ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΟ 2: REPORT

Φοιτητές:

Ον/μο: Κρομμύδας Αστέριος, ΑΜ: 2471

Ον/μο: Παπαστεριάδης Γεώργιος, ΑΜ: 2615

Ερώτημα 1ο

Υλοποίηση

Ψάχνοντας για την κλήση fork() από την διεργασία pm, βρήκαμε στο αρχείο servers/pm/forkexit.c οτι κατασκευάζεται το μήνυμα m, το οποίο στέλνεται με την εντολή tell_vfs(), και επιπλέον βρήκαμε την εντολή “m.m_type = PM_FORK;”. Έπειτα ψάχνοντας για την PM_FORK βρήκαμε στο servers/pm/main.c την case PM_FORK_REPLY όπου καλείται η handle_vfs_reply(). Ψάξαμε για αυτή την συνάρτηση και εντός της υπάρχει case PM_FORK_REPLY όπου μέσα καλείται η sched_start_user(). Ψάξαμε για αυτη την συνάρτηση. Υπάρχει στο servers/pm/schedule.c. Στο return της καλείται η sched_inherit()που στέλνει το μήνυμα στον sched. Στα ορίσματα της προσθέσαμε επιπλέον το rmp-

>mp_procgrp. Επειτα ψάξαμε να βρούμε που υλοποιείται για να την αλλάξουμε. Αυτό γίνεται στα include/minix/sched.h & lib/libsys/sched_start.c. Στο πρώτο βρήκαμε το PROTOTYPE της συνάρτησης όπου προσθέσαμε το νέο όρισμα. Στην sched_start.c ορίζονται τα 3 πεδία του μηνύματος. Ψάξαμε και βρήκαμε στο include/minix/com.h τους τυπους των πεδίων αυτών που είναι m9_l1, m9_l3 και m9_l4. Για το νέο πεδίο που θέλαμε να προσθέσουμε(procgrp), διαλέξαμε το m9_l5 που ήταν ελεύθερο και το προσθέσαμε στην sched_start.c. Για να επιβεβαιώσουμε οτι όσα κάναμε δουλεύουν, στην do_start_scheduling() του αρχείου servers/sched/schedule.c προσθέσαμε μια εντολή printf() που τυπώνει το procgroup.

Ερώτημα 2ο

Υλοποίηση

Στο αρχείο servers/sched/schedproc.h, στο struct schedproc προσθέσαμε τα 4 πεδία που ζητούνται. Στο αρχείο schedule.c τους δώσαμε τιμές στην do_start_scheduling().

Για να ενημερώσουμε τα πεδία όταν μια διεργασία ολοκληρώνει ενα κβάντο, βρήκαμε την do_noquantum() όπου προσθέσαμε τα εξής:

Προσθέτουμε 200 στα proc_usage & grp_usage, ανανεώνουμε το grp_usage για τις διεργασίες με το ίδιο procgrp στο struct schedproc, αντιγραφουμε το schedproc στον allprocgrps τον οποίο κανουμε sort για να βρουμε το πλήθος των groups και ανανεώνουμε τα πεδία σύμφωνα με τους τύπους που μας δώθηκαν.

Ερώτημα 3ο

Υλοποίηση

Για να μειώσουμε το συνολικό πλήθος ουρών ψάξαμε στο kernel/proc.h σύμφωνα με την εκφώνηση αλλά δεν βρήκαμε κάτι. Επειτα ψάξαμε στο minix για το USER_Q και βρήκαμε το define του στο αρχείο include/minix/config.h. Κάναμε τις αλλαγές που χρειάζονται για να βρίσκονται όλες οι διεργασίες χρήστη σε μια μόνο ουρά.

Για να πάρει ο πυρήνας το fss_priority πρεπει να του το στείλει ο sched. Στην do_noquantum() υπάρχει η schedule_process_local(), variation της schedule_process, οπότε ψάξαμε για την schedule_process(). Εκεί στο σημείο που καλείται η sys_schedule() προσθέσαμε ως επιπλέον όρισμα το fss_priority. Το ίδιο κάναμε και στο PROTOTYPE της schedule_process() στο αρχείο include/minix/syslib.h, όπως και στο lib/libsys/sys_schedule.c . Ψάξαμε για ένα απο τα πεδία αυτής και βρήκαμε οτι ο kernel το λαμβάνει στο kernel/system/do_schedule.c. Εκεί προσθέσαμε στο struct p να παίρνει το fss_priority, το ίδιο κάναμε και στο kernel/proc.h όπου ορίζεται το struct p.