Virtualization and Kernel Modules

Timothy Borunov
Ph.D. Student, Computer Science
Slides by Shriram Raja

BOSTON UNIVERSITY

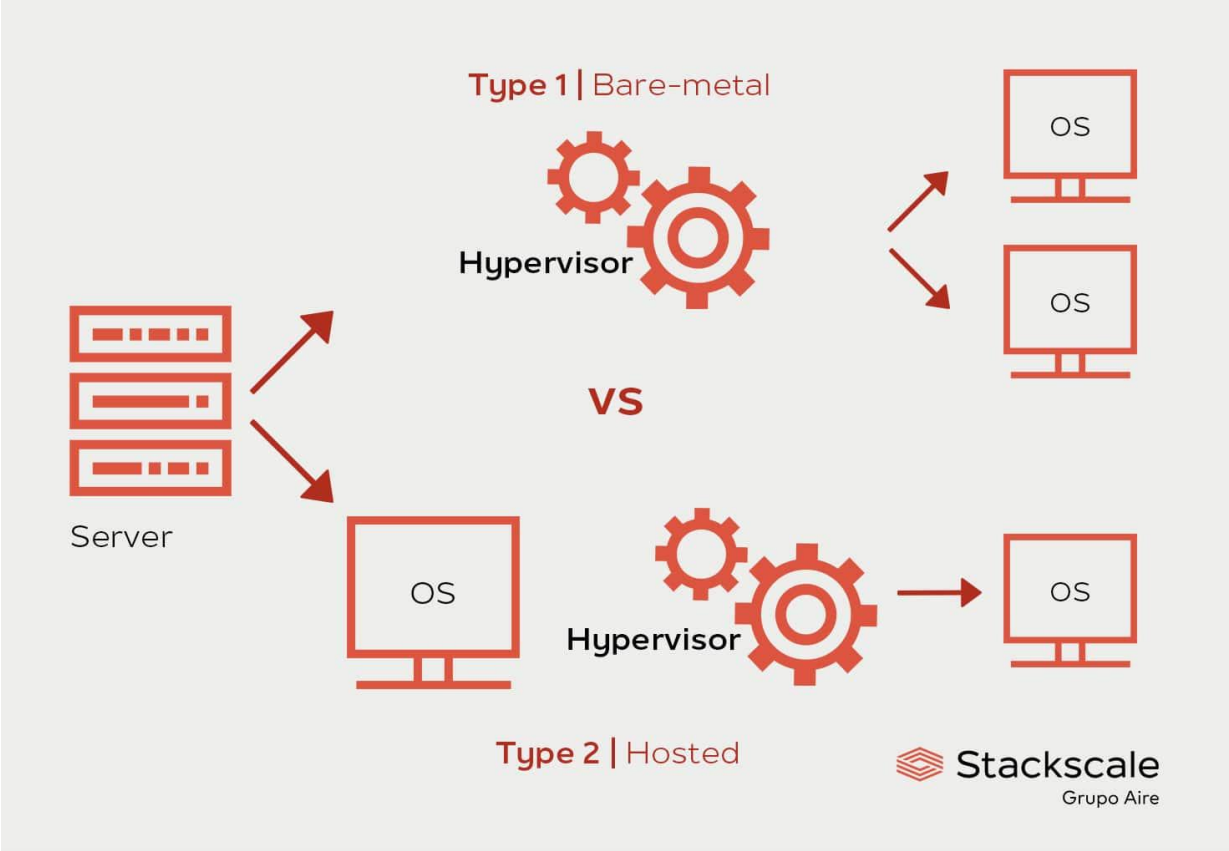
Introduction

- Me:
  - Bachelor's degree in Computer Engineering at BU
  - Ph.D. in Computer Engineering (ongoing)
  - Research: General Operating Systems; Currently Real-time Applications
  - Labs: will cover any topics/practical aspects related to assignments

Virtualization

- Kernel: essential components to keep the system running; OS: kernel + user interface
- Virtualization:
  - imitating underlying hardware
  - Managers: VirtualBox, Quest-V
- Emulation: imitating different hardware

Virtualization



Virtualization

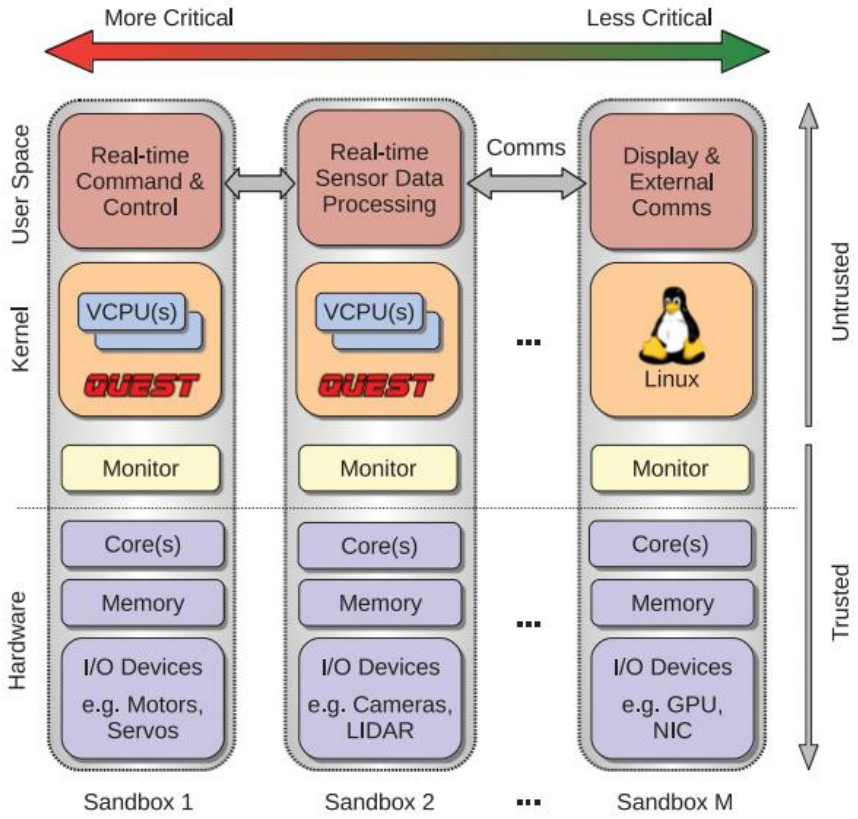


Fig. 1. Example Quest-V architecture overview.

Kernel Modification

- Direct Source code modification: time consuming, error prone
  - Null Pointer De-referencing
- Kernel Modules: quick and suitable to customize a system for an individual or a small group; kernel version dependent; error prone
- BPF: user code is passed through a verifier and then inserted into the kernel; suitable for commercial systems; can be kernel independent
  - Has its own set of security issues

Kernel Modules - Getting Started

```
/*
 *   test module.
 */

#include <linux/module.h>
#include <linux/config.h> // Note: Remove this line if it causes problems with your build
#include <linux/init.h>

MODULE_LICENSE("GPL");

static int __init initialization_routine(void) {

  printk ("Hello, world!\n");

  return 0;
}

static void __exit cleanup_routine(void) {

  printk ("Unloading module!\n");
}

module_init(initialization_routine);
module_exit(cleanup_routine);
```

User-module communication

- Register the module so that user process can access it: proc filesystem
- While registering the module, you can specify the operations allowed on the module

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **);
};
```

User-module communication

```c
static int pseudo_device_ioctl(struct inode *inode, struct file *file,
                               unsigned int cmd, unsigned long arg);

static struct file_operations pseudo_dev_proc_operations;

static struct proc_dir_entry *proc_entry;

static int __init initialization_routine(void) {
  printk("<1> Loading module\n");

  pseudo_dev_proc_operations.ioctl = pseudo_device_ioctl;

  /* Start create proc entry */
  proc_entry = create_proc_entry("ioctl_test", 0444, NULL);
  if(!proc_entry)
  {
    printk("<1> Error creating /proc entry.\n");
    return 1;
  }

  //proc_entry->owner = THIS_MODULE; <-- This is now deprecated
  proc_entry->proc_fops = &pseudo_dev_proc_operations;

  return 0;
}
```

User-module communication

```
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
```

inode: fs specific artifact

```
/***
 * ioctl() entry point...
 */
static int pseudo_device_ioctl(struct inode *inode, struct file *file,
                               unsigned int cmd, unsigned long arg)
{
  struct ioctl_test_t ioc;

  switch (cmd){

  case IOCTL_TEST:
    copy_from_user(&ioc, (struct ioctl_test_t *)arg,
                   sizeof(struct ioctl_test_t));
    printk("<1> ioctl: call to IOCTL_TEST (%d,%c)!\n",
     ioc.field1, ioc.field2);

    my_printk ("Got msg in kernel\n");
    break;

  default:
    return -EINVAL;
    break;
  }

  return 0;
}
```

ioctl call

```
#define _IO(type,nr)            _IOC(_IOC_NONE,(type),(nr),0)
#define _IOR(type,nr,size)      _IOC(_IOC_READ,(type),(nr),(_IOC_TYPECHECK(size)))
#define _IOW(type,nr,size)      _IOC(_IOC_WRITE,(type),(nr),(_IOC_TYPECHECK(size)))
#define _IOWR(type,nr,size)     _IOC(_IOC_READ|_IOC_WRITE,(type),(nr),(_IOC_TYPECHECK(size)))
```

```c
#define IOCTL_TEST _IOW(0, 6, struct ioctl_test_t)

int main () {

  /* attribute structures */
  struct ioctl_test_t {
    int field1;
    char field2;
  } ioctl_test;

  int fd = open ("/proc/ioctl_test", O_RDONLY);

  ioctl_test.field1 = 10;
  ioctl_test.field2 = 'a';

  ioctl (fd, IOCTL_TEST, &ioctl_test);

  return 0;
}
```

Primer

- Copy information from the kernel to the user
- Detect a keypress
- Collect the scan code of the key press
- Way to make your function execute whenever the keyboard interrupt occurs
- Everything you need is in the assignment page; ask questions if unclear.