CS 5264/4224; ECE 5414/4414
(Advanced) Linux Kernel Programming
Lecture 4

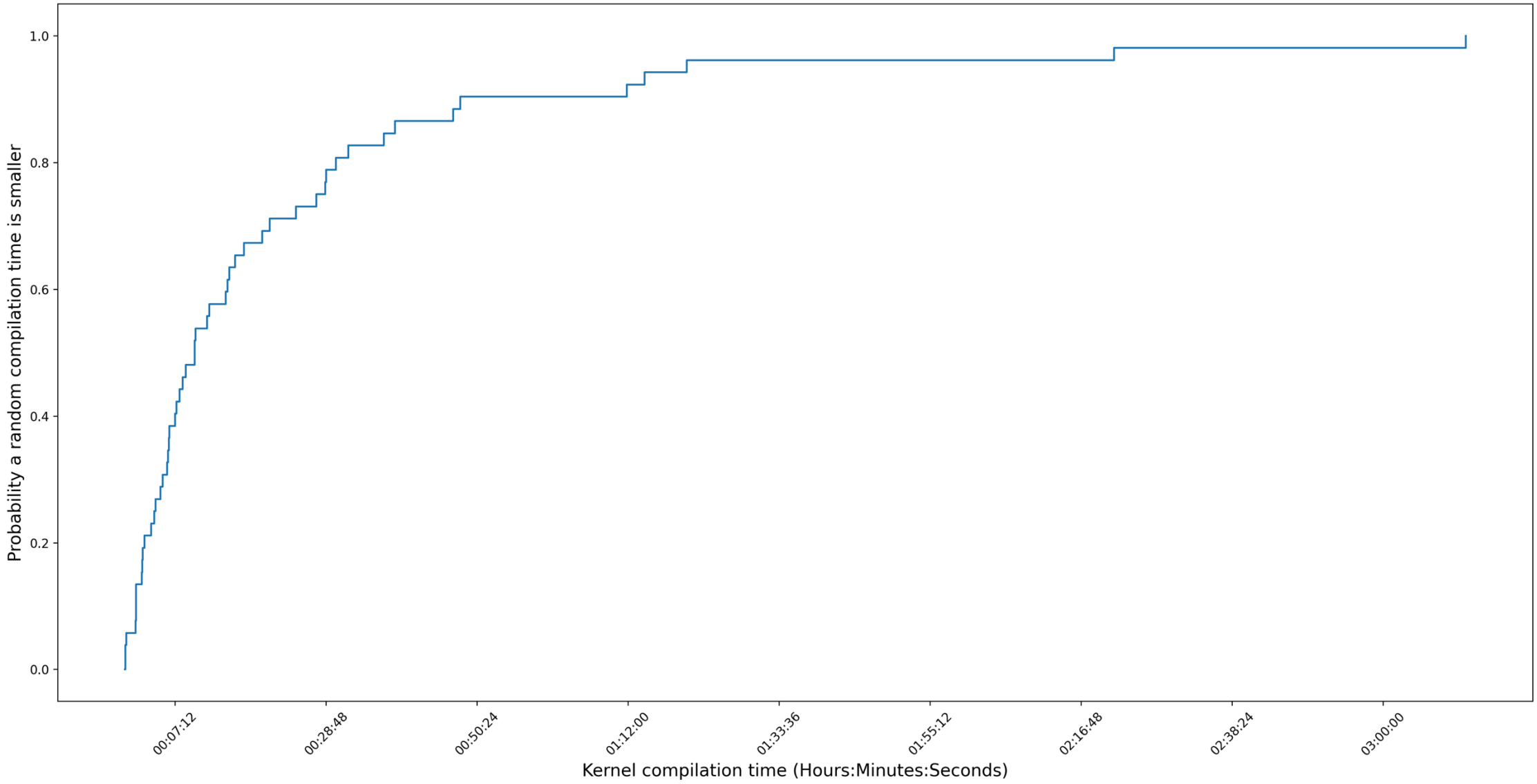
Kernel Modules & Data Structures

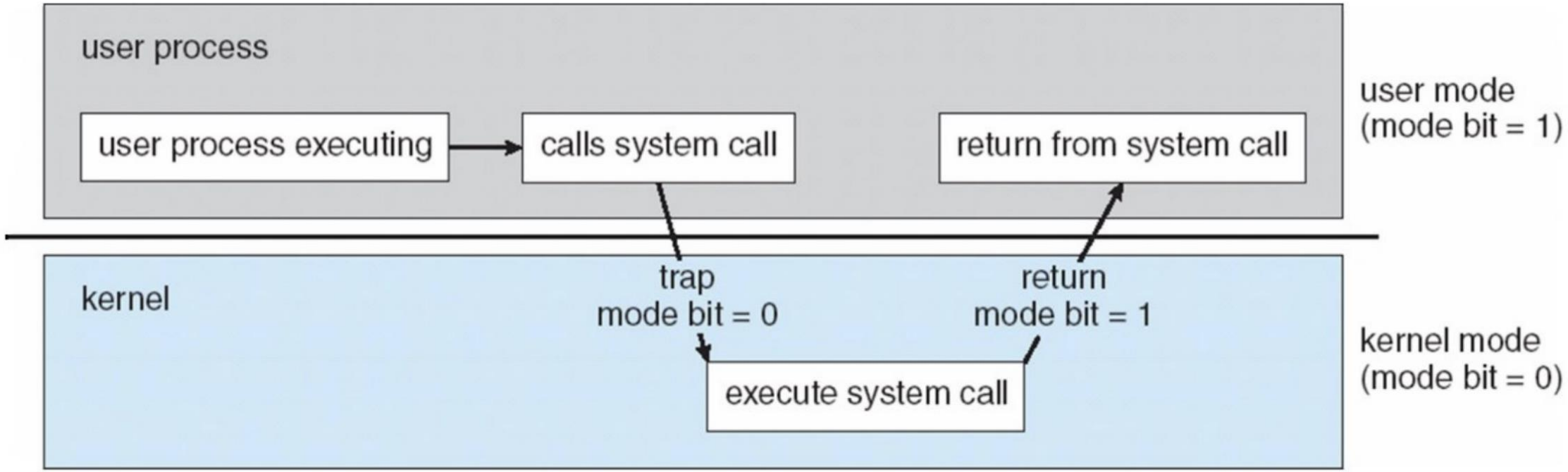
February 4, 2025

Huaicheng Li

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

Ex0 Kernel Compilation Time CDF





From COMS W4118 Operating Systems I

- Example: syscall implemented in linux sources in linux/my_syscall/my_func.c
- Create a linux/my_syscall/Makefile
  - obj-y += my_func.o
- Add “my_syscall” in linux/Makefile
  - core-y += kernel/ certs/ mm/ fs/ .... my_syscall/

Why Not to Add a New Syscall

- Pros
  - Easy to implement and use, fast
- Cons
  - Need an official syscall number
  - Interface cannot change after implementation, need to maintain it forever
  - Must support every architecture
  - Probably too much work for small exchanges of information, easily an overkill!
- Alternative
  - Create a device node and “read()” / “write()”
  - Use “ioctl()”

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

- Memory allocation APIs
- Kernel module basics
- Data structures

- Two types of memory allocation functions are provided
  - kmalloc(size, gfp_mask), kfree(addr)
  - vmalloc(size), vfree(addr)
- gfp_mask is used to specify
  - which types of pages can be allocated
  - whether the allocator can wait for more memory to be free
- Frequently used gfp_mask
  - GFP_KERNEL: a caller might sleep
  - GFP_ATOMIC: a caller will not sleep → higher chance of failure

kmalloc(size, gfp_mask)

- Allocate virtually and physically contiguous memory
  - where physically contiguous memory is necessary
  - e.g., DMA, MMIO, performance in accessing
- The maximum allocatable size through one kmalloc() call is limited
  - e.g., 4MB on x86 (arch 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);
```

vmalloc(size)

- Allocate memory that is virtually contiguous, but not physically contiguous
- No size limit other than the amount of free RAM
  - swapping is not supported for kernel memory
- 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);
```

Kernel Modules

- Modules are pieces of kernel code that can be dynamically loaded and unloaded at runtime
  - No need for 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 code
- No need to compile the entire kernel
- Save 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

Writing a Kernel Module

- Module is linked against the entire kernel
- Module can access all of the kernel global symbols
  - EXPORT_SYMBOL(function or variable name)
- To avoid namespace pollution and involuntary reuse of variable names
  - Put 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*

```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");
```

Building 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
```

Launching a Kernel Module

- Needs root privilege becauses you are executing kernel code
- Loading a kernel module with insmod / modprobe
  - sudo insmod file.ko
  - Module is loaded and 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

- Remove the module with rmmod / modprobe –r
  - sudo rmmod file
  - 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>/ ...

- Modprobe
  - sudo modprobe <module name> ← no need to give the file name
  - Contrary to insmod, modprobe handles module dependencies
    - » dependency list generated in /lib/modules/<kernel version>/modules.dep
  - Unload a module using modprobe –r <module name>
  - Sunc installed modules can be loaded automatically at boot time by editing /etc/modules or the files in /etc/modprobe.d

Module Parameters

- command line arguments for module

```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);
        /* ... */
}
```

Module metadata

- modinfo [module name | file name]
- lsmod: list currently running modules

```
→  hello-world git:(master) modinfo ./hello-world.ko
filename:       /home/femu/git/lkm/hello-world/./hello-world.ko
alias:          LKP a simple example
description:    LKP hello world kernel module
version:        v0.1
author:         Huaicheng Li<huaicheng@cs.vt.edu>
license:        GPL v2
srcversion:     254A10B34EDF8BC76C4CBCA
depends:
name:           hello_world
retpoline:      Y
vermagic:       6.12.0-huaicheng-lkp-gadc218676eef-dirty SMP preempt mod_unload
```

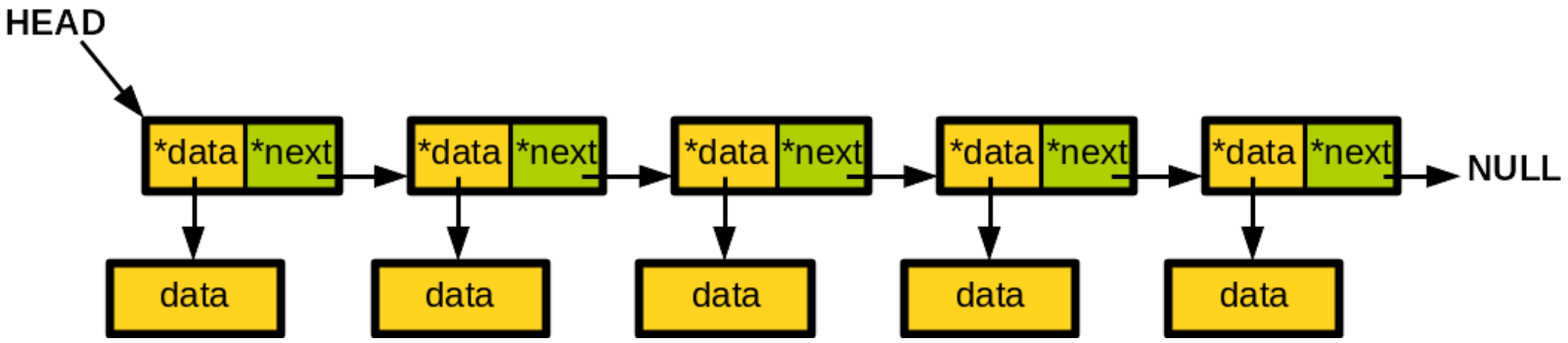
Further Readings

- The Linux Kernel Module Programming Guide
- Passing Command Line Arguments to a Module
- Building External Modules

- Kernel Data Structure!

- Essential kernel data structures
  - list, hash table, red-black tree, ...
- Design patterns of kernel data structures
  - Embedding its pointer structure
  - Too box rather than a complete solution for generic service
  - Caller locks (not thread-safe)

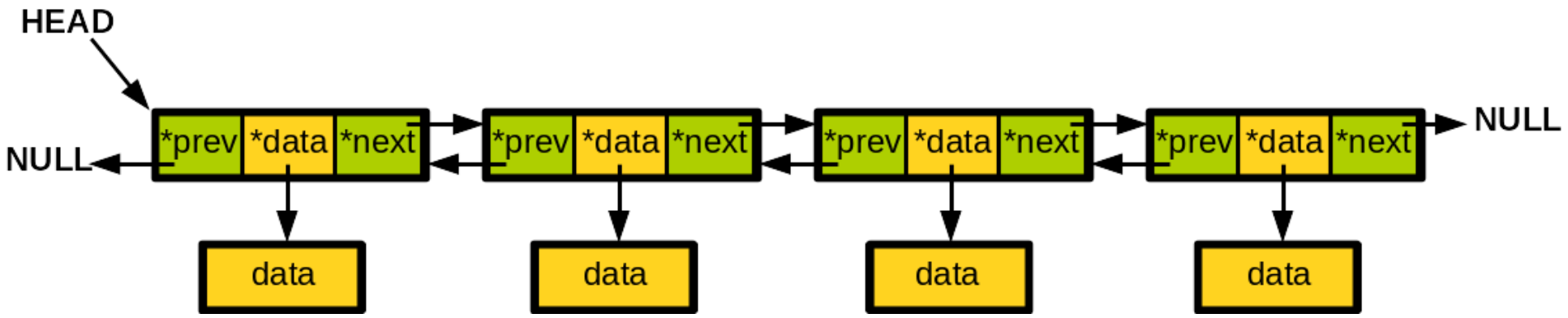
Singly Linked List (CS101)




```
struct my_list_element {
    void *data; /* pointer to generic data */
    struct my_list_element *next;
}
```

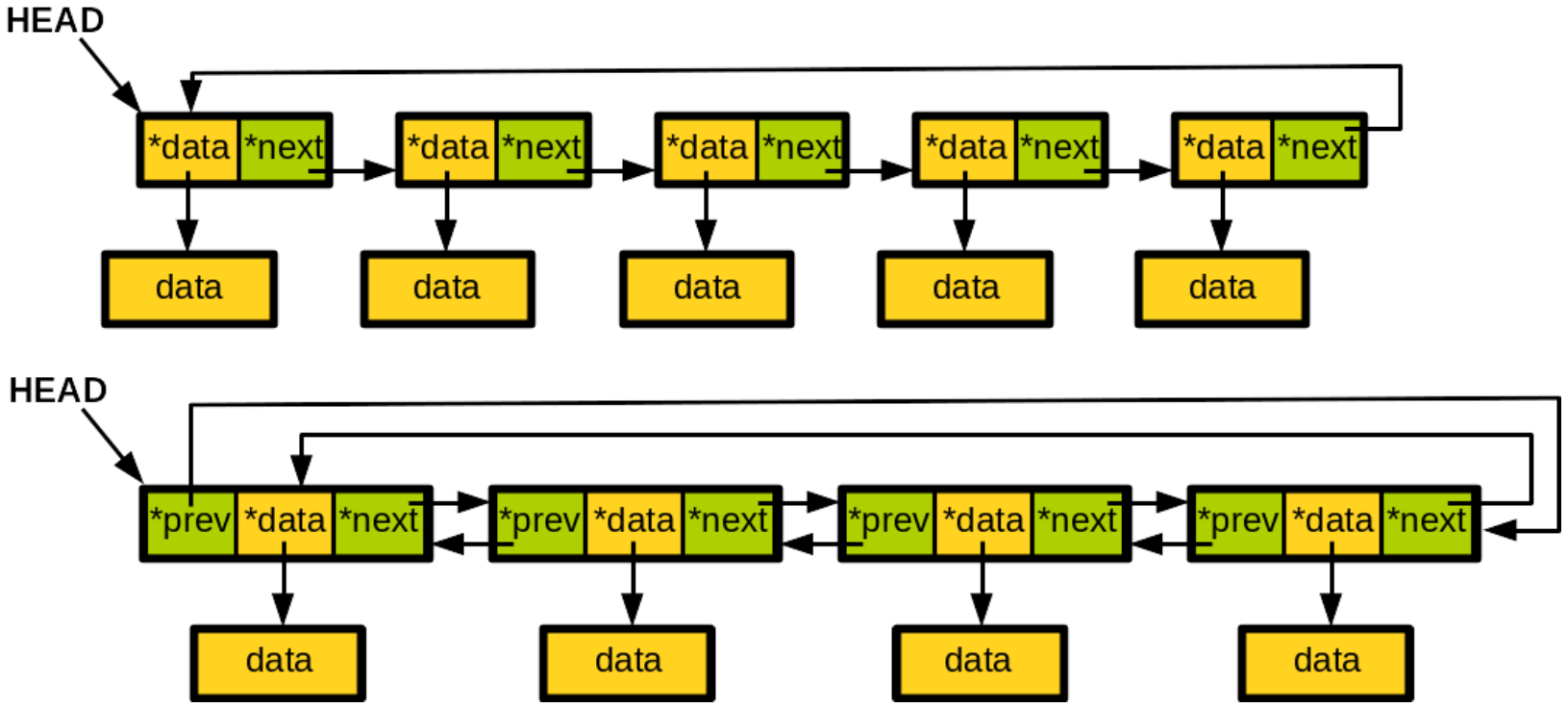
- Starts from HEAD and terminates at NULL
- Traverses forward only
- When empty, HEAD is NULL

Doubly Linked List (CS101)



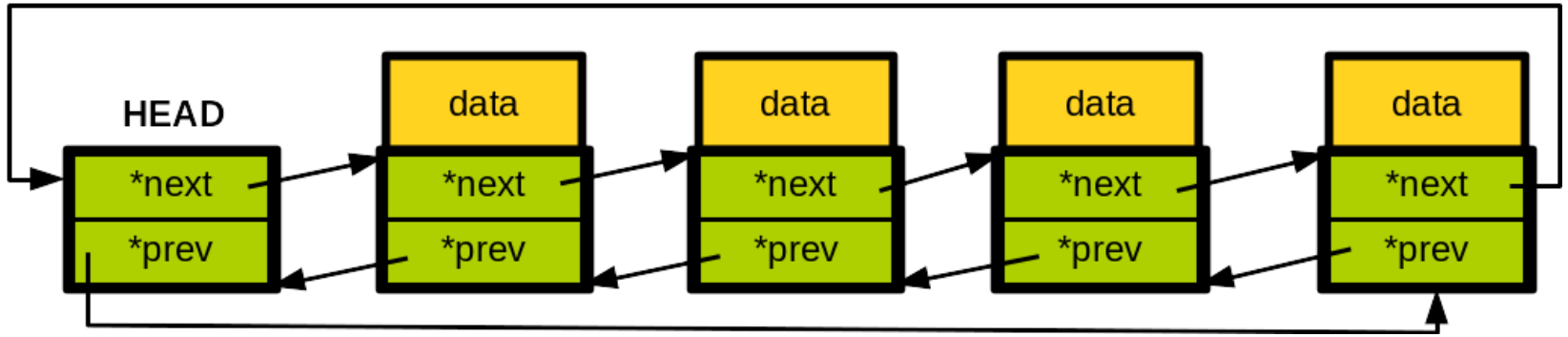
- Starts from HEAD and terminates at NULL
- Traverses forward and backward
- When empty, HEAD is NULL

Circular Linked List (CS1010



- Starts from HEAD and terminates at HEAD
- When empty, HEAD is NULL
- Easy to insert a new element at the end of a list

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

Linux Linked List

- A circular doubly linked list
- Tow differences from the typical design
  - Embedding a linked list node in the structure
  - Using a sentinel node as a list header
- *linux/include/linux/list.h*

```
static inline void __list_del(struct list_head * prev, struct list_head * next)
{
    next->prev = prev;
    WRITE_ONCE(prev->next, next);
}
static inline void __list_del_entry(struct list_head *entry)
{
    __list_del(entry->prev, entry->next);
}
```

```c
struct list_head {                  /* kernel linked list data structure */
    struct list_head *next, *prev;
};

struct car {
    struct list_head  list; /* add list_head instead of prev and next */
    unsigned int      max_speed; /* put data directly */
    unsigned int      drive_wheen_num;
    unsigned int      price_in_dollars;
};

struct list_head my_car_list; /* HEAD is also list_head */
```

- “struct list_head” is the key data structure
- ”list_head” is embedded in the data structure
- Start of a list is also “list_head”, my_car_list → sentinel node

Getting Data from list_head

- How to get the pointer of "struct car" from its list
  - use list_entry(ptr, type, member)
  - just a pointer arithmetic

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

/**
 * list_entry - get the struct for this entry
 * @ptr:    the &struct list_head pointer.
 * @type:   the type of the struct this is embedded in.
 * @member: the name of the list_head within the struct.
 */
#define list_entry(ptr, type, member) container_of(ptr, type, member)
#define container_of(ptr, type, member) ({ \
        const typeof( ((type *)0)->member ) *__mptr = (ptr); \
        (type *)( (char *)__mptr - offsetof(type,member) );})
#define offsetof(TYPE, MEMBER)  ((size_t)&((TYPE *)0)->MEMBER)
```

Defining a List

```
struct car *my_car = kmalloc(sizeof(*my_car), GFP_KERNEL);
my_car->max_speed = 150;
my_car->drive_wheel_num = 2;
my_car->price_in_dollars = 10000.0;
INIT_LIST_HEAD(&my_car->list);  /* initialize an element */

struct car my_car {
    .max_speed = 150,
    .drive_wheel_num = 2,
    .price_in_dollars = 10000,
    .list = LIST_HEAD_INIT(my_car.list), /* initialize an element */
}

LIST_HEAD(my_car_list); /* initialize the HEAD of a list */
```

- Initializing a list_head
  - list_head->prev = &list_head
  - list_head->next = &list_head

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)

Iterating while Removing Entries

```
#define list_for_each_safe(pos, next, head) ...
#define list_for_each_entry_safe(pos, next, head, member) ...

/* This will point on the actual data structures
 * (struct car) during the iteration: */
struct car *current_car, *next;
list_for_each_entry_safe(current_car, next, my_car_list, list) {
    printk(KERN_INFO "Price: %lf\n", current_car->price_in_dollars);
    list_del(current_car->list);
    kfree(current_car); /* if dynamically allocated using kmalloc */
}
```

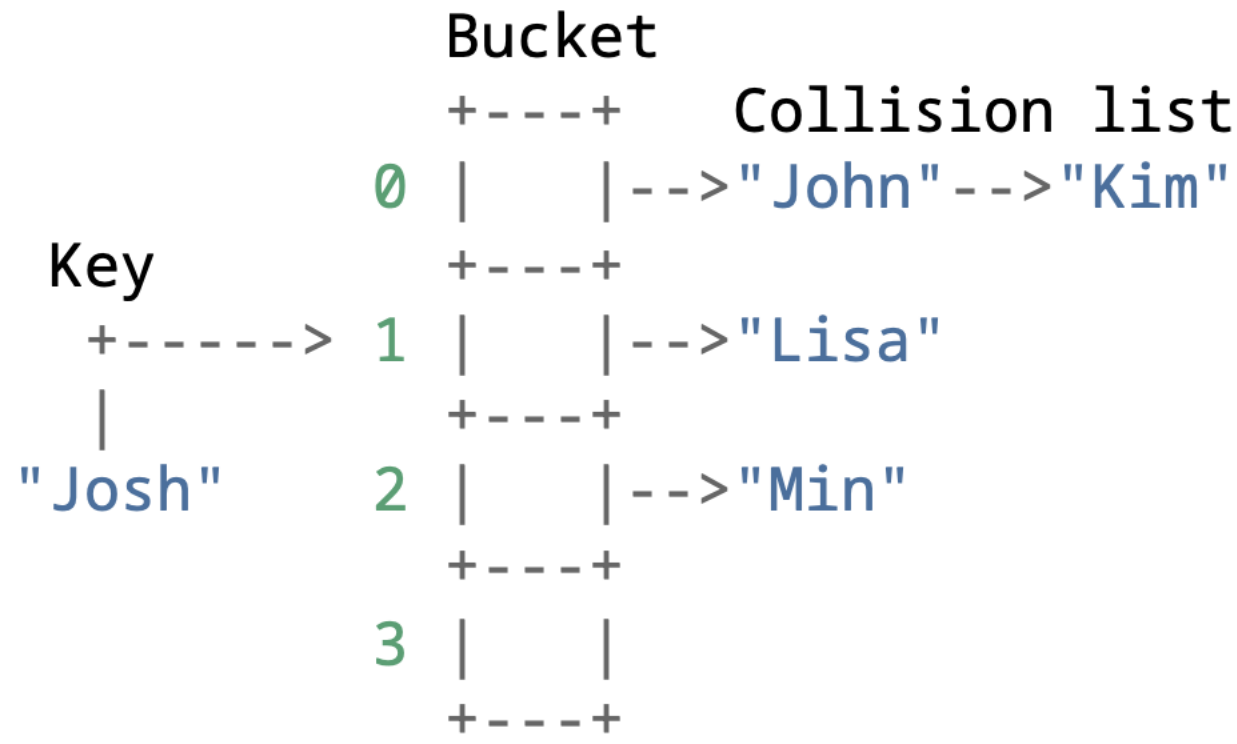
- For each iteration, next points to the next node
  - Can safely remove the current node
  - Otherwise, can cause a use-after-free bug

Linked List Usage in the Kernel

- Kernel code makes extensive use of linked lists
  - a list of threads under the same parent PID
  - a list of superblocks of a file systems
  - ...

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);
    /* ... */
}
```