ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΠΛΗΡΟΦΟΡΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΤΕΥΘΥΝΣΗ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση και Προσομοίωση του Χρονοδρομολογητή Completely Fair Scheduler

Τσακατάνης Χρήστος Γερμανός
Α.Μ.: ics20026

ΘΕΣΣΑΛΟΝΙΚΗ, ΙΑΝΟΥΑΡΙΟΣ 2023

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΣΟΥΡΑΒΛΑΣ ΣΤΑΥΡΟΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Ανάλυση και Προσομοίωση του Χρονοδρομολογητή Completely Fair Scheduler

Τσακατάνης Χρήστος Γερμανός

Προπτυχιακός Φοιτητής του Τμήματος Εφαρμοσμένης Πληροφορικής

Πανεπιστήμιο Μακεδονίας

Πτυχιακή Εργασία

Επιβλέπων Καθηγητής

Σουραβλάς Σταύρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή:

Σουραβλάς Σταύρος	Σιφαλέρας Άγγελος	Κολωνίαρη Γεωργία
………………………	………………………	………………………

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον υπεύθυνο για την επίβλεψη της παρούσας πτυχιακής εργασίας, Επίκουρο καθηγητή κύριο Σταύρο Σουραβλά τόσο για την ευκαιρία που μου έδωσε για να την πραγματοποιήσω όσο και για τις γνώσεις που μου μετέδωσε από το πρώτο κιόλας έτος στη σχολή, καθώς επίσης και για την συνεχή και αδιάκοπη επικοινωνία που είχαμε και τις συμβουλές του.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη τους όλα αυτά τα χρόνια καθώς και τους φίλους μου και συγκεκριμένα τον Γιάννη, τον Λέοναρντ, την Αραβέλλα και τον Κωνσταντίνο, διότι χωρίς τη δική τους ηθική και ψυχολογική υποστήριξη δε θα τα είχα καταφέρει.

Στα αδέρφια μου, Νίκο και Γιάννη

Περίληψη

Το Λειτουργικό Σύστημα του Linux εκδόθηκε για πρώτη φορά τον Σεπτέμβριο του 1991. Δημιουργός του ο Linus Torvalds ο οποίος επηρεάστηκε από τα Λειτουργικά Συστήματα Unix και Minix.

Ένας αλγόριθμος χρονοδρομολόγησης περιγράφει τον τρόπο με τον οποίο θα επιλέξει ο χρονοδρομολογητής μία διεργασία για να την παραχωρήσει στην Κεντρική Μονάδα Επεξεργασίας (CPU), για πόσο θα την απασχολήσει, ποιες είναι οι λειτουργίες της διεργασίας και πόση κατανάλωση θα υπάρχει στους πόρους του συστήματος. Ο αλγόριθμος επίσης θα κρίνει πότε η διεργασία θα απομακρυνθεί από την ΚΜΕ.

Ο σκοπός και ο στόχος της συγκεκριμένης έρευνας είναι η μελέτη και η ανάλυση του Χρονοδρομολογητή Completely Fair Scheduler (CFS), ο τρόπος λειτουργίας του με τη χρήση των Ερυθρόμαυρων Δέντρων (Red Black Trees) και η απόδοση δικαιοσύνης ως προς τις διεργασίες του συστήματος. Στη συνέχεια η συγγραφή του κώδικα του Χρονοδρομολογητή CFS, ώστε να γίνει εφικτή μία πλήρης προσομοίωση του αλγορίθμου με την παρουσίαση των διεργασιών σε ένα Red Black Tree, καθώς και των δεδομένων που χρειάζονται ώστε να είναι εφικτό αυτό.

Μέσω αυτής της έρευνας, είναι εφικτό να σχολιαστεί η δικαιοσύνη του αλγορίθμου, η κατανάλωση των πόρων που χρησιμοποιεί το σύστημα, καθώς επίσης και το πως ο CFS έχει καλύτερη πολυπλοκότητα από τους προγόνους του.

Λέξεις Κλειδιά: Λειτουργικό Σύστημα, Χρονοδρομολόγηση, Χρονοδρομολογητής O(1), O(n), Completely Fair Scheduler (CFS), Ερυθρόμαυρα Δέντρα (Red Black Trees), Πολυπλοκότητα.

Abstract

The Linux Operating System was first released in September of 1991. Its creator Linus Torvalds was inspired by the Unix and Minix Operating Systems.

A Scheduling Algorithm describes the way in which the Scheduler will move a process to the CPU and make use of it, for how much time it is going to use it, which are the functionalities of the process and how many quantums it will use. The Algorithm also decides when the process will be moved away from the CPU.

The purpose and the goal of this research is to study and analyze the Completely Fair Scheduler (CFS), how it operates with the use of Red Black Trees and how justice is rendered towards the rest of the processes. Additionally the writing of the code of the CFS Scheduler is of great importance in order to have a complete simulation of the algorithm and how it operates by presenting the processes in a Red Black Tree, as well as the data needed to make this possible.

Through this research, it is possible to comment on the fairness of the algorithm, the resources that are consumed by the system, as well as how the CFS Scheduler has a better complexity than its predecessors.

Keywords: Operating System, Scheduling, O(1), O(n) Scheduler, Completely Fair Scheduler (CFS), Red Black Trees, Complexity.

Περιεχόμενα

Ευχαριστίες ... 2
Περίληψη ... 4
Abstract ... 5
Περιεχόμενα ... 6
Κατάλογος Εικόνων ... 7
Κεφάλαιο 1ο : Εισαγωγή ... 10
  1.1. Έννοιες και ορισμοί ... 10
  1.2. Στόχος της Έρευνας ... 11
  1.3. Δομή της Διπλωματικής ... 11
Κεφάλαιο 2ο : Ο Χρονοδρομολογητής Completely Fair Scheduler ... 12
  2.1. Ο Χρονοδρομολογητής Ο(n) ... 12
  2.2. Ο Χρονοδρομολογητής Ο(1) ... 12
    2.2.1. Βήματα εκτέλεσης του Ο(1) ... 13
    2.2.2. Σφάλματα του Ο(1) ... 13
  2.3. Ο Χρονοδρομολογητής CFS ... 13
    2.3.1. Δυαδικά Δέντρα Αναζήτησης: Εισαγωγή στα Red Black Trees ... 14
    2.3.2. Βασικές Έννοιες ... 15
    2.3.3 Βήματα εκτέλεσης του CFS με τη χρήση Red Black Trees ... 17
    2.3.4 Κανόνες Περιστροφής των Red Black Trees ... 19
Κεφάλαιο 3ο : Περιγραφή και Ανάλυση του Κώδικα ... 19
  3.1. Η κλάση Configuration.java ... 21
  3.2. Η κλάση Main.java ... 22
  3.3. Η κλάση Node.java ... 22
  3.4. Η κλάση RedBlackTree.java ... 24
  3.5. Η κλάση Simulation.java ... 28
Κεφάλαιο 4ο : Αποτελέσματα τις Προσομοίωσης ... 35
Κεφάλαιο 5ο : Τελικά Συμπεράσματα της Έρευνας ... 51
Βιβλιογραφία ... 52

Κατάλογος Εικόνων

Εικόνα 1.1: Το λογότυπο του Linux 10

Εικόνα 2.1: Χρονοδρομολόγηση διεργασιών σε ένα Λειτουργικό Σύστημα. 12

Εικόνα 2.2: Απεικόνιση της λειτουργίας του CFS, με τη χρήση των Red Black Trees . . . 14

Εικόνα 2.3: Red Black Tree 15

Σχήμα 1 17

Σχήμα 2α 17

Σχήμα 2β 17

Σχήμα 3α 17

Σχήμα 3β 17

Σχήμα 3γ 17

Σχήμα 4α 18

Σχήμα 4β 18

Σχήμα 4γ 18

Σχήμα 5α 18

Σχήμα 5β 18

Σχήμα 6α 18

Σχήμα 6β 18

Εικόνα 2.4: Οι συναρτήσεις πολυπλοκότητας 19

Εικόνα 2.5: Δεξιά Περιστροφή. 20

Εικόνα 2.6: Αριστερή Περιστροφή. 20

Εικόνα 3.1: Η συνάρτηση Configuration 21

Εικόνα 3.2: Η συνάρτηση Main 22

Εικόνα 3.3: Η συνάρτηση Node 22

Εικόνα 3.4: Οι συναρτήσεις giveRandomNice, setNiceFromUser, calculateVruntime, calculateWeight, calculateQuantum 23

Εικόνα 3.5: Η συνάρτηση giveRunTime 23

Εικόνα 3.6: Η συνάρτηση RedBlackTree 24

Εικόνα 3.7: Η συνάρτηση minimum 24

Εικόνα 3.8: Η συνάρτηση insert 25

Εικόνα 3.9: Η συνάρτηση fixInsert 26

Εικόνα 3.10: Οι συναρτήσεις rightRotate και leftRotate. 27

Εικόνα 3.11: Η συνάρτηση Simulation. 28

Εικόνα 3.12: Η συνάρτηση initialize. 28

Εικόνα 3.13: Η συνάρτηση simulate. 29

Εικόνα 3.14: Οι συναρτήσεις createProcesses και createProcess 30

Εικόνα 3.15: Η συνάρτηση moreProcesses 31

Εικόνα 3.16: Η συνάρτηση runNewProcessesFirst 31

Εικόνα 3.17: Οι συναρτήσεις checkOrDeleteRuntime και fixTree. 32

Εικόνα 3.18: Οι συναρτήσεις inorderHelper και runInorder. 33

Εικόνα 3.19: Η συνάρτηση addProcesses. 33

Εικόνα 3.20: Η συνάρτηση printTLUsage 34

Εικόνα 3.21: Οι συναρτήσεις calculateTotalQuantum, printTotalQuantum, prinProcesses, printList. 34

Εικόνα 4.1: Εκκίνηση προσομοίωσης. 35

Εικόνες 4.2α, 4.2β, 4.2γ: Η κατασκευή του αρχικού Red Black Tree. .36

Εικόνα 4.3: Ο πίνακας των διεργασιών και των χαρακτηριστικών του (πάνω), ο πίνακας χρήσης TL (κάτω) 37

Εικόνα 4.4: Οι χρόνοι των πρώτων δέκα διεργασιών, πριν και αφού τρέξουν. 37

Εικόνα 4.5: Προσθήκη επιπλέον διεργασιών και τιμής nice. 37

Εικόνα 4.6: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν. 38

Εικόνα 4.7α: Εισαγωγή των διεργασιών P10 έως P13 στο Red Black Tree. 39

Εικόνα 4.7β: Εισαγωγή των διεργασιών P14 έως P16 στο Red Black Tree. 40

Εικόνα 4.7γ: Εισαγωγή των διεργασιών P17 και P18 στο Red Black Tree. 41

Εικόνα 4.7δ: Εισαγωγή της διεργασίας P19 στο Red Black Tree, Πίνακας χρήσης TL. . .42

Εικόνα 4.8: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν. 43

Εικόνα 4.9: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν. 44

Εικόνα 4.10: Η διαγραφή των διεργασιών P4 και P0. 45

Εικόνα 4.11: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν. 46

Εικόνες 4.12α, 4.12β: Διαγραφή των διεργασιών P5, P12, P9, P3, P7, P6. 47

Εικόνα 4.12γ: Διαγραφή των διεργασιών P8, P2 και P1. 48

Εικόνα 4.13: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν. 49

Εικόνα 4.14: Διαγραφή των διεργασιών P15, P16, P19 και P14. 49

Εικόνα 4.15: Διαγραφή των διεργασιών P13, P11, P17, P10 και P18 και τερματισμός της προσομοίωσης. 50

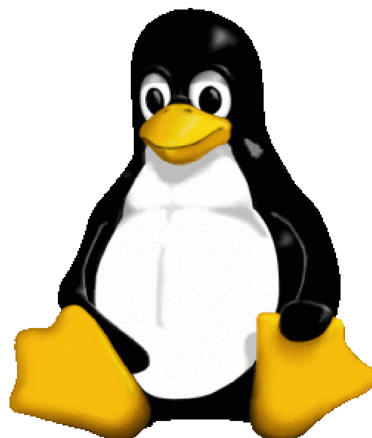
Κεφάλαιο 1ο : Εισαγωγή

Η έννοια της Χρονοδρομολόγησης-Scheduling, προέρχεται από το αγγλικό Schedule που σημαίνει πρόγραμμα. Για τη λειτουργία και την οργάνωση μίας εργασίας, μίας ομάδας ή στη προκειμένη περίπτωση ενός συστήματος, η δημιουργία ενός τέτοιου προγράμματος/πλάνου είναι απαραίτητη για να εφαρμοστούν σωστά όλες οι διαδικασίες. Ειδικά στον κλάδο της πληροφορικής, αν δεν είχαν σχεδιαστεί αλγόριθμοι και Χρονοδρομολογητές, τότε δεν θα υπήρχε σωστή κατανομή πόρων και ισχύος σε ένα υπολογιστικό σύστημα, αλλά αντιθέτως θα επικρατούσε χάος.

1.1. Έννοιες και ορισμοί

Ως Λειτουργικό Σύστημα (Λ.Σ), (Operating System – OS), ονομάζουμε το σύνολο των προγραμμάτων που ελέγχουν την εκτέλεση των εφαρμογών και δρα ως μέσω επικοινωνίας ανάμεσα στις εφαρμογές του χρήστη και το υλικό του υπολογιστή. Ο σκοπός ενός Λ.Σ. είναι να μοιράζει και να χρησιμοποιεί σωστά τους πόρους ενός Υπολογιστικού Συστήματος, να βοηθάει τον χρήστη να χειρίζεται πιο ευκολά έναν υπολογιστή, καθώς επίσης και να έχει τη δυνατότητα να αναπτυχθεί και να εξελιχθεί, δηλαδή να προστίθενται νέες λειτουργίες σε αυτό, χωρίς να επηρεάζονται οι υπηρεσίες που προσφέρει[1].Τα Λ.Σ. διευκολύνουν τους προγραμματιστές στην ανάπτυξη και εκτέλεση προγραμμάτων, παρέχοντας τους εργαλεία, όπως επεξεργαστές κειμένων και διορθωτές λαθών.

Με τον όρο Χρονοδρομολόγηση (Scheduling) και κυρίως Χρονοδρομολόγηση Κεντρικής Μονάδας Επεξεργασίας (ΚΜΕ), (CPU Scheduling), εννοούμε τις «αποφάσεις» που λαμβάνει το Λειτουργικό Σύστημα για την ανάθεση της ΚΜΕ στις διεργασίες. Με αυτό τον τρόπο στοχεύει στην ικανοποίηση κάποιον στόχων του συστήματος, όπως ο χρόνος απόκρισης, η ρυθμοαπόδοση και η αποτελεσματικότητα του επεξεργαστή. Υπάρχουν συνολικά τέσσερα είδη χρονοδρομολόγησης: η Μακροπρόθεσμη, η Μεσοπρόθεσμη, η Βραχυπρόθεσμη και η Χρονοδρομολόγηση Εισόδου/Εξόδου (Ε/Ε).



Εικόνα 1.1: Το λογότυπο του Linux

1.2. Στόχος της Έρευνας

Ο σκοπός της συγκεκριμένης έρευνας είναι η μελέτη και η ανάλυση του Χρονοδρομολογητή Completely Fair Scheduler (CFS), ο τρόπος λειτουργίας του με τη χρήση των Ερυθρόμαυρων Δέντρων (Red Black Trees) και η απόδοση δικαιοσύνης ως προς τις διεργασίες του συστήματος.

Ο στόχος της συγκεκριμένης έρευνας είναι η συγγραφή του κώδικα του Χρονοδρομολογητή CFS, ώστε να γίνει εφικτή μία πλήρης προσομοίωση του αλγορίθμου με την παρουσίαση των διεργασιών σε ένα Red Black Tree, καθώς και των δεδομένων που χρειάζονται ώστε να είναι εφικτό αυτό.

1.3. Δομή της Διπλωματικής

Η δομή της διπλωματικής εργασίας έχεις ως εξής:

- Στο **Κεφάλαιο 2** γίνεται η πλήρης ανάλυση του Χρονοδρομολογητή CFS, ποιος ήταν ο σκοπός της δημιουργίας του, τι είναι ο Χρονοδρομολογητής O(n), τι είναι ο Χρονοδρομολογητής O(1), τι είναι τα Red Black Trees και πως υλοποιούν των CFS.
- Στο **Κεφάλαιο 3** παρουσιάζεται η δομή του κώδικα που έχει συγγραφεί και η παρουσίαση των συναρτήσεων του μαζί με τις λειτουργίες τους.
- Στο **Κεφάλαιο 4** παρουσιάζονται τα αποτελέσματα της προσομοίωσης .
- Στο **Κεφάλαιο 5** σχολιάζονται τα αποτελέσματα και παρουσιάζονται τα συμπεράσματα της έρευνας**.**

Κεφάλαιο 2ο : Ο Χρονοδρομολογητής Completely Fair Scheduler

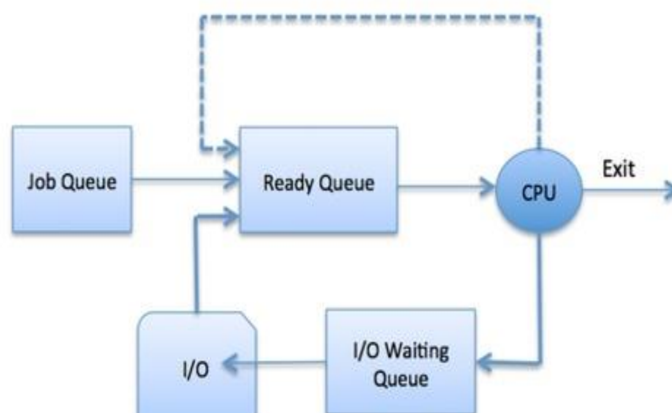
Ιστορικά έχουν υπάρξει μέχρι στιγμής τρία είδη Χρονοδρομολογητών, ο πρώτος O(n), στη συνέχεια ο O(1) και τέλος ο CFS. Με την ανάλυση του καθενός μπορούμε να αντιληφθούμε καλύτερα γιατί ήταν απαραίτητη η αντικατάσταση-εξέλιξη τους στο Λειτουργικό Σύστημα του Linux.

2.1. O Χρονοδρομολογητής O(n)

Ο O(n) ήταν ένας άμεσος και απλός στη λειτουργία Χρονοδρομολογητής. Βασιζόταν σε μία λίστα με αποθηκευμένες διεργασίες, οι οποίες περίμεναν με σειρά να πάρουν κβάντα για να εκτελεστούν, οπότε ο χρόνος επιλογής της επόμενης εξαρτώνταν από το n, δηλαδή από τον συνολικό αριθμό των διεργασιών. Μέχρι να εμφανιστεί η Java, δεν είχε γίνει αντιληπτό αυτό το πρόβλημα καθυστέρησης και ο O(n) συνέχισε να χρησιμοποιείται μεταξύ της έκδοσης 2.4 και πριν την εμφάνιση της έκδοσης 2.6 του Linux Kernel.

2.2. O Χρονοδρομολογητής O(1)

Με την νέα έκδοση του Linux Kernel 2.6, αποφασίστηκε να χρησιμοποιηθεί ένας διαφορετικός και εξ ολοκλήρου νέος Χρονοδρομολογητής με το όνομα O(1). Κατασκευάστηκε με τέτοιο τρόπο ώστε να έχει τη δυνατότητα να επιλέξει την πιο κατάλληλη διεργασία και να την αναθέσει σε έναν επεξεργαστή, ανεξαρτήτως από το πλήθος των διεργασιών που μπορεί να βρίσκονται εκείνη τη χρονική στιγμή στο σύστημα ή από τον φόρτο του ίδιου του συστήματος, εξού και το όνομα O(1). Ο αλγόριθμος πάνω στον οποίο βασίζεται, χρησιμοποιεί δύο Array Lists, την Active, όπου περιέχει 40 ουρές οι οποίες έχουν αποθηκευμένες τις διεργασίες που τρέχουν, και την Expired, όπου περιέχει 40 ουρές οι οποίες έχουν αποθηκευμένες τις διεργασίες των οποίων τα κβάντα έχουν τελειώσει[3][4].



Εικόνα 2.1: Χρονοδρομολόγηση διεργασιών σε ένα Λειτουργικό Σύστημα

2.2.1. Βήματα εκτέλεσης του Ο(1)

Ο Χρονοδρομολογητής ξεκινάει ταξινομώντας τις ουρές από τη μικρότερη στη μεγαλύτερη και με τη χρήση του αλγορίθμου FIFO (First-In-First-Out), τρέχει τις διεργασίες που βρέθηκαν στην πρώτη. Αφού μία διαδικασία τρέξει τα κβάντα της, θα μεταφερθεί σε μία ουρά Expired (και κατά πάσα πιθανότητα σε άλλη σειρά προτεραιότητας). Η διαδικασία αυτή συνεχίζεται μέχρι να τρέξουν όλες οι διεργασίες, δηλαδή, όλες οι διεργασίες από όλες τις ουρές θα έχουν μεταφερθεί από τη λίστα Active, στη λίστα Expired. Αφού ολοκληρωθεί αυτό το βήμα, θα υλοποιηθεί ένα Context Switch, που σημαίνει πως η λίστα Expired θα μετατραπεί στην λίστα Active και το αντίθετο, ώστε οι διεργασίες να έχουν τη δυνατότητα να ξανατρέξουν. Οι προτεραιότητες πλέον αλλάζουν δυναμικά.

2.2.2. Σφάλματα του Ο(1)

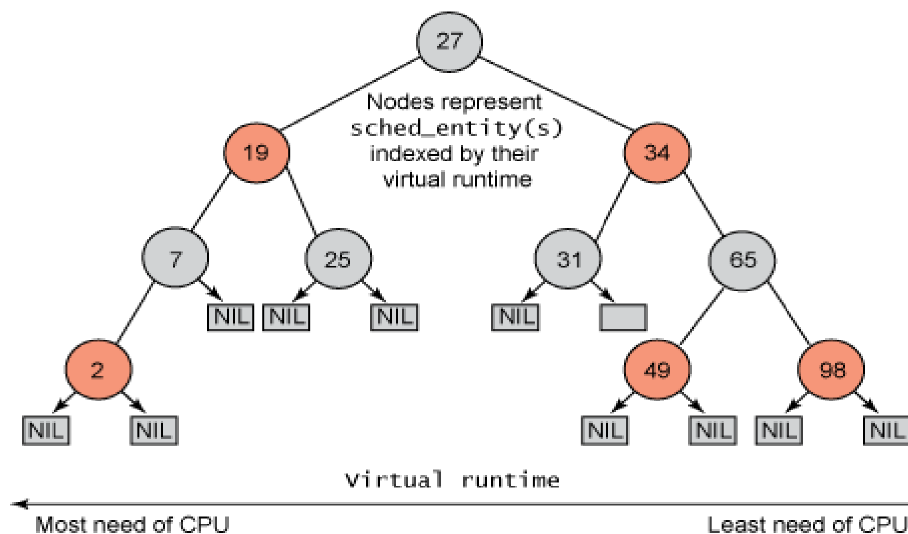
Ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζει ο Χρονοδρομολογητής Ο(1), είναι η αδυναμία ίσης κατανομής κβάντων στις διεργασίες. Αυτό συμβαίνει διότι ο αλγόριθμος είχε περίπλοκες μεθόδους για να ξεχωρίσει τις Interactive (διεργασίες που περιμένουν κάποια είσοδο από τον χρήστη) και non-Interactive (διεργασίες που δεν περιμένουν κάποια είσοδο από τον χρήστη) διεργασίες. Αποτέλεσμα ήταν να δίνεται bonus προτεραιότητα σε «λάθος» διεργασίες και οι Interactive να έχουν συμπεριφορά non-Interactive και αντιστρόφως. Επιπλέον οι νέες διεργασίες που εισέρχονται στο σύστημα θα πρέπει να περιμένουν για να εκτελεστούν[3]. Ο Χρονοδρομολογητής Ο(1) τελικά αποδείχτηκε δυσκίνητος και σύνθετος.

2.3. Ο Χρονοδρομολογητής CFS

Τον Οκτώβριο του 2007 δημοσιεύτηκε η έκδοση 2.6.23 του Linux Kernel και μαζί με αυτή άρχισε να χρησιμοποιείται ένας καινούργιος Χρονοδρομολογητής, ο Completely Fair Scheduler. Ο Ingo Molnar, προγραμματιστής γνωστός για τη συνεισφορά του στο Λειτουργικό Σύστημα του Linux και κατασκευαστής των Χρονοδρομολογητών Ο(1) και CFS, δήλωσε πως «Ο CFS προσημειώνει έναν ιδανικό και ακριβές πολυδιεργασιακό επεξεργαστή σε πραγματικό υλικό»[9][10][11]. Σκοπός του CFS είναι να έχει τη δυνατότητα να δίνει δίκαιο ποσοστό της ΚΜΕ σε όλες τις διεργασίες που βρίσκονται στο σύστημα, με αποτέλεσμα αυτές να τρέχουν σχεδόν ταυτόχρονα κι έτσι να αποδίδεται δικαιοσύνη και να υπάρχει ίση κατανομή κβάντων[2][4][5]. Αν είχαμε για παράδειγμα μία διεργασία, τότε αυτή θα αξιοποιούσε το 100% της ΚΜΕ, αν είχαμε δύο διεργασίες, θα διαμοιραζόταν η ΚΜΕ και κάθε διεργασία θα αξιοποιούσε από 50% της ισχύος και ούτω καθεξής. Αυτό προφανώς είναι ακόμη αδύνατο, παρόλα αυτά ο CFS μας δείχνει τον τρόπο με τον οποίο θα μπορούσε να γίνει εφικτό.

Οι διεργασίες στον CFS κατανέμονται με βάση μία ιδεατή τιμή χρόνου εκτέλεσης, το Virtual Runtime ή πιο απλά vruntime, το οποίο αποτελεί τον χρόνο που έχει καταναλωθεί μέχρι στιγμής για την εκτέλεση, (αφού πρώτα έχει κανονικοποιηθεί ως προς το σύνολο των διεργασιών). Όσο μικρότερο είναι το vruntime, τόσο μεγαλύτερη είναι η ανάγκη της διεργασίας να χρησιμοποιήσει τον επεξεργαστή.

Ο αλγόριθμος αναζητά κάθε φορά την διεργασία με το μικρότερο vruntime. Αυτή θα αρχίσει να εκτελείται για έναν χρόνο q ,(υπολογίζεται από τον χρονοδρομολογητή), και αφού ολοκληρωθούν τα κβάντα της, θα ξαναγίνει έλεγχος για την εύρεση του μικρότερου vruntime. Αν αυτό ανήκει στην ίδια διεργασία, τότε θα συνεχίσει την εκτέλεση της. Ακόμη κι αν συμβεί αυτό πολλαπλές φορές, ο CFS μας δίνει τη βεβαίωση και την ασφάλεια, πως όλες οι διεργασίες θα περάσουν από τη θέση να έχουν μικρότερο vruntime, με ισορροπημένο σχετικά τρόπο. Οπότε θα αποδίδεται δικαιοσύνη για τις διεργασίες. Για την υλοποίηση του Χρονοδρομολογητή γίνεται αξιοποίηση των Red Black Trees. [6][7][8]



Εικόνα 2.2: Απεικόνιση της λειτουργίας του CFS, με τη χρήση των Red Black Trees

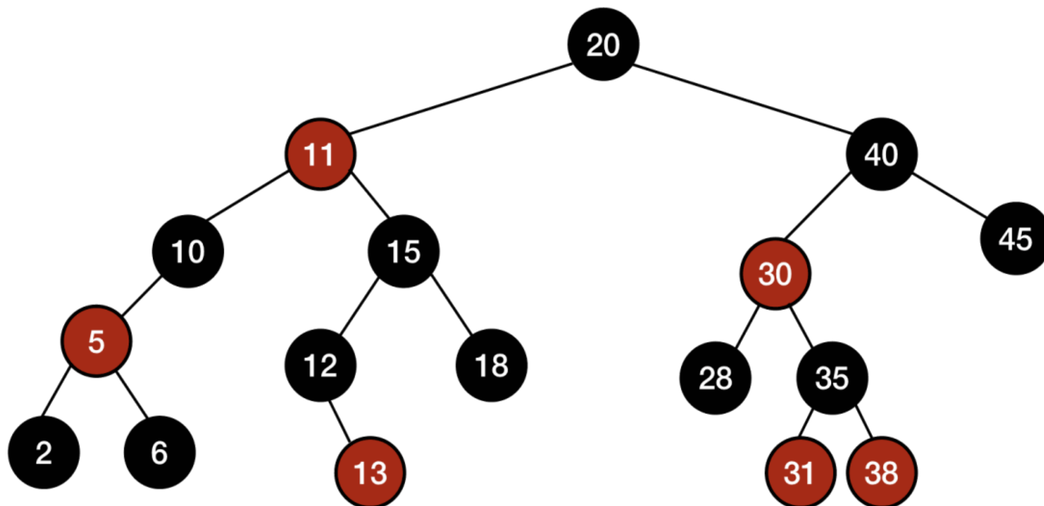
2.3.1. Δυαδικά Δέντρα Αναζήτησης: Εισαγωγή στα Red Black Trees

Ως Δυαδικό Δέντρο ορίζουμε ένα σύνολο κόμβων το οποίο μπορεί να είναι κενό ή να υπάρχει ένας συγκεκριμένος κόμβος, ο οποίος θα αποτελεί τη ρίζα του δέντρου και δεν έχει κάποιον προηγούμενο, ενώ οι υπόλοιποι κόμβοι θα χωριστούν σε δύο ξεχωριστά υποσύνολα και θα αποτελούν δύο νέα δυαδικά δέντρα, τα οποία θα χωριστούν σε δεξί και αριστερό υπόδεντρο της ρίζας αντιστοίχως.

Ένα Red Black Tree είναι ένα αυτό-ισορροπούμενο δυαδικό δέντρο αναζήτησης. Κάθε ένας από τους κόμβους του είναι είτε μαύρος είτε κόκκινος. Αυτή η ρύθμιση βοηθάει το δέντρο να κρατήσει μια συγκεκριμένη ισορροπία κατά τη διάρκεια της εισχώρησης ή της διαγραφής των κόμβων. Αφού γίνουν αυτές οι διαδικασίες, λαμβάνουν χώρα μία σειρά περίπλοκων αλγορίθμων για να ελέγξουν κατά πόσο ισχύουν οι κανόνες ισορροπίας του δέντρου, με σκοπό να αλλάξουν, (αν χρειαστεί), τα χρώματα των κόμβων, καθώς και τις θέσεις τους. Η πολυπλοκότητα του αλγορίθμου των Red Black Trees είναι O(logN), όπου N ο αριθμός των κόμβων στο δέντρο. Αυτό συμβαίνει διότι στα περισσότερα Δυαδικά Δέντρα αναζήτησης, λειτουργίες όπως αναζήτηση,

εισχώρηση, διαγραφή…κ.λ.π. χρειάζονται O(h) χρόνο, όπου h το ύψος του Δυαδικού Δέντρου Αναζήτησης. Αντιθέτως, το ύψος ενός Red Black Tree παραμένει πάντα $O(\log_2 N)$, (άνω όριο), μετά από οποιαδήποτε λειτουργία, με N των αριθμό των κόμβων που βρίσκονται μέσα στο δέντρο.[12][13]

- Κανόνες ισορροπίας των Red Black Trees
    - Κάθε κόμβος είναι, είτε κόκκινος, είτε μαύρος
    - Η ρίζα είναι μαύρη
    - Όλα τα NIL φύλλα, (κόμβοι που δεν περιέχουν κάποια τιμή), είναι μαύρα
    - Ένας κόκκινος κόμβος δεν μπορεί να έχει κόκκινα παιδιά ή κόκκινο γονέα
    - Όλα τα μονοπάτια από έναν κόμβο προς τα φύλλα, περιέχει τον ίδιο αριθμό μαύρων κόμβων



Εικόνα 2.3: Red Black Tree

2.3.2. Βασικές Έννοιες

- CPU Bursts: Ο χρόνος που χρησιμοποιεί μία διεργασία τη CPU.
    - Αν έχω μικρή χρήση της CPU, τότε έχω και μικρά CPU Bursts.
    - Το δέντρο κατασκευάζεται με βάση τις τιμές που έχουν τα vruntime:
        - Όσο πιο μικρό, τόσο πιο αριστερά βρίσκεται στο Δέντρο. Ο Χρονοδρομολογητής θεωρεί πως οι Interactive διεργασίες χρησιμοποιούν λιγότερο την CPU, έχουν μικρότερο vruntime, άρα και μεγαλύτερη προτεραιότητα εκτέλεσης.
        - Όσο πιο μεγάλο, τόσο πιο δεξιά βρίσκεται στο Δέντρο. Ο Χρονοδρομολογητής θεωρεί πως οι Non-Interactive διεργασίες χρησιμοποιούν περισσότερο τη CPU, έχουν μεγαλύτερο vruntime, άρα και μικρότερη προτεραιότητα εκτέλεσης.

- Κάθε διεργασία που εισέρχεται στο σύστημα έχει αρχικά vruntime ίσο με 1. Μετά την πρώτη εκτέλεση της αυτό αλλάζει και υπολογίζεται ως εξής:

- vruntime = vruntime + (t*weight), όπου
  - t, ο χρόνος που έτρεξε η διεργασία.
  - Weight, είναι ένα στάθμισμα αυτού το χρόνου και υπολογίζεται από τη σχέση: Weight = 1.25^{nice} * 1024.
  - Το nice είναι μία μεταβλητή που δίνεται από το σύστημα και έχει πεδίο ορισμού το [-20, 19].
- Όσο μικραίνει το nice, μικραίνει και το weight, με συνέπεια να έχουμε μικρότερο vruntime, δηλαδή μεγαλύτερη προτεραιότητα για την διεργασία.

- Target Latency (TL), (Στοχευμένη Καθυστέρηση): Έστω ότι έχω N διεργασίες οι οποίες περιμένουν να εκτελεστούν. Το σύστημα θα ορίσει έναν ελάχιστο χρόνο για να αξιοποιήσουν όλες οι διεργασίες την CPU με τη σειρά. Αν έχω TL = 20ms και τέσσερις διεργασίες (N=4), τότε κάθε διεργασία μέσα στο διάστημα τον 20ms θα πάρει από 5ms. Ορίζουμε τυπικά TL = 20ms.

- Minimum Granulatiry (MG), (Ελάχιστος Χρόνος Εκτέλεσης): Είναι η ελάχιστη μονάδα ή ελάχιστος χρόνος εκτέλεσης μια διεργασίας, ορίζουμε τυπικά MG = 4ms.

- Το Target Latency και το Minimum Granularity είναι δυο μεταβλητές τιμές, οι οποίες βοηθάνε στο να μην υπάρξει υπερφόρτωση του συστήματος. Αν ισχύει η σχέση N*MG > TL, τότε το σύστημα είναι υπερφορτωμένο. Για να αποτραπεί αυτό αναγκαζόμαστε να αυξήσουμε το TL (TL = N*MG). Αν το πλήθος των διεργασιών N υπερβεί το όριο, τότε θα μειωθεί και το MG.

  - Παρατήρηση: Δεν θέλουμε το TL να αυξηθεί κατά ένα μεγάλο ποσοστό, καθώς επίσης δεν θέλουμε και το MG να πάρει τιμή μικρότερη από 1ms, διότι θα υπάρξει καθυστέρηση στο σύστημα.

- Κάθε διεργασία έχει τον δικό της αριθμό κβάντων και τον συμβολίζουμε με την μεταβλητή K. Υπολογίζεται από τον λόγο K = 1024/1.25^{nice}. Αφού υπολογίσουμε όλα τα K τα προσθέτουμε για να βρούμε τη μεταβλητή M (M = N*K). Έπειτα, χρησιμοποιώντας τη σχέση TL*(K/M), βρίσκουμε ποιο κομμάτι του TL χρησιμοποιεί μία διεργασία. Αφού έχουμε αυτό τον αριθμό τον διαιρούμε από το TL, το πολλαπλασιάζουμε με το 100 και βρίσκουμε το ποσοστό τοις εκατό που χρησιμοποιεί μία διεργασία.

2.3.3 Βήματα εκτέλεσης του CFS με τη χρήση Red Black Trees

Έστω οι διεργασίες P0 – P6 με P0<P1<…<P6, όσον αφορά τα Vruntimes τους.

- Η πρώτη διεργασία που εισέρχεται στο σύστημα είναι η P0. Ως νέα εισαγωγή ο κόμβος θα είναι κόκκινος, επειδή όμως αποτελεί τη ρίζα του Red Black Tree θα γίνει μαύρος.

- Το P1 έχει μεγαλύτερο Vruntime, άρα θα πάει δεξιά του P0.

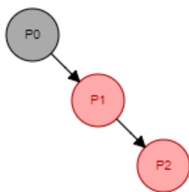
- Το P2 ως μεγαλύτερο του P1, θα πάει δεξιά του. Έχουμε δύο συνεχόμενους κόκκινους κόμβους (Σχήμα 1), οπότε ελέγχουμε τον θείο του καινούργιου κόμβου. Ο θείος του P2 είναι NULL, άρα μαύρος, επομένως θα έχουμε περιστροφή του δέντρου (Σχήμα 2α) και στη συνέχεια θα αλλάξουμε τα χρώματα των κόμβων.

- Η καινούργια ρίζα P1 θα γίνει μαύρη και τα παιδιά θα γίνουν κόκκινα (Σχήμα 2β).

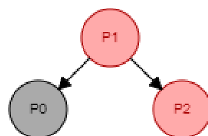
- Το P3 είναι μεγαλύτερο του P2, άρα θα γίνει δεξί παιδί του. Έχουμε δυο διαδοχικούς κόκκινους κόμβους (Σχήμα 3α). Ελέγχουμε τον θείο του P3, P0, ο οποίος είναι κόκκινος, άρα έχουμε αλλαγή χρώματος, η ρίζα του υποδέντρου που βρισκόμαστε θα γίνει κόκκινη και τα παιδιά μαύρα (Σχήμα 3β).

- Τέλος, αλλάζουμε τη ρίζα του δέντρου σε μαύρη, ώστε να έχουμε ένα Red Black Tree.

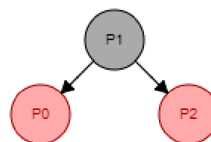
Σχήμα 1



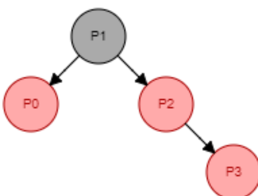
Σχήμα 2α



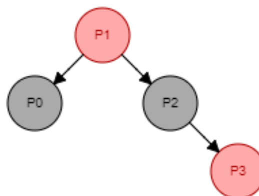
Σχήμα 2β



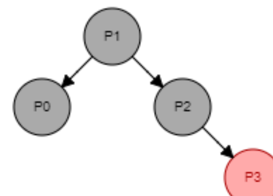
Σχήμα 3α



Σχήμα 3β



Σχήμα 3γ



- Το P4 είναι μεγαλύτερο του P3, θα πάει στα δεξιά του και θα είναι κόκκινο (Σχήμα 4α).

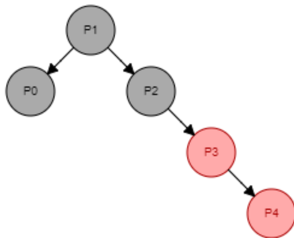
- Έχω δύο διαδοχικούς κόκκινους κόμβους, άρα εξετάζουμε τον θείο του P4, ο οποίος είναι NULL, δηλαδή μαύρος, οπότε πρέπει να κάνω περιστροφή στο δέντρο. Το P4 θα πάει στη θέση του P3, το P3 στη θέση του P2 και το P2 θα γίνει αριστερό παιδί του P3 (Σχήμα 4β).

- Κάνουμε τη ρίζα του υποδέντρου μαύρη και τα παιδιά κόκκινα (Σχήμα 4γ).

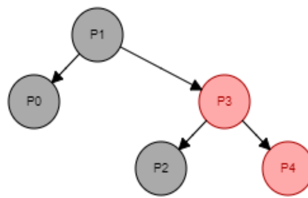
- Το P5 ως μεγαλύτερο του P4 θα πάει και αυτό στα δεξιά. Έχω δύο κόκκινους κόμβους στη σειρά (Σχήμα 5α). Ο θείος του P5 είναι κόκκινος, επομένως θα υπάρξουν μόνο αλλαγές χρωμάτων (Σχήμα 5β)

- Εισάγεται το P6 στο δέντρο και έχουμε δύο συνεχόμενους κόκκινους κόμβους (Σχήμα 6α). Ο θείος του είναι NULL (μαύρος), οπότε θα γίνει περιστροφή του υποδέντρου με αποτέλεσμα να έχουμε μαύρη ρίζα και κόκκινα παιδιά (Σχήμα 6β).

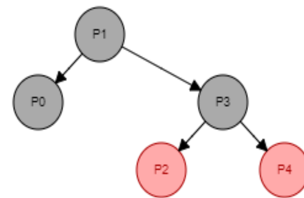
Σχήμα 4α



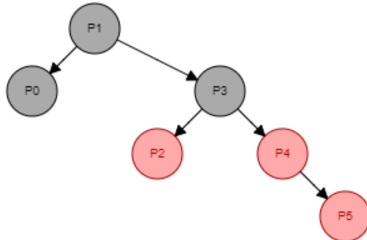
Σχήμα 4β



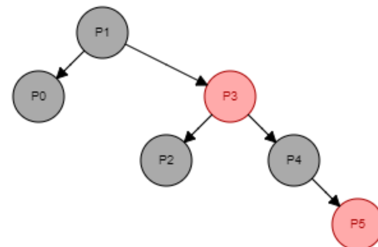
Σχήμα 4γ



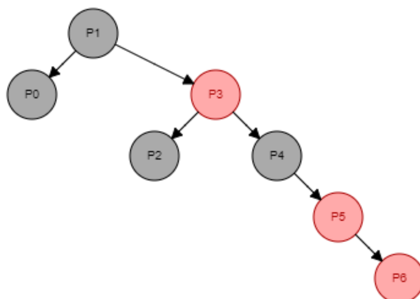
Σχήμα 5α



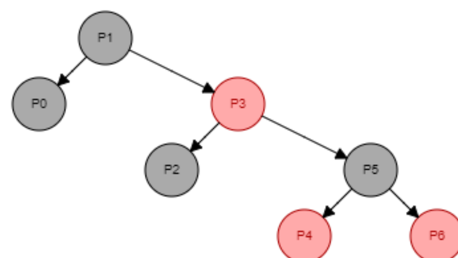
Σχήμα 5β



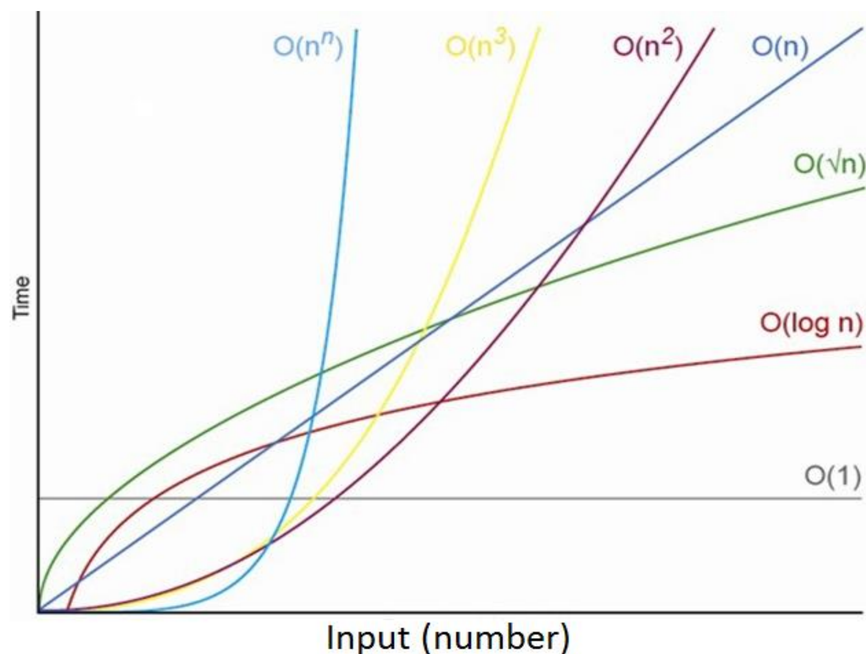
Σχήμα 6α



Σχήμα 6β



- Με τη βοήθεια του παραπάνω παραδείγματος, είναι εύκολο να αποδείξουμε ότι η πολυπλοκότητα του αλγορίθμου είναι $O(\log_2 N)$. Έστω ότι θέλω να προσθέσω έναν ακόμη κόμβο P7 > P6. Αρχικά συγκρίνω το P7 με το P1. P7 > P1 οπότε κατευθύνομαι προς τα δεξιά. Αυτό σημαίνει επίσης πως «κόβω» το δέντρο στα δύο. Οτιδήποτε υπάρχει στην αριστερή μεριά του δεν θα συμπεριληφθεί στις συγκρίσεις. Στη συνέχεια συγκρίνω με το P3. P7 > P3, άρα μετακινούμαστε πάλι δεξιά. Ομοίως με πριν, το δέντρο «κόβεται» πάλι στα δύο. Πραγματοποιούνται άλλες δύο παρόμοιες συγκρίσεις με τα P5 και P6. Στο σύνολο υπήρξαν 4 συγκρίσεις, με 4 = [ακέραιος του $\log_2 7$] + 1 = 3 + 1 = 4. Υπάρχει περίπτωση να έχω λιγότερες συγκρίσεις, αλλά ποτέ περισσότερες. Επειδή στη σχέση [ακέραιος του ακέραιος του $\log_2 N$] + 1, το +1 είναι σταθερού χρόνου, λέμε ότι η πολυπλοκότητα είναι $O(\log_2 N)$, όπου N οι διεργασίες που μπήκαν στο δέντρο.

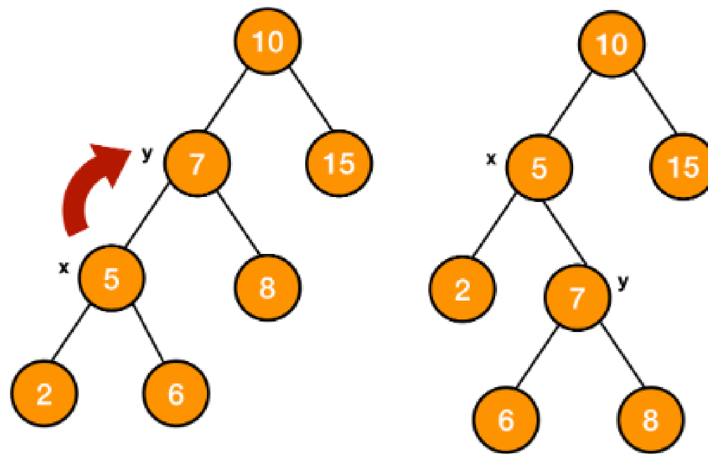


Εικόνα 2.4: Οι συναρτήσεις πολυπλοκότητας

2.3.4 Κανόνες Περιστροφής των Red Black Trees[14]

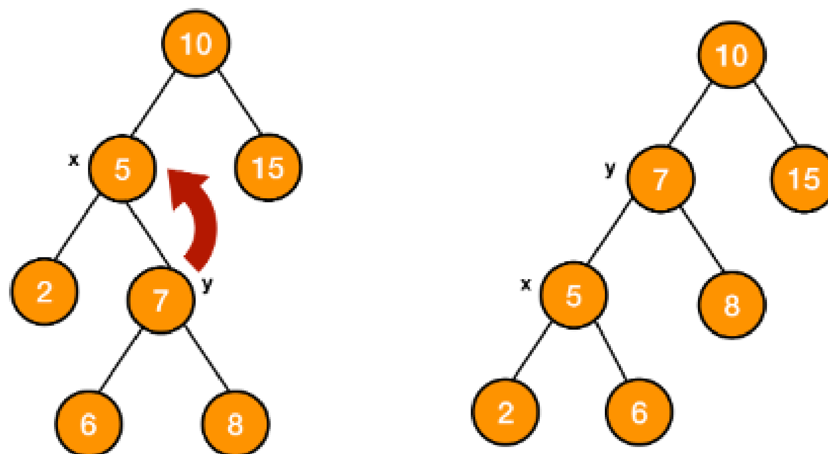
Υπάρχουν δύο ειδών περιστροφές στα Red Black Trees: η Δεξιά και η Αριστερή.

- Στη Δεξιά Περιστροφή γίνεται η υπόθεση ότι το αριστερό παιδί του υποδέντρου που θα περιστραφεί, δεν είναι κενό (NULL). Εφαρμόζεται η περιστροφή στον κόμβο y, αυτό θα έχει ως αποτέλεσμα ο κόμβος x να γίνει η νέα ρίζα του υποδέντρου, το y δεξί παιδί του x και το προηγούμενο δεξιό παιδί του x, θα είναι πλέον αριστερό παιδί του y.



Εικόνα 2.5: Δεξιά Περιστροφή

- Στην Αριστερή Περιστροφή γίνεται η υπόθεση ότι το δεξί παιδί του υποδέντρου που θα περιστραφεί, δεν είναι κενό (NULL). Εφαρμόζεται η περιστροφή στον κόμβο x, αυτό θα έχει ως αποτέλεσμα ο κόμβος y να γίνει η νέα ρίζα του υποδέντρου, το x αριστερό παιδί του y και το προηγούμενο αριστερό παιδί του y, θα είναι πλέον δεξί παιδί του x.



Εικόνα 2.6: Αριστερή Περιστροφή

Κεφάλαιο 3ο : Περιγραφή και Ανάλυση του Κώδικα

Η υλοποίηση του Completely Fair Scheduler, πραγματοποιήθηκε με τη χρήση της Java, στο εικονικό περιβάλλον του Eclipse. Ο κώδικας αποτελείται από πέντε κλάσεις:

- Configuration.java
- Main.java
- Node.java
- RedBlackTree.java
- Simulation.java

3.1. Η κλάση Configuration.java

Η κλάση Configuration είναι μία κλάση η οποία συνεισφέρει στις ρυθμίσεις του συστήματος προσαρμόζοντας, ανάλογα με την είσοδο των διεργασιών n, τις μεταβλητές MG (Minimum Granularity) και TL (Target Latency). Αν το TL ανήκει στο διάστημα [80, 160], (δηλαδή n∈[20,40]), τότε το MG θα μειωθεί σε δύο από τέσσερα, ενώ αν ξεπεράσει την τιμή 160, τότε θα έχουμε υπερφόρτωση συστήματος και θα τρέξουν οι διεργασίες που βρίσκονται ήδη μέσα στο σύστημα χωρίς την προσθήκη νέων.

```java
public class Configuration {
    public int MG = 4;
    public int TL = 20;

    public void calculateConfigurations(int n) {                     //apofygh yperfortwshs
        TL = n*MG;
        if(TL >= 80 && TL<=160){
            MG = 2;
        }else if(TL > 160) {
            System.out.println("The system is overloaded, number of proccesses must 40 or less.");
            System.exit(0);
        }else if (TL > 0 && TL < 80){
            MG = 4;
        }
    }

    public int getTL() {
        return TL;
    }
}
```

Εικόνα 3.1: Η συνάρτηση Configuration

3.2. Η κλάση Main.java

Η main έχει τη μοναδική λειτουργία εκκίνησης του προγράμματος. Δημιουργεί ένα αντικείμενο τύπου Simulation και το χρησιμοποιεί για να πραγματοποιήσει την προσομοίωση.

```
public class Main {
    public static void main(String[] args) {
        Simulation sim = new Simulation();
        sim.initialize();
        sim.simulate();
    }
}
```

Εικόνα 3.2: Η συνάρτηση Main

3.3. Η κλάση Node.java

Η κλάση Node έχει ως σκοπό να δομήσει και να δημιουργήσει σωστά μία διεργασία, δηλαδή έναν κόμβο που εισέρχεται στο σύστημα. Τα χαρακτηριστικά που βοηθάνε στην εισαγωγή του στο Red Black tree είναι: το όνομα του, ο γονέας του και τα παιδιά του (αν υπάρχουν), το χρώμα (κόκκινο όταν καταφθάνει στο σύστημα), καθώς και η μεταβλητή nice.

```
public class Node {
    public double vruntime = 1; // holds the key
    public String name;
    public Node parent; // pointer to the parent
    public Node left; // pointer to left child
    public Node right; // pointer to right child
    public int color; // 1 . Red, 0 . Black
    public int nice = giveRandomNice();
    public double weight = calculateWeight();
    public double quantum = calculateQuantum();
    public double runTime = giveRunTime();
    public boolean niceFromUser = false;

    public Node(String name, int MG) {
        this.name = name;
        parent = null;
        vruntime = calculateVruntime(MG);
        left = RedBlackTree.TNULL;
        right = RedBlackTree.TNULL;
        color = 1; // new node must be red
    }
```

Εικόνα 3.3: Η συνάρτηση Node

Παρακάτω δίνονται οι συναρτήσεις που βοηθούν στη δημιουργία ενός Node. To nice δίνεται είτε από τον χρήστη, είτε από τη συνάρτηση Random(). Επίσης υπολογίζονται οι μεταβλητές Vruntime, Weight και Quantum.

```
public int giveRandomNice() {
    int min = -20;
    int max = 20;
    return new Random().nextInt(max - min) + min;
}

public void setNiceFromUser(Scanner scan) {
    int nice = scan.nextInt();
    while(nice < -20 || nice > 19) {
        System.out.println("Nice needs to be between [-20, 19]");
        nice = scan.nextInt();
    }
    this.nice = nice;
    this.niceFromUser = true;
}

public double calculateVruntime(int MG) {
    return vruntime + MG * weight;
}

public double calculateWeight() {
    return Math.pow(1.25, nice) * 1024;
}

public double calculateQuantum() {
    return quantum = 1024/Math.pow(1.25, nice);
}
```

Εικόνα 3.4: Οι συναρτήσεις giveRandomNice, setNiceFromUser, calculateVruntime, calculateWeight, calculateQuantum

Από εκεί και πέρα, όπως είδαμε και στην Ενότητα 2.3.2, κάθε διεργασία περιλαμβάνει επίσης ένα δικό της Weight, δικά της Κβάντα και δικό της Χρόνο Εκτέλεσης (Runtime σε ms). Δεν είναι απίθανο όμως πολλές διεργασίες να έχουν ίδιες αυτές τις τιμές με άλλες διεργασίες που υπάρχουν στο σύστημα.

```
public double giveRunTime() {
    int min = 10;
    int max = 15;
    return new Random().nextInt(max - min) + min;
}
```

Εικόνα 3.5: Η συνάρτηση giveRunTime

3.4. Η κλάση RedBlackTree.java

- Η κλάση RedBlackTree ξεκινάει με έναν κατασκευαστή, ο οποίος δημιουργεί το δέντρο που χρειάζεται ο αλγόριθμος, το οποίο στην αρχική του μορφή, είναι απλά ένας άδειος κόμβος.

```
public RedBlackTree() {
    TNULL = new Node("NULL", 0);
    TNULL.color = 0;
    TNULL.left = null;
    TNULL.right = null;
    this.root = TNULL;
}
```

Εικόνα 3.6: Η συνάρτηση RedBlackTree

- Η συνάρτηση minimum είναι βοηθητική, βρίσκει ποιος κόμβος βρίσκεται στο αριστερότερο άκρο του δέντρου, δηλαδή ποιος κόμβος έχει το μικρότερο Vruntime.

```
public Node minimum(Node node) {
    while (node.left != TNULL) {
        node = node.left;
    }
    return node;
}
```

Εικόνα 3.7: Η συνάρτηση minimum

- Στη συνέχεια, ακολουθεί η συνάρτηση Insert. Η λειτουργία της είναι να δέχεται μια μεταβλητή τύπου Node (Κόμβου) και να την τοποθετεί στο δέντρο. Η συγκεκριμένη συνάρτηση έχει ως σκοπό μόνο να τοποθετεί τον νέο κόμβο στο δέντρο και στη θέση που πρέπει με βάση το Vruntime. Δεν διαχειρίζεται άλλες λειτουργίες, όπως περιστροφή του δέντρου ή αλλαγές χρωμάτων. Αυτές είναι διαδικασίες που εκτελεί η fixInsert και για αυτόν τον λόγο καλείται στο τέλος.

```
public void insert(Node node) {
    Node y = null;
    Node x = this.root;

    while (x != TNULL) {
        y = x;
        if (node.vruntime< x.vruntime) {
            x = x.left;
        } else {
            x = x.right;
        }
    }

    // y is parent of x
    node.parent = y;
    if (y == null) {
        root = node;
    } else if (node.vruntime < y.vruntime) {
        y.left = node;
    } else {
        y.right = node;
    }

    // if new node is a root node, simply return
    if (node.parent == null){
        node.color = 0;
        return;
    }

    // if the grandparent is null, simply return
    if (node.parent.parent == null) {
        return;
    }

    // Fix the tree
    fixInsert(node);
}
```

Εικόνα 3.8: Η συνάρτηση insert

- Όταν ένας κόμβος εισέρχεται, η Insert τον τοποθετεί σε μία θέση με βάση το Vruntime. Αυτό όμως δε σημαίνει πως το δέντρο που έχουμε είναι Red Black Tree. Εδώ έρχεται σε λειτουργία η συνάρτηση fixInsert, η οποία είναι υπεύθυνη για τη δομή του δέντρου. Κάνει τους κατάλληλους ελέγχους και με βάση τους κανόνες των Red Black Trees, αλλάζει τα χρώματα των κόμβων και καλεί την ανάλογη συνάρτηση για να γίνει περιστροφή στο δέντρο.

```
private void fixInsert(Node k){
    Node u;
    while (k.parent.color == 1) {
        if (k.parent == k.parent.parent.right) {
            u = k.parent.parent.left; // uncle
            if (u.color == 1) {
                // case 3.1
                u.color = 0;
                k.parent.color = 0;
                k.parent.parent.color = 1;
                k = k.parent.parent;
            } else {
                if (k == k.parent.left) {
                    // case 3.2.2
                    k = k.parent;
                    rightRotate(k);
                }
                // case 3.2.1
                k.parent.color = 0;
                k.parent.parent.color = 1;
                leftRotate(k.parent.parent);
            }
        } else {
            u = k.parent.parent.right; // uncle

            if (u.color == 1) {
                // mirror case 3.1
                u.color = 0;
                k.parent.color = 0;
                k.parent.parent.color = 1;
                k = k.parent.parent;
            } else {
                if (k == k.parent.right) {
                    // mirror case 3.2.2
                    k = k.parent;
                    leftRotate(k);
                }
                // mirror case 3.2.1
                k.parent.color = 0;
                k.parent.parent.color = 1;
                rightRotate(k.parent.parent);
            }
        }
        if (k == root) {
            break;
        }
    }
    root.color = 0;
}
```

Εικόνα 3.9:Η συνάρτηση fixInsert

- Οι συναρτήσεις rightRotate και leftRotate περιστρέφουν τους κόμβους ώστε να έρθει το δέντρο στη σωστή μορφή. Οι συγκρίσεις γίνονται με βάση τους κανόνες των Red Black Trees.

```
// rotate right at node x
public void rightRotate(Node x) {
    Node y = x.left;
    x.left = y.right;
    if (y.right != TNULL) {
        y.right.parent = x;
    }
    y.parent = x.parent;
    if (x.parent == null) {
        this.root = y;
    } else if (x == x.parent.right) {
        x.parent.right = y;
    } else {
        x.parent.left = y;
    }
    y.right = x;
    x.parent = y;
}
// rotate left at node x
public void leftRotate(Node x) {
    Node y = x.right;
    x.right = y.left;
    if (y.left != TNULL) {
        y.left.parent = x;
    }
    y.parent = x.parent;
    if (x.parent == null) {
        this.root = y;
    } else if (x == x.parent.left) {
        x.parent.left = y;
    } else {
        x.parent.right = y;
    }
    y.left = x;
    x.parent = y;
}
```

Εικόνα 3.10: Οι συναρτήσεις rightRotate και leftRotate

3.5. Η κλάση Simulation.java

- Η κλάση Simulation είναι αρμόδια για την προσομοίωση του Completely Fair Scheduler. Σε πρώτη φάση δημιουργείται ένα άδειο Red Black Tree και μια άδεια λίστα που θα περιλαμβάνει τους κόμβους.

```
public class Simulation {
    private RedBlackTree rbt;
    private int numberOfProcesses = 0;
    private boolean choseNo = false;
    private Scanner scan;
    private double totalQuantum;
    private ArrayList<Node> allNodes = new ArrayList<>();
    private Configuration configuration = new Configuration();

    public Simulation() {
        scan = new Scanner(System.in);
        rbt = new RedBlackTree();
    }
```

Εικόνα 3.11: Η συνάρτηση Simulation

- Στη συνάρτηση initialize, γίνεται η αρχικοποίηση των διεργασιών. Όλες οι διεργασίες εισέρχονται στο σύστημα με Vruntime ίσο με 1. Υπολογίζονται τα νέα Vruntimes καθώς επίσης και οι υπόλοιπες μετρικές, όπως η χρήση του TL που κάνει κάθε διεργασία, τα κβάντα της, ο χρόνος εκτέλεσης και το σύνολο των κβάντων που καταναλώθηκαν από όλες τις διεργασίες και στη συνέχεια μπαίνουν στο δέντρο με σωστή σειρά.

```
public void initialize() {
    requireNumberOfProcesses();
    createProcesses(numberOfProcesses);
    printProcesses();
    addProcesses(allNodes, true);

    configuration.calculateConfigurations(numberOfProcesses);

    System.out.println("Calculating new Vruntimes and inserting the processes to the tree.");

    printList();
    calculateTotalQuantum(allNodes);

    System.out.println();
    printTotalQuantum();
    System.out.println();
    printTLUsage(allNodes);
}
```

Εικόνα 3.12: Η συνάρτηση initialize

- Με τη συνάρτηση simulate γίνεται επίσημα η εκκίνηση της προσομοίωσης. Όπως φαίνεται και στον επαναληπτικό βρόγχο while, όσο το δέντρο δεν είναι κενό, δηλαδή υπάρχει μέσα σίγουρα ένας κόμβος για να αποτελέσει ρίζα του, η simulate καλεί νέες συναρτήσεις που βοηθούν στη σωστή λειτουργία του "CFS" με αφαίρεση διεργασιών όταν τελειώσει ο χρόνος εκτέλεσής τους ή και προσθήκη νέων που μόλις έχουν καταφθάσει στο σύστημα.

```
public void simulate() {
    ArrayList<Node> newNodes = new ArrayList<Node>();
    System.out.println("\n***** Simulation *****");
    while(rbt.getRoot() != RedBlackTree.TNULL) {

        System.out.println("*** Processes Running ***");
        runNewProcessesFirst(newNodes);

        runInorder(rbt.getRoot());

        System.out.println();

        configureNewProcesses(newNodes);
        checkOrDeleteRuntime();

        if(!newNodes.isEmpty()) {
            totalQuantum = 0;
            totalQuantum = calculateTotalQuantum(allNodes);
        }

        printTotalQuantum();

        if(!newNodes.isEmpty()) {
            printTLUsage(allNodes);
        }

        newNodes = moreProcesses();
    }
    System.out.println("***** Simulation Finished *****");
}
```

Εικόνα 3.13: Η συνάρτηση simulate

- Η συνάρτηση createProcesses αρχικά ρωτάει τον χρήστη αν θέλει να δώσει τιμή στην μεταβλητή nice ο ίδιος, ενώ στη συνέχεια δημιουργείται η νέα διεργασία. Η συνάρτηση createProcess, δημιουργεί το node που θα μπει στο δέντρο και τη διεργασία που θα μπει σε μία λίστα μαζί με τις υπόλοιπες, του δίνονται τα χαρακτηριστικά που έχει κάθε διεργασία και αναλόγως την απάντηση που έδωσε ο χρήστης δίνεται και το nice, είτε από την κονσόλα είτε από το σύστημα. Οι παρακάτω συναρτήσεις χρησιμοποιούνται μόνο για την δημιουργία των πρώτων διεργασιών που θα τρέξουν (όσες κι αν είναι αυτές, έως και 40).

```java
private void createProcesses(int n) {
    System.out.println("Give a custom nice value to processes? (1 : YES / 0 : NO)");
    int niceAnswer = scan.nextInt();
    for (int i=0; i<n; i++) {
        allNodes.add(createProcess(niceAnswer, allNodes.size()));
    }
}

private Node createProcess(int niceAnswer, int size) {
    Node node = new Node("P" + size, configuration.MG);
    if (niceAnswer == 1) {
        System.out.println("Give a nice [-20, 19] for Process P" + size);
        node.setNiceFromUser(scan);
    }
    node.vruntime = 1;
    return node;
}
```

Εικόνα 3.14: Οι συναρτήσεις createProcesses και createProcess

- Η συνάρτηση moreProcesses, αφού τρέξουν πρώτα μία φορά οι διεργασίες που μπήκανε πρώτες στο σύστημα, ρωτάει τον χρήστη αν θέλει να προσθέσει κι άλλες, αν απαντήσει όχι (0) τότε δεν προστίθενται καινούργιες, ο αλγόριθμος τρέχει τις ήδη υπάρχουσες και ολοκληρώνεται. Αν αντιθέτως απαντήσει ναι (1), τότε το σύστημα τον ενημερώνει, για το πόσες ακόμη μπορεί να προσθέσει, ο αριθμός που εισάγεται ελέγχεται από έναν βρόγχο while για το αν μπορεί να γίνει δεκτός. Έπειτα ακολουθεί η ίδια διαδικασία για την μεταβλητή nice και τέλος ο νέος αριθμός διεργασιών μεταφέρεται στη συνάρτηση της κλάσης Configuration, ώστε να γίνουν οι απαραίτητες αλλαγές στο TL ή στο MG.

```java
public ArrayList<Node> moreProcesses() {
    ArrayList<Node> newNodes = new ArrayList<Node>();
    // meta apo ena oloklhro treksimo
    int answer = 0;
    if (!choseNo && allNodes.size() < 40) {
        System.out.println("Would you like to add more processes to the system? (1 : YES / 0 : NO)");
        answer = scan.nextInt();

        if(answer == 0) {
            choseNo = true;
        }
    }
    if(answer == 1) {
        System.out.println("You can add " + (40 - allNodes.size()) + " processes.");
        System.out.println("How many more would you like to add?");
        int extraProcesses = scan.nextInt();

        if (extraProcesses <= 0) {
            return new ArrayList<>();
        }
```

```
        while(40 - allNodes.size() - extraProcesses < 0) {
            System.out.println("Insert a different number of processes.");
            extraProcesses = scan.nextInt();
        }
        numberOfProcesses = allNodes.size() + extraProcesses;
        System.out.println("Give a custom nice value to processes? (1 : YES / 0 : NO)");
        int niceAnswer = scan.nextInt();
        for (int i=0; i<extraProcesses; i++) {
            // create new nodes
            Node newNode = new Node("P" + (allNodes.size() + newNodes.size()), configuration.MG);
            newNode.vruntime = 1;
            if (niceAnswer == 1) {
                System.out.println("Give a nice [-20, 19] for Process "+ newNode.name);
                newNode.setNiceFromUser(scan);
            }
            newNodes.add(newNode);
        }
        configuration.calculateConfigurations(allNodes.size() + newNodes.size());
        return newNodes;
    }else {
        return new ArrayList<Node>();
    }
}
```

Εικόνα 3.15: Η συνάρτηση moreProcesses

- Η συνάρτηση runNewProcessesFirst, τρέχει πρώτα μία φορα τις νέες διεργασίες που εισήλθαν στο σύστημα και έχουν Vruntime ίσο με 1. Η configureNewProcesses, είναι υπεύθυνη για να δώσει στις νέες διεργασίες τα χαρακτηριστικά τους (nice, Weight, Κβάντα, Vruntime) και στη συνέχεια να τις προσθέσει στο Red Black Tree.

```
public void runNewProcessesFirst(ArrayList<Node> newNodes) {
    for(Node n: newNodes) {
        System.out.println("Process: " + n.name + " Runtime before: " + n.runTime);
        if(n.runTime < configuration.MG) {
            n.runTime = 0;
        }else {
            n.runTime -= configuration.MG;
        }
        System.out.println("Process: " + n.name + " Runtime after: " + n.runTime);
    }
}

public void configureNewProcesses(ArrayList<Node> newNodes) {
    for(Node n: newNodes) {
        if (!n.niceFromUser) {
            n.nice = n.giveRandomNice();
        }
        n.weight = n.calculateWeight();
        n.quantum = n.calculateQuantum();
        n.vruntime = n.calculateVruntime(configuration.MG);
        n.printNode();
        rbt.insert(n);
        allNodes.add(n);
        rbt.printStep(n, true);
    }
    numberOfProcesses = allNodes.size();
}
```

Εικόνα 3.16: Η συνάρτηση runNewProcessesFirst

- Η συνάρτηση checkOrDeleteRuntime, ελέγχει μετά από κάθε τρέξιμο αν τελείωσε το Runtime κάποιες διεργασίας που βρίσκεται στο δέντρο. Σε περίπτωση που αυτό ισχύει, τότε την διαγράφει από τη λίστα διεργασιών και καλεί την fixTree. Εμφανίζει στη κονσόλα το δέντρο χωρίς τη διεργασία που διαγράφηκε και υπολογίζονται ξανά το TL και το MG.

- Η συνάρτηση fixTree «μηδενίζει» τα nodes του δέντρου, δηλαδή αφαιρεί όλους τους κόμβους, διαγράφει τους pointers που δείχνουν στον γονέα και στα παιδιά τους, και τους ξαναβάζει από την αρχή στο δέντρο με σωστό τρόπο χωρίς τις διεργασίες που έχουν διαγραφεί.

```java
public void checkOrDeleteRuntime() {
    ArrayList<Node> nodesToBeDeleted = new ArrayList<Node>();
    inorderHelper(rbt.getRoot(), nodesToBeDeleted);
    boolean deleted = false;
    for (Node node: nodesToBeDeleted) {
        // delete node from tree
        allNodes.remove(node);
        fixTree(allNodes);
        deleted = true;
        rbt.printStep(node, false);
    }
    if(deleted) {
        configuration.calculateConfigurations(allNodes.size());
    }
}

private void fixTree(ArrayList<Node> nodes) {
    this.rbt = new RedBlackTree();
    for(Node n: nodes) {
        n.parent = null;
        n.left = RedBlackTree.TNULL;
        n.right = RedBlackTree.TNULL;
        n.color = 1;
        this.rbt.insert(n);
    }
}
```

Εικόνα 3.17: Οι συναρτήσεις checkOrDeleteRuntime και fixTree

- Η συνάρτηση inorderHelper είναι αναδρομική και ψάχνει μέσα στο δέντρο τις διεργασίες που πλέον έχουν Runtime ίσο με μηδέν, με σκοπό να της βάλει σε μία καινούργια λίστα, nodesToBeDeleted, ώστε να διαγραφούν διότι ολοκληρώθηκαν.

- Η συνάρτηση runInorder είναι επίσης αναδρομική, τρέχει τις διεργασίες με βάση το Vruntime τους, από το μικρότερο (μεγαλύτερη ανάγκη χρήσης CPU) στο μεγαλύτερο (μικρότερη ανάγκη χρήσης CPU) και ενημερώνει τον χρήστη για Runtime κάθε διεργασίας πριν και αφού τρέξει.

```
public void inorderHelper(Node node, ArrayList<Node> nodesToBeDeleted) {
    if(node != RedBlackTree.TNULL) {
        inorderHelper(node.left, nodesToBeDeleted);
        if(node.runTime == 0) {
            nodesToBeDeleted.add(node);
        }
        inorderHelper(node.right, nodesToBeDeleted);
    }
}

public void runInorder(Node node) {
    if(node != RedBlackTree.TNULL) {
        runInorder(node.left);
        System.out.println("Process: " + node.name + " Runtime before: " + node.runTime);
        if(node.runTime < configuration.MG) {
            node.runTime = 0;
        }else {
            node.runTime -= configuration.MG;
        }
        System.out.println("Process: " + node.name + " Runtime after: " + node.runTime);
        runInorder(node.right);
    }
}
```

Εικόνα 3.18: Οι συναρτήσεις inorderHelper και runInorder

- Η συνάρτηση addProcesses υπολογίζει τα Vruntimes των διεργασιών που καταφθάνουν στο σύστημα και έπειτα τις προσθέτει στο Red Black Tree

```
public void addProcesses(ArrayList<Node> nodesToAdd, boolean randomizeVruntime) {
    for(Node node: nodesToAdd) {
        if(randomizeVruntime) {
            node.vruntime = node.calculateVruntime(configuration.MG);
        }
        rbt.insert(node);
        rbt.printStep(node, true);
    }
}
```

Εικόνα 3.19: Η συνάρτηση addProcesses

- Η συνάρτηση TL Usage δείχνει το ποσοστό της εκατό που χρησιμοποιεί από το TL κάθε διεργασία.

```
public void printTLUsage(ArrayList<Node> nodes) {
    for(Node node: nodes) {
        double a = configuration.getTL()*(node.getQuantum()/totalQuantum);
        double P = a/configuration.getTL()*100;
        System.out.print("Proccess " + node.name + " is using ");
        System.out.printf("%.2f", P);
        System.out.print("% which is: ");
        System.out.printf("%.2f", a);
        System.out.print(" of the TL");
        System.out.println();
    }
}
```

Εικόνα 3.20: Η συνάρτηση printTLUsage

- Οι παρακάτω συναρτήσεις δείχνουν αναλυτικά τη κατάσταση που βρίσκεται κάθε διεργασία καθ' όλη τη διάρκεια της διαδικασίας του Completely Fair Scheduler.

```
public double calculateTotalQuantum(ArrayList<Node> nodes) {
    for (Node node: nodes) {
        this.totalQuantum += node.getQuantum();
    }
    return this.totalQuantum;
}

public void printTotalQuantum() {
    System.out.printf("The number of total quantums is: " + "%.2f" , totalQuantum );
    System.out.println();
}

public void printProcesses() {
    for (Node n: allNodes) {
        System.out.println("Proccess " + n.name + " Vruntime: " + n.vruntime );
    }
    System.out.println();
}

public void printList() {
    for (Node n: allNodes) {
        n.printNode();
    }
}
```

Εικόνα 3.21: Οι συναρτήσεις calculateTotalQuantum, printTotalQuantum, prinProcesses, printList

Κεφάλαιο 4ο : Αποτελέσματα τις Προσομοίωσης

Στο Κεφάλαιο 3 παρουσιάστηκε ο κώδικας που συγγράφηκε και οι λειτουργίες που εξυπηρετεί κάθε κλάση για να διεξαχθεί σωστά η διαδικασία της προσομοίωσης. Σε αυτό το Κεφάλαιο θα παρουσιαστούν τα αποτελέσματα της και θα επαληθευτεί η ορθότητα της λειτουργίας του συστήματος που μελετάται.

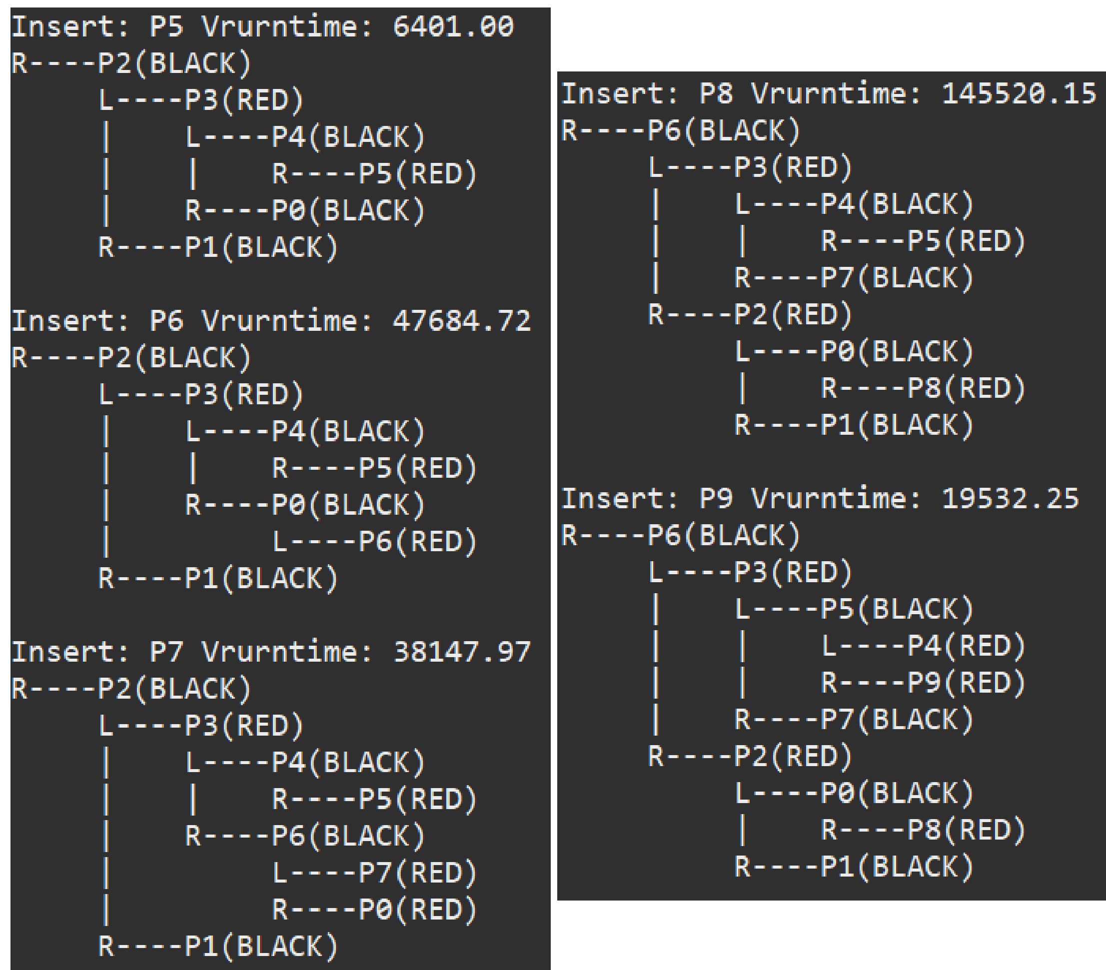
```
Please enter the number of processes (40 or less): 10
Give a custom nice value to processes? (1 : YES / 0 : NO)
0
Proccess P0 Vruntime: 1.0
Proccess P1 Vruntime: 1.0
Proccess P2 Vruntime: 1.0
Proccess P3 Vruntime: 1.0
Proccess P4 Vruntime: 1.0
Proccess P5 Vruntime: 1.0
Proccess P6 Vruntime: 1.0
Proccess P7 Vruntime: 1.0
Proccess P8 Vruntime: 1.0
Proccess P9 Vruntime: 1.0
```

Εικόνα 4.1: Εκκίνηση προσομοίωσης

Η προσομοίωση ξεκίνα με τον χρήστη να βάζει τον αριθμό των διεργασιών (10) που θα εισέλθουν στο σύστημα. Ακόμη του δίνεται η επιλογή να δώσει μία ξεχωριστή τιμή nice στην κάθε διεργασία (στη συγκεκριμένη περίπτωση το nice δίνεται από το σύστημα). Όλες οι διεργασίες μπαίνουν στο σύστημα με Vruntime ίσο με 1.

```
Insert: P0 Vrurntime: 74506.81
R----P0(BLACK)

Insert: P1 Vrurntime: 284218.09
R----P0(BLACK)
     R----P1(RED)

Insert: P2 Vrurntime: 227374.68
R----P2(BLACK)
     L----P0(RED)
     R----P1(RED)

Insert: P3 Vrurntime: 24415.06
R----P2(BLACK)
     L----P0(BLACK)
     |    L----P3(RED)
     R----P1(BLACK)

Insert: P4 Vrurntime: 145.12
R----P2(BLACK)
     L----P3(BLACK)
     |    L----P4(RED)
     |    R----P0(RED)
     R----P1(BLACK)
```

Οι διεργασίες εισέρχονται στο σύστημα, πλέον, με ανανεωμένο Vruntime και αρχίζουν να δομούν το Red Black Tree.



Εικόνες 4.2α, 4.2β, 4.2γ: Η κατασκευή του αρχικού Red Black Tree

Δημιουργείται ο αρχικός πίνακας με τις διεργασίες και με τα χαρακτηριστικά που έχει η κάθε μια, δηλαδή Όνομα, Vruntime, nice, Quantums και Runtime καθώς επίσης και ο συνολικός αριθμός κβάντων του συστήματος μέχρι στιγμής. Ακολουθεί ένας δεύτερος πίνακας που εμφανίζει το ποσοστό επί τοις εκατό και πιο κομμάτι του TL χρησιμοποιεί κάθε διεργασία. Στη συνέχεια φαίνεται η σειρά με την οποία τρέχουν οι διεργασίες. Ξεκινάει η P4, με το μικρότερο Vruntime και τελειώνει η P1, με το μεγαλύτερο. Οι διεργασίες είναι λιγότερες από 20, οπότε θα τρέξουν για 4ms.

```
Calculating new Vruntimes and inserting the processes to the tree.
Name: P0| Vruntime: 74506.81| Nice: 13| Quantums: 56.29| Runtime: 10.0ms
Name: P1| Vruntime: 284218.09| Nice: 19| Quantums: 14.76| Runtime: 14.0ms
Name: P2| Vruntime: 227374.68| Nice: 18| Quantums: 18.45| Runtime: 12.0ms
Name: P3| Vruntime: 24415.06| Nice: 8| Quantums: 171.80| Runtime: 13.0ms
Name: P4| Vruntime: 145.12| Nice: -15| Quantums: 29103.83| Runtime: 10.0ms
Name: P5| Vruntime: 6401.00| Nice: 2| Quantums: 655.36| Runtime: 12.0ms
Name: P6| Vruntime: 47684.72| Nice: 11| Quantums: 87.96| Runtime: 11.0ms
Name: P7| Vruntime: 38147.97| Nice: 10| Quantums: 109.95| Runtime: 13.0ms
Name: P8| Vruntime: 145520.15| Nice: 16| Quantums: 28.82| Runtime: 14.0ms
Name: P9| Vruntime: 19532.25| Nice: 7| Quantums: 214.75| Runtime: 12.0ms

The number of total quantums is: 30461.97

Proccess P0 is using 0.18% which is: 0.07 of the TL
Proccess P1 is using 0.05% which is: 0.02 of the TL
Proccess P2 is using 0.06% which is: 0.02 of the TL
Proccess P3 is using 0.56% which is: 0.23 of the TL
Proccess P4 is using 95.54% which is: 38.22 of the TL
Proccess P5 is using 2.15% which is: 0.86 of the TL
Proccess P6 is using 0.29% which is: 0.12 of the TL
Proccess P7 is using 0.36% which is: 0.14 of the TL
Proccess P8 is using 0.09% which is: 0.04 of the TL
Proccess P9 is using 0.70% which is: 0.28 of the TL
```

Εικόνα 4.3: Ο πίνακας των διεργασιών και των χαρακτηριστικών του (πάνω), ο πίνακας χρήσης TL (κάτω)

```
***** Simulation *****
*** Processes Running ***
Process: P4 Runtime before: 10.0
Process: P4 Runtime after: 6.0
Process: P5 Runtime before: 12.0
Process: P5 Runtime after: 8.0
Process: P9 Runtime before: 12.0
Process: P9 Runtime after: 8.0
Process: P3 Runtime before: 13.0
Process: P3 Runtime after: 9.0
Process: P7 Runtime before: 13.0
Process: P7 Runtime after: 9.0
Process: P6 Runtime before: 11.0
Process: P6 Runtime after: 7.0
Process: P0 Runtime before: 10.0
Process: P0 Runtime after: 6.0
Process: P8 Runtime before: 14.0
Process: P8 Runtime after: 10.0
Process: P2 Runtime before: 12.0
Process: P2 Runtime after: 8.0
Process: P1 Runtime before: 14.0
Process: P1 Runtime after: 10.0
```

Εικόνα 4.4: Οι χρόνοι των πρώτων δέκα διεργασιών, πριν και αφού τρέξουν

Ο χρήστης ρωτάτε εάν θέλει να προσθέσει κι άλλες διεργασίες και να τους δώσει ξεχωριστές τιμές nice. Γίνεται προσθήκη άλλων δέκα διεργασιών, με το nice να δίνεται ξανά από το σύστημα. Οι διεργασίες είναι πλέον είκοσι, οπότε το MG γίνεται ίσο με δύο και κάθε διεργασία τρέχει για 2ms.

```
The number of total quantums is: 30461.97
Would you like to add more processes to the system? (1 : YES / 0 : NO)
1
You can add 30 processes.
How many more would you like to add?
10
Give a custom nice value to processes? (1 : YES / 0 : NO)
0
```

Εικόνα 4.5: Προσθήκη επιπλέον διεργασιών και τιμής nice

```
*** Processes Running ***
Process: P10 Runtime before: 13.0
Process: P10 Runtime after: 11.0
Process: P11 Runtime before: 12.0
Process: P11 Runtime after: 10.0
Process: P12 Runtime before: 10.0
Process: P12 Runtime after: 8.0
Process: P13 Runtime before: 12.0
Process: P13 Runtime after: 10.0
Process: P14 Runtime before: 14.0
Process: P14 Runtime after: 12.0
Process: P15 Runtime before: 12.0
Process: P15 Runtime after: 10.0
Process: P16 Runtime before: 13.0
Process: P16 Runtime after: 11.0
Process: P17 Runtime before: 11.0
Process: P17 Runtime after: 9.0
Process: P18 Runtime before: 12.0
Process: P18 Runtime after: 10.0
Process: P19 Runtime before: 11.0
Process: P19 Runtime after: 9.0
Process: P4 Runtime before: 6.0
Process: P4 Runtime after: 4.0
Process: P5 Runtime before: 8.0
Process: P5 Runtime after: 6.0
Process: P9 Runtime before: 8.0
Process: P9 Runtime after: 6.0
Process: P3 Runtime before: 9.0
Process: P3 Runtime after: 7.0
Process: P7 Runtime before: 9.0
Process: P7 Runtime after: 7.0
Process: P6 Runtime before: 7.0
Process: P6 Runtime after: 5.0
Process: P0 Runtime before: 6.0
Process: P0 Runtime after: 4.0
Process: P8 Runtime before: 10.0
Process: P8 Runtime after: 8.0
Process: P2 Runtime before: 8.0
Process: P2 Runtime after: 6.0
Process: P1 Runtime before: 10.0
Process: P1 Runtime after: 8.0
```

Εικόνα 4.6: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν

Οι διεργασίες P10 έως P19 μπαίνουν τη πρώτη φορά στο σύστημα με Vruntime ίσο με ένα και για αυτό τον λόγο τρέχουν πρώτες.

```
Name: P10| Vruntime: 72760.58| Nice: 16| Quantums: 28.82| Runtime: 11.0ms
Insert: P10 Vrurntime: 72760.58
R----P6(BLACK)
     L----P3(RED)
     |    L----P5(BLACK)
     |    |    L----P4(RED)
     |    |    R----P9(RED)
     |    R----P7(BLACK)
     R----P2(RED)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P11| Vruntime: 430.50| Nice: -7| Quantums: 4882.81| Runtime: 10.0ms
Insert: P11 Vrurntime: 430.50
R----P6(BLACK)
     L----P3(BLACK)
     |    L----P5(RED)
     |    |    L----P4(BLACK)
     |    |    |    R----P11(RED)
     |    |    R----P9(BLACK)
     |    R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P12| Vruntime: 12208.03| Nice: 8| Quantums: 171.80| Runtime: 8.0ms
Insert: P12 Vrurntime: 12208.03
R----P6(BLACK)
     L----P3(BLACK)
     |    L----P5(RED)
     |    |    L----P4(BLACK)
     |    |    |    R----P11(RED)
     |    |    R----P9(BLACK)
     |    |         L----P12(RED)
     |    R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P13| Vruntime: 275.88| Nice: -9| Quantums: 7629.39| Runtime: 10.0ms
Insert: P13 Vrurntime: 275.88
R----P6(BLACK)
     L----P3(BLACK)
     |    L----P5(RED)
     |    |    L----P13(BLACK)
     |    |    |    L----P4(RED)
     |    |    |    R----P11(RED)
     |    |    R----P9(BLACK)
     |    |         L----P12(RED)
     |    R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)
```

Εικόνα 4.7α: Εισαγωγή των διεργασιών P10 έως P13 στο Red Black Tree

```
Name: P14| Vruntime: 176.92| Nice: -11| Quantums: 11920.93| Runtime: 12.0ms
Insert: P14 Vrurntime: 176.92
R----P6(BLACK)
     L----P5(BLACK)
     |    L----P13(RED)
     |    |    L----P4(BLACK)
     |    |    |    R----P14(RED)
     |    |    R----P11(BLACK)
     |    R----P3(RED)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P15| Vruntime: 30.51| Nice: -19| Quantums: 71054.27| Runtime: 10.0ms
Insert: P15 Vrurntime: 30.51
R----P6(BLACK)
     L----P5(BLACK)
     |    L----P13(RED)
     |    |    L----P4(BLACK)
     |    |    |    L----P15(RED)
     |    |    |    R----P14(RED)
     |    |    R----P11(BLACK)
     |    R----P3(RED)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P16| Vruntime: 73.06| Nice: -15| Quantums: 29103.83| Runtime: 11.0ms
Insert: P16 Vrurntime: 73.06
R----P6(BLACK)
     L----P5(RED)
     |    L----P13(BLACK)
     |    |    L----P4(RED)
     |    |    |    L----P15(BLACK)
     |    |    |    |    R----P16(RED)
     |    |    |    R----P14(BLACK)
     |    |    R----P11(BLACK)
     |    R----P3(BLACK)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)
```

Εικόνα 4.7β: Εισαγωγή των διεργασιών P14 έως P16 στο Red Black Tree

```
Name: P17| Vruntime: 23842.86| Nice: 11| Quantums: 87.96| Runtime: 9.0ms
Insert: P17 Vrurntime: 23842.86
R----P6(BLACK)
     L----P5(RED)
     |    L----P13(BLACK)
     |    |    L----P4(RED)
     |    |    |    L----P15(BLACK)
     |    |    |    |    R----P16(RED)
     |    |    |    R----P14(BLACK)
     |    |    R----P11(BLACK)
     |    R----P3(BLACK)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         |    R----P17(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Name: P18| Vruntime: 72760.58| Nice: 16| Quantums: 28.82| Runtime: 10.0ms
Insert: P18 Vrurntime: 72760.58
R----P6(BLACK)
     L----P5(RED)
     |    L----P13(BLACK)
     |    |    L----P4(RED)
     |    |    |    L----P15(BLACK)
     |    |    |    |    R----P16(RED)
     |    |    |    R----P14(BLACK)
     |    |    R----P11(BLACK)
     |    R----P3(BLACK)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         |    R----P17(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(RED)
          |    L----P10(BLACK)
          |    |    R----P18(RED)
          |    R----P8(BLACK)
          R----P1(BLACK)
```

Εικόνα 4.7γ: Εισαγωγή των διεργασιών P17 και P18 στο Red Black Tree

```
Name: P19| Vruntime: 91.07| Nice: -14| Quantums: 23283.06| Runtime: 9.0ms
Insert: P19 Vrurntime: 91.07
R----P6(BLACK)
     L----P5(RED)
     |    L----P13(BLACK)
     |    |    L----P4(RED)
     |    |    |    L----P16(BLACK)
     |    |    |    |    L----P15(RED)
     |    |    |    |    R----P19(RED)
     |    |    |    R----P14(BLACK)
     |    |    R----P11(BLACK)
     |    R----P3(BLACK)
     |         L----P9(BLACK)
     |         |    L----P12(RED)
     |         |    R----P17(RED)
     |         R----P7(BLACK)
     R----P2(BLACK)
          L----P0(RED)
          |    L----P10(BLACK)
          |    |    R----P18(RED)
          |    R----P8(BLACK)
          R----P1(BLACK)

The number of total quantums is: 178653.68
Proccess P0 is using 0.03% which is: 0.03 of the TL
Proccess P1 is using 0.01% which is: 0.01 of the TL
Proccess P2 is using 0.01% which is: 0.01 of the TL
Proccess P3 is using 0.10% which is: 0.08 of the TL
Proccess P4 is using 16.29% which is: 13.03 of the TL
Proccess P5 is using 0.37% which is: 0.29 of the TL
Proccess P6 is using 0.05% which is: 0.04 of the TL
Proccess P7 is using 0.06% which is: 0.05 of the TL
Proccess P8 is using 0.02% which is: 0.01 of the TL
Proccess P9 is using 0.12% which is: 0.10 of the TL
Proccess P10 is using 0.02% which is: 0.01 of the TL
Proccess P11 is using 2.73% which is: 2.19 of the TL
Proccess P12 is using 0.10% which is: 0.08 of the TL
Proccess P13 is using 4.27% which is: 3.42 of the TL
Proccess P14 is using 6.67% which is: 5.34 of the TL
Proccess P15 is using 39.77% which is: 31.82 of the TL
Proccess P16 is using 16.29% which is: 13.03 of the TL
Proccess P17 is using 0.05% which is: 0.04 of the TL
Proccess P18 is using 0.02% which is: 0.01 of the TL
Proccess P19 is using 13.03% which is: 10.43 of the TL
```

Εικόνα 4.7δ: Εισαγωγή της διεργασίας P19 στο Red Black Tree, Πίνακας χρήσης TL

Αφού το δέντρο έρθει στην τελική του μορφή, παρουσιάζεται ο νέος πίνακας χρήσης του TL και τα συνολικά κβάντα που έχουν όλες οι διεργασίες μαζί.

Ο χρήστης στη συνέχεια αποφασίζει να μη προσθέσει άλλες διεργασίες στο σύστημα, οπότε οι ήδη υπάρχουσες συνεχίζουν να τρέχουν.

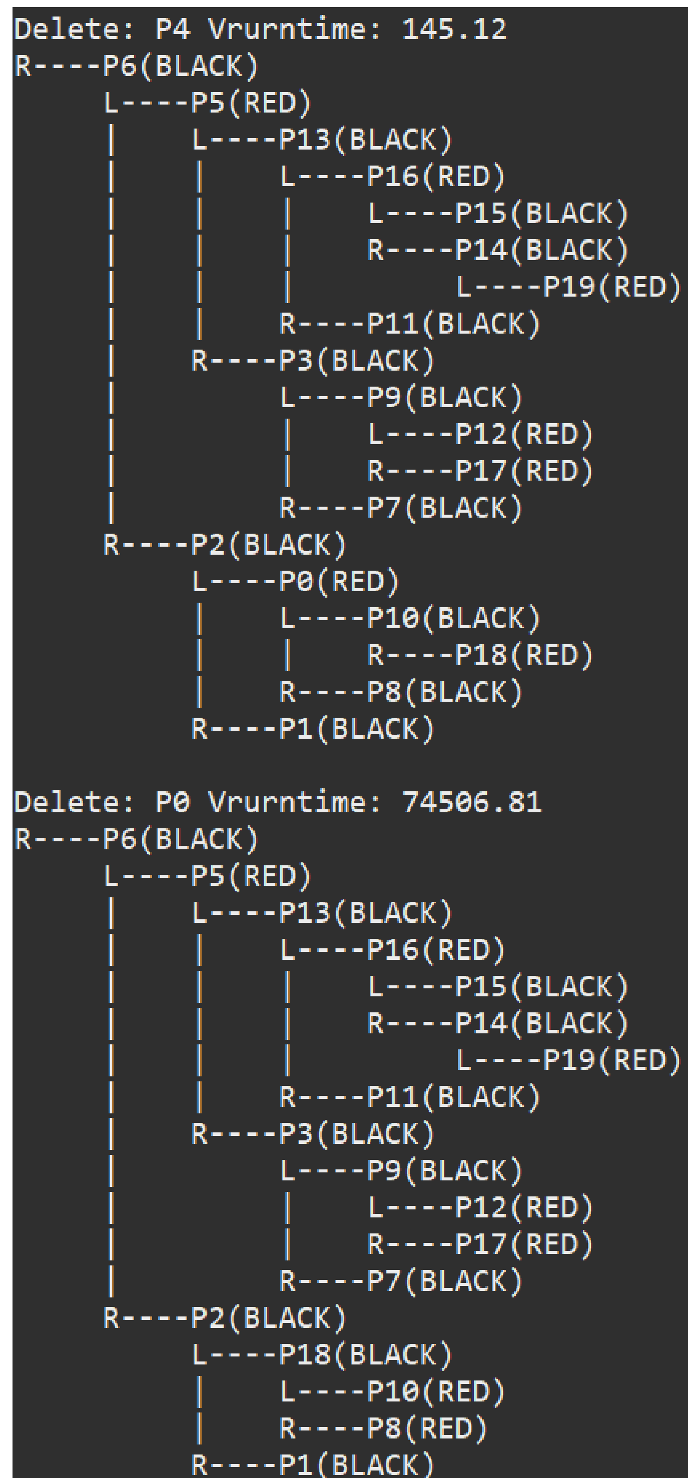
```
Would you like to add more processes to the system? (1 : YES / 0 : NO)
0
*** Processes Running ***
Process: P15 Runtime before: 10.0
Process: P15 Runtime after: 8.0
Process: P16 Runtime before: 11.0
Process: P16 Runtime after: 9.0
Process: P19 Runtime before: 9.0
Process: P19 Runtime after: 7.0
Process: P4 Runtime before: 4.0
Process: P4 Runtime after: 2.0
Process: P14 Runtime before: 12.0
Process: P14 Runtime after: 10.0
Process: P13 Runtime before: 10.0
Process: P13 Runtime after: 8.0
Process: P11 Runtime before: 10.0
Process: P11 Runtime after: 8.0
Process: P5 Runtime before: 6.0
Process: P5 Runtime after: 4.0
Process: P12 Runtime before: 8.0
Process: P12 Runtime after: 6.0
Process: P9 Runtime before: 6.0
Process: P9 Runtime after: 4.0
Process: P17 Runtime before: 9.0
Process: P17 Runtime after: 7.0
Process: P3 Runtime before: 7.0
Process: P3 Runtime after: 5.0
Process: P7 Runtime before: 7.0
Process: P7 Runtime after: 5.0
Process: P6 Runtime before: 5.0
Process: P6 Runtime after: 3.0
Process: P10 Runtime before: 11.0
Process: P10 Runtime after: 9.0
Process: P18 Runtime before: 10.0
Process: P18 Runtime after: 8.0
Process: P0 Runtime before: 4.0
Process: P0 Runtime after: 2.0
Process: P8 Runtime before: 8.0
Process: P8 Runtime after: 6.0
Process: P2 Runtime before: 6.0
Process: P2 Runtime after: 4.0
Process: P1 Runtime before: 8.0
Process: P1 Runtime after: 6.0
```

Εικόνα 4.8: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν

```
The number of total quantums is: 178653.68
*** Processes Running ***
Process: P15 Runtime before: 8.0
Process: P15 Runtime after: 6.0
Process: P16 Runtime before: 9.0
Process: P16 Runtime after: 7.0
Process: P19 Runtime before: 7.0
Process: P19 Runtime after: 5.0
Process: P4 Runtime before: 2.0
Process: P4 Runtime after: 0.0
Process: P14 Runtime before: 10.0
Process: P14 Runtime after: 8.0
Process: P13 Runtime before: 8.0
Process: P13 Runtime after: 6.0
Process: P11 Runtime before: 8.0
Process: P11 Runtime after: 6.0
Process: P5 Runtime before: 4.0
Process: P5 Runtime after: 2.0
Process: P12 Runtime before: 6.0
Process: P12 Runtime after: 4.0
Process: P9 Runtime before: 4.0
Process: P9 Runtime after: 2.0
Process: P17 Runtime before: 7.0
Process: P17 Runtime after: 5.0
Process: P3 Runtime before: 5.0
Process: P3 Runtime after: 3.0
Process: P7 Runtime before: 5.0
Process: P7 Runtime after: 3.0
Process: P6 Runtime before: 3.0
Process: P6 Runtime after: 1.0
Process: P10 Runtime before: 9.0
Process: P10 Runtime after: 7.0
Process: P18 Runtime before: 8.0
Process: P18 Runtime after: 6.0
Process: P0 Runtime before: 2.0
Process: P0 Runtime after: 0.0
Process: P8 Runtime before: 6.0
Process: P8 Runtime after: 4.0
Process: P2 Runtime before: 4.0
Process: P2 Runtime after: 2.0
Process: P1 Runtime before: 6.0
Process: P1 Runtime after: 4.0
```

Εικόνα 4.9: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν

Οι διεργασίες P4 και P0, ολοκλήρωσαν τον χρόνο τους και αφαιρούνται από το δέντρο. Πραγματοποιείται επαναδόμηση του, για να υπάρχει ισορροπία.



Εικόνα 4.10: Η διαγραφή των διεργασιών P4 και P0

```
The number of total quantums is: 178653.68
*** Processes Running ***
Process: P15 Runtime before: 6.0
Process: P15 Runtime after: 2.0
Process: P16 Runtime before: 7.0
Process: P16 Runtime after: 3.0
Process: P19 Runtime before: 5.0
Process: P19 Runtime after: 1.0
Process: P14 Runtime before: 8.0
Process: P14 Runtime after: 4.0
Process: P13 Runtime before: 6.0
Process: P13 Runtime after: 2.0
Process: P11 Runtime before: 6.0
Process: P11 Runtime after: 2.0
Process: P5 Runtime before: 2.0
Process: P5 Runtime after: 0.0
Process: P12 Runtime before: 4.0
Process: P12 Runtime after: 0.0
Process: P9 Runtime before: 2.0
Process: P9 Runtime after: 0.0
Process: P17 Runtime before: 5.0
Process: P17 Runtime after: 1.0
Process: P3 Runtime before: 3.0
Process: P3 Runtime after: 0.0
Process: P7 Runtime before: 3.0
Process: P7 Runtime after: 0.0
Process: P6 Runtime before: 1.0
Process: P6 Runtime after: 0.0
Process: P10 Runtime before: 7.0
Process: P10 Runtime after: 3.0
Process: P18 Runtime before: 6.0
Process: P18 Runtime after: 2.0
Process: P8 Runtime before: 4.0
Process: P8 Runtime after: 0.0
Process: P2 Runtime before: 2.0
Process: P2 Runtime after: 0.0
Process: P1 Runtime before: 4.0
Process: P1 Runtime after: 0.0
```

Εικόνα 4.11: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν

Οι διεργασίες P5, P12, P9, P3, P7, P6, P8, P2 και P1, ολοκλήρωσαν τον χρόνο εκτέλεσης τους, αποχωρούν από το σύστημα και διαγράφονται από το δέντρο.

```
Delete: P5 Vrurntime: 6401.00
R----P7(BLACK)
     L----P11(BLACK)
     |    L----P14(RED)
     |    |    L----P16(BLACK)
     |    |    |    L----P15(RED)
     |    |    |    R----P19(RED)
     |    |    R----P13(BLACK)
     |    R----P9(RED)
     |         L----P12(BLACK)
     |         R----P3(BLACK)
     |              L----P17(RED)
     R----P2(BLACK)
          L----P10(RED)
          |    L----P6(BLACK)
          |    R----P8(BLACK)
          |         L----P18(RED)
          R----P1(BLACK)

Delete: P12 Vrurntime: 12208.03
R----P7(BLACK)
     L----P13(BLACK)
     |    L----P16(RED)
     |    |    L----P15(BLACK)
     |    |    R----P14(BLACK)
     |    |         L----P19(RED)
     |    R----P9(RED)
     |         L----P11(BLACK)
     |         R----P3(BLACK)
     |              L----P17(RED)
     R----P2(BLACK)
          L----P10(RED)
          |    L----P6(BLACK)
          |    R----P8(BLACK)
          |         L----P18(RED)
          R----P1(BLACK)

Delete: P9 Vrurntime: 19532.25
R----P7(BLACK)
     L----P11(BLACK)
     |    L----P14(RED)
     |    |    L----P16(BLACK)
     |    |    |    L----P15(RED)
     |    |    |    R----P19(RED)
     |    |    R----P13(BLACK)
     |    R----P3(BLACK)
     |         L----P17(RED)
     R----P2(BLACK)
          L----P10(RED)
          |    L----P6(BLACK)
          |    R----P8(BLACK)
          |         L----P18(RED)
          R----P1(BLACK)
```

```
Delete: P3 Vrurntime: 24415.06
R----P6(BLACK)
     L----P11(BLACK)
     |    L----P14(RED)
     |    |    L----P16(BLACK)
     |    |    |    L----P15(RED)
     |    |    |    R----P19(RED)
     |    |    R----P13(BLACK)
     |    R----P7(BLACK)
     |         L----P17(RED)
     R----P2(BLACK)
          L----P18(BLACK)
          |    L----P10(RED)
          |    R----P8(RED)
          R----P1(BLACK)

Delete: P7 Vrurntime: 38147.97
R----P10(BLACK)
     L----P11(BLACK)
     |    L----P14(RED)
     |    |    L----P16(BLACK)
     |    |    |    L----P15(RED)
     |    |    |    R----P19(RED)
     |    |    R----P13(BLACK)
     |    R----P6(BLACK)
     |         L----P17(RED)
     R----P2(BLACK)
          L----P8(BLACK)
          |    L----P18(RED)
          R----P1(BLACK)

Delete: P6 Vrurntime: 47684.72
R----P10(BLACK)
     L----P13(BLACK)
     |    L----P16(RED)
     |    |    L----P15(BLACK)
     |    |    R----P14(BLACK)
     |    |         L----P19(RED)
     |    R----P11(BLACK)
     |         R----P17(RED)
     R----P2(BLACK)
          L----P8(BLACK)
          |    L----P18(RED)
          R----P1(BLACK)
```

Εικόνες 4.12α, 4.12β: Διαγραφή των διεργασιών P5, P12, P9, P3, P7, P6

```
Delete: P8 Vrurntime: 145520.15
R----P11(BLACK)
     L----P14(RED)
     |    L----P16(BLACK)
     |    |    L----P15(RED)
     |    |    R----P19(RED)
     |    R----P13(BLACK)
     R----P2(RED)
          L----P10(BLACK)
          |    L----P17(RED)
          |    R----P18(RED)
          R----P1(BLACK)

Delete: P2 Vrurntime: 227374.68
R----P13(BLACK)
     L----P16(RED)
     |    L----P15(BLACK)
     |    R----P14(BLACK)
     |         L----P19(RED)
     R----P10(RED)
          L----P11(BLACK)
          |    R----P17(RED)
          R----P1(BLACK)
               L----P18(RED)

Delete: P1 Vrurntime: 284218.09
R----P11(BLACK)
     L----P14(RED)
     |    L----P16(BLACK)
     |    |    L----P15(RED)
     |    |    R----P19(RED)
     |    R----P13(BLACK)
     R----P10(BLACK)
          L----P17(RED)
          R----P18(RED)
```

Εικόνα 4.12γ: Διαγραφή των διεργασιών P8, P2 και P1

Οι τελευταίες εννέα διεργασίες τρέχουν για τελευταία φορά και ολοκληρώνουν τον χρόνο εκτέλεσης τους.

```
The number of total quantums is: 178653.68
*** Processes Running ***
Process: P15 Runtime before: 2.0
Process: P15 Runtime after: 0.0
Process: P16 Runtime before: 3.0
Process: P16 Runtime after: 0.0
Process: P19 Runtime before: 1.0
Process: P19 Runtime after: 0.0
Process: P14 Runtime before: 4.0
Process: P14 Runtime after: 0.0
Process: P13 Runtime before: 2.0
Process: P13 Runtime after: 0.0
Process: P11 Runtime before: 2.0
Process: P11 Runtime after: 0.0
Process: P17 Runtime before: 1.0
Process: P17 Runtime after: 0.0
Process: P10 Runtime before: 3.0
Process: P10 Runtime after: 0.0
Process: P18 Runtime before: 2.0
Process: P18 Runtime after: 0.0
```

Εικόνα 4.13: Οι χρόνοι των διεργασιών, πριν και αφού τρέξουν

```
Delete: P15 Vrurntime: 30.51
R----P11(BLACK)
     L----P14(RED)
     |    L----P16(BLACK)
     |    |    R----P19(RED)
     |    R----P13(BLACK)
     R----P10(BLACK)
          L----P17(RED)
          R----P18(RED)

Delete: P16 Vrurntime: 73.06
R----P11(BLACK)
     L----P14(BLACK)
     |    L----P19(RED)
     |    R----P13(RED)
     R----P10(BLACK)
          L----P17(RED)
          R----P18(RED)

Delete: P19 Vrurntime: 91.07
R----P11(BLACK)
     L----P13(BLACK)
     |    L----P14(RED)
     R----P10(BLACK)
          L----P17(RED)
          R----P18(RED)

Delete: P14 Vrurntime: 176.92
R----P11(BLACK)
     L----P13(BLACK)
     R----P10(BLACK)
          L----P17(RED)
          R----P18(RED)
```

Εικόνα 4.14: Διαγραφή των διεργασιών P15, P16, P19 και P14

```
Delete: P13 Vrurntime: 275.88
R----P17(BLACK)
     L----P11(BLACK)
     R----P10(BLACK)
          R----P18(RED)

Delete: P11 Vrurntime: 430.50
R----P10(BLACK)
     L----P17(RED)
     R----P18(RED)

Delete: P17 Vrurntime: 23842.86
R----P10(BLACK)
     R----P18(RED)

Delete: P10 Vrurntime: 72760.58
R----P18(BLACK)

Delete: P18 Vrurntime: 72760.58

The number of total quantums is: 178653.68
***** Simulation Finished *****
```

Εικόνα 4.15: Διαγραφή των διεργασιών P13, P11, P17, P10 και P18 και τερματισμός της προσομοίωσης

Όλες οι διεργασίες ολοκληρώθηκαν και διαγράφηκαν από το δέντρο. Εφόσον δεν υπάρχουν άλλες διεργασίες που να έχουν υπολειπόμενο χρόνο εκτέλεσης και ο χρήστης πλέον δεν μπορεί να προσθέσει άλλες, το Red Black Tree είναι κενό, που σημαίνει ότι η προσομοίωση ολοκληρώθηκε με επιτυχία.

Κεφάλαιο 5ο : Τελικά Συμπεράσματα της Έρευνας

Όπως αναφέρθηκε ήδη στο δεύτερο Κεφάλαιο, ο CFS έχει ως κύριο σκοπό να μοιράζει ισάξια την Κεντρική Μονάδα Επεξεργασίας σε όλες τις διεργασίες ώστε να μπορούν να εκτελούνται σχεδόν ταυτοχρόνως. Η κατανομή γίνεται με βάση το vruntime κάθε διεργασίας και έτσι κατασκευάζεται και το Red Black Tree. Στο τρίτο Κεφάλαιο αναλύθηκαν οι πέντε κλάσεις του κώδικα καθώς και οι συναρτήσεις που τις δομούν και κάνουν εφικτή την διαδικασία της προσομοίωσης. Τέλος, στο τέταρτο Κεφάλαιο, παρουσιάζεται μία πλήρης προσομοίωση του Χρονοδρομολογητή CFS. Εισάγονται δέκα διεργασίες, η κάθε μία με τα δικά της χαρακτηριστικά, δομούν το πρώτο Ερυθρόμαυρο Δέντρο και ξεκινούν τη διαδικασία εκτέλεσης. Στη συνέχεια προστίθενται ακόμη δέκα διεργασίες, κάνουν χρήση της ΚΜΕ, καθώς ξεκινούν με vruntime ίσο με ένα, δημιουργούνται τα χαρακτηριστικά τους και εισέρχονται και αυτές στο δέντρο. Από εκεί και πέρα δεν εισέρχονται άλλες διεργασίες και οι ήδη υπάρχουσες εκτελούνται μέχρι να μην μείνει καμία στο δέντρο. Όλες οι διεργασίες έτρεξαν τον χρόνο που τους είχε δοθεί, διαγράφηκαν και το δέντρο είναι πλέον άδειο. Έτσι, σηματοδοτείτε η πλήρης ολοκλήρωση της προσομοίωσης, η οποία είναι επιτυχής.

Βιβλιογραφία

[1] Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. *Operating System Concepts, 10e Abridged Print Companion*. John Wiley & Sons, 2018

[2] Pabla, Chandandeep Singh. "Completely fair scheduler." *Linux Journal* 2009.184 (2009): 4

[3] Pabla, C. S. (2009). Completely fair scheduler. *Linux Journal*, *2009*(184), 4

[4] PABLA, Chandandeep Singh. Completely fair scheduler. *Linux Journal*, 2009, 2009.184: 4.

[5] Jamale, R. S., Dhotre, S., & Patil, P. T. (2016). A Survey on Response Time Analysis Using Linux Kernel Completely Fair Scheduler for Data Intensive Tasks. In *5 th International Conference on Communications, Electrical, Electronics and Computer Engineering (ICEEC 2017) Paper ID: ICEEC802017*.

[6] Patil, Pooja Tanaji, and P. S. Dhotre. "Response Time Analysis Using Linux Completely Fair Scheduler for Compute-Intensive Tasks." *vol* 5: 377-380.

[7] Patil, P. T., & Dhotre, P. S. Response Time Analysis Using Linux Completely Fair Scheduler for Compute-Intensive Tasks. *vol*, *5*, 377-380.

[8] PATIL, Pooja Tanaji; DHOTRE, P. S. Response Time Analysis Using Linux Completely Fair Scheduler for Compute-Intensive Tasks. *vol*, 5: 377-380.

[9] Patila, Pooja Tanaji, Sunita Dhotreb, and Rucha Shankar Jamalec. "A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel."

[10] Patila, P. T., Dhotreb, S., & Jamalec, R. S. A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel.

[11] PATILA, Pooja Tanaji; DHOTREB, Sunita; JAMALEC, Rucha Shankar. A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel.

[12] https://www.baeldung.com/cs/red-black-trees.

[13] https://iq.opengenus.org/time-and-space-complexity-of-red-black-tree/

[14] https://www.codesdope.com/course/data-structures-red-black-trees-insertion/