Γιώργος Δημόπουλος Α.Μ. 2964
Άγγελος Κατσαλήρος Α.Μ. 2997

2η ΑΣΚΗΣΗ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

1ο Ερώτημα:

- ✔ Γνωρίζουμε ότι μια διεργασία δημιουργείται στην **do_fork()**, η οποία βρίσκεται στο path: *usr/src/servers/pm/forkexit.c. Στην συνάρτηση* **do_fork()** βλέπουμε ότι δημιουργείται ένα μήνυμα message m; Το μήνυμα m βλέπουμε ότι είναι πεδίο του μηνύματος m.m_type = PM_FORK; και επίσης στέλνεται μέσω της tell_vfs(rmc, &m); η οποία παίρνει σαν όρισμα το μήνυμα και το δείκτη, ο οποίος δείχνει στο struct mproc. *Παρατηρούμε κάτι το οποίο θα είναι πολύ χρήσιμο για μετά: ο rmp είναι pointer στην διεργασία του πατέρα και δείχνει στο struct mproc.*

- ✔ Τώρα πρέπει να δω που χρησιμοποιείται η PM_FORK() για αυτό κάνω αναζήτηση από το minix με την εντολή search PM_FORK *usr*/src. Έτσι βλέπω ότι χρησιμοποιείται στο path: usr/src/pm/main.c με την μορφή PM_FORK_REPLY. Μέσα σε αυτό το αρχείο, στην συνάρτηση handle_vfs_reply() και μέσα στο PM_FORK_REPLY() βλέπουμε ότι καλείται η συνάρτηση sched_start_user() η οποία παίρνει σαν όρισμα την δομή μιας διεργασίας.

- ✔ Επομένως τώρα πρέπει να κάνω αναζήτηση για να δω που χρησιμοποιείται η sched_start_user(). Και κάνω αναζήτηση από το minix με την εντολή search sched_start_user *usr*/src. Έτσι βλέπουμε ότι η συνάρτηση υπάρχει στο path usr/src/pm/schedule.c. Τώρα μέσα στο αρχείο αυτό και στην συνάρτηση αυτή βλέπω ότι υπάρχει η συνάρτηση sched_inherit(). Η συνάρτηση sched_inherit() βλέπουμε ότι έχει όλες τις πληροφορίες της διεργασίας. Έχει το rmp το οποίο όπως αναφέρεται στην πρώτη βουλίτσα δείχνει στο struct mproc.

- ✔ Τώρα πρέπει να δούμε που ορίζεται το struct. Αυτό το κάνουμε με την εντολή στο minix search mproc *usr*/src και βρίσκουμε ότι υπάρχει στο usr/src/servers/pm/mproc.h Πατάμε πάνω στο mproc από το πηγαιο κώδικα που μας δίνεται και βλέπουμε τα πεδία του struct. Βρίσκουμε στα πεδία μέσα το pid_t mp_procgrp; το οποίο είναι η ταυτότητα του οδηγού ομάδας, που θέλουμε να στείλουμε στο shed.

- ✔ Τώρα πρέπει να το περάσουμε στην sched_inherit() σαν όρισμα επειδή έχει όλες τις πληροφορίες της διεργασίας έτσι ώστε να έχει και την πληροφορία της ταυτότητας ομάδος. Αφού βάλουμε το κανούριο όρισμα pid_t mp_procgrp; θα αλλάξουμε και την sched_inherit() μέσα στην schedule.c. Για να κάνουμε τις παραπάνω αλλαγές πρέπει να κάνουμε search την sched_inherit() που αυτό γίνεται με το minix με την εντολή search sched_inherit *usr*/src. Η εντολή σαν αποτέλεσμα μας βγάζει δύο αρχεία: *usr*/src/include/minix/sched.h που εδώ απλά ορίζεται. Το δεύτερο αρχείο είναι στο path usr/src/lib/libsys/sched_start.c όπου εδώ θα προσθέσουμε το όρισμα int group_pid. Επίσης μέσα στην shedule εκεί που καλείται η sched_inherit() προσθέτουμε το rmp→mp_procgrp.

- ✔ Μέσα στην sched_inherit έχουμε το SCHEDULING_PARENT. Για να βρούμε ποιο πεδίο είναι ελεύθερο για να στείλουμε το μήνυμα κάνουμε search από το minix με την εντολή search SCHEDULING_PARENT usr/src και σαν αποτέλεσμα η εντολή μας βγάζει usr/src/include/minix/com.h και το usr/src/servers/sched/schedule.c. Στο com.h

παρατηρούμε ότι έχουμε τα εξής: 1. SCHEDULING_ENDPOINT m9_l1 2. SCHEDULING_QUANTUM m9_l2 3. SCHEDULING_PARENT m9_l3 3. SCHEDULING_MAXPRIO m9_l4. Στο αρχείο sched_start.c στην συνάρτηση sched_inherit δεν χρησιμοποιούμε το m9_l5 επομένως επειδή αυτό είναι κενό θα στείλουμε την ταυτότητα του οδηγού ομάδας με το μήνυμα m ως m9_l5=group_pid.

- ✔ Τώρα θα δούμε και το αρχείο schedule.c που και εκεί βρίσκεται το SCHEDULING_PARENT όπως μας έδειξε η αναζήτηση μας. Μέσα στο schedule.c έχω την συνάρτηση do_start_scheduling όπου αυτή εισάγει μια νέα διεργασία στο struct schedproc. Πρέπει να κάνω αναζήτηση από το minix να δω που βρίσκεται με την εντολή: search schedproc usr/src και μας δίνει σαν αποτέλεσμα usr/src/servers/sched/schedproc.h, βλέπουμε ότι το struct έχει όλες τις πληροφορίες για την χρονοδρομολόγηση της διεργασίας, επομένως θα πρέπει να προσθέσουμε ένα πεδίο int procgrp για τον οδηγό ομάδας, επειδή θέλουμε να ενημερώνουμε και την ταυτότητα του οδηγού ομάδας. Τέλος στο schedule.c προσθέτουμε και την εντολή rmp ->procgrp = m_ptr -> m9_l5;

- ✔ Αντιμετωπίσαμε ένα πρόβλημα: μας έβγαζε ότι η συνάρτηση sched_inherit() παίρνει 6 ορίσματα ενώ έχει 5. Για αυτό έπρεπε να ψάξουμε να δούμε που είναι το πρότυπο της συνάρτησης. Βρήκαμε ότι είναι στο path: usr/src/include/minixsched.h οπότε πήγαμε εκεί με το vi και προσθέσαμε αυτό που έπρεπε στο πρωτότυπο.

- ✔ Βάλαμε ένα printf("The id of group is %ld\n",m_ptr→SCEDULING_QUANTUM); που τυπώνει τον αριθμό της ταυτότητας ομάδας. Αλλά το βάλαμε σε σχόλια επειδή τυπωνόταν συνέχεια.

2ο ΕΡΩΤΗΜΑ:

- ➢ Στην αρχή μπαίνουμε στο αρχείο schedproc.h και στην δομή schedproc προσθέτουμε τα πεδία int procgrp, int proc_usage, int grp_usage και fss_priority.

- ➢ Τώρα πρέπει να μπούμε στο αρχείο schedule.c του διακομιστή sched. Πρέπει να πάμε στην συνάρτηση do_start_scheduling, έτσι ώστε να αρχικοποιήσουμε τα πεδία της διεργασίας, η οποία προορίζεται για χρονοδρομολόγηση. Περνάμε το procgrp μέσω μηνύματος rmp->procgrp=m_ptr->m9_l5. Βάζουμε το proc_usage=0. Για το grp_usage αυτό που κάνουμε είναι να ελέγξουμε κάθε διεργασία αν έχει το ίδιο procgrp. Αν έχει το ίδιο procgrp με άλλη διεργασία απλά της βάζουμε το ίδιο grp_usage με αυτή την διεργασία, διαφορετικά θα είναι 0. Για να διατρέξουμε τον πίνακα χρησιμοποιήσαμε την ίδια λογική με την συνάρτηση balance_ queue. Τώρα επειδή θέλουμε να υπολογίσουμε το fss_priority θα πρέπει να βρούμε το number_of_groups, δηλαδή πόσες διαφορετικές ομάδες υπάρχουν. Για να το πετύχουμε αυτό αντιγράφουμε το procgrp κάθε διεργασίας του schedproc σε έναν άλλο πίνακα temp_array. Τώρα ταξινομούμε τον temp_array με τον αλγόριθμο ταξινόμησης selection. Δημιουργήσαμε την συνάρτηση swap γιατί την χρειαζόμαστε για την ταξινόμηση. Τώρα πλέον στον ταξινομημένο πίνακα ελέγχουμε κάθε στοιχείο με το προηγούμενο του και αν έχουν διαφορετικό procgrp τώρα αυξάνουμε το number_of_groups κατά 1. Τέλος αφού βρήκαμε και το number_of_groups ενημερώνουμε κατάλληλα το fss_priority.

- ➢ Το επόμενο βήμα είναι να πάμε στην συνάρτηση do_no_ quantum. Αρχικά θα προσθέσουμε στο grp_usage και στο proc_usage 200. Τώρα διατρέχουμε τον πίνακα schedproc σύμφωνα με την συνάρτηση balance_queue. Αυτό το κάνουμε για να ενημερώσουμε το grp_usage όλων των διεργασιών που βρίσκονται στην ίδια ομάδα.

Ακριβώς με τον ίδιο τρόπο προσπαθούμε να βρούμε το number_of_groups όπως στην do_start_scheduling, έτσι ώστε να το χρησιμοποιήσουμε στην ενημέρωση του fss_priority. Στην συνέχεια θέλουμε να ενημερώσουμε όλα τα πεδία που ανήκουν στις διεργασίες χρήστη. Για αυτόν τον λόγο τρέχουμε τον πίνακα ακριβώς με τον ίδιο τρόπο απλά του λέμε ότι θέλουμε να κάνεις τις ενημερώσεις μόνο όταν το priority=7, δηλαδή για τις διεργασίες που ανήκουν στον χρήστη. Βάζουμε priority=7 γιατί στο 3^{ο} ερώτημα θα πρέπει να μειώσουμε τις ουρές χρήστη σε μόνο μια. Μετά από όλες αυτές τις ενέργειες μπορούμε να ενημερώσουμε κατάλληλα τα πεδία: grp_usage, proc_usage και fss_priority.

3ο ΕΡΩΤΗΜΑ:

- Αρχικά στο kernel/proc.h πρέπει να μειώσουμε το συνολικό πλήθος ουρών έτσι ώστε όλες οι διεργασίες χρήστη να βρίσκονται σε μόνο μια ουρά. Για να υπάρχει μόνο μια ουρά θα πρέπει το MAX_USER_Q=MIN_USER_Q=USER_Q. Επειδή οι ουρές από 0-6 είναι ουρές συστήματος θα βάλουμε το USER_Q=7. Θέλουμε ακόμη μια ουρά IDLE, οπότε θα βάλουμε το NR_SCHED_QUEUES=8.

- Προκειμένου ο πυρήνας να διαλέξει την διεργασία με το χαμηλότερο fss_priority, θα πρέπει να βρούμε τρόπο να περάσουμε το fss_priority από τον διακομιστή sched στον πυρήνα kernel και συγκεκριμένα στην συνάρτηση pick_proc του αρχείου proc.c.

- Στη do_no_ quantum μόλις γίνει η ενημέρωση των πεδίων βλέπουμε να καλείται η schedule_process_local. Ψάχνοντας αυτή την συνάρτηση στο ίδιο αρχείο βλέπουμε ότι καλεί την συνάρτηση sys_schedule, η οποία βρίσκεται στον κατάλογο lib/libsys. Αυτή η συνάρτηση βλέπουμε ότι περνάει ένα μήνυμα στον kernel. Άρα σκοπός μας είναι να μεταφέρουμε όλα τα fss_priority των διεργασιών χρήστη μέσω μηνύματος στον kernel.

- Αυτό θα το πετύχουμε κάνοντας μια for στην do_no_quantum στο τέλος καλώντας την schedule_process_local για κάθε διεργασία χρήστη. Μετά στην schedule_process εκεί που καλείται η sys_schedule θα της βάλουμε ένα επιπλέον όρισμα, το οποίο θα είναι το fss_priority της διεργασίας. Εφόσον τώρα αλλάξαμε την sys_schedule θα πρέπει να την αλλάξουμε και στον κατάλογο lib/libsys και στην syslib.h (εκεί βρίσκεται το PROTOTYPE της συνάρτησης), που βρίσκεται στον κατάλογο include/minix.

- Αφού περάσαμε το fss_priotity στην sys_schedule σαν παράμετρο πρέπει να το περάσουμε και σαν μήνυμα στον kernel. Βλέπουμε ότι χρησιμοποιεί ήδη τους εξής τύπους μηνύματος: m9_l1, m9_s1, m9_l2 και m9_l4. Επομένως εμείς θα χρησιμοποιήσουμε το μήνυμα m9_l5 για να στείλουμε το fss_priority. Τώρα πρέπει να βρούμε που καλείται αυτό το μήνυμα στον πυρήνα.

- Επειδή στόχος μας είναι να βρούμε την διεργασία με το μικρότερο fss_priority στην pick_proc θα πάμε να δούμε τι δομές χρησιμοποιεί η pick_proc. Βλέπουμε ότι χρησιμοποιεί το struct proc. Το struct proc ορίζεται στο αρχείο proc.h. Το struct proc περιέχει τις πληροφορίες της διεργασίας. Επειδή θέλουμε κάθε διεργασία-proc να έχει το fss_priority θα φτιάξουμε σε αυτό ένα πεδίο. Αφού φτιάξαμε και το πεδίο στο struct θέλουμε να περάσουμε το fss_priority που στάλθηκε σαν μήνυμα από το διακομιστή sched.

- Ψάχνοντας στα προτεινόμενα αρχεία της εκφώνησης βλέπουμε ότι καλείται στα αρχεία του do_schedule και του do_fork, τα οποία βρίσκονται στον κατάλογο system.

- Έτσι είδαμε ότι στο αρχείο kernel/system/do_schedule.c περνιέται το μήνυμα. Μέσα στο αρχείο στην συνάρτηση sched_proc περνιούνται ως ορίσματα τα μηνύματα που στάλθηκαν. Επειδή τώρα θέλουμε να περάσουμε το μήνυμα m9_l5 στο οποίο είναι η προτεραιότητα της

διεργασίας θα το βάλουμε στο stuct_proc στο πεδίο fss_priority. Άρα με αυτόν τον τρόπο περάσαμε το μήνυμα στην διεργασία του πυρήνα.

- Παρατηρούμε ότι το struct proc έχει 3 pointers: *p_nextready, *p_caller_q, *p_q_link και έτσι συμπεράναμε ότι είναι συνδεδεμένη λίστα. Αυτό που θέλουμε να κάνουμε είναι ότι στην for που διατρέχει όλες τις ουρές διεργασιών(στην συνάρτηση pick_proc του proc.c) να πάμε στην ουρά που έχει τις διεργασίες χρήστη. Όταν φτάσουμε σε αυτήν πρέπει να βρούμε ποια διεργασία έχει το μικρότερο fss_priority και να το επιστρέψουμε. Αρχικά λέμε ότι η διεργασία με το μικρότερο fss_priority είναι αυτή που είναι πρώτη στην ουρά και την αποθηκεύουμε στην μεταβλητή rp και στην μεταβλητή temp_fss_priority αποθηκεύουμε την τιμή της προτεραιότητας. Σε μια μεταβλητή register struct proc *temp αποθηκεύουμε την επόμενη διεργασία μέσω της εντολής rdy_head[q]->p_nextready. Η *p_nextready ουσιαστικά είναι ένας pointer ο οποίος δείχνει στην επόμενη έτοιμη διεργασία. Τέλος μέσω μιας while διαπερνάμε όλη την λίστα και αν βρούμε διεργασία με χαμηλότερο fss_priortity ενημερώνουμε τις κατάλληλες μεταβλητές μας.

4ο ΕΡΩΤΗΜΑ:

- Δημιουργούμε ένα αρχείο test.sh μέσα στο path usr/src και μέσα στο αρχείο βάζουμε τον κώδικα:
  #!/bin/sh
  while true
  do

  done

- Τέλος το τρέχουμε με την εντολή sh test.sh.

```
# ps
  PID TTY  TIME CMD
  749  c1  0:02 sh test.sh
  750  c2  0:01 sh test.sh
  751  c3  0:01 sh test.sh
  752  co  0:00 ps
  151  co  0:00 -sh
  152  c1  0:00 -sh
  153  c2  0:00 -sh
  154  c3  0:00 -sh
```

```
# ps
  PID TTY  TIME CMD
  749  c1  0:10 sh test.sh
  750  c2  0:10 sh test.sh
  751  c3  0:10 sh test.sh
  753  co  0:00 ps
  151  co  0:00 -sh
  152  c1  0:00 -sh
  153  c2  0:00 -sh
  154  c3  0:00 -sh
#
```

```
# ps
  PID TTY  TIME CMD
  749  c1  0:32 sh test.sh
  750  c2  0:32 sh test.sh
  751  c3  0:32 sh test.sh
  756  co  0:00 ps
  151  co  0:00 -sh
  152  c1  0:00 -sh
  153  c2  0:00 -sh
  154  c3  0:00 -sh
#
```

Τρέξαμε σε 3 τερματικά το αρχείο test.sh και βλέπουμε ότι ο χρόνος ισομοιράζεται μεταξύ των τριών τερματικών.