Εισαγωγή στον Προγραμματισμό του Πυρήνα του Linux

Εργαστήριο Λειτουργικών Συστημάτων
7ο εξάμηνο, ΣΗΜΜΥ
ακ. έτος 2024-2025

Εργαστήριο Υπολογιστικών Συστημάτων (CSLab)
ΕΜΠ

Οκτώβριος 2024

Περιεχόμενα Παρουσίασης

1. Εισαγωγή
2. Καταστάσεις χρήστη/πυρήνα
3. Process context/interrupt context
4. PCB – task_struct
5. Διαχείριση μνήμης
6. Συγχρονισμός
7. Kernel vs. user programming
8. Περιβάλλον ανάπτυξης (Qemu-KVM)

Πυρήνας Linux

- `http://www.kernel.org`
- `https://github.com/torvalds/linux/`

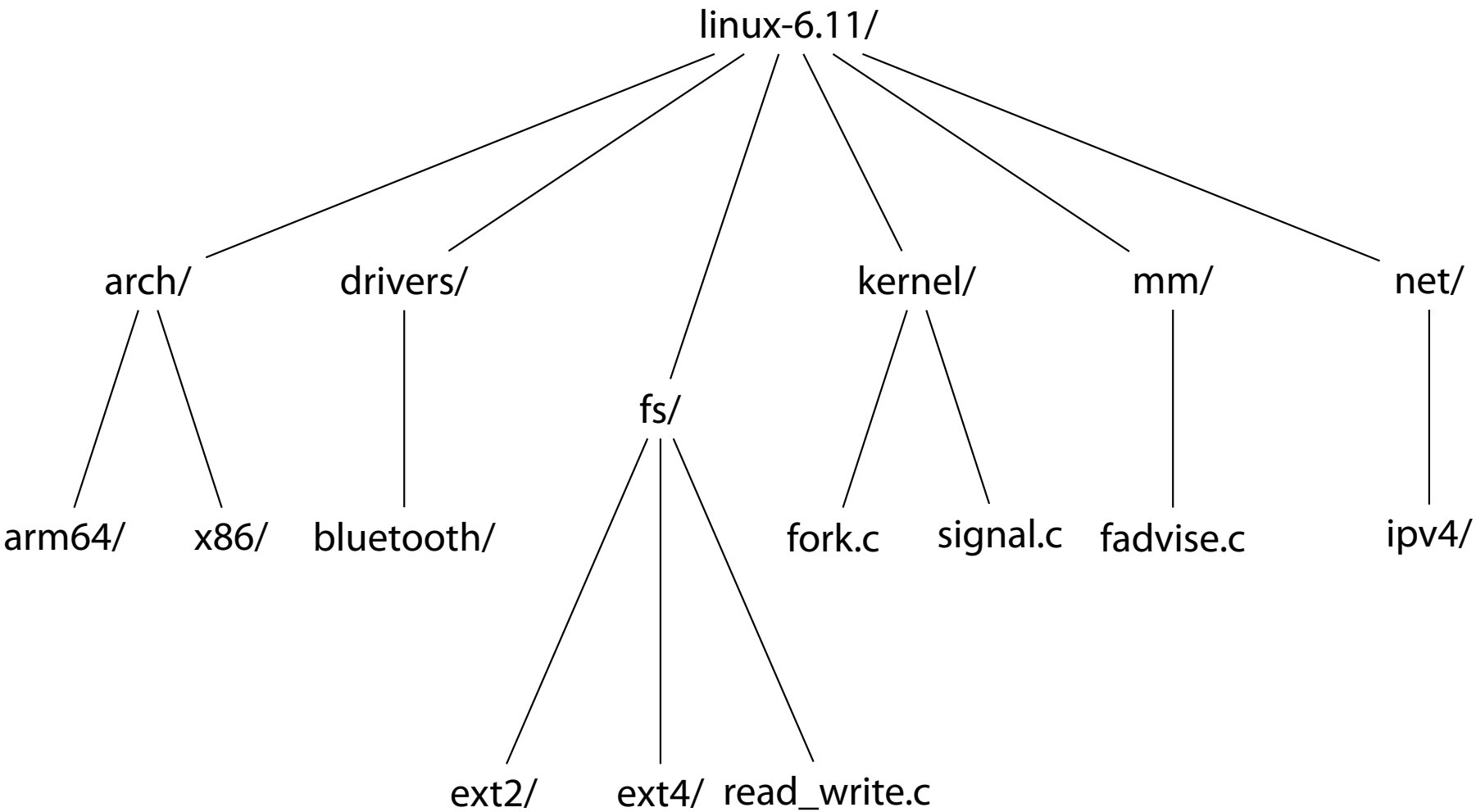
- “I’m doing a (free) operating system (just a hobby, won’t be big and professional like gnu) for 386(486) AT clones” – Linus Torvalds ’91

- Linux Kernel v6.11 (released @ 15/9/2024)*:
  - > 73K unique files
  - > 27 MLOC
  - 1970 προγραμματιστές [1]
  - Τρέχει ουσιαστικά παντού (απο κινητά μέχρι και σε υπερ-υπολογιστές)

[1] `https://lwn.net/Articles/989528/` - ”Some 6.11 development statistics”

* Οι εργαστηριακές (2 και 3) άσκησεις είναι βασισμένες στην έκδοση 6.11

Οργάνωση κώδικα πυρήνα



Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, void *buf, size_t count);
```

Αρχείο `fs/read_write.c:627`

```
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, void *buf, size_t count);
```

Αρχείο `fs/read_write.c:627`

```
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    return ksys_read(fd, buf, count);
}
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:608`

```
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:608`

```
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:608`

```
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;

    if (f.file) {
        loff_t pos, *ppos = file_ppos(f.file);
        if (ppos) {
            pos = *ppos;
            ppos = &pos;
        }
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:608`

```
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;

    if (f.file) {
        loff_t pos, *ppos = file_ppos(f.file);
        if (ppos) {
            pos = *ppos;
            ppos = &pos;
        }
        ret = vfs_read(f.file, buf, count, ppos);
        if (ret >= 0 && ppos)
            f.file->f_pos = pos;
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:608`

```
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;

    if (f.file) {
        loff_t pos, *ppos = file_ppos(f.file);
        if (ppos) {
            pos = *ppos;
            ppos = &pos;
        }
        ret = vfs_read(f.file, buf, count, ppos);
        if (ret >= 0 && ppos)
            f.file->f_pos = pos;
        fdput_pos(f);
    }
    return ret;
}
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:456`

```
ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
    ssize_t ret;

    if (!(file->f_mode & FMODE_READ))
        return -EBADF;
    if (!(file->f_mode & FMODE_CAN_READ))
        return -EINVAL;
    if (unlikely(!access_ok(buf, count)))
        return -EFAULT;

    ret = rw_verify_area(READ, file, pos, count);
    if (ret)
        return ret;
    if (count > MAX_RW_COUNT)
        count =  MAX_RW_COUNT;
    ...
}
```

Πλοήγηση στον κώδικα του πυρήνα

Παράδειγμα

Υλοποίηση της κλήσης συστήματος:

```
ssize_t read(int fd, char __user *buf, size_t count);
```

Αρχείο `fs/read_write.c:456`

```
ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
    ...
    if (file->f_op->read)
        ret = file->f_op->read(file, buf, count, pos);
    else if (file->f_op->read_iter)
        ret = new_sync_read(file, buf, count, pos);
    else
        ret = -EINVAL;
    if (ret > 0) {
        fsnotify_access(file);
        add_rchar(current, ret);
    }
    inc_syscr(current);
    return ret;
}
```

Πλοήγηση στον κώδικα του πυρήνα

Linux Cross Reference

Χρήσιμο εργαλείο: `http://lxr.free-electrons.com`

- Online browser του κώδικα του πυρήνα.
- Κώδικας από διάφορες εκδόσεις του πυρήνα.
- Εύκολη αναζήτηση στον κώδικα.
- Διαχείριση και s/w projects εκτός από τον πυρήνα (π.χ., QEMU).

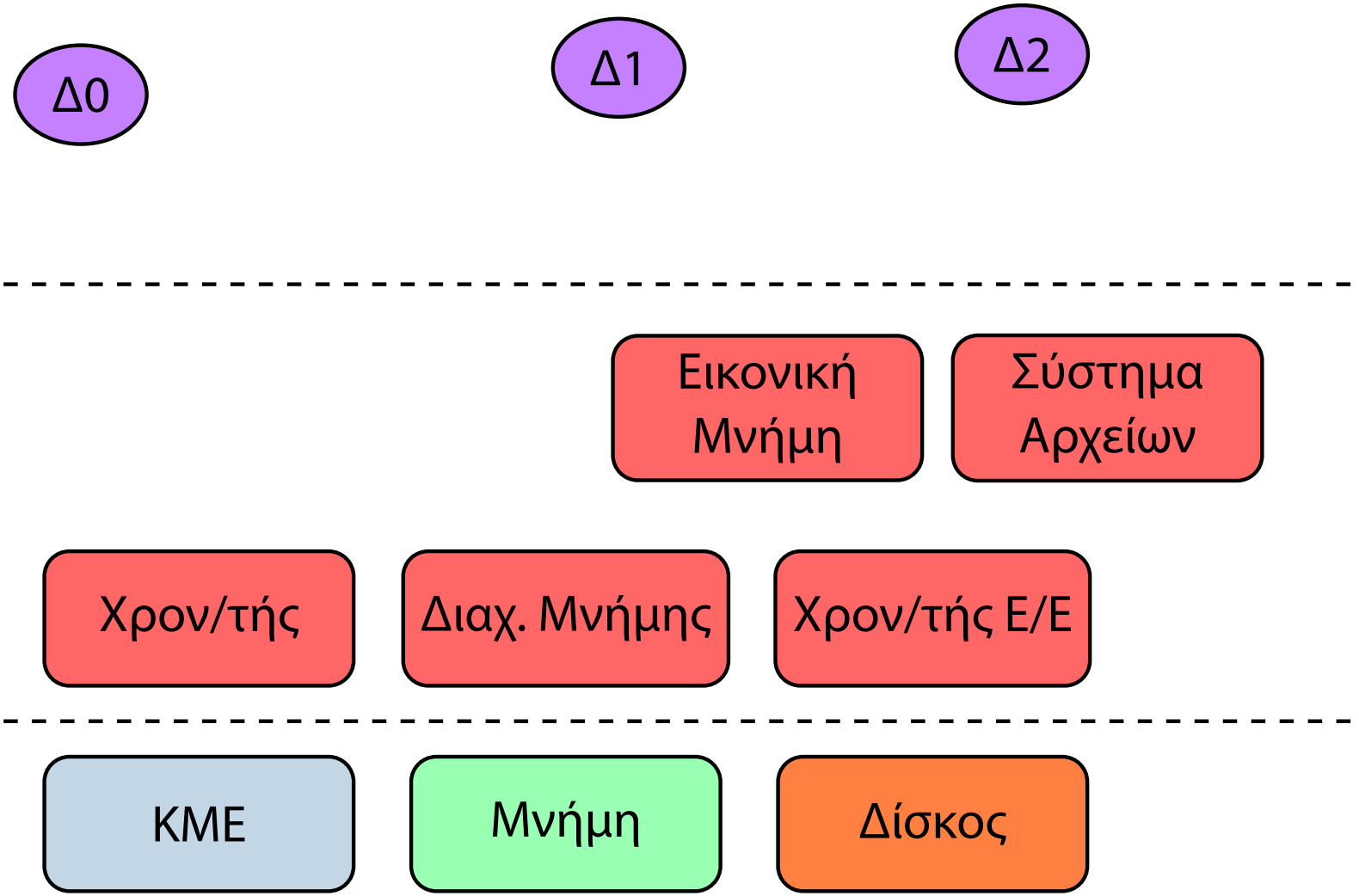
Καταστάσεις χρήστη/πυρήνα

Μία διεργασία μπορεί να βρίσκεται σε δύο καταστάσεις:

- Κατάσταση χρήστη (user mode).
  - Περιορισμένες δυνατότητες.
- Κατάσταση πυρήνα (kernel mode).
  - Πλήρης έλεγχος του συστήματος.
- **Ο πυρήνας δεν είναι διεργασία, αλλά κώδικας που εκτελείται σε kernel mode …**
  - **είτε εκ μέρους κάποιας διεργασίας χρήστη**
  - **είτε ως απόκριση σε κάποιο hardware event.**

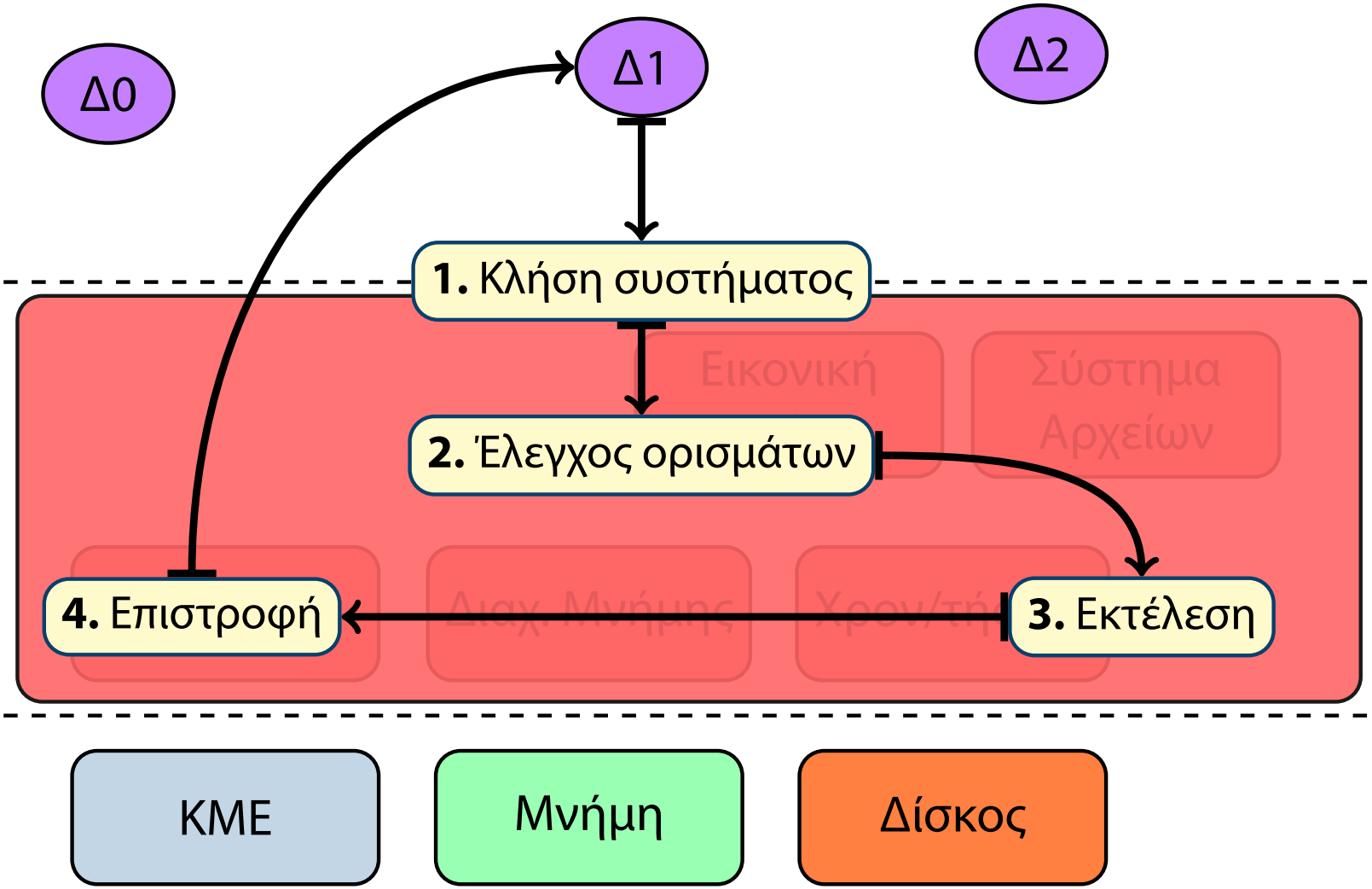
Καταστάσεις χρήστη/πυρήνα

Ροή εκτέλεσης κλήσης συστήματος



Καταστάσεις χρήστη/πυρήνα

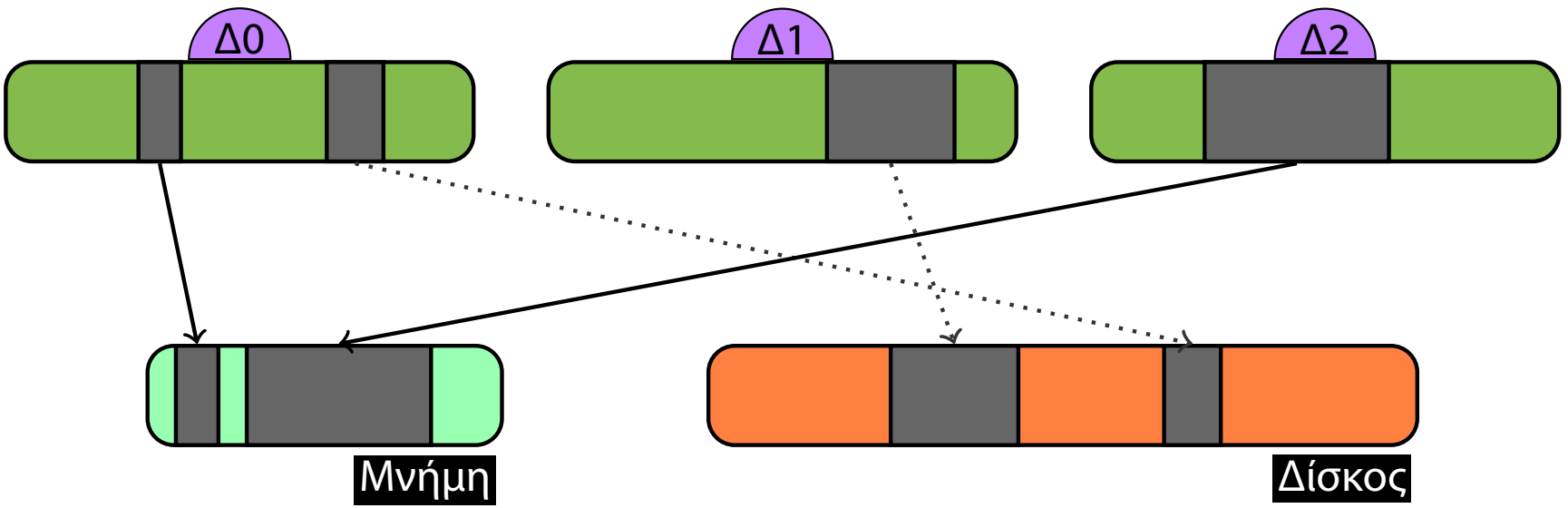
Ροή εκτέλεσης κλήσης συστήματος



Καταστάσεις χρήστη/πυρήνα

Διαχωρισμός χώρων χρήστη/πυρήνα

Το Linux είναι ένα σύγχρονο Λ.Σ. που χρησιμοποιεί εικονική μνήμη.



Καταστάσεις χρήστη/πυρήνα

Διαχωρισμός χώρων χρήστη/πυρήνα

- Ο εικονικός χώρος διευθύνσεων ενός μηχανήματος χωρίζεται σε δύο μέρη:
  - Χώρος χρήστη (εφαρμογές και δεδομένα χρήστη).
  - Χώρος πυρήνα (δεδομένα του πυρήνα).
- Μία διεργασία που τρέχει στον χώρο χρήστη έχει πρόσβαση **μόνο** στο χώρο χρήστη.
- Μία διεργασία που τρέχει στον χώρο πυρήνα έχει **απεριόριστη** πρόσβαση σε όλο το σύστημα.

Καταστάσεις χρήστη/πυρήνα

Διαχωρισμός χώρων χρήστη/πυρήνα

- Ο εικονικός χώρος διευθύνσεων ενός μηχανήματος χωρίζεται σε δύο μέρη:
  - Χώρος χρήστη (εφαρμογές και δεδομένα χρήστη).
  - Χώρος πυρήνα (δεδομένα του πυρήνα).
- Μία διεργασία που τρέχει στον χώρο χρήστη έχει πρόσβαση **μόνο** στο χώρο χρήστη.
- Μία διεργασία που τρέχει στον χώρο πυρήνα έχει **απεριόριστη** πρόσβαση σε όλο το σύστημα.

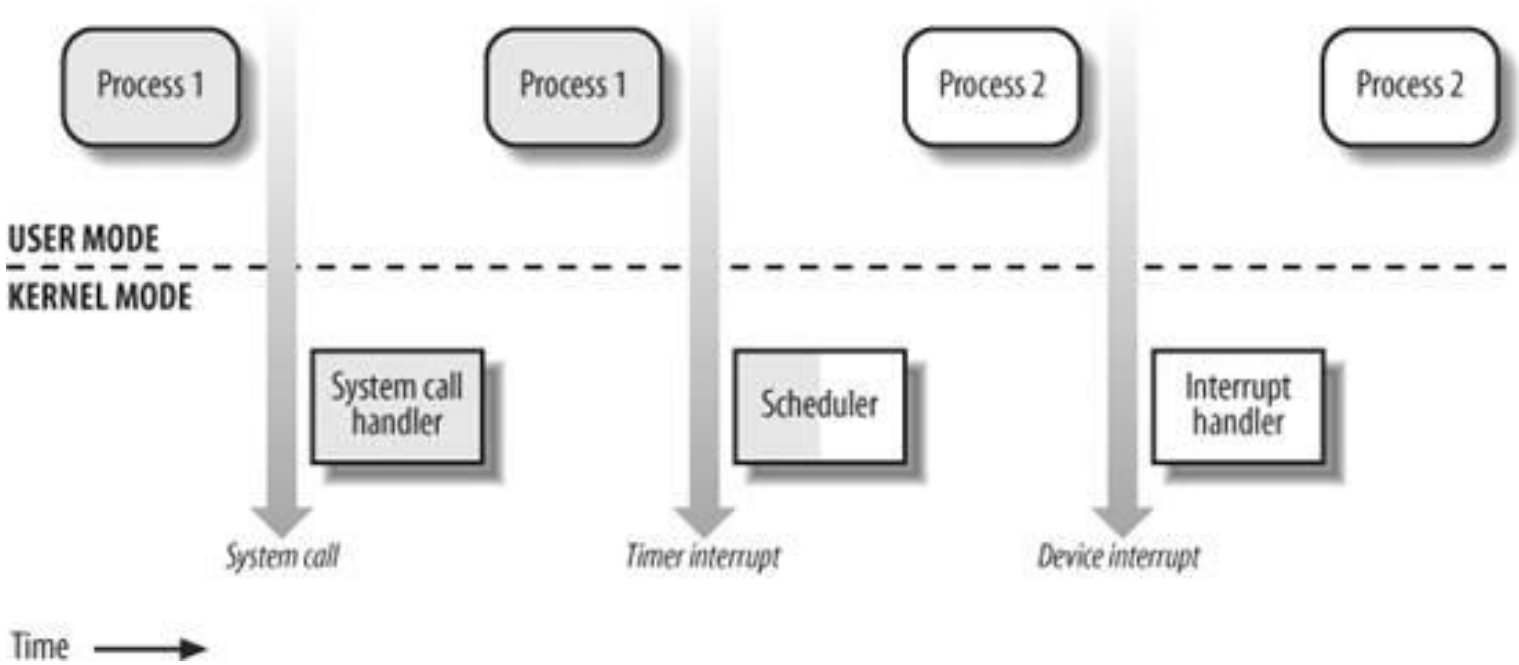
Μεταφορά δεδομένων από/προς χώρο χρήστη

- `copy_from_user`: userspace $\rightarrow$ kernelspace.
- `copy_to_user`: kernelspace $\rightarrow$ userspace.

Kernel contexts

Ο πυρήνας μπορεί να εκτελείται …

1. εκ μέρους κάποιας διεργασίας χρήστη (process context)
2. ως απόκριση σε κάποιο hardware event (interrupt context)
3. …Υπάρχουν και kernel threads



Σημεία εισόδου στον πυρήνα

- Κλήσεις συστήματος (system calls)
- Οδηγοί συσκευών (device drivers)
- Pseudo filesystem `/proc`

Process Control Block – `task_struct`

Αρχείο `include/linux/sched.h:1182`

```
struct task_struct {
    volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
    void *stack;
    ...
    int prio, static_prio, normal_prio;
    unsigned int rt_priority;
    struct mm_struct *mm, *active_mm;
    ...
    pid_t pid;
    const struct cred __rcu *cred;  /* effective (overridable) subjective task
                    * credentials (COW) */
    ...
    /* open file information */
    struct files_struct *files;
    ...
};
```

Διαχείριση μνήμης στον πυρήνα του Linux

Σε χαμηλό επίπεδο

- Βασική μονάδα διαχείρισης της φυσικής μνήμης, η σελίδα (`struct page`).
- Ζώνες μνήμης (DMA, Highmem, Normal).
- Διαχείριση σελίδων φυσικής μνήμης: `alloc_pages`, `__get_free_pages`, `__free_pages`.
- Συνεχόμενες σελίδες φυσικής μνήμης.

Διαχείριση μνήμης στον πυρήνα του Linux

Σε υψηλότερο επίπεδο

- `kmalloc` (πόσα bytes θέλουμε + flags).
  - Συνεχόμενες σελίδες **φυσικής** μνήμης.
  - Μηχανισμοί caching (Slab).
- `vmalloc` (σαν την γνωστή userspace `malloc`).
  - Συνεχόμενες σελίδες **εικονικής** μνήμης.
- Αποδέσμευση μνήμης: `kfree`, `vfree`.

slabinfo

cat /proc/slabinfo

```
ext2_inode
ext2_xattr
ext3_inode
ext3_xattr
tcp_bind_bucket
blkdev_requests
inode_cache
size-4096(DMA)
size-4096
size-2048(DMA)
size-2048
size-1024(DMA)
```

Γιατί χρειάζεται συγχρονισμός;

- Πολυεπεξεργασία (συστήματα μοιραζόμενης μνήμης)
- Ασύγχρονες διακοπές
- Διακοπτός πυρήνας (preemptible kernel)

Επομένως, πρόσβαση σε μοιραζόμενες δομές πρέπει να προστατεύεται με κάποιο είδος κλειδώματος.

Μηχανισμοί συγχρονισμού στον πυρήνα

Μερικοί από τους μηχανισμούς συγχρονισμού που υλοποιούνται στο χώρο πυρήνα είναι οι εξής:

- Ατομικές εντολές (Atomic Operations)
  Interface: `atomic_read()`, `atomic_set()`, ...
- Περιστροφικά Κλειδώματα (Spinlocks)
  Interface: `spin_unlock()`, `spin_lock()`, `spin_unlock_irqrestore()`, `spin_lock_irqsave()`, ...
- Σημαφόροι (Semaphores)
  Interface: `down()`, `down_interruptible()`, `up()`, ...

Kernel vs. user programming

- Μικρή στατική στοίβα (προσοχή στις τοπικές μεταβλητές).
- Δεν μπορούμε να χρησιμοποιούμε πράξεις κινητής υποδιαστολής.
  - Ή πλέον μπορούμε αλλά με μεγάλο κόστος σε επίδοση.
- Δεν μπορούμε να χρησιμοποιήσουμε την libc.
  - Δεν υπάρχει libc στον πυρήνα.
- Στον πυρήνα παρ' όλα αυτά υλοποιούνται πολλές συναρτήσεις με interface παρόμοιο με των συναρτήσεων της libc, π.χ.,
  - `printk()`
  - `kmalloc()`
  - `kfree()`

Kernel vs. user programming (2)

ΠΡΟΣΟΧΗ: Δεν βρισκόμαστε πλέον στον “προστατευμένο” χώρο χρήστη.

Παράδειγμα: αποδεικτοδότηση δείκτη σε NULL

- Στο χώρο χρήστη: Segmentation Fault
- Στον πυρήνα: Kernel Oops

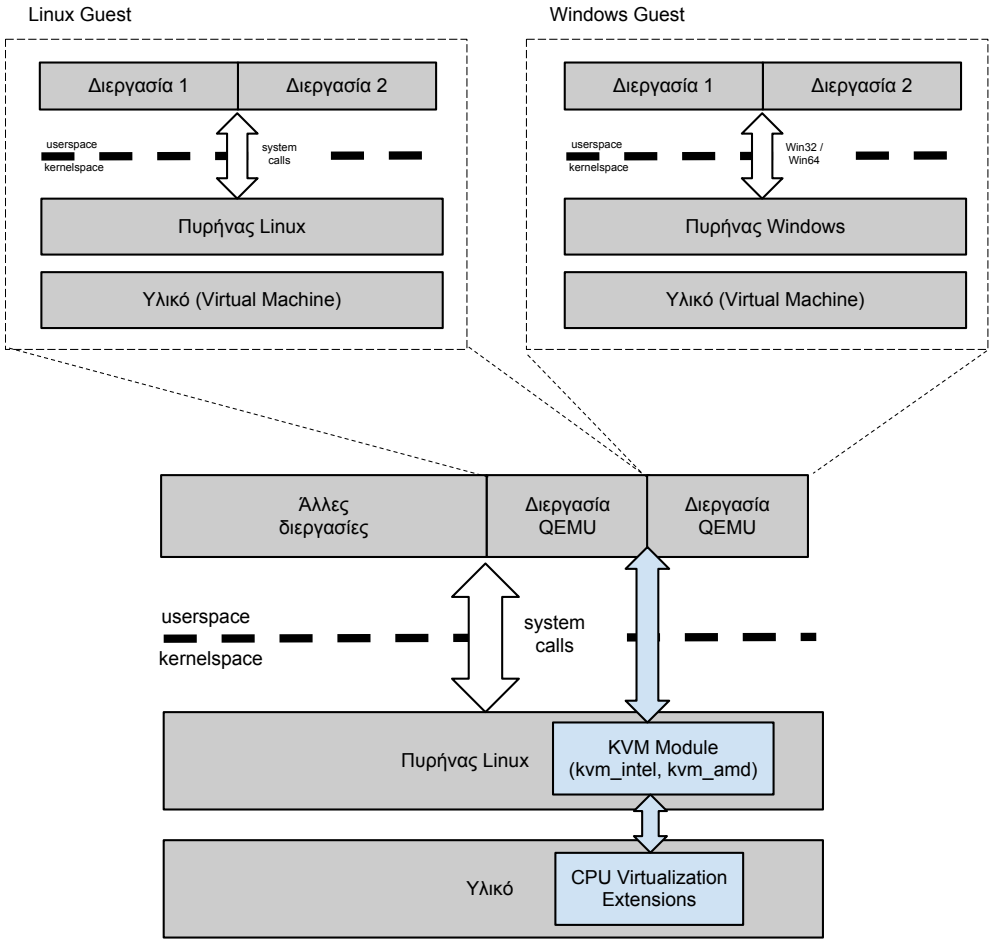
Περιβάλλον ανάπτυξης (Qemu-KVM)

- Για τη διαδικασία ανάπτυξης κώδικα στον πυρήνα δε χρειάζονται αυξημένα δικαιώματα, αλλά …
- εγκατάσταση ενός νέου πυρήνα στο σύστημα και φόρτωση ενός νέου module μπορεί να κάνει μόνο ο χρήστης **root**.

Πώς δοκιμάζουμε ένα νέο πυρήνα με ασφάλεια;

- Με χρήση εικονικής μηχανής, που θα «τρέχει» τον νέο πυρήνα στο χώρο χρήστη.
- Το qemu (Quick EMUlator) είναι ένας emulator που προσωμοιώνει τη λειτουργία ενός πραγματικού υπολογιστή.
- Το KVM είναι ένα σύνολο από modules του πυρήνα που επιτρέπουν στο χρήστη να εκμεταλλευτεί τις επεκτάσεις των σύγχρονων επεξεργαστών για virtualization.

Qemu-KVM Virtualization



Σχήμα: Αρχιτεκτονική του Qemu-KVM.

Χρήση του Qemu-KVM

Βοηθητικά αρχεία:

- `utopia.sh`: εκκινεί την εικονική μηχανή.
- `utopia.config`: απαραίτητες ρυθμίσεις.

  `QEMU_BUILD_DIR` Ο φάκελος στον οποίο έχει εγκατασταθεί το qemu.
  `ROOTFS_FILE` Το root filesystem που θα χρησιμοποιήσει η εικονική μηχανή.
- Πιο αναλυτικές οδηγίες στον οδηγό που δίνεται στο site του μαθήματος.

Βιβλιογραφία

- Linux Kernel Development, Robert Love, Novell Press, 2005
- Linux Device Drivers, Jonathan Corbet, Alessandro Rubin, Greg Kroah-Hartman, O'Reilly Media, 3rd Edition, 2005, `http://lwn.net/Kernel/LDD3/`
- Understanding the Linux kernel, Daniel Bovet, Marco Cesati, O' Reilly Media, 3rd edition, 2005

Ευχαριστούμε

Λίστα μαθήματος:
os-lab@lists.cslab.ece.ntua.gr