Εισαγωγή στον προγραμματισμό στον πυρήνα του Linux

Εργαστήριο Λειτουργικών Συστημάτων
8ο εξάμηνο, ΣΗΜΜΥ

Εργαστήριο Υπολογιστικών Συστημάτων (CSLab)

Μάρτιος 2018

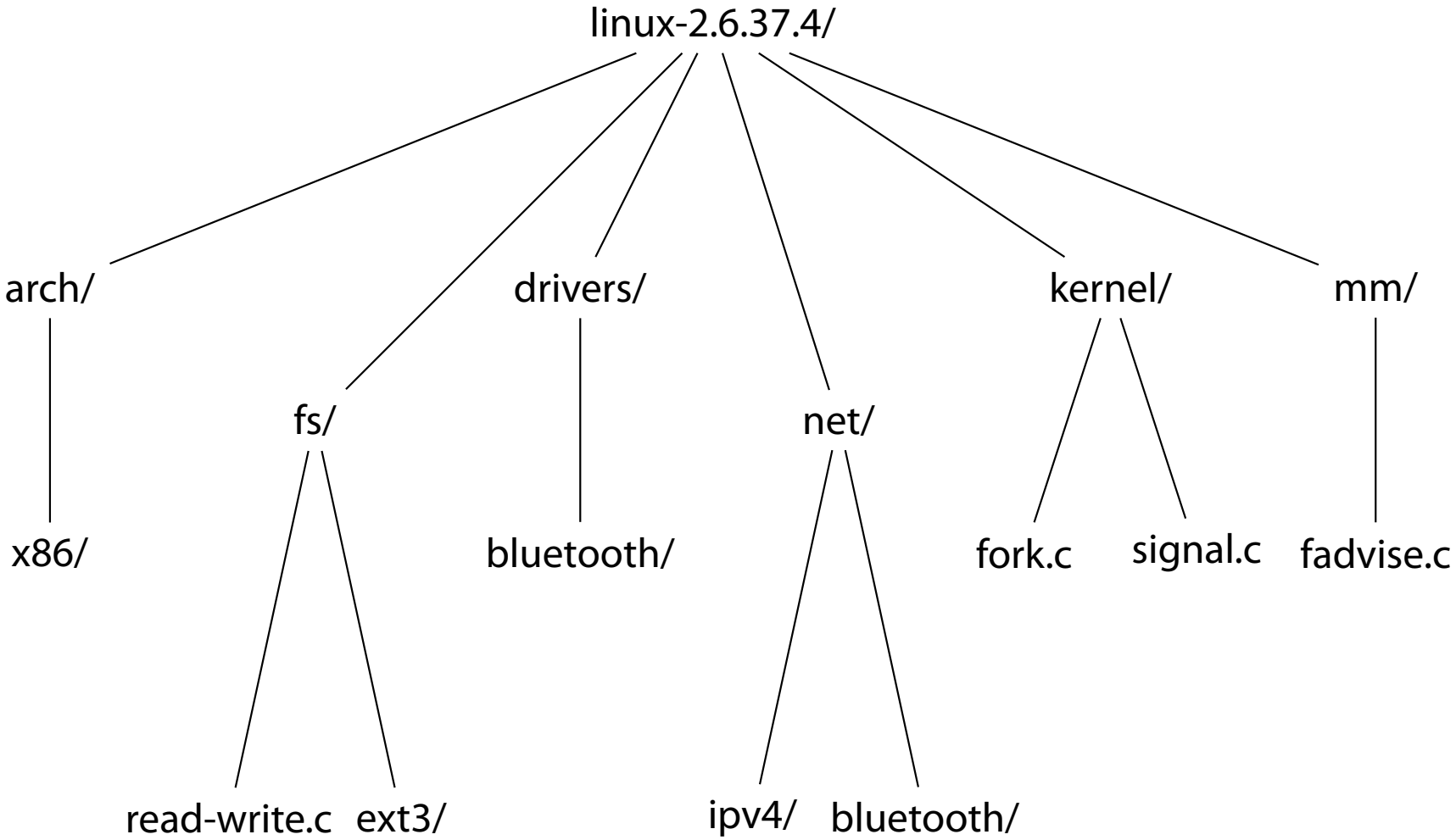
Περίγραμμα παρουσίασης

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

Πυρήνας Linux

- `http://www.kernel.org`
- “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
- **Τώρα**:
  - 19.1 MLOC
  - Εκατοντάδες προγραμματιστές
  - Τρέχει σε κινητά, αλλά και σε υπερ-υπολογιστές

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:407`

```
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:407`

```
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    struct file *file;
    ssize_t ret = -EBADF;
```

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:407`

```
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    struct file *file;
    ssize_t ret = -EBADF;
    int fput_needed;

    file = fget_light(fd, &fput_needed);
    if (file) {
        loff_t pos = file_pos_read(file);
        ret = vfs_read(file, buf, count, &pos);
        file_pos_write(file, pos);
        fput_light(file, fput_needed);
    }

    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.

Χρήσιμα εργαλεία

Κλήσεις συστήματος/σήματα – `strace`

- Πληροφορίες για τις κλήσεις συστήματος που καλεί μια διεργασία και για τα σήματα που λαμβάνει.
- Δυνατότητα να γίνουν trace και τα παιδιά της διεργασίας.

Παράδειγμα χρήσης

```
user@utopia:~$ strace cat foo
execve("/bin/cat", ["cat", "foo"], [/* 35 vars */]) = 0
...
open("foo", O_RDONLY)                   = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=22, ...}) = 0
read(3, "I am an example file!\n"..., 4096) = 22
write(1, "I am an example file!\n"..., 22I am an example file!
) = 22
read(3, ""..., 4096)                    = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
...
```

Χρήσιμα Εργαλεία

Στατιστικά συστήματος – `vmstat`

- Χρήση μνήμης (active/inactive, buffers, cache, free, swap).
- Χρήση CPU (user/system times, idle, αναμονή για Ε/Ε).
- Χρήση δίσκων (αναγνώσεις/εγγραφές, τρέχουσες λειτουργίες Ε/Ε).
- Άλλα στατιστικά (caching αντικειμένων του πυρήνα, πλήθος `forks`).

Παράδειγμα

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
0  0      0 741084  12712 189216    0    0   140     0  643 1133  3  2 91  4
1  1      0 737232  13656 190336    0    0  1732     0 1060 3216  8  5 57 30
2  1      0 736116  15080 190004    0    0  1456     0 1182 4584 18 10 25 47
0  1      0 737160  15244 190504    0    0   212     0  878 3185 24  5 63  9
```

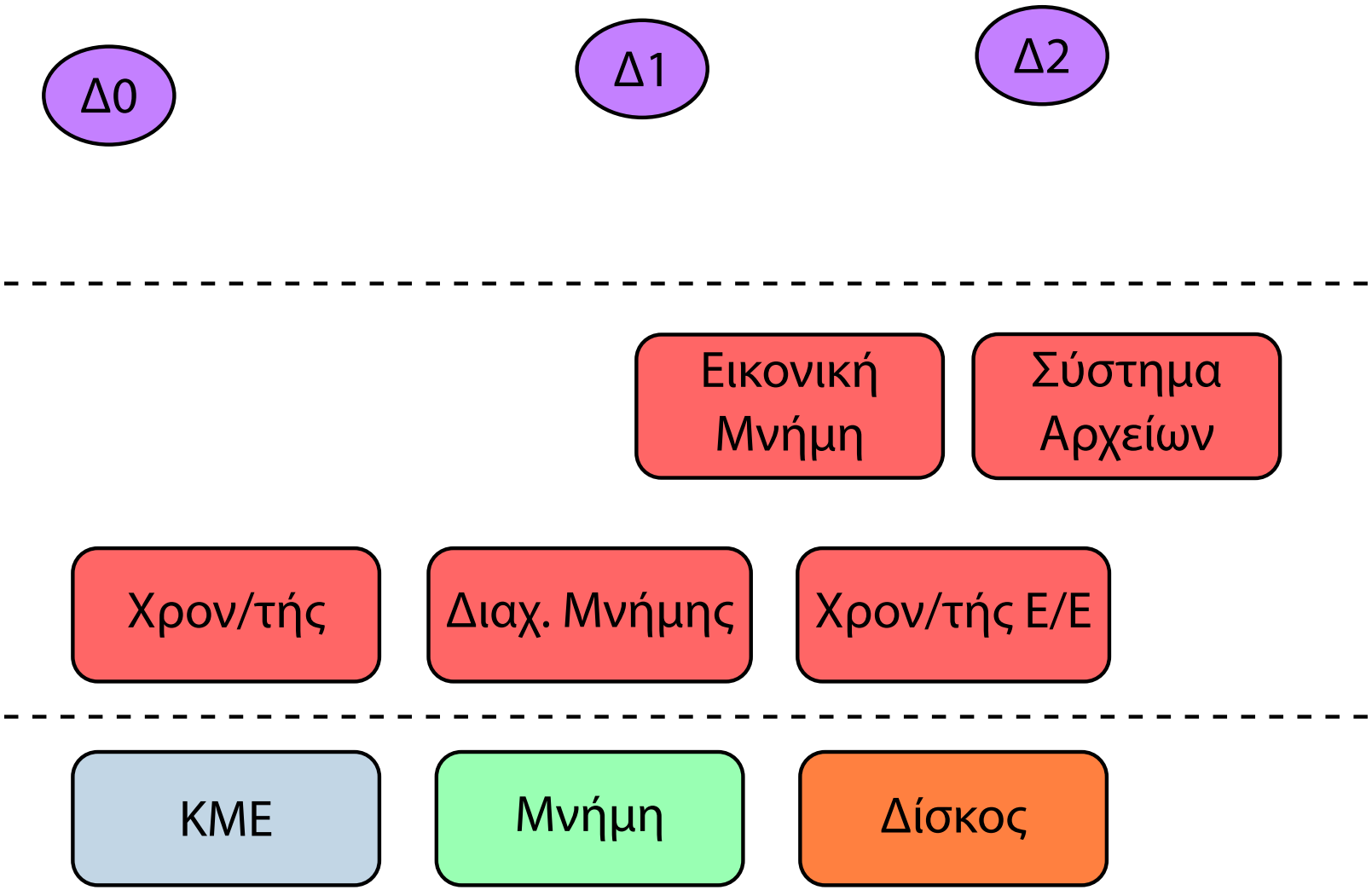
Κάταστασεις χρήστη/πυρήνα

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.**

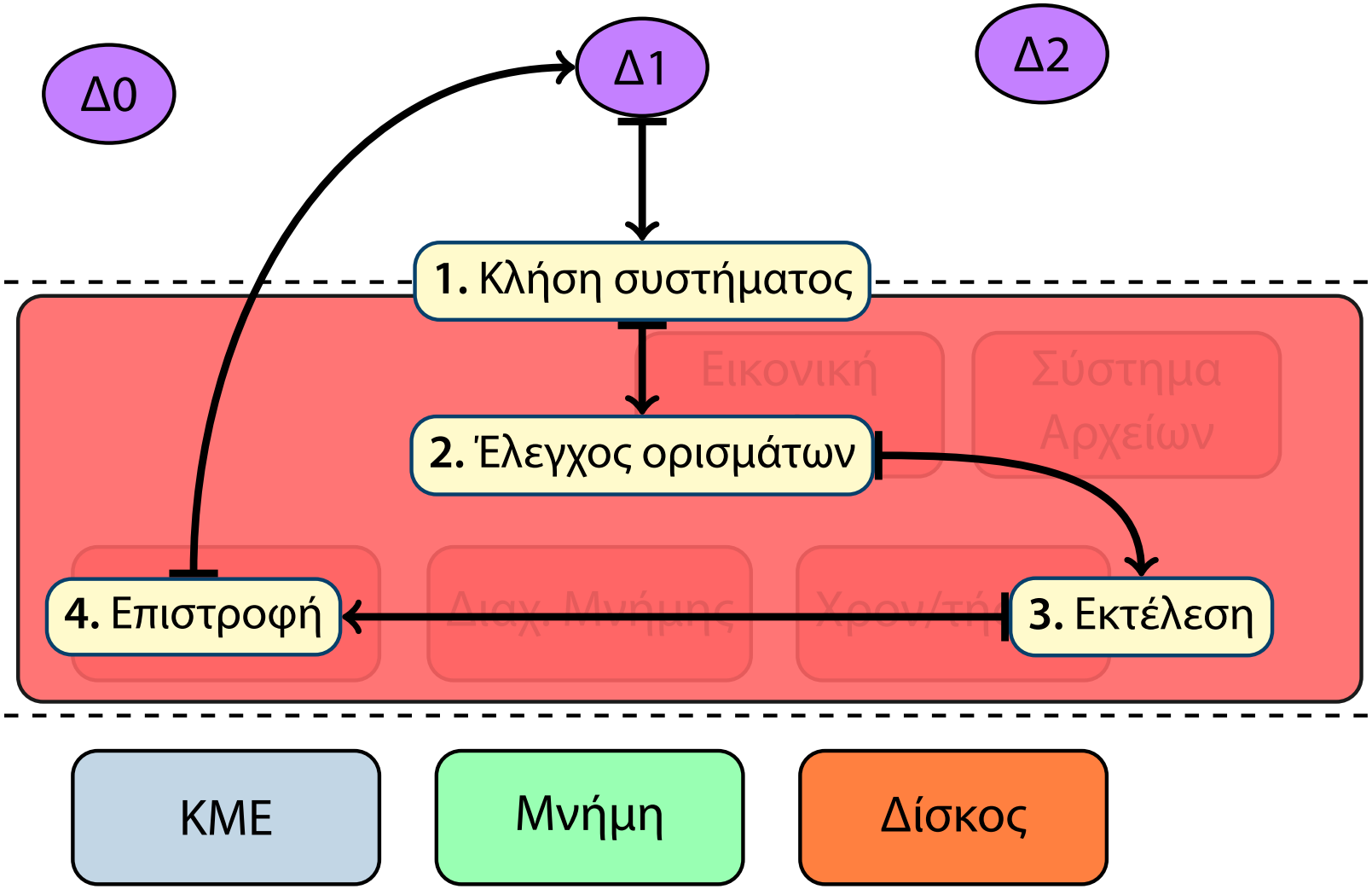
Καταστάσεις χρήστη/πυρήνα

Ροή εκτέλεσης κλήσης συστήματος



Καταστάσεις χρήστη/πυρήνα

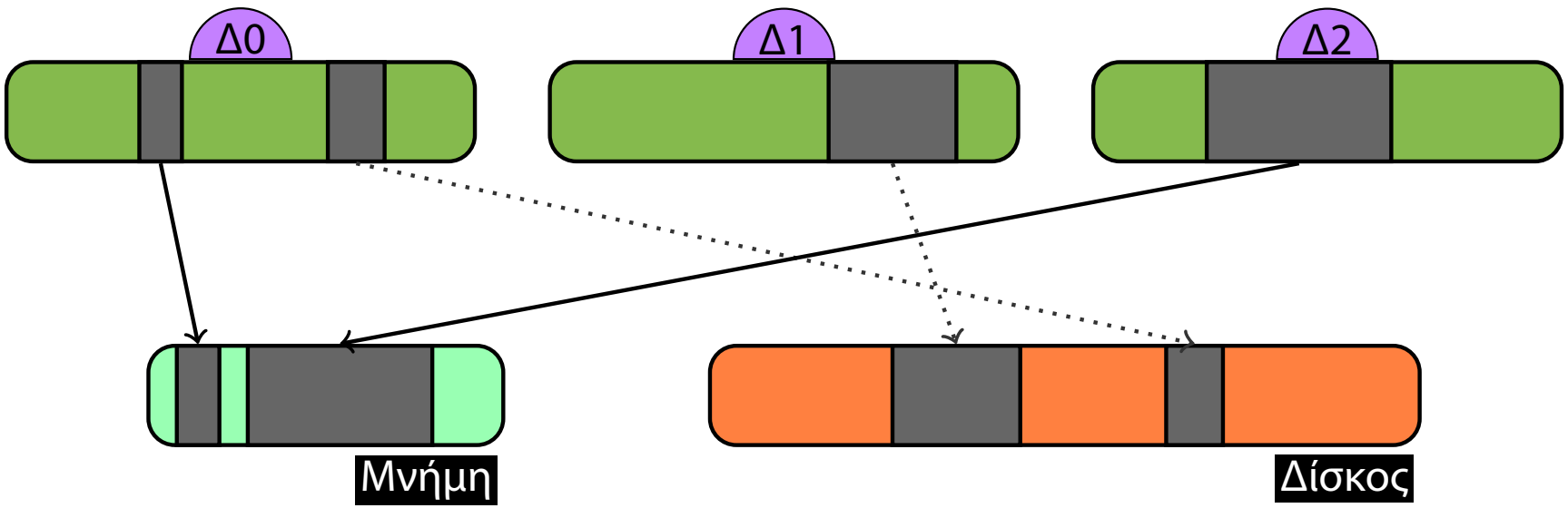
Ροή εκτέλεσης κλήσης συστήματος



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.

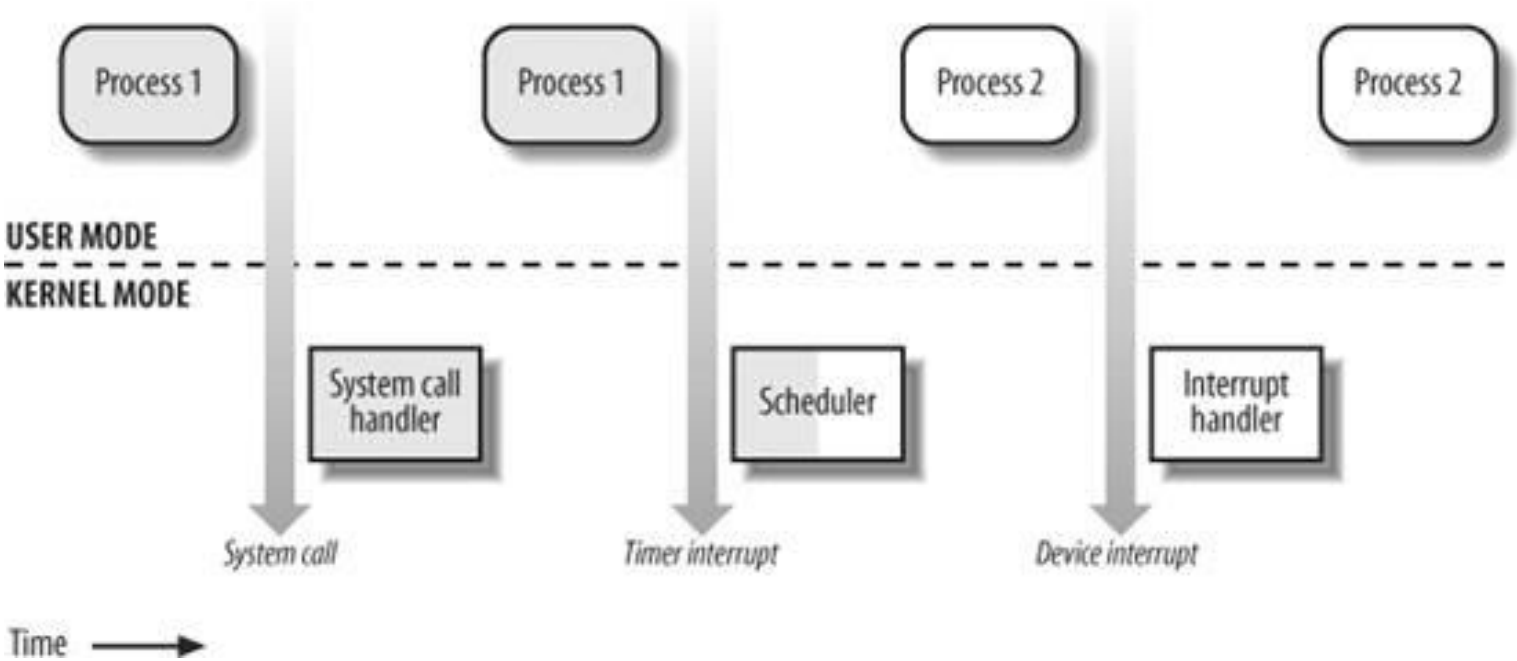
Μεταφορά δεδομένων από/προς χώρο χρήστη

- `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;
    ...
};
```

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;
    ...
};
```

Παράδειγμα: struct cred

- Locking
- Reference counting
- Copy-On-Write (COW)

Παράδειγμα: struct cred

Get

include/linux/cred.h:224

```
/**
 * get_cred - Get a reference on a set of credentials
 * @cred: The credentials to reference
 *
 * Get a reference on the specified set of credentials.  The caller must
 * release the reference.
 * ...
 */
static inline const struct cred *get_cred(const struct cred *cred)
```

Παράδειγμα: struct cred

Put

include/linux/cred.h:244

```
/**
 * put_cred - Release a reference to a set of credentials
 * @cred: The credentials to release
 *
 * Release a reference to a set of credentials, deleting them when the last ref
 * is released.
 * ...
 */
static inline void put_cred(const struct cred *_cred)
```

Παράδειγμα: struct cred

Prepare

kernel/cred.c:269

```
/**
 * prepare_creds - Prepare a new set of credentials for modification
 *
 * Prepare a new set of task credentials for modification.
 * A task's creds shouldn't generally be modified directly, therefore
 * this function is used to prepare a new copy, which the caller then
 * modifies and then commits by calling commit_creds().
 *
 * Preparation involves making a copy of the objective creds for
 * modification.
 *
 * Returns a pointer to the new creds-to-be if successful, NULL otherwise.
 *
 * Call commit_creds() or abort_creds() to clean up.
 */
struct cred *prepare_creds(void)
```

Διαχείριση μνήμης στον πυρήνα του Linux

Σε χαμηλό επίπεδο

- Βασική μονάδα διαχείρισης της φυσικής μνήμης, η σελίδα (`page_struct`).
- 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.
- Δεν μπορούμε να χρησιμοποιήσουμε την libc.
- 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

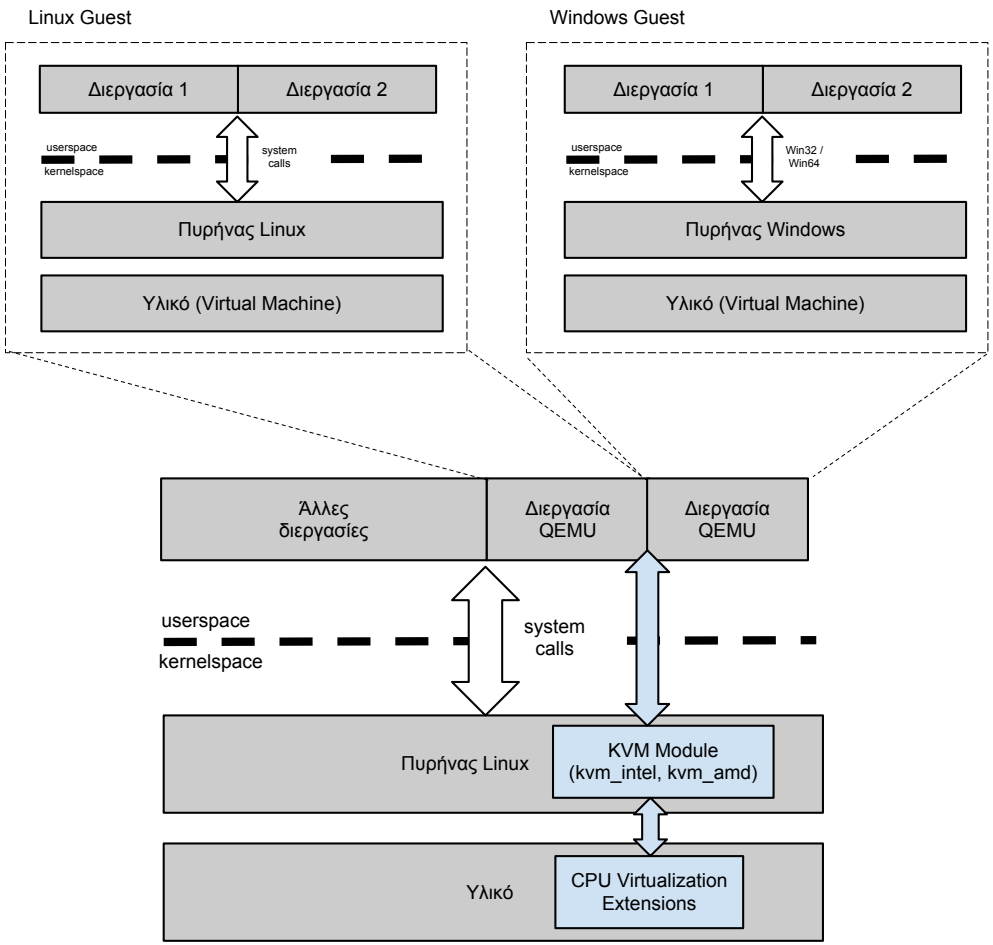
Περιβάλλον ανάπτυξης (Qemu-KVM)

- 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