

Αφού ξεκινήσω το Minix:

```
done.

Minix Release 3 Version 2.0 (console)

192.168.1.103 login: root
Password:


To install additional packages, run 'pkgin'.

To install packages from the online package repository, if you have a
working network connection from MINIX: first do a 'pkgin update' to
update the list of available packages, and then do a 'pkgin' to get a
list of commands. For example, 'pkgin install vim' installs the 'vim'
package, and 'pkgin available' will list all available packages.

To install packages from the installation CD: same, but use pkgin_cd.
To switch to the online repository, do 'pkgin update' again.

MINIX 3 supports multiple virtual terminals. Just use ALT+F1, F2, F3
and F4 to navigate among them.

For more information on how to use MINIX 3, see the wiki:
http://wiki.minix3.org.

#  Γραμμη εντολων
```

ΠΑΝΤΑ μπαίνω στο φάκελο “/usr/src”

(Αυτό γιατί μέσα εκεί είναι όλοι οι **υποφάκελοι**, που αντιστοιχούν στα **επίπεδα-διεργασίες** του Minix. ΒΛΕΠΕ ΑΡΧΕΙΟ: check.pdf)

```
# cd /usr/src
# _
```

(ΓΕΝΙΚΗ ΠΑΡΑΤΗΡΗΣΗ όταν θελω να «βγει» το ποντίκι από το Minix, τότε παταω τα 2 πλήκτρα: Control και μετά το Alt, δηλ: **Ctrl και Alt**.

Προφανώς, μετά αν θέλω να «μπω» πάλι στο Minix τότε αρκεί ένα απλό κλικ πανω στη μαυρη οθόνη του παραθύρου.)

Στα παρακάτω θα εξηγήσω με χρήση “vi” (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf) κάποια πράγματα που θα ΠΡΕΠΕΙ να ξερετε. Τα “print screen” θα τα κανω αρχικά σε Minix και επείτα από περιβαλλον Windows. Δηλαδή, αφού τα έχετε μεταφέρει σαν αρχεία (όσα χρειαστούν) και εκτός του Minix. ΠΡΕΠΕΙ να ξερετε vi, απλα τις αλλαγές θα τις κάνετε έξω από το Minix για να ειστε σιγουροι, οποτε μετα ξαναφερετε τα αρχεια μεσα στο minix.

ΜΕΡΟΣ 1

1) Στην εκφωνηση (ΒΛΕΠΕ ΑΡΧΕΙΟ: ekfonisi.pdf) εχει την παρακατω παραγραφο.

ΑΥΤΗ η παραγραφος θα εξηγηθει παρακατω αναλυτικα, οποτε οσα εχουν φοντο **κιτρινο** είναι κομματα από αυτή την παραγραφο.

Όταν μια διεργασία καλεί **fork()** για να δημιουργήσει μια καινούρια διεργασία, τότε ο διακομιστής διεργασιών (**pm**) καλεί την συνάρτηση **do_fork()** στο αρχείο **forkexit.c**. Η συνάρτηση αυτή καλεί **vm_fork()** για να επικοινωνήσει με τον διακομιστή μνήμης (**vm**) που αναλαμβάνει να δεσμεύσει την μνήμη και μετά να ενημερώσει τον πυρήνα για την νέα διεργασία. Στην συνέχεια ο **pm** καλεί **tell_vfs()** για να επικοινωνήσει με τον διακομιστή συστήματος αρχείων (**vfs**). Όταν ολοκληρωθεί η κλήση του **vfs** και λάβει ασύγχρονη απάντηση ο **pm**, τότε ζητάει από τον διακομιστή **sched** να χρονοδομολογήσει τη διεργασία στον πυρήνα.

Όταν μια διεργασία καλεί fork(): Εδώ θεωρούμε ότι ξεκινάει μία νέα **διεργασία χρήστη**, το οποίο γίνεται στο επίπεδο 4 (**Layer 4**, ΒΛΕΠΕ ΑΡΧΕΙΟ: check.pdf)

[Αυτό τελικά ΘΑ γίνει στο ερώτημα 4 της ασκήσης, που τότε ΟΝΤΩΣ θα ξεκινήσουμε μια τετοια **διεργασία χρηστη**. ΑΛΛΑ προς το παρον θεωρούμε ότι ξεκινάει μια **διεργασία χρήστη**, ΧΩΡΙΣ να μας νοιάζει ποια είναι αυτή].

Αυτό που μας ενδιαφέρει ΤΩΡΑ είναι ΤΙ θα κάνει ο ΔΙΑΧΕΙΡΙΣΤΗΣ διεργασιων PM (**Layer 3**) όταν θα καταλάβει ότι τον «ζητησει» (μεσω μηνύματος: ΒΛΕΠΕ ΑΡΧΕΙΟ: check.pdf) μια τυχαια διεργασια χρηστη.

Αυτό που θα κάνει τότε ο PM, είναι ότι καλεί τη συνάρτηση **do_fork** στο αρχείο **forkexit.c**.

Μπαίνω με τον “vi” σε αυτό το αρχείο:

```
# vi servers/pm/forkexit.c (+ENTER)
```

Ψάχνω εκεί μέσα τη συναρτηση **do_fork** κανοντας **/do_fork (+ENTER)** (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf)

```

/* This file deals with creating processes (via FORK) and deleting them (via
 * EXIT/WAIT). When a process forks, a new slot in the 'mproc' table is
 * allocated for it, and a copy of the parent's core image is made for the
 * child. Then the kernel and file system are informed. A process is removed
 * from the 'mproc' table when two events have occurred: (1) it has exited or
 * been killed by a signal, and (2) the parent has done a WAIT. If the process
 * exits first, it continues to occupy a slot until the parent does a WAIT.
 *
 * The entry points into this file are:
 * do_fork:      perform the FORK system call
 * do_srv_fork:  special FORK, used by RS to create sys services
 * do_exit:      perform the EXIT system call (by calling exit_proc())
 * exit_proc:    actually do the exiting, and tell VFS about it
 * exit_restart: continue exiting a process after VFS has replied
 * do_waitpid:   perform the WAITPID or WAIT system call
 * wait_test:    check whether a parent is waiting for a child
 */

#include "pm.h"
#include <sys/wait.h>
#include <assert.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/sched.h>
do_fork_

```

Για να μην γραφω συνέχεια (+ENTER) , παταω παντα ENTER αφου γραψω κατι. Αν νωρίτερα πατησετε λαθος π.χ. καποιο γραμμα τότε σβηνει κανονικα με το πληκτρο "Backspace" κι ας φαίνεται ότι δεν σβηνει. Επισης γενικα μπορείτε να πατατε το πληκτρο "escape" (Esc) , το οποιο αναιρει οσα γραψατε και ξαναπατατε μετα π.χ. /do_fork (+ENTER)

Αφου πατησετε /do_fork (βλεπε πανω φώτο με κοκκινο), θα παει ο κέρσορας στην πρωτη εμφανιση της. Μπορει με το ματι να μην βλεπετε τον κερσορα που πηγε, οποτε παταμε εναλλαξ τα βελακια (αριστερο-δεξι) από το πληκτρολογιο για να καταλαβετε που είναι ο κερσορας μεσω της κινησης του (αριστερα-δεξια) εναλλαξ γρηγορα.

Για να ξαναψαξω την «do_fork» ΑΥΤΟΜΑΤΑ, απλα πατατε το μικρο γραμμα η (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf) και ουτω καθεξης.

Μεσα στην do_fork υπαρχουν η vm_fork και tell_vfs (αυτή μας νοιάζει τελικά μόνο).

```

rmc->mp_child_stime = 0;          /* reset administration */
rmc->mp_exitstatus = 0;
rmc->mp_sigstatus = 0;
rmc->mp_endpoint = child_ep;      /* passed back by UM */
for (i = 0; i < NR_ITIMERS; i++)
    rmc->mp_interval[i] = 0;      /* reset timer intervals */

/* Find a free pid for the child and put it in the table. */
new_pid = get_free_pid();
rmc->mp_pid = new_pid;           /* assign pid to child */

m.m_type = PM_FORK;
m.m_proc = rmc->mp_endpoint;
m.m_pproc = rmc->mp_endpoint;
m.m_cpuid = rmc->mp_pid;
m.m_reuid = -1;                 /* Not used by PM_FORK */
m.m_regid = -1;                 /* Not used by PM_FORK */

tell_vfs(rmc, m);

#ifdef USE_TRACE
/* Tell the tracer, if any, about the new child */
if (rmc->mp_tracer != NO_TRACER)
    sig_proc(rmc, SIGSTOP, TRUE /*trace*/, FALSE /* ksig */);

```

τύπος μηνύματος που θα στείλει ο PM στον VFS, οπότε πρέπει να ψάξω στο Minix με search να βρω σε ποια αρχεία υπάρχει αυτό το PM_FORK

το μήνυμα είναι η μεταβλητή m στον κώδικα

Βγαίνουμε από το αρχείο με την εντολή :q! (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf)

```
rmc->mp_child_stime = 0;          /* reset administration */
rmc->mp_exitstatus = 0;
rmc->mp_sigstatus = 0;
rmc->mp_endpoint = child_ep;      /* passed back by VM */
for (i = 0; i < NR_ITIMERS; i++)
    rmc->mp_interval[i] = 0;      /* reset timer intervals */

/* Find a free pid for the child and put it in the table. */
new_pid = get_free_pid();
rmc->mp_pid = new_pid;            /* assign pid to child */

m.m_type = PM_FORK;
m.PM_PROC = rmc->mp_endpoint;
m.PM_PPROC = rmp->mp_endpoint;
m.PM_CPID = rmc->mp_pid;
m.PM_REUID = -1;                 /* Not used by PM_FORK */
m.PM_REGID = -1;                 /* Not used by PM_FORK */

tell_vfs(rmc, &m);

#if USE_TRACE
/* Tell the tracer, if any, about the new child */
if (rmc->mp_tracer != NO_TRACER)
    sig_proc(rmc, SIGSTOP, TRUE /*trace*/, FALSE /* ksig */);
:q!_
```

Είμαι στη γραμμή εντολών (βγηνκα από το αρχείο) και κανω “search” για να βρω το PM_FORK

```
/* Tell the tracer, if any, about the new child */
if (rmc->mp_tracer != NO_TRACER)
    sig_proc(rmc, SIGSTOP, TRUE /*trace*/, FALSE /* ksig */);

#
#
#
#
#
# search PM_FORK . : more
#define PM_FORK (PM_RQ_BASE + 7) /* Newly forked process */
#define PM_FORK_REPLY (PM_RS_BASE + 7)
/* Additional parameters for PM_FORK and PM_SRV_FORK */
-rw-r--r-- 1 root operator 51162 Feb 28 2012 ./include/minix/com.h
m.m_type = PM_FORK;
m.PM_REUID = -1; /* Not used by PM_FORK */
m.PM_REGID = -1; /* Not used by PM_FORK */
-rw-r--r-- 1 root operator 25403 Feb 28 2012 ./servers/pm/forkexit.c
case PM_FORK_REPLY:
case PM_FORK_REPLY:
-rw-r--r-- 1 root operator 15931 Feb 28 2012 ./servers/pm/main.c
case PM_FORK:
m_out.m_type = PM_FORK_REPLY;
-rw-r--r-- 1 root operator 27525 Feb 28 2012 ./servers/vfs/main.c
#
#
```

λέξη φάκελος (η τελεία σημαίνει ότι ψάχνω στο φακελο που ειμαι, αρα στον /usr/src)

το βαζω για να τα βγαλει λιγα λιγα (αν ειναι πολλα, αλλα εδω τελικα ΔΕΝ ηταν πολλα. Αν ηταν πολλα θα πατουσατε συνεχως Enter μεχρι να τα βγαλει ΟΛΑ)

τα αρχεία μέσα στα οποία το βρήκε, θα χρειαστει να μπω στο "κιτρινο"

Μπαίνω στον κιτρινο γιατι στην εφκωνηση λεει «Όταν ολοκληρωθεί η κλήση του **vfs** και λάβει ασύγχρονη απάντηση ο **pm**», αρα ειμαι στο αρχείο **servers/pm/main.c** γιατι παρατηρηστε μεσα εκει εχει το case **PM_FORK_REPLY** [που σημαινει ότι του ηρθε απαντηση (**REPLY**) από τον VFS, γιατι θυμηθειτε ότι ο τυπος μηνυματος που κοιτουσαμε στη do_fork στο αρχειο forkexit.c ηταν **PM_FORK**]

Οποτε:

```
# vi servers/pm/main.c_
```

Μολις ανοιξει θα ψάξω με /PM_FORK_REPLY (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf).

```
/* This file contains the main program of the process manager and some related
 * procedures. When MINIX starts up, the kernel runs for a little while,
 * initializing itself and its tasks, and then it runs PM and VFS. Both PM
 * and VFS initialize themselves as far as they can. PM asks the kernel for
 * all free memory and starts serving requests.
 *
 * The entry points into this file are:
 *   main:      starts PM running
 *   setreply:  set the reply to be sent to process making an PM system call
 */

#include "pm.h"
#include <minix/keymap.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/ds.h>
#include <minix/type.h>
#include <minix/endpoint.h>
#include <minix/minlib.h>
#include <minix/type.h>
#include <minix/vm.h>
#include <minix/crtso.h>
#include <signal.h>
#include <stdlib.h>
/PM_FORK_REPLY_
```

(Από εδώ και κατω τα print screen θα είναι από Windows, για να τα ψαχνω πιο ανετα. Και εσεις εννοειται ότι θα κανετε τις αλλαγες σε Windows (ή Linux ή Mac) για μεγαλυτερη ασφαλεια.)

Ξαναψάχνω το PM_FORK_REPLY με το γραμμα n (ΒΛΕΠΕ ΑΡΧΕΙΟ: vi.pdf) και τελικα οδηγουμαι εδώ:

```
switch(call_nr)
{
case PM_SETUID_REPLY:
case PM_SETGID_REPLY:
case PM_SETSID_REPLY:
case PM_EXEC_REPLY:
case PM_EXIT_REPLY:
case PM_CORE_REPLY:
case PM_FORK_REPLY:
case PM_SRV_FORK_REPLY:
case PM_UNPAUSE_REPLY:
case PM_REBOOT_REPLY:
case PM_SETGROUPS_REPLY:
if (who_e == VFS_PROC_NR)
{
handle_vfs_reply();
result= SUSPEND;      /* don't reply */
}
else
result= ENOSYS;
break;
case COMMON_GETSYSINFO:
result = do_getsysinfo();
break;
default:
/* Else, if the system call number is valid, perform the
 * call.
```

Αυτη η συνθήκη σημαίνει ότι έλαβε απαντηση ο pm (γιατί ειμαστε μέσα στο αρχείο servers/pm/main.c) απο τον VFS. Εκτελειται η handle_vfs_reply

Ψάχνω με /handle_vfs_reply και οδηγουμαι εδώ:

```
/*=====
* handle_vfs_reply *
*=====
PRIVATE void handle_vfs_reply()
{
    struct mproc *rmp;
    endpoint_t proc_e;
    int r, proc_n;

    /* PM_REBOOT is the only request not associated with a process.
     * Handle its reply first
     */
}
```

Μεσα στον κωδικα της παρατηρω ότι εχει το case PM_FORK_REPLY, το οποιο θελουμε:

```
case PM_FORK_REPLY:
    /* Schedule the newly created process ... */
    r = (OK);
    if (rmp->mp_scheduler != KERNEL && rmp->mp_scheduler != NONE) {
        r = sched_start_user(rmp->mp_scheduler, rmp);
    }

    /* If scheduling the process failed, we want to tear down the process
     * and fail the fork */
    if (r != (OK)) {
        /* Tear down the newly created process */
        rmp->mp_scheduler = NONE; /* don't try to stop scheduling */
    }
}
```

Ο scheduler είναι ο SCHED (που θελουμε να στείλουμε μήνυμα τελικά στο α' ερωτημα), αρα μπαίνει στο "if" και καλει την sched_start_user

Ψάχνω με /sched_start_user και ΔΕΝ τη βρίσκει μέσα στο αρχείο. Αρα πρέπει να βγω και να κανω search στο minix. Βγαίνω (:q!) και κανω μετα search.

```
# search sched_start_user . : more
      r = sched_start_user(rmp->mp_scheduler, rmp);
-rw-r--r-- 1 root operator 15931 Feb 28 2012 ./servers/pm/main.c
_PROTOPTYPE( int sched_start_user, (endpoint_t ep, struct mproc *rmp) );
-rw-r--r-- 1 root operator 3600 Feb 28 2012 ./servers/pm/proto.h
*
PUBLIC int sched_start_user(endpoint_t ep, struct mproc *rmp)
-rw-r--r-- 1 root operator 3502 Feb 28 2012 ./servers/pm/schedule.c
#
```

εδω είναι ο κωδικας της

Μπαίνω με "vi" σε αυτό:

```
# vi servers/pm/schedule.c_
```

ΕΠΕΙΤΑ:

```
#include "pm.h"
#include <assert.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/config.h>
#include <minix/sched.h>
#include <minix/sysinfo.h>
#include <minix/type.h>
#include <machine/archtypes.h>
#include <lib.h>
#include "mproc.h"

#include <machine/archtypes.h>
#include <timers.h>
#include "kernel/proc.h"

/*=====
 *                               init_scheduling                               *
 *=====*/
PUBLIC void sched_init(void)
{
    struct mproc *trmp;
    endpoint_t parent_e;
    int proc_nr, s;
    /sched_start_user_
```

και οδηγουμε εδώ:

```
/*
 *                               sched_start_user                               *
 *=====*/
PUBLIC int sched_start_user(endpoint_t ep, struct mproc *rmp)
{
    unsigned maxprio;
    endpoint_t inherit_from;
    int rv;

    /* convert nice to priority */
    if ((rv = nice_to_priority(rmp->mp_nice, &maxprio)) != OK) {
        return rv;
    }

    /* scheduler must know the parent, which is not the case for a child
     * of a system process created by a regular fork; in this case the
     * scheduler should inherit settings from init rather than the real
     * parent
     */
    if (mproc[rmp->mp_parent].mp_flags & PRIV_PROC) {
        assert(mproc[rmp->mp_parent].mp_scheduler == NONE);
        inherit_from = INIT_PROC_NR;
    } else {
        inherit_from = mproc[rmp->mp_parent].mp_endpoint;
    }

    /* inherit quantum */
    return sched_inherit(ep,
        rmp->mp_endpoint,
        inherit_from,
        maxprio,
        &rmp->mp_scheduler);
    /* scheduler_e */
    /* schedulee_e */
    /* parent_e */
    /* maxprio */
    /* *newsched_e */
}
```

την ψάχνω να δω τι κάνει

Κανω /sched_inherit και ΔΕΝ τη βρίσκω. Αρα πρέπει να βγω και να κανω search στο minix.
Βγαίνω (:q!) και κανω μετα search.

```
# search sched_inherit . : more
_PROTOTYPE(int sched_inherit, (endpoint_t scheduler_e,
-rw-r--r-- 1 root operator 496 Feb 28 2012 ./include/minix/sched.h
*
PUBLIC int sched_inherit(endpoint_t scheduler_e,
-rw-r--r-- 1 root operator 2821 Feb 28 2012 ./lib/libsys/sched_start.c
return sched_inherit(ep,
/* scheduler_e */
-rw-r--r-- 1 root operator 3582 Feb 28 2012 ./servers/pm/schedule.c
#
_
```

εδώ είναι ΜΟΝΟ η
δήλωση της, θα την
αλλάξω παρακάτω

εκεί ήμουν πριν, τελικά θα χρειαστεί να της περάσω ένα ακόμη
όρισμα π.χ. στην τελευταία θέση

εδώ είναι ο κώδικας
της, θα τον αλλάξω

Μπαίνω στο sched_start.c

```
# vi lib/libsys/sched_start.c_
```

και τελικά παρατηρω τα εξής:

```
/* =====
*
* sched_inherit
*
* =====
PUBLIC int sched_inherit(endpoint_t scheduler_e,
endpoint_t schedulee_e, endpoint_t parent_e, unsigned maxprio,
endpoint_t *newscheduler_e)
{
int rv;
message m;

assert(_ENDPOINT_P(scheduler_e) >= 0);
assert(_ENDPOINT_P(schedulee_e) >= 0);
assert(_ENDPOINT_P(parent_e) >= 0);
assert(maxprio >= 0);
assert(maxprio < NR_SCHED_QUEUES);
assert(newscheduler_e);

m.SCHEDULING_ENDPOINT = schedulee_e;
m.SCHEDULING_PARENT = parent_e;
m.SCHEDULING_MAXPRIO = (int) maxprio;

/* Send the request to the scheduler */
if ((rv = _taskcall(scheduler_e, SCHEDULING_INHERIT, m)) {
return rv;
}

/* Store the process' scheduler. Note that this might not be the
* scheduler we sent the SCHEDULING_INHERIT message to. That scheduler
* might have forwarded the scheduling message on to another scheduler
* before returning the message.
*/
*newscheduler_e = m.SCHEDULING_SCHEDULER;
return (OK);
}
```

Επίσης παρατηρω ότι οι τιμες που κυκλωσα είναι ορισματα, αρα
και εγω πρέπει να βαλω ενα ορισμα ακομα που θελω για τον
κωδικο ομαδας (ΒΛΕΠΕ: ekfonisi.pdf, ερωτημα "α")

Αυτά τα πεδία πρέπει να τα ψάξετε ξεχωριστα την κάθε λέξη στο
minix, ώστε να δείτε οτι αντιστοιχουν στα:

m9_I1, m9_I3, m9_I4.

Απο αυτο καταλαβαινω οτι ο τυπος του μηνυματος είναι ο "m9"
(ΒΛΕΠΕ: check.pdf) και οτι εχω ΕΛΕΥΘΕΡΑ-ΔΙΑΘΕΣΙΜΑ τα m9_I2
και m9_I5

εδώ στέλνεται ΤΕΛΙΚΑ το
μηνυμα απο τον pm
στον sched

Κάνω τις παρακατω αλλαγες στον κωδικα του:

```
/*=====
 *          sched_inherit          *
 *=====*/
PUBLIC int sched_inherit(endpoint_t scheduler_e,
    endpoint_t schedulee_e, endpoint_t parent_e, unsigned maxprio,
    endpoint_t *newscheduler_e, int kodikos_omadas)
{
    int rv;
    message m;

    assert(_ENDPOINT_P(scheduler_e) >= 0);
    assert(_ENDPOINT_P(schedulee_e) >= 0);
    assert(_ENDPOINT_P(parent_e) >= 0);
    assert(maxprio >= 0);
    assert(maxprio < NR_SCHED_QUEUES);
    assert(newscheduler_e);

    m.SCHEDULING_ENDPOINT = schedulee_e;
    m.SCHEDULING_PARENT = parent_e;
    m.SCHEDULING_MAXPRIO = (int) maxprio;
    m.m9_l2 = kodikos_omadas;

    /* Send the request to the scheduler */
    if ((rv = _taskcall(scheduler_e, SCHEDULING_INHERIT, &m))) {
        return rv;
    }

    /* Store the process' scheduler. Note that this might not be the
     * scheduler we sent the SCHEDULING_INHERIT message to. That scheduler
     * might have forwarded the scheduling message on to another scheduler
     * before returning the message.
     */
    *newscheduler_e = m.SCHEDULING_SCHEDULER;
    return (OK);
}
```

Ειναι ακεραιος (int) και το εβαλα τυχαia τελευταιιο ορισμα. Μπορειτε ο καθενας να το βαλει σε οποια θεση θελει, αλλα θα το βαλει στην αντιστοιχη θεση στα αλλα αρχεια που θα δειτε παρακατω

Αντιστοιχα ο καθενας σας μπορει να του δωσει αλλο ονομα π.χ. omada, xristis, odigos

Περνάω στο m9_l2 τον κωδικο

(θα μπορουσα στο m9_l5 που και αυτο ειναι διαθεσιμο)

Βγαίνετε (:q!) και μετα παω στο άλλο αρχείο:

```
# vi include/minix/sched.h_
```

```
#ifndef _MINIX_SCHED_H
#define _MINIX_SCHED_H

#include <minix/ipc.h>

_PROTOTYPE(int sched_stop, (endpoint_t scheduler_e, endpoint_t schedulee_e));
_PROTOTYPE(int sched_start, (endpoint_t scheduler_e, endpoint_t schedulee_e,
    endpoint_t parent_e, int maxprio, int quantum, int cpu,
    endpoint_t *newscheduler_e));
_PROTOTYPE(int sched_inherit, (endpoint_t scheduler_e,
    endpoint_t schedulee_e, endpoint_t parent_e, unsigned maxprio,
    endpoint_t *newscheduler_e, int kodikos_omadas));

#endif /* _MINIX_SCHED_H */
```

πρεπει να βαλετε το ιδιο ονομα που ειχατε δωσει και στην ιδια θεση που το ειχατε βαλει (εγω το ειχα βαλει στην τελευταια θεση)

Βγαίνετε (:q!) και μετα παω στο άλλο αρχείο:

```
# vi servers/pm/schedule.c_
```

και παω στην sched_inherit να περασω (στην τελευταία θέση σε μένα) την τιμή που χρειάζεται το «α» ερωτήμα.

```
/* ===== sched_start_user ===== */
/* ===== */
PUBLIC int sched_start_user(endpoint_t ep, struct mproc *rmp)
{
    unsigned maxprio;
    endpoint_t inherit_from;
    int rv;

    /* convert nice to priority */
    if ((rv = nice_to_priority(rmp->mp_nice, &maxprio)) != OK) {
        return rv;
    }

    /* scheduler must know the parent, which is not the case for a child
     * of a system process created by a regular fork; in this case the
     * scheduler should inherit settings from init rather than the real
     * parent
     */
    if (mproc[rmp->mp_parent].mp_flags & PRIV_PROC) {
        assert(mproc[rmp->mp_parent].mp_scheduler == NONE);
        inherit_from = INIT_PROC_NR;
    } else {
        inherit_from = mproc[rmp->mp_parent].mp_endpoint;
    }

    /* inherit quantum */
    return sched_inherit(ep,
                        rmp->mp_endpoint, /* scheduler_e */
                        inherit_from, /* schedlee_e */
                        maxprio, /* parent_e */
                        &rmp->mp_scheduler, /* maxprio */
                        rmp->mp_procgrp); /* *newsched_e */
}
```

Το mp_procgrp είναι αυτό που θέλουμε (υπάρχει στο αρχείο: servers/pm/mproc.h ΜΕΣΑ στο struct mproc). Για να το παρουμε εχουμε τον δείκτη rmp που δείχνει σε αυτό το struct.

Το εβαλα στην τελευταία θέση (γιατί το είχα βάλει εκεί και στα άλλα αρχεία)

Μετά από αυτές τις αλλαγές στα 3 αρχεία, το μήνυμα (με επιπλέον τον κωδικό ομάδας) έχει ΦΥΓΕΙ πλέον από τον pm και παει στον sched.

Τώρα μένει να παρουμε (λαβουμε) το μήνυμα στον sched και να εμφανίσουμε στην οθόνη τον κωδικό (ώστε να καταλάβουμε ότι ΟΝΤΩΣ ήρθε μέσα στο μήνυμα και ο κωδικός).

Στην εκφώνηση λέει:

Επιβεβαιώστε ότι η επικοινωνία δουλεύει εμφανίζοντας στην οθόνη το αναγνωριστικό ομάδας του δημιουργούμενου παιδιού. Για τον σκοπό αυτό καλέστε την printf στη συνάρτηση do_start_scheduling() του αρχείου servers/sched/schedule.c.

Άρα πρέπει να μπω στον SCHED (από εδώ και πέρα δεν ασχολούμαστε με τον PM).

```
# vi servers/sched/schedule.c_
```

ΕΠΕΙΤΑ:

```
/* This file contains the scheduling policy for SCHED
 *
 * The entry points are:
 * do_noquantum:      Called on behalf of process' that run out of quantum
 * do_start_scheduling Request to start scheduling a proc
 * do_stop_scheduling  Request to stop scheduling a proc
 * do_nice             Request to change the nice level on a proc
 * init_scheduling     Called from main.c to set up/prepare scheduling
 */
#include "sched.h"
#include "schedproc.h"
#include <assert.h>
#include <minix/com.h>
#include <machine/archtypes.h>
#include "kernel/proc.h" /* for queue constants */

PRIVATE timer_t sched_timer;
PRIVATE unsigned balance_timeout;

#define BALANCE_TIMEOUT 5 /* how often to balance queues in seconds */

FORWARD _PROTOTYPE( int schedule_process, (struct schedproc * rmp,
                                           unsigned flags));
FORWARD _PROTOTYPE( void balance_queues, (struct timer *tp)           );
do_start_scheduling_
```

και οδηγουμαι εδώ:

```
/* =====
 * do_start_scheduling
 * ===== */
PUBLIC int do_start_scheduling(message *m_ptr)
{
    register struct schedproc *rmp;
    int rv, proc_nr_n, parent_nr_n;

    /* we can handle two kinds of messages here */
    assert(m_ptr->m_type == SCHEDULING_START ||
           m_ptr->m_type == SCHEDULING_INHERIT);

    /* check who can send you requests */
    if (!accept_message(m_ptr))
        return EPERM;

    /* Resolve endpoint to proc slot. */
    if ((rv = sched_isemptyendpt(m_ptr->SCHEDULING_ENDPOINT, &proc_nr_n))
        != OK) {
        return rv;
    }
    rmp = &schedproc[proc_nr_n];

    /* Populate process slot */
    rmp->endpoint = (m_ptr->SCHEDULING_ENDPOINT);
    rmp->parent   = (m_ptr->SCHEDULING_PARENT);
    rmp->max_priority = (unsigned)(m_ptr->SCHEDULING_MAXPRIO);
    if (rmp->max_priority >= NR_SCHED_QUEUES) {
        return EINVAL;
    }

    /* Inherit current priority and time slice from parent. Since there
     * is currently only one scheduler scheduling the whole system, this
     * value is local and we assert that the parent endpoint is valid */
    if (rmp->endpoint == rmp->parent) {
        Παρατηρώ ότι έχει δείκτη σε μήνυμα
        (άρα εκεί μέσα βρίσκεται η πληροφορία
        που έστειλα)

        Παρατηρώ ότι είναι ΑΚΡΙΒΩΣ οι ίδιες
        σταθερές που υπήρχαν στον κώδικα
        της sched_inherit, οπότε τις τραβάει εδώ
        μέσω του μυνύματος που ήρθε (m_ptr)

        Άρα, μπορώ να τραβήξω αντίστοιχα
        και να εμφανίσω το δικό μου:
        Είτε m_ptr->m9_l2, είτε m_ptr->m9_l5
        (ανάλογα τι έβαλα στη sched_inherit)
    }
}
```

Κάνω τις παρακατω αλλαγες στον κωδικα του (τα πολλά σχόλια που γράφω για τον "rmp" δε χρειάζονται για το πρώτο ερώτημα):

```
/*=====
 *
 *      do_start_scheduling
 *
 *=====*/
PUBLIC int do_start_scheduling(message *m_ptr)
{
    register struct schedproc *rmp;
    int rv, proc_nr_n, parent_nr_n;

    /* we can handle two kinds of messages here */
    assert(m_ptr->m_type == SCHEDULING_START ||
           m_ptr->m_type == SCHEDULING_INHERIT);

    /* check who can send you requests */
    if (!accept_message(m_ptr))
        return EPERM;

    /* Resolve endpoint to proc slot. */
    if ((rv = sched_isemtyendpt(m_ptr->SCHEDULING_ENDPOINT, &proc_nr_n))
        != OK) {
        return rv;
    }
    rmp = &schedproc[proc_nr_n];

    /* Populate process slot */
    rmp->endpoint = m_ptr->SCHEDULING_ENDPOINT;
    rmp->parent = m_ptr->SCHEDULING_PARENT;
    rmp->max_priority = (unsigned) m_ptr->SCHEDULING_MAXPRIO;

    printf("Κωδικος = %d \n", m_ptr->m_l2);

    if (rmp->max_priority >= NR_SCHED_QUEUES) {
        return EINVAL;
    }

    /* Inherit current priority and time slice from parent. Since there
     * is currently only one scheduler scheduling the whole system, this

```

Θα χρειαστεί στο δεύτερο ερώτημα. Ο "rmp" δείχνει στη ΝΕΑ διεργασία που μόλις ΞΕΚΙΝΗΣΕ (δηλαδή δεν υπήρχε νωρίτερα) και αυτή η ΝΕΑ διεργασία εισάγεται στον πίνακα "schedproc", όπου βρίσκονται ΗΔΗ κι άλλες διεργασίες (που είχαν ξεκινήσει νωρίτερα). Ο SCHED ενημερώνει κάποια πεδία της ΝΕΑΣ διεργασίας και μετά στέλνει αίτημα στον πυρήνα(KERNEL) ώστε να τη δρομολογήσει, δηλαδή να την εισάγει σε κάποια ουρά που έχει επιλεγεί από τον SCHED (διακομιστής χρονοδρομολόγησης)

Θα γίνει κατανοητό στο τρίτο ερώτημα

ή m9_l5 (το εξήγησα παραπάνω)

Δικο σας μήνυμα (αλλάξτε το)

Για να καταλάβετε ότι το «α'» ερώτημα είναι OK, πρέπει να κάνετε τα παρακάτω:
(ΑΦΟΥ πρώτα εννοείται έχετε μεταφέρει τα 4 αλλαγμένα αρχεία στο Minix)

Είμαι στο φάκελο: /usr/src

- 1) #make libraries
- 2) #make includes
- 3) #cd tools
- 4) #make install
- 5) #shutdown

Ξαναανοίγετε το Vmware-Minix Και βλεπετε στην οθονη αριθμους που ειναι οι κωδικοι ομαδας. Κανετε κανονικα login(root,root) και ξαναεμφανιζεται ενας αριθμος.

Πατατε "Alt και F2", παλι login(root, root).

Τελος, πατατε "Alt και F1" και βλεπετε ενα ΑΛΛΟ κωδικό.

(απο κοντα θα σας πω τι σημαινει αυτος ο νέος κωδικός)

Θα περιμένετε κάποια δευτερόλεπτα ΜΕΧΡΙ να σταματήσουν να εκτυπώνονται στην οθόνη. ΤΟΤΕ, θα πατήσετε root (Enter) και πάλι root (Enter). (κι ας φαίνεται ότι έχει γεμίσει η οθόνη αριθμούς, έχετε μόλις κάνει Login)