ΜΕΡΟΣ 2

Στην εκφωνηση (ΒΛΕΠΕ ΑΡΧΕΙΟ: ekfonisi.pdf) στο 2ο ερώτημα ξεκινάει με το παρακάτω:

Τροποποιήστε τη δομή **schedproc** στο **schedproc.h** για να συμπεριλάβετε τα πεδία **procgrp**, **proc_usage**, **grp_usage** και **fss_priority**.

Μπαίνω με τον "vi" σε αυτό το αρχείο:

```
# vi servers/sched/schedproc.h_
```

```
EXTERN struct schedproc {
    endpoint_t endpoint;    /* process endpoint id */
    endpoint_t parent;  /* parent endpoint id */
    unsigned flags;     /* flag bits */

    /* User space scheduling */
    unsigned max_priority;  /* this process' highest allowed priority */
    unsigned priority;      /* the process' current priority */
    unsigned time_slice;        /* this process's time slice */
    unsigned cpu;       /* what CPU is the process running on */
    bitchunk_t cpu_mask[BITMAP_CHUNKS(CONFIG_MAX_CPUS)]; /* what CPUs is hte
                                                            process allowed
                                                            to run on */
} schedproc[NR_PROCS];
```

Τα πεδία αυτά (αλλά και τα παρακάτω) παίρνουν τιμές στον κώδικα του sched (βλέπε do_start_scheduling) με τα πεδία του μηνύματος m_ptr

Στο struct "schedproc" αποθηκεύονται οι ΠΛΗΡΟΦΟΡΙΕΣ κάθε διεργασίας στον SCHED. (εδώ πρέπει να προσθέσουμε τα 4 πεδία που ζητάει)

Αν το κάνετε "search" στο Minix, είναι ίσο με 256

Ο πίνακας που περιέχει ΟΛΕΣ τις διεργασίες που ήρθαν στον SCHED για να εξυπηρετηθούν

(Ερώτηση: Που αποθηκεύονται οι ΠΛΗΡΟΦΟΡΙΕΣ κάθε διεργασίας στον PM? (Στο servers/pm/mproc.h , θυμηθείτε ότι εκεί μέσα ήταν το "mp_procgrp" που χρειαστήκαμε στο ΠΡΩΤΟ ερώτημα)

(Ερώτηση: Που αποθηκεύονται οι ΠΛΗΡΟΦΟΡΙΕΣ κάθε διεργασίας στον KERNEL? (Στο kernel/proc.h , θα χρειαστεί στο ΤΡΙΤΟ ερώτημα)

Κάνω τις παρακάτω αλλαγές:

```
EXTERN struct schedproc {
    endpoint_t endpoint;    /* process endpoint id */
    endpoint_t parent;  /* parent endpoint id */
    unsigned flags;     /* flag bits */

    /* User space scheduling */
    unsigned max_priority;  /* this process' highest allowed priority */
    unsigned priority;      /* the process' current priority */
    int procgrp;
    int proc_usage;
    int grp_usage;
    int fss_priority;
    unsigned time_slice;        /* this process's time slice */
    unsigned cpu;       /* what CPU is the process running on */
    bitchunk_t cpu_mask[BITMAP_CHUNKS(CONFIG_MAX_CPUS)]; /* what CPUs is hte
                                                            process allowed
                                                            to run on */
} schedproc[NR_PROCS];
```

Μπορείτε να τα δηλώσετε με οποιαδήποτε σειρά και με "long" όλα ή κάποια από αυτά. Πρέπει να βρίσκονται ΚΑΤΩ από..

ΑΡΧΙΚΑ για κάθε νέα διεργασία στην "do_start_scheduling" του SCHED, σε καθένα από τα 4 πεδία ΠΡΕΠΕΙ να αποθηκεύσω:

1) Στο procgrp: Τον κωδικό ομάδας που μου έστειλε ο PM στο μήνυμα m_ptr

2) Στο proc_usage: 0 (Δηλώνει πόσο χρόνο χρησιμοποίησε η διεργασία τον επεξεργαστή. Επομένως αρχικά είναι 0 γιατί μόλις εμφανίστηκε η ΝΕΑ διεργασία. Το proc_usage θα καταλάβετε παρακάτω στην "do_noquantum" πως και πότε αλλάζει

3)Στο grp_usage: 0 ή καλύτερα την τιμή που έχει σε αυτό το πεδίο μία άλλη διεργασία με ίδιο "procgrp", άρα μία άλλη διεργασία που ανήκει στον ίδιο χρήστη (Δηλώνει πόσο χρόνο χρησιμοποίησε η ομάδα της διεργασίας τον επεξεργαστή. ΠΡΕΠΕΙ να εξασφαλίσουμε να έχει την ίδια τιμή για τις διεργασίες που ΑΝΗΚΟΥΝ ΣΤΗΝ ΙΔΙΑ ΟΜΑΔΑ, δηλαδή έχουν ίδια τιμή στο procgrp μεταξύ τους)

4) Στο fss_priority: 0 (Δηλώνει την προτεραίοτητα της κάθε διεργασίας, αρχικά θεωρούμε ότι είναι 0 γιατί η διεργασία μόλις ξεκίνησε. Όσο μικρότερη τιμή στο fss_priority έχει η διεργασία, τόσο πιο γρήγορα θα επιλεγεί από τον πυρήνα ώστε να εκτελεστεί. Το fss_priority τελικά θα χρειαστεί να το στείλει ο SCHED μήνυμα στον KERNEL στο τρίτο ερώτημα, ώστε έπειτα να το χρησιμοποιήσει ο KERNEL για να πετύχει δίκαιη χρονοδρομολόγηση)

Επανέρχομαι στο schedule.c και κάνω τις εξής αλλαγές στην do_start_scheduling:

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

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

    /* 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;

    rmp->procgrp = m_ptr->m9_l2;
    //printf("Kodikos =  %d \n", m_ptr->m9_l2);
    rmp->proc_usage = 0;
    rmp->grp_usage = 0;
    for (proc_nr=0, rmt=schedproc; proc_nr < NR_PROCS; proc_nr++, rmt++) {
        if (rmt->flags & IN_USE) {
            if (rmp->procgrp == rmt->procgrp) {
                rmp->grp_usage = rmt->grp_usage;
                break;
            }
        }
    }

    rmp->fss_priority = 0;
```

Βοηθητικές μεταβλητές, βάλτε άλλα ονόματα αν τελικά τις χρειαστείτε παρακάτω εδώ

ΠΡΟΣΟΧΗ: Ο "rmp" ΘΥΜΗΘΕΙΤΕ ότι είναι δείκτης στη ΝΕΑ διεργασία που μόλις ήρθε στον SCHED. Για αυτό όπου βλέπετε "rmp" στον κώδικα εδώ να θυμάστε ότι είναι ΜΟΝΟ αυτή η ΝΕΑ διεργασία.

Είχα γράψει παραπομπή στο "schedproc.h" ότι τα πεδία αυτά παίρνουν εδώ τιμές απο το m_ptr (όπως βλέπετε).
Το ίδιο ΠΡΕΠΕΙ να κάνουμε παρακάτω και για τα 4 πεδία που προσθέσαμε.

Σε σχόλια, το τσεκάραμε στο 1ο ερώτημα

Διασχίζει τον πίνακα "schedproc"
(αυτές τις 2 γραμμές τις δανείζομαι σαν ιδέα από την balance_queues που την αναφερει στο check.pdf)

Αν βρήκα διεργασία της ίδια ομάδας (δηλαδή με ίδιο procgrp)

Βάζω στο grp_usage το ίδιο με αυτή της ομάδας μου

Ο κώδικας αυτός διασχίζει τον πίνακα "schedproc" όπου είπαμε είναι μέσα όλες οι διεργασίες που εξυπηρετεί ο SCHED (τον κώδικα τον πήραμε copy-paste από τη συνάρτηση balance_queues, που βρίσκεται στο ίδιο αρχείο, δλδ schedule.c . Η διαφορά είναι ότι εδώ ο "rmp" είναι η ΝΕΑ διεργασία και ο "rmt" είναι βοηθητικός δείκτης για να διασχίσω τον πίνακα).

Στην εκφώνηση (ΒΛΕΠΕ ΑΡΧΕΙΟ: ekfonisi.pdf) στο 2^{ο} ερώτημα ΣΥΝΕΧΙΖΕΙ με το παρακάτω:

Όταν μια διεργασία ολοκληρώνει ένα κβάντο, ενημερώστε τα πεδία **procgrp**, **proc_usage**, **grp_usage** και **fss_priority** για όλες τις διεργασίες που περιμένουν στην ουρά χρήστη. Θα χρειαστεί να υπολογίσετε δυναμικά το πλήθος των διεργασιών που ανήκουν στην ίδια ομάδα, για να ισομοιράσετε στην συνέχεια τον χρόνο στις ομάδες σύμφωνα με τον αλγόριθμο της δίκαιης χρονοδρομολόγησης.

[ΓΕΝΙΚΑ όπου αναφέρουμε την έννοια «κβάντο» είναι ένας σταθερός χρόνος (εδώ 200 ms) που δίνει ο επεξεργαστής σε κάθε διεργασία που θέλει να εκτελεστεί. Με αυτόν τον τρόπο, μετά από ΚΑΘΕ κβάντο (δηλαδή μετά από κάθε 200 ms), ο επεξεργαστής ΑΛΛΑΖΕΙ τη διεργασία που τρέχει εκείνη τη στιγμή και ΒΑΖΕΙ μια ΑΛΛΗ στη θέση της. Επομένως, με αυτόν τον τρόπο ο επεξεργαστής εξυπηρετεί «κυκλικά» τις διεργασίες, ώστε καμία να μην περιμένει.]

Όπως λέει η εκφώνηση, όταν μια διεργασία ολοκληρώνει ένα κβάντο, καλείται η do_noquantum (ο κώδικας της είναι στο ίδιο αρχείο που βρισκόμαστε).

```
/*===========================================================================*
 *              do_noquantum                 *
 *===========================================================================*/

PUBLIC int do_noquantum(message *m_ptr)
{
    register struct schedproc *rmp;
    int rv, proc_nr_n;

    if (sched_isokendpt(m_ptr->m_source, &proc_nr_n) != OK) {
        printf("SCHED: WARNING: got an invalid endpoint in OOQ msg %u.\n",
        m_ptr->m_source);
        return EBADEPT;
    }

    rmp = &schedproc[proc_nr_n];
    if (rmp->priority < MIN_USER_Q) {
        rmp->priority += 1; /* lower priority */
    }

    if ((rv = schedule_process_local(rmp)) != OK) {
        return rv;
    }
    return OK;
}
```

ΠΡΟΣΟΧΗ: Ο "rmp" εδώ δείχνει στη διεργασία που μόλις ΤΕΛΕΙΩΣΕ το κβάντο της.

ΤΩΡΑ θα γίνουν τα εξής:
1) Αύξησε κατα 200 τα proc_usage, grp_usage της διεργασίας που μόλις τελείωσε.
2) Αύξησε κατα 200 τα grp_usage των διεργασίων που έχουν ΙΔΙΟ procgrp με αυτή που τελείωσε.
3) Υπολόγισε το πλήθος των διαφορετικών ομάδων (χρηστών) διεργασιών.
4) Ενημέρωσε τα proc_usage, grp_usage, fss_priority για όλες τις διεργασίες και χρονοδρομολόγησε τες.

Αυτά πλέον ΔΕ θα χρειαστούν, γιατί αφορούσαν ΜΟΝΟ αυτή τη διεργασία που δείχνει ο "rmp".

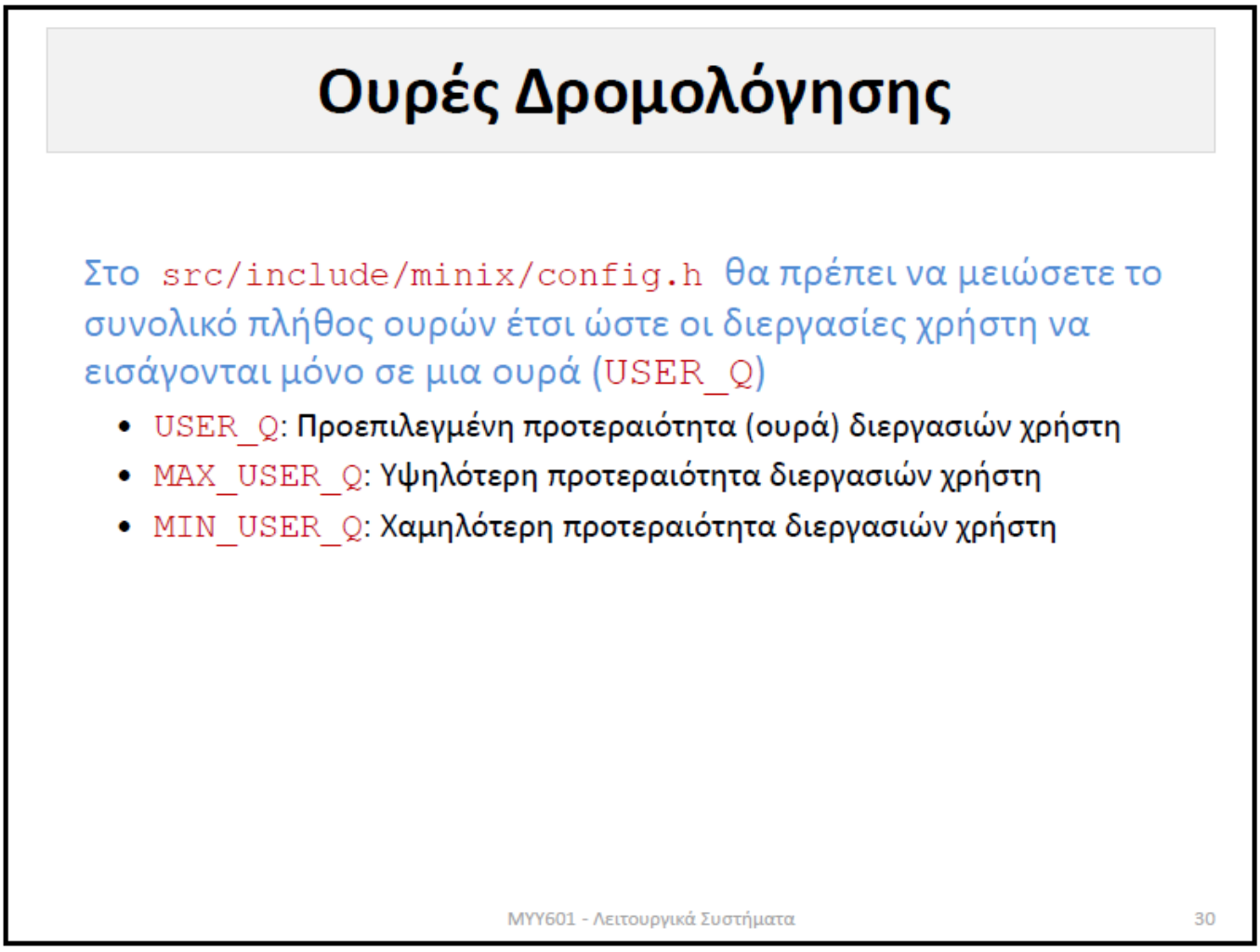
[Για τα βήματα 1, 2, 3, 4 σας επισυνάπτω τρεις «παραλλαγές» για να πάρει ο καθένας σας πρωτοβουλία και να τα κάνει συνδυαστικά με δικό του τρόπο, ώστε να ΜΗΝ έχει θέμα με ομοιότητες]

ΜΕΡΟΣ 3

Στην εκφώνηση (ΒΛΕΠΕ ΑΡΧΕΙΟ: ekfonisi.pdf) στο 3^{ο} ερώτημα ξεκινάει με το παρακάτω:

Στο **kernel/proc.h** μειώστε το συνολικό πλήθος ουρών έτσι ώστε όλες οι διεργασίες χρήστη να βρίσκονται σε μία μόνο ουρά.

Ωστόσο, το αρχείο που πρέπει να γίνει η αλλαγή (ΒΛΕΠΕ ΑΡΧΕΙΟ: check.pdf) είναι το "config.h"
(στην εκφώνηση έχει λάθος αρχείο, γιατί αυτό το λάθος αρχείο ήταν παλιότερα στο Minix version 2, ενώ τώρα είμαστε στο Minix version 3)



Μπαίνω στο αρχείο αυτό:

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

Αρχικά το αρχείο περιέχει τα παρακάτω σε όσα πρέπει να αλλάξουμε:

```
/* Scheduling priorities. Values must start at zero (highest
 * priority) and increment.
 */
#define NR_SCHED_QUEUES   16    /* MUST equal minimum priority + 1 */
#define TASK_Q          0    /* highest, used for kernel tasks */
#define MAX_USER_Q          0    /* highest priority for user processes */
#define USER_Q          ((MIN_USER_Q - MAX_USER_Q) / 2 + MAX_USER_Q)  /* default
                          (should correspond to nice 0) */
#define MIN_USER_Q     (NR_SCHED_QUEUES - 1) /* minimum priority for user
                           processes */
/* default scheduling quanta */
#define USER_QUANTUM 200
```

Τυπικά παρατηρώ ότι το κβάντο είναι όντως 200 ms

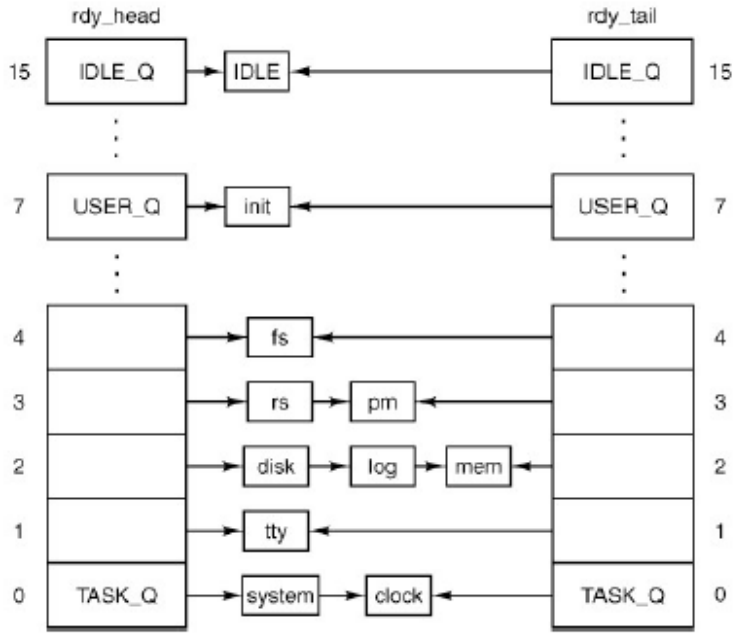
Η τιμή του USER_Q είναι ίση με 7, αν κάνω αντικατάσταση τα MAX_USER_Q (0) και MIN_USER_Q (15).
Γενικά οι διεργασίες χρήστη μπαίνουν σε ΠΟΛΛΕΣ ουρές του πυρήνα με τιμές ΑΠΟ "MAX_USER_Q" ΕΩΣ "MIN_USER_Q" (βλέπε σχήμα παρακάτω).
Δηλαδή: MAX_USER_Q (0) <= ουρά χρήστη <= MIN_USER_Q (15)

Εμείς ΟΜΩΣ θέλουμε να μπούνε ΜΟΝΟ στην ουρά USER_Q, άρα πρέπει τα MAX_USER_Q και MIN_USER_Q να τα κάνω ίσα με 7, οπότε: MAX_USER_Q (7) <= ουρά χρήστη <= MIN_USER_Q (7)
Άρα, ουρά χρήστη = 7 (μοναδική).

Επίσης, πρέπει να ΜΕΙΩΣΩ το NR_SCHED_QUEUES (συνολικό πλήθος ουρών του πυρήνα) σε 8, γιατί πρέπει να είναι ακριβώς ένα παραπάνω από την ελάχιστη προτεραιότητα (MIN_USER_Q)

ΓΕΝΙΚΗ ΠΑΡΑΤΗΡΗΣΗ: Στον πυρήνα στις διεργασίες χρήστη τη μέγιστη προτεραίοτητα την έχουν οι διεργασίες στην ουρά MAX_USER_Q (δλδ με την μικρότερο αριθμό ουράς, αρχικά 0)
και την ελάχιστη προτεραίοτητα την έχουν οι διεργασίες στην ουρά MIN_USER_Q (δλδ με τον μεγαλύτερο αριθμό ουράς, αρχικά 15)

ΕΝΔΕΙΚΤΙΚΟ ΣΧΗΜΑ ΜΕ **ΟΛΕΣ** (όχι μόνο του χρήστη) ΤΙΣ ΟΥΡΕΣ ΤΟΥ ΠΥΡΗΝΑ



Κάνω τις παρακάτω αλλαγές, ώστε να συμφωνούν με όσα λέει στο check.pdf .

Για να μπούνε οι διεργασίες χρήστη στην ίδια και ΜΟΝΟ ουρά (USER_Q):

```
/* Scheduling priorities. Values must start at zero (highest
 * priority) and increment.
 */
#define NR_SCHED_QUEUES   8 /* MUST equal minimum priority + 1 */
#define TASK_Q            0    /* highest, used for kernel tasks */
#define MAX_USER_Q        7    /* highest priority for user processes */
#define USER_Q            7 /* default
                               (should correspond to nice 0) */
#define MIN_USER_Q        7 /* minimum priority for user
                               processes */
/* default scheduling quanta */
#define USER_QUANTUM 200
```

Θα μπορούσα να είχα αφήσει τον τύπο που είχε ΗΔΗ, γιατί πάλι η τιμή του USER_Q θα ήταν είσαι με 7.