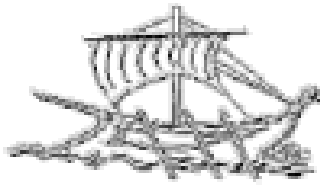


ΤΕΙ ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. "ΕΦΑΡΜΟΣΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Χρονοπρογραμματισμού σε Μεταγωγείς Ευρυζωνικών Δικτύων

Υλοποίηση και Προσομοίωση σε Γραφικό Περιβάλλον

Παναγιώτης Η. Καφόπουλος

Εισηγητής: Δρ. Νικόλαος Δ. Τσελίκας, Επίκουρος Καθηγητής

ΠΕΙΡΑΙΑΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Χρονοπρογραμματισμού σε Μεταγωγείς Ευρυζωνικών Δικτύων Υλοποίηση και Προσομοίωση σε Γραφικό Περιβάλλον

Παναγιώτης Η. Καφόπουλος
Α.Μ. ais-0123

Εισηγητής:

Δρ. Νικόλαος Δ. Τσελίκας, Επίκουρος Καθηγητής

Εξεταστική Επιτροπή:

Ημερομηνία εξέτασης:

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος **Καφόπουλος Παναγιώτης** του **Ηλία**, με αριθμό μητρώου **AIS-0123**, φοιτητής του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ., πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία εκπονήθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό των ευρυζωνικών δικτύων. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, Δρ. Νικόλαος Τσελίκας, τον οποίο θα ήθελα να ευχαριστήσω.

Ακόμα θα ήθελα να ευχαριστήσω τους συμφοιτητές μου για τη βοήθεια και τη συντροφιά τους στο κουραστικό αλλά όμορφο αυτό εκπαιδευτικό ταξίδι, καθώς επίσης την οικογένεια και τους φίλους μου για την αμέριστη στήριξη και υπομονή τους.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με τους αλγορίθμους χρονοπρογραμματισμού στα ευρυζωνικά δίκτυα, την υλοποίησή τους σε μια γλώσσα προγραμματισμού και την προσομοίωσή τους σε γραφικό περιβάλλον. Αλγόριθμοι χρονοπρογραμματισμού είναι οι τεχνικές με τις οποίες γίνεται η μετάβαση των πακέτων δεδομένων από τις εισόδους προς τις εξόδους ενός μεταγωγέα ευρυζωνικών δικτύων, εξαλείφοντας τους κινδύνους των συγκρούσεων και των άσκοπων καθυστερήσεων. Ο σκοπός της εργασίας είναι κυρίως εκπαιδευτικός και στόχο έχει την κατανόηση των αλγορίθμων αυτών από τους σπουδαστές και την εξοικείωσή τους με τη λειτουργία τους.

Κατόπιν μιας σύντομης αναφοράς στο ιστορικό υπόβαθρο των μεταγωγέων ευρυζωνικών δικτύων, δίνεται βάση σε θέματα επίδοσής τους. Ένας από τους σημαντικότερους τρόπους βελτίωσης της επίδοσης των μεταγωγέων είναι οι μηχανισμοί δίκαιης και σωστής διαιτησίας των πακέτων δεδομένων, αντικείμενο με το οποίο ασχολούνται οι αλγόριθμοι χρονοπρογραμματισμού. Αναφέρονται και αναλύονται οι πέντε αλγόριθμοι (PM, PIM, iRRM, iSLIP και DRRM) και εξηγείται η λειτουργία τους. Στη συνέχεια παρατίθεται και επεξηγείται διεξοδικά ο κώδικας υλοποίησής τους σε γλώσσα JAVA και η εργασία ολοκληρώνεται με την απεικόνιση ενός πλήρους παραδείγματος εκτέλεσης της εφαρμογής.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Ευρυζωνικά Δίκτυα
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: χρονοπρογραμματισμός, PM, PIM, iRRM, iSLIP, DRRM

ABSTRACT

This diploma thesis deals with the scheduling algorithms in broadband networks, their implementation in a programming language and their simulation in a graphical environment. Scheduling algorithms are the techniques by which data packets are moved from the inputs to the outputs of a broadband switch, eliminating the risks of collisions and unnecessary delays. The purpose of this work is mainly educational and aims to understanding of these algorithms by the students and familiarizing them with their function.

After a brief reference to the historical background of broadband network switches, it focuses on their performance issues. One of the most important ways to improve the performance of switches is the mechanisms of fair and proper arbitration of data packets, which is the subject of scheduling algorithms. The five algorithms (PM, PIM, iRRM, iSLIP and DRRM) are reported and analyzed and their operation is explained. The JAVA implementation code is then explained in detail, and this thesis is completed by depicting a full implementation example of the application.

SCIENTIFIC AREA: Broadbands Networks
KEYWORDS: scheduling, PM, PIM, iRRM, iSLIP, DRRM

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

HOL Head Of Line

FIFO First In First Out

VOQ Virtual Output Queue

PM Parallel Matching

PIM Parallel Iterative Matching

iRRM Iterative Round-Robin Matching

DRRM Dual Round-Robin Matching

IDE Intergrated Development Environment

1. Γενικά

Αν θέλαμε να δώσουμε έναν γενικό ορισμό στην έννοια του μεταγωγέα (switch), θα λέγαμε ότι είναι η ηλεκτρονική συσκευή που κάνει μεταγωγή (switching) ενός σήματος από ένα σημείο σε ένα άλλο, ανασχηματίζει και αναδιαρθρώνει, δηλαδή, τις πολυπλεγμένες ροές εισόδου (input streams) σε αντίστοιχες ροές εξόδου (output streams). Με άλλα λόγια, η πληροφορία μιας συγκεκριμένης φυσικής σύνδεσης που κατέχει συγκεκριμένη «πολυπλεγμένη θέση», αναλαμβάνεται πλέον από μια άλλη φυσική σύνδεση (κατοχή νέας «πολυπλεγμένης θέσης»).

Οι πρώτοι μεταγωγείς υψηλών ταχυτήτων που κατασκευάστηκαν για χρήση στα ευρυζωνικά δίκτυα, χρησιμοποιούσαν είτε εσωτερικούς κοινόχρηστους ενταμιευτές (buffers), είτε ενταμιευτές στις μονάδες εισόδου, με αποτέλεσμα να υπάρχει περιορισμός στο ρυθμό διέλευσης πακέτων (throughput). Επειδή η αρχική ζήτηση για χωρητικότητα στους μεταγωγείς κυμαινόταν από μερικά μέχρι και δεκάδες Gbits/s, οι μεταγωγείς με ενταμίευση στις μονάδες εξόδου φάνηκε να αποτελούν την καλύτερη λύση, λόγω της υψηλής επίδοσής τους ως προς τη σχέση καθυστέρησης - ρυθμού διέλευσης πακέτων και τη χρήση μνήμης (για μεταγωγείς διαμοιραζόμενης μνήμης). Έτσι, στα πρώτα χρόνια της χρήσης των μεταγωγέων πακέτου, κυριάρχησαν οι μεταγωγείς με ενταμίευση στις μονάδες εξόδου, συμπεριλαμβανομένων και αυτών με διαμοιραζόμενη μνήμη.

Όμως, η ραγδαία ζήτηση για μεγαλύτερη χωρητικότητα στους μεταγωγείς, απαιτούσε αντίστοιχα μεγαλύτερη ταχύτητα στις μνήμες. Το γεγονός αυτό είχε ως αποτέλεσμα τον περιορισμό στη χωρητικότητα των μεταγωγέων με ενταμίευση στις μονάδες εξόδου. Επομένως, με σκοπό να κατασκευαστούν μεγάλης κλίμακας μεταγωγείς υψηλών ταχυτήτων, η έρευνα επικεντρώθηκε σε αρχιτεκτονικές μεταγωγέων με ενταμίευση στις μονάδες εισόδου ή σε αρχιτεκτονικές που συνδύαζαν και τις δύο τεχνολογίες. Οι μεταγωγείς που διαθέτουν ενταμίευση τόσο στις μονάδες εξόδου όσο και στις μονάδες εισόδου, σε συνδυασμό με μηχανισμούς προγραμματισμού εκπομπής πακέτου και τεχνικές δρομολόγησης, αποτελούν το αντικείμενο του παρόντος κεφαλαίου.

Οι μεταγωγείς με ενταμίευση στις μονάδες εισόδου παρουσιάζουν δύο κύρια προβλήματα:

α) περιορισμό στη διέλευση εξαιτίας του γεγονότος της φραγής από το πακέτο - επικεφαλής σειράς (head-of-line, HOL blocking)

β) ανάγκη για δίκαιη αντιμετώπιση των πακέτων, ανεξάρτητα από τη θύρα εισόδου που προήλθαν.

Το πρώτο πρόβλημα μπορεί να αντιμετωπιστεί με βαθμιαία αύξηση της ταχύτητας επεξεργασίας στη Μονάδα Μεταγωγής πακέτου ή με αύξηση του αριθμού των διαδρομών δρομολόγησης που αντιστοιχούν σε κάθε μονάδα εξόδου. Η αύξηση του αριθμού των διαδρομών δρομολόγησης που αντιστοιχούν σε κάθε μονάδα εξόδου, επιτρέπει σε πολλαπλά πακέτα να φτάσουν στην ίδια μονάδα εξόδου κατά τη διάρκεια της ίδιας χρονικής θυρίδας. Σύμφωνα με το νόμο του Moore, η πυκνότητα των μνημών διπλασιάζεται κάθε 18 μήνες. Δεν ισχύει όμως το ίδιο για την ταχύτητά τους, η οποία αυξάνει με πολύ πιο αργό ρυθμό. Έτσι η ταχύτητα της CMOS στατικής RAM μνήμης ήταν περίπου 15 nsec στις αρχές της δεκαετίας του ‘90 και τριπλασιάστηκε μετά από περίπου 10 χρόνια (5 nsec, στις αρχές του 2001). Επίσης, η ταχύτητα των λογικών κυκλωμάτων αυξάνει με μεγαλύτερο ρυθμό από αυτό των μνημών. Ένας εναλλακτικός τρόπος λύσης και των δύο ανωτέρω προβλημάτων, είναι η χρήση αλγορίθμων χρονοπρογραμματισμού που περιγράφονται στην παράγραφο 3.3. Τα τελευταία χρόνια έχει γίνει προσπάθεια προς τη δημιουργία σχημάτων χρονοπρογραμματισμού με σκοπό τη διαιτησία εκπομπής των πακέτων από τις θύρες εισόδου, στις θύρες εξόδου.

Οι παράγοντες που χρησιμοποιούνται για τη σύγκριση των διαφόρων προτεινόμενων σχημάτων χρονοπρογραμματισμού είναι:

α) ο ρυθμός διέλευσης πακέτου,

β) η καθυστέρηση,

γ) η δίκαιη αντιμετώπιση όλων των πακέτων ανεξάρτητα από την θύρα εισόδου από την οποία προήλθαν,

δ) η πολυπλοκότητα υλοποίησης και

ε) η κλιμάκωση (scalability), καθώς αυξάνει το μέγεθος του μεταγωγέα και οι ρυθμοί μετάδοσης των γραμμών.

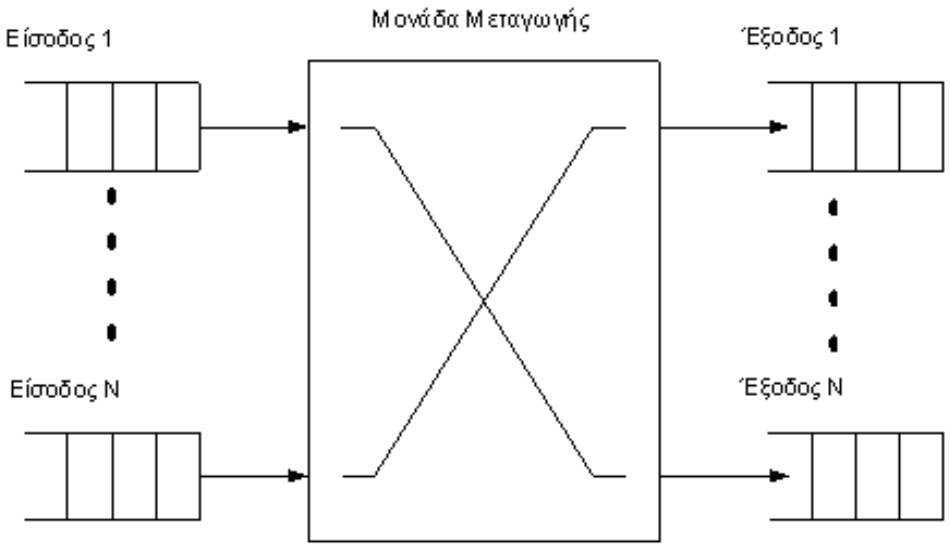
Επίσης, κάποια σχήματα χρονοπρογραμματισμού εφαρμόζουν προγραμματισμό (scheduling) στις θύρες εισόδου ανά ροή πακέτων (per-flow), με στόχο να ικανοποιήσουν τις απαιτήσεις για καθυστέρηση και ρυθμό διέλευσης της κάθε ροής. Σε τέτοια σχήματα, σοβαρό μειονέκτημα είναι ο σημαντικός βαθμός

πολυπλοκότητας και το υψηλό κόστος. Επίσης, ο χρονοπρογραμματισμός πακέτων ανά ροή στις θύρες εισόδου είναι αρκετά πιο δύσκολη διαδικασία σε σύγκριση με το χρονοπρογραμματισμό στις θύρες εξόδου. Για παράδειγμα, στα πακέτα σε μια θύρα εξόδου μπορεί να προστεθεί χρονική σήμανση (timestamped packets), με τιμές ανάλογα με το εύρος ζώνης που έχει κρατηθεί για αυτά και να μεταδοθούν με βάση την αύξουσα σειρά των τιμών αυτών. Όμως, σε μια θύρα εισόδου, για τη μετάδοση των πακέτων πρέπει να ληφθεί υπόψη και ο ανταγωνισμός των πακέτων για μετάδοση σε μια συγκεκριμένη θύρα εξόδου. Το γεγονός αυτό κάνει το πρόβλημα αρκετά πολύπλοκο.

2. Θέματα επίδοσης μεταγωγέων

Στο **Σχήμα 2.1** φαίνεται ένα απλό μοντέλο μεταγωγέα πακέτου αποτελούμενου από τρία κύρια λειτουργικά μέρη, τη μονάδα εισόδου, τη μονάδα εξόδου και τη Μονάδα Μεταγωγής Πακέτου που είναι υπεύθυνη για την μεταφορά των πακέτων από τις εισόδους στις εξόδους. Αρχικά θεωρούμε ότι η Μονάδα Μεταγωγής εσωτερικά δεν παρουσιάζει φραγή εσωτερικού διαύλου και ότι απαιτείται ένα συγκεκριμένο χρονικό διάστημα για την μεταφορά μιας ομάδας μη αλληλοσυγκρουόμενων (conflict-free) πακέτων στον προορισμό τους. Όταν μεγάλος αριθμός πακέτων κατευθύνονται στην ίδια θύρα εξόδου, δημιουργείται μια κατάσταση ανταγωνισμού μεταξύ τους για τη μεταφορά στην έξοδο αυτή.

Εξαιτίας του ανταγωνισμού αυτού, κάποια από τα πακέτα δε θα καταφέρουν να μεταφερθούν ταυτόχρονα και θα πρέπει να αποθηκευθούν σε ενταμιευτές στις εισόδους για να μεταφερθούν σε επόμενη χρονική θυρίδα. Θεωρούμε επίσης ότι κάθε γραμμή εισόδου και γραμμή εξόδου που συνδέει το μεταγωγέα με το υπόλοιπο δίκτυο έχει την ίδια ταχύτητα μετάδοσης, ενώ η Μονάδα Μεταγωγής μπορεί να έχει μεγαλύτερη ταχύτητα για να βελτιωθεί η απόδοση του μεταγωγέα. Στη τελευταία περίπτωση μπορεί να απαιτείται κάθε έξοδος να έχει έναν ενταμιευτή, αφού η ταχύτητα μετάδοσης της γραμμής εξόδου είναι μικρότερη από την ταχύτητα της Μονάδας Μεταγωγής.



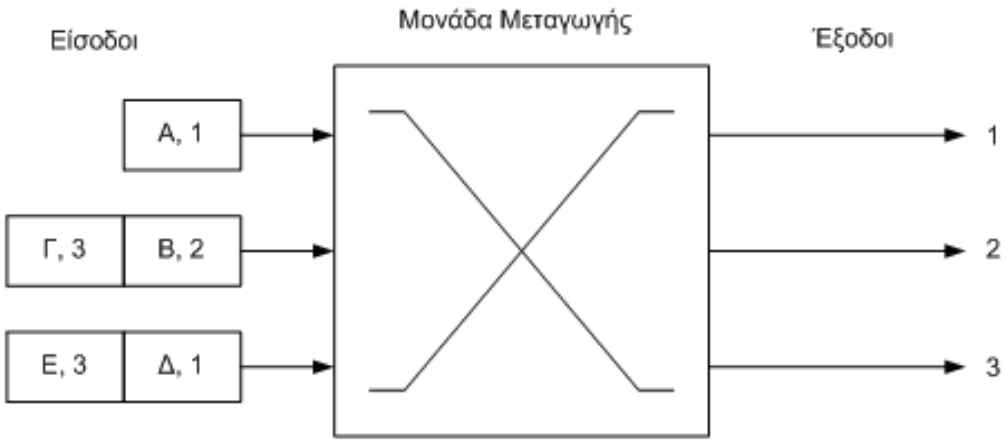
Σχήμα 2.1: Μοντέλο Μεταγωγέα Πακέτου

2.1 Φραγή του Επικεφαλής Σειράς

Ο Επικεφαλής Σειράς (HOL) είναι το πρώτο πακέτο στον ενταμιευτή που περιμένει να εξυπηρετηθεί. Ο πιο απλός τρόπος για την εξυπηρέτηση των πακέτων κάθε εισόδου είναι βάσει του σχήματος First In First Out (FIFO). Στο σχήμα αυτό, σε κάθε χρονική θυρίδα μπορεί να μεταδοθεί μόνο ο Επικεφαλής Σειράς σε κάθε θύρα εισόδου. Έτσι, όταν ένα πακέτο Επικεφαλής Σειράς δε μεταδοθεί λόγω του ανταγωνισμού με τα άλλα πακέτα Επικεφαλής Σειράς σε άλλες θύρες εισόδου, μπορεί να εμποδίσει τη μετάδοση επόμενων πακέτων στον ενταμιευτή, κάποια από τα οποία ενδεχομένως να κατευθύνονται σε μια άδεια θύρα εξόδου.

Για παράδειγμα στην περίπτωση του **Σχήματος 2.2**, τα πακέτα επικεφαλής σειράς Α και Δ είναι ανταγωνιστικά πακέτα γιατί κατευθύνονται στην ίδια έξοδο 1. Όμως, μόνο ένα από αυτά μπορεί να μεταδοθεί τη συγκεκριμένη χρονική θυρίδα. Αν υποθέσουμε ότι μεταδίδεται το πακέτο Α, τότε το πακέτο Δ πρέπει να περιμένει την επόμενη χρονική θυρίδα. Το πακέτο Ε, αν και κατευθύνεται σε μια άδεια έξοδο (έξοδο στην οποία δεν κατευθύνεται κανένα άλλο πακέτο, άρα δεν υπάρχει ανταγωνισμός), δεν μπορεί να μεταδοθεί, γιατί έχει εγκλωβιστεί από το πακέτο επικεφαλής σειράς Δ που βρίσκεται μπροστά του. Το φαινόμενο αυτό ονομάζεται Φραγή του Επικεφαλής Σειράς (HOL blocking). Σε μια τέτοια περίπτωση, το πρόβλημα μπορεί να βελτιωθεί, εάν αρθούν κάποιοι από τους περιορισμούς του σχήματος FIFO. Για παράδειγμα, το πακέτο Ε μπορεί να επιλεγεί να μεταφερθεί την πρώτη χρονική θυρίδα μαζί με το Α, αν και δεν είναι πακέτο Επικεφαλής Σειράς. Στην παράγραφο 3.3 περιγράφονται τα σημαντικότερα σχήματα χρονοπρογραμματισμού, για τη βελτίωση της ευφυΐας και κατ' επέκταση της ταχύτητας των μεταγωγέων, που είναι απαραίτητη σε μεταγωγείς δικτύων ευρείας ζώνης.

Στο **Σχήμα 2.2**, στη δεύτερη είσοδο, τα δύο πακέτα Β και Γ δεν μπορούν να μεταφερθούν την ίδια χρονική θυρίδα, αν και κατευθύνονται σε άδειες εξόδους. Κανένα σχήμα χρονοπρογραμματισμού δεν μπορεί να βελτιώσει την κατάσταση αυτή και ο μόνος τρόπος είναι με αύξηση του εύρους ζώνης εσωτερικά της Μονάδας Μεταγωγής Πακέτου.



Σχήμα 2.2: Φραγή Επικεφαλής Σειράς

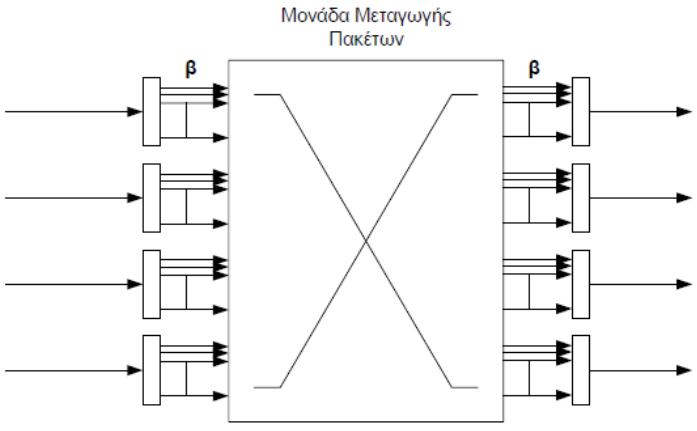
3. Μέθοδοι για βελτίωση της επίδοσης

Οι περιορισμοί στο ρυθμό διέλευσης ενός μεταγωγέα με ενταμιευτές στις εισόδους, προέρχονται κυρίως από τους περιορισμούς στο εύρος ζώνης και την αναποτελεσματικότητα του σχήματος χρονοπρογραμματισμού για τη μεταφορά των πακέτων από τις εισόδους στις εξόδους. Για να βελτιωθεί η διέλευση σε ένα μεταγωγέα με ενταμιευτές στις εισόδους, πρέπει είτε να αναπτυχθούν πιο αποτελεσματικά σχήματα χρονοπρογραμματισμού, είτε να αυξηθεί η εσωτερική χωρητικότητα του μεταγωγέα.

3.1 Αύξηση της εσωτερικής χωρητικότητας

3.1.1 Πολλαπλές Γραμμές ανά Είσοδο

Σε ένα μεταγωγέα NxN με πολλαπλές γραμμές ανά είσοδο (Multiline), δύο η περισσότερα πακέτα που βρίσκονται σε μια είσοδο, μπορούν να μεταφερθούν την ίδια χρονική θυρίδα στη Μονάδα Μεταγωγής Πακέτου. Στο **Σχήμα 3.1** φαίνεται ένας τέτοιος μεταγωγέας, με β γραμμές ανά είσοδο και ανά έξοδο και Μονάδα Μεταγωγής NβxNβ. Η Μονάδα Μεταγωγής του συστήματος μπορεί να δεχθεί μέχρι Nβ πακέτα την ίδια χρονική θυρίδα, β από τα οποία μπορούν να μεταφερθούν ταυτόχρονα στην ίδια έξοδο. Στην αρχιτεκτονική αυτή, μπορεί να παρουσιαστεί το πρόβλημα να φτάσουν σε μια μονάδα εξόδου πακέτα εκτός σειράς (out-of-sequence problem). Μια τέτοια αρχιτεκτονική μεταγωγέων δεν φαίνεται να έχει πρακτική αξία.



Σχήμα 3.1: Πολλαπλές Γραμμές ανά Είσοδο

3.1.2 Επιτάχυνση

Παράγοντας επιτάχυνσης με τιμή γ, σημαίνει ότι η Μονάδα Μεταγωγής λειτουργεί γ φορές ταχύτερα από τις μονάδες εισόδου και τις μονάδες εξόδου. Κάθε χρονική θυρίδα αποτελείται από γ κύκλους και κάθε πακέτο μεταφέρεται από τις μονάδες εισόδου στις μονάδες εξόδου στη διάρκεια κάθε κύκλου. Έτσι, κάθε είσοδος ή έξοδος μπορεί να μεταδώσει ή να δεχθεί αντίστοιχα γ πακέτα σε κάθε χρονική θυρίδα. Μελέτες βάσει εξομοιώσεων έδειξαν ότι ένας παράγοντας επιτάχυνσης 2 επιτυγχάνει 100% διέλευση.

Συχνά, ο όρος «επιτάχυνση» (speedup) χρησιμοποιείται για να δηλώσει ότι ενώ μόνο ένα πακέτο μπορεί να μεταφερθεί από μια είσοδο στη Μονάδα Μεταγωγής, κάθε έξοδος μπορεί να δεχθεί μέχρι γ πακέτα την ίδια χρονική θυρίδα. Στην περίπτωση εκρηκτικής κίνησης, ένας παράγοντας επιτάχυνσης 2 επιτυγχάνει διέλευση που κυμαίνεται από 82.8% μέχρι 88.5%, ανάλογα με το βαθμό εκρηκτικότητας της κίνησης.

3.1.3 Παράλληλος Μεταγωγέας

Ο Παράλληλος Μεταγωγέας αποτελείται από K ταυτόσημα επίπεδα μεταγωγής (switch planes). Κάθε επίπεδο μεταγωγής έχει τους δικούς του ενταμιευτές εισόδου, αλλά μοιράζεται τους ενταμιευτές εξόδου με τα άλλα επίπεδα μεταγωγής. Ένας Παράλληλος Μεταγωγέας με K=2 επίπεδα μεταγωγής, επιτυγχάνει τη μέγιστη τιμή διέλευσης 1.0. Αυτό συμβαίνει γιατί η μέγιστη τιμή της διέλευσης σε κάθε επίπεδο μεταγωγής είναι περίπου 0.586, για οποιαδήποτε τιμή του αριθμού εισόδων/εξόδων N. Επειδή κάθε μονάδα εισόδου μεταδίδει πακέτα σε διαφορετικά επίπεδα μεταγωγής, η σειρά των πακέτων δε διατηρείται στις μονάδες εξόδου. Για το λόγο αυτό, ο μεταγωγέας πρέπει να έχει μηχανισμό χρονικής σήμανσης (timestamp) των πακέτων στις εισόδους και μηχανισμό ταξινόμησης των πακέτων, ανάλογα με την χρονική σήμανση τους, στις εξόδους. Ένα άλλο αρνητικό σημείο, είναι ότι για την κατασκευή ενός Παράλληλου Μεταγωγέα K επιπέδων απαιτείται K φορές περισσότερο υλικό (hardware) από ό,τι για έναν απλό μεταγωγέα.

3.2 Βελτιώσεις του μηχανισμού FIFO

3.2.1 Μηχανισμός Παραθύρου στις Εισόδους

Ο ρυθμός διέλευσης ενός μεταγωγέα μπορεί να αυξηθεί, αν αρθούν ή γίνουν πιο ελαστικοί κάποιοι από τους περιορισμούς του σχήματος FIFO που εφαρμόζεται στις μονάδες εισόδου. Σύμφωνα με τα παραπάνω, κάθε είσοδος θα μπορεί να στείλει στη Μονάδα Μεταγωγής μόνο ένα πακέτο σε κάθε χρονική θυρίδα, αλλά δεν είναι απαραίτητο αυτό το πακέτο να είναι το πρώτο πακέτο της ουράς (επικεφαλής σειράς). Επίσης, όταν πολλαπλά πακέτα κατευθύνονται στην ίδια θύρα εξόδου, μόνο ένα από αυτά μπορεί να μεταφερθεί στη Μονάδα Μεταγωγής κατά τη διάρκεια μιας χρονικής θυρίδας. Στην αρχή κάθε χρονικής θυρίδας, τα πρώτα w πακέτα (όπου w η τιμή του παραθύρου) κάθε ουράς εισόδου ανταγωνίζονται διαδοχικά για τη μεταφορά τους στις μονάδες εξόδου. Τα πακέτα που βρίσκονται πρώτα στις ουρές (επικεφαλής σειράς) ανταγωνίζονται πρώτα. Εξαιτίας του γεγονότος ότι πολλά πακέτα μπορεί να κατευθύνονται στην ίδια έξοδο, κάποια από τα πακέτα επικεφαλής σειράς δε θα επιλεχθούν για μετάδοση. Στις ουρές όπου τα πακέτα επικεφαλής σειράς δεν επιλέχθηκαν, τα δεύτερα πακέτα στη σειρά θα ανταγωνιστούν για μετάδοση στις εναπομείνασες εξόδους, αυτές δηλαδή που δεν έχουν δεσμευτεί από τα πακέτα επικεφαλής σειράς. Αυτή η διαδικασία ανταγωνισμού και δέσμευσης επαναλαμβάνεται μέχρι w φορές για κάθε χρονική θυρίδα και επιτρέπει στα πρώτα w πακέτα κάθε ουράς εισόδου να ανταγωνιστούν διαδοχικά για τις εξόδους που δε δεσμεύτηκαν από τα προηγούμενα στη σειρά πακέτα. Όταν η τιμή του παραθύρου είναι w=1 τότε καταλήγουμε στο σχήμα FIFO.

Ο **Πίνακας 3.1** δείχνει το μέγιστο ρυθμό διέλευσης που έχει επιτευχθεί σε μεταγωγείς, για μεταβλητό αριθμό εισόδων N και για διάφορα μεγέθη παραθύρου w, μέσω εξομοιώσεων. Παρατηρείται ότι ο ρυθμός διέλευσης αυξάνεται σημαντικά καθώς αυξάνεται το μέγεθος του παραθύρου από w=1 (σχήμα FIFO) σε w = 2, 3, 4. Για μεγαλύτερα μεγέθη παραθύρου, η διέλευση αυξάνεται με μικρό ρυθμό. Ακόμα και στην περίπτωση ενός θεωρητικά άπειρου παραθύρου (w =∞), η διέλευση δε φτάνει την τιμή της διέλευσης που παρουσιάζουν οι μεταγωγείς με ενταμίευση στις μονάδες εξόδου. Αυτό συμβαίνει, γιατί η ενταμίευση στις μονάδες εισόδου επιτρέπει στο σύστημα να στείλει μόνο ένα πακέτο από κάθε είσοδο στη

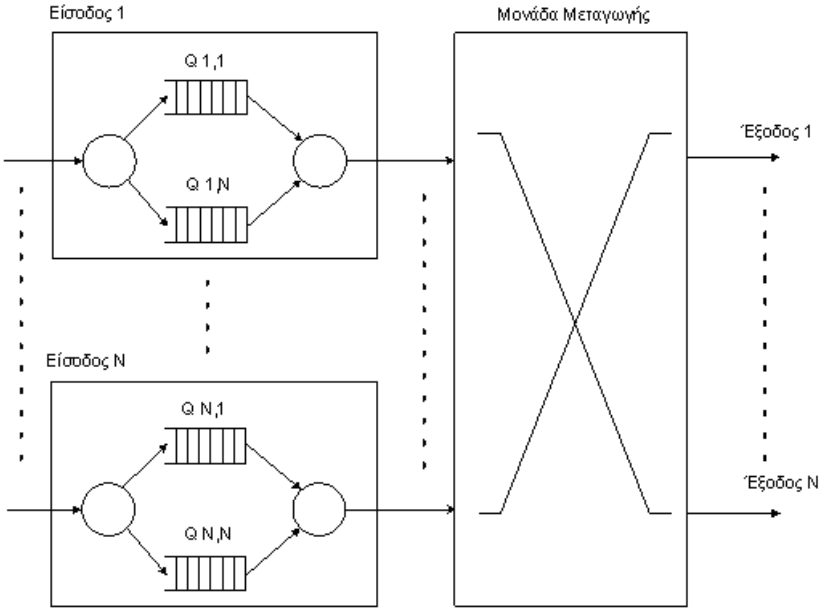
Μονάδα Μεταγωγής, εμποδίζοντας έτσι περισσότερα πακέτα να μεταφερθούν σε ανενεργές εξόδους.

Αριθμός Εισόδων N	Μέγεθος Παραθύρου w							
	1	2	3	4	5	6	7	8
2	0.75	0.84	0.89	0.92	0.93	0.94	0.95	0.96
4	0.66	0.76	0.81	0.85	0.87	0.89	0.91	0.92
8	0.62	0.72	0.78	0.82	0.85	0.87	0.88	0.89
16	0.6	0.71	0.77	0.81	0.84	0.86	0.87	0.88
32	0.59	0.7	0.76	0.8	0.83	0.85	0.87	0.88
64	0.59	0.7	0.76	0.8	0.83	0.85	0.86	0.88
128	0.59	0.7	0.76	0.8	0.83	0.85	0.86	0.88

Πίνακας 3.1: Μέγιστος ρυθμός διέλευσης σε μεταγωγείς, για μεταβλητό αριθμό εισόδων N και για διάφορα μεγέθη παραθύρου w

3.2.2 Εικονικές Ουρές Εξόδου

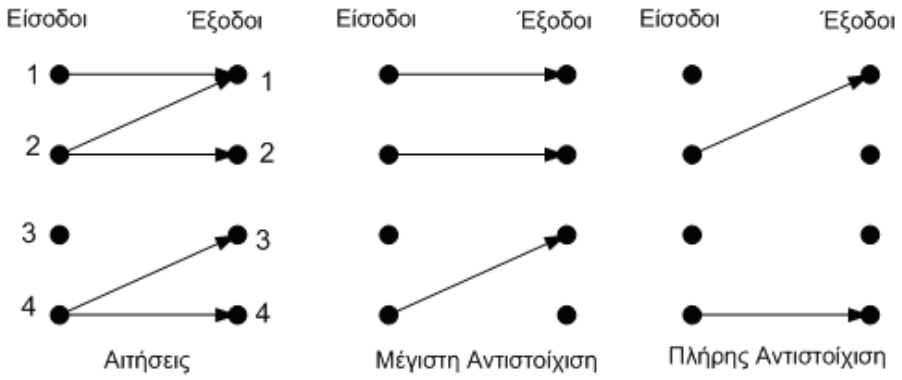
Ένας άλλος τρόπος για την αντιμετώπιση του προβλήματος της Φραγής Επικεφαλής Σειράς (HOL blocking) στους μεταγωγείς με ενταμίευση στις μονάδες εισόδου, είναι η δημιουργία σε κάθε είσοδο μιας ουράς FIFO για κάθε έξοδο. Αυτή η μορφή FIFO ονομάζεται εικονική ουρά εξόδου (virtual output queue, VOQ) και απεικονίζεται στο **Σχήμα 3.2**. Όπως φαίνεται, στην εικονική ουρά εξόδου $Q_{i,j}$ αποθηκεύονται πακέτα που φθάνουν στην είσοδο i και κατευθύνονται στην έξοδο j.



Σχήμα 3.2: Εικονικές Ουρές Εξόδου στις Μονάδες Εισόδου

Στο σχήμα αποθήκευσης σε εικονικές ουρές εξόδου, σε μια είσοδο μπορεί να υπάρχουν πακέτα προς μετάδοση που κατευθύνονται σε διαφορετικές θύρες εξόδου. Επειδή, όμως, κάθε είσοδος μπορεί να στείλει μόνο ένα πακέτο ανά χρονική θυρίδα, τα υπόλοιπα πακέτα πρέπει να περιμένουν για μετάδοση σε επόμενη χρονική θυρίδα, ενώ οι αντίστοιχες έξοδοι παραμένουν ανενεργές. Το μειονέκτημα αυτό μπορεί να αντιμετωπιστεί, εάν ο αλγόριθμος εκτελείται επαναληπτικά κατά τη διάρκεια μιας χρονικής θυρίδας. Σε πιο πολύπλοκες εκδόσεις του αλγορίθμου, χρησιμοποιούνται μέθοδοι αντιστοίχισης για να γίνει πιο αποδοτικός ο χρονοπρογραμματισμός. Έχουν παρουσιαστεί τρεις μέθοδοι αντιστοίχισης: α) πλήρης β) μέγιστη και γ) σταθερή αντιστοίχιση.

Στη μέγιστη αντιστοίχιση δημιουργείται ο μέγιστος δυνατός αριθμός ζευγαριών μεταξύ εισόδων και εξόδων. Η πλήρης αντιστοίχιση είναι η εικόνα αντιστοίχισης την οποία λαμβάνουμε, αν δεχτούμε ότι δεν μπορούμε να έχουμε περαιτέρω αντιστοιχίσεις, χωρίς να αλλάξουμε τις υπάρχουσες αντιστοιχίσεις. Στο **Σχήμα 3.3** φαίνονται η μέγιστη και η πλήρης αντιστοίχιση για συγκεκριμένες αιτήσεις.



Σχήμα 3.3: Μέγιστη Αντιστοίχιση και Πλήρης Αντιστοίχιση

Στη σταθερή αντιστοίχιση χρησιμοποιείται λίστα προτεραιότητας σε κάθε είσοδο και κάθε έξοδο. Η λίστα προτεραιότητας μιας εισόδου περιέχει όλα τα πακέτα που βρίσκονται στην ουρά της εισόδου, ενώ η λίστα προτεραιότητας μιας εξόδου περιέχει όλα τα πακέτα που θέλουν να μεταφερθούν στη συγκεκριμένη έξοδο (κάποια από αυτά μπορεί να βρίσκονται ακόμα στις εισόδους). Η

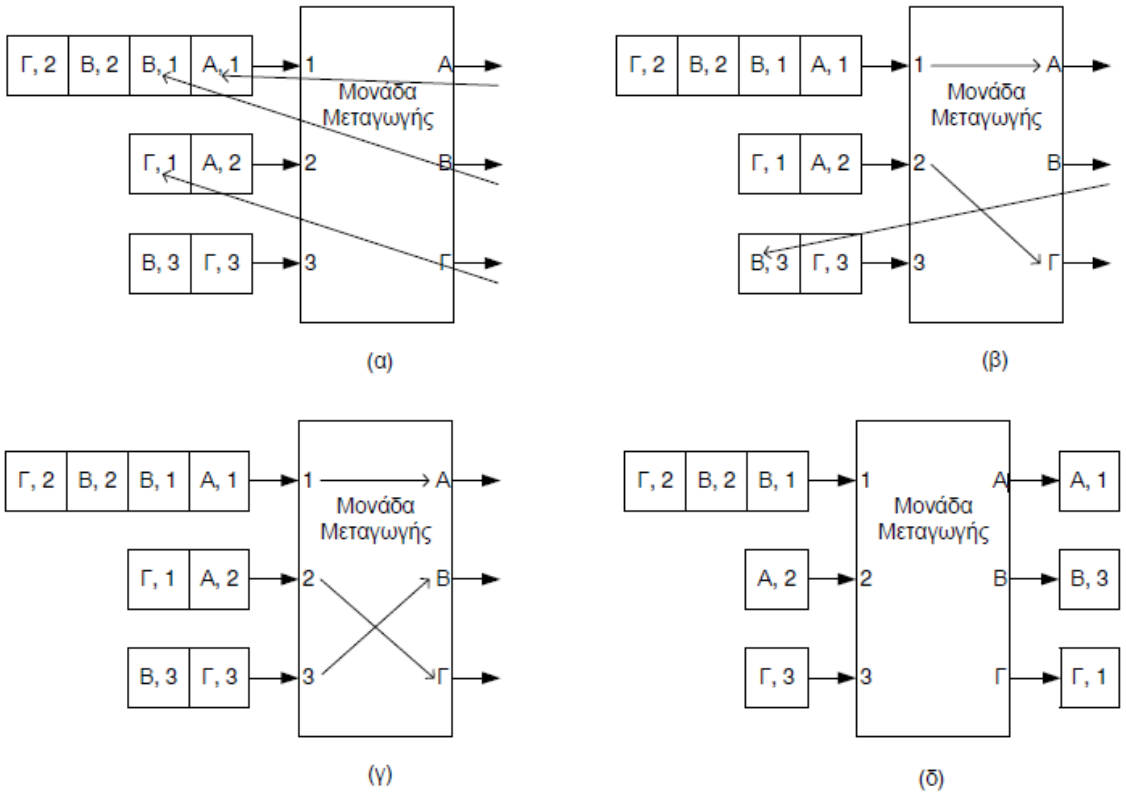
αντιστοίχιση θεωρείται σταθερή, όταν για κάθε πακέτο γ που περιμένει στην ουρά της μονάδας εισόδου, ισχύει μία από τις παρακάτω προϋποθέσεις:

(α) Το πακέτο γ ανήκει στην αντιστοίχιση, δηλαδή το πακέτο πρόκειται να μεταδοθεί από την είσοδο στην έξοδο κατά τη διάρκεια της τρέχουσας φάσης.

(β) Το πακέτο που βρίσκεται μπροστά από το πακέτο γ στη λίστα προτεραιότητας της εισόδου ανήκει στην αντιστοίχιση.

(γ) Το πακέτο που βρίσκεται μπροστά από το πακέτο γ στη λίστα προτεραιότητας της εξόδου ανήκει στην αντιστοίχιση.

Οι προϋποθέσεις (β) και (γ) μπορεί να ικανοποιούνται ταυτόχρονα, ενώ η προϋπόθεση (α) αναιρεί τις άλλες δύο. Στο **Σχήμα 3.4** απεικονίζεται η εφαρμογή του αλγόριθμου σταθερής αντιστοίχισης, σε ένα μεταγωγέα 3x3.



Σχήμα 3.4: Σταθερή αντιστοίχιση σε μεταγωγέα 3x3

Το γράμμα σε κάθε πακέτο συμβολίζει την έξοδο στην οποία θα κατευθυνθεί, ενώ ο αριθμός συμβολίζει τη θέση του πακέτου στη λίστα προτεραιότητας της εξόδου αυτής. Τα βέλη δείχνουν αιτήσεις που γίνονται από τις εισόδους ή τις εξόδους, ανάλογα με τη φορά τους. Κατά τη διάρκεια της πρώτης

επανάληψης, κάθε έξοδος αιτείται για το πακέτο που βρίσκεται πρώτο στη λίστα προτεραιότητάς της, δηλαδή για το πακέτο με αριθμό 1 και γράμμα τη συγκεκριμένη έξοδο (**Σχήμα 3.4.α**). Σε απάντηση των αιτήσεων, η είσοδος 2 δέχεται τη μοναδική αίτηση από την έξοδο Γ για το πακέτο Γ1. Η είσοδος 1 έχει να επιλέξει μεταξύ δυο αιτήσεων από τις εξόδους Α και Β. Τελικά δέχεται την αίτηση από την έξοδο Α για το πακέτο Α1, αφού αυτό το πακέτο βρίσκεται πιο πάνω στη δική της λίστα προτεραιότητας, επειδή βρίσκεται πιο μπροστά στην λίστα της. Έτσι, μετά την πρώτη επανάληψη, οι έξοδοι Α και Γ έχουν αντιστοιχηθεί με τα πακέτα Γ1 και Α1 αντίστοιχα. Κατά τη διάρκεια της επόμενης επανάληψης, η έξοδος Β - που δεν έχει αντιστοιχηθεί ακόμα - αιτείται για το πακέτο από την είσοδο 3 που βρίσκεται πιο πάνω στη λίστα προτεραιότητάς της, αφού τα Β1 και Β2 ανήκουν στην είσοδο 1 που έχει ήδη αντιστοιχηθεί με την έξοδο Α. Αυτό είναι το πακέτο Β3 (**Σχήμα 3.4.β**). Η είσοδος 3 δέχεται την αίτηση και η αντιστοίχιση συμπληρώνεται. Τα πακέτα που επιλέχθηκαν μεταφέρονται στις αντίστοιχες εξόδους, σύμφωνα με τη συμφωνημένη αντιστοίχιση (**Σχήμα 3.4.γ**). Η κατάσταση στο μεταγωγέα μετά την πρώτη χρονική θυρίδα απεικονίζεται στο **Σχήμα 3.4.δ**.

4. Αλγόριθμοι Χρονοπρογραμματισμού

Ο μηχανισμός διαιτησίας ενός μεταγωγέα είναι υπεύθυνος για την επιλογή και τη μεταφορά των πακέτων από τις μονάδες εισόδου στις μονάδες εξόδου. Κάθε έξοδος έχει τον δικό της μηχανισμό διαιτησίας, ο οποίος λειτουργεί ανεξάρτητα από τους υπόλοιπους. Ένα σχήμα διαιτησίας αποφασίζει ποιες πληροφορίες θα μεταφερθούν από τις εισόδους στους μηχανισμούς διαιτησίας, βάσει των οποίων κάθε ένας θα επιλέξει το πακέτο που τελικά θα μεταφερθεί στην αντίστοιχη έξοδο, ανάμεσα στα πακέτα που κατευθύνονται σε αυτήν.

Το σχήμα διαιτησίας είναι ουσιαστικά ένας μηχανισμός που κανονίζει και καθορίζει τη σειρά εξυπηρέτησης των πακέτων στις μονάδες εισόδου. Τα βασικά σχήματα διαιτησίας είναι τα εξής:

α) τυχαία επιλογή

β) FIFO και

γ) κυκλική εναλλαγή (round-robin).

Στο σχήμα τυχαίας επιλογής, ένα πακέτο επιλέγεται τυχαία ανάμεσα στα πακέτα που κατευθύνονται στην ίδια έξοδο. Στο σχήμα FIFO, ο μηχανισμός διαιτησίας επιλέγει το πακέτο που περιμένει τον περισσότερο χρόνο (δηλαδή το πρώτο στην ουρά). Το σχήμα κυκλικής εναλλαγής είναι πιο πολύπλοκο από τα προηγούμενα δύο, εντούτοις είναι ευρέως διαδεδομένο λόγω της διασφάλισης δικαιοσύνης στο χειρισμό όλων των πακέτων.

Ας θεωρήσουμε ότι οι είσοδοι αριθμούνται από το 1 μέχρι το N. Σύμφωνα με το σχήμα κυκλικής εναλλαγής, κάθε μηχανισμός διαιτησίας της εξόδου γνωρίζει την είσοδο από την οποία έφυγε το τελευταίο πακέτο, έστω την είσοδο i. Στην επόμενη φάση, τη μεγαλύτερη προτεραιότητα να μεταφέρει πακέτο έχει η είσοδος i+1. Αν η είσοδος i+1 είναι άδεια, τότε τη μεγαλύτερη προτεραιότητα θα έχει η είσοδος i+2 κ.ο.κ.

Στα πιο απλά σχήματα διαιτησίας, οι πληροφορίες που ανταλλάσσονται μεταξύ εισόδων και εξόδων είναι απλές. Έτσι, μια θύρα εισόδου πληροφορεί μια θύρα εξόδου όταν έχει ένα πακέτο να στείλει σε αυτήν ή αντίστροφα μια θύρα εξόδου πληροφορεί μια θύρα εισόδου ότι μπορεί να στείλει το πακέτο. Σε πιο πολύπλοκα σχήματα χρησιμοποιούνται και άλλες παράμετροι, όπως είναι η

προτεραιότητα και η χρονική σήμανση (timestamp), ώστε οι μηχανισμοί διαιτησίας να δρουν πιο αποτελεσματικά.

4.1 Παράλληλη Αντιστοίχιση (PM)

Ο αλγόριθμος Παράλληλης Αντιστοίχισης (Parallel Matching, PM) χρησιμοποιεί το μηχανισμό της τυχαίας επιλογής για να διευθετήσει το φαινόμενο του ανταγωνισμού μεταξύ των πακέτων στις εισόδους και τις εξόδους. Αρχικά, τα πακέτα στις μονάδες εισόδου αποθηκεύονται σε Εικονικές Ουρές Εξόδου. Ο αλγόριθμος ακολουθεί τρία βήματα. Τα βήματα αυτά εκτελούνται παράλληλα σε κάθε είσοδο και έξοδο και περιγράφονται παρακάτω:

Βήμα 1. Κάθε είσοδος στέλνει μια αίτηση σε κάθε έξοδο για την οποία έχει πακέτο που περιμένει στην ουρά για να μεταφερθεί («ΑΙΤΗΣΕΙΣ»).

Βήμα 2. Όταν μια έξοδος λαμβάνει περισσότερες από μία αιτήσεις αποστολής, τότε επιλέγει να κάνει κράτηση τυχαία για μία από αυτές («ΚΡΑΤΗΣΕΙΣ»). Όλες οι αιτήσεις έχουν την ίδια πιθανότητα να γίνουν αποδεκτές.

Βήμα 3. Όταν μια είσοδος λαμβάνει περισσότερες από μία αιτήσεις παραλαβής από εξόδους, τότε επιλέγει να αποδεχθεί τυχαία μια από αυτές («ΑΠΟΔΟΧΕΣ»).

4.2 Παράλληλη Επαναληπτική Αντιστοίχιση (PIM)

Παραλλαγή του αλγορίθμου Παράλληλης Αντιστοίχισης (PM) αποτελεί ο αλγόριθμος Παράλληλης Επαναληπτικής Αντιστοίχισης (Parallel Iterative Matching, PIM). Η μόνη διαφορά του PIM με τον PM είναι ότι στην περίπτωση του PIM, τα τρία βήματα του αλγορίθμου PM επαναλαμβάνονται για τις μονάδες εισόδου και εξόδου που δε συμμετέχουν στην αντιστοίχιση, μέχρι να επέλθει πλήρης αντιστοίχιση. Επομένως, τα 3 βήματα της κάθε επανάληψης (μέχρι να επέλθει πλήρης αντιστοίχιση) είναι τα εξής:

Βήμα 1. Κάθε είσοδος που δε συμμετέχει σε αντιστοίχιση στέλνει μια αίτηση σε κάθε έξοδο για την οποία έχει πακέτο που περιμένει στην ουρά για να μεταφερθεί («ΑΙΤΗΣΕΙΣ»).

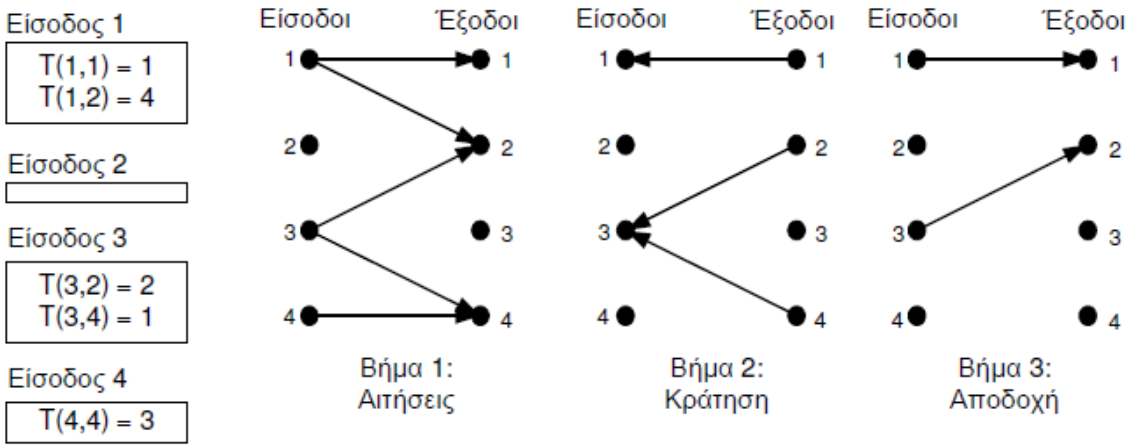
Βήμα 2. Όταν μια έξοδος που δε συμμετέχει σε αντιστοίχιση λαμβάνει περισσότερες από μία αιτήσεις αποστολής, τότε επιλέγει να κάνει κράτηση τυχαία

για μία από αυτές («ΚΡΑΤΗΣΕΙΣ»). Όλες οι αιτήσεις έχουν την ίδια πιθανότητα να γίνουν αποδεκτές.

Βήμα 3. Όταν μια είσοδος λαμβάνει περισσότερες από μία αιτήσεις παραλαβής από εξόδους, τότε επιλέγει να αποδεχθεί τυχαία μια από αυτές («ΑΠΟΔΟΧΕΣ»).

Σχετικές μελέτες έχουν δείξει ότι περίπου το 75% των αιτήσεων που υπολείπονται, θα ικανοποιηθούν στην επόμενη επανάληψη. Ο αλγόριθμος έχει λογαριθμική πολυπλοκότητα (τείνει σε logN επαναλήψεις). Εξαιτίας της τυχαίας επιλογής των αιτήσεων, δεν απαιτείται ο αλγόριθμος να γνωρίζει την τελευταία είσοδο που έστειλε πακέτο (αλγόριθμος χωρίς μνήμη). Από την άλλη, η υλοποίηση ενός μηχανισμού τυχαίας επιλογής σε μεγάλες ταχύτητες, μπορεί να αποδειχθεί πολύ δαπανηρή.

Στο **Σχήμα 4.1** φαίνεται η λειτουργία του αλγόριθμου PM (ή αντίστοιχα η πρώτη επανάληψη του αλγορίθμου PIM). Ο συμβολισμός Τ(κ,λ) = μ σημαίνει ότι υπάρχουν μ πακέτα στην Εικονική Ουρά Εξόδου που κατευθύνονται από την είσοδο κ στην έξοδο λ. Όταν η εισερχόμενη κίνηση ακολουθεί ομοιόμορφη κατανομή, ο αλγόριθμος επιτυγχάνει ρυθμό διέλευσης από 63% μέχρι 100% για αριθμό επαναλήψεων από 1 μέχρι Ν αντίστοιχα, για μεταγωγέα NxN.



Σχήμα 4.1: Πιθανό στιγμιότυπο μεταγωγέα που χρησιμοποιεί αλγόριθμο χρονοπρογραμματισμού PM (μπορεί να ταυτίζεται με το στιγμιότυπο της πρώτης επανάληψης μεταγωγέα που χρησιμοποιεί αλγόριθμο χρονοπρογραμματισμού PIM).

4.3 Επαναληπτική Αντιστοίχιση με Κυκλική Εναλλαγή (iRRM)

Το σχήμα Επαναληπτικής Αντιστοίχισης με Κυκλική Εναλλαγή (Iterative Round-Robin Matching, iRRM) είναι παρόμοιο με το σχήμα Παράλληλης Επαναληπτικής Αντιστοίχισης (PIM), με τη μόνη διαφορά ότι χρησιμοποιεί μηχανισμό κυκλικής εναλλαγής και όχι τυχαίας επιλογής για την επιλογή των εισόδων και των εξόδων. Κάθε μηχανισμός διαιτησίας διατηρεί ένα δείκτη που «δείχνει» τη θύρα με τη μεγαλύτερη προτεραιότητα. Αυτός ο δείκτης ονομάζεται δείκτης αποδοχής αj στην είσοδο j ή δείκτης παραχώρησης (grant pointer) βi στην έξοδο i. Ο αλγόριθμος λειτουργεί ως εξής:

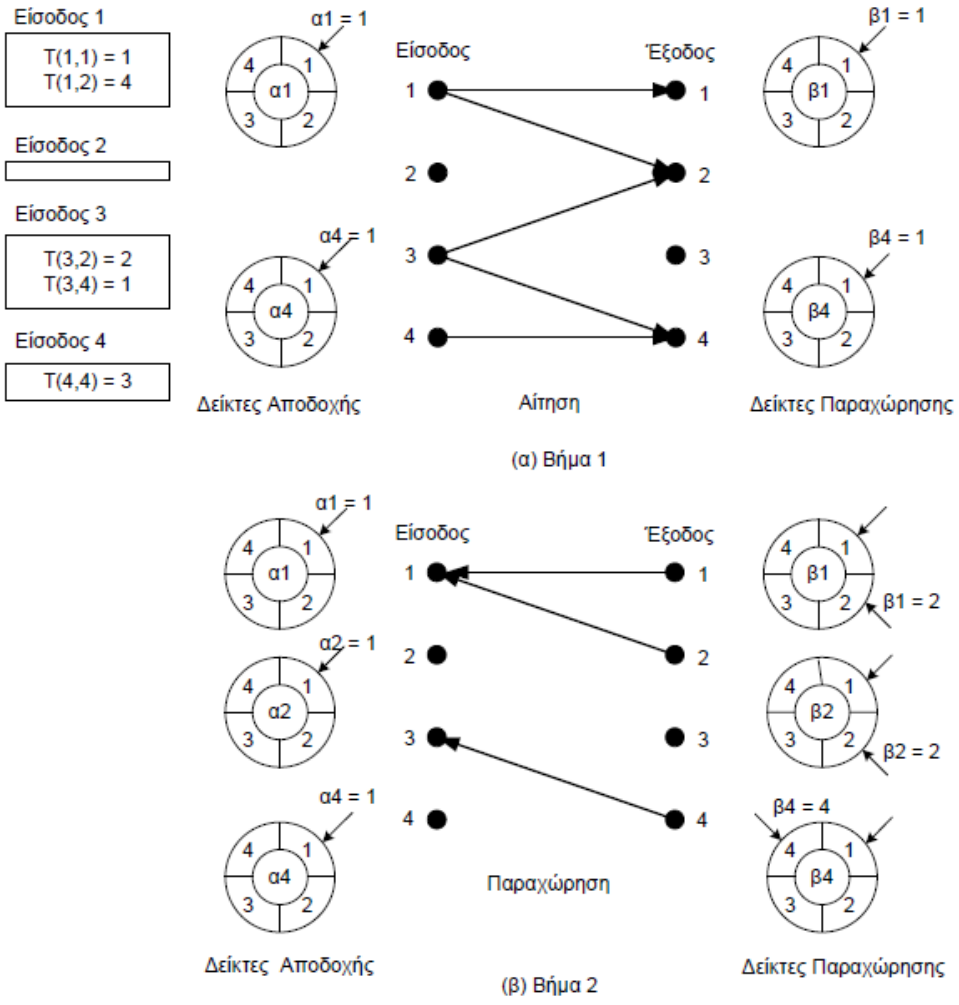
Βήμα 1. Κάθε είσοδος που δε συμμετέχει σε αντιστοίχιση στέλνει μια αίτηση αποστολής σε κάθε έξοδο για την οποία έχει πακέτο που περιμένει στην ουρά για να μεταφερθεί («ΑΙΤΗΣΕΙΣ»).

Βήμα 2. Όταν μια έξοδος που δε συμμετέχει σε αντιστοίχιση λαμβάνει περισσότερες από μία αιτήσεις, τότε επιλέγει την αίτηση που βρίσκεται σε επόμενη θέση στο σχήμα κυκλικής εναλλαγής, ξεκινώντας από τη θέση μέγιστης προτεραιότητας. Η έξοδος ειδοποιεί όλες τις αιτούμενες εισόδους για το αν έγινε δεκτή ή όχι η αίτηση τους («ΠΑΡΑΧΩΡΗΣΕΙΣ»). Ο δείκτης παραχώρησης αυξάνεται και τοποθετείται μία θέση πάνω από την θέση της εισόδου στην οποία έγινε η κράτηση. Ο δείκτης παραχώρησης αλλάζει μόνο κατά την πρώτη επανάληψη.

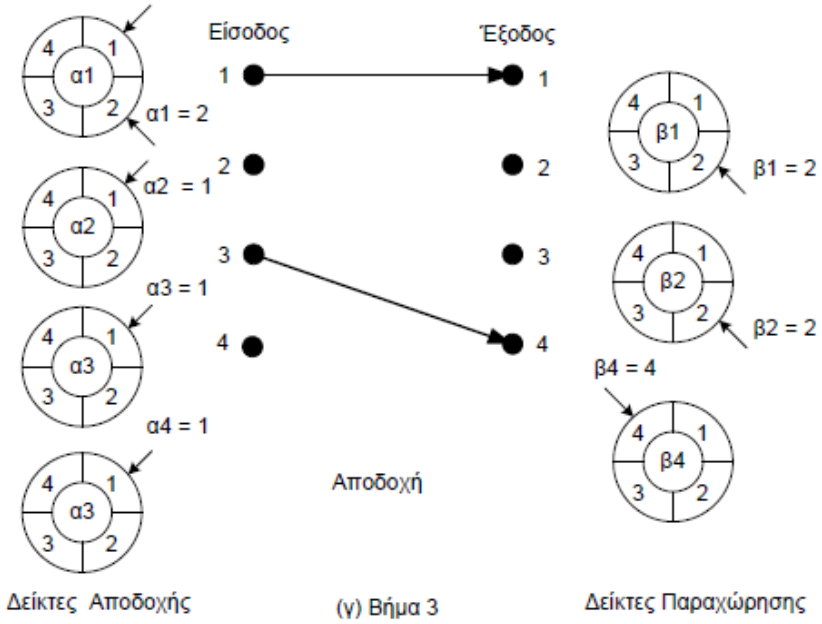
Βήμα 3. Όταν μια είσοδος λάβει περισσότερες από μία αιτήσεις, δέχεται αυτή που βρίσκεται πιο κοντά στη θέση μέγιστης προτεραιότητας, σε ένα σχήμα κυκλικής εναλλαγής («ΑΠΟΔΟΧΕΣ»). Ο δείκτης αποδοχής αυξάνεται και μετακινείται στην επόμενη θέση από τη θέση της εξόδου που έγινε δεκτή. Ο δείκτης αποδοχής αλλάζει μόνο κατά την πρώτη επανάληψη.

Στο **Σχήμα 4.2** φαίνεται ένα παράδειγμα του αλγόριθμου iRRM. Θεωρούμε αρχικά ότι κάθε δείκτης παραχώρησης δείχνει στην είσοδο 1 (βi = 1). Ομοίως κάθε δείκτης αποδοχής αρχικά δείχνει στην έξοδο 1 (αj = 1). Στη διάρκεια του πρώτου βήματος (βήμα «ΑΙΤΗΣΕΙΣ»), οι είσοδοι αιτούνται μετάδοση σε κάθε έξοδο για την οποία έχουν κάποιο πακέτο στην ουρά που περιμένει να μεταδοθεί σε αυτή. Στο δεύτερο βήμα (βήμα «ΠΑΡΑΧΩΡΗΣΕΙΣ»), κάθε μηχανισμός στην έξοδο επιλέγει εκείνη την αίτηση που είναι πιο κοντά στη θέση που δείχνει ο δείκτης

παραχώρησης. Έτσι, στην περίπτωση του σχήματος, η έξοδος 1 επιλέγει την είσοδο 1, η έξοδος 2 την είσοδο 1, η έξοδος 3 δεν έχει δεχθεί καμιά αίτηση, ενώ η έξοδος 4 επιλέγει την είσοδο 3. Στη συνέχεια, κάθε δείκτης παραχώρησης μετακινείται στην επόμενη της επιλεγμένης θέση. Επομένως β1 = 2, β2 = 2, β3 = 1 και β4 = 4. Στο τρίτο βήμα (βήμα «ΑΠΟΔΟΧΕΣ»), κάθε δείκτης αποδοχής αποφασίζει ποια αίτηση από τις εξόδους θα γίνει δεκτή με τον ίδιο τρόπο που επέλεξε και ο δείκτης παραχώρησης προηγουμένως (κυκλική εναλλαγή). Στο συγκεκριμένο παράδειγμα η είσοδος 1 επιλέγει την έξοδο 1 και η είσοδος 3 την έξοδο 4. Κατ' αντιστοιχία με τους δείκτες παραχώρησης, οι δείκτες αποδοχής αυξάνονται κατά ένα και γίνονται α1 = 2, α2 = 1, α3 = 1 και α4 = 1. Παρατηρούμε ότι η είσοδος 3 δέχεται την αίτηση της εξόδου 4, οπότε ο δείκτης α3 επιστρέφει στη θέση 1, κάνει δηλαδή μία «ολόκληρη στροφή» γύρω από τον εαυτό του.



Σχήμα 4.2: Επαναληπτική Αντιστοίχιση με Κυκλική Εναλλαγή, iRRM (1η επανάληψη)



Σχήμα 4.2 (συνέχεια): Επαναληπτική Αντιστοίχιση με Κυκλική Εναλλαγή, iRRM (1η επανάληψη)

4.4 Επαναληπτική Κυκλική Εναλλαγή με SLIP (iSLIP)

Ο αλγόριθμος της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP αποτελεί ένα πιο αποτελεσματικό σχήμα σε σχέση με τα προηγούμενα. Η διαφορά έγκειται στο γεγονός ότι στον αλγόριθμο αυτό οι δείκτες παραχώρησης αλλάζουν τη θέση τους, μόνο όταν οι αντίστοιχες αιτήσεις τους γίνουν δεκτές. Έτσι, αποτρέπεται η μονοπώληση του εύρους ζώνης από μία ροή σε βάρος κάποιων άλλων, επειδή το τελευταίο ζευγάρι αντιστοίχισης έχει τη μικρότερη προτεραιότητα στην επόμενη επανάληψη. Τα βήματα του σχήματος της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP είναι:

Βήμα 1. Κάθε είσοδος που δε συμμετέχει σε αντιστοίχιση στέλνει μια αίτηση σε κάθε έξοδο για την οποία έχει πακέτο που περιμένει στην ουρά για να μεταφερθεί («ΑΙΤΗΣΕΙΣ»).

Βήμα 2. Όταν μία έξοδος που δε συμμετέχει σε αντιστοίχιση λάβει περισσότερες από μία αιτήσεις, δέχεται αυτή που βρίσκεται πιο κοντά στη θέση μέγιστης προτεραιότητας, σε ένα σχήμα κυκλικής εναλλαγής. Η έξοδος ειδοποιεί κάθε είσοδο για το αν έγινε δεκτή ή όχι η αίτησή της («ΠΑΡΑΧΩΡΗΣΕΙΣ»).

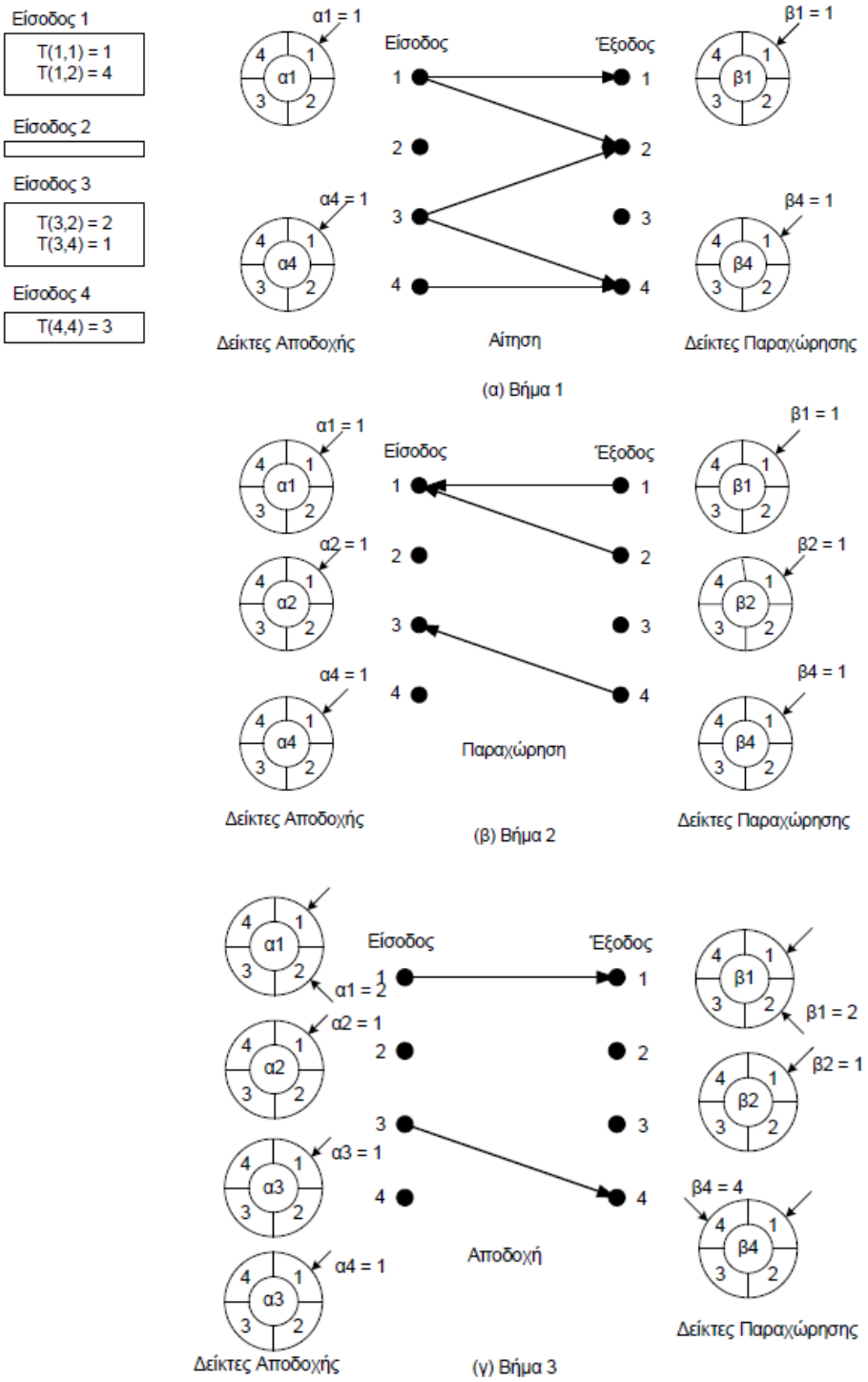
Βήμα 3. Όταν μια είσοδος λάβει περισσότερες από μία αιτήσεις, δέχεται αυτή που βρίσκεται πιο κοντά στη θέση μέγιστης προτεραιότητας, σε ένα σχήμα κυκλικής εναλλαγής («ΑΠΟΔΟΧΕΣ»). Ο δείκτης αποδοχής αυξάνεται και μετακινείται στην επόμενη θέση από τη θέση της εξόδου που έγινε δεκτή. Ο δείκτης αποδοχής αλλάζει μόνο κατά την πρώτη επανάληψη.

Βήμα 4. Ο δείκτης παραχώρησης βi αλλάζει μόνο αν η αίτηση της εξόδου γίνει δεκτή από την είσοδο, κατά τη διάρκεια του τρίτου βήματος της πρώτης επανάληψης. Σε μια τέτοια περίπτωση, ο δείκτης βi αυξάνεται και μετακινείται στην επόμενη θέση από τη θέση της εισόδου που έγινε δεκτή. Αντίστοιχα με το δείκτη αποδοχής και ο δείκτης παραχώρησης αλλάζει μόνο στην πρώτη επανάληψη.

Εξαιτίας της κυκλικής εναλλαγής της θέσης των δεικτών, ο αλγόριθμος εξασφαλίζει δίκαιη κατανομή του εύρους ζώνης, ανάμεσα σε όλες τις ροές πακέτων. Το σχήμα αυτό αποτελείται από 2N μηχανισμούς διαιτησίας, καθένας από τους οποίους παρουσιάζει μικρή πολυπλοκότητα στην υλοποίηση.

Ένα παράδειγμα του αλγόριθμου της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP εικονίζεται στο **Σχήμα 4.3**. Αρχικά, στο πρώτο βήμα (βήμα «ΑΙΤΗΣΕΙΣ»), όλοι οι δείκτες αj και βi βρίσκονται στη θέση 1. Στο δεύτερο βήμα (βήμα «ΠΑΡΑΧΩΡΗΣΕΙΣ») του αλγορίθμου της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP, κάθε έξοδος δέχεται την αίτηση της εισόδου που είναι πιο κοντά στη θέση του δείκτη (αν κινούμαστε δεξιόστροφα). Όμως, σε αντίθεση με τον αλγόριθμο Επαναληπτικής Αντιστοίχισης με Κυκλική Εναλλαγή (iRRM), ο δείκτης παραχώρησης βi δεν αλλάζει σε αυτό το βήμα, αλλά στο τελευταίο βήμα και μόνο στην περίπτωση που η αίτηση της εξόδου γίνει δεκτή. Στο τρίτο βήμα (βήμα «ΑΠΟΔΟΧΕΣ»), κάθε είσοδος δέχεται την αίτηση της εξόδου που είναι πιο κοντά στη θέση του δείκτη αποδοχής αj. Έτσι, οι δείκτες μεταφέρονται κατά μία θέση πάνω από τη θέση της εξόδου που εγκρίθηκε. Επομένως, στο **Σχήμα 4.3** ο δείκτης α1 γίνεται 2, οι δείκτες α2 και α4 παραμένουν στην ίδια θέση 1, ενώ ο δείκτης α3 μεταφέρεται μια θέση πάνω από τη θέση 4, οπότε μεταφέρεται στη θέση 1 κάνοντας έναν πλήρη κύκλο. Αφού οι είσοδοι αποφασίσουν ποιες αιτήσεις των εξόδων θα γίνουν δεκτές, τότε αλλάζουν οι δείκτες παραχώρησης των εξόδων αυτών. Στο συγκεκριμένο παράδειγμα, οι νέες τιμές των δεικτών παραχώρησης θα είναι β1 = 2, β2 = 1, β3 = 1 και β4 = 4. Στις επόμενες επαναλήψεις, μπορούν

να συμμετέχουν μόνο οι είσοδοι και έξοδοι που δεν ανήκουν ήδη σε κάποια αντιστοίχιση, ενώ οι δείκτες παραμένουν αμετάβλητοι, αφού αλλάζουν μόνο στην πρώτη επανάληψη.



Σχήμα 4.3: Επαναληπτική Κυκλική Εναλλαγή με SLIP (1η επανάληψη)

4.5 Αντιστοίχιση με Διπλή Κυκλική Εναλλαγή (DRRM)

Το σχήμα της Αντιστοίχισης με Διπλή Κυκλική Εναλλαγή (Dual Round-Robin Matching, DRRM) μοιάζει πολύ με το σχήμα της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP, αφού χρησιμοποιεί και αυτό μηχανισμό επιλογής κυκλικής εναλλαγής, αντί για μηχανισμό τυχαίας επιλογής. Διαφέρει στο ότι η επιλογή κυκλικής εναλλαγής αρχίζει από τις εισόδους. Επίσης, σε κάθε είσοδο χρησιμοποιείται ένας μηχανισμός διαιτησίας για την επιλογή μιας Εικονικής Ουράς Εξόδου, σύμφωνα με το σχήμα της κυκλικής εναλλαγής. Μόλις πραγματοποιηθεί η επιλογή, κάθε είσοδος στέλνει μια αίτηση στο μηχανισμό διαιτησίας της εξόδου για την οποία προορίζεται το πακέτο. Ο μηχανισμός διαιτησίας στην έξοδο μπορεί να λάβει μέχρι N αιτήσεις. Επιλέγει μία από αυτές, σύμφωνα με το σχήμα κυκλικής εναλλαγής και στέλνει μια νέα αίτηση στην αντίστοιχη μονάδα εισόδου. Εξαιτίας της χρήσης δύο ομάδων μηχανισμών διαιτησίας κυκλικής εναλλαγής, μία για τις εισόδους και μία για τις εξόδους, ο αλγόριθμος αυτός ονομάζεται διαιτησία με διπλή κυκλική εναλλαγή.

Το σχήμα διαιτησίας με διπλή κυκλική εναλλαγή (DRRM) αποτελείται από τέσσερα βήματα σε κάθε επανάληψη:

Βήμα 1. Κάθε μηχανισμός διαιτησίας στην μονάδα εισόδου πραγματοποιεί επιλογή της αίτησης που θα στείλει.

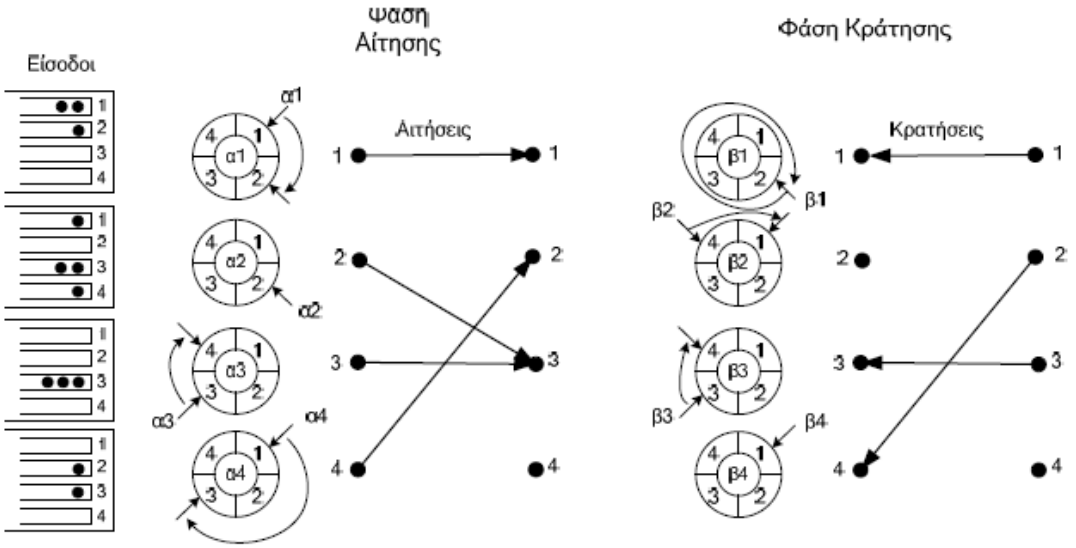
Βήμα 2. Στέλνει την αίτηση στον αντίστοιχο μηχανισμό διαιτησίας της μονάδας εξόδου για την οποία προορίζεται το πακέτο («ΑΙΤΗΣΕΙΣ»).

Βήμα 3. Κάθε μηχανισμός διαιτησίας της εξόδου επιλέγει την αίτηση σύμφωνα με σχήμα κυκλικής εναλλαγής.

Βήμα 4. Οι έξοδοι στέλνουν μηνύματα αποδοχής της αίτησης στις αντίστοιχες εισόδους («ΚΡΑΤΗΣΕΙΣ»).

Στο **Σχήμα 4.4** φαίνεται ένα παράδειγμα του αλγόριθμου DRRM. Κατά τη διάρκεια της φάσης αίτησης (πρώτο και δεύτερο βήμα του αλγόριθμου), κάθε είσοδος επιλέγει μια Εικονική Ουρά Εξόδου και στέλνει μια αίτηση στον αντίστοιχο μηχανισμό διαιτησίας της εξόδου. Σύμφωνα με το **Σχήμα 4.4** η μονάδα εισόδου 1 έχει πακέτα που προορίζονται για τις εξόδους 1 και 2. Επειδή ο δείκτης κυκλικής εναλλαγής α1 δείχνει στη θέση 1, ο μηχανισμός διαιτησίας επιλέγει την πρώτη Εικονική Ουρά Εξόδου και στέλνει μια αίτηση στην έξοδο 1, ενώ ο δείκτης α1

μετακινείται στη θέση 2. Ας υποθέσουμε ότι βρισκόμαστε στη φάση κράτησης (τρίτο και τέταρτο βήμα του αλγόριθμου) και εξετάζουμε τη μονάδα εξόδου 3. Επειδή ο δείκτης κυκλικής εναλλαγής β3 βρίσκεται στη θέση 3, ο μηχανισμός διαιτησίας δέχεται την αίτηση της εισόδου 3, ενώ ο δείκτης β3 μετακινείται στη θέση 4.



Σχήμα 4.4: Αλγόριθμος Αντιστοίχισης με Διπλή Κυκλική Εναλλαγή, DRRM (1η επανάληψη)

Όπως και στον αλγόριθμο της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP, έτσι και στο σχήμα DRRM εμφανίζεται το ευεργετικό φαινόμενο του αποσυγχρονισμού, αφού οι δείκτες στους μηχανισμούς διαιτησίας στις εισόδους, σε διαφορετικές χρονικές θυρίδες, βρίσκονται σε διαφορετική θέση και παράλληλα οι μηχανισμοί διαιτησίας των εισόδων αιτούνται σε διαφορετικές εξόδους. Στην πραγματικότητα ο αλγόριθμος DRRM υπερτερεί της iSLIP γιατί απαιτείται λιγότερος χρόνος από ότι στο σχήμα iSLIP για να γίνει διαιτησία, ενώ είναι πιο εύκολο να υλοποιηθεί. Αυτό συμβαίνει επειδή απαιτείται η ανταλλαγή λιγότερου όγκου πληροφορίας στους μηχανισμούς διαιτησίας μεταξύ των εισόδων και των εξόδων. Δηλαδή, το σχήμα της Αντιστοίχισης με Διπλή Κυκλική Εναλλαγή εξοικονομεί τον χρόνο που απαιτείται για τη μετάδοση των αιτήσεων από τις εισόδους στις εξόδους στο σχήμα της Επαναληπτικής Κυκλικής Εναλλαγής με SLIP.

5. Περιγραφή και επεξήγηση κώδικα

Με σκοπό να υλοποιηθούν οι πέντε αλγόριθμοι χρονοπρογραμματισμού που αναφέρθηκαν και περιγράφηκαν στο κεφάλαιο 4, καθώς και να προσομοιωθεί η λειτουργία τους σε γραφικό περιβάλλον, αναπτύχθηκε κώδικας στη γλώσσα **JAVA** με χρήση του ολοκληρωμένου περιβάλλοντος ανάπτυξης (Intergrated Development Environment, IDE) **NetBeans IDE 8.2**. Το πρόγραμμα έχει εκπαιδευτικό χαρακτήρα και στόχο έχει την κατανόηση της λειτουργίας των αλγορίθμων προγραμματισμού. Στο παρόν κεφάλαιο θα παρατεθεί τμηματικά ολόκληρος ο κώδικας και θα επεξηγηθεί η λειτουργία του με τη βοήθεια εικόνων (screenshots).

Μετά την εισαγωγή των απαραίτητων για το πρόγραμμα κλάσεων της Java, δημιουργήθηκε η κλάση **Item** για να περιγράψει το βασικό αντικείμενο του προγράμματος που είναι το πακέτο μεταγωγής (**Εικόνα 5.1**). Η κλάση έχει δύο πεδία (fields), έναν τύπου χαρακτήρα (**head**) και έναν ακεραίου τύπου (**dest**). Ο χαρακτήρας περιγράφει μοναδικά το κάθε πακέτο, ενώ ο ακέραιος τον προορισμό του, δηλαδή σε ποια θύρα εξόδου απευθύνεται. Για παράδειγμα, A4 είναι το πακέτο με όνομα A που κατευθύνεται προς τη θύρα εξόδου 4. Αν μια θέση στις εισόδους είναι κενή, τότε του δίνουμε το όνομα X9. Κατά την εκκίνηση του προγράμματος, όλες οι θέσεις στις εξόδους έχουν το όνομα X9, αφού όλες αρχικά είναι κενές.

Στην κύρια κλάση του προγράμματος **GCHRONO** που ακολουθεί, πριν την κεντρική μέθοδο (**main**), θα περιγραφεί η λειτουργία όλων των μεθόδων που χρησιμοποιήθηκαν. Να σημειωθεί ότι, χάριν ευκολίας και ελέω εκπαιδευτικού χαρακτήρα, οι κλάσεις Item και GCHRONO γράφτηκαν στο ίδιο αρχείο, αν και η πιστή ακολουθία των κανόνων ορθής χρήσης του αντικειμενοστραφούς προγραμματισμού θα επέβαλλε να γραφτούν σε ξεχωριστά.

Προηγουμένως όμως να ρίξουμε μια ματιά στις **global** μεταβλητές της κλάσης GCHRONO, αυτές δηλαδή που χρησιμοποιούνται από όλες τις μεθόδους (**Εικόνα 5.2**). Οι μεταβλητές αυτές έχουν δηλωθεί ως **static** ώστε να μπορούν να χρησιμοποιηθούν και από την static κεντρική μέθοδο main. Η μεταβλητή **N** είναι το πλήθος εισόδων και εξόδων του μεταγωγέα. Η μεταβλητή **M** είναι ο μέγιστος αριθμός χρονοθυρίδων που αναμένεται να χρειαστεί, ώστε να σχεδιαστεί γραφικά

ο κατάλληλος πίνακας εξόδου. Οι μεταβλητές **A** και **D** είναι βοηθητικές κι έχουν να κάνουν με μεγέθη στο σχεδιαστικό κομμάτι. Η A είναι το μήκος πλευράς του «γραφικού» πακέτου και D είναι η απόσταση του πίνακα εισόδου από αυτόν της εξόδου. Η μεταβλητή **lb** είναι ένας πίνακας NxN από **JLabels**, των γραφικών εκείνων στοιχείων δηλαδή που χρησιμοποιούνται για εμφάνιση και κίνηση ετικετών. Ο lb πίνακας αποθηκεύει όλα τα προς μεταγωγή πακέτα. Οι NxN πίνακες **arr1**, **arr2**, **arr3** και **arr4** αποθηκεύουν τις πληροφορίες για τις αιτήσεις, τις κρατήσεις ή παραχωρήσεις, τις αποδοχές και την πλήρη αντιστοίχηση που εξηγήθηκαν αναλυτικά στα προηγούμενα κεφάλαια. Οι πίνακες **a[N]** και **b[N]** αποθηκεύουν τους δείκτες αποδοχής και παραχώρησης αντίστοιχα. Τέλος, η μεταβλητή **choice** αποθηκεύει την επιλογή του χρήστη στα μενού που εμφανίζονται στο πρόγραμμα, ενώ η boolean μεταβλητή **cont**, ελέγχει τη μετάβαση ανάμεσα στις χρονοθυρίδες.

```
package gchrono;

import AppPackage.AnimationClass;
import java.util.Scanner;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JComponent;

class Item {

   char head;
   int dest;

   public Item (char head, int dest) {
      this.head = head;
      this.dest = dest;
   }

}
```

Εικόνα 5.1: Οι βιβλιοθήκες και η κλάση Item

```
public class GCHRONO extends JComponent {

   static final int N = 4;
   static final int M = 8;

   static final int A = 38;
   static final int D = 400;

   static JLabel[][] lb = new JLabel[N][N];

   static int[][] arr1 = new int[N][N];
   static int[][] arr2 = new int[N][N];
   static int[][] arr3 = new int[N][N];
   static int[][] arr4 = new int[N][N];

   static int[] a = new int[N];
   static int[] b = new int[N];

   static int choice;

   static boolean cont = true;
```

Εικόνα 5.2: Οι global μεταβλητές

```
@Override
   public void paintComponent (Graphics g) {
      int x=10;
      int y=50;
      int s2=150;
      int d1=s2/5;
      int d2=300;

      if (choice==5){
         x=150;
         d2=350;
      }

      g.setColor(Color.black);
      g.drawString("ΑΙΤΗΣΕΙΣ", x, y-20);
      if ((choice==3)||(choice==4)){
         g.drawString("ΠΑΡΑΧΩΡΗΣΕΙΣ", x+d2, y-20);
      }else{
         g.drawString("ΚΡΑΤΗΣΕΙΣ", x+d2, y-20);
      }
      if (choice!=5){
         g.drawString("ΑΠΟΔΟΧΕΣ", x+2*d2, y-20);
      }

      g.drawRect (x, y, s2, s2);
      g.drawString("0.", x+2, y+d1);
      g.drawString(".0", x+s2-12, y+d1);
      g.drawString("1.", x+2, y+2*d1);
      g.drawString(".1", x+s2-12, y+2*d1);
      g.drawString("2.", x+2, y+3*d1);
      g.drawString(".2", x+s2-12, y+3*d1);
      g.drawString("3.", x+2, y+4*d1);
      g.drawString(".3", x+s2-12, y+4*d1);

      g.drawRect (x+d2, y, s2, s2);
      g.drawString("0.", x+2+d2, y+d1);
      g.drawString(".0", x+d2+s2-12, y+d1);
      g.drawString("1.", x+2+d2, y+2*d1);
      g.drawString(".1", x+d2+s2-12, y+2*d1);
      g.drawString("2.", x+2+d2, y+3*d1);
      g.drawString(".2", x+d2+s2-12, y+3*d1);
      g.drawString("3.", x+2+d2, y+4*d1);
      g.drawString(".3", x+d2+s2-12, y+4*d1);
```

Εικόνα 5.3: Η μέθοδος paintComponent

```
        if (choice!=5){
            g.drawRect (x+2*d2, y, s2, s2);
            g.drawString("0.", x+2+2*d2, y+d1);
            g.drawString(".0", x+2*d2+s2-12, y+d1);
            g.drawString("1.", x+2+2*d2, y+2*d1);
            g.drawString(".1", x+2*d2+s2-12, y+2*d1);
            g.drawString("2.", x+2+2*d2, y+3*d1);
            g.drawString(".2", x+2*d2+s2-12, y+3*d1);
            g.drawString("3.", x+2+2*d2, y+4*d1);
            g.drawString(".3", x+2*d2+s2-12, y+4*d1);
        }

        if (choice==3){
            g.setColor(Color.magenta);
            for (int i = 0; i < N; i++) {
                g.drawString("b"+i+"="+b[i], x+d2+s2-12+17, y+d1+i*d1);
                g.drawString("a"+i+"="+a[i], x+2*d2-32, y+d1+i*d1);
            }
        }else if (choice==4){
            g.setColor(Color.magenta);
            for (int i = 0; i < N; i++) {
                g.drawString("a"+i+"="+a[i], x+2*d2-32, y+d1+i*d1);
                g.drawString("b"+i+"="+b[i], x+2*d2+s2-12+17, y+d1+i*d1);
            }
        }else if (choice==5){
            g.setColor(Color.magenta);
            for (int i = 0; i < N; i++) {
                g.drawString("a"+i+"="+a[i], x-32, y+d1+i*d1);
                g.drawString("b"+i+"="+b[i], x+d2+s2-12+17, y+d1+i*d1);
            }
        }

        g.setColor(Color.black);
        drawArrows (arr1, g, 0);
        g.setColor(Color.red);
        drawArrows (arr2, g, 1);
        if (choice!=5){
            g.setColor(Color.blue);
            drawArrows (arr3, g, 2);
            if (choice!=1){
                g.setColor(Color.green);
                drawArrows (arr4, g, 2);
            }
        }
    }
```

Εικόνα 5.3 (συνέχεια): Η μέθοδος paintComponent

5.1 Μέθοδοι

Η **paintComponent** (**Εικόνα 5.3**) είναι μέθοδος της JComponent κλάσης της Java, την οποία και υπερκαλύπτουμε. Είναι η μέθοδος που εκτελείται αυτόματα κάθε φορά που δημιουργείται ή ανανεώνεται ένα «παράθυρο» γραφικών, για να εμφανίσει τα στοιχεία που περιέχονται σε αυτό. Στο συγκεκριμένο πρόγραμμα, χρησιμοποιείται για να σχεδιάσει τα πλαίσια όπου φαίνονται οι αιτήσεις, οι κρατήσεις ή παραχωρήσεις και οι αποδοχές σε όλα τα στάδια των αλγορίθμων, όπου απαιτούνται, καθώς και οι απαραίτητες συμβολοσειρές. Όλες οι μεταβλητές που χρησιμοποιούνται αφορούν σε μεγέθη του γραφικού σχεδιασμού, ούτως ώστε ο προγραμματιστής να μπορεί εύκολα να μεταβάλλει την εμφάνιση του παραθύρου, χωρίς να χρειαστεί να αλλάζει και τις παρακάτω εντολές. Οι μεταβλητές **x** και **y** ορίζουν την πάνω αριστερά γωνία του πρώτου πλαισίου, αυτό των αιτήσεων. Η **s2** είναι το μήκος πλευράς των πλαισίων. Η **d1** είναι η κάθετη απόσταση μεταξύ των εισόδων που φυσικά εξαρτάται από την s2, ενώ **d2** είναι η οριζόντια απόσταση μεταξύ των πλαισίων.

Οι βασικές εντολές σχεδιασμού που χρησιμοποιούνται είναι οι εξής:

- **setColor**: ορίζει το χρώμα που θα χρησιμοποιηθεί στη σχεδίαση στο εξής, μέχρι αυτό να αλλάξει ξανά με νεότερη εντολή.
- **drawString**: ορίζει το κείμενο και τη θέση εμφάνισης μιας συμβολοσειράς.
- **drawRect**: ορίζει την θέση της πάνω αριστερά γωνίας και το μήκος πλευρών ενός ορθογωνίου πλαισίου.

Επίσης καλείται η μέθοδος drawArrows, που περιγράφεται αμέσως μετά, για να σχεδιάσει τα βελάκια μέσα στα πλαίσια. Σχεδιάζονται μαύρα βελάκια στο πρώτο πλαίσιο των αιτήσεων, κόκκινα στο δεύτερο πλαίσιο των κρατήσεων ή παραχωρήσεων, μπλε στο τρίτο για τις αποδοχές και πράσινα στο ίδιο πλαίσιο για την πλήρη αντιστοίχηση.

Η **drawArrows** (**Εικόνα 5.4**) σχεδιάζει βελάκια μέσα στα πλαίσια που δημιουργήθηκαν προηγουμένως από την paintComponent, που ενώνουν τις εισόδους με τις εξόδους και δείχνουν τις συμμετέχουσες και την κατεύθυνση των αιτήσεων, κρατήσεων κ.λ.π. Έχει ως όρισμα έναν δισδιάστατο πίνακα **p** ο οποίος

περιέχει «1» όπου υπάρχει συμμετοχή και «0» παντού αλλού. Για παράδειγμα, αν η είσοδος 2 κάνει αίτηση στην έξοδο 3, τότε υπάρχει «1» στη θέση [2,3] του πίνακα και επομένως σχεδιάζεται το αντίστοιχο βελάκι. Έχει επίσης όρισμα και έναν ακέραιο αριθμό **k**, ο οποίος εκφράζει τον αριθμό του πλαισίου - και άρα του σταδίου - στο οποίο αναφερόμαστε τη δεδομένη στιγμή, αν δηλαδή είμαστε στο στάδιο των αιτήσεων ή των αποδοχών κ.ο.κ.

Η drawArrows χρησιμοποιεί τις ίδιες μεταβλητές μεγέθους που χρησιμοποιεί και η paintComponent αφού βασικά αναφέρεται στο ίδιο γραφικό παράθυρο.

Οι βασικές εντολές σχεδιασμού που χρησιμοποιούνται εδώ είναι οι εξής:

- **drawLine**: σχεδιάζει μια γραμμή ανάμεσα σε δύο δεδομένα σημεία.
- **fillOval**: σχεδιάζει μια συμπαγή έλλειψη σε συγκεκριμένη θέση και ορισμένου μεγέθους.

Η fillOval χρησιμοποιείται για να σχεδιάσει την «αιχμή» του βέλους.

```
public void drawArrows (int[][] p, Graphics g, int k) {
        int i, j;
        int x=10;
        int y=50;
        int s2=150;
        int d1=s2/5;
        int d2=300;

        if (choice==5){
           x=150;
           d2=350;
        }

        for (i = 0; i < N; i++) {
           for (j = 0; j < N; j++) {
              if (p[i][j] == 1) {
                 int x1 = (x+10)+k*d2;
                 int y1 = (y+d1)+i*d1;
                 int x2 = (x+s2-12)+k*d2;
                 int y2 = (y+d1)+j*d1;
                 g.drawLine(x1, y1, x2, y2);
                 if (k == 1) {
                    g.fillOval (x1-3,y1-3,7,7);
                 }else{
                    g.fillOval (x2-3,y2-3,7,7);
                 }
              }
           }
        }
     }
```

Εικόνα 5.4: Η μέθοδος drawArrows

Η μέθοδος **request** (**Εικόνα 5.5**) παίρνει ως όρισμα τον πίνακα των εισόδων **in** και φτιάχνει τον πίνακα των αιτήσεων **p**. Υπάρχουν δύο μέθοδοι request. Η δεύτερη, που χρησιμοποιείται μόνο για τον αλγόριθμο DRRM (**Εικόνα 5.6**), έχει ένα όρισμα επιπλέον – έναν **a** πίνακα N στοιχείων – που είναι ο πίνακας κυκλικής εναλλαγής, καθώς όπως είδαμε στο προηγούμενο κεφάλαιο, οι αιτήσεις στη μέθοδο DRRM, χρησιμοποιούν το σχήμα κυκλικής εναλλαγής για να επιλέξουν την έξοδο η οποία έχει προτεραιότητα για κάθε είσοδο.

```
static void request (int[][] p, Item[][] in) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            p[i][j] = 0;
         }
      }
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            k = in[i][j].dest;
            if ((k>=0)&&(k<=N)) {
               p[i][k] = 1;
            }
         }
      }
      System.out.println("\n*** REQUESTS ***\n");
      print_table(p);
   }
```

Εικόνα 5.5: Η μέθοδος request

```
static void request (int[][] p, Item[][] in, int[] a) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            p[i][j] = 0;
         }
      }
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            k = in[i][j].dest;
            if ((k>=0)&&(k<=N)) {
               p[i][k] = 1;
            }
         }
      }
      for (i = 0; i < N; i++) {
         int sum_row = 0;
         for (j = 0; j < N; j++) {
            sum_row += p[i][j];
         }
         if (sum_row > 0) {
            k = a[i];
            while (p[i][k] == 0) {
               if (k == N - 1) {
                  k = 0;
               } else {
                  k++;
               }
            }
            for (j = 0; j < N; j++) {
               p[i][j] = 0;
            }
            p[i][k] = 1;
            if (k == N - 1) {
               k = 0;
            } else {
               k++;
            }
            a[i] = k;
         }
      }
      System.out.println("\n*** REQUESTS ***\n");
      print_table(p);
   }
```

Εικόνα 5.6: Η μέθοδος request για DRRM

Η λογική έχει ως εξής: αφού μηδενιστεί ο πίνακας p, σαρώνεται ο πίνακας των εισόδων και για κάθε πακέτο που κατευθύνεται προς μια έξοδο, γράφεται «1» στην αντίστοιχη θέση του p. Δηλαδή για παράδειγμα, αν υπάρχει πακέτο από την είσοδο 3 που κατευθύνεται στην έξοδο 4, γράφεται «1» στη θέση [3,4] του πίνακα p. Η επιπλέον εργασία που γίνεται στη δεύτερη request έχει να κάνει με το σχήμα κυκλικής εναλλαγής. Με δεδομένο τον πίνακα a, για κάθε είσοδο επιλέγεται η έξοδος που έχει για αυτήν προτεραιότητα. Αν λοιπόν σε κάποια σειρά του πίνακα p υπάρχουν περισσότερα του ενός «1», επιλέγεται το σωστό σύμφωνα με τον πίνακα a και τα υπόλοιπα ξαναγίνονται «0». Κατόπιν αυτού, η αντίστοιχη θέση του πίνακα a πηγαίνει «κυκλικά» στον επόμενο αριθμό. Τέλος, γίνεται κλήση της μεθόδου print_table για να εμφανίσουμε στην κονσόλα των αποτελεσμάτων τα περιεχόμενα του πίνακα p των αιτήσεων, τη λειτουργία της οποίας θα δούμε στη συνέχεια.

```
static void reservation (int[][] p) {
      int i, j, k;
      for (j = 0; j < N; j++) {
         int sum_col = 0;
         for (i = 0; i < N; i++) {
            sum_col += p[i][j];
         }
         if (sum_col > 0) {
            k = (int) (Math.random()*N);
            while (p[k][j] == 0) {
               k = (int) (Math.random()*N);
            }
            for (i = 0; i < N; i++) {
               p[i][j] = 0;
            }
            p[k][j] = 1;
         }
      }
      System.out.println("\n*** RESERVATIONS ***\n");
      print_table(p);
   }
```

Εικόνα 5.7: Η μέθοδος reservation για PM/PIM

Η επόμενη μέθοδος είναι η **reservation** η οποία έχει επίσης 2 «εκδόσεις». Στην πρώτη, που χρησιμοποιείται για τους PM και PIM (**Εικόνα 5.7**), όταν έχουμε περισσότερες από μία αιτήσεις σε κάποια συγκεκριμένη έξοδο, επιλέγεται τυχαία η μία από αυτές. Πρακτικά αυτό σημαίνει ότι όταν ο πίνακας των αιτήσεων που

κατασκευάστηκε στο προηγούμενο βήμα έχει περισσότερα του ενός «1» σε κάποια στήλη, καλείται η μέθοδος **Math.random** της Java για να επιλέξει το ένα από αυτά, ενώ τα υπόλοιπα ξαναγίνονται «0». Μοναδικό όρισμα είναι ο πίνακας των αιτήσεων **p**, ο οποίος στο τέλος της μεθόδου θα έχει – πιθανόν – μεταβληθεί.

```
static void reservation (int[][] p, int[] b) {
      int i, j, k;
      for (j = 0; j < N; j++) {
         int sum_col = 0;
         for (i = 0; i < N; i++) {
            sum_col += p[i][j];
         }
         if (sum_col > 0) {
            k = b[j];
            while (p[k][j] == 0) {
               if (k == N - 1) {
                  k = 0;
               } else {
                  k++;
               }
            }
            for (i = 0; i < N; i++) {
               p[i][j] = 0;
            }
            p[k][j] = 1;
            if (choice != 4){
               if (k == N - 1) {
               k = 0;
               } else {
                  k++;
               }
               b[j] = k;
            }
         }
      }
      if (choice == 5){
        System.out.println("\n*** RESERVATIONS ***\n");
      }else{
         System.out.println("\n*** GRANTS ***\n");
      }
      print_table(p);
   }
```

Εικόνα 5.8: Η μέθοδος reservation για iRRM/iSLIP/DRRM

Στη δεύτερη «έκδοση», για τους υπόλοιπους τρεις αλγορίθμους (**Εικόνα 5.8**), η επιλογή του «1» που θα μείνει δε γίνεται τυχαία, αλλά σύμφωνα με το σχήμα της κυκλικής εναλλαγής. Για το λόγο αυτό, η μέθοδος έχει ως επιπλέον όρισμα τον πίνακα **b**, σύμφωνα με τον οποίο επιλέγεται για κάθε έξοδο η αίτηση που έχει προτεραιότητα, όπως ακριβώς έγινε και στη μέθοδο request για την DRRM. Έτσι, κρατάμε το «1» που μας υποδηλώνει ο πίνακας b και μηδενίζουμε όλα τα υπόλοιπα. Όπως και προηγουμένως, ο πίνακας b «δείχνει» στη συνέχεια το επόμενο στη σειρά στοιχείο. Το σημείο που χρειάζεται ιδιαίτερη προσοχή είναι ότι στον τέταρτο αλγόριθμο iSLIP, οι δείκτες παραχώρησης δεν μεταβάλλονται σε αυτό το στάδιο, αλλά στο επόμενο της αποδοχής. Έτσι, με τη συνθήκη **if (choice != 4)**, όταν η μέθοδος καλείται από τον iSLIP, ο πίνακας b δεν μεταβάλλεται. Σε κάθε περίπτωση πάντως, η μέθοδος τελειώνει και πάλι με κλήση της μεθόδου print_table, όπου εμφανίζεται ο πίνακας p ανανεωμένος.

Η μέθοδος **acceptance** που ακολουθεί, έχει αυτή τη φορά 3 «εκδόσεις». Η πρώτη είναι και πάλι για τους PM και PIM (**Εικόνα 5.9**), η δεύτερη είναι για τον iRRM (**Εικόνα 5.10**), ενώ η τρίτη για τον iSLIP (**Εικόνα 5.11**). Ο αλγόριθμος DRRM δεν έχει στάδιο αποδοχής, γι' αυτό και η μέθοδος αυτή δε χρησιμοποιείται καθόλου. Η διαφορά στις τρεις αυτές μεθόδους έγκειται και πάλι στον τρόπο επιλογής της εξόδου που έχει προτεραιότητα. Σε κάθε περίπτωση εξετάζεται αν σε κάποια σειρά του πίνακα **p** υπάρχουν περισσότερα από ένα «1». Στην πρώτη μέθοδο των PM και PIM καλείται και πάλι η **Math.random** μέθοδος για να επιλέξει τυχαία ένα από αυτά και να μηδενίσει τα υπόλοιπα. Στη δεύτερη μέθοδο της iRRM, χρησιμοποιείται το επιπλέον όρισμα του πίνακα **a** και το σχήμα της κυκλικής εναλλαγής με τον τρόπο που είδαμε προηγουμένως, ενώ στην τρίτη, αυτή της iSLIP, έχουμε τα δύο επιπλέον ορίσματα των a και **b** πινάκων που εκφράζουν τους πίνακες αποδοχής και παραχώρησης αντίστοιχα, οι οποίοι αμφότεροι ανανεώνονται κατάλληλα εδώ, όπως ακριβώς επιτάσσει ο αλγόριθμος. Η κλήση της print_table, ολοκληρώνει και πάλι τη μέθοδο.

```
static void acceptance (int[][] p) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         int sum_row = 0;
         for (j = 0; j < N; j++) {
            sum_row += p[i][j];
         }
         if (sum_row > 0) {
            k = (int) (Math.random()*N);
            while (p[i][k] == 0) {
               k = (int) (Math.random()*N);
            }
            for (j = 0; j < N; j++) {
               p[i][j] = 0;
            }
            p[i][k] = 1;
         }
      }
      System.out.println("\n*** ACCEPTANCES ***\n");
      print_table(p);
   }
```

Εικόνα 5.9: Η μέθοδος acceptance για PM/PIM

```
static void acceptance (int[][] p, int[] a) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         int sum_row = 0;
         for (j = 0; j < N; j++) {
            sum_row += p[i][j];
         }
         if (sum_row > 0) {
            k = a[i];
            while (p[i][k] == 0) {
               if (k == N - 1) {
                  k = 0;
               } else {
                  k++;
               }
            }
            for (j = 0; j < N; j++) {
               p[i][j] = 0;
            }
            p[i][k] = 1;
            if (k == N - 1) {
               k = 0;
            } else {
               k++;
            }
            a[i] = k;
         }
      }
      System.out.println("\n*** ACCEPTANCES ***\n");
      print_table(p);
   }
```

Εικόνα 5.10: Η μέθοδος acceptance για iRRM

```
static void acceptance (int[][] p, int[] a, int[] b) {
      int i, j, k, l;
      for (i = 0; i < N; i++) {
         int sum_row = 0;
         for (j = 0; j < N; j++) {
            sum_row += p[i][j];
         }
         if (sum_row > 0) {
            k = a[i];
            while (p[i][k] == 0) {
               if (k == N - 1) {
                  k = 0;
               } else {
                  k++;
               }
            }
            for (j = 0; j < N; j++) {
               p[i][j] = 0;
            }
            p[i][k] = 1;
            l = i;
            if (l == N - 1) {
               l = 0;
            } else {
               l++;
            }
            b[k] = l;
            if (k == N - 1) {
               k = 0;
            } else {
               k++;
            }
            a[i] = k;
         }
      }
      System.out.println("\n*** ACCEPTANCES ***\n");
      print_table(p);
   }
```

Εικόνα 5.11: Η μέθοδος acceptance για iSLIP

Αφού λοιπόν ο πίνακας **p** περάσει από τα τρία προαναφερθέντα στάδια, καλείται η μέθοδος **makeFull** (**Εικόνα 5.12**) για να ολοκληρώσει τη διαδικασία, να εκτελέσει δηλαδή πλήρη αντιστοίχηση, σε όποιον αλγόριθμο από τους πέντε αυτή απαιτείται. Για το σκοπό αυτό σαρώνεται ο πίνακας p. Για κάθε στοιχείο του p που βρίσκεται σε «μηδενική» γραμμή και στήλη (για τον έλεγχο αυτό καλείται η zeroRowCol που θα αναφερθεί παρακάτω), δεν υπάρχει δηλαδή κανένα «1» σε αυτές και άρα κανένα πακέτο που να έχει επιλεγεί από αυτήν την είσοδο για να κατευθυνθεί προς αυτήν την έξοδο, η μέθοδος ελέγχει στον πίνακα **in** των εισόδων - που είναι και το δεύτερο όρισμά της - αν υπάρχει τέτοιο πακέτο, πακέτο δηλαδή σε αυτήν την είσοδο προς αυτήν την έξοδο. Αν υπάρχει, αλλάζει την περί ης ο λόγος θέση του p από «0» σε «1», δρομολογώντας έτσι τη μετακίνηση του πακέτου άμεσα. Η print_table θα κληθεί για ακόμα μία φορά για να εμφανίσει τον ανανεωμένο με την πλήρη αντιστοίχηση πίνακα.

```
static void makeFull (int[][] p, Item[][] in) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            k = 0;
            while ((zeroRowCol(p, i, j)) && (k < N)) {
               if (in[i][k].dest == j) {
                  p[i][j] = 1;
               } else {
                  k++;
               }
            }
         }
      }
      System.out.println("\n*** FULL ALLIGNEMENT ***\n");
      print_table(p);
   }
```

Εικόνα 5.12: Η μέθοδος makeFull

Η μέθοδος **splitArrayFull** (**Εικόνα 5.13**) χρησιμοποιείται ουσιαστικά για να διαχωρίσει τις μετακινήσεις που οφείλονται στην πλήρη αντιστοίχηση από τις υπόλοιπες, έτσι ώστε να μπορέσουν τα αντίστοιχα βελάκια να εμφανιστούν με διαφορετικό χρώμα στο παράθυρο των γραφικών. Για το λόγο αυτό έχει ως ορίσματα 3 πίνακες NxN. Αφού μηδενιστεί ο τρίτος πίνακας, τοποθετούνται «1» στις θέσεις εκείνες όπου στις αντίστοιχες του πρώτου υπάρχει «1», ενώ του

δεύτερου «0». Στην πράξη, ο πρώτος πίνακας είναι αυτός που δημιουργήθηκε μετά την πλήρη αντιστοίχηση, ενώ ο δεύτερος αμέσως πριν. Έτσι, ο τρίτος θα αποθηκεύσει μόνο τα «1» που δημιουργήθηκαν από την πλήρη αντιστοίχηση.

```
static void splitArrayFull (int[][] p, int[][] q, int[][] n) {
        int i, j;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                n[i][j]=0;
                if ((p[i][j]==1)&&(q[i][j]==0)){
                    n[i][j]=1;
                }
            }
        }
    }
```

Εικόνα 5.13: Η μέθοδος splitArrayFull

Η **zeroRowCol** (**Εικόνα 5.14**), με ορίσματα τον γνωστό πίνακα **p** καθώς και μία σειρά και μία στήλη του, επιστρέφει μια boolean τιμή που είναι αληθής όταν οι συγκεκριμένες σειρά και στήλη είναι μηδενικές, δεν περιέχουν δηλαδή κανένα «1». Χρησιμοποιείται όπως είδαμε από τη μέθοδο makeFull για την πλήρη αντιστοίχηση.

```
static boolean zeroRowCol (int[][] p, int r, int c) {
        int i;
        int sum_row = 0;
        int sum_col = 0;
        for (i = 0; i < N; i++) {
            sum_row += p[r][i];
            sum_col += p[i][c];
        }

        return (sum_row + sum_col == 0);
    }
```

Εικόνα 5.14: Η μέθοδος zeroRowCol

Η **print_table** (**Εικόνα 5.15**) που χρησιμοποιήσαμε και σε τέσσερις από τις παραπάνω μεθόδους, δεν κάνει τίποτε άλλο παρά να εμφανίζει στην κονσόλα των αποτελεσμάτων τα περιεχόμενα του πίνακα **p** που έχει ως μοναδικό όρισμα, σχηματοποιημένα σε μορφή δισδιάστατου NxN πίνακα. Χρησιμοποιείται για να

εμφανίζει σε κάθε στάδιο τους πίνακες των αιτήσεων, κρατήσεων, παραχωρήσεων ή αποδοχών κατά περίπτωση.

```
static void print_table (int[][] p) {
      int i, j;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            System.out.printf("%3d", p[i][j]);
         }
         System.out.println();
      }
   }
```

Εικόνα 5.15: Η μέθοδος print_table

```
static void print_point (int[] a, char c) {
      int i;
      System.out.println();
      for (i = 0; i < N; i++) {
         System.out.printf("%c%d=%d ", c, i, a[i]);
      }
      System.out.println();
   }
```

Εικόνα 5.16: Η μέθοδος print_point

Η **print_point** (**Εικόνα 5.16**) εμφανίζει στην κονσόλα των αποτελεσμάτων τα περιεχόμενα ενός μονοδιάστατου πίνακα N ακεραίων με τη μορφή a1=0, b2=3 κ.ο.κ. Χρησιμοποιείται για να εμφανίζει τα περιεχόμενα των δεικτών αποδοχής και παραχώρησης **a** και **b**.

Η **print_switch** (**Εικόνα 5.17**) εμφανίζει στην κονσόλα των αποτελεσμάτων όλα τα πακέτα στις εισόδους και στις εξόδους που υπάρχουν τη δεδομένη χρονική στιγμή, με μια σχηματοποιημένη μορφή.

```
static void print_switch (Item[][] in, Item[][] out, int pos) {
      int i, j;
      System.out.println();
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            System.out.printf("%c %d | ", in[i][j].head, in[i][j].dest);
         }
         System.out.printf(" |--o %d %d o--| ", i, i);
         for (j = 0; j < pos; j++) {
            System.out.printf("%c %d | ", out[i][j].head, out[i][j].dest);
         }
         System.out.println();
      }
   }
```

Εικόνα 5.17: Η μέθοδος print_switch

Η μέθοδος **copyArrays** (**Εικόνα 5.18**) αντιγράφει όλα τα στοιχεία του **p** στον **q**, όπου p και q δύο δισδιάστατοι NxN πίνακες.

Η **endOfTable** (**Εικόνα 5.19**) έχει ως όρισμα έναν NxN πίνακα από στοιχεία τύπου Item και επιστρέφει μια boolean τιμή. Κατά την κλήση της σαρώνει τον πίνακα και μόνο αν όλες οι θέσεις είναι κενές (dest=9), τότε επιστρέφει την τιμή true. Αν βρεθεί έστω και μία μη κενή θέση σε αυτόν, επιστρέφεται η τιμή false. Η μέθοδος χρησιμοποιείται για να ελεγχθεί πότε τελειώνουν όλα τα πακέτα στις εισόδους, ώστε να τερματιστεί η βασική επανάληψη του προγράμματος.

```
static void copyArrays (int[][] p, int[][]q) {
      int i, j;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            q[i][j] = p[i][j];
         }
      }
   }
```

Εικόνα 5.18: Η μέθοδος copyArrays

```
static boolean endOfTable (Item[][] in) {
      int i, j;
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            if (in[i][j].dest != 9) {
               return false;
            }
         }
      }
      return true;
   }
```

Εικόνα 5.19: Η μέθοδος endOfTable

Η τελευταία μέθοδος **updateTable** (**Εικόνα 5.20**) έχει να κάνει με τη μετακίνηση των «πακέτων» από τις εισόδους προς τις εξόδους του μεταγωγέα στο γραφικό περιβάλλον, καθώς και με την ενημέρωση με τα καινούρια δεδομένα των πινάκων εισόδου και εξόδου τόσο στο γραφικό παράθυρο, όσο και στην κονσόλα των αποτελεσμάτων. Η μέθοδος έχει τέσσερις παραμέτρους: τον πίνακα **p** με τα «0» και «1» που περιέχει τις πληροφορίες για το ποια πακέτα θα μετακινηθούν και προς πού, τους πίνακες **in** και **out** με τις πληροφορίες για τα πακέτα που υπάρχουν στις εισόδους και τις εξόδους του μεταγωγέα, καθώς και έναν ακέραιο αριθμό **pos** που εκφράζει τον αριθμό της χρονοθυρίδας στην οποία βρισκόμαστε.

Στο πρώτο μπλοκ της διπλής επανάληψης, όλα τα πακέτα στον πίνακα εξόδου out μετακινούνται κατά μία θέση «δεξιά», ούτως ώστε η πρώτη στήλη του πίνακα να «ελευθερωθεί» για να δεχτεί τα πακέτα που θα έρθουν από τις εισόδους κατά την τρέχουσα χρονοθυρίδα.

```
static void updateTable (int[][] p, Item[][] in, Item[][] out, int pos) {
      AnimationClass AC = new AnimationClass();
      int i, j, k;
      for (i = 0; i < N; i++) {
         for (j = pos; j > 0; j--) {
            if (out[i][j-1].dest != 9) {
               out[i][j].head = out[i][j-1].head;
               out[i][j].dest = out[i][j-1].dest;
               out[i][j-1].head = 'X';
               out[i][j-1].dest = 9;
            }
         }
      }
      for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
            if (p[i][j] == 1) {
               k = 0;
               while (in[i][k].dest != j) {
                  k++;
               }
               out[j][0].head = in[i][k].head;
               out[j][0].dest = in[i][k].dest;
               in[i][k].head = 'X';
               in[i][k].dest = 9;

               AC.jLabelXRight((k+1)*(A+2),(A+2)+D,10,2,lb[i][k]);
               if (i<j) {
                  AC.jLabelYDown((i+1)*(A+2),(j+1)*(A+2),10,2,lb[i][k]);
               }else if (i>j) {
                  AC.jLabelYUp((i+1)*(A+2),(j+1)*(A+2),10,2,lb[i][k]);
               }
            }
            int x = lb[i][j].getX();
            if (x > D) {
               AC.jLabelXRight(x, x+(A+2),10,2,lb[i][j]);
            }
         }
      }
   }
```

Εικόνα 5.20: Η μέθοδος updateTable

Στο δεύτερο μπλοκ επαναλήψεων σαρώνεται ο πίνακας p. Για κάθε θέση με «1» που δηλώνει μετακίνηση, εντοπίζεται το πακέτο που πρέπει να μετακινηθεί και τοποθετείται στη θέση του, στην πρώτη στήλη του πίνακα out. Η προηγούμενη θέση του στον πίνακα in «αδειάζει», δηλαδή παίρνει την τιμή Χ9. Ταυτόχρονα γίνεται το ίδιο και στο γραφικό περιβάλλον: όλα τα πακέτα στον χώρο εξόδου μετακινούνται κατά μία θέση δεξιά, ενώ το προς μεταφορά πακέτο «πετάει»

θεαματικά προς την καινούρια θέση του. Για το λόγο αυτό, έχει δημιουργηθεί το αντικείμενο **AC** τύπου **AnimationClass** της Java και έχουν χρησιμοποιηθεί οι μέθοδοί του **jLabelXRight**, **jLabelYDown**, και **jLabelYUp** για τη γραφική μετακίνηση των labels προς τα δεξιά, προς τα κάτω και προς τα πάνω αντίστοιχα.

5.2 Κεντρική μέθοδος - main

Η κεντρική μέθοδος της εφαρμογής ξεκινάει εμφανίζοντας ένα πρώτο μενού επιλογής, κατά το οποίο ο χρήστης καλείται να επιλέξει αν επιθυμεί να εισαγάγει τα πακέτα στις εισόδους του μεταγωγέα από το πληκτρολόγιο ή να χρησιμοποιήσει το έτοιμο παράδειγμα πακέτων που διατίθεται (**Εικόνα 5.21**). Αν κάνει την πρώτη επιλογή, οφείλει να δώσει τις ταμπέλες των πακέτων ανά γραμμή – και άρα ανά είσοδο – αφήνοντας ένα κενό μεταξύ τους και χρησιμοποιώντας τη μορφή CN, όπου C ένας χαρακτήρας που προσδιορίζει μοναδικά το πακέτο και N ένας ακέραιος αριθμός που δηλώνει τον προορισμό του, δηλαδή προς ποια έξοδο κατευθύνεται. Για παράδειγμα, δίνει F4 για το πακέτο με όνομα F που κατευθύνεται στην έξοδο 4. Για να δηλώσει το κενό πακέτο χρησιμοποιεί, όπως είδαμε, το συμβολισμό X9, ενώ μετά από την ολοκλήρωση κάθε εισόδου πατάει Enter.

```
public static void main(String[] args) {

    Item[][] sinput = new Item[N][N];
    Item[][] soutput = new Item[N][M];

    Scanner kb = new Scanner(System.in);
    String token;

    System.out.println();
    System.out.println("***Author: Kafopoulos Panagiotis, postgraduate student***");
    System.out.println();
    System.out.println("Κάντε την επιλογή σας:");
    System.out.println("1. Δώστε τις ετικέτες και τους προορισμούς των πακέτων από
                        το πληκτρολόγιο");
    System.out.println("2. Χρησιμοποιήστε το default παράδειγμα");
    choice = kb.nextInt();
    System.out.println();
```

Εικόνα 5.21: Κεντρική μέθοδος main. Μεταβλητές

Τα πακέτα αποθηκεύονται στον πίνακα **sinput** ο οποίος παραμένει αναλλοίωτος καθ' όλη τη διάρκεια του προγράμματος, για να μπορεί να χρησιμοποιηθεί σε όλους τους αλγορίθμους χρονοπρογραμματισμού, χωρίς να χρειαστεί να δοθεί εκ νέου. Με τη μέθοδο **Character.getNumericValue** της Java, ο δεύτερος χαρακτήρας του κάθε πακέτου που δίνεται και δηλώνει τον προορισμό του, μετατρέπεται σε ακέραιο αριθμό. Αφού ολοκληρωθεί ο πίνακας sinput με τα πακέτα, δημιουργείται ο πίνακας των πακέτων εξόδου **soutput** ο οποίος είναι όλος κενός, περιέχει δηλαδή μόνο πακέτα X9 (**Εικόνα 5.22**).

```
    if (choice == 1) {
        System.out.println("Δώστε τα πακέτα κατά γραμμή, αφήνοντας κενό ανάμεσά τους");
        System.out.println("Χρησιμοποιήστε έναν χαρακτήρα και έναν αριθμό για κάθε πακέτο");
        System.out.println("Για κενό πακέτο δώστε το χαρακτήρα 'X' και τον αριθμό '9'");
        System.out.println();
        for (int i = 0; i < N; i++) {
           System.out.print("Δώστε την "+(i+1)+"η γραμμή πακέτων: ");
           for (int j = 0; j < N; j++) {
              token = kb.next();
              sinput[i][j] = new Item(token.charAt(0), Character.getNumericValue(token.charAt(1)));
           }
        }
        System.out.println();
     }else{
        sinput[0][0] = new Item('X', 9); sinput[0][1] = new Item('A', 3); sinput[0][2] = new Item('B', 0);
sinput[0][3] = new Item('C', 2);
        sinput[1][0] = new Item('X', 9); sinput[1][1] = new Item('D', 2); sinput[1][2] = new Item('E', 1);
sinput[1][3] = new Item('Z', 3);
        sinput[2][0] = new Item('X', 9); sinput[2][1] = new Item('X', 9); sinput[2][2] = new Item('H', 3);
sinput[2][3] = new Item('U', 0);
        sinput[3][0] = new Item('X', 9); sinput[3][1] = new Item('I', 3); sinput[3][2] = new Item('K', 2);
sinput[3][3] = new Item('L', 1);
     }

for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
           soutput[i][j] = new Item('X', 9);
        }
}
```

Εικόνα 5.22: Πρώτο μενού επιλογής

Στη συνέχεια εμφανίζεται ένα δεύτερο μενού επιλογής στο οποίο ο χρήστης επιλέγει ποιον αλγόριθμο χρονοπρογραμματισμού από τους πέντε θέλει να εκτελεστεί ή αν επιθυμεί την έξοδο από την εφαρμογή (**Εικόνα 5.23**).

```
System.out.println("Επιλέξτε τον αλγόριθμο χρονοπρογραμματισμού:");
System.out.println("1. PM");
System.out.println("2. PIM");
System.out.println("3. IRRM");
System.out.println("4. ISLIP");
System.out.println("5. DRRM");
System.out.println("6. EXIT");
choice = kb.nextInt();
System.out.println();
```

Εικόνα 5.23: Δεύτερο μενού επιλογής

Ξεκινάει ένας βρόχος επανάληψης ο οποίος συνεχίζεται μέχρι να επιλεγεί η έξοδος και δίνει τη δυνατότητα στην εφαρμογή να εκτελέσει όσους αλγορίθμους επιθυμεί ο χρήστης, με το ίδιο σύνολο πακέτων μεταγωγής (**Εικόνα 5.24**). Στο ξεκίνημα του βρόχου, δημιουργούνται δύο νέοι πίνακες **input** και **output** οι οποίοι αντιγράφουν τους sinput και soutput, ούτως ώστε να μεταβάλλονται κατά τη διάρκεια εκτέλεσης του αλγορίθμου, χωρίς να μεταβάλλονται οι αρχικοί και να μπορούν έτσι να χρησιμοποιηθούν ξανά στον επόμενο αλγόριθμο που θα επιλεγεί. Αρχικοποιούνται επίσης και όλοι οι υπόλοιποι πίνακες που θα χρησιμοποιηθούν, μηδενίζοντας όλα τα στοιχεία τους: οι **arr1**, **arr2**, **arr3** και **arr4** που είναι αντίστοιχα οι πίνακες αιτήσεων, κρατήσεων ή παραχωρήσεων, αποδοχών και πλήρους αντιστοίχησης, καθώς και οι πίνακες των δεικτών αποδοχής και παραχώρησης **a** και **b**.

Το επόμενο βήμα είναι να δημιουργηθούν τα δύο παράθυρα γραφικής απεικόνισης (**new JFrame**) (**Εικόνα 5.25**). Το πρώτο απεικονίζει το μεταγωγέα και τα πακέτα στην είσοδο και την έξοδό του, καθώς και τη μεταφορά των πακέτων από τη μία πλευρά στην άλλη. Το άλλο εμφανίζει τα πλαίσια των αιτήσεων, κρατήσεων ή παραχωρήσεων και αποδοχών με τα αντίστοιχα βελάκια τους, καθώς και τους δείκτες παραχώρησης και αποδοχής, όπου αυτοί απαιτούνται. Στα δύο παράθυρα γίνονται οι αρχικές ρυθμίσεις που αφορούν τη θέση τους (**setLocation**), το μέγεθός τους (**setSize**), τον τίτλο τους κ.λ.π.

```
while ((choice>=1)&&(choice<=5)){

        Item[][] input = new Item[N][N];
        Item[][] output = new Item[N][M];

        for (int i = 0; i < N; i++) {
           for (int j = 0; j < N; j++) {
              input[i][j] = new Item(sinput[i][j].head, sinput[i][j].dest);
              arr1[i][j]=0;arr2[i][j]=0;arr3[i][j]=0;arr4[i][j]=0;
           }
        }

        for (int i = 0; i < N; i++) {
           for (int j = 0; j < M; j++) {
              output[i][j] = new Item(soutput[i][j].head, soutput[i][j].dest);
           }
           a[i] = 0;
           b[i] = 0;
        }

        switch (choice) {
           case 1:
              token = "PM";
              break;
           case 2:
              token = "PIM";
              break;
           case 3:
              token = "IRRM";
              break;
           case 4:
              token = "ISLIP";
              break;
           default:
              token = "DRRM";
              break;
        }
```

Εικόνα 5.24: Επανάληψη και αρχικοποίηση πινάκων

```
        JFrame window = new JFrame(token);
        window.setLayout(null);
        window.setSize(820, 300);
        window.setLocation(0, 0);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);

        JFrame window2 = new JFrame(token);
        window2.setSize(820, 300);
        window2.setLocation(0, 300);
        window2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window2.setVisible(true);
```

Εικόνα 5.25: Δημιουργία παραθύρων γραφικών

```
        for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
            lb[i][j] = new JLabel();
            lb[i][j].setSize(A,A);
            lb[i][j].setLocation((j+1)*(A+2),(i+1)*(A+2));
            switch (input[i][j].dest) {
               case 0:
                  lb[i][j].setBackground(Color.yellow);
                  break;
               case 1:
                  lb[i][j].setBackground(Color.blue);
                  break;
               case 2:
                  lb[i][j].setBackground(Color.red);
                  break;
               case 3:
                  lb[i][j].setBackground(Color.green);
                  break;
               default:
                  lb[i][j].setBackground(Color.gray);
                  break;
            }
            lb[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
            lb[i][j].setOpaque(true);
            String text="";
            if (input[i][j].dest!=9){
               text = String.valueOf(input[i][j].head) + String.valueOf(input[i][j].dest);
            }
            lb[i][j].setText(text);
            lb[i][j].setHorizontalAlignment(0);
            lb[i][j].setVisible(true);
            window.add(lb[i][j]);
          }
        }
```

Εικόνα 5.26: Δημιουργία πακέτων εισόδου

Στη διπλή επανάληψη που ακολουθεί, ο σκοπός είναι να κατασκευαστούν ένα-ένα όλα τα πακέτα στο γραφικό περιβάλλον και να μπουν στην αρχική τους θέση στο παράθυρο, σύμφωνα με τις πληροφορίες του πίνακα input που είδαμε προηγουμένως (**Εικόνα 5.26**). Αφού λοιπόν δημιουργηθεί ένα καινούριο αντικείμενο τύπου ετικέτας (**new JLabel**), ορίζουμε το μέγεθος (**setSize**) και τη θέση του στο παράθυρο (**setLocation**). Στη συνέχεια καθορίζεται το χρώμα του (**setBackground**) με βάση τον προορισμό του, έτσι ώστε όλα τα πακέτα που κατευθύνονται προς την ίδια έξοδο να έχουν το ίδιο χρώμα. Ορίζονται επίσης το χρώμα περιγράμματος (**setBorder**), η ορατότητα (**setVisible**), το ανάγλυφο (**setOpaque**), το κείμενο της ετικέτας (**setText**) και η στοίχισή του

(**setHorizontalAlignment**). Τα κενά πακέτα γίνονται χρώματος γκρι και δεν περιέχουν κείμενο. Τέλος, τοποθετείται μέσα στο γραφικό παράθυρο (**window.add**). Με τον ίδιο ακριβώς τρόπο δημιουργούνται και οι κενές θέσεις στις εξόδους του μεταγωγέα (**Εικόνα 5.27**).

```
JLabel[][] lb2 = new JLabel[N][M];

for (int i = 0; i < N; i++) {
   for (int j = 0; j < M; j++) {
      lb2[i][j] = new JLabel();
      lb2[i][j].setSize(A,A);
      lb2[i][j].setLocation((j+1)*(A+2)+D,(i+1)*(A+2));
      lb2[i][j].setBackground(Color.GRAY);
      lb2[i][j].setOpaque(true);
      lb2[i][j].setVisible(true);
      window.add(lb2[i][j]);
   }
}
```

Εικόνα 5.27: Δημιουργία θέσεων εξόδου

```
JButton jb = new JButton("ΕΝΑΡΞΗ");
jb.setSize(100, 40);
jb.setLocation(260, 200);
jb.setVisible(true);
window.add(jb);

jb.addActionListener(new ActionListener(){
   @Override
   public void actionPerformed(ActionEvent e) {
      if (jb.getText().equals("ΤΕΛΟΣ")) {
         window.dispose();
         window2.dispose();
      } else {
         cont = true;
      }
   }
});
```

Εικόνα 5.28: Δημιουργία κουμπιού ελέγχου

Μένει τώρα να δημιουργήσουμε ένα κουμπί στο γραφικό περιβάλλον για να ελέγχουμε κατά βούληση την κίνηση των πακέτων ανά χρονοθυρίδα (**Εικόνα 5.28**). Όπως έγινε και στις ετικέτες, αφού δημιουργηθεί το κουμπί (**new JButton**), ορίζουμε τις ιδιότητές του (setSize, setLocation, setVisible) και το τοποθετούμε

στο παράθυρο (window.add). Το κουμπί εμφανίζει αρχικά το όνομα «ΕΝΑΡΞΗ», κατά τη διάρκεια όλης της διαδικασίας φέρει το όνομα «ΕΠΟΜΕΝΟ» και όταν αυτή τελειώσει με τη μεταγωγή όλων των πακέτων, αναγράφει το όνομα «ΤΕΛΟΣ». Το τελευταίο που μένει να ορίσουμε είναι την αντίδραση του κουμπιού κατά το πάτημα του ποντικιού. Για το λόγο τούτο δημιουργούμε ένα νέο **ActionListener** (addActionListener, new ActionListener). Σε αυτό ορίζουμε ότι όταν το κουμπί αναγράφει τη λέξη «ΤΕΛΟΣ» πρέπει να κλείσει τα δύο γραφικά παράθυρα (**window.dispose**), ώστε να επιλεγεί ο επόμενος αλγόριθμος ή η έξοδος από το πρόγραμμα, ενώ στις άλλες δύο περιπτώσεις απλώς δίνει την τιμή true στην global μεταβλητή **cont** που είδαμε στην αρχή του κεφαλαίου, ώστε να συνεχιστεί η επανάληψη και η διαδικασία.

Σε αυτό το σημείο έχει ολοκληρωθεί η κατασκευή του γραφικού περιβάλλοντος. Δημιουργούμε ένα νέο αντικείμενο της κλάσης που τώρα σχεδιάζουμε και το προσθέτουμε στο δεύτερο παράθυρο γραφικών ώστε να κληθεί για πρώτη φορά η paintComponent. Ορίζουμε μερικές ακόμα μεταβλητές, όπως ο πίνακας **array** όπου θα αποθηκευτούν οι αιτήσεις και η συνεχής μετατροπή τους και η μεταβλητή **pos** που θα εκφράζει τον αριθμό της τρέχουσας χρονοθυρίδας (**Εικόνα 5.29**).

```
GCHRONO dr = new GCHRONO();
window2.add(dr);

int[][] array = new int[N][N];
int pos = 0;

System.out.println("Αρχική μορφή:");
System.out.println("-------------------------");
print_switch(input, output, pos);
```

Εικόνα 5.29: Δημιουργία αντικειμένου GCHRONO. Μεταβλητές και αρχικές εκτυπώσεις.

Γίνονται οι απαραίτητες πρώτες εκτυπώσεις και ξεκινάει η βασική επανάληψη της εφαρμογής για να πραγματοποιηθεί η διαδικασία. Η επανάληψη ελέγχεται από την τιμή της **endOfTable** που είδαμε προηγουμένως κι έχει να κάνει με το αν παραμένουν ή όχι πακέτα στις εισόδους του μεταγωγέα. Αμέσως μετά γίνεται false η μεταβλητή **cont** και η διαδικασία μπαίνει σε αναμονή, μέχρις ότου πατηθεί το κουμπί και ξαναγίνει η μεταβλητή true. Με γνώμονα την επιλογή του

χρήστη (**choice**) στο δεύτερο μενού, καλούνται οι κατάλληλες μέθοδοι όπως καθορίζονται από τον αλγόριθμο χρονοπρογραμματισμού που επιλέχτηκε. Με την **updateTable** γίνεται, όπως είδαμε, η κίνηση των πακέτων και η ανανέωση των πινάκων εισόδου και εξόδου, ενώ με την **repaint** της Java ανανεώνεται το γραφικό παράθυρο των πλαισίων με τις νέες σχεδιαστικές λεπτομέρειες. Πριν επιστρέψουμε στην αρχή του βρόχου, αν έχουν «αδειάσει» οι είσοδοι το κουμπί παίρνει το όνομα «ΤΕΛΟΣ», ενώ παραμένει «ΕΠΟΜΕΝΟ» στην αντίθετη περίπτωση. Η μεταβλητή των χρονοθυρίδων **pos** αυξάνεται κατά ένα (**Εικόνα 5.30**).

```
while (!endOfTable(input)) {
      cont = false;
      while (!cont) {}
      System.out.println();
      System.out.printf("%dη Χρονοθυρίδα:\n", pos+1);
      System.out.println("-------------------------");
      window2.setTitle ((pos+1)+"η ΧΡΟΝΟΘΥΡΙΔΑ");

      switch (choice) {
         case 1:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array);
            copyArrays(array, arr2);
            acceptance(array);
            copyArrays(array, arr3);
            break;
         case 2:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array);
            copyArrays(array, arr2);
            acceptance(array);
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
         case 3:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array, b);
            print_point(b, 'b');
            copyArrays(array, arr2);
            acceptance(array, a);
            print_point(a, 'a');
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
         case 4:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array, b);
            copyArrays(array, arr2);
            acceptance(array, a, b);
            print_point(a, 'a');
            print_point(b, 'b');
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
         default:
            request(array, input, a);
            print_point(a, 'a');
            copyArrays(array, arr1);
            reservation(array, b);
            print_point(b, 'b');
            copyArrays(array, arr2);
            break;
      }
      dr.repaint();
      updateTable(array, input, output, pos);
      if (!endOfTable(input)) {
         jb.setText("ΕΠΟΜΕΝΟ");
         print_switch(input, output, pos+1);
      }else {
         jb.setText("ΤΕΛΟΣ");
      }
      pos++;
} //while
```

Εικόνα 5.30: Βασική επανάληψη

Όταν τελειώσουν τα πακέτα και ολοκληρωθεί η επανάληψη, γίνονται οι τελικές εκτυπώσεις, κλείνουν τα παράθυρα των γραφικών με το πάτημα του κουμπιού «ΤΕΛΟΣ» και ο χρήστης επιστρέφει στην κονσόλα των αποτελεσμάτων, με το μενού ξανά στην οθόνη, για να επιλέξει τον επόμενο αλγόριθμο ή την έξοδο από την εφαρμογή (**Εικόνα 5.31**).

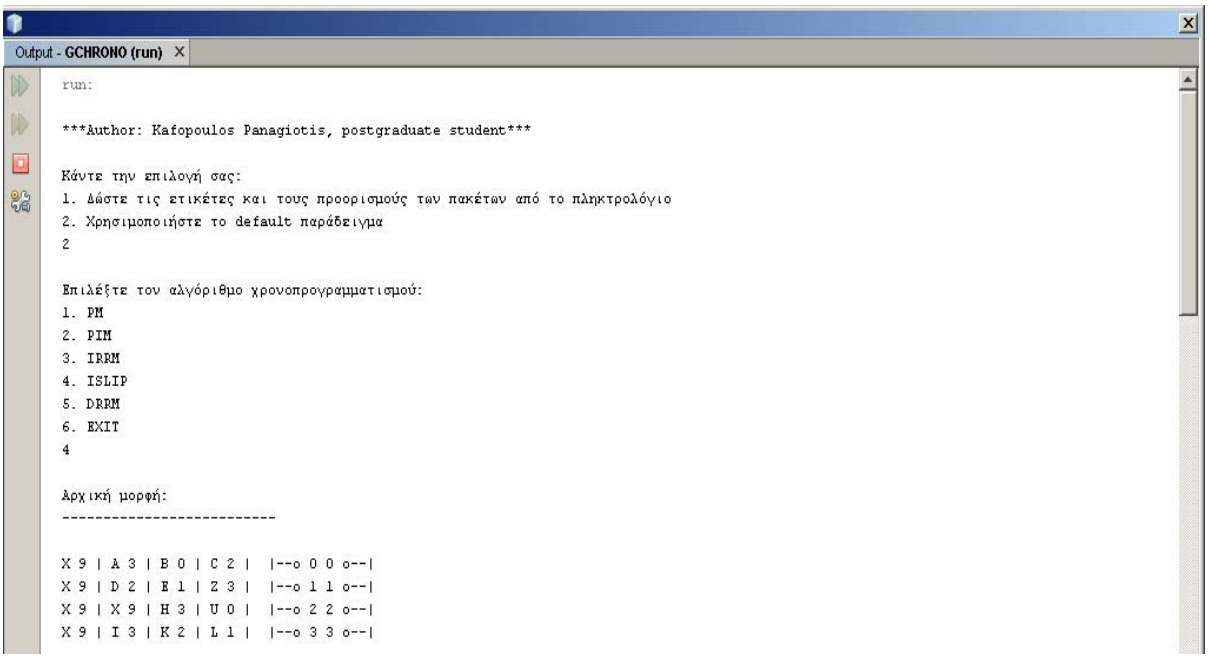
```
        System.out.println();
        System.out.println("Τελική μορφή:");
        System.out.println("-------------------------");
        print_switch(input, output, pos);
        System.out.println();

        System.out.println("Επιλέξτε τον αλγόριθμο χρονοπρογραμματισμού:");
        System.out.println("1. PM");
        System.out.println("2. PIM");
        System.out.println("3. IRRM");
        System.out.println("4. ISLIP");
        System.out.println("5. DRRM");
        System.out.println("6. EXIT");
        choice = kb.nextInt();
        System.out.println();
    } //first while
    System.out.println("***Author: Kafopoulos Panagiotis, postgraduate student***");
    System.out.println();
```

Εικόνα 5.31: Τελικές εκτυπώσεις

6. Παράδειγμα εκτέλεσης κώδικα

Στο κεφάλαιο αυτό θα δούμε ένα ολοκληρωμένο παράδειγμα εκτέλεσης της εφαρμογής, με τη βοήθεια εκτυπώσεων οθόνης (screenshots).

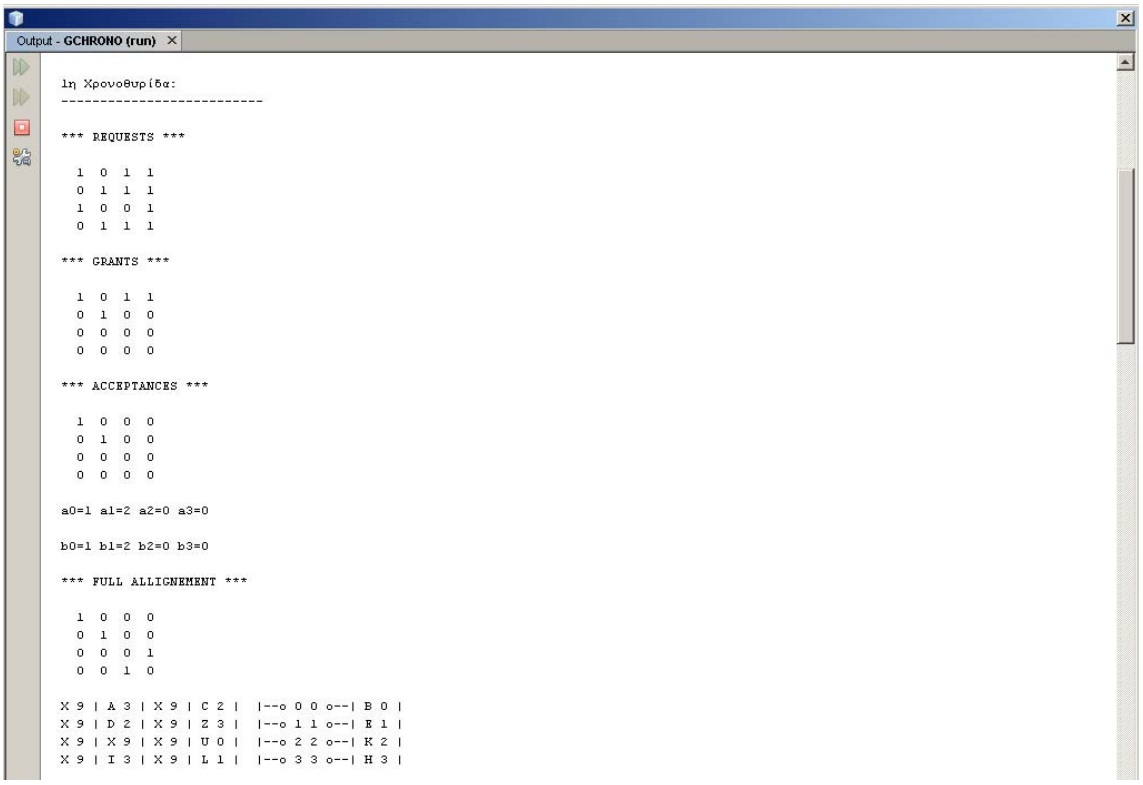
Ξεκινάμε την εφαρμογή και στο πρώτο μενού επιλέγουμε το default παράδειγμα. Από τους πέντε αλγορίθμους χρονοπρογραμματισμού επιλέγουμε αυτόν που επιθυμούμε. Εδώ επιλέξαμε τον αλγόριθμο 4, δηλαδή τον iSLIP. Εμφανίζεται η αρχική μορφή του πίνακα των πακέτων εισόδου-εξόδου (**Εικόνα 6.1**).



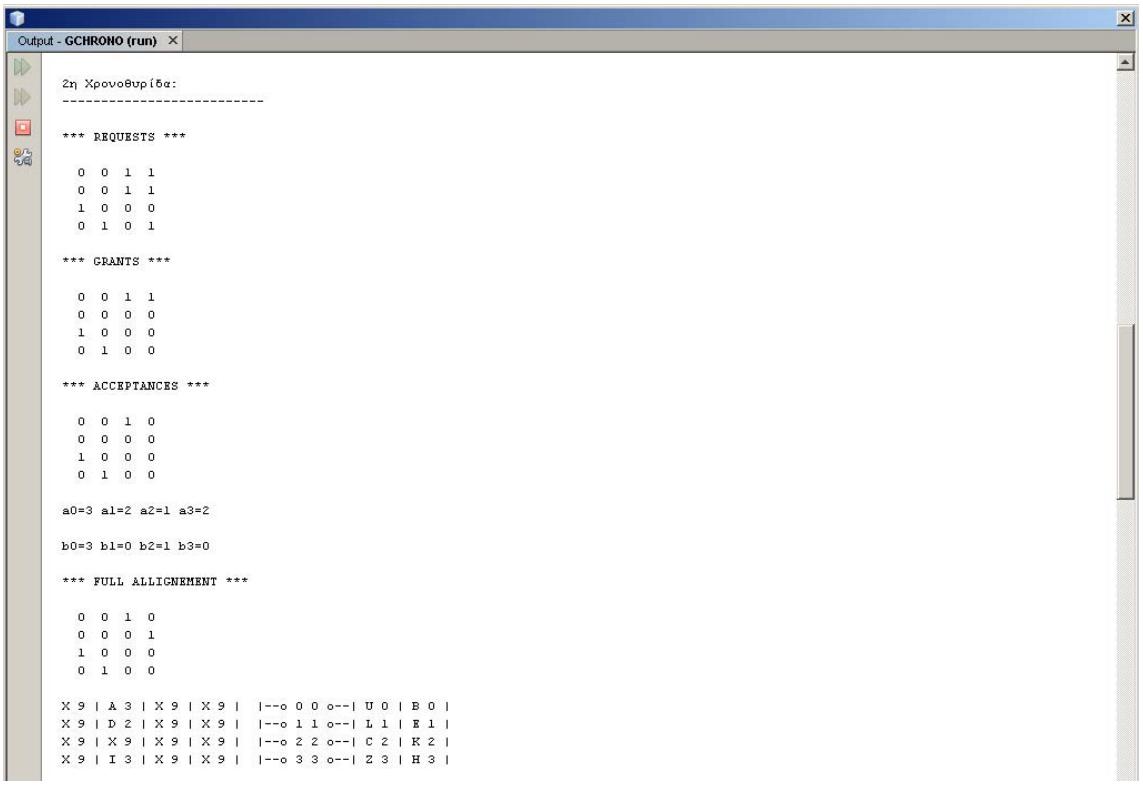
Εικόνα 6.1: Επιλογή default παραδείγματος

Κατά την πρώτη χρονοθυρίδα, εμφανίζονται στην οθόνη ο πίνακας των αιτήσεων, των παραχωρήσεων, των αποδοχών, οι δείκτες παραχώρησης και αποδοχής, ο πίνακας πλήρους αντιστοίχισης και ο διαμορφωμένος πίνακας των πακέτων (**Εικόνα 6.2**).

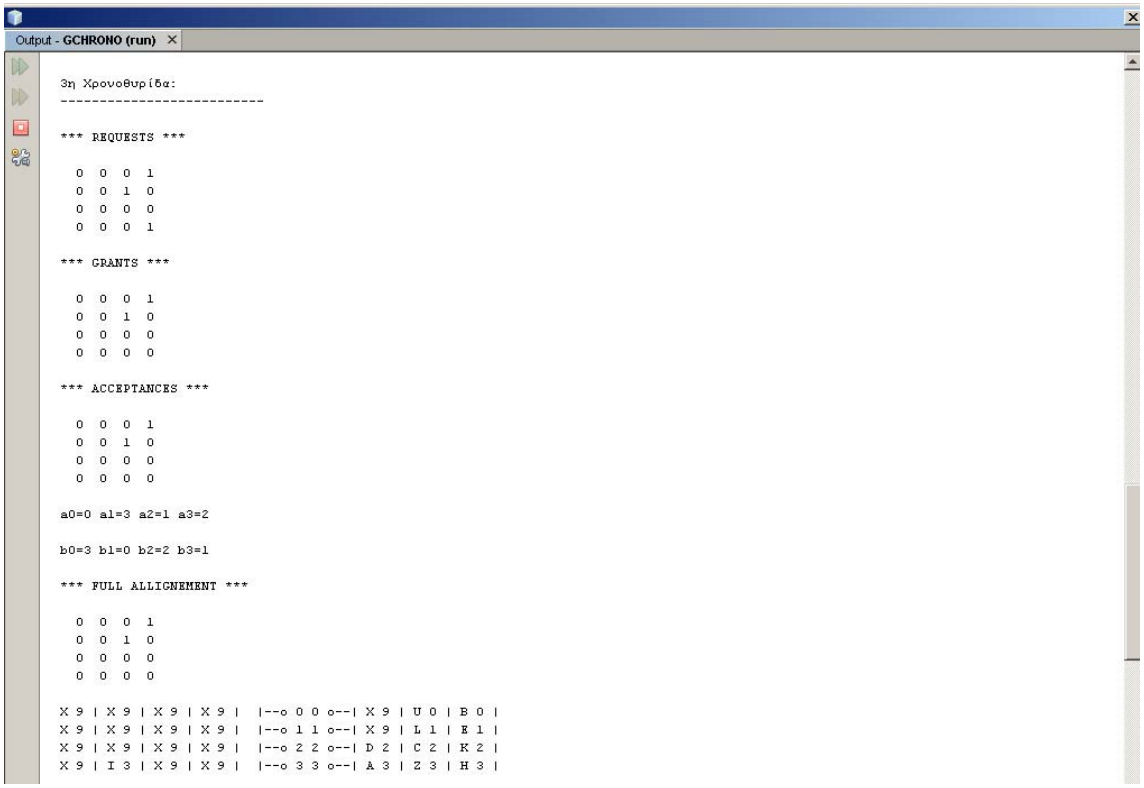
Όμοια εμφανίζονται και για τις υπόλοιπες χρονοθυρίδες που στο συγκεκριμένο παράδειγμα είναι τέσσερις (**Εικόνα 6.3**) (**Εικόνα 6.4**) (**Εικόνα 6.5**).



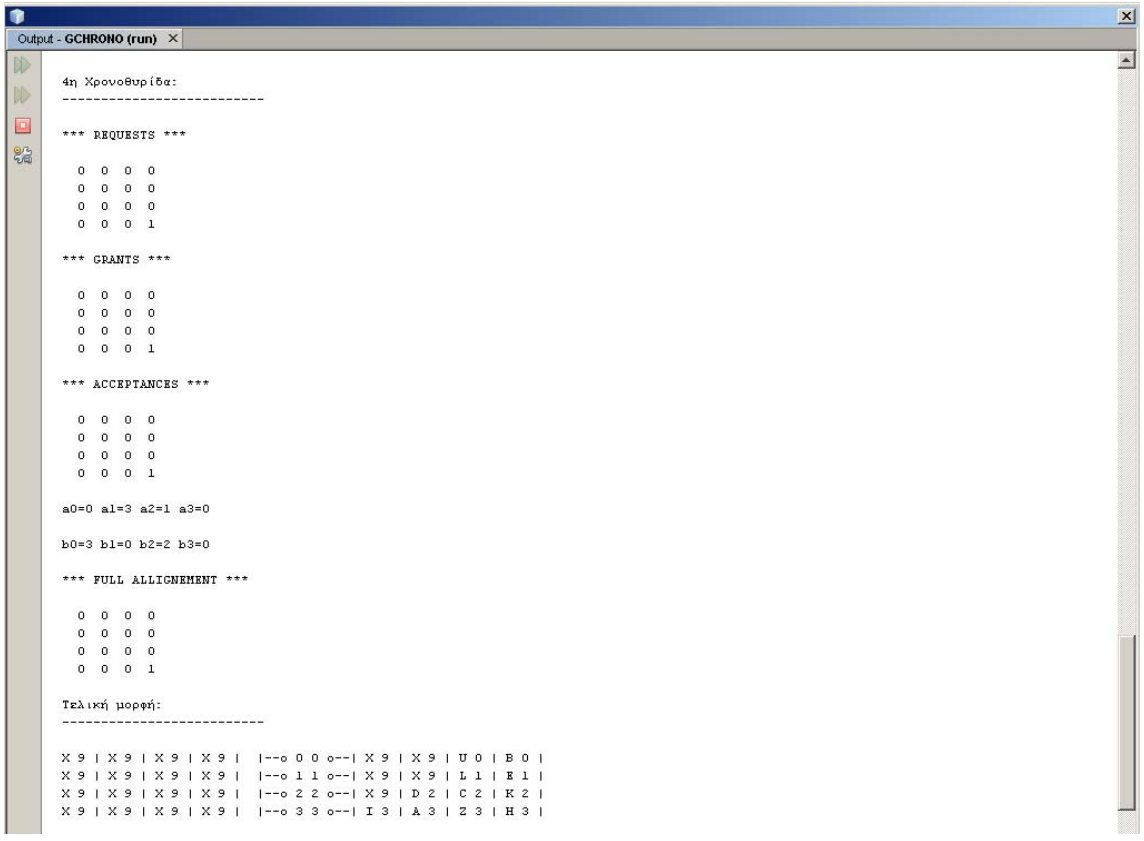
Εικόνα 6.2: Πρώτη χρονοθυρίδα



Εικόνα 6.3: Δεύτερη χρονοθυρίδα

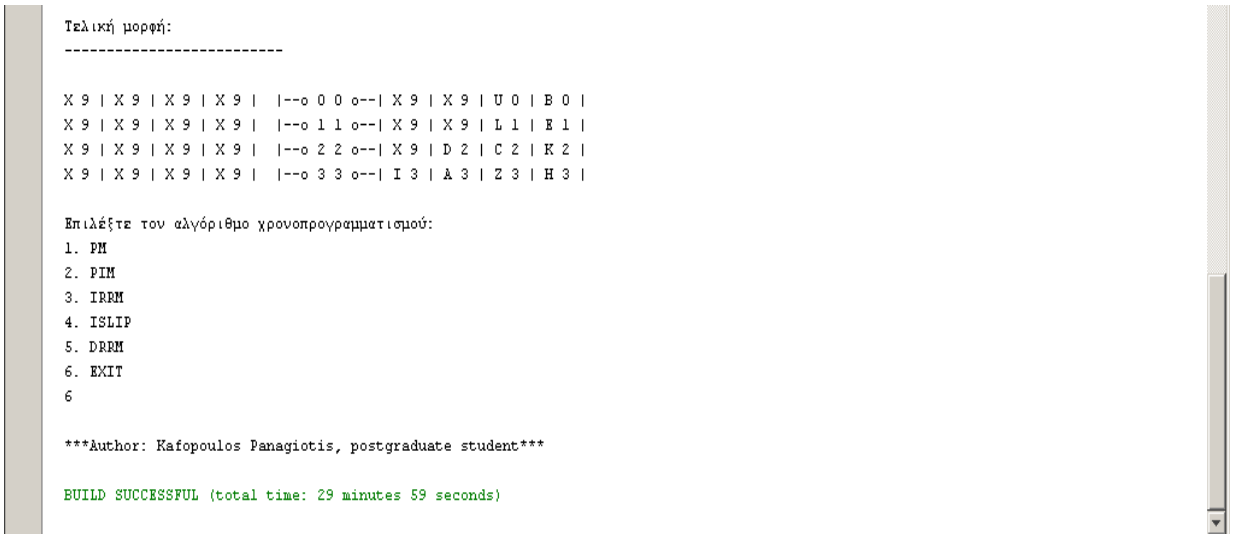


Εικόνα 6.4: Τρίτη χρονοθυρίδα



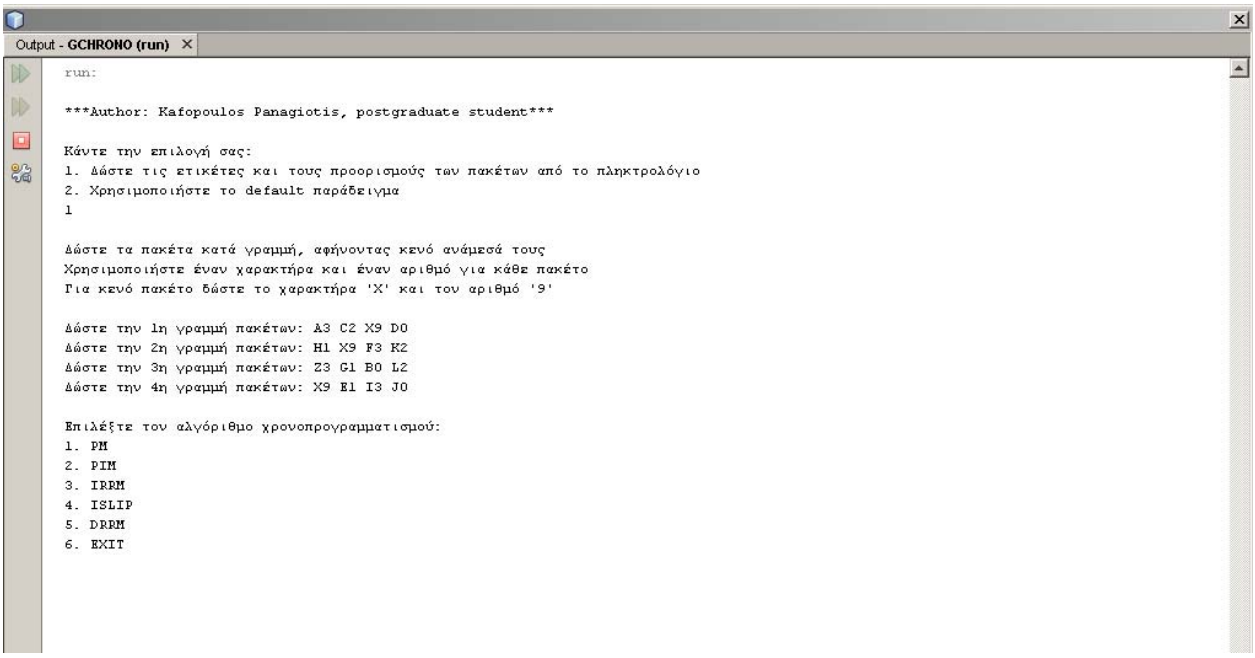
Εικόνα 6.5: Τέταρτη χρονοθυρίδα

Αφού ολοκληρωθεί η μετακίνηση όλων των πακέτων και εμφανιστεί ο πίνακας στην τελική του μορφή, επιστρέφουμε στο μενού και επιλέγουμε έναν άλλον αλγόριθμο για το ίδιο σύνολο πακέτων μεταγωγής ή την έξοδο από την εφαρμογή (**Εικόνα 6.6**).



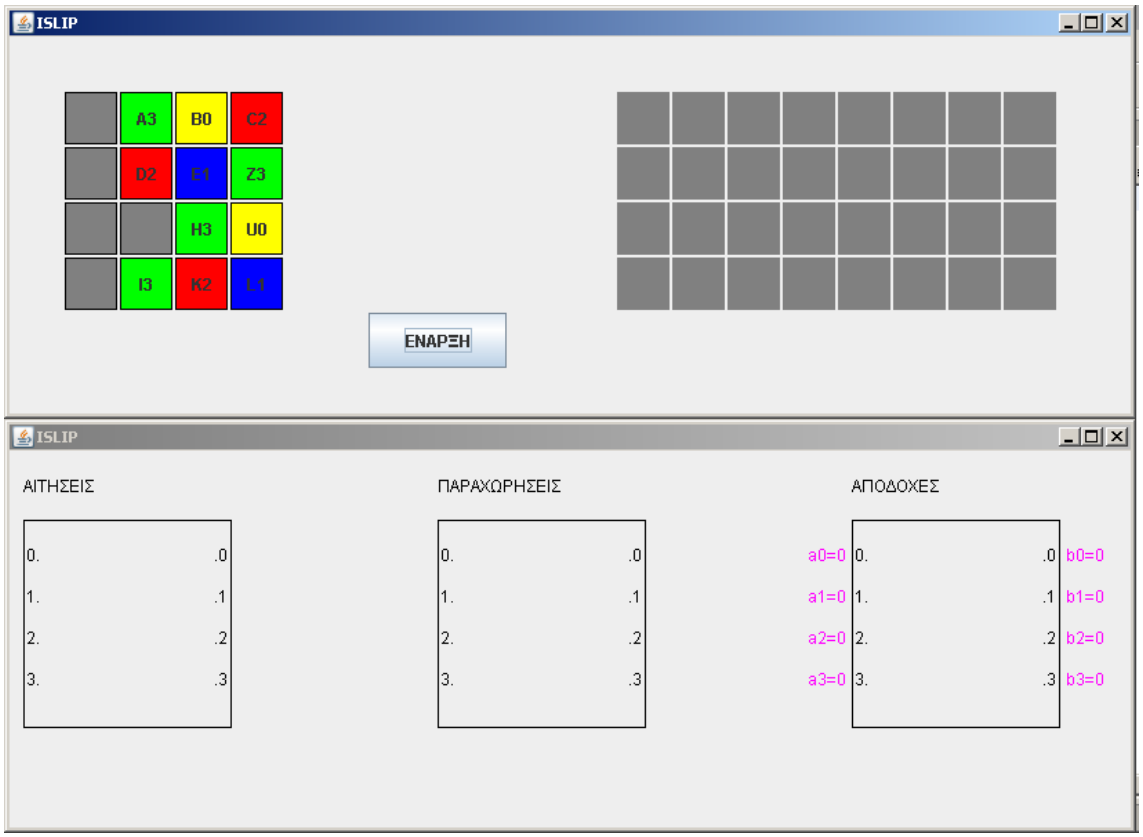
Εικόνα 6.6: Τελικός πίνακας και επιστροφή στο μενού

Στην περίπτωση που στο αρχικό μενού επιλέξουμε την εισαγωγή των πακέτων από το πληκτρολόγιο, τα πακέτα πρέπει να δοθούν με τον παρακάτω τρόπο (**Εικόνα 6.7**).



Εικόνα 6.7: Εισαγωγή πακέτων από το πληκτρολόγιο

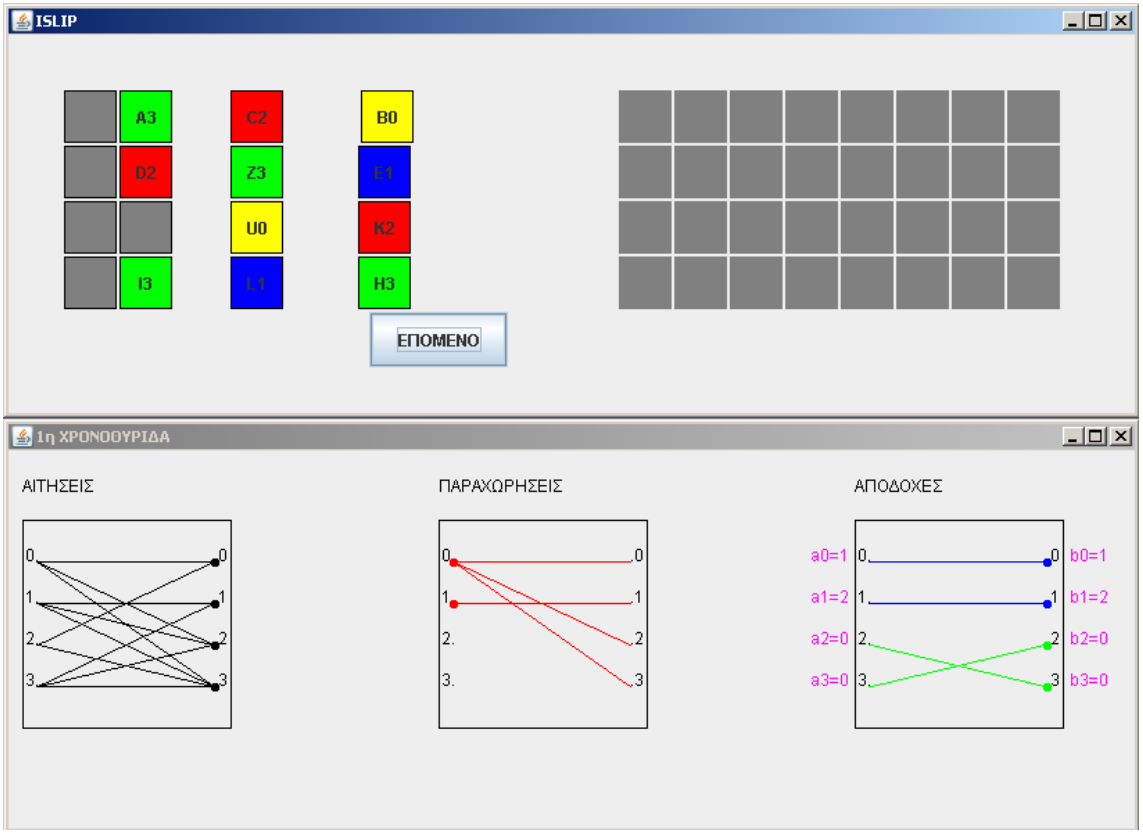
Βλέπουμε την ίδια διαδικασία και στο γραφικό περιβάλλον. Στο πρώτο παράθυρο εμφανίζεται ο μεταγωγέας και τα πακέτα που μετακινούνται από τις εισόδους προς τις εξόδους με βάση τον αλγόριθμο που έχει επιλεγεί. Εμφανίζεται επίσης και το κουμπί ελέγχου της διαδικασίας. Στο δεύτερο παράθυρο είναι τα πλαίσια των αιτήσεων, των κρατήσεων ή παραχωρήσεων και των αποδοχών με τα αντίστοιχα βελάκια. Στο τρίτο πλαίσιο, των αποδοχών, με πράσινο χρώμα εμφανίζονται τα βελάκια της πλήρους αντιστοίχισης (**Εικόνα 6.8**).



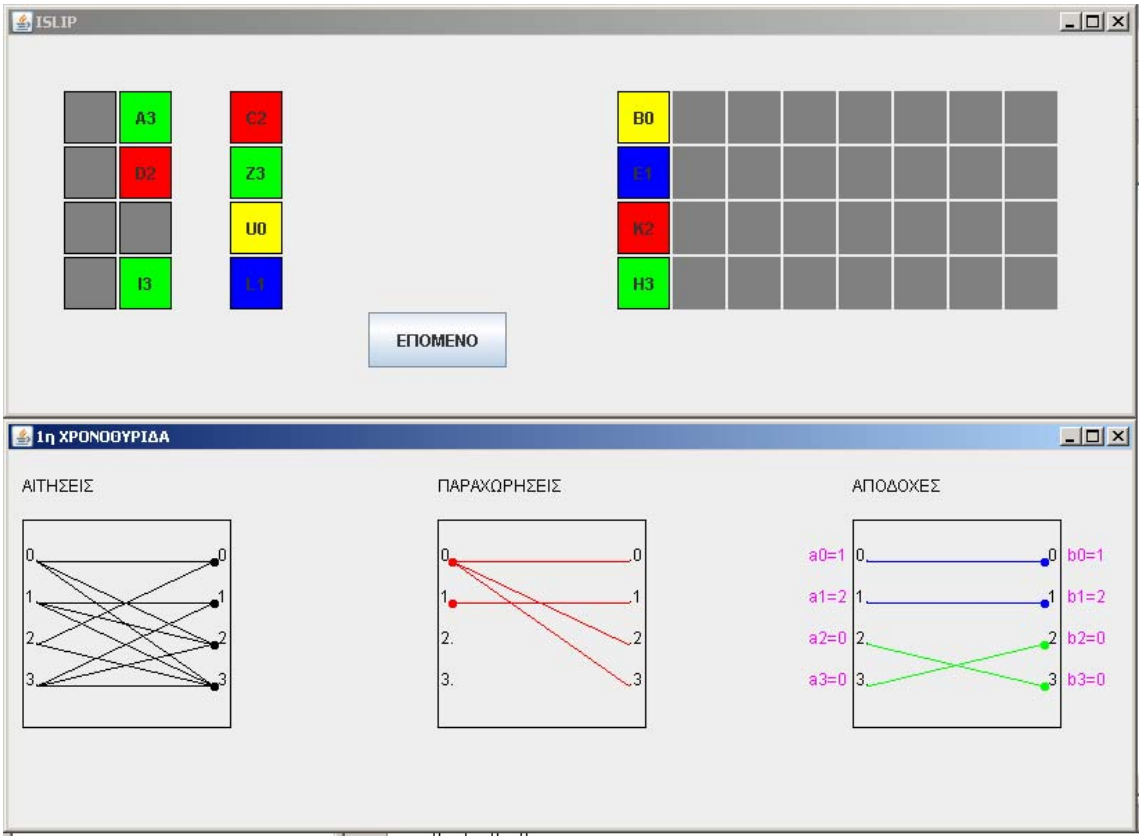
Εικόνα 6.8: Γραφικό περιβάλλον

Πατάμε το κουμπί «ΕΝΑΡΞΗ» και ξεκινάει η πρώτη χρονοθυρίδα (**Εικόνα 6.9**). Αμέσως το κουμπί μεταλλάσσεται σε «ΕΠΟΜΕΝΟ» και με αυτό ξεκινάμε την κάθε μια από τις επόμενες χρονοθυρίδες (**Εικόνα 6.10**) (**Εικόνα 6.11**) (**Εικόνα 6.12**).

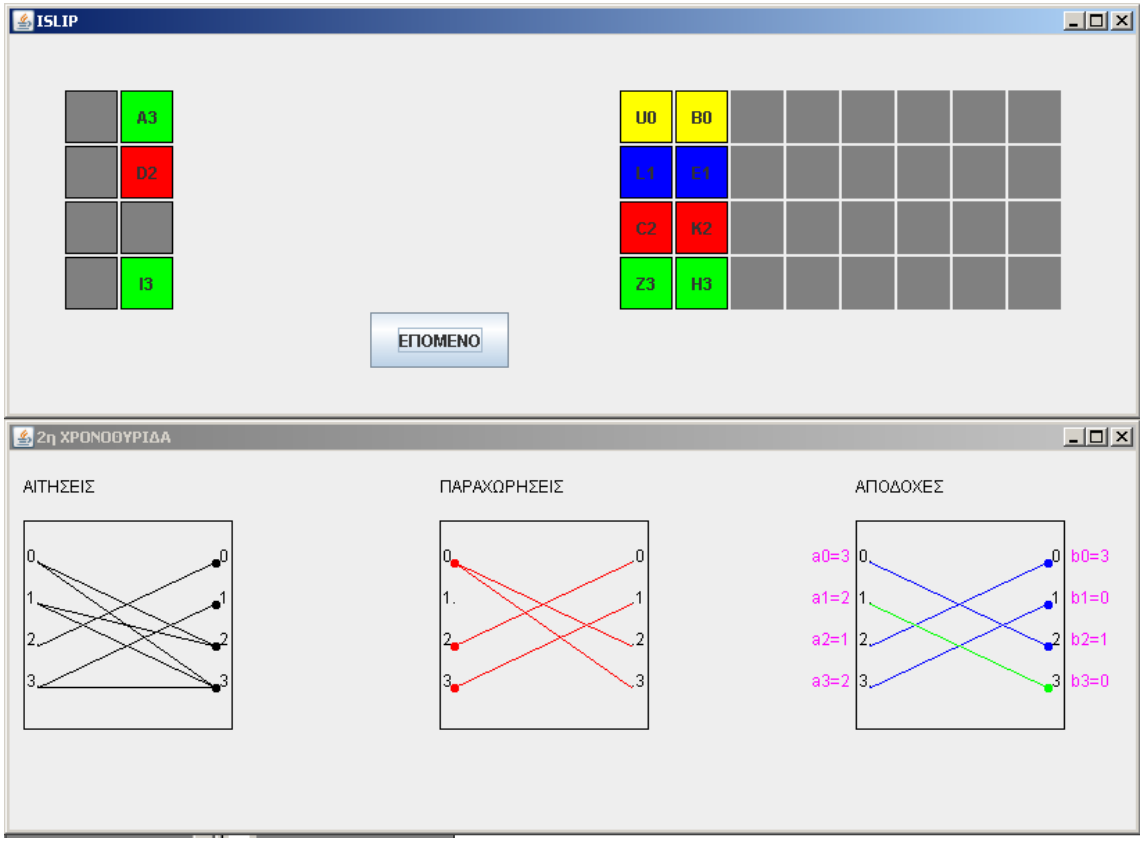
Οι χρονοθυρίδες τελειώνουν και το κουμπί μετονομάζεται σε «ΤΕΛΟΣ». Πατώντας το, κλείνουν τα παράθυρα των γραφικών και επιστρέφουμε στην κονσόλα και στο μενού επιλογής (**Εικόνα 6.13**).



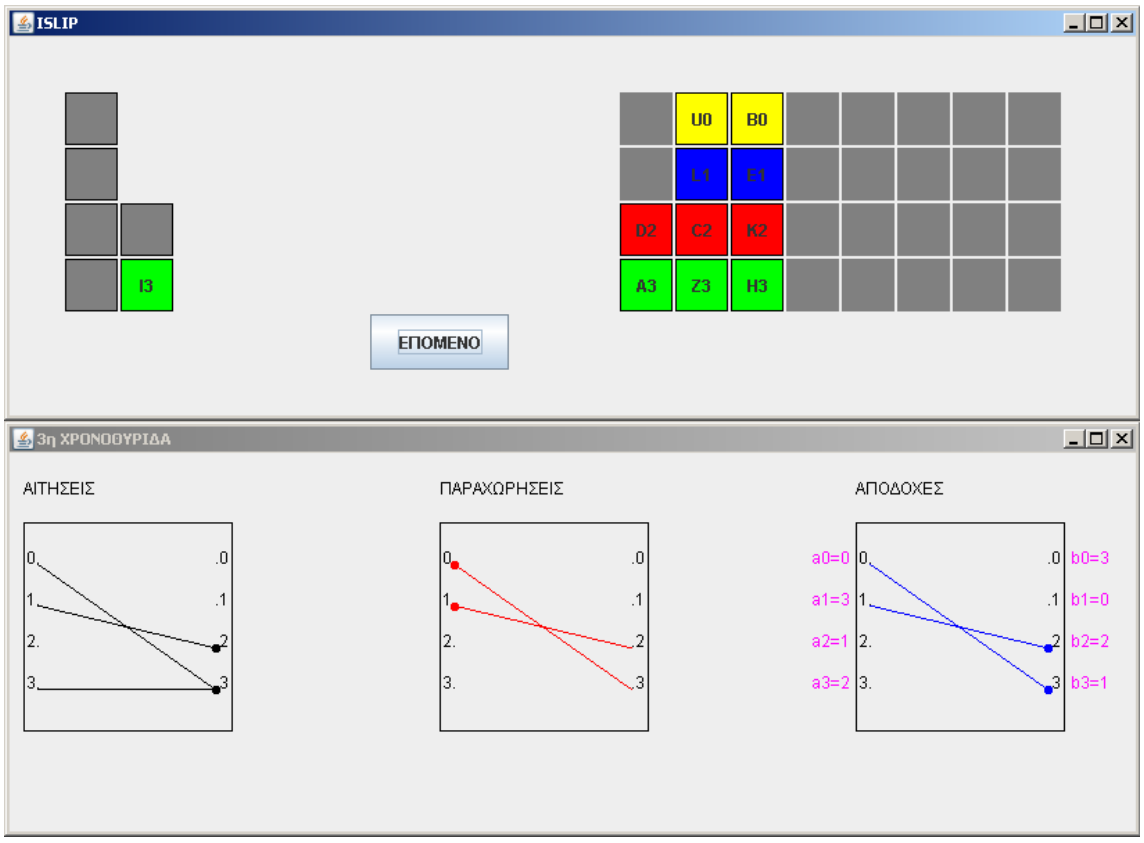
Εικόνα 6.9: Μετακίνηση πακέτων



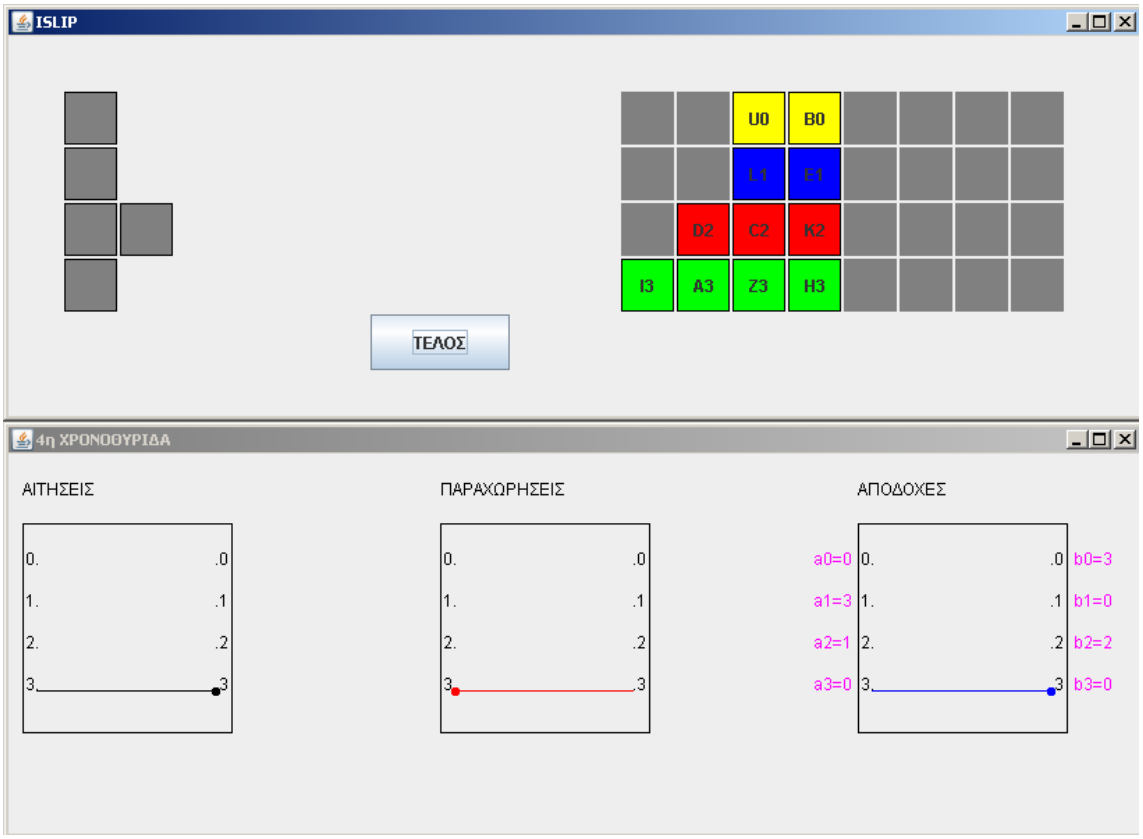
Εικόνα 6.10: Πρώτη χρονοθυρίδα



Εικόνα 6.11: Δεύτερη χρονοθυρίδα



Εικόνα 6.12: Τρίτη χρονοθυρίδα



Εικόνα 6.13: Τέταρτη χρονοθυρίδα

ΠΑΡΑΡΤΗΜΑ

Παρατίθεται ολόκληρος ο κώδικας της εφαρμογής σε **JAVA**:

```
package gchrono;

import AppPackage.AnimationClass;
import java.util.Scanner;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JComponent;

class Item {
   char head;
   int dest;

   public Item (char head, int dest) {
      this.head = head;
      this.dest = dest;
   }
}

public class GCHRONO extends JComponent {

   static final int N = 4;
   static final int M = 8;

   static final int A = 38;
   static final int D = 400;
```

```
static JLabel[][] lb = new JLabel[N][N];

static int[][] arr1 = new int[N][N];
static int[][] arr2 = new int[N][N];
static int[][] arr3 = new int[N][N];
static int[][] arr4 = new int[N][N];

static int[] a = new int[N];
static int[] b = new int[N];

static int choice;

static boolean cont = true;

public static void main(String[] args) {

   Item[][] sinput = new Item[N][N];
   Item[][] soutput = new Item[N][M];

   Scanner kb = new Scanner(System.in);
   String token;

   System.out.println();
   System.out.println("***Author: Kafopoulos Panagiotis, postgraduate student***");
   System.out.println();
   System.out.println("Κάντε την επιλογή σας:");
   System.out.println("1. Δώστε τις ετικέτες και τους προορισμούς των πακέτων από το
                       πληκτρολόγιο");
   System.out.println("2. Χρησιμοποιήστε το default παράδειγμα");
   choice = kb.nextInt();
   System.out.println();

   if (choice == 1) {
      System.out.println("Δώστε τα πακέτα κατά γραμμή, αφήνοντας κενό ανάμεσά
                          τους");
      System.out.println("Χρησιμοποιήστε έναν χαρακτήρα και έναν αριθμό για κάθε
                          πακέτο");
```

```
        System.out.println("Για κενό πακέτο δώστε το χαρακτήρα 'X' και τον αριθμό '9'");
        System.out.println();
        for (int i = 0; i < N; i++) {
            System.out.print("Δώστε την "+(i+1)+"η γραμμή πακέτων: ");
            for (int j = 0; j < N; j++) {
                token = kb.next();
                sinput[i][j] = new Item(token.charAt(0),
                                Character.getNumericValue(token.charAt(1)));
            }
        }
        System.out.println();
    }else{
        sinput[0][0] = new Item('X', 9); sinput[0][1] = new Item('A', 3);
        sinput[0][2] = new Item('B', 0); sinput[0][3] = new Item('C', 2);
        sinput[1][0] = new Item('X', 9); sinput[1][1] = new Item('D', 2);
        sinput[1][2] = new Item('E', 1); sinput[1][3] = new Item('Z', 3);
        sinput[2][0] = new Item('X', 9); sinput[2][1] = new Item('X', 9);
        sinput[2][2] = new Item('H', 3); sinput[2][3] = new Item('U', 0);
        sinput[3][0] = new Item('X', 9); sinput[3][1] = new Item('I', 3);
        sinput[3][2] = new Item('K', 2); sinput[3][3] = new Item('L', 1);
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            soutput[i][j] = new Item('X', 9);
        }
    }

    System.out.println("Επιλέξτε τον αλγόριθμο χρονοπρογραμματισμού:");
    System.out.println("1. PM");
    System.out.println("2. PIM");
    System.out.println("3. IRRM");
    System.out.println("4. ISLIP");
    System.out.println("5. DRRM");
    System.out.println("6. EXIT");
    choice = kb.nextInt();
    System.out.println();
```

```
while ((choice>=1)&&(choice<=5)){

   Item[][] input = new Item[N][N];
   Item[][] output = new Item[N][M];

   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         input[i][j] = new Item(sinput[i][j].head, sinput[i][j].dest);
         arr1[i][j]=0;arr2[i][j]=0;arr3[i][j]=0;arr4[i][j]=0;
      }
   }

   for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
         output[i][j] = new Item(soutput[i][j].head, soutput[i][j].dest);
      }
      a[i] = 0;
      b[i] = 0;
   }

   switch (choice) {
      case 1:
         token = "PM";
         break;
      case 2:
         token = "PIM";
         break;
      case 3:
         token = "IRRM";
         break;
      case 4:
         token = "ISLIP";
         break;
      default:
         token = "DRRM";
         break;
   }
```

```
JFrame window = new JFrame(token);
window.setLayout(null);
window.setSize(820, 300);
window.setLocation(0, 0);
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window.setVisible(true);

JFrame window2 = new JFrame(token);
window2.setSize(820, 300);
window2.setLocation(0, 300);
window2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window2.setVisible(true);

for (int i = 0; i < N; i++) {
   for (int j = 0; j < N; j++) {
      lb[i][j] = new JLabel();
      lb[i][j].setSize(A,A);
      lb[i][j].setLocation((j+1)*(A+2),(i+1)*(A+2));
      switch (input[i][j].dest) {
         case 0:
            lb[i][j].setBackground(Color.yellow);
            break;
         case 1:
            lb[i][j].setBackground(Color.blue);
            break;
         case 2:
            lb[i][j].setBackground(Color.red);
            break;
         case 3:
            lb[i][j].setBackground(Color.green);
            break;
         default:
            lb[i][j].setBackground(Color.gray);
            break;
      }
      lb[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
      lb[i][j].setOpaque(true);
```

```
            String text="";
            if (input[i][j].dest!=9){
               text = String.valueOf(input[i][j].head) + String.valueOf(input[i][j].dest);
            }
            lb[i][j].setText(text);
            lb[i][j].setHorizontalAlignment(0);
            lb[i][j].setVisible(true);
            window.add(lb[i][j]);
         }
      }

      JLabel[][] lb2 = new JLabel[N][M];

      for (int i = 0; i < N; i++) {
         for (int j = 0; j < M; j++) {
            lb2[i][j] = new JLabel();
            lb2[i][j].setSize(A,A);
            lb2[i][j].setLocation((j+1)*(A+2)+D,(i+1)*(A+2));
            lb2[i][j].setBackground(Color.GRAY);
            lb2[i][j].setOpaque(true);
            lb2[i][j].setVisible(true);
            window.add(lb2[i][j]);
         }
      }

      JButton jb = new JButton("ΕΝΑΡΞΗ");
      jb.setSize(100, 40);
      jb.setLocation(260, 200);
      jb.setVisible(true);
      window.add(jb);

      jb.addActionListener(new ActionListener(){
         @Override
         public void actionPerformed(ActionEvent e) {
            if (jb.getText().equals("ΤΕΛΟΣ")) {
               window.dispose();
               window2.dispose();
```

```
        } else {
            cont = true;
        }
    }
});

GCHRONO dr = new GCHRONO();
window2.add(dr);

int[][] array = new int[N][N];
int pos = 0;

System.out.println("Αρχική μορφή:");
System.out.println("--------------------------");
print_switch(input, output, pos);

while (!endOfTable(input)) {
    cont = false;
    while (!cont) {}
    System.out.println();
    System.out.printf("%dη Χρονοθυρίδα:\n", pos+1);
    System.out.println("--------------------------");
    window2.setTitle ((pos+1)+"η ΧΡΟΝΟΘΥΡΙΔΑ");

    switch (choice) {
        case 1:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array);
            copyArrays(array, arr2);
            acceptance(array);
            copyArrays(array, arr3);
            break;
        case 2:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array);
```

```
            copyArrays(array, arr2);
            acceptance(array);
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
        case 3:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array, b);
            print_point(b, 'b');
            copyArrays(array, arr2);
            acceptance(array, a);
            print_point(a, 'a');
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
        case 4:
            request(array, input);
            copyArrays(array, arr1);
            reservation(array, b);
            copyArrays(array, arr2);
            acceptance(array, a, b);
            print_point(a, 'a');
            print_point(b, 'b');
            copyArrays(array, arr3);
            makeFull(array, input);
            splitArrayFull(array,arr3,arr4);
            break;
        default:
            request(array, input, a);
            print_point(a, 'a');
            copyArrays(array, arr1);
            reservation(array, b);
            print_point(b, 'b');
            copyArrays(array, arr2);
```

```
                    break;
                }
                dr.repaint();
                updateTable(array, input, output, pos);
                if (!endOfTable(input)) {
                    jb.setText("ΕΠΟΜΕΝΟ");
                    print_switch(input, output, pos+1);
                }else {
                    jb.setText("ΤΕΛΟΣ");
                }
                pos++;
            }

            System.out.println();
            System.out.println("Τελική μορφή:");
            System.out.println("-------------------------");
            print_switch(input, output, pos);
            System.out.println();

            System.out.println("Επιλέξτε τον αλγόριθμο χρονοπρογραμματισμού:");
            System.out.println("1. PM");
            System.out.println("2. PIM");
            System.out.println("3. IRRM");
            System.out.println("4. ISLIP");
            System.out.println("5. DRRM");
            System.out.println("6. EXIT");
            choice = kb.nextInt();
            System.out.println();
        }
        System.out.println("***Author: Kafopoulos Panagiotis, postgraduate student***");
        System.out.println();
    }
```

```
@Override
  public void paintComponent (Graphics g) {
     int x=10;
     int y=50;
     int s2=150;
     int d1=s2/5;
     int d2=300;

     if (choice==5){
        x=150;
        d2=350;
     }

     g.setColor(Color.black);
     g.drawString("ΑΙΤΗΣΕΙΣ", x, y-20);
     if ((choice==3)||(choice==4)){
        g.drawString("ΠΑΡΑΧΩΡΗΣΕΙΣ", x+d2, y-20);
     }else{
        g.drawString("ΚΡΑΤΗΣΕΙΣ", x+d2, y-20);
     }
     if (choice!=5){
        g.drawString("ΑΠΟΔΟΧΕΣ", x+2*d2, y-20);
     }

     g.drawRect (x, y, s2, s2);
     g.drawString("0.", x+2, y+d1);
     g.drawString(".0", x+s2-12, y+d1);
     g.drawString("1.", x+2, y+2*d1);
     g.drawString(".1", x+s2-12, y+2*d1);
     g.drawString("2.", x+2, y+3*d1);
     g.drawString(".2", x+s2-12, y+3*d1);
     g.drawString("3.", x+2, y+4*d1);
     g.drawString(".3", x+s2-12, y+4*d1);

     g.drawRect (x+d2, y, s2, s2);
     g.drawString("0.", x+2+d2, y+d1);
     g.drawString(".0", x+d2+s2-12, y+d1);
```

```
g.drawString("1.", x+2+d2, y+2*d1);
g.drawString(".1", x+d2+s2-12, y+2*d1);
g.drawString("2.", x+2+d2, y+3*d1);
g.drawString(".2", x+d2+s2-12, y+3*d1);
g.drawString("3.", x+2+d2, y+4*d1);
g.drawString(".3", x+d2+s2-12, y+4*d1);

if (choice!=5){
   g.drawRect (x+2*d2, y, s2, s2);
   g.drawString("0.", x+2+2*d2, y+d1);
   g.drawString(".0", x+2*d2+s2-12, y+d1);
   g.drawString("1.", x+2+2*d2, y+2*d1);
   g.drawString(".1", x+2*d2+s2-12, y+2*d1);
   g.drawString("2.", x+2+2*d2, y+3*d1);
   g.drawString(".2", x+2*d2+s2-12, y+3*d1);
   g.drawString("3.", x+2+2*d2, y+4*d1);
   g.drawString(".3", x+2*d2+s2-12, y+4*d1);
}

if (choice==3){
   g.setColor(Color.magenta);
   for (int i = 0; i < N; i++) {
      g.drawString("b"+i+"="+b[i], x+d2+s2-12+17, y+d1+i*d1);
      g.drawString("a"+i+"="+a[i], x+2*d2-32, y+d1+i*d1);
   }
}else if (choice==4){
   g.setColor(Color.magenta);
   for (int i = 0; i < N; i++) {
      g.drawString("a"+i+"="+a[i], x+2*d2-32, y+d1+i*d1);
      g.drawString("b"+i+"="+b[i], x+2*d2+s2-12+17, y+d1+i*d1);
   }
}else if (choice==5){
   g.setColor(Color.magenta);
   for (int i = 0; i < N; i++) {
      g.drawString("a"+i+"="+a[i], x-32, y+d1+i*d1);
      g.drawString("b"+i+"="+b[i], x+d2+s2-12+17, y+d1+i*d1);
   }
```

```
    }

    g.setColor(Color.black);
    drawArrows (arr1, g, 0);
    g.setColor(Color.red);
    drawArrows (arr2, g, 1);
    if (choice!=5){
       g.setColor(Color.blue);
       drawArrows (arr3, g, 2);
       if (choice!=1){
          g.setColor(Color.green);
          drawArrows (arr4, g, 2);
       }
    }
}

public void drawArrows (int[][] p, Graphics g, int k) {
    int i, j;
    int x=10;
    int y=50;
    int s2=150;
    int d1=s2/5;
    int d2=300;

    if (choice==5){
       x=150;
       d2=350;
    }

    for (i = 0; i < N; i++) {
       for (j = 0; j < N; j++) {
          if (p[i][j] == 1) {
             int x1 = (x+10)+k*d2;
             int y1 = (y+d1)+i*d1;
             int x2 = (x+s2-12)+k*d2;
             int y2 = (y+d1)+j*d1;
             g.drawLine(x1, y1, x2, y2);
```

```
                if (k == 1) {
                    g.fillOval (x1-3,y1-3,7,7);
                }else{
                    g.fillOval (x2-3,y2-3,7,7);
                }
            }
        }
    }
}

static void request (int[][] p, Item[][] in) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            p[i][j] = 0;
        }
    }
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            k = in[i][j].dest;
            if ((k>=0)&&(k<=N)) {
                p[i][k] = 1;
            }
        }
    }
    System.out.println("\n*** REQUESTS ***\n");
    print_table(p);
}

static void request (int[][] p, Item[][] in, int[] a) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            p[i][j] = 0;
        }
    }
    for (i = 0; i < N; i++) {
```

```
            for (j = 0; j < N; j++) {
                k = in[i][j].dest;
                if ((k>=0)&&(k<=N)) {
                    p[i][k] = 1;
                }
            }
        }
        for (i = 0; i < N; i++) {
            int sum_row = 0;
            for (j = 0; j < N; j++) {
                sum_row += p[i][j];
            }
            if (sum_row > 0) {
                k = a[i];
                while (p[i][k] == 0) {
                    if (k == N - 1) {
                        k = 0;
                    } else {
                        k++;
                    }
                }
                for (j = 0; j < N; j++) {
                    p[i][j] = 0;
                }
                p[i][k] = 1;
                if (k == N - 1) {
                    k = 0;
                } else {
                    k++;
                }
                a[i] = k;
            }
        }
        System.out.println("\n*** REQUESTS ***\n");
        print_table(p);
    }
```

```
static void reservation (int[][] p) {
   int i, j, k;
   for (j = 0; j < N; j++) {
      int sum_col = 0;
      for (i = 0; i < N; i++) {
         sum_col += p[i][j];
      }
      if (sum_col > 0) {
         k = (int) (Math.random()*N);
         while (p[k][j] == 0) {
            k = (int) (Math.random()*N);
         }
         for (i = 0; i < N; i++) {
            p[i][j] = 0;
         }
         p[k][j] = 1;
      }
   }
   System.out.println("\n*** RESERVATIONS ***\n");
   print_table(p);
}

static void reservation (int[][] p, int[] b) {
   int i, j, k;
   for (j = 0; j < N; j++) {
      int sum_col = 0;
      for (i = 0; i < N; i++) {
         sum_col += p[i][j];
      }
      if (sum_col > 0) {
         k = b[j];
         while (p[k][j] == 0) {
            if (k == N - 1) {
               k = 0;
            } else {
               k++;
            }
```

```
            }
            for (i = 0; i < N; i++) {
               p[i][j] = 0;
            }
            p[k][j] = 1;
            if (choice != 4){
               if (k == N - 1) {
               k = 0;
               } else {
                  k++;
               }
               b[j] = k;
            }
         }
      }
      if (choice == 5){
        System.out.println("\n*** RESERVATIONS ***\n");
      }else{
         System.out.println("\n*** GRANTS ***\n");
      }
      print_table(p);
   }

   static void acceptance (int[][] p) {
      int i, j, k;
      for (i = 0; i < N; i++) {
         int sum_row = 0;
         for (j = 0; j < N; j++) {
            sum_row += p[i][j];
         }
         if (sum_row > 0) {
            k = (int) (Math.random()*N);
            while (p[i][k] == 0) {
               k = (int) (Math.random()*N);
            }
            for (j = 0; j < N; j++) {
               p[i][j] = 0;
```

```
            }
            p[i][k] = 1;
        }
    }
    System.out.println("\n*** ACCEPTANCES ***\n");
    print_table(p);
}

static void acceptance (int[][] p, int[] a) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        int sum_row = 0;
        for (j = 0; j < N; j++) {
            sum_row += p[i][j];
        }
        if (sum_row > 0) {
            k = a[i];
            while (p[i][k] == 0) {
                if (k == N - 1) {
                    k = 0;
                } else {
                    k++;
                }
            }
            for (j = 0; j < N; j++) {
                p[i][j] = 0;
            }
            p[i][k] = 1;
            if (k == N - 1) {
                k = 0;
            } else {
                k++;
            }
            a[i] = k;
        }
    }
    System.out.println("\n*** ACCEPTANCES ***\n");
```

```
    print_table(p);
}

static void acceptance (int[][] p, int[] a, int[] b) {
    int i, j, k, l;
    for (i = 0; i < N; i++) {
        int sum_row = 0;
        for (j = 0; j < N; j++) {
            sum_row += p[i][j];
        }
        if (sum_row > 0) {
            k = a[i];
            while (p[i][k] == 0) {
                if (k == N - 1) {
                    k = 0;
                } else {
                    k++;
                }
            }
            for (j = 0; j < N; j++) {
                p[i][j] = 0;
            }
            p[i][k] = 1;
            l = i;
            if (l == N - 1) {
                l = 0;
            } else {
                l++;
            }
            b[k] = l;
            if (k == N - 1) {
                k = 0;
            } else {
                k++;
            }
            a[i] = k;
        }
```

```
    }
    System.out.println("\n*** ACCEPTANCES ***\n");
    print_table(p);
}

static void makeFull (int[][] p, Item[][] in) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            k = 0;
            while ((zeroRowCol(p, i, j)) && (k < N)) {
                if (in[i][k].dest == j) {
                    p[i][j] = 1;
                } else {
                    k++;
                }
            }
        }
    }
    System.out.println("\n*** FULL ALLIGNEMENT ***\n");
    print_table(p);
}

static void splitArrayFull (int[][] p, int[][] q, int[][] n) {
    int i, j;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            n[i][j]=0;
            if ((p[i][j]==1)&&(q[i][j]==0)){
                n[i][j]=1;
            }
        }
    }
}

static boolean zeroRowCol (int[][] p, int r, int c) {
    int i;
```

```
        int sum_row = 0;
        int sum_col = 0;
        for (i = 0; i < N; i++) {
            sum_row += p[r][i];
            sum_col += p[i][c];
        }

        return (sum_row + sum_col == 0);
    }

    static void print_table (int[][] p) {
        int i, j;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                System.out.printf("%3d", p[i][j]);
            }
            System.out.println();
        }
    }

    static void print_point (int[] a, char c) {
        int i;
        System.out.println();
        for (i = 0; i < N; i++) {
            System.out.printf("%c%d=%d ", c, i, a[i]);
        }
        System.out.println();
    }

static void print_switch (Item[][] in, Item[][] out, int pos) {
        int i, j;
        System.out.println();
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                System.out.printf("%c %d | ", in[i][j].head, in[i][j].dest);
            }
            System.out.printf(" |--o %d %d o--| ", i, i);
```

```
            for (j = 0; j < pos; j++) {
                System.out.printf("%c %d | ", out[i][j].head, out[i][j].dest);
            }
            System.out.println();
        }
    }

    static void copyArrays (int[][] p, int[][]q) {
        int i, j;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                q[i][j] = p[i][j];
            }
        }
    }

    static boolean endOfTable (Item[][] in) {
        int i, j;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                if (in[i][j].dest != 9) {
                    return false;
                }
            }
        }
        return true;
    }

    static void updateTable (int[][] p, Item[][] in, Item[][] out, int pos) {
        AnimationClass AC = new AnimationClass();
        int i, j, k;
        for (i = 0; i < N; i++) {
            for (j = pos; j > 0; j--) {
                if (out[i][j-1].dest != 9) {
                    out[i][j].head = out[i][j-1].head;
                    out[i][j].dest = out[i][j-1].dest;
                    out[i][j-1].head = 'X';
```

```
                out[i][j-1].dest = 9;
            }
        }
    }
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (p[i][j] == 1) {
                k = 0;
                while (in[i][k].dest != j) {
                    k++;
                }
                out[j][0].head = in[i][k].head;
                out[j][0].dest = in[i][k].dest;
                in[i][k].head = 'X';
                in[i][k].dest = 9;

                AC.jLabelXRight((k+1)*(A+2),(A+2)+D,10,2,lb[i][k]);
                if (i<j) {
                    AC.jLabelYDown((i+1)*(A+2),(j+1)*(A+2),10,2,lb[i][k]);
                }else if (i>j) {
                    AC.jLabelYUp((i+1)*(A+2),(j+1)*(A+2),10,2,lb[i][k]);
                }
            }
            int x = lb[i][j].getX();
            if (x > D) {
                AC.jLabelXRight(x, x+(A+2),10,2,lb[i][j]);
            }
        }
    }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Νικόλαος Δ. Τσελίκας, Αλγόριθμοι Χρονοπρογραμματισμού σε Μεταγωγείς Ευρυζωνικών Δικτύων, Σεπτέμβριος 2014

[2] Adisak Mekkittikul, D. Sadot, L.G. Kazovsky, Nick McKeown, "8 Tb/s ATM Interconnection through optical WDM networks", High-Speed Semiconductor Laser Sources, San Jose, CA, Vol. 2684, pp. 186-98, 1-2 February, 1996.

[3] Nick McKeown and Balaji Prabhakar, "Scheduling Multicast Cells in an Input-Queued Switch", Proceedings of IEEE Infocom '96, San Francisco, Vol 1, pp. 271-278, March 1996.

[4] Nick McKeown, Venkat Anantharam and Jean Walrand, "Achieving 100% Throughput in an Input-Queued Switch", Proceedings of IEEE Infocom '96, Vol 1, pp. 296-302, San Francisco, March 1996.

[5] Rietsh Ahuja, Balaji Prabhakar, Nick McKeown, "Multicast Scheduling for Input-Queued Switches", IEEE Journal on Selected Areas in Communications, Vol 15, No. 15, pp. 885-866, June 1997.

[6] Anthony Hung, George Kesidis, and Nick McKeown, "ATM Input-Buffered Switches with Guaranteed-Rate Property", Proc. IEEE ISCC '98, Athens, Jul 1998, pp 331-335.

[7] Nick McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches", IEEE Transactions on Networking, Vol 7, No.2, April 1999.

[8] Thomas Anderson, Susan Owicki, James Saxe and Charles Thacker, "High speed switch scheduling for local area networks," ACM Transactions on Computer Systems, vol. 11,no. 4, pp. 319–352, Nov. 1993.

[9] Shang-Tse Chuang, Sundar Iyer, Nick McKeown, "Practical Algorithms for Performance Guarantees in Buffered Crossbars", Proceedings of IEEE INFOCOM 2005, Miami, Florida, March 2005.