Embedded Systems

Ch 14B
Linux Kernel

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- *1. Bootloader*
- *2. EzBoot*
- *3. Boot Sequence*
- *4. Linux Boot Process*
- *5. Linux Kernel*

4. Linux Boot Process

- **The chain of events at boot in PC**
  - CPU -> VGA -> Power-On-Self-Test ->
  - SCSI -> Boot Manager -> Lilo boot loader ->
  - kernel-> init-> bash.
  - The firmware and software programs output *various messages* as the computer and Linux come to life.

Linux Boot Process (II)

- **Linux boot procedure (PC with Disk)**
  - 1. The Motherboard BIOS Triggers the Video Display Card BIOS Initialization
  - 2. Motherboard BIOS Initializes Itself
  - 3. SCSI Controller BIOS Initializes
  - 4. Hardware Summary:
    - The motherboard BIOS then displays the summary of its hardware inventory. And runs its Virus checking code that looks for changed boot sectors.
  - 5. BootManager Menu:
    - The Master Boot Record (MBR) on the first hard disk is read, by DOS tradition, into address 0x00007c00, and the processor starts executing instructions there.
    - This MBR boot code loads the first sector of code on the active DOS partition.

Linux Boot Process (III)

- **Linux boot procedure (II)**
  - 6. Lilo is started:
    - If the Linux selection is chosen and if Linux has been installed with Lilo, Lilo is loaded into address 0x00007c00.
    - Lilo prints LILO with its progress revealed by individually printing the letters.
      - The first "L" is printed after Lilo moves itself to a better location at 0x0009A000.
      - The "I" is printed just before it starts its secondary boot loader code.
      - Lilo's secondary boot loader prints the next "L", loads descriptors pointing to parts of the kernel, and then prints the final "O".
    - The descriptors are placed at 0x0009d200.
    - The boot message and a prompt line, if specified, are printed.
    - The pressing "Tab" at the prompt, allows the user to specify a system and to provide command-line specifications to the Linux Kernel, its drivers, and the "init" program. Also, environment variables may be defined at this point.

Linux Boot Process (IV)

- **Linux boot procedure (III)**
  - 7. The kernel code in /linux/arch/i386/boot/setup.S arranges the transition from the processor running in real mode (DOS mode) to protected mode (full 32-bit mode).
    - Blocks of code named Trampoline.S and Trampoline32.S help with the transition.
    - Small kernel images (zImage) are decompressed and loaded at 0x00010000.
    - Large kernel images (bzImage) are loaded instead at 0x00100000.
    - This code sets up the registers, decompresses the compressed kernel (which has linux/arch/i386/head.S at its start), printing the following 2 lines from linux/arch/i386/boot/compressed/misc.c
      - Uncompressing Linux… Ok.
      - Booting the kernel.
    - The i386-specific setup.S code has now completed its job and it jumps to 0x00010000 (or 0x00100000) to start the generic Linux kernel code.

Linux Boot Process (V)

- **Linux boot procedure (IV)**
  - 8. Generic Linux kernel code
    - Processor, Console, and Memory Initialization:
      - This runs linux/arch/i386/head.S which in turn jumps to start_kernel(void) in linux/init/main.c where the interrupts are redefined.
      - Linux/kernel/module.c then loads the drivers for the console and pci bus.
      - From this point on the kernel messages are also saved in memory and available using /bin/dmesg.
      - They are then usually transferred to /var/log/message for a permanent record.
    - PCI Bus Initialization:
      - mpci_init() in linux/init/main.c causes lines from linux/arch/i386/kernel/bios32.c to be printed.
    - Network Initialization:
      - socket_init() in linux/init/main.c causes network initializations.

Linux Boot Process (VI)

- **Linux boot procedure (V)**
  - 8B. Generic kernel code (cont'd)
    - The Kernel Idle Thread (Process 0) is Started : At this point a kernel thread is started running init() which is one of the routines defined in linux/init/main.c.
      - This init() must not be confused with the program /sbin/init that will be run after the Linux kernel is up and running.
      - mkswapd_setup() in linux/init/main.c causes the following line from linux/mm/vmscan.c to be printed:
        - Starting kswapd v 1.5

Linux Boot Process (VII)

- **Linux boot procedure (VI)**
  - 8C. Generic kernel code (Cont'd)
    - Device Driver Initialization : The kernel routine linux/arch/i386/kernel/setup.c then initializes devices and file systems. It produces the following lines and then forks to run /sbin/init:
      - Generic Parallel Port Initialization: The parallel port initialization routine linux/drivers/misc/parport_pc.c prints.
      - Character Device Initializations: from linux/drivers/char/serial.c:
      - Block Device Initializations : linux/drivers/block/rd.c prints:
        - RAM disk driver initialized: 16 RAM disks of 8192K size
      - linux/drivers/block/loop.c prints:
        - loop: registered device at major 7
      - linux/drivers/block/floppy.c prints:
        - Floppy drive(s): fd0 is 1.44M, fd1 is 1.44M FDC 0 is a post-1991 82077
      - SCSI Bus Initialization: aic7xxx.c, scsi.c, sg.c, sd.c or sr.c in the subdirectory linux/drivers/scsi.

Linux Boot Process (VIII)

- **Linux boot procedure (VII)**
  - 8D. Generic kernel code (Cont'd)
    - Initialization of Kernel Support for Point-to-Point Protocol : The initialization is done by linux/drivers/net/ppp.c.
    - Examination of Fixed Disk Arrangement : from linux/drivers/block/genhd.c:
  - 9. Init Program (Process 1) Startup:
    - The program /sbin/init is started by the "idle" process (Process 0) code in linux/init/main.c and becomes process 1.
    - /sbin/init then completes the initialization by running scripts and forking additional processes as specified in /etc/inittab.
    - It starts by printing: INIT: version 2.76 booting and reads /etc/inittab.
  - 10. The Bash Shell is Started:
    - The bash shell, /bin/bash is then started up. Bash initialization begins by executing script in /etc/profile which set the system-wide environment variables. *Login:*

5. Linux Kernel

- 주요 특징
  - Multi-tasking (다중작업)
    - Preemptive, mutually independent
  - Multi-user access (다중 사용자 접근)
  - Multi-tasking (다중 처리)
    - Multi-task time sharing
    - Distribution to multiple processors possible
  - Architecture independence (구조 독립성)
    - Pc, Amiga, DEC Alpha, Sparc, Power PC, ARM, ...
  - Demand load executables (요구 적재 실행 가능성)
    - Loaded into memory only when required. Copy-on-write.
  - Paging (페이징)
    - Memory full: disk swap in 4K bytes unit (not a whole process).
  - Dynamic cache for hard disk
    - 동적으로 사용중인 disk cache memory 의 크기를 조정 가능.

Linux Kernel (II)

- **주요 특징 (cont'd)**
  - Shared Libraries (**공유된** library)
    - **여러 프로세스에서 요구하는** Library code **들을 한번만 적재하여 공유**
  - POSIX 1003.1 standard, System V, BSD support
    - POSIX 1003.1: Unix **형태의 운영체제 최소의** interface
    - System V, BSD **를 위한 부가적인** system interface
  - **실행가능한 파일들에 대한 다양한 형식들**
    - MS-DOS, Windows emulator
  - Memory protection mode
    - Access protection to other processes and system kernel
  - Internationalization
    - Character sets and keyboard drivers for various countries
  - **여러** file system **지원**
    - Ext2, VFAT, ISO, NFS
    - AFF for Amiga, UPS, SysV, HPFS for OS/2, Sambe, Windows NT
  - TCP/IP, SLIP, PPP **지원**.

Linux Kernel (III)

- **Kernel Architecture**
  - **Microkernel**
    - Windows NT, Minix, Hurd
    - 실제 kernel은 inter-process communication, memory 관리 등의 최소의 기능만을 가지며 작고 compact 하다.
    - OS의 기타 기능들은 microkernel과 통신에 의하여 정보를 교환한다.
    - 장점:
      - 유지, 관리 문제가 덜 발생한다.
      - 각 구성요소들은 독립적으로 작동하며 교체가 용이하다
      - 새로운 구성요소의 개발이 간단하다.
    - 단점
      - Overall optimization이 어렵다.
      - IPC가 광범위하다.
  - **Single kernel**
    - Linux (But modular construction)
    - 느린 processor에서도 동작한다 (i386)
    - Run-time 최적화

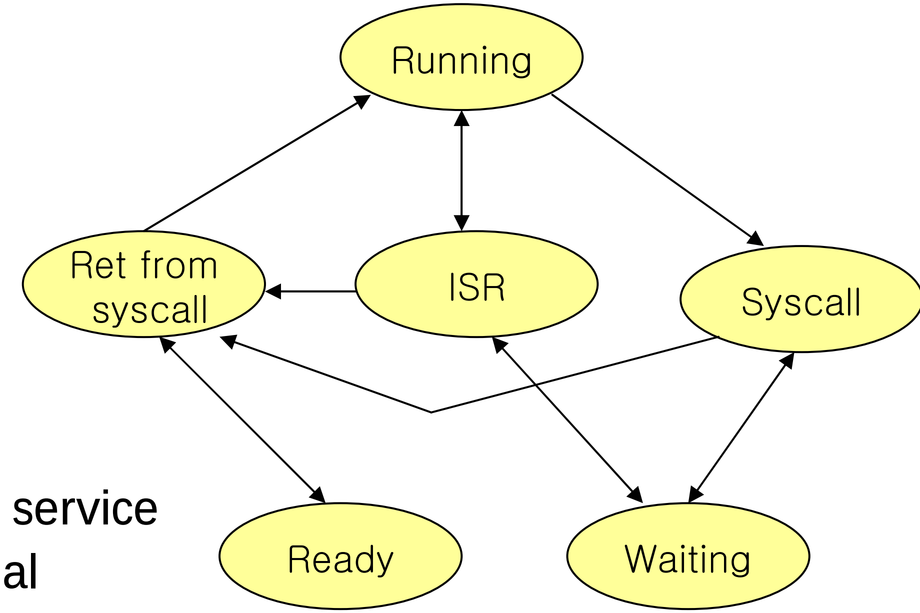
Linux Kernel (IV)

- **Linux kernel 2.0 for Intel architecture**
  - 470,000 lines of C code
    - 165,000 lines for 1.0
    - 5% for Kernel (process and memory management)
  - 8,000 lines of Assembly code
  - Components

Item	C code	ASM lines
Device driver	377,000	100
Network	25,000	
VFS	13,500	
File systems	50,000	
Initialization	4,000	2,800
Math Coprocessor		3,550
Miscellaneous	20,000	

Linux Kernel (V)

- **Linux Process Status**
  - Running
    - In user mode
  - System call
    - Via software interrupt
    - Can wait for a specific event
  - Return from system call
    - After system call or interrupt service
    - Check for scheduler and signal
  - Interrupt routine
    - Generated by hardware
  - Waiting
    - Wait for an external event
  - Ready
    - Compete with other process to obtain the processor.



Linux Kernel (VI)

- **Data Structure**
  - Task structure: struct task_struct {...} for each task
    - Volatile long state; TASK_RUNNING, TASK_STOPPED, ...
    - Long counter; Process tick. Sub-priority
    - Long priority; Process priority
    - Unsigned long signal; Bit mask for signal reception
    - Unsigned long blocked; Another bit mask for other signals
    - Unsigned long flags; System status flag. PF_PTRACRED etc.
    - Int errno; Error code for the last system call
    - Int debugreg[8]; x86 debug registers for ptrace
    - Struct exec_domain exec_domain;
      각 프로세스들이 emulate **되어야 하는** unix**에 대한 기술 정보**.
    - Struct task_struct *next_task;
    - Struct task_struct *prev_task; Double linked list.
    - …… Parent & child, memory management
    - Int pid, pg**게**, session, leader; Process id, group, session, leader
    - …… File, timing, semaphore, wait

Linux Kernel (VII)

Data Structure (II)

- Process table: struct task_struct *task[NR_TASKS];
  - Struct task_struct init_task; Start task for double linked list
  - Struct task_struct current; Current task
  - Task_struct *current_set[NR_CPUS] SMP

- File structure: Struct file { ... }
  - Mode_t f_mode; Access mode: R, W, RW
  - Loff_t f_pos; Read/write pointer (64-bits)
  - Unsigned short f_flags; File access control
  - Unsigned short f_count; Reference counter
  - Struct *file *f_next, *f_prev; Double linked list
  - Struct inode *f_inode; Inode structure
  - Struct file_operations *f_op; File operations table pointer
  - ...

Linux Kernel (VIII)

- **Data Structure (III)**
  - Queue — Wait
    - Struct wait_queue {
      - Struct task_struct *task;
      - Struct wait_queue *next;
    - }
  - Semaphore — **접근 허가**
    - Struct semaphore {
      - Int count;
      - Struct wait_queue *wait;
    - }
  - Timer — Timed action
    - Struct timer_list {
      - Struct timer_list *nest, *prev;
      - Unsigned long expires;
      - Unsigned long data;
      - Void (*function) (unsigned long);
    - }
  - ...

Linux Kernel (IX)

- **Scheduler**
  - Scheduler classes   Set by sched_setscheduler()
    - SCHED_FIFO
      - First-In First_Out
      - Run from start to finish
    - SCHED_RR
      - Round robin
      - Run during a specified time slot
    - SCHED_OTHER
      - Classic Unix scheduling
  - Schedule() **함수**   kernel/sched.c
    - **정기적으로 호출되어야 하는** routine (Timer interrupt)
    - **높은 우선권의** process **결정**
    - **새로운 최우선권** process**로 이양**.

Linux Kernel (X)

- **System Call Mechanism**
  - User mode to system mode
  - Via software interrupt 0x80

  - Pseudo code system_call(int sys_call_num, sys_call_args) {
    - SAVE_ALL;   // Macro in entry.S
    - If (sys_call_num >= NR_syscalls)
      - errno = -ENOSYS;
    - Else {
      - If (current->flags & PT_TRACESYS) {
        - Syscall_trace;
        - Errno = (*sys_call_table[sys_call_num])(sys_call_args);
        - Syscall trace;
      - } else
        - Errno = (*sys_call_table[sys_call_num])(sys_call_args);
      - }

Linux Kernel (XI)

- **System Call Mechanism (II)**
  - Pseudo code for return form system call
    - If (need_resched) {
      - Reschedule;
      - scehdule();
      - Goto ret_from _sys_call;
    - }
    - If (current->singal & ~current->blocked) {
      - Signal_return;
      - Do_signal();
    - }
    - Exit_now:
      - RESTORE_ALL;

Linux Kernel (XII)

- **System Call Examples**
  - Getpid
    - Asmlikage int sys_getpid(void)
      - {
      -     Return current->pid;
      - }

  - Pause
    - Asmlinkage int sys_pause(void)
      - {
      -     current->state = TASK_INTERRUPTIBLE;
      -     schedule();
      -     return –ERESTARTNOHAND;
      - }

Linux Kernel (XIII)

- **Memory Manager**
  - Virtual memory management
    - Memory page
      - 4 Kbytes/page for PC, 8 Kbytes for Alpha
    - Linear memory mapping
      - Linear address =
        - Page directory +
        - Page middle directory +
        - Page table +
        - Offset
  - Dynamic memory allocation in kernel
    - Void *kmalloc(size_t size, int priority);
    - Void kfree(void *obj);

Linux Kernel (XIV)

- **Inter-Process Communication (IPC)**

	Kernel	Process	Network
Resource sharing	Data structure, buffer	Shared memory, file, mmap	-
Synchronization	Wait queue, semaphore	SysV semaphore, file locking	-
Connectionless data exchange	Signal	SysV message, Unix domain sockets in datagram mode	Datagram sockets (UDP)
Connection-oriented data exchange	-	Pipes, Named pipes, Unix domain sockets in stream mode	Stream sockets (TCP)

References

- Linux Boot Process
  - Search Internet
- Linux Kernel
  - R. Magnus, et al., “Linux Kernel Internals”, 1999, Addison Wesley