Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Λειτουργικά Συστήματα (ΗΥ321)

Ακαδημαϊκό Έτος 2017-2018

2η Εργαστηριακή Άσκηση

Τελευταία Ενημέρωση: 8 Μαρτίου 2018

Περιεχόμενα

1 Εισαγωγικά

Η απόδοση του λειτουργικού συστήματος εξαρτάται σε πολύ μεγάλο βαθμό από την υλοποίηση του χρονοδρομολογητή διεργασιών, ο οποίος αποτελεί και ένα από τα βασικότερα τμήματα του λειτουργικού. Ο χρονοδρομολογητής είναι υπεύθυνος για να αποφασίζει πότε η κάθε διεργασία θα λάβει τον επεξεργαστή ώστε να εκτελεστεί και είναι το τμήμα που υλοποιεί την ”αφαίρεση” του multitasking σε ένα λειτουργικό σύστημα. Αυτές οι λειτουργίες είναι ιδιαιτέρως σημαντικές προκειμένου να διασφαλιστεί η δίκαιη χρήση των πόρων του συστήματος από όλες τις διεργασίες, η καλή αλληλεπίδραση με τον χρήστη και η έγκαιρη εκτέλεση κρίσιμων για το σύστημα διεργασιών.

Σε αυτή την εργασία καλείστε να υλοποιήσετε ένα χρονοδρομολογητή διεργασιών που βασίζεται στην πολιτική Shortest Job First. Ο κώδικας που υλοποιεί τον χρονοδρομολογητή του Linux είναι αρκετά περίπλοκος καθώς πρέπει να ελέγχει διεργασίες διαφόρων τύπων, πολλαπλούς επεξεργαστές και είναι εν μέρει υπεύθυνος για την απαραίτητη διαχείριση μνήμης. Γι’ αυτό το λόγο, για τους σκοπούς της παρούσας εργασίας, σας παρέχεται μία εικονική μηχανή που προσομοιώνει ένα μονο-επεξεργαστικό σύστημα. Η εικονική μηχανή δεν απαιτεί εκτενή διαχείριση μνήμης και διακοπών, ενώ παράλληλα παρέχει όλη την διεπαφή που χρησιμοποιείται από τον χρονοδρομολογητή του Linux και εκτελεί τις διάφορες συναρτήσεις όπως ακριβώς και ο πυρήνας του Linux. Με αυτό τον τρόπο είναι δυνατό να υλοποιήσετε τον αλγόριθμο χρονοδρομολόγησης στο επίπεδο χρήστη, επιτρέποντάς σας να εμβαθύνετε στις λεπτομέρειες του αλγορίθμου χρονοδρομολόγησης και έχοντας στην διάθεσή σας επιπλέον εργαλεία αποσφαλμάτωσης.

1.1 Υλικό Μελέτης

Πριν ξεκινήσετε την υλοποίηση θα πρέπει να έχετε διαβάσει τις οδηγίες που σας έχουν δοθεί και αφορούν στην εικονική μηχανή που υλοποιεί τον χρονοδρομολογητή του Linux και να έχετε εξοικειωθεί με τα αρχεία *schedule.c* και *schedule.h*. Σε αυτά **και μόνο** τα αρχεία θα εκτελέσετε τις όποιες τροποποιήσεις για την υλοποίηση του χρονοδρομολογητή σας. Γι’ αυτό το σκοπό, έχουν τεκμηριωθεί σε μεγάλο βαθμό. Επίσης, θα πρέπει να έχετε μελετήσει το κεφάλαιο 4 από το βιβλίο ”Linux Kernel Development” όπως επίσης και το αρχείο *sched.c*, το οποίο βρίσκεται στον πηγαίο κώδικα του πυρήνα και υλοποιεί τον χρονοδρομολογητή διεργασιών του Linux[1].

[1]Στην ιστοσελίδα http://lxr.linux.no/linux+v3.2.39/ υπάρχει ο πηγαίος κώδικας του πυρήνα του Linux στο σύνολό του, με δυνατότητες παραπομπής μεταξύ αρχείων.

1.2 Προαπαιτούμενα

Σε αυτή την άσκηση θα υλοποιήσετε τον αλγόριθμο χρονοδρομολόγησης *Shortest Job First*. Σε αντίθεση με την πρώτη άσκηση, δεν θα εργαστείτε απευθείας στον πηγαίο κώδικα του Linux. Αντί αυτού, σας έχει δοθεί μία εικονική μηχανή η οποία παρέχει το σύνολο των συναρτήσεων που χρησιμοποιούνται από τον πυρήνα του Linux για την χρονοδρομολόγηση διεργασιών. Ο κώδικας για την εικονική μηχανή βρίσκεται στο αρχείο *Scheduler_VM.zip*, το οποίο είναι διαθέσιμο στην σελίδα του μαθήματος (έγγραφα -> εργασίες). Κατεβάστε το αρχείο *Scheduler_VM.tar.gz* και αποσυμπιέστε το στο home directory σας.

1.3 Προθεσμία και Τρόπος Παράδοσης

Η άσκηση θα πρέπει να παραδοθεί έως τα **μεσάνυχτα της Τετάρτης 21/3/2018**. Η προθεσμία είναι τελική και δεν πρόκειται να δοθεί καμία παράταση – συνολικά ή ατομικά – για κανένα λόγο. Ο χρόνος που σας δίνεται υπερεπαρκεί. Χρησιμοποιήστε τον ”σοφά”.

Η παράδοση θα γίνει στο e-class.

Στο συνοδευτικό κείμενο θα πρέπει να υπάρχουν τα ονοματεπώνυμα, τα Α.Ε.Μ. και τα e-mail των μελών της ομάδας. Επίσης μπορείτε να συμπεριλάβετε και οτιδήποτε άλλο μήνυμα θέλετε να μας μεταφέρετε σχετικά με την άσκησή σας.

Τέλος, η εργασία σας θα πρέπει να συμπεριληφθεί ως ένα συνημμένο αρχείο. Για τις λεπτομέρειες δημιουργίας του αρχείου δείτε την παράγραφο 4.

1.4 Κώδικας Ηθικής

Κάθε ομάδα θα πρέπει να εργαστεί ανεξάρτητα. Είναι δεκτό και θεμιτό διαφορετικές ομάδες να ανταλλάξουν απόψεις σε επίπεδο γενικής ιδέας ή αλγόριθμου. Απαγορεύεται όμως κάθε συζήτηση / ανταλλαγή κώδικα ή ψευδοκώδικα. Σημειώστε ότι οι ασκήσεις θα ελεγχθούν τόσο από αυτόματο σύστημα όσο και από εμάς για ομοιότητες μεταξύ των ομάδων αλλά και για ομοιότητες με τυχόν λύσεις που μπορεί να βρεθούν στο διαδίκτυο ή λύσεις που έχουν δοθεί σε προηγούμενα έτη. Σε περίπτωση αντιγραφής οι ασκήσεις (όλων των φάσεων) όλων των εμπλεκόμενων φετινών ομάδων μηδενίζονται χωρίς καμία περαιτέρω συζήτηση.

Όλα τα μέλη στο εσωτερικό κάθε ομάδας θα πρέπει να έχουν ισότιμη συμμετοχή στην ανάπτυξη κάθε φάσης. Διατηρούμε το δικαίωμα να επιβεβαιώσουμε αν αυτό τηρείται με προσωπικές συνεντεύξεις / προφορική εξέταση.

2 Ζητούμενα

2.1 Υλοποίηση Παραλλαγής του Αλγορίθμου Χρονοδρομολόγησης Shortest Job First (SJF)

Στα πλαίσια αυτής της εργασίας θα πρέπει να υλοποιήσετε τον προεκτοπιστικό αλγόριθμο χρονοδρομολόγησης *Shortest Job First*. Η πολιτική σας θα πρέπει να λαμβάνει υπόψη και τον χρόνο που έχει μεσολαβήσει από την τελευταία εκτέλεση κάθε διεργασίας, ευνοώντας τις διεργασίες που δεν έχουν εκτελεστεί για μεγάλο χρονικό διάστημα, ενώ ήταν στην ready queue.

Υπενθυμίζεται ότι ο αλγόριθμος SJF επιλέγει πάντα προς εκτέλεση, από τις έτοιμες (ready) διεργασίες, εκείνη με τον μικρότερο εναπομείναντα χρόνο εκτέλεσης. Η προεκτοπιστική έκδοση του SJF επιτρέπει τη διακοπή μιας εκτελούμενης διεργασίας, εφόσον εμφανιστεί μία με μικρότερο εναπομείναντα χρόνο εκτέλεσης.

Η βασική αδυναμία του αλγορίθμου SJF, που καθιστά προβληματική τη χρήση του, είναι ότι δεν είναι ρεαλιστική η απαίτηση της εκ των προτέρων γνώσης του εναπομείναντα χρόνου εκτέλεσης κάποιας διεργασίας. Μία μέθοδος παράκαμψης αυτής της αδυναμίας είναι η ιστορική παρακολούθηση της συμπεριφοράς των εφαρμογών. Πιο συγκεκριμένα, το λειτουργικό καταγράφει τα προηγούμενα "ξεσπάσματα" (bursts) κάθε εφαρμογής. Ως ξέσπασμα ορίζουμε το χρόνο από τη στιγμή που μια διεργασία λαμβάνει τον έλεγχο του επεξεργαστή, έως τη στιγμή που θα αφήσει (για οποιονδήποτε λόγο, οικειοθελώς ή μη) τον επεξεργαστή. Με βάση αυτή την πληροφορία είναι δυνατόν ο χρονοδρομολογητής να εκτιμήσει τον αναμενόμενο μελλοντικό χρόνο ξεσπάσματος κάθε διεργασίας, εφαρμόζοντας έναν απλό μαθηματικό τύπο που φροντίζει για τη "γήρανση" των παλαιότερων παρατηρήσεων (και άρα για τη μείωση της επίδρασής τους στον αναμενόμενο μελλοντικό χρόνο ξεσπάσματος). Ο τύπος που θα χρησιμοποιηθεί στα πλαίσια της εργασίας είναι:

$$Exp\_Burst_i = \frac{Burst_{i-1} + \alpha * Exp\_Burst_{i-1}}{1 + \alpha}$$

όπου Exp_Burst_i είναι ο εκτιμώμενος μελλοντικός χρόνος ξεσπάσματος κατά την i-οστή εκτέλεση της διεργασίας, $Burst_{i-1}$ ο χρόνος ξεσπάσματος που μετρήθηκε κατά την (i-1)-στή εκτέλεση της διεργασίας, Exp_Burst_{i-1} ο χρόνος ξεσπάσματος που είχε προβλεφθεί για την (i-1)-στή εκτέλεση της διεργασίας και α ένας συντελεστής γήρανσης. Για τους σκοπούς της εργασίας μπορείτε να θεωρήσετε ότι $\alpha = 0.5$.

Με βάση τον εκτιμώμενο μελλοντικό χρόνο ξεσπάσματος κάποιας διεργασίας και το χρόνο κατά τον οποίο η διεργασία είναι ανενεργή (από την τελευταία φορά που μπήκε στη ready queue), μπο-

ρούμε να υπολογίσουμε μία “προτεραιότητα” για τη διεργασία ως εξής:

$$Goodness(k)_i = \frac{1 + Exp\_Burst(k)_i}{\min_{m=0}^{N}(1 + Exp\_Burst(m)_i)} * \frac{\max_{m=0}^{N}(1 + WaitingInRQ(m)_i)}{1 + WaitingInRQ(k)_i}$$

όπου $Goodness(k)_i$ είναι η “προτεραιότητα” της διεργασίας k τη χρονική στιγμή i, $WaitingInRQ(k)_i$ είναι ο χρόνος που έχει περάσει η διεργασία k από τη στιγμή που μπήκε στην ουρά των έτοιμων διεργασιών, έως τη χρονική στιγμή i και N ο αριθμός των διεργασιών στη ready queue. Προφανώς προτιμάται η διεργασία με το χαμηλότερο Goodness.

2.2 Υλοποίηση Προγραμμάτων Ελέγχου - Πειραματική Αξιολόγηση

Αφού υλοποιήσετε τον αλγόριθμο χρονοδρομολόγησης, θα πρέπει να ελέγξετε τη λειτουργία του χρησιμοποιώντας κατάλληλα υπολογιστικά φορτία (εφαρμογές). Πιο συγκεκριμένα, θα πρέπει να χρησιμοποιήσετε το interface που σας παρέχει η virtual machine και να σχεδιάσετε κάποια profiles εκτέλεσης που να περιλαμβάνουν:

- Μόνο προγράμματα που πραγματοποιούν υπολογισμούς (χωρίς, ή με όσο το δυνατό λιγότερο E/E, σφάλματα σελίδων κλπ) - non-interactive.
- Μόνο προγράμματα που πραγματοποιούν E/E (με όσο το δυνατό λιγότερο υπολογισμό) - interactive.
- Τόσο non-interactive όσο και interactive προγράμματα.

Επίσης, θα πρέπει να σχεδιάσετε profiles τα οποία θα δείχνουν τις αδυναμίες του SJF στην περίπτωση που δε λαμβάνει υπόψη του το χρόνο παραμονής μιας διεργασίας στη ready queue και να αποδεικνύουν (πειραματικά) τις βελτιώσεις που φέρνει ο συνυπολογισμός της εν λόγω πληροφορίας.

Τα προγράμματα θα πρέπει να έχουν περίπου τον ίδιο χρόνο εκτέλεσης (μη αμελητέο - κάποιων δευτερολέπτων). Θα χρησιμοποιήσετε αυτά τα profiles προκειμένου να αξιολογήσετε τον αλγόριθμο SJF. Επίσης, θα πρέπει να αξιολογήσετε την επίδοση του standard αλγορίθμου SJF (αφαιρώντας από τη σχέση υπολογισμού του Goodness το κλάσμα που αφορά το χρόνο WaitingInRQ). Θα πρέπει να καταγράψετε και να μελετήσετε τον τρόπο εκτέλεσης των εφαρμογών σε κάθε περίπτωση και να ερμηνεύσετε τα αποτελέσματα. Λάβετε υπόψη σας ότι η ποιότητα της πειραματικής αξιολόγησης είναι ένας από τους **βασικότερους** παράγοντες που βαθμολογούνται σε αυτή την άσκηση.

3 Λεπτομέρειες Υλοποίησης

3.1 Χρονοδρομολόγηση στην Εικονική Μηχανή

Όπως αναφέρθηκε και παραπάνω, η εικονική μηχανή που σας παρέχεται για την υλοποίηση της παρούσας εργασίας αποτελεί μία απλοποιημένη έκδοση του χρονοδρομολογητή του Linux. Παρέχει ως διεπαφή το βασικό σύνολο των συναρτήσεων που χρησιμοποιεί και ο πραγματικός δρομολογητής διεργασιών του Linux προκειμένου να μπορεί να υλοποιηθεί και να ελεγχθεί οποιοσδήποτε αλγόριθμος δρομολόγησης.

Ο πηγαίος κώδικας της εικονικής μηχανής που σας δίνεται υλοποιεί μία απλοποιημένη παραλλαγή της πολιτικής δρομολόγησης *Round-Robin*. Εσείς καλείστε να μελετήσετε και να τροποποιήσετε κατάλληλα αυτή την υλοποίηση προκειμένου να υλοποίησετε το ζητούμενο αλγόριθμο δρομολόγησης.

Όπως αναφέρθηκε και στην πρώτη εργασία, κάθε διεργασία του συστήματος αναπαρίσταται από μία δομή *struct task_struct*. Η δομή αυτή περιέχει μεταξύ άλλων, την κλάση δρομολόγησης, την προτεραιότητα, όπως επίσης και την τιμή του τρέχοντος timeslice της διεργασίας. Σε αυτή την δομή θα προσθέσετε τα απαραίτητα πεδία για την τήρηση των λογιστικών στοιχείων δρομολόγησης της διεργασίας.

Όσον αφορά στις συναρτήσεις που θα πρέπει να υλοποιήσετε (ή να μετατρέψετε) αυτές είναι οι ακόλουθες:

- initschedule
- killschedule
- schedule
- activate_task
- deactivate_task
- scheduler_tick
- sched_fork
- wake_up_new_task

Η βασική συνάρτηση που είναι υπεύθυνη για την δρομολόγηση των διεργασιών είναι η συνάρτηση *schedule()*. Αυτή η συνάρτηση καλείται είτε απευθείας, όταν μία διεργασία απελευθερώνει τον επεξεργαστή, είτε μέσω της συνάρτησης *scheduler_tick()*. Η συνάρτηση *scheduler_tick()* καλείται από τον πυρήνα σε περιοδικά χρονικά διαστήματα για την ανανέωση της μεταβλητής time_slice της τρέχουσας διεργασίας. Ο δρομολογητής έχει πρόσβαση στην τρέχουσα διεργασία μέσω της global μεταβλητής **current**. Σε περίπτωση που κριθεί απαραίτητη η εκ νέου δρομολόγηση της διεργασίας (π.χ. η διεργασία έχει εξαντλήσει το timeslice που της έχει ανατεθεί), καλείται έμμεσα η συνάρτηση *schedule()*.

Η ουρά εκτέλεσης των διεργασιών περιγράφεται από μία δομή τύπου runqueue. Στον κώδικα που σας δίνεται, η δομή αυτή αποθηκεύει τις έτοιμες προς εκτέλεση διεργασίες σε μία απλά συνδεδεμένη λίστα. Μπορείτε όμως να υλοποιήσετε οποιαδήποτε δομή δεδομένων επιθυμείτε εσείς. Επίσης, στη δομή αποθηκεύεται και ο συνολικός αριθμός των διεργασιών που είναι αποθηκευμένες στην ουρά (μεταβλητή *nr_running*). Η συνάρτηση *schedule* είναι υπεύθυνη για να διατρέξει την λίστα των ενεργών διεργασιών, να επιλέξει (βάσει του αλγορίθμου δρομολόγησης) την επόμενη διεργασία προς εκτέλεση και να εκτελέσει το απαραίτητο context switch καλώντας την συνάρτηση *context_switch()*.

Πλέον της λειτουργικότητας που περιγράφηκε παραπάνω, η εικονική μηχανή προσομοιώνει και τις λειτουργίες Εισόδου/Εξόδου των interactive διεργασιών. Όταν μία interactive διεργασία χρειάζεται να περιμένει για την ολοκλήρωση μίας λειτουργίας Εισόδου/Εξόδου, εισάγεται στην κατάλληλη **WaitQueue** και καλείται η συνάρτηση *deactivate_task()* για την αφαίρεση της διεργασίας από την λίστα των έτοιμων προς εκτέλεση διεργασιών. Όταν μία διεργασία ολοκληρώσει μία λειτουργία Εισόδου/Εξόδου και είναι έτοιμη προς εκτέλεση, καλείται η συνάρτηση *activate_task()* για την προσθήκη της διεργασίας στην ουρά εκτέλεσης του δρομολογητή. Στην συνέχεια, καλείται η συνάρτηση *schedule()* ώστε να πραγματοποιηθεί η εκ νέου χρονοδρομολόγηση των διεργασιών.

Οι δύο τελευταίες συναρτήσεις που θα πρέπει να υλοποιήσετε (ή να μετατρέψετε) είναι οι συναρτήσεις *sched_fork()* και *wake_up_new_task()*. Η πρώτη συνάρτηση είναι αυτή που καλείται κατά την δημιουργία μίας νέας διεργασίας (μέσω της συνάρτησης fork()) και είναι υπεύθυνη για να θέσει το timeslice της διεργασίας. Η δεύτερη συνάρτηση καλείται μετά τη συνάρτηση *sched_fork()* για να εισάγει την διεργασία στην runqueue.

3.2 Συνιστώμενη Πορεία Υλοποίησης

Η υλοποίησή σας θα πρέπει να ακολουθήσει τα εξής βήματα:

- Υλοποιήστε τις συναρτήσεις *initschedule() killschedule()*, οι οποίες εκτελούνται κατά την αρχικοποίηση και τον τερματισμό του δρομολογητή αντίστοιχα.

- Προσθέστε τα απαιτούμενα πεδία στη δομή κάθε διεργασίας ώστε να διατηρείται ο χρόνος του τελευταίου ξεσπάσματος, η τρέχουσα πρόβλεψη για το επόμενο ξέσπασμα και ο χρόνος τελευταίας εκτέλεσης της διεργασίας ή τελευταίας εισόδου της στη runqueue (το μεταγενέστερο από τα δύο). Φροντίστε για τη σωστή αρχικοποίηση των πεδίων αυτών (σε 0) για κάθε νέα διεργασία που δημιουργείται. Η αρχικοποίηση μπορεί να πραγματοποιηθεί εντός της συνάρτησης *sched_fork()*.

- Τήρηση "λογιστικών" σχετικά με τον τελευταίο χρόνο ξεσπάσματος κάθε διεργασίας. Για το σκοπό αυτό αρκεί να διατηρείτε τη χρονική στιγμή κατά την οποία δόθηκε ο επεξεργαστής σε μια διεργασία, και να αφαιρείτε αυτή από την τρέχουσα χρονική στιγμή, όταν η διεργασία χάσει τον επεξεργαστή (και μόνο τότε). Η συνάρτηση που επιστρέφει την τρέχουσα χρονική στιγμή είναι η *sched_clock()*. Επίσης θα πρέπει κάθε φορά που έχετε μια νέα τιμή τελευταίου ξεσπάσματος να υπολογίζετε τον εκτιμώμενο χρόνο επόμενου ξεσπάσματος.

- Τήρηση "λογιστικών" σχετικά με τον τελευταίο χρόνο εκτέλεσης μιας διεργασίας (δηλαδή το πότε έχασε τον επεξεργαστή της για τελευταία φορά) ή το πότε μπήκε για τελευταία φορά στη runqueue. Διατηρείτε το μεταγενέστερο από τα δύο!

- Προσθέστε κώδικα που εντοπίζει το minimum Exp_Burst και το maximum WaitingInRQ μεταξύ των διεργασιών της runqueue.

- Στο νέο χρονοδρομολογητή έχουμε εισάγει ένα μικρό σφάλμα. Κάθε φορά που εκτελούμε το χρονοδρομολογητή, υπάρχουν νεότερα πραγματικά στοιχεία για το τελευταίο ξέσπασμα της διεργασίας που εκτελείται, από αυτά που είχαν χρησιμοποιηθεί για την προηγούμενη εκτίμηση του αναμενόμενου ξεσπάσματός του. Παρόλα αυτά, δεν είναι δυνατό να παγιώσουμε αυτά τα στοιχεία και να υπολογίσουμε ένα νέο αναμενόμενο ξέσπασμα, καθώς δεν είμαστε βέβαιοι αν η εκτελούμενη διεργασία θα απολέσει τον επεξεργαστή σε αυτό το γύρο δρομολόγησης ή αν θα συνεχίσει την εκτέλεσή της. Για το λόγο αυτό, ειδικά για τη διεργασία που κατείχε τον επεξεργαστή τη στιγμή που κλήθηκε ο χρονοδρομολογητής, υπολογίζουμε μια προσωρινή τιμή εκτιμώμενου επόμενου ξεσπάσματος – βασισμένοι στην προσωρινή τιμή τελευταίου πραγματικού ξεσπάσματος – και χρησιμοποιούμε την τιμή αυτή κατά τη διάρκεια της δρομολόγησης. Σημειώστε ότι η διεργασία που κατείχε τον επεξεργαστή τη στιγμή που

ενεργοποιήθηκε ο χρονοδρομολογητής “δείχνεται” από την καθολική μεταβλητή του kernel με όνομα current.

- Μεταβάλετε την πολιτική χρονοδρομολόγησης ώστε πλέον κριτήριο για την επιλογή μιας διεργασίας να είναι το Goodness και όχι το Exp_Burst. Με άλλα λόγια, συμπεριλάβετε το χρόνο που η διεργασία έχει περάσει στη ready queue από την τελευταία χρονική στιγμή που εκτελέστηκε – ή εναλλακτικά από τη χρονική στιγμή που μπήκε για τελευταία φορά στη ready queue, εάν δεν έχει ακόμα εκτελεστεί μετά την τελευταία της είσοδο.

- Υλοποιήστε τα 3 profiles ελέγχου για τον έλεγχο λειτουργικότητας και την πειραματική αξιολόγηση του χρονοδρομολογητή σας. Σχεδιάστε και εκτελέστε τα κατάλληλα πειράματα, υπό τον έλεγχο του νέου χρονοδρομολογητή που υλοποιήσατε. Θα πρέπει να αξιολογήστε τόσο το χρονοδρομολογητή που λαμβάνει υπόψη του το Exp_Burst, όσο και αυτόν που λαμβάνει υπόψη του το Goodness.

- Εφόσον ολοκληρώσατε και ελέγξατε την υλοποίηση σας, θα πρέπει να σβήσετε τυχόν βοηθητικά μηνύματα εκτύπωσης που εσείς εισάγατε, ώστε η τελική έξοδος να περιέχει μόνο μηνύματα που προυπήρχαν στην υλοποίηση.

3.3 Συμβάσεις Υλοποίησης

Για την υλοποίησή σας θα πρέπει να λάβετε υπόψη τις ακόλουθες συμβάσεις. Όπως θα παρατηρήσετε, και ο κώδικας που σας δίνεται βασίζεται σε αυτές.

- Στην κεφαλή της δομής runqueue αποθηκεύεται η διεργασία *init*. Αυτή δημιουργείται κατά την αρχικοποίηση της εικονικής μηχανής. Η διεργασιά αυτή αντιστοιχεί στην διεργασιά *init* του λειτουργικού συστήματος Linux. Σε περίπτωση που δεν υπάρχει κάποια άλλη διεργασία προς εκτέλεση, δηλαδή η ουρά έτοιμων διεργασιών έχει αποθηκευμένο μόνο ένα στοιχείο (*nr_running = 1*), θα πρέπει να επιλέγετε την διεργασία *init* προς εκτέλεση. Με άλλα λόγια, η διεργασία *init* παίζει ταυτόχρονα και το ρόλο της *idle* process.

- Όταν μία διεργασία αφαιρείται από την δομή runqueue, δηλαδή δεν είναι έτοιμη προς εκτέλεση, θα πρέπει να θέτετε τα πεδία *next* και *prev* της αντίστοιχης δομής task_struct στην τιμή *NULL*.

4 Δημιουργία του προς Υποβολή Συνημμένου Αρχείου

4.1 Πακετάρισμα των Προς Υποβολή Στοιχείων σε Ένα Αρχείο

Το τελευταίο βήμα είναι να πακετάρετε όλα τα αρχεία που πρέπει να μας στείλετε σε ένα αρχείο. Πηγαίνετε στο home directory του λογαριασμού σας. Φτιάξτε εκεί με την εντολή mkdir έναν κατάλογο με όνομα project_2_AEM1_AEM2_AEM3. Για παράδειγμα, η ομάδα με μέλη τους φοιτητές με ΑΕΜ 1234 2345 3456 θα πρέπει να δώσει την εντολή mkdir project_2_1234_2345_3456.

Αντιγράψτε τα αρχεία schedule.c και schedule.h του φακέλου Project2_VM σε αυτό τον κατάλογο με την εντολή cp. Π.χ. η ομάδα με μέλη 1234 2345 3456 θα δώσει την εντολή:

cp Project2_VM/schedule.[ch] project_2_1234_2345_3456

Κάντε το ίδιο με όποια επιπλέον αρχεία έχετε δημιουργήσει, το Makefile που χρησιμοποιήσατε καθώς και τα αρχεία (profiles) που χρησιμοποιήσατε κατά την διαδικασία της αξιολόγησης. Επίσης, δημιουργήστε μέσα στον κατάλογο ένα αρχείο με όνομα *README.txt* στον οποίο θα γράψετε τα ονόματα, ΑΕΜ και e-mail των μελών της ομάδας. Μπορείτε να συμπεριλάβετε στο αρχείο και οτιδήποτε άλλο θέλετε να έχουμε υπόψη μας κατά τη διόρθωση. Το αρχείο θα πρέπει να είναι απλό αρχείο κειμένου (όχι .doc, .docx, .pdf ή οτιδήποτε παρόμοιο). Επίσης, προσέξτε το encoding να είναι UTF-8 (unicode).

Ακολούθως, δώστε την εντολή

tar -cvf project_2_AEM1_AEM2_AEM3.tar project_2_AEM1_AEM2_AEM3

Για παράδειγμα, η ομάδα με μέλη 1234 2345 3456 θα δώσει την εντολή

tar -cvf project_2_omada_1.tar project_2_1234_2345_3456

Με την εντολή tar θα πακεταριστούν τα περιεχόμενα του καταλόγου σε ένα αρχείο με κατάληξη .tar.

Τέλος, δώστε την εντολή bzip2 project_2_omada_<arithmos_omadas>.tar. Θα δημιουργηθεί ένα αρχείο με κατάληξη .bz2, το οποίο περιέχει συμπιεσμένο το αρχείο με κατάληξη .tar. Για παράδειγμα, η ομάδα 1234 2345 3456 θα δώσει την εντολή bzip2 project_2_1234_2345_3456.tar. **Το αρχείο με κατάληξη .bz2 είναι αυτό που θα πρέπει να επισυνάψετε κατά την παράδοση της άσκησης στο e-class**.