

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 αλλάξαμε το προτοτυπο της μέσω του αρχείου `usr/src/include/minix/sched.h` ώστε τώρα να δεχεται 6 ορίσματα όπου το τελευταίο όρισμα 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 όρισμα 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 τελείωσε 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, ενημερωνούμε 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 μινix). 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 μηνύματος το `fss_priority` που περναμε από το βήμα 1. Μετεπείτα 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()` (οι οποίες βρίσκονται στο `/usr/src/kernel/system` και εκεί μέσα βάζουμε το `fss_priority` ίσο με το πεδίο του μηνύματος `m9_l5` όπου περναμε το `fss_priority` του βήματος 2) όπου αυτές στέλνουν το

fss_priority στη sched_process του αρχείου system.c(όπου άλλαξαμε το prototype της συνάρτησης sched_proc() μέσω του αρχείου /usr/src/kernel/proto.h ώστε η συνάρτηση sched_proc() πια να δέχεται 5 ορίσματα, όπου το 5^ο όρισμα θα είναι το fss_priority) και εκεί κάθε διεργασία χρήστη περνεί το αντίστοιχο fss_priority ενημερώνοντας τον πίνακα proc.h.

Εχοντας πια για κάθε διεργασία χρήστη το fss_priority(αυτό εξασφαλίζεται μέσω της συνάρτησης schedule_process() του αρχείου usr/src/servers/sched/schedule.c, όπου αν έχουμε διεργασία χρήστη τότε καλούμε μια φορά ώστε να στείλουμε στον πυρήνα τα fss_priority όλων των διεργασιών χρήστη) ώστε ο πυρήνας να διαλέξει προς εκτέλεση την διεργασία χρήστη με το μικρότερο fss_priority.

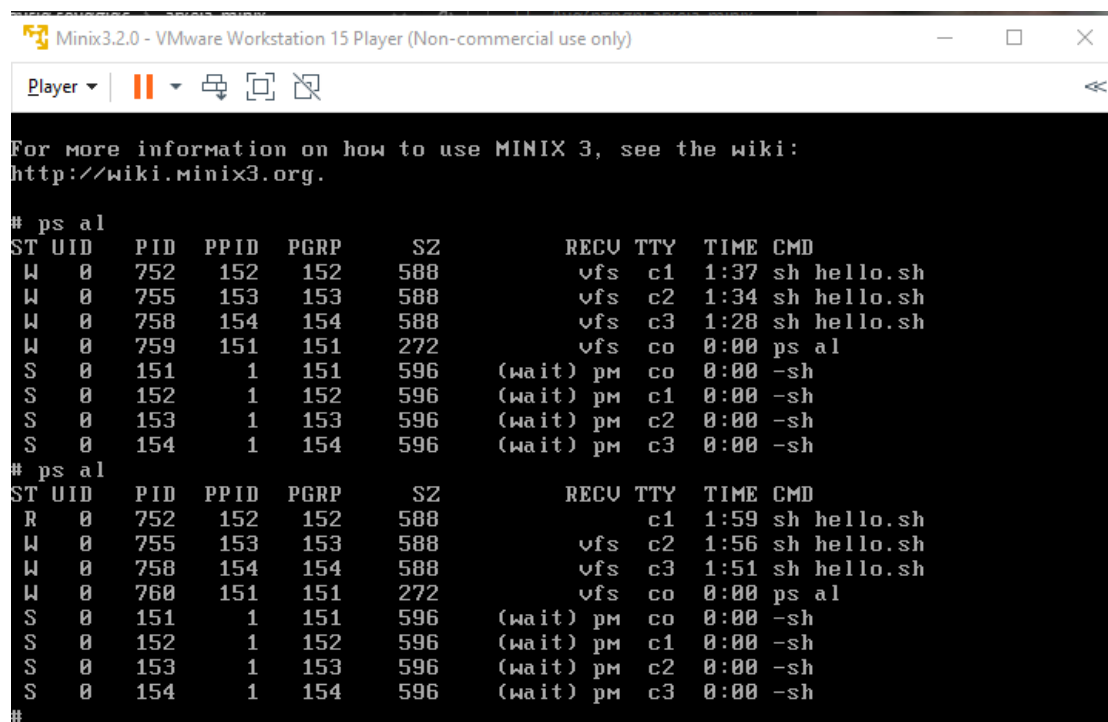
Στη συνέχεια για να διαλέγει ο πυρήνας την διεργασία χρήστη με το χαμηλότερο fss_priority άλλαξαμε την συνάρτηση pick_proc() στο αρχείο /usr/src/kernel/proc.c. Αρχικά είχαμε από πριν για κάθε ουρά την 1^η διεργασία κάθε ουράς που ήταν έτοιμη προς εκτέλεση, εμείς προσθέσαμε και να έχουμε και την τελευταία διεργασία κάθε ουράς που είναι έτοιμη προς εκτέλεση(rdy_tail). Στη συνέχεια επειδή θέλουμε να αλλάξουμε την πολιτική μόνο για τις διεργασίες χρήστη βάλαμε μια if η οποία ελέγχει αν είμαστε στην ουρά χρήστη, αλλιώς η μέθοδος κάνει ότι έκανε και πριν. Αν βρούμε ότι είμαστε στην ουρά χρήστη, τότε ελέγχουμε αν η 1^η διεργασία της ουράς χρήστη είναι μη κενή(δηλαδή είναι έτοιμη προς εκτέλεση), αν δεν είναι τότε συνεχίζουμε στην επόμενη ουρά(ουσιαστικά βγαίνουμε από την φάση) ενώ αν είναι έτοιμη προς εκτέλεση τότε το min γίνεται ίσο με το p_fss_priority της 1^{ης} διεργασίας που είναι έτοιμη για εκτέλεση. Στη συνέχεια όσο έχουμε έτοιμες διεργασίες προς εκτέλεση στην ουρά χρήστη, ελέγχουμε αν το p_fss_priority αυτών των διεργασιών είναι μικρότερο της 1^{ης} διεργασίας και αν είναι τότε θα εκτελέσουμε κάποια από αυτές τις διεργασίες. Το όσο έχουμε έτοιμες διεργασίες προς εκτέλεση το βρίσκουμε μέσω του πεδίου p->p_nextready της δομής proc.h και ουσιαστικά ελέγχουμε αν το p είναι διαφορετικό του end(όπου το end θα είναι η τελευταία διεργασία της ουράς που θα είναι έτοιμη προς εκτέλεση). Αρχικά το p θα είναι ίσο με την 1^η διεργασία της ουράς που είναι έτοιμη προς εκτέλεση(rdy_head[q]) ενώ το end θα είναι ίσο με την τελευταία διεργασία της ουράς 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:



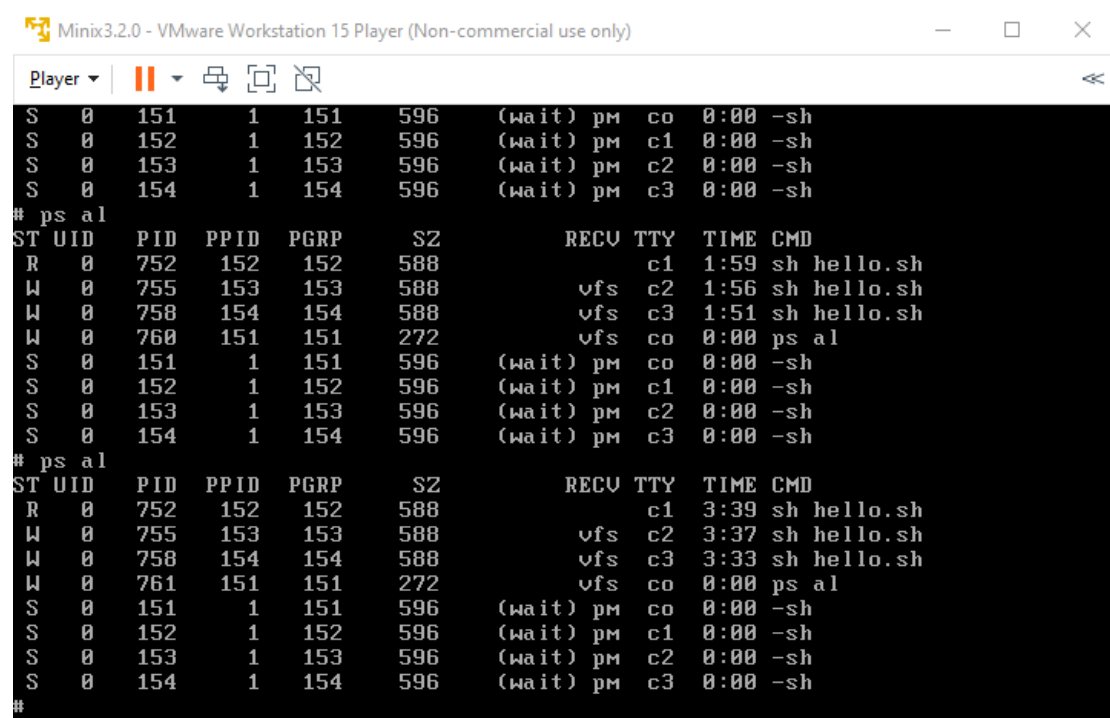
The screenshot shows a VMware Workstation 15 Player window titled "Minix3.2.0 - VMware Workstation 15 Player (Non-commercial use only)". Inside the window is a terminal window with a black background and white text. The terminal displays the output of the command "# ps al" twice. The first output shows processes running at 1:37, 1:34, and 1:28. The second output shows processes running at 1:59, 1:56, and 1:51. The processes are listed with columns for ST, UID, PID, PPID, PGRP, SZ, RECV, TTY, TIME, and CMD. The CMD column shows "sh hello.sh" for the first three processes and "ps al" for the fourth process in each output. The terminal also shows a prompt "# _" at the bottom.

```
# 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

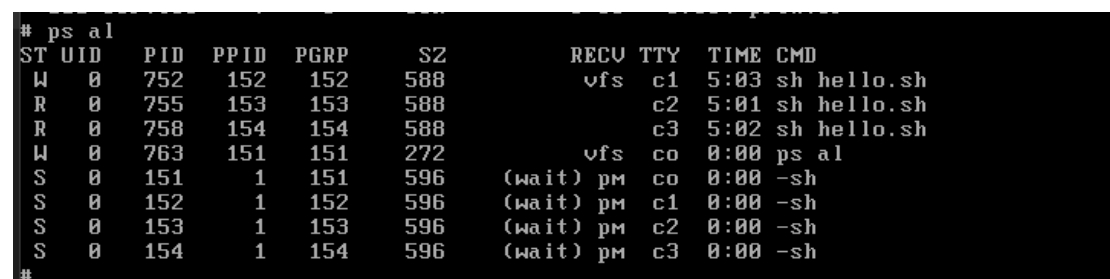
# _
```

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



```
Minix3.2.0 - VMware Workstation 15 Player (Non-commercial use only)
Player
# 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.