ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μια τεχνική χρονοπρογραμματισμού νημάτων με σκοπό την αποφυγή εξόντωσης

Μανούσος - Γαβριήλ Γ. Αθανασούλης

Επιβλέπων: **Ευστάθιος Χατζηευθυμιάδης**, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2008

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μια τεχνική χρονοπρογραμματισμού νημάτων με σκοπό την αποφυγή εξόντωσης

Μανούσος - Γαβριήλ Γ. Αθανασούλης

Α.Μ.: Μ800

ΕΠΙΒΛΕΠΩΝ:
Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:
Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ
Αλέξιος Δελής, Αναπληρωτής Καθηγητής ΕΚΠΑ

Μάρτιος 2008

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή πραγματεύεται το φαινόμενο της εξόντωσης (page-thrashing). Το φαινόμενο αυτό, το οποίο μπορεί να συμβεί σε οποιονδήποτε υπολογιστή εάν αυξηθεί το επίπεδο πολυπρογραμματισμού, χωρίς να έχουν εξασφαλιστεί οι απαραίτητοι πόροι, οδηγεί σε πολύ κακή εμπειρία χρήστη αλλά και σε πολύ κακή συνολική λειτουργία του υπολογιστικού συστήματος στο οποίο συμβαίνει.

Στα πλαίσια της εργασίας αυτής σχεδιάστηκε και υλοποιήθηκε ένας μηχανισμός αξιολόγησης των νημάτων με βάση την τρέχουσα – κυρίως – επίδραση της εκτέλεσης του κάθε νήματος στην κατάσταση του συστήματος. Με βάση την αξιολόγηση αυτή, ένας μηχανισμός που εδράζεται σε κάθε διεργασία, αποδίδει εσωτερικές προτεραιότητες στα νήματά της εκάστοτε διεργασίας και, εάν οι συνθήκες του συστήματος το απαιτούν, επιλέγεται το νήμα που έχει να επιδείξει τη θετικότερη επίδραση στην κατάσταση του συστήματος.

Με αυτόν τον τρόπο επιτυγχάνεται ένα κατανεμημένο σχήμα αντιμετώπισης της εξόντωσης, το οποίο βασίζεται στην επιμέρους, δηλαδή στην ανά διεργασία, βελτίωση των συνθηκών εκτέλεσης. Στο τέλος της εργασίας παρουσιάζεται μια πληθώρα πειραμάτων καθώς και τα ειδικότερα συμπεράσματα που προέκυψαν.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Λειτουργικά Συστήματα, Κατανεμημένοι Αλγόριθμοι

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: εξόντωση, χρονοπρογραμματισμός, νήματα, διεργασίες, διαχείριση μνήμης

ABSRACT

Page thrashing is the result of the increase of multiprogramming level without having the necessary resources to support it. This phenomenon can happen to every computer and it causes very bad user experience and utilization of the computer resources.

In this text, a new thread scheduling mechanism was introduced, implemented and studied. The proposed inter-process mechanism assigns a utility at every thread of each process every time the parent process is scheduled, and the subsequent thread-scheduling decision is based upon the assigned utilities. The utility value is based on the contribution of each thread on the increase or the decrease of the system's condition metrics (processor utilization and page fault rate). Depending on the system's conditions either the thread that maximizes or the thread that minimizes the utility value is chosen to be executed. The core idea is to schedule threads that cause low page fault rates and high processor utilization when the system's conditions are hard and the opposite when the system's conditions are normal.

The designed mechanism is a distributed paradigm of thread scheduling technique aiming at the reduction of page-thrashing. In the end of the thesis, simulation results are presented, where it is shown that the proposed mechanism increases the processor utilization and reduces the average turnaround time.

SUBJECT AREA: Operating Systems, Distributed Algorithms

KEYWORDS: page thrashing, time scheduling, threads, processes, memory management

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Στάθη Χατζηευθυμιάδη για την επίβλεψη της εργασίας αυτής, για τις συμβουλές και τις συζητήσεις που είχαμε, καθώς και τον καθηγητή του τμήματός Αλέξη Δελή, που έλαβε μέρος στην εξεταστική επιτροπή της διπλωματικής εργασίας μου.

Επίσης, σημαντική συμβολή στη δημιουργία ενός ευχάριστου και παραγωγικού κλίματος εργασίας έχουν όλα τα παιδιά από την ερευνητική ομάδα διάχυτου υπολογισμού (http://p-comp.di.uoa.gr) με αρκετούς από τους οποίους είμαστε εδώ και ενάμιση χρόνο στο ίδιο γραφείο.

Σημαντική είναι η στήριξη και η βοήθεια που είχα και έχω από τους γονείς μου και τον αδερφό μου, τους οποίους ευχαριστώ γιατί ποτέ δεν έχουν σταματήσει να με παρακινούν και να με βοηθούν κατά τις δυνάμεις τους.

Θα ήθελα, επίσης, να ευχαριστήσω το Κοινωφελές Ίδρυμα «Αλέξανδρος Ωνάσης» για την οικονομική υποστήριξη που μου παρείχε για ένα ακαδημαϊκό έτος μέσω του θεσμού «Μεταπτυχιακές Υποτροφίες Εσωτερικού».

Μάρτιος 2008

Μανούσος – Γαβριήλ Αθανασούλης

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Η εργασία αυτή εκπονήθηκε στα πλαίσια της ολοκλήρωσης των μεταπτυχιακών μου σπουδών στο πρόγραμμα μεταπτυχιακών σπουδών (ΠΜΣ) του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών.

Το παρόν κείμενο οργανώθηκε σε δύο μέρη. Στο Α' Μέρος ορίζεται το πρόβλημα καθώς και οι βασικές έννοιες που χρειάζονται μέσα από τρία κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μία γενική εισαγωγή που βοηθά τον αναγνώστη να καταλάβει το πρόβλημα που πραγματεύεται η εργασία αυτή. Στο δεύτερο κεφάλαιο γίνεται μια αναδρομή στις τεχνικές χρονοπρογραμματισμού νημάτων, διεργασιών καθώς και στη διαχείριση μνήμης. Παρουσιάζονται τόσο θεωρητικά οι αλγόριθμοι που χρησιμοποιούνται όσο και οι αντίστοιχες ειδικές περιπτώσεις για δημοφιλή λειτουργικά συστήματα, όπως το Linux, λειτουργικά συστήματα τύπου Unix, τα Windows NT και άλλα λειτουργικά. Στο τρίτο κεφάλαιο ορίζεται αυστηρά το πρόβλημα της εξόντωσης (page-thrashing) και περιγράφονται άλλες εργασίες που αποσκοπούν στην μείωση της έντασης του φαινομένου αυτού.

Στο Β' Μέρος παρουσιάζεται η στρατηγική αντιμετώπισης του φαινομένου που σχεδιάστηκε και η προσομοίωση που έγινε. Πιο συγκεκριμένα στο τέταρτο κεφάλαιο παρουσιάζεται ο προτεινόμενος μηχανισμός χρονοπρογραμματισμού νημάτων και στο πέμπτο κεφάλαιο αναλύεται η προσομοίωση που σχεδιάστηκε για να δοκιμαστεί το προτεινόμενο σχήμα. Στο έκτο κεφάλαιο παρουσιάζονται τα αποτελέσματα της προσομοίωσης μαζί με τα συμπεράσματα από αυτά και στο έβδομο κεφάλαιο γίνονται αναφορές για μελλοντική εργασία.

Μέρος Α΄: Βασικές έννοιες και ορισμός το προβλήματος

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Το κεντρικό τμήμα ενός υπολογιστή ως προς το υλικό είναι ο επεξεργαστής. Για να έχει νόημα ο επεξεργαστής πρέπει να υλοποιεί μια πραγματικότητα που περιγράφεται από ένα σύνολο τιμών και καταστάσεων. Αυτό ακριβώς αναπαριστά μια διεργασία. Κάθε διεργασία είναι η ενεργή υλοποίηση ενός προγράμματος σε ένα δεδομένο επεξεργαστή.

Κάθε λειτουργικό που υποστηρίζει πολυπρογραμματισμό (multitasking operating system), εναλλάσσει την ενεργή διεργασία πολύ συχνά ώστε να δίνεται η εντύπωση της ταυτόχρονης εκτέλεσης. Η εκτέλεση διεργασιών εκ περιτροπής (round robin) οδηγεί στην ανάγκη για δημιουργία ενός αποδοτικού μηχανισμού εναλλαγής διεργασιών (context switch). Σε κλασσικούς αλγορίθμους χρονοδρομολόγησης διεργασιών επιλέγεται ένα «κβάντο» εκτέλεσης διεργασίας (δηλαδή, ο συνεχόμενος χρόνος κατά τον οποίο μια διεργασία εκτελείται) το οποίο είναι 10 και πάνω φορές μεγαλύτερο από το χρόνο που χρειάζεται η εναλλαγή μεταξύ δύο διεργασιών. Όπως θα αναλυθεί και στη συνέχεια, οι αλγόριθμοι για χρονοδρομολόγηση διεργασιών γίνονται πιο περίπλοκοι, λαμβάνουν υπόψη πολλές ακόμα παραμέτρους και χρησιμοποιούν πιο σύνθετες δομές.

Ο κώδικας και τα δεδομένα κάθε διεργασίας εδράζουν στη μνήμη κατά τη διάρκεια της εκτέλεσης. Πιο συγκεκριμένα, πριν μια διεργασία εκτελεστεί, φορτώνεται από το δίσκο στη μνήμη (ολόκληρη ή κατά τμήματα), όπου και παραμένει μέχρι να ολοκληρωθεί η εκτέλεσή της. Σε όλα το πολυπρογραμματιστικά περιβάλλοντα, ανά πάσα στιγμή, βρίσκονται στην κύρια μνήμη αρκετές διεργασίες, αφού εναλλάσσονται ως προς την εκτέλεση. Αυτό έχει ως συνέπεια να έρχονται δεδομένα και κώδικας στη μνήμη για όλες τις ενεργές διεργασίες. Εάν η μνήμη γεμίσει, για να εκτελεστεί μια διεργασία της οποίας ο κώδικας δεν είναι φορτωμένος, πρέπει να έρθει στη μνήμη ο κώδικάς της αντικαθιστώντας κώδικα άλλων διεργασιών.

Η παραπάνω διαδικασία ονομάζεται διαχείριση μνήμης (Memory Management) και υλοποιείται σε κάθε σύστημα από το αντίστοιχο Memory Management Unit. Τόσο ο κώδικας όσο και τα δεδομένα, όπως θα δούμε στη συνέχεια, είναι οργανωμένα σε σελίδες οι οποίες φορτώνονται από το δίσκο στην κύρια μνήμη και εάν χρειαστεί να αντικατασταθούν υλοποιείται ένας αλγόριθμος αντικατάστασης σελίδων. Εάν, όμως, απαιτείται περισσότερος χώρος από όσο έχει η κύρια μνήμη για ενεργές διεργασίες,

τότε χρησιμοποιείται η εικονική μνήμη (virtual memory). Όλα τα σύγχρονα λειτουργικά συστήματα δεσμεύουν στο δίσκο κάποιο χώρο στον οποίο αποθηκεύουν σελίδες που είναι πιθανό να χρησιμοποιηθούν στο μέλλον από διεργασίες που τρέχουν αλλά δεν υπάρχει χώρος για να παραμείνουν στην κύρια μνήμη. Τόσο η εικονική όσο και η κύρια μνήμη είναι οργανωμένες σε σελίδες (pages).

Όπως είναι γνωστό από την αρχιτεκτονική υπολογιστών, υπάρχουν και άλλα επίπεδα μνήμης, όπως η κρυφή μνήμη (cache memory), η οποία όμως δεν αποτελεί ενδιαφέρον στοιχείο για την παρούσα μελέτη.

Όταν η κύρια μνήμη είναι «γεμάτη» και ένα σύστημα αρχίσει να χρησιμοποιεί εντατικά την εικονική μνήμη αρχίζει ένα πρόβλημα απόδοσης. Αυτό είναι άμεση συνέπεια του ότι η προσπέλαση σελίδων που είναι στο δίσκο (όπου εδράζει η εικονική μνήμη), απαιτεί πολύ περισσότερο χρόνο από ότι η προσπέλαση σελίδων που είναι στη μνήμη. Εάν, λοιπόν, με κάποια μη αμελητέα πιθανότητα για κάθε διεργασία χρειάζεται μια σελίδα από την εικονική μνήμη (εάν, δηλαδή συμβεί ένα σφάλμα σελίδας – page fault) τότε ο επεξεργαστής πριν προλάβει να εκτελέσει εντολές αρχίζει να ζητά σελίδες από την εικονική μνήμη, οι οποίες είναι στο δίσκο και περιμένει την ολοκλήρωση της μεταφοράς τους. Η αύξηση του λόγου σφαλμάτων σελίδων αυτή, ρίχνει κατακόρυφα το CPU utilization και καθιστά το σύστημα προβληματικό. Αυτό είναι το πρόβλημα με το οποίο ασχολείται η παρούσα μελέτη, γνωστό και ως εξόντωση (page-thrashing).

Για την αντιμετώπιση του προβλήματος αυτού σχεδιάστηκε, μελετήθηκε και προσομοιώθηκε ένας μηχανισμός χρονοπρογραμματισμού νημάτων. Ο μηχανισμός αυτός όταν εντοπίζει ότι οι συνθήκες του συστήματος δυσχαιρένουν, προσπαθεί να επιλέξει προς εκτέλεση νήματα τα οποία επιδεικνύουν θετική συνεισφορά στις συνθήκες του συστήματος. Ο μηχανισμός αυτός υλοποιείται σε επίπεδο διεργασίας, δημιουργώντας, έτσι, ένα κατανεμημένο σχήμα αντιμετώπισης της εξόντωσης. Ο μηχανισμός κάθε διεργασίας αποφασίζει εάν θεωρεί το σύστημα επιβαρυμένο ή όχι και ανάλογα με αυτήν την απόφαση χρονοπρογραμματίζει νήματα που πιστεύει ότι θα οδηγήσουν σε βελτίωση των συνθηκών του συστήματος, χωρίς να έχει πληροφορίες από άλλες διεργασίες.

Συνοψίζοντας τη βασική περιγραφή του μηχανισμού, πρέπει να τονιστεί ότι προϋποτίθεται η ύπαρξη ενός κεντρικού χρονοπρογραμματιστή (διεργασιών) και ενός μηχανισμού χρονοπρογραμματισμού νημάτων που λειτουργεί ανεξάρτητα σε κάθε

διεργασία. Ακριβώς, αυτός ο δεύτερος μηχανισμός μελετήθηκε, τροποποιήθηκε και προσομοιώθηκε στην εργασία αυτή.

ΚΕΦΑΛΑΙΟ 2

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΔΙΕΡΓΑΣΙΩΝ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ

Σε αυτό το κεφάλαιο περιγράφεται η διαδικασία εκτέλεσης των διεργασιών σε ένα υπολογιστικό σύστημα, οι αλγόριθμοι που έχουν κατά καιρούς χρησιμοποιηθεί, καθώς και τα βασικά λειτουργικά συστατικά (υλικό και λογισμικό, με έμφαση στο λογισμικό) που απαρτίζουν το μηχανισμό χρονοπρογραμματισμού και διαχείρισης μνήμης.

2.1 Ορισμοί - Έννοιες

Αρχίζουμε δίνοντας τους ορισμούς των εννοιών που θα χρησιμοποιηθούν, καθώς και μια σχηματική περιγραφή της δομής της εκτέλεσης.

2.1.1 Διεργασία – Νήμα

Η *διεργασία (process)* είναι το βασικότερο υπολογιστικό στοιχείο. Αποτελεί ένα στιγμιότυπο του επεξεργαστή και των δεδομένων που είναι αποθηκευμένα σε αυτόν. Αναπαρίσταται από μια δομή δεδομένων που ονομάζεται *Δομή Ελέγχου Διεργασίας (Process Control Block – PCB)*, στην οποία δομή περιλαμβάνεται:

- η ταυτότητα της διεργασίας (Process ID),
- οι τρέχουσες τιμές των καταχωρητών και, ιδίως, η τιμή του Program Counter,
- ο χώρος διευθύνσεων της επεξεργασίας (address space),
- η προτεραιότητα της διεργασίας (priority)
- μια λίστα από ανοιχτά αρχεία και sockets
- στοιχεία και στατιστικά εκτέλεσης (process accounting information)
- ένα δείκτη στο επόμενο προς εκτέλεση PCB.

Πολλές γλώσσες προγραμματισμού και λειτουργικά συστήματα υποστηρίζουν τα λεγόμενα *νήματα εκτέλεσης (threads of execution)*. Τα νήματα (όπως θα τα καλούμε) είναι παρόμοια με τις διεργασίες αφού αναπαριστούν μια ακολουθία εντολών που εκτελούνται ταυτόχρονα με άλλες ακολουθίες είτε λόγω εκτέλεσης εκ περιτροπής, είτε

λόγω εκτέλεσης σε περιβάλλον με πολλούς επεξεργαστές. Η βασική διαφορά είναι ότι κάθε διεργασία μπορεί να έχει περισσότερα του ενός νήματα, τα οποία έχουν κοινό χώρο διευθύνσεων και άρα μπορούνε να μοιράζονται δεδομένα εκ κατασκευής.

Σε κλασσικά λειτουργικά συστήματα, όπως το Unix, οι διεργασίες είναι το βασικό συστατικά χρονοπρογραμματισμού και τα νήματα είναι απλά μια προσθήκη που επιτρέπει αύξηση του πολυπρογραμματισμού και «σπάσιμο» ενός προγράμματος σε δύο ή παραπάνω οντότητες που εκτελούνται. Σε κάποια σύγχρονα λειτουργικά συστήματα όπως το Linux, οι δύο έννοιες είναι πιο συγκεχυμένες. Η διεργασίες μπορούν να μοιράζονται το χώρο διευθύνσεων και άλλα στοιχεία με άλλες διεργασίες, μοιάζοντας, έτσι, περισσότερο με τα κλασσικά νήματα. Αυτές οι διεργασίες καλούνται *Light-weight processes (LWP)* και μπορούν να υλοποιήσουν τη λογική των νημάτων, εάν σε κάθε ομάδα διεργασιών που μοιράζονται πόρους αναθέτουμε έναν κωδικό και τα καθιστούμε μια ομάδα, η οποία αντιστοιχεί με την κλασσική έννοια της διεργασίας.

2.1.2 Χρονοπρογραμματιστής διεργασιών – νημάτων

Κάθε λειτουργικό σύστημα έχει ένα μηχανισμό με τον οποίο χρονοπρογραμματίζονται οι διεργασίες ή/και τα νήματα. Στα λειτουργικά συστήματα πραγματικού χρόνου κυριαρχούν αλγόριθμοι που βασίζονται στην εκ περιτροπής εκτέλεση. Βέβαια αυτό δεν αρκεί για αποδοτική εκτέλεση και έτσι εισάγεται η έννοια της προτεραιότητας. Ο χρονοπρογραμματιστής ενός λειτουργικού έχει τους παρακάτω στόχους:

- Να μειώσει το μέσο χρόνο απόκρισης (mean response time)
- Να μειώσει τη διασπορά του χρόνου απόκρισης (variance of response time)
- Να μεγιστοποιήσει τη ρυθμαπόδοση του συστήματος (throughput)
    - μειώνοντας το πλεονάζοντα φόρτο (overhead), και
    - με αποτελεσματική χρήση των διαθέσιμων πόρων
- Να μεγιστοποιήσει τη χρησιμοποίηση του επεξεργαστή (CPU utilization)
- Να παραμείνει δίκαιος
    - Λόγω του ότι η μείωση του μέσου χρόνου απόκρισης οδηγεί σε λιγότερο δίκαιο χρονοπρογραμματισμό, εδώ δημιουργείται ένα αντιστάθμισμα (tradeoff) μεταξύ δικαιοσύνης και μείωσης χρόνου απόκρισης

Οι αλγόριθμοι χρονοπρογραμματισμού καθώς και τα κριτήρια για την επιλογή του καθενός θα περιγραφούν αναλυτικά σε επόμενη ενότητα.

2.1.3 Κύρια μνήμη

Η κύρια μνήμη (main memory) είναι η μνήμη στην οποία βρίσκονται όλα τα προγράμματα προς εκτέλεση και είναι άμεσα προσβάσιμη από τον επεξεργαστή. Στη κύρια μνήμη αποθηκεύονται, επίσης, τα δεδομένα που απαιτούνται και παράγονται από τα προγράμματα που εκτελούνται. Η κύρια μνήμη είναι αρκετά μεγάλη ώστε να χωράει πολλά προγράμματα ταυτόχρονα αλλά και αρκετά γρήγορη ώστε να μην καθυστερεί η εκτέλεση προγραμμάτων αναμένοντας των κώδικα από αυτήν. (Η αλήθεια, όπως είναι γνωστό, είναι λίγο πιο πολύπλοκη. Τα αποθηκευτικά μέσα διαχωρίζονται σε ιεραρχίες. Πάνω από το σκληρό δίσκο είναι η κύρια μνήμη, πάνω από την οποία είναι η κρυφή μνήμη – η οποία έχει δύο επίπεδα – και, τέλος, βρίσκονται οι καταχωρητές του επεξεργαστή.)

Η κύρια μνήμη αν και αρκετά μεγάλη για να χωράει πολλά προγράμματα δεν είναι όσο μεγάλος είναι ο σκληρός δίσκος και έτσι κατά την, εκ περιτροπής, εκτέλεση διαφόρων προγραμμάτων είναι πιθανό να ζητηθούν για εκτέλεση σελίδες που δε χωράνε στη μνήμη. Σε αυτήν την περίπτωση με έναν αλγόριθμό αντικατάστασης σελίδων αντικαθίστανται κάποιες σελίδες με τις νέες σελίδες που απαιτούνται για την εκτέλεση του τρέχοντος προγράμματος. Οι σελίδες που φεύγουν από τη μνήμη πηγαίνουν στην εικονική μνήμη.

2.1.4 Εικονική μνήμη

Η εικονική μνήμη (virtual memory) είναι μια «επέκταση» της κύριας μνήμης. Η αποθήκευση σελίδων στην εικονική μνήμης γίνεται ακριβώς όπως και στην κύρια μνήμη. Η μόνη διαφορά είναι η ταχύτητα πρόσβασης. Η εικονική μνήμη βρίσκεται στο δίσκο και συχνά τη θεωρούμε πολύ μεγάλη. Βέβαια, σε πραγματικά συστήματα η εικονική μνήμη έχει πεπερασμένο μέγεθος και είναι σημαντικό να δεσμευθεί/χρησιμοποιηθεί το κατάλληλο μέγεθος για να έχουμε ένα λειτουργικό σύστημα.

Σκοπός της εικονικής μνήμης είναι το λειτουργικό σύστημα να μπορεί να διαχειριστεί ως μνήμη περισσότερο χώρο από αυτόν που τα παρέχεται από την κύρια μνήμη. Γι’ αυτό το λόγο η Μονάδα Διαχείρισης μνήμης δημιουργεί έναν αδιαφανή τρόπο πρόσβασης σε μια σελίδα στη μνήμη, είτε αυτή είναι στην κύρια μνήμη είτε είναι στην εικονική μνήμη.

2.1.5 Μονάδα Διαχείρισης Μνήμης

Όπως προαναφέρθηκε, τα δεδομένα οργανώνονται τόσο στην κύρια όσο και στην εικονική μνήμη με τον ίδιο τρόπο. Κάθε πρόγραμμα χωρίζεται σε σελίδες (με τυπικό μέγεθος 4096 kB ή 8192 kB) και μεταφέρονται στη μνήμη οι σελίδες που πρόκειται να εκτελεστούν (και πιθανόν και άλλες, ανάλογα με την πολιτική με την οποία φορτώνονται σελίδες – fetch policy). Η μονάδα διαχείρισης μνήμης (Memory Management Unit) αναλαμβάνει να μεταφέρει αυτές τις σελίδες από και προς την εικονική μνήμη, έτσι ώστε να επιτυγχάνεται βέλτιστη λειτουργία. Γι’ αυτό το λόγο υπάρχει μια πληθώρα από αλγορίθμους αντικατάστασης σελίδων (page replacement algorithms), εκ των οποίων επιλέγεται αυτός που ικανοποιεί καλύτερα το σύστημα στο οποίο αναφερόμαστε. Οι αλγόριθμοι αυτοί θα περιγραφούν αναλυτικά σε επόμενη ενότητα.

Η μονάδα διαχείρισης μνήμης χρησιμοποιεί τον πίνακα αντιστοίχησης διεργασιών-μνήμης (Process Memory Map), στον οποίο βρίσκονται οι πληροφορίες που αναφέρονται που είναι αποθηκευμένη η κάθε σελίδα καθώς και διάφορα στατιστικά για την κάθε σελίδα που συμμετέχουν στην απόφαση αντικατάστασης.

2.2 Χρονοπρογραμματισμός Διεργασιών

Σε αυτήν την ενότητα περιγράφεται πιο διεξοδικά ο τρόπος με τον οποίο γίνεται χρονοπρογραμματισμός διεργασιών. Πρέπει να τονιστεί πως κάθε λειτουργικό χρησιμοποιεί διαφορετικές παραλλαγές των αλγορίθμων χρονοπρογραμματισμού διεργασιών. Θα αναφερθούμε στους βασικούς αλγόριθμους χρονοπρογραμματισμού, και στη συνέχεια στους αλγορίθμους που χρησιμοποιούν τα Windows, το Unix και το Linux. Πριν προχωρήσουμε αξίζει να αναφερθεί ότι τα σύγχρονα λειτουργικά συστήματα περιλαμβάνουν έως τριών ειδών χρονοπρογραμματιστές (time schedulers):

- Μακροπρόθεσμοι χρονοπρογραμματιστές (Long-term schedulers), οι οποίοι υποβάλλουν εργασίες στην ουρά έτοιμων διεργασιών (ready queue) και υπάρχουν σε συστήματα πραγματικού χρόνου με υψηλό φόρτο εργασίας,
- Μεσοπρόθεσμοι χρονοπρογραμματιστές (Mid-term schedulers), οι οποίοι αναλαμβάνουν να ολοκληρώσουν το “swap in” και “swap out” των διεργασιών και υπάρχουν σε όλα τα συστήματα, καθώς και,
- Βραχυπρόθεσμοι χρονοπρογραμματιστές (Short-term schedulers ή dispatchers), για τους οποίους θα μιλήσουμε στη συνέχεια.

Όπως αναφέρθηκε νωρίτερα, οι χρονοπρογραμματιστές διέπονται από κάποιες βασικές αρχές, οι οποίες περιληπτικά είναι:

- Μείωση του μέσου χρόνου απόκρισης
- Μείωση της διασποράς του χρόνου απόκρισης
- Μεγιστοποίηση της ρυθμαπόδοσης του συστήματος (throughput)
- Μεγιστοποίηση της χρησιμοποίησης του επεξεργαστή (CPU utilization)
- Δικαιοσύνη

Αυτές οι αρχές εμφανίζουν κάποιες αντιφάσεις. Λόγου χάρη, η δικαιοσύνη μεταξύ του χρόνου που δίνουμε ανά διεργασία οδηγεί σε μείωση του μέσου χρόνου απόκρισης, οπότε σε τέτοιες περιπτώσεις πρέπει να καταφύγουμε σε συμβιβασμούς.

Οι ανωτέρω αρχές ορίστηκαν στην δεκαετία του 1970. Εκείνη την εποχή, οι πιο πολλοί υπολογιστές ακολουθούσαν κάποιες υποθέσεις απλοποίησης που σήμερα απέχουν πολύ από την πραγματικότητα. Τέτοιες υποθέσεις είναι, για παράδειγμα, ότι υπάρχει μία CPU, ότι κάθε χρήστης τρέχει ένα πρόγραμμα και ότι κάθε διεργασία έχει ένα νήμα.

Σήμερα, όμως, κάθε υπολογιστής μπορεί να τρέχει ταυτόχρονα, πολλά προγράμματα από πολλούς χρήστες, και κάθε διεργασία μπορεί να έχει πολλά νήματα. Παρά την παλαιότητά τους, οι αρχές που διατυπώσαμε νωρίτερα δε χάνουν την αξία τους, εξάλλου οι απλοποιητικές υποθέσεις που προαναφέρονται δεν είναι δεσμευτικές για τον τρόπο με τον οποίο προσπαθούμε να φτάσουμε στους στόχους μας, απλά τίθεται ένα καινούργιο ερώτημα. Η σύγχρονη έρευνα σε λειτουργικά συστήματα καλείται να απαντήσει με ποιο τρόπο μπορούμε να εισάγουμε λιγότερο περιοριστικές υποθέσεις και προς ποια κατεύθυνση πρέπει να τροποποιήσουμε τους στόχους αυτούς.

Έχουν ήδη αναφερθεί κάποιες βασικές υποθέσεις αλλά και κάποιους βασικούς στόχους. Σε αυτό το σημείο θα αναφερθούμε στα εργαλεία που μπορούν να αποδώσουμε μια ποσοτική εκτίμηση της απόδοσης του κάθε αλγορίθμου χρονοπρογραμματισμού. Οι μετρικές αυτές, οι οποίες προκύπτουν και απο τους στόχους, είναι οι εξής:

- Χρόνος απόκρισης
    - Μέση τιμή
    - Διασπορά
- Ρυθμαπόδοση (Throughput), δηλαδή ο αριθμός διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου,
- Χρησιμοποίηση του επεξεργαστή/κάποιας άλλης συσκευής (CPU/device utilization), δηλαδή το ποσοστό του χρόνου κατά το οποίο ο επεξεργαστής (ή κάποιο άλλο τμήμα του υπολογιστή) είναι σε χρήση, και,
- Δικαιοσύνη, η οποία ποσοτικοποιείται πιο δύσκολα από τις άλλες μετρικές.

Στη συνέχεια παρουσιάζονται οι βασικές αρχές των παρακάτω πολιτικών:

1. First-In First-Out (FIFO)
2. Shortest Job First (SJF)
3. Shortest (Remaining) Time to Completion First (SRTCF ή STCF)
4. Multi-level Feedback (MLF) queues
5. Priority Scheduling (PS)
6. Round Robin (RR)

Οι παραπάνω τεχνικές είναι μερικές από αυτές που έχουν προταθεί ή και συζητηθεί για υλοποίηση σε λειτουργικά συστήματα. Είναι, όμως, αυτές που συναντά κανείς στη βιβλιογραφία πιο συχνά, οι υπόλοιπες εμφανίζονται πολύ σπάνια.

Οι τεχνικές αυτές μπορούν να οδηγήσουν και σε δημιουργία υβριδικών τεχνικών συνδυάζοντας ιδιότητες από δύο ή παραπάνω. Κάθε πολιτική μπορεί να είναι είτε προεκχωρητική (preemptive) είτε μη-προεκχωρητική (non-preemptive). Μια μη-προεκχωρητική πολιτική χρονοπρογραμματισμού από τη στιγμή που θα αναθέσει μια διεργασία στον επεξεργαστή προς εκτέλεση, ο επεξεργαστής εκτελεί αυτήν τη διεργασία μέχρι να ολοκληρωθεί. Η τεχνική αυτή είναι πολύ απλή στην υλοποίηση αλλά καθόλου δίκαιη, αφού π.χ. μια μικρή διεργασία μπορεί να περιμένει πολύ μια άλλη, μεγάλη, διεργασία πριν εκτελεστεί. Αντίθετα, μια προεκχωρητική πολιτική μπορεί να αναθέσει μια διεργασία προς εκτέλεση στον επεξεργαστή και στη συνέχεια να αναστείλει την εκτέλεση της διεργασίας αυτής και να αναθέσει άλλη διεργασία προς εκτέλεση. Έτσι, κάθε διεργασία εναλλάσσεται μεταξύ των καταστάσεων “ενεργή” και “σε αναμονή” (σε πιο περίπλοκα συστήματα υπάρχουν και άλλες καταστάσεις όπως “σταματημένη για I/O”). Η προεκχωρητικές τεχνικές είναι πιο αποδοτικές, όμως απαιτούν πιο περίπλοκη υλοποίηση. Τέλος, η υποστήριξη από το υλικό είναι απαραίτητη αφού χρειάζονται timers και interrupts. Από τις παραπάνω πολιτικές μπορούμε να πούμε ότι η FIFO, η SJF, και η PS είναι μη-προεκχωρητικές ενώ η MLF, η SRTF, και φυσικά η RR είναι προεκχωρητικές.

2.2.1 First-In First-Out (FIFO)

Ο χρονοπρογραμματισμός FIFO, δεν είναι προεκχωρητικός, έτσι, κάθε διεργασία που αρχίζει να εκτελείται παραμένει σε αυτήν την κατάσταση μέχρι να ολοκληρωθεί. Όπως αφήνει να εννοηθεί και το όνομα του αλγορίθμου αυτού, η σειρά εκτέλεσης των διεργασιών καθορίζεται από τη σειρά με την οποία οι διεργασίες πηγαίνουν στον χρονοπρογραμματιστή. Η υλοποίηση είναι πολύ απλή, όμως δεν καταφέρνουμε να έχουμε ικανοποιητικό χρόνο απόκρισης σε περιπτώσεις που μια διεργασία, ιδιαιτέρως μία μικρή διεργασία, κληθεί να περιμένει την ολοκλήρωση μιας μεγάλης, σε διάρκεια, διεργασίας. Εάν το σύστημα έχει μόνο μικρές διεργασίες, τότε ο αλγόριθμος FIFO οδηγεί σε καλύτερο χρησιμοποίηση του επεξεργαστή. Ακόμη, δεν έχουμε καλό χρόνο απόκρισης όταν εκτελούνται διεργασίες ίδιου μήκους που στάλθηκαν στο χρονοπρογραμματιστή μαζί και, τέλος, σε ένα σύστημα πραγματικού χρόνου με χρήστες

που λειτουργούν διαδραστικά, μια FIFO πολιτική μπορεί να οδηγήσει σε πολύ κακή εμπειρία χρήστη.

2.2.2 Shortest Job First (SJF)

Ο αλγόριθμος αυτός είναι μη-προεκχωρητικός και χρονοπρογραμματίζει σύμφωνα με τη διάρκεια της κάθε διεργασίας, Πιο συγκεκριμένα, από τις τρέχουσες ανά πάσα στιγμή διεργασίες, πρώτη εκτελείται αυτή που έχει τη μικρότερη διάρκεια. Με αυτόν τον τρόπο επιτυγχάνεται η ελαχιστοποίηση του μέσου χρόνου απόκρισης. Το προφανές μειονέκτημα αυτού του αλγορίθμου είναι ότι δεν είναι υλοποιήσιμος, αφού, γενικά, δεν μπορούμε να γνωρίζουμε πόσος είναι ο χρόνος που πρόκειται μια διεργασία να διαρκέσει. Σε ένα σύστημα που η φύση των διεργασιών είναι γνωστή και είναι εφικτό να έχουμε καλές εκτιμήσεις για τη διάρκεια της κάθε διεργασίας ένας τέτοιος αλγόριθμος μπορεί να χρησιμοποιηθεί με καλά αποτελέσματα. Όπως, όμως, προαναφέρθηκε, σε ένα σύστημα γενικής χρήσης, δεν μπορούμε να έχουμε τέτοιες εκτιμήσεις.

2.2.3 Shortest Remaining Time to Completion First (SRTCF)

Ο αλγόριθμος SRTCF είναι η προεκχωρητική εκδοχή του SJF. Είναι βέλτιστος ως προς το μέσο χρόνο απόκρισης, επιτρέπει την ταυτόχρονη εκτέλεση διεργασιών αφού είναι προεκχωρητικός, όμως, δεν είναι υλοποιήσιμος σε συστήματα γενικής χρήσης αφού δεν μπορούμε να έχουμε μια γενική πρόβλεψη διάρκειας μιας διεργασίας εάν οι διεργασίες δεν ακολουθούν ένα πολύ συγκεκριμένο πρότυπο λειτουργίας.

2.2.4 Priority Scheduling (PS)

Η πολιτική χρονοπρογραμματισμού αυτή, εισάγει την έννοια της προτεραιότητας κατά τον χρονοπρογραμματισμό. Κατά την εκτέλεση αλγορίθμων που υλοποιούν την πολιτική αυτή ανατίθεται σε κάθε διεργασία μια προτεραιότητα και, στη συνέχεια, εκτελούνται οι διεργασίες κατά σειρά προτεραιότητας. Η πολιτική αυτή είναι μη-προεκχωρητική, και έτσι, κάθε φορά που εκτελείται μια διεργασία, αυξάνει η προτεραιότητα των διεργασιών που παραμένουν προς εκτέλεση, προκειμένου να αποφευχθεί το ενδεχόμενο νέες διεργασίες με αυξημένη προτεραιότητα να μην επιτρέπουν την εκτέλεση διεργασιών

που είναι για μεγάλο χρονικό διάστημα στο σύστημα αλλά ξεκίνησαν από χαμηλή προτεραιότητα.

2.2.5 Round Robin (RR)

Ο χρονοπρογραμματισμός εκ περιτροπής είναι μια γενίκευση του χρονοπρογραμματισμού FIFO “εξοπλισμένου” με ένα χρονόμετρο και ένα μηχανισμό προεκχώρησης. Ορίζεται ένα κβάντο, δηλαδή η μέγιστη διάρκεια κατά την οποία μια διεργασία μπορεί να εκτελείται χωρίς διακοπή, και οι διεργασίες εκτελούνται με τη σειρά που εισέρχονται στην ουρά έτοιμων διεργασιών (ready queue). Εάν μια διεργασία δεν έχει ολοκληρωθεί κατά το τέλος ενός κβάντου μεταφέρεται στο τέλος της ουράς. Έτσι, μια διεργασία θα εκτελείται κάθε (n-1)*q μονάδες του χρόνου (n: το πλήθος των διεργασιών, q: η διάρκεια του κβάντου).

Η επιλογή της διάρκειας του κβάντου είναι πολύ σημαντική επιλογή. Εάν η διάρκεια του κβάντου είναι μεγάλη τότε το αλγόριθμος μπορεί να οδηγήσει σε μεγάλο χρόνο αναμονής για μια μεμονωμένη διεργασία, και, όπως είναι λογικό, στο όριο να εκφυλιστεί σε έναν αλγόριθμο χρονοπρογραμματισμού FIFO. Εάν το κβάντο είναι μικρό, τότε, το σύστημα ανταποκρίνεται (είναι responsive) αλλά ενδέχεται να έχουμε μειωμένη ρυθμαπόδοση λόγω του ότι η εναλλαγή μεταξύ διεργασιών οδηγεί ένα σημαντικό ποσοστό του χρόνου να αναμένουμε τη φόρτωση της νέας διεργασίας. Έτσι, κάνοντας ένα συμβιβασμό μεταξύ ανταπόκρισης και ρυθμαπόδοσης επιλέγουμε ένα κβάντο που συνήθως έχει διάρκεια τέτοια ώστε ο χρόνος που χρειάζεται για την εναλλαγή διεργασιών (context switch time) να μην υπερβαίνει το 1% της διάρκεια του κβάντου. Με αυτόν τον τρόπο πετυχαίνουμε την ελαχιστοποίηση του επιπλέον κόστους εναλλαγής διεργασιών χωρίς, όμως, να χάσουμε την ανταποκρισιμότητα (responsiveness) του συστήματος.

2.2.6 Multi-level Feedback (MLF) queues

Η πολιτική χρονοπρογραμματισμού με πολυεπίπεδες ουρές με ανάδραση, χρησιμοποιεί την έννοια της προτεραιότητας μιας διεργασίας. Σε μια τέτοια πολιτική, ορίζουμε ένα πλήθος από ουρές, που η κάθε μία ουρά αντιστοιχεί σε μια προτεραιότητα. Μια τέτοια πολιτική είναι προεκχωρητική και, συχνά, το κβάντο εκτέλεσης δεν είναι σταθερό. Οι διεργασίες με μικρή προτεραιότητα εκτέλεσης έχουν μεγάλο κβάντο και οι διεργασίες με

μικρότερη προτεραιότητα εκτέλεσης έχουν μικρότερο κβάντο. Το παραδοσιακό UNIX scheduling είναι μια παραλλαγή της πολιτικής χρονοπρογραμματισμού MLF. Όταν μια διεργασία δημιουργείται, αρχικοποιείται σε μια προτεραιότητα, δηλαδή προστίθεται στο τέλος της ουράς που αντιστοιχεί στην προτεραιότητα αυτή. Όταν πρόκειται να εκτελεστεί μια διεργασία ελέγχονται οι ουρές με τη σειρά (προτεραιότητας) και επιλέγεται η πρώτη διεργασία από την πρώτη ουρά που δεν είναι άδεια. Στη συνέχεια, αφού εκτελεστεί το τρέχον κβάντο, εάν δεν έχει ολοκληρωθεί η διεργασία, μειώνεται η προτεραιότητα της διεργασίας και εισάγεται στην κατάλληλη ουρά.

2.3 Χρονοπρογραμματισμός σε πραγματικές συνθήκες

Σε αυτήν την ενότητα γίνεται μια σύντομη περιγραφή των αλγορίθμων χρονοπρογραμματισμού διεργασιών που χρησιμοποιών μερικά δημοφιλή λειτουργικά συστήματα. Θα αναφερθούμε στις πολιτικές που χρησιμοποιούνται από το MS-DOS, τα Windows 3.1 και NT, το Linux αλλά και το Unix.

2.3.1 MS-DOS και Windows 3.1

Στο πρώτο λειτουργικό δεν υπάρχει ανάγκη χρονοπρογραμματισμού, αφού δεν υποστηρίζει τον πολυπρογραμματισμό. Κάθε διεργασία εκτελείται εξ ολοκλήρου και στη συνέχεια εκτελείται κάθε επόμενη. Η έκδοση των Windows 3.1 περιείχε έναν πολύ απλό μη-προεκχωρητικό χρονοπρογραμματιστή, ο οποίος για να δώσει στη CPU μια άλλη διεργασία έπρεπε να ο προγραμματιστής μέσα στον κώδικα να έχει δώσει μια Yield εντολή. Επί της ουσίας, το περιβάλλον αυτό αν και υποστήριζε τον πολυπρογραμματισμό δεν παρείχε καμία δυνατότητα έξυπνου χρονοπρογραμματισμού μεταξύ των διεργασιών.

2.3.2 Windows NT

Στα Windows NT η Microsoft έφτιαξε για πρώτη φορά ένα πιο ολοκληρωμένο σχήμα χρονοπρογραμματισμού. Αφενός, το ελάχιστο στοιχείο χρονοπρογραμματισμού είναι το νήμα αντί για τη διεργασία. Αφετέρου το σχήμα αυτό χρησιμοποιεί προτεραιότητες. Οι επιτρεπτές τιμές είναι από 1 έως 31. Οι τιμές 1 έως 15 αναφέρονται ως κανονικές προτεραιότητες (normal priorities) και οι τιμές 16 έως 31 αναφέρονται ως προτεραιότητες πραγματικού χρόνου (real-time priorities). Επιπλέον, υπάρχει ένα ειδικό νήμα με προτεραιότητα 0 που “τρέχει” όταν δεν εκτελείται κανένα άλλο νήμα. Η πολιτική που χρησιμοποιείται είναι εκτέλεση εκ περιτροπής με προεκχώρηση και προτεραιότητες (Priority-based Preemptive Round Robin).

Ο χρονοπρογραμματιστής παίρνει μια νέα απόφαση κάθε φορά που το κβάντο ενός νήματος ολοκληρωθεί, κάθε φορά που ένα νήμα σταματά τη λειτουργία του (suspends the execution) περιμένοντας για ένα γεγονός (event), καθώς και κάθε φορά που ένα νήμα γίνεται έτοιμο προς εκτέλεση (ready). Εάν ένα νέο νήμα προς εκτέλεση έχει

μεγαλύτερη προτεραιότητα από το νήμα που εκτελείται τη δεδομένη χρονική στιγμή, τότε το τρέχον νήμα παραχωρεί τη θέση του στο νήμα με τη μεγαλύτερη προτεραιότητα.

Ο χρονοπρογραμματιστής υλοποιεί μια στρατηγική που βοηθά τα νήματα με χαμηλή προτεραιότητα να μην περιέλθουν σε κατάσταση έλλειψης πόρων (starvation). Η τεχνική που ακολουθείται είναι η εξής: ο χρονοπρογραμματιστής ελέγχει για κάθε προτεραιότητα (αρχίζοντας από την 31) για κάποιο νήμα που δεν έχει εκτελεστεί για πάνω από 3s. Εάν βρεθεί κάποιο τέτοιο νήμα, παίρνει τη βέλτιστη προτεραιότητα και διπλασιάζεται το κβάντο του. Επομένως, χρονοπρογραμματίζεται άμεσα. Μετά τη λήξη του κβάντου αυτού επανέρχεται στην προηγούμενη προτεραιότητα και στο προηγούμενο κβάντο.

2.3.3 Traditional Unix

Σε αυτήν την ενότητα περιγράφεται η πολιτική χρονοπρογραμματισμού του παραδοσιακού Unix. Μεμονωμένες υλοποιήσεις μπορεί να υιοθετούν διαφορετική πολιτική, ενώ το ίδιο μπορεί να συμβαίνει και με νέες εκδόσεις του Unix. Το ελάχιστο στοιχείο χρονοπρογραμματισμού είναι η διεργασία.

Η πολιτική που υιοθετήθηκε αρχικά στο Unix είναι MLF ουρές με εκ περιτροπής εκτέλεση σε κάθε ουρά. Αρχικά, η διάρκεια του κβάντου ήταν 1s ενώ από το BSD4.3 το κβάντο έγινε 0.1s και οι προτεραιότητες επαναϋπολογίζονται κάθε 1s. Η προτεραιότητα της κάθε διεργασίας, η οποία υπολογίζεται με συγκεκριμένες φόρμουλες που δίνονται στη συνέχεια, εξαρτάται από το είδος της διεργασίας και το παρελθόν της εκτέλεσης της συγκεκριμένης διεργασίας. Το πλήθος των προτεραιοτήτων ορίστηκε αρχικά ίσο με 40, αλλά σήμερα συναντά κανείς μέχρι και 160 επίπεδα (Solaris). Τέλος, υλοποιείται και ένα σχήμα γήρανσης (aging scheme) προκειμένου να αποφευχθεί η έλλειψη πόρων (starvation).

Ο πυρήνας του σχήματος αυτού είναι ο υπολογισμός των νέων προτεραιοτήτων. Τόσο το σχήμα γήρανσης, όσο και η πολιτική προτεραιοτήτων καθαυτή βασίζονται πάνω στο συγκεκριμένο υπολογισμό.

$$P(j,i) = Base(j) + {}^{CPU(j,i-1)}/_{2} + nice(j)$$

$$CPU(j,i) = {}^{U(j,i)}/_{2} + {}^{CPU(j,i-1)}/_{2}$$

όπου:

j : διεργασία

i : (χρονική) διάστημα

$P(j,i)$: προτεραιότητα της διεργασίας j στην αρχή του διαστήματος i

$Base(j)$: αρχική προτεραιότητα της διεργασίας j

$U(j,i)$: η χρησιμοποίηση του επεξεργαστή από τη διεργασία j στο διάστημα i

$CPU(j,i)$: η χρησιμοποίηση του επεξεργαστή από τη διεργασία j έως το διάστημα i (υπολογισμένη με βάρη εκθετικά κατανεμημένα)

$nice(j)$: ένας ελεγχόμενος από το χρήστη παράγοντας για μείωση της προτεραιότητας της διεργασίας j.

2.3.4 Linux

Ο χρονοπρογραμματισμός διεργασιών στο Linux είναι κάπως πιο σύνθετος. Αρχικά, η το ελάχιστο στοιχείο της διαδικασία αυτής είναι το νήμα, όπως ακριβώς και στα Windows NT. Επίσης, υπάρχει μια διαδικασία χρονοπρογραμματισμού για κάθε επεξεργαστή. Αυτή η πρόβλεψη υπάρχει γιατί πολλές φορές το λειτουργικό αυτό χρησιμοποιείται σε clusters και σε ισχυρά υπολογιστικά συστήματα που μπορεί να έχουν πάνω από έναν επεξεργαστή ανά μητρική κάρτα.

Κάθε διαδικασία χρονοπρογραμματισμού έχει 140 ουρές εκτέλεσης. Οι πρώτες 100 έχουν δεσμευτεί για διεργασίες πραγματικού χρόνου και οι επόμενες 40 για εφαρμογές των χρηστών. Σε κάθε νήμα αντιστοιχεί ένα χρονικό διάστημα (time-slice) κατά το οποίο μπορεί να εκτελείται. Η ομάδα ουρών εκτέλεσης αυτή ονομάζεται ενεργή ομάδα ουρών εκτέλεσης (active runqueue) και υπάρχει μια ακόμα τέτοια ομάδα ουρών εκτέλεσης που δεν είναι ενεργή (expired runqueue).

Όταν ένα νήμα έχει εκτελεστεί για όλο το χρονικό διάστημα που του έχει ανατεθεί και δεν έχει ολοκληρωθεί τότε μεταφέρεται στην αντίστοιχη μη ενεργή ουρά όπου του ανατίθεται ένα νέο χρονικό διάστημα. Εάν όλες οι ουρές της ενεργής ομάδας ουρών είναι άδειες τότε εναλλάσσονται οι δείκτες των δύο ουρών. Με αυτόν τον τρόπο η

διαδικασία χρονοπρογραμματισμού είναι πολύ απλή. Επιλέγεται η πρώτη διεργασία της ουράς με τη μεγαλύτερη προτεραιότητα από την ενεργή ομάδα ουρών. Συνεπώς, η πολυπλοκότητα επιλογής διεργασίας είναι ντετερμινιστική και σταθερή αφού είναι ανεξάρτητη από το πλήθος των διεργασιών και εξαρτάται μόνο από το πλήθος ουρών. Επίσης, η εναλλαγή των ουρών είναι μια απλή εναλλαγή δύο δεικτών με σταθερό και πολύ μικρό χρονικό κόστος.

Ο πυρήνας του χρονοπρογραμματισμού του Linux είναι η επιλογή της προτεραιότητας και του χρονικού διαστήματος εκτέλεσης. Ο χρονοπρογραμματισμός αυτού του τύπου είναι πολύ αποδοτικός για SMP συστήματα (δηλαδή Symmetric Multi-Processing συστήματα, τα οποία αποτελούνται από πολλούς επεξεργαστές που έχουν, όμως, κοινή μνήμη) αφού παρέχουν διαφορετικό κλείδωμα για κάθε ομάδα ουρών εκτέλεσης. Εξάλλου, όπως προαναφέραμε κάθε επεξεργαστής έχει τη δική του ομάδα ουρών εκτέλεσης. Επιπλέον, υποστηρίζεται η προεκχώρηση (preemption) νημάτων καθώς και η δυναμική απόδοση προτεραιοτήτων. Αυτό γίνεται “αμείβοντας” τις διεργασίες που δεν χρησιμοποιούν τον επεξεργαστή πολύ με μεγαλύτερη προτεραιότητα (συχνά αυτό συμβαίνει σε διεργασίες που έχουν πολύ I/O μιας και αποσύρονται από τον επεξεργαστή όταν ξεκινήσει μία αίτηση για είσοδο ή έξοδο δεδομένων) και “τιμωρώντας” τις διεργασίες που χρησιμοποιούν τον επεξεργαστή με μείωση προτεραιότητας (προφανώς πρόκειται για διεργασίες που εκτελούν υπολογισμούς και εν γένει δεν έχουν επικοινωνία με το δίσκο). Τέλος, αξίζει να αναφερθεί ότι σε πολυ-επεξεργαστικά συστήματα το Linux αναθέτει σε έναν επεξεργαστή να ελέγχει κάθε 200ms εάν το φορτίο είναι κατανεμημένο στους διαθέσιμους επεξεργαστές και εάν δεν είναι, αναλαμβάνει να αναδιανείμει τα νήματα στους επεξεργαστές ώστε να βελτιωθεί η ισοκατανομή φορτίου (load balancing).

2.4 Διαχείριση Μνήμης

Το πρόβλημα που διαπραγματεύεται η παρούσα διατριβή αναφέρεται άμεσα στην κύρια μνήμη του υπολογιστή. Είναι επόμενο να είναι πολύ σημαντικό η εξοικείωση με τη διαχείριση της μνήμης αυτής. Σε αυτήν την ενότητα θα παρουσιαστούν αλγόριθμοι που έχουν προταθεί και εν γένει χρησιμοποιούνται για διαχείριση της μνήμης, καθώς και οι βασικές έννοιες της διαχείρισης μνήμης. Πιο συγκεκριμένα, θα παρουσιαστούν αλγόριθμοι που αναλαμβάνουν την αντικατάσταση σελίδων της κύριας μνήμης, δηλαδή που επιλέγουν ποια σελίδα πρέπει ανά πάσα στιγμή να αντικατασταθεί. Αξίζει να σημειωθεί ότι η διαχείριση μνήμης δεν είναι απαραίτητη σε ένα σύστημα στο οποίο δεν υπάρχει εκ περιτροπής εκτέλεσης ή, τουλάχιστον, πιθανότητα προεκχώρησης κατά την εκτέλεση.

2.4.1 Μονάδα διαχείρισης μνήμης

Σε κάθε υπολογιστικό σύστημα υπάρχει μια μονάδα διαχείρισης μνήμης (Memory Management Unit – MMU), η οποία αναλαμβάνει την ανάθεση και τη χρήση σελίδων μνήμης από τον επεξεργαστή. Η μονάδα αυτή μπορεί να βρίσκεται είτε στον επεξεργαστή είτε στη μητρική κάρτα. Ακολουθεί μια απλουστευμένη περιγραφή της λειτουργίας της μονάδας διαχείρισης μνήμης.

Ο λόγος που υλοποιείται μια τέτοια μονάδα είναι για να μπορεί ο επεξεργαστής να χρησιμοποιεί σελίδες χωρίς να χρειάζεται να γνωρίζει εάν αυτές είναι φορτωμένες στη μνήμη ή όχι. Η μνήμη χωρίζεται σε κύρια μνήμη (Main Memory) και εικονική μνήμη (Virtual Memory). Και στα δύο ήδη μνήμης αποθηκεύονται δεδομένα με τη μορφή σελίδων. Δηλαδή, ένα πρόγραμμα χωρίζεται σε σελίδες σταθερού μεγέθους (συχνά 4096 kB ή 8192 kB) και κατά την εκτέλεση της κάθε διεργασίες φορτώνονται στη μνήμη οι κατάλληλες σελίδες. Η χρήση της εικονικής μνήμης είναι η παρακάτω. Σε περίπτωση που η κύρια μνήμη είναι γεμάτη με σελίδες από τις διεργασίες που εκτελούνται και πρέπει μια νέα σελίδα να φορτωθεί, πρέπει, νωρίτερα, να απομακρυνθεί μια – τουλάχιστον – σελίδα από την κύρια μνήμη. Η μονάδα διαχείρισης μνήμης αναλαμβάνει να επιλέξει τη σελίδα αυτή, σύμφωνα με κάποιον από τους αλγορίθμους που θα συζητηθούν αργότερα, την οποία μεταφέρει στην εικονική μνήμη. Με αυτόν τον τρόπο όταν ξαναζητηθεί η σελίδα αυτή, η σελίδα θα ξαναφορτωθεί στην κύρια μνήμη, με μια διαδικασία που δεν γίνεται αντιληπτή από τον επεξεργαστή. Η διαδικασία της

μεταφοράς της σελίδας από την εικονική μνήμη προς την κύρια μπορεί να μην απαιτεί από τον επεξεργαστή κάποια ενέργεια, εισάγει, όμως, μεγάλη καθυστέρηση με αποτέλεσμα να προεκχωρείται η τρέχουσα διεργασία μέχρι να φορτωθεί η σελίδα ώστε, ενδιαμέσως, να εκτελεστεί μια άλλη διεργασία που ο κώδικάς της βρίσκεται φορτωμένος στην κύρια μνήμη. Η καθυστέρηση αυτή οφείλεται στο ότι η πρόσβαση δεδομένων που εδράζουν στην κύρια μνήμη είναι τρεις και πλέον τάξεις μεγέθους πιο γρήγορη από την πρόσβαση δεδομένων που εδράζουν στο σκληρό δίσκο.

Η διαδικασία κατά την οποία ο επεξεργαστής ζητά μια σελίδα που δε βρίσκεται στην κύρια μνήμη, η οποία οδηγεί σε παύση εκτέλεσης της τρέχουσας διεργασία μέχρι να φορτωθεί από την εικονική μνήμη η κατάλληλη σελίδα ονομάζεται Σφάλμα Σελίδας (Page Fault) και θα μας απασχολήσει ιδιαίτερα στη συνέχεια της εργασίας.

Η λειτουργία της μονάδας διαχείρισης μνήμης που περιγράφηκε δημιουργεί την ανάγκη για τον ορισμό δύο διευθύνσεων ανά σελίδα. Τη φυσική διεύθυνση και τη λογική διεύθυνση. Η φυσική διεύθυνση (physical address) είναι η θέση στο δίσκο στην οποία είναι αποθηκευμένη η σελίδα που βρίσκεται στην εικονική μνήμη. Η λογική διεύθυνση (logical address) είναι η διεύθυνση με την οποία ο επεξεργαστής καλεί την κάθε σελίδα. Η διεύθυνση αυτή συνήθως αποτελείται από κάποιο χαρακτηριστικό της διεργασίας στην οποία ανήκει η σελίδα καθώς και τη θέση της σελίδας στον κώδικα της διεργασίας αυτής. Προκειμένου να μην απαιτείται κάποια ειδική ενέργεια από τον επεξεργαστή, η μονάδα διαχείρισης μνήμης διατηρεί μια δομή στην οποία βρίσκονται οι αντιστοιχίσεις μεταξύ των δύο αυτών διευθύνσεων. Η δομή αυτή καλείται Δομή Αντιστοίχησης Διεργασιών Μνήμης (Process Memory Map).

2.4.2 Αναλυτική περιγραφή της διαχείριση μνήμης

Στην παρούσα ενότητα περιγράφεται με πιο μεγάλη λεπτομέρεια η διαχείριση μνήμης, προσπαθώντας να καλυφθούν οι διάφορες επιλογές που μπορεί να συναντήσει κάποιος σε ένα πραγματικό σύστημα. Η διαχείριση μνήμη αποτελεί, ίσως, το πιο περίπλοκο και δύσκολο τμήμα ενός λειτουργικού συστήματος. Είδαμε νωρίτερα διάφορες πολιτικές χρονοπρογραμματισμού διεργασιών, που εν τέλει καταλήγουν σε σχετικά απλά σχήματα. Αντίθετα, η αποτελεσματική διαχείριση μνήμης απαιτεί πολύπλοκες δομές και αποδοτικούς αλγορίθμους προκειμένου να μη βάζει εμπόδια στην ομαλή εκτέλεση διεργασιών.

Βασικές έννοιες της διαχείρισης μνήμης είναι η μετατόπιση (relocation), η προστασία (protection), ο διαμοιρασμός (sharing) καθώς και η λογική και η φυσική οργάνωση (logical and physical organization).

Μετατόπιση

Η μετατόπιση είναι απαραίτητη όταν πρόκειται για ένα σύστημα με πολλούς επεξεργαστές. Συχνά θέλουμε να έχουμε ένα σύστημα με όσο το δυνατόν πιο πολλές διεργασίες έτοιμες προς εκτέλεσης ώστε να μη μένει ποτέ ο επεξεργαστής αδρανής. Εάν απομακρυνθεί από την κύρια μνήμη ο κώδικας μιας διεργασίας, δεν μπορεί η διεργασία αυτή να είναι έτοιμη προς εκτέλεση άμεσα. Αντίθετα, μπορεί να είναι προτιμότερο να μετατοπιστεί ο κώδικας σε άλλο τμήμα της κύριας μνήμης.

Προστασία

Κάθε διεργασία πρέπει να προστατεύεται από ανεπιθύμητες παρεμβάσεις από άλλες διεργασίες, είτε είναι εσκεμμένες είτε όχι. Αυτό απαιτεί ένα μηχανισμό με τον οποίο απαγορεύεται σε μια διεργασία να προσπελάσει διευθύνσεις μνήμης που ανήκουν σε άλλες διεργασίες. Γενικά, αυτός είναι ο κανόνας για κάθε διεργασία χρήστη. Επιπλέον, προκειμένου αυτή η διαδικασία να μπορεί να γίνει αποδοτικά είναι σημαντικό να υπάρχει υποστήριξη για προστασία σε επίπεδο υλικού (επεξεργαστής) παρά σε επίπεδο λογισμικού (λειτουργικό σύστημα). Το ευτύχημα είναι ότι οι διαδικασίες που υλοποιούν την μετατόπιση υποστηρίζουν και τις απαιτήσεις προστασίας.

Διαμοιρασμός

Μία έννοια συνδεδεμένη με την προστασία είναι ο διαμοιρασμός. Κάθε σύστημα διαχείρισης μνήμης πρέπει να επιτρέπει την πρόσβαση σε ίδιο τμήμα της μνήμης από διαφορετικές διεργασίες εάν κάτι τέτοιο είναι σκόπιμο. Για παράδειγμα, εάν δυο δύο διεργασίες εκτελέσουν το ίδιο πρόγραμμα, εάν επιτρέψουμε και στις δύο να έχουν πρόσβαση στον ίδιο κώδικα, εξοικονομούμε μνήμη. Επιπλέον, συχνά δύο διεργασίες μπορεί να έχουν κοινές δομές για αποθήκευση δεδομένων και επικοινωνία.

Λογική οργάνωση

Νωρίτερα έγινε αναφορά στις λογικές διευθύνσεις. Ο κώδικας των προγραμμάτων πρέπει να οργανωθεί λογικά σύμφωνα με τον τρόπο με τον οποίο ο επεξεργαστής μπορεί να καλέσει ένα πρόγραμμα, μία ρουτίνα ή ακόμα και δομές δεδομένων. Η καλή λογική οργάνωση επιτρέπει να γίνεται άμεσα προσπέλαση από μια διεργασία σε δεδομένα άλλη διεργασίας (εάν αυτό είναι επιτρεπτό) χωρίς να χρειάζεται πληροφορία

για το που είναι αυτά αποθηκευμένα. Επιπλέον, μια λογική και ιεραρχική οργάνωση των δεδομένων επιτρέπει την ύπαρξη πιο σύνθετων μορφών προστασίας.

Φυσική οργάνωση

Μία βασική υπόθεση εργασίας όταν αναφερόμαστε στην κύρια μνήμη ενός συστήματος είναι ότι είναι πιθανό να μην επαρκεί για όλα τα προγράμματα που μπορεί να είναι έτοιμα προς εκτέλεση ανά πάσα στιγμή. Ο προγραμματιστής, επομένως, δε γνωρίζει εάν θα υπάρχει αρκετός χώρος για τον κώδικα της διεργασίας του. Άρα, είναι σημαντικό να είναι εφικτό να σπάσει ο κώδικας σε μικρότερα τμήματα (τμηματοποίηση μνήμης – memory partitioning). Για να υποστηρίζεται κάτι τέτοιο πρέπει να υπάρχει ένας μηχανισμός ο οποίος αποθηκεύει τις πληροφορίες για τη φυσική θέση κάθε τμήματος του κώδικα μιας διεργασίας.

Ένα σημαντικό χαρακτηριστικό της διαδικασίας διαχείρισης μνήμης, λοιπόν, είναι η τμηματοποίηση της μνήμης. Αρχικά, σε κάθε υπολογιστικό σύστημα υπάρχει ένα τμήμα της μνήμης το οποίο είναι κατειλημμένο από το λειτουργικό σύστημα. Το υπόλοιπο τμήμα της μνήμης είναι διαθέσιμο στις διάφορες διεργασίες. Πρέπει, όμως, να καθοριστεί ένας τρόπος με τον οποίο κάθε διεργασία επιλέγει σε ποιο σημείο της μνήμης και πόσο χώρο μπορεί να δεσμεύσει. Αυτό μπορεί να γίνει έξι βασικούς τρόπους.

1. Σταθερή Τμηματοποίηση (Fixed Partitioning)
2. Δυναμική Τμηματοποίηση (Dynamic Partitioning)
3. Απλή Σελιδοποίηση (Simple Paging)
4. Απλή Κατάτμηση (Simple Segmentation)
5. Σελιδοποίηση Εικονικής Μνήμης (Virtual-Memory Paging)
6. Κατάτμηση Εικονικής Μνήμης (Virtual-Memory Segmentation)

Οι πρώτοι τέσσερις τρόποι υποθέτουν την απουσία εικονικής μνήμης ενώ οι δύο τελευταίοι προϋποθέτουν την ύπαρξη εικονικής μνήμης.

Σταθερή Τμηματοποίηση

Σύμφωνα με την τεχνική αυτή, η κύρια μνήμη χωρίζεται σε στατικά τμήματα κατά την έναρξη λειτουργίας του συστήματος. Κάθε διεργασία καταλαμβάνει στη μνήμη χώρο ίσο με ένα τέτοιο τμήμα ή περισσότερα. Το μέγεθος του κάθε τμήματος μπορεί να είναι σταθερό και ίσο για κάθε τμήμα ή, εναλλακτικά, μπορεί τμήματα με αυξανόμενο μέγεθος, έτσι ώστε να παρέχεται μια ευκολία πρόσβασης για μεγαλύτερες διεργασίες.

Το σχήμα αυτό αν και είναι πολύ απλό στην υλοποίηση και δεν επιβαρύνει το λειτουργικό σύστημα σε πολυπλοκότητα, είναι αναποτελεσματικό σε χρησιμοποίηση μνήμης (memory utilization) γιατί, αφενός, υπάρχει εσωτερικός κατακερματισμός (internal fragmentation) και, αφετέρου, υπάρχει ένα δεδομένο μέγιστο πλήθος διεργασιών (μία για κάθε τμήμα).

Ο *εσωτερικός κατακερματισμός* είναι το φαινόμενο κατά το οποίο μία διεργασία έχει δεσμεύσει χώρο στη μνήμη αλλά δεν τον χρησιμοποιεί όλο. Σε ένα σχήμα όπως αυτό της σταθερής τμηματοποίησης κάτι τέτοιο είναι πολύ πιθανό.

Δυναμική Τμηματοποίηση

Όταν εφαρμόζεται δυναμική τμηματοποίηση, κάθε διεργασία καταλαμβάνει ακριβώς όσο χώρο χρειάζεται αποφεύγοντας έτσι εντελώς τον εσωτερικό κατακερματισμό. Η χρήση της μνήμης είναι πολύ αποδοτική, όμως, έχουμε μη ικανοποιητική χρήση του επεξεργαστή γιατί προκειμένου να αποφεύγεται ο εξωτερικός κατακερματισμός (external fragmentation) γίνεται συμπίεση (compaction).

Ο *εξωτερικός κατακερματισμός* είναι το φαινόμενο κατά το οποίο η δέσμευση της μνήμης για κάθε διεργασία οδηγεί σε κενά (συνήθως μικρά) μεταξύ των τμημάτων μνήμης που αντιστοιχούν ανά διεργασία με αποτέλεσμα σε αυτά τα κενά να μη χωράει μια ολόκληρη διεργασία.

Για να αποφευχθεί ο εξωτερικός κατακερματισμός, όπως προαναφέρθηκε, γίνεται συμπίεση. Η *συμπίεση* είναι μια διαδικασία κατά την οποία μεταφέρεται ο κώδικας της κάθε διεργασίας ώστε όλες οι διεργασίες να καταλαμβάνουν συνεχόμενο κομμάτια μνήμης. Αυτή η διαδικασία είναι αρκετά “ακριβή” σε πόρους και, άρα, καθιστά την τεχνική αυτή μη αποδοτική.

Τόσο στην απλή τμηματοποίηση όσο και στη δυναμική τμηματοποίηση υπάρχουν αλγόριθμοι τοποθέτησης διεργασίας οι οποίοι παίζουν πολύ σημαντικό ρόλο στην απόδοση των τεχνικών αυτών.

Απλή Σελιδοποίηση

Στην απλή σελιδοποίηση, η μνήμη χωρίζεται σε ίσου μεγέθους πλαίσια. Κάθε διεργασία χωρίζεται σε σελίδες με μέγεθος ίσο με ένα πλαίσιο. Όταν μια διεργασία πρέπει να φορτωθεί στη μνήμη φορτώνονται όλες οι σελίδες, όχι, όμως, υποχρεωτικά σε συνεχόμενα πλαίσια. Με αυτήν την τεχνική αποφεύγουμε εντελώς των εξωτερικό κατακερματισμό, αλλά υπάρχει ένα μικρό ποσό εσωτερικού κατακερματισμού.

Απλή κατάτμηση

Σύμφωνα με την τεχνική αυτή, κάθε διεργασία χωρίζεται σε ένα πλήθος από τμήματα, τα οποία έχουν δυναμικό μέγεθος και δε χρειάζεται να είναι συνεχόμενα. Για να τρέξει η διεργασία φορτώνονται όλα τα τμήματα αυτά. Έτσι, αποφεύγεται ο εσωτερικός κατακερματισμός, βελτιώνεται η χρησιμοποίηση της μνήμης και μειώνεται η επιβάρυνση σε σχέση με τη δυναμική τμηματοποίηση.

Σελιδοποίηση Εικονικής Μνήμης

Η τεχνική αυτή είναι μια επέκταση της σελιδοποίησης. Πιο συγκεκριμένα, δεν είναι απαραίτητο να φορτωθούν όλες οι σελίδες μιας διεργασίας για να αρχίσει να εκτελείται. Οι σελίδες που θα ζητηθούν θα φορτωθούν στη συνέχεια αυτόματα. Η τεχνική αυτή εξαλείφει τον εξωτερικό κατακερματισμό, επιτρέπει αυξημένο επίπεδο πολυπρογραμματισμού και παρέχει ένα μεγάλο εύρος εικονικού χώρου διευθύνσεων. Εισάγει , βέβαια, την ανάγκη για πιο σύνθετες δομές και μονάδες διαχείρισης μνήμης, αφού εισάγει την έννοια της εικονικής μνήμης. Ο μηχανισμός που φροντίζει να είναι προσπελάσιμες οι σελίδες που είναι στην εικονική μνήμη, και άρα στο δίσκο, έχει σημαντική πολυπλοκότητα αλλά τα προαναφερθέντα οφέλη κρίνονται συνήθως πιο σημαντικά.

Κατάτμηση εικονικής μνήμης

Η τεχνική αυτή είναι μια επέκταση της απλής κατάτμησης. Η διαφορά, είναι και πάλι, ότι δε χρειάζεται να φορτωθούν όλα τα τμήματα της διεργασίας. Σε αντίθεση με την σελιδοποίηση εικονικής μνήμης, εξαλείφεται ο εσωτερικός κατακερματισμός ενώ και πάλι επιτρέπεται αυξημένο επίπεδο πολυπρογραμματισμού και παρέχεται ένα μεγάλο εύρος εικονικού χώρου διευθύνσεων, ενώ υποστηρίζεται τόσο η προστασία όσο και ο διαμοιρασμός διεργασιών.

Στις τεχνικές εικονικής μνήμης κάνουν την εμφάνισή τους αλγόριθμοι αντικατάστασης σελίδων (ή τμημάτων). Οι αλγόριθμοι αυτοί παίζουν πολύ σημαντικό ρόλο στην απόδοση του συστήματος αφού καθορίζουν ποιες διεργασίες μπορούν ανά πάσα στιγμή να εκτελεστούν και ποιες πρέπει να περιμένουν τις σελίδες τους από την εικονική μνήμη.

Στις τεχνικές σελιδοποίησης πρέπει το σύστημα να συντηρεί πληροφορίες σχετικά με το ποιο πλαίσιο αντιστοιχεί σε ποια σελίδα. Οι πληροφορίες αυτές κρατούνται σε κάποιον πίνακα σελίδων (page table). Τέτοιοι πίνακες υπάρχουν για κάθε διεργασία. Η μονάδα

διαχείρισης μνήμης αναλαμβάνει να χειρίζεται και να φυλάσσει τους πίνακες αυτούς. Μια αντίστοιχη δομή πρέπει να φυλάσσεται και στα συστήματα που υποστηρίζουν κατάτμηση.

2.4.2.1 Εικονική μνήμη

Έχοντας περιγράψει όλες τις βασικές έννοιες για τη διαχείριση μνήμης μπορούμε να περάσουμε στη λεπτομερή περιγραφή της εικονικής μνήμης (virtual memory). Όπως ήδη αναφέρεται η εικονική μνήμη είναι μια επέκταση της κύριας μνήμης που εδράζεται στο σκληρό δίσκο και αποσκοπεί στο να καθιστά την κύρια μνήμη δυνάμει άπειρη. Θεωρούμε, γενικά, το μέγεθος της εικονικής μνήμης αρκετά μεγάλο ώστε να μη μας απασχολεί το ενδεχόμενο του να γεμίσει με προγράμματα. Η δημιουργία της εικονικής μνήμης βασίζεται στις δύο αρχές που περιγράφονται παρακάτω.

Κάθε διεύθυνση μνήμης μέσα σε ένα πρόγραμμα εκφράζεται με λογικό τρόπο και υπάρχει πάντα ένας αλγόριθμος για να αντιστοιχηθεί με τη φυσική διεύθυνση της εν λόγω πληροφορίας. Συνεπώς δεν είναι απαραίτητο να είναι πάντα στο ίδιο σημείο της μνήμης τα δεδομένα ενός προγράμματος. Μπορούν να ξεφορτώνονται από τη μνήμη και να φορτώνονται πάλι όταν το πρόγραμμα είναι έτοιμο για εκτέλεση, αρκεί να είναι εφικτή η αντιστοίχηση της λογικής με τη φυσική διεύθυνση. Επίσης, από τη στιγμή που είναι εφικτό να διαμεριστεί ένα πρόγραμμα σε υποτμήματα (είτε σε σελίδες είτε σε τμήματα), σε συνδυασμό με τη προαναφερθείσα δυνατότητα μετατροπής της λογικής διεύθυνσης σε φυσική, δεν είναι απαραίτητο να είναι οι σελίδες (ή τα τμήματα) μιας διεργασίας συνεχόμενα.

Οι δύο αυτές παρατηρήσεις μας οδηγούν στο συμπέρασμα ότι δεν είναι απαραίτητο να είναι συνεχώς φορτωμένος στη μνήμη ο κώδικας μιας διεργασίας στο σύνολό του. Έτσι, μπορούμε να μεταφέρουμε ανά πάσα στιγμή ένα υποσύνολο του κώδικα σε ένα άλλο κομμάτι μνήμης, το οποίο είναι η εικονική μνήμη, επιτρέποντας τη φόρτωση περισσότερων διεργασιών προς εκτέλεσης ταυτόχρονα. Η εικονική μνήμη, δηλαδή, ενισχύει το επίπεδο πολύ-προγραμματισμού αλλά και αυξάνοντας το ποσοστό χρησιμοποίησης του επεξεργαστή (αφού, έχοντας πιο πολλές διεργασίες αυξάνεται η πιθανότητα ανά χρονική στιγμή κάποια από αυτές να είναι έτοιμη προς εκτέλεση). Επιπλέον, με χρήση εικονική μνήμης είναι, πια, εφικτό, ένα πρόγραμμα να έχει πιο μεγάλο μέγεθος από το μέγεθος της κύριας μνήμης.

Πριν προχωρήσουμε στην περιγραφή των σχημάτων διαχείρισης μνήμης σε πραγματικά λειτουργικά συστήματα πρέπει να περιγράψουμε μερικές βασικές έννοιες

της εικονικής μνήμης. Σύμφωνα με την *αρχή της τοπικότητας* (principle of locality) οι αναφορές μέσα σε ένα πρόγραμμα έχουν την τάση να ομαδοποιούνται σε γειτονικές περιοχές. Επίσης, σπάνια σε ένα πρόγραμμα οι αναφορές αφορούν όλο το εύρος του προγράμματος αυτού. Συνεπώς, οδηγούμαστε στο συμπέρασμα ότι η εικονική μνήμη είναι όντως, αποτελεσματική. Για την υλοποίηση ενός σχήματος εικονικής μνήμης χρειάζεται το κατάλληλο υλικό ώστε να υποστηρίζεται η σελιδοποίηση (ή η κατάτμηση), καθώς και το κατάλληλο λογισμικό το οποίο θα οργανώνει τη μετακίνηση σελίδων μεταξύ της κύριας και της δευτερεύουσας μνήμης. Ένα σημαντικό συστατικό ακόμα, είναι ο πίνακας σελίδων (page table). Ο πίνακας αυτός, που συνήθως υπάρχει ένας ανά διεργασία, διατηρεί πληροφορίες σχετικά με τη μετατροπή της λογικής σε φυσική διεύθυνση, αλλά και με το εάν μια σελίδα είναι φορτωμένη στην κύρια μνήμη ή στην εικονική μνήμη. Σε συστήματα που η εικονική μνήμη μπορεί να έχει μεγάλο μέγεθος, η διατήρηση ενός πίνακα σελίδων ανά διεργασία χάνει την πρακτικότητά του, αφού χρειάζονται πολλαπλά αντίγραφα μεγάλων πινάκων που καταλαμβάνουν πολύ χώρο. Για να επιλυθεί αυτό το πρόβλημα πολλές φορές οι πίνακες σελίδων οργανώνονται σε μια διεπίπεδη ιεραρχία που αποθηκεύεται στην εικονική μνήμη και άρα μπορεί ανά πάσα στιγμή να απομακρυνθεί από την κύρια μνήμη. Επίσης, σε μερικά συστήματα χρησιμοποιούνται οι ανεστραμμένοι πίνακες σελίδων (inverted page table), στους οποίους χρησιμοποιείται ένα σύστημα κατακερματισμού (hashing) για να διατηρηθεί το απαιτούμενο μέγεθος σταθερό και να μην είναι απαραίτητη η αποθήκευση των πινάκων αυτών στην εικονική μνήμη.

Μια ακόμη παράμετρος που παίζει πολύ σημαντικό ρόλο είναι το μέγεθος της κάθε σελίδας. Σε περίπτωση που επιλεγεί μεγάλο μέγεθος, τότε μπορεί να έχουμε σημαντικό εσωτερικό κατακερματισμό, κάτι που θέλουμε να αποφύγουμε. Όμως, εάν το μέγεθος της κάθε σελίδας είναι αρκετά μικρό, τότε, κάθε διεργασία θα απαιτεί μεγάλο αριθμό σελίδων με αποτέλεσμα να δημιουργούνται μεγαλύτεροι πίνακες σελίδων. Επιπλέον, σημαντικό ρόλο παίζει η συσχέτιση μεταξύ του μεγέθους της σελίδας και του λόγου σφαλμάτων σελίδων, καθώς και η συσχέτιση μεταξύ του πλήθους πλαισίων που δεσμεύονται για κάθε διεργασία και του λόγου σφαλμάτων σελίδων. Το σύνολο των παρατηρήσεων δεν οδηγεί σε ένα καθαρό συμπέρασμα και έτσι οι κατασκευαστές επιλέγουν μέγεθος σελίδας ανάλογα με την υφή των εφαρμογών για τις οποίες προορίζεται ένα σύστημα. Έτσι, σε συστήματα του εμπορίου μπορεί να συναντήσει κανείς μέγεθος σελίδας από 512 48-bit λέξεις έως 256 MB. Ένα πιο συγκεκριμένο

παράδειγμα είναι πως σε επεξεργαστές Pentium μπορεί να συναντήσει κάποιος μέγεθος σελίδας από 4 kB έως 4 MB.

Σχεδιαστικές επιλογές εικονικής μνήμης

Κατά το σχεδιασμό της εικονικής μνήμης ενός συστήματος πρέπει να ληφθούν αποφάσεις στους παρακάτω βασικούς άξονες:

- Πολιτική φόρτωσης σελίδων (Fetch policy)
- Πολιτική τοποθέτησης σελίδων (Placement policy)
- Πολιτική αντικατάστασης σελίδων (Replacement policy)
- Διαχείριση φορτωμένων σελίδων (Resident Set management)
- Πολιτική εκκαθάρισης (Cleaning policy)
- Έλεγχος φόρτου (Load control)

Η φόρτωση σελίδων υλοποιείται με δύο διαφορετικούς τρόπους. Σύμφωνα με τη μία επιλογή κάθε φορά που ένα πρόγραμμα ετοιμάζεται προς εκτέλεση, φορτώνονται στη μνήμη μόνο οι σελίδες που αναφέρονται και οι επόμενες σελίδες θα φορτωθούν μόνο εάν γίνει κάποια αναφορά στο περιεχόμενό τους. Αυτή η πολιτική ονομάζεται φόρτωση σελίδων κατ' απαίτηση (demand paging). Αντίθετα, σύμφωνα με την άλλη πολιτική κάθε φορά που φορτώνεται ένα πρόγραμμα φορτώνονται όλες οι σελίδες του στη μνήμη. Η πολιτική αυτή ονομάζεται προληπτική φόρτωση σελίδων (prepaging). Η πρώτη πολιτική ενισχύει το επίπεδο πολυπρογραμματισμού αλλά μπορεί να οδηγήσει σε περισσότερα σφάλματα σελίδων, ειδικά κατά την αρχική φάση εκτέλεσης του προγράμματος, μέχρι δηλαδή να φορτωθεί το σύνολο εργασίας (working set) της τρέχουσας διεργασίας. Αντίθετα, η δεύτερη πολιτική μηδενίζει την πιθανότητα να συμβεί σφάλμα σελίδας πέραν του αρχικού, αλλά, έτσι γεμίζει η μνήμη πιο γρήγορα και μειώνεται το υποστηριζόμενο επίπεδο πολύ-προγραμματισμού.

Η πολιτική τοποθέτησης σελίδων είναι πολύ σημαντική σε μια κατηγορία συστημάτων και, αντίθετα, λιγότερο σημαντική, σε μια άλλη κατηγορία. Πιο συγκεκριμένα, σε συστήματα που κάνουν κατάτμηση (και άρα κάθε τμήμα δεν έχει σταθερό μέγεθος), η επιλογή αλγορίθμου τοποθέτησης παίζει σημαντικό ρόλο. Κλασσικοί αλγόριθμοι υλοποιούν για παράδειγμα καλύτερο ταίριασμα, πρώτο ταίριασμα, κ.α.. Σε συστήματα που κάνουν σελιδοποίηση ή ακολουθούν ένα υβριδικό μοντέλο σελιδοποίησης και κατάτμησης η απόφαση αυτή δεν παίζει κανένα ρόλο. Αντίθετα σε ένα σύστημα με πολλούς επεξεργαστές και κοινή κατανεμημένη μνήμη η απόδοση μπορεί να αλλάξει ραγδαία από τη σωστή η μη επιλογή αλγορίθμου τοποθέτησης, αφού ένας

επεξεργαστής μπορεί να προσπελάσει ανά πάσα στιγμή όλη τη μνήμη ως κύρια μνήμη, ενώ μόνο ένα υποσύνολο αυτής είναι κύρια μνήμη του δεδομένου επεξεργαστή σε επίπεδο υλικό, στο οποίο είναι προτιμότερο να εδράζουν τα προγράμματα που τρέχουν στον τρέχοντα επεξεργαστή.

Η αντικατάσταση σελίδων είναι ένα πολύ σημαντικό τμήμα της διαχείρισης μνήμης. Υλοποιείται ακολουθώντας κάποιον ή κάποιους από μια ομάδα αλγορίθμων που θα παρουσιαστούν σε ξεχωριστή ενότητα. Η πολιτική αντικατάσταση σελίδων και η πολιτική διαχείρισης φορτωμένων σελίδων έχουν συγκεχυμένα όρια. Αυτό συμβαίνει γιατί κατά την απόφαση για αντικατάσταση μιας σελίδας, όταν μια νέα σελίδα πρόκειται να φορτωθεί στη μνήμη, πρέπει να αποφασιστεί:

- πόσα πλαίσια θα δεσμευτούν για κάθε ενεργή διεργασία,
- εάν το σύνολο των σελίδων που περιέχει σελίδες που μπορούν να αντικατασταθούν πρέπει να περιοριστεί στις σελίδες της διεργασία που προκάλεσε το σφάλμα σελίδας ή εάν πρέπει να περιέχει όλες τις σελίδες, και, τελικά,
- από το σύνολο αυτό, ποια σελίδα θα αντικατασταθεί τελικά.

Τα δύο πρώτα ερωτήματα αναφέρονται στη διαδικασία διαχείρισης των φορτωμένων σελίδων και, έτσι, μόνο η τρίτη έννοια αναφέρεται ως πολιτική αντικατάστασης σελίδων. Σε επόμενη ενότητα θα αναφερθούμε στη βέλτιστη (optimal) πολιτική, στην πολιτική αντικατάστασης τις λιγότερο πρόσφατα χρησιμοποιημένης σελίδας (Least Recently Used), στην πολιτική εξυπηρέτησης σύμφωνα με τη σειρά άφιξης (First-in-first-out ή First-come-first-serve), στην πολιτική ρολογιού (clock) καθώς και σε μερικές άλλες, λιγότερο σημαντικές πολιτικές.

Η διαχείριση των σελίδων που βρίσκονται στην κύρια μνήμη συνίσταται στην επιλογή μία από τις δύο βασικές πολιτικές δέσμευσης μνήμης. Η σταθερή δέσμευση σελίδων (fixed-allocation) αντιστοιχεί σε κάθε διεργασία ένα σταθερά πλήθος σελίδων στη μνήμη, το οποίο υπολογίζεται κατά τη δημιουργία της διεργασίας στη μνήμη (process creation time), και άρα μπορεί να είναι διαφορετικό ανά διεργασία. Η ακριβής τιμή του πλήθος σελίδων, ή πιο σωστά πλαισίων, ανά διεργασία εξαρτάται από τις ιδιότητες της διεργασίας αυτής (διαδραστικότητα, είδος εφαρμογής, κ.α.) αλλά μπορεί να βασίζεται σε καθοδήγηση από τον προγραμματιστή του λειτουργικού ή το διαχειριστή του συστήματος. Μια διεργασία που χρησιμοποιεί όλα τα διαθέσιμα πλαίσια και έχει ανάγκη να φορτώσει μια νέα σελίδα αναγκάζεται να απομακρύνει μία από τις δικές της σελίδες

στην εικονική μνήμη. Αντίθετα, στη μεταβλητή δέσμευση σελίδων (variable-allocation) το πλήθος σελίδων που μπορεί κάθε διεργασία να έχει ταυτόχρονα στην κύρια μνήμη είναι μεταβλητό. Σε ένα ιδεατό σύστημα, σε μια διεργασία που έχει μεγάλο λόγο σφαλμάτων σελίδων θα δοθεί περιθώριο να φέρει πολλές σελίδες στη μνήμη προκειμένου να φορτώσει όλο το σύνολο εργασίας (working set) στη μνήμη, ενώ σε μία διεργασία που έχει μικρό λόγο σφαλμάτων σελίδων, και άρα η αρχή της τοπικότητας είναι ισχυρή, θα έχει στη διάθεσή της μικρότερο αριθμό πλαισίων.

Όταν έχουμε μεταβλητή δέσμευση, τίθεται ένα ακόμα ερώτημα. Σε περίπτωση που έχουμε ανάγκη για αντικατάσταση σελίδας, από ποιο σύνολο σελίδων θα γίνει αυτή. Αν και αυτό το ερώτημα άπτεται περισσότερο στο θέμα της αντικατάστασης σελίδων είναι αλληλένδετο με την επιλογή της διαχείρισης φορτωμένων σελίδων. Οι επιλογές είναι δύο: πολιτική τοπικής αντικατάστασης (local replacement policy) και πολιτική καθολικής αντικατάστασης (global replacement policy). Σύμφωνα με την πρώτη πολιτική αντικαθίσταται σελίδα της διεργασίας που προκάλεσε το σφάλμα σελίδας, ενώ, σύμφωνα με τη δεύτερη πολιτική όλες οι σελίδες που δεν είναι κλειδωμένες είναι υποψήφιες για αντικατάσταση. Οι αλγόριθμοι που θα περιγραφούν μπορεί να χρησιμοποιηθούν πάνω σε όποιο από τα δύο σύνολα επιλέξει ο προγραμματιστής του λειτουργικού, δηλαδή, η επιλογή της πολιτικής αντικατάστασης δεν αλλάζει τη δομή των αλγορίθμων αντικατάστασης σελίδων.

Προκειμένου να υπάρχει ανά πάσα στιγμή ελεύθερος χώρος στη μνήμη εφαρμόζεται εκκαθάριση στη μνήμη σύμφωνα με κάποιες πολιτικές. Δύο συνήθεις εναλλακτικές είναι η εκκαθάριση κατ' απαίτηση (demand cleaning) και η προεκκαθάριση (precleaning). Σύμφωνα με την πρώτη πολιτική μια σελίδα απομακρύνεται από την κύρια μνήμη και εγγράφεται στη δευτερεύουσα μόνο σε περίπτωση που τοποθετηθεί προς αντικατάσταση σύμφωνα με τους αλγορίθμους αντικατάστασης μετά από ένα σφάλμα σελίδας. Αντίθετα, η πολιτική προεκκαθάρισης, εγγράφει τις τροποποιημένες σελίδες στη δευτερεύουσα μνήμη πριν ζητηθούν ώστε να μπορούν να εγγραφούν πολλές μαζί. Το γεγονός ότι ακολουθώντας την πολιτική αυτή μπορεί να γράψεις πολλές σελίδες στο δίσκο και να τις αφήσεις στη μνήμη και μετά από λίγο να χρειαστεί να τις ξαναγράψεις καθιστά προβληματική την αντιμετώπιση αυτή. Αντίστοιχα, κατά την εκκαθάριση κατ' απαίτηση κάθε εισαγωγή νέας σελίδας με αντικατάσταση τροποποιημένης σελίδας πρέπει να περιμένει πρώτα την εγγραφή της τροποποιημένης σελίδας πριν να είναι διαθέσιμη η νέα σελίδα. Είναι φανερό ότι ένας συνδυασμός των δύο πολιτικών θα οδηγήσει σε καλύτερα αποτελέσματα. Ένα υβρίδιο που έχει προκύψει από τις δύο αυτές

πολιτικές λειτουργεί ως εξής. Δημιουργούνται δύο λίστες σελίδων, η λίστα τροποποιημένων σελίδων και η λίστα μη-τροποποιημένων σελίδων. Επίσης, χρησιμοποιείται προσωρινή αποθήκευση (buffering) των σελίδων που αντικαταστάθηκαν, οι οποίες μπορεί να αποθηκευτούν σε μία από τις δύο προαναφερθείσες λίστες. Οι τροποποιημένες σελίδες αντιγράφονται περιοδικά και τοποθετούνται στη λίστα μη-τροποποιημένων σελίδων. Από τη λίστα αυτή μια σελίδα είτε αφαιρείται όταν έχει αντιστοιχηθεί το πλαίσιο στο οποίο άνηκε σε άλλη διεργασία, είτε ανακτάται όταν ξαναζητηθεί άμεσα, από τη διεργασία στην οποία ανήκει, προς εκτέλεση.

Ο έλεγχος φόρτου στη διαχείριση μνήμης είναι μια διαδικασία που καθορίζει πόσες διαφορετικές διεργασίες θα παραμένουν ενεργές στην κύρια μνήμη. Όταν επιτρέπονται λίγες διεργασίες τότε είναι πιθανό σε μία χρονική στιγμή όλες οι διεργασίες να έχουν διακόψει τη λειτουργία τους αναμένοντας κάποιο γεγονός ή κάποια σελίδα, οπότε θα έχουμε μικρή χρησιμοποίηση του επεξεργαστή. Αντίθετα, όταν επιτρέπονται πολλές διεργασίες ταυτόχρονα στην κύρια μνήμη τότε μπορεί να μην υπάρχει αρκετός χώρος για το σύνολο εργασίας της κάθε διεργασίας με αποτέλεσμα να συμβαίνουν συχνά σφάλματα σελίδων και να οδηγούμαστε σε εξόντωση (page thrashing) του συστήματος, δηλαδή σε δραματική μείωση της χρησιμοποίησης του επεξεργαστή (process utilization collapse). Στη ενότητα ΚΕΦΑΛΑΙΟ 3 θα ασχοληθούμε με περισσότερες λεπτομέρειες πάνω στο φαινόμενο της εξόντωσης καθώς και σε διάφορες λύσεις που προτείνονται.

2.5 Αλγόριθμοι Αντικατάστασης Σελίδων

Στην ενότητα αυτή παρουσιάζονται οι βασικοί αλγόριθμοι αντικατάστασης σελίδων. Όπως είδαμε νωρίτερα οι αλγόριθμοι αυτοί μπορεί να εφαρμόζονται με τοπική ή καθολική πολιτική. Η απόφαση έχει να κάνει με το εκάστοτε σύστημα και το διαχειριστή του συστήματος.

2.5.1 Βέλτιστος αλγόριθμος (Optimal)

Ο βέλτιστος αλγόριθμος επιλέγει, σε ένα ιδανικό σύστημα, για αντικατάσταση τη σελίδα, η οποία θα ζητηθεί ξανά για προσπέλαση πιο μακριά στο μέλλον από οποιαδήποτε άλλη. Έχει αποδειχθεί […] ότι μια τέτοια πολιτική οδηγεί σε ελάχιστο αριθμό σφαλμάτων σελίδων. Όπως γίνεται εύκολα αντιληπτό αυτή η πολιτική δεν μπορεί να υλοποιηθεί σε πραγματικά συστήματα, αφού δεν είναι δυνατόν να γνωρίζουμε με βεβαιότητα τις μέλλουσες αναφορές σε σελίδες της κάθε διεργασίας. Σε ελεγχόμενο περιβάλλον προσομοίωσης μπορεί, όμως, να αποτελέσει ένα καλό κριτήριο σύγκρισης άλλων υλοποιήσιμων αλγορίθμων.

2.5.2 Least Recently Used (LRU)

Ο αλγόριθμος LRU αντικαθιστά τη σελίδα που δεν έχει χρησιμοποιηθεί (αναγνωστεί ή εγγραφεί) για το μεγαλύτερο διάστημα. Σύμφωνα με την αρχή της τοπικότητας αυτή μια τέτοια σελίδα συγκεντρώνει τη μικρότερη πιθανότητα να ζητηθεί στο άμεσο μέλλον. Όντως, λοιπόν, ο αλγόριθμος αυτός έχει παρόμοια αποτελέσματα με το βέλτιστο αλγόριθμο αλλά συναντά μερικές δυσκολίες κατά την υλοποίηση του. Δύο είναι οι βασικές παραλλαγές για την υλοποίηση του LRU. Είτε αποθηκεύεται μαζί με τη σελίδα μια χρονοσφραγίδα, κάθε φορά που γίνεται αναφορά στην εν λόγω σελίδα, και γίνεται μια αναζήτηση σύμφωνα με τους αποθηκευμένους χρόνους είτε δημιουργείται μια δομή στοίβας στην οποία αποθηκεύονται οι αναφορές στις σελίδες. Και οι δύο παραλλαγές στην υλοποίηση οδηγούν σε μεγάλο επιπλέον κόστος. Τελικά, το επιπλέον αυτό κόστος είναι συχνά προτιμότερο από έναν χειρότερο αλγόριθμο και, έτσι, ο αλγόριθμος LRU είναι και αυτός που υλοποιείται σε αρκετά συστήματα.

2.5.3 First-in-first-out (FIFO)

Ο αλγόριθμος FIFO είναι, ίσως, ο πιο απλός αλγόριθμος από όσον αφορά την υλοποίηση. Ο αλγόριθμος θεωρεί πως τα δεσμευμένα πλαίσια είναι τοποθετημένα σε μία κυκλική διάταξη (ρολόι) και οι σελίδες αφαιρούνται εκ περιτροπής. Δεδομένου μιας κατάλληλης δομής αποθήκευσης σελίδων το μόνο που χρειάζεται για την υλοποίηση αυτού του αλγορίθμου είναι ένας δείκτης που θα εναλλάσσεται κυκλικά στα δεσμευμένα πλαίσια, σύμφωνα με τον οποίο θα αφαιρείται μια σελίδα από την κύρια μνήμη όταν αυτό είναι αναγκαίο. Η λογική σύμφωνα με την οποία σχεδιάστηκε αυτός ο αλγόριθμος, εκτός από την απλότητά του, είναι η εξής: μια σελίδα που έχει έρθει στην κύρια μνήμη πιθανόν πια να μη χρειάζεται. Η λογική αυτή, όμως, είναι συχνά λάθος γιατί μπορεί μια σελίδα να ήρθε για πρώτη φορά στην κύρια μνήμη πριν από πολλές αναφορές, αλλά μπορεί να ξαναγίνονται αναφορές συνεχώς. Οι σελίδες, για τις οποίες συμβαίνει κάτι τέτοιο, θα απομακρύνονται και θα επαναφέρονται στην κύρια μνήμη συχνά.

2.5.4 Clock

Δεδομένου ότι ο αλγόριθμος LRU, αν και έχει απόδοση πολύ κοντά στο βέλτιστο αλγόριθμο, έχει μεγάλο κόστος υλοποίησης και ο αλγόριθμος FIFA, αν και έχει μικρό κόστος υλοποίησης, έχει κακή απόδοση, δημιουργήθηκε η ανάγκη για το σχεδιασμό ενός αλγορίθμου που θα συνδυάζει τα πλεονεκτήματα των δύο αλγορίθμων. Σχεδιάστηκε, λοιπόν, μια ομάδα αλγορίθμων που υιοθετούν την πολιτική ρολογιού (clock policy).

Η πιο απλή μορφή του αλγορίθμου αυτού απαιτεί ένα, μόλις, bit επιπλέον. Το bit αυτό, το οποίο ονομάζεται bit χρήσης, τίθεται ίσο με 1 όταν η σελίδα φορτώνεται στη μνήμη αρχικά και κάθε φορά που γίνεται αναφορά σε αυτήν, εάν δεν είναι ήδη ίσο με 1. Όταν καταστεί απαραίτητο να γίνει αντικατάσταση μιας σελίδας, τα πλαίσια που είναι υποψήφια προς αντικατάσταση (ανάλογα με την πολιτική διαχείρισης φορτωμένων σελίδων, πρόκειται για σελίδες της διεργασίας ή σελίδες από κάθε διεργασία), θεωρούνται οργανωμένα σε μια κυκλική δομή, την οποία ο αλγόριθμος διατρέχει με χρήση ενός δείκτη. Κάθε φορά που συναντά ένα πλαίσιο με τιμή 1 στο bit χρήσης, το θέτει ίσο με μηδέν και εξετάζει το επόμενο πλαίσιο. Εάν βρεθεί πλαίσιο με bit χρήση ίσο με 0 τότε το πλαίσιο αυτό αντικαθίσταται. Εάν όλα τα πλαίσια έχουν bit χρήσης ίσο με 1 τότε ο αλγόριθμος θα διατρέξει το σύνολο των πλαισίων και θα απομακρύνει την πρώτη

σελίδα, ενεργώντας όπως ο αλγόριθμος FIFO. Μια άλλη ονομασία του αλγορίθμου αυτού είναι, αλγόριθμος δεύτερης ευκαιρίας (second chance algorithm).

Μια άλλη παραλλαγή του αλγορίθμου του ρολογιού απαιτεί την προσθήκη 2 bits. Το bit χρήσης (**u**se bit) και το bit τροποποίησης (**m**odified bit). Οι συνδυασμοί που προκύπτουν είναι τέσσερις.

Πίνακας 1: Συνδυασμός τιμών use bit και modified bit

όχι πρόσφατη προσπέλαση, όχι τροποποίηση	u=0	m=0
πρόσφατη προσπέλαση, όχι τροποποίηση	u=1	m=0
όχι πρόσφατη προσπέλαση, τροποποίηση	u=0	m=1
πρόσφατη προσπέλαση, τροποποίηση	u=1	m=1

Η λειτουργία του αλγορίθμου που χρησιμοποιεί αυτήν την κατηγοριοποίηση χωρίζεται σε τρία βήματα.

1. Αναζητούμε το πρώτο πλαίσιο του οποίου τα δύο bit είναι ίσα με μηδέν (δηλαδή ούτε έχει γίνει αναφορά σε αυτό πρόσφατα ούτε είναι τροποποιημένο). Εάν βρεθεί, τότε αντικαθίσταται, εάν όχι, τότε, προχωράμε στο βήμα 2.
2. Αναζητούμε το πρώτο πλαίσιο για το οποίο ισχύει ότι u=0 και m=1 (δηλαδή δεν έχει χρησιμοποιηθεί πρόσφατα αλλά έχει τροποποιηθεί πρόσφατα). Εάν βρεθεί επιλέγεται προς αντικατάσταση. Επίσης, σε όσα πλαίσια εξετάζονται μέχρι να καταλήξουμε τίθεται το bit χρήσης ίσο με μηδέν (u=0).
3. Εάν αποτύχει το βήμα 2, τότε, ο δείκτης πρέπει να έχει επανέλθει στην αρχική του θέση και τα bit χρήσης είναι σε όλα τα πλαίσια ίσα με 0. Επαναλαμβάνουμε τα βήματα 1 και 2. Αυτή τη φορά σίγουρα θα βρεθεί ένα πλαίσιο, αφού έχουμε θέσει όλα τα bit χρήσης ίσα με 0.

Τα πλαίσια που αναζητά το πρώτο βήμα του αλγορίθμου είναι η πιο κατάλληλη ομάδα πλαισίων για να επιλέξουμε κάποιο προς αντικατάσταση. Αυτό ισχύει, γιατί οι σελίδες που βρίσκονται στα πλαίσια αυτά, όχι μόνο δεν έχουν προσπελαστεί πρόσφατα αλλά και δεν έχουν τροποποιηθεί, και άρα, πριν την απομάκρυνση δε χρειάζεται να επανεγγραφούν στη δευτερεύουσα μνήμη. Στο δεύτερο βήμα αναζητούνται πλαίσια, που δεν έχουν προσπελαστεί πρόσφατα και άρα, σύμφωνα με την αρχή της

τοπικότητας, δε θα χρειαστούν στο άμεσο μέλλον έστω κι αν χρειάζεται να γίνει ενημέρωση των αντιστοίχων σελίδων στη δευτερεύουσα μνήμη.

2.6 Διαχείριση μνήμης σε πραγματικά συστήματα

Σε αυτήν την ενότητα παρουσιάζονται οι μηχανισμοί διαχείρισης μνήμης σε πραγματικά συστήματα. Πιο συγκεκριμένα, αναλύονται οι μηχανισμοί στα λειτουργικά Unix, Linux και Windows, τα οποία είναι ιδιαιτέρως διαδεδομένα, τόσο για επαγγελματική και ακαδημαϊκή χρήση όσο και για οικιακή.

2.6.1 Διαχείριση μνήμης σε συστήματα Unix και Solaris

Το Unix είναι το λειτουργικό με τη μεγαλύτερη ιστορία από το προαναφερθέντα και έτσι, όπως είναι λογικό, οι μηχανισμοί διαχείρισης μνήμης έχουν αλλάξει με το πέρασμα του χρόνου. Οι πιο πρόσφατες εκδόσεις του Unix (και πιο ειδικά, του Solaris) χρησιμοποιούν εικονική μνήμη με σελιδοποίηση. Πιο ειδικά χρησιμοποιείται η σελιδοποίηση με εικονική μνήμη για τη διαχείριση μνήμης των διεργασιών χρηστών και ένα άλλο σχήμα για τη δέσμευση μνήμης για τον πυρήνα (kernel memory allocator).

Σελιδοποίηση σε Unix

Οι δομές δεδομένων που χρησιμοποιούνται για την υλοποίηση της σελιδοποίησης είναι οι παρακάτω.

- Page table, που υπάρχει ένας για κάθε διεργασία με μία εγγραφή για κάθε σελίδα της διεργασίας στην εικονική μνήμη.
- Disk block descriptor, για κάθε σελίδα μιας διεργασίας υπάρχει μια εγγραφή στον πίνακα αυτό που περιγράφει την αντιγραφή για την σελίδα στην εικονική μνήμη.
- Page frame data table, το οποίο περιγράφει τα πλαίσια κύριας μνήμης που έχουν αντιστοιχηθεί με κάποια σελίδα. Ο πίνακας αυτός χρησιμοποιείται από τον αλγόριθμο αντικατάστασης.
- Swap-use table, που υπάρχει για κάθε δίσκο ή άλλη συσκευή στην οποία είναι εφικτό να αντιγράφονται σελίδες.

Η αντικατάσταση σελίδων στο Unix πραγματοποιείται με τη βοήθεια του Page frame data table. Ο αλγόριθμος αντικατάστασης σελίδων είναι μια τροποποίηση του αλγορίθμου Clock, γνωστός ως two-handed clock (ρολόι με δύο δείκτες). Ο αλγόριθμος αναζητά σελίδες προς αντικατάσταση βάσει του bit χρήσης. Το bit αυτό είναι ίσο με 0 όταν φορτώνεται η σελίδα αρχικά στη μνήμη αλλά τίθεται ίσο με 1 εάν γίνει αναφορά στη σελίδα αυτή για εγγραφή ή ανάγνωση. Ο πρώτος δείκτης του ρολογιού (fronthand) διατρέχει τις σελίδες – οργανωμένες στην κυκλική δομή του ρολογιού – για διαθέσιμες

σελίδες και θέτει το bit χρήσης ίσο με 0 σε κάθε τέτοια σελίδα. Αργότερα, ο δεύτερος δείκτης (backhand) διατρέχει για άλλη μία φορά τις σελίδες και κάθε σελίδα που το bit χρήσης της είναι 1 (δηλαδή έχει αναφορά σε αυτές μετά το πέρασμα του πρώτου δείκτη) αγνοείται, ενώ όταν συναντήσει σελίδα με bit χρήσης ίσο με 0, τότε η σελίδα αυτή δεν έχει προσπελαστεί από τη στιγμή του σαρώματος του πρώτου δείκτη. Τέτοιες σελίδες τοποθετούνται σε μια λίστα προς αντικατάσταση. Ο αλγόριθμος αυτός έχει δύο παραμέτρους: το ρυθμό σάρωσης (scanrate), ο οποίος μετριέται σε σελίδες ανά δευτερόλεπτο και το εύρος των δεικτών (handspread), το οποίο είναι το χρονικό διάστημα που περνά από τη σάρωση του πρώτου δείκτη μέχρι τη σάρωση του δεύτερου. Ο ρυθμός σάρωσης αρχικοποιείται κατά την έναρξη του συστήματος και μεταβάλλεται παίρνοντας μια τιμή μεταξύ του *slowscan* και του *fastscan*, ανάλογα με το ποσό της μνήμης που παραμένει ελεύθερο (μεταξύ του *lotsfree* και του *minfree*), δηλαδή, ο αλγόριθμος επιταχύνει τις σαρώσεις όσο η μνήμη γεμίζει προκειμένου να διώχνει σελίδες που θεωρούνται μη χρήσιμες στη μνήμη πιο γρήγορα. Η μεταβολή του εύρους των δεικτών μπορεί να χρησιμοποιηθεί σε καταστάσεις όπου υπάρχει έλλειψη σελίδων καθιστώντας μικρότερο το παράθυρο που επιτρέπεται να μένει αχρησιμοποίητη μια σελίδα. Η πολιτική αντικατάστασης σελίδων που ακολουθείται στα Unix συστήματα είναι μια πολιτική προεκκαθάρισης, δηλαδή απομακρύνονται σελίδες από την κύρια μνήμη πριν να δημιουργηθεί ανάγκη απομάκρυνσης μιας – τουλάχιστον – σελίδας λόγω έλλειψης χώρου, απλά με βάση την παλαιότητα της προσπέλασης της εκάστοτε σελίδας.

Δέσμευση μνήμης για τον πυρήνα του Unix

Ο πυρήνας του Unix δημιουργεί και καταστρέφει με μεγάλη συχνότητα μια σειρά από μικρούς πίνακες και δομές δεδομένων κατά την εκτέλεση διεργασιών, που είναι πολύ πιο μικρά από το τυπικό μέγεθος μιας σελίδας. Επομένως, μια πολιτική σελιδοποίησης θα οδηγούσε σε μη αποδοτική διαχείριση μνήμης. Ο στόχος όσον αφορά τη δέσμευση και αποδέσμευση μνήμης στον πυρήνα είναι ένας: η ταχύτητα. Σε κάποια Unix συστήματα (στα SVR4) έχει χρησιμοποιηθεί μια παραλλαγή του αλγορίθμου buddy. Η απλή εκδοχή του αλγορίθμου δεν έχει τόσο καλή απόδοση όσον αφορά την ταχύτητα, λόγω του χρόνου που χρειάζεται για να αναδιατάξει τα δεδομένα μέσα στη μνήμη. Η παραλλαγή που χρησιμοποιείται ονομάζεται lazy buddy system η οποία βασίζεται στη συμπεριφορά του Unix όπως έχει παρατηρηθεί πειραματικά. Πιο συγκεκριμένα, οι απαιτήσεις μνήμης του πυρήνα του Unix παρουσιάζουν μια σταθερότητα με το χρόνο. Έτσι εάν σε μια χρονική στιγμή αποδεσμευτεί ένα κομμάτι μνήμης και ο αλγόριθμος

buddy το συνενώσει, υπάρχει μεγάλη πιθανότητα σε μικρό χρονικό διάστημα να ζητηθεί εκ νέου ένα κομμάτι μνήμης ίσου μεγέθους με συνέπεια να πρέπει να ξαναδημιουργηθεί ο διαθέσιμος χώρος στη μνήμη. Όπως είναι αναμενόμενο, το lazy buddy system συνενώνει μόνο όταν γίνεται επιτακτικό να πραγματοποιηθεί συνένωση, χρησιμοποιώντας όσο περισσότερα μπλοκ είναι δυνατόν.

[εδώ μπορώ να πω κι άλλα, αλλά δε μου φαίνεται αναγκαίο]

2.6.2 Διαχείριση μνήμης σε συστήματα Linux

Η διαχείριση μνήμης σε συστήματα Linux έχει πολλά κοινά χαρακτηριστικά με το Unix, διατηρώντας όμως και μερικές βασικές διαφορετικές σχεδιαστικές επιλογές. Συνολικά το σύστημα διαχείρισης μνήμης είναι αρκετά περίπλοκο και διατηρεί δυο κύριους άξονες: την εικονική μνήμη που χρησιμοποιεί σελιδοποίηση και τη διαχείριση μνήμης για τον πυρήνα που σε αντίθεση με το Unix χρησιμοποιεί τον ίδιο μηχανισμό που χρησιμοποιείται για τη δέσμευση μνήμης για διεργασίες χρηστών.

Σελιδοποίηση σε Linux

Στο λειτουργικό Linux υπάρχει μια δομή οργάνωσης σελίδων που χρησιμοποιεί μια τρι-επίπεδη δομή πινάκων σελίδων, η οποία είχα αρχικά σχεδιαστεί για την υποστήριξη 64-bit Alpha επεξεργαστών. Οι επεξεργαστές Pentium x86 32-bit μπορούν να υποστηριχτούν χωρίς πρόβλημα στην απόδοση αφού όλες οι βελτιστοποιήσεις αναφορών γίνονται κατά το compile και όχι κατά την εκτέλεση. Η δομή αυτή αποτελείται από τρεις τύπους πινάκων.

- **Page directory**: Κάθε ενεργή διεργασία έχει ένα τέτοιο πίνακα ο οποίος έχει το μέγεθος μίας σελίδας. Κάθε εγγραφή του πίνακα αυτού δείχνει σε μία σελίδα του Page middle directory και πρέπει να βρίσκεται στην κύρια μνήμη για μία ενεργή διεργασία.
- **Page middle directory**: Ο πίνακας αυτός μπορεί να εκτείνεται σε πολλές σελίδες και κάθε εγγραφή του δείχνει σε μία σελίδα του Page directory.
- **Page directory**: Και αυτός ο πίνακας μπορεί να εκτείνεται σε πολλές σελίδες. Κάθε εγγραφή του πίνακα αυτού δείχνει σε μία σελίδα (το περιεχόμενο ενός πλαισίου της μνήμης) της διεργασίας.

Η δέσμευση μνήμης γίνεται με ένα μηχανισμό ο οποίος υλοποιεί το σύστημα φίλων (*buddy system*). Ο πυρήνας διατηρεί μια λίστα με ομάδες συνεχόμενων πλαισίων σταθερού μεγέθους. Κάθε ομάδα μπορεί να έχει 1,2,4,8,16 ή 32 πλαίσια σελίδων, και, σύμφωνα με το αλγόριθμο των φίλων (buddy algorithm), καθώς οι σελίδες αποδεσμεύονται και επαναδεσμεύονται, οι ομάδες διασπώνται και συνενώνονται.

Η πολιτική αντικατάστασης σελίδων που βρίσκονται στη μνήμη ακολουθεί τον αλγόριθμο ρολογιού με κάποιες τροποποιήσεις. Στο σχήμα που ακολουθείται στο Linux, το μπιτ χρήσης έχει αντικατασταθεί από μία μεταβλητή ηλικίας μεγέθους οκτώ μπιτ, έχει, δηλαδή 256 διαφορετικές τιμές. Κάθε φορά που γίνεται μια προσπέλαση στην τρέχουσα σελίδα η μεταβλητή αυτή αυξάνεται. Παράλληλα, το λειτουργικό σαρώνει περιοδικά όλες τις σελίδες και μειώνει την ηλικία κάθε σελίδας καθώς εναλλάσσεται κυκλικά μεταξύ των σελίδων που είναι στη μνήμη. Μία σελίδα που έχει ηλικία μηδέν είναι μια “παλιά” σελίδα που είναι υποψήφια προς αντικατάσταση αφού δεν έχει προσπελαστεί για ένα ικανό διάστημα χρόνου. Όσο πιο μεγάλη είναι η ηλικία μιας σελίδας τόσο πιο πολλές φορές έχει προσπελαστεί η σελίδα αυτή σε πρόσφατες χρονικές στιγμές και τόσο λιγότερο υποψήφια προς αντικατάσταση είναι. Υλοποιείται, λοιπόν, μια λογική που μοιάζει με τη Λιγότερο Συχνά Χρησιμοποιούμενη (Least Frequently Used).

Δέσμευση μνήμης για τον πυρήνα του Linux

Το λειτουργικό σύστημα Linux χρησιμοποιεί και στον πυρήνα πλαίσια σελίδων. Σε επίπεδο πυρήνα χρειάζεται να δεσμευτεί και να αποδεσμευτεί μνήμη για συγκεκριμένους σκοπούς (δυναμική δέσμευση δεδομένων πυρήνα, στατικός κώδικας πυρήνα, κρυφή μνήμη σελίδων).

Χρησιμοποιείται εκ νέου ο αλγόριθμος των φίλων για τη δέσμευση σελίδων, αλλά επειδή συχνά οι ανάγκες του πυρήνα είναι για πολύ μικρά τμήματα μνήμης έχει προταθεί και χρησιμοποιείται ένα σχήμα που ονομάζεται slab allocation []. Το σχήμα αυτό διαχειρίζεται τη μνήμη εσωτερικά μίας σελίδας, χωρίζοντάς την σε μικρά υπο-τμήματα (*chunks*), μεγέθους 32 έως 4080 bytes, ενώ το μέγεθος μιας σελίδας σε ένα Pentium/x86 είναι 4Kbytes.

2.6.3 Διαχείριση μνήμης σε συστήματα Windows

Το σύστημα διαχείρισης μνήμες στα Windows, έχει σχεδιαστεί ώστε να μπορεί να λειτουργεί σε διάφορα συστήματα με το μέγεθος της σελίδας να κυμαίνεται μεταξύ 4 Kbytes και 64 Kbytes.

Κάθε διεργασία μπορεί να χρησιμοποιήσει ένα διαφορετικό 32-bit χώρο διευθύνσεων, που επιτρέπει μέχρι 4 GBytes ανά διεργασία. Το σύστημα δεσμεύει 2 GByte από τα διαθέσιμα στον εικονικό χώρο διευθύνσεων και έτσι κάθε διεργασία μπορεί να διαχειριστεί μέχρι 2 GByte.

Κάθε φορά που δημιουργείται μια διεργασία, αυτή μπορεί να φέρει στη μνήμη έως και σελίδες συνολικού μεγέθους 2 GByte ενώ το ελάχιστο είναι 128 KBytes. Οι σελίδες έχουν σταθερό μέγεθος και μπορεί να βρίσκονται σε μία από τις παρακάτω καταστάσεις:

- Διαθέσιμη (*available*): Σελίδες που δε χρησιμοποιούνται από καμία διεργασία.
- Κρατημένη (reserved): Ένα σύνολο συνεχόμενων σελίδων που δε χρησιμοποιούνται από κάποια διεργασία, προορίζονται όμως για μία διεργασία και δεν υπολογίζονται ως δεσμευμένη μνήμη από τη διεργασία αυτή μέχρι να χρειαστεί να γράψει σε αυτές η εν λόγω διεργασία.
- Δεσμευμένη (committed): Σελίδες που το σύστημα διαχείρισης μνήμης θεωρεί πως δεν έχουν ελεύθερο χώρο.

Η χρήση κρατημένων και δεσμευμένων σελίδων είναι πολύ χρήσιμη γιατί μειώνει τη μνήμη που δεσμεύεται για μία διεργασία στις αναγκαίες σελίδες αφήνοντας περισσότερες ελεύθερες σελίδες για άλλες διεργασίες και επιτρέπει σε ένα νήμα ή σε μία διεργασία να δηλώσει ότι χρειάζεται ένα κομμάτι μνήμης το οποίο θα δεσμευτεί άμεσα.

Η διαχείριση των φορτωμένων σελίδων γίνεται με δυναμική δέσμευση αλλά και πολιτική τοπικής αντικατάστασης. Έτσι, όταν μια διεργασία φορτώνεται στη μνήμη για πρώτη φορά της ανατίθενται ένας συγκεκριμένος αριθμός πλαισίων τα οποία μπορεί να χρησιμοποιήσει. Έτσι, όταν πρέπει να αντικαταστήσει μια σελίδα, μια ήδη φορτωμένη σελίδα απομακρύνεται από τη μνήμη για να φορτωθεί νέα σελίδα. Στη λογική αυτή εφαρμόζονται και οι δύο παρακάτω κανόνες, στην προσπάθεια το σχήμα δέσμευσης και διαχείρισης μνήμης να είναι πιο αποδοτικό:

- Όταν υπάρχει πολλή ελεύθερη κύρια μνήμη, επιτρέπεται στο σύνολο σελίδων μια διεργασίας να μεγαλώσει. Αυτό επιτυγχάνεται όταν σε κάθε σφάλμα σελίδα φορτώνεται η νέα σελίδα χωρίς να απομακρύνεται κάποια από τις ήδη φορτωμένες.
- Στην αντίθετη περίπτωση που η μνήμη έχει αρχίσει να μην επαρκεί, το σύστημα αφαιρεί τις σελίδες που δε χρησιμοποιήθηκαν πρόσφατα από τα τμήματα μνήμης

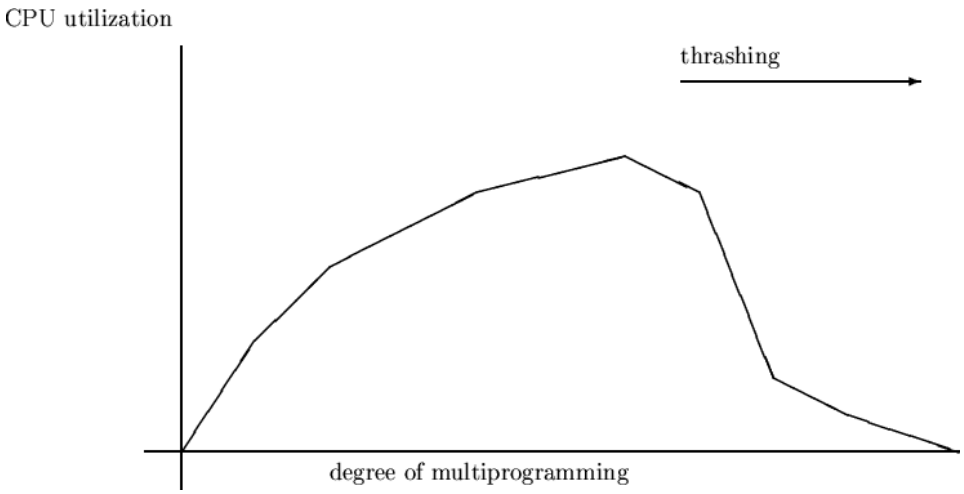
που έχουν δεσμευτεί ανά διεργασία, μειώνοντας έτσι και το μέγιστο αριθμό σελίδων που μπορεί να έχει φορτωμένο μια διεργασία.

ΚΕΦΑΛΑΙΟ 3

ΕΞΟΝΤΩΣΗ

3.1 Ορισμός του προβλήματος

Όπως ορίστηκε νωρίτερα, το πρόβλημα της εξόντωσης (page thrashing) είναι η δραματική μείωση της χρησιμοποίησης του επεξεργαστή (process utilization collapse) λόγω της αύξησης του επιπέδου του πολυπρογραμματισμού (multiprogramming level). Εξάλλου τα μεγέθη αυτά συνδέονται ποιοτικά με το παρακάτω σχήμα.



Σχήμα 1: Χρησιμοποίηση επεξεργαστή ως προς επίπεδο πολυπρογραμματισμού

Το πρόβλημα αυτό έχει εμφανιστεί από τα πρώτα βήματα των λειτουργικών συστημάτων και ήδη υπάρχουν υλοποιημένες κάποιες πολιτικές αποφυγής ή μείωσης της έντασης του προβλήματος. Βασικές έννοιες που συναντά κανείς στις πολιτικές αυτές είναι:

- Page Fault Frequency, δηλαδή το ρυθμό με τον οποίο συμβαίνουν σφάλματα σελίδων
- Process Working Set, δηλαδή το σύνολο των σελίδων μιας διεργασίες που κατά τεκμήριο εκτελεί και πρόκειται να εκτελέσει (τεκμήριο είναι, επί της ουσίας η εκτέλεσή τους κατά το πρόσφατο παρελθόν)

- Process Resident Set, δηλαδή το σύνολο των σελίδων μιας διεργασίας που βρίσκονται διαθέσιμες στην κύρια μνήμη

3.2 Τεχνικές αποφυγής εξόντωσης

Σε αυτήν την ενότητα περιγράφονται εργασίες που περιγράφουν μεθόδους αποφυγής του εξόντωσης λόγω σελιδοποίησης (page thrashing), οι οποίες είτε αναφέρονται στο ίδιο ακριβώς πρόβλημα είτε δίνουν απαντήσεις σε παρεμφερή προβλήματα.

3.2.1 Load control policy

Η πολιτική αυτή βασίζεται στη διατήρηση ενός επιπέδου πολυπρογραμματισμού ικανού να εκτελεστεί. Το κριτήριο με βάση το οποίο λαμβάνεται η απόφαση αυτή είναι εάν μια διεργασία έχει φορτωμένο στη μνήμη (resident set) ένα ικανό υποσύνολο του συνόλου εργασίας της διεργασίας (working set), δηλαδή των σελίδων που προσπέλασε πρόσφατα.

Στην ουσία, η πολιτική αυτή όταν υπάρχει συμφόρηση μειώνει το επίπεδο πολυπρογραμματισμού έτσι ώστε όλες οι ενεργές διεργασίες να είναι αυτές που εάν εκτελεστούν δε θα χρειαστεί να φορτωθεί στην κύρια μνήμη μία νέα σελίδα.

3.2.2 L=S criterion

Μία άλλη προσέγγιση έχει προταθεί στο [5] και βασίζεται στη μεταβολή του επιπέδου του πολυπρογραμματισμού. Η μεταβολή αυτή γίνεται έτσι ώστε να είναι ίσος ο μέσος χρόνος μεταξύ δύο σφαλμάτων σελίδας με το μέσο χρόνο εξυπηρέτησης ενός σφάλματος σελίδας. Σύμφωνα με μελέτες της απόδοσης του αλγορίθμου αυτού, το κριτήριο αυτό οδηγεί σε μεγιστοποίηση της χρησιμοποίησης του επεξεργαστή (utilization) δεδομένων των συνθηκών εξόντωσης.

3.2.3 Κριτήριο 50%

Το κριτήριο 50% είναι μια πολιτική που προτάθηκε στο [6] και αποσκοπεί στη διατήρηση της χρησιμοποίησης της συσκευής της μνήμης σε ποσοστό τουλάχιστον 50%. Αντίστοιχες μελέτες της απόδοσης του αλγορίθμου αυτού δείχνουν ότι σε αυτό το σημείο επιτυγχάνεται μέγιστη χρησιμοποίηση του επεξεργαστή.

3.2.4 Τροποποίηση του αλγορίθμου Clock page replacement

Μια άλλη προσέγγιση είναι η τροποποίηση του αλγορίθμου αντικατάστασης σελίδων ρολογιού. Στο [8] περιγράφεται μια τεχνική που χρησιμοποιεί καθολική αντικατάσταση σελίδων και περιλαμβάνει την παρακολούθηση του ρυθμού με τον οποίο σαρώνει ο δείκτης τις σελίδες που είναι φορτωμένες στη μνήμη. Ορίζονται δύο κατώφλια: κατώφλι ελάχιστου και μέγιστου ρυθμού σάρωσης των σελίδων.

Όταν ο ρυθμός σάρωσης είναι μικρότερος από το κατώφλι ελάχιστου ρυθμού τότε είτε συμβαίνουν λίγα σφάλματα σελίδας με αποτέλεσμα να είναι λίγες οι αιτήσεις για προώθηση του δείκτη, είτε, για κάθε αίτηση προώθησης του δείκτη ο μέσος αριθμός πλαισίων που ελέγχονται για απομάκρυνση είναι μικρός, κάτι που σημαίνει ότι υπάρχουν πολλές σελίδες που βρίσκονται στη μνήμη και δεν έχουν προσπελαστεί πρόσφατα και άρα είναι αντικαταστάσιμες. Τόσο στην πρώτη περίπτωση, όσο και στη δεύτερη, είναι σαφές ότι μπορούμε με ασφάλεια να αυξήσουμε το επίπεδο πολυπρογραμματισμού. Αντίθετα όταν ο ρυθμός σάρωσης ξεπεράσει το κατώφλι μέγιστου ρυθμού, τότε, αυτό συμβαίνει είτε λόγω αύξησης του λόγου σφαλμάτων σελίδων είτε λόγω αύξησης της δυσκολίας εύρεσης σελίδας διαθέσιμης προς αντικατάσταση, κάτι που είναι ένδειξη για βεβαρημένη κατάσταση στη μνήμη, οπότε και οδηγούμαστε στη μείωση του επιπέδου του πολυπρογραμματισμού.

3.2.5 Process suspension

Σε μερικές από τις παραπάνω στρατηγικές αναφερθήκαμε στη μείωση του επιπέδου του πολυπρογραμματισμού. Αυτό υλοποιείται, με αναστολή εκτέλεσης κάποιων διεργασιών. Στο [8] παρατίθεται μια λίστα με έξι δυνατότητες ως προς την επιλογή διεργασίας προς αναστολή:

1. **Διεργασία ελάχιστης προτεραιότητας** (lowest-priority process): Αυτό υλοποιεί μια πολιτική χρονοδρομολόγησης και δεν είναι απαραίτητα συνδεδεμένο με θέματα απόδοσης.
2. **Διεργασία που υποπίπτει σε σφάλματα** (faulting process): Η λογική πίσω από την επιλογή μιας τέτοιας διεργασίας είναι κατά πάσα πιθανότητα ο λόγος που γίνονται σφάλματα είναι ότι δεν είναι όλο το σύνολο εργασίας φορτωμένο στην κύρια μνήμη και έτσι θα έχουμε το μικρότερο κόστος απόδοσης εάν αναστείλουμε τη διεργασία αυτή. Επιπλέον, η επιλογή αυτή έχει το άμεσο κέρδος ότι σταματάμε

μια διεργασία πριν σταματήσει τη λειτουργία της μόνη της σε ένα επόμενο σφάλμα σελίδας δεσμεύοντας πόρους για αντικατάσταση σελίδας.

3. **Πιο πρόσφατα ενεργοποιηθείσα διεργασία** (Last process activated): Η επιλογή αυτή γίνεται με τη λογική ότι μια διεργασία που μόλις ξεκίνησε δε θα έχει προλάβει να φορτώσει όλο το σύνολο εργασίας στην κύρια μνήμη.
4. **Διεργασία με το μικρότερο σύνολο φορτωμένων σελίδων** (Process with the smallest resident set): Η επιλογή αυτή είναι κάπως αντιφατική. Πιο συγκεκριμένα, ενώ μια τέτοια διεργασία έχει ισχυρή τοπικότητα και, λογικά, θα εκτελεστεί απρόσκοπτα, είναι η διεργασία για την οποία είναι πολύ εύκολο να ξαναφορτωθεί.
5. **Μεγαλύτερη διεργασία** (Largest process): Με την επιλογή μιας τέτοιας διεργασία ελευθερώνονται πολλά πλαίσια στη μνήμη με αποτέλεσμα να αυξάνεται η πιθανότητα να αποφύγουμε την ανάγκη για νέες αναστολές διεργασιών.
6. **Διεργασία με το μεγαλύτερο εναπομείναν παράθυρο εκτέλεσης** (Process with the largest remaining execution window): Αυτή η επιλογή έχει να κάνει με τα σχήματα χρονοπρογραμματισμού που υλοποιούν ένα παράθυρο εκτέλεσης με τη λήξη του οποίου η διεργασία τοποθετείται στο τέλος της ουράς έτοιμων διεργασιών. Με την επιλογή αυτή ενισχύεται τη λογική της πολιτικής χρονοπρογραμματισμού σύμφωνα με τον εναπομείναντα χρόνο εκτέλεσης. Πρόκειται για προσομοίωση της συμπεριφοράς της Shortest Remaining Time to Completion First.

3.2.6 Μείωση εξόντωσης σε συστήματα με κατανεμημένη μνήμη

Στο άρθρο [2] οι συγγραφείς αναφέρονται στην εξόντωση που μπορεί να δημιουργηθεί σε συστήματα με πολλούς επεξεργαστές και κοινή, κατανεμημένη μνήμη. Σε τέτοια συστήματα το γεγονός ότι όλοι οι επεξεργαστές θεωρούν ότι το σύνολο της κύριας μνήμης είναι τοπικό ενώ στην πραγματικότητα αυτό είναι μια αφαίρεση για να διευκολυνθεί η επικοινωνία των διεργασιών, η οποία μπορεί να οδηγήσει σε κακή απόδοση.

Σε ένα σύστημα κατανεμημένης μνήμης (Distributed Shared Memory) κάθε διεργασία έχει στη διάθεσή της ένα χώρο διευθύνσεων τον οποίο μπορούν να χρησιμοποιήσουν όλα τα νήματα που ανήκουν στη διεργασία αυτή. Με αυτόν τον τρόπο είναι πολύ εύκολο

να εκτελεστούν παλαιότερα προγράμματα που είχαν σχεδιαστεί για ένα σύστημα με πολλούς επεξεργαστές (χωρίς κοινή μνήμη). Επιπλέον, υπάρχει ένας μηχανισμός μετανάστευσης νημάτων (thread migration). Το σύστημα μπορεί δυναμικά να μεταφέρει ένα νήμα που εκτελείται σε έναν επεξεργαστή, σε κάποιον άλλο είτε για να αυξήσει την απόδοση του συστήματος συνολικά είτε για να επιτευχθεί καλύτερος διαμοιρασμός φόρτου (load balancing). Παράλληλα, όμως, η μετανάστευση νημάτων χρησιμοποιείται για να μειώνεται ο λόγος σφαλμάτων σελίδων, κάτι που ούτως ή άλλως συνεισφέρει στην βελτίωση της απόδοσης. Η μεταφορά ενός νήματος σε έναν άλλο επεξεργαστή μεταβάλλει τον τρόπο με τον οποίο γίνεται προσπέλαση της μνήμης σε κάθε υποσύστημα και, έτσι, μια προσεκτική μελέτη μπορεί να οδηγήσει σε μείωση της συμφόρησης κατά την προσπέλαση σελίδων, καθώς, και τη μείωση της εξόντωσης του συστήματος (page thrashing).

Στο [2], παρουσιάζεται ένα σχήμα που αποσκοπεί στη μείωση της έντασης του φαινόμενου της εξόντωσης. Το σχήμα αυτό χρησιμοποιεί πληροφορίες προσπέλασης σελίδων από κάθε νήμα και χρονοπρογραμματίζει το κάθε νήμα ανάλογα με τις πληροφορίες αυτές. Αποτελείται από δύο μεθόδους χρονοπρογραμματισμού νημάτων:

1. Χρονοπρογραμματισμός συσχετισμού (correlation scheduling), και
2. Χρονοπρογραμματισμός αναστολής (suspension scheduling)

Για το χρονοπρογραμματισμό συσχετισμού χρειάζεται να οριστεί μια μετρική συσχετισμού μεταξύ νημάτων. Έτσι, δεδομένης αυτής τα μετρικής, όταν δύο νήματα που είναι “αρκετά” συσχετισμένα τρέχουν στον ίδιο επεξεργαστή, τότε δε συμβαίνουν πολλά σφάλματα σελίδων στο σύστημα αυτό και άρα δεν συμβαίνει εξόντωση στο σύστημα. Έτσι γίνεται μια προσπάθεια να χρονοπρογραμματιστούν έντονα συσχετισμένα νήματα στον ίδιο επεξεργαστή. Όταν ένας επεξεργαστής είναι διαθέσιμος τότε επιλέγεται το νήμα το οποίο προκύπτει από τη μετρική συσχετισμού το πιο κατάλληλο, και αντίστοιχα όταν ένα νέο νήμα δημιουργείται τότε αυτό χρονοπρογραμματίζεται στον ελεύθερο επεξεργαστή για τον οποίο η μετρική συσχετισμού θα δώσει καλύτερα αποτελέσματα.

Συσχετισμός μεταξύ νημάτων (θεωρητικός υπολογισμός)

Το αναλυτικό μοντέλο που χρησιμοποιήθηκε για τον υπολογισμό του συσχετισμού μεταξύ νημάτων i, j , CA_{ij} , θεωρεί τη συνάρτηση $D_i^p \ t$, η οποία είναι η συνάρτηση

συχνότητας προσπέλασης της σελίδας p από το νήμα i τη χρονική στιγμή t. Ο συσχετισμός μεταξύ νημάτων εξαρτάται από τις σελίδες που προσπελαύνουν. Συνεπώς, εξαρτάται από τη συμπεριφορά προσπέλασης των δύο νημάτων στις κοινές του σελίδες.

$$CA_{ij} = \sum_{\forall p} \int_{T_1}^{T_2} D_i^p\ t\ \cdot D_j^p\ t\ dt$$

Συσχετισμός μεταξύ νήματος και επεξεργαστή

Με χρήση του παραπάνω μεγέθους ορίζεται από τους συγγραφείς και ο συσχετισμός μεταξύ νήματος i και επεξεργαστή n, CN_i^n.

$$CN_i^n = \sum_{\forall j} \begin{cases} 0 & n_j = 0 \text{ (thread j is not running)} \\ CA_{ij} & n_j = n \\ -CA_{ij} & n_j \neq n \end{cases}$$

(όπου n_j είναι ο αριθμός του επεξεργαστή στον οποίο τρέχει το νήμα j).

Συσχετισμός μεταξύ νημάτων (υπολογισμός με βάση τα πραγματικά δεδομένα)

Οι παραπάνω εξισώσεις είναι αναλυτική μοντελοποίηση του υπολογισμού του συσχετισμού που χρησιμοποιεί πληροφορίες που δεν είναι διαθέσιμες. Οι μόνες διαθέσιμες πληροφορίες είναι πληροφορίες που αναφέρονται στα σφάλματα σελίδων, από τις οποίες γίνεται προσέγγιση της συχνότητας προσπέλασης. Έτσι ο υπολογισμός συσχετισμού δύο νημάτων γίνεται ως εξής:

$$CA_{ij} \approx \sum_{\forall p} \int_{T_1}^{T_2} PT_{ij}^p\ t\ dt$$
$$\approx \sum_{\forall p} \int_{0}^{T_1} Age\ PT_{ij}^p\ t\ ,t\ dt$$

όπου $PT_{ij}^p\ t$ είναι η συχνότητα μεταφοράς της σελίδας p από το νήμα i στο νήμα j τη χρονική στιγμή t και θεωρείται ότι προσεγγίζει το γινόμενο μεταξύ των συναρτήσεων συχνότητας προσπέλασης των δύο νημάτων που είδαμε στην αναλυτική σχέση. Επιπλέον εφαρμόζεται μια συνάρτηση $Age\ x,t$ που δίνει διαφορετικό βάρος στη συχνότητα προσπέλασης x ανάλογα με την παλαιότητα (μεγαλύτερο βάρος στα πιο πρόσφατα δεδομένα).

Ο χρονοπρογραμματισμός αναστολής υλοποιείται από έναν μηχανισμό ανατολής εκτέλεσης νημάτων. Νήματα που είναι υποψήφια για αναστολή είναι αυτά που τρέχουν σε διαφορετικούς επεξεργαστές και προκαλούν πολλά σφάλματα σελίδας λόγω του ότι αναφέρονται στην ίδια σελίδα. Όταν συμβαίνει αυτό, τότε η σελίδα πρέπει να μετακινείται από την τοπική κύρια μνήμη του ενός επεξεργαστή στην τοπική κύρια μνήμη του δεύτερου και έτσι γίνονται συνεχόμενα σφάλματα σελίδων που οδηγούν σε συμπεριφορά εξόντωσης. Σε μια τέτοια περίπτωση επιλέγεται ένα από τα νήματα αυτά και αναστέλλεται η λειτουργία του. Ο εντοπισμός του φαινομένου αυτού βασίζεται στο λόγο σφαλμάτων σελίδων των νημάτων, όταν ο λόγος αυτός για ένα νήμα αυξάνει δραστικά τότε το νήμα αυτό παύει να εκτελείται.

Το σχήμα που περιγράφεται υλοποιήθηκε στο περιβάλλον PARSEC, το οποίο παρέχει τη δυνατότητα για κατανεμημένη εκτέλεση νημάτων με υψηλή απόδοση και δυναμική κατανομή φόρτου. Το περιβάλλον αυτό αποτελείται από:

- DSM manager: αναλαμβάνει τη συλλογή στοιχείων των νημάτων.
- Dtask manager: διαχειρίζεται τη δημιουργία και την καταστροφή διεργασιών, νημάτων πυρήνα, καθώς και την διαχείριση φόρτου.
- D-threads library: είναι το πακέτο νημάτων επιπέδου χρήστη που χρησιμοποιείται.

Η υλοποίηση του χρονοπρογραμματισμού συσχετισμού ακολουθεί τα παρακάτω βήματα:

1. Όταν υπάρχει ένα νέο έτοιμο νήμα αυτό αποστέλλεται στον καθολικό χρονοπρογραμματιστή (global scheduler).
2. Εάν υπάρχουν ελεύθερα νήματα σε επίπεδο πυρήνα, τότε ο καθολικός χρονοπρογραμματιστής υπολογίζει το συσχετισμό μεταξύ του νέου νήματος και των επεξεργαστών που έχουν ελεύθερα νήματα επιπέδου πυρήνα και αντιστοιχεί το νέο νήμα με τον επεξεργαστή για τον οποίο ο συσχετισμός είναι μέγιστος.
3. Εάν δε βρεθεί κάποιο νήμα επιπέδου πυρήνα ελεύθερο τότε το νέο νήμα τοποθετείται στην καθολική ουρά έτοιμων νημάτων (global ready queue).
4. Όταν ένα νήμα πυρήνα σε κάποιον επεξεργαστή ελευθερωθεί τότε ο τοπικός χρονοπρογραμματιστής (local scheduler) στέλνει μια αίτηση στον καθολικό για αντιστοίχηση με νήμα χρήστη. Ο καθολικός χρονοπρογραμματιστής υπολογίζει το συσχετισμό μεταξύ το νήματος πυρήνα που ελευθερώθηκε και όλων των νημάτων χρήστη που υπάρχουν στην ουρά έτοιμων νημάτων και αντιστοιχεί το

πιο κατάλληλο νήμα, δηλαδή το νήμα που μεγιστοποιεί τη συνάρτηση συσχετισμού.

Ο χρονοπρογραμματισμός αναστολής υλοποιείται ακολουθώντας τα παρακάτω βήματα:

1. Όταν ένα νήμα προκαλέσει σφάλμα σελίδας, ο τοπικός DSM manager αυξάνει μια μεταβλητή που αντιστοιχεί στο νήμα αυτό. Όταν η μεταβλητή αυτή για κάποιο νήμα υπερβεί ένα δοθέν κατώφλι, τότε ο DSM manager δίνει την εντολή στον τοπικό χρονοπρογραμματιστή να αναστείλει τη λειτουργία του νήματος αυτού.
2. Όταν ο τοπικός χρονοπρογραμματιστής λάβει το μήνυμα αναστολής νήματος "σημαδεύει" το νήμα ως νήμα προς αναστολή.
3. Όταν το νήμα καλέσει μια ρουτίνα της βιβλιοθήκης νημάτων χρήστη ελέγχεται εάν το νήμα είναι προς αναστολή και, εάν ναι, αφού αποθηκευτεί η τρέχουσα κατάσταση του νήματος, αυτό αναστέλλεται.
4. Το νήμα του οποίου η λειτουργία μόλις ανεστάλη μεταφέρεται στην καθολική ουρά έτοιμων νημάτων.

Τα αποτελέσματα της προσομοίωσης που έγινε από τους συγγραφείς έδειξαν σαφή βελτίωση της απόδοσης του συστήματος με το σχήμα που πρότειναν. Η επιτυχία του βασίζεται στο ότι οι πιο πολλές πολυνηματικές εφαρμογές αποτελούνται από νήματα με μεγάλη συσχέτιση, οπότε ο χρονοπρογραμματισμός συσχετισμού είναι πολύ αποδοτικός. Στις περιπτώσεις που πολυνηματικές εφαρμογές έχουν νήματα με μικρό συσχετισμό, τότε "αναλαμβάνει" ο χρονοπρογραμματισμός αναστολής. Ο αλγόριθμος αυτός εντοπίζει όλα τα νήματα που προκαλούν εξόντωση τα οποία αναστέλλονται όταν εκτελέσουν κάποια ρουτίνα της βιβλιοθήκης νημάτων. Αυτό είναι αναγκαίο γιατί τα νήματα επιπέδου χρήστη είναι μη-προεκχωρητικά και έτσι το σύστημα δεν μπορεί να σταματήσει άμεσα ένα νήμα. Ένας μηχανισμός με προεκχωρητικά νήματα θα μπορούσε να βελτιώσει ακόμα περισσότερο την απόδοση σύμφωνα με τους συγγραφείς.

3.2.7 Βελτίωση χρονοπρογραμματισμού του Linux με σκοπό την αποφυγή εξόντωσης

Στα άρθρα [3] και [4] οι συγγραφείς προτείνουν μία μέθοδο βελτίωσης του χρονοπρογραμματισμού του πυρήνα του λειτουργικού Linux. Το λειτουργικό Linux υιοθετεί τον αλγόριθμο ρολογιού, που έχει παρουσιαστεί εκτενώς, για την

αντικατάσταση σελίδων. Η αλλαγή που προτείνεται από τους συγγραφείς αναφέρεται στη συμπεριφορά του συστήματος όταν συμβαίνει σφάλμα σελίδας. Έτσι, υπό προϋποθέσεις, μια διεργασία που προκαλεί σφάλματα σελίδας προσπαθώντας να φέρει στην κύρια μνήμη όλες τις σελίδες που ανήκουν στο σύνολο εργασίας, αντί να ανασταλεί η λειτουργία της, της επιτρέπεται να φέρει όλες τις σελίδες που χρειάζεται για να εκτελεστεί και, με αυτόν τον τρόπο, να μειώσει σταδιακά το λόγο σφαλμάτων σελίδων και να αυξήσει τη χρησιμοποίηση του επεξεργαστή.

Κατά τη διαδικασία του χρονοπρογραμματισμού διεργασιών και διαχείρισης της μνήμης το Linux προσπαθεί να διατηρήσει υψηλή τη χρησιμοποίηση του επεξεργαστή, μερικές φορές σε βάρος της χρησιμοποίησης της μνήμης. Γενικά, αυτό δεν αποτελεί πρόβλημα, όμως σε περίπτωση που υπάρχει συμφόρηση στη μνήμη μπορεί η σχεδιαστική επιλογή αυτή να επιφέρει εξόντωση πιο γρήγορα από ότι θα ήταν εφικτό εάν υιοθετούταν ένας πιο αποδοτικός αλγόριθμος. Έτσι, οι συγγραφείς προτείνουν μια βελτίωση του πυρήνα του Linux, η οποία οδηγεί σε καλύτερη χρησιμοποίηση της μνήμης όταν η χρησιμοποίηση του επεξεργαστή δεν έχει πρόβλημα και μεταβάλλει τη συμπεριφορά του πυρήνα όταν μειωθεί πολύ η χρησιμοποίηση του επεξεργαστή.

Ποιοτικά, η βελτίωση αυτή βασίζεται στην παρακάτω απλή και διαισθητική λογική. Όταν υπάρχουν πολλές διεργασίες ικανές να καταναλώσουν επεξεργαστική ισχύ (CPU-bound processes), καθώς και υψηλοί ρυθμοί σφαλμάτων σελίδων και χαμηλή χρησιμοποίηση του επεξεργαστή, επιλέγεται μια από αυτές τις διεργασίες για φέρει στη μνήμη όλο το σύνολο εργασίας της και να προχωρήσει στην εκτέλεση της άμεσα. Εάν δεν συντρέχουν οι παραπάνω συνθήκες, η βελτίωση που προτείνεται από τους συγγραφείς του [3] δεν επηρεάζει το σύστημα στη λειτουργία του.

Για τις ανάγκες του σχήματος αυτού δημιουργούνται δύο νέες ρουτίνες πυρήνα. Μια ρουτίνα εντοπισμού και μια ρουτίνα προστασίας. Η πρώτη παρακολουθεί τις παραμέτρους του συστήματος και αποφασίζει εάν πρέπει να επιλεγεί μια διεργασία για προστασία. Οι παράμετροι που χρησιμοποιούνται είναι ο λόγος σφάλματος σελίδας της κάθε διεργασίας καθώς και η χρησιμοποίηση του επεξεργαστή που επιτυγχάνεται. Επιπλέον, ορίζονται τέσσερα κατώφλια και μία λίστα διεργασιών στον πυρήνα:

- CPU_Low: η ελάχιστη επιτρεπτή χρησιμοποίηση του επεξεργαστή.
- CPU_High: ο στόχος της χρησιμοποίησης του επεξεργαστή στον οποίο αποσκοπεί η ρουτίνα προστασίας, όταν ενεργοποιηθεί.

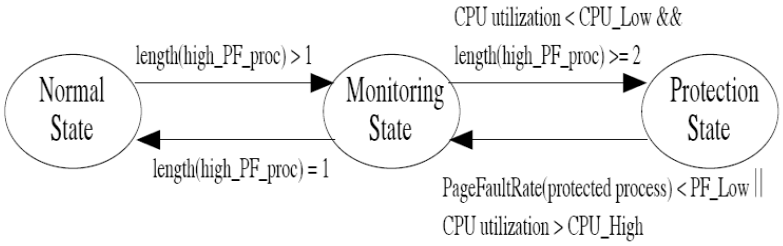
- PF_Low: ο στόχος του λόγου σφαλμάτων σελίδων της διεργασίας που έχει τεθεί υπό προστασία, στον οποίο αποσκοπεί η ρουτίνα προστασίας, όταν ενεργοποιηθεί.
- PF_High: ο μέγιστος λόγος σφαλμάτων σελίδων που χρειάζεται να φτάσει μια διεργασία για να θεωρηθεί επικίνδυνη για δημιουργία φαινομένου εξόντωσης.
- high_PF_proc: μια συνδεδεμένη λίστα αποτελούμενη από τις διεργασίες που έχουν εντοπιστεί από τη ρουτίνα εντοπισμού ως διεργασίες με αυξημένο λόγο σφαλμάτων σελίδων. Μια διεργασία προστίθεται στη λίστα αυτή όταν ξεπεράσει το PF_High και αφαιρείται όταν ο λόγος σφαλμάτων σελίδων μειωθεί κάτω από το PF_Low.

Ορίζονται, λοιπόν τρεις καταστάσεις στις οποίες μπορεί να βρεθεί η μηχανή καταστάσεων το σχήματος αυτού:

1. **Κανονική κατάσταση** (normal state): Σε αυτήν την κατάσταση το σύστημα απλά διατηρεί τη λίστα high_PF_proc και προσθαφαιρεί σε αυτήν τις κατάλληλες διεργασίες.
2. **Κατάσταση παρακολούθησης** (monitoring state): Όταν το μήκος της λίστας high_PF_proc ξεπεράσει το 1, τότε το σύστημα μεταβαίνει στην κατάσταση αυτή, στην οποία ενεργοποιείται η ρουτίνα παρακολούθησης, η οποία παρακολουθεί τη χρησιμοποίηση του επεξεργαστή καθώς και τη λίστα “επικινδύνων” διεργασιών. Εάν, επιπλέον, μειωθεί η καθολική χρησιμοποίηση κάτω από το κατώφλι CPU_Low και το μήκος της λίστας high_PF_proc είναι μεγαλύτερο ή ίσο από 2, τότε επιλέγεται μια διεργασία για να προστατευθεί και το σύστημα μεταβαίνει στην κατάσταση προστασίας. Αντίθετα, εάν το μήκος της λίστας high_PF_proc μειωθεί σε 1, τότε γίνεται μετάβαση στην κανονική κατάσταση.
3. **Κατάσταση προστασίας** (protection state): Στην κατάσταση αυτή η ρουτίνα προστασίας που ενεργοποιείται σημαδεύει τη διεργασία που την ενεργοποίησε και διατηρεί τη μεταβλητή του συστήματος για τη συγκεκριμένη διεργασία, swap_cnt ίση μονίμως με μηδέν ώστε εάν χρειαστεί να φορτωθεί μία σελίδα από το δίσκο, να επιτραπεί να συνεχίσει η διεργασία να καλεί τις σελίδες που χρειάζεται. Επίσης, η ρουτίνα προστασίας παρακολουθεί τη χρησιμοποίηση του επεξεργαστή καθώς και το λόγο σελίδων όλων των διεργασιών που είναι στη λίστα high_PF_proc. Έτσι, εάν, είτε η χρησιμοποίηση του επεξεργαστή περάσει το κατώφλι CPU_High, είτε ο λόγος

σφαλμάτων σελίδων της διεργασίας υπό προστασία μειωθεί κάτι από το κατώφλι PF_Low, τότε γίνεται μετάβαση στην κατάσταση παρακολούθησης.

Στο παρακάτω σχήμα φαίνεται σχηματικά η προηγούμενη περιγραφή της λειτουργίας του συστήματος.



Σχήμα 2: Καταστάσεις του μηχανισμού Linux kernel patch

Το σχήμα που προτάθηκε στο [3] και στο [4] δεν είναι ενεργό γενικά και έτσι δεν επιβαρύνει το σύστημα, παρά μόνο όταν ήδη υπάρχει πρόβλημα, με σκοπό τη βελτίωση της κατάστασης του συστήματος. Επίσης, η επιβάρυνση αυτή είναι αμελητέα μπροστά στις καθυστερήσεις που επιφέρονται από τα σφάλματα σελίδων και, έτσι, κρίνεται λογική.

Μέρος Β΄: Στρατηγική αντιμετώπισης και προσομοίωση

ΚΕΦΑΛΑΙΟ 4

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΝΗΜΑΤΩΝ ΣΤΟ ΕΣΩΤΕΡΙΚΟ ΜΙΑΣ ΔΙΕΡΓΑΣΙΑΣ

Στην ενότητα αυτή παρουσιάζεται ο πυρήνας της εργασίας, δηλαδή ο προτεινόμενος μηχανισμός χρονοπρογραμματισμού νημάτων. Το φαινόμενο της εξόντωσης, όπως περιγράφηκε στο προηγούμενο κεφάλαιο δημιουργείται κάτω από συγκεκριμένες προϋποθέσεις και έχουν προταθεί διάφορες τεχνικές που αποσκοπούν στη μείωσή του ή στην εξάλειψή του, συνήθως, λαμβάνοντας υπόψη ειδικά χαρακτηριστικά των εφαρμογών που το προκαλούν ή του λειτουργικού συστήματος. Ο προτεινόμενος μηχανισμός είναι μια τεχνική γενικού σκοπού, η οποία αποσκοπεί στη μείωση (έως και την εξάλειψη, υπό προϋποθέσεις) του φαινομένου, υλοποιώντας ένα μηχανισμό απόδοσης προτεραιοτήτων νημάτων σε επίπεδο διεργασίας, χρησιμοποιώντας παραμέτρους που κάθε διεργασία έχει ούτως ή άλλως στη διάθεσή της.

Η υλοποίηση του μηχανισμού αυτού μπορεί να πραγματοποιηθεί μόνο στην περίπτωση που υπάρχου δύο επίπεδα χρονοπρογραμματισμού από το λειτουργικό σύστημα:

1. Χρονοπρογραμματισμός διεργασιών (process scheduling) που γίνεται από το λειτουργικό σύστημα και αναθέτει τον επεξεργαστή σε μία διεργασία, το οποίο λειτουργεί με συμβατικό τρόπο (για παράδειγμα οι διεργασίες μπορεί να χρονοπρογραμματίζονται εκ περιτροπής ή σύμφωνα με ουρές προτεραιοτήτων)

2. Χρονοπρογραμματισμός νημάτων στο εσωτερικό μιας διεργασίας (in-process thread scheduling), που αποφασίζει ποιο νήμα θα εκτελεστεί από τα διαθέσιμα νήματα της διεργασίας που πήρε το κβάντο.

Ο δεύτερος μηχανισμός, εν γένει, στα λειτουργικά συστήματα υλοποιείται με εκ περιτροπής εκτέλεση. Στην παρούσα εργασία προτείνεται ένας μηχανισμός που χρησιμοποιεί προτεραιότητες, τις οποίες αποδίδει σε κάθε νήμα (από τα διαθέσιμα νήματα της τρέχουσας διεργασίας) σύμφωνα με τις τιμές των εξής παραμέτρων (τις οποίες θα ονομάζουμε και *εξωτερικές συνθήκες*):

1. Καθολικός λόγος σφάλματος σελίδων (Page Fault Ratio), το οποίο θα συμβολίζουμε με PFR, και $PFR\ i$ για το διάστημα $i-\Delta i, i$

2. Χρησιμοποίηση επεξεργαστή (CPU Utilization), το οποίο θα συμβολίζουμε με CPU , και $CPU\ \ i$ για το διάστημα $i-\Delta i, i$

Ο χρονοπρογραμματισμός διεργασιών όπως υλοποιείται στα σύγχρονα λειτουργικά συστήματα τιμωρεί τις διεργασίες που παρουσιάζουν χαμηλή χρησιμοποίηση του επεξεργαστή ή/και αυξημένο λόγο σφαλμάτων σελίδων, ειδικά εάν το σύστημα στο σύνολο του έχει μεταβεί σε κακή κατάσταση, μειώνοντας την προτεραιότητά τους.

Ο εσωτερικός χρονοπρογραμματιστής νημάτων προσπαθεί να χρονοπρογραμματίσει το νήμα που είναι πιο κατάλληλο ανάλογα με τις εξωτερικές συνθήκες. Πιο συγκεκριμένα διαχωρίζουμε τη λειτουργία του συστήματος σε **κανονική** και **επιβαρυμένη**, ανάλογα με τις προαναφερθείσες μετρικές.

4.1 Καταστάσεις συστήματος

Οι δύο καταστάσεις που περιγράψαμε εξαρτώνται από τις τιμές της χρησιμοποίησης του επεξεργαστή και του ρυθμού σφαλμάτων σελίδων. Έτσι, ορίζουμε δύο κατώφλια:

- CPU_LOWER_BOUND, το οποίο είναι το ελάχιστο κατώφλι ανεκτής χρησιμοποίησης του επεξεργαστή
- PFR_UPPER_BOUND, το οποίο είναι το μέγιστο κατώφλι ανεκτού ρυθμού σφαλμάτων σελίδων

Εάν η χρησιμοποίηση του επεξεργαστή είναι μικρότερη από το CPU_LOWER_BOUND **και** ο λόγος σφαλμάτων σελίδων είναι μεγαλύτερος από το PFR_UPPER_BOUND, τότε η κατάσταση του συστήματος θεωρείται **επιβαρυμένη**, σε διαφορετική περίπτωση **κανονική**.

Οι τιμές των κατωφλίων, είναι παράμετροι του αλγορίθμου. Ο λόγος που θα πρέπει να παραβιάζονται και τα δύο κατώφλια είναι ότι, μπορεί η παραβίαση μίας εκ των δύο συνθηκών να συμβαίνει για άλλο λόγο. Πιο συγκεκριμένα, όταν μια νέα διεργασία φορτώνεται στη μνήμη, σίγουρα έχουμε μια – περιορισμένη χρονική διάρκειας – αύξηση του ρυθμού σφαλμάτων σελίδων. Επιπλέον, όταν το σύστημα δεν έχει διεργασίες που χρειάζονται συνεχή εκτέλεση, τότε μπορεί να μειώνεται πολύ η χρησιμοποίηση του επεξεργαστή χωρίς να σημαίνει ότι έχει επιβαρυνθεί η λειτουργία του συστήματος.

4.2 Συνεισφορά νήματος

Το πιο σημαντικό σημείο του μηχανισμού είναι ο υπολογισμός της *συνεισφοράς* του κάθε νήματος στην τρέχουσα κατάσταση του συστήματος. Επειδή ένα σύστημα είναι αδύνατον να γνωρίζει τη μελλοντική συμπεριφορά μιας διεργασίας ή ενός νήματος, προτείνεται ένας μηχανισμός υπολογισμού της συνεισφοράς του κάθε νήματος και της κάθε διεργασίας στην τρέχουσα κατάσταση του συστήματος. Πιο συγκεκριμένα, υπολογίζεται η συνεισφορά του κάθε νήματος σε κάθε μία από τις δύο βασικές μετρικές που προαναφέραμε δηλαδή, τη χρησιμοποίηση επεξεργαστή και το λόγο σφαλμάτων σελίδων.

Συνεισφορά στη μεταβολή της CPU χρήσης (Contribution in CPU Utilization, ccu)

Η συνεισφορά ενός νήματος στη μεταβολή της χρησιμοποίησης του επεξεργαστή είναι ο εκθετικός μέσος όρος της μεταβολής που παρατηρήθηκε στην καθολική χρησιμοποίηση του επεξεργαστή κατά την εκτέλεση ενός κβάντου από το δεδομένο νήμα.

$$ccu\ t,i\ = \alpha \cdot ccu\ t,i-1\ +\ 1-\alpha\ \cdot\ CPU\ i\ -CPU\ i-1$$

Συνεισφορά στη μεταβολή της PFR (Contribution in PFR, $cpfr$)

Αντίστοιχα, η συνεισφορά ενός νήματος στη μεταβολή του ρυθμού σφαλμάτων σελίδων είναι ο εκθετικός μέσος όρος της μεταβολής που παρατηρήθηκε στο λόγο σφαλμάτων σελίδων κατά την εκτέλεση ενός κβάντου από το δεδομένο νήμα.

$$cpfr\ t,i\ = \alpha \cdot cpfr\ t,i-1\ +\ 1-\alpha\ \cdot\ PFR\ i\ -PFR\ i-1$$

Επίσης υπολογίζονται οι αντίστοιχες τιμές για κάθε διεργασία ($cpfr\ p,i$ και $ccu\ p,i$), οι οποίες χρησιμοποιούνται για τη εξομάλυνση των αντίστοιχων τιμών των νημάτων, ώστε να είναι πιο φυσική η μεταξύ τους σύγκριση.

4.3 Υπολογισμός προτεραιότητας νήματος

Σε κάθε κβάντο, και αφού επιλεγεί η διεργασία που έχει το κβάντο αυτό στη διάθεσή της, με χρήση των παραπάνω τιμών υπολογίζεται η προτεραιότητα κάθε νήματος της διεργασίας αυτής, ώστε, τελικά, να επιλεγεί το νήμα με τη μεγαλύτερη προτεραιότητα (δηλαδή την μικρότερη αριθμητική τιμή) για να εκτελεστεί.

Η προτεραιότητα του κάθε νήματος εξαρτάται από τις εξωτερικές συνθήκες και από τις συνεισφορές του νήματος και δίνεται από τη συνάρτηση απόδοσης προτεραιότητας νήματος (thread priority function):

$$tpf\ \ t,i\ = sign\ \ sc\ \cdot\left(w_1 \cdot \frac{ccu\ \ t,i}{ccu\ \ p,i} - w_2 \cdot \frac{cpf\ \ t,i}{cpf\ \ p,i} \right),\ w_i > 0$$

όπου το $sign\ sc$ είναι μια συνάρτηση που παίρνει τιμές -1 και +1, ανάλογα με την κατάσταση στην οποία βρίσκεται το σύστημα. Πιο συγκεκριμένα, κατά το χρονοπρογραμματισμό νημάτων επιλέγεται το νήμα με τη μικρότερη αριθμητική τιμή προτεραιότητας. Η συνάρτηση απόδοσης προτεραιότητας νημάτων, επί της ουσίας, βαθμολογεί τη συμπεριφορά του νήματος ως προς τη συμμετοχή του σε πιθανή αύξηση ή μείωση της καθολική χρησιμοποίησης του επεξεργαστή καθώς και των σφαλμάτων σελίδων.

Όσο πιο “θετική” είναι η συνεισφορά ενός νήματος τόσο πιο μεγάλη θα είναι οι τιμή του $\frac{ccu\ \ t,i}{ccu\ \ p,i}$ και τόσο πιο μικρή θα είναι η τιμή του $\frac{cpf\ \ t,i}{cpf\ \ p,i}$. Έτσι θέλοντας να υπολογίσουμε τη συμπεριφορά ενός νήματος υπολογίζουμε τη σταθμισμένη διαφορά των δύο τιμών. Ένα νήμα που συνεισφέρει θετικά στην απόδοση του συστήματος θα έχει “μεγάλη τιμή” και ένα νήμα που συνεισφέρει αρνητικά στην απόδοση του συστήματος θα έχει “μικρή τιμή”.

Η συνάρτηση $sign\ sc$ χρησιμοποιείται γιατί ανάλογα με τις εξωτερικές συνθήκες θέλουμε να προωθήσουμε νήματα με εκ διαμέτρου αντίθετη συμπεριφορά. Όταν το σύστημα βρίσκεται σε **κανονική** κατάσταση η συνάρτηση προσήμου έχει τιμή +1 γιατί θέλουμε να δώσουμε προτεραιότητα σε νήματα με “μικρή τιμή”, δηλαδή με κακή συνεισφορά, τα οποία όταν επιβαρύνεται το σύστημα δεν επιλέγονται προς εκτέλεση. Αντίθετα, όταν το σύστημα βρίσκεται σε **επιβαρυμένη** κατάσταση η τιμή της συνάρτησης προσήμου είναι -1 με σκοπό να δοθεί προτεραιότητα σε νήματα με

«μεγάλη τιμή», δηλαδή με καλή συνεισφορά στις μετρικές που περιγράφουν την κατάσταση του συστήματος.

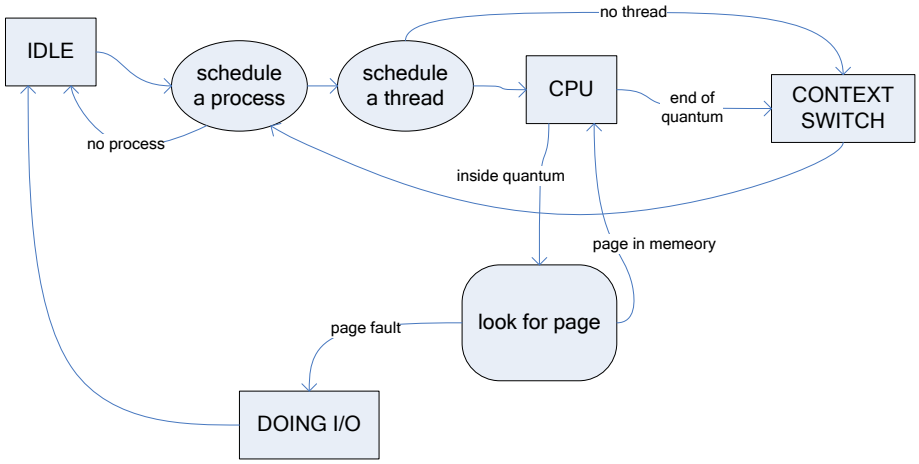
Σκοπός του εσωτερικού χρονοπρογραμματιστή είναι, όταν το σύστημα είναι σε **επιβαρυμένη** κατάσταση να χρονοπρογραμματίσει τα νήματα με την πιο καλή συμπεριφορά ώστε να μην “τιμωρηθεί” από τον εξωτερικό χρονοπρογραμματιστή (διεργασιών), ο οποίος σε περίπτωση επιβάρυνσης του συστήματος δίνει πιο μεγάλη προτεραιότητα σε διεργασίες με λίγα σφάλματα σελίδων.

Με αυτόν τον τρόπο επιτυγχάνουμε ένα κατανεμημένο σύστημα αντιμετώπισης του φαινομένου της εξόντωσης, το οποίο δεν απαιτεί επικοινωνία μεταξύ των διεργασιών και η μοναδική πληροφορία που χρειάζεται κάθε διεργασία για να υλοποιήσει το μηχανισμό αυτό είναι η καθολική χρησιμοποίηση του επεξεργαστή και ο καθολικός λόγος σφαλμάτων σελίδων, μετρικές που, γενικά, είναι διαθέσιμες σε κάθε διεργασία.

ΚΕΦΑΛΑΙΟ 5

ΠΡΟΣΟΜΟΙΩΣΗ ΕΚΤΕΛΕΣΗΣ ΕΠΕΞΕΡΓΑΣΤΗ

Σε αυτήν την ενότητα παρουσιάζεται η δομή της προσομοίωσης της λειτουργίας του επεξεργαστή. Στο παρακάτω σχήμα φαίνεται η εναλλαγή καταστάσεων που πραγματοποιείται κατά την εκτέλεση της προσομοίωσης.



Σχήμα 3: Καταστάσεις του προσομοιωτή του επεξεργαστή

Ο χρόνος της προσομοίωσης χωρίζεται στην ελάχιστη δυνατή μονάδα (tick) και σε κάθε βήμα προσομοίωσης περνά ένα tick. Το βήμα της προσομοίωσης είναι ένα μονοπάτι στον παραπάνω γράφο που αρχίζει από μία κατάσταση και καταλήγει πάλι σε κατάσταση (καταστάσεις είναι τα 4 ορθογώνια παραλληλόγραμμα). Οι υπόλοιποι κόμβοι είναι ενέργειες και τα βέλη είναι μεταβάσεις (είτε υπό συνθήκη είτε ανεξαρτήτως συνθήκης). Η ενέργεια “schedule a process” αναπαριστά τον εξωτερικό χρονοπρογραμματιστή διεργασιών ενώ η ενέργεια “schedule a thread” αναπαριστά τον χρονοπρογραμματιστή νημάτων που επιλέγει, εντός μιας διεργασίας, ποιο νήμα θα εκτελεστεί. Τέλος, η ενέργεια “look for page” αναζητά την τρέχουσα σελίδα στη μνήμη και ανάλογα με το εάν συμβεί σφάλμα σελίδας ή όχι οδηγούμαστε στην κατάλληλη κατάσταση (DOING I/O ή CONTEXT SWITCH).

Τα δυνατά μονοπάτια (βήματα προσομοίωσης), λοιπόν, έχουν τις εξής καταστάσεις ως αρχική και τελική:

1. IDLE → CPU
2. CPU → CPU
3. CPU → CONTEXT SWITCH
4. CPU → DOING I/O
5. CONTEXT SWITCH → CPU
6. CONTEXT SWITCH → IDLE
7. DOING I/O → IDLE

Παράλληλα με την ανωτέρω διαδικασία, μια άλλη διαδικασία παράγει διεργασίες που τοποθετούνται στην ουρά έτοιμων διεργασιών. Κάθε φορά που ένα νήμα – ή μία διεργασία – δημιουργείται, υπολογίζεται η χρονική στιγμή κατά την οποία θα δημιουργηθεί το επόμενο νήμα. Έτσι, σε κάθε βήμα προσομοίωσης συγκρίνεται η τρέχουσα χρονική στιγμή με τη χρονική στιγμή που έχει αποθηκευτεί ως η στιγμή της επόμενη δημιουργίας νήματος/διεργασίας. Εάν, οι δύο τιμές συμπίπτουν ενεργοποιείται η διαδικασία δημιουργίας νέου νήματος όπως περιγράφεται παρακάτω.

5.1 Χρονοπρογραμματισμός διεργασιών

Στην ενέργεια “schedule a process” καλείται ο χρονοπρογραμματιστής διεργασιών του συστήματος που έχει επιλεχθεί μια Multi-level Feedback Queue, η οποία είναι η δομή που επιλέγεται στα συστήματα Unix. Παράμετροι του συστήματος είναι:

- Το πλήθος των προτεραιοτήτων (και άρα και των ουρών)
- Η προτεραιότητα βάσης (base priority) δηλαδή η προτεραιότητα που έχει κάθε διεργασία κατά τη δημιουργία της

Η επιλογή της διεργασίας προς εκτέλεση γίνεται επιλέγοντας, από την πρώτη μη άδεια ουρά τη διεργασία που βρίσκεται στην κορυφή. Με αυτόν τον τρόπο μπορεί να προσομοιωθεί τόσο μία ουρά χωρίς προτεραιότητες (απλή εκ περιτροπής εκτέλεση) σε περίπτωση που όλες οι διεργασίες έχουν την ίδια προτεραιότητα, όσο και περισσότερες ουρές ανάλογα με τις τιμές των παραμέτρων που θα επιλέξει ο προγραμματιστής.

5.2 Χρονοπρογραμματισμός νημάτων

Η ενέργεια “schedule a thread” καλείται εσωτερικά της εκάστοτε διεργασίας με την έννοια ότι καλείται αφού έχει επιλεχθεί ποια διεργασία έχει το κβάντο στη διάθεσή της. Έτσι, όταν μια διεργασία έχει το κβάντο καλείται να “αποφασίσει” ποιο νήμα της θα το χρησιμοποιήσει.

Το νήμα το οποίο επιλέγεται εξαρτάται από μία σειρά από παραμέτρους:

- Την κατάσταση του συστήματος
  - κανονική, ή
  - επιβαρυμένη
- Την επιλογή αλγορίθμου
  - εκ περιτροπής, ή
  - το προτεινόμενο σχήμα (βλέπε κεφάλαιο 4)
- Στην περίπτωση που χρησιμοποιείται το προτεινόμενο σχήμα, η επιλογή νήματος εξαρτάται, ακόμα, από τη συνεισφορά του νήματος στις μετρικές που περιγράφουν τις συνθήκες του συστήματος (χρησιμοποίηση επεξεργαστή και λόγος σφαλμάτων σελίδων)

Ο εκ περιτροπής αλγόριθμος χρονοπρογραμματισμού νημάτων επιλέγει ανά κβάντο το νήμα που βρίσκεται πρώτο στην ουρά έτοιμων νημάτων τις διεργασίες, το δίνει στον επεξεργαστή, και όταν ολοκληρωθεί η εκτέλεσή του το προσθέτει στο τέλος της ουράς αυτής. Από την άλλη μεριά το προτεινόμενο σχήμα, όπως έχουμε δει στο προηγούμενο κεφάλαιο, αναθέτει σε κάθε ένα από τα έτοιμα – προς εκτέλεση – νήματα μία αριθμητική τιμή προτεραιότητας και επιλέγει το νήμα που μεγιστοποιεί ή ελαχιστοποιεί αυτήν την τιμή ανάλογα με τις συνθήκες του συστήματος.

Κάθε φορά που ένα νήμα εκτελείται αποθηκεύονται σε τοπικές (ως προς τη διεργασία στην οποία ανήκει το νήμα) μεταβλητές οι μεταβολές των τιμών των μετρικών του συστήματος ώστε να υπολογιστεί ο γεωμετρικός μέσος όρος των συνεισφορών του νήματος.

Κατά τη διάρκεια της προσομοίωσης της εκτέλεσης ενός νήματος ζητείται από τη μνήμη η κατάλληλη σελίδα της διεργασίας στην οποία ανήκει το νήμα. Αυτή η διαδικασία επαναλαμβάνεται μερικές φορές (αφήνοντας τη δυνατότητα στον προγραμματιστή να τροποποιήσει τον αριθμό αυτό) σε κάθε κβάντο εκτέλεσης.

5.3 Αναζήτηση σελίδας

Στην ενέργεια αυτή αναζητείται μια σελίδα στην κύρια μνήμη. Η τρέχουσα διεργασία αφού υπολογιστεί ποια είναι η νέα σελίδα ζητάει από τη μονάδα διαχείρισης μνήμης τη σελίδα αυτή και εάν η σελίδα είναι διαθέσιμη προχωράει στην εκτέλεση.

Σε διαφορετική περίπτωση έχουμε σφάλμα σελίδας και η προσομοίωση μεταπίπτει σε κατάσταση DOING I/O. Πριν μεταπέσει στην κατάσταση αυτή, το νήμα έχει σταματήσει τη λειτουργία του και για ένα χρονικό διάστημα που υπολογίζεται πιθανοθεωρητικά (το χρονικό διάστημα αυτό μοντελοποιεί το χρόνο που χρειάζεται για να μεταφερθεί η σελίδα από το σκληρό δίσκο στην κύρια μνήμη) το νήμα δεν είναι διαθέσιμο προς χρονοπρογραμματισμό.

Άμεσα, δηλαδή στην επόμενη ελάχιστη μονάδα του χρόνου, ο επεξεργαστής μεταβαίνει σε κατάσταση CONTEXT SWITCH προσπαθώντας να βρει την επόμενη διεργασία προς χρονοπρογραμματισμό.

5.4 Δημιουργία νέου νήματος

Η προσομοίωση αυτή μας παρέχει τη δυνατότητα να παράγουμε ένα αρχικό φόρτο εργασίας (διεργασίες) αλλά και τη δυνατότητα να παράγουμε δυναμικά διεργασίες κατά τη διάρκεια της προσομοίωσης της λειτουργίας του επεξεργαστή. Για το λόγο αυτό υπάρχει μια χωριστή διαδικασία που παράγει νήματα.

Κάθε φορά που ένα νήμα δημιουργείται γίνονται τα εξής:

- Με βάση το αποτέλεσμα ενός πειράματος Bernoulli, το νέο νήμα μπορεί να ανήκει σε μία διεργασία ή να είναι το πρώτο νήμα μιας άλλης διεργασίας.
- Εάν το νήμα ανήκει σε υπάρχουσα διεργασία επιλέγεται μια διεργασία από τις τρέχουσες (βάσει της ομοιόμορφης κατανομής)
- Με βάση ένα άλλο πείραμα Bernoulli η διεργασία αυτή είναι ένα μια διεργασία-δαίμονας, η οποία έχει άπειρο χρόνο εκτέλεσης (συγκεκριμένα θα τρέχει μέχρι το τέλος της προσομοίωσης)
- Εάν δεν έχουμε τη δημιουργία μιας διεργασίας-δαίμονα, υπολογίζεται ο πραγματικός χρόνος εκτέλεσης του νήματος (βάσει της εκθετικής κατανομής)
- Υπολογίζεται η χρονική στιγμή κατά την οποία θα δημιουργηθεί το επόμενο νήμα. Αυτή τιμή υπολογίζεται προσθέτοντας στην τρέχουσα χρονική στιγμή μία τυχαία μεταβλητή που κατανέμεται με βάση την εκθετική (εκθετικός χρόνος μεταξύ διαδοχικών αφίξεων)

5.5 Τυχαίες μεταβλητές που χρησιμοποιούνται στην προσομοίωση

Στην προσομοίωση που περιγράφεται παραπάνω χρησιμοποιούνται διάφορες τυχαίες μεταβλητές που παίζουν πολύ σημαντικό ρόλο μοντελοποιώντας τη συμπεριφορά διαφόρων παραμέτρων του συστήματος. Πιο συγκεκριμένα χρησιμοποιούνται οι παρακάτω τυχαίες μεταβλητές:

- Χρόνος μεταξύ της δημιουργίας νέων νημάτων
- Χρόνος εκτέλεσης που απαιτεί ένα νήμα μέχρι να τερματίσει
- Εάν το νέο νήμα θα ανήκει σε νέα ή σε ήδη υπάρχουσα διεργασία
- Μέγεθος (σε σελίδες μιας διεργασίας)
- Εάν κάποιο νήμα θα σταματήσει την εκτέλεσή του αναμένοντας κάποια άλλη λειτουργία/πληροφορία
- Χρόνος κατά τον οποίο ένα νήμα θα παραμένει σταματημένο
- Απόσταση (σε σελίδες) από την τρέχουσα σελίδα όταν πρόκειται να γίνει το επόμενο βήμα εκτέλεσης του εκάστοτε νήματος

5.5.1 Δημιουργία νέων νημάτων/διεργασιών

Η διαδικασία που δημιουργεί νέα νήματα, μετά από κάθε δημιουργία ενός νήματος υπολογίζει το χρονικό διάστημα που θα μεσολαβήσει μέχρι τη δημιουργία ενός επόμενου νήματος. Αυτό το χρονικά διάστημα υπολογίζεται με τη χρήση της εκθετικής κατανομής [5]. Ανάλογα με χρόνους που μεσολαβούν μέχρι την άφιξη ενός πακέτου στη θεωρία ουρών, μοντελοποιείται και η δημιουργία νημάτων (και διεργασιών). Η μέση τιμή του χρόνου μεταξύ δημιουργίας δύο νημάτων δίνεται από την παράμετρο *meanInterCreationTime* αυξημένη κατά ένα (*meanInterCreationTime*+1), και αυτό γιατί μια τυχαία μεταβλητή που η οποία κατανέμεται σύμφωνα με την εκθετική κατανομή μπορεί να έχει ως πιθανή τιμή το 0. Η προσομοίωση του συστήματος, όμως, βασίζεται σε προσομοίωση διακριτού χρόνου και στο χρόνο μεταξύ δημιουργίας νημάτων υπολογίζεται σε πόσες υποδιαιρέσεις του χρόνου (ticks) θα δημιουργηθεί ένα νέο νήμα, γεγονός που πρέπει να συμβεί στο μέλλον (δηλαδή τουλάχιστον 1 tick μετά από το παρόν).

Έτσι, σύμφωνα με αυτούς τους χρόνους δημιουργούνται νέα νήματα. Μία άλλη τυχαία απόφαση είναι εάν το νέο νήμα θα ανήκει σε μια υπάρχουσα διεργασία ή σε μία νέα διεργασία. Αυτό το ερώτημα απαντάται με την διεξαγωγή ενός πειράματος Bernoulli το οποίο με πιθανότητα *probabilityOfNewProcess* αποφασίζει ότι θα δημιουργηθεί μια νέα διεργασία, ενώ με τη συμπληρωματική πιθανότητα αποφασίζει ότι το νέο νήμα θα ανήκει σε μία υπάρχουσα διεργασία. Η υπάρχουσα διεργασία στην οποία θα προστεθεί το νήμα επιλέγεται τυχαία με ίση πιθανότητα μεταξύ όλων των υπαρχουσών διεργασιών.

Με τη δημιουργία μίας νέας διεργασίας υπάρχει ένα ακόμη μέγεθος που πρέπει να επιλεχθεί η τιμή του, το μέγεθος της διεργασίας (σε σελίδες). Χρησιμοποιείται η εκθετική κατανομή με μέση τιμή την τιμή της παραμέτρου *meanPagesPerProcess*+1, αφού δεν έχει νόημα να υπάρχει μία διεργασία με μηδενικό πλήθος σελίδων.

5.5.2 Χρόνος εκτέλεσης που απαιτεί ένα νήμα μέχρι να τερματίσει

Μία πολύ σημαντική τυχαία μεταβλητή είναι η διάρκεια του κάθε νήματος. Το μέγεθος αυτό κατανέμεται σύμφωνα με την εκθετική κατανομή [5] και η μέση τιμή δίνεται από την παράμετρο *meanThreadExecutionTime*. Η διάρκεια της εκτέλεσης της κάθε διεργασίας προκύπτει έμμεσα από την χρονική στιγμή έναρξης του πρώτου νήματος και τη χρονική στιγμή λήξης του τελευταίου νήματος.

5.5.3 Διακοπή εκτέλεσης νήματος

Ένα νήμα μπορεί να πρέπει να διακόψει τη λειτουργία του για να δεχθεί κάποια δεδομένα, λόγω συγχρονισμού ή λόγω εκτέλεσης εισόδου/εξόδου. Το ενδεχόμενο της διακοπής αποφασίζεται με ένα πείραμα Bernoulli με πιθανότητα *probabilityOfBlockThread*. Η διάρκεια της διακοπής της λειτουργίας του νήματος είναι μια νέα τυχαία μεταβλητή που κατανέμεται σύμφωνα με την εκθετική κατανομή και η μέση τιμή δίνεται από την παράμετρο *meanThreadBlockedTime*.

5.5.4 Υπολογισμός νέας σελίδας

Σε κάθε βήμα της προσομοίωσης υπολογίζεται η επόμενη σελίδα που πρόκειται να χρειαστεί κάθε νήμα προκειμένου να εκτελεστεί. Η επόμενη σελίδα ανά πάσα στιγμή

είναι μία τυχαία μεταβλητή που κατανέμεται σύμφωνα με την κατανομή Gaussian, έχει μέση τιμή την τρέχουσα σελίδα και η διασπορά της δίνεται από την παράμετρο *locality* [5]. Σε περίπτωση που κατά τον υπολογισμό της Gaussian φύγουμε έξω από το διάστημα [0, <αριθμός σελίδων διεργασίας>], τότε τα δεδομένα αναδιπλώνονται μέσα στο διάστημα αυτό με χρήση του τελεστή υπολοίπου.

ΚΕΦΑΛΑΙΟ 6

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

Στην ενότητα αυτή παρουσιάζονται τα αποτελέσματα της προσομοίωσης. Σε πρώτη φάση, κύριος στόχος, ήταν η επιβεβαίωση της ορθής λειτουργίας του προσομοιωτή. Όπως φαίνεται και από την περιγραφή της προσομοίωσης στο προηγούμενο κεφάλαιο, ο προσομοιωτής ενσωματώνει μια πληθώρα παραμέτρων και η ορθή λειτουργία για όλες τις πιθανές τιμές είναι η πρώτη φάση δοκιμών που έλαβαν χώρα. Στη συνέχεια προσομοιώθηκε η δημιουργία του φαινομένου της εξόντωσης, προκειμένου να δοκιμαστεί το προτεινόμενο σχήμα αντιμετώπισής του.

6.1 Εμφάνιση φαινομένου εξόντωσης

Η εμφάνιση της εξόντωσης είναι κάτι που απαιτεί ένα σύνολο από «ατυχή» συμβάντα να λάβουν χώρα. Πρέπει, λοιπόν, να ικανοποιούνται οι παρακάτω συνθήκες:

- Να υπάρχουν πολλές διεργασίες που εκτελούνται ταυτόχρονα
- Να είναι τουλάχιστον το μεγαλύτερο κομμάτι της κύριας μνήμης δεσμευμένο με σελίδες από διεργασίες που δυνάμει θα εκτελεστούν
- Να υπάρχουν κάποιες διεργασίες που δεν έχουν μεγάλη τοπικότητα ώστε να ζητούν συχνά νέες σελίδες ή να έχει γεμίσει πλήρως η μνήμη ώστε να πρέπει να αντικατασταθούν κάποιες σελίδες.

6.2 Πειράματα

Τα πειράματα που θα παρουσιαστούν είναι όλα πειράματα προσομοίωσης και μπορούν να χωριστούν σε δύο κατηγορίες. Στα πειράματα με πανομοιότυπο φόρτο εργασίας, ο οποίος δημιουργήθηκε πριν αρχίσει η εκτέλεση των διεργασιών και πειράματα κατά το οποία υπήρχε δημιουργία νημάτων ή και διεργασιών κατά τη διάρκεια της λειτουργίας του επεξεργαστή που προσομοιώνεται. Για να παραχθεί το φαινόμενου της εξόντωσης επιλέχθηκε μικρό μέγεθος της μνήμης και μείωση της τοπικότητας ώστε να αποφευχθεί η δημιουργία πολύ μεγάλου πλήθος διεργασιών (ήδη με 40 διεργασίες μπορούμε να πετύχουμε εξόντωση). Επίσης, κατά την προσομοίωση δημιουργούνται και διεργασίες «δαίμονες» που συνεχίζουν να εκτελούνται συνεχώς και, μάλιστα, οι σελίδες τους δεν μπορούν να απομακρυνθούν από τη μνήμη, γιατί προσομοιώνουν κρίσιμες διεργασίες για το λειτουργικό.

Σε κάθε μία από τις προαναφερθείσες κατηγορίες έγινε μια σειρά πειραμάτων για διάφορες τιμές παραμέτρων, τόσο με απλό χρονοπρογραμματισμό νημάτων (εκ περιτροπής) όσο και με χρήση του προτεινόμενου μηχανισμού. Στις επόμενες σελίδες παρουσιάζονται αναλυτικά τα πειράματα που έλαβαν χώρα, τα αποτελέσματά τους καθώς και τα συμπεράσματα που προέκυψαν από αυτά.

6.2.1 Πειράματα με φόρτο εργασίας ορισμένο μόνο στην αρχή της προσομοίωσης

6.2.1.1 Σύνολο πειραμάτων A1

Στο σύνολο πειραμάτων A1 πραγματοποιήθηκε μια σειρά από 10 πειράματα, όπου σε κάθε ένα δημιουργήθηκαν 40 διεργασίες και ο χρόνος προσομοίωσης ήταν 10.000 sec. Η μέση διάρκεια εκτέλεσης νήματος ήταν 100.000 ticks (100 sec), μέσο πλήθος σελίδων ανά διεργασία 30 και όρια για την εκτέλεση του προτεινόμενου αλγορίθμου έναντι του Round Robin, PFR_UPPER_BOUND = 0.1 και CPU_LOWER_BOUND = 0.49. Το μέγεθος μνήμης που επιλέχθηκε προκειμένου να δημιουργηθεί συμφόρηση ήταν 100 σελίδες.

Το κάθε πείραμα πραγματοποιήθηκε με χρήση τόσο του χρονοπρογραμματισμού νημάτων εκ περιτροπής, όσο και του προτεινόμενου χρονοπρογραμματισμού νημάτων. Στον παρακάτω πίνακα, οι τρεις πρώτες σειρές αναφέρονται σε μεγέθη κοινά και στις δύο εκδοχές των πειραμάτων, δηλαδή στη μέση τιμή διεργασιών, διεργασιών δαιμόνων και νημάτων που παρήχθησαν, ενώ οι υπόλοιπες σειρές αναφέρονται στα μεγέθη μέσω των οποίων συγκρίνουμε τους δύο μηχανισμούς. Το πράσινο χρώμα υποδηλώνει καλύτερη τιμή ενώ το κόκκινο χειρότερη τιμή.

Πίνακας 2: Στατιστικά του πειράματος A1

Παράμετρος	Round Robin	Proposed
Processes Created	40,00	40,00
Daemon Processes Created	2,00	2,00
Thread Created	194,70	194,70
Processed Finished	13,00	12,80
Thread Finished	33,60	65,10
Average CPU%	60,71%	81,09%
Average IO%	18,07%	6,51%
Average Context Switch%	21,22%	12,39%
Average PFR%	31,40%	8,66%
Thread Penalty Ratio	88,93	86,46
Process Penalty Ratio	44,73	54,21

Παρατηρώντας τον παραπάνω πίνακα βγαίνουν τα εξής συμπεράσματα:

- Ο προτεινόμενος μηχανισμός χρονοπρογραμματισμού νημάτων οδηγεί σε σημαντική αύξηση του ποσοστού του πραγματικού χρόνου που λειτουργεί ο επεξεργαστής (81,09% από 60,71%)
- Οδηγεί σε σημαντική μείωση του ρυθμού σφαλμάτων σελίδων (8,66% από 31,40%)
- Έχει ελαφρώς αρνητική επίπτωση στο throughput διεργασιών (στα όρια του στατιστικού λάθους)
- Έχει ελαφρώς θετική επίπτωση στο λόγο ποινής εκτέλεσης νημάτων (Thread Penalty Ratio) - στα όρια του στατιστικού λάθους
- Έχει, ως αντιστάθμισμα, την αύξηση του λόγου ποινής εκτέλεσης διεργασιών (Process Penalty Ratio) κατά 25,6% (56,14 έναντι 44,7)

6.2.1.2 Σύνολο πειραμάτων Α2

Στο σύνολο πειραμάτων Α2 επιλέχθηκαν να παρουσιαστούν προσομοιώσεις κατά τις οποίες το σύστημα είναι πολύ πιο επιβαρυμένο από ότι στο σύνολο πειραμάτων Α1. Έτσι, ενώ σε όλες τις παραμέτρους διατηρήθηκαν οι ίδιες τιμές, το αρχικό πλήθος διεργασιών αυξήθηκε από 40 σε 80.

Πίνακας 3: Στατιστικά του πειράματος Α2

Παράμετρος	Round Robin	Proposed
Processes Created	80,00	80,00
Daemon Processes Created	4,30	4,30
Thread Created	377,20	377,20
Processed Finished	2,50	2,10
Thread Finished	9,00	16,80
Average CPU%	45,40%	50,52%
Average IO%	26,49%	23,97%
Average Context Switch%	28,11%	25,51%
Average PFR%	60,50%	48,94%
Thread Penalty Ratio	171,77	254,64
Process Penalty Ratio	142,05	198,93

Στον πίνακα που παρουσιάζεται στη συνέχεια φαίνονται οι αντίστοιχοι μέσοι όροι των δέκα πειραμάτων που έλαβαν με χώρα με πλήθος διεργασιών προς εκτέλεση 80, μέγεθος μνήμης 100 σελίδες και χρόνο προσομοίωσης 10.000 sec.

Τα συμπεράσματα που προκύπτουν εξετάζοντας τα παραπάνω αποτελέσματα είναι όμοια με τα προηγούμενα. Ο προτεινόμενος μηχανισμός:

- Οδηγεί σε σημαντική αύξηση του χρόνου χρησιμοποίησης του επεξεργαστή (50,52% έναντι 45,40%)
- Μειώνει το μέσο λόγο σφαλμάτων σελίδων από 60,50% σε 48,94%
- Έχει πολύ μικρή αρνητική επίδραση στο throughput διεργασιών
- Έχει σημαντική θετική επίδραση στο throughput νημάτων αυξάνοντας τα νήματα που ολοκληρώθηκαν από 9,00 σε 16,80 κατά μέσον όρο
- Έχει ως αντιστάθμισμα την αύξηση τόσο του λόγου ποινής εκτέλεσης νημάτων (κατά 48,2%: από 171,77 σε 254,64) όσο και του λόγου ποινής εκτέλεσης διεργασιών (κατά 40%: από 142,05 σε 198,93)

6.2.1.3 Σύνολο πειραμάτων A3

Πίνακας 4: Στατιστικά πειράματος A3

Παράμετρος	Round Robin	Proposed
Processes Created	120,00	120,00
Daemon Processes Created	5,30	5,30
Thread Created	585,50	585,50
Processed Finished	16,80	16,60
Thread Finished	62,20	80,00
Average CPU%	80,44%	89,88%
Average IO%	6,67%	0,70%
Average Context Switch%	12,90%	9,42%
Average PFR%	8,99%	0,86%
Thread Penalty Ratio	334,57	168,92
Process Penalty Ratio	136,32	141,80

Στο σύνολο πειραμάτων A3 παρουσιάζονται τα αποτελέσματα μια προσομοίωσης κατά την οποία το φαινόμενο της εξόντωσης ήταν σχεδόν ανύπαρκτο. Στην προσομοίωση

αυτή το μέγεθος της μνήμης ήταν 500 σελίδες και το πλήθος των διεργασιών που δημιουργήθηκαν 120. Ο χρόνος προσομοίωσης ήταν και πάλι 10.000 sec.

Τα αποτελέσματα του συνόλου πειραμάτων Α3 δείχνουν ότι ο προτεινόμενος μηχανισμός βοηθά στην καλύτερη εκτέλεση και κατά τη στιγμή που το φαινόμενο της εξόντωσης είναι πολύ ήπιο. Σε ένα σύστημα με 80,44% μέση χρησιμοποίηση του επεξεργαστή, η εφαρμογή του μηχανισμού απόδοσης προτεραιότητας νημάτων αύξησε τη μέση χρησιμοποίηση σε 89,88% και μείωσε το λόγο σφαλμάτων σελίδων από 8,99% σε 0,86%. Επιπλέον:

- Η ρυθμαπόδοση διεργασιών έμεινε πρακτικά σταθερό
- Η ρυθμαπόδοση νημάτων αυξήθηκε αφού κατά μέσον όρο ολοκληρώθηκαν 80 έναντι 62,2 νημάτων
- Ο λόγος ποινής εκτέλεσης νημάτων μειώθηκε από 334,57 σε 168,92, δηλαδή υπέστη μείωση κατά 49,51%
- Το μοναδικό αντιστάθμισμα ήταν ο λόγος ποινής εκτέλεσης διεργασιών που αυξήθηκε κατά μόλις 4,01% (από 136,32 σε 141,80)

6.2.2 Πειράματα με δυναμικό φόρτο εργασίας

Στην ενότητα αυτή παρουσιάζονται πειράματα κατά τα οποία υπήρχε δημιουργία φόρτου εργασίας πριν από την έναρξη της προσομοίωσης, όμοια με τα προηγούμενα πειράματα, αλλά και δημιουργία φόρτου κατά τη διάρκεια της εκτέλεσης της προσομοίωσης, σύμφωνα με τα μοντέλα που έχουν περιγραφεί στην προηγούμενη ενότητα.

6.2.2.1 Σύνολο πειραμάτων B1

Στο σύνολο πειραμάτων B1, προσομοιώθηκε ένα σύστημα στο οποίο ο αρχικός φόρτος διεργασιών ήταν 100 διεργασίες, και κατά την εκτέλεση δημιουργήθηκαν άλλες 100 διεργασίες. Ο χρόνος προσομοίωσης ήταν 10.000 sec και το μέγεθος της μνήμης ήταν 500 σελίδες. Στις λοιπές παραμέτρους (CPU_LOWER_BOUND, PFR_UPPER_BOUND, μέση διάρκεια εκτέλεσης νήματος) διατηρήθηκαν οι ίδιες τιμές.

Σε σχέση με τα προηγούμενα σύνολα πειραμάτων, το γεγονός ότι ο φόρτος διεργασιών είναι δυναμικός, μοιραία οδηγεί σε διαφορετικό πλήθος παραγόμενων νημάτων και διεργασιών δαιμόνων, αφού η γεννήτρια τυχαίων τιμών έχει χρησιμοποιηθεί στην πορεία για τον υπολογισμό πολλών τιμών τυχαίων μεταβλητών που το σύστημα χρειάζεται για να προχωρήσει η προσομοίωση του επεξεργαστή.

Πίνακας 5: Στατιστικά πειράματος B1

Παράμετρος	Round Robin	Proposed
Processes Created	200	200
Daemon Processes Created	10,90	10,40
Thread Created	976,40	979,80
Processed Finished	16,90	16,00
Thread Finished	63,90	71,80
Average CPU%	78,01%	87,09%
Average IO%	7,91%	2,55%
Average Context Switch%	14,08%	10,36%
Average PFR%	11,01%	3,52%
Thread Penalty Ratio	230,82	140,46
Process Penalty Ratio	115,32	126,05

Στον παραπάνω πίνακα φαίνονται τα στατιστικά αποτελέσματα του συνόλου πειραμάτων Β1 (που όπως και τα προηγούμενα αποτελείται από 10 επαναλήψεις του πειράματος με ίδιες παραμέτρους).

Η σύγκριση του προτεινόμενου μηχανισμού με την εκ περιτροπής εκτέλεση νημάτων για άλλη μία φορά έχει παρόμοια αποτελέσματα. Πιο συγκεκριμένα:

- Ο προτεινόμενος μηχανισμός οδήγησε σε ελάχιστη μείωση της ρυθμαπόδοσης διεργασιών (16 έναντι 16,90) και σε σημαντική αύξηση της ρυθμαπόδοσης νημάτων (71,80/979,80 έναντι 63,90/976,40, δηλαδή 7,33% έναντι 6,54%)
- Οδήγησε σε αύξηση του πραγματικού χρόνου εκτέλεσης από 78,01% σε 87,09% και προφανώς στη μείωση του χρόνου εισόδου/εξόδου από 7,91% σε 2,55% αλλά και του χρόνου εναλλαγής διεργασιών/νημάτων από 14,08% σε 10,36%
- Σε σημαντική μείωση του ρυθμού σφαλμάτων σελίδων από 11,01% σε 3,52%
- Σε σημαντική μείωση του λόγου ποινής εκτέλεσης νημάτων από 230,82 σε 140,46, δηλαδή μείωση κατά 39,15%
- Ενώ το μοναδικό αντιστάθμισμα ήταν η – μικρή – αύξηση του λόγου ποινής εκτέλεσης διεργασιών κατά 9,31%, από 115,32 σε 126,05.

6.2.2.2 Σύνολο πειραμάτων Β2

Στο σύνολο πειραμάτων Β2, προσομοιώθηκε ένα σύστημα στο οποίο ο αρχικός φόρτος διεργασιών ήταν 100 διεργασίες, και κατά την εκτέλεση δημιουργήθηκαν άλλες 300 διεργασίες. Ο χρόνος προσομοίωσης ήταν 100.000 sec και το μέγεθος της μνήμης ήταν 500 σελίδες. Στις λοιπές παραμέτρους (CPU_LOWER_BOUND, PFR_UPPER_BOUND, μέση διάρκεια εκτέλεσης νήματος) διατηρήθηκαν οι ίδιες τιμές.

Στο σύνολο πειραμάτων Β2 παρατηρήθηκε βελτίωση των συνθηκών του συστήματος με χρήση του προτεινόμενου μηχανισμού. Οι αρχικές παράμετροι των πειραμάτων αυτών ήταν τέτοιες ώστε να δημιουργήσουν αρκετά δύσκολες συνθήκες εκτέλεσης των διεργασιών. Έτσι η μέση χρησιμοποίηση του επεξεργαστή ήταν μόνο 49,29% και τα σφάλματα σελίδων ήταν 51,50% όταν χρησιμοποιήθηκε η εκ περιτροπής εκτέλεση νημάτων. Στα πειράματα στα οποία χρησιμοποιήθηκε ο προτεινόμενος μηχανισμός επιτεύχθηκε μέση χρησιμοποίηση του επεξεργαστή της τάξης του 58,60%, ενώ τα σφάλματα σελίδων μειώθηκαν στο 34,75% των αιτήσεων που έγιναν.

Πίνακας 6: Στατιστικά πειράματος Β2

Παράμετρος	Round Robin	Proposed
Processes Created	400	400
Daemon Processes Created	21,20	21,40
Thread Created	1980,30	2002,00
Processed Finished	39,50	50,70
Thread Finished	149,20	291,30
Average CPU%	49,29%	58,60%
Average IO%	24,26%	19,59%
Average Context Switch%	26,45%	21,82%
Average PFR%	51,50%	34,75%
Thread Penalty Ratio	668,40	904,70
Process Penalty Ratio	401,91	702,30

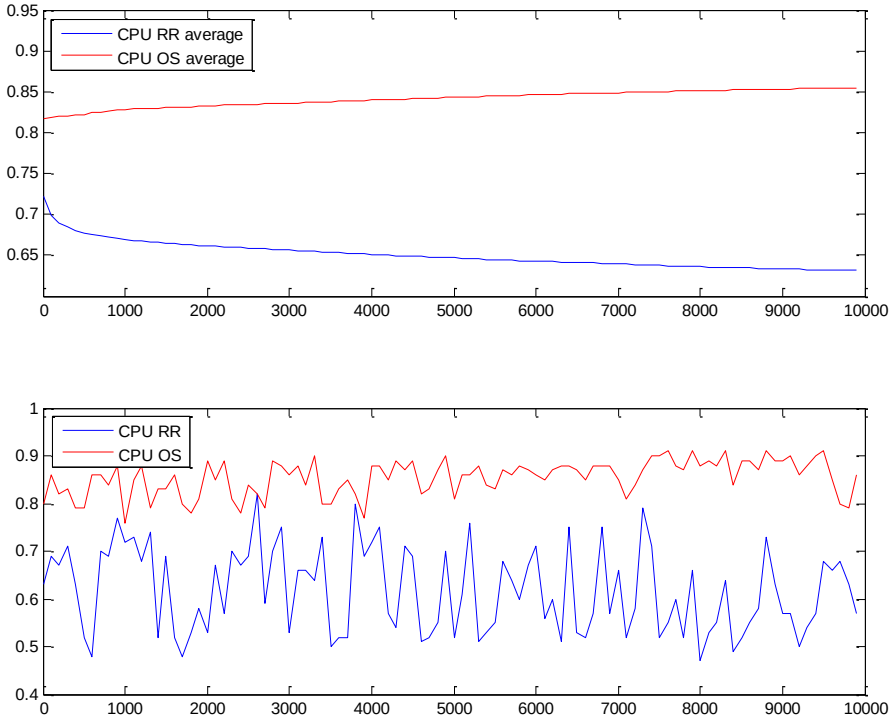
Στο σύνολο πειραμάτων παρατηρήθηκε αύξηση του πλήθος ολοκληρωμένων νημάτων αλλά και διεργασιών. Τα νήματα που ολοκληρώθηκαν με χρήση εκ περιτροπής εκτέλεσης ήταν κατά μέσον όρο 149,20, ενώ στα πειράματα με τον προτεινόμενο μηχανισμό τα νήματα που ολοκληρώθηκαν σχεδόν διπλασιάστηκαν φτάνοντας τα 291,30. Επίσης, βελτίωση παρατηρήθηκε και στον αριθμό διεργασιών που ολοκληρώθηκαν, αφού από 39,50 διεργασίες ολοκληρωμένες κατά μέσον όρο, η χρησιμοποίηση του μηχανισμού απόδοση προτεραιοτήτων στα νήματα οδήγησε σε 50,70 ολοκληρωμένες διεργασίες κατά μέσον όρο. Μοναδικό αντιστάθμισα ήταν η αύξηση των λόγων ποινής νημάτων και διεργασιών από 668,40 σε 904,70 όσον αφορά το λόγο ποινής εκτέλεσης νημάτων και από 401,91 σε 702,30.

6.2.3 Πειράματα με παράλληλη σύγκριση μετρικών του συστήματος

Σε αυτήν την ενότητα παρουσιάζεται η εξέλιξη της τιμής των μετρικών του συστήματος (χρησιμοποίηση του επεξεργαστή και λόγος σφαλμάτων σελίδων) σε πειράματα με πανομοιότυπο φόρτο εργασίας. Σκοπός των πειραμάτων αυτών είναι να φανεί το αποτέλεσμα του προτεινόμενου μηχανισμού σε αντίστοιχες καταστάσεις.

6.2.3.1 Πείραμα Γ1

Στο πείραμα Γ1 προσομοιώθηκε η συμπεριφορά του συστήματος με στατικό φόρτο εργασίας 100 διεργασιών και μέγεθος μνήμης 300 σελίδες. Στα παρακάτω σχήματα φαίνεται η χρονική εξέλιξη των τιμών των μετρικών που χαρακτηρίζουν τη συμπεριφορά του συστήματος.

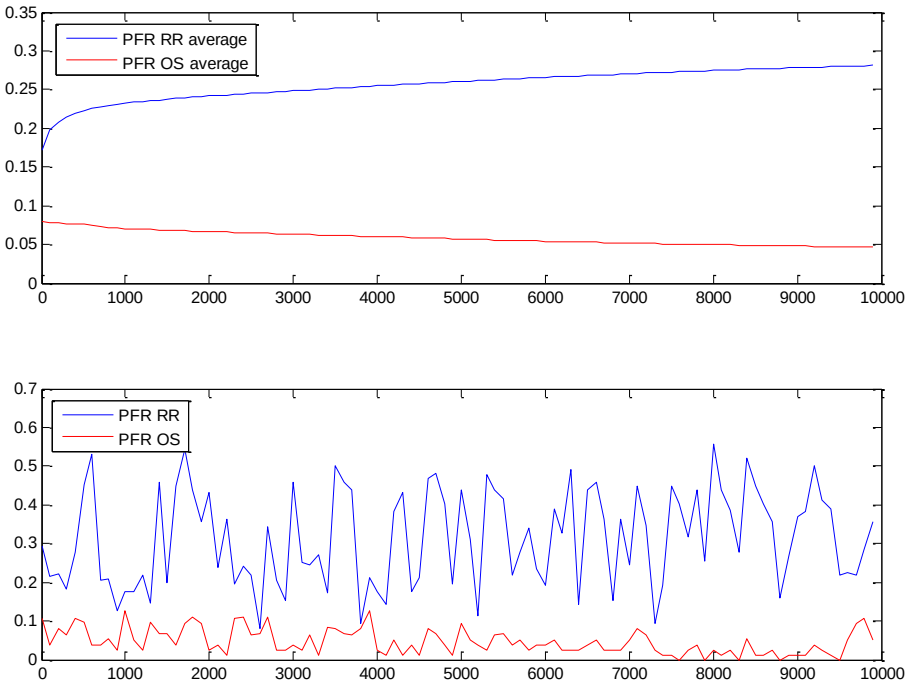


Σχήμα 4: Μέση και τρέχουσα χρησιμοποίηση του επεξεργαστή για το πείραμα Γ1

Ο άξονας των y είναι καθαρός αριθμός από 0 έως 1 και ο άξονας των x είναι δευτερόλεπτα και αντιστοιχούν στα 10.000 sec της προσομοίωσης. Με μπλε χρώμα είναι οι τιμές της εκ περιτροπής εκτέλεσης νημάτων και με κόκκινο χρώμα είναι οι τιμές που προκύπτουν από τη χρήση του προτεινόμενου μηχανισμού. Σε κάθε ζευγάρι σχημάτων το πρώτο σχήμα αναπαριστά τη χρονική εξέλιξη της μέσης τιμής της

αντίστοιχης μετρικής, ενώ το δεύτερο σχήμα αναπαριστά τη χρονική εξέλιξη της τρέχουσας τιμής της ίδιας μετρικής[1].

Στο παραπάνω σχήμα φαίνεται η χρονική μεταβολή της χρησιμοποίησης του επεξεργαστή. Στο πρώτο από τα δύο σχήματα φαίνεται η μέση χρησιμοποίηση ενώ στο δεύτερο η τρέχουσα (που προκύπτει από ένα παράθυρο τιμών, με διάρκεια ίση με δέκα κβάντα). Στο επόμενο σχήμα φαίνεται η χρονική μεταβολή του ρυθμού σφαλμάτων σελίδων για το ίδιο πείραμα (πρώτα η μέση και μετά η τρέχουσα).



Σχήμα 5: Μέσος και τρέχων λόγος σφαλμάτων σελίδων για το πείραμα Γ1

Στον παρακάτω πίνακα συνοψίζεται η σύγκριση των δύο μεθόδων για το πείραμα Γ1. Τα αποτελέσματα του πειράματος αυτού είναι παρόμοια με τα προηγούμενα πειράματα. Παρατηρούμε ότι το throughput σε νήματα αυξάνεται σημαντικά χάρη στη χρήση του προτεινόμενου μηχανισμού, μέσω της αύξησης της μέσης χρησιμοποίησης του επεξεργαστή και μείωσης του μέσου ρυθμού σφαλμάτων σελίδων. Αντίθετα υπάρχει μία ελάχιστη μείωση του throughput διεργασιών. Αντίστοιχα, ο λόγος ποινής εκτέλεσης

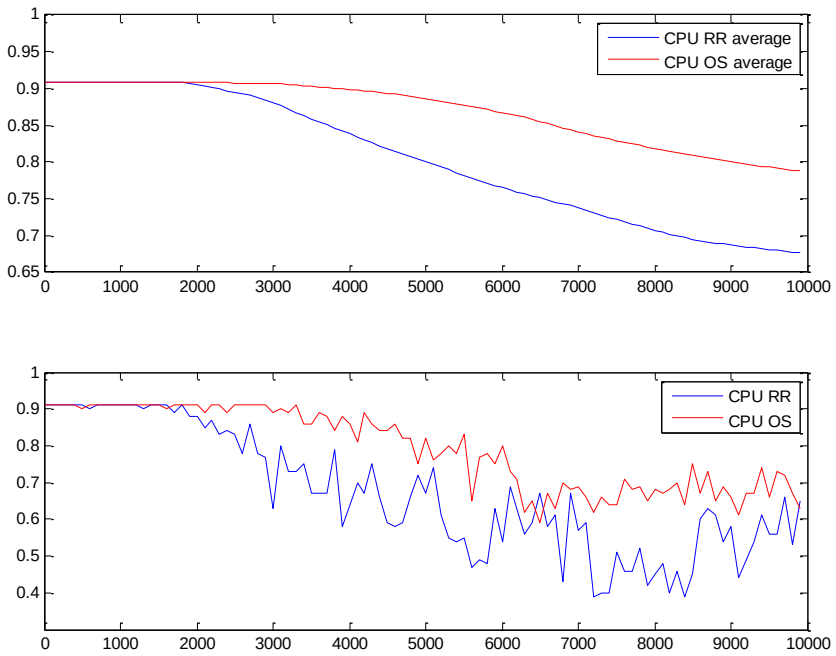
[1] Αποφεύχθηκε η χρήση της ορολογίας στιγμιαία τιμή αφού η «τρέχουσα» τιμή είναι ο μέσος όρος της αντίστοιχης μετρικής κατά τη διάρκεια χρόνου ίσου με τη διάρκεια 10 κβάντων του συστήματος.

νημάτων βελτιώνεται με χρήση του προτεινόμενου μηχανισμού ενώ ο λόγος ποινής εκτέλεσης διεργασιών επιβαρύνεται.

Πίνακας 7: Στατιστικά πειράματος Γ1

Παράμετρος	Round Robin	Proposed
Processes Created	100	100
Daemon Processes Created	5	5
Thread Created	420	420
Processed Finished	21	20
Thread Finished	43	67
Average CPU%	63,12%	85,48%
Average IO%	16,85%	3,64%
Average Context Switch%	20,03%	10,88%
Average PFR%	28,11%	4,65%
Thread Penalty Ratio	171,35	151,63
Process Penalty Ratio	94,52	125,74

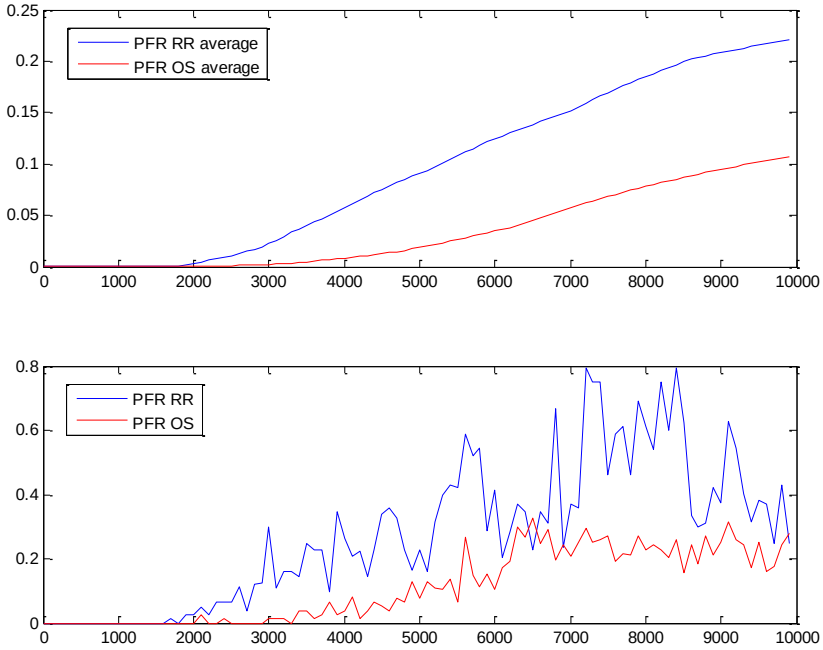
6.2.3.2 Πείραμα Γ2



Σχήμα 6: Μέση και τρέχουσα χρησιμοποίηση του επεξεργαστή για το πείραμα Γ2

Στο πείραμα Γ2 προσομοιώθηκε η συμπεριφορά του συστήματος με αρχικό φόρτο διεργασιών 20 και στη συνέχεια, κατά τη διάρκεια της προσομοίωσης δημιουργούνται διεργασίες μέχρι να παραχθούν 150 συνολικά. Το μέγεθος της μνήμης είναι και πάλι 300 σελίδες.

Στο πρώτο σχήμα φαίνεται η εξέλιξη της χρησιμοποίησης του επεξεργαστή. Αυτό το οποίο γίνεται αντιληπτό είναι ότι όταν αρχίσουν να υπάρχουν συνθήκες που επιβαρύνουν την κατάσταση του συστήματος η χρήση του προτεινόμενου μηχανισμού (κόκκινο χρώμα στα σχήματα) βοηθά στην διατήρηση των μετρικών σε καλύτερα επίπεδα από ότι η χρήση της εκ περιτροπής εκτέλεσης νημάτων (μπλε χρώμα στα σχήματα).



Σχήμα 7: Μέσος και τρέχων λόγος σφαλμάτων σελίδων για το πείραμα Γ2

Στον παρακάτω πίνακα συνοψίζεται η σύγκριση των δύο μεθόδων για το πείραμα Γ2.

Πίνακας 8: Στατιστικά πειράματος Γ2

Παράμετρος	Round Robin	Proposed
Processes Created	150	150
Daemon Processes Created	6	12
Thread Created	730	693

Processed Finished	19	9
Thread Finished	50	58
Average CPU%	67,64%	78,76%
Average IO%	14,00%	7,86%
Average Context Switch%	18, 63%	13,39%
Average PFR%	22,12%	10,73%
Thread Penalty Ratio	155,76	135,48
Process Penalty Ratio	123,46	89,88

Το πείραμα Γ2 μας δείχνει ότι η χρήση του προτεινόμενου μηχανισμού έχει ευεργετικές συνέπειες για το σύστημα ακόμα και όταν ο χρόνος κατά τον οποίο το σύστημα είναι επιβαρυμένο είναι περιορισμένος. Έτσι, όχι μόνο αυξήθηκε ο αριθμός νημάτων που ολοκληρώθηκαν και η χρησιμοποίηση του επεξεργαστή αλλά βελτιώθηκε τόσο ο λόγος ποινής εκτέλεσης διεργασιών όσο και ο λόγος ποινής εκτέλεσης νημάτων.

6.2.3.3 Πείραμα Γ3

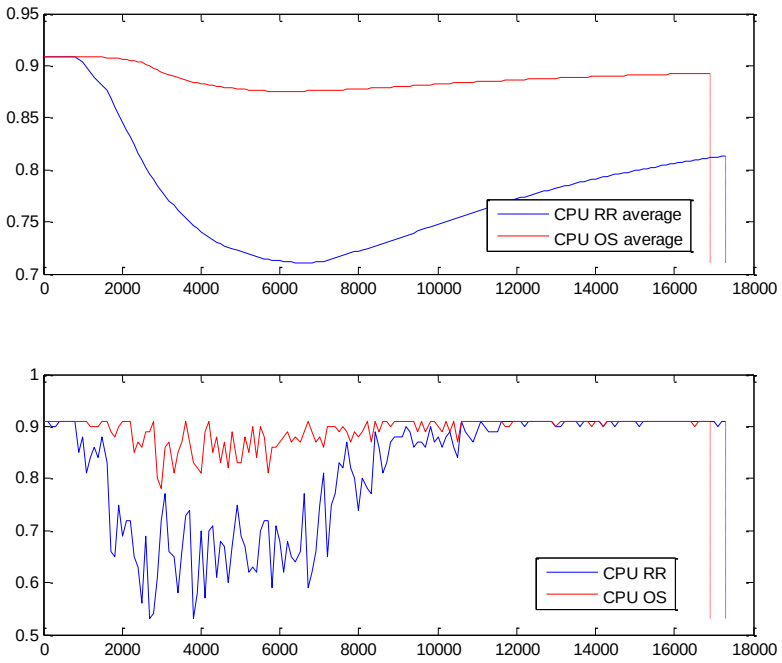
Στο πείραμα Γ3 προσομοιώθηκε η λειτουργία του επεξεργαστή έχοντας αρχικό φόρτο 20 διεργασίες, ο οποίος κατά τη διάρκεια της προσομοίωσης αυξήθηκε έως τις 60 διεργασίες. Η προσομοίωση έτρεξε για όσο χρόνο χρειάστηκε, μέχρι να ολοκληρωθεί η εκτέλεση όλων των διεργασιών που δημιουργήθηκαν (για το παράδειγμα αυτό δεν υπήρξαν διεργασίες δαίμονες προκειμένου να ολοκληρωθεί η προσομοίωση). Το μέγεθος της μνήμης ήταν 150 σελίδες και η μέση διάρκεια νήματος, σε αντίθεση με τα προηγούμενα πειράματα ήταν 50.000 ticks (δηλαδή 50 δευτερόλεπτα). Σκοπός του πειράματος αυτού είναι να φανεί η θετική επίδραση του αλγορίθμου ως προς το χρόνο της ολοκλήρωσης της εκτέλεσης του φόρτου εργασιών.

Πίνακας 9: Στατιστικά πειράματος Γ3

Παράμετρος	Round Robin	Proposed
Processes Created	60	60
Daemon Processes Created	0	0
Thread Created	286	314
Processed Finished	60	60
Thread Finished	286	314
Average CPU%	81,40%	89,30%
Average IO%	5,86%	1,08%

Average Context Switch%	12,74%	9,62%
Average PFR%	7,87%	1,34%
Thread Penalty Ratio	184,70	154,70
Process Penalty Ratio	35,79	35,68
Time of simulation end	17459,965 s	17092,383 s

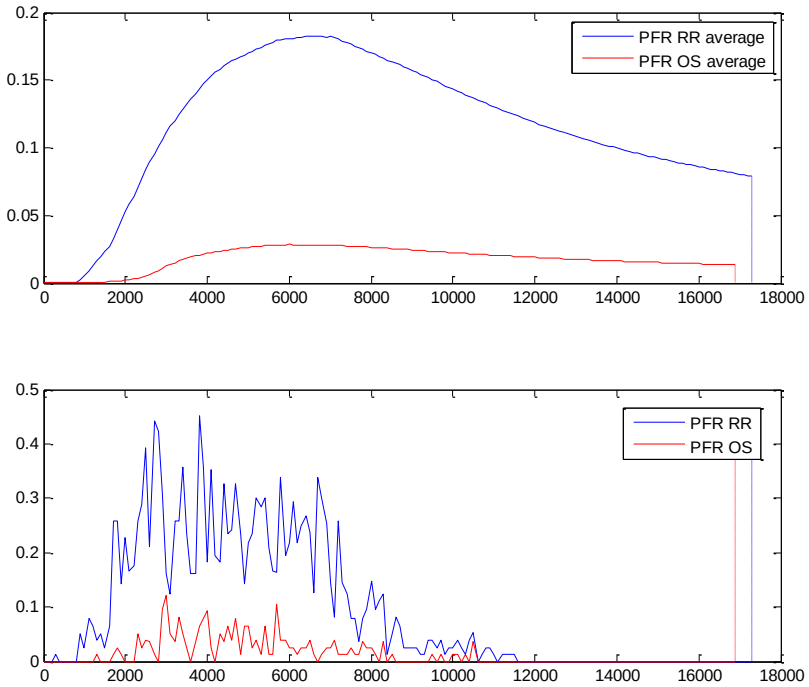
Στα σχήματα που ακολουθούν φαίνεται το αποτέλεσμα της χρήσης του προτεινόμενου αλγορίθμου. Ίδιο πλήθος διεργασιών που παράγεται δυναμικά κατά την εκτέλεση της προσομοίωσης ολοκληρώνονται σε μικρότερο χρόνο, επιτυγχάνοντας καθ' όλη τη διάρκεια της προσομοίωσης υψηλότερη χρησιμοποίηση του επεξεργαστή και χαμηλότερο λόγο σφαλμάτων σελίδων. Με μπλε χρώμα παρουσιάζονται τα στοιχεία που αφορούν την εκ περιτροπής εκτέλεση νημάτων και με κόκκινο χρώμα παρουσιάζονται τα στοιχεία που αφορούν την εκτέλεση νημάτων με βάση το προτεινόμενο αλγόριθμο. Οι κατακόρυφες διακεκομμένες γραμμές υποδηλώνουν ακολουθώντας την ίδια χρωματική σύμβαση το τέλος κάθε προσομοίωσης.



Σχήμα 8: Μέση και τρέχουσα χρησιμοποίηση του επεξεργαστή για το πείραμα Γ3

Στο πείραμα αυτό παρατηρήθηκε η αναμενόμενη, με βάση τα προηγούμενα πειράματα, βελτίωση των μετρικών του συστήματος. Ο μέση χρήση του επεξεργαστή αυξήθηκε από

81,40% σε 89,30% ενώ ο λόγος σφαλμάτων σελίδων μειώθηκε, δραματικά, από 7,87% σε 1,34%. Αξίζει να σημειωθεί ότι με την εκ περιτροπής εκτέλεση νημάτων όταν το σύστημα ξεκινούσε να μπαίνει σε επιβαρυμένη κατάσταση, οι συνθήκες χειροτέρευαν ραγδαία και μόνη λύση ήταν η ολοκλήρωση – συν τω χρόνω – των διεργασιών που προκαλούσαν το πρόβλημα. Αντίθετα, όπως βλέπουμε στο σχήμα με το λόγο σφαλμάτων σελίδων, όταν η τρέχουσα τιμή του λόγου σφαλμάτων σελίδων ξεπερνούσε το κατώφλι (10%) και ο μηχανισμός έμπαινε σε λειτουργία, ο λόγος αυτός μειωνόταν άμεσα και παρέμενε σε επίπεδα κάτω από το κατώφλι. Αποτέλεσμα της βελτίωσης των συνθηκών του συστήματος είναι και η ολοκλήρωση όλων των διεργασιών που παρήχθησαν κατά την προσομοίωση σε λιγότερο χρόνο. Πιο συγκεκριμένα, τα 17.460 περίπου δευτερόλεπτα μειώθηκαν κατά 2,11%, σε 17.092 δευτερόλεπτα.



Σχήμα 9: Μέσος και τρέχων λόγος σφαλμάτων σελίδων για το πείραμα Γ3

6.3 Συμπεράσματα

Το σύνολο των πειραμάτων που παρουσιάστηκε δείχνει ότι η προτεινόμενη μέθοδος οδηγεί σε πολύ θετικά αποτελέσματα. Ο χρονοπρογραμματισμός νημάτων με βάση την πρόσφατη – κυρίως – επίδραση του κάθε νήματος επί των συνθηκών που περιγράφουν το σύστημα οδηγεί σε βελτίωση της κατάστασης του συστήματος, είτε εξαλείφοντας το φαινόμενο της εξόντωσης είτε μειώνοντας την έντασή του, ανάλογα με τις αρχικές συνθήκες του συστήματος.

Σημαντικό, ακόμα, είναι ότι στα πειράματα που ολοκληρώθηκε το σύνολο του φόρτου εργασίας, παρατηρήθηκε ότι η ρυθμαπόδοση ήταν μεγαλύτερη, αφού ο χρόνος που χρειάστηκε για να ολοκληρωθούν οι ίδιες διεργασίες ήταν μικρότερος. Ο προτεινόμενος μηχανισμός πετυχαίνει:

- Την αύξηση της μέσης χρησιμοποίησης του επεξεργαστή
    - μέσω της μείωσης του λόγου σφαλμάτων σελίδων
- Τη μείωση του μέσου χρόνου απόκρισης όσον αφορά το σύνολο των αρχικών διεργασιών

Μία ακόμα σημαντική παρατήρηση είναι ότι στα πειράματα κατά το οποία το σύστημα ήταν πολύ επιβαρυμένο, η ρυθμαπόδοση όσον αφορά τις διεργασίες τόσο με χρήση της εκ περιτροπής εκτέλεσης, όσο και με χρήση του προτεινόμενου μηχανισμού είχε σχεδόν την ίδια τιμή, ενώ η ρυθμαπόδοση όσον αφορά τα νήματα είχε σε όλες τις περιπτώσεις σημαντική βελτίωση. Από τη στιγμή που ο προτεινόμενος μηχανισμός αναφέρεται στο χρονοπρογραμματισμό νημάτων είναι αναμενόμενο να υπάρχει πάντα βελτίωση στις μετρικές που αφορούν νήματα, ενώ για να επέλθει βελτίωση στις μετρικές που αφορούν διεργασίες πρέπει είτε να είναι μεγάλος ο φόρτος εργασίας και να αθροιστεί η βελτιωμένη συμπεριφορά του συστήματος είτε η βελτίωση, λόγω της υφής των ενεργών διεργασιών, να είναι πολύ έντονη.

ΚΕΦΑΛΑΙΟ 7

ΙΔΕΕΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

7.1 Θεωρητική μοντελοποίηση του προτεινόμενου μηχανισμού

Στην παρούσα εργασία μελετήθηκε το πρόβλημα της εξόντωσης και προτάθηκε ένας κατανεμημένος μηχανισμός αντιμετώπισης του φαινομένου. Μελετήθηκε πειραματικά και κρίνεται σκόπιμο να γίνει μια θεωρητική μοντελοποίηση της αντιμετώπισης του προβλήματος της εξόντωσης που προτείνεται. Μια πιθανή μοντελοποίηση είναι παιγνιοθεωρητική.

Σε αυτό το σημείο είναι σημαντικό να γίνει μια μικρή παρέκβαση ώστε να ορίσουμε τι είναι ένα παίγνιο:

Ορισμός[15]: Ένα παίγνιο είναι μια κατάσταση κατά την οποία

i. Ν άτομα (όπου Ν>1), τα οποία αποκαλούνται «παίκτες» κάνουν κάποιες επιλογές με σκοπό να ικανοποιήσουν το «ατομικό» τους συμφέρον, και,

ii. το αποτέλεσμα δεν εξαρτάται μόνο από τις επιλογές του κάθε παίκτη αλλά και από τις επιλογές των υπολοίπων Ν-1 παικτών.

Ο ανωτέρω ορισμός είναι ένας πολύ γενικός ορισμός, ο οποίος αποσκοπεί στο να μπορεί να περιγράψει όλα δυνατά παίγνια. Ακριβώς για το λόγο αυτό απέχει κατά τι από το να μπορεί να συσχετιστεί με τον προτεινόμενο μηχανισμό.

Ήδη, όμως, με χρήση αυτού του γενικού ορισμού μπορεί να γίνει μια πρώτη μοντελοποίηση ως εξής:

- Παίκτες είναι οι διεργασίες
- Το «ατομικό» συμφέρον των διεργασιών είναι ο χρόνος εκτέλεσης που θα έχουν, και ένας άλλος τρόπος για να το αυξήσουν είναι να μην υποστούν μείωση προτεραιότητας
- Ο μηχανισμός με χρήση του οποίου κάθε διεργασία προσπαθεί να μεγιστοποιήσει το «ατομικό» της όφελος είναι ο προτεινόμενος μηχανισμός απόδοσης προτεραιοτήτων στα νήματα, που αποσκοπεί στην αύξηση της μέσης χρησιμοποίησης του επεξεργαστή

7.1.1 Ταλαντώσεις στις αποφάσεις των παικτών (μηχανισμών χρονοπρογραμματισμού νημάτων)

Η θεώρηση που παρουσιάστηκε νωρίτερα, μπορεί να οδηγήσει σε παίκτες που θα κάνουν ταλαντώσεις όσον αφορά τις αποφάσεις τους. Αυτό μπορεί να οφείλεται στο ότι όλες οι διεργασίες όταν εντοπίζουν την επιβαρυμένη κατάσταση του συστήματος αλλάζουν τρόπο λειτουργίας, και οδηγούν το σύστημα σε γρήγορη βελτίωση των συνθηκών, οπότε και όλες οι διεργασίες εντοπίζουν ότι η κατάσταση του συστήματος βελτιώθηκε και, άρα, αποφασίζουν ότι μπορούν πάλι να αλλάξουν τρόπο λειτουργίας, θεωρώντας ότι το σύστημα βρίσκεται σε ομαλή κατάσταση.

Επειδή το πρόβλημα αφορά σε χρονοπρογραμματισμό διεργασιών και σε δεδομένα που ενημερώνονται συνεχώς, η υπόθεση ότι όλες οι διεργασίες θα λάβουν την ίδια απόφαση είναι λίγο βεβιασμένη. Ο κίνδυνος εμφάνισης των προαναφερθεισών ταλαντώσεων μειώνεται σημαντικά λόγω του ότι κάθε διεργασία διαγιγνώσκει εάν το σύστημα βρίσκεται σε επιβαρυμένη ή όχι κατάσταση με βάση το γεωμετρικό μέσο όρο των μετρικών του συστήματος (χρησιμοποίηση επεξεργαστή και λόγος σφαλμάτων σελίδων) μόνο αφού χρονοπρογραμματιστεί από τον κεντρικό χρονοπρογραμματιστή. Έτσι, ανά πάσα στιγμή – ανά κάθε κβάντο εκτέλεσης – μόνο μία απόφαση λαμβάνεται, από τη διεργασία που έχει το κβάντο εκτέλεσης στη διάθεσή της.

7.1.2 Μοντελοποίηση με βάση το παράδειγμα γεράκι-περιστέρι

Το παίγνιο γεράκι-περιστέρι [15] περιγράφει αρκετά πετυχημένα τη στρατηγική που υλοποιείται από τον προτεινόμενο μηχανισμό. Το παίγνιο αυτό διέπεται από τις εξής βασικές αρχές:

1. Θεωρεί ότι υπάρχουν δύο παίκτες που έχουν μια διαφωνία
2. Κάθε παίκτης μπορεί να συμπεριφερθεί επιθετικά (σαν γεράκι), ή
3. συναινετικά (σαν περιστέρι)
4. Στα τέσσερα δυνατά ζευγάρια συμπεριφορών έχουμε:
    i. Εάν συμπεριφερθούν και οι δύο επιθετικά, χάνουν και οι δύο

ii. Εάν συμπεριφερθεί ο ένας από τους δύο επιθετικά (εδώ έχουμε δύο περιπτώσεις, ανάλογα με το ποιος συμπεριφερθεί επιθετικά), τότε αυτός κερδίζει το «τρόπαιο» της διαφωνίας

iii. Εάν είναι και οι δύο συναινετικοί, τότε κερδίζουν και οι δύο, κάτι λιγότερο από αυτό που θα κέρδιζε ο καθένας εάν ήταν **μόνο** αυτός επιθετικός.

Για να φανεί καθαρά ποια είναι η αναλογία του ανωτέρω παίγνιου με τον προτεινόμενο μηχανισμό πρέπει να καθοριστεί τι σημαίνει για μια διεργασία να φερθεί επιθετικά ή συναινετικά.

Θεωρούμε ότι μια διεργασία «φέρεται επιθετικά» εάν ο εσωτερικός μηχανισμός χρονοπρογραμματισμού νημάτων επιλέγει νήματα που προκαλούν αύξηση των σφαλμάτων σελίδων ή και μείωση της χρησιμοποίησης του επεξεργαστή. Αντίθετα, όταν ο μηχανισμός αυτός επιλέγει νήματα που οδηγούν σε μείωση των σφαλμάτων σελίδων ή και αύξηση της χρησιμοποίησης του επεξεργαστή, τότε θεωρούμε ότι η διεργασία «φέρεται συναινετικά».

ΟΡΟΛΟΓΙΑ

context switch time	χρόνος εναλλαγής διεργασιών/νημάτων
CPU utilization	χρησιμοποίηση του επεξεργαστή
demand paging	αντικατάσταση σελίδων κατ’ απαίτηση
page thrashing	εξόντωση
level of multiprogramming	επίπεδο πολυπρογραμματισμού
load control	φόρτος εργασίας
logical address	λογική διεύθυνση (σελίδας στη μνήμη της διεργασίας)
main memory	κύρια μνήμη
non-preemptive	μη-προεκχωρητικός
page	σελίδα
physical address	φυσική διεύθυνση (σελίδας στο δίσκο)
preemptive	προεκχωρητικός
pre-paging	προεκχωρητική αντικατάσταση σελίδων
process	διεργασία
process penalty ratio	λόγος ποινής εκτέλεσης διεργασιών
ready-queue	ουρά έτοιμων διεργασιών/νημάτων
responsive	σύστημα που έχει λογικό χρόνο απόκρισης
scheduler	χρονοπρογραμματιστής
starvation	έλλειψη πόρων
thread	νήμα
thread penalty ratio	λόγος ποινής εκτέλεσης νημάτων
throughput	ρυθμαπόδοση
virtual memory	εικονική μνήμη

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ

FIFO	First In First Out
LFU	Least Frequently Used
LRU	Least Recently Used
MLF	Multi-Level Feedback
MMU	Memory Management Unit
PCB	Process Control Block
PFR	Page Fault Ratio
PMM	Process Memory Map
PS	Priority Scheduling
RR	Round Robin
SJF	Shortest Job First
SMP	Symmetric Multi-Processor
SRTCF	Shortest Remaining Time to Completion First

ΑΝΑΦΟΡΕΣ

1. Stallings William, *Operating Systems: Internal and Design Principles*, 5th edition, Prentice Hall, 2005.
2. Sudo Y., Suzuki S. and Shibayama S., *Distributed-Thread Scheduling Methods for Reducing Page-Thrashing*, HPDC 1997, p.p. 356-364.
3. Jiang S. and Zhang X., *Adaptive Page Replacement to Protect Thrashing in Linux*, 5th Annual Linux Showcase & Conference, November 2001.
4. Jiang S. and Zhang X., *TPF: a dynamic system thrashing protection facility*, In Software - Practice & Experience, March 2002, 32 (3): p.p. 295-318.
5. Finkel Raphael, *An Operating Systems Vade Mecum*, Prentice Hall, 1986.
6. Denning P., *Working Sets Past and Present*, IEEE Transactions on Software Engineering, January 1980.
7. Leroudier J. and Potier D., *Principles of Optimality for Multiprogramming*, Proceedings, International Symposium on Computer Performance Modeling, Measurement, and Evaluation, March 1976.
8. Carr R., *Virtual Memory Management*, Ann Arbor, MI: UMI Research Press, 1984.
9. Baer J., *Computer System Architecture*, Rockville, MD: Computer Science Press, 1980.
10. Belady L., *A Study of Replacement Algorithms for a Virtual Storage Computer*, IBM Systems Journal, No. 2 1966.
11. Duda K. and Cheriton D., *Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler*, SOSP 1999, p.p. 261-276.
12. Jain Rah, *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling*, Wiley Computer Publishing, 1991.
13. Fortier P. and Michel H., *Computer Systems Performance Evaluation and Prediction,* Digital Press, 2003.
14. Saucier Richard, *Computer Generation of Statistical Distributions*, Army Research Laboratory, 2000.
15. Hargreaves-Heap S. and Varoufakis Y., *Game Theory: A Critical Introduction*, Routledge, 1995.