CS 5264/4224; ECE 5414/4414
(Advanced) Linux Kernel Programming
Lecture 5

Kernel Data Structures & Debugging

February 6, 2025

Huaicheng Li

Acknowledgement: Credits to Dr. Changwoo Min for the original LKP lecture slides.

Improving Syscall Performance

- System call performance is critical in many applications
  - Web server: select(), poll()
  - Game engine: gettimeofday()
- Hardware: add a new fast system call instruction
  - int 0x80 → syscall
- Software: vDSO (virtual dynamically linked shared object)
  - A kernel mechanism for exporting a kernel space routines to user space applications
  - No context switching overhead
  - e.g., "gettimeofday()"
    - » the kernel allows the page containing the current time to be mapped read-only into user space
- Software: FlexSC: Exception-less system call, OSDI 2010
  - Optimizing system call performance for large multi-core systems
  - "FlexSC improves performance of Apache by up to 116%, MySQL by up to 40%, and BIND by up to 105% while requing no modifications to the applications

Readings

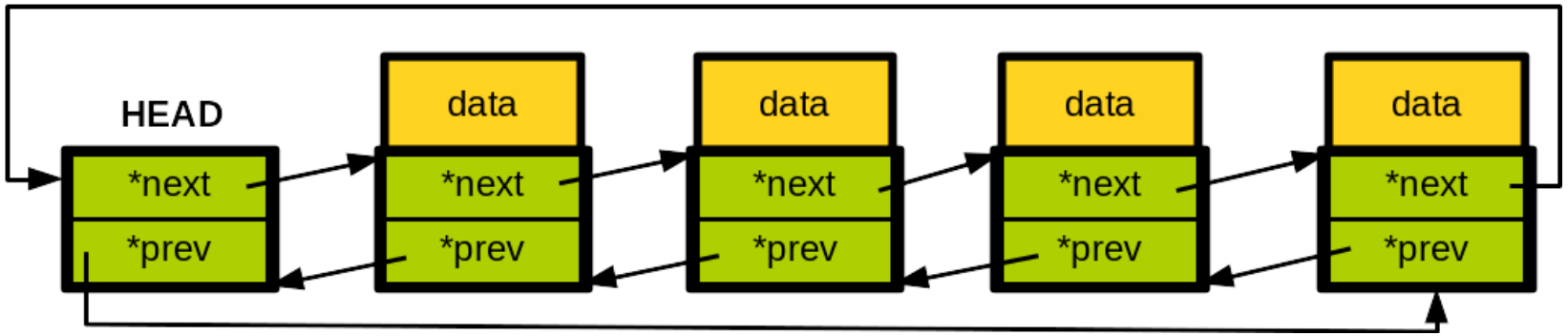
- LWN: Anatomy of a system call: part 1 and part 2
- LWN: On vsyscalls and the vDSO
- Linux Inside: system calls
- Linux Performance Analysis: New Tools and Old Secrets

Today's Agenda

- Data structures
- Kernel debugging

Linux Linked List

- Starts from HEAD and terminates at HEAD
- When empty, HEAD is not NULL
  - prev and next of HEAD points to HEAD
  - HEAD is a sentinel node
- Easy to insert a new element at the end of a list
- There is no exceptional case to handle NULL



Manipulating a List: O(1)

```c
/* Insert a new entry after the specified head */
void list_add(struct list_head *new, struct list_head *head);

/* Insert a new entry before the specified head */
void list_add_tail(struct list_head *new, struct list_head *head);

/* Delete a list entry
 * NOTE: You still have to take care of the memory deallocation if needed */
void list_del(struct list_head *entry);

/* Delete from one list and add as another's head */
void list_move(struct list_head *list, struct list_head *head);

/* Delete from one list and add as another's tail */
void list_move_tail(struct list_head *list, struct list_head *head);

/* Tests whether a list is empty */
int list_empty(const struct list_head *head);

/* Join two lists (merge a list to the specified head) */
void list_splice(const struct list_head *list, struct list_head *head);
```

Iterating over a List: O(n)

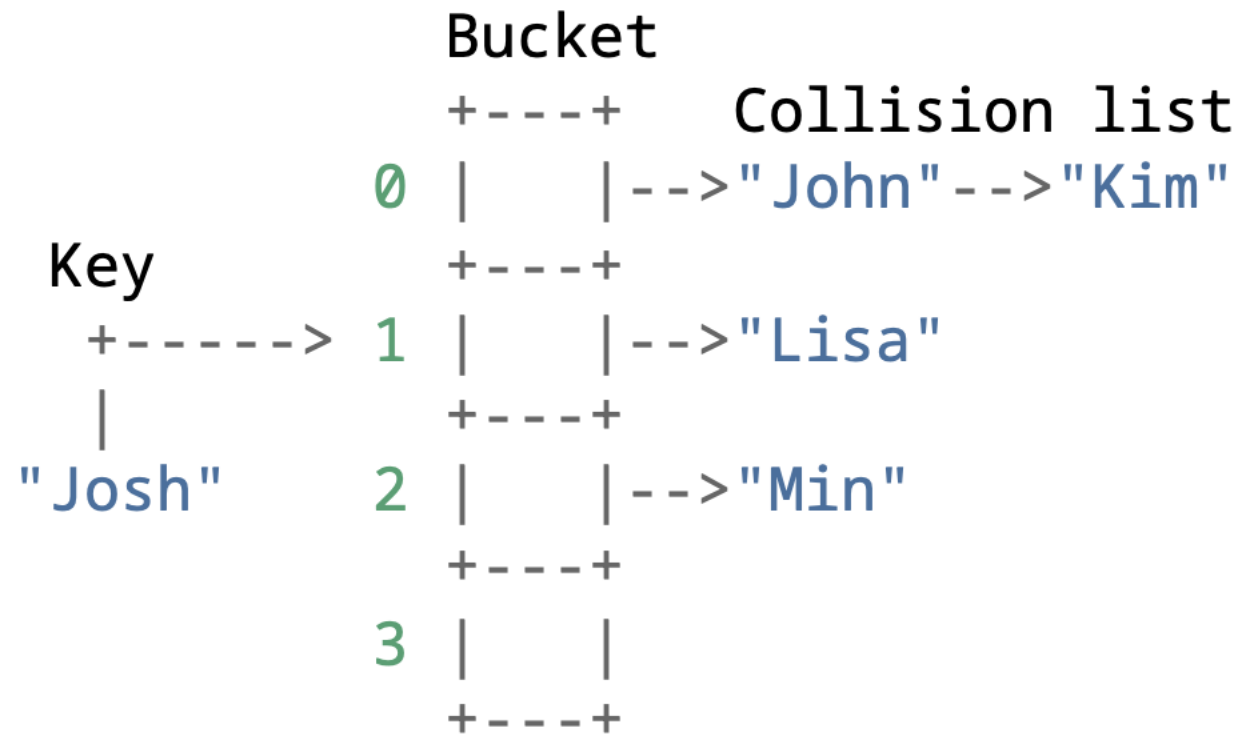
```
/**
 * list_for_each    -   iterate over a list
 * @pos:    the &struct list_head to use as a loop cursor.
 * @head:   the head for your list.
 */
#define list_for_each(pos, head) \
    for (pos = (head)->next; pos != (head); pos = pos->next)

/**
 * list_for_each_entry  -   iterate over list of given type
 * @pos:    the type * to use as a loop cursor.
 * @head:   the head for your list.
 * @member: the name of the list_head within the struct.
 */
#define list_for_each_entry(pos, head, member)                  \
    for (pos = list_first_entry(head, typeof(*pos), member); \
         &pos->member != (head);                                \
         pos = list_next_entry(pos, member))
```

```c
/* Temporary variable needed to iterate: */
struct list_head p;

/* This will point on the actual data structures
 * (struct car)during the iteration: */
struct car *current_car;

list_for_each(p, &my_car_list) {
    current_car = list_entry(p, struct car, list);
    printk(KERN_INFO "Price: %lf\n", current_car->price_in_dollars);
}

/* Simpler: use list_for_each_entry */
list_for_each_entry(current_car, &my_car_list, list) {
    printk(KERN_INFO "Price: %lf\n", current_car->price_in_dollars);
}
```

Backward iteration? → list_for_each_entry_reverse(pos, head, member)

Linux Hash Table

- A simple fixed-size open chaining hash table
  - The size of bucket array is fixed at initialization as a 2^N
  - Each bucket has a singly linked list to resolve hash collision
  - Time Complexity: O(1)



```c
/* linux/include/linux/hashtable.h, types.h */
/* hash bucket */
struct hlist_head {
  struct hlist_node *first;
};

/* collision list */
struct hlist_node {
  /* Similar to list_head, hlist_node is embedded
   * into a data structure. */
  struct hlist_node *next;
  struct hlist_node **pprev; /* &prev->next */
};
```

```
  Bucket: array of hlist_head
  +---+   Collision list: hlist_node
0 |   |-->"John"-->"Kim"
  +---+
1 |   |-->"Josh"-->"Lisa"
  +---+
2 |   |-->"Min"
  +---+
```

Linux Hash Table API

```
/**
 * Define a hashtable with 2^bits buckets
 */
#define DEFINE_HASHTABLE(name, bits) ...

/**
 * hash_init - initialize a hash table
 * @hashtable: hashtable to be initialized
 */
#define hash_init(hashtable) ...

/**
 * hash_add - add an object to a hashtable
 * @hashtable: hashtable to add to
 * @node: the &struct hlist_node of the object to be added
 * @key: the key of the object to be added
 */
#define hash_add(hashtable, node, key) ...
```

Linux hash table API

```
/**
 * hash_for_each - iterate over a hashtable
 * @name: hashtable to iterate
 * @bkt: integer to use as bucket loop cursor
 * @obj: the type * to use as a loop cursor for each entry
 * @member: the name of the hlist_node within the struct
 */
#define hash_for_each(name, bkt, obj, member) ...
```

```
  +---+
0 |   |-->"John"<-->"Kim"
  +---+
1 |   |-->"Josh"<-->"Lisa"
  +---+
2 |   |-->"Min"
  +---+
3 |   |
  +---+
```

```c
/**
 * hash_for_each_possible - iterate over all possible objects hashing to the
 * same bucket
 * @name: hashtable to iterate
 * @obj: the type * to use as a loop cursor for each entry
 * @member: the name of the hlist_node within the struct
 * @key: the key of the objects to iterate over
 */
#define hash_for_each_possible(name, obj, member, key) ...
```

```
          +---+
        1 |   |-->"Josh"<-->"Lisa"
          +---+
```

```c
/**
 * hash_del - remove an object from a hashtable
 * @node: &struct hlist_node of the object to remove
 */
void hash_del(struct hlist_node *node);

/* Q. No lookup? */
```

Linux Hash Table Examples

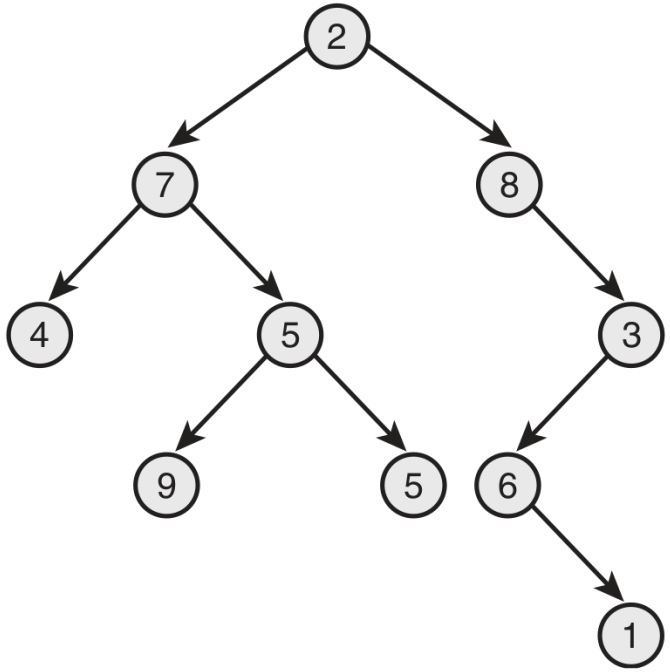
- Transparent hugepages
  - finds physically consecutive 4KB pages
  - rempas consecutive 4KB pages to a 2MB page (huge page)
  - saves TLB entries and improves memory access performance by reducing TLB miss
  - maintains per-process memory structure, “struct mm_struct”

```c
/* linux/mm/khugepaged.c */

#define MM_SLOTS_HASH_BITS 10
static DEFINE_HASHTABLE(mm_slots_hash, MM_SLOTS_HASH_BITS);

/* struct mm_slot - hash lookup from mm to mm_slot
 * @hash: hash collision list
 * @mm: the mm that this information is valid for
 */
struct mm_slot {
    struct hlist_node hash;  /* hlist_node is embedded like list_head */
    struct mm_struct *mm;
};
```

```c
/* add an mm_slot into the hash table
 * use the mm pointer as a key */
static void insert_to_mm_slots_hash(struct mm_struct *mm,
                    struct mm_slot *mm_slot)
{
    mm_slot->mm = mm;
    hash_add(mm_slots_hash, &mm_slot->hash, (long)mm);
}

/* iterate the chained list of a bucket to find an entry */
static struct mm_slot *get_mm_slot(struct mm_struct *mm)
{
    struct mm_slot *mm_slot;

    hash_for_each_possible(mm_slots_hash, mm_slot, hash, (unsigned long)mm)
        if (mm == mm_slot->mm)
            return mm_slot;

    return NULL;
}
```

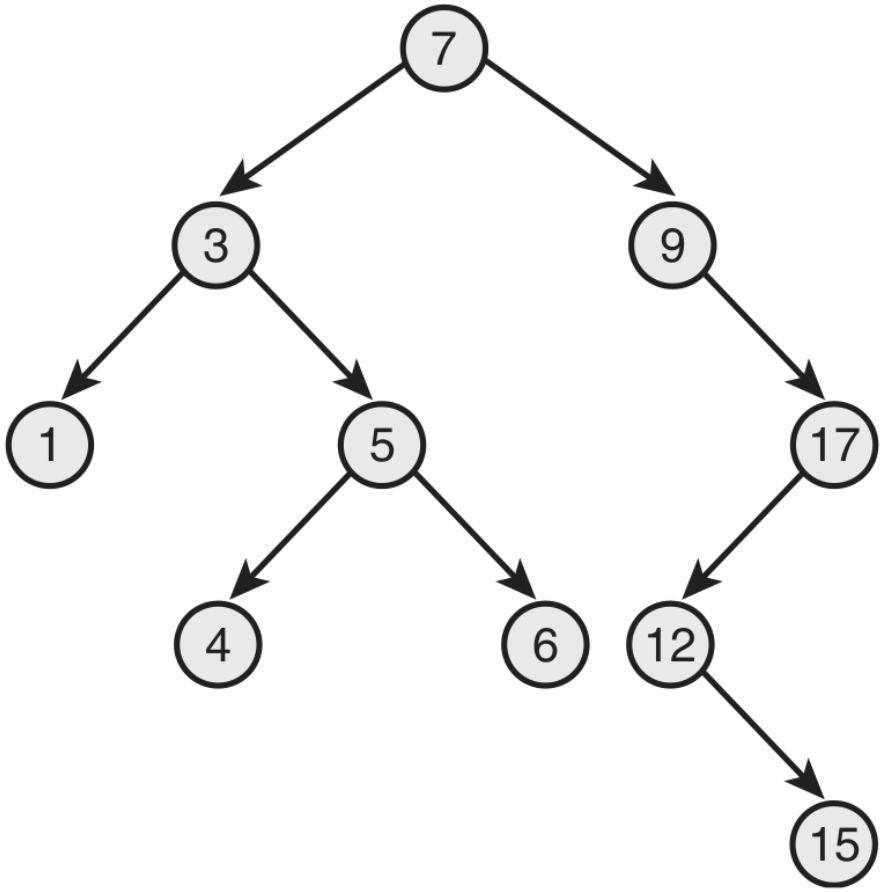
```c
/* remove an entry after finding it */
void __khugepaged_exit(struct mm_struct *mm)
{
    struct mm_slot *mm_slot;

    spin_lock(&khugepaged_mm_lock);
    mm_slot = get_mm_slot(mm);
    if (mm_slot && khugepaged_scan.mm_slot != mm_slot) {
        hash_del(&mm_slot->hash);
        list_del(&mm_slot->mm_node);
        free = 1;
    }
    spin_unlock(&khugepaged_mm_lock);

    clear_bit(MMF_VM_HUGEPAGE, &mm->flags);
    free_mm_slot(mm_slot);
    mmdrop(mm);
    /* ... */
}
```

Binary Tree



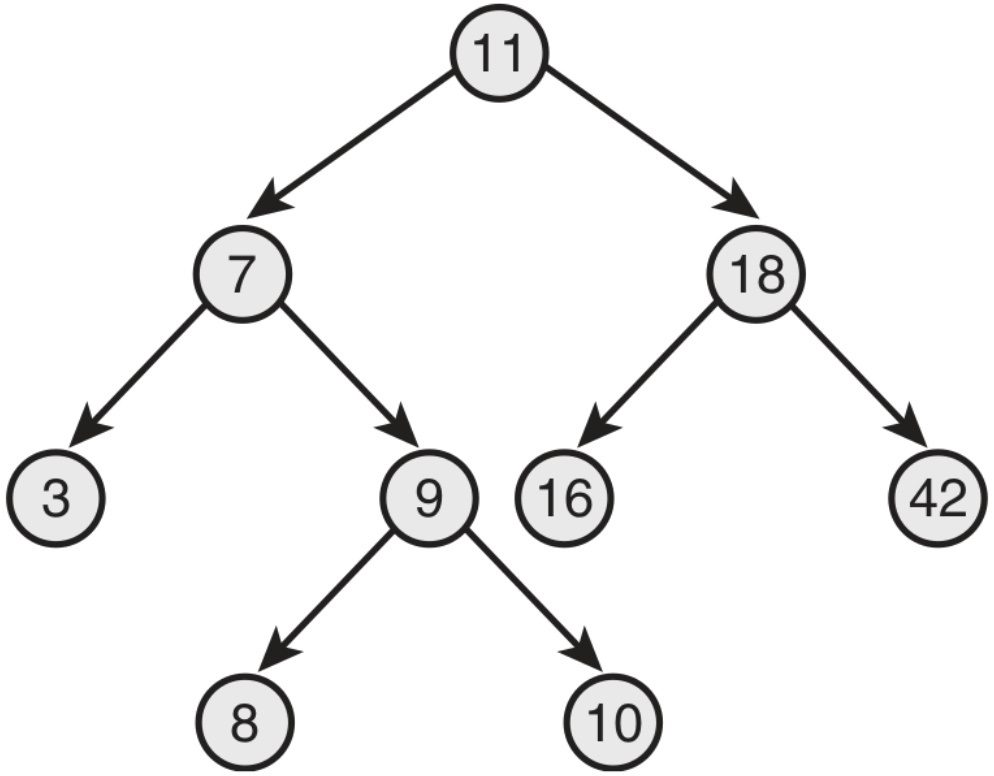
- Nodes have zero, one, or two children
- Root has no parent, other nodes have one

Binary Search Tree



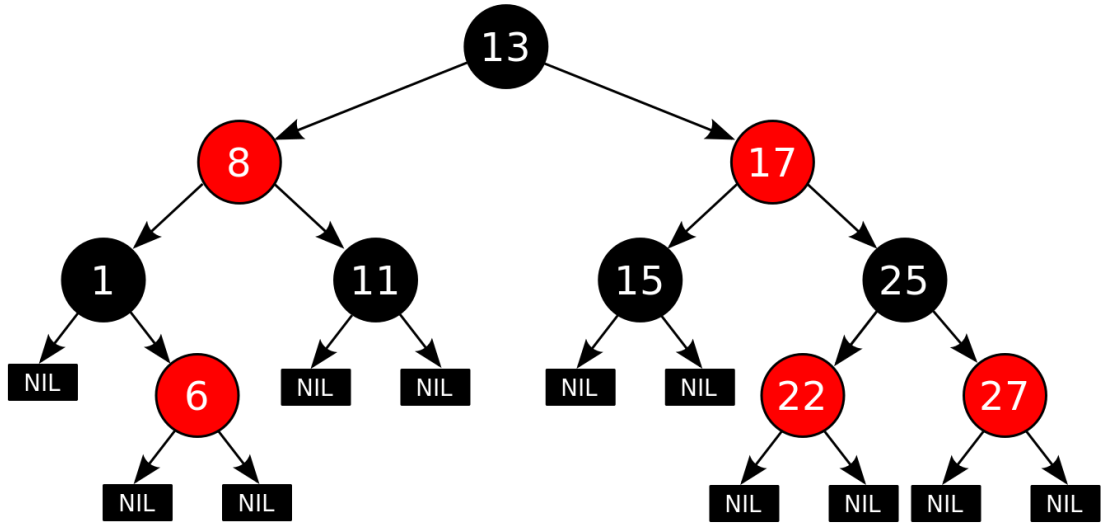
- Left children < parent
- Right children > parent
- Search and ordered traversal are efficient

Balanced Binary Search Tree



- Depth of all leaves differs by at most one
- Puts a boundary on the worst-case operations

Tree Basics: Red-Black Tree



- A type of self-balancing binary search tree
  - Nodes: red or black
  - Leaves: black, no data
- The following properties are maintained during tree modifications:
  - The path from a node to one of its leaves contains the same number of black nodes as the shortest path to nay of its other leaves
- Fast search, insert, delete operations: O(log N)

Linux Red-Black Tree (or rbtree)

```c
/* linux/include/linux/rbtree.h
 * linux/lib/rbtree.c */

/* Rbtree node, which is embedded to your data structure like
 * list_head and hlist node */
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
};

/* Root of a rbtree */
struct rb_root {
    struct rb_node *rb_node;
};

#define RB_ROOT (struct rb_root) { NULL, }

/* A macro to access data from rb_node */
#define rb_entry(ptr, type, member) container_of(ptr, type, member)
#define rb_parent(r) ((struct rb_node *)((r)->__rb_parent_color & ~3))
```

```c
/* Find logical next and previous nodes in a tree */
struct rb_node *rb_next(const struct rb_node *);
struct rb_node *rb_prev(const struct rb_node *);
struct rb_node *rb_first(const struct rb_root *);
struct rb_node *rb_last(const struct rb_root *);

/* Insert a new node under a parent connected via rb_link */
void rb_link_node(struct rb_node *node, struct rb_node *parent,
                  struct rb_node **rb_link);

/* Re-balance an rbtree after inserting a node if necessary */
void rb_insert_color(struct rb_node *, struct rb_root *);

/* rb_add() - insert @node into @tree
 * @node: node to insert
 * @tree: tree to insert @node into
 * @less: operator defining the (partial) node order
 */
static __always_inline void
rb_add(struct rb_node *node, struct rb_root *tree,
       bool (*less)(struct rb_node *, const struct rb_node *));
```

```c
/* rb_find_add() - find equivalent @node in @tree, or add @node
 * @node: node to look-for / insert
 * @tree: tree to search / modify
 * @cmp: operator defining the node order
 * Returns the rb_node matching @node, or NULL when no match is found
 * and @node is inserted.
 */
static __always_inline struct rb_node *
rb_find_add(struct rb_node *node, struct rb_root *tree,
        int (*cmp)(struct rb_node *, const struct rb_node *));

/* Delete a node */
void rb_erase(struct rb_node *, struct rb_root *);
```

```c
/* rb_find() - find @key in tree @tree
 * @key: key to match
 * @tree: tree to search
 * @cmp: operator defining the node order
 * Returns the rb_node matching @key or NULL.
 */
static __always_inline struct rb_node *
rb_find(const void *key, const struct rb_root *tree,
    int (*cmp)(const void *key, const struct rb_node *));

/* rb_find_add() - find equivalent @node in @tree, or add @node
 * @node: node to look-for / insert
 * @tree: tree to search / modify
 * @cmp: operator defining the node order
 * Returns the rb_node matching @node, or NULL when no match is found
 * and @node is inserted.
 */
static __always_inline struct rb_node *
rb_find_add(struct rb_node *node, struct rb_root *tree,
        int (*cmp)(struct rb_node *, const struct rb_node *));
```

Linux Red-Black Tree Example

- Completely Fair Scheduling (CFS)
  - Default task scheduler in Linux for a long time …
  - Each task has "vruntime", which presents how much time a task has run
  - CFS always picks a process with the smallest "vruntime" for fairness
  - Per-task "vruntime" structure is maintained in a rbtree

```c
/* linux/include/linux/sched.h
 * linux/kernel/sched/fair.c, sched.h */

/* Define an rbtree */
struct cfs_rq {
    struct rb_root tasks_timeline; /* contains sched_entity */
};

/* Data structure of a task */
struct sched_entity {
    struct rb_node  run_node; /* embed a rb_node */
    u64             vruntime; /* vruntime is the key of task_timeline */
};

/* Initialize an rbtree */
void init_cfs_rq(struct cfs_rq *cfs_rq)
{
    cfs_rq->tasks_timeline = RB_ROOT;
}
```

```c
/* Enqueue an entity into the rb-tree: */
void __enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
    struct rb_node **link = &cfs_rq->tasks_timeline.rb_node; /* root node */
    struct rb_node *parent = NULL;
    struct sched_entity *entry;

    /* Traverse the rbtree to find the right place to insert */
    while (*link) {
        parent = *link;
        entry = rb_entry(parent, struct sched_entity, run_node);
        if (se->vruntime < entry->vruntime) {
            link = &parent->rb_left;
        } else {
            link = &parent->rb_right;
        }
    }

    /* Insert a new node */
    rb_link_node(&se->run_node, parent, link);
    /* Re-balance the rbtree if necessary */
    rb_insert_color(&se->run_node, &cfs_rq->tasks_timeline);
}
```

```c
/* Delete a node */
void __dequeue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
    rb_erase(&s->run_node, &cfs_rq->tasks_timeline);
}

/* Pick the first enitiy, which has the smallest vruntime,
 * for scheduling */
struct sched_entity *__pick_first_entity(struct cfs_rq *cfs_rq)
{
    return rb_first(&cfs_rq->tasks_timeline);
}
```

Kernel Debugging

- tools, techniques, and tricks

Kernel Development Cycle

- Write code → build kernel/modules → Deploy → Test and debug
- Debugging is the real bottleneck even for experienced kernel developers due to limitations in kernel debugging
- It is important to be familiar with kernel debugging techniques to save time

Kernel Debugging Techniques

- Print debug message: printk()
- Assert your code: BUG_ON(c), WARN_ON(c)
- Analyze kernel panic message
- Debug with QEMU/gdb

Print Debugging Message: printk()

- Similar to printf() in C library
- Need to specify a log level (the default level is KERN_WARNING or KERN_ERR)

```
KERN_EMERG    /* 0: system is unusable                    */
KERN_ALERT    /* 1: action must be taken immediately      */
KERN_CRIT     /* 2: critical conditions                   */
KERN_ERR      /* 3: error conditions                      */
KERN_WARNING  /* 4: warning conditions                    */
KERN_NOTICE   /* 5: normal but significant condition      */
KERN_INFO     /* 6: informational                         */
KERN_DEBUG    /* 7: debug-level messages                  */

e.g., printk(KERN_DEBUG "debug message from %s:%d\n", __func__, __LINE__);
```

- Prints out only messages whose log level is higher than the current
- The kernel message buffer is a fixed-size circular buffer.
- If the buffer fills up, it wraps around and you can lose messages
- Increasing the buffer size would help a bit
  - e.g., add "log_buf_len=1M" to kernel boot parameter (2^N)

```
# Check current kernel log level
$ cat /proc/sys/kernel/printk
    4       4       1       7
#   |       |       |       |
# current default minimum  boot-time-default
# Enable all levels of messages:
$ echo 7 > /proc/sys/kernel/printk
```

- Support additional specifiers
- Reference: How to get printk format specifiers right

```
/* function pointers with function name */
"%pF"     versatile_init+0x0/0x110 /* symbol+offset/length */
"%pf"     versatile_init

/* direct code address (e.g., regs->ip) */
"%pS"     versatile_init+0x0/0x110
"%ps"     versatile_init

/* direct code address in stack (e.g., return address) */
"%pB"     prev_fn_of_versatile_init+0x88/0x88

/* Example */
printk("Going to call: %pF\n", p->func);
printk("Faulted at %pS\n", (void *)regs->ip);
printk(" %s%pB\n", (reliable ? "" : "? "), (void *)*stack);
```

BUG_ON(), WARN_ON()

- Similar to assert(c) in userspace
- BUG_ON(c)
  - if c is false, kernel panics with its call stack
- WARN_ON(c)
  - if c is false, kernel prints out its call stack and keeps running

Kernel Panic Message

```
[  174.507084] Stack:
[  174.507163]  ce0bd8ac 00000008 00000000 ce4a7e90 c039ce30 ce0bd8ac c0718b04 c07185a0
[  174.507380]  ce4a7ea0 c0398f22 ce0bd8ac c0718b04 ce4a7eb0 c037deee ce0bd8e0 ce0bd8ac
[  174.507597]  ce4a7ec0 c037dfe0 c07185a0 ce0bd8ac ce4a7ed4 c037d353 ce0bd8ac ce0bd8ac
[  174.507888] Call Trace:
[  174.508125]  [<c039ce30>] ? sd_remove+0x20/0x70
[  174.508235]  [<c0398f22>] ? scsi_bus_remove+0x32/0x40
[  174.508326]  [<c037deee>] ? __device_release_driver+0x3e/0x70
[  174.508421]  [<c037dfe0>] ? device_release_driver+0x20/0x40
[  174.508514]  [<c037d353>] ? bus_remove_device+0x73/0x90
[  174.508606]  [<c037bccf>] ? device_del+0xef/0x150
[  174.508693]  [<c0399207>] ? __scsi_remove_device+0x47/0x80
[  174.508786]  [<c0399262>] ? scsi_remove_device+0x22/0x40
[  174.508877]  [<c0399324>] ? __scsi_remove_target+0x94/0xd0
[  174.508969]  [<c03993c0>] ? __remove_child+0x0/0x20
[  174.509060]  [<c03993d7>] ? __remove_child+0x17/0x20
[  174.509148]  [<c037b868>] ? device_for_each_child+0x38/0x60
```

Analyze Kernel Panic Message

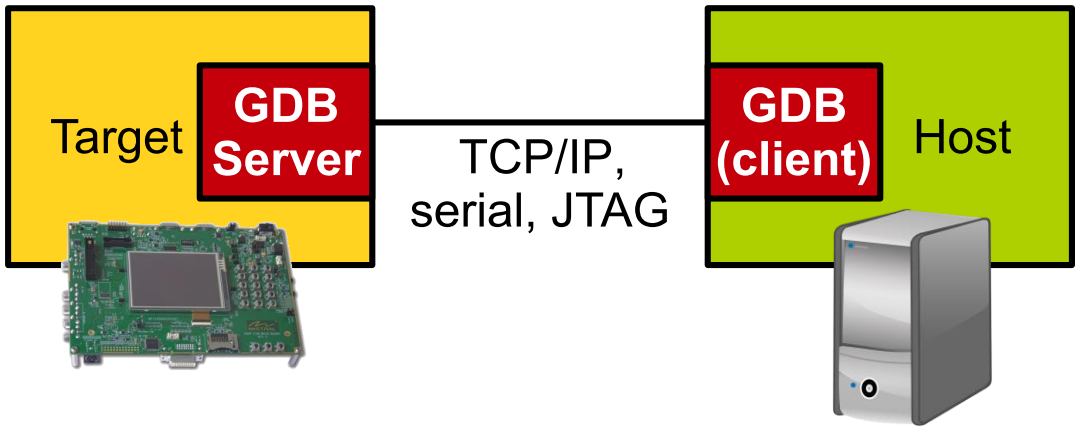
- Find where sd_remove() is, e.g., in linux/driver/scsi/sd.c
- Load its object file with gdb
- Use gdb to identify the offending code, “list *(function+offset)”

QEMU

- Full system emulator: emulates and entire virtual machine
  - Using a software model for the CPU, memory, devices
  - Emulation is slow ...
- Can be used in conjunction with hardware virtualization extensions to provide high performance virtualization
  - KVM: in-kernel support for virtualization + extensions to QEMU

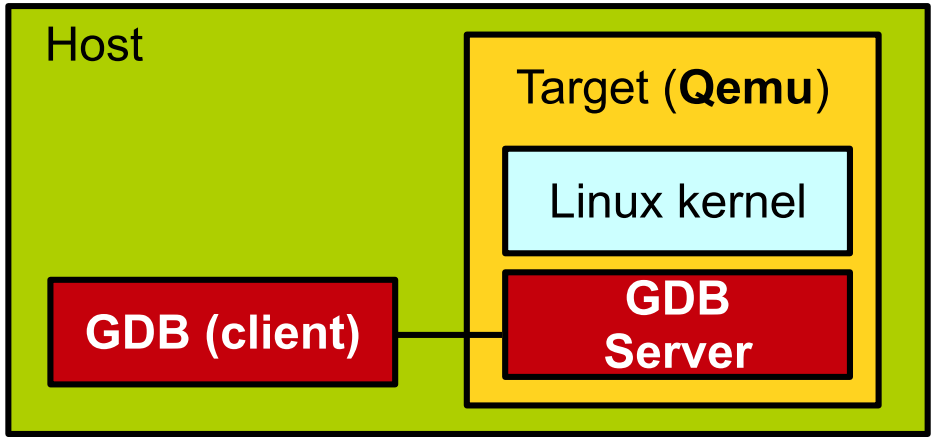
GDB Server

- Originally used to debug a program executing on a remote machine
- for example, when GDB is not available on that remote machine
  - e.g., low performance embedded systems



Debugging with QEMU/GDB

- Linux kernel runs in a virtual machine (KVM or emulated on QEMU)
- Hardware devices are emulated with QEMU
- GDB server runs at QEMU, emulated VM, so it can fully control Linux kernel running on QEMU
- Powerful for debugging and code exploration ...



Two Ways for Kernel Debugging

- Running minimal Linux dis
  - Use a “debootstrap”-ed distribution
  - Root file system is a directory in a host system
  - Limited functional in userspace applications
- Running full Linux distro
  - Use a QEMU disk image (qcow2, or raw disk)
  - Root file system is on the disk image
  - Able to run full userspace applications

Build Kernel for QEMU Debugging

- Rebuild kernel with gdb script, 9p, and virtio enabled
- Following should be built-in not built as a kernel module

```
$ cat .config
CONFIG_DEBUG_INFO=y          # debug symbol
CONFIG_GDB_SCRIPTS=y         # qemu/gdb support
CONFIG_E1000=y               # default network card

CONFIG_VIRTIO=y              # file sharing with host
CONFIG_NET_9P=y              # file sharing with host
CONFIG_NET_9P_VIRTIO=y       # file sharing with host
CONFIG_9P_FS=y               # file sharing with host
CONFIG_9P_FS_POSIX_ACL=y     # file sharing with host
CONFIG_9P_FS_SECURITY=y      # file sharing with host
```

```
# or (use `/ GDB_SCRIPTS` in `make menuconfig`
$ make menuconfig
  | Prompt: Provide GDB scripts for kernel debugging                    |
  |   Location:                                                          |
  |     -> Kernel hacking                                                |
  |       -> Compile-time checks and compiler options                    |
  |         -> Provide GDB scripts for kernel debugging                  |

# then build the kernel
$ make -j8; make -j8 modules
# No need to `make modules_install; make install`
# because all necessary features are embedded into the kernel.
```

Debootstrap Linux Distribution

- install-debian.sh

```
#!/bin/bash

# debootstrap the latest, minimal debian linux
sudo debootstrap --no-check-gpg --arch=amd64 \
     --include=kmod,net-tools,iproute2,iputils-ping \
     --variant=minbase \ # minimal base packages
     sid \               # latest debian distribution
     linux-chroot        # target directory

# copy the minimal initialization code to the target directory
sudo cp start.sh linux-chroot/

# allow gdb to load gdb script files
echo 'set auto-load safe-path ~/' > ~/.gdbinit
```

QEMU Options for Kernel Debugging

- **"-kernel vmlinux", path to the vmlinux of the kernel to debug**
- **"-s": enable the GDB server and open a port 1234**
- **"-S": (optional) pause on the first kernel instruction waiting for a GDB client connection to continue**

$ cd /path/to/linux-build
$ gdb vmlinux
(gdb) target remote :1234

```
  - [b]reak <function name or filename:line# or *memory addres>
- [hbreak] <start_kernel or any function name> # to debug boot code
- [d]elete <breakpoint #>
- [c]continue
- [b]ack[t]race
- [i]nfo [b]reak
- [n]ext
- [s]tep
- [p]rint <variable or *memory address>
- Ctrl-x Ctrl-a: TUI mode
```

Tips

- Disable optimizations ...
- Terminate QEMU with “halt” to avoid corrupting the disk image
- Run QEMU with KVM, “enable-kvm” (only for Linux host)

  - `(gdb) p my_var` → `$1 = <value optimized out>`
    - `my_var` is optimized out
    - Since it is not possible to disable optimization for the entire kernel, we need to disable optimization for a specific file.

```
# linux/fs/ext4/Makefile
obj-$(CONFIG_EXT4_FS) += ext4.o

CFLAGS_bitmap.o = -O0  # disable optimization of bitmap.c

ext4-y := balloc.o bitmap.o dir.o file.o \
#...
```

```
#!/bin/bash
KNL_SRC=~/workspace/research/linux
BZIMAGE=${KNL_SRC}/arch/x86_64/boot/bzImage
VMIMAGE=${PWD}/linux-vm.qcow2
sudo qemu-system-x86_64 -s \                    # enable qemu-gdb debugging
     -nographic \                               # without graphic
     -hda ${VMIMAGE} \                          # disk image in qcow2 or raw format
     -kernel ${BZIMAGE} \                       # kernel binary
     \       # boot parameter: no KASLR, set HDD, enable serial console
     -append "nokaslr root=/dev/sda1 console=ttyS0" \
     -smp cpus=2 \                              # set num of CPUs
     -device e1000,netdev=net0 \                # enable network adapater
             # forward TCP:5555 to 22 for ssh
     -netdev user,id=net0,hostfwd=tcp::5555-:22 \
     -m 2G                                      # set memory size

# Converting vmdk to qcow2:
#   qemu-img convert -O qcow2 linux-vm.vmdk linux-vm.qcow2
# Ctrl-a x: terminating QEMU
```

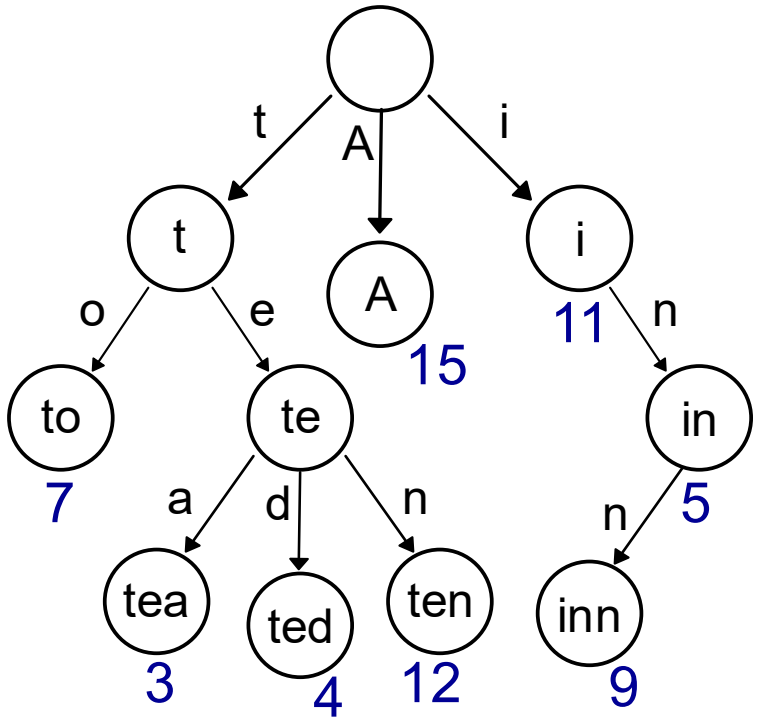
Other Tools

- ftrace
- kprobes
- dump_stack()
- ...

Further Readings

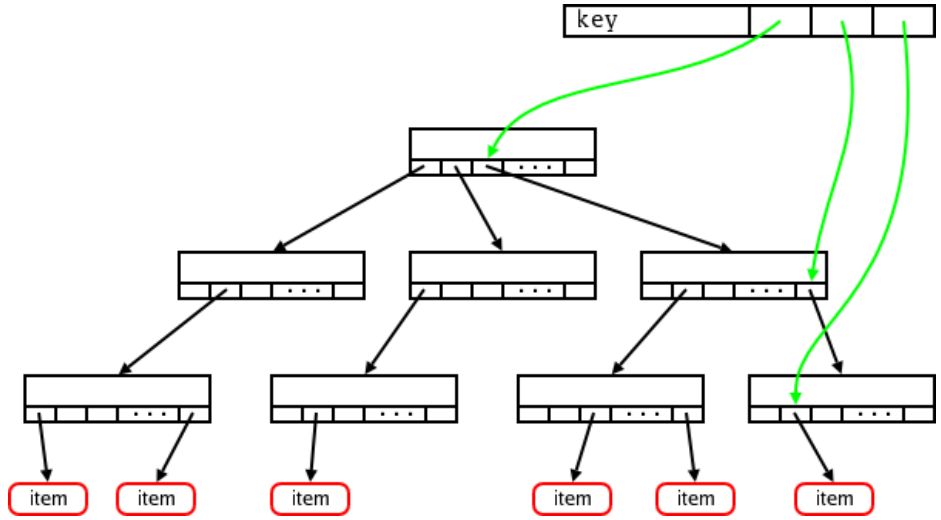
- Debugging by printing
- Kernel Debugging Tricks
- Kernel Debugging Tips
- Debugging kernel and modules via gdb
- gdb Cheatsheet
- Speed up your kernel development cycle with QEMU
- Migrate a VirtualBox Disk Image (.vdi) to a QEMU Image

Radix Tree (or tries)



- The key at each node is compared chunk-of-bits by chunk-of-bits
- All descendents of a node have a common prefix
- Values are only associated with leaves
- See Wiki

Linux Radix Tree



- Mapping between "unsigned long" and "void *"
- Each node has 64 slots
- Slots are indexed by a 6-bit portion of the key
- Source: LWN
- At leaves, a slot points to an address of data
- At non-leaf nodes, a slot points to another node in a lower layer
- Other metadata is also stored at each node: tags, parent pointer, offset in parent, etc.
- tags: specific bits can be set on items in the trees (0, 1, 2)
  - e.g., set the status of memory pages, which are dirty or under writeback

Linux Radix Tree API

```c
/* linux/include/linux/radix-tree.h, linux/lib/radix-tree.c */
#define RADIX_TREE_MAX_TAGS 3
#define RADIX_TREE_MAP_SIZE (1UL << 6)

/* Root of a radix tree */
struct radix_tree_root {
    gfp_t                   gfp_mask; /* used to allocate internal nodes */
    struct radix_tree_node  *rnode;
};

/* Radix tree internal node,
 * which is composed of slot and tag array */
struct radix_tree_node {
    unsigned char           offset;      /* Slot offset in parent */
    struct radix_tree_node *parent;      /* Used when ascending tree */
    void                   *slots[RADIX_TREE_MAP_SIZE];
    unsigned long           tags[RADIX_TREE_MAX_TAGS][RADIX_TREE_TAG_LONGS];
    /* ... */
};
```

- **Q: Is `radix_tree_node` embedded to user data (`list_head`)?**

```c
/* Root of a radix tree */
struct radix_tree_root {
    gfp_t                   gfp_mask; /* used to allocate internal nodes */
    struct radix_tree_node  *rnode;
};

/* Radix tree internal node,
 * which is composed of slot and tag array */
struct radix_tree_node {
    unsigned char           offset;     /* Slot offset in parent */
    struct radix_tree_node *parent;     /* Used when ascending tree */
    void                   *slots[RADIX_TREE_MAP_SIZE];
    unsigned long           tags[RADIX_TREE_MAX_TAGS][RADIX_TREE_TAG_LONGS];
    /* ... */
};
```

- **Q: Is `radix_tree_node` embedded to user data (`list_head`)?**
  - It is dynamically allocated when inserting an item.

```c
/* Declare and initialize a radix tree
 * @gfp_mask: how memory allocations are to be performed
 *            (e.g., GFP_KERNEL, GFP_ATOMIC, GFP_FS, etc) */
RADIX_TREE(name, gfp_mask);

/* Initialize a radix tree at runtime */
struct radix_tree_root my_tree;
INIT_RADIX_TREE(my_tree, gfp_mask);

/* Insert an item into the radix tree at position @index.
 * @root:       radix tree root
 * @index:      index key
 * @item:       item to insert */
int radix_tree_insert(struct radix_tree_root *root,
        unsigned long index, void *item);
```

- **Q: What happens if memory allocation fails?**

```
/* 1. Allocate sufficient memory (using the given gfp_mask) to guarantee
 * that the next radix tree insertion cannot fail. When successful,
 * it disables preemption so the pre-allocated memory can be used for
 * subsequent radix_tree_insert() operations. */
int radix_tree_preload(gfp_t gfp_mask);

/* 2. Insert an item into the radix tree at position @index.
 * @root:  radix tree root
 * @index: index key
 * @item:  item to insert */
int radix_tree_insert(struct radix_tree_root *root,
        unsigned long index, void *item);

/* 3. Enable preemption again. */
void radix_tree_preload_end(void);
```

- When failure to insert an item into a radix tree can be a significant problem, use “radix_tree_preload”

```
/* Remove the entry at @index from the radix tree rooted at @root.
 * @root:  radix tree root
 * @index: index key
 * Return: The deleted entry, or NULL if it was not present. */
void *radix_tree_delete(struct radix_tree_root *root, unsigned long index);

/* radix_tree_lookup    -    perform lookup operation on a radix tree
 * @root:  radix tree root
 * @index: index key
 * Return: data pointer corresponding to the position @index */
void *radix_tree_lookup(const struct radix_tree_root *root, unsigned long index);

/* radix_tree_lookup_slot    -    lookup a slot in a radix tree
 * @root:  radix tree root
 * @index: index key
 * Return: the slot corresponding to the position @index */
void __rcu **radix_tree_lookup_slot(const struct radix_tree_root *root,
                unsigned long index);
```

```
/* radix_tree_gang_lookup - perform multiple lookup on a radix tree
 * @root:         radix tree root
 * @results:      where the results of the lookup are placed
 * @first_index:  start the lookup from this key
 * @max_items:    place up to this many items at *results
 *
 * Performs an index-ascending scan of the tree for present items.  Places
 * them at *@results and returns the number of items which were placed at
 * *@results. */
unsigned int
radix_tree_gang_lookup(const struct radix_tree_root *root, void **results,
            unsigned long first_index, unsigned int max_items);
```

Linux Radix Tree Example

- The most important user is the page cache
  - Every time, we look up a page in a file, we consult the radix tree to see if the page is already in the cache
  - Use tags to maintain the status of the page, e.g.,
    - » PAGECACHE_TAG_DIRTY
    - » PAGECACHE_TAG_WRITEBACK

```c
/* linux/include/linux/fs.h */

/* inode: a metadata of a file */
struct inode {
    umode_t                 i_mode;
    struct super_block      *i_sb;
    struct address_space    *i_mapping;
};

/* address_space: a page cache of a file */
struct address_space {
    struct inode            *host;      /* owner: inode, block_device */
    struct radix_tree_root  page_tree;  /* radix tree of all pages
                                         * (i.e., page cache of an inode) */
    spinlock_t              tree_lock;  /* and lock protecting it */
};

/* address space in the recent kernel */
struct address_space {
    struct inode        *host;
    struct xarray       i_pages; /* xarray = radix tree + spinlock */
};
```

- Shared memory virtual file system
  - shared memory among process (`shmget()` and `shmat()`)
  - `tmpfs` memory file system

```c
/* linux/fs/inode.c */
/* page_tree is initialized at associated address_space is inialized */
void address_space_init_once(struct address_space *mapping)
{
    INIT_RADIX_TREE(&mapping->page_tree, GFP_ATOMIC | __GFP_ACCOUNT);
}

/* linux/mm/shmem.c */
/* Radix operations are performed on page_tree for file system operations */
static int shmem_add_to_page_cache(struct page *page,
    struct address_space *mapping, pgoff_t index, void *expected)
{
    error = radix_tree_insert(&mapping->page_tree, index, page);
}
```

XArray

- A nicer API wrapper for linux radix tree (merged to 4.19)
- An automatically resizing array of pointers indexed by an unsigned long
- Entries may have up to three tag bits (get/set/clear)
- You can iterate over entires
- You can extract a batch of entires
- Embeds a spinlock
- Loads are store-free using RCU
- Reference: XArray API reference

XArray API

```
#include <linux/xarray.h>

/** Define an XArray */
DEFINE_XARRAY(array_name);
/* or */
struct xarray array;
xa_init(&array);

/** Storing a value into an XArray is done with: */
void *xa_store(struct xarray *xa, unsigned long index, void *entry,
    gfp_t gfp);

/** An entry can be removed by calling: */
void *xa_erase(struct xarray *xa, unsigned long index);

/** Storing a value only if the current value stored there matches old: */
void *xa_cmpxchg(struct xarray *xa, unsigned long index, void *old,
    void *entry, gfp_t gfp);
```

```c
/** Fetching a value from an XArray is done with xa_load(): */
void *xa_load(struct xarray *xa, unsigned long index);

/** Up to three single-bit tags can be set on any non-null XArray
entry; they are managed with: */
void xa_set_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);
void xa_clear_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);
bool xa_get_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);

/** Iterate over present entries in an XArray: */
xa_for_each(xa, index, entry) {
    /* Process "entry" */
}

/** Iterate over marked entries in an XArray: */
xa_for_each_marked(xa, index, entry, filter) {
    /* Process "entry" which marked with "filter" */
}
```

```c
/* linux/include/linux/fs.h */

/* inode: a metadata of a file */
struct inode {
    umode_t                 i_mode;
    struct super_block      *i_sb;
    struct address_space    *i_mapping;
};

/* address_space: a page cache of a file */
struct address_space {
    struct inode            *host;   /* owner: inode, block_device */
    struct xarray           i_pages; /* XArray of all pages
                                      * (i.e., page cache of an inode) */
};
```

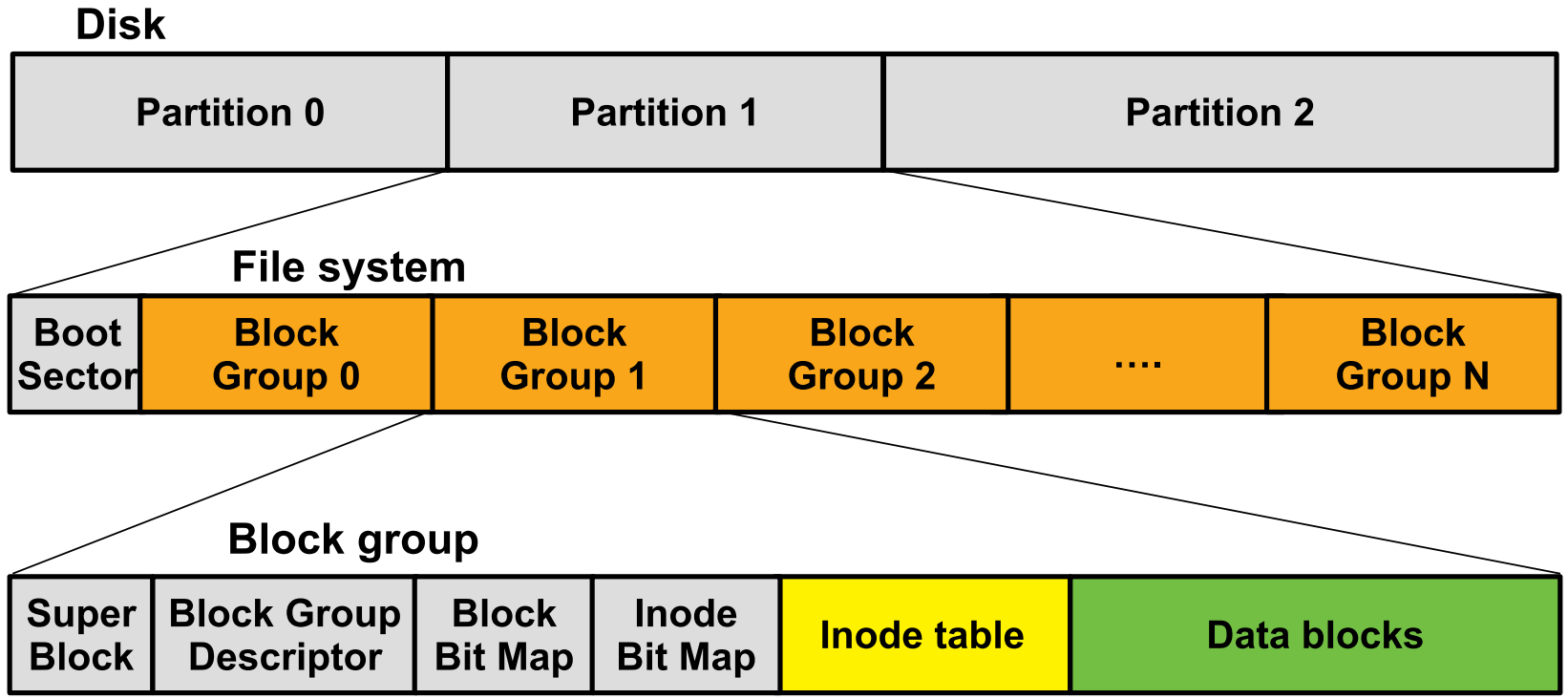
Linux bitmap

- A bit array that consumes one or more “unsigned long”
- Used in many places in the kernel
  - a set of online/offline processors for ssytems which support hot-plug cpu
  - a set of allocated IRQs during initialization of the Linux kernel

```c
/* linux/include/linux/bitmap.h
 * linux/lib/bitmap.c
 * arch/x86/include/asm/bitops.h */

/* Declare an array named 'name' of just enough unsigned longs to
 * contain all bit positions from 0 to 'bits' - 1 */
#define DECLARE_BITMAP(name,bits) \
    unsigned long name[BITS_TO_LONGS(bits)]

/* set_bit - Atomically set a bit in memory
 * @nr: the bit to set
 * @addr: the address to start counting from */
void set_bit(long nr, volatile unsigned long *addr);
void clear_bit(long nr, volatile unsigned long *addr);
void change_bit(long nr, volatile unsigned long *addr);

/* clear nbits from dst */
void bitmap_zero(unsigned long *dst, unsigned int nbits);
void bitmap_fill(unsigned long *dst, unsigned int nbits);
```

```c
/* find_first_bit - find the first set bit in a memory region
 * @addr: The address to start the search at
 * @size: The maximum number of bits to search
 *
 * Returns the bit number of the first set bit.
 * If no bits are set, returns @size.
 */
unsigned long find_first_bit(const unsigned long *addr, unsigned long size);
unsigned long find_first_zero_bit(const unsigned long *addr, unsigned long size);

/* iterate bitmap */
#define for_each_set_bit(bit, addr, size) \
    for ((bit) = find_first_bit((addr), (size));        \
         (bit) < (size);                        \
         (bit) = find_next_bit((addr), (size), (bit) + 1))
#define for_each_set_bit_from(bit, addr, size)   ...
#define for_each_clear_bit(bit, addr, size)      ...
#define for_each_clear_bit_from(bit, addr, size) ...
```

Linux bitmap Example



- Free inode/disk block management in ext2/3/4 file system