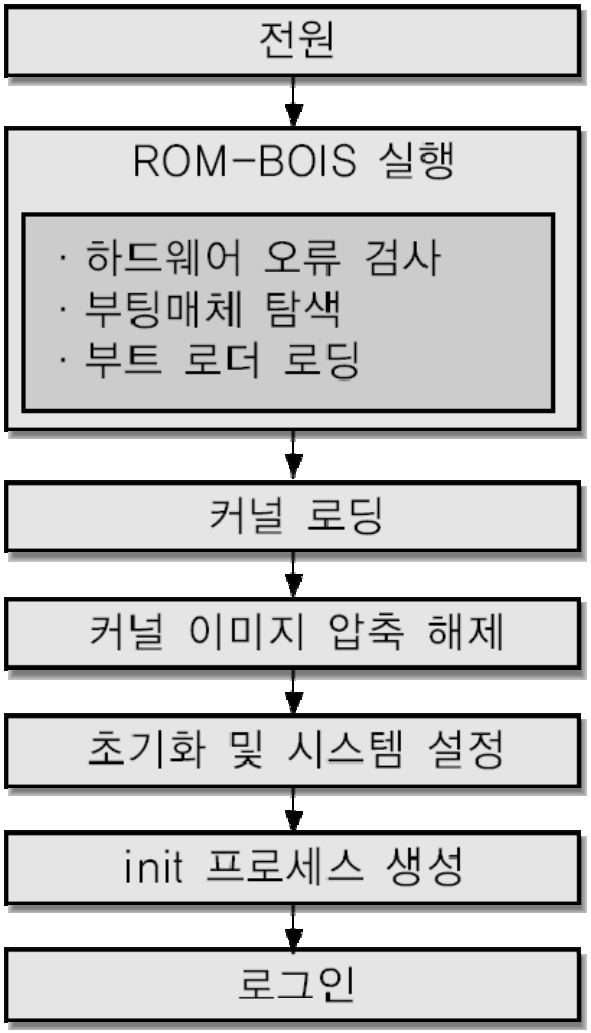
3장 리눅스 부팅 과정

- 리눅스 시스템 전체 부팅 과정
- start_kernel() 함수
- inittab 파일

리눅스 시스템 전체 부팅 과정



[그림 3-1] 리눅스 시스템 부팅 과정

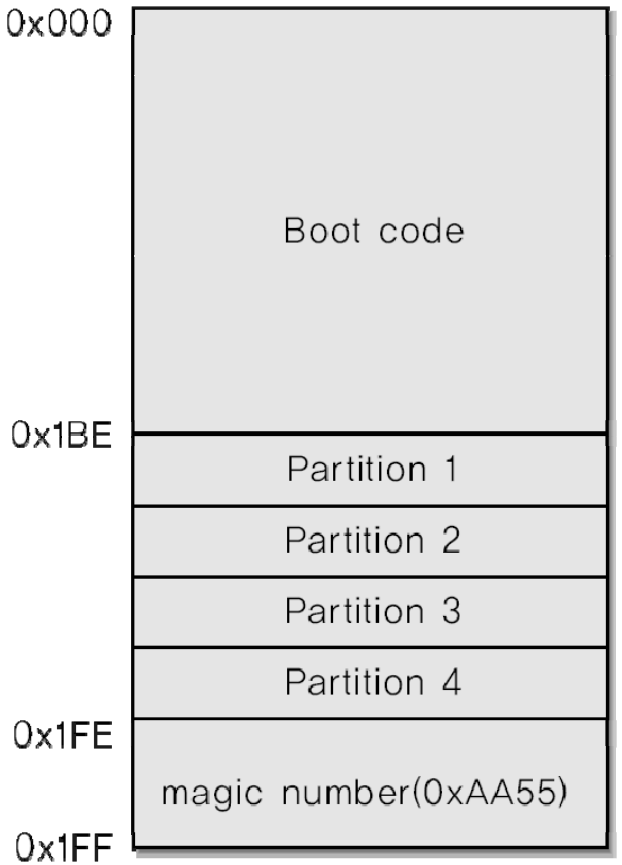
리눅스 시스템 전체 부팅 과정

- ROM-BIOS 실행
  - 메인보드의 ROM-BIOS가 실행
  - POST(power-on self test)
  - 장치들을 초기화하고 부팅이 가능한 장치 탐색

- 메모리에 부트 프로그램 적재
  - MBR(Master Boot Record)에서 부트 프로그램(부트로더)을 읽어들여 메모리로 적재
  - 부트로더로 제어권을 넘겨줌
  - 부트로더
    - LILO, GRUB, ppcboot, uboot
  - GRUB
    - /boot/grub

리눅스 시스템 전체 부팅 과정

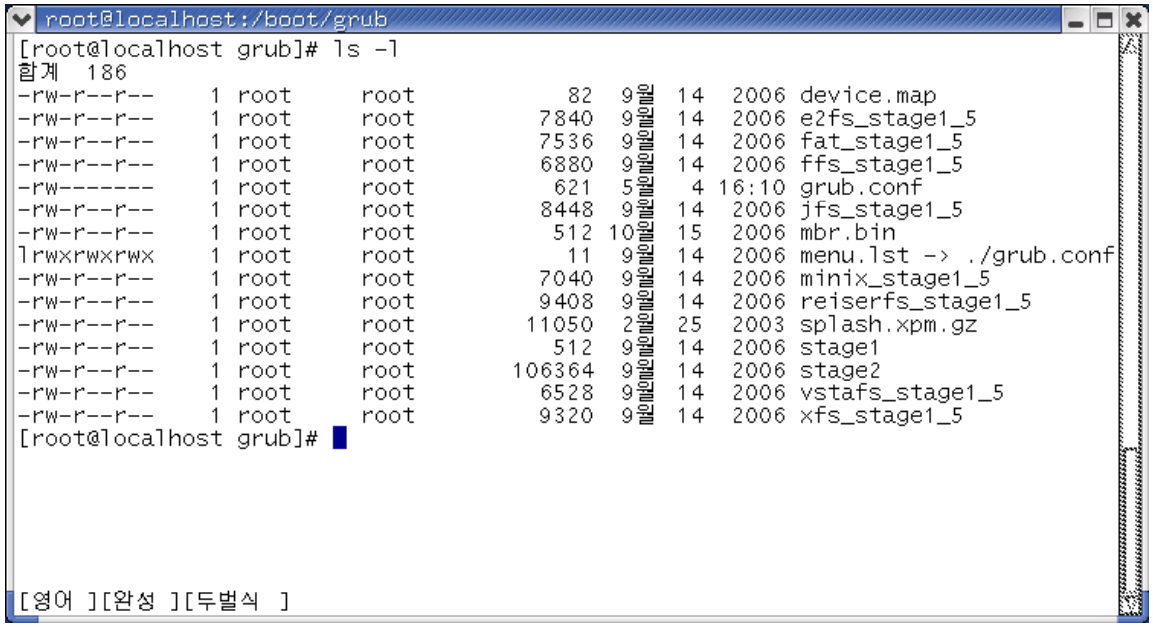
- 부팅 매체의 0번 섹터를 지칭
- grub을 설치할 때 MBR에 자신을 위한 로더를 기록해 부팅할 때 grub이 실행되도록 함
- 부트 프로그램 코드와 파티션 테이블 및 매직 넘버로 구성
- 매직 넘버 : 0xAA55
  - 실제로 이 영역이 MBR이 맞는지 확인하는 값
- 크기 : 512byte



[그림 3-2] MBR의 구조

리눅스 시스템 전체 부팅 과정

- 부트로더
  - MBR(주 부트로더)
  - Stage1cd (512byte)→ stage1.5 → stage2
- 화면에서 부팅 메뉴를 선택하는 부분은 stage2에 해당



[그림 3-3] /boot/grub 디렉토리

리눅스 시스템 전체 부팅 과정

- 커널 이미지 로딩 및 수행
  - GRUB으로 해당 커널 이미지를 로딩
  - 압축 이미지 해제
  - 커널로 제어권이 넘어옴
  - 커널이 부팅 시 관련 코드 (Redhat 2.4.20-8)
    - arch/i386/boot/bootsect.S
    - arch/i386/boot/setup.S (start())
    - arch/i386/boot/compressed/head.S (startup_32())
    - arch/i386/boot/compressed/misc.c (decompress_kernel())
    - arch/i386/kernel/head.S (startup_32())
    - init/main.c (start_kernel())
  - 커널이 부팅 시 관련 코드 (Fedora 2.6.29.4)
    - arch/x86/boot/header.S (bootsect_start)
    - arch/x86/boot/compressed/head_32.S (startup_32)
    - arch/x86/boot/compressed/misc.c (decompress_kernel())
    - arch/x86/kernel/head_32.S (startup_32)
    - init/main.c (start_kernel())

리눅스 시스템 전체 부팅 과정

- arch/x86/boot/header.S (_start 또는 bootsect_start)
  - 커널 이미지가 호출되면 _start 루틴 또는 bootsect_start 루틴 (플로피용 부트 섹터) 수행
  - 기본적인 하드웨어 설정
  - arch/x86/boot/compressed/head_32.S의 startup_32 루틴 호출

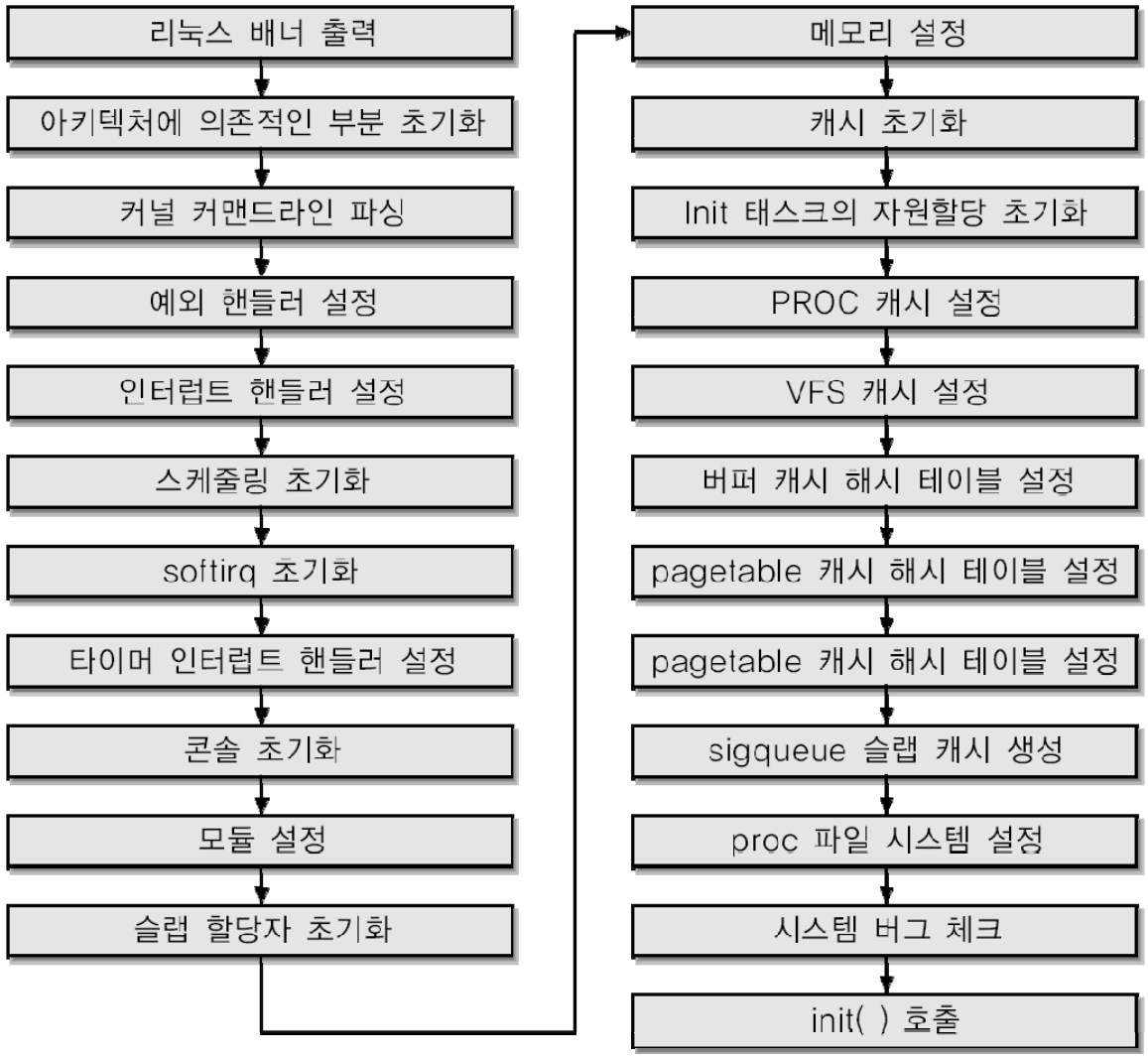
- arch/x86/boot/compressed/head_32.S (startup_32 ; 압축된 커널 해제)
  - startup_32 수행
    - ✓ 스택 설정
    - ✓ BSS(Block Started by Symbol)을 초기화
    - ✓ 147행: arch/x86/boot/compressed/misc.c 내의 389행 decompress_kernel() 함수를 호출하여 압축된 커널 해제

리눅스 시스템 전체 부팅 과정

- arch/x86/kernel/head_32.S (startup_32)
  - startup_32 수행
    - ✓압축이 해제된 커널에서의 시작점
    - ✓하드웨어 종속적인 초기화
      - » 세그먼트 값들의 설정
      - » 페이지 테이블 초기화 및 페이징 인에이블(enable)
      - » BSS (Block Started by Symbol) 섹션 초기화
      - » IDT (Interrupt Descriptor Table) 설정
      - » 부트 파라미터와 커널 커맨드 라인의 복사
      - » CPU 유형 검사 및 FPU 검사
      - » GDT (Global Descriptor Table)와 LDT (Local Descriptor Table) 로딩
      - » 447행: 커널의 메인 시작 함수인 start_kernel()을 호출

- start_kernel() 함수
  - init/main.c
  - 실질적인 리눅스 커널의 엔트리 포인트
  - 리눅스 커널의 거의 모든 초기화를 처리하는 방대한 함수
  - init 프로세스를 생성하는 init_post() 함수를 호출

리눅스 시스템 전체 부팅 과정



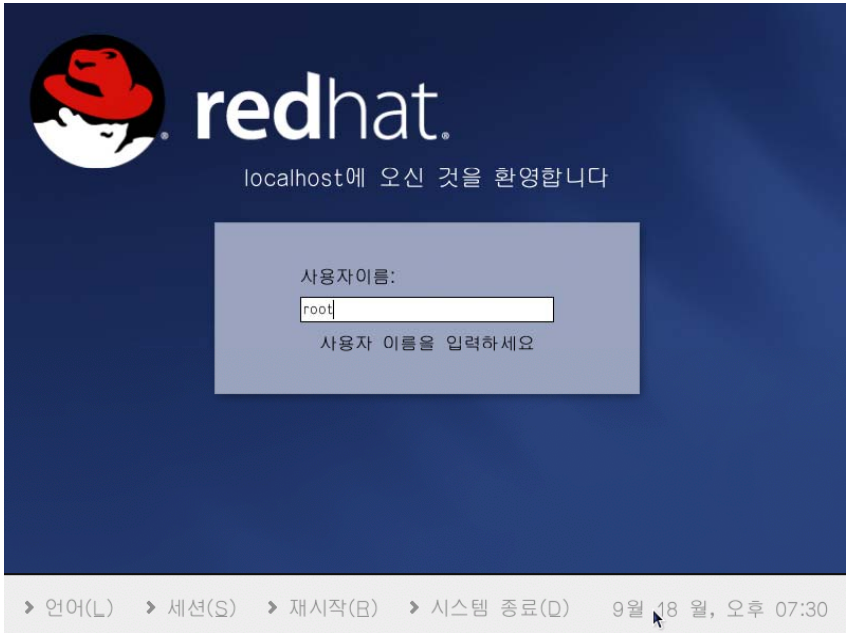
[그림 3-4] start_kernel() 함수의 수행 흐름

리눅스 시스템 전체 부팅 과정

- init 프로세스(스레드)의 생성
  - init/main.c내의 init_post() 함수
    - start_kernel( ) 함수에서 아직 처리되지 않은 나머지 부분들 초기화한 후 다음 루트 파일 시스템을 마운트
    - 초기 콘솔을 열고 루트 파일시스템에서 init 파일(/sbin/init)을 실행시킴
    - /sbin/init
      - ✓ 최초의 사용자 레벨의 프로그램
    - 만약 init을 찾지 못하면 커널은 부팅 시 패닉(panic)
    - dup 시스템 콜
      - ✓ stdin(표준입력)의 파일 디스크립터를 비어 있는 다음 파일 디스크립터에 복사하고, 파일 디스크립터 1, 2를 표준 출력과 표준 에러에 차례대로 할당

리눅스 시스템 전체 부팅 과정

- inittab 파일 수행
  - /sbin/init이 /etc/inittab 파일을 읽어 수행
    - init 프로세스가 수행해야 할 처리 방법을 기술해놓은 스크립트 파일
  - 런 레벨(run level)에 따라 필요한 기능들을 수행 후 마지막으로 getty(mingetty)를 호출하여 로그인 프롬프트를 띄우고 사용자들이 로그인할 수 있도록 함
  - 최종적으로 사용자가 로그인하면 쉘(shell)을 띄우고 사용자의 다음 명령을 기다림



[그림 3-5] 리눅스 로그인 화면

start_kernel( ) 함수

소스 코드 (Redhat 2.4.20-8)

p.91~100

```
01  asmlinkage void __init start_kernel(void)
02
03      char * command_line
04      extern char saved_command_line[];
05      lock_kernel();
06      printk(linux_banner);
07      setup_arch(&command_line);
08      printk("Kernel command line: %s \n",
        saved_command_line);
09      parse_options(command_line);
10      trap_init();
11      init_IRQ();
12      sched_init();
13      softirq_init();
14      time_init();
15      console_init();
16      init_modules();
17      kmem_cache_init();
18      sti();
19      calibrate_delay();
20      mem_init();
21      kmem_cache_sizes_init();
22      pgtable_cache_init();
23      if (num_mappedpages == 0)
24              num_mappedpages = num_physpages
25      fork_init(num_mappedpages);
26      proc_caches_init();
27      vfs_caches_init(num_physpages);
28      buffer_init(num_physpages);
29      page_cache_init(num_physpages);
30      signals_init();
31      proc_root_init();
32      check_bugs();
33      acpi_early_init(); /* before LAPIC and SMP init
        */
34      printk("POSIX conformance testing by UNIFIX
        \n");
35      smp_init();
36      rest_init();
37  }
```

start_kernel( ) 함수

소스 코드 (Fedora 2.6.29.4)

```
asmlinkage void __init start_kernel(void)
{
        char * command_line;
        extern struct kernel_param __start___param[], __stop___param[];
        smp_setup_processor_id();
        /*
         * Need to run as early as possible, to initialize the
         * lockdep hash:
         */
        lockdep_init();
        debug_objects_early_init();
        cgroup_init_early();
        local_irq_disable();
        early_boot_irqs_off();
        early_init_irq_lock_class();
/*
 * Interrupts are still disabled. Do necessary setups, then
 * enable them
 */
        lock_kernel();
        tick_init();
        boot_cpu_init();
        page_address_init();
        printk(KERN_NOTICE);
        printk(linux_banner);
        setup_arch(&command_line);
        mm_init_owner(&init_mm, &init_task);
        setup_command_line(command_line);
        setup_per_cpu_areas();
        setup_nr_cpu_ids();
        smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */
```

```
/*
 * Set up the scheduler prior starting any interrupts (such as the
 * timer interrupt). Full topology setup happens at smp_init()
 * time - but meanwhile we still have a functioning scheduler.
 */
sched_init();
/*
 * Disable preemption - early bootup scheduling is extremely
 * fragile until we cpu_idle() for the first time.
 */
preempt_disable();
build_all_zonelists();
page_alloc_init();
printk(KERN_NOTICE "Kernel command line: %s\n",
    boot_command_line);
parse_early_param();
parse_args("Booting kernel", static_command_line, __start___param,
          __stop___param - __start___param,
          &unknown_bootoption);
if (!irqs_disabled()) {
        printk(KERN_WARNING "start_kernel(): bug: interrupts were "
                        "enabled *very* early, fixing it\n");
        local_irq_disable();
}
sort_main_extable();
trap_init();
rcu_init();
```

start_kernel( ) 함수

소스 코드 (Fedora 2.6.29.4)

```
/* init some links before init_ISA_irqs() */
        early_irq_init();
        init_IRQ();
        pidhash_init();
        init_timers();
        hrtimers_init();
        softirq_init();
        timekeeping_init();
        time_init();
        sched_clock_init();
        profile_init();
        if (!irqs_disabled())
                printk(KERN_CRIT "start_kernel(): bug: interrupts were "
                                "enabled early\n");
        early_boot_irqs_on();
        local_irq_enable();

        /*
         * HACK ALERT! This is early. We're enabling the console before
         * we've done PCI setups etc, and console_init() must be aware of
         * this. But we do want output early, in case something goes wrong.
         */
        console_init();
        if (panic_later)
                panic(panic_later, panic_param);

        lockdep_info();
```

```
        /*
         * Need to run this when irqs are enabled, because it wants
         * to self-test [hard/soft]-irqs on/off lock inversion bugs
         * too:
         */
        locking_selftest();

#ifdef CONFIG_BLK_DEV_INITRD
        if (initrd_start && !initrd_below_start_ok &&
            page_to_pfn(virt_to_page((void *)initrd_start)) < min_low_pfn) {
                printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
                    "disabling it.\n",
                    page_to_pfn(virt_to_page((void *)initrd_start)),
                    min_low_pfn);
                initrd_start = 0;
        }
#endif
        vmalloc_init();
        vfs_caches_init_early();
        cpuset_init_early();
        page_cgroup_init();
        mem_init();
        enable_debug_pagealloc();
        cpu_hotplug_init();
        kmem_cache_init();
        debug_objects_mem_init();
        idr_init_cache();
        setup_per_cpu_pageset();
        numa_policy_init();
```

start_kernel( ) 함수

소스 코드 (Fedora 2.6.29.4)

```
if (late_time_init)
        late_time_init();
    calibrate_delay();
    pidmap_init();
    pgtable_cache_init();
    prio_tree_init();
    anon_vma_init();
#ifdef CONFIG_X86
    if (efi_enabled)
        efi_enter_virtual_mode();
#endif
    thread_info_cache_init();
    cred_init();
    fork_init(num_physpages);
    proc_caches_init();
    buffer_init();
    key_init();
    security_init();
    vfs_caches_init(num_physpages);
    radix_tree_init();
    signals_init();
```

```
/* rootfs populating might need page-writeback */
    page_writeback_init();
#ifdef CONFIG_PROC_FS
    proc_root_init();
#endif
    cgroup_init();
    cpuset_init();
    taskstats_init_early();
    delayacct_init();

    check_bugs();

    acpi_early_init(); /* before LAPIC and SMP init */

    ftrace_init();

    /* Do the rest non-__init'ed, we're now alive */
    rest_init();
}
```

start_kernel( ) 함수

- init/main.c
  - 리눅스 커널의 거의 모든 초기화 부분을 처리하는 방대한 함수
  - 시스템 전체에 사용될 하드웨어뿐만 아니라 각각의 서브 시스템에서 사용되는 자원들을 초기화하는 루틴을 수행

- 코드 내용
  - 530행: asmlinkage void __init start_kernel(void)
    - __init : Include/linux/init.h 참조
  - 553행: lock_kernel(); 리눅스가 SMP 환경에서 실행되는 경우를 위해 먼저 락(lock)을 설정
    - include/linux/smplock.h
  - 558행: printk(linux_banner); linux_banner 출력
    - init/version.c
    - /var/log/dmesg에 기록
    - #cat /proc/version

start_kernel( ) 함수

- 559행 : setup_arch( );
  - 부트 커맨드라인을 인수로 받아 아키텍처에 의존적인 초기화 루틴 수행
    - ✓시스템의 여러 정보들을 수집
    - ✓메모리 레이아웃을 분석 및 설정
    - ✓페이징 시스템 초기화
    - ✓전원 관리 등
  - arch/"CPU이름"/kernel/setup.c → arch/x86/kernel/setup.c
- 590행: trap_init( ); 트랩(예외) 핸들러를 설정
  - 시스템 콜 소프트웨어 인터럽트를 위한 핸들러도 설정
  - 리눅스에서의 예외 처리는 자체 인터럽트 디스크립터 사용
  - 0x00 ~ 0x19
  - 0x80 : 시스템 콜
  - arch/x86/kernel/traps.c 953행

start_kernel( ) 함수

- 571행 : sched_init( ) 함수
  - 프로세스 스케줄링을 위한 초기화 작업을 수행
  - kernel/sched.c
  - init 태스크가 사용하는 CPU 번호 할당
  - pid 해시 테이블(hash table)을 초기화
  - 시스템의 타이머를 초기화
  - "bottom-half" 큐들을 초기화
  - lazy TLB(translation lookaside buffer) 생성

- 594행 : init_IRQ( )
  - arch/x86/kernel/i8259.c
  - 인터럽트 핸들러들을 설정
  - init_ISA_irqs() 함수를 호출하여 두 개의 8259A 인터럽트 컨트롤러를 초기화, ISA IRQ들을 위한 기본 인터럽트 핸들러들을 설정
  - 인터럽트 벡터들을 위한 인터럽트 게이트들을 세팅

start_kernel( ) 함수

- 598행 : softirq_init( )
  - kernel/softirq.c
  - tasklet, softirq을 초기화
    - ✓ 리눅스에서는 여러 개의 CPU에서 bottom-half를 수행할 수 있도록 softirq와 tasklet을 사용
- 600행 : time_init( ) 함수
  - arch/i386/kernel/time.c
  - CMOS에서 시스템의 현재 시간을 읽어 옴
  - IRQ0 타이머 틱 인터럽트 핸들러를 설정
  - 커널의 3가지 시간을 측정 방법
    - ✓ PIT 타이머 이용
    - ✓ RTC(Real Time Clock) 이용
    - ✓ TSC(Time Stamp Counter) 이용
- 614행 : console_init( )
  - drivers/char/tty_io.c
  - 콘솔 초기화

start_kernel( ) 함수

- ??? 행 : module_init( ) 함수
  - kernel/module.c
  - 커널 모듈을 설정
  - 커널 심볼 테이블의 크기(수) 초기화

- 644행 : kmem_cache_init( ) 함수
  - mm/slab.c
  - 슬랩 할당자(slab allocator)를 초기화
    - ✓ 페이지 단위로 메모리를 할당받아 생성되는 커널 오브젝트들의 크기에 맞게 메모리를 할당해주는 알고리즘

- ??? 행 : sti( ) 함수
  - 인터럽트 가능 상태로 설정

- 651행 : calibrate_delay( ) 함수
  - "loops_per_jiffy" 지연 루프 값을 계산
  - BogoMIPS 출력

start_kernel( ) 함수

- 641행 : mem_init( ) 함수
  - arch/x86/mm/init.c
  - empty_zero_page를 초기화
  - 메모리 영역을 해제(free_pages_init())
  - 예약된 RAM 페이지 수를 카운트
  - 시스템의 메모리 크기, 커널 코드 크기, 예약된 메모리 크기, 커널 데이터 크기, init의 크기, High 메모리의 크기들의 정보를 출력
- 644행 : kmem_cache_sizes_init( ) 함수
  - mm/slab.c
  - 일반적인 캐시들을 초기화
- 653행 : pgtable_cache_init( ) 함수
  - arch/x86/mm/init.c
  - 페이지 테이블 캐시를 초기화

start_kernel( ) 함수

- 662행 : fork_init() 함수
  - kernel/fork.c
  - 최대 스레드의 수를 메모리 크기에 따라 결정
  - init 태스크의 자원 할당과 관련된 부분을 초기화
- 663행 : proc_caches_init( ) 함수
  - kernel/fork.c
  - signal_act, files_cache, fs_cache, vm_area_struct, mm_struct에 대한 slab 캐시들을 생성
- 667행 : vfs_caches_init( ) 함수
  - fs/dcache.c
  - bh_cache, names_cache, filp_cache, dquot에 대한 slab 캐시를 생성
  - dentry_cache와 dentry_hashtable을 생성

start_kernel( ) 함수

- 664행 : buffer_init( ) 함수
  - fs/buffer.c
  - 버퍼 캐시 해시 테이블을 할당하고 초기화
    - ✓ buffer_head 구조체가 초기화
    - ✓ 해시 체인과 LRU 리스트를 셋업
- 669행 : signals_init( ) 함수
  - kernel/signal.c
  - 시그널 처리와 관련된 sigqueue 슬랩 캐시를 생성
- 673행 : proc_root_init( ) 함수
  - fs/proc/root.c
  - /proc 파일시스템 설정
  - /proc/net, /proc/fs, /proc/driver 등

start_kernel( ) 함수

- 680행 : check_bugs() 함수
  - linux/include/asm-i386/bugs.h
  - 커널 설정 체크, FPU 체크, Halt 체크 등 시스템의 문제점을 체크
- 682행 : acpi_early_init() 함수
  - 디바이스들이 예약된 I/O 포트를 사용할 수 있도록 설정
  - 커맨드라인에 acpi=off가 설정된 경우에는 skip
- 856행 : smp_init() 함수
  - init/main.c
  - 다른 SMP 프로세서들을 시작시킴
- 687행 : rest_init( ) 함수
  - init 프로세스를 생성
  - 커널의 락을 해제
  - 대기(idle) 상태로 들어감(CPU_idle() 함수, idle process, pid 0)

inittab 파일

/etc/inittab 파일

- 시스템의 상태에 따라 해당 런 레벨(run level)에서 init 프로세스가 수행해야 할 일들을 기술해 놓은 파일

[표 3-1] 런레벨 종류

런레벨	기능
0	시스템 종료
1	싱글 유저 모드
2	NFS를 제외한 다중 사용자 모드 만약 네트워크를 사용하지 않으면 런 레벨 3과 동일
3	풀(Full) 다중 사용자 모드
4	사용되지 않음(unused)
5	X 윈도우
6	시스템 재부팅

inittab 파일

소스 코드 (Redhat 2.4.20-8)

p.107~109

```
01
02    # inittab   This file describes how the
      INIT process should set up
03    #  the system in a certain run-level.
04
05  # Author:  Miquel van Smoorenburg,
      <miquels@drinkel.nl.mugnet.org>
06  # Modified for RHS Linux by Marc Ewing
      and Donnie Barnes
07
08
09  # Default runlevel. The runlevels used by
      RHS are:
10  #   0 - halt (Do NOT set initdefault to this)
11  #   1 - Single user mode
12  #   2 - Multiuser, without NFS (The same as
      3, if you do not have networking)
13  #   3 - Full multiuser mode
14  #   4 - unused
15  #   5 - X11
16  #   6 - reboot (Do NOT set initdefault to this)
17
18  id:5:initdefault:
19
20  # System initialization.
21  si::sysinit:/etc/rc.d/rc.sysinit
22
23  l0:0:wait:/etc/rc.d/rc 0
24  l1:1:wait:/etc/rc.d/rc 1
25  l2:2:wait:/etc/rc.d/rc 2
26  l3:3:wait:/etc/rc.d/rc 3
27  l4:4:wait:/etc/rc.d/rc 4
28  l5:5:wait:/etc/rc.d/rc 5
29  l6:6:wait:/etc/rc.d/rc 6
30
31  # Trap CTRL-ALT-DELETE
32  ca::ctrlaltdel:/sbin/shutdown -t3 -r now
33
34  # When our UPS tells us power has failed,
      assume we have a few minutes
```

inittab 파일

소스 코드 (Redhat 2.4.20-8)

p.107~109

```
35  # of power left.  Schedule a shutdown for 2
      minutes from now.
36  # This does, of course, assume you have
      powerd installed and your
37  # UPS connected and working correctly.
38  pf::powerfail:/sbin/shutdown -f -h +2 "Power
      Failure; System Shutting Down"
39
40  # If power was restored before the shutdown
      kicked in, cancel it.
41  pr:12345:powerokwait:/sbin/shutdown -c
      "Power Restored; Shutdown Cancelled"
42
43  # Run gettys in standard runlevels
44  1:2345:respawn:/sbin/mingetty tty1
45  2:2345:respawn:/sbin/mingetty tty2
46  3:2345:respawn:/sbin/mingetty tty3
47  4:2345:respawn:/sbin/mingetty tty4
48  5:2345:respawn:/sbin/mingetty tty5
49  6:2345:respawn:/sbin/mingetty tty6
50
51  # Run xdm in runlevel 5
52  x:5:respawn:/etc/X11/prefdm -nodaemon
```

inittab 파일

소스 코드 (Fedora 2.6.29.4)

```
# inittab is only used by upstart for the default runlevel.
#
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# System initialization is started by /etc/event.d/rcS
#
# Individual runlevels are started by /etc/event.d/rc[0-6]
#
# Ctrl-Alt-Delete is handled by /etc/event.d/control-alt-delete
#
# Terminal gettys (tty[1-6]) are handled by /etc/event.d/tty[1-6] and
# /etc/event.d/serial
#
# For information on how to write upstart event handlers, or how
# upstart works, see init(8), initctl(8), and events(5).
#
# Default runlevel. The runlevels used 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:5:initdefault:
```

inittab 파일

각 엔트리들의 구성

identifier: run-levels: action: process

❶ ❷ ❸ ❹

❶ identifier : 각 엔트리를 구분하기 위한 식별자
(1개 문자 또는 1개 이상의 문자열)

❷ run-levels : 각 엔트리가 수행되어지는 런 레벨 목록

❸ action : process 필드에 기술된 프로세스를 어떻게 처리할 지의 방식

❹ process : 실행 시키고자 하는 프로세스 및 인자

inittab 파일

action의 종류

- initdefault : 디폴트 런 레벨을 지정
  - ✓ init 프로세스가 실행될 때 가장 먼저 실행되며 소스에서는 런 레벨 5를 디폴트 런 레벨로 지정한다는 의미
- sysinit : 시스템 부트 시에 실행되며 런 레벨 필드는 무시
  - ✓ boot 또는 bootwait 엔트리들이 수행되기 전에 실행됨
- wait : 프로세스를 실행한 후 init 프로세스가 실행한 프로세스가 종료하기까지 기다리라는 의미
- once : 프로세스를 실행한 후 wait처럼 기다리지 않도록 지정
  - ✓ 만약 프로세스가 죽더라도 재실행하지 않음
- ctrlaltdel : init 프로세스가 SIGINT 시그널을 받으면 프로세스 실행
  - ✓ 시스템 콘솔 상에서 CTRL+ALT+DEL 키를 누르면 실행된다.
- powerwait : 전원에 문제가 있는 경우 실행
  - ✓ init은 프로세스가 종료될 때까지 기다림

inittab 파일

- powerokwait : 전원이 회복되면 실행됨
- powerfail : powerwait 항목과 다른 것은 init가 프로세스의 종료를 기다리지 않는다는 점
- respawn : 실행한 프로세스가 죽게 되면 다시 실행하도록 지정
  - ✓ 예 : getty 프로세스의 실행에 적용
- kbrequest : 키보드 핸들러로부터 init이 시그널을 받으면 프로세스 실행
- off : 프로세스가 현재 실행 중이 아니면 엔트리는 무시
  - ✓ 만약 이 엔트리와 연관된 프로세스가 현재 실행 중이면 경고 시그널을 보내고 종료되기 전에 20초 동안 대기
- boot : 시스템 부팅 동안 프로세스가 실행. 런 레벨 필드는 무시
- bootwait : boot와 다른 점은 프로세스가 종료할 때까지 기다린다는 점

inittab 파일

- /etc/event.d/rcS 16행 : /etc/rc.d/rc.sysinit
  - 가장 먼저 실행되는 스크립트로서, 시스템을 초기화하는 스크립트
  - 부팅할 때 화면에 나타나는 시스템의 초기화와 관련된 거의 모든 메시지들이 이 파일의 실행 과정에서 나타나는 메시지
  - 런 레벨과 무관하게 부팅 시 한번만 실행
    - ✓ 폰트 설정
    - ✓ 네트워크 설정 적용
    - ✓ proc 파일시스템 마운트
    - ✓ 커널 파라미터 구성
    - ✓ 시스템 클록 세팅
    - ✓ 호스트 이름 설정
    - ✓ USB 콘트롤러, HID 디바이스 초기화
    - ✓ 루트 파일 시스템 재마운트(Read-Write 모드)
    - ✓ LVM 초기화
    - ✓ 스왑 파티션 활성화
    - ✓ 모듈 적재
    - ✓ 파일 시스템 점검
    - ✓ 모든 다른 파일 시스템 마운트
    - ✓ 시리얼 포트 초기화
    - ✓ 기타 시스템 초기화

inittab 파일

- /etc/event.d/rc 5 19행 : exec /etc/rc.d/rc 5
  - 결정된 런 레벨에 따라 조건에 맞는 엔트리를 실행
    - ✓ 런 레벨이 5면 /etc/rc.d/rc5.d의 내용들을 순차적으로 실행
    - ✓ 런 레벨이 0이면 /etc/rc.d/rc0.d의 파일들이 순차대로 실행됨
    - ✓ 런 레벨이 6이면 /etc/rc.d/rc6.d의 파일들이 순차대로 실행됨
  - 각 디렉토리 내에는 각 런 레벨에 따라 죽여야 할 프로세스들과 실행시켜야할 프로세스들이 명시
    - ✓ K와 S로 시작하는 파일들이 링크 파일 형태로 존재
      - » 파일명이 K로 시작하면 죽여야 할 프로세스, S로 시작하면 실행해야 할 프로세스를 의미
      - » K나 S 뒤에 명시되어 있는 두 자리 숫자는 실행 순서