# Minix 3.2.0

Η βασική ιδέα της ειναι να πραγματοποιήσουμε δίκαιη χρονοδρομολόγηση για τις διεργασίες χρήστη.Για να το πετύχουμε αυτο τοποθετήσαμε ολες τις διεργασίες χρήστη σε μόνο μια ουρά και ορίσαμε σε αυτες προταιρεότητα εκτέλεσης.
Συνεπώς μείωσαμε το συνολικό πλήθος ουρων σε 8,και ορίσαμε ως ουρα χρήστη την 7 στον κατάλογο include/minix/config.h.
Ο βασίκος κωδίκας για την λύση της άσκησης βρισκεται στους καταλογους include,servers,lib,kernel συνεπώς πραγματοποιήσαμε τα εξής βήματα :

## Βήμα 1ο : Επικοινωνία PM με SCHED

Ο διακομιστής διεργασιών (PM) διατηρεί εναν πίνακα (mproc) με τις πληροφορίες της κάθε νέας διεργασίας.Σε αυτον τον πίνακα ειναι αποθηκευμένο και το mp_procgrp το οποίο θα χρησιμοποιήσουμε για να διαχωρίσουμε τις διεργασίες σε ομάδες.Οι διεργασίες με το ίδιο mp_procgrp ανήκουν δηλαδή στην ιδιά ομάδα.
Για να στείλουμε αυτη την πληροφορία στον διακομιστή χρονοδρομολόγησης (sched),θα αποθηκέυσουμε το mp_procgrp σε ενα πεδίο του μηνήματος που στέλνει ο PM.Βάση του include/minix/com.h διαπιστώσαμε οτι ο τύπος μηνύματος ειναι mess 7,οπότε αποθηκέυσαμε το mp_procgrp εκει,αφου πρώτα εχουμε κάνει define το PM_PROCGRP στο include/minix/com.h

## Βήμα 2ο : Χρονοδρομολόγηση SCHED

Ο sched διατηρέι ενα πίνακα διεργασιώ (schedproc).Για τις ανάγκες της άσκησης προσθέσαμε  4 επιπλέον
πεδία ,grp_usage,proc_usage,fss_priority,procgrp.

Ο sched ξεκινάει την χρονοδρομολόγηση στη do_start_scheduling() ,συνεπώς πρέπει να αρχικοποιήσουμε τα παραπάνω πεδία εκέι.Η do_start_scheduling() μπορεί να χειριστέι δύο τύπους μηνυμάτων (SCHEDULING_START και SCHEDULING_INHERIT) και οι αρχικοποιήσεις θα πρέπει να γίνουν σε κάθε περίπτωση.

Proc_usage :
Δηλώνει πόσα κβάντα εχουν ολοκληρωθέι για την κάθε διεργασία,αρα αρχίκα θα είναι μηδέν μιας και δεν εχει εκτελεστέι ακόμα

Grp_usage:
Δηλώνει πόσα κβάντα εχουν ολοκληρωθέι συνολικά για την εκτέλεση των διεργασίων  μιας ομάδας,οπότε ολες οι διεργασίες της ίδιας ομάδας εχουν το ιδιο grp_usage. Αν ειναι η πρώτη διεργασία της ομάδας τοτε θα ισούτε με μηδέν, αν οχι τότε θα της αναθέσουμε το ιδιο grp_usage με τις υπολοιπες διεργασίες της ομάδας,μεσω μιας απλής αναζήτης στον πίνακα schedproc.

Procgrp.
Θα το βρούμε μεσω του μηνύματος που ελαβε ο sched απο τον pm (m_ptr→PM_PROCGRP)

Fss_priority:
Βαση αυτου θα μπορέσουμε να δώσουμε προτεραιότητα στις διεργασίες χρήστη  επειτα απο τον πυρήνα.Χρησιμοποιούμε τον τύπο του πίνακα για τον υπολογισμό της.  Για τις ανάγκες του τύπου πρεπει να υπολογίσουμε ποσες διαφορετικές ομάδες εχουμε στην ουρά χρήστη.Αυτο το κάνουμε με την χρήση ενος βοηθητικού πίνακα(array_procgrps).
Σε αυτον τον πίνακα τοποθετούμε ολα τα procgrp των διεργασιών χρήστη.Επειτα με την χρήση του αλγόριθμου ταξινόμισης merge sort ταξινομούμε τον πίνακα.Εχουμε επιπλέον δημιουργήσει μια συνάρτηση(number_of_groups) η οπόια δεχέται τον ταξινομημένο πίνακα και υπολογίζει το πλήθος των procgrp.Αυτές τις βοηθητικές συναρτήσεις τις εχουμε αποθηκέυσει στο servers/pm/utility.

Μέσω της  schedule_process() η do_start_scheduling() στέλνει μήνυμα στον πυρήνα και προσθέτει στον πινακα proc την διεργασία που έλαβε.

Ο sched επιπλέον εχει την συναρτηση do_noquantum() η οποία καλείται κάθε φορα που μια διεργασία ολοκληρώνει ενα κβάντο.
Εκει εμείς κάναμε τις απαραίτητες ενημερώσεις των πεδίων grp_usage,proc_usage,fss_priority για την επίτευξη της δίκαιης χρονοδρομολόγησης με τα εξής βήματα μόνο για τις διεργασίες χρήστη.

- Θα αυξήσουμε το proc_usage της διεργασίας που έληξε το κβάντο κατα το USER_QUANTUM
- Θα αυξήσουμε το grp_usage ολων των διεργασιών που ανήκουν στην ίδια ομάδα κατα USER_QUANTUM,μεσω μιας απλής αναζήτης στον πίνακα schedproc και παράλληλα,επειδη θα μας χρειαστέι για τον υπολόγισμο του fss_priority,αποθηκέυουμε το procgrp της κάθε διεργασίας που διατρέχουμε σε ενα βοηθητικό πίνακα(groups_array
- Υπολογίζουμε το πλήθος των διεργασίων με την χρήση της num_of_groups()

- Διατρέχουμε ξανα τον πινακα schedproc ετσι ωστε να ενημερωσουμε όλες τις διεργασίες χρήστη βαση του αλγοριθμου που μας δώθηκε,δηλαδη: proc_usage=proc_usage/2 grp_usage=grp_usage/2 fss_priority=proc_usage/2 + (grp_usage*num_of_groups)/4

Αφου εχουμε ολοκληρώσει τις απαραίτητες ενημερώσεις στελνουμε μήνυμα στον πύρήνα με την χρήση της schedule_process_local() για να πραγματοποιήσει και εκεινος τις κατάλληλες ενημερώσεις των πεδίων διεργασιών που άλλαξαν.

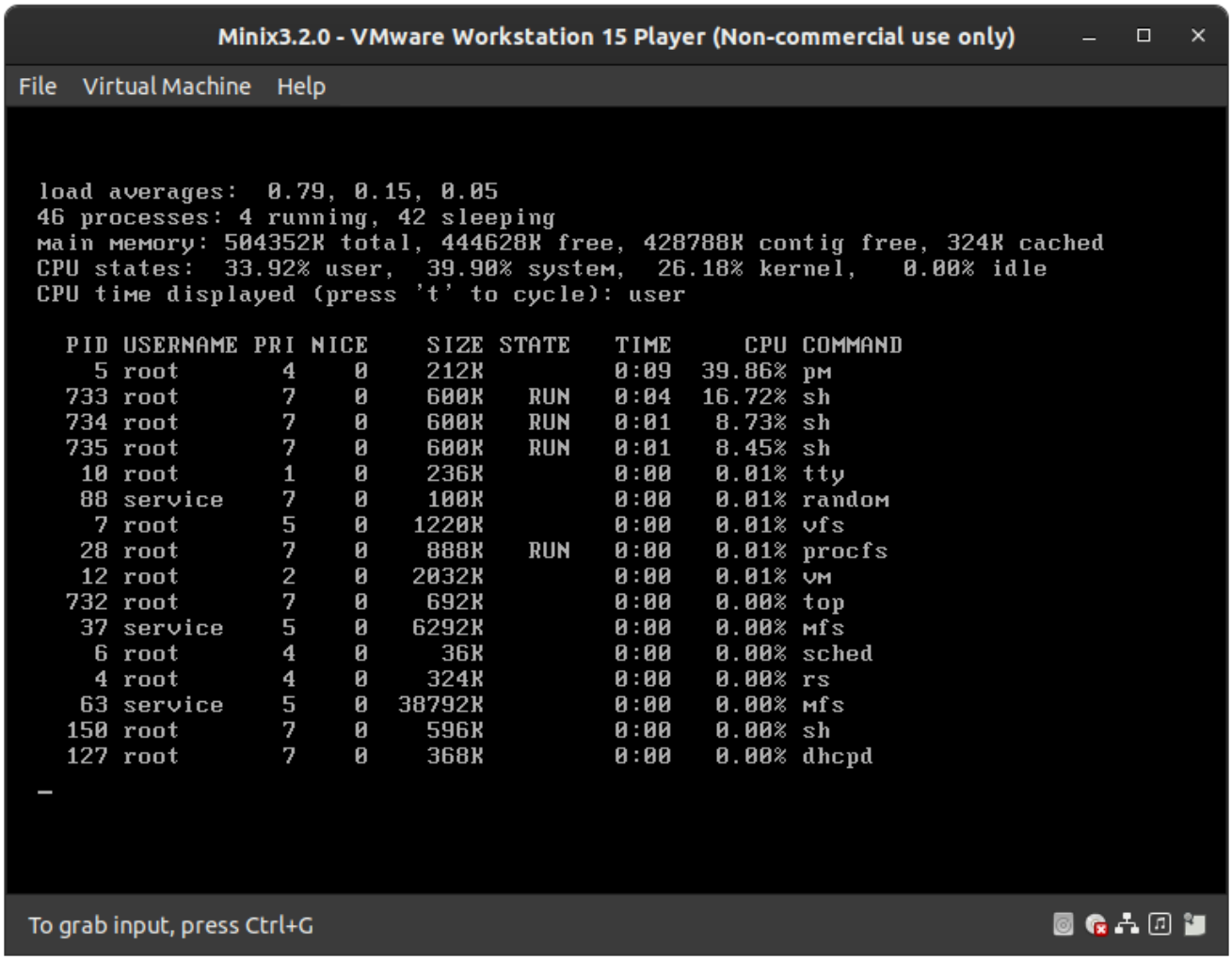
Αυτο ως συνέπεια είχε να προσθέσουμε το πεδίο fss_priority στις συναρτήσεις που πραγματοποιούν την μεταφορα μηνύματος:
- sys_schedule του lib/libsys
- schedule_process στου servers/sched/schedule.c
- do_schedule του kernel/system
- sched_proc του kernel/system.c

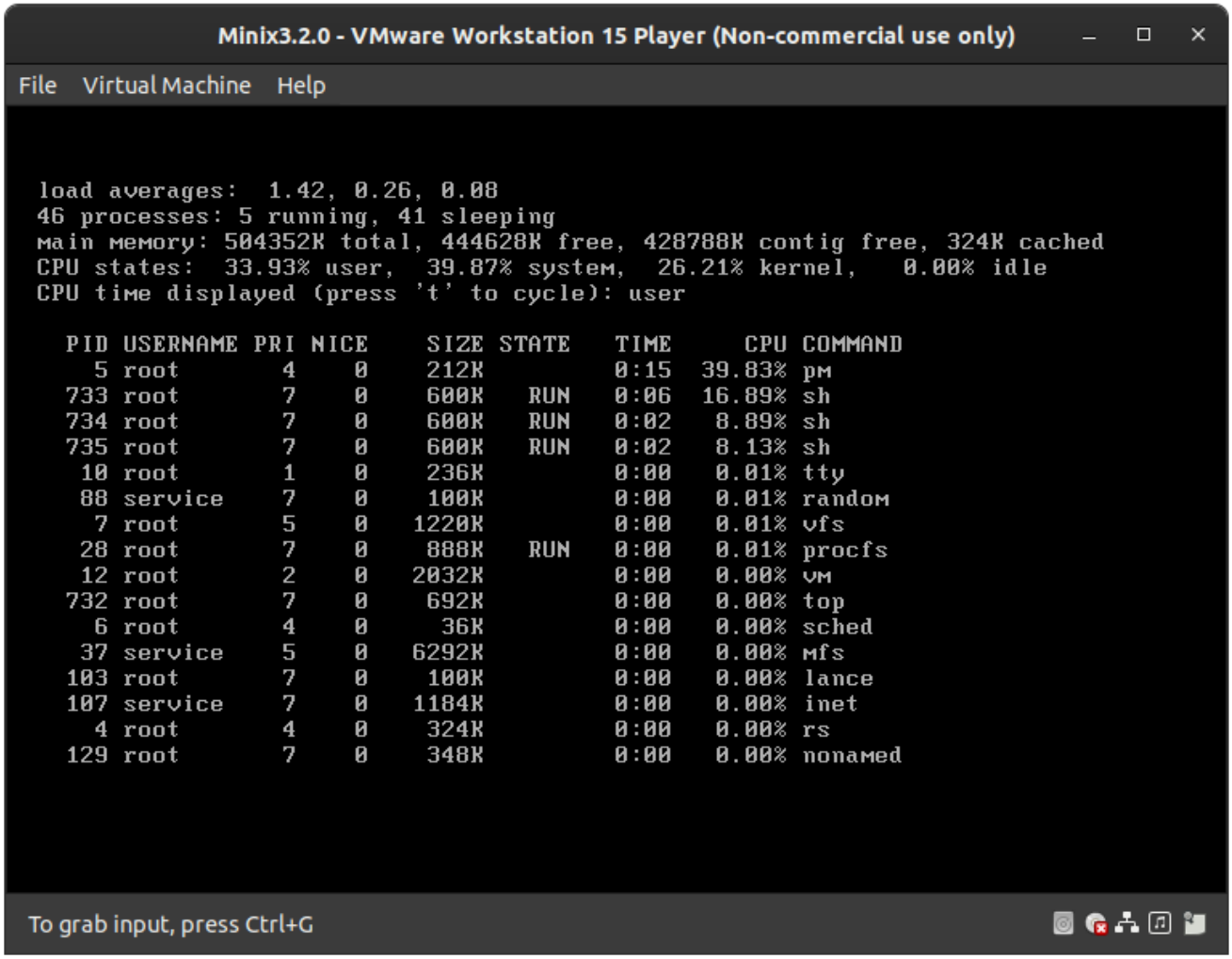
## Βήμα 3ο : Επιλογή επόμενης διεργασίας προς εκτέλεση απο τον πυρήνα

Ο πυρήνας αναλαμβανει ποια ειναι η επόμενη διεργασία που θα τρέξει.Συνεπώς για να πετύχουμε δικαιη χρονοδρομολόγηση επεξεργαστήκαμε την pick_proc. Η pick_proc διατρέχει τον πίνακα rdy_head και επιλέγει την πρώτη διεργασία ,ετοιμη προς εκτέλεση με το χαμηλότερο priority.Οταν λοιπόν φτάσει στην ουρα χρήστη (USER_Q=7) τότε εμείς διατρέχουμε ολη την ουρα χρήστη και επιλέξαμε την διεργασία με το μικρότερο p_fss_priority.
Για να το πραγματοποιήσουμε αυτο εχουμε 2 βοηθητικές μεταβλήτες( int fss_priority και proc
*rp2).Οσο λοιπόν διατρέχουμε την ουρα .με την βοήθεια του p_nextready, μεχρι το τέλος της (while(rp != NULL)) ελέγχουμε καθε φορα αν η επόμενη διεργασία εχει μικροτερο p_fss_priority και αν εχει επιλέγουμε αυτη τη διεργασία ως επόμενη προς εκελεση.

## Εκτέλεση εργασιας και δοκιμη ορθης λειτουργιας

Ανοιγοντας λοιπον το MINIX πηγαμε στο path που ειχαμε το σκριπτ και το εκτελεσαμε με & (τρεχει στο background απ οτι ειδαμε στο ιντερνετ) ωστε να μπορουμε απο κατω να τρεξουμε κ αλλο αν χρειαστει. Ετσι λοιπον και καναμε. Τρεξαμε αλλη μια φορα απο κατω το σκριπτ.
Στην συνεχεια με ALT F2 ανοιξαμε ενα αλλο τερματικο στο MINIX και πηγαμε παλι στο path με το σκριπτ. Τρεξαμε και εκει μια φορα το σκριπτ μας.
Τελευταιο βημα ηταν να παμε με ALT F3 και σε ενα αλλο τερματικο και να εκτελεσουμε top -s 10 (η top ενημερωνεται καθε 10 δευτερα) και παρατηρησαμε οτι οι διεργασιες παιρνουν ακριβως το ποσοστο που περιμενα (δηλαδη το αθροισμα των 2 διεργασιων του ενος τερματικου θα πρεπει να ειναι ισο με το ποσοστο που παιρνει η διεργασια στο δευτερο τερματικο). Αυτο λοιπον θα δουμε και παρακατω με καποια screenshots που βγαλαμε οσο ετρεχαν (παρατηρουμε τις διεργασιες με ονομα sh).


Minix3.2.0 - VMware Workstation 15 Player (Non-commercial use only)
File Virtual Machine Help
load averages: 0.79, 0.15, 0.05
46 processes: 4 running, 42 sleeping
main memory: 504352K total, 444628K free, 428788K contig free, 324K cached
CPU states: 33.92% user, 39.90% system, 26.18% kernel, 0.00% idle
CPU time displayed (press 't' to cycle): user
PID USERNAME PRI NICE SIZE STATE TIME CPU COMMAND
5 root 4 0 212K 0:09 39.86% pm
733 root 7 0 600K RUN 0:04 16.72% sh
734 root 7 0 600K RUN 0:01 8.73% sh
735 root 7 0 600K RUN 0:01 8.45% sh
10 root 1 0 236K 0:00 0.01% tty
88 service 7 0 100K 0:00 0.01% random
7 root 5 0 1220K 0:00 0.01% vfs
28 root 7 0 888K RUN 0:00 0.01% procfs
12 root 2 0 2032K 0:00 0.01% vm
732 root 7 0 692K 0:00 0.00% top
37 service 5 0 6292K 0:00 0.00% mfs
6 root 4 0 36K 0:00 0.00% sched
4 root 4 0 324K 0:00 0.00% rs
63 service 5 0 38792K 0:00 0.00% mfs
150 root 7 0 596K 0:00 0.00% sh
127 root 7 0 368K 0:00 0.00% dhcpd
To grab input, press Ctrl+G


Minix3.2.0 - VMware Workstation 15 Player (Non-commercial use only)
File Virtual Machine Help
load averages: 1.42, 0.26, 0.08
46 processes: 5 running, 41 sleeping
main memory: 504352K total, 444628K free, 428788K contig free, 324K cached
CPU states: 33.93% user, 39.87% system, 26.21% kernel, 0.00% idle
CPU time displayed (press 't' to cycle): user
PID USERNAME PRI NICE SIZE STATE TIME CPU COMMAND
5 root 4 0 212K 0:15 39.83% pm
733 root 7 0 600K RUN 0:06 16.89% sh
734 root 7 0 600K RUN 0:02 8.89% sh
735 root 7 0 600K RUN 0:02 8.13% sh
10 root 1 0 236K 0:00 0.01% tty
88 service 7 0 100K 0:00 0.01% random
7 root 5 0 1220K 0:00 0.01% vfs
28 root 7 0 888K RUN 0:00 0.01% procfs
12 root 2 0 2032K 0:00 0.00% vm
732 root 7 0 692K 0:00 0.00% top
6 root 4 0 36K 0:00 0.00% sched
37 service 5 0 6292K 0:00 0.00% mfs
103 root 7 0 100K 0:00 0.00% lance
107 service 7 0 1184K 0:00 0.00% inet
4 root 4 0 324K 0:00 0.00% rs
129 root 7 0 348K 0:00 0.00% noname
To grab input, press Ctrl+G