. 단지, init 가 그 프로세스가 종료되기를 기다린다는 점에서 boot 와 다릅니다. 예를들어 /etc/rc 와 같은 것이 있습니다.

sysinit

프로세스가 시스템 부트시에 실행되게 합니다. 그리고, 이 엔트리는 다른 boot 나 bootwait 엔트리들이 실행되기 전에 실행되는 엔트리가 됩니다. 런레벨 필드는 무시합니다.

powerwait

init 프로세스가 SIGPWR 시그널을 받으면 실행되는 프로세스입니다. SIGPWR 시그널은 전원과 관련해서 무엇인가 문제가 있음을 가리키는 시그널입니다. 이때 init 는 프로세스가 종료되기까지 대기합니다.

powerfail

powerwait 항목과 마찬가지이지만, 프로세스가 종료되기까지 기다리지 않는다는 점이 다릅니다.

powerokwait

이 엔트리 역시 init 가 SIGPWR 시그널을 받으면 실행될 프로세스를 지정합니다. 그러나, 이 엔트리에서 지정된 프로세스는 /etc/powerstatus 파일에 OK 라는 단어가 있을 때만 실행됩니다. 즉, 전원이 다시 돌아왔을 때만 실행됩니다.

ctrlaltdel

init 가 SIGINT 시그널을 받게 되면 실행할 프로세스를 지정합니다. 즉, 시스템 콘솔에서 누군가가 CTRL-ALT-DEL 키를 눌렀을 때 이 엔트리에서 지정한 프로세스가 실행되는 것입니다.

kbrequest

이 엔트리에서 지정하는 프로세스는 init 가 키보드 핸들러로부터 콘솔에서 특수키 조합이 눌려졌다는 시그널을 받으면 실행되는 프로세스입니다. 키맵 파일과 함께 쓰일 수 있습니다.

2.5. /etc/inittab 의 분석

아마 앞절의 내용만 보셔도 자신의 inittab 파일의 대부분은 이해하실 수 있을 겁니다. 이제, 쓸데없을지도 모르지만, 제 inittab 파일을 한줄씩 분석해 보겠습니다.

```
id:3:initdefault:
```

이 줄은 디폴트 런레벨을 3으로 지정하는 줄입니다. initdefault 키워드 뒤에는 process 필드가 무시된다고 윗부분에서 설명되어 있지요? ^^; 이 줄에 의해서 부팅한 후에 런레벨 3 으로 커널의 런레벨이 조정됩니다. 만약, 처음에 리눅스를 부팅하면, 바로 X 로 들어가시는 분이라면 이부분이 다음과 같이 되어 있을 겁니다.

```
id:5:initdefault:
```
[8]

런레벨 5에 대한 설명을 inittab 파일에서 보십시오. X11 이라고 되어 있지요?

그러면, 그 다음줄입니다.

```
si::sysinit:/etc/rc.d/rc.sysinit
```
[9]

이 라인에는 sysinit 라는 키워드가 action 필드에 지정되었습니다. action 필드에 지정할 수 있는 키워드 중 sysinit 를 찾아보면, 프로세스가 시스템 부팅시에 실행되며, 런레벨은 무시한다는 내용과, 다른 boot 나 bootwait 가 action 필드에 지정된 프로세스들보다 먼저 실행된다는 내용이 있습니다. 즉, 제일 처음 init 프로세스가 실행시키는 명령입니다. 정확히 말하면, start-up 스크립트라고 할 수 있습니다. process 필드에 /etc/rc.d/rc.sysinit 라고 명시되어 있습니다. 다시 한번 단순화시켜서 이야기하면, 위에 예시한 inittab 파일을 사용하는 제 리눅스 박스는 전원을 올리면, /etc/rc.d/rc.sysinit [10] 가 실행된다는 것입니다. rc.sysinit 의 내용은 나중에 살펴보기로 하고, 다음줄로 진행하겠습니다.

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

보시고, 짐작하신 바와 같이 각 런레벨별로 지정된 시작 스크립트들입니다. /etc/rc.d/rc 스크립트에 해당 런레벨을 인수로 넘겨주는군요. /etc/rc.d/rc 스크립트의 내용은 나중에 하나씩 분석해 보겠습니다. 각 런레벨 별로 지정된 스크립트는 그 실행이 종료될 때까지 init 프로세스가 진행하지 않고, 기다리라는 의미의 wait 키워드를 action 필드에 지정해 두었군요.

```
ud::once:/sbin/update
```

그 다음줄에는 action 필드에 once 키워드가 쓰였습니다. 그래서 각 런레벨별로 한번씩만 실행되는 프로세스입니다. /sbin/update 를 실행하는군요. /sbin/update 는 bdflush 라는 커널 데몬(커널 스레드)을 실행시킵니다. 이 데몬은 버퍼를 갱신한다든지 하는 일을 합니다. 자세한 내용은 man update 를 해 보시면 알 수 있습니다.

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

그 다음으로 나오는 것은 사용자가 콘솔에서 CTRL-ALT-DEL 키의 조합을 눌려서 싱글 유저 모드로 들어가거나 혹은 시스템을 셧다운 하려고 할 때 불려지는 프로세스를 정의합니다. /sbin/shutdown -t3 -r now 로 정의했군요. 3초 후에 재부팅하게 됩니다. shutdown 이 재부팅을 수행하기 위해서는 init 프로세스에게 런레벨을 바꾸도록 하라는 시그널을 보내게 됩니다. 런레벨 0은 시스템 halt 에 사용되고, 런레벨 6은 재시작, 1은 싱글유저 모드에 사용됩니다.

재미있는 것으로, 사용자가 (root 사용자일 경우) init 프로세스에게 구체적으로 어떤 런레벨에서 init 를 수행시킬지 커맨드라인에서 명시해 줄 수 있습니다.

/sbin/init N

이라고 명령을 내리면, N 에 명시된 실행레벨로 init 가 진입하게 됩니다.

/sbin/init 6

이라고 하면, init 는 런레벨 6 으로 진입하게 되면서 시스템은 재부팅 되게 됩니다. 다른 System V 계열의 운영체제에서는 이러한 경우를 위해서 특별히 telinit 라는 명령어도 준비해 놓고 있습니다. (리눅스도 마찬가지입니다.)

```
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
```

다음 두줄은 전원에 관계된 것으로, UPS 를 사용하는 분들에게만 해당되는거 같군요.

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

그리고, 거의 마지막 부분에 붙어 있는 여섯줄, 이 줄은 각 런레벨 2,3,4,5 일 때 실행되는 것으로써, 콘솔을 열고, 응답을 기다리는 부분입니다. 앞서의 모든 초기화 과정이 끝나고 마지막으로 실행되는 부분이죠. 컴을 켰을 때 바로 X 가 시작되는 분들은

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

등과 같이 런레벨 5에만 해당되는 로그온 프로세스가 명시되어 있을 겁니다.

주석

1. 이와 관련해서 정헌학님(hunhak94@somedomain) 께서 다음과 같은 질문을 주셨습니다 :

   > 이 부분에서 init 다음에 파일시스템 마운트fstab 이 실행된다고 하면 초기 부팅시 /sbin 이 마운트 되지 않은 상태 즉, fstab 이 실행되지 않은 상태에서 어찌 /sbin/init 를 찾아서 실행을 할 수가 있죠?? 결국 마운트 되지 않은 상태에서 파일 시스템에 접근을 할 수 있다는 이야기인데... 잘 이해가 안갑니다.

   다음은 저의 답장입니다 :

   > 한번 훈학님의 시스템에서 dmesg 를 해 보세요. 아마도 NET, 즉, TCP/IP 프로토콜 스택과 유닉스 도메인 소켓을 초기화 한 후에 root 를 readonly 로 마운트 하는 부분이 있을 겁니다. 이것은, mount 프로그램을 쓰는 것이 아니라, 커널의 VFS(Virtual File System) 에서 처리하는 것이죠, 그리고, fstab 을 참조한 마운트는 start_kernel() 함수가 종료되면서, execve() 함수로 init 를 실행시킨 후에 일어납니다. 이때, readonly 로 마운트되어 있던 root 파일 시스템이 다시 read/write 로 마운트되게 됩니다. 흠... 그러면, /sbin 을 root 파일시스템에 두지 않고 다른 파일 시스템이 둔다면 어떻게 될까요... 궁금해지네요.. 이 문제는 시스템을 한번 밀고, /sbin 을 다른 파일시스템으로 두어서 테스트 해 본 후에 다시 메일을 드리겠습니다. 다행히도 테스트할 수 있는 장난감용 컴퓨터가 있거든요 ^^ 메일 다시한번 감사드립니다. 즐거운 나날들 되세요 ^^*** 꼬랑지. 답장이 좀 늦었네요... 그간 바빠서, 훈학님의 메일에 대해 곰곰히 생각해 볼 시간이 별로 없었답니다 -_-;;;

2. 이처럼 런레벨을 정의하고, 런레벨 별로 디렉토리를 할당해서 init 를 실행시키는 방식의 init 를 system V 스타일의 init 라고 합니다. init 프로그램은 크게 BSD 스타일과 system V 스타일의 두종류가 있는데, 우리나라에서 사용하는 대부분의 리눅스 배포본 (실제 제가 써본 것은 레드햇 계열과 데비안밖에는 없습니다. _._)에서는 system V 스타일의 init 를 사용합니다.
3. lilo 에 주는 명령은 lilo HOWTO 를 참조하세요.
4. 데비안 배포본의 경우, 레벨 2 가 디폴트 런레벨(Full-multiuser mode)
5. A runlevel is a software configuration of the system which allows only a selected group of processes to exist.
6. 레드햇의 경우
7. 데비안의 경우 `/etc/rcN.d` 입니다.
8. 이렇게 되어 있지 않을 경우는 xdm 혹은 gdm 을 사용하는 경우입니다.
9. 데비안의 경우는 si::sysinit:/etc/init.d/rcS 로 되어 있습니다.
10. 당연히 데비안은 /etc/init.d/rcS 이겠죠.

3장. init : 그 이후

앞 절에서 설명한 바와 같이 init 프로세스가 자신에게 할당된 초기화 과정을 모두 끝마치고 나면, inittab 파일의 맨 끝부분에 명시된 것처럼 mingetty 혹은 getty 프로세스를 실행시킵니다. 실행시키는 방법은 최초에 pid 0 의 프로세스 [1] 에 의해 실행된 init 가 fork() 시스템 콜을 한번 수행하고, 그에 의해 생성된 자식 init 가 exec() 시스템 콜을 이용해서 getty 혹은 mingetty 등의 프로세스를 수행하게 됩니다.

getty 가 하는 일은 화면에

```
login :
```

프롬프트를 띄우고서 사용자의 userid 가 입력되기를 기다리는 일입니다. 사용자가 자신의 userid 를 입력하고서 엔터키를 치게 되면, getty 는 `/bin/login` 을 실행시키게 됩니다.

/bin/login 프로그램은

```
Password:
```

라는 프롬프트를 띄우고, 사용자가 password 를 입력하기를 기다립니다. 사용자가 암호를 입력하면, `/etc/passwd` 혹은 `/etc/shadow` 파일로부터 패스워드를 읽어서 암호화 한 패스워드와 일치하는지를 검사합니다. 만약 패스워드가 일치하지 않으면, login 프로그램은 에러를 내고 종료합니다. 그러면, init 프로세스는 다시 자신을 fork() 하고, /sbin/getty 프로그램을 exec() 해서 화면에 로그인 프롬프트를 출력하도록 하고 사용자의 입력을 기다리게 됩니다. 만약 패스워드가 일치하면, login 프로그램은 홈디렉토리를 바꾸는 등 여러가지 작업을 하고, 최종적으로 exec()를 하여, 로그인 쉘 [2] 을 띄우게 됩니다.

그러면, 사용자는 자신의 쉘 (주로 bash) 로 작업을 하고, 쉘에서 입력하는 명령어들은 먼저 쉘이 fork() 를 한 후 쉘의 자식프로세스가 exec() 를 하여 실행되고, 종료하는 것입니다. 이 과정을 간단하게 그림으로 그리면 다음과 같습니다.

```
+-------------+
|    init     |
|   (pid 1)   |
+------+------+
       | <------------------------------------+
       | fork()                               | respawn
      \|/                                     |
+------+------+                               |
| "자식" init |                              |
|  (pid 266)  |                               |
+------+------+                               |
       |                                      |
       | exec()                               |
      \|/                                     |
+------+------+                               |
|  mingetty   |                               |
```

```
|  (pid 266)  |                                 |
+------+------+                                 |
       |                                        |
       | exec()                                 |
      \|/                                       |
+------+------+                                 |
|    login    |                                 |
|  (pid 266)  |                                 |
+------+------+                                 |
       |                                        |
       | exec()                                 |
      \|/                                       |
+------+------+                                 |
|    bash     +----------------------------- 쉘 종료
|  (pid 266)  |          로그아웃
+------+------+
       | 명령어 수행
       | fork()
      \|/
+------+------+
|    bash     |
|  (pid5909)  |
+------+------+
       |
       | exec()
      \|/
+------+------+
|다른 프로세스|
|  (pid5909)  |
+------+------+
```

주석

1. 최초의 init 는 커널의 init 쓰레드에 의해 실행됩니다.
2. ps 를 수행했을 때 bash 등의 앞에 '-' 기호가 붙어서 -bash 등과 같이 표시된 것이 로그인 쉘입니다. 자세한 것은 다른 문서를 찾아보세요.

4장. /etc/rc.d/rc.sysinit, /etc/rc.d/rc

rc.sysinit 스크립트는 살펴본 결과 크게 특이할 것은 없는 것 같습니다. 하는 일들은 레드햇 기반 배포본을 모델로 설명하겠습니다. [1] 이 파일은 inittab 의 정의에 따라, 시스템 초기화시 맨 먼저 딱 한 번 실행되는 초기화 스크립트입니다.

- 기본적인 path 설정
- /etc/sysconfig/network 파일이 있으면 그 스크립트를 실행시킴
- 키맵의 로딩
- 시스템 폰트의 로딩
- 스왑 영역의 활성화
- 디스크 검사 (fsck)
- /proc 파일시스템의 마운트
- 루트 파일시스템을 rw 모드로 다시 마운트하기
- /etc/HOSTNAME 파일의 설정
- /etc/mtab 파일에 루트와 /proc 파일시스템의 엔트리 추가하기
- 커널 모듈들 로드하기
- 시스템 시간 설정
- 등등....

의 일을 수행합니다. 자세한 것은 rc.sysinit 파일을 찬찬히 읽어 보면 알 수 있습니다.

정작, 우리가 주의를 기울일 초기화 스크립트는 각종 데몬들을 실행시키고, 죽이는데 사용되는 /etc/rc.d/rc 스크립트입니다. 이 스크립트는 인수로 실행 레벨을 받습니다.(-_-; 표현이 좀 매끄럽지 못하군요) 즉,

```
/etc/rc.d/rc n
```

과 같은 형식으로 실행시킵니다. n 에는 실행레벨이 들어갑니다. 좀 더 나은 이해를 위해서 다음의 명령을 실행시켜 보십시오.

```
ls /etc/rc.d
```

실행 결과로 /etc/rc.d 디렉토리의 내용이 나오는데, 그 중에 다음과 같은 디렉토리들이 있습니다.

```
rc0.d/
rc1.d/
```

```
rc2.d/
rc3.d/
rc4.d/
rc5.d/
rc6.d/
```

각각의 디렉토리는 rc런레벨.d 로 이름이 지어져 있습니다. 각 디렉토리 아래에는 해당 런레벨에서 실행할 서비스나 프로세스들이 정의되어 있습니다. 한 디렉토리를 정해서 어떤 파일들이 있는지 살펴보겠습니다.

```
$ ls -laF /etc/rc.d/rc3.d
drwxr-xr-x  ./
drwxr-xr-x  ../
lrwxrwxrwx  K05keytable -> ../init.d/keytable*
lrwxrwxrwx  K15gpm -> ../init.d/gpm*
lrwxrwxrwx  K15proftpd -> ../init.d/proftpd*
lrwxrwxrwx  K60atd -> ../init.d/atd*
lrwxrwxrwx  K60crond -> ../init.d/crond*
lrwxrwxrwx  K80random -> ../init.d/random*
lrwxrwxrwx  K89portmap -> ../init.d/portmap*
lrwxrwxrwx  K92apmd -> ../init.d/apmd*
lrwxrwxrwx  K96pcmcia -> ../init.d/pcmcia*
lrwxrwxrwx  S10network -> ../init.d/network*
lrwxrwxrwx  S30syslog -> ../init.d/syslog*
lrwxrwxrwx  S50inet -> ../init.d/inet*
lrwxrwxrwx  S55named -> ../init.d/named*
lrwxrwxrwx  S80sendmail -> ../init.d/sendmail*
lrwxrwxrwx  S90mysql -> ../init.d/mysql*
lrwxrwxrwx  S91smb -> /etc/rc.d/init.d/smb*
lrwxrwxrwx  S99local -> ../rc.local*
```

보시는 바와 같이 S 로 시작하는 파일들과, K 로 시작하는 파일들이 주욱~ 있는데, 거의 /etc/rc.d/init.d 아래의 파일들에 링크가 되어 있습니다.

이처럼, rcN.d 디렉토리의 모든 파일들은 오로지 링크로만 되어 있으며, 실제 서비스를 시작하거나 종료하는 스크립트는 /etc/rc.d/init.d 디렉토리 아래에 존재합니다.

S 로 시작하는 파일은 해당 서비스를 실행시키는데 사용되고, (Start) K 로 시작하는 파일은 해당 서비스를 죽이는데 사용됩니다.(Kill) 제 컴퓨터에서는 디폴트 런레벨인 3 에서 network, syslog, inet, named, sendmail, mysql, smb 등을 실행하도록 구성해 두었습니다. (아파치와 proftpd도 실행이 되는데, 일일이 링크 걸기가 귀찮(^^;)아서 S99local 이 가리키는 /etc/rc.d/rc.local 안에 시작 스크립트를 적어 두었습니다.)

그럼 재미삼아서, /etc/rc.d/rc0.d 디렉토리의 내용도 살펴 보겠습니다.

```
$ ll /etc/rc.d/rc0.d
drwxr-xr-x  ./
drwxr-xr-x  ../
lrwxrwxrwx  K05keytable -> ../init.d/keytable*
lrwxrwxrwx  K15gpm -> ../init.d/gpm*
```

```
lrwxrwxrwx  K15proftpd -> ../init.d/proftpd*
lrwxrwxrwx  K30sendmail -> ../init.d/sendmail*
lrwxrwxrwx  K45named -> ../init.d/named*
lrwxrwxrwx  K50inet -> ../init.d/inet*
lrwxrwxrwx  K60atd -> ../init.d/atd*
lrwxrwxrwx  K60crond -> ../init.d/crond*
lrwxrwxrwx  K80random -> ../init.d/random*
lrwxrwxrwx  K85netfs -> ../init.d/netfs*
lrwxrwxrwx  K89portmap -> ../init.d/portmap*
lrwxrwxrwx  K90killall -> ../init.d/killall*
lrwxrwxrwx  K90mysql -> ../init.d/mysql*
lrwxrwxrwx  K90network -> ../init.d/network*
lrwxrwxrwx  K92apmd -> ../init.d/apmd*
lrwxrwxrwx  K96pcmcia -> ../init.d/pcmcia*
lrwxrwxrwx  K99syslog -> ../init.d/syslog*
lrwxrwxrwx  S00halt -> ../init.d/halt*
```

예상했던 대로 거의 모두가 K 로 시작하는 링크만 존재합니다. 단 하나 S 로 시작하는 링크는 halt 입니다. 시스템 종료를 하는 스크립트에 링크가 걸려 있습니다.

이제, 이렇게 실행레벨 별로 적절한 디렉토리를 찾아서 그 디렉토리 아래의 링크가 가리키는 스크립트들을 실행시키는 스크립트인 /etc/rc.d/rc 를 살펴보겠습니다.

아래에 제 컴퓨터의 /etc/rc.d/rc 파일의 일부를 발췌해 보았습니다. 레드햇 인스톨한 후에 하나도 바꾸지 않았습니다. (^^;)

```
--------------------------------------------------------------------
#!/bin/bash
#
# rc            This file is responsible for starting/stopping
#               services when the runlevel changes. It is also
#               responsible for the very first setup of basic
#               things, such as setting the hostname.
#
# Original Author:
#               Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#

# Source function library.
. /etc/rc.d/init.d/functions

... 생략 ...

# 만약, 첫번째 인자로 넘어온 런레벨에 해당하는 디렉토리가 있다면, 우선,
# 지금 실행중인 프로세스를 죽입니다.
# 프로세스를 죽이는 데 사용되는 명령어는 K 라는 접두어가 붙은 파일입니다.
# 런레벨에 해당하는 디렉토리는 /etc/rc.d/rcn.d 로써, n 에는 0, 1, 2, 3, 4,
# 5, 6등 런레벨의 숫자가 붙습니다. ls /etc/rc.d 를 실행해 보시면 rc0.d
# rc1.d... 와 같은 디렉토리들이 존재하는 것을 볼 수 있습니다.

if [ -d /etc/rc.d/rc$runlevel.d ]; then
    # 먼저, kill 스크립트를 실행시킨다.
    for i in /etc/rc.d/rc$runlevel.d/K*; do
```

```
        # 스크립트가 존재하는지 체크한다.
        [ ! -f $i ] && continue

        # [KS]??foo.{rpmsave,rpmorig} 스크립트를 실행시키지 않는다.
        # [KS]??foo.{rpmsave,rpmorig} 의 뜻은... 정규표현식을 조금 공부해
        # 보면 알 수 있다.
        # 첫 글자가 K 혹은 S 로 시작하며, 그 뒤에 임의의 문자 두개가 오고,
        # 그 뒤에 foo. 이 오며, 끝에 rpmsave 나 rpmorig 가 붙은 형태의 파일
        # 이름을 뜻한다.
        [ "${i%.rpmsave}" != "${i}" ] && continue
        [ "${i%.rpmorig}" != "${i}" ] && continue
        [ "${i%.rpmnew}" != "${i}" ] && continue

        # Check if the subsystem is already up.
        subsys=${i#/etc/rc.d/rc$runlevel.d/K??}
        [ ! -f /var/lock/subsys/$subsys ] && \
            [ ! -f /var/lock/subsys/${subsys}.init ] && continue

        # Bring the subsystem down.
        if egrep -q "(killproc |action )" $i ; then
            $i stop
else
            action "Stopping $subsys" $i stop
        fi
    done

# 프로세스를 모두 죽였다면, 이제, 해당 런레벨에서 실행되어야 하는
# 프로세스들을 실행시킬 차례입니다. 프로세스를 실행하는 명령어는 S 라는
# 접두어가 붙은 파일입니다.

    for i in /etc/rc.d/rc$runlevel.d/S*; do
        # 스크립트가 존재하는지 확인한다.
        [ ! -f $i ] && continue

        # Don't run [KS]??foo.{rpmsave,rpmorig} scripts
        [ "${i%.rpmsave}" != "${i}" ] && continue
        [ "${i%.rpmorig}" != "${i}" ] && continue
        [ "${i%.rpmnew}" != "${i}" ] && continue

        # Check if the subsystem is already up.
        subsys=${i#/etc/rc.d/rc$runlevel.d/S??}
        [ -f /var/lock/subsys/$subsys ] || \
            [ -f /var/lock/subsys/${subsys}.init ] && continue

        # 서비스를 실행시킬지 실행 시키지 않을지를 사용자에게 물어보는
        # "confirm" 모드인지를 확인한다. 만약 confirm 모드이면, 사용자에게
        # 해당 프로세스의 실행 여부를 질의한다.
        # confirm 함수는 /etc/rc.d/init.d/functions 에 정의되어 있다.
        [ -n "$CONFIRM" ]  &&
            {
                confirm $subsys
                case $? in
                    0)
                        :
```

```
                    ;;
                2)
                    CONFIRM=
                    ;;
                *)
                    continue
                    ;;
            esac
        }

        # Bring the subsystem up.
        if egrep -q "(daemon |action )" $i ; then
            $i start
        else
          if [ "$subsys" = "halt" -o "$subsys" = "reboot" -o \
          "$subsys" = "single" ]; then
              $i start
          else
              action "Starting $subsys" $i start
          fi
        fi
    done
fi
```

세부적인 내용은 스크립트를 차분히 분석해 보시면 아실 수 있을 것입니다. 여기서 중요한 것은, 자신이 부팅시에 실행시키기를 원하는 데몬을 지정할 수 있다는 것입니다. rc 스크립트에 보면, rc 에 인수로 건네진 런레벨의 숫자를 보고, 해당하는 디렉토리에서 K 로 시작하거나, S 로 시작하는 모든 파일을 찾아서 죽이거나 실행시킵니다.

```
for i in /etc/rc.d/rc$runlevel.d/K*; do
....
$i stop
```

과

```
for i in /etc/rc.d/rc$runlevel.d/S*; do
....
$i start
```

부분이 그 일을 하게 됩니다.

우리가 다중 사용자 모드로 부팅했을 때, 즉, 컴퓨터를 켰을 때 자동으로 실행시키고자 하는 데몬 중에, 예를들어, httpd 가 있다고 가정하면, httpd 가 컴을 켰을 때 자동으로 실행되게 하기 위해서 /etc/rc.d/rc3.d 디렉토리 아래에 SNNhttpd 라는 파일과 KNNhttpd 파일을 심볼릭 링크로 httpd 혹은 apachectl 에 링크시켜 주면 되는 것입니다.

그러면, 부팅시에 런레벨 3으로 진입하면서, /etc/rc.d/rc3.d 디렉토리의 S* 파일들을 실행하게 되고, (S 뒤에 번호가 있는데 그 순서로 실행됩니다.) 우리가 만든 SNNhttpd 링크도 실행하게

됩니다. (여기서 NN 은 적절한 번호입니다. 네트워크가 활성화 된 후에 apache 데몬이 실행되어야 하는것을 고려해 적절하게 번호를 할당해 주시면 됩니다.

참고할 부분은, 해당 프로세스를 실행시키거나 종료시키는 스크립트는 반드시, 항상, 인수로 start 혹은 stop 이라는 것을 넘겨 받도록 작성해야 한다는 것입니다. 만약 여러분이 해당 프로세스를 실행 혹은 종료시키는 스크립트를 직접 만들기를 원하신다면, 다음과 같은 코드가 반드시 포함되어서 /etc/rc.d/rc 스크립트에서 실행할 수 있도록 작성을 하셔야 합니다. 그리고, 시스템 관리상, 일관성 있게, 그 스크립트는 /etc/rc.d/init.d 에 넣어 주시는것이 보기에 좋을 것 같습니다.

```
. /etc/rc.d/init.d/functions

... 중략 ...

case "$1" in
 start)
     # 이 부분에 시작하는 코드를 넣는다.
     ;;
 stop)
     # 이 부분에 종료하는 코드를 넣는다.
     ;;
 *)
     # 만약 start 혹은 stop 이 명시되지 않으면 다음줄을 실행한다.
     echo "Usage: skeleton {start|stop}"
     exit 1
esac

exit 0
```

이게 좀 복잡하다고 느끼시는 분들은 /etc/rc.d/rc.local 스크립트가 있는데, 이 스크립트 마지막 부분에 실행시키기를 원하는 데몬을 실행하는 코드를 넣어 주시면 됩니다. 단, rc.local 스크립트는 실행레벨 2, 3, 5 일 때만 실행되도록 설정되어 있습니다. rc.local 스크립트는 rcN.d 디렉토리에서 S99local 이라는 이름으로 링크되어 실행됩니다. S99local 이라는 파일이 rc2.d rc3.d rc5.d 에만 있죠?

주석

1. 데비안의 경우도 파일의 위치와 이름만 조금 다를 뿐 원리는 같습니다.

5장. 꼬릿말

여러가지로 부족한 문서이지만, 나름대로 리눅스의 부트 프로세스에 대해 알고 있는 내용들을 정리해 보았습니다. 제가 처음 리눅스를 공부할 때 이에 관한 자료가 없어서 여기저기 삽질하면서 고생한 것이 생각나는군요 T_T 부디 초보자(나도 아직 초보를 못벗어난거 같지만 -_-;)님들이 이 문서를 보고 자신의 리눅스 시스템 설정을 바꾸는 일을 좀 더 수월하게 느꼈으면 좋겠습니다. 이 문서가 다른 많은 문서들 사이에서 쓸데없는(!) 문서 하나를 더하는 꼴이 되어서 자료를 찾으시는 분들께 혼란을 끼치게 되지나 않을까 하는 걱정도 있지만.. ^^; 감히 이렇게 kldp 에 제 저작물을 올려 봅니다.

읽으시는 분들의 따끔한 지적을 기다리겠습니다. - 2000. 8

애초에 계획한 것은 데비안의 경우를 모두다 포함시키는 것이었는데, 하다보니 게으름이 나는군요. 헤헤... 원리는 같기 때문에 조금만 삽질을 해 보시면 데비안에서도 무난히 적용할 수 있을 것입니다. - 2001. 7

참고문헌

Essential System Administration, FrischAEleen, O'REILLY .

The Design of the UNIX Operating System, BachMorris, Prentice Hall .

또다른 리눅스의 세계, 이만용, kldp .

AGETTY(8), INIT(8), INITTAB(5), Linux MAM PAGE .

Understanding the Linux Kernel, Bovet et al.Daniel P., O'REILLY .

Advanced Programming in UNIX environment, StevensRichard, Addison Wesley .