Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Πτυχιακή Εργασία

Κατανεμημένες Συναλλαγές Ασφαλείας Πολλών Επιπέδων

Πολυμέρου Γεώργιος

Επιβλέπων Λέκτορας Παναγιώτης Κατσαρός

Θεσσαλονίκη 2006

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή στη Διαχείριση των Συναλλαγών

Κεφάλαιο 2: Μοντέλα ελέγχου προσπέλασης βασισμένα σε πλέγματα

Κεφάλαιο 3: Κριτήρια Ορθότητας για κατανεμημένες συναλλαγές πολλών επιπέδων ασφαλείας

Κεφάλαιο 4: ASEP-Ένα ασφαλές και ευέλικτο πρωτόκολλο εκτέλεσης για κατανεμημένες συναλλαγές πολλών επιπέδων ασφαλείας

Κεφάλαιο 1
Εισαγωγή στη Διαχείριση των Συναλλαγών
(Transaction management)

1. Εισαγωγή

Σε πολλές περιπτώσεις, οι χρήστες των πληροφοριακών συστημάτων, πρέπει να εκτελέσουν ένα σύνολο λειτουργιών (που αναφέρονται και ως *νήματα*). Η εκτέλεση αυτή πρέπει υποχρεωτικά να γίνει με έναν ατομικό τρόπο. Με ένα τρόπο, δηλαδή, έτσι ώστε αν και μία μόνο λειτουργία αποτύχει, η συνολική επεξεργασία αποτυγχάνει. Ένα τέτοιο σύνολο λειτουργιών-νημάτων αποτελεί μία λογική μονάδα και ονομάζεται *συναλλαγή* (*transaction*). Οι εντολές μιας συναλλαγής μπορούν να ποικίλουν, συμπεριλαμβανομένων εντολών ανάγνωσης, εγγραφής διαγραφής και ενημέρωσης των δεδομένων κάποιας βάσης δεδομένων (*ΒΔ*). Τον ρόλο για την ατομική διαχείριση των συναλλαγών τον έχει το Σύστημα Διαχείρισης της ΒΔ (*ΣΔΒΔ*), και πιο συγκεκριμένα το τμήμα του ΣΔΒΔ το οποίο είναι υπεύθυνο για τη λειτουργία των συναλλαγών, δηλαδή ο διαχειριστής συναλλαγών (*transaction manager*). Μετά το πέρας μιας συναλλαγής, τα δεδομένα πρέπει να ικανοποιούν όλους τους περιορισμούς ακεραιότητας οι οποίοι έχουν οριστεί κατά τη σχεδίαση της ΒΔ.

Το θέμα της διαχείρισης των συναλλαγών γίνεται ακόμα πιο πολύπλοκο λαμβάνοντας υπόψη πως τα σύγχρονα ΣΔΒΔ μπορούν να εξυπηρετούν πολλούς χρήστες ταυτόχρονα. Έτσι, πρέπει να υπάρχουν αξιόπιστοι μηχανισμοί ελέγχου της ακεραιότητας των δεδομένων καθώς πολλές συναλλαγές μπορούν να εκτελούνται παράλληλα, επεξεργαζόμενες συγχρόνως δεδομένα (που πιθανώς μπορεί να είναι και κοινά). Αν επιτραπεί σε δύο, για παράδειγμα, συναλλαγές να επεξεργάζονται τα ίδια δεδομένα ταυτόχρονα, τότε υπάρχει μεγάλη πιθανότητα αλλοίωσης των δεδομένων. Το ΣΔΒΔ, με τον έλεγχο ταυτοχρονισμού που θα δούμε στη συνέχεια, πρέπει να διαχειρίζεται τις συναλλαγές με τέτοιον τρόπο ώστε να αποφεύγονται καταστάσεις όπως η προηγούμενη. Επίσης, η ΒΔ μπορεί να βρεθεί σε «ασταθή» κατάσταση, ιδιαίτερα μετά από μία βλάβη του συστήματος, η οποία μπορεί να προκληθεί για διάφορους λόγους που θα εξετάσουμε στη συνέχεια. Υπάρχει, δηλαδή, η ανάγκη για τη λειτουργία μηχανισμών επανάκτησης των δεδομένων, έτσι ώστε η ΒΔ να επανέλθει στη κανονική της κατάσταση, χωρίς να χαθούν τα δεδομένα της, με πλήρη ικανοποίηση των περιορισμών ακεραιότητάς τους.

2. Συναλλαγές

2.1 Καταστάσεις Συναλλαγών

Μία συναλλαγή είτε θα τερματιστεί κανονικά προκαλώντας μόνιμες αλλαγές στη ΒΔ είτε θα απορριφθεί με τα δεδομένα να επανέρχονται στη προηγούμενη τους κατάσταση. Οι δύο παραπάνω καταστάσεις αποτελούν τις καταστάσεις τερματισμού των συναλλαγών. Παρόλα αυτά, πριν τη μετάβαση μιας συναλλαγές σε μία από τις τελικές καταστάσης της, μία συναλλαγή μπορεί να βρεθεί και σε κάποιες ενδιάμεσες καταστάσεις (σχήμα 1), οι οποίες είναι οι παρακάτω :

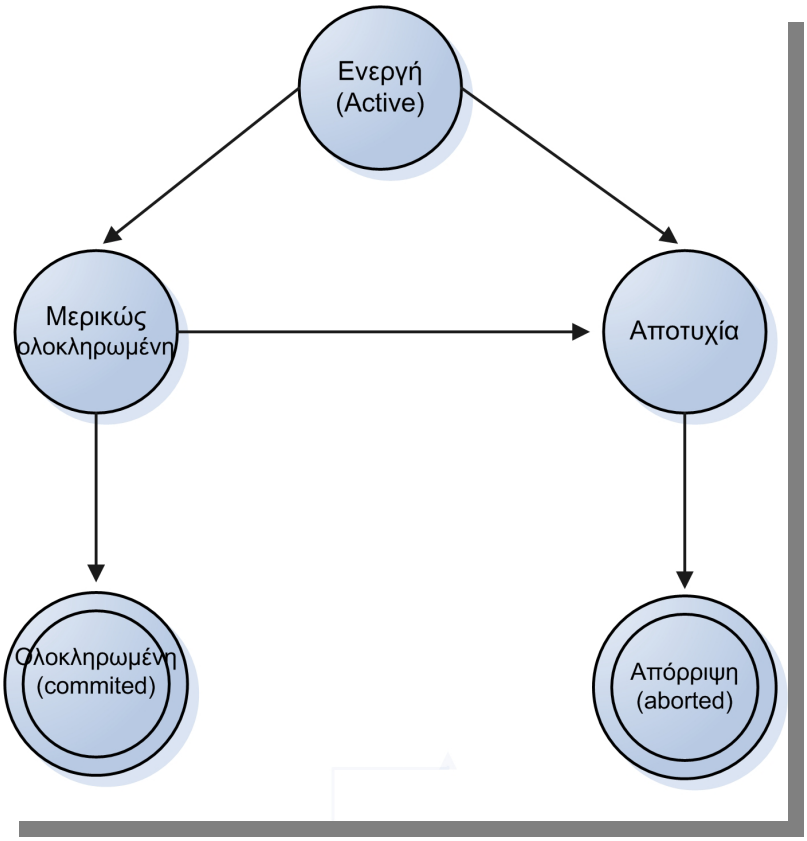
- *Ενεργή κατάσταση*: Η συναλλαγή εισέρχεται στη κατάσταση αυτή κατά την εκκίνησή της και παραμένει σε αυτή όσο βρίσκεται σε εκτέλεση.

- *Κατάσταση μερικής ολοκλήρωσης*: Η συναλλαγή μεταβαίνει σε αυτή τη κατάσταση όταν έχει εκτελεστεί και η τελευταία εντολή της, αλλά δεν έχει ακόμα ολοκληρωθεί.

- *Κατάσταση αποτυχίας*: Η συναλλαγή μεταβαίνει στη κατάσταση αυτή όταν δεν μπορεί να εκτελέσει με επιτυχία τις εντολές της.

- *Κατάσταση απόρριψης*: Η συναλλαγή μεταβαίνει στη κατάσταση απόρριψης όταν τα δεδομένα που αυτή επεξεργάστηκε έχουν επανέλθει στη προηγούμενη τους κατάσταση, πριν την αρχή της εκτέλεσής της.

- *Κατάσταση ολοκλήρωσης*: Η συναλλαγή έχει ολοκληρώσει την εκτέλεσή της με επιτυχία και προκαλεί μόνιμες αλλαγές στη ΒΔ..



Σχήμα 1

2.2 Οι ιδιότητες των συναλλαγών

Ο διαχειριστής συναλλαγών του ΣΔΒΔ πρέπει να εκτελεί τις συναλλαγές που του ανατίθενται με βάση τις παρακάτω ιδιότητες, οι οποίες είναι γνωστές στη βιβλιογραφία ως *ACID ιδιότητες*, από τις λέξεις **ατομικότητα** (*atomicity*), **συνέπεια** (*consistency*), **απομόνωση** (*isolation*) και **αντοχή** (*durability*).

- *Ατομικότητα:* Η ύπαρξη τουλάχιστον μιας εντολής της συναλλαγής η οποία αποτυγχάνει να εκτελεστεί, οδηγεί στην αποτυχία ολόκληρης της συναλλαγής.

- *Συνέπεια:* Η συναλλαγή πρέπει να μετατρέπει τη ΒΔ από μία κατάσταση συνέπειας σε μία άλλη. Η συνέπεια σχετίζεται άμεσα με την ορθότητα των δεδομένων της ΒΔ.

- *Απομόνωση:* Κάθε συναλλαγή πρέπει να εκτελείται ανεξάρτητα από άλλες συναλλαγές, σαν να ήταν η μόνη συναλλαγή προς εκτέλεση στο σύστημα.

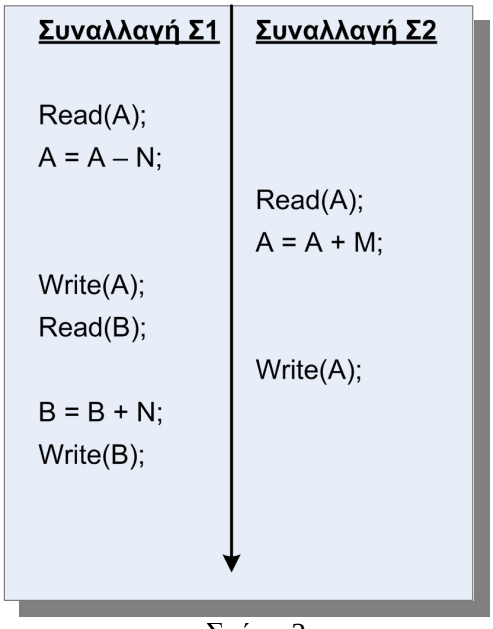
- *Αντοχή:* Αν μία συναλλαγή ολοκληρωθεί επιτυχώς, τότε οι αλλαγές που προκύπτουν στη ΒΔ πρέπει να είναι μόνιμες και να μη μπορούν να ανακληθούν με κανέναν τρόπο.

3. Έλεγχος ταυτοχρονισμού (*concurrency control*)

Σε ένα πληροφοριακό σύστημα μπορούν να βρίσκονται σε ταυτόχρονη εκτέλεση περισσότερες από μία συναλλαγές. Το γεγονός αυτό ενδέχεται να προκαλέσει προβλήματα στην ακεραιότητα και στη συνέπεια των δεδομένων της ΒΔ. Σε περίπτωση όπου όλες οι συναλλαγές που εκτελούνται ταυτόχρονα, εκτελούν μόνον εντολές ανάγνωσης δεδομένων, τότε δεν υπάρχει κίνδυνος για τα δεδομένα. Σε αντίθετη όμως περίπτωση, όπου υπάρχουν εντολές οι οποίες τροποποιούν και επεξεργάζονται δεδομένα που διαβάζονται από άλλες συναλλαγές, τότε πιθανότατα πρέπει να αντιμετωπιστεί το πρόβλημα του συντονισμού όλων των συναλλαγών. Ο σωστός συντονισμός των συναλλαγών είναι ευθύνη του ΣΔΒΔ, το οποίο μέσω του *μηχανισμού ελέγχου ταυτοχρονισμού* (*concurrency control*) εξασφαλίζει την ομαλή και ορθή εκτέλεση των συναλλαγών. Παρακάτω θα περιγράψουμε τρεις περιπτώσεις όπου ο συντονισμός των συναλλαγών είναι απαραίτητος για την ταυτόχρονη εκτέλεση τους.

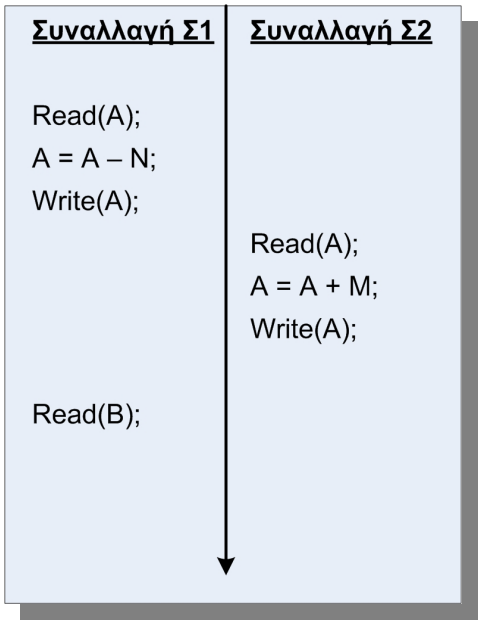
3.1 Περιπτώσεις προβλημάτων από την ταυτόχρονη εκτέλεση συναλλαγών

Ως πρώτη περίπτωση προβλήματος που προκύπτει από την ταυτόχρονη εκτέλεση των συναλλαγών, είναι το πρόβλημα της χαμένης ενημέρωσης. Το πρόβλημα αυτό εμφανίζεται όταν δύο συναλλαγές εκτελούν εντολές ανάγνωσης και εγγραφής πάνω στα ίδια δεδομένα. Έστω ότι έχουμε τις δύο συναλλαγές που φαίνονται στο παρακάτω σχήμα (σχήμα 2). Η συναλλαγή *Σ2* διαβάζει την τιμή του *Α* πριν καταχωρηθεί η νέα τιμή από τη συναλλαγή *Σ2*, και αποθηκεύει τη νέα τιμή του *Α* στη ΒΔ μετά από τη *Σ1*. Έτσι, η ενημέρωση του *Α* από τη συναλλαγή *Σ1* έχει χαθεί.



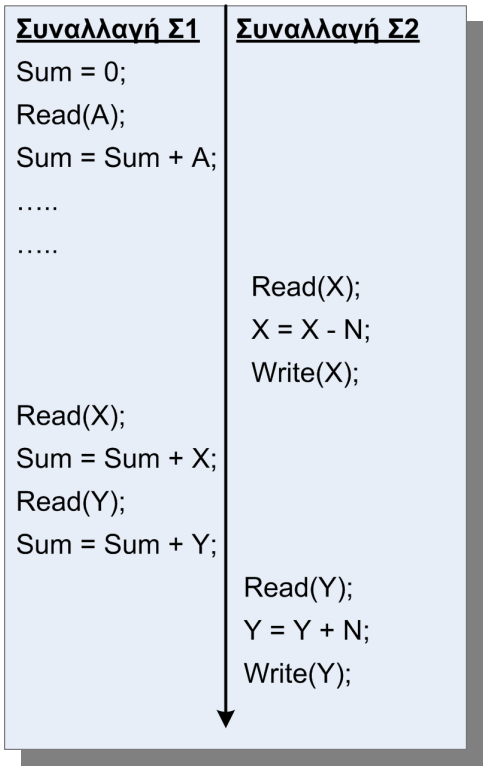
Σχήμα 2

Στη δεύτερη περίπτωση έχουμε το πρόβλημα της λανθασμένης ανάγνωσης, το οποίο φαίνεται στο παρακάτω σχήμα (σχήμα 3). Το πρόβλημα αυτό εμφανίζεται όταν μία συναλλαγή τροποποιεί τα δεδομένα και για κάποιο λόγο αποτυγχάνει. Πριν όμως την επαναφορά των δεδομένων αυτών στην αρχική τους κατάσταση, κάποια άλλη συναλλαγή διαβάζει τα δεδομένα με αποτέλεσμα τη λανθασμένη ανάγνωσή τους. Στο σχήμα, η συναλλαγή *Σ1* αλλάζει τη τιμή του *A* και έστω πως μετά αποτυγχάνει. Αμέσως μετά όμως, η συναλλαγή *Σ2* διαβάζει το *A*, και αφού προσθέσει το *M* αποθηκεύει τη νέα τιμή του *A*. Η τιμή του *A* που διαβάζεται από τη συναλλαγή *Σ2* είναι λανθασμένη και αυτή έπρεπε να είχε αντικατασταθεί με την τιμή πριν την εκτέλεση της *Σ1*.



Σχήμα 3

Η τελευταία περίπτωση προβλήματος από την ταυτόχρονη εκτέλεση των συναλλαγών, είναι το πρόβλημα της λανθασμένης άθροισης. Στο σχήμα 4, η συναλλαγή *Σ1* διαβάζει κάποια δεδομένα και παράγει το άθροισμά τους. Ωστόσο, κάποια δεδομένα ενημερώνονται από τη συναλλαγή *Σ2* πριν διαβαστούν από τη *Σ1* (όπως το *X*), ενώ κάποια άλλα δεδομένα ενημερώνονται από τη *Σ2* μετά την ανάγνωσή τους από την *Σ1* (όπως το *Y*). Κατά συνέπεια, το τελικό αποτέλεσμα της άθροισης είναι λανθασμένο.



Σχήμα 4

4. Σειριοποίηση συναλλαγών (*serializability*)

Με σκοπό να επιτευχθεί η συνέπεια των δεδομένων τα οποία αποθηκεύονται σε μία ΒΔ, το ΣΔΒΔ πρέπει να συντονίσει την εκτέλεση των ταυτόχρονων συναλλαγών. Ο τρόπος και η σειρά με την οποία εκτελείται ένα σύνολο συναλλαγών ονομάζεται *χρονοδιάγραμμα* (*schedule*). Έτσι, το ΣΔΒΔ πρέπει να εκτελέσει κάποιο χρονοδιάγραμμα το οποίο θα εγγυάται τη συνέπεια των δεδομένων που επεξεργάζονται οι συναλλαγές.

Ένα χρονοδιάγραμμα συναλλαγών ονομάζεται σειριακό χρονοδιάγραμμα σε περίπτωση όπου οι εντολές μιας συναλλαγής δεν επικαλύπτονται χρονικά από τις εντολές των άλλων συναλλαγών. Αν υποθέσουμε πως έχουμε *n* συναλλαγές, ο αριθμός των δυνατών σειριακών χρονοδιαγραμμάτων ισούται με τον αριθμό των διατάξεων των *n* συναλλαγών. Συνεπώς, υπάρχουν *n*! δυνατά σειριακά χρονοδιαγράμματα εκτέλεσης. Το ερώτημα που τίθεται εδώ είναι το πως το ΣΔΒΔ θα επιλέξει το σωστό χρονοδιάγραμμα για να εξασφαλιστεί η συνέπεια των δεδομένων. Για να γίνει αυτό, πρέπει το χρονοδιάγραμμα το οποίο θα επιλεχθεί να είναι ισοδύναμο με κάποιο σειριακό χρονοδιάγραμμα. Τα αποτελέσματα, δηλαδή, του χρονοδιαγράμματος που θα επιλεχθεί να είναι τα ίδια στη ΒΔ με αυτά ενός σειριακού χρονοδιαγράμματος. Η παραγωγή αυτή ενός χρονοδιαγράμματος ισοδύναμου με ένα σειριακό, ονομάζεται σειριοποίηση.

5. Μηχανισμοί κλειδώματος

Μία από τις πιο χρήσιμες τεχνικές που έχουν σαν σκοπό τον συντονισμό της εκτέλεσης ενός συνόλου συναλλαγών είναι η χρήση *κλειδαριών* (*locks*). Η κλειδαριές είναι μεταβλητές οι οποίες περιγράφουν τη κατάσταση των δεδομένων σε σχέση με τις λειτουργίες που ενεργούν στα συγκεκριμένα δεδομένα. Δύο είναι οι βασικοί τύποι κλειδαριών :

- *Κλειδαριά ανάγνωσης* (*read lock*): Επιτρέπει σε μια συναλλαγή να διαβάσει δεδομένα αλλά όχι να τα τροποποιήσει ή να τα διαγράψει και

- *Κλειδαριά αποθήκευσης* (*write lock*): Επιτρέπει σε μια συναλλαγή να διαβάσει ή να ενημερώσει τα δεδομένα.

Κάθε φορά που μία συναλλαγή επιθυμεί να προσπελάσει κάποια δεδομένα ζητά να της χορηγηθεί μία κλειδαριά ανάγνωσης. Επειδή η ανάγνωση των ίδιων δεδομένων από πολλές συναλλαγές δεν δημιουργεί κανένα πρόβλημα συνέπειας, μπορούν να χορηγηθούν περισσότερες από μία κλειδαριές για τα ίδια δεδομένα. Σε περίπτωση όμως που ζητηθεί από μία συναλλαγή μία κλειδαριά αποθήκευσης, η κλειδαριά αυτή δίνεται σε μία μόνο συναλλαγή και σε καμία άλλη συγχρόνως. Μια συναλλαγή απελευθερώνει την κλειδαριά που έχει δεσμεύσει όταν η εκτέλεση της ολοκληρωθεί είτε με επιτυχία είτε όχι. Οι μόνιμες αλλαγές στη ΒΔ, είναι ορατές από τις άλλες συναλλαγές μόνο μετά την αποδέσμευση της κλειδραριάς αποθήκευσης από κάποια συναλλαγή. Επίσης, κάποια πληροφοριακά συστήματα επιτρέπουν στις συναλλαγές να αλλάξουν τον τύπο της κλειδαριάς που έχουν αποκτήσει. Δηλαδή, μία κλειδαριά ανάγνωσης που συμβαίνει να πρέπει να ενημερώσει κάποια δεδομένα, μπορεί να αλλάξει απο κλειδαριά ανάγνωσης σε κλειδαριά αποθήκευσης. Η αλλαγή αυτή της κλειδαριάς ονομάζεται αναβάθμιση (u*pgrade*), ενώ το αντίστροφο υποβάθμιση (*downgrade*).

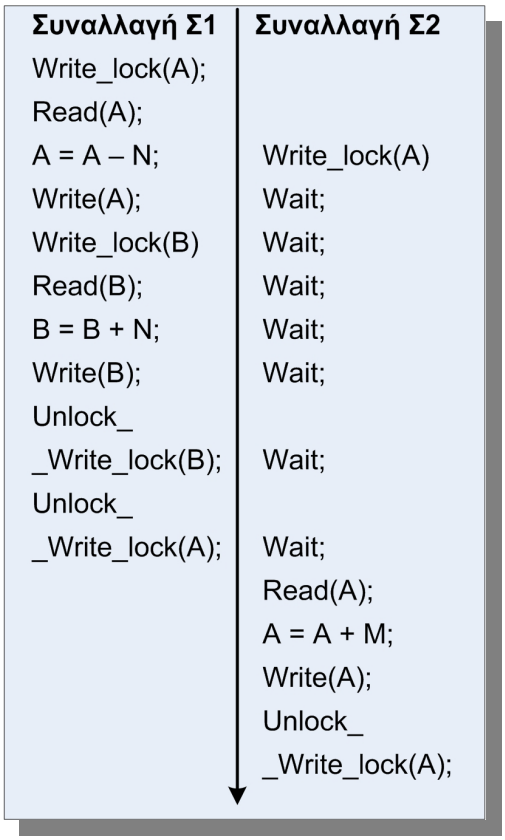
Πρωτόκολλο κλειδώματος δύο φάσεων (2-phase-locking protocol)

Το πρωτόκολλο κλειδώματος δύο φάσεων ορίζει δύο φάσεις λειτουργίας μιας συναλλαγής :

1. *Φάση ανάπτυξης* (*growing phase*): Η συναλλαγή ζητά να αποκτήσει τις κλειδαριές που χρειάζεται και δεν επιτρέπεται η απελευθέρωσή αυτών.

2. *Φάση συρρίκνωσης* (*shrinking phase*): Η συναλλαγή απελευθερώνει τις κλειδαριές της και δεν της επιτρέπεται η χορήγηση νέων κλειδαριών.

Σύμφωνα με τον Eswaran [*Eswaran, "2-phase locking techniques",1976*], για κάθε συναλλαγή η οποία ικανοποιεί τους κανόνες του πρωτοκόλλου κλειδώματος δύο φάσεων, το χρονοδιάγραμμα που προκύπτει είναι πάντοτε σειριοποιήσημο. Παρακάτω, στο σχήμα 5, μπορούμε να δούμε τον τρόπο με τον οποίο το πρωτόκολλο κλειδώματος δύο φάσεων λύνει το πρόβλημα της χαμένης ενημέρωσης που παρουσιάστηκε παραπάνω.



Σχήμα 5

6. Μηχανισμός χρονικών σφραγίδων (*timestamp ordering*)

Προηγουμένως, εξετάσαμε την τεχνική χρήσης των κλειδαριών με σκοπό να εξασφαλιστεί η ορθότητα της συνέπειας των δεδομένων τα οποία επεξεργάζονται οι συναλλαγές. Ένας εναλλακτικός τρόπος ελέγχου των ταυτόχρονων συναλλαγών που εκτελούνται είναι *ο μηχανισμός χρονικών σφραγίδων* (*timestamps technique*), ο οποίος διαφέρει σε πολλά σημεία από τη τεχνική χρήσης κλειδαριών. Η χρονική σφραγίδα αποτελεί μία μεταβλητή η οποία σχετίζει μία συγκεκριμένη συναλλαγή με τη χρονική στιγμή της αρχής της συναλλαγής αυτής. Η χρονική αυτή σφραγίδα παράγεται είτε από το ρολόι του συστήματος είτε ενημερώνοντας την τιμή της κάθε φορά που εκκινεί κάποια συναλλαγή. Για κάθε συναλλαγή T_i υπάρχει η χρονική σφραγίδα της $TS(T_i)$.

Έστω ότι μία συναλλαγή επιθυμεί την ανάγνωση ή την ενημέρωση κάποιων δεδομένων στη ΒΔ. Η προσπέλαση στα δεδομένα αυτά επιτρέπεται μόνο αν η τελευταία ενημέρωση αυτών πραγματοποιήθηκε από παλαιότερη συναλλαγή, από κάποια δηλαδή συναλλαγή με μικρότερη χρονική σφραγίδα. Διαφορετικά, η συναλλαγή τερματίζεται και γίνεται η επανεκκίνησή της. Κατά την επανεκκίνηση αυτή δημιουργείται μία νέα χρονική σφραγίδα. Επιπλέον, χρονικές σφραγίδες δεν ορίζονται μόνο για τις συναλλαγές αλλά και για τα δεδομένα. Σε κάθε στοιχείο δεδομένων αντιστοιχεί μία χρονική *σφραγίδα ανάγνωσης* (*read timestamp*) και μία χρονική *σφραγίδα εγγραφής* (*write timestamp*). Ο συμβολισμός αυτών των σφραγίδων είναι *timestamp-R* και *timestamp-W* , αντίστοιχα. Συνεπώς, η χρονική σφραγίδα ανάγνωσης κάποιου στοιχείου δεδομένων είναι ίση με τη χρονική σφραγίδα της συναλλαγής που εκτέλεσε την ανάγνωση. Το ίδιο σκεπτικό ισχύει και για τη χρονική σφραγίδα εγγραφής των δεδομένων. Ο συμβολισμός αυτών των χρονικών σφραγίδων για ένα στοιχείο δεδομένων *D* είναι *timestamp-R(D)* και *timestamp-W(D)*, αντίστοιχα.

Πρωτόκολλο διάταξης χρονικών σφραγίδων

Το πρωτόκολλο αυτό εκτελεί τις λειτουργίες ανάγνωσης και αποθήκευσης δεδομένων από τις συναλλαγές, με βάση τις χρονικές σφραγίδες που δημιουργούνται κατά την εκτέλεσή τους. Έτσι, για μία συναλλαγή T_i και κάποιο στοιχείο δεδομένων *D*, έχουμε τις παρακάτω περιπτώσεις :

- Εκτέλεση λειτουργίας ανάγνωσης του στοιχείου *D* από τη συναλλαγή T_i .
    1. Αν ισχύει $TS(T_i)$ < timestamp-W(D), τότε η συναλλαγή διαβάζει τη τιμή του *D*, η οποία έχει αλλάξει από κάποια άλλη συναλλαγή. Στη περίπτωση αυτή η ανάγνωση διακόπτεται.

    2. Αν ισχύει $TS(T_i)$ ≥ timestamp-W(D), τότε η ανάγνωση ολοκληρώνεται και ενημερώνεται η τιμή timestamp-R(D) [timestamp-R(D) = max{ timestamp-R(D) , $TS(T_i)$ }].

- Εκτέλεση λειτουργίας αποθήκευσης του στοιχείου *D* από τη συναλλαγή T_i .
    1. Αν $TS(T_i)$ < timestamp-W(D), τότε η λειτουργία προσπαθεί να ενημερώσει μία τιμή που έχει ήδη αλλάξει από άλλη συναλλαγή. Έτσι, η αποθήκευση

τερματίζεται και εκτελείται οπισθοδρόμηση για την T_i (την λειτουργία της οπισθοδρόμησης θα την εξετάσουμε αναλυτικότερα στη πορεία). Ας σημειώσουμε εδώ πως, κατά τη εκτέλεση κάποιας οπισθοδρόμησης, δημιουργείται νέα χρονική σφραγίδα για τη συναλλαγή, και αυτή εκκινεί ξανά από την αρχή.

2. Αν $TS(T_i)$ < timestamp-R(D), τότε η τρέχουσα τιμή διαβάστηκε προηγουμένως από άλλη συναλλαγή. Συνεπώς, και εδώ η λειτουργία αποθήκευσης διακόπτεται και πραγματοποιείται οπισθοδρόμηση.

3. Αν δεν ισχύει καμία από τις παραπάνω περιπτώσεις, η αποθήκευση ολοκληρώνεται με επιτυχία και η τιμή της timestamp-W(D) ενημερώνεται [timestamp-W(D) = $TS(T_i)$].

7. Κατανεμημένες και εμφωλευμένες συναλλαγές (*distributed and nested transactions*)

Τα στοιχεία δεδομένων (*data items*) τα οποία απαιτούνται για την εκτέλεση μίας συναλλαγής μπορούν να είναι κατανεμημένα σε διαφορετικούς *διακομιστές* (*servers*). Συνεπώς, σε γενικότερο επίπεδο, μία συναλλαγή μπορεί να απαιτεί πολλούς διακομιστές για την ορθή εκτέλεσή της. Για παράδειγμα, μία συναλλαγή μπορεί να εκτελεί περισσότερες από μία λειτουργίες, οι οποίες με τη σειρά τους πρέπει να προσπελάσουν περισσότερους από έναν διακομιστές. Σε κάποιες άλλες περιπτώσεις, ο διακομιστής, ο οποίος προσπελάστηκε από μία συναλλαγή, πρέπει να εκτελέσει περαιτέρω προσπελάσεις σε άλλους διαφορετικούς και περισσότερους από έναν διακομιστές. Κάθε συναλλαγή η οποία απαιτεί και εμπλέκει τη συμμετοχή πολλαπλών διακομιστών για την εκτέλεση των λειτουργιών της, ονομάζεται *κατανεμημένη συναλλαγή* (*distributed transaction*).

Όταν μία κατανεμημένη συναλλαγή ολοκληρώνεται, η ιδιότητα της ατομικότητας των συναλλαγών απαιτεί πως είτε όλοι οι διακομιστές που εμπλέκονται θα ολοκληρώσουν τη συναλλαγή είτε όλοι θα την απορρίψουν. Για να επιτευχθεί κάτι τέτοιο, ένας από τους διακομιστές αναλαμβάνει να παίξει το ρόλο του συντονιστή (*coordinator*), όπου βεβαιώνει την κοινή κατάληξη της κατανεμημένης συναλλαγής σε όλους τους διακομιστές που συμμετέχουν. Ο τρόπος με τον οποίο ο συντονιστής θα πετύχει τον στόχο του ποικίλει ανάλογα με το πρωτόκολλο που χρησιμοποιείται σε κάθε περίπτωση. Όπως θα δούμε και στη

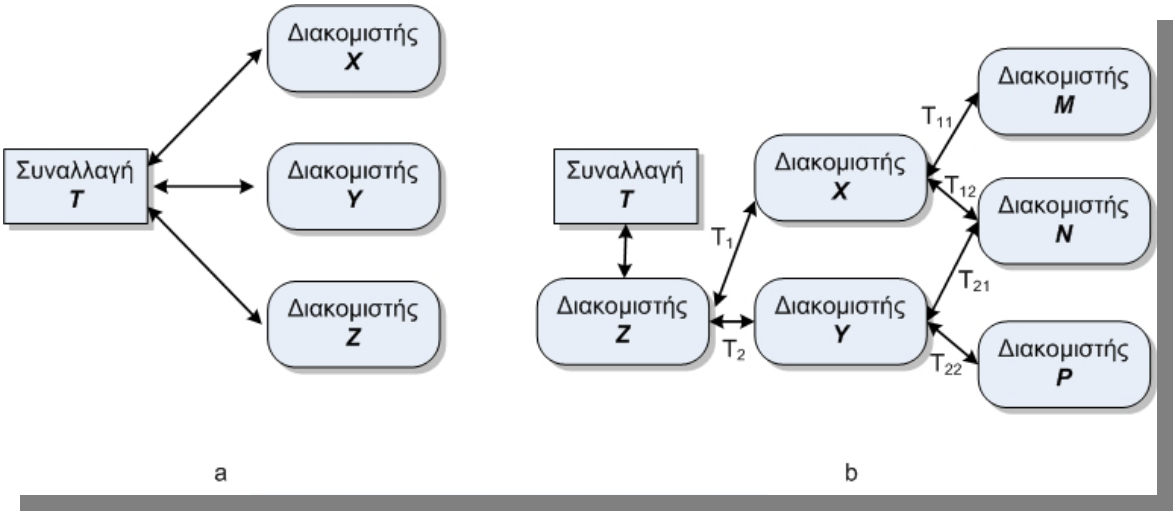
συνέχεια, το γνωστό *Πρωτόκολλο Ολοκλήρωσης Δύο Φάσεων* (*Two-Phase Commit Protocol*) χρησιμοποιείται πολύ συχνά. Το πρωτόκολλο αυτό επιτρέπει στους διακομιστές να επικοινωνούν μεταξύ τους έτσι ώστε να καταλήξουν στη λήψη της απόφασης για την ολοκλήρωσή ή την απόρριψη της κατανεμημένης συναλλαγής.

Όσον αφορά τον έλεγχο ταυτοχρονισμού στις κατανεμημένες συναλλαγές, κάθε διακομιστής εφαρμόζει τοπικά τεχνικές ελέγχου ταυτοχρονισμού στα δικά του στοιχεία δεδομένων, γεγονός που εξασφαλίζει την *τοπική* (*local*) σειριοποίηση των συναλλαγών. Οι κατανεμημένες όμως συναλλαγές πρέπει να είναι σειριοποιήσημες και στο γενικότερο τους *σύνολο* (*global*). Ο τρόπος με τον οποίο επιτυγχάνεται η σειριοποίηση αυτή ποικίλει από τη χρήση κλειδαριών μέχρι τη χρήση χρονοσφραγίδων ή άλλων τεχνικών ελέγχου ταυτοχρονισμού. Παρόλα αυτά, μπορεί μία κατανεμημένη συναλλαγή να σειριοποιήται επιτυχώς σε τοπικό επίπεδο, αλλά σε συνολικό επίπεδο, λόγω διαφόρων περιορισμών ή αλληλεξαρτήσεων, να μη μπορεί να σειριοποιηθεί.

Υπάρχουν γενικά δύο τρόποι με τους οποίους μπορεί να δομηθεί μία κατανεμημένη συναλλαγή: ως μία απλή κατανεμημένη συναλλαγή (*simple distributed transaction*) και ως μία *εμφωλευμένη συναλλαγή* (*nested distributed transaction*). Σε μία απλή κατανεμημένη συναλλαγή, η αρχική συναλλαγή εκτελεί αιτήσεις προσπέλασης σε περισσότερους από έναν διακομιστές, αλλά ο κάθε διακομιστής δεν απαιτεί προσπελάσεις σε άλλους διακομιστές. Για παράδειγμα, στο σχήμα 6a η συναλλαγή *T* είναι μία απλή κατανεμημένη συναλλαγή η οποία απαιτεί προσπελάσεις στους διακομιστές *X*, *Y* και *Z*. Σε μία τέτοιου είδους συναλλαγή, η κάθε προσπέλαση ολοκληρώνεται πριν εκτελεστεί η επόμενη προσπέλαση. Επομένως, κάθε συναλλαγή εκτελεί τις προσπελάσεις της στους διακομιστές που απαιτούνται σειριακά. Έτσι, σε περίπτωση που οι διακομιστές χρησιμοποιούν τεχνικές κλειδώματος, η συναλλαγή μπορεί να περιμένει για ένα μόνο στοιχείο δεδομένων τη φορά.

Σε πολλές περιπτώσεις, η εκτέλεση μιας λειτουργίας σε κάποιον διακομιστή μπορεί να απαιτεί την εκτέλεση κάποιας άλλης λειτουργίας σε διαφορετικό, όμως, διακομιστή (ο οποίος με τη σειρά του να απαιτεί την εκτέλεση λειτουργιών σε άλλον διακομιστή και ου τω καθεξής). Για να διαχειριστούμε την κατάσταση αυτή, κάθε τέτοιου είδους συναλλαγή δομείται ως ένα σύνολο εμφωλευμένων συναλλαγών. Το σχήμα 6b απεικονίζει μία συναλλαγή *T* η οποία απαιτεί την εκτέλεση λειτουργιών στον διακομιστή *Z*. Ο τελευταίος απαιτεί εκτέλεση λειτουργιών στους διακομιστές *X* και *Y*, οι οποίοι με τη σειρά τους απαιτούν εκτελέσεις λειτουργιών στους διακομιστές M,N και N,P αντίστοιχα. Γενικά, μία συναλλαγή αποτελείται από μία ιεραρχία εμφωλευμένων συναλλαγών. Εμφωλευμένες

συναλλαγές οι οποίες βρίσκονται στο ίδιο επίπεδο εμφώλευσης, μπορούν να εκτελεστούν συγχρόνως η μία με την άλλη.



Σχήμα 6

8. Ανάκτηση συναλλαγών (*transaction and data recovery*)

8.1. Εισαγωγή

Η *ανάκτηση* των δεδομένων αποτελεί μία σημαντικότατη λειτουργία για την ορθή διαχείριση της ΒΔ. Η λειτουργία αυτή απαιτείται σε περιπτώσεις βλάβης του συστήματος που προκαλούν την προσωρινή διακοπή λειτουργίας. Οι συχνότερες αιτίες που οδηγούν στη διακοπή αυτή του συστήματος είναι οι εξής :

- Πτώση συστήματος, λόγω εσφαλμένης λειτουργίας του υλικού ή του λογισμικού.
- Πρόβλημα στο μέσο αποθήκευσης, το οποίο μπορεί να οφείλεται στη καταστροφή του μέσου αποθήκευσης που χρησιμοποιείται, για παράδειγμα πρόβλημα στον σκληρό δίσκο.
- Σφάλμα λογισμικού εφαρμογής, το οποίο οφείλεται σε σφάλμα εκτέλεσης κατά τη διάρκεια ενημέρωσης των δεδομένων.
- Άλλα φυσικά αίτια, όπως για παράδειγμα σεισμοί ή πυρκαγιές.

Όποια και να είναι η αιτία για τη διακοπή της κανονικής λειτουργίας του συστήματος, το ΣΔΒΔ πρέπει να εκτελεί με ορθό τρόπο τη λειτουργία ανάκτησης με σκοπό να μειωθεί στο ελάχιστο η πιθανότητα οποιασδήποτε αλλοίωσης των δεδομένων. Το θέμα της ανάκτησης των δεδομένων θα το εξετάσουμε σε μεγαλύτερο βάθος στη πορεία της εργασίας.

Η ανάκτηση των συναλλαγών αφορά την εξασφάλιση της αποτυχημένης ατομικότητας σε περίπτωση εμφάνισης κάποιων αποτυχιών από την πλευρά των διακομιστών (*servers)*. Οι διακομιστές που παρέχονται στις συναλλαγές περιλαμβάνουν έναν *διαχειριστή ανάκτησης* (*recovery manager*), ο ρόλος του οποίου είναι να εξασφαλίζει πως τα αποτελέσματα των συναλλαγών πάνω στα στοιχεία δεδομένων που αποθηκεύονται στον διακομιστή. Μπορούν να επανακτηθούν σε περίπτωση που κάποια-ες συναλλαγές ή κάποιοι διακομιστές αποτύχουν να ολοκληρώσουν την εργασία τους. Ο *διαχειριστή ανάκτησης* αποθηκεύει τα στοιχεία δεδομένων σε μόνιμη μνήμη μαζί με λίστες προεκτάσεων για τα στοιχεία αυτά και πληροφοριών για την κατάσταση κάθε συναλλαγής. Παρακάτω θα εξετάσουμε δύο προσεγγίσεις :

- Την μόνιμη αποθήκευση (*stable storage*).

- Την οργάνωση του πρωτεύοντα διακομιστή (*primary server*) και την οργάνωση του εφεδρικού διακομιστή (*backup server*).

8.2. Ανάκτηση συναλλαγών (*transaction recovery*)

Η ιδιότητα της ατομικότητας των συναλλαγών απαιτεί πως τα αποτελέσματα όλων των ολοκληρωμένων συναλλαγών, και κανένα από τα αποτελέσματα των μη-ολοκληρωμένων ή αυτών που έχουν απορριφθεί, έχουν αντίκρισμα πάνω στα στοιχεία δεδομένων. Η ιδιότητα αυτή μπορεί να περιγραφεί από δύο διαφορετικές γωνίες : την *αντοχή* (*durability*) και την ατομικότητα *αποτυχίας* (*failure atomicity*). Η αντοχή απαιτεί πως τα στοιχεία δεδομένων αποθηκεύονται σε κάποια μόνιμη μνήμη και πως θα είναι διαθέσιμα οποιαδήποτε στιγμή. Επομένως, η επιβεβαίωση ενός συμμετέχοντα για ολοκλήρωση κάποιας συναλλαγής απαιτεί όλα τα αποτελέσματα της συναλλαγής αυτής πως θα αποθηκευτούν σε κάποια μόνιμη μνήμη. Η ατομικότητα αποτυχίας απαιτεί τα αποτελέσματα των συναλλαγών να είναι ατομικά, ακόμα και αν αποτύχει κάποιος διακομιστής. Η ανάκτηση αφορά την αντοχή των στοιχείων δεδομένων του διακομιστή και την παροχή ατομικότητας αποτυχίας.

Παρά το γεγονός πως οι διακομιστές αρχείων και οι διακομιστές ΒΔ διατηρούν τα στοιχεία δεδομένων σε μόνιμη μνήμη, κάποια άλλα είδη διακομιστών δεν χρειάζεται να λειτουργήσουν κατά τον παραπάνω τρόπο για να επιτύχουν ανάκτηση δεδομένων. Εδώ, θα υποθέσουμε πως, όταν ο διακομιστής βρίσκεται σε λειτουργία, διατηρεί όλα τα στοιχεία δεδομένων σε προσωρινή

μνήμη, και αποθηκεύει σε μόνιμη μνήμη τα αποτελέσματα των ολοκληρωμένων συναλλαγών πάνω στα στοιχεία δεδομένων, σε ένα *αρχείο ανάκτησης* (*recovery file*). Έτσι, η ανάκτηση αφορά την επαναφορά του διακομιστή στη τελευταία ή μη κατάσταση των στοιχείων δεδομένων του από τον μόνιμο χώρο αποθήκευσής τους.

Οι δύο απαιτήσεις για αντοχή και ατομικότητα αποτυχίας δεν εξαρτώνται σε μεγάλο βαθμό η μία από την άλλη, και μπορούν να επιτευχθούν με έναν απλό μηχανισμό, τον *διαχειριστή ανάκτησης*. Οι υποχρεώσεις ενός *διαχειριστή ανάκτησης* είναι οι εξής :

- Να αποθηκεύει τα στοιχεία δεδομένων σε μόνιμη μνήμη, όσο αφορά τις ολοκληρωμένες συναλλαγές,

- Να επαναφέρει την κατάσταση των στοιχείων δεδομένων μετά από μία αποτυχία κάποιου διακομιστή.

- Να αναδιοργανώνει το *αρχείο ανάκτησης* με σκοπό να βελτιώνεται η απόδοση της ανάκτησης

- Να αναδιοργανώνει τον χώρο αποθήκευσης για το *αρχείο ανάκτησης*.

8.3. Λίστες πορείας (*intentions lists*)

Κάθε διακομιστής, ο οποίος χρησιμοποιείται για την εκτέλεση οποιουδήποτε είδους συναλλαγών, πρέπει να διατηρεί την πορεία (να κρατά, δηλαδή, τα ίχνη) των στοιχείων δεδομένων που επεξεργάζονται από τις συναλλαγές αυτές. Για παράδειγμα, όταν ένας συμμετέχον πρόκειται να εκκινήσει μία συναλλαγή, όπως έχουμε δει, ο διακομιστής του αποδίδει ένα αναγνωριστικό (πχ *TID-Transaction Identification*) και του το αποστέλλει. Κάθε συμμετέχον που εκτελεί κάποια εργασία και την αποστέλλει στον διακομιστή, πρέπει να αποστέλλει και το αναγνωριστικό της συναλλαγής της οποίας την εργασία εκτέλεσε. Κατά την πορεία εκτέλεσης της συναλλαγής, οι λειτουργίες ανανέωσης της κατάστασής της εφαρμόζονται σε ένα «ιδιωτικό» σύνολο των στοιχείων δεδομένων, που σχετίζονται με την εκτέλεση της συναλλαγής.

Κάθε διακομιστής δημιουργεί μία *λίστα πορείας* για όλες τις τρέχοντες συναλλαγές σε εκτέλεση. Μία *λίστα πορείας* για μία συγκεκριμένη συναλλαγή περιέχει το ονόματα και τις τιμές των στοιχείων δεδομένων που σχετίζονται και αλλάζουν με την εκτέλεση της συγκεκριμένης συναλλαγής. Όταν μία συναλλαγή ολοκληρωθεί με επιτυχία, ο διακομιστής χρησιμοποιεί την *λίστα πορείας* της συναλλαγής με τα στοιχεία που διαφοροποιήθηκαν από αυτή. Η ολοκληρωμένη έκδοση του κάθε στοιχείου δεδομένων αντικαθίσταται με την τελευταία έκδοση, και οι νέες τιμές

των στοιχείων αποθηκεύονται στο *αρχείο ανάκτησης*. Σε περίπτωση που μία συναλλαγή απορριφθεί, ο διακομιστής χρησιμοποιεί ξανά την λίστα πορείας για να διαγράψει τις εκδόσεις των στοιχείων δεδομένων που άλλαξαν από την συναλλαγή αυτή.

Όπως έχουμε δει, για μία κατανεμημένη συναλλαγή, πρέπει να εκτελεστεί ένα πρωτόκολλο ολοκλήρωσης προτού αυτή ολοκληρωθεί ή απορριφθεί. Για την εξέταση της ανάκτησης θα βασιστούμε στο *πρωτόκολλο εκτέλεσης δύο-φάσεων* (*two-phase-commit protocol*), σύμφωνα με το οποίο όλοι οι διακομιστές που εμπλέκονται σε μία κατανεμημένη συναλλαγή δηλώνουν πρώτα αν είναι έτοιμοι να ολοκληρώσουν τις υπο-συναλλαγές τους, και αφού συμφωνήσουν όλοι, εκτελούν την εντολή ολοκλήρωσης. Διαφορετικά, οι διακομιστές απορρίπτουν τις υπο-συναλλαγές τους.

Στο σημείο όπου ο διακομιστής δηλώνει πως είναι έτοιμος να ολοκληρώσει την σχετική υπο-συναλλαγή, πρέπει να έχει αποθηκεύσει την *λίστα πορείας* και τα στοιχεία δεδομένων της στο *αρχείο ανάκτησης*, έτσι ώστε να μπορεί να εκτελέσει την ολοκλήρωση ακόμα και αν αργότερα υπάρξει κάποια αποτυχία στον διακομιστή.

Όταν όλοι οι διακομιστές συμφωνήσουν να ολοκληρώσουν τις υπο-συναλλαγές τους, ο συντονιστής ενημερώνει τους συμμετέχοντες και αποστέλλει μηνύματα σε όλους με την εντολή ολοκλήρωσης. Μόλις ο συμμετέχον ενημερωθεί για την ολοκλήρωση, τα *αρχεία ανάκτησης* των διακομιστών που συμμετέχουν στην υπο-συναλλαγή πρέπει να περιέχουν όλες τις απαραίτητες πληροφορίες για να εξασφαλιστεί πως η υπο-συναλλαγή ολοκληρώνεται από όλους τους διακομιστές, ακόμα και αν κάποιοι από αυτούς αποτύχουν στο διάστημα που μεσολαβεί ανάμεσα στην προετοιμασία και την ολοκλήρωση.

8.4. Καταχωρήσεις στο *αρχείο ανάκτησης*

Για να εφαρμοστεί η ανάκτηση σε έναν διακομιστή ο οποίος συμμετέχει σε μία κατανεμημένη συναλλαγή, επιπλέον πληροφορίες πρέπει να αποθηκευτούν στο *αρχείο ανάκτησης*, πέρα από τα ονόματα και τις τιμές που παίρνουν τα στοιχεία δεδομένων, τα οποία σχετίζονται με την συναλλαγή. Οι πληροφορίες αυτές αφορούν την κατάσταση της συναλλαγής, άσχετα με το αν έχει ολοκληρωθεί, απορριφθεί ή προετοιμάζεται να ολοκληρωθεί. Επιπλέον, κάθε στοιχείο δεδομένων σε ένα *αρχείο ανάκτησης* συσχετίζεται με μία συγκεκριμένη υπο-συναλλαγή με το να αποθηκευτεί η *λίστα πορείας* της στο *αρχείο ανάκτησης*. Γενικά, το αρχείο ανάκτησης περιέχει τους παρακάτω τύπους καταχωρήσεων :

Τύπος καταχώρησης	Περιγραφή περιεχομένων καταχώρησης
Στοιχείο δεδομένων	Η τιμή ενός στοιχείου δεδομένων
Κατάσταση συναλλαγής	*Αναγνωριστικό* συναλλαγής, κατάσταση συναλλαγής (*προετοιμασία*, *ολοκληρωμένη*, *απορρίφθηκε*) και άλλες πληροφορίες για το πρωτόκολλο ολοκλήρωσης
Λίστα πορείας	*Αναγνωριστικό* συναλλαγής και ένα σύνολο λειτουργιών κατά την πορεία της συναλλαγής, αποτελούμενο από <*Αναγνωριστικό* στοιχείου δεδομένων>, <θέση της τιμής του στοιχείου στο *αρχείο ανάκτησης*>.

Σχήμα 1

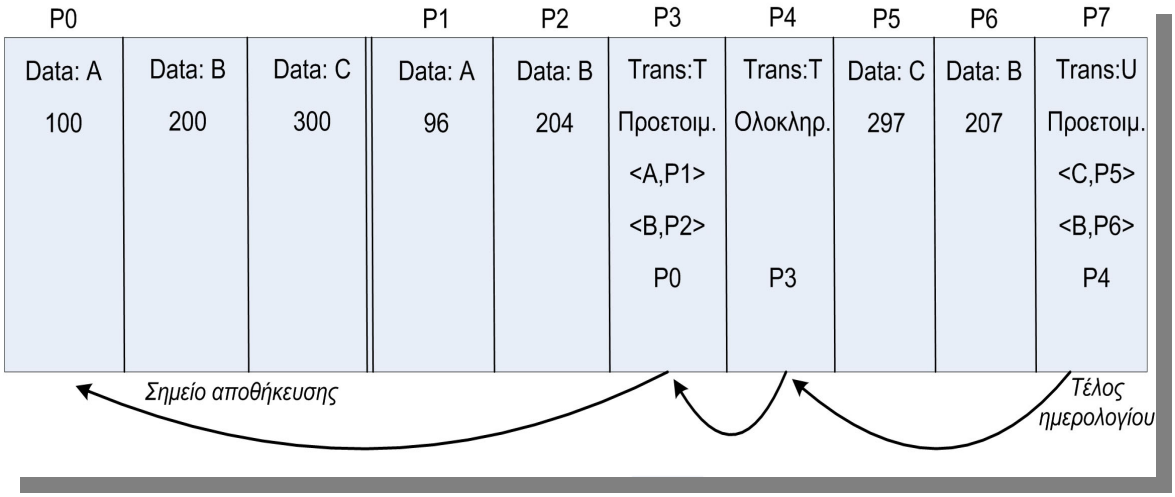
Στη συνέχεια θα παρουσιάσουμε δύο προσεγγίσεις για τη χρήση των *αρχείων ανάκτησης*, τη χρήση *ημερολογίου* (*logging*) και την *έκδοση σκιών* (*shadow versions*).

8.5. Χρήση *ημερολογίου* για την ανάκτηση συναλλαγών (*logging*)

Στην τεχνική του *ημερολογίου*, το *αρχείο ανάκτησης* αναπαριστά ένα ημερολόγιο όλων των συναλλαγών που πραγματοποιήθηκαν από έναν διακομιστή. Το ιστορικό του αποτελείται από τις τιμές των στοιχείων δεδομένων, καταχωρήσεις σχετικά με την κατάσταση της συναλλαγής και τις *λίστες πορείας* των συναλλαγών αυτών. Η σειρά κατάταξης των καταχωρήσεων στο *ημερολόγιο* αντικατοπτρίζει την σειρά με την οποία οι συναλλαγές προετοιμάστηκαν, ολοκληρώθηκαν ή απορρίφθηκαν στον διακομιστή. Σε πρακτικό επίπεδο, το αρχείο ανάκτησης περιέχει ένα πρόσφατο στιγμιότυπο όλων των τιμών των στοιχείων δεδομένων στον διακομιστή συνοδευμένο με το ιστορικό των συναλλαγών.

Κατά τη διάρκεια της κανονικής λειτουργίας του διακομιστή, ο *διαχειριστής ανάκτησής* του καλείται όταν μία συναλλαγή προετοιμάζεται, ολοκληρώνεται ή απορρίπτεται. Όταν ο διακομιστής προετοιμάζεται να ολοκληρώσει μία συναλλαγή, ο *διαχειριστής ανάκτησης* προσάπτει όλα τα στοιχεία δεδομένων της *λίστας πορείας* στο *αρχείο ανάκτησης*, μαζί με την παρούσα κατάσταση της συναλλαγής . Όταν η συναλλαγή πρόκειται να ολοκληρωθεί ή να απορριφθεί, ο *διαχειριστής ανάκτησης* προσάπτει την ανάλογη κατάσταση της συναλλαγής στο *αρχείο ανάκτησής* του.

Ο *διαχειριστής ανάκτησης* αναθέτει ένα αναγνωριστικό σε κάθε ένα από τα στοιχεία δεδομένων έτσι ώστε οι επιτυχής εκδόσεις των στοιχείων στο *αρχείο ανάκτησης* να σχετίζονται με τα στοιχεία δεδομένων που βρίσκονται στον διακομιστή. Για παράδειγμα, σε ένα τραπεζικό σύστημα, οι τιμές των τραπεζικών συναλλαγών μπορεί να είναι στοιχεία ενός πίνακα, και τα αναγνωριστικά τους να είναι οι δείκτες αυτού του πίνακα.. Το σχήμα 2 απεικονίζει έναν μηχανισμό *ημερολογίου* για τις συναλλαγές T και U .



Σχήμα 2

Το *ημερολόγιο* έχει πρόσφατα αναδιοργανωθεί και οι καταχωρήσεις στα αριστερά της διπλής γραμμής αποτελούν ένα στιγμιότυπο των τιμών των μεταβλητών *A, B* και *C*, πριν εκκινήσουν οι συναλλαγές T και U . Τα γράμματα *A, B* και *C* χρησιμοποιούνται ως μοναδικά αναγνωριστικά των στοιχείων δεδομένων. Παρουσιάζουμε μία περίπτωση όπου η συναλλαγή T έχει ολοκληρωθεί, και η U έχει προετοιμαστεί αλλά δεν έχει ολοκληρωθεί. Όταν η συναλλαγή T προετοιμάζεται να ολοκληρωθεί, οι τιμές των μεταβλητών *A* και *B* γράφονται στις θέσεις *P1* και *P2* στο *ημερολόγιο*, μαζί με την κατάσταση της συναλλαγής και τη *λίστα πορείας* της (<A,P1>, <B,P2>). Όταν η συναλλαγή T ολοκληρωθεί, η καταχώρηση της κατάστασης της συναλλαγής γράφεται στη θέση *P4* του *ημερολογίου*. Όταν η συναλλαγή U προετοιμάζεται να ολοκληρωθεί, οι τιμές των στοιχείων C και B γράφονται στη θέση *P5* και *P6* του *ημερολογίου*, μαζί με την κατάσταση της συναλλαγής και τη *λίστα πορείας* της (<C,P5>, <B,P6>).

Κάθε καταχώρηση κατάστασης μιας συναλλαγής, περιέχει έναν δείκτη προς το *αρχείο ανάκτησης* της προηγούμενης καταχώρησης κατάστασης συναλλαγής, έτσι ώστε να μπορεί ο *διαχειριστής ανάκτησης* να ακολουθεί με αντίστροφο τρόπο τις καταχωρήσεις αυτές μέσα στο *αρχείο ανάκτησης*.

8.5.1 Ανάκτηση των στοιχείων δεδομένων με τη χρήση *ημερολογίου*

Όταν ένας διακομιστής επανεκκινείται, αρχικοποιεί πρώτα τις τιμές των στοιχείων δεδομένων του και τις προωθεί στον *διαχειριστή ανάκτησης*. Ο *διαχειριστή ανάκτησης* είναι υπεύθυνος να επαναφέρει τα στοιχεία δεδομένων του διακομιστή, ώστε να περιέχονται όλα τα αποτελέσματα όλων των ολοκληρωμένων συναλλαγών, με τη σωστή σειρά, και κανένα από τα αποτελέσματα των μη ολοκληρωμένων ή των συναλλαγών που απορρίφθηκαν.

Οι πιο πρόσφατες πληροφορίες για τις συναλλαγές βρίσκονται στο τέλος του *ημερολογίου*. Συνεπώς, ο *διαχειριστή ανάκτησης* θα επαναφέρει την κατάσταση των στοιχείων δεδομένων διαβάζοντας από τέλος προς τα πίσω το *αρχείο ανάκτησης*. Χρησιμοποιεί τις συναλλαγές με κατάσταση συναλλαγής *ολοκληρωμένη* για να επαναφέρει τα στοιχεία τα οποία δεν έχουν επανακτηθεί ακόμα, και συνεχίζει μέχρι να επανακτηθούν όλα τα στοιχεία του διακομιστή.

Για να επανακτήσει τα αποτελέσματα μιας συναλλαγής, ο *διαχειριστής ανάκτησης* λαμβάνει τη σχετική *λίστα πορείας* από το *αρχείο ανάκτησής* της. Η *λίστα πορείας* περιέχει τα αναγνωριστικά και τις θέσεις στο *αρχείο ανάκτησης*, για όλες τις τιμές των στοιχείων δεδομένων που τροποποιήθηκαν από τη συναλλαγή αυτή.

Σε περίπτωση που για κάποιο λόγο ο διακομιστής αποτύχει, ο *διαχειριστής* του *ανάκτησης* επαναφέρει την κατάσταση των στοιχείων δεδομένων του ως εξής: Ξεκινάει από την τελευταία καταχώρηση κατάστασης συναλλαγής στο *ημερολόγιο* (στη *P7*), και αφού δει πως η συναλλαγή U δεν έχει ακόμα ολοκληρωθεί, αγνοεί τα αποτελέσματά της. Έπειτα, μετακινείται στην προηγουμένη καταχώρηση κατάστασης συναλλαγής στο *ημερολόγιο* (στη *P4*) και βλέπει πως η συναλλαγή T έχει ολοκληρωθεί. Για να καλύψει τα στοιχεία δεδομένων τα οποία έχουν επηρεαστεί από τη συναλλαγή αυτή, μετακινείται στην προηγούμενη καταχώρηση κατάστασης συναλλαγής στο *ημερολόγιο* (στη *P3*) και βρίσκει τη *λίστα πορείας* για τη συναλλαγή T , <A,P1>, <B,P2>. Τότε, επαναφέρει τις τιμές των στοιχείων δεδομένων *A* και *B* από τις τιμές *P1* και *P2*. Αφού δεν έχει ανακτήσει ακόμα τη τιμή του στοιχείου *C*, μετακινείται στη θέση *P0* του *ημερολογίου* , που αποτελεί *σημείο αποθήκευσης*, και επαναφέρει και το *C*.

Παρ'όλ'αυτά, ο διακομιστής μπορεί να αποτύχει κατά τη διάρκεια της διαδικασίας ανάκτησης. Είναι σημαντικό η ανάκτηση να είναι μοναδική, από τη άποψη πως μπορεί να πραγματοποιηθεί πολλές φορές με τα ίδια όμως πάντα αποτελέσματα. Αυτό επιτυγχάνεται εδώ με τη υπόθεσή μας, πως όλα τα στοιχεία δεδομένων αποθηκεύονται σε μία μόνιμη μνήμη. Έτσι, ο *διαχειριστής ανάκτησης* θα επαναφέρει τα στοιχεία δεδομένων από την μόνιμη αυτή μνήμη. Αν κατά τη διάρκεια της διαδικασίας ο διακομιστής αποτύχει, τα στοιχεία προς ανάκτηση θα βρίσκονται ακόμα στη μνήμη αυτή.

8.6. Αναδιοργανώνοντας το *αρχείο ανάκτησης*

Ο *διαχειριστής ανάκτησης* είναι υπεύθυνος, πέρα από τα άλλα, να αναδιοργανώνει το *αρχείο ανάκτησής* του, έτσι ώστε να κάνει τη διαδικασία ανάκτησης αποδοτικότερη και γρηγορότερη, και για να μειώνει τον χώρο που αυτό χρησιμοποιεί. Αν το *αρχείο ανάκτησης* δεν έχει ποτέ αναδιοργανωθεί, τότε πρέπει η διαδικασία ανάκτησης να ψάξει προς τα πίσω στο *αρχείο ανάκτησης*, μέχρι να βρει μία τιμή για όλα τα στοιχεία δεδομένων που χρειάζεται. Συνεπώς, η

μόνη πληροφορία που διατίθεται για την ανάκτηση είναι ένα αντίγραφο των στοιχείων των ολοκληρωμένων εκδόσεων στον διακομιστή. Ο όρος *δημιουργία σημείου αποθήκευσης* (*check pointing*) χρησιμοποιείται εδώ για να δηλώσει τη διαδικασία εγγραφής της τρέχουσας κατάστασης των τιμών των στοιχείων του διακομιστή σε ένα νέο *αρχείο ανάκτησης*, μαζί με τις καταχωρίσεις των καταστάσεων των συναλλαγών και των λιστών πορείας αυτών. Ο όρος *σημείο αποθήκευσης* (*check point*) χρησιμοποιείται για να αναφερθούμε στη πληροφορία που αποθηκεύεται κατά την παραπάνω διαδικασία. Ο σκοπός της δημιουργίας *σημείων αποθήκευσης* είναι να μειώσουμε τον αριθμό των συναλλαγών οι οποίες λαμβάνουν μέρος στη διαδικασία ανάκτησης και για να απαιτηθεί νέος χώρος αποθήκευσης.

Τα *σημεία αποθήκευσης* μπορούν να δημιουργούνται μετά την ανάκτηση, αλλά πριν εκκινήσει κάθε νέα συναλλαγή. Υπάρχει όμως η περίπτωση της μη συχνής εμφάνισης ανάκτησης. Συνεπώς, είναι συνετό να δημιουργούνται *σημεία αποθήκευσης* κατά τη διάρκεια της κανονικής δραστηριότητας ενός διακομιστή. Το *σημείο αποθήκευσης* γράφεται σε ένα νέο *αρχείο ανάκτησης* και το παρών *αρχείο ανάκτησης* παραμένει σε χρήση μέχρι αυτό να ολοκληρωθεί. Ένα *σημείο αποθήκευσης* αποτελείται από τη δημιουργία ενός σημείου-σημάδι (*mark*) που εισάγεται στο *αρχείο ανάκτησης* όταν αυτό ξεκινά, έπειτα γράφονται τα στοιχεία δεδομένων του διακομιστή στο νέο *αρχείο ανάκτησης*, και γράφονται όλα τα στοιχεία μετά το σημείο αυτό στο μελλοντικό *αρχείο ανάκτησης*.

Το σύστημα ανάκτησης μπορεί να μειώσει τον χώρο που χρησιμοποιεί διαγράφοντας το παλιό *αρχείο ανάκτησης*. Όταν ο διαχειριστής ανάκτησης εκτελεί τη διαδικασία ανάκτησης, μπορεί να συναντήσει ένα σημείο αποθήκευσης μέσα στο *αρχείο ανάκτησης* . Αν συμβεί αυτό, ο διαχειριστής ανάκτησης μπορεί αμέσως να ανακτήσει όλα τα σχετικά στοιχεία δεδομένων με τη χρήση αυτού του *σημείου αποθήκευσης*.

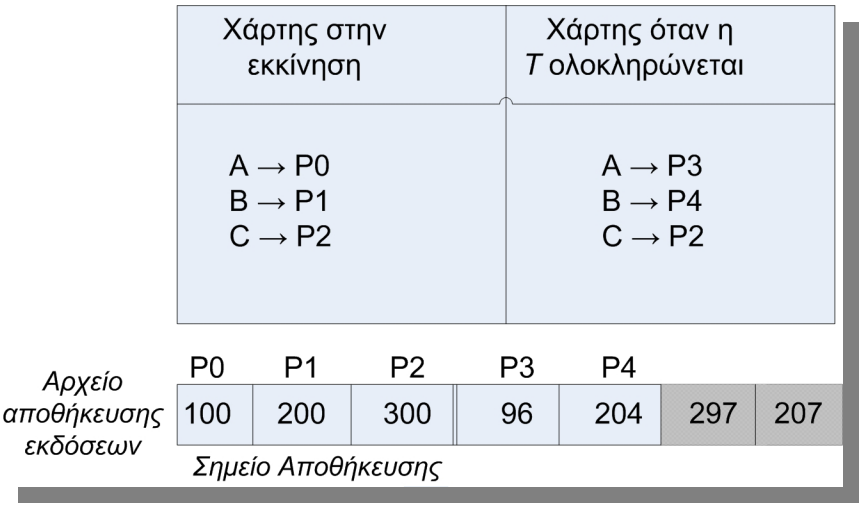
8.7. Ανάκτηση των στοιχείων δεδομένων με τη χρήση *εκδόσεων σκιών* (*shadow versions*)

Σύμφωνα με την τεχνική ανάκτησης με τη χρήση *ημερολογίου*, οι καταχωρήσεις κατάστασης των συναλλαγών, οι λίστες πορείας και τα στοιχεία δεδομένων αποθηκεύονται όλα στο ίδιο αρχείο, στο *ημερολόγιο*. Η τεχνική *εκδόσεων σκιών* είναι ένας εναλλακτικός τρόπος οργάνωσης του *αρχείου ανάκτησης*. Η μέθοδος αυτή, χρησιμοποιεί έναν *χάρτη* (*map*) για να εντοπίσει τις εκδόσεις των στοιχείων δεδομένων του διακομιστή σε ένα αρχείο το οποίο ονομάζεται *αρχείο αποθήκευσης εκδόσεων (version store)*. Ο *χάρτης* συσχετίζει τα αναγνωριστικά των στοιχείων δεδομένων με τις θέσεις των τρεχόντων εκδόσεών τους στο *αρχείο αποθήκευσης εκδόσεων*. Οι εκδόσεις που γράφονται από κάθε συναλλαγή, είναι «σκιές» των προηγούμενων ολοκληρωμένων εκδόσεων. Οι

καταχωρήσεις της κατάστασης συναλλαγών και οι λίστες πορείας τους μεταχειρίζονται ξεχωριστά. Θα εξετάσουμε πρώτα τις εκδόσεις σκιών.

Όταν μία συναλλαγή προετοιμάζεται να ολοκληρωθεί, κάθε στοιχείο δεδομένων που τροποποιείται από αυτή αποθηκεύεται στο *αρχείο αποθήκευσης σκιών*, αφήνοντας έτσι τις συσχετιζόμενες ολοκληρωμένες εκδόσεις τους ανέπαφες. Αυτές οι νέες μη ολοκληρωμένες εκδόσεις ονομάζονται *εκδόσεις σκιών*. Όταν μία συναλλαγή ολοκληρωθεί, ένας νέος *χάρτης* δημιουργείται με την αντιγραφή του παλιού και τη προσθήκη σε αυτόν των *εκδόσεων σκιών*. Για να τελειώσει η ολοκλήρωση, ο νέος αυτός *χάρτης* παίρνει τη θέση του παλιού.

Για να ανακτηθούν τα στοιχεία δεδομένων όταν ο διακομιστής επανεκκινείται, ο *διαχειριστής ανάκτησής* του διαβάζει τον *χάρτη* και χρησιμοποιεί τις πληροφορίες που του παρέχονται από αυτόν για να εντοπίσει τα στοιχεία δεδομένων στο *αρχείο αποθήκευσης εκδόσεων*. Η τεχνική αυτή περιγράφεται στο σχήμα 3, όπου έχουμε τις συναλλαγές T και U . Η πρώτη στήλη του πίνακα δείχνει τον *χάρτη* πριν από την εκτέλεση των δύο συναλλαγών, όταν οι τιμές των στοιχείων δεδομένων *A*, *B* και *C* είναι 100, 200 και 300, αντίστοιχα. Η δεύτερη στήλη του πίνακα δείχνει τον χάρτη μετά την ολοκλήρωση της συναλλαγής T .



Σχήμα 3

Το αρχείο αποθήκευσης εκδόσεων περιέχει ένα σημείο αποθήκευσης, ακολουθούμενο από τις εκδόσεις των A και B, στις θέσεις P3 και P4, που δημιουργήθηκαν από τη συναλλαγή T . Περιέχει, επίσης, τις εκδόσεις σκιών των στοιχείων B και C, που δημιουργήθηκαν από τη συναλλαγή U .

Ο χάρτης πρέπει πάντα να αποθηκεύεται σε ένα γνωστό σημείο (για παράδειγμα στην αρχή του αρχείου αποθήκευσης εκδόσεων ή σε ένα ξεχωριστό αρχείο), έτσι ώστε να μπορεί να βρεθεί εύκολα και γρήγορα σε περίπτωση πραγματοποίησης μιας λειτουργίας ανάκτησης. Για να γίνει η

μετάβαση από τον παλιό χάρτη στον καινούριο πρέπει να γίνει με ένα απλό αλλά ατομικό βήμα. Για να επιτευχθεί αυτό, ο χάρτης πρέπει να είναι αποθηκευμένος σε μία μόνιμη μνήμη, έτσι ώστε να εγγυηθεί η χρήση ενός έγκυρου πίνακα, ακόμα και όταν η λειτουργία εγγραφής του αποτύχει.

Η μέθοδος των *εκδόσεων σκιών* παρέχει γρηγορότερη ανάκτηση σε σύγκριση με τη χρήση της μεθόδου του *ημερολογίου*. Αυτό συμβαίνει γιατί στη πρώτη περίπτωση, οι θέσεις των στοιχείων δεδομένων που έχουν ολοκληρωθεί βρίσκονται στον χάρτη, σε αντίθεση με το *ημερολόγιο*, όπου η ανάκτηση απαιτεί να γίνει αναζήτηση των στοιχείων σε όλη την έκταση του *ημερολογίου*. Παρ'όλ'αυτά, η μέθοδος του *ημερολογίου* είναι πιο γρήγορη από τις *εκδόσεις σκιών*, κατά την κανονική λειτουργία του συστήματος, επειδή η χρήση του *ημερολογίου* απαιτεί επεξεργασία και λειτουργίες πάνω σε ένα και μόνο αρχείο, ενώ η μέθοδος των *εκδόσεων σκιών* απαιτεί την επεξεργασία πιθανός πρόσθετων αρχείων και μνήμης (σε περίπτωση, για παράδειγμα, που οι *χάρτες* είναι αποθηκευμένοι σε διαφορετικά αρχεία σε σχέση με το *αρχείο αποθήκευσης εκδόσεων*).

8.8. Η αναγκαιότητα για τη χρήση των καταχωρήσεων *κατάσταση συναλλαγής* και *λίστα πορείας* στο *αρχείο ανάκτησης*

Είναι δυνατό να σχεδιάσουμε ένα απλό *αρχείο ανάκτησης* το οποίο δεν θα περιέχει καταχωρήσεις για την κατάσταση των συναλλαγών αλλά ούτε και τις *λίστες πορείας* τους. Ένα *αρχείο ανάκτησης* τέτοιας μορφής μπορεί να χρησιμοποιηθεί, και είναι μάλιστα ιδανικό για χρήση, σε περίπτωση που όλες οι συναλλαγές δρομολογούνται σε έναν απλό διακομιστή, για εκτέλεση σε μία μόνο τοποθεσία. Παρ'όλ'αυτά, η χρήση των καταχωρήσεων *κατάσταση συναλλαγής* και της *λίστας πορείας* της, είναι απαραίτητες για έναν διακομιστή που δρομολογεί κατανεμημένες συναλλαγές σε διαφορετικές τοποθεσίες. Η προσέγγιση αυτή μπορεί να φανεί χρήσιμη ακόμα και για διακομιστές μη κατανεμημένων συναλλαγών για πολλούς και ποικίλους λόγους. Μερικοί από τους λόγους αυτούς είναι οι εξής :

- Κάποιοι διαχειριστές ανάκτησης είναι σχεδιασμένοι να γράφουν τα στοιχεία δεδομένων στο *αρχείο ανάκτησης* πρόωρα, με την υπόθεση πως η συναλλαγή θα ολοκληρωθεί κανονικά.

- Αν οι συναλλαγές χρησιμοποιούν μεγάλο αριθμό στοιχείων δεδομένων, η ανάγκη για την γειτονική εγγραφή τους στο *αρχείο ανάκτησης* μπορεί να βελτιώσει και να τελειοποιήσει ακόμα τη σχεδίαση του διακομιστή.

- Στα πρωτόκολλα ελέγχου συνέπειας στα οποία χρησιμοποιείται η μέθοδος των *χρονικών σφραγίδων* (*timestamp ordering concurrency control*), ο διακομιστής μερικές φορές γνωρίζει πως μία συναλλαγή πρόκειται να ολοκληρωθεί και ενημερώνει τον

συμμετέχοντα. Στο σημείο αυτό, τα στοιχεία δεδομένων εγγράφονται στο *αρχείο ανάκτησης*. Παρά το γεγονός του ότι πρόκειται να ολοκληρωθεί, μία συναλλαγή μπορεί να χρειαστεί να περιμένει, όπως έχουμε δει, για να ολοκληρωθεί μέχρι να έχουν ολοκληρωθεί κάποιες προηγούμενες συναλλαγές. Σε μία τέτοια περίπτωση, η σχετική καταχώρηση κατάσταση συναλλαγής στο *αρχείο ανάκτησης* θα περιμένει για να ολοκληρωθεί, και έπειτα θα ολοκληρωθεί για να εξασφαλίσει τη σειρά των *χρονικών σφραγίδων* των ολοκληρωμένων συναλλαγών στο *αρχείο ανάκτησης*. Κατά τη διάρκεια της λειτουργίας ανάκτησης, κάθε συναλλαγή σε κατάσταση αναμονής επιτρέπεται να ολοκληρωθεί επειδή αυτές που περίμεναν μπορεί είτε να έχουν μόλις ολοκληρωθεί είτε να μην πρέπει να απορριφθούν λόγω κάποιας αποτυχίας της λειτουργίας του διακομιστή.

8.9. Ανοχή των συναλλαγών και των κατανεμημένων συστημάτων σε σφάλματα

Κάθε συστατικό μέρος ενός συστήματος αποτελείται, γενικά, από ένα σύνολο λογισμικού και υλικού, από το οποίο κάποιο μπορεί να αποτύχει για διαφόρους λόγους κάποια χρονική στιγμή. Για παράδειγμα, ένα ελαττωματικό κομμάτι του υλικού μπορεί να εκτελέσει ακόμα και ένα σωστό πρόγραμμα σε μία, λάθος όμως, χρονική στιγμή. Το γεγονός του ότι ένα κατανεμημένο σύστημα αποτελείται από διεργασίες οι οποίες εκτελούνται συγχρόνως σε διάφορες τοποθεσίες που ενώνονται μεταξύ τους με κάποιο δίκτυο, οδηγεί σε δύο σημαντικά σημεία για τη σχεδίαση υπηρεσιών με τη παρουσία άλλων ελαττωματικών συστατικών :

1. Η λειτουργία μιας υπηρεσίας σε ένα κατανεμημένο σύστημα γενικά βασίζεται στη λειτουργία άλλων υπηρεσιών οι οποίες εκτελούνται σε διαφορετικές τοποθεσίες. Οι τελευταίες υπηρεσίες κάποιες φορές καταλήγουν σε αποτυχία ανταπόκρισης, είτε επειδή το σύστημα απέτυχε είτε επειδή η επικοινωνία, το δίκτυο δηλαδή, δεν είναι έμπιστο και αποτελεσματικό. Επιπλέον, είναι δύσκολο για μία υπηρεσία να αναγνωρίσει αν μία άλλη έχει πραγματικά αποτύχει ή είναι υπερφορτωμένη και καθυστερεί υπερβολικά, από την άποψη της χρονικής διάρκειάς της.

2. Ένα σύνολο υπηρεσιών σε διαφορετικές τοποθεσίες μπορούν να συνδυαστούν με τέτοιον τρόπο ώστε η ομαδική τους εκτέλεση να είναι λιγότερο πιθανή προς αποτυχία.

Το πρώτο σημείο μας δείχνει ότι οι σχεδιαστές κατανεμημένων συστημάτων πρέπει να λάβουν υπόψη τους πως κάποιες υπηρεσίες μπορούν να αποτύχουν με διάφορους τρόπους. Το δεύτερο σημείο μας δείχνει πως οι σχεδιαστές κατανεμημένων συστημάτων μπορούν να εκμεταλλευτούν τη δυνατότητα των πολλαπλών συστημάτων να «καλύψουν» (*mask*) ενδεχόμενα λάθη του συστήματος.

Και στις δύο παραπάνω περιπτώσεις, οι σχεδιαστές κατανεμημένων συστημάτων πρέπει να γνωρίζουν το εύρος των ενδεχομένων λαθών που μπορεί να προκύψουν κατά τη λειτουργία των συστημάτων αυτών. Αυτό σημαίνει πως δεν πρέπει μόνο να ορίσουν τη σωστή λειτουργία των συστημάτων που σχεδιάζουν, αλλά επίσης και τη λανθασμένη. Η περιγραφή των τρόπων με τους οποίους μία υπηρεσία μπορεί να αποτύχει ονομάζεται *σημασιολογία αποτυχίας*. Η γνώση αυτή μπορεί να οδηγήσει στη δημιουργία κάποιων υπηρεσιών που να μπορούν να καλύψουν τη λανθασμένη συμπεριφορά κάποιων άλλων υπηρεσιών. Ένα σύστημα ανεκτικό σε σφάλματα μπορεί είτε να προβλέψει τα ενδεχόμενα λάθη και να αποτύχει είτε να καλύψει τα λάθη αυτά. Μία *υπηρεσία ανοχής σφαλμάτων* λειτουργεί σύμφωνα με έναν προκαθορισμένο τρόπο σε περίπτωση εμφάνισης λαθών. Ένας διακομιστής καλύπτει (*mask*) τα σφάλματα είτε με το να τα αποκρύπτει είτε με το να τα μετατρέπει σε μία από τις γνωστές μορφές λαθών να τα εκτελεί όπως έχει οριστεί.

8.9.1. Τα χαρακτηριστικά των σφαλμάτων

Με σκοπό να ορίσουμε τη *σημασιολογία αποτυχίας* πρέπει να έχουμε κάποιον τρόπο ώστε να περιγράφουμε τα σφάλματα. Ο Cristian [Cristian, “Characteristics of faults”, 1991] παραθέτει μία πολύ χρήσιμη κατάταξη των σφαλμάτων. Ο Cristian υποθέτει πως, για να εκτελείται μία υπηρεσία με ορθό τρόπο πρέπει να είναι σωστά και τα αποτελέσματα του διακομιστή αλλά και η ανταπόκριση των συμμετεχόντων. Ένα μέρος της κατάταξης που πρότεινε είναι η εξής:

Κλάση σφάλματος	Υπο-κλάση σφάλματος	Περιγραφή
Σφάλμα παράλειψης		Ο διακομιστής παραλείπτει να ανταποκριθεί σε μία αίτηση
Σφάλμα ανταπόκρισης		Ο διακομιστής ανταποκρίνεται με λάθος τρόπο σε μία αίτηση
	Σφάλμα τιμής	Επιστροφή λανθασμένης τιμής
	Σφάλμα μετάβασης	Λάθος αποτέλεσμα στα δεδομένα (για παράδειγμα, λάθος ανάθεση τιμής σε ένα στοιχείο δεδομένων)

Σχήμα 4

8.9.1.1. Τα *χρονικά σφάλματα* (*timing failure*)

Η κατάταξη σύμφωνα με τον Cristian, ασχολείται και με τα *χρονικά σφάλματα*. Τα σφάλματα αυτά αναφέρονται σε οποιαδήποτε ανταπόκριση η οποία δεν είναι διαθέσιμη μέσα σε ένα προκαθορισμένο χρονικό διάστημα. Μια χρονική αποτυχία μπορεί να περιγράψει την ανταπόκριση η οποία είναι είτε πολύ καθυστερημένη είτε πολύ πρόωρη, σε σχέση με το παραπάνω χρονικό διάστημα. Οι υπηρεσίες που χρησιμοποιούνται με κάποιου είδους χρονικό περιορισμό ,πρέπει ανακτηθούν επίσης στο καθορισμένο χρονικό διάστημα. Επιπλέον, δεν πρέπει να βασίζονται σε πρωτόκολλα όπως το πρωτόκολλο ατομικής εκτέλεσης, τα οποία στη χειρότερη

περίπτωση μπορεί να χρειαστούν χρονικό διάστημα το οποίο δεν θα περιορίζεται από κάποιο ανώτατο ή κατώτατο όριο.

8.9.1.2. Τα σφάλματα λόγω αποτυχίας του διακομιστή (*server crash failure*)

Ο Cristian ορίζει ένα σφάλμα λόγω αποτυχίας του διακομιστή ως ένα επαναλαμβανόμενο σφάλμα παράλειψης. Μία τέτοιου είδους αποτυχία προκαλεί στον διακομιστή παύση της αποστολής μηνυμάτων έτσι ώστε να φανεί στους συμμετέχοντες μιας συναλλαγής πως σταμάτησε. Έτσι, οι συμμετέχοντες δεν μπορούν να ξεχωρίσουν και να καταλάβουν αν ο διακομιστής απέτυχε ή αν ο διακομιστής ανταποκρίνεται πολύ αργά, ή ακόμα αν υπάρχει κάποιο σφάλμα στη μεταξύ τους σύνδεση. Οι *χρονικοί περιορισμοί* (*time-outs*) και η επαναποστολή μηνυμάτων χρησιμοποιούνται γενικά για την ανίχνευση και ανακάλυψη σφαλμάτων λόγω αποτυχίας του διακομιστή.

Μία σημαντική όψη ενός διακομιστή μετά την επανεκκίνησή του (μετά από τη αποτυχία του για διάφορους λόγους), αποτελεί η κατάσταση στην οποία θα βρεθεί. Για παράδειγμα, ένας τυπικός διακομιστής επανεκκινείται ένα σημείο σύμφωνα με τα στοιχεία δεδομένων των ήδη ολοκληρωμένων συναλλαγών (καλύπτει δηλαδή όλα τα στοιχεία δεδομένων που έχουν τροποποιηθεί από τις ολοκληρωμένες συναλλαγές). Ο Cristian δίνει την παρακάτω κατάταξη των σφαλμάτων σε σχέση με την αποτυχία του διακομιστή :

Κλάση σφάλματος	Υπο-κλάση σφάλματος	Περιγραφή
Σφάλμα αποτυχίας(crash)	*Αποτυχία αμνησίας*	Επαναλαμβανόμενο σφάλμα παράλειψης: Ο διακομιστής αποτυγχάνει επανηλημένα να ανταποκριθεί σε αιτήσεις μέχρι να επαννακινηθεί. Ο διακομιστής επανέρχεται στην αρχική του κατάσταση, έχοντας ξεχάσει τη κατάστασή του τη στιγμή της αποτυχίας του.
	Αποτυχία παύσης	Ο διακομιστής εκκινεί από τη κατάσταση πριν την αποτυχία του.
	Αποτυχία σταματήματος	Ο διακομιστής δεν επανεκκινήται ποτέ.

Σχήμα 5

Η αποτυχία αμνησίας (*amnesia crash*) είναι πολύ χειρότερη από το επαναλαμβανόμενο σφάλμα παράλειψης, επειδή ο διακομιστής χάνει την κατάστασή του (χάνονται, για παράδειγμα, οι τρέχουσες τιμές των στοιχείων δεδομένων του). Η λειτουργία της ανάκτησης μπορεί εδώ να

χρησιμοποιηθεί για να εξασφαλιστεί πως αν ο διακομιστής αποτύχει, δεν θα πέσει σε συμπεριφορά αμνησίας , όπως παραπάνω.

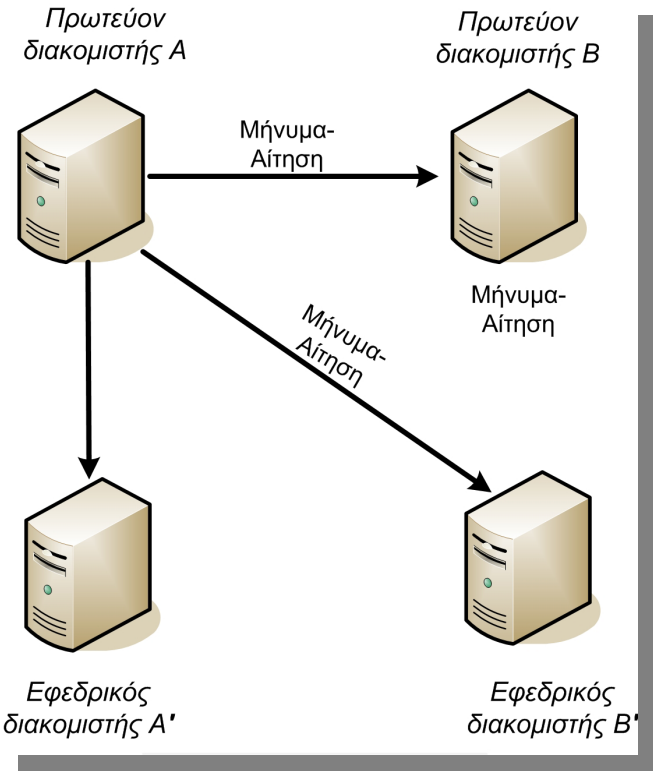
Η πρόβλεψη της *υπηρεσίας ανοχής σφαλμάτων* μπορεί να απλοποιηθεί αν υποθέσουμε πως οι υπηρεσίες αποτυγχάνουν με «καθαρό» τρόπο. Αυτό σημαίνει πως, ένας διακομιστής είτε θα λειτουργεί με τον ορθό τρόπο είτε θα αποτυγχάνει. Ένας *διακομιστής σταματήματος σε περίπτωση αποτυχίας* (***fail-stop server***) είναι ο διακομιστής ο οποίος όταν πρόκειται να αποτύχει για οποιονδήποτε λόγο, μεταβαίνει σε μία κατάσταση από όπου επιτρέπει στους υπόλοιπους διακομιστές να εντοπίσουν την αποτυχία του, και έπειτα σταματά τη λειτουργία του. Οι συμμετέχοντες σε αυτόν τον διακομιστή μπορούν να υποθέσουν πως δεν θα υπάρχει κανένα σφάλμα τιμής, παράλειψης ή χρονικό σφάλμα. Μία αποτυχία στην ανταπόκριση του διακομιστεί οδηγεί μόνο στη αποτυχία του. Έτσι, οι συμμετέχοντες μπορούν να εντοπίσουν οποιαδήποτε αποτυχία λόγω της επαναλαμβανόμενης αποτυχίας του διακομιστή να ανταποκριθεί στις αιτήσεις τους.

8.10. Οργάνωση *πρωτεύον διακομιστή* (*primary server*) και *εφεδρικού διακομιστή* (*backup server*).

Πολλά συστήματα (για παράδειγμα το σύστημα *Stratus*[Cristian, 1991]) είναι σχεδιασμένα με τέτοιο τρόπο ώστε να παρέχουν ανοχή σε σφάλματα με τη χρήση υλικού (hardware), το οποίο χρησιμοποιείται για τη διατήρηση ενός αντιγράφου ολόκληρου του χώρου μνήμης που βρίσκεται υπό επεξεργασία. Μία τέτοιου είδους υπηρεσία περιγράφεται από ένα ζεύγος διακομιστών, με τον καθένα τους να παρέχει υπηρεσίες σε περίπτωση αποτυχίας του συστήματος. Οι αιτήσεις των συμμετεχόντων εκτελούνται και από τους δύο διακομιστές, και η ανταπόκριση μπορεί να προέρχεται από οποιονδήποτε από αυτούς. Σε περίπτωση που ένας από τους διακομιστές, για κάποιο λόγο αποτύχει στη λειτουργία του, ο άλλος συνεχίζει να παρέχει τις υπηρεσίες που πρέπει.

Η διατήρηση αυτή ενός αντιγράφου του υλικού καθιστά τη λειτουργία της ανάκτησης άμεση. Αν το σύστημα χρησιμοποιείται για περιπτώσεις πραγματικού χρόνου (για παράδειγμα παρακολούθηση των παλμών ενός ασθενή ή παρακολούθηση ενός αισθητήρα), τότε είναι σημαντικό το να μπορεί η ανάκτηση να πραγματοποιηθεί μέσα σε ένα προκαθορισμένο χρονικό διάστημα. Πολλές εφαρμογές μπορούν να είναι περισσότερο ανεκτικές σε κάποια καθυστέρηση στη διαδικασία της ανάκτησης. Σε τέτοιες περιπτώσεις, είναι προτιμότερο να σχεδιαστεί με τέτοιο τρόπο το ζεύγος πρωτεύον-εφεδρικού διακομιστή, έτσι ώστε ο δεύτερος να είναι ανενεργός κατά τη διάρκεια της κανονικής λειτουργίας του πρώτου, κατά το μεγαλύτερο δηλαδή διάστημα. λειτουργίας του συστήματος. Αυτό δίνει τη δυνατότητα χρήσης του εφεδρικού διακομιστή για κάποια άλλη, περισσότερο παραγωγική εργασία.

Ένα μήνυμα το οποίο περιέχει κάποια αίτηση από ένα πρωτεύον διακομιστή σε έναν άλλο, πάντα αποστέλλεται προς τρεις κατευθύνσεις : στον εφεδρικό διακομιστή του αποστολέα, στον πρωτεύον διακομιστή-παραλήπτη, και στον εφεδρικό του τελευταίου. Το σχήμα 6 απεικονίζει την αποστολή ενός μηνύματος από τον πρωτεύοντα διακομιστή *A* σε έναν άλλο πρωτεύοντα *B*, το οποίο παράλληλα αποστέλλεται τόσο στον εφεδρικό του *A* (*A'*), όσο και στον εφεδρικό του *B* (*B'*).



Σχήμα 6

Κάθε μήνυμα-αίτηση που αποστέλλεται τρεις φορές, όπως είδαμε παραπάνω, αποστέλλεται με πλήρως ατομικό τρόπο. Το γεγονός αυτό εξασφαλίζει πως τόσο ο πρωτεύον όσο και ο εφεδρικός διακομιστής λαμβάνουν τα ίδια ακριβώς μηνύματα με την ίδια επίσης σειρά. Επίσης, με τον τρόπο αυτό, ο εφεδρικός διακομιστής του αποστολέα-διακομιστή μπορεί και μετρά τον αριθμό των μηνυμάτων που αυτός έστειλε. Κάθε διακομιστής, σε ένα τέτοιο σύστημα, εκτελεί κάποια ενέργεια με τη λήψη ενός μηνύματος-αίτησης. Η ενέργεια αυτή καθορίζεται από τον «ρόλο» της, όπως φαίνεται στο σχήμα 7 :

Ρόλος ενέργιας	Περιγραφή ενέργιας
Πρωτεύον	Εκτελείται η λειτουργία που ζητήθηκε και αποστέλλεται. η *απάντηση/ανταπόκριση*
Εφεδρικός	Αποθηκεύεται το μήνυμα σε ένα *ημερολόγιο* για χρήση του σε περίπτωση μελλοντικής ανάκτησης.
Εφεδρικός αποστολέα	Μετριέται ο αριθμός των μηνυμάτων από τον αποστολέα, από το σημείο που δημιουργήθηκε το τελευταίο *σημείο αποθήκευσης*.

Σχήμα 7

Ο πρωτεύον διακομιστής και ο εφεδρικός του συγχρονίζονται συχνά με σκοπό να αποτραπεί ο εφεδρικός από το να μείνει ανενημέρωτος, σε σχέση με τον πρωτεύοντα. Ο συγχρονισμός αυτό περιλαμβάνει τα παρακάτω:

- Ο πρωτεύον διακομιστής δημιουργεί ένα *σημείο αποθήκευσης*, αποθηκεύοντας την κατάσταση των στοιχείων δεδομένων του για χρήση από τον εφεδρικό του.

- Ο πρωτεύον διακομιστής ενημερώνει τον εφεδρικό του για τη δημιουργία του *σημείου αποθήκευσης* και έπειτα ο εφεδρικός διαγράφει όλα τα μηνύματα του από το ημερολόγιο (αφού είναι πλέον συγχρονισμένος με τον πρωτεύοντα) και θέτει τον μετρητή των μηνυμάτων που στάλθηκαν από τον πρωτεύοντα, μετά το τελευταίο *σημείο αποθήκευσης*, στην τιμή μηδέν.

Ο συγχρονισμός πρωτεύοντα-εφεδρικού διακομιστή μπορεί να γίνεται αυτόματα μετά από το πέρας ενός συγκεκριμένου χρονικού διαστήματος, ή όποτε το *ημερολόγιο* μηνυμάτων του εφεδρικού φτάσει σε κάποιο μέγιστο μέγεθος.

Όταν ο πρωτεύον διακομιστής αποτύχει για οποιονδήποτε λόγο, ο εφεδρικός αρχικοποιεί τα στοιχεία δεδομένων του από το πιο πρόσφατο *σημείο αποθήκευσης* και έπειτα εκτελεί τα μηνύματα-αιτήσεις τα οποία είναι αποθηκευμένα στο *ημερολόγιό* του. Καθώς ο εφεδρικός διακομιστής εκτελεί κάθε μήνυμα-αίτηση, θα παραλείψει τα μηνύματα που εστάλησαν ξανά από τον πρωτεύοντα διακομιστή χρησιμοποιώντας τον μετρητή μηνυμάτων από το τελευταίο *σημείο αποθήκευσης*. Τέλος, ο εφεδρικός διακομιστής διαγράφει το *ημερολόγιο* μηνυμάτων του, θέτει τον μετρητή μηνυμάτων από το τελευταίο σημείο αποθήκευσης μηδέν και παίζει πλέον τον ρόλο του πρωτεύοντα διακομιστή.

Κεφάλαιο 2

Μοντέλα ελέγχου προσπέλασης βασισμένα σε πλέγματα

(Lattice-Based Access Control models)

1. Εισαγωγή

Κατασκευαστές και χρήστες ηλεκτρονικών συστημάτων κατάλαβαν την ανάγκη για την ασφάλεια της πληροφορίας με την άφιξη των πρώτων υπολογιστικών συστημάτων που υποστήριζαν πολλούς χρήστες. Η ανάγκη αυτή απόκτησε ακόμα μεγαλύτερη σημασία καθώς τα συστήματα αυτά εξελίχθηκαν από απομονωμένους υπολογιστές σε αποκεντρωμένα αλλά και συνδεδεμένα συστήματα. Η ασφάλεια της πληροφορίας έχει τρεις ξεχωριστούς αλλά αλληλένδετους στόχους :

1. *Εμπιστευτικότητα* (ή *μυστικότητα*), σε σχέση με την αποκάλυψη του περιεχομένου της πληροφορίας,
2. *Ακεραιότητα*, σε σχέση με την τροποποίηση του περιεχομένου της πληροφορίας, και
3. *Διαθεσιμότητα* (availability), σε σχέση με την άρνηση της πρόσβασης στο περιεχόμενο της πληροφορίας.

Αυτοί οι στόχοι εμφανίζονται στην πράξη σε οποιοδήποτε πληροφοριακό σύστημα. Σε ένα σύστημα πληρωμών, για παράδειγμα, η εμπιστευτικότητα σχετίζεται με το να εμποδίσει σε έναν υπάλληλο να μάθει τον μισθό του αφεντικού του, η ακεραιότητα στο να εμποδίσει τον υπάλληλο να αλλάξει το ποσό του μισθού του, και η διαθεσιμότητα στο να εξασφαλίσει ότι όλες οι πληρωμές γίνονται στη χρονική στιγμή που είναι αυτές ορισμένες, δηλ. στην ώρα τους.

Οι Bell και LaPadula [D.E Bell and LaPadula “Secure Computer Systems, foundations and models”, 1975] ανέπτυξαν ένα σύνολο από μοντέλα ελέγχου προσπέλασης βασισμένα σε πλέγματα με σκοπό να χειριστούνε την ροή των πληροφοριών μέσα στα υπολογιστικά (πληροφοριακά) συστήματα.. Η εμπιστευτικότητα και η ακεραιότητα είναι οι κεντρικές απαιτήσεις στην ροή των πληροφοριών. Για αυτόν το λόγο, τα μοντέλα αυτά είναι κυρίως επικεντρωμένα στην εμπιστευτικότητα και μπορούν να αντεπεξέλθουν και σε κάποιες απαιτήσεις ακεραιότητας.

Οι Bell , LaPadula και Denning [D.E Denning “A Lattice Model of Secure Information Flow”, 1976] έκαναν την έρευνα σε αυτό το πεδίο κατά τη δεκαετία του ’70. Από τότε τα

μοντέλα αυτά έχουν βρει χρήση σε πολλά συστήματα, κυρίως για τις ανάγκες της εθνικής άμυνας των ΗΠΑ. Παρ' όλ' αυτά η θεωρία και οι γενικές αρχές είναι εφαρμόσιμες σε κάθε περίπτωση που σχετίζεται με την ροή πληροφοριών. Ο οικονομικός τομέας έχει μοναδικές πολιτικές για την ροή αυτή. Έτσι γίνεται απόλυτα κατανοητό ότι ο έλεγχος προσπέλασης στις πληροφορίες είναι ένα από τα σημαντικότερα συστατικά για την ασφάλεια των υπολογιστικών συστημάτων.

2. Η ροή της πληροφορίας

Οι πολιτικές για την ροή της πληροφορίας αφορούν ροή πληροφορίας από την μία κλάση ασφάλειας σε μία άλλη. Σε ένα σύστημα, η πληροφορία ρέει από ένα αντικείμενο σε ένα άλλο. Τα μοντέλα με τα οποία θα ασχοληθούμε χειρίζονται την έννοια «αντικείμενο» ως ένα γενικής σημασίας όρο. Ένα αντικείμενο μπορεί να οριστεί ως μία δεξαμενή πληροφορίας. Τυπικά παραδείγματα τέτοιων αντικειμένων είναι οι φάκελοι, και τα αρχεία σε ένα λειτουργικό σύστημα, και οι πίνακες σε μία βάση δεδομένων.

Η ροή της πληροφορίας ελέγχεται συνήθως με το να αναθέτουμε σε κάθε αντικείμενο μία κλάση ασφάλειας, η οποία καλείται και ετικέτα ασφάλειας. 'Οποτε η πληροφορία ρέει από ένα αντικείμεχο x σε ένα αντικείμενο y, υπάρχει μία συνοδευτική ροή πληροφορίας από την κλάση ασφάλειας x στη κλάση ασφάλειας y. Στο εξής, όταν αναφερόμαστε σε κάποια ροή πληροφορίας από μία κλάση ασφάλειας A σε μία άλλη B, πρέπει να φανταζόμαστε μία ροή από ένα αντικείμενο με ετικέτα A σε ένα άλλο με ετικέτα B.

O Denning έχει ορίσει μία πολιτική ροής πληροφορίας ως εξής :

Ορισμός 1 [Πολιτική ροής πληροφορίας]

Μία τριπλέτα $< SC, \rightarrow, \oplus >$, όπου SC είναι ένα σύνολο από κλάσεις ασφάλειας,

$\rightarrow \subseteq$ SC × SC είναι μία δυαδική σχέση πάνω στο SC × SC που δηλώνει πως είναι δυνατή η ροή της πληροφορίας , και $\oplus$: SC × SC → SC είναι ένας τελεστής ένωσης κλάσεων (class-combining or join operator) πάνω στο SC.

Και τα τρία συστατικά μίας πολιτικής ροής πληροφορίας είναι σταθερά, δηλαδή δεν αλλάζουν με τη πάροδο του χρόνου. Αυτός ο ορισμός επιτρέπει τα αντικείμενα να μπορούν να δημιουργηθούν και να καταστραφούν δυναμικά (όπως θα ήταν αναμενόμενο σε χρήσιμα συστήματα). Παρ'όλ'αυτά, οι κλάσεις ασφάλειας δεν μπορούν να δημιουργηθούν και να καταστραφούν και αυτές δυναμικά.

Είναι πιο βολικό και εύκολο να χρησιμοποιούμε κάποια σημειογραφία για την δυαδική σχέση $\rightarrow$, έτσι ώστε γράφοντας $A \rightarrow B$ (που είναι όμοιο με το $(A,B) \in \rightarrow$) να εννοούμε ότι η πληροφορία μπορεί να ρέει από το Α στο Β. Γράφουμε επίσης ότι $A \not\rightarrow B$ (που είναι όμοιο με το $(A, B) \notin \rightarrow$) , για να δηλώσουμε ότι η ροή της πληροφορίας από το Α στο Β δεν είναι δυνατή. Με άλλα λόγια, η πληροφορία μπορεί να ρέει από μία κλάση ασφάλειας Α σε μία άλλη κλάση Β, δοσμένης πολιτικής ροής πληροφορίας, αν και μόνο αν $A \rightarrow B$.

Όμοια, σημειογραφία χρησημοποιήται και για τον τελεστή ένωσης κλάσεων $\oplus$, έτσι ώστε $A \oplus B = \Gamma$ να σημαίνει $\oplus (A, B) = \Gamma$. Ο τελεστής ένωσης κλάσεων ασφάλειας καθορίζει τον τρόπο με τον οποίο θα βάλουμε μία ετικέτα ασφάλειας σε μία πληροφορία που αποκτήθηκε με τον συνδυασμό δύο κλάσεων ασφάλειας . Έτσι, η έκφραση $A \oplus B = \Gamma$ μας λέει πως τα αντικείμενα που περιέχουν πληροφορία από τις κλάσεις ασφάλειας Α και Β πρέπει να φέρουν ετικέτα της κλάσης ασφάλειας Γ.

Ορισμός 2 [Τα αξιώματα του Denning]

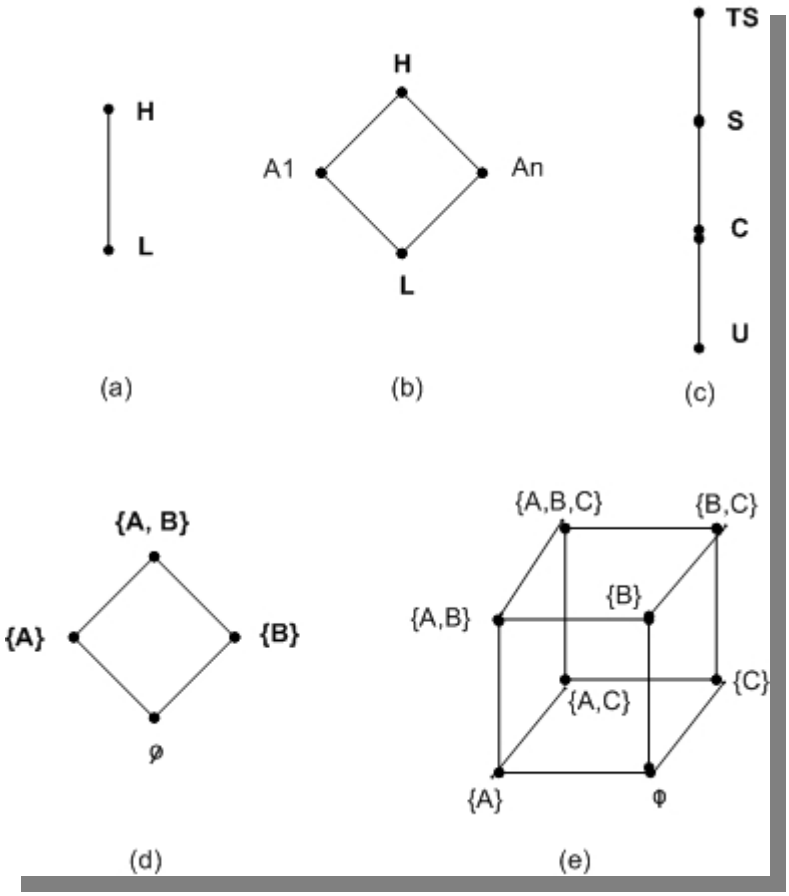
1. Το σύνολο των κλάσεων ασφάλειας *SC* είναι πεπερασμένο.
2. Η ροή πληροφορίας (ή καλύτερα η δυνατότητα ροής) $\rightarrow$ είναι μία μερική διάταξη πάνω στο *SC*.
3. Ο τελεστής ένωσης $\oplus$ είναι ένας ολικά ορισμένος τελεστής ελάχιστου υπέρτατου ορίου.

Ορισμός 3 [Επικράτηση]

Γράφουμε $A \geq B$ (διαβάζεται το Α επικρατεί στο Β) αν και μόνον αν $B \rightarrow A$. Η απόλυτη (*strict dominance* ή *sdom*) επικράτηση $>$ ορίζεται ως $A > B$ και μόνο αν $A \geq B$ και $A \neq B$. Λέμε επίσης πως τα Α και Β μπορούν να συγκριθούν αν $A \geq B$ ή $B \geq A$,αλλιώς τα Α και Β δεν μπορούν να συγκριθούν.

3. Το «στρατιωτικό πλέγμα» (The military lattice)

Τα πιο απλά παραδείγματα πολιτικών ροής πληροφορίας εμφανίζονται όταν έχουμε ολική ή γραμμική διάταξη των κλάσεων ασφάλειας. Το πιο κοινό παράδειγμα ολικής διάταξης κλάσεων ασφάλειας είναι η *TS* για άκρως απόρρητο (top secret), *S* για απόρρητο (secret) , *C* για εμπιστευτικό (confidential) και *U* για μη απόρρητο (unclassified) επίπεδο ασφάλειας που συναντούνται σε στρατιωτικούς και κυβερνητικούς τομείς. (Σχήμα 1). Γενικά, όμως μπορούμε να έχουμε οποιοδήποτε αριθμό από ολικές ή γραμμικές διατάξεις κλάσεων ασφάλειας.



Σχήμα 1

Πρέπει να σημειώσουμε ότι το ≥ (ή επικράτηση) είναι μία ολική διάταξη αν και μόνο αν το αντίστροφό του → (ή δυνατή ροή πληροφορίας) είναι ολικά διατεταγμένο. Ο ορισμός τότε του A ⊕ B είναι απλά το μεγαλύτερο από τα A και B όσο αφορά την επικράτηση. Με άλλα λόγια, όταν η πληροφορία δύο κλάσεων ασφάλειας συνδυαστεί , η μεγαλύτερη ετικέτα των δύο, θα χρησιμοποιηθεί για το αποτέλεσμα. Για παράδειγμα, S ⊕ U = S .

Στο σχήμα 1d βλέπουμε ένα μερικώς διατεταγμένο πλέγμα. Οι κλάσεις ασφάλειας ορίζονται ως το δυναμοσύνολο (δηλ. το σύνολο όλων των υποσυνόλων) του {A, B}. Έστω ότι το A δηλώνει πληροφορίες πληρωμής και το B δηλώνει ιατρικές πληροφορίες σε μία βάση δεδομένων προσωπικού. Η χαμηλή κλάση του συστήματος (system-low) είναι ένα κενό σύνολο, το οποίο μπορεί να περιέχει public πληροφορίες για το προσωπικό. Οι ετικέτες ασφάλειας {A} και {B} είναι μονοσύνολα (singleton sets), τα οποία αναφέρονται στις μισθοδοτικές και ιατρικές πληροφορίες αντίστοιχα. Όταν συνδυαστούν μισθοδοτικές και ιατρικές πληροφορίες, το αποτέλεσμα θα έχει την ετικέτα {A, B}. Σημειώνουμε εδώ ότι το {A} και {B} δεν μπορούν να συγκριθούν μεταξύ τους και ότι {A} ⊕ {B} = {A. B}. Σε αυτή τη πολιτική, η δυνατότητα ροής πληροφοριών → είναι όμοια με τη σχέση του υποσυνόλου,

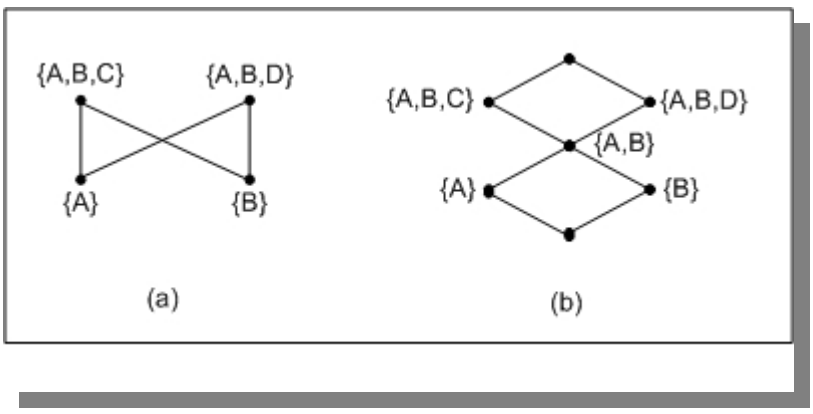
η επικράτηση με το υπερσύνολο και ο τελεστής ένωσης $\oplus$ όμοιος με την τομή των ετικετών. Ένα τέτοιο πλέγμα ονομάζεται πλέγμα-υποσυνόλου (subset lattice).

Στον στρατιωτικό και κυβερνητικό τομέα, τα στοιχεία των συνόλων (δηλ. τα A και B) είναι γνωστά ως *κατηγορίες* (*categories*) και οι *κλάσεις ασφάλειας* (που είναι σύνολα από κατηγορίες) ως *τμήματα* (*compartments*).

Το σχήμα 1e απεικονίζει ένα πλέγμα υποσυνόλου με τρεις κατηγορίες A, B, C που μπορούν να αναφέρονται σε μισθοδοτικές, ιατρικές και εκπαιδευτικές πληροφορίες αντίστοιχα. Σε αυτή τη περίπτωση οι κλάσεις ασφάλειας {A} και {B} έχουν δύο άνω όρια, το {A, B}και το {A, B, C}, με το πρώτο να είναι το κατώτερο άνω όριο.

Όμοια, είναι δυνατό να οριστούν πλέγματα υποσυνόλου κάθε μεγέθους. Παρά το ότι υπάρχουν 2^n υποσύνολα για ένα σύνολο μεγέθους n, υπάρχει μία εκθετική αύξηση του στον αριθμό των κλάσεων ασφάλειας, καθώς ο αριθμός των κατηγοριών αυξάνεται. Στη πράξη, μόνο ένα μικρό μέρος των κλάσεων ασφάλειας θα χρησιμοποιηθεί για ένα μεγάλο n.

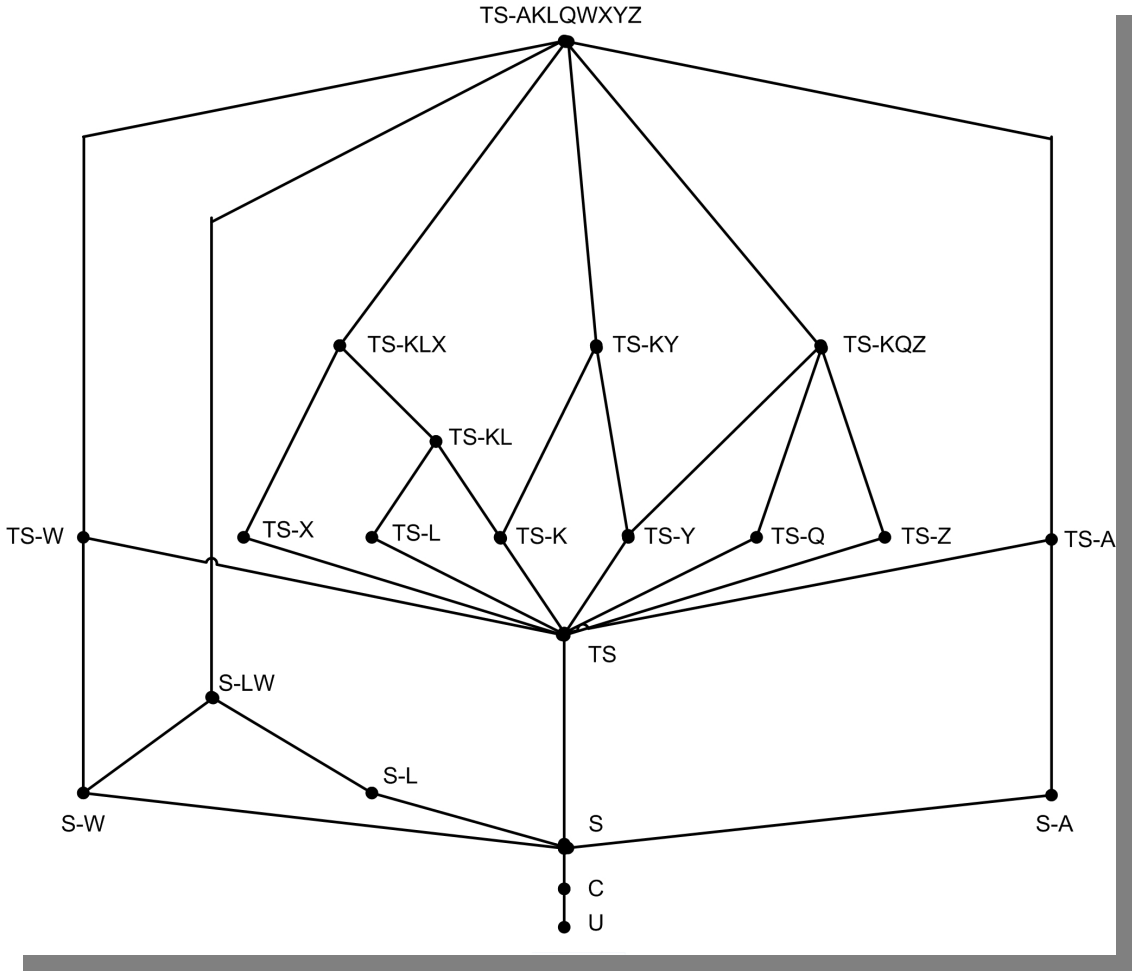
Επιλέγοντας ένα αυθαίρετο υποσύνολο από ένα πλέγμα, δεν σημαίνει απαραίτητα ότι σχηματίζεται ένα πλέγμα. Για παράδειγμα, η μερική διάταξη του σχήματος 2a προκύπτει επιλέγοντας τις τέσσερεις αυτές κλάσεις ασφάλειας από το πλέγμα υποσυνόλου στο {A, B, C, D}. Η μερική διάταξη του 2a αποτυγχάνει να χαρακτηριστεί ως πλέγμα για δύο λόγους. Πρώτον, λείπουν οι χαμηλές και υψηλές κλάσεις του συστήματος (system-low, system-high). Δεύτερον, τα δύο άνω όρια των {A} και {B} δεν μπορούν να συγκριθούν. Παρ' όλ' αυτά, δεν υπάρχει κατώτερο άνω όριο των {A} και {B}. Με το να συμπληρώσουμε αυτές τις κλάσεις ασφάλειας που λείπουν, μπορούμε να αποκτήσουμε τη διάταξη του σχήματος 2a με σκοπό να καταλήξουμε στο πλέγμα το σχήματος 2b. Μία τέτοια δυνατότητα είναι πάντοτε εφικτή για κάθε μερική διάταξη.



Σχήμα 2

Δύο πλέγματα, σαν αυτά που συζητήθηκαν παραπάνω, είναι δυνατό να συνδυαστούνε. Τότε κάθε κλάση ασφάλειας έχει δύο συστατικά: ένα από το ολικά διατεταγμένο πλέγμα ασφάλειας του σχήματος 1c, και ένα από το πλέγμα υποσυνόλου πάνω σε ένα πλήθος κατηγοριών. Μία ετικέτα τότε επικρατεί σε σχέση με μία άλλη, αν κάθε συστατικό της πρώτης επικρατεί σε σχέση με το αντίστοιχο συστατικό της δεύτερης. Για παράδειγμα, <TS,{A}> επικρατεί σε σχέση με την <S,{A}>, αλλά δεν μπορεί να συγκριθεί με την < S,{B}>. Η ένωση των δύο ετικετών ασφάλειας ορίζεται απλά ως η ένωση των συστατικών τους (π.χ. < TS,{A}>⊕< S,{B}>=< TS,{A,B}>). Είναι εύκολο να δούμε πως το αποτέλεσμα είναι ένα πλέγμα. Αυτό το πλέγμα είναι γνωστό ως παράγωγο πλέγμα των δύο υπό εξέταση πλεγμάτων. Το παράδειγμα αυτό αποδεικνύει τον γενικό κανόνα ότι το παράγωγο δύο πλεγμάτων είναι και αυτό ένα πλέγμα.

Με αυτόν τον τρόπο είναι δυνατό να παράγουμε πολύ μεγάλα πλέγματα. Όπως αναφέραμε και προηγουμένως, μόνο ένα μικρό υποσύνολο από ολόκληρο το πλέγμα στην πραγματικότητα θα χρησιμοποιηθεί σε τέτοιες περιπτώσεις. Ο Smith [G.W. Smith, “The Modeling and Representation of Security Semantics for Database Applications”, 1990] κατασκεύασε ένα τέτοιο πλέγμα με βάση τη θεωρία που χρησιμοποιείται στον στρατιωτικό τομέα. Το πλέγμα αυτό αποτελείται από τις τέσσερεις γραμμικά διατεταγμένες κλάσεις ασφάλειας TS >S >C >U και οκτώ κατηγορίες {A, K, L, Q, W, X, Y, Z } αναφερόμενες, σύμφωνα με αυτόν, σε οκτώ διαφορετικές εργασίες του συστήματος. Το πλέγμα αυτό (βλ. σχήμα 3) έχει 21 ετικέτες ασφάλειας που χρησιμοποιούνται για να συστήσουν το πλέγμα. Με εξαίρεση τις χαμηλές και υψηλές κλάσεις ασφάλειας, οι συνδυασμοί των κατηγοριών εμφανίζονται μόνο σε ζευγάρια και τριάδες. Επίσης, η χρήση των κατηγοριών εμφανίζεται κυρίως πάνω από το top-secret επίπεδο και καμία κάτω από το secret επίπεδο.



Σχήμα 3

4. Μοντέλα ελέγχου πρόσβασης (Access Control Models)

Οι ενεργές οντότητες σε ένα σύστημα είναι συνήθως διαδικασίες που εκτελούν προγράμματα εκ μέρους των χρηστών. Συνεπώς, η ροή των πληροφοριών μεταξύ των αντικειμένων, και κατ' επέκταση μεταξύ των κλάσεων ασφάλειας, διεξάγονται από τις διαδικασίες. Οι πληροφορίες μπορούν ενδεχομένως να ρέουν από κάθε αντικείμενο που μία διαδικασία διαβάζει (*read*) σε κάθε αντικείμενο που αυτή γράφει (*write*). Σε περίπτωση που δεν γνωρίζουμε τι ακριβώς κάνει ένα πρόγραμμα, πρέπει να φανταστούμε τη χειρότερη κατάσταση και να ισχυριστούμε πως όποτε υπάρχει ένα, έστω και ένα μικρό, ενδεχόμενο ροής πληροφοριών, η ροή πραγματοποιείται. Με άλλα λόγια, πρέπει να είμαστε απόλυτα σίγουροι πως τα προγράμματα απλά δεν έχουν την ικανότητα να προκαλούν οποιαδήποτε ροή πληροφοριών η οποία είναι αντίθετη σε μία δεδομένη πολιτική ροής. Πριν δούμε πώς το μοντέλο Bell-LaPadula πραγματεύεται αυτόν τον στόχο, θα δούμε κάποιες βασικές αρχές που αφορούν τα μοντέλα ελέγχου προσπέλασης.

Για να κατανοήσουμε πλήρως τον έλεγχο προσπέλασης και την ασφάλεια των συστημάτων, πρέπει να καταλάβουμε τη διαφορά μεταξύ χρήστη (*user*) και υποκειμένου (*subject*) στα συστήματα. Αυτός ο διαχωρισμός είναι πολύ σημαντικός, αλλά πολλές φορές δεν χρησιμοποιείται σωστά, οδηγώντας σε υπερβολική σύγχυση όσον αφορά την ασφάλεια των πληροφοριακών συστημάτων.

Είναι γνωστό ότι ένας χρήστης είναι μία ανθρώπινη οντότητα. Επίσης υποθέτουμε πως κάθε τέτοια οντότητα που είναι γνωστή σε ένα σύστημα, αναγνωρίζεται ως ένας μοναδικός χρήστης. Με άλλα λόγια, ο μοναδικός άνθρωπος που ονομάζεται «Νίκος Παπαδόπουλος» δεν μπορεί να είναι πάνω από μία ταυτότητα χρήστη στο σύστημα. Αν ο «Νίκος Παπαδόπουλος» δεν είναι εξουσιοδοτημένος χρήστης του συστήματος, τότε αυτός δεν έχει ταυτότητα χρήστη. Επιπλέον, αν είναι εξουσιοδοτημένος χρήστης τότε αυτός έχει αποκλειστικά μία ταυτότητα χρήστη, π.χ. ΝΠαπαδ.

Η προηγούμενη υπόθεση μπορεί να εξασφαλιστεί μόνο με την είσοδο διαχειριστικών ελέγχων, που εδώ υποθέτουμε πως είναι ήδη τοποθετημένοι. Αυτή η υπόθεση πάντως δεν προαπαιτείται για μερικές πολιτικές ροής που θα ακολουθήσουν στη συνέχεια. Παράλληλα όμως, είναι σημαντική για ορισμένες άλλες πολιτικές όπως το πλέγμα (ακεραιότητας) του Lipher και το Chinese wall που θα συναντήσουμε. Αυτή η υπόθεση συχνά παραβιάζεται στα τρέχοντα συστήματα.

Αντιλαμβανόμαστε την έννοια ενός υποκειμένου να είναι μία διαδικασία σε ένα σύστημα. Πράγματι, ένα υποκείμενο είναι ένα πρόγραμμα σε εκτέλεση. Κάθε υποκείμενο σχετίζεται με έναν χρήστη, έχοντας το πρώτο να εκτελείται εκ μέρους του χρήστη. Γενικά, ένας χρήστης μπορεί να έχει περισσότερα από ένα υποκείμενα σε εκτέλεση την ίδια χρονική στιγμή. Κάθε φορά που ένας χρήστης *δηλώνεται* (*logs in*) σε ένα σύστημα, δηλώνεται σαν ένα συγκεκριμένο υποκείμενο. (Σημειώνουμε πως για τα μοντέλα ελέγχου προσπέλασης που εξετάζουμε, παίρνουμε σαν δεδομένο πως η ταυτοποίηση και εξουσιοδότηση των χρηστών γίνεται με ασφαλή και σωστό τρόπο).

Διαφορετικά υποκείμενα που σχετίζονται με τον ίδιο χρήστη, μπορούν να αποκτήσουν διαφορετικά σύνολα δικαιωμάτων προσπέλασης. Ας υποθέσουμε πως ο *top-secret* χρήστης «Γιάννης» δηλώνεται στο *secret* επίπεδο. Στην τεχνική ορολογία χρήστη-υποκειμένου περιγράφουμε το παραπάνω σενάριο ως εξής : Πρώτον, υπάρχει ένας μοναδικός χρήστης «Γιάννης», ο οποίος έχει δηλωθεί στο *top-secret* επίπεδο ασφάλειας. Δεύτερον, ο «Γιάννης» μπορεί να έχει σε εκτέλεση υποκείμενα σε οποιοδήποτε επίπεδο που επικρατείται από το top-secret. Στο εξής, θα θεωρούμε ότι κάθε υποκείμενο εκτελείται σε ένα σταθερό επίπεδο

ασφάλειας (Παρακάτω θα δούμε ότι το επίπεδο ασφάλειας των υποκειμένων μπορεί να αλλάξει σε κάποια μοντέλα.)

Τα δικαιώματα προσπέλασης των υποκειμένων στα αντικείμενα μπορούν εύκολα να αναπαρασταθούν από ένα *πίνακα προσπέλασης* (*access matrix*). Ο πίνακας αυτός έχει μία γραμμή για κάθε υποκείμενο, και μία στήλη για κάθε αντικείμενο που υπάρχει στο σύστημα. Επίσης, δεν αποκλείεται ένα υποκείμενο να είναι ένα αντικείμενο μέσα στο σύστημα.

Γενικά, όλα τα υποκείμενα είναι συγχρόνως και αντικείμενα, αλλά δεν ισχύει πάντα και το αντίστροφο. Το κελί (*cell*) για τη γραμμή c και τη στήλη o στον πίνακα προσπέλασης συμβολίζεται [c,o], και περιέχει ένα σύνολο από δικαιώματα προσπέλασης καθορίζοντας έτσι την εξουσιοδότηση ενός υποκειμένου c να λειτουργεί πάνω σε ένα αντικείμενο o. Για παράδειγμα, read∈ [s,o], εξουσιοδοτεί το s να διαβάζει το o. Οι λειτουργίες που καθορίζονται μέσω του πίνακα, είναι αυστηρά αυτές και μόνο που μπορούν να εκτελεστούν. Στον πίνακα, όλοι οι χρήστες θεωρούνται ως υποκείμενα στα δικά τους δικαιώματα. Ένα υποκείμενο αποκτά τα δικαιώματα ενός χρήστη, ακόμα και όταν ο χρήστης δεν έχει εμπλακεί σε καμία δραστηριότητα μέσα στο σύστημα. Ο πίνακας προσπέλασης είναι συχνά αραιός και αποθηκεύεται στο σύστημα χρησιμοποιώντας λίστες, σχέσεις και άλλες δομές δεδομένων για έξυπνη και αποδοτικότερη αποθήκευση αραιών πινάκων. Είναι λογικό ο πίνακας προσπέλασης να αποτελεί μία δυναμική οντότητα, με τα κελιά του να μπορούν να αλλάξουν από τα υποκείμενα. Για παράδειγμα, αν το υποκείμενο s είναι ο ιδιοκτήτης του αντικειμένου o (δηλαδή own∈ [s, o]), τότε το s μπορεί να διαφοροποιήσει όλα τα περιεχόμενα των κελιών που βρίσκονται στη στήλη o. Σε αυτή τη περίπτωση, ο ιδιοκτήτης ενός αντικειμένου έχει τον απόλυτο έλεγχο, όσον αφορά τη προσπέλαση στο αντικείμενο αυτό από τα άλλα υποκείμενα Τέτοιου είδους έλεγχοι προσπέλασης ονομάζονται *προαιρετικοί* (discretionary access controls). Ο πίνακας προσπέλασης μπορεί επίσης να αλλάξει με τη προσθήκη ή διαγραφή υποκειμένων και αντικειμένων. (Ο τυπικός έλεγχος προσπέλασης σε φακέλους και αρχεία σε ένα λειτουργικό σύστημα είναι ένα παράδειγμα προαιρετικού ελέγχου).

Από μόνοι τους, οι προαιρετικοί έλεγχοι προσπέλασης δεν είναι αρκετοί για να παρέχουν πολιτικές ροής πληροφοριών. Το βασικό τους πρόβλημα είναι πως δεν παρέχουν κανένα περιορισμό στην αντιγραφή πληροφοριών από ένα αντικείμενο σε ένα άλλο (Υπάρχουν και άλλα περισσότερο περίπλοκα προβλήματα με τον προαιρετικό έλεγχο, όπως το γνωστό και πολύ σημαντικό *πρόβλημα ασφάλειας*, ή *safety problem*, για την αντιγραφή και πολλαπλασιασμό των δικαιωμάτων προσπέλασης).

Ας υποθέσουμε ότι οι Κώστας, Γιώργος και Χάρης είναι χρήστες και ότι ο Κώστας έχει έναν εμπιστευτικό φάκελο *Private* που θέλει ο Γιώργος να τον διαβάσει, όχι όμως και ο Χάρης. Ο Κώστας μπορεί να εξουσιοδοτήσει τον Γιώργο να διαβάσει τον φάκελο προσθέτοντας read∈[Γιώργος, Private], θεωρώντας πως οι άλλοι χρήστες δεν μπορούν να παρέχουν το δικαίωμα read σε άλλους χρήστες. Ο Γιώργος μπορεί πολύ εύκολα να εξαπατήσει τον Κώστα, δημιουργώντας ένα αντίγραφο του *Private*, το *Copy-of-Private*, και να αντιγράψει τα περιεχόμενα του *Private* σε αυτό. Τώρα, ο Γιώργος ως ιδιοκτήτης του αντιγράφου μπορεί να δώσει το δικαίωμα στον Χάρη να διαβάσει τον φάκελο, με το read∈[Χάρης, Copy-of-Private]. Έτσι, ο Χάρης μπορεί να διαβάζει έμμεσα τον *Private*, όσο το αντίγραφο ενημερώνεται, και να το χρησιμοποιήσει με όποιον τρόπο αυτός θέλει. Ας θεωρήσουμε τώρα τον Γιώργο ως έμπιστο συνεργάτη του Κώστα, και επίσης ότι δεν παραβιάζει σκόπιμα την επιθυμία του Κώστα για τον φάκελο *Private*. Επίσης, ο Γιώργος χρησιμοποιεί ένα επεξεργαστή κειμένου που του τον έδωσε ο Χάρης. Ο επεξεργαστής παρέχει όλες τις απαραίτητες λειτουργίες που χρειάζεται ο Γιώργος. Παράλληλα όμως, ο Χάρης έχει προγραμματίσει τον επεξεργαστή να δημιουργεί το *Copy-of-Private* και να παρέχει στον Χάρη το δικαίωμα read∈[Χάρης, Copy-of-Private]. Αυτό το είδος *Δούρειων Ίππων* λογισμικού εκτελεί τις προβλεπόμενες λειτουργίες, αλλά παράλληλη λαμβάνουν χώρα κακόβουλες δραστηριότητες με σκοπό να παραβιάσουν την ασφάλεια. Ένας άλλος Δούρειος Ίππος που δημιούργησε ο Χάρης, μπορεί να παρέχει στον ίδιο το δικαίωμα να διαβάζει απευθείας τον φάκελο Private. Η λύση είναι να χρησιμοποιήσουμε υποχρεωτικούς ελέγχους προσπέλασης (mandatory access controls), οι οποίοι δεν μπορούν να παραβιαστούν, ακόμα και από Δούρειους Ίππους.

5. Το μοντέλο Bell-LaPadula

Οι Bell και LaPadula τυποποίησαν την ιδέα των υποχρεωτικών ελέγχων προσπέλασης (mandatory access controls) [D.E Bell and LaPadula “Secure Computer Systems, foundations and models”, 1975], ορίζοντας ένα μοντέλο το οποίο φέρει τα ονόματά τους. Πολυάριθμες διαφοροποιήσεις έχουν γίνει στο μοντέλο αυτό από τότε που εκδόθηκε για πρώτη φορά.

Στη συνέχεια θα πραγματοποιήσουμε μία «μινιμαλιστική» προσέγγιση και ορισμό του μοντέλου, που καλείται *BLP*, που είναι το μικρότερο μοντέλο το οποίο παρέχει τους απαραίτητους ελέγχους προσπέλασης που θέλουμε εδώ να περιγράψουμε. Η σημειογραφία που χρησιμοποιείται για αυτό το μοντέλο είναι διαφορετική από αυτή του πρωτότυπου μοντέλου, και είναι αυτή που χρησιμοποιείται σήμερα στη βιβλιογραφία.

Η κεντρική ιδέα του μοντέλου BLP είναι να επαυξήσουμε τους *προαιρετικούς* (*discretionary*) *ελέγχους* προσπέλασης με υποχρεωτικούς ελέγχους προσπέλασης με σκοπό να μας δοθεί η δυνατότητα χρήσης πολιτικών ροής πληροφορίας. Το BLP προσεγγίζει σε δύο βήματα τον έλεγχο προσπέλασης: Στο πρώτο, υπάρχει ένας προαιρετικός πίνακας προσπέλασης D,τα περιεχόμενα του οποίου μπορούν να αλλάξουν από τα υποκείμενα. Παρόλα αυτά, η εξουσιοδότηση μέσα στον D δεν είναι αρκετή συνθήκη για την εκτέλεση μίας λειτουργίας. Στο δεύτερο βήμα, η λειτουργία πρέπει να εξουσιοδοτηθεί από την υποχρεωτική πολιτική ελέγχου προσπέλασης, την οποία οι χρήστες δεν μπορούν να ελέγξουν.

Η πολιτική υποχρεωτικού ελέγχου προσπέλασης εκφράζεται σε σχέση με τις ετικέτες ασφάλειας που είναι συνδεδεμένες με τα υποκείμενα και τα αντικείμενα. Μία ετικέτα αντικειμένου ονομάζεται *ταξινόμηση ασφάλειας* (*security classification*), ενώ μία ετικέτα υποκειμένου ονομάζεται *δήλωση ασφάλειας*. Ένας χρήστης με ετικέτα secret μπορεί να εκτελεί το ίδιο πρόγραμμα με ένα υποκείμενο με ετικέτα secret ή με ένα υποκείμενο με ετικέτα unclassified. Παρά το γεγονός ότι και τα δύο υποκείμενα εκτελούν το ίδιο πρόγραμμα, αποκτούν διαφορετικά προνόμια λόγω των ετικετών τους. Συχνά υποθέτουμε πως, αφού δηλωθούν, οι ετικέτες των υποκειμένων και αντικειμένων δεν μπορούν να αλλάξουν. Αυτή η υπόθεση, γνωστή ως υπόθεση ισορροπίας, μπορεί να οριστεί στη συνέχεια πιο χαλαρά με έναν ασφαλή τρόπο.

Με λ συμβολίζουμε την ετικέτα ασφάλειας ενός υποκειμένου ή αντικειμένου. Οι κανόνες του υποχρεωτικού ελέγχου προσπέλασης του BLP είναι οι εξής :

- Απλή-Ιδιότητα-Ασφαλείας : Το υποκείμενο s μπορεί διαβάσει το αντικείμενο o μόνο αν $\lambda(\mathrm{s}) \geq \lambda(o)$.
- Ιδιότητα-* : Το υποκείμενο s μπορεί να γράψει στο αντικείμενο o μόνο αν $\lambda(\mathrm{s}) \leq \lambda(o)$.

Η προσπέλαση ανάγνωσης (read access) σημαίνει ροή πληροφοριών από ένα αντικείμενο σε ένα υποκείμενο, γι' αυτό και η απαίτηση $\lambda(\mathrm{s}) \geq \lambda(o)$. Η προσπέλαση εγγραφής, (write access) αντιθέτως, σημαίνει ροή πληροφοριών από ένα υποκείμενο σε ένα αντικείμενο, γι' αυτό και η απαίτηση $\lambda(\mathrm{s}) \leq \lambda(o)$ ή όμοια $\lambda(\mathrm{s}) \rightarrow \lambda(o)$. Σε μερικά μοντέλα, η προσπέλαση εγγραφής σημαίνει ανάγνωση και εγγραφή συγχρόνως, μόνο με την εξουσιοδότηση της εγγραφής , π.χ. write $\in$ [s, o] .

Σε πραγματικά συστήματα, υπάρχουν φυσικά πρόσθετες λειτουργίες, όπως η δημιουργία και καταστροφή αντικειμένων. Εδώ χρησιμοποιούμε την ανάγνωση και εγγραφή για να περιγράψουμε τα βασικά στοιχεία των μοντέλων. Για παράδειγμα, η δημιουργία και η καταστροφή αντικειμένων περιορίζονται επίσης από τη Ιδιότητα-* επειδή αλλάζουν την κατάστασή τους. Οι υποχρεωτικοί έλεγχοι διατυπώνονται ως «μόνο αν» συνθήκες, οι οποίες είναι μεν απαραίτητες αλλά όχι και αρκετές για τη προσπέλαση. Αν ο προαιρετικός πίνακας προσπέλασης D δεν εξουσιοδοτεί μία λειτουργία, τότε δεν είναι απαραίτητο να εξετάσουμε τους υποχρεωτικούς ελέγχους, μιας και η λειτουργία σίγουρα θα ακυρωθεί. Όμοια, οι υποχρεωτικοί έλεγχοι μπορούν να πραγματοποιηθούν πρώτα , ακολουθημένοι από τους προαιρετικούς ελέγχους προσπέλασης.

Η Απλή-Ιδιότητα-Ασφαλείας αναφέρεται ισότιμα στους χρήστες και στα προγράμματα. Αντιθέτως, η Ιδιότητα-* δεν αναφέρεται στους χρήστες αλλά στα προγράμματα. Οι χρήστες δεν είναι έμπιστοι για τη ροή των πληροφοριών. Ένας secret χρήστης μπορεί να γράψει ένα unclassified έγγραφο επειδή όμως υποθέτουμε πως αυτός θα γράψει μόνο unclassified πληροφορίες σε αυτό. Τα προγράμματα από την άλλη, δεν είναι έμπιστα γιατί μπορούν να περιέχουν Δουρείους Ίππους. Η Ιδιότητα-* εμποδίζει ένα πρόγραμμα σε εκτέλεση στο secret επίπεδο να γράψει σε unclassified αντικείμενα, ακόμα και αν αυτό επιτρέπεται από τους προαιρετικούς ελέγχους. Ένας χρήστης με ετικέτα secret που επιθυμεί να γράψει σε ένα unclassified αντικείμενο, είναι πρώτα υποχρεωμένος να δηλωθεί ως unclassified υποκείμενο.

Ένα περίεργο χαρακτηριστικό της Ιδιότητας-* είναι πως ένα unclassified υποκείμενο μπορεί να γράψει σε ένα secret αντικείμενο. Αυτό σημαίνει πως οι secret πληροφορίες μπορούν να αλλάξουν ή και να διαγραφούν ,ίσως τυχαία, από unclassified υποκείμενα. Για να λυθεί αυτό το πρόβλημα ακεραιότητας , μία παραλλαγή του Ιδιότητα-* μπορεί να χρησιμοποιηθεί και απαιτεί $\lambda(\mathrm{s}) = \lambda(o)$, έτσι ώστε τα υποκείμενα να μπορούν να γράψουν στο δικό τους επίπεδο αλλά όχι και στα ανώτερά τους. Ας εξετάσουμε εδώ τις συνέπειες των παραπάνω στο παράδειγμα των Κώστα, Γιώργο και Χάρη. Υποθέτουμε πως ο Κώστας και ο Γιώργος είναι secret χρήστες και ο Χάρης unclassified. Έτσι, οι δύο πρώτοι μπορούν να έχουν secret αντικείμενα, ενώ ο τελευταίος μόνο unclassified. Ο Κώστας τώρα δημιουργεί τον secret φάκελο Private, μέσω ενός secret αντικειμένου. Η Απλή-Ιδιότητα-Ασφαλείας θα εμποδίσει τα υποκείμενα του Χάρης να διαβάσουν άμεσα τον φάκελο Private. Επίσης η Απλή-Ιδιότητα-Ασφαλείας και η Ιδιότητα-* εξασφαλίζουν πως τα αντικείμενα του Χάρης δεν θα μπορούν να διαβάσουν το Copy-of-Private, γιατί αυτό θα έχει ετικέτα secret ή ανώτερη. Συγκεκριμένα, αν ο Δούρειος Ίππος προσβάλει secret υποκείμενα και δημιουργήσει το Copy-of-Private, αυτό θα έχει ετικέτα secret ή ανώτερη, έτσι ώστε τα υποκείμενα του Χάρης να μην μπορούν

να το διαβάσουν. Από την άλλη, τα unclassified υποκείμενα με τον Δούρειο Ίππο δεν θα μπορούσαν να διαβάσουν και να αντιγράψουν για τον Χάρης τον Private. Το BLP μοντέλο αποτρέπει μόνο τη ροή πληροφοριών μεταξύ κλάσεων ασφάλειας, Έτσι, αν ο Χάρης ήταν secret χρήστης όπως επίσης και οι Γιώργος και Κώστας, αυτοί οι έλεγχοι δεν θα μπορούσαν να λύσουν το παραπάνω πρόβλημα.

Δυστυχώς, οι υποχρεωτικοί έλεγχοι δεν λύνουν το πρόβλημα των Δούρειων Ίππων τελείως. Ένα secret υποκείμενο εμποδίζεται στο να γράψει άμεσα σε ένα unclassified αντικείμενο. Υπάρχουν άλλοι, όμως, έμμεσοι τρόποι στην επικοινωνία πληροφοριών με unclassified αντικείμενα. Για παράδειγμα, το secret υποκείμενο μπορεί να αποκτήσει μεγάλα ποσά μνήμης στο σύστημα. Αυτό μπορεί να ανιχνευθεί από ένα unclassified υποκείμενο το οποίο μπορεί να παρατηρεί το ποσό της διαθέσιμης μνήμης. Αν και το unclassified υποκείμενο εμποδίζεται στο να παρατηρεί άμεσα τη μνήμη, μπορεί να το κάνει έμμεσα πραγματοποιώντας μία αίτηση για πολύ μεγάλο ποσό μνήμης. Αποδέχοντας ή απορρίπτοντας την αίτηση αυτή, θα αποκαλυφθεί μερικές πληροφορίες για την ελεύθερη μνήμη στο unclassified υποκείμενο.

Τέτοιου είδους επικοινωνία ονομάζεται *κρυφό* (ή *συγκεκαλυμμένο-covert channel*) κανάλι. Τα κρυφά κανάλια αποτελούν ένα σοβαρότατο πρόβλημα στη χρήση πολιτικών ροής πληροφοριών. Είναι πολύ δύσκολο να ανιχνευθούν, και αν ανιχνευθούν είναι δύσκολο να κλείσουν χωρίς σημαντικές παραβιάσεις πληροφοριών και απώλειες απόδοσης. Το πρόβλημα των κρυφών καναλιών είναι έξω από τη σκοπιά των μοντέλων προσπέλασης που βασίζονται στα πλέγματα, όπως το Bell-LaPadula. Αυτά τα μοντέλα ψάχνουν να εμποδίσουν τη μη ασφαλή ροή πληροφοριών μέσω αντικειμένων που είναι διαθέσιμα για δια-διαδικαστικό διαμοιρασμό δεδομένων και επικοινωνίας. Το πρόβλημα της αποφυγής , ανάληψης και ανοχής των κρυφών καναλιών αποτελεί ένα μηχανικό θέμα, απαιτώντας ανάλυση του συστήματος , της αρχιτεκτονικής του και του κώδικα. Μία άλλη ομάδα μοντέλων, τα Μοντέλα Ροής Πληροφοριών, προσπαθούν να ασχοληθούν με όλες τις ροές των πληροφοριών σε ένα σύστημα, συμπεριλαμβάνοντας και το πρόβλημα των κρυφών καναλιών.

Παραπάνω αναφέραμε την υπόθεση *ισορροπίας* (*tranquility*) που απαιτεί τις ετικέτες ασφάλειας των υποκειμένων και των αντικειμένων να μην αλλάζουν. Αυτή η υπόθεση μπορεί να γίνει πιο χαλαρή είτε με ασφαλείς είτε με μη ασφαλείς τρόπους (επιτρέποντας ροές αντίθετες με το πλέγμα που χρησιμοποιείται). Θεωρούμε ένα υποκείμενο s να αλλάζει την ετικέτα ενός αντικειμένου από $\lambda(o)$ σε $\lambda(o)'$, με την προϋπόθεση όμως $\lambda(\mathrm{s}) = \lambda(o)$, και

$\lambda(o)' > \lambda(o)$. Για παράδειγμα, ένα unclassified υποκείμενο αναβαθμίζει την ετικέτα ενός φακέλου από unclassified σε secret. Μία τέτοια αλλαγή είναι ασφαλής γιατί δεν προκαλεί καμία ροή πληροφοριών από secret σε unclassified. Ας υποθέσουμε τώρα πως έχουμε ένα secret υποκείμενο που πραγματοποιεί την ίδια αλλαγή, έχοντας όμως $\lambda(s) > \lambda(o)$ (αντί για $\lambda(\mathrm{s}) = \lambda(o)$). Αυτή η περίπτωση είναι μη ασφαλής γιατί αναβαθμίζεται ένας φάκελος από unclassified σε secret από ένα secret υποκείμενο , εξαφανίζοντας τον φάκελο από την όψη των unclassified υποκειμένων, ανοίγοντας έτσι όμως δρόμους επικοινωνίας από το secret στο unclassified (γεγονός που θα μπορούσε να εκμεταλλευτεί ένας Δούρειος Ίππος). Ένας secret χρήστης θα μπορούσε να αναβαθμίσει με ασφάλεια αυτόν τον φάκελο δηλώνοντας τον εαυτό του πρώτα ως unclassified χρήστη-υποκείμενο.

6. Το μοντέλο Biba και ο δυισμός (duality)

Προβληματισμοί σχετικά με την *εμπιστευτικότητα* (*confidentiality*) ώθησαν τους υποχρεωτικούς ελέγχους στο μοντέλο Bell-LaPadula. Ο Biba προτείνε [K.J. Biba "Integrity Considerations for Secure Computer Systems", 1977] πως παρόμοιοι έλεγχοι θα μπορούσαν να οριστούν και να χρησιμοποιηθούν και για την ακεραιότητα. Η βασική ιδέα του μοντέλου Biba είναι πως οι πληροφορίες χαμηλής ακεραιότητας, ή low-integrity, δεν πρέπει να μπορούν να ρέουν προς αντικείμενα υψηλής ακεραιότητας, ή high-integrity, όπου το αντίστροφο όμως είναι αποδεκτό. Ο Biba πρότεινε πολλές μεθόδους για τη χρήση τέτοιων ελέγχων για αντικείμενα ακεραιότητας. Η πιο γνωστή από αυτές ονομάζεται αυστηρή ακεραιότητα, ή strict integrity.

Στη συνηθισμένη μορφή του μοντέλου Biba, η υψηλή ακεραιότητα τοποθετείται στη κορυφή του πλέγματος των ετικετών ασφάλειας, και η χαμηλή στη βάση του. Σύμφωνα με αυτή τη μορφή επιτρέπεται η ροή πληροφοριών από πάνω προς τα κάτω, η αντίθετη δηλαδή από αυτή του μοντέλου Bell-LaPadula και των αξιωμάτων του Denning. Αυτό οδήγησε τον Biba να ορίσει τους παρακάτω υποχρεωτικούς ελέγχους, σε αναλογία με τους υποχρεωτικούς ελέγχους του μοντέλου BLP (με ω δηλώνεται η ετικέτα ακεραιότητας ενός υποκειμένου ή αντικειμένου) :

- Απλή-Ιδιότητα-Ακεραιότητας : Το υποκείμενο s μπορεί να διαβάσει το αντικείμενο ο μόνο αν $\omega(s) \le \omega(o)$.
- Ιδιότητα-ακεραιότητας-* : Το υποκείμενο s μπορεί να γράψει στο αντικείμενο ο μόνο αν $\omega(s) \ge \omega(o)$.

Αυτές οι ιδιότητες είναι δυικές των αντίστοιχων ιδιότητας στο μοντέλο BLP. Έτσι μπορούμε να χρησιμοποιήσουμε τους υποχρεωτικούς ελέγχους του BLP για να προσθέσουμε τη ροή πληροφοριών που απαιτείται στο Biba. Η κατάσταση αυτή είναι συμμετρική. Μπορούμε δηλαδή να αντιστρέψουμε τα πλέγματα BLP-Denning βάζοντας τη χαμηλή εμπιστευτικότητα στη κορυφή και την υψηλή στη βάση, και να ορίσουμε υποχρεωτικούς ελέγχους για τη ροή. Δεν υπάρχει κάποια βασική διαφορά μεταξύ των μοντέλων Biba και BLP. Και τα δύο επιτρέπουν τη ροή πληροφοριών σε ένα πλέγμα κλάσεων ασφάλειας προς τη μία μόνο κατεύθυνση. Το BLP επιτρέπει την προς τα πάνω ροή, ενώ το Biba την προς τα κάτω.

Συχνά, σε περιπτώσεις όπου και η εμπιστευτικότητα αλλά και η ακεραιότητα μας ενδιαφέρουν, ο συνδυασμός των δύο παραπάνω μοντέλων θα μπορούσε να αποδειχθεί αρκετά χρήσιμος. Αν όμως χρησιμοποιηθεί μία ετικέτα και για τα δύο στοιχεία, τότε θα προκύψουν συγκρούσεις. Ένα αρνητικό χαρακτηριστικό του συνδυασμού αυτού, είναι πως ένα υποκείμενο μπορεί να διαβάσει ή να γράψει ακριβώς στα αντικείμενα με την ίδια ετικέτα. Πιο εύκολη είναι η χρήση δύο διαφορετικών ετικετών : της εμπιστευτικότητας λ και της ακεραιότητας ω, με τους υποχρεωτικούς ελέγχους του BLP να εφαρμόζονται στη πρώτη και του Biba στη δεύτερη.

Έτσι, οι συνδυασμένοι υποχρεωτικοί έλεγχοι είναι οι εξής :

- Το υποκείμενο s μπορεί να διαβάσει το αντικείμενο ο μόνο αν $\lambda(s) \geq \lambda(o)$ και $\omega(s) \leq \omega(o)$.
- Το υποκείμενο s μπορεί να γράψει στο αντικείμενο ο μόνο αν $\lambda(s) \leq \lambda(o)$ και $\omega(s) \geq \omega(o)$.

Κεφάλαιο 3

Κριτήρια Ορθότητας για κατανεμημένες συναλλαγές πολλών επιπέδων ασφαλείας

(Correctness Criteria for multilevel secure transactions)

1. Εισαγωγή

Τα πλεονεκτήματα των ανοικτών κατανεμημένων συστημάτων και των κατανεμημένων Βάσεων Δεδομένων (ΒΔ) έχουν γίνει ευρέως αποδεκτά, αλλά συχνά δεν μπορούν να τα εκμεταλλευτούν οι χρήστες οι οποίοι πρέπει να προστατεύσουν τα δεδομένα τους χρησιμοποιώντας ελέγχους πρόσβασης σε αυτά βασισμένους σε ετικέτες ασφαλείας πχ. Απόρρητο, Εμπιστευτικό κτλ. Σε συστήματα τα οποία χειρίζονται «ομαδοποιημένα» δεδομένα, το να εμποδιστεί η ροή ευαίσθητων πληροφοριών από ευαίσθητα αντικείμενα σε αντικείμενα χαμηλότερης ευαισθησίας, είναι μια σημαντική πρόκληση. Για να εμποδιστούν τέτοιου είδους ροές πληροφοριών, τα συστήματα πολλών επιπέδων ασφάλειας δεν πρέπει απλά να διασφαλίζουν μόνο το ότι οι χρήστες διαβάζουν αποκλειστικά τα δεδομένα για τα οποία είναι εξουσιοδοτημένοι, αλλά και αφού τα έχουν διαβάσει σε κάποιο επίπεδο ασφάλειας, να μη μπορούν να γράψουν σε ένα χαμηλότερο επίπεδο (τουλάχιστο όχι στη διάρκεια της ίδιας διαδικασίας).

Οι χρήστες δεδομένων με ετικέτες συχνά πρέπει να διαβάζουν και να γράφουν δεδομένα με ένα εύρος ετικετών ασφαλείας. Για παράδειγμα, από το σύνολο των δεδομένων που απαιτεί μία διαδικασία, μπορεί να υπάρχουν ευαίσθητα ή όχι δεδομένα, αλλά όλα αυτά να σχετίζονται και να πρέπει να χρησιμοποιηθούν από τους χρήστες. Το ερευνητικό πρόγραμμα MUSET (*Multilevel Secure Transactions-Ασφαλείς Συναλλαγές Πολλών Επιπέδων Ασφαλείας*) εστιάζει στην ανάγκη των χρηστών να γράφουν σε δεδομένα με διαφορετικά επίπεδα ασφαλείας κατά την εκτέλεση μιας απλής κατανεμημένης συναλλαγής. Στη συνέχεια θα περιγράψουμε ένα μοντέλο και τα κριτήρια ορθότητας για τις κατανεμημένες συναλλαγές.

2. Το πρόγραμμα MUSET

Το *MUSET* θεμελιώνει τη δυνατότητα εκτέλεσης κατανεμημένων συναλλαγών πολλών επιπέδων, όπου δίνει την δυνατότητα στους χρήστες να λειτουργούν σε όλο το δηλωμένο εύρος επιπέδων ασφαλείας χρησιμοποιώντας κατανεμημένες πηγές δεδομένων.

Η λειτουργία του MUSET μπορεί να περιγραφεί απλά με το παρακάτω παράδειγμα.. Υποθέτουμε ότι το site_1 (του κατανεμημένου συστήματος) αποθηκεύει χαμηλής

ευαισθησίας (*Low*) δεδομένα (περιέχοντας το στοιχείο x), και το site_2 αποθηκεύει υψηλής ευαισθησίας (*High*-περιέχοντας το y). Τα συστήματα διαχείρισης των κατανεμημένων ΒΔ (DBMS) εξασφαλίζουν πως οι χρήστες που είναι δηλωμένοι ως Low, θα μπορούν να διαβάσουν και να γράψουν Low δεδομένα, ενώ οι χρήστες που είναι δηλωμένοι ως High μπορούν να έχουν πρόσβαση και στα δύο επίπεδα.. Ας υποθέσουμε πως ένας High χρήστης εκτελεί την παρακάτω συναλλαγή πολλών επιπέδων ασφαλείας:

y : = 1
x : = x + 2
z := x + y

Το MUSET αναλύει την συναλλαγή και παράγει τις παρακάτω διαδικασίες με ετικέτες ασφαλείας :

High: w(y)
Low: R(x)
Low: W(x)
High: R(x)
High: R(y)
High: W(z)

Ας σημειώσουμε πως παρά το γεγονός του ότι το x είναι Low, η διαδικασία read x έχει ετικέτα High. Αυτό οφείλεται στο ότι μια High διαδικασία πρέπει να διαβάσει (High) x ώστε να γράψει High z.

Το MUSET χωρίζει τις διαδικασίες στα παρακάτω σύνολα απλού επιπέδου ασφαλείας (τομείς, sections)

Low	High
R(x)	w(y)
W(x)	R(x)
	R(y)
	W(z)

Οι διαδικασίες πλέον εκτελούνται από τα συστήματα διαχείρισης των κατανεμημένων ΒΔ των sites, ενώ το MUSET συντονίζει την εκτέλεσή τους για τον έλεγχο του ταυτοχρονισμού.

3. Κριτήρια ορθότητας

Στόχος του *MUSET* είναι να εκτελεί ξεχωριστούς τομείς έτσι ώστε να πετύχει το ίδιο τελικό αποτέλεσμα, σαν μία ατομική, συνεπή, απομονωμένη και ασφαλή εκτέλεση της αρχικής συναλλαγής σε ένα απλό site. Όμοια με τις ACID ιδιότητες των συναλλαγών και των χρονοδιαγραμμάτων (*schedules*) των ΒΔ, ορίζουμε τέσσερις ACIS ιδιότητες ορθότητας: (Ατομικότητα, συνέπεια, απομόνωση και ασφάλεια, αντίστοιχα). *A-ορθότητα* σημαίνει πως κάθε συναλλαγή σε ένα χρονοδιάγραμμα είναι πλήρως ατομική: είτε κάθε κομμάτι της συναλλαγής ολοκληρώνεται (*commits*) είτε κανένα. *C-ορθότητα* σημαίνει ότι κάθε συναλλαγή σε ένα χρονοδιάγραμμα είναι ισοδύναμη όσον αφορά τις συγκρούσεις (*conflict equivalent*) με την αρχική σειρά των διαδικασιών μιας συναλλαγής. *I-ορθότητα* (σειριοποίηση) σημαίνει πως ένα χρονοδιάγραμμα είναι ισοδύναμο όσον αφορά τις συγκρούσεις με την σειριοποιημένη μορφή της αρχικής συναλλαγής. *S-ορθότητα* είναι όμοια με τη συνθήκη της μη παρέμβασης, δηλαδή να μην είναι δυνατό High διαδικασίες να μπορούν να επηρεάσουν διαδικασίες σε χαμηλότερα επίπεδα ασφάλειας.

Ενώ η Ατομικότητα, η Συνέπεια και η Απομόνωση είναι αμοιβαία κατορθώσημοι στόχοι σε ένα απλό site, σε κατανεμημένες συναλλαγές πολλών επιπέδων ασφάλειας συγκρούονται με κάποιούς από τους παραπάνω στόχους, αναγκάζοντας μας να καταφεύγουμε σε αντισταθμίσεις (*trade-offs*) μεταξύ των ACIS ιδιοτήτων στις συναλλαγές αυτές.

Λόγω αυτών των αντισταθμίσεων, ένα σημαντικό πρόβλημα που τίθεται είναι ο σχεδιασμός πρωτόκολλων εκτέλεσης με τα οποία να πετυχαίνουμε το απαιτούμενο επίπεδο ορθότητας μεταξύ των ορίων των ACIS ιδιοτήτων. Τα πρωτόκολλα αυτά πρέπει να έχουν μεγάλη ποικιλία προσεγγίσεων στις αντισταθμίσεις, έτσι ώστε να ικανοποιούν τις διαφορετικές προτεραιότητες των διαφόρων χρηστών.

Παρακάτω θα ορίσουμε κάποιες βασικές έννοιες του μοντέλου MUSET, που είναι οι συναλλαγές πολλών επιπέδων ασφάλειας, τα χρονοδιαγράμματα και τα πρωτόκολλα εκτέλεσης για αυτού του είδους τις συναλλαγές. Ορίζουμε, επίσης, τέσσερεις επιθυμητές ιδιότητες ορθότητας για αυτά τα πρωτόκολλα και για τα χρονοδιαγράμματα που αυτά παράγουν. Αυτό το μοντέλο χρησιμοποιείται επίσης για τη μελέτη των αντισταθμίσεων μεταξύ των τεσσάρων ιδιοτήτων που θα περιγράψουμε.

Ορισμός [Συναλλαγές πολλών επιπέδων ασφάλειας - Multilevel Transactions]

Μία συναλλαγή πολλών επιπέδων ασφάλειας T_i αποτελείται από ένα σύνολο read και write λειτουργιών, μερικώς διατεταγμένες κατά $<_i$, οι οποίες εκτελούνται σε πολλαπλά επίπεδα, και που η κάθε μία υπακούει στο Bell-LaPadula , και στην Ιδιότητα-* και στη Απλή-Ιδιότητα-Ασφαλείας. Σε μια συναλλαγή T_i, η λειτουργία O_{ij} δηλώνει το επίπεδο ταξινόμησης j. Έτσι η r_{ij} περιγράφει τη λειτουργία read στο επίπεδο ασφάλειας j, μέρος της συναλλαγής i. Όταν η συναλλαγή εύκολα εννοείται, γράφουμε απλά O_j. Γενικά, δύο λειτουργίες βρίσκονται σε σύγκρουση αν και οι δύο λειτουργούν πάνω στα ίδια δεδομένα και τουλάχιστο μία από αυτές είναι write λειτουργία..

Μία *συναλλαγή πολλών επιπέδων ασφάλειας* είναι ένα σύνολο λειτουργιών $Oij \in \{r_{ij}[x_k] \mid j \geq k\} \cup \{w_{ij}[x_j]\}$ το οποίο είναι μερικώς διατεταγμένο κατά $<_i$.(Με x_k εννοούμε το στοιχείο δεδομένων x στο επίπεδο ασφάλειας κ).

Όταν μία συναλλαγή πολλών επιπέδων ασφάλειας εκτελείται, τα γεγονότα σε αυτή είτε απορρίπτονται (*abort*) είτε ολοκληρώνονται (*commit*). Επίσης, η φάση της πρόωρης ολοκλήρωσης (precommit, θα αναλυθεί παρακάτω) είναι απαραίτητη για τον συγχρονισμό γεγονότων σε διαφορετικά επίπεδα.. Σε μια συναλλαγή T_i η λειτουργία P_{ik} σημαίνει ότι όλες οι λειτουργίες της συναλλαγής στο επίπεδο k έχουν εκτελεστεί και δεν υπάρχουν κάποια περαιτέρω εμπόδια για να ολοκληρωθούν. Επίσης, σε ένα χρονοδιάγραμμα περιέχονται οι λειτουργίες **a** (*abort*), **c** (*commit*) και **p** (*precommit*). Επειδή οι συναλλαγές απλού επιπέδου είναι ατομικές, τα αποτελέσματα των write λειτουργιών δεν είναι ορατά σε άλλες συναλλαγές μέχρι αυτές να ολοκληρωθούν. Τέλος, ακόμα και κατά τη διάρκεια εκτέλεσης μίας συναλλαγής πολλών επιπέδων ασφάλειας , οι λειτουργίες write σε χαμηλότερα επίπεδα δεν είναι ορατές από τα υψηλότερα επίπεδα, μέχρις ότου τα χαμηλότερα επίπεδα να ολοκληρώσουν τις λειτουργίες τους.

Ορισμός [Χρονοδιάγραμμα πολλών επιπέδων ασφάλειας - Multilevel Transaction Schedule]

Ένα χρονοδιάγραμμα S_{Ti} (S_i όταν εννοείται το όνομα της συναλλαγής) για μια συναλλαγή T_i, είναι μία ολική διάταξη λειτουργιών κατά $<_{S_{Ti}}$ έτσι ώστε :

1. $S_i \subseteq T_i \cup \{p_{ij}, a_{ij}, c_{ij}\}$, όπου p_{ij} είναι μία λειτουργία πρόωρης ολοκλήρωσης, a_{ij} είναι μία λειτουργία απόρριψης και c_{ij} είναι μία λειτουργία ολοκλήρωσης μίας

διαδικασίας. Εδώ δηλαδή ορίζεται το σύνολο των λειτουργιών που παίρνουν μέρος σε ένα χρονοδιάγραμμα.

2. $a_{ij} \in S_i$ μόνο και μόνο αν $c_{ij} \notin S_i$ και αντίστροφα.. Δηλαδή, το παραπάνω σύνολο περιέχει μία λειτουργία ολοκλήρωσης c_{ij} ή μία απόρριψης a_{ij}, αλλά ποτέ και τις δύο στο ίδιο επίπεδο.
3. Για τις λειτουργίες a_{ij} ή c_{ij}, οπουδήποτε στο S_i, για κάθε άλλη λειτουργία $o_{ij} \in S_i$, πρέπει $o_{ij} <_{Si} a_{ij}, c_{ij}$. Δηλαδή η λειτουργία ολοκλήρωσης c_{ij} ή απόρριψης a_{ij} έρχεται μετά από όλες τις άλλες διαδικασίες.
4. Για τις λειτουργίες a_{ij} ή c_{ij}, οπουδήποτε στο S_i ,και για την λειτουργία p_{ij}, δεν υπάρχει καμία λειτουργία o_{ij} έτσι ώστε $p_{ij} <_{Si} o_{ij} <_{Si} a_{ij}, c_{ij}$. Δηλαδή αν σε ένα χρονοδιάγραμμα περιέχεται η λειτουργία p_{ij}, αυτή εκτελείται αμέσως πριν από την λειτουργία ολοκλήρωσης c_{ij} ή απόρριψης a_{ij}, χωρίς άλλες λειτουργίες στο ενδιάμεσο.
5. Οι συγκρουόμενες λειτουργίες $o_{ij} \in S_i$ και $o_{ij}' \in S_i$, πρέπει να πληρούν την $<_i$ διάταξη. Δηλαδή η εκτέλεση ενός χρονοδιαγράμματος πρέπει να διατηρεί την διάταξη των συγκρουόμενων λειτουργιών σε κάθε επίπεδο.
6. Αν $w_{ik}[x_k] <_i r_{ij}[x_k]$, τότε $c_{ik} <_{Si} r_{ij}[x_k]$. Δηλαδή όταν εκτελείται μία λειτουργία write σε ένα επίπεδο ασφαλείας k η οποία ακολουθείται από μία διαδικασία read υψηλότερου επιπέδου j (δηλ $j > k$) στα ίδια στοιχεία δεδομένων, η εκτέλεση της λειτουργίας read πρέπει να ακολουθεί την ολοκλήρωση της write.

Τα κριτήρια για τα χρονοδιαγράμματα απλών συναλλαγών χρησιμοποιούνται ως θεμέλιοι λίθοι για την κατασκευή χρονοδιαγραμμάτων πάνω σε σύνολα συναλλαγών πολλών επιπέδων ασφάλειας.

Είναι πιθανό να υπάρχει ένα μεγάλο σύνολο από δυνατά χρονοδιαγράμματα για την ίδια συναλλαγή ή σύνολο συναλλαγών. Οι έννοιες ορθότητας που θα περιγράψουμε στη συνέχεια βασίζονται στην ιδέα των *ισοδύναμων χρονοδιαγραμμάτων* (*equivalent schedules*).

Ορισμός [Ισοδύναμα Χρονοδιαγράμματα πολλών επιπέδων ασφάλειας - Equivalent Multilevel Transaction Schedule]

Έστω ότι ένα στιγμιότυπο μιας ΒΔ αποτελείται από ένα σύνολο γεγονότων σε μια συγκεκριμένη χρονική στιγμή. Τα χρονοδιαγράμματα S και S' είναι ισοδύναμα αν για κάθε

στιγμιότυπο D της βάσης δεδομένων, εκτελώντας το S στην D παράγεται η ίδια ακριβώς κατάσταση D', αν εκτελούσαμε το S' στην D.

Ορισμός [Πρωτόκολλο εκτέλεσης πολλών επιπέδων ασφάλειας - Multilevel Execution Protocol]

Ένα πρωτόκολλο εκτέλεσης πολλών επιπέδων ασφάλειας (ή απλά πρωτόκολλο) P μετατρέπει ένα σύνολο από συναλλαγές T σε ένα χρονοδιάγραμμα S_T. Αυτή η μετατροπή συμβολίζεται με $P(T) = S_T$.

4. Ορισμός κριτηρίων ορθότητας

4.1. Ατομικότητα [atomicity]

Σε ένα πλήρως *ατομικό* χρονοδιάγραμμα (*A-correct*) οι λειτουργίες σε μία συναλλαγή πρέπει είτε να ολοκληρώνονται όλες, είτε καμία. Σε ένα χρονοδιάγραμμα πολλών επιπέδων ασφαλείας, υπάρχει μία λειτουργία ολοκλήρωσης ή απόρριψης που αναφέρεται σε κάθε επίπεδο όπου γίνονται εγγραφές (*writes*). Ακολουθεί ο προσαρμοσμένος ορισμός της ατομικότητας σε χρονοδιαγράμματα πολλών επιπέδων ασφάλειας.

Ορισμός [A-ορθότητα]

Έστω ότι το S_T είναι ένα χρονοδιάγραμμα για ένα σύνολο συναλλαγών T πολλών επιπέδων. Το S_T είναι A-ορθό αν για όλα τα χρονοδιαγράμματα S_i στο S_T, υπάρχει j τέτοιο ώστε αν $c_{ij} \in S_i$, τότε $c_{ik} \in S_i$ για όλα τα επίπεδα κ που αντιστοιχούν στις λειτουργίες εγγραφής $w_{ik} \in T_i$. Πράγμα που σημαίνει πως σε κάθε επίπεδο όπου εκτελούνται λειτουργίες εγγραφής, το επίπεδο αυτό είτε πρέπει να ολοκληρώνεται είτε να απορρίπτεται στο σύνολό του.

Κατ' επέκταση, τα κριτήρια ορθότητας που εφαρμόζονται στα χρονοδιαγράμματα, μπορούν επίσης να εφαρμοστούν και στα πρωτόκολλα που τα παράγουν.

Ορισμός [A-ορθό πρωτόκολλο]

Ένα πρωτόκολλο P είναι A-ορθό αν για κάθε σύνολο T συναλλαγών πολλών επιπέδων ασφαλείας, το P(T) είναι A-ορθό.

Η πλήρης A-ορθότητα είναι συχνά δύσκολο να επιτευχθεί όταν εκτελούνται συναλλαγές πολλών επιπέδων ασφαλείας. Θα δούμε στη συνέχεια πως τα πρωτόκολλα μπορεί να

χρειαστεί να *αντισταθμίσουν* (*trade-off*) μέρος της A-ορθότητας για S-ορθότητα . Σε περιβάλλον πολλών επιπέδων ασφαλείας, οι χρήστες κάθε επιπέδου «βλέπουν» την κατάσταση των δεδομένων στο επίπεδο το οποίο βρίσκονται και κάτω από αυτό, έτσι η μερική A-ορθότητα ορίζεται σε σχέση με τα ορατά δεδομένα σε κάθε επίπεδο. Η ατομικότητα σε πολλά επίπεδα ασφαλείας ορίζει ότι σε ένα χρονοδιάγραμμα, αν οι λειτουργίες σε ένα επίπεδο ολοκληρωθούν, τότε όλες οι λειτουργίες στα χαμηλότερα επίπεδα επίσης θα ολοκληρωθούν. Οι λειτουργίες στα υψηλότερα επίπεδα θα ολοκληρωθούν μόνο αν όλες οι λειτουργίες στα χαμηλότερα επίπεδα έχουν ολοκληρωθεί. Παρ'όλ'αυτά, αυτό το κριτήριο ορθότητας επιτρέπει στις συναλλαγές να ολοκληρώνουν λειτουργίες σε κάποιο επίπεδο που δεν είναι γνωστό εκ των προτέρων. Η ατομικότητα σε πολλά επίπεδα ασφαλείας είναι το κριτήριο που χρησιμοποιούμε για να ορίσουμε την μερική ατομικότητα (A^--*ορθότητα*).

Ορισμός [A^--ορθότητα]

Έστω S_T είναι ένα χρονοδιάγραμμα για ένα σύνολο συναλλαγών T πολλών επιπέδων, ασφαλείας. Το S_T είναι A^--ορθό αν για όλα τα χρονοδιαγράμματα S_i στο S_T, $c_{ij} \in S_T$ συνεπάγεται $c_{ik} \in S_T$ για όλα τα $k < j$ που αντιστοιχούν στη write λειτουργία στη T_i συναλλαγή.

Στα πλαίσια του MUSET θεωρούμε ότι η A^--ορθότητα είναι η ελάχιστη απαίτηση σε ατομικότητα για κάθε πρωτόκολλο, καθώς αυτή εξασφαλίζει αν παρέχεται σταθερή άποψη (*view*) του αποτελέσματος της συναλλαγής από το επίπεδο L, όπου L είναι το υψηλότερο επίπεδο που μπόρεσε να ολοκληρωθεί. Χωρίς την ατομικότητα σε πολλά επίπεδα ασφαλείας, δεν εξασφαλίζεται σταθερή παρακολούθηση του αποτελέσματος.

4.2. Συνέπεια [Consistency]

Θα ορίσουμε αρχικά την C-ορθότητα και στη συνέχεια θα το ορίσουμε πιο συγκεκριμένα για χρονοδιαγράμματα πολλών επιπέδων ασφαλείας.

Ορισμός [C-ορθότητα]

Ένα χρονοδιάγραμμα S_i για μία συναλλαγή πολλών επιπέδων ασφαλείας T_i είναι C-ορθό αν είναι ισοδύναμο με ένα χρονοδιάγραμμα S_i', έτσι ώστε για όλες τις συγκρουόμενες λειτουργίες $o, o' \in T_i$, να διατηρείται η διάταξη των συγκρούσεων κατά $<_i$.

Η C-ορθότητα απαιτεί το πρωτόκολλο να διατηρεί τα αποτελέσματα της αρχικής διάταξης όλων των συγκρούσεων εντός της ίδιας συναλλαγής. Μέσα σε μια συναλλαγή T_i υπάρχουν δύο πιθανοί τύποι συγκρούσεων στο ίδιο επίπεδο ασφαλείας:

1. $r_j[x_k] <_i w_k[x_k]$, όπου $j > k$ (η ανισότητα αυτή μπορεί εναλλακτικά να συμβολίζεται j sdom k), που ονομάζεται read-first σύγκρουση.
2. $w_k[x_k] <_i r_j[x_k]$, όπου $j > k$ (ή εναλλακτικά j sdom k), που ονομάζεται write-first σύγκρουση.

Σε ένα χρονοδιάγραμμα S_i για τη συναλλαγή T_i, οι παραπάνω συγκρούσεις μεταφράζονται σε συγκρούσεις μεταξύ της $r_j[x_k]$ και της λειτουργίας ολοκλήρωσης c_k. Επομένως, όπως μας λέει και το παρακάτω πόρισμα, ένα χρονοδιάγραμμα είναι C-ορθό αν είναι ισοδύναμο με ένα άλλο χρονοδιάγραμμα, όπου η διάταξη του υψηλού επιπέδου αναγνώσεων και του χαμηλού επιπέδου εγγραφών κατά τη διάρκεια μιας συναλλαγής, διατηρείται σε σχέση με τις υψηλού επιπέδου αναγνώσεις και τις χαμηλού επιπέδου ολοκληρώσεις σε ένα χρονοδιάγραμμα.

Πόρισμα

Ένα χρονοδιάγραμμα S_i για μια συναλλαγή πολλών επιπέδων ασφαλείας T_i είναι C-ορθό αν είναι ισοδύναμο με ένα χρονοδιάγραμμα S_i', έτσι ώστε για όλες τις συγκρουόμενες λειτουργίες $r_j[x_k], w_k[x_k]$, όπου $j > k$ (ή εναλλακτικά j sdom k), να ισχύει $r_j[x_k] <_i w_k[x_k]$ αν $r_j[x_k] <_{S\_i'} c_k$.

Ορισμός [C-ορθό πρωτόκολλο]

Ένα πρωτόκολλο είναι C-ορθό αν για κάθε σύνολο συναλλαγών πολλών επιπέδων ασφάλειας T_i, το P(T) είναι C-ορθό.

4.3. Απομόνωση [Isolation]

Ο ορισμός της *I-ορθότητας* βασίζεται στις γενικές αρχές σειριοποίησης.

Ορισμός [Σειριακό χρονοδιάγραμμα]

Έστω S_T είναι ένα χρονοδιάγραμμα ενός συνόλου συναλλαγών πολλών επιπέδων ασφαλείας T_i. Το S_T είναι σειριακό αν για όλα τα χρονοδιαγράμματα S_i και S_j μέσα στο S_T, αν $\exists o_i \in S_i$, $o_j \in S_j$ τέτοιο ώστε $o_i <_{S_T} o_j$, τότε $\forall o_i' \in S_i$, $o_j' \in S_j, o_i' <_{S_T} o_j'$.

Δηλαδή, ένα χρονοδιάγραμμα S_T είναι I-ορθό αν αυτό είναι ισοδύναμο σε ένα σειριακό χρονοδιάγραμμα S_T'.

Ορισμός [I-ορθό πρωτόκολλο]

Ένα πρωτόκολλο είναι I-ορθό αν για κάθε σύνολο συναλλαγών πολλών επιπέδων ασφάλειας T_i, το P(T) είναι I-ορθό.

4.4. Ασφάλεια [Security]

Οι συναλλαγές πολλών επιπέδων ασφαλείας πρέπει να συμμορφώνονται με τους κανόνες της Απλής-Ιδιότητας-Ασφαλείας και της Ιδιότητας-* οι οποίοι αποτρέπουν τη ροή πληροφοριών από τα υψηλότερα επίπεδα ασφαλείας στα χαμηλότερα. Παρόλ'αυτά, η σειρά εκτέλεσης των λειτουργιών σε μία συναλλαγή πολλών επιπέδων ασφαλείας ίσως από μόνη της να προκαλέσει κάποιες παραβιάσεις σε ότι αφορά τα θέματα ασφάλειας. Για παράδειγμα, αν οι λειτουργίες στα υψηλά επίπεδα ασφάλειας μπορούν να προκαλέσουν την απόρριψη λειτουργιών σε ένα χαμηλότερο, τότε αυτή η παρέμβαση παραβιάζει την ασφάλεια.. Έτσι ένας παρατηρητής στο χαμηλό επίπεδο δεν θα μπορούσε να συμπεράνει την ύπαρξη υψηλότερων λειτουργιών κατά τη διάρκεια της ίδιας συναλλαγής.

Για να κατανοήσουμε την έννοια της μη-παρέμβασης, το κριτήριο ασφαλείας που ορίζεται παρακάτω, συγκρίνει ένα χρονοδιάγραμμα με λειτουργίες σε υψηλά και χαμηλά επίπεδα, με το ίδιο χρονοδιάγραμμα έχοντας όμως αφαιρέσει όλες τις υψηλές λειτουργίες που υπάρχουν σε αυτό. Αν τα αποτελέσματα των χαμηλών λειτουργιών είναι τα ίδια και στα δύο χρονοδιαγράμματα, τότε οι υψηλές λειτουργίες δεν παρεμβαίνουν στις χαμηλότερες. Παρακάτω θα χρησιμοποιήσουμε τη συνάρτηση *purge* για να αναφερθούμε σε ένα χρονοδιάγραμμα ή μία συναλλαγή με διαγραμμένες τις υψηλού επιπέδου ασφάλειας λειτουργίες.

Για ένα χρονοδιάγραμμα S_T, ένα πρωτόκολλο P και ένα σύνολο συναλλαγών πολλών επιπέδων ασφαλείας T_i, η συνάρτηση purge(S_T, L) θα επιστρέφει ένα νέο χρονοδιάγραμμα S_T', όπου όλες οι λειτουργίες του δεν θα επικρατούνται από το επίπεδο L, του οποίου οι λειτουργίες έχουν απομακρυνθεί . Όμοια, η purge(T_i, L) θα επιστρέφει ένα νέο σύνολο συναλλαγών με όλος τις λειτουργίες του να μην επικρατούνται από το επίπεδο ασφαλείας L .

Ορισμός [I-ορθότητα]

Έστω το χρονοδιάγραμμα S_T = P(T) για ένα πρωτόκολλο P και ένα σύνολο συναλλαγών T. Το S_T είναι S-ορθό (σε σχέση με το P), αν για όλα τα επίπεδα L, ισχύει purge(S_T, L) = purge(T_i, L) και καμία λειτουργία στο S_T δεν πρέπει να περιμένει για να ξεκινήσει μέχρι να ολοκληρωθεί μία υψηλότερη λειτουργία στο ίδιο χρονοδιάγραμμα.

Ο ορισμός αυτός της ορθότητας παρέχει με αυστηρό τρόπο τον αποκλεισμό της παρέμβασης ανάμεσα στα επίπεδα ασφαλείας, συμπεριλαμβάνοντας και την αποφυγή χρονικών καναλιών (*timing channels*). Ένα χρονικό κανάλι εμφανίζεται όταν μία χαμηλή λειτουργία δεν μπορεί να ξεκινήσει αν δεν έχει ολοκληρωθεί κάποια υψηλότερη. Ένας πιο χαλαρός ορισμός παρέχει την μη-παρέμβαση που επιτρέπει έναν απλό τύπο χρονικών καναλιών, όπου η απόρριψη λειτουργιών στα υψηλά επίπεδα ασφαλείας ίσως αποτρέψει την εκκίνηση όλων των λειτουργιών σε κάποιο χαμηλότερο επίπεδο. Η Μερική-Ορθότητα (S^--*ορθότητα*) επιτυγχάνεται με αυτή τη χαλάρωση του αρχικού ορισμού.

Ορισμός [S-ορθό πρωτόκολλο]

Ένα πρωτόκολλο P είναι S-ορθό αν για κάθε σύνολο συναλλαγών πολλών επιπέδων ασφαλείας T, το S_T = P(T) είναι S-ορθό.

Γενικά, περιγράφουμε τα πρωτόκολλα εκτέλεσης με το να αναφέρουμε τις ιδιότητες τους, για παράδειγμα ένα CIS-πρωτόκολλο παράγει χρονοδιαγράμματα τα οποία είναι C-, I- και S-ορθά, αλλά όχι ατομικά ένα A^-CIS πρωτόκολλο έχει «χαλαρή» ατομικότητα, και πλήρη συνέπεια, απομόνωση και ασφάλεια. Τα κριτήρια ορθότητας που περιγράψαμε δεν προστίθενται απλά σε ένα πρωτόκολλο. Όπως θα δούμε παρακάτω, κάποιοι συνδυασμοί δεν είναι δυνατόν να επιτευχθούν μιας και περιορίζονται από κάποια θεωρήματα ή λήματα.

5. Αντισταθμίσεις μεταξύ των κριτηρίων ορθότητας για συναλλαγές πολλών επιπέδων ασφαλείας

Ενώ τα κριτήρια ορθότητας που περιγράψαμε παραπάνω αποτελούν την ιδανική εκτέλεση συναλλαγών πολλών επιπέδων ασφαλείας, στην πραγματικότητα δεν είναι εφικτό για οποιοδήποτε πρωτόκολλο να πετύχει πλήρη *ACIS* ορθότητα. Παρά το γεγονός πως ένα συγκεκριμένο χρονοδιάγραμμα μπορεί να πετύχει αυτούς τους στόχους, κανένα πρωτόκολλο δεν μπορεί να εγγυηθεί πως θα παράγει πάντα ένα πλήρως *ACIS* ορθό χρονοδιάγραμμα. Στη συνέχεια θα παρουσιάσουμε την θεωρητική βάση για τις αντισταθμίσεις μεταξύ των κριτηρίων ορθότητας, και έπειτα θα δούμε τα πρωτόκολλα που εφαρμόζουν τις αντισταθμίσεις αυτές. Θα εστιάσουμε στην αντιστάθμιση μεταξύ της ασφάλειας και των άλλων κριτηρίων.

5.1. Αντιστάθμιση Ατομικότητας και Ασφάλειας

Τα κριτήρια για την ατομικότητα και την ασφάλεια έρχονται σε αντίθεση. Η A-ορθότητα συνεπάγεται ισχυρή εξάρτηση μεταξύ των λειτουργιών σε διαφορετικά επίπεδα ασφαλείας, ενώ η S-ορθότητα εμποδίζει τις λειτουργίες στα υψηλά επίπεδα να παρέμβουν στις χαμηλότερες. Πριν δούμε αυτή την αντίθεση, ας δούμε εν συντομία κάποιες μεθόδους για να πετύχουμε ατομικότητα.

- Ένα τμήμα λειτουργιών μπορεί να απορριφθεί οποιαδήποτε στιγμή μέχρι αυτό να ολοκληρωθεί.
- Ένα τμήμα λειτουργιών δεν μπορεί να εγγυηθεί πως θα ολοκληρωθεί, λόγω παραγόντων που δεν ελέγχει το πρωτόκολλο, όπως lock time-outs.

Η λειτουργία της *πρόωρης ολοκλήρωσης* (*precommit*) ορίζει ένα σημείο στο οποίο όλα τα επίπεδα είναι έτοιμα να ολοκληρωθούν, αλλά κανένα δεν έχει ολοκληρωθεί ακόμα. Έτσι, η πρόωρη ολοκλήρωση επιτρέπει την A-ορθότητα για συναλλαγές πολλών επιπέδων ασφαλείας, όπως θα δούμε στο παρακάτω λήμμα.. Αυτό το λήμμα δείχνει πως σε ένα A-ορθό πρωτόκολλο, οι λειτουργίες σε όλα τα επίπεδα ασφαλείας θα ολοκληρωθούν πρόωρα, πριν ολοκληρωθεί οποιοδήποτε επίπεδο.

Λήμμα πρόωρης ολοκλήρωσης

Ένα πρωτόκολλο είναι A-ορθό αν για κάθε σύνολο συναλλαγών T πολλών επιπέδων ασφαλείας και για κάθε χρονοδιάγραμμα S_T = P(T) ισχύει:

Για κάθε $S_{Ti} \in S_T, \forall c_{ij} \in S_{Ti_i}$, και $\forall k$ στο T_i ώστε $p_{ik} <_{S_T} c_{ij}$.

Το παραπάνω λήμμα δείχνει πως ένα A-ορθό πρωτόκολλο πρέπει πρώτα να προσπαθήσει να ολοκληρώσει πρόωρα όλες τις λειτουργίες, πριν τις ολοκληρώσει. Όταν η μια λειτουργία ολοκληρωθεί, τότε ακολουθούν και όλες οι υπόλοιπες, οδηγώντας σε ένα A-ορθό πρωτόκολλο. Αν οποιαδήποτε λειτουργία κατά τη διάρκεια όλης της παραπάνω διαδικασίας απορριφθεί, τότε απορρίπτονται όλες οι διαδικασίες, είτε έχουν ολοκληρωθεί, είτε όχι. Αυτό έχει ως συνέπεια την παρέμβαση μεταξύ λειτουργιών σε διαφορετικά επίπεδα ασφαλείας, γεγονός που έρχεται σε σύγκρουση με τις βασικές αρχές της ασφάλειας.

Θεώρημα περιορισμού A-ορθότητας και S-ορθότητας

Από τα παραπάνω καταλήγουμε στο ότι κανένα πρωτόκολλο δεν είναι δυνατό να είναι συγχρόνως A-ορθό και S-ορθό. Παρ΄ολ'αυτά, με το να εκτελεστούν οι λειτουργίες κατά αύξουσα σειρά, σε σχέση με το επίπεδο ασφαλείας, μπορεί να επιτευχθεί η S-ορθότητα καθώς διατηρείται η A^--ορθότητα, όπως ορίζεται στο παρακάτω λήμμα:

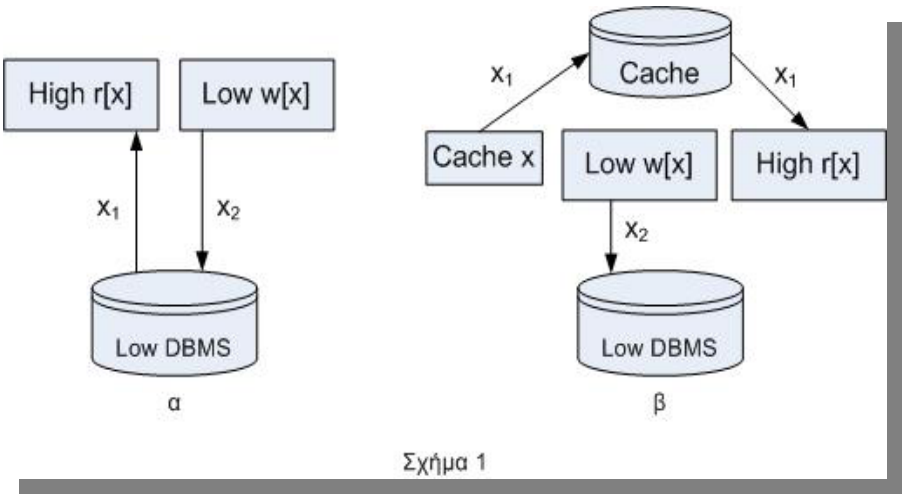
Λήμμα χαμηλής εκτέλεσης (low-first lemma)

Έστω P είναι ένα πρωτόκολλο χαμηλής εκτέλεσης που παράγει χρονοδιαγράμματα S για τα οποία για όλες τις λειτουργίες o_{ij} και o_{ik}, τέτοιες ώστε $k > j$ ή $k \text{ sdom } j$, ισχύει $o_{ij} <_S o_{ik}$. Αυτό το πρωτόκολλο χαμηλής εκτέλεσης είναι S-ορθό και A^--ορθό.

5.2. Αντιστάθμιση Συνέπειας και Ασφάλειας

Για να εκπληρώσουν την απαίτηση για S-ορθότητα, οι λειτουργίες στα διαφορετικά επίπεδα σε μία συναλλαγή πολλών επιπέδων ασφαλείας T_i ίσως χρειαστεί να αλλάξουν τη διάταξή τους στο χρονοδιάγραμμα S_{Ti}. Ωστόσο, μία τέτοια αναδιάταξη θα μπορούσε να παραβιάσει τις αρχές της C-ορθότητας. Ας σημειώσουμε εδώ πως σε ένα *χρονοδιάγραμμα χαμηλής εκτέλεσης (low-first schedule)*, η διάταξη των write-first συγκρούσεων $w_k[x_k] <_i r_j[x_k]$, όπου $j > k$ ή $j \text{ sdom } k$,διατηρείται. Συνεπώς, για χρονοδιαγράμματα χαμηλής εκτέλεσης, οι μόνες συγκρούσεις που μπορούν να αναδιαταχθούν, είναι οι read-first $r_j[x_k] <_i w_k[x_k]$, όπου $j > k$ ή $j \text{ sdom } k$. Στη συνέχεια θα περιγράψουμε τη χρήση της μνήμης cache με σκοπό την ορθή εκτέλεση κάθε συναλλαγής πολλών επιπέδων ασφαλείας με ένα χρονοδιάγραμμα χαμηλής εκτέλεσης και τον σχεδιασμό (με τη βοήθεια της μνήμης cache) πρωτοκόλλων τα οποία είναι συγχρόνως C-ορθά και S-ορθά.

Ας υποθέσουμε πως υπάρχει μία read-first σύγκρουση για το στοιχείο δεδομένων (data item) x. Όπως δείχνει το σχήμα 1.α η υψηλού επιπέδου read $r[x]$ διαβάζει την τιμή x_1 για x, και έπειτα η χαμηλού επιπέδου $w[x]$ γράφει την τιμή x_2. Αν οι συγκρουόμενες λειτουργίες αναδιαταζόταν για να επιτρέψουν την χαμηλού επιπέδου $w[x]$ να ολοκληρώσει πριν την υψηλού επιπέδου read $r[x]$, τότε θα διαβάζονταν η λάθος τιμή. Οι εγγραφές στη μνήμη cache *επικαλύπτοντα* (*overwritten*) κατά τη διάρκεια της εκτέλεσης μιας συναλλαγής. Οι αναγνώσεις έτσι οδηγούνται στις τιμές που είναι αποθηκευμένες στη μνήμη cache, αντί της άμεσης ανάγνωσης. Όπως βλέπουμε στο σχήμα 1.β, η τιμή x_1 αποθηκεύεται στη μνήμη cache προτού ανανεωθεί σε x_2. Όταν γίνει νέα αναδιάταξη, και εμφανιστεί η υψηλού επιπέδου read $r[x]$, διαβάζεται (η αρχική τιμή) από την cache, αντί της x_2.



Σχήμα 1

Έτσι η χρήση της μνήμης cache επιτρέπει σε μία υψηλού επιπέδου read να εκτελεστεί μετά από μία χαμηλού επιπέδου write, εξαλείφοντας την αρχική read-first σύγκρουση. Αυτή η προσέγγιση μπορεί να εφαρμοστεί και στις απλές συναλλαγές (single transactions), όπως περιγράφουν οι O.Costich ["Transaction processing with replicated architecture", 1992] και K.P.Smith ["Execution reordering for multilevel rules", 1994].

Στο τέλος της συναλλαγής πολλών επιπέδων ασφαλείας η μνήμη cache μπορεί να διαγραφεί και να μη χρειαστεί ποτέ να αποθηκευτεί στη μόνιμη ΒΔ.. Ας σημειώσουμε εδώ πως αν το χρονοδιάγραμμα δεν είναι χαμηλής εκτέλεσης, η προσέγγιση αυτή δεν μπορεί να εξαλείψει τις write-first συγκρούσεις, μιας και η κατάλληλη τιμή μπορεί να μην έχει εγγραφεί προηγουμένως στη μνήμη cache.

Το παρακάτω λήμμα δείχνει πως ένα πρωτόκολλο μετατρέπεται χρησιμοποιώντας τη μνήμη cache και αποδεικνύει πως το αποτέλεσμα είναι ένα ισοδύναμο χρονοδιάγραμμα με το αρχικό, πριν τη μετατροπή. Χρησιμοποιούμε την εντολή w_c για να γράψουμε στην μνήμη cache και την r_c για να διαβάσουμε αντικείμενα δεδομένων.

Λήμμα ισοδυναμίας με τη χρήση μνήμης cache

Έστω το χρονοδιάγραμμα S χαμηλής εκτέλεσης όπου $<...r_j[x_k], w_k[x_k]...>$,με $j > k$ ή j sdom k. Έστω τώρα το S' είναι ένα χρονοδιάγραμμα που δημιουργήθηκε με χρήση της μνήμης cache και είναι πανομοιότυπο με το S, εκτός από τις παρακάτω αλλαγές που προκάλεσε η χρήση της cache για τη τιμή του x_k:

- Η λειτουργία $w_h[x_k]$ αντικαταστάθηκε από την $w_c[x'_k]$ $w_h[x_k]$.
- Η λειτουργία $r_j[x_k]$ αντικαταστάθηκε από την $r_c[x'_k]$ και μετατοπίστηκε μετά την $w_c[x'_k]$.

Συνεπώς, με όλες τις υπόλοιπες λειτουργίες πανομοιότυπες με το S, το S' είναι το $< w_c[x'_k], r_c[x''_k], w_k[x_k] >$. Τότε, τα S και S' είναι ισοδύναμα.

5.3. Αντιστάθμιση Απομόνωσης και Ασφάλειας

Οι *έλεγχοι συνέπειας* (*concurrency controls*) χρησιμοποιούνται με σκοπό να εξασφαλίσουμε πως το παραγόμενο χρονοδιάγραμμα είναι *1-ορθό*. Παρακάτω θα παρουσιάσουμε δύο βασικές προσεγγίσεις, κλειδώματος (*locking*) και πολλαπλής-έκδοσης (*multi-versioning*), μαζί με ένα υβριδικό μοντέλο για το καθένα.

5.3.1. Τεχνικές κλειδώματος και 2PL

Στο κλείδωμα-δύο-φάσεων (*2PL*), οι συναλλαγές σειριοποιούνται με το να απαιτήσουν από κάθε συναλλαγή να αποκτήσει όλες τις κλειδαριές πριν απελευθερώσει έστω και μία [P.A.Bernstein, “Concurrency Control and Recovery in Database Systems”, 1987]. Κάθε πρωτόκολλο το οποίο χρησιμοποιεί το 2PL θα παράγει μόνο σειριοποιημένα χρονοδιαγράμματα. Ο βασικός ορισμός της μεθόδου 2PL πρέπει να τροποποιηθεί ελαφρά για να είναι κατάλληλη προς χρήση για τις συναλλαγές πολλών επιπέδων ασφαλείας. Ας υποθέσουμε πως τα χρονοδιαγράμματα περιέχουν τις λειτουργίες ol_i και ou_i, τις εντολές κλειδώματος και ξεκλειδώματος δηλαδή που χρειάζεται μία λειτουργία o_i (στη συναλλαγή i). Το 2PL θέτει τις παρακάτω απαιτήσεις στα χρονοδιαγράμματα:

Ορισμός [2-Phase-Locking]

Ένα χρονοδιάγραμμα είναι 2PL αν:

1. Υπάρχει $r_i \in S$, τότε $rl_i <_S r_i <_S ru_i$. Δηλαδή, κάθε εντολή ανάγνωσης συνοδεύεται από ένα ζευγάρι εντολών κλειδώματος-ξεκλειδώματος.
2. Υπάρχει $w_i \in S$, τότε $wl_i <_S w_i <_S c_i <_S wu_i$, ή $wl_i <_S w_i <_S a_i <_S wu_i$. Δηλαδή, κάθε εντολή γραψίματος συνοδεύεται από ένα ζευγάρι εντολών κλειδώματος-ξεκλειδώματος, ωστόσο η λειτουργία ολοκλήρωσης ή απόρριψης πρέπει επίσης να συμπεριλαμβάνεται στο ζεύγος αυτό.
3. Υπάρχουν συγκρουόμενες λειτουργίες $o_i[x]$ και $o_j[x] \in S$, και για τις συναλλαγές αυτές ισχύει $T_i \neq T_j$, τότε είτε $ou_i[x] <_S ol_j[x]$ ή $ou_j[x] <_S ol_i[x]$. Δηλαδή, μία λειτουργία πρέπει να απελευθερώσει την κλειδαριά της προτού μία συγκρουόμενη λειτουργία να μπορεί να κλειδώσει το ίδιο στοιχείο δεδομένων.
4. Υπάρχουν $o_{ij}[x], o_{ik}[y] \in S$, τότε $ol_{ij}[x] <_S ou_{ik}[y]$. Αυτό αποτελεί τον κανόνα πολλαπλών-επιπέδων-2PL, ορίζοντας πως καμία απελευθέρωση κλειδαριάς (σε κανένα επίπεδο) κατά την εκτέλεση μιας συναλλαγής δεν μπορεί να συνυπάρχει με μία απόκτηση κλειδαριάς από την ίδια συναλλαγή.

Όσον αφορά τη μέθοδο κλειδώματος, οι συναλλαγές πολλών επιπέδων ασφαλείας διαφέρουν από τις απλές συναλλαγές, σε σχέση με τους δύο διαφορετικούς τύπους κλειδώματος:

1. *Εσωτερικές κλειδαριές* (*Intra-level locks*): Είναι όλες οι κλειδαριές ανάγνωσης και εγγραφής μέσα σε ένα επίπεδο.
2. *Εξωτερικές κλειδαριές* (*Inter-level locks*): Είναι οι κλειδαριές ανάγνωσης οι οποίες καλύπτουν μία υψηλή εντολή ανάγνωσης ενός χαμηλού στοιχείου.

Οι εσωτερικές κλειδαριές εμφανίζονται στο εξάρτημα του DBMS το οποίο είναι υπεύθυνο για την εκτέλεση των λειτουργιών σε κάθε επίπεδο ασφαλείας ξεχωριστά.

Οι εξωτερικές κλειδαριές εμφανίζονται όταν μία λειτουργία ανάγνωσης $r_{ij}[x_l]$, όπου j sdom l εκτελεστεί. Οι εξωτερικές κλειδαριές αποτελούν ένα πολύ γνωστό και σοβαρό κίνδυνο [INFORMIX, “Guide to SQL”, 1993]. Μία υψηλή εντολή ανάγνωσης πάνω σε ένα χαμηλό στοιχείο δεδομένων μπορεί να τροποποιηθεί σε ένα κανάλι ασφαλείας (*security channel*) με σκοπό να «επικοινωνήσει» με ένα χαμηλότερο επίπεδο. Για να το αποφύγουμε

αυτό, οι συγκρούσεις μεταξύ των υψηλών κλειδαριών ανάγνωσης και χαμηλών κλειδαριών εγγραφής μπορούν να λυθούν με τη χρήση κάποιων «χαλαρών» κλειδαριών ανάγνωσης (από ένα έμπιστο πάντα DBMS). Οι «χαλαρές» κλειδαριές χορηγούνται εκ μέρους της εντολής $r_{ij}[x_l]$ και απελευθερώνονται όποτε μία άλλη συναλλαγή T_m απαιτεί να αποκτήσει μία αποκλειστική κλειδαριά πάνω στο στοιχείο δεδομένων x_l. Η συναλλαγή T_m δεν γνωρίζει την ύπαρξη της «χαλαρής» κλειδαριάς που σπάει, και η T_m μερικώς αποκτά την κλειδαριά. Ενώ είναι απαραίτητες για να εγγυηθούν την ασφάλεια, οι «χαλαρές» κλειδαριές περιπλέκουν την επίτευξη της I-ορθότητας για τα πρωτόκολλα πολλών επιπέδων ασφαλείας. Με τις «χαλαρές» κλειδαριές, μία απελευθέρωση κλειδαριάς μπορεί να εμφανιστεί απρόβλεπτα (πριν ή μετά από μία λειτουργία ανάγνωσης). Στη συνέχεια θα αποδείξουμε πως η χρήση των «χαλαρών» κλειδαριών συνεπάγεται την αντισυμβατότητα του 2PL και της S-ορθότητας.

Η συνθήκη 1 του 2PL δεν επηρεάζεται από τη χρήση των «χαλαρών» κλειδαριών. Αν η κλειδαριά σπάσει πριν την λειτουργία ανάγνωσης $r_{ij}[x_l]$, τότε η διαδικασία της ανάγνωσης δεν θα μπορεί να προχωρήσει. Όταν εμφανιστεί η ανάγνωση, η συνθήκη 1 τηρείται με οποιαδήποτε απελευθέρωση πραγματοποιηθεί, πρόωρη ή μη. Η συνθήκη 2 αφορά μόνο τις κλειδαριές εγγραφής. Συνεπώς, η συνθήκη 2 δεν επηρεάζεται από τη χρήση των «χαλαρών» κλειδαριών. Η συνθήκη 3 απαιτεί την απελευθέρωση της σπασμένης κλειδαριάς ανάγνωσης πριν από την παροχή μιας χαμηλής κλειδαριάς εγγραφής, και επομένως τηρείται εξ' ορισμού όποτε μία «χαλαρή» κλειδαριά σπάσει. Τέλος, η συνθήκη 4 επίσης τηρείται άσχετα με το πότε μία «χαλαρή» κλειδαριά θα σπάσει.

Ακόμα και με την παραπάνω δυνατότητα να χρησιμοποιήσουμε «χαλαρές» κλειδαριές στα 2PL χρονοδιαγράμματα, υπάρχει πρόβλημα σχετικά με την I-ορθότητα. Το 2PL εξασφαλίζει την I-ορθότητα για συναλλαγές απλού επιπέδου, ωστόσο, στη γενική μορφή του δεν είναι αρκετό για να εξασφαλίσει την I-ορθότητα στις συναλλαγές πολλών επιπέδων ασφαλείας. Τα χρονοδιαγράμματα πολλών επιπέδων ασφαλείας μπορεί να περιέχουν λειτουργίες σε μη-ισοδύναμα επίπεδα, γεγονός που θέτει κάθε συνεργασία-επικοινωνία μεταξύ αυτών των επιπέδων κίνδυνο για την εμφάνιση καναλιών ασφαλείας.

Θεώρημα ασυμβατότητας 2PL και S-ορθότητας

Κανένα πρωτόκολλο, το οποίο χρησιμοποιεί τη μέθοδο του 2PL, δεν είμαι δυνατό να παράγει χρονοδιαγράμματα τα οποία να είναι S-ορθά.

5.3.2. Τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων

Ένα πρωτόκολλο διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων παράγει one-copy σειριοποιήσημα χρονοδιαγράμματα (και όχι απλά σειριοποιήσημα) των απλών συναλλαγών. Σε αντίθεση με τις τεχνικές κλειδώματος, η τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων μπορεί να τροποποιηθεί για να χρησιμοποιηθεί σε συναλλαγές πολλών επιπέδων ασφαλείας, με τρόπο τέτοιο ώστε να επιτύχουμε I- και S-ορθότητα. Σε αυτή τη τεχνική, το *σημείο συνέπειας (consistency point)* αποτελεί το σημείο έναρξης της συναλλαγής. Κάθε συναλλαγή σφραγίζεται χρονικά σύμφωνα με το παραπάνω σημείο. Κάθε λειτουργία εγγραφής παράγει ένα νέο χρονικά σφραγισμένο σημείο, σε σχέση με το σημείο συνέπειας της συναλλαγής στην οποία γράφει. Οι λειτουργίες ανάγνωσης οδηγούνται στην τελευταία έκδοση του σημείου συνέπειας της συναλλαγής στην οποία ανήκουν. Μπορεί, ωστόσο, αυτή η έκδοση να μην είναι διαθέσιμη τη συγκεκριμένη χρονική στιγμή της ανάγνωσης. Ας υποθέσουμε πως μία συναλλαγή T_1 διαβάζει ένα στοιχείο δεδομένων το οποίο *αργότερα* γράφεται από μία άλλη συναλλαγή T_2, αλλά χρονική_σφραγίδα(T_2)< χρονική_σφραγίδα(T_1). Αυτή η αργοπορημένη εγγραφή της T_2 δημιουργεί μία νέα έκδοση για το στοιχείο δεδομένων που η T_1 έπρεπε να είχε διαβάσει. Η λύση που προτείνεται με την τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων είναι να οπισθοδρομήσουμε πίσω (*roll back*) στην T_2.

Μόνο δύο σημεία της τεχνικής επηρεάζονται από την εισαγωγή πολλών επιπέδων ασφαλείας:

1.Η ίδια χρονική σφραγίδα (σημείο συνέπειας) πρέπει να «μοιράζεται» από όλα τα επίπεδα ασφαλείας της συναλλαγής για να υπάρχει I-ορθότητα. Πρέπει, άσχετα με το αν μία λειτουργία εγγραφής είναι υψηλού ή χαμηλού επιπέδου, κάθε τιμή να δέχεται την ίδια χρονοσφραγίδα.

2. Όπως και στη τεχνική κλειδώματος, υψηλές λειτουργίες ανάγνωσης πάνω σε χαμηλά δεδομένα απαιτούν ειδική μεταχείριση. Στη τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων ένα *κανάλι ασφαλείας* (*security channel*) εμφανίζεται

όταν μία χαμηλή εγγραφή οπισθοδρομείται επειδή μία υψηλή ανάγνωση δεν είδε τη σωστή τιμή του στοιχείου δεδομένων.

Για να επιτύχουμε S-ορθότητα στη τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων για πολλά επίπεδα ασφαλείας, οι λειτουργίες εκτελούνται σε χαμηλή πρώτα διάταξη (*low-first execution*). Επειδή η χρονική σφραγίδα πρέπει να «μοιράζεται» σε όλα τα επίπεδα, για χρονοσφραγίδα (σημείο συνέπειας) της συναλλαγής επιλέγεται η στιγμή πριν (η ακριβώς όταν) αρχίσουν οι χαμηλότερες λειτουργίες. Αυτή η χρονική σφραγίδα περνιέται έπειτα προς τα πάνω στα επίπεδα εκτέλεσης, εξασφαλίζοντας παράλληλα και την I-ορθότητα. (Σημείωση: Επειδή το σημείο συνέπειας δεν εξαρτάται από το χρόνο των προηγούμενων λειτουργιών στη συναλλαγή, όπως συμβαίνει στο 2PL, στη τεχνική διάταξης πολλαπλών-εκδόσεων-χρονικών-σφραγίδων για πολλά επίπεδα ασφαλείας δεν εμφανίζεται κανένα *χρονικό κανάλι [timing channel]* λόγω του ότι πετύχαμε I-ορθότητα.).

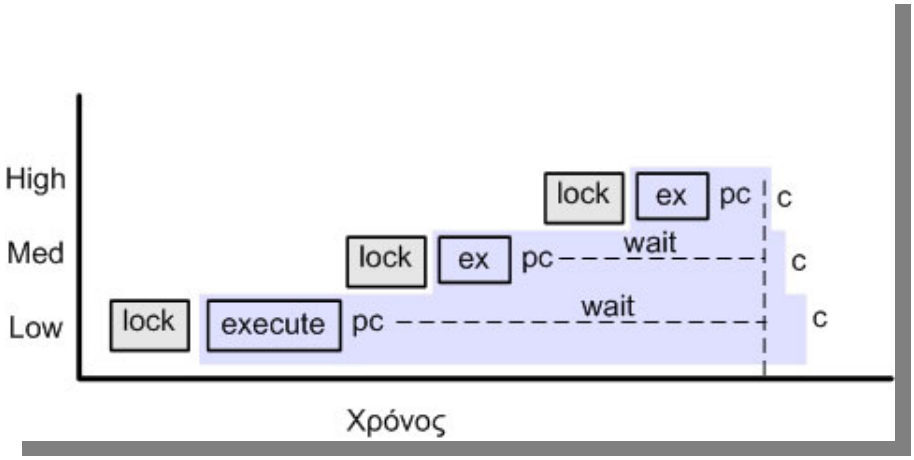
6. Πρωτόκολλα εκτέλεσης

Όπως έχουμε προαναφέρει, ένα πρωτόκολλο λαμβάνει ένα σύνολο από συναλλαγές πολλών επιπέδων ασφαλείας και παράγει ένα χρονοδιάγραμμα για την εκτέλεσή τους. Κάθε ένα πρωτόκολλο μπορεί να χαρακτηριστεί από την προσέγγισή του στο να επιτύχει ατομικότητα, συνέπεια, απομόνωση και ασφάλεια. Παρά το γεγονός του ότι κανένα πρωτόκολλο δεν μπορεί να είναι πλήρως ACIS ορθό, είναι ιδιαίτερα επιθυμητό να πετύχουμε το μεγαλύτερο πιθανό ποσοστό ορθότητας κάθε κατηγορίας κριτηρίων. Παρακάτω θα παρουσιάσουμε τρία πρωτόκολλα εκτέλεσης τα οποία πετυχαίνουν ένα υψηλό επίπεδο ορθότητας. Το πρώτο και το δεύτερο πρωτόκολλο παρουσιάζουν τα δύο ενδεχόμενα, όσον αφορά το θεώρημα περιορισμού της ατομικότητας και της ασφάλειας: το πρώτο είναι $ACIS^-$ και το δεύτερο A^-CIS ορθό. Το τρίτο προσφέρει μία απλή αλλά και πρακτική, πλήρως S-ορθή, εναλλακτική λύση στα άλλα δύο πρωτόκολλα. Κάθε πρωτόκολλο χρησιμοποιεί ένα γενικό πλέγμα, και δεν περιορίζονται από τη οποιαδήποτε δομή του πλέγματος αυτού. Επίσης, κάθε πρωτόκολλο χρησιμοποιεί μνήμη cache για να πετύχει C-ορθότητα και κάθε ένα εκτελεί τις λειτουργίες του με αύξουσα διάταξη (χαμηλή πρώτα εκτέλεση-low first execution). Μία διαφορετική προσέγγιση της I-ορθότητας γίνεται σε κάθε πρωτόκολλο. Τα τρία πρωτόκολλα που θα παρουσιάσουμε είναι :

- *Low-Ready-Wait-2PL (LRW-2PL): το οποίο είναι* $ACIS^-$ *ορθό.*
- *Low-First Multiversion Timestamp Ordering (LF-MVTO): το οποίο είναι* A^-CIS *ορθό.*
- *Low-First Hybrid Multiversioning (LF-HM): το οποίο είναι* A^-CI^-S *ορθό.*

6.1. Low-Ready-Wait-2PL ($ACIS^-$ ορθό)

Στο *LRW-2PL* οι λειτουργίες εκτελούνται κατά αύξουσα διάταξη, σε σχέση με το επίπεδο ασφαλείας στο οποίο ανήκουν, και ολοκληρώνονται κατά φθίνουσα διάταξη (σχήμα 1). Οι λειτουργίες σε ένα νέο επίπεδο ασφαλείας ξεκινούν, όταν όλες οι πρόωρα ολοκληρωμένες λειτουργίες στα παρακάτω επίπεδα έχουν εκτελεστεί. Οι πρώτες λειτουργίες σε κάθε επίπεδο αποκτούν όλες τις κλειδαριές, οι σκιασμένη περιοχή στο παρακάτω σχήμα δηλώνει την περίοδο όπου οι κλειδαριές είναι δεσμευμένες. Επιπλέων με τις εσωτερικές κλειδαριές ανάγνωσης και εγγραφής, οι «χαλαρές» κλειδαριές, που περιγράψαμε παραπάνω, υποστηρίζουν τις λειτουργίες ανάγνωσης. Όταν οι άλλες λειτουργίες σε ένα επίπεδο ασφαλείας έχουν εκτελεστεί, η λειτουργία που έχει πρόωρα ολοκληρωθεί σε αυτό το επίπεδο εκτελείται κανονικά. Οι λειτουργίες ολοκληρώνονται, αφού πρώτα όλα τα επίπεδα έχουν πρόωρα ολοκληρωθεί , σε φθίνουσα διάταξη.



Σχήμα 1

Το βήμα πρόωρης ολοκλήρωσης / αναμονής επιτρέπει στα επίπεδα είτε να ολοκληρωθούν είτε να απορριφθούν μαζί, πετυχαίνοντας πλήρη A-ορθότητα. Όπως είδαμε και στο Θεώρημα Περιορισμού της A- και S- ορθότητας, η φάση αναμονής εισάγει ένα χρονικό κανάλι (timing channel) επειδή η ικανότητα του κάθε πεδίου να ολοκληρωθεί εξαρτάται από την επιτυχημένη ολοκλήρωση του υψηλότερου πεδίου. Επιπλέον, σύμφωνα με το Θεώρημα Ασυμβατότητας 2PL και S-ορθότητας, το κανάλι αυτό πρέπει να υπάρχει, αφού χρησιμοποιούμε 2PL.

Η συνέπεια πετυχαίνεται με την χαμηλή πρώτα εκτέλεση και τη χρήση μνήμης cache. Η χρήση «χαλαρών» κλειδαριών εμποδίζει τη δημιουργία καναλιών ασφαλείας (security channels), αλλά πρέπει να απελευθερωθούν πριν από κάθε ολοκλήρωση και μετά από όλες τις

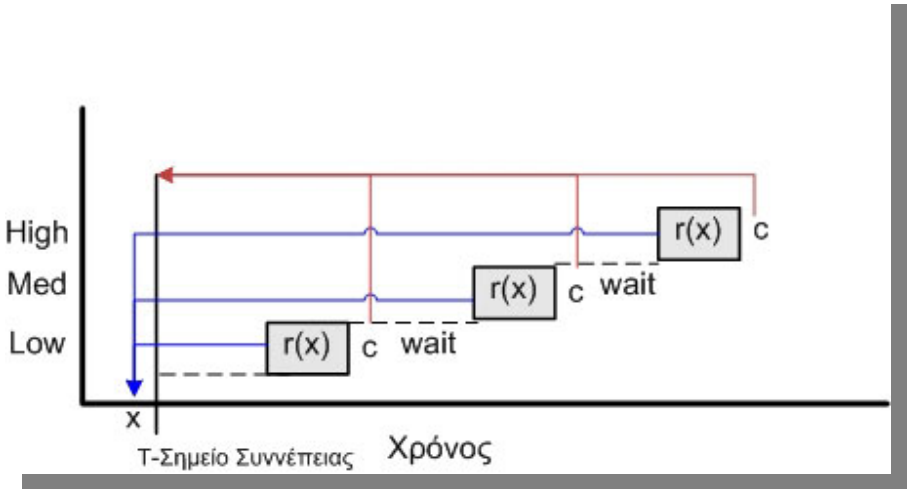
πρόωρες ολοκληρώσεις, επειδή ένα «προειδοποιητικό» ξεκλείδωμα στη φάση της ολοκλήρωσης θα κατάστρεφε την Α-ορθότητα.

6.2. Low-First with Multiversion Timestamp Order ($A^{-}CIS$ ορθό)

Το πρωτόκολλο *LF-MVTO* ξεκινάει με την επιλογή μιας χρονοσφραγίδας t (σε εύρος συναλλαγής) για να χρησιμοποιηθεί από τις λειτουργίες σε όλα τα επίπεδα ασφαλείας. Οι λειτουργίες εκτελούνται σε σχέση με τη χρονοσφραγίδα t, και ολοκληρώνονται σε αύξουσα διάταξη (σε σχέση με το επίπεδο στο οποίο αυτές ανήκουν), όπως φαίνεται στο σχήμα 2. Σύμφωνα με το πρωτόκολλο, η πρώτη λειτουργία σε κάθε επίπεδο ξεκινάει όταν:

1. όλες οι χαμηλότερες λειτουργίες της ίδιας συναλλαγής έχουν ολοκληρωθεί και
2. όλες οι λειτουργίες των άλλων συναλλαγών, στα επίπεδα από τα οποία η τρέχουσα συναλλαγή πραγματοποιεί αναγνώσεις, έχουν ολοκληρωθεί.

Η δεύτερη συνθήκη είναι υπεύθυνη για την φάση αναμονής που εμφανίζεται στο σχήμα 2. Όταν οι λειτουργίες σε ένα επίπεδο ασφαλείας ξεκινούν, όλες οι αναγνώσεις και οι εγγραφές πραγματοποιούνται σε σχέση με τη χρονοσφραγίδα t . Οι αργοπορημένες εγγραφές κατά τη διάρκεια της ίδιας συναλλαγής, θα προκαλέσουν την οπισθοδρόμηση κάποιων λειτουργιών. Μόλις μία λειτουργία απόρριψης (*abort*) σε οποιοδήποτε επίπεδο εμφανιστεί, όλες οι λειτουργίες οι οποίες δεν έχουν προλάβει να ολοκληρωθούν όμοια θα απορριφθούν.



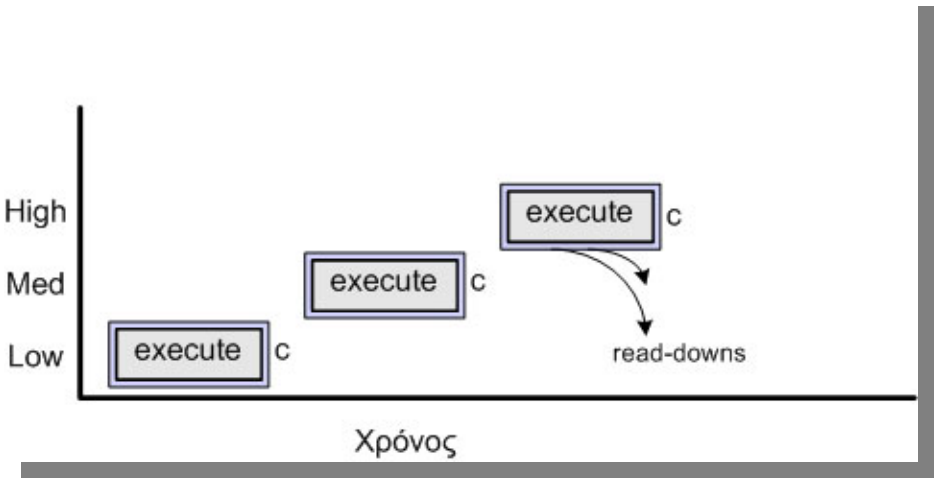
Σχήμα 2

Λόγω της απουσίας της φάσης της πρόωρης ολοκλήρωσης, οι λειτουργίες στα διαφορετικά επίπεδα ασφαλείας δεν μπορούν να ολοκληρωθούν ταυτόχρονα. Το πρωτόκολλο LF-MVTO

είναι A^- ορθό επειδή κανένας τομέας λειτουργιών δεν μπορεί να ολοκληρωθεί αν κάθε χαμηλότερος τομέας δεν έχει επίσης ολοκληρωθεί ή αν έχει απορριφθεί. Η χαμηλή-πρώτα εκτέλεση των λειτουργιών το κάνει C- και S-ορθό, σύμφωνα με το Λήμμα χαμηλής εκτέλεσης (*low-first lemma*). Είναι επίσης I-ορθό μιας και διατηρεί ένα σημείο συνέπειας, και σε αντίθεση με το κλείδωμα-δύο-φάσεων, όλα τα επίπεδα μπορούν να συγχρονιστούν χωρίς την εμφάνιση χρονικών καναλιών. Τα μειονεκτήματά του είναι η ανάγκη για τη καθυστέρηση κάποιων τομέων και η ανάγκη της αποθήκευσης των χρονοσφραγίδων εγγραφής και ανάγνωσης για κάθε στοιχείο δεδομένων. Αξίζει να σημειώσουμε εδώ πως, παρά το γεγονός ότι είδη του MVTO έχουν χρησιμοποιηθεί σε πολλά προϊόντα, κανένα MLS DBMS μέχρι σήμερα δεν έχει εφαρμόσει την αυθεντική μορφή του MVTO, όπως περιγράψαμε παραπάνω.

6.3. Low-First with Hybrid Multiversioning (A^-CI^-S ορθό)

Στο πρωτόκολλο *LF-HM*, που απεικονίζεται στο σχήμα 3, οι λειτουργίες εκτελούνται και ολοκληρώνονται κατά αύξουσα διάταξη, σε σχέση με το επίπεδο ασφαλείας στο οποίο ανήκουν. Μία λειτουργία σ ένα νέο επίπεδο μπορεί να ξεκινήσει όταν όλες οι λειτουργίες στα χαμηλότερα επίπεδα έχουν ολοκληρωθεί. Σε κάθε επίπεδο, οι κλειδαριές αποκτώνται για όλα τα στοιχεία δεδομένων τα οποία διαβάζονται ή εγγράφονται στο συγκεκριμένο επίπεδο. Καμία κλειδαριά δεν αποκτάται για τα στοιχεία δεδομένων τα οποία διαβάζονται από χαμηλότερα επίπεδα . Αντί για αυτό, οι αναγνώσεις πραγματοποιούνται σε σχέση με τη χρονική σφραγίδα για το επίπεδο αυτό. Όταν όλες οι αναγνώσεις και εγγραφές έχουν πραγματοποιηθεί, το επίπεδο ολοκληρώνεται και απελευθερώνει τις κλειδαριές του. Όπως και στο πρωτόκολλο LF-MVTO, η χαμηλή-πρώτα εκτέλεση καθιστά το LF-HM A^--ορθό, C-ορθό λόγω της χρήσης μνήμης cache και S-ορθό. Το LF-HM δεν είναι πλήρως I-ορθό λόγω του γεγονότος ότι οι επαναλαμβανόμενες αναγνώσεις δεν είναι εγγυημένες. Η υβριδική προσέγγιση κλειδώματος / multiversioning που χρησιμοποιεί το πρωτόκολλο αυτό είναι ευρέως αποδεκτή λόγω της αύξησης της παραγωγής συναλλαγών πάνω στην τεχνική κλειδώματος-δύο-φάσεων.



Σχήμα 3

Παρά το γεγονός ότι το LF-HM δεν είναι ούτε πλήρως A-ορθό ούτε πλήρως I-ορθό, έχει πολλά πλεονεκτήματα :

1. Οι υψηλές λειτουργίες οι οποίες δεν έχουν ολοκληρωθεί μπορούν να ξαναξεκινήσουν πολλές φορές με σκοπό να επιτευχθεί πλήρης ατομικότητα.. Οι επαναλήψεις δεν επηρεάζουν τον βαθμό της I-ορθότητας , όπως στο LF-MVTO. Το «τίμημα» των επαναλήψεων αυτών είναι ότι, με κάθε επανάληψη, η χρονική απόσταση μεταξύ των τιμών που διαβάζονται από διαφορετικά επίπεδα αυξάνεται.

2. Το LF-HM έχει λίγα από τα μειονεκτήματα του LF-MVTO. Δεν υπάρχει η ανάγκη να καθυστερήσουν λειτουργίες σε υψηλότερα επίπεδα ασφαλείας, δεν απαιτείται χρονική σφραγίδα ανάγνωσης – μόνο χρονική σφραγίδα εγγραφής και καμία οπισθοδρόμηση δεν εμφανίζεται.

3. Λόγω της χρήσης της μνήμης cache, οι επαναλαμβανόμενες αναγνώσεις είναι εγγυημένα ορθές για αυτές τις τιμές που έχει εγγράψει η ίδια η συναλλαγή.

4. Το πρωτόκολλο απαιτεί να ανταλλαχθούν μόνο έναν μικρό αριθμό μηνυμάτων, επομένως η υλοποίηση είναι σχετικά εύκολη και η επίδοσή του βελτιωμένη.

Κεφάλαιο 4

ASEP - Ένα ασφαλές και ευέλικτο πρωτόκολλο εκτέλεσης για κατανεμημένες συναλλαγές πολλών επιπέδων ασφαλείας

1. Εισαγωγή

Σε μία κατανεμημένη Βάση Δεδομένων (ΒΔ), υπάρχουν πολλαπλά και ποικίλα λογικά αντικείμενα-δεδομένα τα οποία είναι τοποθετημένα με φυσικό τρόπο σε διαφορετικές *τοποθεσίες ή κόμβους* (*sites-nodes*). Ένας χρήστης μπορεί να ξεκινήσει μία κατανεμημένη συναλλαγή σε οποιαδήποτε τοποθεσία ή κόμβο. Αν απαιτείται η πρόσβαση σε αντικείμενα τα οποία είναι αποθηκευμένα σε απομακρυσμένες τοποθεσίες, τότε η κατανεμημένη συναλλαγή εκκινεί μία *υπο-συναλλαγή* (*sub-transaction*) στη συγκεκριμένη τοποθεσία. Για να εγγυηθεί την ορθή εκτέλεση της κατανεμημένης συναλλαγής, κάθε τοποθεσία της κατανεμημένης ΒΔ είναι εξοπλισμένη με ένα *πρωτόκολλο συνέπειας* (*concurrency control protocol*) και με ένα *πρωτόκολλο εκτέλεσης* (*commit protocol*). Το πρωτόκολλο συνέπειας εγγυάται πως η εκτέλεση όλων των τοπικών συναλλαγών και υπο-συναλλαγών σε κάθε τοποθεσία είναι σειριοποιήσημη. Από την άλλη, το πρωτόκολλο εκτέλεσης εγγυάται την ατομικότητα όλων των κατανεμημένων συναλλαγών : όλες οι υπο-συναλλαγές πρέπει είτε να *ολοκληρωθούν* είτε να *απορριφθούν* (*commit-abort*). Όπως γνωρίζουμε, υπάρχουν πολλά πρωτόκολλα συνέπειας και εκτέλεσης με *κλείδωμα-δύο-φάσεων* και *ολοκλήρωση δύο φάσεων*, αντίστοιχα [P.A Berstein, V. Hadzilacos “Concurrency Control and Recovery in Database Systems”, 1987].

Ένα από τα σημαντικότερα προβλήματα των πρωτοκόλλων κλειδώματος σε συστήματα πολλών επιπέδων ασφαλείας είναι πως, για να αποφευχθεί ένα *συγκεκαλυμένο κανάλι* (covert channel), κάθε κλειδαριά ανάγνωσης που παρέχεται σε μία υψηλού επιπέδου ασφαλείας λειτουργία για ανάγνωση πάνω σε ένα χαμηλό στοιχείο δεδομένων, πρέπει να απελευθερωθεί, οπότε μία χαμηλή συναλλαγή προσπαθεί να πάρει μία κλειδαριά εγγραφής πάνω στο ίδιο στοιχείο. Δυστυχώς, το γεγονός αυτό καθιστά το πρωτόκολλο εκτέλεσης πρόωρης προετοιμασίας *EP* (θα δούμε μία τροποποίησή του στη συνέχεια), αδύναμο σε αυτό το θέμα [J.K Millen “The Cascading Problem for Interconnected Systems”, 1988]. Το πρόβλημα που προκύπτει είναι πως οι κλειδαριές ανάγνωσης μπορεί να απελευθερωθούν κατά τη διάρκεια του *παραθύρου αβεβαιότητας* (*window of uncertainty*) μιας συναλλαγής (την περίοδο κατά την οποία ένας συμμετέχων ψήφισε ναι στην ολοκλήρωση μιας υπο-συναλλαγής, αλλά πριν λάβει την *ναι ή όχι* απάντηση για την ολοκλήρωση από τον συντονιστή), γεγονός το οποίο μπορεί να οδηγήσει σε μη-σειριοποιήσημη εκτέλεση.

Για να επιτευχθεί η σειριοποίηση ο Atluri πρότεινε ένα ασφαλές πρωτόκολλο πρόωρης προετοιμασίας *SEP* [V. Atluri, "Degrees of Isolation, concurrency control protocols and commit protocols", 1994]. Το *SEP* εισάγει τη *φάση επιβεβαίωσης* (*confirmation phase*) πριν τη λήψη της απόφασης του *EP* με σκοπό να εγγυηθεί πως αν μία κλειδαριά ανάγνωσης έχει απελευθερωθεί πρόωρα κατά το παράθυρο αβεβαιότητας, η υπο-συναλλαγή και κατ' επέκταση ολόκληρη η κατανεμημένη συναλλαγή απορρίπτεται. Όπως όμως φαίνεται, η στρατηγική αυτή απόρριψής της συναλλαγής είναι υπερβολικά «συντηρητική». Ένα από τα σημαντικότερα πλεονεκτήματα της κατανεμημένης επεξεργασίας είναι η ανεκτικότητα σε σφάλματα, και το *SEP* αδυνατεί να το εκμεταλλευτεί. Μπορούμε να εισάγουμε χαρακτηριστικά *μερικής οπισθοδρόμησης* (*partial rollback*) με σκοπό να αποτρέψουμε τις συναλλαγές από το να απορρίπτονται κατά τον παραπάνω τρόπο.

Το *SEP* πάσχει από μία ακόμα αδυναμία: μία υπο-συναλλαγή μπορεί να πέσει σε κατάσταση λιμοκτονίας (*starvation*). Αυτή η κατάσταση εμφανίζεται όταν μία υψηλή υπο-συναλλαγή απορρίπτεται επανελημένα λόγω κάποιων «κακόβουλων» χαμηλών υπο-συναλλαγών. Ενώ αυτό είναι αναπόφευκτο αν θέλουμε να εγγυηθούμε την σειριοποίηση, μπορούμε να εισάγουμε μέτρα *εμπρόσθιας ανάκτησης* (*forward recovery*) τα οποία θα αποτρέπουν την λιμοκτονία (*ASEP*), παρά το κόστος σε σειριοποίηση, όπως θα δούμε στη συνέχεια. Για τον σκοπό αυτό, έχει προταθεί ένα *προηγμένο ασφαλές πρωτόκολλο πρόωρης προετοιμασίας* (*ASEP*), μαζί με ένα σύνολο βασικών στοιχείων-χαρακτηριστικών για να υποστηρίξουν το πρωτόκολλο αυτό. Τα στοιχεία αυτά μπορούν να χρησιμοποιηθούν και να τροποποιηθούν από τον προγραμματιστή για να «χαλαρώσει» την συνέπεια, να εγγυηθεί ατομικότητα και παράλληλα να αποτρέψει την λιμοκτονία των κατανεμημένων συναλλαγών.

2. Μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας

Ένα μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας αποτελείται από ένα σύνολο N *τοποθεσιών* (*sites*), όπου κάθε τοποθεσία $N \in N$ στο μοντέλο αυτό. Οι τοποθεσίες στο σύστημα επικοινωνούν μεταξύ τους μέσω *συνδέσεων* (*links*). Υποθέτουμε εδώ πως οι συνδέσεις αυτές είναι ασφαλείς (πιθανώς με χρήση κάποιων μεθόδων κρυπτογράφησης), έτσι ώστε καμία επικοινωνία μεταξύ δύο τοποθεσιών δεν μπορεί να πέσει θύμα κρυφής παρακολούθησης ή παραβίασης της ακεραιότητας της .

Ορίζουμε ένα μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας να είναι η τετράδα $< D,T,S,L >$, όπου D ένα σύνολο στοιχείων δεδομένων, T ένα σύνολο κατανεμημένων συναλλαγών, S ένα μερικώς διατεταγμένο σύνολο κλάσεων ή επιπέδων ασφαλείας

διατεταγμένα κατά $\leq$ και L η *αντιστοίχηση* (*mapping*) από το $D \cup T$ στο S. Όπως γνωρίζουμε, το επίπεδο ασφαλείας s_i κυριαρχεί στο επίπεδο s_j αν $s_j \leq s_i$ και κυριαρχεί αυστηρά το s_k αν $s_k \leq s_i$ και $s_k \neq s_i$. Για κάθε $x \in D$, $L(x) \in S$ και για κάθε $T \in T, L(T) \in S$. Με άλλα λόγια, κάθε στοιχείο δεδομένων, όπως επίσης και κάθε κατανεμημένη συναλλαγή, έχει μία κλάση ασφαλείας με την οποία σχετίζεται.

Περιγράφουμε την αντιστοίχηση L ,έτσι ώστε κάθε αντιστοίχηση σε ένα μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας N να είναι ένα διατεταγμένο σύνολο ζευγών κλάσεων ασφαλείας $L_{\min}(N)$ και $L_{\max}(N)$. Πρέπει πάντα να ισχύει $L_{\min}(N), L_{\max}(N) \in S$ και $L_{\min}(N) \leq L_{\max}(N)$. Με άλλα λόγια, κάθε μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας έχει ένα εύρος από κλάσεις ασφαλείας με τις οποίες σχετίζεται. Για κάθε στοιχείο δεδομένων x που είναι αποθηκευμένο σε ένα μοντέλο N ισχύει:

- $L_{\min}(N) \leq L(x) \leq L_{\max}(N)$.

Όμοια για κάθε συναλλαγή T που εκτελείται στο N ισχύει:

- $L_{\min}(N) \leq L(T) \leq L_{\max}(N)$.

Μία τοποθεσία N_i επιτρέπεται να επικοινωνήσει με μία άλλη N_j μόνο αν $L_{\max}(N_i) = L_{\max}(N_j)$ (αυτό αποτρέπει κάποιον κακόβουλο χρήστη από το να εκμεταλλευτεί τις διασυνδέσεις του δικτύου και να μεταφέρει δεδομένα από ένα υψηλό επίπεδο ασφαλείας σε έναν άλλο χρήστη ο οποίος έχει δηλωθεί σε χαμηλότερο επίπεδο ασφαλείας). Η πολιτική που ακολουθείται βασίζεται στο μοντέλο *Bell-LaPadula* και εισάγει τους παρακάτω περιορισμούς:

- Μία συναλλαγή T επιτρέπεται να διαβάσει ένα στοιχείο δεδομένων x μόνο αν $L(x) \leq L(T)$.
- Μία συναλλαγή T επιτρέπεται να γράψει σε ένα στοιχείο δεδομένων x μόνο αν $L(x) = L(T)$.

Έτσι, μια συναλλαγή μπορεί να διαβάζει δεδομένα στο επίπεδό της και παρακάτω, αλλά μπορεί να γράφει μόνο στα δεδομένα τα οποία ανήκουν στο επίπεδό της. Οι έμπιστοι

διαχειριστές της ΒΔ (*DBMS*) πρέπει να συμφωνούν με τον ίδιο περιορισμό. Επιπρόσθετα στους παραπάνω περιορισμούς, όπως ήδη γνωρίζουμε, ένα ασφαλές σύστημα κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας πρέπει να ελέγχει και για παράνομες μεταφορές δεδομένων μέσω συγκεκαλυμένων καναλιών.

2.1 Μοντέλο κατανεμημένων συναλλαγών

Για κάθε επίπεδο ασφαλείας s μέσα στο εύρος $\{L_{\min}, L_{\max}\}$ σε μία συγκεκριμένη τοποθεσία, υπάρχει ένας *διαχειριστής συναλλαγών* (*Transaction Manager-TM*) στο επίπεδο αυτό. Ένας χρήστης ο οποίος έχει δηλωθεί στο επίπεδο s, υποβάλει μία συναλλαγή T στον *TM* στο επίπεδο αυτό. Αν η συναλλαγή T δεν απαιτεί πρόσβαση σε απομακρυσμένες τοποθεσίες, ο *TM* απλά την εκτελεί ως μία τοπική συναλλαγή. Αν, από την άλλη, η T είναι μία κατανεμημένη συναλλαγή, ο *TM* αποσυνθέτει την κατανεμημένη συναλλαγή σε ένα σύνολο υπο-συναλλαγών T_i , με κάθε μία να κληρονομεί το επίπεδο ασφαλείας της κύριας συναλλαγής *T*, και προωθεί κάθε T_i στη κατάλληλη τοποθεσία N_i. Μία υπο-συναλλαγή T_i μπορεί να εκτελεστεί στην τοποθεσία N_i μόνο αν :

- Όποτε η T_i επιθυμεί να διαβάσει ένα στοιχείο δεδομένων x, $L_{\min}(N_i) \le L(x) \le L(T)$.
- Όποτε η T_i επιθυμεί να γράψει σε ένα στοιχείο δεδομένων x, $L(x) = L(T) \le L_{\max}(N_i)$.

Μία κατανεμημένη συναλλαγή περιγράφεται ως μία *εμφωλευμένη συναλλαγή* (*nested transaction*) [J.Eliot, “Nested Transactions”, 1985]. Κάθε εμφωλευμένη συναλλαγή έχει μία ομαδοποιημένη ιεραρχική δομή: κάθε εμφωλευμένη συναλλαγή αποτελείται είτε από βασικές λειτουργίες (ανάγνωση-εγγραφή) είτε από μερικές άλλες εμφωλευμένες συναλλαγές. Για απλότητα, υποθέτουμε πως μία κατανεμημένη συναλλαγή T στο μοντέλο μας έχει μόνο ένα επίπεδο από εμφωλευμένες συναλλαγές, στο πρώτο επίπεδο. Το πρώτο επίπεδο μπορεί να έχει επίσης λειτουργίες ανάγνωσης και εγγραφής οι οποίες μπορεί να εκτελούνται τοπικά στην αρχική τοποθεσία . Σε αυτή τη περίπτωση, οι παραπάνω λειτουργίες εκτελούνται ως μία υπο-συναλλαγή η οποία λαμβάνει χώρα στην αρχική τοποθεσία . Η υψηλότερη σε επίπεδο εμφώλευσης συναλλαγή T δρα σαν συντονιστής (*coordinator*) και επιτηρητής (*supervisory*) των λειτουργιών. Επίσης, η συναλλαγή αυτή εκκινεί το *ατομικό πρωτόκολλο εκτέλεσης* και θα

ονομάζεται πλέον συντονιστής. Κάθε υπο-συναλλαγή θα παρουσιάζεται στο πρωτόκολλο εκτέλεσης ως μία διαδικασία την οποία θα ονομάσουμε *συμμετέχων* (participant ή worker).

Οι εμφωλευμένες συναλλαγές επιλέχτηκαν για δύο βασικούς λόγους :

1. Αποτελεί έναν απλό και κομψό τρόπο να αναπαραστήσουμε ασύγχρονη και ανεξάρτητη εκτέλεση των υπο-συναλλαγών και

2. Σε μία εμφωλευμένη συναλλαγή, οι υπο-συναλλαγές μπορούν να αποτύχουν ανεξάρτητα από τις άλλες και από την κύρια συναλλαγή. Το χαρακτηριστικό αυτό μας δίνει τη δυνατότητα να εισάγουμε νέα εργαλεία, όπως η οπισθοδρόμηση και η επανεκτέλεση σε μία μοναδική τοποθεσία, σε περίπτωση τέτοιας αποτυχίας.

Σε σχέση με το πρωτόκολλο ελέγχου συνέπειας (*Concurrency Control Protocol*), υποθέτουμε πως ο διαχειριστής συναλλαγών *TM* χρησιμοποιεί ένα ασφαλές πρωτόκολλο κλειδώματος. Όπως έχουμε δει, τα πρωτόκολλα αυτά έχουν το κοινό χαρακτηριστικό πως όποτε μία συναλλαγή T_i επιθυμεί να διαβάσει (η να γράψει) ένα στοιχείο δεδομένων x, αυτή πρέπει πρώτα να αποκτήσει μία κλειδαριά ανάγνωσης (ή κλειδαριά εγγραφής) πάνω στο x. Επίσης, μία συναλλαγή T_i πρέπει να απελευθερώσει την κλειδαριά ανάγνωσης του στοιχείου δεδομένων x, όποτε μία άλλη συναλλαγή T_j, με $L(T_j) < L(T_i)$, απαιτεί μια κλειδαριά εγγραφής στο ίδιο στοιχείο δεδομένων. Γενικά, υποθέτουμε πως μία υπο-συναλλαγή αποκτά μία κλειδαριά την κατάλληλη στιγμή με ορθό τρόπο πριν επεξεργαστεί ένα στοιχείο δεδομένων.

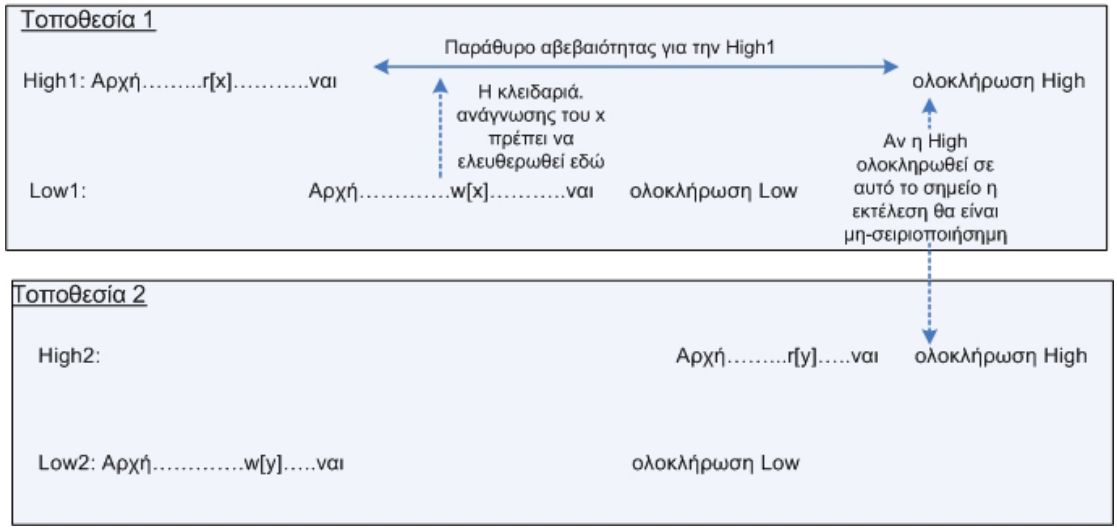
3. Ασφαλές πρωτόκολλο πρόωρης προετοιμασίας (*SEP*)

Όπως αναφέραμε στην αρχή, το τυπικό πρωτόκολλο *EP* αδυνατεί να δράσει ορθά σε ένα μοντέλο κατανεμημένης ΒΔ πολλών επιπέδων ασφαλείας μιας και το *EP* απαιτεί καμία κλειδαριά να μην ελευθερωθεί από μία υπο-συναλλαγή κατά το παράθυρο αβεβαιότητάς της. Παρ'όλ'αυτά, σε ένα περιβάλλον πολλών επιπέδων ασφαλείας, οι κλειδαριές ανάγνωσης σε ένα χαμηλού επιπέδου ασφαλείας στοιχείο δεδομένων, πρέπει να ελευθερωθούν όποτε μία χαμηλότερη συναλλαγή επιθυμεί να γράψει πάνω στο ίδιο στοιχείο.

Ας υποθέσουμε πως έχουμε το σενάριο του σχήματος 1, όπου υπάρχουν δύο κατανεμημένες συναλλαγές *High* και *Low*, τέτοιες ώστε $L(Low) < L(High)$.Κάθε κατανεμημένη

συναλλαγή αποτελείται από δύο υπο-συναλλαγές : *Low1, Low2 και High1, High2*, με τις *High1*

Low1 να εκτελούνται στην τοποθεσία 1 και τις *High2, Low2* να εκτελούνται στην τοποθεσία 2. Τα στοιχεία δεδομένων στα οποία έχουν πρόσβαση οι *High* και *Low* είναι τα x και y, με $L(x) = L(y) = L(Low)$. Το στοιχείο x βρίσκεται στην τοποθεσία 1 και το στοιχείο y στην τοποθεσία 2. Η σειρά εκτέλεσης των λειτουργιών φαίνεται στο σχήμα 1. Το γεγονός «ναι» σημαίνει πως η υπο-συναλλαγή έχει ολοκληρώσει την εκτέλεσή της και αποστέλλει μία ψήφο «ναι» στον συντονιστή. Σημειώνουμε πως, όταν η λειτουργία *w[x]* εκτελείται από την *Low1*, η λειτουργία δεν μπορεί να καθυστερήσει αναμένοντας την *High1* να ελευθερώσει την κλειδαριά ανάγνωσης του στοιχείου x. Αυτό γίνεται με σκοπό να αποφύγουμε την εμφάνιση συγκεκαλυμένου καναλιού ανάμεσα στα δύο επίπεδα ασφαλείας. Συνεπώς, παρά το γεγονός του ότι η *High1* βρίσκεται στο παράθυρο αβεβαιότητάς της όταν η *Low1* απαιτεί την κλειδαριά εγγραφής στο x, η κλειδαριά ανάγνωσης του x πρέπει να ελευθερωθεί από την *High1*. Το *EP* αδυνατεί να λάβει υπόψη του ότι μία κλειδαριά ανάγνωσης πρέπει να ελευθερωθεί σε περίπτωση ανάλογη με την παραπάνω. Αντίθετα, το *EP* θα ολοκληρώσει και τις δύο κατανεμημένες συναλλαγές *High* και *Low*, οδηγώντας όμως έτσι σε μία μη-σειριοποιήσημη εκτέλεση.



Σχήμα 1

Το πρωτόκολλο *SEP* που περιγράφεται στο [V.Alturi, S.Jajodia, "Degrees of Isolation, Concurency Ontrol and Commit Protocols", 1994] εφαρμόζει απελευθέρωση των κλειδαριών ανάγνωσης κατά τη διάρκεια του παραθύρου αβεβαιότητας και έτσι λύνει το παραπάνω πρόβλημα. Πιο συγκεκριμένα, στο σχήμα 1, το *SEP* ολοκληρώνει την κατανεμημένη συναλλαγή *Low* αλλά απορρίπτει την *High* αποφεύγοντας έτσι την μη-σειριοποιήσημη εκτέλεση. Το *SEP* εισάγει την *φάση επιβεβαίωσης* (*confirmation phase*) πριν από τη φάση

απόφασης του *EP*. Σε αυτή τη φάση, το *SEP* ψάχνει αν υπάρχει καμία κλειδαριά ανάγνωσης που να έχει ελευθερωθεί κατά τη διάρκεια του παραθύρου αβεβαιότητας μιας υπο-συναλλαγής. Αν ναι, η συγκεκριμένη κατανεμημένη συναλλαγή απορρίπτεται.. Η πλήρης περιγραφή του *SEP* είναι η παρακάτω:

1. Ο διαχειριστής των συναλλαγών *TM* στην αρχική τοποθεσία παράγει τις υπο-συναλλαγές $T_1, T_2, ..., T_n$, και τις στέλνει στους συμμετέχοντες $P_1, P_2, ..., P_n$, αντίστοιχα.
2. Κάθε συμμετέχων P_i εκτελεί την T_i και στέλνει στον συντονιστή *C* είτε μία ψήφο «ναι» είτε μία ψήφο «όχι». Μία ψήφος «ναι» αποστέλλεται αν ο P_i μπόρεσε επιτυχώς να ολοκληρώσει την υπο-συναλλαγή. Επιπλέον, αν ο P_i έχει διαβάσει κάποιο χαμηλού επιπέδου στοιχείο δεδομένων , στέλνει έναν χαμηλής-ανάγνωσης-δείκτη στον συντονιστή *C* μαζί με τη ψήφο του.
3. Αν ο συντονιστής *C* λάβει ψήφους «ναι» από όλους τους συμμετέχοντες, ελέγχει αν έχει λάβει κάποιον χαμηλής-ανάγνωσης-δείκτη. Αν ναι, ο *C* στέλνει σε κάθε συμμετέχων P_j που έστειλε χαμηλής-ανάγνωσης-δείκτη ένα μήνυμα επιβεβαίωσης , για να σιγουρευτεί ότι αυτοί δεν ελευθέρωσαν καμία κλειδαριά ανάγνωσης σε χαμηλού επιπέδου ασφαλείας στοιχεία δεδομένων, πριν λάβουν το μήνυμα επιβεβαίωσης.
4. Αν ο P_j δεν ελευθέρωσε καμία κλειδαριά ανάγνωσης σε χαμηλού επιπέδου ασφαλείας στοιχεία δεδομένων, πριν λάβει το μήνυμα επιβεβαίωσης., τότε απαντά στον συντονιστή με ένα μήνυμα επιβεβαίωσης. Αλλιώς, απαντά με ένα μήνυμα μη-επιβεβαίωσης.
5. Αν ο συντονιστής *C* λάβει ένα μήνυμα επιβεβαίωσης από όλους τους συμμετέχοντες P_j στους οποίους ο *C* είχε στείλει μήνυμα, τότε αυτός ολοκληρώνει τη κατανεμημένη συναλλαγή και αποστέλλει ένα μήνυμα ολοκλήρωσης σε όλους τους συμμετέχοντες. Διαφορετικά, απορρίπτει την συναλλαγή και αποστέλλει ένα μήνυμα απόρριψής σε όλους τους συμμετέχοντες.

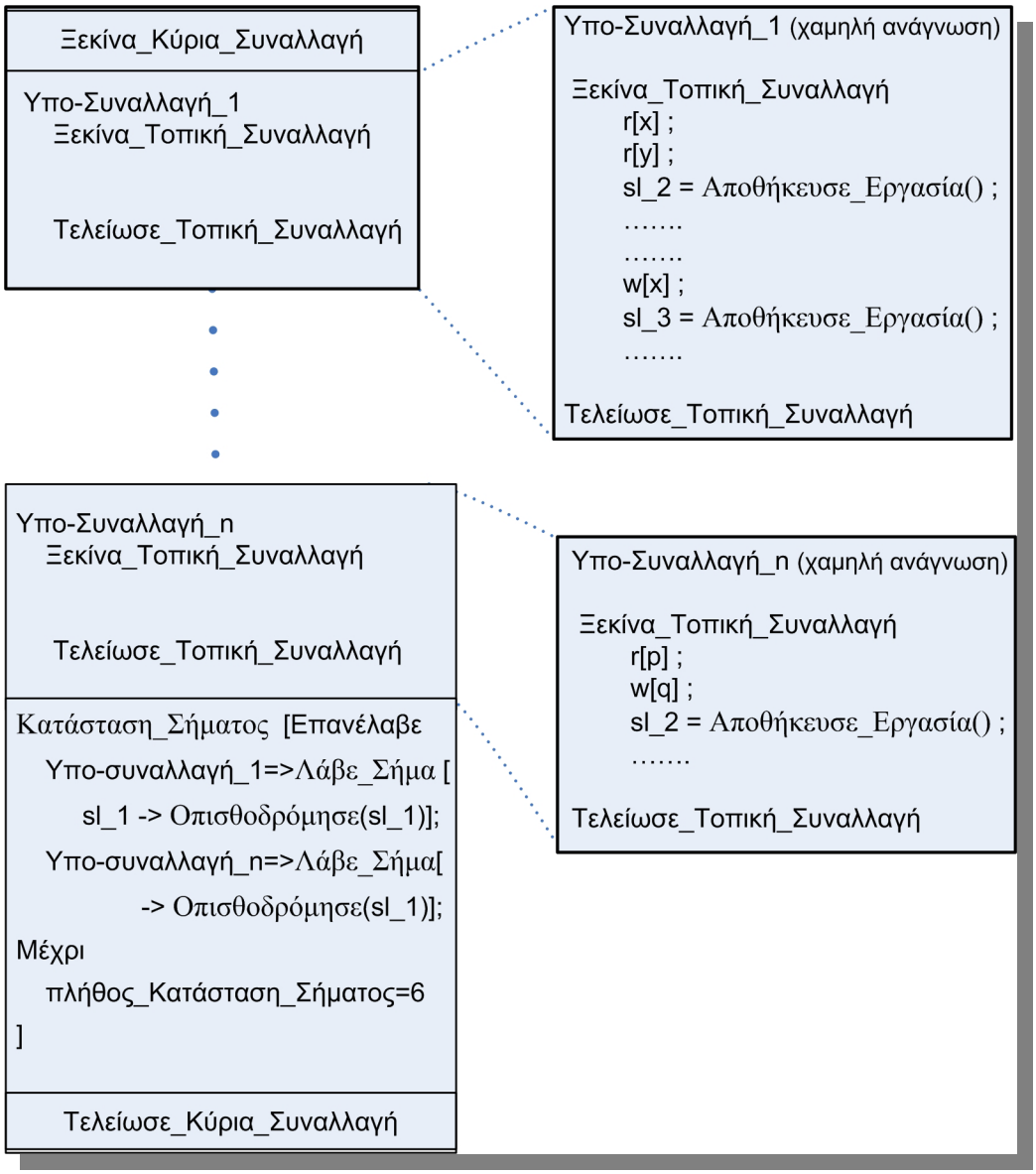
Έτσι, βλέπουμε ότι το *SEP* όποτε οποιαδήποτε από τις υπο-συναλλαγές *T* ελευθερώνει μία κλειδαριά ανάγνωσης πάνω σε χαμηλού επιπέδου ασφάλειας στοιχείο δεδομένων πριν λάβει το μήνυμα επιβεβαίωσης από τον συντονιστή, η *T* απορρίπτεται . Στη χειρότερη περίπτωση, το παραπάνω μπορεί να οδηγήσει σε λιμοκτονία συγκεκριμένων υψηλής ασφαλείας συναλλαγών από κακόβουλες χαμηλής ασφαλείας συναλλαγές. Είναι φανερό πως μία τέτοια

πολιτική, του να απορρίπτονται τόσο εύκολα συναλλαγές, είναι πολύ «συντηρητική». Παρακάτω, θα εισάγουμε την έννοια της *μερικής οπισθοδρόμησης* (*partial rollback*) με στόχο να αποτρέψουμε τις συναλλαγές από το να απορριφθούν τόσο εύκολα. Στη πράξη, στο τροποποιημένο πρωτόκολλο εκτέλεσης που θα περιγράψουμε, στέλνεται, σε κάθε τοποθεσία χαμηλού επιπέδου ανάγνωσης, οδηγίες που έχουν να κάνουν με το τι πρέπει να γίνει αν πρέπει να ελευθερωθούν χαμηλές κλειδαριές ανάγνωσης, σε αντίθεση με την απλή απόρριψη της κατανεμημένης συναλλαγής.

4. Βασικά στοιχεία του συστήματος (*System Primitives*)

Όπως έχουμε αναφέρει, περιγράφουμε μία κατανεμημένη συναλλαγή υπό τη μορφή εμφωλευμένης συναλλαγής, με ένα επίπεδο εμφώλευσης. Μία τυπική κατανεμημένη συναλλαγή του μοντέλου που περιγράφουμε φαίνεται στο σχήμα 2. Η κύρια συναλλαγή φαίνεται στα αριστερά και οι λεπτομέρειες των δύο τυπικών υπο-συναλλαγών φαίνονται στα δεξιά. Μία κατανεμημένη συναλλαγή κατατίθεται από έναν χρήστη, και περιλαμβάνεται ανάμεσα στις εντολές του συστήματος *Ξεκίνα_Κύρια_Συναλλαγή*, *Τελείωσε_Κύρια_Συναλλαγή*. Ολόκληρος ο κώδικας για την κατανεμημένη συναλλαγή βρίσκεται ανάμεσα σε αυτές τις δύο εντολές. Κάθε υπο-συναλλαγή περιλαμβάνεται ανάμεσα στις εντολές του συστήματος *Ξεκίνα_Τοπική_Συναλλαγή, Τελείωσε_Τοπική_Συναλλαγή* και αναγνωρίζεται από την ετικέτα που ακολουθεί την εντολή *Ξεκίνα_Τοπική_Συναλλαγή*.

Η εκτέλεση της εντολής *Ξεκίνα_Τοπική_Συναλλαγή_n* εκκινεί την υπο-συναλλαγή στην τοπική τοποθεσία *n*. Η κατάσταση της υπο-συναλλαγής αρχικοποιείται και τοποθετείται ένας δείκτης συναλλαγής. Η επιτυχής εκτέλεση της εντολής *Τελείωσε_Τοπική_Συναλλαγή* σηματοδοτεί την αίτηση από τον συμμετέχοντα προς τον συντονιστή να εκκινήσει το πρωτόκολλο εκτέλεσης. Για να γίνει αυτό, ο συμμετέχων στέλνει μία ψήφο «ναι» στον συντονιστή. Η υπο-συναλλαγή αναμένει στην εντολή *Τελείωσε_Τοπική_Συναλλαγή* μέχρι ο συμμετέχων να λάβει απάντηση από τον συντονιστή, υπό τη μορφή μιας οδηγίας. Όταν ο συμμετέχων λάβει μία τέτοια οδηγία, η υπο-συναλλαγή εκτελεί την οδηγία αυτή. Αν η οδηγία λέει ότι πρέπει να γίνει ολοκλήρωση ή απόρριψη, η υπο-συναλλαγή ολοκληρώνεται ή απορρίπτεται ανάλογα. Για άλλου είδους οδηγίες, όταν η εκτέλεσή τους έχει ολοκληρωθεί, ο συμμετέχων απαντά με μία ψήφο «ναι» ή «όχι», ανάλογα με την τελική κατάσταση στην οποία έχει περιέλθει η υπο-συναλλαγή.



Σχήμα 2

Για να εξειδικεύσει την λήξη μιας υπο-συναλλαγής, ο προγραμματιστής μπορεί να χρησιμοποιήσει τις εντολές *Ολοκλήρωσε_Εργασία* και *Απέρριψε_Εργασία*. Μία εντολή *Ολοκλήρωσε_Εργασία* ή *Απέρριψε_Εργασία* σε οποιοδήποτε σημείο του κώδικα της υπο-συναλλαγής έχει την παρακάτω έννοια: Αν η υπο-συναλλαγή συναντήσει την εντολή *Ολοκλήρωσε_Εργασία* προσπερνά τον υπόλοιπο κώδικα και εκτελεί την εντολή *Τελείωσε_Τοπική_Συναλλαγή*. Όπως και πριν, αυτό σηματοδοτεί την αίτηση από τον συμμετέχοντα στον συντονιστή να εκκινήσει το πρωτόκολλο εκτέλεσης, αφού πρίν του έστειλε μία ψήφο «ναι». Αν, από την άλλη, η υπο-συναλλαγή συναντήσει την εντολή *Απέρριψε_Εργασία*, απορρίπτεται μετά την αποστολή μιας ψήφου «όχι» στον συντονιστή.

Αφού ο συντονιστής έχει λάβει τις ψήφους από όλους τους συμμετέχοντες στη κατανεμημένη συναλλαγή, ο πρώτος εκκινεί το πρωτόκολλο εκτέλεσης. Σε περίπτωση που υπάρχει έστω και μία ψήφος «όχι» η κατανεμημένη συναλλαγή απορρίπτεται. Αν υπήρξαν υπο-συναλλαγές οι οποίες να διάβασαν χαμηλού επιπέδου ασφαλείας στοιχεία δεδομένων, η νέα εντολή του συστήματος *Κατάσταση_Σήματος* θέτει σε εφαρμογή το πρωτόκολλο εκτέλεσης. Η εντολή αυτή είναι μία από τις πέντε νέες εντολές που εισάγουμε και μπορούν να χρησιμοποιηθούν μόνο από τον συντονιστή. Με την εκτέλεση της *Κατάσταση_Σήματος*, ο συντονιστής στέλνει σε όλους τους συμμετέχοντες τις οδηγίες που οι υπο-συναλλαγές αναμένουν καθώς βρίσκονται στην εντολή *Τελείωσε_Τοπική_Συναλλαγή*. Μετά την ολοκλήρωση της εντολής *Κατάσταση_Σήματος*, ο συντονιστής αποστέλλει μία εντολή ολοκλήρωσης ή απόρριψης σε όλους τους συμμετέχοντες στη κατανεμημένη συναλλαγή.

Η σύνταξη των πέντε νέων εντολών είναι η εξής :

1. sl = *Αποθήκευσε_Εργασία*()
2. *Οπισθοδρόμησε*(sl)
3. *Λάβε_Σήμα* [[$\mathrm{sl}_1 \rightarrow$ Διαχειτιστής $_1$],….., [$sl_n \rightarrow$ Διαχειριστής $_n$]]
4. *Κατάσταση_Σήματος* [Επανέλαβε

   TID $_P \Rightarrow$ *Λάβε_Σήμα* […];

   ……...

   ……..

   TID $_q \Rightarrow$ *Λάβε_Σήμα* […];

   Μέχρι <συνθήκη>]
5. *Μη_Λήψη_Σήματος*

Σχετικά με την κύρια συναλλαγή, υπάρχουν τρεις μεταβλητές για χρήση από τον προγραμματιστή: Ο μετρητής *Πλήθος_Κατάσταση_Σήματος*, και οι λογικές μεταβλητές *Έτοιμο_Να_Ολοκληρωθεί* και *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαραριάς*. Όταν η κύρια συναλλαγή εκτελεί την εντολή *Ξεκίνα_Κύρια_Συναλλαγή* οι τρεις μεταβλητές αρχικοποιούνται : η *Πλήθος_Κατάσταση_Σήματος* γίνεται μηδέν, και οι *Έτοιμο_να_ολοκληρωθεί* και *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς* παίρνουν τη τιμή TRUE. Η ακριβής χρήση των τριών αυτών μεταβλητών θα γίνει κατανοητή όταν περιγράψουμε την λειτουργία *Κατάσταση_Σήματος*.

Σχετικά με την κάθε υπο-συναλλαγή, υπάρχει ένας μετρητής, ο *Μετρητής_Οπισθδρομήσεων*, ο οποίος είναι τοπικός για κάθε υπο-συναλλαγή. Όταν μία υπο-συναλλαγή εκτελεί την εντολή

Ξεκίνα_Τοπική_Συναλλαγή, αρχικοποιείται η μεταβλητή αυτή με την τιμή μηδέν. Αυτή η μεταβλητή αυξάνεται κατά ένα, κάθε φορά που η υπο-συναλλαγή οπισθοδρομεί. Η εντολή *Τελείωσε_Κύρια_Συναλλαγή* σηματοδοτεί το σημείο που λήγει η κατανεμημένη συναλλαγή, έτσι ώστε ο διαχειριστής της συναλλαγής *T* να μπορεί τώρα να πάψει να ασχολείται με την *T*.

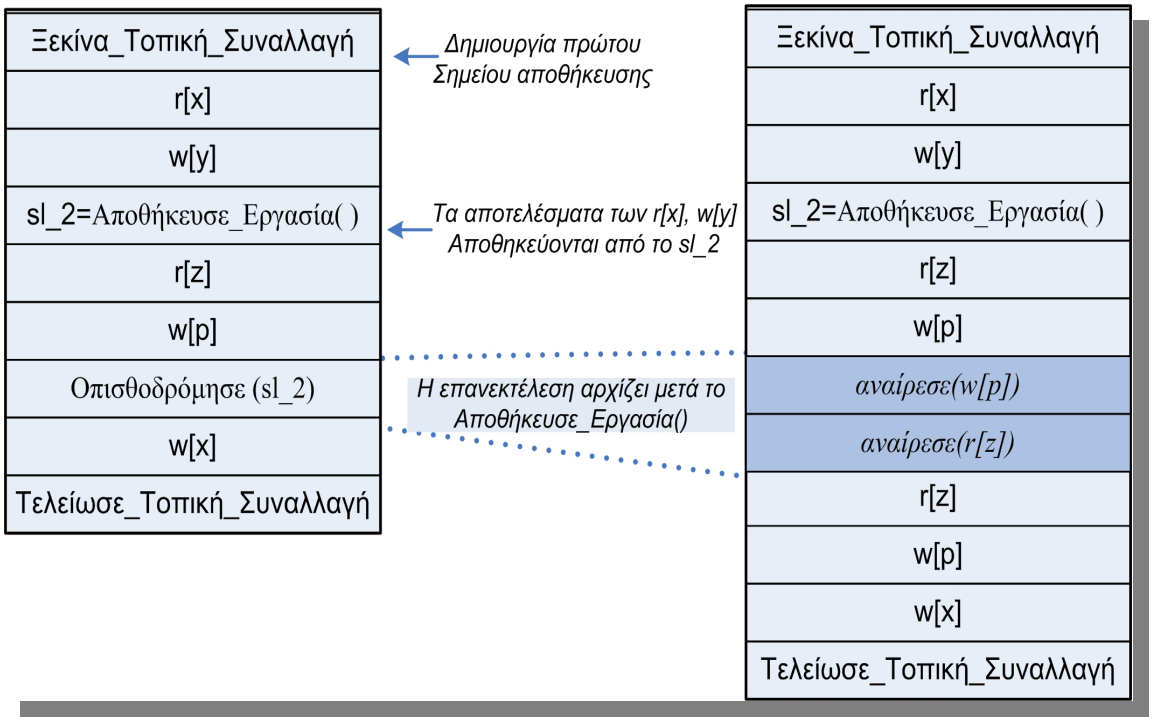
Στη συνέχεια περιγράφουμε τη σημασία των παραπάνω πέντε νέων λειτουργιών και πως αυτές μπορούν να τροποποιηθούν από τον προγραμματιστή για να καθοδηγούν κάθε συμμετέχουσα τοποθεσία, σε περίπτωση που η χαμηλές κλειδαριές ανάγνωσης πρέπει να ελευθερωθούν.

4.1 Σημασία *Αποθήκευσε_Εργασία()* και *Οπισθοδρόμησε()*

Η εντολή *Αποθήκευσε_Εργασία()* τοποθετεί ένα *σημείο αποθήκευσης* (*save point*) και οδηγεί το σύστημα να αποθηκεύσει την τρέχουσα κατάσταση της εκτέλεσης της κατανεμημένης συναλλαγής. Κάθε διαχειριστής συναλλαγών εγγράφει ένα *σημείο αποθήκευσης* στο τοπικό *ημερολόγιο* (*log*) της τοπικής συναλλαγής, ενώ ένας *διαχειριστής της διαδικασίας* (*process manager*) αποθηκεύει τις τρέχουσες τιμές των τοπικών μεταβλητών σε μία μεταβλητή (*volatile memory*) μνήμης. Η εντολή *Αποθήκευσε_Εργασία()* επιστρέφει στην υπο-συναλλαγή έναν *διαχειριστή* (*handle*) ο οποίος σχετίζεται με μία ετικέτα *sl* (*signal label*) που έτσι μπορεί να χρησιμοποιηθεί για να αναφερθούμε στο συγκεκριμένο *σημείο αποθήκευσης*. Ο διαχειριστής αυτός είναι τυπικά ένας μονοτονικά αύξοντας αριθμός. Η υπο-συναλλαγή μπορεί να επιστέψει σε οποιοδήποτε *σημείο αποθήκευσης* με χρήση της εντολής *Οπισθοδρόμησε (sl)*, περνώντας σε αυτή τη ετικέτα *sl* του *σημείου αποθήκευσης* στο οποίο επιθυμεί να οπισθοδρομήσει. Ας σημειώσουμε εδώ πως κάθε *σημείο αποθήκευσης* που δημιουργείται από μία υπο-συναλλαγή, έχει τοπική την εμβέλεια αυτής της υπο-συναλλαγής. Υποθέτουμε, επίσης, πως με την επιτυχή εκτέλεση της εντολής *Ξεκίνα_Τοπική_Συναλλαγή*, δημιουργείται το πρώτο σημείο αποθήκευσης, το οποίο θα ονομάζεται συμβολικά $sl_{default}$.

Η εντολή *Οπισθοδρόμησε (sl)* λαμβάνει ως παράμετρο την ετικέτα *sl* και επαναφέρει το σύστημα στην κατάσταση που είχε αυτό τη χρονική στιγμή που δημιουργήθηκε το *σημείο αποθήκευσης sl*. (όταν δηλαδή εκτελέστηκε η εντολή *sl=Αποθήκευσε_Εργασία( )*). Η συναλλαγή τότε επανεκκινεί την εκτέλεσή της από το σημείο μετά την εντολή *sl=Αποθήκευσε_Εργασία( )*. Με άλλα λόγια, το αποτέλεσμα της εντολής *Οπισθοδρόμησε (sl)* είναι η εκτέλεση μιας σειράς από εντολές αναίρεσης, του τύπου *αναίρεσε*(op_i) . Το αποτέλεσμα μιας εντολής *αναίρεσε*(op_i), είναι να ελευθερωθεί κάθε κλειδαριά που η

λειτουργία op_i έχει αποκτήσει πάνω σε κάποιο στοιχείο δεδομένων, και να απομακρύνει το αποτέλεσμα της op_i από το σύστημα, σαν η λειτουργία op_i να μην είχε ποτέ εκτελεστεί. Συνεπώς, τα στοιχεία δεδομένων, όπως και οι τοπικές μεταβλητές, επανέρχονται στην κατάσταση που βρίσκονταν την χρονική στιγμή του *sl*. Μόλις η «αποκατάσταση» αυτή έχει ολοκληρωθεί, η *Οπισθοδρόμησε (sl)* τερματίζεται. Τώρα πλέον η συναλλαγή είναι έτοιμη να ξεκινήσει πάλι από το σημείο που ακολουθεί την εντολή *sl=Αποθήκευσε_Εργασία( )*.



Σχήμα 3

Το σχήμα 3 απεικονίζει το πώς η εντολή *Αποθήκευσε_Εργασία( )* και *Οπισθοδρόμησε (sl)* ελέγχουν τη ροή της συναλλαγής. Η αρχική συναλλαγή βρίσκεται στα αριστερά του σχήματος. Η εντολή *Ξεκίνα_Τοπική_Συναλλαγή* δημιουργεί το πρώτο σημείο αποθήκευσης, το $sl_{default}$. Τα αποτελέσματα των λειτουργιών *r[x]* και *w[y]* αποθηκεύονται ως αποτέλεσμα της *Αποθήκευσε_Εργασία( )* με την ετικέτα sl_2. *Αφού η εντολή Οπισθοδρόμησε (..) έχει ως παράμετρο την* sl_2, η συναλλαγή εκτελεί με τη σειρά τις *αναίρεσε(w[p]) και αναίρεσε(r[z])*. Μόλις ολοκληρωθεί αυτό, συνεχίζει την κανονική της εκτέλεση με τα βήματα *r[z], w[p], w[x]* κτλ. Το αποτέλεσμα της συναλλαγής, που περιλαμβάνει τα παραπάνω βήματα, φαίνεται στα δεξιά του σχήματος.

4.2 Σημασία *Λάβε_Σήμα*

Στο πρωτόκολλό μας υποθέτουμε πώς ένας *διαχειριστής κλειδαριών* (*lock manager*) στέλνει ένα μήνυμα στον *διαχειριστή των συναλλαγών* κάθε φορά που μία χαμηλή κλειδαριά ανάγνωσης, η οποία έχει αποκτηθεί από μία υψηλού επιπέδου ασφαλείας υπο-συναλλαγή, πρέπει να ελευθερωθεί κατά τη διάρκεια του παραθύρου αβεβαιότητάς της. Η εντολή *Λάβε_Σήμα* χρησιμοποιείται για να καθορίσει πως, τα παραπάνω μηνύματα θα εξυπηρετηθούν (*serviced*) από την υψηλή υπο-συναλλαγή. Η *Λάβε_Σήμα* αποτελείται από δύο σημεία : Ένα τυπικό σημείο το οποίο είναι αμέσως μετά από την εντολή *Λάβε_Σήμα*, και δεύτερο που περιγράφεται από την έκφραση:

$$[\,sl_1 \rightarrow \text{Διαχειριστής}_{\ 1}\,], \ldots., [\,sl_n \rightarrow \text{Διαχειριστής}_{\ n}\,].$$

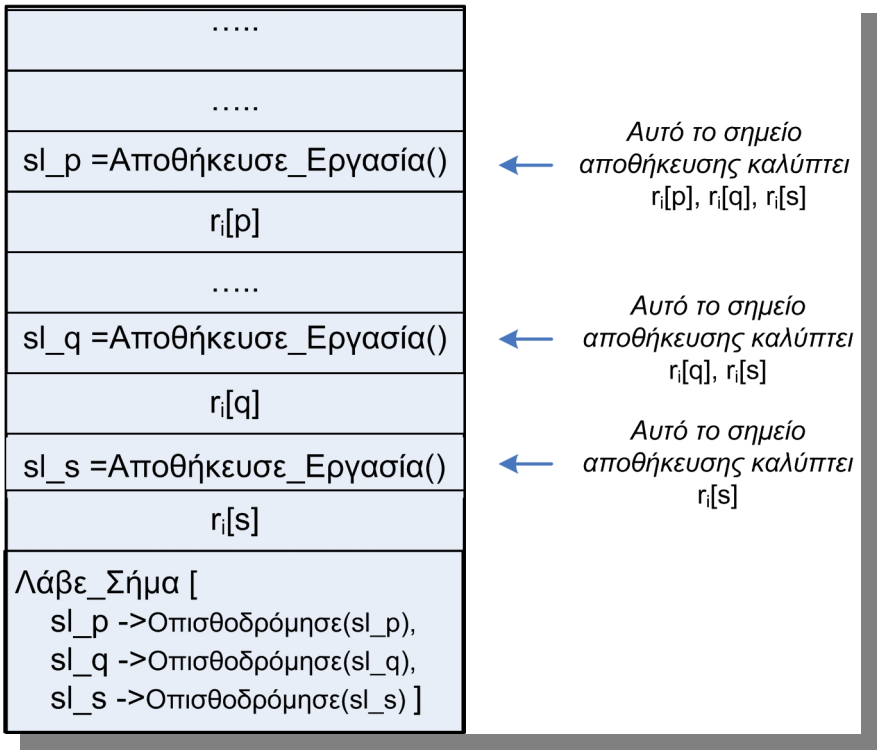
Κάθε sl_i αναπαριστά μία ετικέτα *σημείου αποθήκευσης* και ο συσχετισμένος διαχειριστής αναπαριστά τον κώδικα που πρέπει να εκτελεστεί για την συγκεκριμένη ετικέτα.

Κάθε φορά που μία χαμηλή κλειδαριά ανάγνωσης, η οποία έχει αποκτηθεί από μία υψηλού επιπέδου ασφαλείας υπο-συναλλαγή πρέπει να ελευθερωθεί, ο *διαχειριστής κλειδαριών* στέλνει ένα μήνυμα στον υψηλό *διαχειριστή συναλλαγών* για τις T_i, δείχνοντας το συγκεκριμένο στοιχείο δεδομένων. Κάθε μήνυμα που λαμβάνεται από τον διαχειριστή συναλλαγών υποδεικνύει μία νέα τιμή για ένα χαμηλού επιπέδου ασφαλείας στοιχείο δεδομένων, το οποίο έχει προηγουμένως αναγνωστεί από την T_i. Μόλις ληφθεί το μήνυμα, ο διαχειριστής συναλλαγών εντοπίζει το *σημείο αποθήκευσης* *sl* που προηγήθηκε αμέσως πριν από την ανάγνωση του στοιχείου δεδομένων που έδειξε το μήνυμα. Λέμε ότι το σημείο αποθήκευσης *sl* «*καλύπτει*» το συγκεκριμένο στοιχείο δεδομένων. Για παράδειγμα, αν το μήνυμα υποδεικνύει μία νέα τιμή για το στοιχείο δεδομένων x, τότε επιλέγεται η ετικέτα sl_i που δημιουργήθηκε από την εντολή sl_i = *Αποθήκευσε_Εργασία( )* και η οποία αναφέρεται στο τελευταίο *σημείο αποθήκευσης* πριν από την λειτουργία $r_i[x]$.

Σε περίπτωση που πολλές χαμηλές κλειδαριές ανάγνωσης «σπάσουν», ο *διαχειριστής συναλλαγών* δέχεται πολλαπλά μηνύματα, ένα για κάθε κλειδαριά, από τον *διαχειριστή κλειδαριών*. Τα μηνύματα αυτά αποθηκεύονται προσωρινά στον *διαχειριστή συναλλαγών* και όταν αυτός εκτελέσει την εντολή *Λάβε_Σήμα*, τότε επιλέγει ένα από τα μηνύματα αυτά να εξυπηρετήσει, με τον εξής τρόπο: Επιλέγεται η ετικέτα η οποία καλύπτει όλα τα στοιχεία

δεδομένων με «σπασμένες» κλειδαριές. Επιλέγεται, δηλαδή, η ετικέτα η οποία προηγήθηκε όλων των άλλων ετικετών που δημιουργήθηκαν λόγω των «σπασμένων» κλειδαριών.

Για να καταλάβουμε πως ο *διαχειριστής συναλλαγών* επιλέγει ένα μήνυμα για να εξυπηρετήσει, ας υποθέσουμε πως έχουμε το σχήμα 4, τμήμα μιας συναλλαγής. Έτσι, η υψηλού επιπέδου συναλλαγή T_i εκτελεί τρεις αναγνώσεις $r_i[p], r_i[q]$ και $r_i[s]$.Αν η χαμηλή κλειδαριά ανάγνωσης στο p «σπάσει», η ετικέτα που επιλέγεται είναι η *sl_p*, αφού αυτή καλύπτει την εντολή $r_i[p]$.



Σχήμα 4

Όμοια, αν οι κλειδαριές στα q και s «σπάσουν», ο *διαχειριστής συναλλαγών* θα επιλέξει την ετικέτα *sl_q*, η οποία καλύπτει $r_i[q]$ και $r_i[s]$. Αν και οι τρεις κλειδαριές «σπάσουν», θα επιλεγεί η ετικέτα *sl_p* , αφού αυτή καλύπτει και τα τρία στοιχεία δεδομένων.

Όταν ένα μήνυμα sl_i επιλεχτεί για να εξυπηρετηθεί, ο κώδικας για τον αντίστοιχο διαχειριστή, διαχειριστής $_i$, εκτελείται.

4.3 Σημασία *Κατάσταση_Σήματος* και *Μη_Λήψη_Σήματος*

Η λειτουργία της εντολής *Κατάσταση_Σήματος* αποτελεί την καρδιά του πρωτόκολλου εκτέλεσης που παρουσιάζουμε. Όταν ο προγραμματιστής χρησιμοποιήσει την εντολή αυτή, το σύστημα θα εκκινήσει το ασφαλές πρωτόκολλο εκτέλεσης.

Η εντολή *Κατάσταση_Σήματος* χρησιμοποιείται πάντα από την πρώτη (σε επίπεδο εμφώλευσης) συναλλαγή στην τοποθεσία που παίζει τον συντονιστικό ρόλο για την εκτέλεση της συναλλαγής αυτής. Το κυρίως μέρος της εντολής αυτής αποτελείται από έναν βρόγχο *Επανέλαβε…Μέχρι* < *συνθήκη* >, πράγμα που σημαίνει πως αυτό εκτελείται τουλάχιστον μία φορά. Μέσα στον βρόγχο, υπάρχει ένας αριθμός βημάτων υπό τη μορφή $TID_i \Rightarrow$ *Λάβε_Σήμα […]*. Κάθε ένα από τα βήματα αυτά αναπαριστά ένα τρέχον νήμα της λειτουργίας. Όταν η εντολή *Λάβε_Σήμα* εκτελεστεί, στον συντονιστή συμβαίνουν τα παρακάτω γεγονότα :

a. Για κάθε βήμα $TID_i \Rightarrow$ *Λάβε_Σήμα […]* μέσα στο σώμα του βρόγχου, ο συντονιστής εξυπηρετεί το τρέχον νήμα της λειτουργίας.

b. Κάθε ένα από αυτά τα νήματα στέλνει ένα μήνυμα στην αντίστοιχη υπο-συναλλαγή TID_i. Με αυτό το μήνυμα, ζητείται από την υπο-συναλλαγή να εκτελέσει μία εντολή *Λάβε_Σήμα*, με τον διαχειριστή που καθορίζεται από την κλήση της *Λάβε_Σήμα*. Το νήμα, τότε, περιμένει έως ότου έρθει μία απάντηση από την συγκεκριμένη υπο-συναλλαγή.

c. Μόλις λάβει το μήνυμα από τον συντονιστή, η υπο-συναλλαγή TID_i εκτελεί την εντολή *Λάβε_Σήμα* σαν να είχε κληθεί τοπικά.

    a. Αν δεν υπάρχει κανένα σήμα (*sl*) για να εξυπηρετηθεί (πράγμα που σημαίνει ότι καμία χαμηλή κλειδαριά ανάγνωσης δεν έχει «σπάσει» πρόωρα), ο συμμετέχων αποστέλλει ένα *Μη_Λήψη_Σήματος* μήνυμα στον συντονιστή και περιμένει για την απάντησή του.
    b. Αν υπάρχει σήμα (*sl*) για να εξυπηρετηθεί (πράγμα που σημαίνει ότι μία ή περισσότερες χαμηλές κλειδαριές ανάγνωσης έχουν «σπάσει» πρόωρα), η υπο-συναλλαγή εκτελεί τον κώδικα που αντιστοιχεί στον διαχειριστή ο οποίος καθορίζεται από την *Λάβε_Σήμα*.

i. Αφού η υπο-συναλλαγή εκτελέσει τον κώδικα του διαχειριστή επιτυχώς και φτάσει στην εντολή *Τελείωσε_Τοπική_Συναλλαγή*, αποστέλλει μία ψήφο «ναι» στον συντονιστή.

ii. Αν, σε αντίθετη περίπτωση, η υπο-συναλλαγή δεν μπορεί να εκτελέσει τον κώδικα του διαχειριστή με επιτυχία, αποστέλλει μία ψήφο «όχι» στον συντονιστή.

d. Το νήμα που βρίσκεται στον συντονιστή βγαίνει από την κατάσταση αναμονής όταν αυτό λάβει την απάντηση από την υπο-συναλλαγή και ανανεώνει τις τιμές των καθολικών μεταβλητών *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς* και *Έτοιμο_Να_Ολοκληρωθεί* ως εξής :

ΑΝ(*απάντηση_από_την_υποσυναλλαγή*) **ΕΙΝΑΙ** (*Μη_Λήψη_Σήματος)* **ΤΟΤΕ**

Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς = (*Έτοιμο_Να_Ολοκληρωθεί*) $\wedge$ (TRUE)

ΑΛΛΙΩΣ

Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς = FALSE

ΑΝ (*απάντηση _από_την_υποσυναλλαγή*) **ΕΙΝΑΙ** (ναι) **ΤΟΤΕ**

Έτοιμο_Να_Ολοκληρωθεί = (*Έτοιμο_Να_Ολοκληρωθεί*) $\wedge$ (TRUE)

ΑΛΛΙΩΣ *Έτοιμο_Να_Ολοκληρωθεί* = FALSE

ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ_ΑΝ

e. Όταν έχουν ληφθεί όλες οι ψήφοι, εξετάζεται πρώτα η τιμή της μεταβλητής *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς*. Αν είναι TRUE, ο συντονιστής εκτελεί-ολοκληρώνει την κατανεμημένη συναλλαγή και ενημερώνει όλους τους συμμετέχοντες για την απόφαση ολοκλήρωσης (*commit decision*). Αν η τιμή της μεταβλητής *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαριάς* είναι FALSE, εξετάζεται η τιμή της μεταβλητής *Έτοιμο_Να_Ολοκληρωθεί*. Αν είναι FALSE (σε περίπτωση, δηλαδή, όπου ένας ή περισσότεροι συμμετέχοντες ψήφισαν «όχι») ο συντονιστής απορρίπτει την κατανεμημένη συναλλαγή και ενημερώνει τους συμμετέχοντες για την απόφαση αυτή (*abort decision*). Αν η τιμή της *Έτοιμο_Να_Ολοκληρωθεί* είναι TRUE, εξετάζεται το μέρος του βρόγχου *Επανάλαβε….Μέχρι*, δηλαδή η <*συνθήκη*>. Αν η συνθήκη είναι FALSE, οι τιμές των *Έτοιμο_Να_Ολοκληρωθεί* και *Όχι_Πρόωρη_ Ελευθέρωση_Κλειδαριάς* επαναφέρονται στην αρχική τιμή TRUE, η τιμή του μετρητή

Πλήθος_Κατάσταση_Σήματος αυξάνεται κατά ένα και ο βρόγχος εκτελείται ξανά από το βήμα b, που περιγράψαμε παραπάνω. Αν, σε αντίθετη περίπτωση, η συνθήκη είναι ίση με TRUE, ο συντονιστής απορρίπτει την κατανεμημένη συναλλαγή και στέλνει το ανάλογο μήνυμα σε κάθε συμμετέχων.

Ας σημειώσουμε εδώ πως, ο προγραμματιστής μπορεί να τροποποιήσει τη μεταβλητή *Πλήθος_Κατάσταση_Σήματος* στην συνθήκη του βρόγχου επανάληψης, με σκοπό να καθορίσει το πλήθος των επαναλήψεων που αυτός θα εκτελέσει πριν η κατανεμημένη συναλλαγή απορριφθεί. Στη συνέχεια παρουσιάζουμε το προηγμένο ασφαλές πρωτόκολλο πρόωρης προετοιμασίας (*ASEP*). Το πρωτόκολλο *ASEP* συναντάται σε δύο διαφορετικές μορφές : Την βασική του μορφή η οποία εγγυάται για τη συνέπεια της ΒΔ και τη προηγμένη μορφή του η οποία εγγυάται την εκτέλεση της κατανεμημένης συναλλαγής.

Ο προγραμματιστής χρησιμοποιεί τις παραπάνω εντολές για να ελέγχει τη συμπεριφορά του αλγορίθμου του πρωτοκόλλου. Παράλληλα, οι παράμετροι που χρησιμοποιούνται στην εντολή *Κατάσταση_Σήματος* καθορίζουν την συμπεριφορά του *ASEP*. Αν η εντολή *Κατάσταση_Σήματος* περιέχει *καλώς ορισμένες κλήσεις* (η έννοια αυτή περιγράφεται παρακάτω) της *Λάβε_Σήμα,* καθώς επίσης, αν οι παράμετροι συμφωνούν με τις απαιτήσεις *μιας καλώς ορισμένης ASEP* κατανεμημένης συναλλαγής, το πρωτόκολλο *ASEP* εγγυάται την συνέπεια της ΒΔ. Σε άλλες περιπτώσεις, το πρωτόκολλο προσπαθεί να εγγυηθεί την εκτέλεση των υψηλών συναλλαγών παρά το κόστος σε συνέπεια. Για να περιγράψουμε το πρωτόκολλο, θα υποθέσουμε πως χρησιμοποιείται ένα ασφαλές πρωτόκολλο κλειδώματος για να πετύχουμε έλεγχο συνέπειας. Σύμφωνα με το πρωτόκολλο αυτό, μία υπο-συναλλαγή πρέπει να αποκτήσει μια κλειδαριά ανάλογα με την εργασία που θα κάνει πάνω σε ένα στοιχείο δεδομένων, πριν της επιτραπεί η πρόσβασή της σε αυτό. Αν μία υψηλή υπο-συναλλαγή έχει μία κλειδαριά ανάγνωσης πάνω σε ένα χαμηλό στοιχείο δεδομένων, όταν μία χαμηλή υπο-συναλλαγή επιθυμεί μία κλειδαριά ανάγνωσης πάνω στο ίδιο στοιχείο δεδομένων, η υψηλή υπο-συναλλαγή πρέπει να ελευθερώσει τη κλειδαριά ανάγνωσης που κατέχει.

5. Το πρωτόκολλο *ASEP* : Η βασική μορφή του.

Πριν περιγράψουμε το πρωτόκολλο *ASEP* πρέπει να δώσουμε κάποιους ορισμούς.

[Ορισμός 5.1]

Ας υποθέσουμε πως T είναι μία κατανεμημένη συναλλαγή, η οποία αποτελείται από τις υπο-συναλλαγές $T_1, T_2, ..., T_n$. Αν η T περιέχει μία κλήση της εντολής *Λάβε_Σήμα* στη παρακάτω μορφή

$$T_i \Rightarrow \textit{Λάβε\_Σήμα}\ [\ [\ sl_p \rightarrow \text{Διαχειριστής}_{\ p}]\ldots.[\ sl_q \rightarrow \text{Διαχειριστής}_{\ q}\ ]\ ],$$

με T_i να είναι η χαμηλότερη υπο-συναλλαγή ανάγνωσης, τότε η εντολή *Λάβε_Σήμα* είναι *καλώς ορισμένη* αν :

1. Για κάθε χαμηλή λειτουργία ανάγνωσης $r_i[x]$ που εκτελείται από την T_i, υπάρχει ένα $[\ sl_x \rightarrow \text{Διαχειριστής}_{\ \chi}\ ]$ μέσα στην κλήση της *Λάβε_Σήμα* τέτοιο ώστε η ετικέτα sl_x να «καλύπτει» την χαμηλή ανάγνωση $r_i[x]$ και

2. Ο Διαχειριστής , Διαχειριστής $_{\chi}$, είναι είτε η εντολή *Οπισθοδρόμησε*(sl_x) είτε μία εντολή *Απέριψε_Εργασία* .

Για να προγραμματιστεί μία *καλώς ορισμένη Λάβε_Σήμα* κλήση για την υπο-συναλλαγή, ο προγραμματιστής πρέπει να ενσωματώσει τον απαιτούμενο διαχειριστή για κάθε χαμηλή λειτουργία ανάγνωσης της υπο-συναλλαγής (μιας και κάθε χαμηλή λειτουργία ανάγνωσης μπορεί να σηματοδοτηθεί με μία ετικέτα). Επιπλέον, ο προγραμματιστής πρέπει να εγγυηθεί πως κάθε χαμηλή λειτουργία ανάγνωσης της υπο-συναλλαγής καλύπτεται από ένα *σημείο αποθήκευσης*. Ας θυμηθούμε εδώ, πως το πρώτο *σημείο αποθήκευσης* δημιουργείται με την εκτέλεση της εντολής *Ξεκίνησε_Τοπική_Συναλλαγή*, το οποίο καλύπτει κάθε χαμηλή λειτουργία ανάγνωσης της υπο-συναλλαγής. Επομένως, ένας απλός τρόπος για να εγγυηθούμε μία *καλώς ορισμένη Λάβε_Σήμα* κλήση είναι να έχουμε μία οπισθοδρόμηση στο παραπάνω *σημείο αποθήκευσης* άσχετα με το συγκεκριμένο σήμα-ετικέτα. Αυτό επιτυγχάνεται με την ακόλουθη κλήση :

- *Λάβε_Σήμα* $[\rightarrow$ *Οπισθοδρόμησε*$(sl_{default})\ ]$

Το πλεονέκτημα της παραπάνω μεθόδου είναι πως ο προγραμματιστής δεν χρειάζεται να καθορήσει το κάθε ένα *σημείο αποθήκευσης* στο σώμα της υπο-συναλλαγής. Παρ'όλ'αυτά, το

μειονέκτημα είναι πως η υπο-συναλλαγή θα οπισθοδρομείται πάντα ολοκληρωτικά. Με το να δημιουργηθούν περισσότερα από ένα *σημεία αποθήκευσης*, ίσως μειωθεί το κόστος των οπισθοδρομήσεων. Ο πιο απλός τρόπος για να εγγυηθείς μία *καλώς ορισμένη Λάβε_Σήμα*, εκμεταλλευόμενος τη χρήση των πολλαπλών *σημείων αποθήκευσης* μέσα στην υπο-συναλλαγή, είναι να χρησιμοποιηθεί η εντολή της μορφής $Λάβε\_Σήμα[\rightarrow Οπισθοδρόμησε()]$

Ένας τρίτος τρόπος με τον οποίο ο προγραμματιστής μπορεί να εγγυηθεί μία *καλώς ορισμένη* εντολή *Λάβε_Σήμα* είναι ο εξής : Ας υποθέσουμε πως η υπο-συναλλαγή T_i περιέχει τις χαμηλές εντολές ανάγνωσης $r_i[x], r_i[y], r_i[z]$ (κατά αυτή τη σειρά εμφάνισης).

```
Ti :
    Ξεκίνα_Τοπική_Συναλλαγή
    ….
    ….
    Sl_x = Αποθήκευσε_Εργασία( )
    ri[x]              /* χαμηλή ανάγνωση */
    ….
    ….
    Sl_y = Αποθήκευσε_Εργασία( )
    ri[y]              /* δεύτερη χαμηλή ανάγνωση */
    ….
    Sl_z = Αποθήκευσε_Εργασία( )
    ri[z]              /* τελευταία χαμηλή ανάγνωση */
    ….
    Τελείωσε_Τοπική_Συναλλαγή

Κατάσταση_Σήματος [ Επανέλαβε
    ….
    Ti ⇒Λάβε_Σήμα [ Sl_x -> Οπισθοδρόμησε(Sl_x)
                    Sl_y -> Οπισθοδρόμησε(Sl_y)
                    Sl_z -> Οπισθοδρόμησε(Sl_z) ]
    ….
    ….
                      Μέχρι < συνθήκη > ]
```

Σχήμα 5

Ο προγραμματιστής μπορεί, πριν από κάθε μία από αυτές τις εντολές, να εκτελέσει μία $sl = Αποθήκευσε\_Εργασία$ εντολή, και να ορίσει διαχειριστές για κάθε μία από τις ετικέτες sl_x, sl_y, sl_z μέσα στο σώμα της εντολής Κατάσταση_Σήματος, όπως φαίνεται στο σχήμα 5. Σε σχέση με το σήμα-ετικέτα που πρέπει να εξυπηρετηθεί, η υπο-συναλλαγή μπορεί να οπισθοδρομήσει σε διαφορετικά *σημεία αποθήκευσης*. Αυτό επιτρέπει στον προγραμματιστή να επιλέξει πόσα σημεία αποθήκευσης θα χρησιμοποιήσει παρά το κόστος της δημιουργίας αυτών και των εκτελέσιμων οπισθοδρομήσεων.

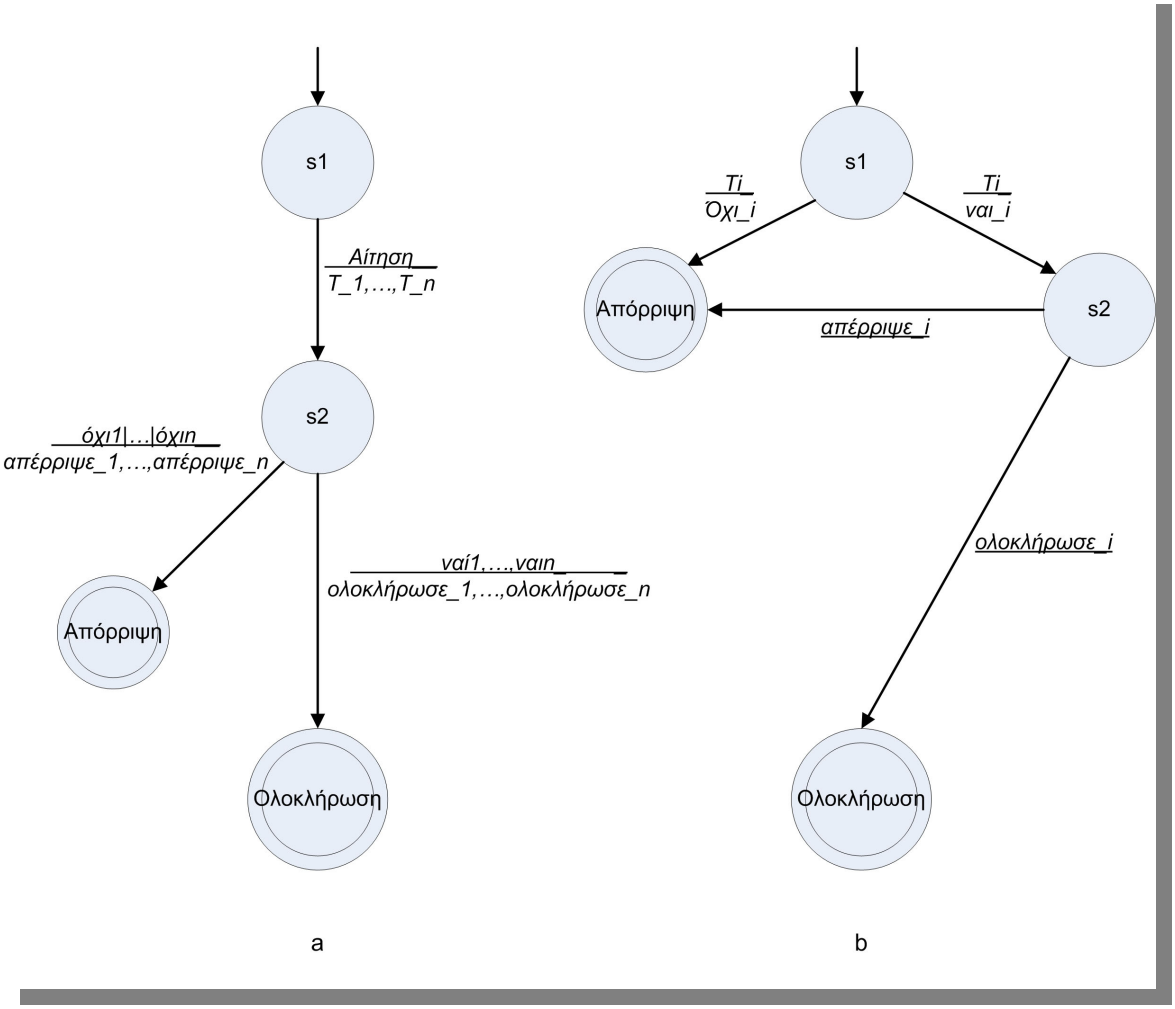
[Ορισμός 5.2]

Μία κατανεμημένη συναλλαγή πολλών επιπέδων ασφαλείας, T ,είναι μία *καλώς ορισμένη ASEP* συναλλαγή αν ικανοποιεί τις παρακάτω συνθήκες :

1. Κάθε υπο-συναλλαγή T_i είναι *καλώς ορισμένη*, σύμφωνα με το ασφαλές πρωτόκολλο συνέπειας που χρησιμοποιείται .

2. Κάθε υπο-συναλλαγή χαμηλής ανάγνωσης T_i, αντιπροσωπεύεται στο σώμα της εντολής *Κατάσταση_Σήματος* και όλες οι κλήσεις *Λάβε_Σήμα* είναι *καλώς ορισμένες*.

Για να περιγράψουμε όλες τις λεπτομέρειες του πρωτοκόλλου *ASEP*, παρουσιάζουμε την εκτέλεση του πρωτοκόλλου σε κάθε τοποθεσία υπό τη μορφή αυτομάτων. Ο συντονιστής ή ο κάθε συμμετέχων μετακινείται από μία κατάσταση του αυτομάτου σε μία άλλη στέλνοντας ή λαμβάνοντας μηνύματα μέσω του δικτύου που υπάρχει και συνδέει τις τοποθεσίες. Για να περιγράψουμε τον συμβολισμό, παρουσιάζουμε αρχικά το κλασσικό *Πρωτόκολλο Πρόωρης Προετοιμασίας* ως ένα αυτόματο. Στο σχήμα 6a αναπαρίσταται ο συντονιστής ενώ στο σχήμα 6b ένας συμμετέχων.



Σχήμα 6

Οι κύκλοι στο διάγραμμα αναπαριστούν τις ενδιάμεσες καταστάσεις, ενώ τα δαχτυλίδια τις τελικές καταστάσεις. Οι ετικέτες στις ακμές των αυτομάτων αναπαριστούν τα μηνύματα τα οποία ανταλλάσσονται και χρησιμοποιούνται για την μετάβαση από την μία κατάσταση σε μία άλλη. Η σημασία των ετικετών αυτών παρουσιάζεται στον παρακάτω πίνακα (πίνακας 1).

Σύμβολο:	Διερμηνεύεται ως:
Αίτηση	Αίτηση υπο-συναλλαγής στον συντονιστή
Μνμ_i	Μήνυμα από/προς τον συμμετέχων P_i
Τ_i	Αίτηση εργασίας και ψήφου προς τον συμμετέχων P_i
όχι_i	Ένα μήνυμα «όχι» από τον συμμετέχων P_i
Ναι_i	Ένα μήνυμα «ναι» από τον συμμετέχων P_i
Απέρριψε_i	Ένα μήνυμα απόρριψης προς τον συμμετέχων P_i
Ολοκλήρωσε_i	Ένα μήνυμα ολοκλήρωσης προς τον συμμετέχων P_i
Μνμ_i | … | Μνμ_n	Τουλάχιστον ένα μήνυμα ελήφθη από τους συμμετέχοντες [1,…,n]
Μνμ_1, … , Μνμ_n	Όλα τα μηνύματα που ελήφθησαν από τους συμμετέχοντες [1,…,n]
ΜΛΣ_i	Ένα μήνυμα Μη_Λήψης_Σήματος από τον P_i
Μνμ_ΛΣ_i	Ένα μήνυμα που περιέχει εντολή Λάβε\-Σήμα στον P_i

Πίνακας 1

Για να γίνουν περισσότερο κατανοητά τα παραπάνω, ας εξετάσουμε το αυτόματο του σχήματος 6a. Όταν ο συντονιστής βρίσκεται στη αρχική κατάσταση *s1*, περιμένει να λάβει μία αίτηση από κάποιον χρήστη για να εκτελέσει μία κατανεμημένη συναλλαγή. Αυτό αναπαρίσταται από ένα «μήνυμα αίτησης» στη ετικέτα η οποία βρίσκεται στην ακμή $s1 \rightarrow s2$. Η λήψη της αίτησης αυτής προκαλεί την μετακίνηση του συντονιστή στην κατάσταση *s2*. Ο συντονιστής με τη σειρά του στέλνει τα μηνύματα $T_1,...,T_n$, τα οποία υπονοούν τις υπο-συναλλαγές οι οποίες είναι κατανεμημένες στους διάφορους συμμετέχοντες, και μαζί αποστέλλει και τις αιτήσεις για να ψηφίσουν οι τελευταίοι.

Από την κατάσταση *s2*, ο συντονιστής μπορεί να μετακινηθεί είτε στην κατάσταση *Απόρριψη* είτε στην κατάσταση *Ολοκλήρωση*, αν λάβει έστω και μία ψήφο *«όχι»* ή αν όλες οι ψήφοι που λαμβάνει είναι *«ναι»*, αντίστοιχα. Αν ο συντονιστής μετακινηθεί στην κατάσταση *Απόρριψη*, αποστείλει *μηνύματα απόρριψης* σε όλους τους συμμετέχοντες, ειδάλλως αποστείλει *μηνύματα ολοκλήρωσης*.

Ένας όμοιος συμβολισμός στο σχήμα 6b περιγράφει τις αντίστοιχες καταστάσεις των συμμετεχόντων. Το αυτόματο του σχήματος 7a παρουσιάζει την εκτέλεση του συντονιστή στη βασική μορφή του πρωτοκόλλου *ASEP*, Το σχήμα 7b παρουσιάζει έναν συμμετέχων χαμηλής ανάγνωσης, ενώ το σχήμα 7c παρουσιάζει έναν συμμετέχων μη χαμηλής ανάγνωσης. Στη συνέχεια περιγράφουμε το πρωτόκολλο με τη βοήθεια αυτών των αυτομάτων.

[Αλγόριθμος 5.1 *ASEP* Η βασική μορφή του πρωτοκόλλου]

Ένας χρήστης ο οποίος έχει δηλωθεί στο επίπεδο ασφαλείας s εκκινεί, μία *καλώς ορισμένη ASEP*, κατανεμημένη συναλλαγή πολλών επιπέδων ασφαλείας T σε μία συγκεκριμένη τοποθεσία. Το επίπεδο ασφαλείας της συναλλαγής αυτής γίνεται s. Μία διαδικασία C, η οποία θα παίξει τον ρόλο του συντονιστή για τη συναλλαγή T, δημιουργείται στη συγκεκριμένη τοποθεσία.

1. Συντονιστής : *s1*$\rightarrow$*s2*. Ο συντονιστής C παράγει τις υπο-συναλλαγές $T_1, T_2, ..., T_n$. Το επίπεδο ασφαλείας της κάθε υπο-συναλλαγής T_i γίνεται ίσο με το επίπεδο της T, δηλαδή ίσο με s. Ο συντονιστής δημιουργεί ένα *ημερολόγιο μελών* (*membership log*) και το προωθεί στους συμμετέχοντες $P_1, P_2, ..., P_n$, μαζί με την αίτησή του για τις ψήφους τους. Αυτό αναπαρίσταται από την μετάβαση του συντονιστή από την κατάσταση *s1* στη κατάσταση *s2*, στο σχήμα 7a.

2. Συμμετέχων: *s1*$\rightarrow$*s2, s1*$\rightarrow$*Απόρριψη*. Ένας συμμετέχων P_i επιπέδου ασφαλείας s, ο οποίος βρίσκεται αρχικά στην κατάσταση *s1* στα σχήματα 7b και7c, λαμβάνει μία υπο-συναλλαγή T_i. Ο P_i προχωράει στο να αποκτήσει τις απαραίτητες κλειδαριές και μετά εκτελεί την υπο-συναλλαγή. Αν η υπο-συναλλαγή είναι ικανή να εκτελεστεί επιτυχώς, αποστέλλει μία ψήφο «ναι» στον συντονιστή C. Σε αυτό το σημείο, ο συμμετέχων μετακινείται στην κατάσταση *s2*. Αν, σε αντίθετη περίπτωση, η υπο-συναλλαγή δεν μπορεί να εκτελεστεί επιτυχώς, απορρίπτεται. Σε αυτή τη περίπτωση, ο συμμετέχων P_i αποστέλλει μία ψήφο «όχι» στον συντονιστή και μετακινείται στην τελική κατάσταση *Απόρριψη*.

3. Συντονιστής: *s2*$\rightarrow$ *Απόρριψη, s2*$\rightarrow$Ολοκλήρωση, *s2*$\rightarrow$ *s3***. Όταν ο συντονιστής C έχει, λάβει από όλους τους συμμετέχοντες μία ψήφο «ναι» είτε έχει τουλάχιστον μία ψήφο «όχι» από οποιονδήποτε συμμετέχων (υποθέτουμε πως αν λείπει κάποια ψήφος από έναν ή

περισσότερους συμμετέχοντες, αυτή θεωρείται από τον συντονιστή ως ψήφος «όχι»), αυτός εκτελεί ένα από τα παρακάτω βήματα :

a. Αν υπάρχει έστω και μία ψήφος «όχι», ο συντονιστής απορρίπτει την κατανεμημένη συναλλαγή και αποστέλλει ένα *μήνυμα απόρριψης* σε κάθε συμμετέχων. Έτσι, γίνεται η μετάβαση $s2 \rightarrow$ *Απόρριψη*.

b. Αν όλες οι ψήφοι που έλαβε ο συντονιστής είναι ψήφοι «ναι», αυτός ελέγχει αν υπήρξε κάποια υπο-συναλλαγή η οποία να διάβασε χαμηλού επιπέδου στοιχεία δεδομένων. Αν δεν υπάρχει καμία τέτοια υπο-συναλλαγή, ο συντονιστής ολοκληρώνει την κατανεμημένη συναλλαγή T και αποστέλλει *μηνύματα ολοκλήρωσης* προς όλους τους συμμετέχοντες .Έτσι, ο συντονιστής εκτελεί την μετάβαση $s2 \rightarrow$ Ολοκλήρωση στο σχήμα 7a.

c. Αν όλες οι ψήφοι είναι «ναι» και υπάρχει τουλάχιστον μία υπο-συναλλαγή η οποία διάβασε χαμηλού επιπέδου στοιχεία δεδομένων, ο συντονιστής εκτελεί μία εντολή *Κατάσταση_Σήματος*, η οποία περιλαμβάνει μία κλήση *Λάβε_Σήμα[…]* για κάθε συμμετέχων P_i που διάβασε χαμηλά στοιχεία δεδομένων. Για τους υπόλοιπους συμμετέχοντες ο συντονιστής δεν αποστέλλει κανένα μήνυμα αυτή τη στιγμή. Η μετάβαση $s2 \rightarrow s3$ αναπαριστά την κατάσταση του συντονιστή στο σημείο αυτό.

4. Συμμετέχων: $s2 \rightarrow$ *Απόρριψη*, $s2 \rightarrow$ Ολοκλήρωση, $s2 \rightarrow s2$, $s2 \rightarrow s3$. Ανάλογα με το μήνυμα που έλαβε ο συμμετέχων από τον συντονιστή, ο πρώτος εκτελεί τα παρακάτω βήματα:

a. Αν ο συμμετέχων λάβει ένα *μήνυμα απόρριψης*, απορρίπτει την υπο-συναλλαγή. Αυτό αναπαρίσταται από την μετάβαση $s2 \rightarrow$ *Απόρριψη* στα σχήματα 7b και 7c.

b. Αν ο συμμετέχων λάβει ένα *μήνυμα ολοκλήρωσης*, ολοκληρώνει την υπο-συναλλαγή. Αυτό αναπαρίσταται από την μετάβαση $s2 \rightarrow$ *Ολοκλήρωση* στα σχήματα 7b και 7c.

c. Ένας συμμετέχων ο οποίος διάβασε χαμηλού επιπέδου ασφαλείας στοιχεία δεδομένων και λαμβάνει μία κλήση *Λάβε_Σήμα […]* , εκτελεί την κλήση και, ανάλογα με το αποτέλεσμα, αποστείλει ένα από τα τρία παρακάτω μηνύματα στον συντονιστή:

i. Μία ψήφο «όχι»,αν ο διαχειριστής είναι εντολή *Απέρριψε_Εργασία* ή αντο αποτέλεσμα της εκτέλεσης του μηνύματος-ετικέτας ήταν ανεπιτυχές. Σε αυτή τη περίπτωση, ο συμμετέχων απορρίπτει την υπο-συναλλαγή και μετακινείται κατά *s2* → *Απόρριψη*.

ii. *Μη_Λήψη_Σήματος*, αν η υπο-συναλλαγή δεν ελευθέρωσε καμία χαμηλή κλειδαριά ανάγνωσης πρόωρα και συνεπώς δεν εκτέλεσε κανένα σήμα-ετικέτα. Ο συμμετέχων τότε παραμένει στην κατάσταση *s2* → *s2*.

iii. Μία ψήφο «ναι», αν η υπο-συναλλαγή ελευθέρωσε κάποια κλειδαριά χαμηλής ανάγνωσης, ο διαχειριστής δεν ήταν εντολή *Απέρριψε_Εργασία* και αφού η υπο-συναλλαγή εκτέλεσε επιτυχώς τον διαχειριστή, κατέληξε στην εντολή *Τελείωσε_Τοπική_Συναλλαγή*. Σε αυτή τη περίπτωση, ο χαμηλής ανάγνωσης συμμετέχων μετακινείται κατά *s2* → *s3*.

d. Αν ο συμμετέχων δεν ήταν χαμηλής ανάγνωσης, δεν λαμβάνει κανένα μήνυμα από τον συντονιστή, και συνεπώς δεν αλλάζει η κατάστασή του και παραμένει στην κατάσταση *s2*.

5. Συντονιστής : *s3* →*Απόρριψη, s3* →*Ολοκλήρωση, s3* → *s3*. Μετά τη λήψη των παραπάνω μηνυμάτων, ο συντονιστής εκτελεί τα παρακάτω βήματα :

a. Αν ο συντονιστής λάβει έστω και μία ψήφο «όχι» από τους συμμετέχοντες, απορρίπτει την κατανεμημένη συναλλαγή. Αποστείλει σε όλους τους συμμετέχοντες ένα μήνυμα απόρριψης και μετακινείται κατά *s3* →*Απόρριψη*.

b. Αν ο συντονιστής λάβει μηνύματα *Μη_Λήψη_Σήματος* από όλους τους συμμετέχοντες χαμηλής ανάγνωσης, ολοκληρώνει την κατανεμημένη συναλλαγή και αποστέλλει *μηνύματα ολοκλήρωσης* σε όλους τους συμμετέχοντες. Έτσι, ο συντονιστής εκτελεί την μετάβαση *s3* →*Ολοκλήρωση*.

c. Σε κάθε άλλη περίπτωση, η οποία περιλαμβάνει μία ή περισσότερες ψήφους «ναι» και τα υπόλοιπα μηνύματα *Μη_Λήψη_Σήματος*, ο συντονιστής ελέγχει τη συνθήκη στο σώμα του βρόγχου *Επανέλαβε….Μέχρι <συνθήκη>*, στην εντολή *Κατάσταση_Σήματος* και εκτελεί ένα από τα παρακάτω βήματα :

i. Αν η συνθήκη είναι FALSE, ο συντονιστής εκτελεί ξανά το σώμα της εντολής *Κατάσταση_Σήματος*. Αυτό έχει ως αποτέλεσμα ο συντονιστής να στείλει μία κλήση της *Λάβε_Σήμα* σε κάθε ένα από τους συμμετέχοντες οι οποίοι διάβασαν χαμηλού επιπέδου στοιχεία δεδομένων. Έτσι ο συντονιστής παραμένει στην κατάσταση *s3*.

ii. Αν η συνθήκη είναι TRUE, ο συντονιστής απορρίπτει την κατανεμημένη συναλλαγή και αποστέλλει *μηνύματα απόρριψης* σε όλους τους συμμετέχοντες. Έτσι, ο συντονιστής εκτελεί τη μετάβαση *s3* $\rightarrow$*Απόρριψη*.

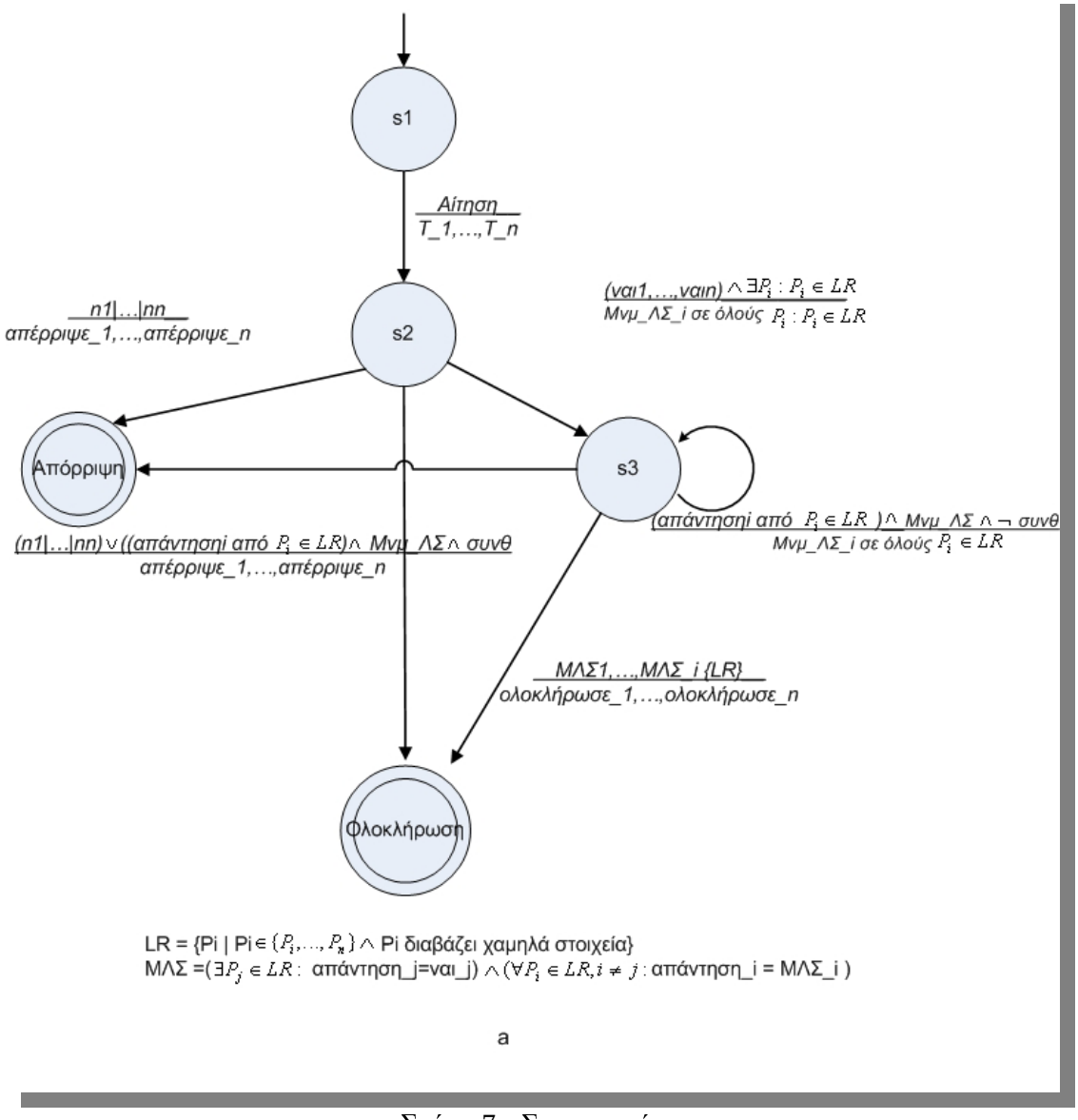
6. Συμμετέχων : *s3* $\rightarrow$*Απόρριψη, s3* $\rightarrow$ *s2, , s3* $\rightarrow$ *s3.* Αν ο συντονιστής εκτελέσει ξανά την εντολή *Κατάσταση_Σήματος*, οι χαμηλής ανάγνωσης συμμετέχοντες εκτελούν ξανά την εντολή *Λάβε_Σήμα*. Σε αυτό το σημείο, ο συμμετέχων μπορεί να βρίσκεται είτε στην κατάσταση *s2* είτε στην κατάσταση *s3*. Αν βρίσκεται στην κατάσταση *s2*, το παραπάνω βήμα 4 εκτελείται ξανά. Διαφορετικά, αν ο συμμετέχων βρίσκεται στην κατάσταση *s3*, τότε ανάλογα από το αποτέλεσμα της εκτέλεσης της εντολής *Λάβε_Σήμα*, ο συμμετέχων αποστείλει ένα από τα παρακάτω μηνύματα προς τον συντονιστή :

a. Μία ψήφο «όχι». Ο συμμετέχων απορρίπτει την υπο-συναλλαγή και μετακινείται κατά *s3* $\rightarrow$*Απόρριψη*.

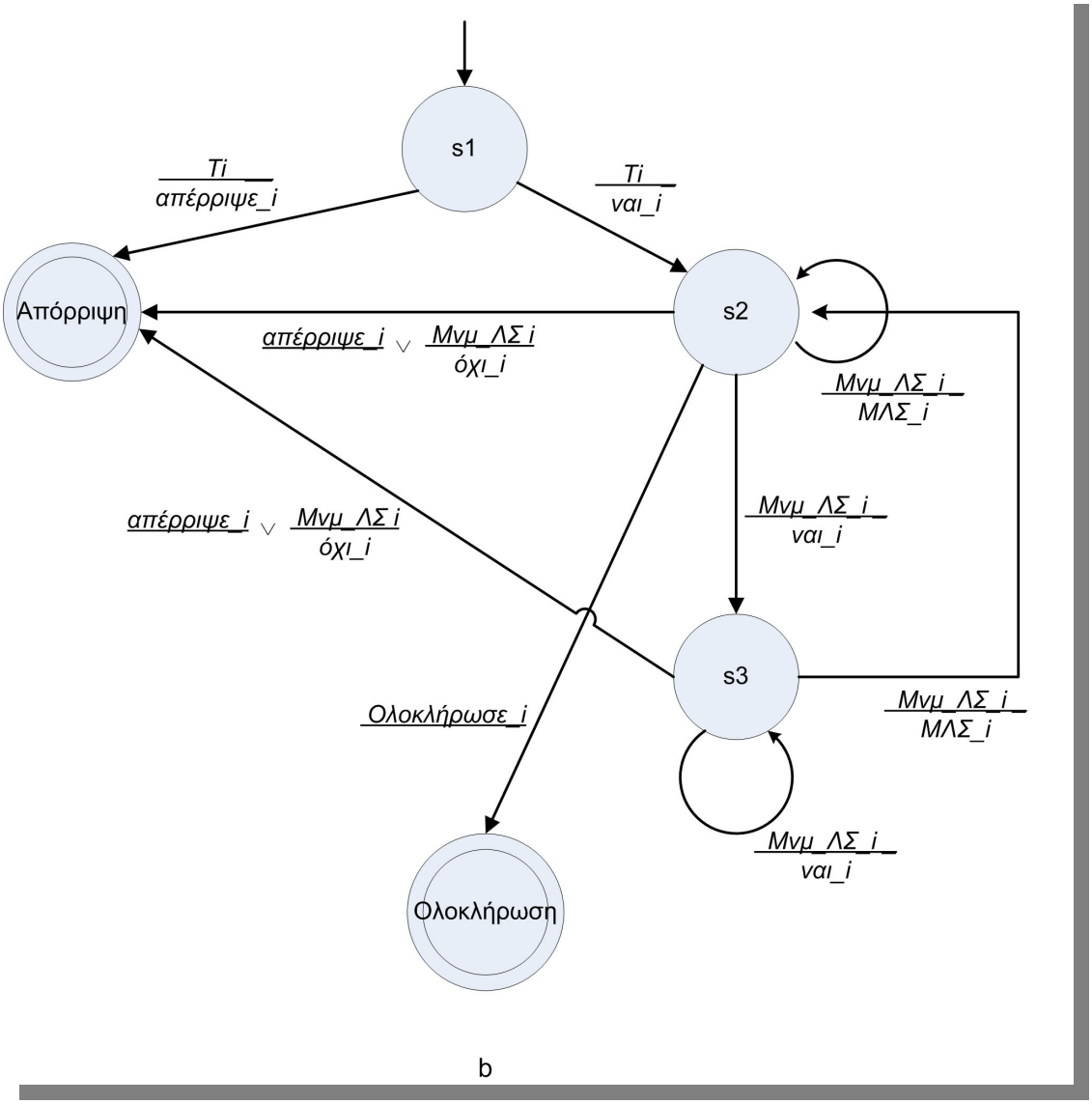
b. Μη_Λήψη_Σήματος. Ο συμμετέχων μετακινείται κατά *s3* $\rightarrow$ *s2*.

c. Μία ψήφο «ναι». Ο συμμετέχων παραμένει στην κατάσταση *s3*.

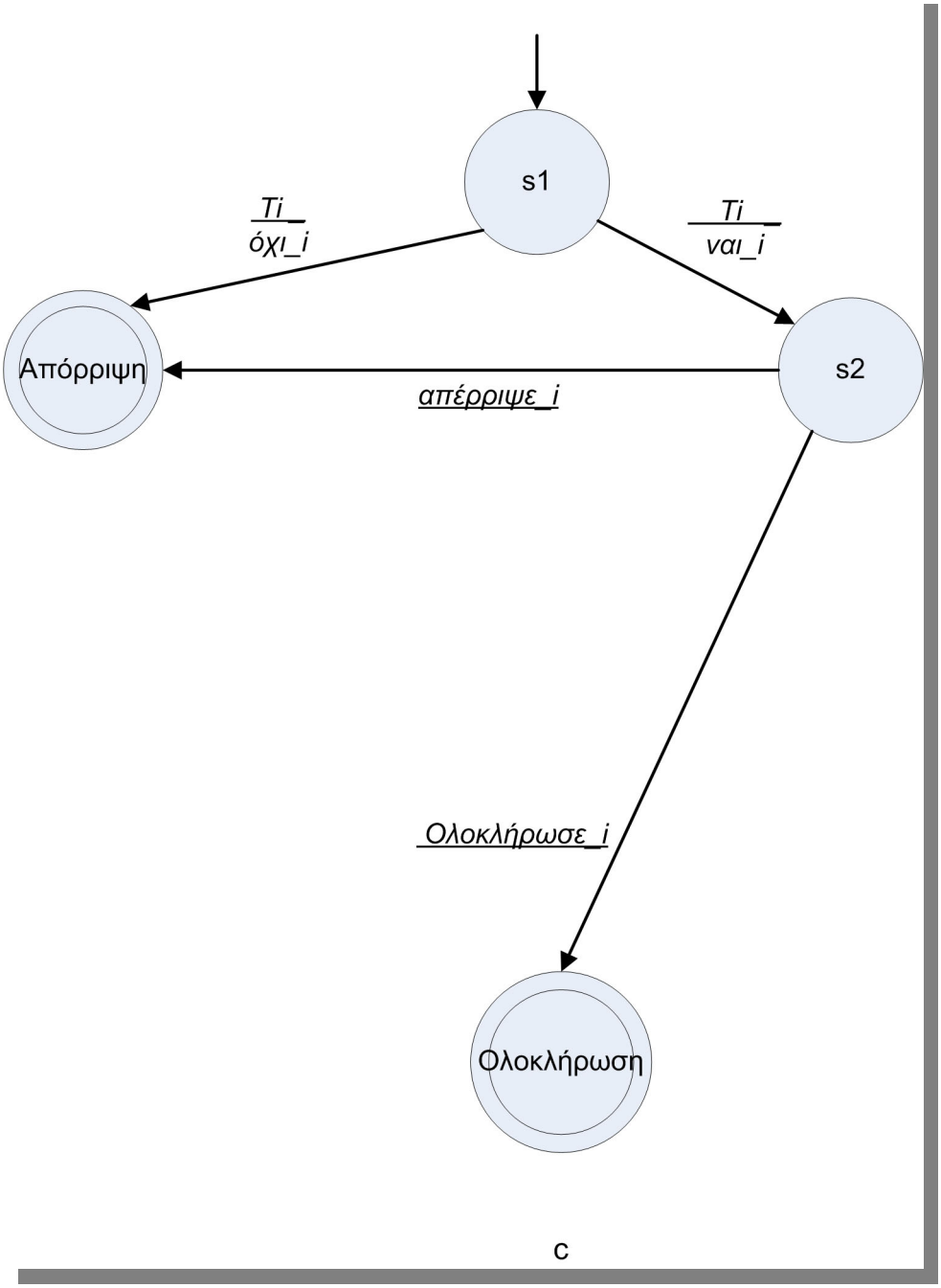
7. Το πρωτόκολλο συνεχίζει την εκτέλεση από το παραπάνω βήμα 5



Σχήμα 7a, Συντονιστής



Σχήμα 7b, Συμμετέχων χαμηλής ανάγνωσης



Σχήμα 7c, Συμμετέχων μη-χαμηλής ανάγνωσης

5.2 Εξασφαλίζοντας Συνέπεια με τη χρήση της προηγμένης μορφής του πρωτοκόλλου *ASEP*

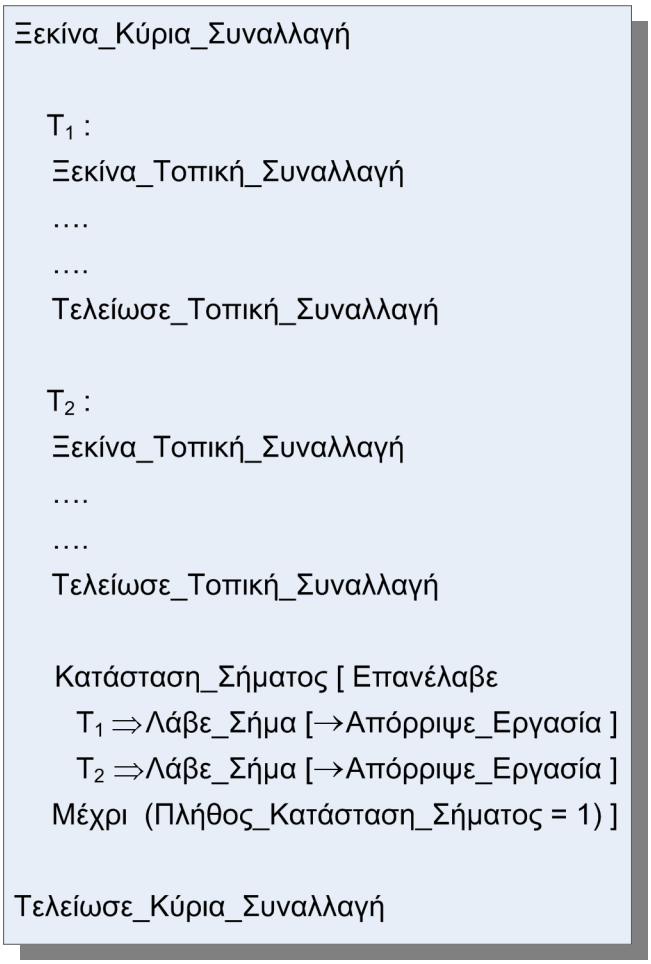
Θα παρουσιάσουμε εδώ τρία σημεία της εντολής *Κατάσταση_Σήματος* με τα οποία η βασική μορφή του πρωτοκόλλου *ASEP* εγγυάται την συνέπεια:

1. Ο βρόγχος της *Επανέλαβε…Μέχρι*, στην εντολή *Κατάσταση_Σήματος*, εκτελείται τουλάχιστον μία φορά. Αν όλοι οι χαμηλής ανάγνωσης συμμετέχοντες στείλουν ένα μήνυμα *Μη_Λήψη_Σήματος* στον συντονιστή, τότε η κατανεμημένη συναλλαγή ολοκληρώνεται, διαφορετικά απορρίπτεται (Σημειώνουμε ότι στην περίπτωση αυτή το *ASEP* ανάγεται στο *EP* πρωτόκολλο).

2. Ο βρόγχος της *Επανέλαβε…Μέχρι*, στην εντολή *Κατάσταση_Σήματος*, εκτελείται ξανά και ξανά με κάθε φορά τους χαμηλής ανάγνωσης συμμετέχοντες να οπισθοδρομούν όποτε εξυπηρετούν ένα μήνυμα-ετικέτα, και εκτελείται ξανά μέχρι όλοι από τους προηγούμενους να απαντήσουν με μήνυμα *Μη_Λήψη_Σήματος*.

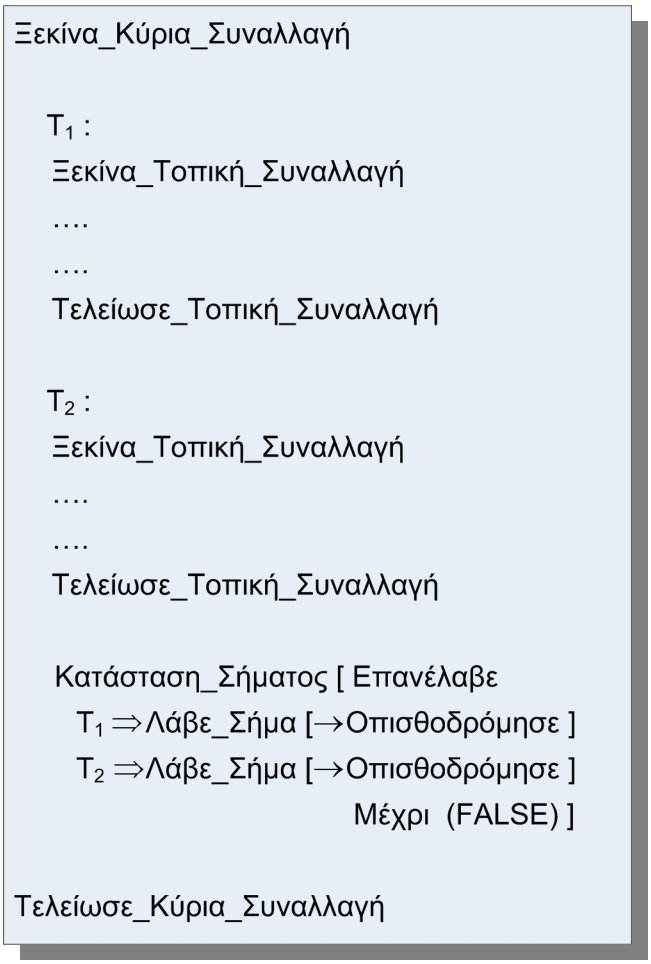
3. Η συνθήκη τερματισμού για την *Επανέλαβε…Μέχρι*, καθορίζει το πλήθος των επαναλήψεων που θα εκτελεστεί ο βρόγχος. Αν στο διάστημα αυτό, όλοι οι συμμετέχοντες απαντήσουν με ένα μήνυμα *Μη_Λήψη_Σήματος*, η κατανεμημένη συναλλαγή ολοκληρώνεται. Διαφορετικά, αν η συνθήκη τερματισμού ικανοποιηθεί και αν υπάρχει έστω και ένας συμμετέχων, ο οποίος δεν απάντησε με μήνυμα *Μη_Λήψη_Σήματος*, η κατανεμημένη συναλλαγή απορρίπτεται.

Τα σημεία του κώδικα που περιγράφουν τις τρεις παραπάνω περιπτώσεις, δίνονται στα σχήματα 8, 9 και 10. Για λόγους απλότητας, θεωρούμε πως η κατανεμημένη συναλλαγή T αποτελείται από δύο μόνο χαμηλής ανάγνωσης υπο-συναλλαγές, την T_1 και T_2. Επιπλέον, θεωρούμε πως το μόνο *σημείο αποθήκευσης* που δημιουργείται είναι το $sl_{default}$ με την εντολή *Ξεκίνα_Τοπική_Συναλλαγή*. Έτσι, αν μία υπο-συναλλαγή πρέπει να οπισθοδρομήσει, επιστρέφει σε αυτό το *σημείο αποθήκευσης*. Μπορούμε φυσικά, όπως είδαμε και παραπάνω, να δημιουργήσουμε παραπάνω από ένα *σημεία αποθήκευσης*, έτσι ώστε να υπάρχει τουλάχιστον ένα *σημείο αποθήκευσης* το οποίο θα καλύπτει όλες τις ετικέτες.



Σχήμα 8

Ας παρατηρήσουμε πως στο σχήμα 8, ο βρόγχος *Επανέλαβε…Μέχρι* θα εκτελεστεί μόνο μία φορά. Αφού ο διαχειριστής για την εντολή *Λάβε_Σήμα* σε κάθε υπο-συναλλαγή είναι εντολή *Απέρριψε_Εργασία*, ο συντονιστής απορρίπτει κάθε συμμετέχων που εξυπηρετεί ένα τέτοιο σήμα-ετικέτα (Όταν δηλαδή, έχει σπάσει πρόωρα μία χαμηλή κλειδαριά ανάγνωσης). Αν κανένας από τους συμμετέχοντες δεν εξυπηρετήσει τέτοιο σήμα (Όταν δηλαδή, δεν έχει σπάσει πρόωρα μία χαμηλή κλειδαριά ανάγνωσης), μόνο τότε ο συντονιστής ολοκληρώνει την κατανεμημένη συναλλαγή.



Σχήμα 9

Στο σχήμα 9, όποτε ένα σήμα-ετικέτα πρέπει να εξυπηρετηθεί, η υπο-συναλλαγή θα οπισθοδρομήται στο *σημείο αποθήκευσης* $sl_{default}$. Στο συγκεκριμένο παράδειγμα, ο βρόγχος της *Επανέλαβε…Μέχρι* εκτελείται με ατέρμον τρόπο. Ο μόνος τρόπος να βγει ο συντονιστής από αυτή την ατελείωτη επανάληψη, είναι αν και οι δύο υπο-συναλλαγές T_1 και T_2 απαντήσουν με μηνύματα *Μη_Λήψη_Σήματος*. Με άλλα λόγια, όταν ούτε η T_1 ούτε η T_2 δεν έχουν ελευθερώσει πρόωρα κάποια κλειδαριά χαμηλής ανάγνωσης και επιθυμούν και οι δύο να ολοκληρωθούν. Μόνο στην παραπάνω περίπτωση ο συντονιστής θα καταφέρει να βγει από τον ατέρμον βρόγχο της *Επανέλαβε…Μέχρι*.

```
Ξεκίνα_Κύρια_Συναλλαγή
   T1 :
   Ξεκίνα_Τοπική_Συναλλαγή
   ….
   ….
   Τελείωσε_Τοπική_Συναλλαγή

   T2 :
   Ξεκίνα_Τοπική_Συναλλαγή
   ….
   ….
   Τελείωσε_Τοπική_Συναλλαγή

   Κατάσταση_Σήματος [ Επανέλαβε
     T1 ⇒Λάβε_Σήμα [→AN Μετρητής_Οπισθοδρ≤3 TOTE Οπισθοδρόμησε
                    ΑΛΛΙΩΣ Απέρριψε_Εργασία ]
     T2 ⇒Λάβε_Σήμα [→AN Μετρητής_Οπισθοδρ≤8 TOTE Οπισθοδρόμησε
                    ΑΛΛΙΩΣ Απέρριψε_Εργασία  ]
                    Μέχρι  (Πλήθος_Κατάσταση_Σήματος = 10) ]
Τελείωσε_Κύρια_Συναλλαγή
```

Σχήμα 10

Στο σχήμα 10, χρησιμοποιείται η μεταβλητή-μετρητής *Μετρητής_Οπισοδρ*. Η μεταβλητή αυτή, όπως έχουμε δει, έχει τοπική εμβέλεια για κάθε υπο-συναλλαγή. Όταν μία υπο-συναλλαγή εκκινεί την εκτέλεσή της, ο μετρητής της *Μετρητής_Οπισοδρ* μηδενίζεται και αυτός αυξάνεται κατά ένα όποτε η υπο-συναλλαγή οπισθοδρομείται σε οποιοδήποτε σημείο του κώδικά της. Αν μία συναλλαγή οπισθοδρομήσει τόσες φορές όσες της επιτρέπεται και έχει ακόμα κάποιο σήμα-ετικέτα να εξυπηρετήσει, εκτελεί την εντολή *Απέρριψε_Εργασία*, με την οποία απορρίπτεται.

6. Το πρωτόκολλο *ASEP* : Η προηγμένη μορφή του.

Για την προηγμένη μορφή του πρωτοκόλλου *ASEP*, το κριτήριο για την *ASEP καλώς ορισμένη* κατανεμημένη συναλλαγή, γίνεται πιο «χαλαρό». Η προηγμένη αυτή μορφή εισάγει νέα, πιο ευέλικτα χαρακτηριστικά, όπως την *μερική οπισθοδρόμηση* (*partial rollback*) και την *εμπρόσθια ανάκτηση* (*forward recovery*). Όπως και με τη βασική μορφή του πρωτοκόλλου, θα χρησιμοποιήσουμε τα σχήματα 7a και 7b για να αναφερθούμε στις καταστάσεις ενός συντονιστή και ενός συμμετέχων.

[Αλγόριθμος 6.1: Το πρωτόκολλο *ASEP* : Η προηγμένη μορφή του]

Τα βήματα 1-4b είναι τα ίδια με τη βασική μορφή του πρωτοκόλλου. Η μόνη διαφορά είναι πως η κλήση της εντολής *Κατάσταση_Σήματος* δεν χρειάζεται να ικανοποιεί τις προϋποθέσεις μιας *ASEP καλώς ορισμένης* κατανεμημένης συναλλαγής.

4. Συμμετέχων : *s2*→ *Απόρριψη*, *s2*→ *s2*, *s2*→ *s3*. Ένας χαμηλής ανάγνωσης συμμετέχων ο οποίος λαμβάνει ένα μήνυμα *Λάβε_Σήμα*, εκτελεί τον κώδικα του διαχειριστή που καθορίζεται από την κλήση της εντολής. Η διαφορά σε σχέση με τη βασική μορφή του πρωτοκόλλου *ASEP*, είναι πως ένας χαμηλής ανάγνωσης συμμετέχων μπορεί μα μη λάβει ένα μήνυμα *Λάβε_Σήμα* και η εντολή *Λάβε_Σήμα* μπορεί να μην είναι *καλώς ορισμένη*. Στην πρώτη περίπτωση, ο χαμηλής ανάγνωσης συμμετέχων συμπεριφέρεται με τον ίδιο ακριβώς τρόπο με έναν απλό συμμετέχων (μη χαμηλής ανάγνωσης). Στην δεύτερη περίπτωση, ο διαχειριστής στην εντολή *Λάβε_Σήμα* είναι ένα οποιοδήποτε τμήμα κώδικα που δημιουργείται από τον προγραμματιστή. Πιο συγκεκριμένα, ο διαχειριστής μπορεί να είναι ένα από τα παρακάτω :

a. *Απέρριψε_Εργασία*. Η απάντηση από τον συμμετέχοντα είναι μία ψήφος «όχι», δείχνοντας την απόφαση για απόρριψη. Ο συμμετέχων τότε μετακινείται στην τελική κατάσταση *Απόρριψη*.

b. *Μη_Λήψη_Σήματος / Ολοκλήρωσε_Εργασία*. Σε αυτή τη περίπτωση, ακόμα και αν η υπο-συναλλαγή ελευθέρωσε μία κλειδαριά, η υπο-συναλλαγή δεν εξυπηρετεί το σήμα-ετικέτα. Αντί αυτού, ο συμμετέχων απαντά με ένα μήνυμα *Μη_Λήψη_Σήματος*. Έτσι, ο συμμετέχων είτε παραμένει στην κατάσταση *s2* είτε αποστέλλει μία ψήφο «ναι» και μεταβαίνει στην τελική κατάσταση *s3*.

c. Οποιοδήποτε τμήμα κώδικα. Αν ένα τέτοιο τμήμα δεν περιλαμβάνει εντολή *Οπισθοδρόμησε*, ο κώδικας εκκινεί μία *εμπρόσθια ανάκτηση* (*forward recovery*). Διαφορετικά, ίσως περιέχει εντολές *Οπισθοδρόμησε* οι οποίες εκτελούν τις διάφορες οπισθοδρομήσεις της υπο-συναλλαγής. Παρόλ'αυτά, οι οπισθοδρομήσεις αυτές μπορεί να οδηγήσουν την υπο-συναλλαγή σε ποικίλα σημεία μέσα στο σώμα της και όχι απαραίτητα στο *σημείο αποθήκευσης* το οποίο καλύπτει όλες τις χαμηλές αναγνώσεις η οποίες σηματοδοτήθηκαν με ετικέτες.

Ο συμμετέχων χαμηλής ανάγνωσης εκτελεί την εντολή *Λάβε_Σήμα*, με τον σχετικό κώδικα, και αποστέλλει μία ψήφο «ναι» ή «όχι» στον συντονιστή. Αν ο συμμετέχων στείλει μία ψήφο

«ναι», τότε μεταβαίνει στην κατάσταση *s3*, διαφορετικά απορρίπτει την υπο-συναλλαγή και μεταβαίνει στην τελική κατάσταση *Απόρριψη*.

5. Συντονιστής : *s3*→ *Απόρριψη*, *s3*→ *Ολοκλήρωση*, *s3*→ *s3*. Η αντίδραση του συντονιστή εξαρτάται από τους συμμετέχοντες, όπως ακριβώς και στη βασική μορφή του πρωτοκόλλου, στο βήμα 5 (a, b, c).

6. Συμμετέχων : *s3*→ *Απόρριψη*, *s3*→ *s2*, *s3*→ *s3*. Όπως και στο βήμα 6 της βασικής μορφής του πρωτοκόλλου, αν ο συντονιστής εκτελέσει ξανά την εντολή *Κατάσταση_Σήματος*, οι χαμηλής ανάγνωσης υπο-συναλλαγές, οι οποίες καθορίζονται στην εντολή *Κατάσταση_Σήματος* , εκτελούν ξανά την *Λάβε_Σήμα*. Ο συμμετέχων μπορεί να βρίσκεται, σε αυτό το σημείο, είτε στην κατάσταση *s2* είτε στην κατάσταση *s3*. Αν βρίσκεται στην κατάσταση *s2*, η εκτέλεση του ακολουθεί το παραπάνω βήμα 4. Διαφορετικά, αν ο συμμετέχων βρίσκεται στην κατάσταση *s3*, αποστέλλει ένα από τα παρακάτω μηνύματα, ανάλογα με το αποτέλεσμα της εκτέλεσης της εντολής *Λάβε_Σήμα* :

a. Μία ψήφο «όχι». Ο συμμετέχων απορρίπτει την υπο-συναλλαγή και μεταβαίνει στην κατάσταση *Απόρριψη*.

b. *Μη_Λήψη_Σήματος*. Ο συμμετέχων μεταβαίνει από την κατάσταση *s2* στη κατάσταση *s3*.

c. Μία ψήφο «ναι». Ο συμμετέχων παραμένει στην κατάσταση *s3*.

7. Το πρωτόκολλο συνεχίζει την εκτέλεση της κατανεμημένης συναλλαγής από το βήμα 5.

Ας σημειώσουμε ξανά εδώ πως ο παραπάνω αλγόριθμός είναι πολύ κοντά στον αλγόριθμο της βασικής μορφής του πρωτοκόλλου *ASEP*. Η μόνη διαφορά είναι ότι στη βασική μορφή, τα σήματα και οι διαχειριστές έχουν πολύ συγκεκριμένα και αυστηρά χαρακτηριστικά, σε αντίθεση με την προηγμένη μορφή του. Στη συνέχεια θα παρουσιάσουμε κάποια παραδείγματα, για να δείξουμε με ποιόν τρόπο εξασφαλίζεται η ολοκλήρωση της κατανεμημένης συναλλαγής με τη χρήση του πρωτοκόλλου *ASEP*.

6.1 Εξασφαλίζοντας ολοκλήρωση με τη χρήση της προηγμένης μορφής του πρωτοκόλλου *ASEP*

Στο σημείο αυτό, θα δώσουμε δύο παραδείγματα με σκοπό να παρουσιάσουμε πως μπορεί το πρωτόκολλο *ASEP* να εξασφαλίσει την ολοκλήρωση μιας κατανεμημένης συναλλαγής, ακόμα και αν οι υπο-συναλλαγές της ελευθερώσουν πρόωρα τις χαμηλής ανάγνωσης κλειδαριές τους. Υποθέτουμε εδώ πως υπάρχουν δύο τοποθεσίες, η τοποθεσία *A* και η τοποθεσία *B*. Στην τοποθεσία *A* αποθηκεύονται τα στοιχεία δεδομένων *q*, *s*, *x*, *y* και *z*, με $L(x) = L(z) = High$ και $L(q) = L(s) = L(y) = Low$. Στην τοποθεσία *B* αποθηκεύονται τα στοιχεία δεδομένων m, n, o και p, με $L(o) = L(p) = High$ και $L(m) = L(n) = Low$. Ο συμβολισμός T_A δηλώνει την υπο-συναλλαγή η οποία εκτελείται στην τοποθεσία *A* και όμοια ο T_B δηλώνει την υπο-συναλλαγή η οποία εκτελείται στην τοποθεσία *B*.

[Παράδειγμα 6.1.1]

Ας υποθέσουμε πως έχουμε την κατανεμημένη συναλλαγή που φαίνεται στο σχήμα 11. Αφού μόνο η υπο-συναλλαγή T_A διαβάζει χαμηλού επιπέδου ασφαλείας στοιχεία δεδομένων, η εντολή *Κατάσταση_Σήματος* θα περιέχει κώδικά μόνο για αυτή την υπο-συναλλαγή. Η T_A περιέχει τρεις λειτουργίες χαμηλής ανάγνωσης, τις $r[s], r[y]$ και $r[q]$. Αν «σπάσει» η κλειδαριά στο στοιχείο δεδομένων *s*, τότε το σήμα που θα επιλεχθεί θα είναι το $sl_{default}$ (το οποίο δημιουργήθηκε με την εκτέλεση της εντολής *Ξεκίνα_Τοπική_Συναλλαγή*, της T_A). Το σήμα θα είναι το sl_2 αν «σπάσει» η κλειδαριά του *y*, και το sl_3 σε περίπτωση που «σπάσει» η κλειδαριά του *q*. Ο προγραμματιστής χρησιμοποιεί τη βασική μορφή της εντολής *Λάβε_Σήμα*, με τη παρουσία μιας συνθήκης τερματισμού. Έτσι, αν ικανοποιηθεί η συνθήκη αυτή, το αποτέλεσμα της *Λάβε_Σήμα* θα είναι το ίδιο με το αποτέλεσμα της *Λάβε_Σήμα*$[\rightarrow Οπισθοδρόμησε]$. Διαφορετικά, θα είναι ίσιο με το αποτέλεσμα της *Λάβε_Σήμα*$[\rightarrow Μη\_Λήψη\_Σήματος]$. Στη τελευταία περίπτωση, για τα σήματα που επιλέχτηκαν, οι συμμετέχοντες απαντούν με ένα μήνυμα *Μη_Λήψη_Σήματος*. Επίσης, οι τιμές που χρησιμοποιούνται για την μεταβλητή-μετρητή *Μετρητής_Οπισθοδ.* και *Πλήθος_Κατάσταση_Σήματος* εγγυώνται ότι η T_A επιτρέπεται να εκτελέσει μία οπισθοδρόμηση πριν η κατανεμημένη συναλλαγή ολοκληρωθεί. Πριν η *Πλήθος_Κατάσταση_Σήματος* πάρει την τιμή 2, η τιμή της μεταβλητής *Μετρητής_Οπισθοδ.* της T_A θα είναι μεγαλύτερη από 1 (ακόμα και αν η T_A πρέπει να εξυπηρετήσει κάποιο σήμα), η T_A θα αποστείλει ένα μήνυμα *Μη_Λήψη_Σήματος* και ο συντονιστής θα ολοκληρώσει. Να σημειώσουμε πως η κατανεμημένη συναλλαγή στο παράδειγμα αυτό δεν είναι *ASEP καλώς ορισμένη*. Αυτό συμβαίνει γιατί η εντολή *Λάβε_Σήμα* δεν είναι *καλώς*

ορισμένη. Παρ'όλ'αυτά, αν η κατανεμημένη συναλλαγή ολοκληρωθεί μετά από την πρώτη εκτέλεση του βρόγχου, η συναλλαγή θα είναι δύο φάσεων. Ενώ αν ο βρόγχος εκτελεστεί και δεύτερη φορά, η κατανεμημένη συναλλαγή μπορεί να είναι ή όχι δύο φάσεων, με βέβαιη όμως την ολοκλήρωσή της.

```
Ξεκίνα_Κύρια_Συναλλαγή
   T_A :
   Ξεκίνα_Τοπική_Συναλλαγή
         r[s]                  /* Χαμηλή ανάγνωση */
         r[x]
         sl_2 = Αποθήκευσε_Εργασία
         w[z]
         r[y]                  /* άλλη μία χαμηλή ανάγνωση */
         sl_3 = Αποθήκευσε_Εργασία
         r[q]                  /* άλλη μία χαμηλή ανάγνωση */
   Τελείωσε_Τοπική_Συναλλαγή

   T_B :
   Ξεκίνα_Τοπική_Συναλλαγή
         r[o]
         w[p]
   Τελείωσε_Τοπική_Συναλλαγή

   Κατάσταση_Σήματος [ Επανέλαβε
     T_A ⇒ Λάβε_Σήμα [   ΑΝ Μετρητής_Οπισθοδρ < 1 ΤΟΤΕ → Οπισθοδρόμησε
                         ΑΛΛΙΩΣ → Μη_Λήψη_Σήματος ]
                          Μέχρι  (Πλήθος_Κατάσταση_Σήματος = 2) ]
Τελείωσε_Κύρια_Συναλλαγή
```

Σχήμα 11

[Παράδειγμα 6.1.2]

Ας εξετάσουμε τώρα την κατανεμημένη συναλλαγή του σχήματος 12. Αφού και οι δύο υπο-συναλλαγές T_A και T_B εκτελούν λειτουργίες χαμηλής ανάγνωσης, οι διαχειριστές στην εντολή *Κατάσταση_Σήματος* αναφέρονται και στις δύο.

Η υπο-συναλλαγή T_B διαβάζει τα στοιχεία δεδομένων *n* και *m*. Η ανάγνωση του *n* καλύπτεται από το *σημείο αποθήκευσης* $sl_{default}$ ενώ το sl_2 καλύπτει την ανάγνωση του

στοιχείου m. Ο προγραμματιστής έχει ορίσει την λειτουργία *εμπρόσθιας ανάκτησης* $r[n], w[o],$ Οπισθοδρόμησε(sl_2), ως τον κώδικα του διαχειριστή $sl_{default}$. Έτσι, αν εξυπηρετηθεί το σήμα-ετικέτα της $r[n]$, η υπο-συναλλαγή θα διαβάσει πρώτα το n και μετά θα γράψει το στοιχείο o πριν οπισθοδρομήσει στο *σημείο αποθήκευσης* sl_2 . Το προηγούμενο είναι διαφορετικό από το να γίνει η οπισθοδρόμηση στο $sl_{default}$ και να γίνει ξανά η εκτέλεση από το σημείο αυτό, επειδή οι λειτουργίες ανάμεσα στις $r[n]$ και $w[o]$ δεν θα εκτελεστούν ξανά εδώ. Επίσης, κανένας διαχειριστής δεν έχει οριστεί για το *σημείο αποθήκευσης* sl_2. Κατά συνέπεια, αν το σήμα της $r[m]$ πρέπει να εξυπηρετηθεί, ο συμμετέχων θα απαντήσει με μήνυμα *Μη_Λήψη_Σήματος*.

Ένα τυπικό σενάριο για τη εκτέλεση της εντολής *Κατάσταση_Σήματος* είναι το ακόλουθο:
Την πρώτη φορά της εκτέλεσης του βρόγχου, υποθέτουμε ότι η T_A εξυπηρετεί ένα σήμα και απαντά με μία ψήφο «ναι» προς τον συντονιστή, ενώ η T_B στέλνει ένα μήνυμα *Μη_Λήψη_Σήματος*. Στο τέλος της εκτέλεσης αυτής, η τιμή της μεταβλητής *Πλήθος_Κατάσταση_Σήματος* γίνεται 1. Επιπλέον, στο σημείο αυτό, η τιμή της μεταβλητής Όχι_Πρόωρη_Ελευθέρωση_Κλειδαραριάς είναι FALSE και η τιμή της Έτοιμο_Να_Ολοκληρωθεί είναι TRUE. Συνεπώς, ο βρόγχος εκτελείται για δεύτερη φορά. Κατά την εκτέλεση αυτή, υποθέτουμε πώς η T_A εξυπηρετεί ένα σήμα και απαντά με ένα μήνυμα *Μη_Λήψη_Σήματος*. Ως αποτέλεσμα, η τιμή της *Πλήθος_Κατάσταση_Σήματος* γίνεται 2, η *Όχι_Πρόωρη_Ελευθέρωση_Κλειδαραριάς* είναι FALSE και η *Έτοιμο_Να_Ολοκληρωθεί* είναι TRUE. Αυτό θα αναγκάσει τον βρόγχο να εκτελεστεί για τρίτη φορά. Αν κατά την τρίτη εκτέλεση του βρόγχου και η T_A αλλά και η T_B στείλουν μήνυμα *Μη_Λήψη_Σήματος* , τότε η κατανεμημένη συναλλαγή θα προχωρήσει και θα ολοκληρωθεί.

Ας παρατηρήσουμε πως, η συνθήκη τερματισμού του βρόγχου επιτρέπει σε αυτόν να εκτελεστεί το πολύ 6 φορές, πριν η κατανεμημένη συναλλαγή απορριφθεί. Φυσικά, αν κάποια από τις δύο (ή και οι δύο) υπο-συναλλαγές στείλουν μία ψήφο «όχι» μετά από την εκτέλεση κάποιου διαχειριστή, η κατανεμημένη συναλλαγή θα απορριφθεί. Τέλος, βλέπουμε πως η κατανεμημένη συναλλαγή δεν είναι *ASEP καλώς ορισμένη*. Αυτό συμβαίνει επειδή, καμία από τις δύο κλήσεις της εντολής *Λάβε_Σήμα* δεν είναι *καλώς ορισμένη*. Ο διαχειριστής για την T_A δεν οπισθοδρομεί πάντα την υπο-συναλλαγή σε σημείο αποθήκευσης που καλύπτει την σηματοδοτημένη ανάγνωση. Η κλήση, επίσης, της *Λάβε_Σήμα* για την υπο-συναλλαγή T_B δεν είναι σε μορφή που αποτελεί *καλό ορισμό* της εντολής.

```
Ξεκίνα_Κύρια_Συναλλαγή
   T_A :
   Ξεκίνα_Τοπική_Συναλλαγή
         r[s]              /* Χαμηλή ανάγνωση */
         r[x]
         sl_2 = Αποθήκευσε_Εργασία
         w[z]
         r[y]              /* άλλη μία χαμηλή ανάγνωση */
         r[q]              /* άλλη μία χαμηλή ανάγνωση */
   Τελείωσε_Τοπική_Συναλλαγή

   T_B :
   Ξεκίνα_Τοπική_Συναλλαγή
         r[n]              /* Χαμηλή ανάγνωση */
         ….
         ….
         w[o]
         sl_2 = Αποθήκευσε_Εργασία
         r[m]              /* Χαμηλή ανάγνωση */
         w[p]
   Τελείωσε_Τοπική_Συναλλαγή

   Κατάσταση_Σήματος [ Επανέλαβε
     T_A ⇒ Λάβε_Σήμα [→ Οπισθοδρόμησε(sl_2) ]
     T_B ⇒ Λάβε_Σήμα [ sl_default → r[n]; w[o]; Οπισθοδρόμησε(sl_2) ]
                       Μέχρι  (Πλήθος_Κατάσταση_Σήματος = 6) ]
Τελείωσε_Κύρια_Συναλλαγή
```

Σχήμα 12

7. Αξιολόγηση του πρωτοκόλλου *ASEP*

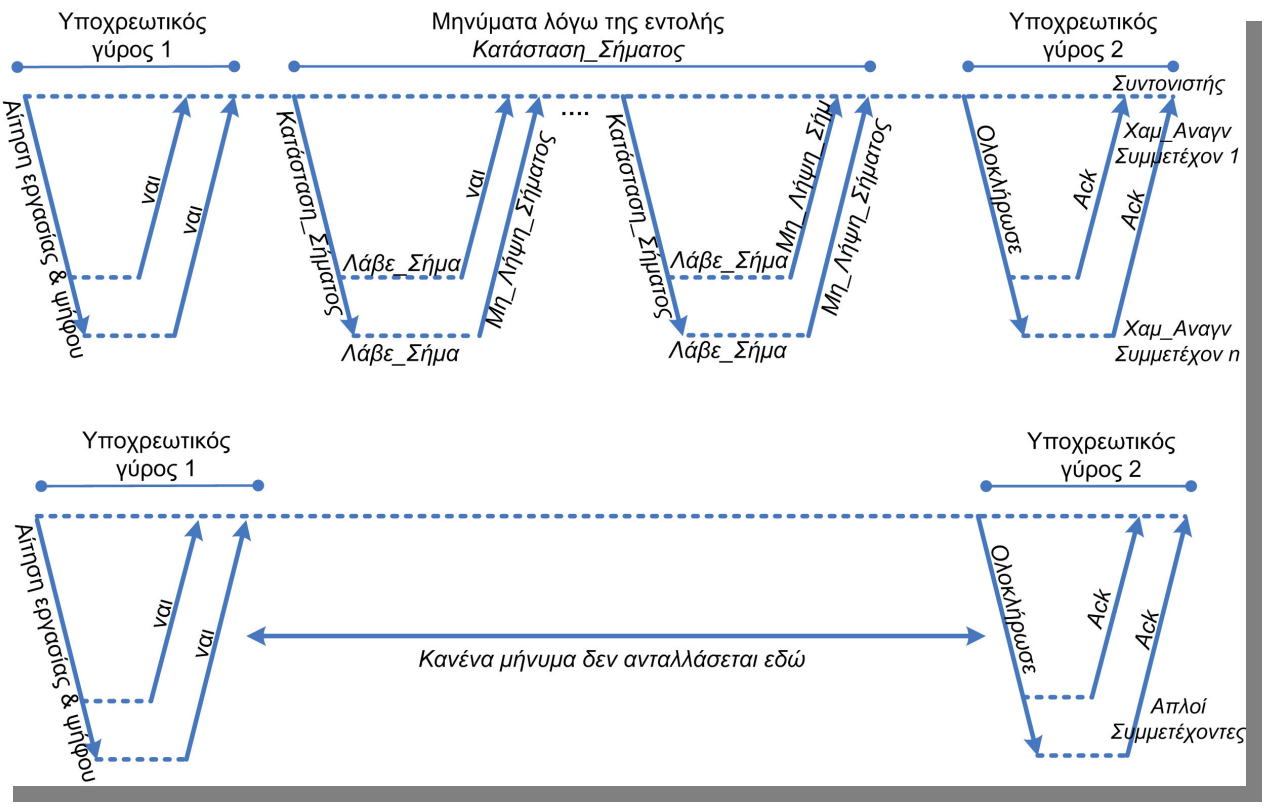
Θα αξιολογήσουμε τη βασική μορφή του πρωτοκόλλου *ASEP* με την υπόθεση πως αυτό θα ολοκληρώσει κάποια στιγμή την κατανεμημένη συναλλαγή, μετά από έναν ορισμένο αριθμό γύρων ανταλλαγής μηνυμάτων. Αν συγκρίνουμε τον αριθμό των μηνυμάτων που απαιτούνται από το πρωτόκολλο *ASEP* σε σχέση με αυτόν που απαιτούνται από το πρωτόκολλο *EP*, το *EP* απαιτεί πάντα περίπου 4n μηνύματα (όπου *n* ο αριθμός των συμμετεχόντων), ενώ το *ASEP* απαιτεί 4n μηνύματα τουλάχιστον όταν δεν υπάρχουν χαμηλής ανάγνωσης συμμετέχοντες. Αν υπάρχει έστω και ένας συμμετέχων χαμηλής ανάγνωσης, τουλάχιστο ένας επιπλέον γύρος μηνυμάτων απαιτείται μεταξύ του συντονιστή και των τοποθεσιών όπου υπάρχουν αυτοί οι συμμετέχοντες. Το πρωτόκολλο *ASEP* στην καλύτερη περίπτωση έχει ακριβώς την ίδια πολυπλοκότητα μηνυμάτων με αυτή του τυπικού πρωτοκόλλου δύο-φάσεων (6n μηνύματα), όποτε υπάρχουν υπο-συναλλαγές χαμηλής ανάγνωσης..

Στη συνέχεια, θα προσπαθήσουμε υπολογίσουμε τον αριθμό των μηνυμάτων που ανταλλάσσονται κατά την εκτέλεση του πρωτοκόλλου *ASEP*. Για την ανάλυση αυτή θα αναφερθούμε στο σχήμα 13. Υποθέτουμε πως η κατανεμημένη συναλλαγή T ολοκληρώνεται. Ας, υποθέσουμε, επίσης, πως ο αριθμός των υπο-συναλλαγών είναι ίσος με n, με n_{low} τον αριθμό των υπο-συναλλαγών που διαβάζουν χαμηλότερα από αυτές στοιχεία δεδομένων, και n_{same} τον αριθμό των υπο-συναλλαγών που διαβάζουν στοιχεία που βρίσκονται στο ίδιο επίπεδο ασφαλείας με αυτές (Δηλαδή, $n = n_{low} + n_{same}$).

Από το σχήμα 13, βρίσκουμε πως μεταξύ του συντονιστή και κάθε συμμετέχοντα, υπάρχουν δύο γύροι ανταλλαγής μηνυμάτων, οι οποίοι είναι υποχρεωτικοί, εδώ ένα σύνολο τεσσάρων μηνυμάτων. Υπάρχουν οι παρακάτω γύροι :

1. Ο πρώτος γύρος ανταλλαγής μηνυμάτων περιέχει την αποστολή της υπο-συναλλαγής T_i στην τοποθεσία i και την απάντηση του συμμετέχοντα προς τον συντονιστή, μετά την επιτυχή εκτέλεση της εργασίας της υπο-συναλλαγής, με μία ψήφο «ναι».

2. Ο δεύτερος υποχρεωτικός γύρος μηνυμάτων περιέχει την αίτηση του συντονιστή προς τον συμμετέχοντα να ολοκληρώσει την υπο-συναλλαγή του και την απάντηση επιβεβαίωσης του συμμετέχοντα (στο σχήμα αυτό συμβολίζεται με το μήνυμα *ack*).



Σχήμα 13

Ας συμβολίσουμε με *R*, τον αριθμό των επαναλήψεων που εκτελεί ο βρόγχος της εντολής *Επανέλαβε…Μέχρι*. Κάθε φορά που ο βρόγχος εκτελείται, υπάρχει ένας γύρος μηνυμάτων μεταξύ του συντονιστή και κάθε χαμηλής ανάγνωσης συμμετέχοντα, ένα σύνολο δύο μηνυμάτων επί του δικτύου. Συνεπώς, ο αριθμός των μηνυμάτων που ανταλλάσσονται μεταξύ του συντονιστή και των συμμετεχόντων, λόγω της εντολής *Κατάσταση_Σήματος*, είναι ίσος με $2*R*n_{low}$. Πέρα από αυτά τα μηνύματα, έχουμε $4*n_{low}$ υποχρεωτικά μηνύματα (τα μηνύματα στον πρώτο γύρο, όταν οι υπο-συναλλαγές αποστέλλονται στις τοποθεσίες τους μαζί με τις ψήφους των συμμετεχόντων, και τα μηνύματα στον τελευταίο γύρο, όταν η απόφαση για ολοκλήρωση αποστέλλεται στους συμμετέχοντες). Έτσι, ο συνολικός αριθμός μηνυμάτων μεταξύ του συντονιστή και των συμμετεχόντων χαμηλής ανάγνωσης ,ισούται με $4*n_{low}+2*R*n_{low}$.

Μεταξύ του συντονιστή και των συμμετεχόντων μη χαμηλής ανάγνωσης, ανταλλάσσονται $4*n_{same}$ μηνύματα. Έτσι, για το πρωτόκολλο *ASEP* ο αριθμός μηνυμάτων επί του δικτύου είναι ίσος με :

Αριθμός μηνυμάτων = $4 * n_{same} + 4 * n_{low} + 2 * R * n_{low}$

$= 4 * n + 2 * R * n_{low}$, με $n = n_{low} + n_{same}$.

Ο αριθμός των μηνυμάτων στο πρωτόκολλο *ASEP* εκφρασμένος ως κλάσμα προς τον αριθμό των μηνυμάτων στο πρωτόκολλο *EP* είναι ο παρακάτω :

$$\frac{ASEP}{EP} = \frac{4 * n + 2 * R * n_{low}}{4 * n}$$

$$= 1 + \frac{R * n_{low}}{2 * n}$$

Μία σημαντική παρατήρηση είναι πως, για τα μηνύματα με μικρό αριθμό R ισχύει $n_{low} << n$. Αυτό συμβαίνει επειδή, αν ο αριθμός των υπο-συναλλαγών χαμηλής ανάγνωσης είναι πολύ μικρότερος από τον αριθμό των υπο-συναλλαγών για μία κατανεμημένη συναλλαγή, τότε το συνολικό πλήθος μηνυμάτων στο πρωτόκολλο *ASEP* είναι σχεδόν ίσο με το ανάλογο πλήθος κατά την εκτέλεση του πρωτοκόλλου *EP*.

Βιβλιογραφία

Βιβλία

1. *Συστήματα Βασεων Δεδομένων, Θεωρία και Πρακτική Εφαρμογή*, Ιωάννης Μανωλόπουλος, Απόστολος Ν. Παπαδόπουλος, Εκδόσεις Αριστοτέλειου Πανεπιστημίου Θεσσαλονίκης.
2. *Distributed Systems, Concepts and Design*, second edition, George Coulouris, Jean Dollimore, Tim Kindberg, Addison and Wesley editions

Αρθρα

1. *Lattice-Based Access Control Models*, Ravi s. Sandhu, George Mason University, 0018-9162/93/1100-0009, 1993 IEEE.
2. *Correctness Criteria for Multilevel Secure Transactions*, Kenneth P. Smith, Barbara T. Blaustein, Sushil Jajodia, LouAnna Notargiacomo, Ieee Transactions on knowledge and data angineering, vol 8, no 1.
3. *ASEP: A Secure and Flexible Commit Protocol for MLS Distributed Database Systems*, Indrajit Ray, Luigi V. Mancini, Sushil Jajodia, Elisa Bertino, Ieee Transactions on knowledge and data angineering, vol 12, no 6.