Kernel Data Structures II and Kernel Module

Xiaoguang Wang

UIC COMPUTER SCIENCE

Questions from hw1 & hw2

#8: the addr of the 1st element of an array `v.s.` the addr of an array

#4: cast an integer value to a pointer (char array); little-endian

#7: The common way to find the page address of an address

```
#define CONVERT(sz) (((sz)+PGSIZE-1) & ~(PGSIZE-1))
```

Tips on hw3

- Explanations in the code comments
- Brief

Recap

Kernel data structures

- list, hash table, red-black tree

Design patterns of kernel data structures

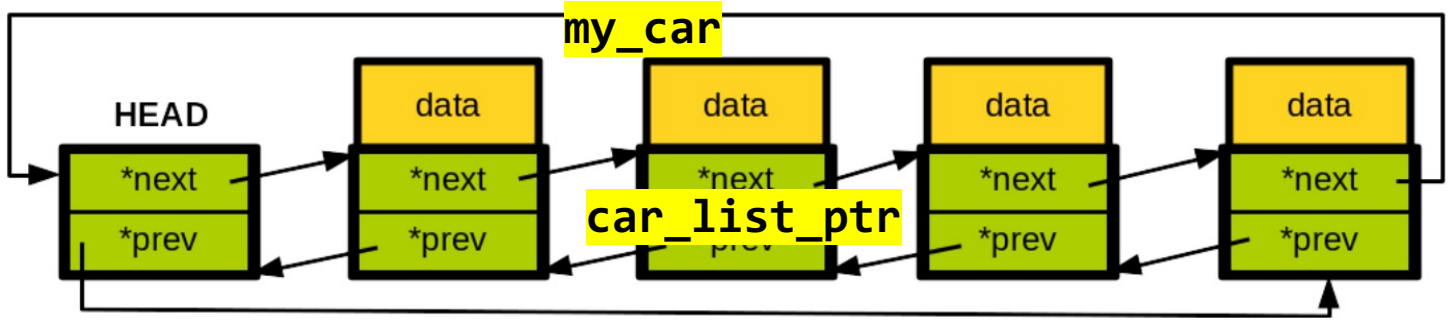
- Embedding its pointer structure
- Toolbox rather than a complete solution for generic service
- Caller locks

Racap: Getting a data element from `list_head`

How to get the pointer of the containing data structure (`struct car`) from its list?

- Use `list_entry(ptr, type, member)`
- Just a pointer arithmetic

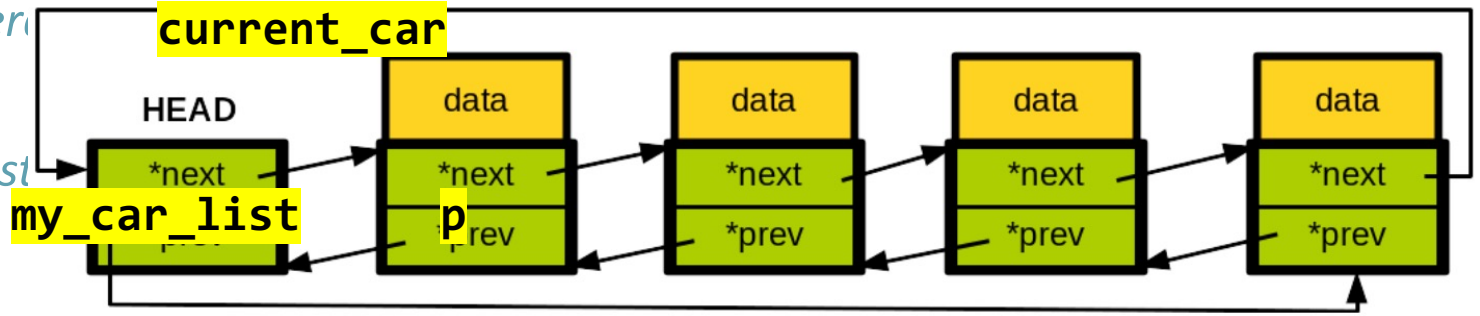
```
struct car *my_car = list_entry(car_list_ptr,
                                struct car, list);
```




```
struct list_head {
    struct list_head *next, *prev;
};
struct car {
    struct list_head list;
    unsigned int max_speed;
    unsigned int price_in_dollars;
};
struct list_head my_car_list;
```

Recap: Iterate over a list: O(n)

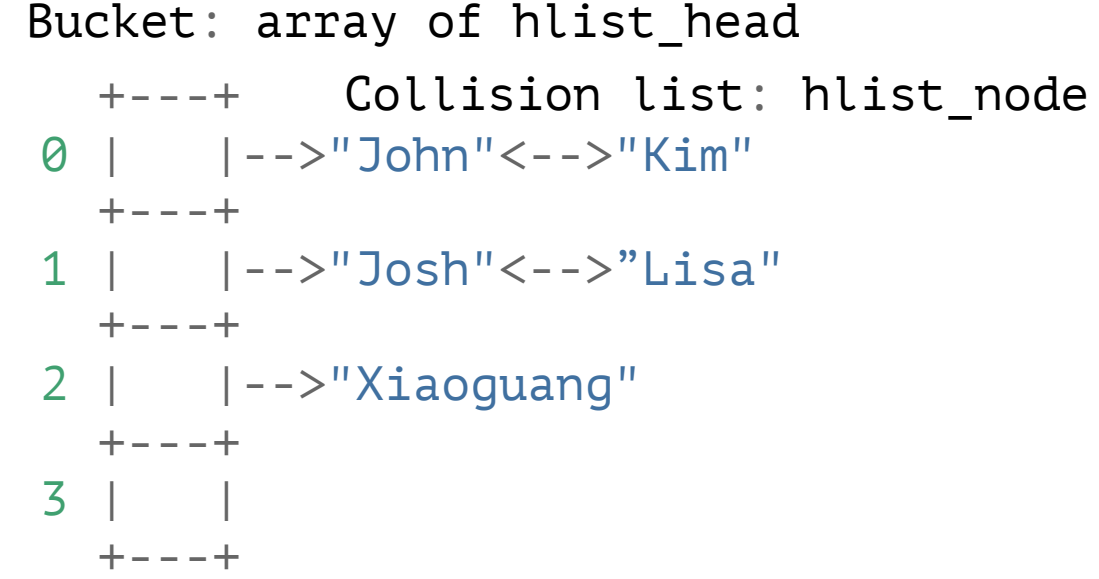
```
/* Temporary variable needed to iter
struct list_head p;
/* This will point to the actual data st
struct car *current_car;
```




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

Recap: Linux hash table

```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 */
};
```



Recap: Linux red-black tree (or rbtree)

```
struct rb_node {                                /* include/linux/rbtree.h,  lib/rbtree.c */
  unsigned long __rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
struct rb_root {                                /* Root of a rbtree */
  struct rb_node *rb_node;
};
#define RB_ROOT (struct rb_root) { NULL, }
#define rb_entry(ptr, type, member) container_of(ptr, type, member)
#define rb_parent(r) ((struct rb_node *)((r)->__rb_parent_color & ~3))
```

Today's agenda

Memory allocation in the kernel

More kernel data structures

- Radix tree
- XArray
- Bitmap

Kernel module

Memory allocation in kernel

Two types of memory allocation functions are provided

- `kmalloc(size, gfp_mask)` - `kfree(address)`
- `vmalloc(size)` - `vfree(address)`

`gfp_mask` is used to specify

- which types of pages can be allocated
- whether the allocator can wait for more memory to be freed

Frequently used `gfp_mask`

- `GFP_KERNEL`: a caller might sleep
- `GFP_ATOMIC`: prevent a caller to sleep → higher chance of failure

kmalloc(size, gfp_mask)

Allocate virtually and physically contiguous memory

- where physically contiguous memory is necessary?
  - E.g., DMA, memory-mapped IO

The maximum allocatable size through one `kmalloc` is limited

- 4MB on x86 (architecture dependent)

```
#include <linux/slab.h>
void my_function()
{
  char *my_string = (char *)kmalloc(128, GFP_KERNEL);
  my_struct *my_struct_ptr = (my_struct *)kmalloc(sizeof(my_struct), GFP_KERNEL);
  /* ... */
  kfree(my_string);
  kfree(my_struct_ptr);
}
```

vmalloc(size)

Allocate memory that is virtually contiguous, but not physically contiguous

No size limit other than the amount of free RAM

Memory allocator might sleep to get more free memory

Unit of allocation is a page (4KB)

```
#include <linux/slab.h>
void my_function()
{
  char *my_string = (char *)vmalloc(128);
  my_struct my_struct_ptr = (my_struct *)vmalloc(sizeof(my_struct));
  /* ... */
  vfree(my_string);
  vfree(my_struct_ptr);
}
```

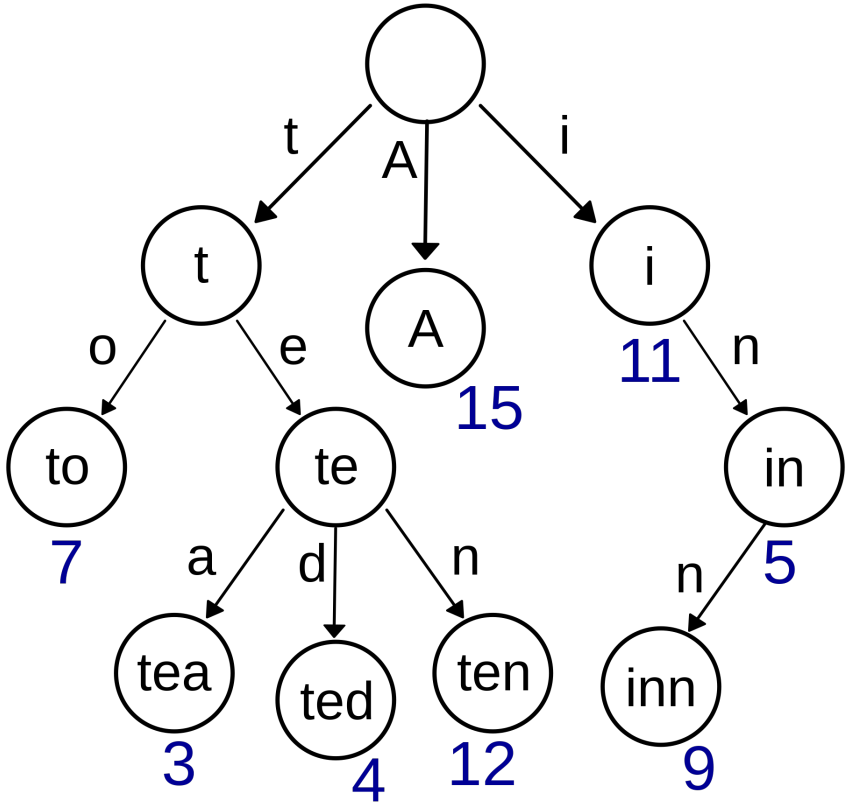
Radix tree (or Trie)

Store key-value pairs (optimized for searching)

Compact prefix tree

All descendants of a node have a common prefix

Values are only associated with leaves



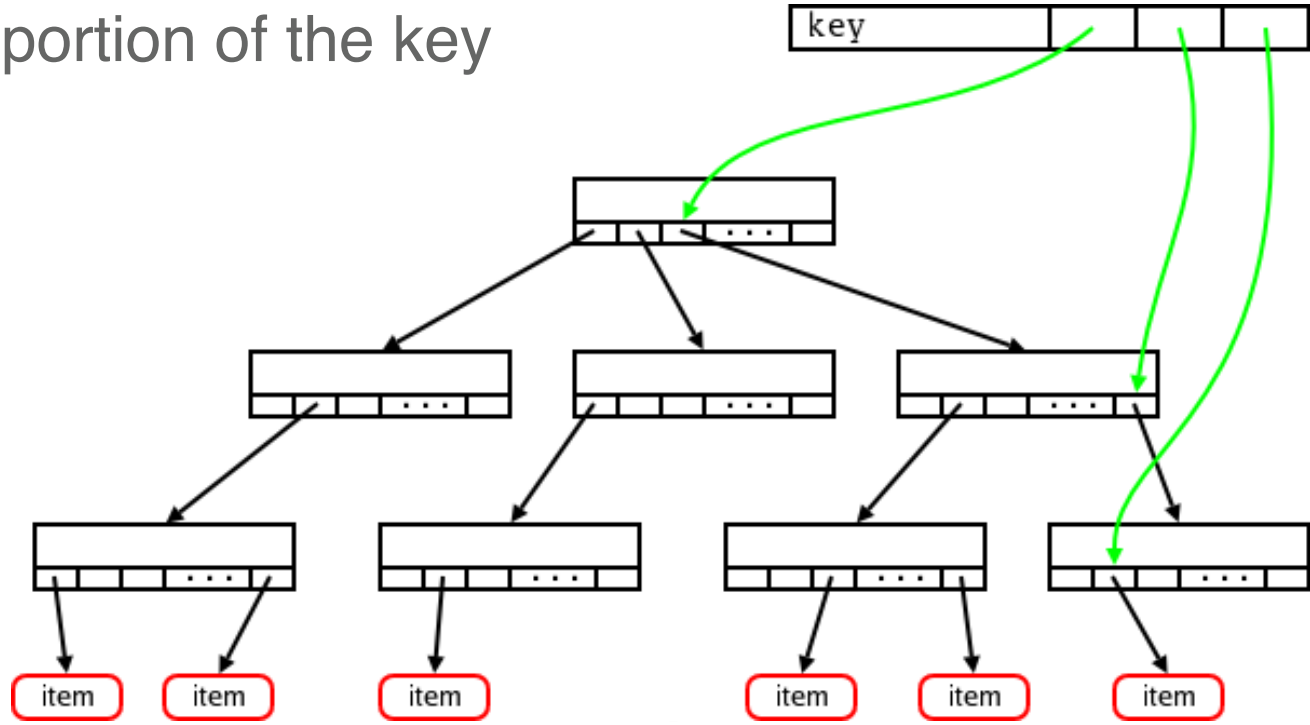
https://en.wikipedia.org/wiki/Trie

Linux radix tree

Mapping between a long integer key and a pointer value

Each node has 64 slots

Slots are indexed by a 6-bit (2^6=64) portion of the key



https://lwn.net/Articles/175432/

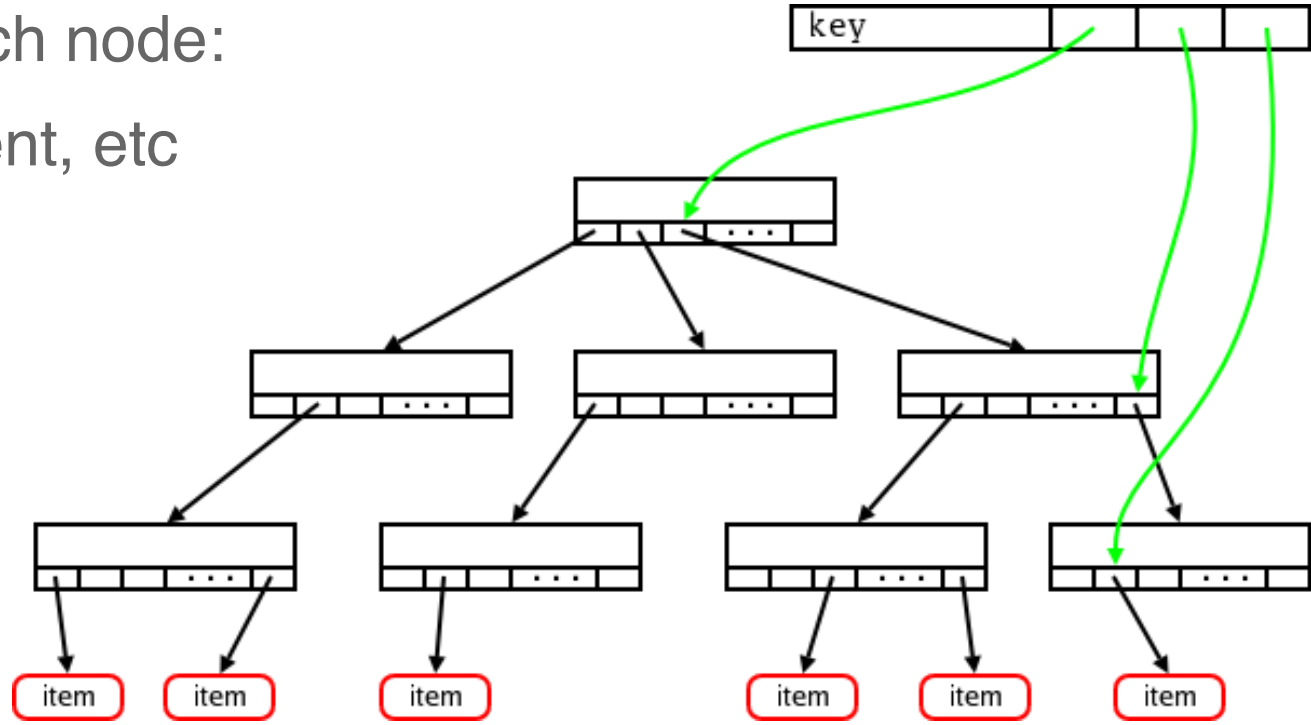
Linux radix tree

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



https://lwn.net/Articles/175432/

Linux radix tree API (old kernel)

```
/* 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];
  /* ... */
};
```

Linux radix tree API (new kernel)

/ Keep unconverted code working */*

```
#define radix_tree_root          xarray
#define radix_tree_node          xa_node
```

```
struct xa_node {
        unsigned char   shift;          /* Bits remaining in each slot */
        unsigned char   offset;         /* Slot offset in parent */
        unsigned char   count;          /* Total entry count */
        unsigned char   nr_values;      /* Value entry count */
        struct xa_node __rcu *parent;   /* NULL at top of tree */
        struct xarray   *array;         /* The array we belong to */
        union {
                struct list_head private_list;  /* For tree user */
                struct rcu_head rcu_head;       /* Used when freeing node */
        };
        void __rcu      *slots[XA_CHUNK_SIZE];
        union {
                unsigned long   tags[XA_MAX_MARKS][XA_MARK_LONGS];
                unsigned long   marks[XA_MAX_MARKS][XA_MARK_LONGS];
        };
};
```

Linux radix tree API

```
/* Declare and initialize a radix tree, gfp_mask: how memory allocations are to be performed. */
RADIX_TREE(name, gfp_mask);

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

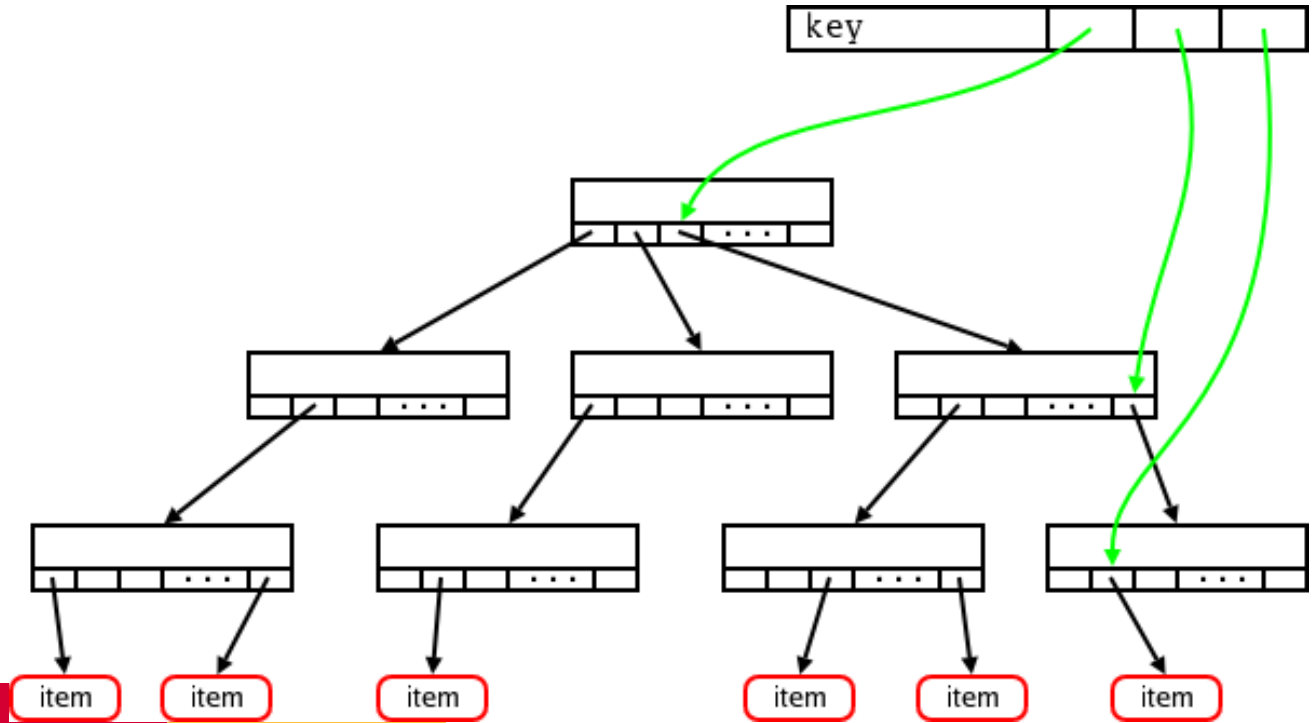
Linux radix tree API

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

/* Remove the entry at index from the radix tree rooted at root */
void *radix_tree_delete(struct radix_tree_root *root, unsigned long
index);

/* radix_tree_lookup - perform lookup operation on a radix tree */
void *radix_tree_lookup(const struct radix_tree_root *root,
unsigned long index);
```

Linux radix tree API

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

```
/* radix_tree_tag_set - set a tag on a radix tree node */
void *radix_tree_tag_set(struct radix_tree_root *root, unsigned
long index, unsigned int tag);

/* radix_tree_tag_clear - clear a tag on a radix tree node */
void *radix_tree_tag_clear(struct radix_tree_root *root,
unsigned long index, unsigned int tag);

/* radix_tree_tag_get - get a tag on a radix tree node */
int radix_tree_tag_get(const struct radix_tree_root *root,
unsigned long index, unsigned int tag);
```

Linux radix tree example

The most important user is the page cache

- page cache: a cache (in RAM) for pages from the disk
- Every time we look up a page of a file, we consult the radix tree to see if the page is already in the cache
- Use tags to maintain the status of page
  (e.g., `PAGECACHE_TAG_DIRTY` or `PAGECACHE_TAG_WRITEBACK`)

Page cache: https://en.wikipedia.org/wiki/Page_cache

Linux radix tree example

```
struct inode {                          /* inode: a metadata of a file */
  umode_t i_mode;
  struct super_block *i_sb;
  struct address_space *i_mapping;
};
struct address_space {                  /* address_space: a page cache of a file */
  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 */
};
/* struct address_space in the recent kernel */
struct address_space {
  struct inode *host;
  struct xarray i_pages;                /* xarray = radix tree + spinlock */
};
```

include/linux/fs.h

rbtree v.s. radix tree v.s. hash table

- Red-black trees are a type of self-balancing binary search tree, used for storing sortable key/value data pairs.
- Radix trees are used to efficiently store sparse arrays and thus use long integer indexes to insert/access/delete nodes.
- Hash tables are not kept sorted to be easily traversed in order and must be tuned for a specific size.

https://docs.kernel.org/core-api/rbtree.html

XArray

A nicer API ~~wrapper~~ replacement for Linux radix tree (since 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 entries

You can extract a batch of entries

Embeds a spinlock

Loads are store-free using RCU

XArray API

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

/** Define an XArray */

DEFINE_XARRAY(array_name);

/* or */

struct xarray array;

xa_init(&array);
```

```c
/** 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);
```

XArray API

```
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" */
}
```

Linux Xarray example (v6.1)

```
/* include/linux/fs.h */
struct inode {                          /* inode: a metadata of a file */
   umode_t i_mode;
   struct super_block *i_sb;
   struct address_space *i_mapping;
};
/* struct 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*/
};
```

Linux bitmap

A bit array that consumes one or more `unsigned long`

Using in many places in kernel

- a set of online/offline processors for systems which support hot-plug CPUs
- a set of allocated IRQs during initialization of the Linux kernel

Linux bitmap

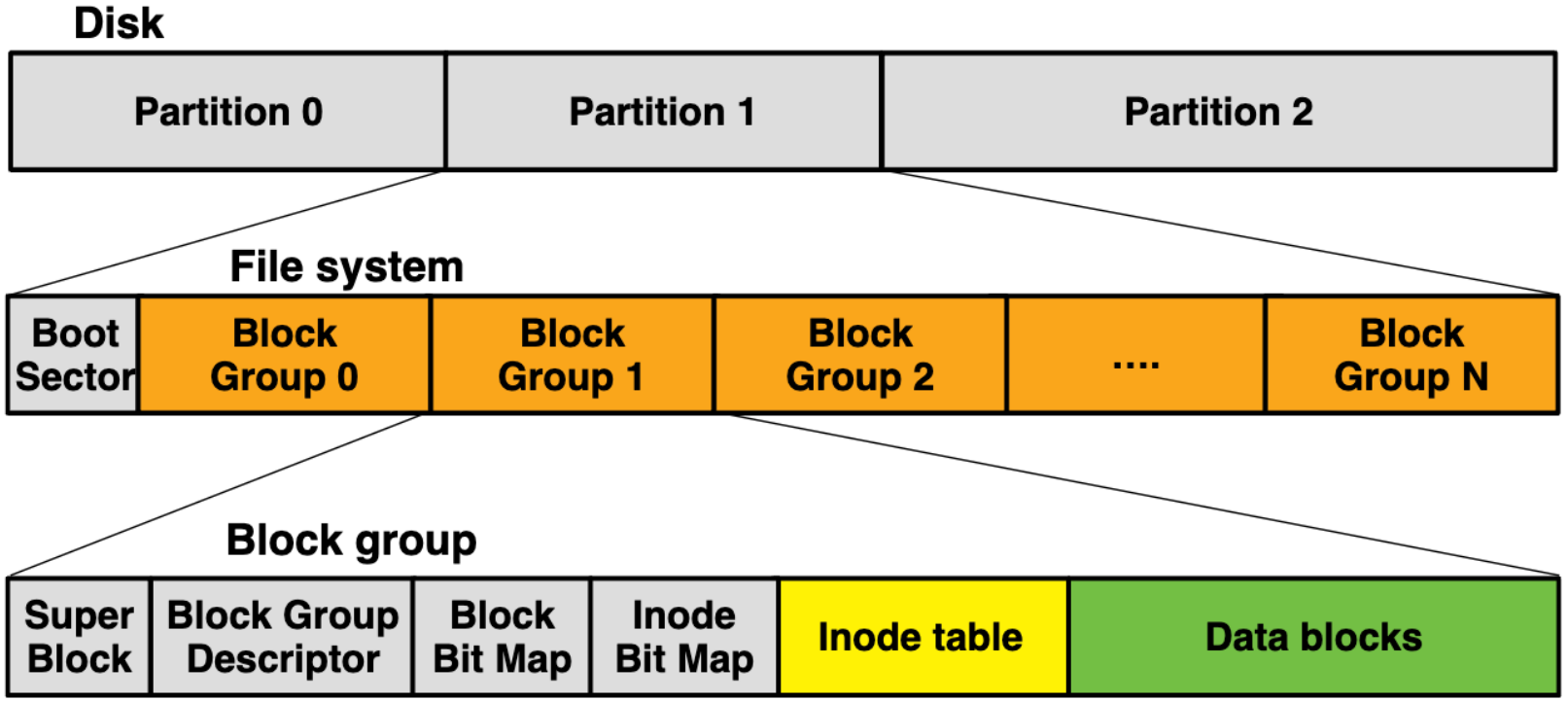
```
/* include/linux/types.h */
#define DECLARE_BITMAP(name,bits) \
        unsigned long name[BITS_TO_LONGS(bits)]
void set_bit(long nr, volatile unsigned long *addr);
void clear_bit(long nr, volatile unsigned long *addr);
void change_bit(long nr, volatile unsigned long *addr);

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

Linux bitmap

```
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



Using these data structure

Read kernel source code, definition first then usage

- Explanations above the definition

vim + cscope

```
:cs find g xa_for_each
:cs find s xa_for_each
```

Kernel modules

Modules are pieces of kernel code that can be **dynamically loaded and unloaded at runtime** → No need to reboot

Appeared in Linux 1.2 (1995)

Numerous Linux features can be compiled as modules

- Selection in the configuration .config file

```
# linux/.config
# CONFIG_XEN_PV is not set
CONFIG_KVM_GUEST=y  # built-in to kernel binary executable, vmlinux
CONFIG_XFS_FS=m  # kernel module
```

Benefit of kernel modules

No reboot → saves a lot of time when developing/debugging

No need to compile the entire kernel

Saves memory and CPU time by running on-demand

No performance difference between module and built-in kernel code

Help identifying buggy code

- E.g., identifying a buggy driver compiled as a module by selectively running them

Write a kernel module

Module is linked against the entire kernel

Module can access all the kernel global symbols

- `EXPORT_SYMBOL(function or variable name)`

To avoid namespace pollution and involuntary reuse of variables names

- Put a prefix of your module name to symbols: `my_module_func_a()`
- Use `static` if a symbol is not global

Kernel symbols list are at `/proc/kallsyms`

Write a kerne

```c
#include <linux/module.h>        /* Needed by all modules */
#include <linux/kernel.h>        /* KERN_INFO */
#include <linux/init.h>          /* Init and exit macros */
static int answer = 42;
static int __init lkp_init(void) {
    printk(KERN_INFO "Module loaded ...\n");
    printk(KERN_INFO "The answer is %d ...\n", answer);
    return 0; /* return 0 on success, something else on error */
}
static void __exit lkp_exit(void) {
    printk(KERN_INFO "Module exiting ...\n");
}
module_init(lkp_init); /* lkp_init() will be called at loading the module */
module_exit(lkp_exit); /* lkp_exit() will be called at unloading the module */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Xiaoguang Wang <xgwang9@uic.edu>");
MODULE_DESCRIPTION("A simple kernel module");
```

Build a kernel module

Source code of a module is out of the kernel source

Put a `Makefile` in the module source directory

After compilation, the compiled module is the file with .ko extension

```
# let's assume the module C file is named lkp.c
obj-m := lkp.o
# obj-m += lkp2.o              # add multiple files if necessary
CONFIG_MODULE_SIG=n
KDIR := /path/to/kernel/sources/root/directory
# KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
all: lkp.c                     # add lkp2.c if necessary
	make -C $(KDIR) M=$(PWD) modules
clean:
	make -C $(KDIR) M=$(PWD) clean
```

Launch a kernel module

Needs `root` privileges because you are executing kernel code!

Loading a kernel module with `insmod`

- `sudo insmod file.ko`
- Module is loaded and the init function is executed

Note that a module is compiled against a specific kernel version and will not load on another kernel

- This check can be bypassed through a mechanism called `modversions` but it can be dangerous

Launch a kernel module

Remove the module with rmmod

- `sudo rmmod file`
- or `sudo rmmod file.ko`
- Module exit function is called before unloading

`make modules_install` from the kernel sources installs the modules in a standard location

- `/lib/modules/<kernel version>/`

Launch a kernel module

These installed modules can be loaded using modprobe

- `sudo modprobe <module name>` ← no need to give a file name
- `find /lib/modules/$(uname -r) -type f -name '*.ko' | less`
- E.g., `modprobe 9p`

Unload a module using `modprobe -r <module name>`

```
/lib/modules/5.15.0-91-generic/kernel/net/mac80211/mac80211.ko
/lib/modules/5.15.0-91-generic/kernel/net/bpfilter/bpfilter.ko
/lib/modules/5.15.0-91-generic/kernel/net/kcm/kcm.ko
/lib/modules/5.15.0-91-generic/kernel/net/nfc/nfc.ko
/lib/modules/5.15.0-91-generic/kernel/net/nfc/hci/hci.ko
/lib/modules/5.15.0-91-generic/kernel/net/nfc/nfc_digital.ko
```

Launch a kernel module

Contrary to `insmod`, `modprobe` handles module dependencies

- Dependency list generated in `/lib/modules/<kernel version>/modules.dep`

Such installed modules can be loaded automatically at boot time by editing `/etc/modules` or the files in `/etc/modprobe.d`

```
kernel/fs/xfs/xfs.ko: kernel/lib/libcrc32c.ko
kernel/fs/9p/9p.ko: kernel/net/9p/9pnet.ko kernel/fs/fscache/fscache.ko
kernel/fs/afs/kafs.ko: kernel/net/rxrpc/rxrpc.ko kernel/net/ipv6/ip6_udp
kernel/fs/nilfs2/nilfs2.ko:
kernel/fs/befs/befs.ko:
```

Module parameters

command line arguments for module

- `sudo insmod lkp.ko int_param=12 string_param="hello"`

```c
#include <linux/module.h>
static int int_param = 42; /* default value */
static char *string_param = "default value";

module_param(int_param, int, 0);
MODULE_PARM_DESC(int_param, "A sample integer kernel module parameter");
module_param(string_param, charp, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(string_param, "Another parameter, a string");

static int __init lkp_init(void)
{
  printk(KERN_INFO "Int param: %d\n", int_param);
  printk(KERN_INFO "String param: %s \n", string_param);
}
```

Get module information

`modinfo [module name | file name]`

`lsmod`: list currently running modules

```
$ modinfo lkp.ko
filename:       /home/xiaoguang/lkp/kern_mod/lkp.ko
description:    A simple kernel module
author:         Xiaoguang Wang <xgwang9@uic.edu>
license:        GPL
srcversion:     1008BB92F3162284F0A4C58
depends:
name:           lkp
vermagic:       5.15.0-89-generic SMP mod_unload modversions aarch64
```

Next step

Take hw4 (Linux kernel module and linked list)

- Due: next Friday (Feb 2nd)

Further reading

The Linux Kernel Module Programming Guide

LKD3: Chap 17: Devices and Modules

LWN: Tree I: Radix trees

LWN: The XArray data structure

Bit arrays and bit operations in the Linux kernel

Next lecture

Kernel debugging techniques