

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^{ου} Εξαμήνου,

Τομέας Λογικού και Υπολογιστών

Ιωάννης Χατζηγιαννάκης

Τετάρτη, 14 Απριλίου, 2010
Αίθουσα ΒΑ



Σύνοψη 8^{ης} Διάλεξης

3η Άσκηση

Minix recompilation

System Task

Επικοινωνία με το System Task

Λειτουργικό Σύστημα Minix

Ανάθεση Μνήμης

Ανταλλαγή Δεδομένων

Παράδειγμα: Αρχικοποίηση Σημάτων



Τρίτη Άσκηση

- ▶ Αφορά το λειτουργικό σύστημα MINIX
- ▶ Οι απαντήσεις πρέπει να ακολουθούν συγκεκριμένη μορφή
- ▶ Ατομική άσκηση
- ▶ Παράδοση γίνεται με την χρήση του εργαλείου `submit` μέσω του συστήματος `zenon`
- ▶ **Μέχρι Δευτέρα 19 Απριλίου, ώρα 23:59**
- ▶ Αν παρατηρηθεί αντιγραφή, τότε όλοι όσοι συνεργάστηκαν και εμπλέκονται στην αντιγραφή, θα μηδενίζονται στο μάθημα



1ο Πρόβλημα

Περιγράψτε τη διαδικασία που πρέπει να ακολουθήσουμε για να δημιουργήσουμε ένα νέο image του minix όπου η κλήση πυρήνα `SYS_TRACE` να είναι απενεργοποιημένη.



2ο Πρόβλημα

Αναπτύξτε μια νέα κλήση του πυρήνα (kernel call) που να επιστρέφει την έκδοση του πυρήνα (OS_VERSION). Υλοποιήστε μια απλή εντολή που χρησιμοποιεί την νέα κλήση του πυρήνα που αναπτύξατε. Τα αρχεία που θα παραδώσετε θα πρέπει να είναι καλά δομημένα (με την χρήση tab, κλπ.) και καλά σχολιασμένα.



3ο Πρόβλημα

Αναπτύξτε μια νέα κλήση του πυρήνα (kernel call) που να δέχεται έναν ακέραιο ως παράμετρο και να επιστρέφει το AM σας πολλαπλασιασμένο με τον ακέραιο που δώθηκε. Υλοποιήστε μια απλή εντολή που χρησιμοποιεί την νέα κλήση του πυρήνα που αναπτύξατε. Τα αρχεία που θα παραδώσετε θα πρέπει να είναι καλά δομημένα (με την χρήση tab, κλπ.) και καλά σχολιασμένα.



Compiling Minix 3

```
bash-3.00# cd /usr/src/tools
bash-3.00# make hdboot
```

- ▶ Αντικατάσταση του `/usr/include/` με το `/usr/src/include/`
- ▶ Recompile των αρχείων κάτω από τους καταλόγους `/usr/src/kernel/`, `/usr/src/servers/` και `/usr/src/drivers/` σε object files.
- ▶ Linking των object files κάθε καταλόγου σε ενιαίο εκτελέσιμο πρόγραμμα
- ▶ Ομοίως για τα υπόλοιπα χρήσιμα (εκτός πυρήνα) sources (driver συσκευής που θα φιλοξενήσει τον πυρήνα, άλλοι servers, log, tty κτλ)
- ▶ Το σύνολο των παραπάνω προγραμμάτων θα αποτελέσει το image του πυρήνα
- ▶ Θα γίνει εγκατάσταση του νέου image



Επεξεργασία Αρχείων

- ▶ Χρήση editor: vi, vim, nano
- ▶ Επεξεργασία αρχείων εκτός minix
 - ▶ Αντιγραφή των αρχείων κάτω από τους καταλόγους `/usr/src/` εκτός minix
 - ▶ Χρήση κάποιου editor π.χ. Visual Studio, notepad++
 - ▶ Αντιγραφή αλλαγμένων αρχείων εντός minix
- ▶ Αντιγραφή αρχείων από/πρός minix
 - ▶ Με την χρήση ftp server/client
 - ▶ Με την χρήση ssh / scp



Δομή κώδικα του System Task

- ▶ Το System Task υλοποιείται από το `system.h` και `system.c`
 - ▶ Τα αρχεία είναι στον φάκελο `/usr/src/kernel`
- ▶ Το Minix δεν είναι **μόνο** για γενική χρήση
 - ▶ Στοιχείει και σε ενσωματωμένα συστήματα
- ▶ Επιτρέπει την ενεργοποίηση/απενεργοποίηση των κλήσεων του πυρήνα – μείωση μεγέθους εκτελέσιμου
- ▶ Το αρχείο `kernel/config.h` ορίζει ποιες κλήσεις είναι ενεργές

```
#define USE_FORK    1 // fork a new process
#define USE_NEWMAP 1 // set a new memory map
#define USE_EXEC   1 // update process after execute
#define USE_EXIT   1 // clean up after process exit
#define USE_NICE   1 // change scheduling priority
```

- ▶ Αν αφαιρέσουμε μια δήλωση, όλα τα τμήματα του κώδικα που σχετίζονται με την συγκεκριμένη κλήση δεν θα γίνουν compile



Ορισμός Κλήσεων Πυρήνα

- ▶ Όλες οι κλήσεις του πυρήνα ορίζονται στο αρχείο `com.h`
 - ▶ Βρίσκεται στον φάκελο `/usr/src/include/minix`

```
#define KERNEL_CALL 0x600 // base for kernel calls
                          // to SYSTEM

#define SYS_FORK      (KERNEL_CALL + 0) // sys_fork()
#define SYS_EXEC     (KERNEL_CALL + 1) // sys_exec()
#define SYS_EXIT     (KERNEL_CALL + 2) // sys_exit()
#define SYS_NICE     (KERNEL_CALL + 3) // sys_nice()

#define NR_SYS_CALLS 28 // number of system calls
```

- ▶ Αν θέλουμε να προσθέσουμε μια νέα κλήση, πρέπει να την ορίσουμε εδώ
- ▶ Ανεξάρτητα από το αν θα την απενεργοποιήσουμε ή όχι



Αρχείο Επικεφαλίδας System Task

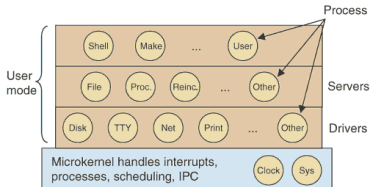
- ▶ Το αρχείο `system.h` ορίζει τις συναρτήσεις που χειρίζονται κάθε κλήση του πυρήνα
 - ▶ Για την κλήση `sys_xxx` αντιστοιχεί την συνάρτηση `do_xxx`

```
/* Default handler for unused kernel calls. */
_PROTOTYPE( int do_unused, (message *m_ptr) );
_PROTOTYPE( int do_exec, (message *m_ptr) );
_PROTOTYPE( int do_fork, (message *m_ptr) );
_PROTOTYPE( int do_trace, (message *m_ptr) );
_PROTOTYPE( int do_nice, (message *m_ptr) );
_PROTOTYPE( int do_copy, (message *m_ptr) );
#define do_vircopy do_copy
#define do_physcopy do_copy
```

- ▶ Η συνάρτηση `do_unused` είναι μια 'κενή' συνάρτηση
- ▶ Η συνάρτηση `do_copy` αντιστοιχεί επίσης στις κλήσεις `sys_vircopy` `sys_physcopy`



Η Εσωτερική Δομή του Minix 3



- ▶ Οι λειτουργίες που προσφέρει το Λ.Σ. υλοποιούνται ως ανεξάρτητες διεργασίες
- ▶ Η επικοινωνία μεταξύ των διεργασιών γίνεται με την ανταλλαγή μηνυμάτων



Επικοινωνία με το System Task

- ▶ Οι διαχειριστές επικοινωνούν μεταξύ τους, με τους οδηγούς και με τον πυρήνα **με την χρήση μηνυμάτων**
- ▶ Το System Task λαμβάνει όλα τα μηνύματα που αφορούν τον πυρήνα
 - ▶ Είναι το σημείο εισόδου / εξόδου στον πυρήνα για τις διεργασίες υψηλότερων επιπέδων
- ▶ Θα μπορούσαμε να ονομάσουμε όλες αυτές τις αιτήσεις ως κλήσεις του συστήματος
 - ▶ Στο Minix τις ονομάζουμε **κλήσεις του πυρήνα**
- ▶ Δεν είναι προσβάσιμες στις 'απλές' διεργασίες
- ▶ Στις περισσότερες περιπτώσεις μια κλήση του συστήματος μετατρέπεται σε κλήση του πυρήνα με παρόμοιο όνομα
- ▶ Απλά διότι σχεδόν όλες οι κλήσεις πρέπει να τις χειριστεί (σε κάποιο βαθμό, έστω μικρό) ο πυρήνας



Η Δομή των Μηνυμάτων

- ▶ Ο ορισμός της δομής των μηνυμάτων βρίσκεται στο αρχείο `/usr/src/include/minix/ipc.h`
- ▶ Συνολικά 7 διαφορετικοί τύποι μηνυμάτων

```
typedef struct {int m11, m12, m13; char *m1p1, *m1p2, *m1p3;} mess_1;
typedef struct {int m211, m212, m213; long m211, m212; char *m2p1;} mess_2;
typedef struct {int m311, m312; char *m3p1; char m3cal[M3_STRING];} mess_3;
typedef struct {long m41, m42, m43, m44, m45;} mess_4;
typedef struct {short m5c1, m5c2; int m511, m512; long m511, m512, m513;} mess_5;
typedef struct {int m711, m712, m713, m714; char *m7p1, *m7p2;} mess_7;
typedef struct {int m811, m812; char *m8p1, *m8p2, *m8p3, *m8p4;} mess_8;

typedef struct {
    int m_source; // who sent the message
    int m_type; // what kind of message is it
    union {
        mess_1 m_m1;
        mess_2 m_m2;
        mess_3 m_m3;
        mess_4 m_m4;
        mess_5 m_m5;
        mess_7 m_m7;
        mess_8 m_m8;
    } m_u;
} message;
```



Οι 7 Δομές Μηνυμάτων του Minix 3

m_source	m_source	m_source	m_source	m_source	m_source	m_source
m_type	m_type	m_type	m_type	m_type	m_type	m_type
m1_11	m2_11	m3_11	m4_11	m5_c1m5_c1	m7_11	m8_11
m1_12	m2_12	m3_12	m4_12	m8_11	m7_12	m8_12
m1_13	m2_13	m3_p1	m4_13	m5_12	m7_13	m8_p1
m1_p1	m2_11		m4_14	m5_11	m7_14	m8_p2
m1_p2	m2_12		m4_15	m5_12	m7_p1	m8_p3
m1_p3	m2_p1	m3_0a1		m5_13	m7_p2	m8_p4

- ▶ Το μέγεθος προκύπτει από την αρχιτεκτονική του συστήματος
- ▶ Το σχήμα απεικονίζει τις δομές για αρχιτεκτονική 32-bit



Αποστολή – Παραλαβή Μηνυμάτων

- ▶ Η υλοποίηση της Send – Receive γίνεται από την συνάρτηση `_sendrec()`
- ▶ Αναλαμβάνει να στείλει το μήνυμα στην διεργασία
- ▶ Ορίζεται στο αρχείο `src/include/minix/ipc.h`

```
#define sendrec _sendrec
__PROTOTYPE(int sendrec, (int src_dest, message *m_ptr))
```

- ▶ Υλοποιείτε σε επίπεδο assembly στο αρχείο `src/lib/i86/rts/_sendrec.s`
- ▶ Βασίζεται στην συνάρτηση `_sendrec` – επίσης υλοποιείτε σε επίπεδο assembly στο αρχείο `src/lib/i386/rts/_ipc.s`
- ▶ **Αντιγράφει το μήνυμα στην διεργασία – παραλήπτης και αμέσως μετά στέλνει ένα interrupt SYSVEC στον πυρήνα**
- ▶ Η εκτέλεση της διεργασίας – αποστολέας διακόπτεται και ενεργοποιείται η διεργασία – παραλήπτης



```
#include <lib.h>

PUBLIC int _syscall(int who, int syscallnr,
                    message *msgptr) {
    int status;
    msgptr->m_type = syscallnr;
    status = _sendrec(who, msgptr);
    if (status != 0) {
        /* 'sendrec' itself failed. */
        msgptr->m_type = status;
    }
    if (msgptr->m_type < 0) {
        errno = -msgptr->m_type;
        return(-1);
    }
    return(msgptr->m_type);
}
```



```
while (TRUE) {
    // Get work. Block until a message arrives
    receive(ANY, &m);
    call_nr = (unsigned) m.m_type - KERNEL_CALL;

    result = (*call_vec[call_nr])(&m);

    if (result != EDONTREPLY) {
        m.m_type = result;
        if (OK != (s=lock_send(m.m_source, &m)))
            kprintf("SYSTEM, reply to %d failed:
    }
}
```



```
#include <lib.h>

PUBLIC int _syscall(int who, int syscallnr,
                    message *msgptr) {
    int status;
    msgptr->m_type = syscallnr;
    status = _sendrec(who, msgptr);
    if (status != 0) {
        /* 'sendrec' itself failed. */
        msgptr->m_type = status;
    }
    if (msgptr->m_type < 0) {
        errno = -msgptr->m_type;
        return(-1);
    }
    return(msgptr->m_type);
}
```



[/usr/src/kernel/table.c](#)

```
#define PM_C ~(c(SYS_DEVIO) | c(SYS_SDEVIO)
| c(SYS_VDEVIO) | c(SYS_IRQCTL) | c(SYS_INT86))

#define FS_C (c(SYS_KILL) | c(SYS_VIRCOPY)
| c(SYS_VIRVCOPY) | c(SYS_UMAP) | c(SYS_GETINFO)
| c(SYS_EXIT) | c(SYS_TIMES) | c(SYS_SETALARM))

#define RS_C ~0
```



Σύνοψη 8^{ης} Διάλεξης

3η Άσκηση

Minix recompilation

System Task

Επικοινωνία με το System Task

Λειτουργικό Σύστημα Minix

Ανάθεση Μνήμης

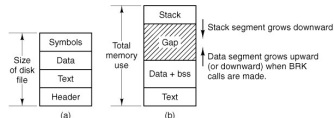
Ανταλλαγή Δεδομένων

Παράδειγμα: Αρχικοποίηση Σημάτων



Τμήματα Διεργασιών

- ▶ Οι διεργασίες στο Minix 3 χωρίζονται σε τρία τμήματα
 1. text -- ο κώδικα ("εκτελέσιμο"), δεν μεταβάλλεται
 2. data -- τα δεδομένα
 3. stack -- το stack εντολών
- ▶ Το τμήμα stack μεταβάλλεται προς το τμήμα data και αντίστροφα
- ▶ Το αρχικό μέγεθος ορίζεται από το αρχείο του εκτελέσιμου (στην επικεφαλίδα)
 - ▶ Με την εντολή `chmem` μπορούμε να μεταβάλουμε το αρχικό μέγεθος



Τμήματα Διεργασιών – Θέματα Υλοποίησης (1)

- ▶ Το μέγεθος του κάθε τμήματος μετριέται σε **clicks**
 - ▶ Κάθε click είναι 1024 bytes
- ▶ Στο αρχείο `/usr/src/include/minix/type.h` ορίζεται η δομή `mem_map` που περιγράφει το κάθε τμήμα
 - ▶ Εικονική θέση του τμήματος στην μνήμη (σε clicks)
 - ▶ Πραγματική θέση του τμήματος στην μνήμη (σε clicks)
 - ▶ Μέγεθος τμήματος (σε clicks)

```
struct mem_map {
    vir_clicks mem_vir;    /* virtual address */
    phys_clicks mem_phys; /* physical address */
    vir_clicks mem_len;    /* length */
};
```

- ▶ Η εικονική θέση και το μέγεθος μετριούνται σε *unsigned int*, η πραγματική θέση σε *unsigned long*



Τμήματα Διεργασιών – Θέματα Υλοποίησης (2)

- ▶ Η πληροφορία για τη θέση του κάθε τμήματος και το μέγεθος διατηρείται στον πυρήνα και στον διαχειριστή διεργασιών
- ▶ Ορίζεται ως ένας πίνακας
 - ▶ Η θέση 0 περιέχει το text (T)
 - ▶ Η θέση 1 περιέχει το data (D)
 - ▶ Η θέση 2 περιέχει το stack (S)

Πληροφορία στον Πυρήνα

```
struct mem_map p_mmap[NR_LOCAL_SEGS];
```

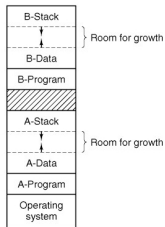
Πληροφορία στον Διαχειριστή Διεργασιών

```
struct mem_map mp_seg[NR_LOCAL_SEGS];
```

- ▶ Η σταθερά `NR_LOCAL_SEGS` είναι 3 και ορίζεται στο αρχείο `/usr/src/include/minix/const.h`
 - ▶ Επίσης ορίζει π 3 βοηθητικές σταθερές (T, D, S)



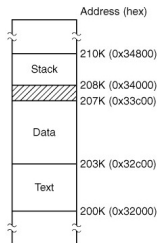
Τμήματα Διεργασιών – Μέγεθος Τμημάτων



- ▶ Κατά την αρχικοποίηση της διεργασίας ορίζονται τα 3 τμήματα (μέγεθος, θέση)
- ▶ Μπορεί ένα τμήμα να χρειαστεί περισσότερο χώρο (αφορά το data, stack)
 - ▶ Λογικό για το μεγαλύτερο μέρος των προγραμμάτων
- ▶ Αφήνουμε ένα 'κενό' μεταξύ data – stack
- ▶ Μειώνουμε το κόστος 'μεταφοράς' των τμημάτων σε νέες θέσεις μνήμης με μεγαλύτερο ελεύθερο χώρο



Τμήματα Διεργασιών – Παράδειγμα

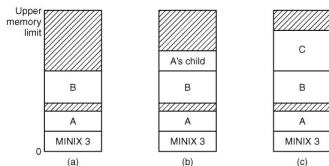


- ▶ Τα τρία τμήματα της διεργασίας:
 1. text – μέγεθος 3K
 2. data – μέγεθος 4K
 3. stack – μέγεθος 2K
- ▶ Κενό μεταξύ data – stack 1K
- ▶ Συνολικό μέγεθος 10K
- ▶ Περιεχόμενα πίνακα:

	Εικ.	Πραγ.	Μεγ.
Text	0	0xc8	0x3
Data	0	0xcb	0x4
Stack	0x5	0xad	0x2



Εκτέλεση Εντολών – Δημιουργία Νέων Διεργασιών



- ▶ Θέλουμε να εκτελέσουμε μια νέα εντολή
- ▶ Αρχικά χρησιμοποιούμε την κλήση του συστήματος fork
 - ▶ Αντιγράφονται τα τμήματα μνήμης της A
- ▶ Αμέσως μετά χρησιμοποιούμε την κλήση του συστήματος exec
 - ▶ Αντικαθίστανται τα τμήματα μνήμης από αυτά της C



Κοινά Τμήματα Μνήμης

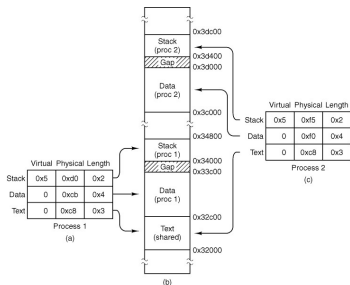
- ▶ Στα αλληλεπιδραστικά συστήματα μια διεργασία μπορεί να 'τρέχει' πολλές φορές
 - ▶ ... αντίγραφα του ίδιου κώδικα στην μνήμη
 - ▶ π.χ. χρησιμοποιώντας την κλήση του συστήματος fork
- ▶ Στην ουσία το τμήμα text είναι κοινό για όλες τις διεργασίες
- ▶ Θέλουμε να εντοπίσουμε κατά πόσο μια διεργασία χρησιμοποιεί το ίδιο τμήμα text
 - ▶ Ανατρέχουμε στο εκτελέσιμο της διεργασίας
 - ▶ Αν δεν έχουν γίνει αλλαγές, πρόκειται για αντίγραφο

```

/* File identification for sharing. */
ino_t mp_ino; // inode number of file
dev_t mp_dev; // device number of file system
time_t mp_ctime; // inode changed time
    
```



Κοινά Τμήματα Μνήμης – Παράδειγμα



Επικοινωνία Διεργασιών

- ▶ Το Λ.Σ. προσφέρει πολλούς τρόπους για την επικοινωνία μεταξύ διεργασιών
 - ▶ Μηνύματα
 - ▶ Σήματα
 - ▶ Sockets / Pipes
 - ▶ Διαμοιραζόμενη μνήμη (shared memory)
- ▶ Οι μηχανισμοί αυτοί είναι υψηλού επιπέδου
 - ▶ Παρέχουν σύνθετη λειτουργικότητα
 - ▶ Προϋποθέτουν συνθετων μηχανισμών
- ▶ Πως μπορούν 2 διεργασίες να επικοινωνήσουν χωρίς την χρήση αυτών των μηχανισμών;
 - ▶ ... θέλουμε ταχύτητα
 - ▶ ... θέλουμε καλύτερο έλεγχο
 - ▶ ... Δεν μπορούμε να χρησιμοποιήσουμε αυτούς τους μηχανισμούς

Επικοινωνία Διεργασιών – Απ' ευθείας Αντιγραφή Μνήμης

- ▶ Ο πιο βασικός τρόπος είναι μέσω απ' ευθείας πρόσβαση στη μνήμη
 - ▶ Οι μεταβλητές διατηρούνται στο data τμήμα μιας διεργασίας
 - ▶ Αρκεί να 'διαβάσουμε' τη θέση μνήμης που βρίσκεται μια μεταβλητή ...
- ▶ Όμως είναι αυτός ο τρόπος ασφαλής;
- ▶ Οι διεργασίες χειρίζονται τα τμήματα μνήμης τους μέσω των εικονικών διευθύνσεων
 - ▶ πως μπορούν να εντοπίσουν την θέση μνήμης σε μια άλλη διεργασία;

Αντιγραφή Μνήμης

- ▶ Ο πυρήνας παρέχει μηχανισμούς (κλήσεις πυρήνα) για την αντιγραφή μνήμης
- ▶ Ασφάλεια:
 - ▶ Αρχιτεκτονική Μικρο-πυρήνα – μόνο οι διαχειριστές / οδηγοί μπορούν να καλέσουν αυτούς τους μηχανισμούς
- ▶ Εικονικές Διευθύνσεις:
 - ▶ Ο πυρήνας γνωρίζει τις πραγματικές διευθύνσεις που έχουν τα τμήματα των διεργασιών ...
 - ▶ Μετατρέπει τις εικονικές διευθύνσεις σε πραγματικές

```
_PROTOTYPE(int sys_vircopy, (int src_proc,
    int src_seg, vir_bytes src_vir,
    int dst_proc, int dst_seg, vir_bytes dst_vir,
    phys_bytes bytes));
```

- ▶ Αντιγραφή από – διεργασία ‘πηγή’ (source)
 - ▶ src_proc – θέση διεργασίας στον πίνακα διεργασιών
 - ▶ src_seg -- τμήμα μνήμης διεργασίας (text, data, stack)
 - ▶ src_vir -- εικονική θέση μνήμης στη διεργασία
- ▶ Αντιγραφή προς – διεργασία ‘προορισμός’ (destination)
 - ▶ dst_proc – θέση διεργασίας στον πίνακα διεργασιών
 - ▶ dst_seg -- τμήμα μνήμης διεργασίας (text, data, stack)
 - ▶ dst_vir -- εικονική θέση μνήμης στη διεργασία
- ▶ bytes -- Πλήθος bytes προς αντιγραφή



```
/usr/src/lib/syslib/sys_vircopy.c
```

```
PUBLIC int sys_vircopy(src_proc, src_seg, src_vir,
    dst_proc, dst_seg, dst_vir,
    bytes)
int src_proc; /* source process */
int src_seg; /* source memory segment */
vir_bytes src_vir; /* source virtual address */
int dst_proc; /* destination process */
int dst_seg; /* destination memory segment */
vir_bytes dst_vir; /* destination virtual address */
phys_bytes bytes; /* how many bytes */
{
    ...
}
```



```
/usr/src/lib/syslib/sys_vircopy.c
```

```
{
    message copy_mess;

    if (bytes == 0L) return(OK);
    copy_mess.CP_SRC_ENDPT = src_proc;
    copy_mess.CP_SRC_SPACE = src_seg;
    copy_mess.CP_SRC_ADDR = (long) src_vir;
    copy_mess.CP_DST_ENDPT = dst_proc;
    copy_mess.CP_DST_SPACE = dst_seg;
    copy_mess.CP_DST_ADDR = (long) dst_vir;
    copy_mess.CP_NR_BYTES = (long) bytes;
    return(_taskcall(SYSTASK, SYS_VIRCOPY, &copy_mess));
}
```



- ▶ Το αρχείο /usr/include/minix/syslib.h
 - ▶ ορίζει ‘παρалаγές’ της sys_vircopy
- ▶ Λιγότερες παράμετροι ανάλογα την περίπτωση
 1. sys_biosin
 2. sys_biosout
 3. sys_datacopy
 4. sys_textcopy
 5. sys_stackcopy
 6. sys_abscopy
- ▶ Όλες καταλήγουν στην do_copy


```
/usr/src/kernel/system/do_copy.c
```



- ▶ Στο αρχείο `/usr/src/kernel/system.h` ορίζονται οι handlers για τις κλήσεις πυρήνα

```
_PROTOTYPE( int do_copy, (message *m_ptr) );
#define do_vircopy do_copy
#define do_physcopy do_copy
```

- ▶ Στο αρχείο `/usr/src/kernel/system.c` γίνεται η αντιστοίχιση των handlers με τους αριθμούς των κλήσεων πυρήνα

```
map(SYS_VIRCOPY, do_vircopy);
map(SYS_PHYSCOPY, do_physcopy);
```



- ▶ Θεωρεί ότι το μήνυμα είναι τύπου m5

- ▶ τύπος (`m_type`) -- `SYS_VIRCOPY` ή `SYS_PHYSCOPY`
- ▶ `m5_c1` (`CP_SRC_SPACE`) -- source virtual segment
- ▶ `m5_l1` (`CP_SRC_ADDR`) -- source offset within segment
- ▶ `m5_i1` (`CP_SRC_PROC_NR`) -- source process number
- ▶ `m5_c2` (`CP_DST_SPACE`) -- destination virtual segment
- ▶ `m5_l2` (`CP_DST_ADDR`) -- destination offset within segment
- ▶ `m5_i2` (`CP_DST_PROC_NR`) -- destination process number
- ▶ `m5_l3` (`CP_NR_BYTES`) -- number of bytes to copy



Συνάρτηση -- do_copy (1)

- ▶ Μετατρέπει τις παραμέτρους του μηνύματος σε 2 μεταβλητές τύπου `vir_addr`

```
PUBLIC int do_copy(m_ptr)
register message *m_ptr;
{
    struct vir_addr vir_addr[2];
    phys_bytes bytes;

    vir_addr[_SRC_].proc_nr_e = m_ptr->CP_SRC_ENDPT;
    vir_addr[_SRC_].segment = m_ptr->CP_SRC_SPACE;
    vir_addr[_SRC_].offset = (vir_bytes) m_ptr->CP_SRC_ADD;
    vir_addr[_DST_].proc_nr_e = m_ptr->CP_DST_ENDPT;
    vir_addr[_DST_].segment = m_ptr->CP_DST_SPACE;
    vir_addr[_DST_].offset = (vir_bytes) m_ptr->CP_DST_ADD;
    bytes = (phys_bytes) m_ptr->CP_NR_BYTES;
```



Συνάρτηση -- do_copy (2)

- ▶ Ελέγχει τις διευθύνσεις -- αν αναφέρονται σε 'πραγματικές' διεργασίες
 - ▶ Χρησιμοποιεί την συνάρτηση `isokendpt_f`
 - ▶ ... ορίζεται στο αρχείο `/usr/src/kernel/proc.c`
 - ▶ Στην ουσία μετατρέπει το αριθμό end-point σε αριθμό διεργασίας (θέση στον πίνακα διεργασιών)
 - ▶ Αν η διεργασία δεν εντοπιστεί -- ή δεν είναι ζωντανή -- επιστρέφει έναν αρνητικό αριθμό
- ▶ Χρησιμοποιεί την συνάρτηση `virtual_copy`
 - ▶ ορίζεται στο αρχείο `/usr/src/kernel/system.c`
 - ▶ Μετατρέπει τις εικονικές διευθύνσεις σε πραγματικές με την χρήση της συνάρτησης `umap_local`
 - ▶ Χρησιμοποιεί την συνάρτηση `phys_copy` για την τελική αντιγραφή -- ενέλεδο assembly (`klib386.s`)



Προετοιμασία Σημάτων

- ▶ Οι ρυθμίσεις ενός τύπου σήματος γίνεται με μεταβλητές τύπου `sigaction`

```
struct sigaction {
    /* SIG_DFL, SIG_IGN, or pointer to function */
    __sighandler_t sa_handler;
    /* signals to be blocked during handler */
    sigset_t sa_mask;
    /* special flags */
    int sa_flags;
};
```

- ▶ Ο signal handler ορίζεται ως
`typedef void _PROTOTYPE((*__sighandler_t),
(int));`



Προετοιμασία Σημάτων – Παράδειγμα

Αρχείο `testsignal.c`

```
void catch_int(int sig_num) {
    printf("Don't do that...");
}

int main() {
    struct sigaction sig;
    sigset_t sset;
    sigemptyset(&sset);
    sig.sa_flags = 0;
    sig.sa_handler = catch_int;
    sig.sa_mask = sset;
    sigaddset(&sig.sa_mask, SIGINT);
    sigaction(SIGINT, &sig, NULL);
}
```



Βοηθητική Συνάρτηση `sigaction`

- ▶ Ορίζεται στο αρχείο `/usr/src/lib/posix/_sigaction.c`

```
PUBLIC int sigaction(sig, act, oact)
int sig;
_CONST struct sigaction *act;
struct sigaction *oact;
{
    message m;
    m.ml_i2 = sig;
    m.ml_p1 = (char *) act;
    m.ml_p2 = (char *) oact;
    m.ml_p3 = (char *) __sigreturn;
    return(_syscall(MM, SIGACTION, &m));
}
```



Κλήση Συστήματος `SIGACTION` (1)

```
PUBLIC int do_sigaction() {
    int r;
    struct sigaction svec;
    struct sigaction *svp;
    ...
    r = sys_datacopy(PM_PROC_NR,
        (vir_bytes) svp,
        who_e, (vir_bytes) m_in.sig_osa,
        (phys_bytes) sizeof(svec));
    ...
}
```



Κλήση Συστήματος SIGACTION (2)

```
...  
  
/* Read in the sigaction structure. */  
r = sys_datacopy(who_e,  
                (vir_bytes) m_in.sig_nsa,  
                PM_PROC_NR, (vir_bytes) &svec,  
                (phys_bytes) sizeof(svec));  
  
...  
}
```

