Τμήμα Μηχ. Η/Υ & Πληροφορικής Εαρινό 2019
Πανεπιστήμιο Ιωαννίνων

ΜΥΥ601 Λειτουργικά Συστήματα

ΟΜΑΔΑ
1)Χαρίσης Γκέκας
2)Δημήτριος Σαμουρέλης

Εργαστήριο 2: Υλοποίηση δίκαιης χρονοδρομολόγησης στο MINIX 3.2.0

ΕΡΩΤΗΜΑ 1

Μας δόθηκε ο πηγαίος κώδικας του λειτουργικού συστήματος MINIX 3.2.0.Αυτό που πρέπει εμείς να κάνουμε είναι να εισάγουμε την δίκαιη χρονοδρομολόγηση μεταξύ διεργασιών και κάποιες άλλες τροποποιήσεις που μας ζητούνται στα επόμενα βήματα.
Καταρχήν,αυτό που μας ζητείται στο πρώτο ερώτημα είναι να βρούμε τρόπο να μεταφέρουμε την ταυτότητα της πρώτης διεργασίας της ομάδας από την διεργασία pm στο sched.Στο path servers/pm/mproc.h υπάρχει σαν πεδίο το pid_t mp_procgrp(οδηγός ομάδας).'Οπως λέει και στην εκφώνηση ένας τρόπος για να συμπεριλάβουμε το πεδίο rmp→mp_procgrpr στο μύνημα που στέλνεται από τον pm στο sched. Αφού το κάναμε αυτό πάμε και βρίσκουμε που αλλού υπάρχει η συνάρτηση που πειράξαμε για να την τροποποιήσουμε.Οπότε εκτελώ μια αναζήτηση της μορφής search sched_start_user()/usr/src |more και search sched_inherit()/usr/src|more .Μπαίνουμε include/minix/sched.h όπου μέσα υπάρχει το πρωτότυπο sched_inherit() και προσθέτω το όρισμα int mp_procgrp για να είναι σωστό.Έπειτα μπαίνω και στην sched_start.c με path lib/libsys/sched_start.c όπου υπάρχει η συνάρτηση sched inherit η οποία είναι υπεύθυνη για την αποστολή μυνημάτων στον sched.Περνάω το Int procgrp σαν όρισμα και στην συνάρτηση καιθα περάσω στο μύνημα m αυτο που θέλω να στείλω δηλαδή το procgrp μέσω του μυνηματος m9.l2 που ψάξαμε με μια αναζήτηση search "τύπος μυνηματος"/usr/src|more .(Επίσης στο path usr/src/include/minix/com.h πάνω πάνω είδαμε οτι τα μυνήματα m9_l1 m9_l3 και λοιπά αναφέρονται σε πεδία SCHEDULING_SCHEDULER,SCHEDULING_PARENT και λοιπά οπότε επιλέγω το m9_l2).Γράφω την εντολή m.m9_l2=(int) procgrp κάνω downcasting .Εδώ είχαμε και ένα printf το που τυπώνει το μυνημα (και επιστρέφει 151) για να δούμε εάν έγινε σωστά η φόρτωση τα οποία τα βάλαμε σε σχόλιο πιο μετα .
Και τέλος η taskcall είναι αυτή που καλείται για να στείλει το μύνημα m.

ΕΡΩΤΗΜΑ 2

Τροποποιήσαμε τη δομή schedproc στο schedproc.h(servers/sched/schedproc.h) για να συμπεριλάμβανει τα πεδία procgrp, proc_usage, grp_usage και fss_priority . Έχω τον οδηγό ομάδας στο sched.*Στο servers/sched/ στην schedule.c στη do_start_scheduling αρχικοποιούμε τα πεδία στο μηδέν χρησιμοποιώντας το δείκτη rmp στο struct schedproc και εκεί που δέχομαι στο rmp→procgrp =m_ptr→m9_l2 βάζουμε ένα printf για να δούμε εαν στάλθηκε σωστά που όντως συνέβει αφού επέστρεψε 151 και στην συνέχεια το βάλαμε σε σχόλιο. Στο schedproc.h έχω το πίνακα μου.Η do_noquantum καλείται κάθε φορά που το quantum μιας διεργασίας τελειώνει.Όπως είναι,την σπρώχνει προσ την επόμενης χαμηλότερης προτεραιότητας ουρά εκτέλεσης και ζητά απο την schedule_process() να την χρονοπρογραμματίσει στην νέα ουρά.Άυτή με την σειρά της καλεί την sys_schedule() για να το επιτύχει.Επίσης εκεί ενημερώνουμε τις τιμές οτάν λήγει το κβάντο,το μονοπατι της είναι servers/sched/schedule.c.Πιο συγκεκριμένα όταν τελικά λήξει το κβάντο καλείται η do_noquantum η οποία θα κάνει την ενημέρωση των τίμων.Καλούμε την update(rmp) οπου rmp είναι δείκτης στο struct schedproc.Ο Ρόλος της update είναι να διατρέχει κάθε διεργασία (for i=0;i<NR_PROCS;i++)*
*και έχουμε βάλει την if(rmp->flags & IN_USE) για όσο μια διεργασία δεν χρησιμοποιείται. Την χρήση της (rmp->flags & IN_USE) την παρατηρούμε και στην balance_queues η οποία ενημερώνεται όταν έχει λήξει το κβάντο μιας διεργασίας.Τώρα στην δομή schedproc(schedproc[i].grp_usage ,schedproc[i].proc_usage) προσθέτω το USER_QUANTUM κάθε φορά που λήγει μια διεργασία.Έχουμε δύο μεταβλητές τις int first_grp_usage και int first_proc_grp που τους δίνω τις τιμές των schedproc[i].grp_usage ,schedproc[i].procgrp αντίστοιχα.Εάν το schedproc[i].procgrp είναι ίσο με το first_proc_grp δηλαδη εάν ανηκουνστην ίδια ομάδα τότε κάνω ενημέρωση στο πίνακα μου και δίνω σε όσες ανήκουν στην ίδια ομάδα το ίδιο grp_usage .Με αυτόν τον τρόπο τα βάζω στην ίδια ομάδα.Εδώ καλούμε την find_number_of_teams η οποία βρίσκει τον συνολικό αριθμό των ομάδων.Έχουμε δύο μεταβλητές I,j που λειτουργούν σαν δείκτες .Κάνω αναζήτηση σε όλες τις διεργασίες μαζί με τον έλεγχο μου rmp→flag &IN_USE και κάνουμε ενα βγόγχο for όπου διατρέχω το j για j<i .Εαν βρώ το ίδιο procgrp κάνω break απο τον βρόγχο γτ δεν θελώ να μετράω τις ίδιες ομάδες.Εάν το i==j τότε σημαίνει οτι βρήκα διαφορετική ομάδα οπότε μπορώ να αυξησω την μεταβλητή numbers .Το break μου μηδενίζει το j καθε φορα που βρίσκω ιδια ομάδα ωστε να μην μετραω τα διπλοτυπα.Έπειτα επιστρεφώ τον αριθμό των ομάδων.(Εδώ βάλαμε ενα printf με τον αριθμό των ομάδων το οποίο την πρώτη φορά ηταν ίσο με 7.Τώρα είμαστε έτοιμοι να βρούμε τον αποτέλεσμα των τύπων που δόθηκαν για την ενημέρωση των διεργασιών.Δηλαδή scheproc[i].proc_usage=schedproc[i].proc_usage/2 ,,,schedproc[i].grp_usage=schedproc[i].grp_usage/2,,,schedproc[i].proc_usage/2 +schedproc[i].grp_usage*number_of_teams/4 +base (int base =0).*

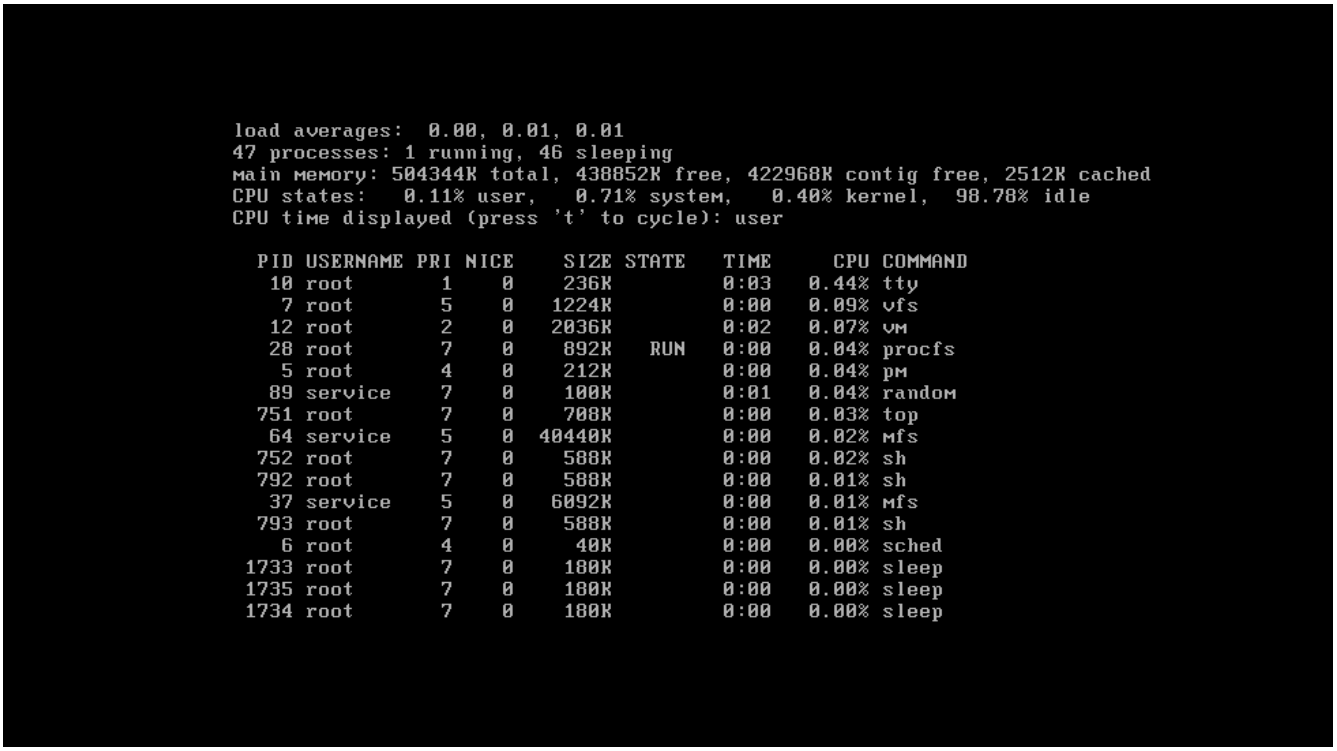
Ερώτημα 3

Στο τρίτο ερώτημα ζητείται να τροποποιήσουμε το πλήθος των ουρών έτσι ωστε όλες οι διεργασίες να βρίσκονται να βρίσκονται στην ουρά χρήστη .Συγκεκριμένα πάμε στο Path include/minix/config.h όπου βρίσκονται αυτες οι μεταβλητές και τις κάνω NR_SCHED_QUEUES 8(Αφού θέλουμε 1 επίπεδο χρήστη αφαιρώ τα υπόλοιπα,,,MAX_USER_Q =7(Δηλαδη για επίπεδο χρήστη που βρίσκεται στο έβδομο επίπεδο) USER_Q=7,,,,MIN_USER_Q=7.Τώρα όλες οι διεργασίες θα πηγαίνουν στο 7 επίπεδο δηλαδη στο επίπεδο χρήστη.Το επόμενο πράγμα που πρέπει να κάνουμε είναι να στείλουμε το fss_priority απο τον sched στον kernel.Βρίσκουμε την διαδρομή που στέλνονται μυνήματα απο τον sched στον kernel .Στο path servers/sched υπάρχει η sys_schedule(μεσα

στην schedule_process) που ειναι για την αποστολή μυνημάτων απο τον sched στον kernel .Αρχικά περνάω το rmp→fss_priority για να το στείλω και η επομένη μας σκεψη ειναι να φτιάξουμε σωστα την sys_schedule.Πάμε στο path lib/libsys/sys_schedule.c όπου έχουμε την υλοποίηση της sys_schedule βάζουε σαν όρισμα το double fss_priority και στέλνουμε με το μύνημα m.m9_l5=fss_priority και η kernelcall με την σειρά της πέρνει σαν όρισμα το μύνημα m στο path lib/libsys/kernel_call.c.Από την σκοπιά του πυρήνα πρέπει να δεχθούμε το μύνημα.Καλείται η map(SYS_SCHEDULE,do_schedule) στο path kernel/system.c η οποία οδηγεί στο kernel/system/do_schedule.c όπου θα αποθηκέυσουμε αυτό που στείλαμε απο το μύνημα στο πυρηνα με την εντολή p→fss_priority=m_ptr→m9_l5.Ο p δείχνει στο struct του kernel οπότε του αποθηκεύω εκεί το fss_priority που στείλαμε.Και τελικά στην kernel/system.c στην schedproc που πέρνει το p τελειώνει αυτη η διαδικασία.Βεβαια ολα αυτά προυποθέτουν να βάλουμε στο struct του kernel το fss_priority στο path kernel/proc.h.Το τελευταίο πράγμα που έχουμε να κανουμε είναι να φτιάξουμε τον kernel ώστε να δεχεται σωστα τα fss_priority.Πάμε στο αρχείο kernel/proc.c στην συνάρτηση pick_proc της οποίας ο ρόλος είναι να βάζει στο head την επόμενη διεργασία για εκτέλεση σύμφωνα με το μικρότερο fss_priority.Δηλώσαμε ενα min_fss=rp→fss_priority εκτελούμε ενα βρόγχο for μεχρί NR_PROCS για να ψάξει όλες τις διεργασίες ,κάνουμε έλεγχω για q=USER_Q ότι είμαστε στο επίπεδο χρήστη.Ελέγχουμε το min_fss >fss_priority και εαν είναι βάζουμε στο min_fss το rp→fss_priority ,όταν βγούμε απο την if αυξάνω τον δείκτη κατά 1 για να πάρει την επόμενη διεργασία και τελικα αφου τελειώσει η for ξαναβάζω τον δείκτη στην θέση του για να ξαναχρησιμοποιήθει απο το σύστημα.Τώρα η ουρά ειναι ταξινομημένη σύμφωνα με το μικροτερο fss_priority .

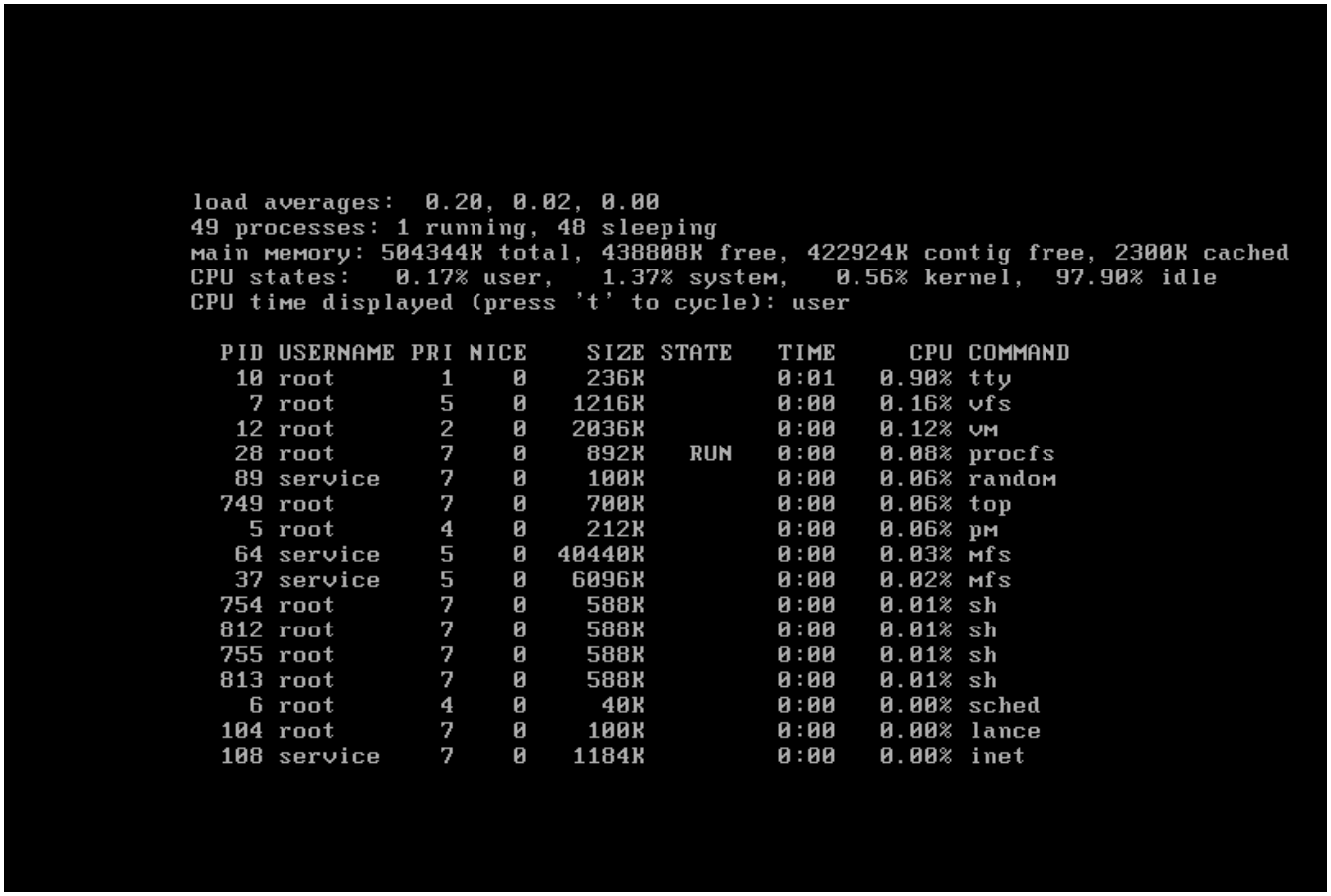
ΕΡΩΤΗΜΑ 4

*Στο path usr/src/bin περάσαμε το script που βρίσκεται στην εκφώνηση με όνομα script2.sh και τρέχουμε σε ένα τερματικό την εντολή top για να δούμε τη χρήση που γίνεται στον επεξέργαστη από κάθε ομάδα και επομένως και κάθε διεργασία της κάθε ομάδας.*Αρχή της δίκαιης χρονοδρομολόγησης είναι να διαιρέσει τις διεργασίες χρήστη σε ομάδες και να ισομοιράσει το χρόνο επεξεργαστή μεταξύ των ομάδων. Επιπλέον ισομοιράζει το χρόνο επεξεργαστή της κάθε ομάδας μεταξύ των διεργασιών της ομάδας. Για παράδειγμα, θεωρούμε δύο ομάδες διεργασιών A και B. Θεωρούμε ότι η A περιέχει μόνο μία διεργασία A1, ενώ η B περιέχει δύο διεργασίες B1 και B2. Τότε ο δίκαιος χρονοδρομολογητής αναθέτει από 50% του χρόνου επεξεργαστή σε κάθε ομάδα. Επομένως η A1 λαμβάνει 50% του συνολικού χρόνου και οι B1, B2 λαμβάνουν από 25% η καθεμία.



Τρέχουμε σε ένα τερματικό ./script2.sh& και σε ένα άλλο ./script2.sh & ./script2.sh & δημιουργώντας έτσι δύο ομάδες όπως στο παράδειγμα που δίνεται στην εκφώνηση.
Πράγματι παρατηρούμε οτί η διεργασία που τρέχει στο ένα τερματικό(πρώτη ομάδα) με pid = 752 χρησιμοποίει το 0.02% του επεξεργαστή . Οι άλλες δυο διεργασίες με pid = 792 , pid = 793 τρέχουν στο ίδιο τερματικό (ανήκουν στην ίδια ομάδα) χρήσιμοποίουν αθροιστικά το 0.02% το επεξεργαστή και από 0.01% η κάθεμία . Άρα πραγματοποιείται σώστα η δίκαιη χρονοδρομολόγηση .

Δημιουργούμε επίσης σε ένα τερματικό δύο διεργασίες της ίδιας ομάδας και σε ένα άλλο δύο διεργασίας άλλης ομάδας . Παρατηρούμε για τους ίδιους λόγους ότι πραγματοποιείται σωστά η δίκαιη χρονοδρομολόγηση



Τέλος Δημιουργούμε σε ένα τερματικό δύο διεργασίες της ίδιας ομάδας και σε ένα άλλο τέσσερις διεργασίας άλλης ομάδας . Παρατηρούμε για τους ίδιους λόγους ότι πραγματοποιείται σωστά η δίκαιη χρονοδρομολόγηση .