

Ch 2. Exploration Toolkit

Oct. 26, 2006

Hyun-koo Jee

Senior researcher, DWE

Contents

- 2.1. Common Kernel Datatypes
- 2.2. Assembly
- 2.3. Assembly Language Example
- 2.4. Inline Assembly
- 2.5. Quirky C Language Usage
- 2.6. A Quick Tour of Kernel Exploration Tools
- 2.7. Kernel Speak: Listening to Kernel Messages
- 2.8. Miscellaneous Quirks

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

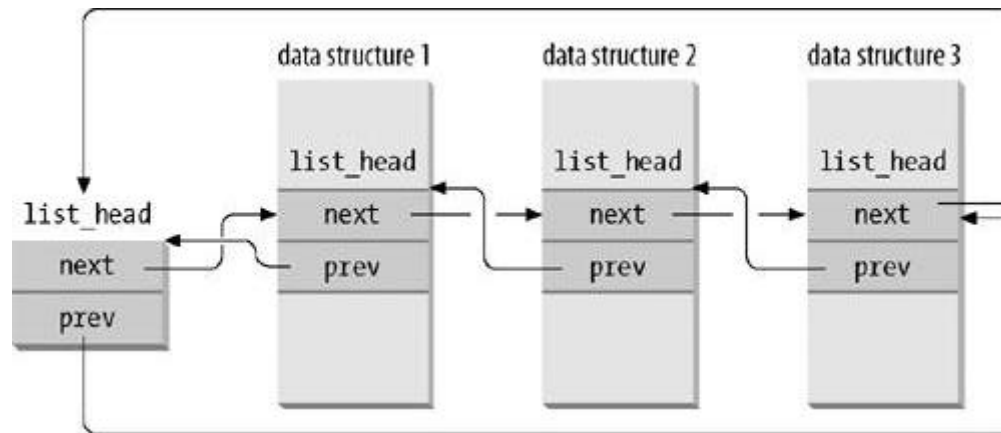
2.8. Miscellaneous Quirks

2.1. Common Kernel Datatypes

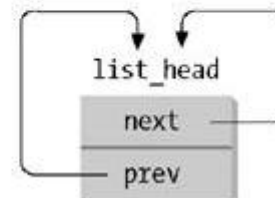
- 2.1.1. Linked Lists
- 2.1.2. Searching
- 2.1.3. Trees

2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
 - Structure



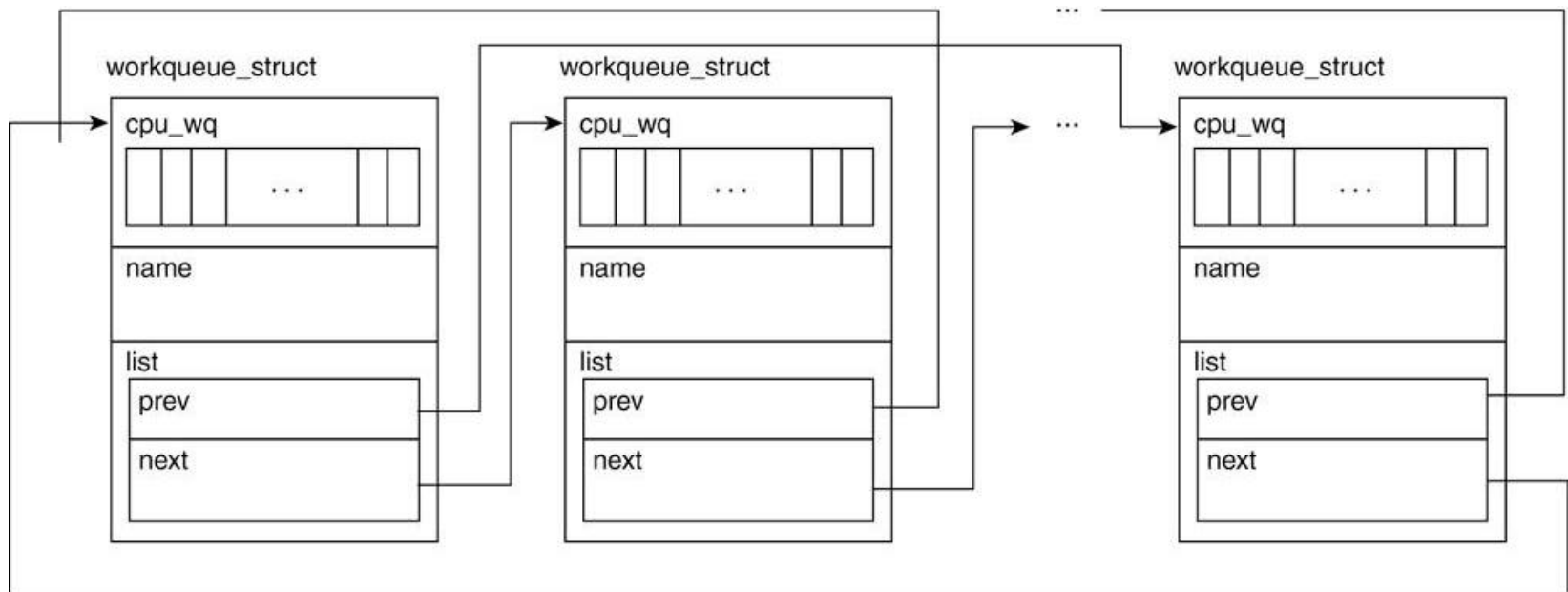
(a) a doubly linked list with three elements



(b) an empty doubly linked list

2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
 - Structure



???

2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists

- Data structure

- struct list_head (in *include/linux/list.h*)

- List handling functions & macros (in *include/linux/list.h*)

- LIST_HEAD(name): list_head
 - LIST_HEAD_INIT(name): list_head
 - INIT_LIST_HEAD(ptr): ptr 가 list_head

- `list_add(n, p)`: `p`가 `n`가
(`p`가 `list head`, `list`)
- `list_add_tail(n, p)`: `p`가 `n`가
(`p`가 `list head`, `list`)
- `list_del(p)`: `p`가
- `list_empty(p)`: `p` `list head`
- `list_entry(p, t, m)`: `t`가 `m` (`list_entry(p, t, m)`가
.`t` `m` `p`가 ,
`list_entry(p, t, m)` `t` `p`가 .
- `list_for_each(p, h)`: `p` iterator `h` `list head`
- `list_for_each_entry(p, h, m)`: `p`가 `list_head`가
`list_head`가 .

* See 'Understanding the Linux Kernel', 3rd ed.,
Table 3-1 in Section 3.2

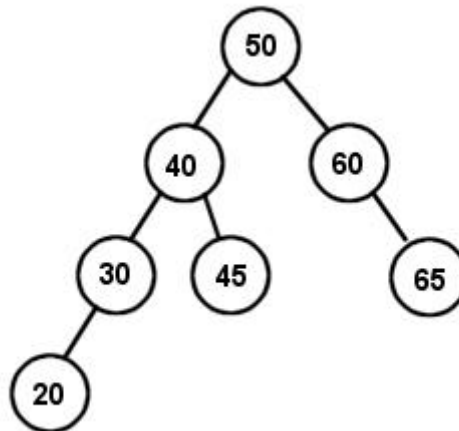

```
1  #include <linux/list.h>
2
3  typedef struct mystruct {
4      char name[20];
5      struct list_head list;
6  };
7
8  myfunc()
9  {
10     LIST_HEAD(mylist);
11     struct mystruct *p;
12     ...
13     list_add(&newnode, &mylist);
14     ...
15     for_each_list_entry(p, &mylist, list) {
16         printk("%s\n", p->name);
17     }
18 }
```

2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
- 2.1.2. Searching
 - Linear search: $O(n/2)$

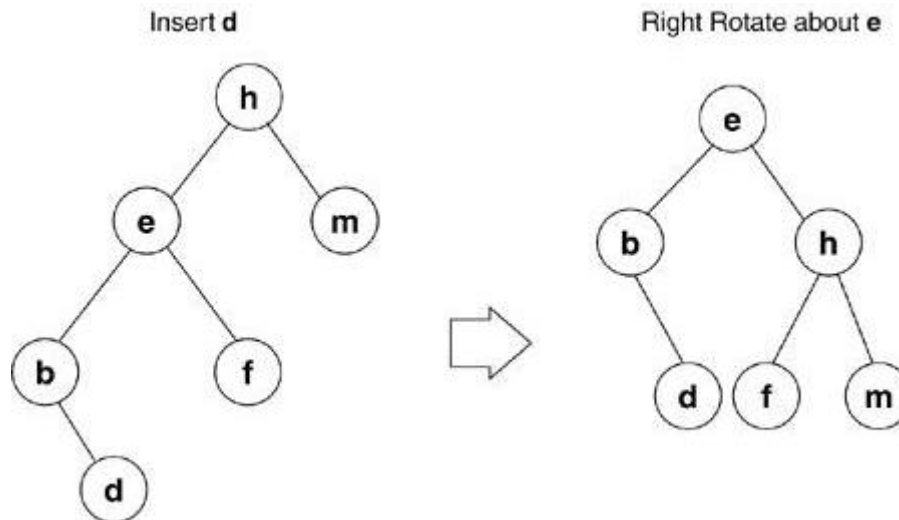
2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
- 2.1.2. Searching
- 2.1.3. Trees
 - BST (binary search tree)



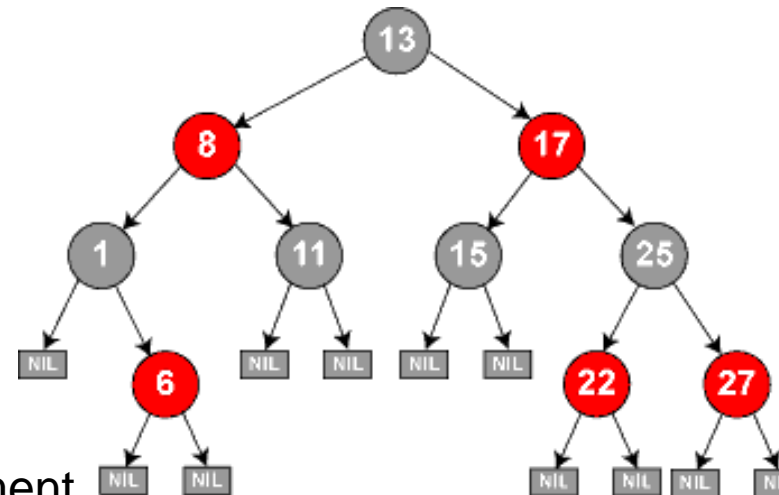
2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
- 2.1.2. Searching
- 2.1.3. Trees
 - Balanced BSD (AVL-Tree)



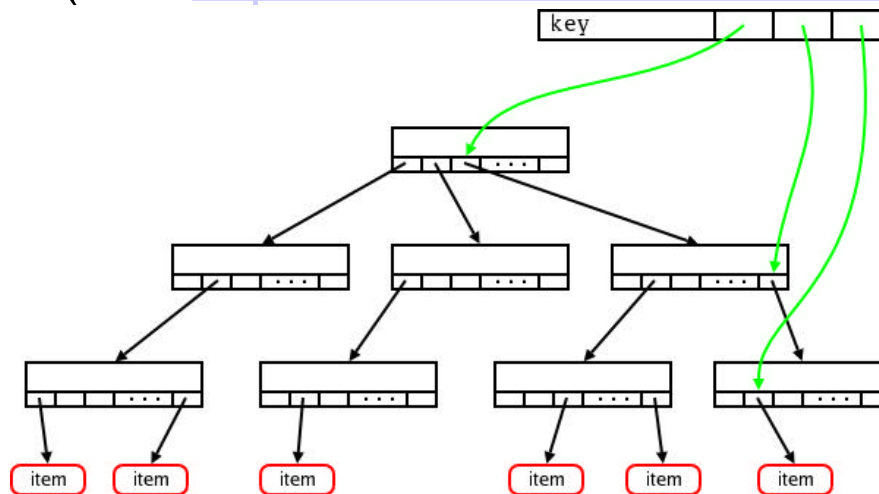
2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
- 2.1.2. Searching
- 2.1.3. Trees
 - Red-Black Tree (See <http://lwn.net/Articles/189122/>)
 - All nodes are either red or black.
 - If a node is red, both its children are black.
 - All leaf nodes are black.
 - When traversing from the root to a leaf, each path contains the same # of black nodes.
 - * Used in Linux memory management.
See `include/linux/rbtree.h`



2.1. Common Kernel Datatypes

- 2.1.1. Linked Lists
- 2.1.2. Searching
- 2.1.3. Trees
 - Radix Tree (See <http://lwn.net/Articles/175432/>)



* Used in Linux memory management and nfs.
See `include/linux/radix-tree.h`

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

2.8. Miscellaneous Quirks

2.2. Assembly

- PPC vs x86
 - PPC: RISC, x86: CISC
 - PPC: Big Endian, x86: Little Endian
 - representation of 0x12345678

Big Endian 32-Bit Byte Ordering (PPC)

12	34	56	78
0	7 8	15 16	23 24 31

Little Endian 32-Bit Byte Ordering (x86)

78	56	34	12
0	7 8	15 16	23 24 31

2.2. Assembly

- x86 Assembly
 - Registers of x86 (1/2)
 - EAX. General purpose accumulator
 - EBX. Pointer to data
 - ECX. Counter for loop operations
 - EDX. I/O pointer
 - ESI. Pointer to data in DS segment
 - EDI. Pointer to data in ES segment
 - ESP. Stack pointer
 - EBP. Pointer to data on the stack

2.2. Assembly

- x86 Assembly
 - Registers of x86 (2/2)
 - CS. Code segment
 - SS. Stack segment
 - ES, DS, FS, GS. Data segment
 - ES:EDI, DS:ESI, SS:ESP, SS:EBP, ...
 - EFLAGS. Status, control, and system flags
 - EIP. The instruction pointer, contains an offset from CS
 - CS:EIP

2.2. Assembly

- x86 Assembly
 - Control instructions
 - ‘cmp’ modifies flags in ‘EFLAGS’
 - OF (overflow), SF (sign flag), ZF (zero flag), PF (parity flag), CF (carry flag).
 - je(Jump if equal): jump when ZF=1
 - jg(Jump if greater): jump when ZF=0 and SF=OF
 - jge (Jump if greater or equal): jump when SF=OF
 - jl (Jump if less): jump when SF!=OF
 - jle (Jump if less or equal): jump when ZF=1
 - jmp(Unconditional jump): jump always

2.2. Assembly

- x86 Assembly
 - Control instructions
 - Example

```
100          pop eax
101 loop2:
102          pop ebx
103          cmp eax, ebx
104          jge loop2
```

2.2. Assembly

- x86 Assembly
 - Control instructions
 - call: push EIP and jump
 - ret: pop EIP
 - * iret

2.2. Assembly

- x86 Assembly
 - Arithmetic instructions
 - add, sub, imul (integer multiply), idiv (integer divide)
 - and, or, not, and xor

2.2. Assembly

- x86 Assembly
 - Data instructions

```
100    mov eax,ebx
101    mov eax,WORD PTR [data3]
102    mov BYTE PTR [char1],al
103    mov eax,0xbeef
104    mov WORD PTR [my_data],0xbeef
```
 - * load/store in RISCs

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

2.8. Miscellaneous Quirks

2.3. Assembly Language Example

- Example in the text: “count”
 - Generate “count.c”
 - Try “gcc -S count.c”
 - Try “gcc count.c && gdb a.out”, then enter “disassemble main”
 - Try “gcc count.c && objdump -S a.out | vim -”, then find “<main>”
 - Give “-g” to gcc and do the same.
- * SS:EBP is the top of stack.

2.3. Assembly Language Example

- AT&T format
 - %... : register
 - \$... : constant
 - OFFSET(BASE REGISTER): indexed addressing
 - The order of operands is reversed.

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

2.8. Miscellaneous Quirks

2.4. Inline Assembly

- Inline assembler construct
asm (assembler_instructions
: output_operands(optional)
: input_operands(optional)
: clobbered_operands(optional)
);
- Examples
asm ("movl %eax, %ebx");
__asm__ ("movl %eax, %ebx" : : :);
- We have to tell the compiler
which operands are clobbered(modified).

```

1 #include <stdio.h>
2 #define fastcall __attribute__((regparm(3)))
3
4 int fastcall setbits(int p1, int p2)
5 {
6     printf("setbits(): p1=0x%x, p2=0x%x\n", p1, p2);
7     return p1 | p2;
8 }
9
10 int foo(void)
11 {
12     int ee = 0x4000, ce = 0x8000, reg;
13
14     __asm__ __volatile__(
15         "movl %1, %%eax\n"
16         "movl %2, %%ebx\n"
17         "call setbits\n"
18         "movl %%eax, %0\n"
19         : "=r" (reg) // reg [param %0] is output
20         : "r" (ce), "r" (ee) // ce [param %1], ee [param %2] are inputs
21         : "%eax", "%ebx" // %eax and % ebx got clobbered
22     );
23     printf("reg=%x", reg);
24 }
25
26 int main()
27 {
28     foo();
29     printf("\n");
30     return 0;
31 }

```

2.4. Inline Assembly

- `__volatile__`
 - Tells the compiler not to optimize it.
 - cf. volatile variable
- Parameter numbering with %

2.4. Inline Assembly

- Example in the text:
switch_to() for i386
 - “\$1f” means the label “1:”
 (“f” means “forward”)
 - Linkage
 - FASTCALL: pass parameters in registers
 - asmlinkage: pass parameters on the stack
 - “pushl + jmp” instead of “call”

2.4. Inline Assembly

- See

<http://web.archive.org/web/20050315182310/http://linuxkernel.net/doc/flyduck/linuxasm/linux-assembly-code.html>

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

2.8. Miscellaneous Quirks

2.5. Quirky C Language Usage

- **asm linkage & FASTCALL**
 - `__attribute__` and `regparm` are nonstandard features of gcc.
 - **Ex:** `asm linkage long sys_gettimeofday(struct timeval __user *tv, struct timezone __user *tz)`
- **UL**
 - unsigned long constant
 - **Ex:**

```
include/linux/hash.h
18          #define GOLDEN_RATIO_PRIME 0x9e370001UL
include/linux/kernel.h
23          #define ULONG_MAX (~0UL)
include/linux/slab.h
39          #define SLAB_POISON 0x00000800U
```

2.5. Quirky C Language Usage

- inline
 - integrates the code of the function into the code of its callers
 - faster, more optimizable
 - ‘static inline’: discards the assembly code of the function
 - Disadvantages
 - bigger binary image
 - Slow down when accessing the CPU’s cache

2.5. Quirky C Language Usage

- **const & volatile**
 - **const**
 - means “read only”
 - Note that:
 - `const int *x` : a pointer to a const integer
 - `int const *x` : a const pointer to an integer
 - **volatile**
 - “Access the real memory, everytime”.
 - Important when using
 - Multithreading
 - Memory-mapped hardware registers

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel Messages

2.8. Miscellaneous Quirks

2.6. A Quick Tour of Kernel Exploration Tools

- objdump/readelf
 - displays informations within
 - object files (objdump)
 - ELF files (readelf)
- hexdump, od, xxd
- nm
 - lists symbols in object files
 - “nm | sort”
 - used when generating *System.map*
 - helpful when debugging libraries

2.6. A Quick Tour of Kernel Exploration Tools

- objcopy
- ar (archive)
 - combines object files into a library
 - separate object files from a library
- ldd
 - displays the dependency of shared libraries
- strings
 - U know 'xray' for MS-DOS?

2.6. A Quick Tour of Kernel Exploration Tools

- `gcc -M`
 - make the list of included header files
- `gcc -E`
 - Stop when preprocessing ends
 - “make `****.i`” in `/usr/src/linux`
- `gcc -S`
 - Stop when the assembly file generated
 - “make `****.s`” in `/usr/src/linux`

2.6. A Quick Tour of Kernel Exploration Tools

- gdb, kdb, kgdb, and ddd
- See
‘Self-Service Linux: Mastering the Art of Problem Determination’
by M. Wilding & D. Behman
or “ ”,

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

**2.7. Kernel Speak: Listening to Kernel
Messages**

2.8. Miscellaneous Quirks

2.7. Kernel Speak

- printk
- dmesg
- /proc/kmsg
- /var/log/messages
- syslogd(), klogd(), sysklogd()
- tail -F /var/log/kern.log -s 0.01
- To find the caller: (* CONFIG_KALLSYMS should be defined)
 - #include <linux/kallsyms.h>


```
#define PRINT_INFO() \
do { \
    printk("%s starting( caller:", __FUNCTION__); \
    print_symbol("%s)\n", (unsigned long) __builtin_return_address(0)); \
} while (0)
```

Contents

2.1. Common Kernel Datatypes

2.2. Assembly

2.3. Assembly Language Example

2.4. Inline Assembly

2.5. Quirky C Language Usage

2.6. A Quick Tour of Kernel Exploration Tools

2.7. Kernel Speak: Listening to Kernel
Messages

2.8. Miscellaneous Quirks

2.8. Miscellaneous Quirks

- `__init`, `__initdata`, `__exit`, `__exitdata`
 - “`__init`” functions and “`__initdata`” variables are placed in ***a special section*** (see *vmlinux.lds.S*).
 - freed after kernel initialization phase ends.
 - `start_kernel()` -> `rest_init()` -> `init()` -> `free_initmem()`
- `initcall`
 - `__initcall`
 - Two definitions of `module_init`
 - `start_kernel()` -> `rest_init()` -> `init()` -> `do_basic_setup()` -> `do_initcalls()`

2.8. Miscellaneous Quirks

- `likely()/unlikely()`
 - to help CPU's branch prediction
 - Branch prediction is important for instruction prefetching in pipelining architectures.
 - `likely()/unlikely()` uses `__builtin_expect()`
 - Ex: `sys_gettimeofday()`

2.8. Miscellaneous Quirks

- IS_ERR, PTR_ERR
 - IS_ERR: encodes a negative error number into a pointer
 - PTR_ERR: retrieves the error number from the pointer
 - Ex: blk_rq_map_user() -> bio_copy_user()

2.8. Miscellaneous Quirks

- Notifier Chains

- Ex:

- watchdog_init() in *drivers/char/watchdog/softdog.c* calls register_reboot_notifier(), which registers 'softdog_notify_sys' to 'reboot_notifier_list'.
 - kernel_shutdown_prepare() calls blocking_notifier_call_chain(), then the functions registered to 'reboot_notifier_list' are called.