

Βάμβας Ιωάννης Α.Μ.: 2943

Γεωργουλας Βασίλης Α.Μ.: 2954

Βήμα 1

Αρχικά στην συνάρτηση `handle_vfs_reply()` του αρχείου `usr/src/servers/pm/main.c`, βλέπουμε το `case PM_FORK_REPLY` δηλαδή το `fork()` που ζητάει η εκφώνηση και εκεί καλείτε η συνάρτηση `sched_start_user()` του αρχείου `/usr/src/servers/pm/schedule.c`. Εκεί είναι η πρώτη αλλαγή που θα κάνουμε για να περάσει το γκρουπ μιας διεργασίας στην `do_start_scheduling()` του αρχείου `usr/src/servers/sched/schedule.c`. Στην `sched_start_user()`, έχουμε την κλήση της συνάρτησης `sched_inherit()`, η οποία δεχόταν 5 ορίσματα και εμείς αλλαξαμε το προτοτυπο της μέσω του αρχείου `usr/src/include/minix/sched.h` ώστε τώρα να δεχεται 6 ορίσματα όπου το τελευταίο ορίσμα θα είναι το γκρουπ της διεργασίας (`rmp->procgrp`). Στη συνέχεια η συνάρτηση `sched_inherit()` υλοποιείται στο αρχείο `usr/src/lib/libsys/sched_start.c` όπου εκεί βάζουμε το 6^ο ορίσμα στο πεδίο `m9_I5` του μηνυματος που θα σταλεί στον `sched` (το οποίο θα είναι το `SCHEDULING_INHERIT`). Το γκρουπ της διεργασίας το βάλουμε μέσα στο πεδίο `m9_I5` γιατί συμπεραναμε πως στο `SCHEDULING_INHERIT` δεν χρησιμοποιείτε αυτό το πεδίο του μηνυματος (πιο συγκεκριμένα το είδαμε μέσω του αρχείου `usr/src/include/minix/com.h`). Μετεπειτα βλέπουμε πως με το μήνυμα `SCHEDULING_INHERIT` καταληγουμε στο αρχείο `usr/src/servers/sched/main.c` όπου από εκεί βλέπουμε ότι για το `case SCHEDULING_INHERIT` καταληγουμε στην συνάρτηση `do_start_scheduling()` που αναφερθηκε και παραπάνω, όπου εκεί θα γίνει η αρχικοποίηση της ομάδας κάθε διεργασίας παίρνοντας το πεδίο `m9_I5` του μηνυματος.

Βήμα 2

Στο αρχείο `usr/src/servers/sched/schedproc.h` προσθεσαμε τα 4 πεδία που ζητούνται στην άσκηση, πιο συγκεκριμένα το `pid_t procgrp` (όπου είναι ο οδηγός ομάδας), το `unsigned proc_usage` που είναι η χρήση

διεργασίας, το `unsigned grp_usage` που είναι η χρήση του `group` διεργασιών και το `unsigned fss_priority` που είναι η προτεραιότητα με βάση τον αλγόριθμο δίκαιης δρομολόγησης. Αυτά τα 4 πεδία, τα αρχικοποιούμε στην συνάρτηση `do_start_scheduling()` του αρχείου `usr/src/servers/sched/schedule.c` και πιο συγκεκριμένα, αρχικοποιούμε το πεδίο `procgrp` ως `m_ptr->m9_15` αμέσως μετά την εισαγωγή της διεργασίας μέσα στην δομή `schedproc.h`. Στη συνέχεια μέσα στο `case SCHEDULING_INHERIT` αρχικοποιούμε τα υπολοιπά 3 πεδία της δομής μας ως εξής:

Το πεδίο `proc_usage` αρχικοποιείτε στο 0 γιατί η χρήση της διεργασίας μόλις μπει στην `do_start_scheduling` είναι 0 (γιατί ουσιαστικά δεν έχει γίνει ακόμα ο «προγραμματισμός» της διεργασίας από τον πυρήνα).

Το πεδίο `grp_usage` αρχικοποιείτε όσο είναι και το `grp_usage` της 1^{ης} διεργασίας (γονέα) γιατί ξέρουμε πως όλες οι διεργασίες του ίδιου γκρουπ θα πρέπει να έχουν το ίδιο `grp_usage`.

Το πεδίο `fss_priority` αρχικοποιείτε με βάση τον τύπο, όπου το `number_of_groups` το βρίσκουμε μέσω της συνάρτησης `num_of_grps()`, η οποία επιστρέφει το `number_of_groups`. Στη συνάρτηση `num_of_grps()` αρχικά βρίσκουμε όλες τις διεργασίες χρήστη και κρατάμε σε 1 πίνακα τους οδηγούς ομάδας των διεργασιών και στη συνέχεια μετράμε τους διαφορετικούς οδηγούς ομάδων που βρίσκουμε μέσα σε αυτόν τον πίνακα όπου θα είναι τελικά και το `number_of_groups` που ψαχνούμε.

Μετέπειτα η ενημέρωση των 4 παραπάνω πεδίων γίνεται στη συνάρτηση `do_noquantum()` του αρχείου `usr/src/servers/sched/schedule.c`, όπου ελέγχουμε αν έχουμε διεργασία χρήστη και αν έχουμε τότε τα πεδία ενημερώνονται ως εξής:

Το πεδίο `proc_usage` της διεργασίας που τελειωσε το κβάντο της, το αυξάνουμε κατά `rmp->time_slice` αν το `time_slice` είναι ίσο με `USER_QUANTUM` αλλιώς το αυξάνουμε κατά `USER_QUANTUM`.

Στη συνέχεια διατρέχουμε όλες τις διεργασίες που έχουμε στην δομή του `schedproc.h` και αυξάνουμε όλα τα `grp_usage` των διεργασιών που έχουν τον ίδιο οδηγό ομάδας με την διεργασία που εληξε το κβάντο της κατά `USER_QUANTUM`.

Τέλος για όλες τις διεργασίες χρήστη, ενημερώνουμε τα πεδία τους ως εξής:

To `proc_usage=proc_usage/2`

To `grp_usage=grp_usage/2`

Και το `fss_priority` με βάση τον τύπο `fss_priority = proc_usage/2 + grp_usage*number_of_groups/4 + base`, όπου `base=0` (είναι ο ίδιος τύπος που αναερίθηκε πιο πάνω στο βήμα 2 ως προς την αρχικοποίηση του `fss_priority`).

Βήμα 3

Για να είναι οι διεργασίες χρήστη σε 1 μόνο ουρά (όπως ζητείτε στην εκφώνηση), αλλάξαμε το αρχείο `usr/src/include/minix/config.h` και πια βάλαμε ως `NR_SCHED_QUEUEUES = 8` και `MAX_USER_Q = MIN_USER_Q = USER_Q=7` έτσι ώστε οι ουρές 0-6 να είναι ίδιες με πριν, η ουρά 7 να είναι η ουρά χρήστη και η ουρά 8 να είναι η ουρά `idle` (όπως ήταν και στην προηγούμενη έκδοση των ουρών του μινιξ). Το `MIN_USER_Q` είναι ίσο με το `MAX_USER_Q` γιατί πια έχουμε μόνο 1 ουρά χρήστη οπότε δεν νοείται το μιν να είναι διαφορετικό από το μαξ. Στη συνέχεια για να περάσουμε το `fss_priority` στον πυρήνα, κάνουμε τα εξής βήματα:

- 1) Αλλάζουμε το `prototype` της συνάρτησης `sys_schedule` ώστε αντί για 4 ορίσματα, να δεχεται 5, ώστε το 5^ο πεδίο της να είναι το `fss_priority`. Αυτό έγινε μέσω του αρχείου `usr/src/include/minix/syslib.h`.
- 2) Στο αρχείο `usr/src/lib/libsys/sys_schedule.c` όπου εκτελείτε η συνάρτηση `sys_schedule()` βάζουμε στο ελεύθερο πεδίο `m9_l5` του μηνυματος το `fss_priority` που περναμε από το βήμα 1. Μετεπειτα η `sys_schedule` κάνει κλήση συστήματος και μεταφέρει το μήνυμα στον κερνελ (`_kernel_call(SYS_SCHEDULE, &m)`).
- 3) Στη συνέχεια μετά την κλήση συστήματος το μήνυμα καταληγει στις συναρτήσεις `do_schedule()` και `do_schedctl()` (`SYS_XXX->do_XXX`) (οι οποίες βρίσκονται στο `/usr/src/kernel/system` και εκεί μέσα βάζουμε το `fss_priority` ίσο με το πεδίο του μηνυματος `m9_l5` όπου περναμε το `fss_priority` του βήματος 2) όπου αυτές

στελνουν το `fss_priority` στη `sched_process` του αρχείου `system.c` (οπου αλλαξαμε το `prototype` της συναρτησης `sched_proc()` μεσω του αρχείου `/usr/src/kernel/proto.h` ώστε η συναρτηση `sched_proc()` πια να δεχεται 5 ορισματα, οπου το 5^ο ορισμα θα είναι το `fss_priority`) και εκει καθε διεργασία χρηστη περνει το αντιστιχο `fss_priority` ενημερωνοντας τον πινακα `proc.h`.

Εχοντας πια για κάθε διεργασία χρηστη το `fss_priority` (αυτό εξασφαλίζεται μεσω της συναρτησης `schedule_process()` του αρχείου `usr/src/servers/sched/schedule.c`, οπου αν εχουμε διεργασία χρηστη τοτε καλουμε μια φορ ώστε να στείλουμε στον πυρηνα τα `fss_priority` όλων των διεργασιων χρηστη) ώστε ο πυρηνας να διαλεξει προς εκτελεση την διεργασία χρηστη με το μικροτερο `fss_priority`.

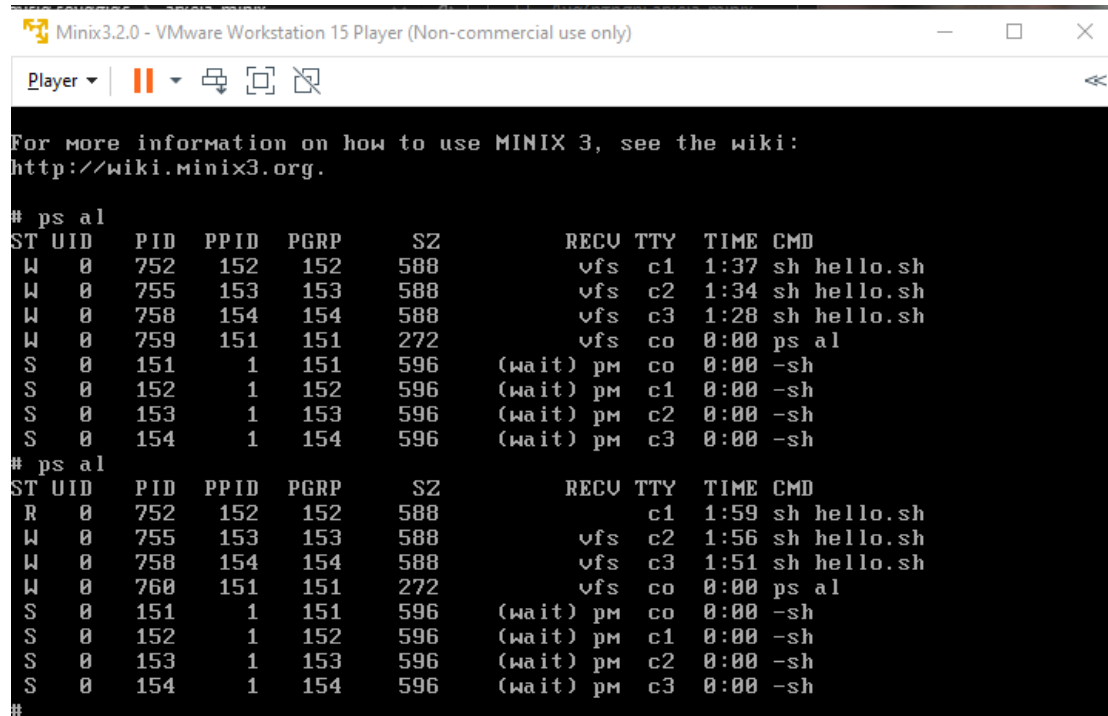
Στη συνεχεια για να διαλεγει ο πυρηνας την διεργασία χρηστη με το χαμηλοτερο `fss_priority` αλλαξαμε την συναρτηση `pick_proc()` στο αρχείο `/usr/src/kernel/proc.c`. Αρχικα ειχαμε από πριν για κάθε ουρα την 1^η διεργασία καθε ουρας που ηταν ετοιμη προς εκτελεση, εμεις προσθεσαμε και να εχουμε και την τελευταια διεργασία κάθε ουρας που είναι ετοιμη προς εκτελεση (`rdy_tail`). Στη συνεχεια επειδη θελουμε να αλλαξουμε την πολιτικη μονο για τις διεργασίες χρηστη βαλαμε μια `if` η οποια ελεγχει αν ειμαστε στην ουρα χρηστη, αλλιως η μεθοδος κανει ότι εκανε και πριν. Αν βρουμε ότι ειμαστε στην ουρα χρηστη, τοτε ελεγχουμε αν η 1^η διεργασία της ουρας χρηστη είναι μη κενη (δηλαδη είναι ετοιμη προς εκτελεση), αν δεν είναι τοτε συνεχιζουμε στην επομενη ουρα (ουσιαστικα βγαινουμε από την φορ) ενώ αν είναι ετοιμη προς εκτελεση τοτε το `min` γινεται ισο με το `p_fss_priority` της 1^{ης} διεργασιας που είναι ετοιμη για εκτελεση. Στη συνεχεια οσο εχουμε ετοιμες διεργασίες προς εκτελεση στην ουρα χρηστη, ελεγχουμε αν το `p_fss_priority` αυτων των διεργασιων είναι μικροτερο της 1^{ης} διεργασιας και αν είναι τοτε θα εκτελεσουμε καποια από αυτές τις διεργασίες. Το οσο εχουμε ετοιμες διεργασίες προς εκτελεση το βρισκουμε μεσω του πεδιου `p->p_nextready` της δομης `proc.h` και ουσιαστικα ελεγχουμε αν το `p` είναι διαφορετικο του `end` (οπου το `end` θα είναι η τελευταια διεργασία της ουρας που θα είναι ετοιμη προς εκτελεση). Αρχικα το `p` θα είναι ισο με την 1^η διεργασία της ουρας που είναι ετοιμη προς εκτελεση (`rdy_head[q]`)

ενώ το end θα είναι ισο με την τελευταία διεργασία της ουρας που είναι ετοιμη προς εκτελεση δηλαδη(rdy_tail[q]). Τελος επιβεβαιωνουμε με την εντολη assert πως η διεργασία χρηστη που επιλεξαμε να εκτελεστει, μπορεί οντως να εκτελεστει(δηλαδη δεν είναι «κενη» ή κατι τετοιο).

Βημα 4

Φτιαξαμε 1 σκριπτ το οποιο το ονομασαμε hello.sh και το τρεξαμε σχεδον ταυτοχρονα σε 3 διαφορετικα τερματικο(στο 2^ο, στο 3^ο και στο 4^ο και στο 1^ο τερματικο με την εντολη ps al βλεπαμε τον χρονο εκτελεσης του σκριπτ στα τερματικα). Το σκριπτ μας απλως εκτυπωνει συνεχως την λεξη τεστ(με while(1)).

Μετα από 5 λεπτα(το 1^ο ps al) και 6 λεπτα εκτελεσης(το 2^ο ps al) εκτελεσης ειχαμε αυτό εδώ το αποτελεσμα:



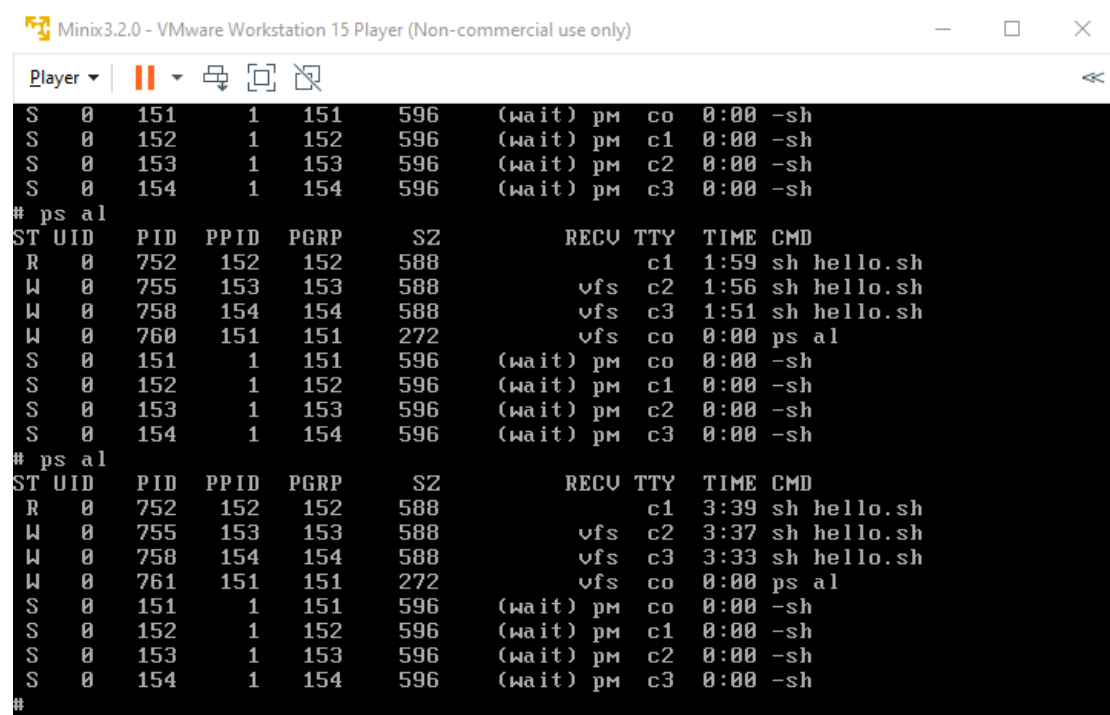
```
For more information on how to use MINIX 3, see the wiki:
http://wiki.minix3.org.

# ps al
ST UID  PID  PPID  PGRP  SZ      RECV TTY  TIME CMD
W  0    752   152   152   588      vfs c1  1:37 sh hello.sh
W  0    755   153   153   588      vfs c2  1:34 sh hello.sh
W  0    758   154   154   588      vfs c3  1:28 sh hello.sh
W  0    759   151   151   272      vfs co 0:00 ps al
S  0    151    1    151   596    (wait) pm co 0:00 -sh
S  0    152    1    152   596    (wait) pm c1 0:00 -sh
S  0    153    1    153   596    (wait) pm c2 0:00 -sh
S  0    154    1    154   596    (wait) pm c3 0:00 -sh

# ps al
ST UID  PID  PPID  PGRP  SZ      RECV TTY  TIME CMD
R  0    752   152   152   588      c1  1:59 sh hello.sh
W  0    755   153   153   588      vfs c2  1:56 sh hello.sh
W  0    758   154   154   588      vfs c3  1:51 sh hello.sh
W  0    760   151   151   272      vfs co 0:00 ps al
S  0    151    1    151   596    (wait) pm co 0:00 -sh
S  0    152    1    152   596    (wait) pm c1 0:00 -sh
S  0    153    1    153   596    (wait) pm c2 0:00 -sh
S  0    154    1    154   596    (wait) pm c3 0:00 -sh

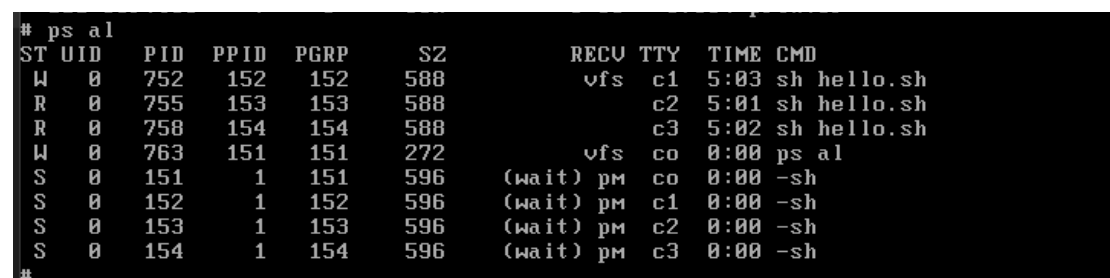
#
```

Μετα από 10 λεπτα(το τελευταία ps al που φαινεται κατω κατω στην εικονα) εκτελεσης ειχαμε αυτό εδώ το αποτελεσμα:



```
Minix3.2.0 - VMware Workstation 15 Player (Non-commercial use only)
Player
S 0 151 1 151 596 (wait) pm co 0:00 -sh
S 0 152 1 152 596 (wait) pm c1 0:00 -sh
S 0 153 1 153 596 (wait) pm c2 0:00 -sh
S 0 154 1 154 596 (wait) pm c3 0:00 -sh
# ps al
ST UID PID PPID PGRP SZ RECV TTY TIME CMD
R 0 752 152 152 588 c1 1:59 sh hello.sh
W 0 755 153 153 588 vfs c2 1:56 sh hello.sh
W 0 758 154 154 588 vfs c3 1:51 sh hello.sh
W 0 760 151 151 272 vfs co 0:00 ps al
S 0 151 1 151 596 (wait) pm co 0:00 -sh
S 0 152 1 152 596 (wait) pm c1 0:00 -sh
S 0 153 1 153 596 (wait) pm c2 0:00 -sh
S 0 154 1 154 596 (wait) pm c3 0:00 -sh
# ps al
ST UID PID PPID PGRP SZ RECV TTY TIME CMD
R 0 752 152 152 588 c1 3:39 sh hello.sh
W 0 755 153 153 588 vfs c2 3:37 sh hello.sh
W 0 758 154 154 588 vfs c3 3:33 sh hello.sh
W 0 761 151 151 272 vfs co 0:00 ps al
S 0 151 1 151 596 (wait) pm co 0:00 -sh
S 0 152 1 152 596 (wait) pm c1 0:00 -sh
S 0 153 1 153 596 (wait) pm c2 0:00 -sh
S 0 154 1 154 596 (wait) pm c3 0:00 -sh
#
```

Μετα από 15 λεπτα εκτελεσης ειχαμε αυτό εδώ το αποτελεσμα:



```
# ps al
ST UID PID PPID PGRP SZ RECV TTY TIME CMD
W 0 752 152 152 588 vfs c1 5:03 sh hello.sh
R 0 755 153 153 588 c2 5:01 sh hello.sh
R 0 758 154 154 588 c3 5:02 sh hello.sh
W 0 763 151 151 272 vfs co 0:00 ps al
S 0 151 1 151 596 (wait) pm co 0:00 -sh
S 0 152 1 152 596 (wait) pm c1 0:00 -sh
S 0 153 1 153 596 (wait) pm c2 0:00 -sh
S 0 154 1 154 596 (wait) pm c3 0:00 -sh
#
```

Όπως φαινεται από τις παραπανω εικονες, ο χρονος ισομοιραζεται μεταξυ των 3 διαφορετικων τερματικων που τρεχουμε το σκριπτ που περιγραφηκε παραπανω.