ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΚΑΤΕΥΘΥΝΣΗ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΒΙΒΛΙΟΓΡΑΦΙΚΟ ΕΡΓΟ

Παράλληλη εκτέλεση και Λ.Σ. Προκλήσεις και Προβλήματα.

ΠΑΠΑΘΑΝΑΣΗΣ ΔΗΜΗΤΡΙΟΣ-ΝΕΚΤΑΡΙΟΣ

Α.Ε.Μ.: 151

ΕΞΑΜΗΝΟ: 10^{ο}

Κοζάνη, Αύγουστος 2012

1

Εισαγωγή

1.1 Παράλληλη επεξεργασία

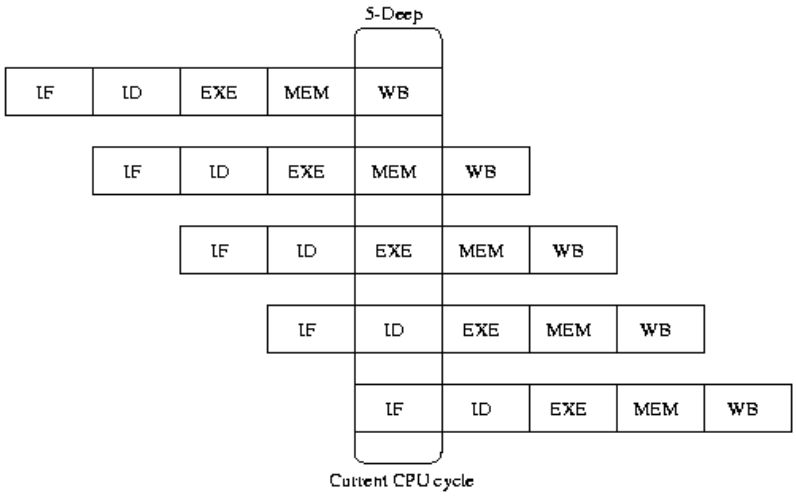
Ο τρόπος εκτέλεσης των εντολών στους πρώτους επεξεργαστές ήταν σειριακός, δηλαδή εκτελούνταν η επόμενη εντολή αφού πρώτα είχε τελειώσει η προηγούμενη. Όπως αποδείχθηκε ο τρόπος αυτός δεν είναι ο πλέον αποδοτικός για την εκτέλεση μεγάλων προγραμμάτων επειδή καθυστερεί υπερβολικά. Αναπτύχθηκε για αυτό τον λόγο η έννοια της παράλληλης επεξεργασίας, που επιτρέπει παράλληλη, δηλαδή ταυτόχρονη, επεξεργασία. Πολλές εναλλακτικές έχουν προκύψει για αυτό το σκοπό αλλά οι κυριότερες είναι η *σωλήνωση* και η *επεξεργασία νημάτων* για τα πολυεπεξεργαστικά συστήματα.

1.1.1 Σωλήνωση

Όλοι οι σύγχρονοι επεξεργαστές χρησιμοποιούν την τεχνική της *σωλήνωσης (pipeline)* για την εκτέλεση των εντολών. Η σωλήνωση εκμεταλλεύεται τα στάδια κάθε εντολής, και επιτρέπει έτσι στον επεξεργαστή να εκτελεί πολλές εντολές παράλληλα. Αν η κάθε εντολή διασπάται σε τέσσερα επίπεδα, τότε ο επεξεργαστής μπορεί κάθε χρονική στιγμή να εκτελεί επίπεδα από τέσσερις εντολές, μειώνοντας

έτσι σημαντικά τον συνολικό χρόνο εκτέλεσης του προγράμματος. Σε μια σωλήνωση τεσσάρων επιπέδων η εξοικονόμηση χρόνου μπορεί να φτάσει μέχρι και 56% της αντίστοιχης σειριακής επεξεργασίας.

Όμως ένα σημαντικό πρόβλημα που προκύπτει από την σωλήνωση είναι αυτό της εξάρτησης των εντολών. Πολλές εντολές προϋποθέτουν για την εκτέλεσή τους να έχει ολοκληρωθεί η προηγούμενη εντολή ώστε να τους επιστρέψει ένα αποτέλεσμα. Αυτό συμβαίνει στην σωλήνωση αφού όλες οι εντολές εκτελούνται σχεδόν παράλληλα. Το πρόβλημα ξεπεράστηκε εν μέρη από τις γλώσσες προγραμματισμού που εξελίχθηκαν ώστε να ανταποκρίνονται σε επεξεργασία σωλήνωσης και εν μέρη από τις ίδιες τις *Κεντρικές Μονάδες Επεξεργασίας (Κ.Μ.Ε.)* που ενσωμάτωσαν διαδικασίες ελέγχου. Έτσι όταν απαιτείται από μια εντολή ο τερματισμός της προηγούμενης, η Κ.Μ.Ε. θέτει σε αναμονή την εντολή μέχρι να ολοκληρωθεί η προηγούμενη.



Σχήμα 1.1 Σωλήνωση πέντε επιπέδων. (IF = Instruction Fetch, ID = Instruction Decode, EXE = Execute, MEM = Memory access, WB = Register write back)

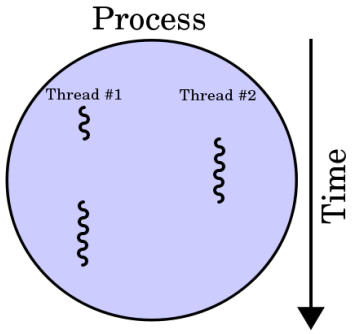
1.1.2 Επεξεργασία νημάτων

Αυτή η μέθοδος παράλληλης εξεργασίας προϋποθέτει συνήθως την ύπαρξη τουλάχιστον δύο επεξεργαστών στο ίδιο υπολογιστικό σύστημα, αν και συνήθως εφαρμόζεται σε πολύ μεγαλύτερα συστήματα. Επειδή υπάρχουν περισσότεροι του ενός επεξεργαστές είναι εφικτή η παράλληλη επεξεργασία διαφορετικών εργασιών. Έτσι κάθε εργασία διασπάται σε μικρότερες ισομεγέθεις εργασίες που ονομάζονται *νήματα (threads)*. Κάθε Κ.Μ.Ε. του συστήματος αναλαμβάνει να ολοκληρώσει ένα

νήμα σε ένα χρόνο (κβάντο) που θα έχει οριστεί. Εάν το νήμα δεν ολοκληρωθεί μέσα στα όρια του χρόνου, τότε η επεξεργασία του νήματος συνεχίζεται και στο επόμενο διάστημα χρόνου. Μπορούν επίσης να εφαρμοστούν διάφορες τεχνικές χρονοπρογραμματισμού, ώστε να υπάρχει εναλλαγή των νημάτων από διαφορετικές εργασίες.

Σε ένα μοναδικό επεξεργαστή, η πολυνηματική εκτέλεση γενικά γίνεται με πολυπλεξία διαίρεσης χρόνου: ο επεξεργαστής εναλλάσσει μεταξύ διαφορετικών νημάτων. Αυτή η *θεματική αλλαγή (context switch)* συμβαίνει αρκετά συχνά ώστε ο χρήστης να αντιλαμβάνεται ότι τα νήματα ή διεργασίες εκτελούνται ταυτόχρονα. Σε ένα πολυεπεξεργαστικό ή πολυπύρηνο σύστημα, τα νήματα ή διεργασίες εκτελούνται γενικά ταυτόχρονα, με κάθε επεξεργαστή ή πυρήνα να εκτελεί ένα συγκεκριμένο νήμα ή εργασία.

Πολλά σύγχρονα *λειτουργικά συστήματα (Λ.Σ.)* υποστηρίζουν τόσο νήματα καταμερισμού χρόνου, όσο και νήματα παράλληλης πολυεπεξεργασίας, στον χρονοδρομολογητή τους. Ο πυρήνας ενός Λ.Σ. επιτρέπει στους προγραμματιστές να χειρίζονται τα νήματα μέσω της διεπαφής κλήσεων συστήματος. Ορισμένες υλοποιήσεις λέγονται *νήμα πυρήνα*, ενώ *ελαφρά διεργασία* είναι ο ειδικός τύπος νήματος πυρήνα που μοιράζεται την ίδια κατάσταση και πληροφορίες.



Σχήμα 1.2 Μία διεργασία με δύο νήματα εκτέλεσης.

1.2 Οργάνωση έργου

Στο Κεφάλαιο 2 παρουσιάζεται η *υπερβαθμωτή αρχιτεκτονική (superscalar architecture)* καθώς και οι θεμελιώδεις περιορισμοί του παραλληλισμού τους οποίους καλείται να ξεπεράσει ένα σύστημα που υλοποιεί υπερβαθμωτή εκτέλεση. Στο

Κεφάλαιο 3 γίνεται αναφορά στην τεχνολογία *Hyper-threading* και στα προβλήματα που εισήγαγε η παράλληλη εκτέλεση μέσω αυτής της αρχιτεκτονικής. Το Κεφάλαιο 4 εξετάζει τον διαρκώς σημαντικό τομέα της παράλληλης επεξεργασίας που προκύπτει από τη συνεργασία πολλαπλών επεξεργαστών σε μία λειτουργική μονάδα *(multi-core on chip)* μη παραλείποντας, ωστόσο, τους περιορισμούς που εισήγαγαν τα πολυπύρηνα συστήματα. Στο Κεφάλαιο 5 παρουσιάζεται μια σειρά προκλήσεων που συνόδευσε την τεχνολογία των πολυπύρηνων συστημάτων ενώ στο Κεφάλαιο 6 προτείνονται οι τρόποι αντιμετώπισης που οι αρχιτέκτονες υπολογιστών ενσωμάτωσαν στα σύγχρονα Λ.Σ.

2

Υπερβαθμωτή αρχιτεκτονική

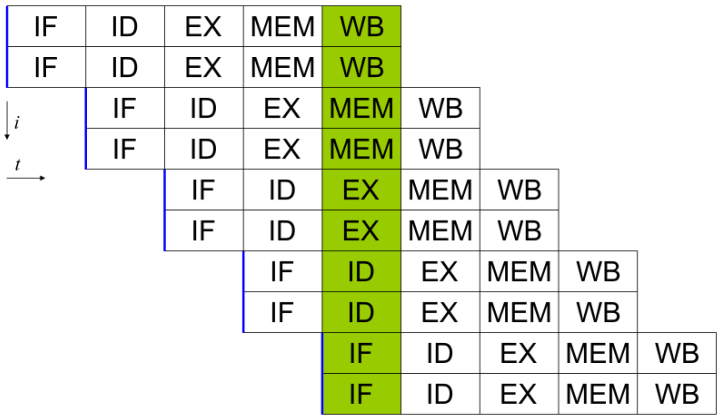
2.1 Από τους scalar στους superscalar επεξεργαστές

Οι *scalar* επεξεργαστές είναι pipelined επεξεργαστές οι οποίοι είναι σχεδιασμένοι να φορτώνουν και να προωθούν προς επεξεργασία το πολύ μία εντολή σε κάθε κύκλο μηχανής. Οι *superscalar* επεξεργαστές είναι pipelined επεξεργαστές οι οποίοι όμως είναι σχεδιασμένοι να φορτώνουν και να προωθούν προς επεξεργασία πολλές εντολές σε κάθε κύκλο μηχανής.

2.2 Υπερβαθμωτή εκτέλεση

Η υπερβαθμωτή αρχιτεκτονική υλοποιεί ένα είδος σωλήνωσης μέσω ενός επεξεργαστή. Ένας υπερβαθμωτός επεξεργαστής εκτελεί περισσότερες της μιας εντολής κατά τη διάρκεια ενός κύκλου ρολογιού, φορτώνοντας τις εντολές σε λειτουργικές μονάδες του επεξεργαστή που πλεονάζουν. Κάθε λειτουργική μονάδα δεν αποτελεί ξεχωριστό πυρήνα αλλά πόρο εκτέλεσης σε έναν ενιαίο επεξεργαστή.

Μολονότι ένας υπερβαθμωτός επεξεργαστής υλοποιεί επιπρόσθετα κλασσική σωλήνωση, -επιτρέποντας να εκτελούνται παράλληλα τα στάδια διαφορετικών εντολών (προσαγωγή, αποκωδικοποίηση, υπολογισμός τελεστών, προσαγωγή τελεστών, εκτέλεση εντολής κτλ.)- η σωλήνωση και η υπερβαθμωτή αρχιτεκτονική θεωρούνται διαφορετικές τεχνικές βελτίωσης της απόδοσης.



Σχήμα 2.1 Υπερβαθμωτή αρχιτεκτονική με σωλήνωση. (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back, *i* = Instruction number, *t* = Clock cycle [i.e., time])

2.3 Περιορισμοί

Οι θεμελιώδεις περιορισμοί του παραλληλισμού τους οποίους καλείται να ξεπεράσει ένα σύστημα που υλοποιεί υπερβαθμωτή εκτέλεση είναι:

- Εξάρτηση από πραγματικά δεδομένα
- Διαδικαστική εξάρτηση
- Συγκρούσεις πόρων
- Εξάρτηση εξόδου
- Αντιεξάρτηση

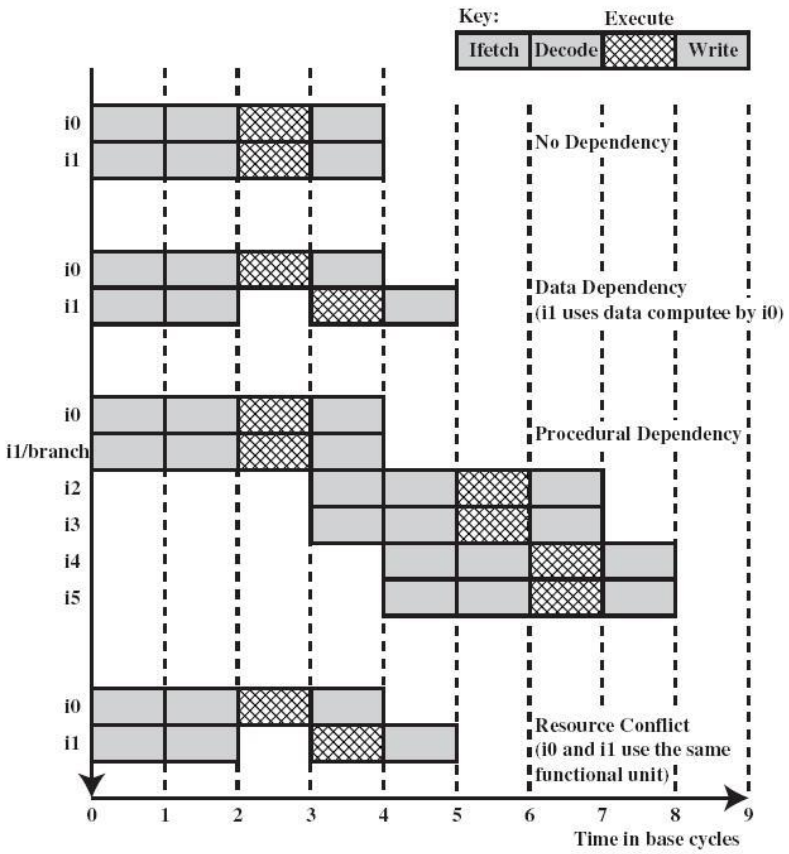
2.3.1 Εξάρτηση από πραγματικά δεδομένα

Έστω ότι έχουμε την παρακάτω σειρά:

```
add r1, r2        ;φόρτωσε στον καταχωρητή r1 τα περιεχόμενα του r2 συν τα
                  ;περιεχόμενα του r1
move r3, r1       ;φόρτωσε στον καταχωρητή r3  τα περιεχόμενα του r1
```

Η δεύτερη εντολή μπορεί να προσαχθεί και να αποκωδικοποιηθεί αλλά δεν μπορεί να εκτελεστεί μέχρι να εκτελεστεί η πρώτη εντολή. Αιτία είναι ότι η δεύτερη εντολή χρειάζεται δεδομένα που παράγονται από την πρώτη εντολή. Η περίπτωση αυτή ονομάζεται *εξάρτηση από πραγματικά δεδομένα* (ονομάζεται και *εξάρτηση ροής* ή *εξάρτηση εγγραφής-ανάγνωσης*).

Στο Σχήμα 2.2 φαίνεται αυτή η εξάρτηση σε μια μηχανή superscalar βαθμού 2. Σε απουσία εξάρτησης, μπορούν να προσαχθούν και να εκτελεστούν παράλληλα δύο εντολές. Αν υπάρχει εξάρτηση δεδομένων μεταξύ της πρώτης και δεύτερης εντολής, τότε η δεύτερη εντολή καθυστερεί κατά τόσους κύκλους ρολογιού όσοι χρειάζονται για να φύγει η εξάρτηση. Γενικά, κάθε εντολή πρέπει να καθυστερήσει μέχρι να έχουν παραχθεί όλες οι τιμές της εισόδου.



Σχήμα 2.2 Επίδραση των εξαρτήσεων.

Σε μια απλή αριθμητική σωλήνωση, η σειρά εντολών που αναφέραμε παραπάνω δεν θα προκαλούσε καθυστέρηση. Ας εξετάσουμε, ωστόσο, τα παρακάτω, όπου η μία από τις φορτώσεις είναι από την μνήμη αντί από καταχωρητή:

```
load r1, eff    ;φόρτωσε στον καταχωρητή r1 τα περιεχόμενα της
                ;αποτελεσματικής μνήμης eff
move r3, r1     ;φόρτωσε στον καταχωρητή r3  τα περιεχόμενα του r1
```

Ένας τυπικός επεξεργαστής RISC χρειάζεται δύο ή περισσότερους κύκλους για να εκτελέσει φόρτωση από την μνήμη εξαιτίας της καθυστέρησης που προκαλείται από προσπέλαση σε μνήμη ή σε μνήμη cache που βρίσκονται εκτός του chip. Ένας τρόπος αντιστάθμισης αυτής της καθυστέρησης είναι ο compiler να αναδιατάξει τις εντολές έτσι ώστε μία ή περισσότερες εντολές που δεν εξαρτώνται από τη φόρτωση από την μνήμη να μπορέσουν να ξεκινήσουν την ροή τους μέσα από την σωλήνωση. Το σχέδιο αυτό είναι λιγότερο αποτελεσματικό στην περίπτωση σωλήνωσης superscalar: Οι ανεξάρτητες εντολές που εκτελούνται κατά την διάρκεια της φόρτωσης είναι πιθανό να εκτελεστούν στον πρώτο κύκλο φόρτωσης, αφήνοντας τον επεξεργαστή χωρίς να κάνει τίποτα μέχρι να ολοκληρωθεί η φόρτωση.

2.3.2 Εξαρτήσεις από διαδικασίες

Η παρουσία διακλαδώσεων σε μια σειρά εντολών περιπλέκει την λειτουργία της σωλήνωσης. Οι εντολές που ακολουθούν μια διακλάδωση (είτε αυτή ακολουθηθεί είτε όχι) εξαρτώνται από πλευράς διαδικασίας από την διακλάδωση και δεν μπορούν να εκτελεστούν μέχρι να εκτελεστεί η διακλάδωση. Στο σχήμα 2.2 φαίνεται η επίδραση μιας διακλάδωσης σε σωλήνωση superscalar βαθμού 2.

Αυτό το είδος εξάρτησης από διαδικασία επηρεάζει και μια αριθμητική σωλήνωση. Και πάλι, οι συνέπειες σωλήνωσης superscalar είναι χειρότερες, επειδή με κάθε καθυστέρηση χάνονται περισσότερες ευκαιρίες.

Αν χρησιμοποιηθούν εντολές μεταβλητού μήκους, τότε εμφανίζεται ένα άλλο είδος εξάρτησης από διαδικασίες. Επειδή δεν είναι γνωστό το μήκος κάθε συγκεκριμένης εντολής, αυτή θα πρέπει να αποκωδικοποιηθεί τουλάχιστο κατά ένα μέρος πριν μπορέσει να προσαχθεί η επόμενη εντολή. Αυτό απαγορεύει την ταυτόχρονη προσαγωγή που χρειάζεται σε σωλήνωση superscalar. Αυτή είναι και μια από τις αιτίες που οι τεχνικές superscalar εφαρμόζονται ευκολότερα σε αρχιτεκτονική RISC ή σε αρχιτεκτονική που μοιάζει με RISC, οι οποίες έχουν σταθερό μήκος εντολών.

2.3.3 Σύγκρουση πόρων

Σύγκρουση πόρων είναι ο ανταγωνισμός δύο ή περισσότερων εντολών για τον ίδιο πόρο ταυτόχρονα. Παραδείγματα πόρων είναι οι μνήμες, οι μνήμες cache, οι δίαυλοι, οι θύρες αρχείων καταχωρητών και οι λειτουργικές μονάδες (π.χ. κύκλωμα πρόσθεσης στην ALU). Από πλευράς σωλήνωσης, η σύγκρουση πόρων εμφανίζει συμπεριφορά παρόμοια με την εξάρτηση από δεδομένα (Σχήμα 2.2). Υπάρχουν, ωστόσο κάποιες διαφορές. Οι συγκρούσεις πόρων μπορούν, τουλάχιστον, να ξεπεραστούν με διπλασιασμό των πόρων, ενώ δεν μπορεί να απαλειφθεί μια πραγματική εξάρτηση από δεδομένα. Επίσης, όταν μια πράξη χρειάζεται πολύ χρόνο για να ολοκληρωθεί, οι συγκρούσεις πόρων μπορούν να ελαχιστοποιηθούν αν βάλουμε σε σωλήνωση την κατάλληλη λειτουργική μονάδα.

2.3.4 Εξάρτηση εξόδου

Μια νέα εξάρτηση η οποία ονομάζεται *εξάρτηση εξόδου* ή *εξάρτηση εγγραφής* εμφανίζεται στο παρακάτω κομμάτι κώδικα (το op παριστάνει οποιαδήποτε πράξη):

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

Η εντολή I2 δεν μπορεί να εκτελεστεί πριν από την εντολή I1, επειδή χρειάζεται το αποτέλεσμα στον καταχωρητή το οποίο παράγεται στην I1. Αυτό είναι ένα παράδειγμα εξάρτησης από πραγματικά δεδομένα όπως το περιγράψαμε στην υποενότητα 2.3.1. Παρόμοια, η I4 θα πρέπει να περιμένει την I3, επειδή χρησιμοποιεί αποτέλεσμα που παράγεται από την I3. Τι συμβαίνει με την σχέση μεταξύ I1 και I3; Εδώ δεν υπάρχει εξάρτηση από δεδομένα, όπως την έχουμε ορίσει. Αν, ωστόσο, η I3 ολοκληρώσει την εκτέλεσή της πριν την I1, τότε για την εκτέλεση της I4 θα προσαχθεί λάθος τιμή των περιεχομένων του R3. Κατά συνέπεια, η I3 θα πρέπει να ολοκληρωθεί μετά την I1 για να δώσει τις σωστές τιμές εξόδου. Για να εξασφαλιστεί κάτι τέτοιο, θα πρέπει να ανασταλεί η έκδοση της τρίτης εντολής επειδή το αποτέλεσμά της θα μπορούσε να καλυφθεί αργότερα από παλαιότερη εντολή που χρειάζεται περισσότερο χρόνο ολοκλήρωσης.

Η ολοκλήρωση εκτός σειράς χρειάζεται πολυπλοκότερη λογική έκδοσης εντολών από την ολοκλήρωση σε σειρά. Επιπλέον, είναι δυσκολότερος ο χειρισμός διακοπών και εξαιρέσεων εντολών. Όταν συμβεί μια *διακοπή*, αναστέλλεται η εκτέλεση εντολών στο τρέχον σημείο και αναλαμβάνεται αργότερα. Ο επεξεργαστής θα πρέπει να εξασφαλίσει ότι η αναστολή θα λάβει υπόψη της ότι, την στιγμή της διακοπής, εντολές που βρίσκονται εμπρός από την εντολή που προκάλεσε την διακοπή ίσως έχουν ήδη ολοκληρωθεί.

2.3.5 Αντιεξάρτηση

Το κομμάτι κώδικα που εξετάσαμε προηγουμένως δείχνει ακόμα μια νέα εξάρτηση που ονομάζεται *αντιεξάρτηση (antidependency)* ή *εξάρτηση ανάγνωσης-εγγραφής (read-write dependency)*.

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

Η εντολή I3 δεν μπορεί να ολοκληρώσει την εκτέλεσή της πριν η εντολή I2 αρχίσει την εκτέλεσή της και έχει προσάγει τους τελεστέους της. Αυτό συμβαίνει επειδή η I3 ανανεώνει-ενημερώνει τον καταχωρητή R3, ο οποίος αποτελεί τελεστέο πηγής για την I2. Ο όρος αντιεξάρτηση χρησιμοποιείται επειδή ο περιορισμός μοιάζει με τον περιορισμό της εξάρτησης από αληθινά δεδομένα, αλλά με αντεστραμμένο τρόπο: Αντί η πρώτη εντολή να παράγει μια τιμή που χρησιμοποιεί η δεύτερη εντολή, η δεύτερη εντολή καταστρέφει μια τιμή την οποία χρησιμοποιεί η πρώτη εντολή.

3

Τεχνολογία Hyper-threading

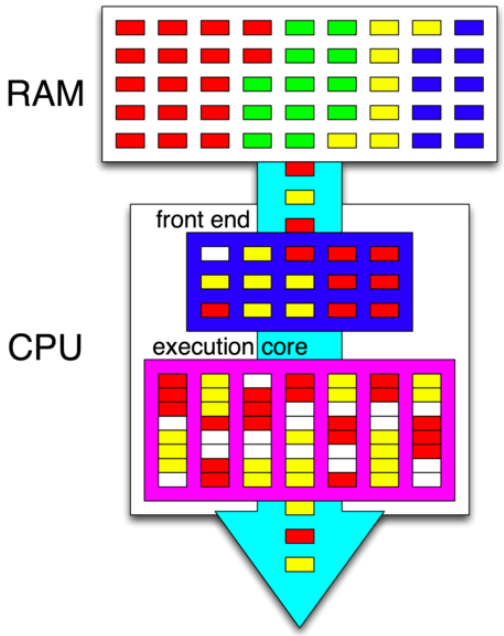
3.1 Η μετάβαση στην τεχνολογία Hyper-threading

Η τεχνολογία *Hyper-threading* (συντ. *HTT* ή *HT*) αποτελεί ορολογία της Intel Corporation, με σκοπό να περιγράψει την ταυτόχρονη εφαρμογή *πολυνημάτωσης (multithreading)*. Εμφανίστηκε για πρώτη φορά τον Φεβρουάριο του 2002 στους επεξεργαστές διακομιστών Xeon και τον Νοέμβριο του 2002 στους επεξεργαστές Pentium 4. Αργότερα η Intel συμπεριέλαβε την τεχνολογία μεταξύ άλλων στις σειρές επεξεργαστών Itanium, Atom και Core.

Σχήμα 3.1 Ο επεξεργαστής *Intel Pentium 4* που ενσωματώνει τεχνολογία Hyper-threading.

3.2 *Εκτέλεση μέσω Hyper-threading*

Η τεχνολογία Hyper-threading υλοποιείται σε συστήματα με έναν πυρήνα κάνοντάς τα να εμφανίζονται σαν να διαθέτουν πολλαπλούς επεξεργαστές, επιτρέποντας έτσι στο Λ.Σ. να εκτελεί δύο νήματα ή διεργασίες ταυτόχρονα. Στην πραγματικότητα, εκείνο το οποίο επιτυγχάνεται είναι η αύξηση του ρυθμού με τον οποίο το σύστημα μπορεί να εναλλάσσεται μεταξύ πολλών νημάτων. Έτσι ενισχύεται ο πολυδιεργασικός χαρακτήρας των προσωπικών υπολογιστών.



Σχήμα 3.2 Λειτουργία μεμονωμένου επεξεργαστή με τεχνολογία Hyper-threading.

3.3 *Περιορισμοί*

Παρόλο που η τεχνολογία HT δεν παρέχει τα πλεονεκτήματα ενός πολυπύρηνου επεξεργαστή, υπερέχει σημαντικά σε σχέση με τους μονοπύρηνους επεξεργαστές που δεν υλοποιούν την τεχνολογία HT. Ωστόσο, η παράλληλη εκτέλεση μέσω αυτής της αρχιτεκτονικής εισήγαγε και ορισμένα προβλήματα.

Στην περίπτωση εφαρμογών όπως περιήγηση στο Διαδίκτυο, επεξεργασία κειμένου και ηλεκτρονικό ταχυδρομείο, η τεχνολογία HT δεν έχει μεγάλη επίδραση. Οι σύγχρονοι επεξεργαστές είναι αρκετά γρήγοροι, ώστε βασικές εφαρμογές να περιορίζονται από την ταχύτητα του επεξεργαστή. Ωστόσο, η διεκπεραίωση

απαιτητικών εφαρμογών όπως 3D προγράμματα και επιστημονικές εφαρμογές, ευνοείται από την τεχνολογία HT.

Η βελτίωση της απόδοσης μπορεί να σημειώσει αύξηση μέχρι και της τάξης του 30%, καθιστώντας σαφές ότι η τεχνολογία HT υστερεί με αντίστοιχο διπλασιασμό πυρήνα του επεξεργαστή.

Ένα μειονέκτημα που παρατηρείται σε πολλές εφαρμογές, είναι και η υψηλή κατανάλωση ενέργειας. Δεδομένου ότι όλες οι περιοχές ενός πυρήνα θα πρέπει να είναι ενεργοποιημένες (ακόμα και σε κατάσταση stand-by), καταναλώνεται υψηλό ποσοστό ενέργειας.

4

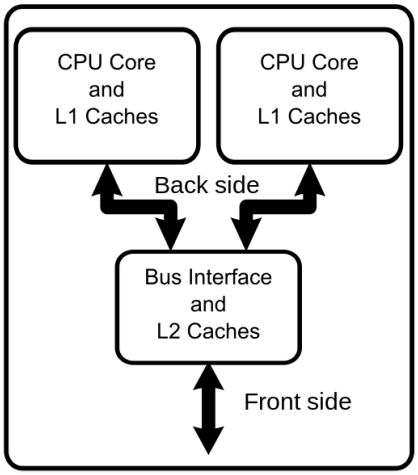
Πολυπύρηνοι επεξεργαστές

4.1 Εισαγωγή στους πολυπύρηνους επεξεργαστές

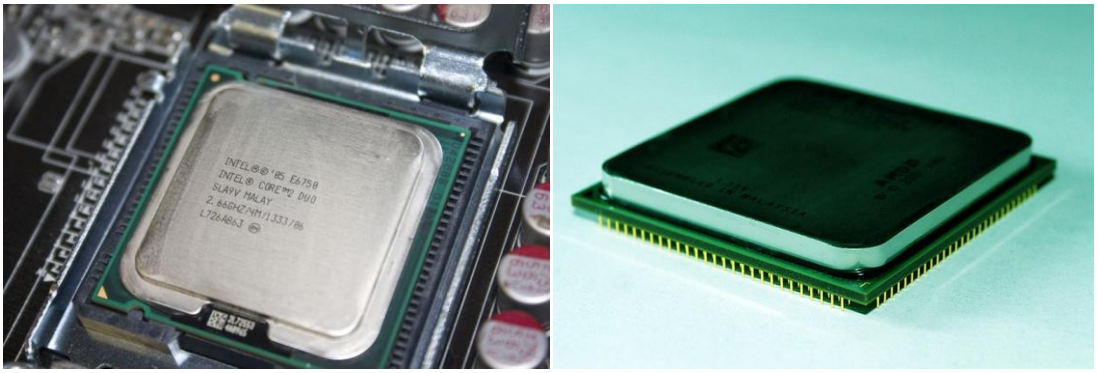
Η συνεχής αύξηση των απαιτήσεων σε ποικίλες εφαρμογές, για ταχύτερες και υψηλότερες αποδόσεις, οδηγεί στο σχεδιασμό των πολυεπεξεργαστικών συστημάτων. Ανεξάρτητα από την απόδοση ενός επεξεργαστή, η συνεργασία του με έναν ή περισσότερους επεξεργαστές υπόσχεται μεγαλύτερες αποδόσεις. Στα πολυεπεξεργαστικά συστήματα κάθε επεξεργαστής εκτελεί ανεξάρτητα μέρος του συνόλου της εργασίας, και επικοινωνεί με τους υπόλοιπους για να συγχρονίζονται και όλοι μαζί να δίνουν ένα αποτέλεσμα, πιο ικανοποιητικό από αυτό που θα έδινε ένας επεξεργαστής από μόνος του.

Ένας *πολυπύρηνος (multi-core)* επεξεργαστής αποτελεί ένα μεμονωμένο συστατικό υπολογιστή με δύο ή περισσότερες ανεξάρτητες Κ.Μ.Ε. οι οποίες καλούνται *πυρήνες (cores),* πάνω στους οποίους υπάρχει η δυνατότητα να τρέχουν πολλές διεργασίες ταυτόχρονα. Αυτό γίνεται κατορθωτό αφού οι διεργασίες που τρέχουν σε ανεξάρτητους πυρήνες μπορούν να επικοινωνούν μεταξύ τους όταν αυτό χρειάζεται. Σε συστήματα *κοινής μνήμης (shared memory)*, οι πυρήνες διασυνδέονται μεταξύ

τους χρησιμοποιώντας κάποια κοινή μνήμη μέσω της οποίας επικοινωνούν και οι διεργασίες για ανταλλαγή πληροφοριών κάνοντας χρήση κοινών μεταβλητών. Αντιθέτως σε συστήματα *κατανεμημένης μνήμης (distributed memory)*, κάθε επεξεργαστής έχει την δική του μνήμη, και η επικοινωνία μεταξύ των πυρήνων και των διεργασιών γίνεται με *ανταλλαγή-διαβίβαση μηνυμάτων (message passing)*.



Σχήμα 4.1 Διάγραμμα επεξεργαστή δύο πυρήνων.



(α) (β)

Σχήμα 4.2 Πολυπύρηνοι επεξεργαστές (α) *Intel Core 2 Duo E6750* δύο πυρήνων, (β) *AMD Athlon X2 6400+* δύο πυρήνων.

4.2 Στάδια εξέλιξης από μονοπύρηνους σε πολυπύρηνους επεξεργαστές

Ένα εύλογο ερώτημα είναι το πώς οδηγηθήκαμε σε πολυπύρηνους (multi-core) επεξεργαστές και δεν συνεχίσαμε να αυξάνουμε την *λειτουργική συχνότητα (operating frequency)* ενός μονοπύρηνου (single-core) επεξεργαστή. Η απάντηση σε αυτό το ερώτημα είναι το γεγονός ότι η αύξηση της λειτουργικής συχνότητας δεν απέδιδε την

ανάλογη αύξηση στην απόδοση του επεξεργαστή. Αυτό οφείλεται κυρίως σε τρεις παράγοντες:

- *Memory Wall:* Η αυξανόμενη διαφορά ανάμεσα στις ταχύτητες του επεξεργαστή και της μνήμης. Το κόστος ανάκτησης δεδομένων από την μνήμη *(memory latency)* αυξανόταν συνεχώς και για να γίνει κάποιος συμβιβασμός αυξάνονταν και τα μεγέθη των μνημών cache. Μετά όμως από κάποιο σημείο, το μέγεθος της cache μνήμης ήταν πολύ μεγάλο και δεν εξυπηρετούσε τον σκοπό της.

- *ILP (Instruction Level Parallelism):* Η αυξανόμενη δυσκολία εύρεσης κάποιου συνδυασμού για παραλληλισμό εντολών σε ικανοποιητικό επίπεδο για να κρατείται ένας μονοπύρηνος επεξεργαστής συνεχώς απασχολημένος. Το ILP ήταν μια τεχνική μεγιστοποίησης της απόδοσης του επεξεργαστή. Στόχος της ήταν να αυξήσει τον αριθμό εντολών που εκτελούνται από τον επεξεργαστή σε κάθε κύκλο ρολογιού.

- *Power Wall:* Η συνεχής αύξηση της λειτουργικής συχνότητας ενός μονοπύρηνου επεξεργαστή έχει ως άμεσο αποτέλεσμα την εκθετική αύξηση κατανάλωσης ενέργειας, καθώς επίσης και της θερμότητας του επεξεργαστή σε σημείο στο οποίο το υλικό δεν άντεχε πλέον τις τόσο ψηλές θερμοκρασίες.

Ακόμα ένας λόγος που οδήγησε στην ανάπτυξη των πολυπύρηνων ήταν το γεγονός ότι σε ένα μονοπύρηνο δεν υπήρχε η δυνατότητα για παράλληλη επεξεργασία. Υπήρχε μόνο η έννοια του *κατανεμημένου υπολογισμού (distributed computing)*, δηλαδή η ταυτόχρονη επεξεργασία μεγάλων όγκων δεδομένων από μια ομάδα ξεχωριστών μονοπύρηνων ηλεκτρονικών υπολογιστών, που ήταν ενωμένα σε κάποιο δίκτυο και επικοινωνούσαν μεταξύ τους μέσω ανταλλαγής μηνυμάτων, αλλά αυτό δεν ήταν αρκετό. Η συνεχής και ραγδαία αύξηση των αποθηκευμένων δεδομένων στις διάφορες βάσεις δεδομένων, αλλά και οι σύνθετοι τρόποι αποθήκευσης τους για εξοικονόμηση χώρου στους διάφορους εξυπηρετητές, έφεραν την επιτακτική ανάγκη για αύξηση στην δύναμη επεξεργασίας του επεξεργαστή ενός υπολογιστή. Επιπλέον, με την ανάπτυξη και εξέλιξη του τρόπου σχεδίασης και υλοποίησης των διάφορων λογισμικών, εμφανίστηκε η δυνατότητα να τρέχουν πολλαπλές διεργασίες

ταυτόχρονα σε κάποιες εφαρμογές. Ως αποτέλεσμα των πιο πάνω ήταν η ανάγκη που δημιουργήθηκε για αύξηση της υπολογιστικής δύναμης των επεξεργαστών, χρησιμοποιώντας παράλληλη επεξεργασία δεδομένων.

Για την υποστήριξη αυτής της δυνατότητας, υιοθετήθηκαν αρχικά κάποιες προσεγγίσεις όπως για παράδειγμα η χρήση Λ.Σ. που επέτρεπαν την ταυτόχρονη εκτέλεση πολλών διεργασιών *(multitasking)*. Στην ουσία όμως αυτές οι προσεγγίσεις το μόνο που πέτυχαν ήταν να "κρύψουν" κατά κάποιο τρόπο το χρόνο ανάκτησης δεδομένων από την μνήμη, δίνοντας την άδεια σε κάποια άλλη διεργασία να χρησιμοποιήσει τον επεξεργαστή, μέχρι να ανακτηθούν από την μνήμη τα δεδομένα που χρειαζόταν η προηγούμενη διεργασία. Δεν εξυπηρετούσε δηλαδή τον αρχικό στόχο που ήταν η παράλληλη επεξεργασία.

Μια άλλη τεχνική που χρησιμοποιήθηκε και η οποία επέτρεπε την παράλληλη επεξεργασία, ήταν ή αύξηση των επεξεργαστών σε κάθε υπολογιστή *(multiprocessor systems)*. Αυτά τα συστήματα όμως είχαν το μεγάλο μειονέκτημα του υψηλού κόστους παραγωγής. Το αμέσως επόμενο βήμα, ήταν η έρευνα που ξεκίνησε για την αποδοτικότητα χρήσης πολλών πυρήνων σε έναν επεξεργαστή. Αφού τα ποσοστά αποδοτικότητας αυτής της αρχιτεκτονικής ήταν άκρως ικανοποιητικά, οι πολυπύρηνοι επεξεργαστές θεωρήθηκαν ως το μέλλον στο πεδίο σχεδίασης επεξεργαστών. Για τους πιο πάνω λόγους, μεγάλες εταιρείες κατασκευής επεξεργαστών όπως η Intel και η AMD, έστρεψαν την προσοχή τους στους πολυπύρηνους επεξεργαστές, θυσιάζοντας τα χαμηλά κόστη παραγωγής (ενός μονοπύρηνου επεξεργαστή) για υψηλότερη απόδοση.

Η Intel προσπάθησε μέσω μιας έρευνας (Tera-scale Computing Research Program) να βρει τρόπους να προσαρμόσει τις υπολογιστικές δυνατότητες ενός υπέρ-υπολογιστή σε συσκευές καθημερινής χρήσης όπως desktops, laptops κτλ. Στην έρευνα αυτή χρησιμοποιούνται εκατοντάδες επεξεργαστές οι οποίοι επεξεργάζονται τεράστιους όγκους δεδομένων παράλληλα. Η Intel στήριξε την ανάπτυξη πολυπύρηνων επεξεργαστών στους εξής παράγοντες:

- *Απόδοση:* Μέσω της παράλληλης επεξεργασίας σε ένα πολυπύρηνο επεξεργαστή θα γίνει κατορθωτή η αύξηση της απόδοσης, αφού με τον

κατάλληλο συγχρονισμό των πυρήνων θα μειωθεί ο χρόνος υπολογισμού σε σχέση με τους μονοπύρηνους επεξεργαστές.

- *Κατανάλωση ενέργειας:* Η σκέψη της Intel είναι να χρησιμοποιήσει μια τεχνική σύμφωνα με την οποία οι πυρήνες που δεν είναι απασχολημένοι θα "σβήνουν" για μείωση της κατανάλωσης ενέργειας.

Σε αυτή την έρευνα της Intel εντοπίστηκαν κάποιες προκλήσεις σε δυο ερευνητικά πεδία:

1. Όσον αφορά την αρχιτεκτονική των μικροεπεξεργαστών έπρεπε να ληφθούν υπόψη τέσσερις παράγοντες :

   I. Βελτιστοποιημένος σχεδιασμός των πυρήνων: Το multithreading μπορεί να βοηθήσει στη μείωση του χρόνου ανάκτησης δεδομένων από την μνήμη και στη μείωση της περιττής πολυπλοκότητας.

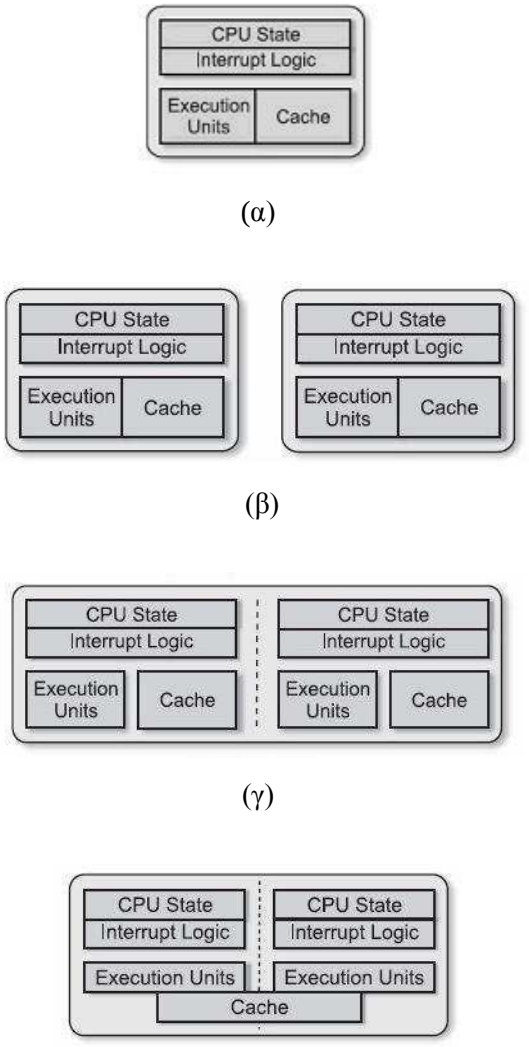
   II. Δημιουργία μονάδων που θα εκτελούν προκαθορισμένες εργασίες π.χ. επιταχυντές δικτύου (network accelerators) ή μηχανές κρυπτογράφησης (cryptography engines). Αυτές οι μονάδες θα αποδίδουν καλύτερα στις συγκεκριμένες εργασίες από κάποια μονάδα γενικής λειτουργίας.

   III. Scapable On-die Interconnect fabric: Ένας επεξεργαστής χρειάζεται να συνδέεται όχι μόνο με ένα μεγάλο αριθμό πυρήνων αλλά και με μνήμες cache και με εξειδικευμένο υλικό π.χ. κάρτες γραφικών. Συνεπώς η σύνδεση αυτή πρέπει να είναι εξελίξιμη, δηλαδή να μπορεί να υποστηρίζει μεταβλητό αριθμό πυρήνων, μνήμων κτλ.

   IV. Σχεδιασμός κυκλωμάτων με ενσωματωμένη μνήμη έτσι ώστε να γίνεται πιο αποδοτική χρήση της ενέργειας καθώς επίσης και με δυνατότητα ρύθμισης της αποδοτικότητας της ενέργειας. Δηλαδή εργασίες οι οποίες χαρακτηρίζονται ως μη κρίσιμες θα μπορούσαν να

τρέχουν σε χαμηλότερες συχνότητες για μείωση της κατανάλωσης της ενέργειας.

2. Στη σχεδίαση και ανάπτυξη νέων εφαρμογών θα πρέπει να λαμβάνεται υπόψη το γεγονός ότι θα τρέχουν σε πολυπύρηνο επεξεργαστή έτσι ώστε να γίνεται εκμετάλλευση της παράλληλης επεξεργασίας για αύξηση της επίδοσης. Αυτό όμως δεν είναι εύκολο, γιατί οι προγραμματιστές θα χρειάζονται παραπάνω χρόνο για κωδικοποίηση και αποσφαλμάτωση τέτοιων προγραμμάτων. Ο λόγος είναι η δυσκολία που παρουσιάζεται στην εφαρμογή σωστού συγχρονισμού ανάμεσα στα διάφορα threads χωρίς να παρουσιάζονται προβλήματα όπως π.χ. deadlocks και starvation.



Σχήμα 4.2 Γραφική απεικόνιση αρχιτεκτονικής (α) Single-core, (β) Multiprocessor system, (γ) Multi-core, (δ) Multi-core με κοινή cache.

4.3 Περιορισμοί

Οι περιορισμοί που εισήγαγε η παράλληλη επεξεργασία μέσω πολυπύρηνων συστημάτων είναι οι εξής:

1. Ο προγραμματισμός παράλληλου κώδικα απαιτεί πολύπλοκο συγχρονισμό των διεργασιών και μπορεί εύκολα να παρουσιάσει μικρά και δυσεύρετα λάθη λόγω της παρεμβολής των διεργασιών σε δεδομένα που βρίσκονται σε κοινή μνήμη. Τέτοιος κώδικας είναι πολύ δύσκολο να ελεγχθεί και να αποσφαλματωθεί. Επίσης οι προγραμματιστές θα πρέπει να ξεφύγουν από τον παραδοσιακό τρόπο σκέψης για σχεδίαση και υλοποίηση προγραμμάτων, και να ακολουθήσουν νέες τεχνικές οι οποίες υποστηρίζουν τη σωστή και εύκολη παραγωγή παράλληλου κώδικα που να μπορεί να τρέχει σε πολλούς πυρήνες.

2. Οι υψηλές θερμοκρασίες που αναγκαστικά θα υπάρχουν λόγω του ότι θα συνυπάρχουν πολλοί πυρήνες σε έναν επεξεργαστή συνεπάγεται μικρότερη διάρκεια ζωής του επεξεργαστή. Επίσης οι υψηλές θερμοκρασίες είναι αποτέλεσμα της ενέργειας που καταναλώνεται για να λειτουργούν οι πυρήνες. Είναι δεδομένο πως ένας πολυπύρηνος επεξεργαστής καταναλώνει περισσότερη ενέργεια από οποιοδήποτε μονοπύρηνο επεξεργαστή, αλλά αυτό είναι το αντίτιμο της υψηλότερης απόδοσης. Προς το παρόν το πρόβλημα υψηλών θερμοκρασιών στους μονοπύρηνους επεξεργαστές αντιμετωπίζεται διατηρώντας τις λειτουργικές τους συχνότητες σε χαμηλά επίπεδα.

3. Για να λειτουργεί ένας γρήγορος πολυπύρηνος επεξεργαστής χρειάζεται μεγάλες ποσότητες ηλεκτρικής ενέργειας γεγονός που κάνει το κύκλωμα επιρρεπή σε ηλεκτρικό θόρυβο, ο οποίος δημιουργεί παρεμβάσεις. Επειδή τα μονοπάτια πάνω σε ένα κύκλωμα είναι πολύ κοντά το ένα στο άλλο, όσο περισσότερη ενέργεια τρέχει μέσα στα μονοπάτια, τόσο περισσότερα λάθη γίνονται στα δεδομένα λόγω της ηλεκτρικής ακτινοβολίας που εκπέμπεται.

4. Η αύξηση του αριθμού των πυρήνων σε κάποιον επεξεργαστή δεν οδηγεί πάντα σε καλύτερες επιδόσεις. Δεν είναι απόλυτο δηλαδή ότι με το να ενσωματώσουμε περισσότερους πυρήνες σε κάποιον επεξεργαστή θα έχουμε και την ανάλογη αύξηση στην απόδοση.

5

Προκλήσεις πολυπύρηνων επεξεργαστών και παράλληλης επεξεργασίας

5.1 Εισαγωγή

Η ιδέα ένα μεμονωμένου συστατικού υπολογιστή με δύο ή περισσότερες ανεξάρτητες Κ.Μ.Ε. που θα επιτρέπει παράλληλη επεξεργασία, ακολουθήθηκε από μια σειρά προκλήσεων όπου οι αρχιτέκτονες υπολογιστών έπρεπε να αντιμετωπίσουν. Η επικοινωνία μεταξύ διεργασιών που τρέχουν παράλληλα σε διαφορετικούς πυρήνες, ο συγχρονισμός μεταξύ των διεργασιών, η συνέπεια ανάμεσα στα διάφορα επίπεδα των μνήμων cache αλλά και ο σχεδιασμός και η υλοποίηση παράλληλων προγραμμάτων είναι μερικά από τα προβλήματα που βρέθηκαν αντιμέτωποι και κλήθηκαν μέσω των σύγχρονων Λ.Σ. να ξεπεράσουν.

5.2 Προκλήσεις

Οι βασικές προκλήσεις του παραλληλισμού όπου καλείται να ξεπεράσει κάθε πολυεπεξεργαστικό σύστημα είναι:

- Η πρόκληση της διαδιεργασικής επικοινωνίας
- Η πρόκληση της μνήμης
- Η πρόκληση των παράλληλων προγραμμάτων

5.2.1 Η πρόκληση της διαδιεργασικής επικοινωνίας

Μία από τις πιο δυνατές προκλήσεις αφορά τα διάφορα κόστη που παρουσιάζονται με τη χρήση πολλών πυρήνων και παράλληλου προγραμματισμού. Για παράδειγμα το κόστος που χρειάζεται για την *εκκίνηση/τερματισμό* παράλληλων διεργασιών και την *αποστολή μηνυμάτων* μεταξύ τους. Ακόμα το κόστος *συγχρονισμού* περιορίζει την επίδοση γιατί στην πραγματικότητα μετατρέπει σε σειριακό κάποιο κομμάτι του προγράμματος και έτσι περιορίζει τον αριθμό των παράλληλων διεργασιών. Επίσης ο *ίσος καταμερισμός φορτίου (load balancing)* είναι μια δυνατή πρόκληση για τον προγραμματιστή. Ο διαμοιρασμός δηλαδή του φόρτου εργασίας πρέπει να γίνεται εξίσου ανάμεσα στους πυρήνες έτσι ώστε ο χρόνος που παραμένει αδρανής ένας πυρήνας να κρατείται στο ελάχιστο και να μειώνεται ο χρόνος υπολογισμού, αυξάνοντας την απόδοση. Όλα τα πιο πάνω κόστη είναι αναπόφευκτα και πρέπει να διατηρούνται στο ελάχιστο.

5.2.2 Η πρόκληση της μνήμης

Όσον αφορά την μνήμη, η μεγαλύτερη πρόκληση βρίσκεται στο σχεδιασμό διάφορων αποδοτικών μηχανισμών που θα εξασφαλίσουν τη συνέπεια ανάμεσα στα διάφορα επίπεδα των μνήμων cache. Αφού ο κάθε πυρήνας έχει την δική του μνήμη cache στην οποία βρίσκεται ένα αντίγραφο των δεδομένων που έχουν και οι άλλοι πυρήνες, είναι απαραίτητο αυτό το αντίγραφο να είναι πάντοτε το πιο ενημερωμένο. Όταν δηλαδή ένας πυρήνας κάνει κάποιες αλλαγές στα δεδομένα της μνήμης cache ή της κύριας μνήμης, αυτές οι αλλαγές πρέπει να υλοποιούνται αμέσως και στις μνήμες των άλλων πυρήνων. Έτσι, όταν κάποιος άλλος πυρήνας προσπαθήσει να διαβάσει τα δεδομένα αυτά, ή να τα τροποποιήσει, θα εκτελέσει την λειτουργία στα πιο ενημερωμένα δεδομένα, και ως αποτέλεσμα θα διατηρούνται ορθοί οι υπολογισμοί που γίνονται κατά την παράλληλη επεξεργασία.

5.2.3 Η πρόκληση των παράλληλων προγραμμάτων

Η πρόκληση αυτή αγγίζει τον τομέα σχεδίασης και δημιουργίας προγραμμάτων τα οποία να επωφελούνται από τα πλεονεκτήματα που προσφέρουν οι πολυπύρηνοι

επεξεργαστές και ο παραλληλισμός. Με την εισαγωγή των πολυπύρηνων επεξεργαστών επήλθε και η ανάγκη μετατροπής αλγορίθμων και προγραμμάτων ώστε να μπορούν να τρέχουν παράλληλα, και να εκμεταλλευτούν την ύπαρξη πολλών πυρήνων στο ίδιο chip. Αυτό απαιτεί από τους προγραμματιστές να σχεδιάζουν και να υλοποιούν παράλληλα προγράμματα, διαδικασία η οποία είναι επίπονη, χρονοβόρα και επιρρεπής σε λάθη. Η παραγωγή τέτοιου κώδικα προβλέπεται ότι θα είναι ο μεγαλύτερος περιορισμός στην πλήρη εκμετάλλευση των δυνατοτήτων των πολυπύρηνων επεξεργαστών.

Παρόλα αυτά υπάρχουν κάποιες τεχνικές με τις οποίες μπορούν να μετατραπούν κάποιοι σειριακοί αλγόριθμοι σε παράλληλους. Μια συνηθισμένη τεχνική που χρησιμοποιείται συχνά έχει τα ακόλουθα βήματα:

1. *Αποσύνθεση (Decomposition):* Σε πρώτο βήμα γίνεται η αναγνώριση κομματιών του προβλήματος τα οποία μπορούν να εκτελεστούν παράλληλα και χωρίζονται σε υπο-προβλήματα.

2. *Χαρτογράφηση (Mapping):* Σε αυτό το βήμα γίνεται αντιστοίχηση των υπο-προβλημάτων που δημιουργήθηκαν στην προηγούμενη φάση στα διαθέσιμα threads.

3. *Έλεγχος πρόσβασης (Access control):* Τέλος γίνεται ο συγχρονισμός και ο έλεγχος πρόσβασης σε κοινά δεδομένα από πολλαπλά threads.

Μία άλλη πρόκληση είναι οι βασικές ιδιότητες που πρέπει να πληρούν όλα τα παράλληλα προγράμματα, δηλαδή ιδιότητες ασφαλείας οι οποίες εγγυώνται ότι τίποτα κακό δε θα συμβεί ποτέ π.χ. *αδιέξοδα (deadlocks)*, και ιδιότητες βιωσιμότητας οι οποίες εγγυώνται ότι ένα πρόγραμμα τελικά θα κάνει πρόοδο π.χ. ότι κάθε πρόγραμμα κάποτε τελειώνει.

6

Αντιμετώπιση στα σύγχρονα Λ.Σ.

6.1 Εισαγωγή

Οι διεργασίες δεν αποτελούν "κλειστά συστήματα", ξεκομμένα και ανεπηρέαστα από την εξέλιξη των υπολοίπων διεργασιών του συστήματος ή του χρήστη. Αντίθετα κάθε διεργασία επηρεάζει και επηρεάζεται από τις υπόλοιπες διεργασίες με διάφορους τρόπους. Μια διεργασία π.χ. που έχει ζητήσει να διαβάσει δεδομένα από το δίσκο, αφού πρώτα περάσει τις κατάλληλες παραμέτρους στο πρόγραμμα-οδηγό του δίσκου (disk driver), αναστέλλει την λειτουργία της περιμένοντας να γίνουν διαθέσιμα τα δεδομένα. Μόλις το πρόγραμμα-οδηγός του δίσκου μεταφέρει τα δεδομένα από το δίσκο στη μνήμη θα στείλει ένα *σήμα (signal)* στην αρχική διεργασία το οποίο θα τη θέσει πάλι σε ενέργεια, ειδοποιώντας τη ότι τα δεδομένα είναι διαθέσιμα. Τα σήματα, εν συντομία, είναι διάφορες "ειδοποιήσεις" που στέλνονται σε μια διεργασία προκειμένου να την ενημερώσουν να εκτελέσει την αντίστοιχη *συνάρτηση χειρισμού* του σήματος για διάφορα σημαντικά γεγονότα. Ένας δεύτερος τρόπος επικοινωνίας των διεργασιών είναι αυτός των *σωληνώσεων (pipes)*, με την χρήση των οποίων επιτυγχάνεται η μεταφορά δεδομένων μεταξύ των διεργασιών.

Επιπρόσθετα, στα Λ.Σ. πολυεπεξεργασίας, ακριβώς λόγω της φύσης τους, εμφανίζονται κάποια προβλήματα εξαιτίας της συνύπαρξης πολλών διεργασιών που ανταγωνίζονται για τους κοινούς πόρους. Ένα απλό αλλά χαρακτηριστικό παράδειγμα αποτελεί το πρόβλημα του *κρίσιμου τμήματος*, όπως ονομάζεται, όταν ένα τμήμα κώδικα πρέπει να εκτελεστεί αδιάκοπτα από μία μόνο διεργασία. Κατά το χρονικό διάστημα αυτό, που η διεργασία εκτελεί το κρίσιμο τμήμα της, οι υπόλοιπες πρέπει να περιμένουν. Συνεπώς, είναι αναγκαία η ύπαρξη κάποιου μηχανισμού συγχρονισμού μεταξύ των διεργασιών που επιτυγχάνεται με τη χρήση των *σημαφόρων (semaphores)*.

Οι παραπάνω τρόποι καθιστούν δυνατή την επικοινωνία μεταξύ διεργασιών με βασική προϋπόθεση οι διεργασίες που επικοινωνούν να εκτελούνται στο ίδιο μηχάνημα.

Τέλος, με την αρχιτεκτονική των πολυπύρηνων συστημάτων να ενσωματώνει μία τοπική μνήμη για κάθε πυρήνα του συστήματος, γεννήθηκε το πρόβλημα της *συνοχής μνήμης (cache coherency)* το οποίο επιλύθηκε με τη χρησιμοποίηση πρωτοκόλλου καταστάσεων και *δικαιωμάτων (permissions)* μεταξύ πυρήνων και τοπικών μνημών.

6.2 Διαδιεργασική επικοινωνία

Διαδιεργασική επικοινωνία (InterProcess Communication, IPC) ονομάζεται ένα σύνολο μηχανισμών που παρέχουν τα Λ.Σ. των ηλεκτρονικών υπολογιστών, οι οποίοι διευκολύνουν την ανταλλαγή δεδομένων και τον συγχρονισμό μεταξύ ταυτοχρόνως εκτελούμενων διεργασιών μέσω δομών δεδομένων του πυρήνα. Τέτοιοι μηχανισμοί είναι απαραίτητοι στα μοντέρνα Λ.Σ. όπου, χάρη στον μηχανισμό της εικονικής μνήμης, κάθε διεργασία έχει τον δικό της ιδιωτικό χώρο εικονικών διευθύνσεων στον οποίον έχει πρόσβαση μόνο αυτή και ο πυρήνας. Προκειμένου να υπάρχει μια στοιχειώδης προστασία μνήμης μεταξύ διαφορετικών διεργασιών, καμία διεργασία δεν έχει δικαίωμα ανάγνωσης ή εγγραφής στον χώρο διευθύνσεων των υπολοίπων. Αν λοιπόν χρειάζεται δύο διαφορετικές διεργασίες να επικοινωνήσουν μεταξύ τους ή να ανταλλάξουν δεδομένα, αυτό μπορεί να γίνει μόνο μέσω του συστήματος αρχείων (π.χ. μια διεργασία να γράψει ένα αρχείο και μια άλλη να το διαβάσει) ή μέσω μιας μεθόδου διαδιεργασική επικοινωνίας.

Σε πολλές περιπτώσεις ένα εκτελούμενο πρόγραμμα (η *μητρική* ή *γονική* διεργασία) δημιουργεί δευτερεύουσες *(θυγατρικές)* διεργασίες ώστε να εκμεταλλευτεί πιθανά οφέλη από τον ταυτοχρονισμό. Με αυτόν τον τρόπο, σε ένα παράλληλο σύστημα οι υπολογισμοί που απαιτούνται από μια εφαρμογή μπορούν να κατανεμηθούν σε πολλαπλούς επεξεργαστές με τον καθένα να εκτελεί διαφορετική διεργασία, ενώ σε ένα σειριακό σύστημα αν μια διεργασία ανασταλεί (π.χ. σε μια κλήση συστήματος) καθώς περιμένει την απελευθέρωση ενός πόρου (π.χ. πρόσβαση στον σκληρό δίσκο) ή μια είσοδο από τον χρήστη), κάποια άλλη διεργασία μπορεί να συνεχίσει τους υπολογισμούς. Είναι φανερό επομένως ότι η διαδιεργασική επικοινωνία δεν είναι απαραίτητη μόνο για την ανταλλαγή δεδομένων μεταξύ ανεξάρτητων διεργασιών, αλλά και για τον συντονισμό στενά συνεργαζόμενων διεργασιών οι οποίες εκτελούνται παράλληλα, σε συστήματα πολλαπλών επεξεργαστών.

6.2.1 Σήμα

Τα *σήματα (signals)* είναι μια περιορισμένη μορφή διαδιεργασικής επικοινωνίας στο Unix και άλλα Λ.Σ. τα οποία υπακούν στο πρότυπο POSIX. Πρόκειται για διακοπές λογισμικού που δρουν ως σινιάλα και μπορούν να αποστέλλονται σε μια διεργασία από κάποια άλλη ή από τον πυρήνα, αναγκάζοντάς την να τα χειριστεί ασύγχρονα μόλις τα λάβει και ακολούθως να επιστρέφει στην κανονική ροή εκτέλεσης. Κάθε σήμα διακρίνεται από έναν ακέραιο με τον οποίο είναι συσχετισμένο κάποιο συμβολικό όνομα (SIGxxxx).

Το μοντέλο είναι το εξής: ένα πρόγραμμα στον κώδικά του δηλώνει ότι μια συνάρτηση είναι *χειριστής* ενός συγκεκριμένου σήματος (εγκατάσταση χειριστή). Όταν ληφθεί το σήμα αυτό κατά την εκτέλεση της αντίστοιχης διεργασίας, αυτομάτως η τελευταία διακόπτει ό,τι έκανε (εντολή Α) και εκτελεί τον χειριστή. Μόλις ο χειριστής επιστρέψει ο έλεγχος δίνεται ξανά στην εντολή Α. Ένας χειριστής εγκαθίσταται με την κλήση συστήματος `sigaction()` (αν δίνεται η ειδική τιμή SIG_IGN ως όρισμα στη `sigaction()` αντί για όνομα συνάρτησης, τότε το αντίστοιχο σήμα αγνοείται από τη διεργασία), ενώ με τις κλήσεις `kill()` ή `sigqueue()` το τρέχον πρόγραμμα αποστέλλει ένα συγκεκριμένο σήμα σε μία συγκεκριμένη διεργασία (η διαφορά τους είναι ότι η `sigqueue()`, μαζί με το σήμα, αποστέλλει και έναν επιπλέον ακέραιο με δεδομένα του προγραμματιστή). Σχεδόν για κάθε σήμα υπάρχει κάποιος προεπιλεγμένος χειριστής ο οποίος παρέχεται από το

πρότυπο POSIX και δεν χρειάζεται να γραφεί ή να εγκατασταθεί χειροκίνητα (συνήθως επιφέρει τον τερματισμό της διεργασίας ή αγνοεί τελείως το σήμα), για κάποια συγκεκριμένα σήματα μάλιστα δεν μπορεί να υποσκελιστεί από χειριστή του προγραμματιστή.

6.2.2 Χειρισμός και αποστολή σημάτων

Συνήθως μια διεργασία μπορεί να αποστείλει σήματα μόνο σε άλλες διεργασίες του ίδιου χρήστη, εκτός κι αν ανήκει στον λογαριασμό του διαχειριστή του συστήματος οπότε μπορεί να αποστείλει σήματα σε οποιαδήποτε άλλη διεργασία. Αν μια διεργασία δεν εκτελείται όταν ο πυρήνας τής παραδίδει κάποιο σήμα (π.χ. αν εκείνη τη στιγμή εκτελείται κάποια άλλη σε συνθήκες πολυδιεργασίας), τότε το σήμα αποθηκεύεται και της παραδίδεται μόλις λάβει τον έλεγχο του επεξεργαστή. Σε ορισμένες περιπτώσεις όμως η αποστολή σήματος προκαλεί άμεση θεματική εναλλαγή και εκτέλεση της διεργασίας-παραλήπτη, ώστε ο χειρισμός του σήματος να γίνει με πολύ μικρή καθυστέρηση. Αν μια διεργασία αρχίσει να χειρίζεται κάποιο σήμα διακόπτοντας έτσι την εκτέλεση κάποιας κλήσης συστήματος, τότε η τελευταία αποτυγχάνει και επιστρέφει τη συμβολική τιμή EINTR μετά τον τερματισμό του χειριστή. Στα περισσότερα Λ.Σ. είναι δυνατόν με χρήση της συμβολικής τιμής SA_RESTART στα ορίσματα της `sigaction()` να προσδιοριστεί ότι οι κλήσεις συστήματος οι οποίες διακόπτονται από τον αντίστοιχο χειριστή κατά τον χρόνο εκτέλεσης, θα επανεκκινούνται αυτομάτως αντί να αποτυγχάνουν. Σε ορισμένα συστήματα αυτή είναι η προεπιλεγμένη συμπεριφορά.

Σημαντικό ζήτημα είναι η αναστολή παράδοσης σημάτων στην τρέχουσα διεργασία ώστε να μην μπορούν να εκτελεστούν οι αντίστοιχοι χειριστές σε χρονικές περιόδους που θα μπορούσαν να προξενήσουν προβλήματα. Αυτή η αναστολή ορίζεται από μια μάσκα μπλοκαρισμένων σημάτων η οποία τροποποιείται με την κλήση `sigprocmask()`, ενώ με την κλήση `sigpending()` μια διεργασία μπορεί κάθε στιγμή να ελέγξει ποια σήματα τής έχουν αποσταλεί αλλά δεν τα έχει παραλάβει επειδή τα έχει μπλοκάρει. Συνήθως είναι απαραίτητη η αναστολή παράδοσης όλων των σημάτων όσο εκτελείται ένας χειριστής ώστε να διασφαλιστεί η ομαλή λειτουργία του προγράμματος. Προκειμένου να μην καλείται χειροκίνητα η `sigprocmask()` μία φορά στην αρχή του χειριστή (όπου τίθεται η νέα μάσκα που περιλαμβάνει όλα τα σήματα) και μία φορά στο τέλος του (όπου επαναφέρεται η παλαιά μάσκα), παρέχεται

η δυνατότητα ενσωμάτωσης της νέας μάσκας στον ίδιο τον χειριστή, μέσω κατάλληλων ορισμάτων της `sigaction()`, ώστε η αλλαγή της μάσκας να συμβαίνει αυτόματα μόλις ενεργοποιείται και μόλις επιστρέφει ο χειριστής. Τη στιγμή που αναιρείται η αναστολή παράδοσης ενός σήματος Α τότε, αν όντως εστάλη τουλάχιστον ένα σήμα Α κατά το χρονικό διάστημα της αναστολής στην τρέχουσα διεργασία, αμέσως παραδίδεται στην τελευταία ένα σήμα Α.

Το πρότυπο POSIX παρέχει επίσης τις κλήσεις **`pause()`** και `sigwait()`, οι οποίες μπλοκάρουν την καλούσα διεργασία επ' αόριστον μέχρι αυτή να λάβει κάποιο σήμα (οποιοδήποτε με την `pause()` ή κάποιο συγκεκριμένο με τη `sigwait()`), την κλήση `sleep()`, η οποία μοιάζει με την `pause()` αλλά μπλοκάρει την καλούσα διεργασία μόνο για πεπερασμένο μέγιστο χρονικό διάστημα το οποίο της δίνεται ως όρισμα, και την οικογένεια κλήσεων `sigXXXset()` που επενεργούν σε δομές δεδομένων οι οποίες αναπαριστούν μάσκες σημάτων, προσθαφαιρώντας συγκεκριμένα σήματα σε αυτές ή ελέγχοντας ποια σήματα περιλαμβάνουν. Η πρότυπη βιβλιοθήκη της C παρέχει εξειδικευμένες δομές δεδομένων οι οποίες αναπαριστούν μάσκες σημάτων αλλά και κάποια από τα ορίσματα της `sigaction()`.

No	Name	Default Action	Description
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort program (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	non-existent system call invoked
13	SIGPIPE	terminate process	write on a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop
20	SIGCHLD	discard signal	child status has changed
21	SIGTTIN	stop process	background read attempted from control terminal
22	SIGTTOU	stop process	background write attempted to control terminal
23	SIGIO	discard signal	I/O is possible on a descriptor
24	SIGXCPU	terminate process	cpu time limit exceeded (see
25	SIGXFSZ	terminate process	file size limit exceeded (see
26	SIGVTALRM	terminate process	virtual time alarm (see
27	SIGPROF	terminate process	profiling timer alarm (see
28	SIGWINCH	discard signal	Window size change
29	SIGINFO	discard signal	status request from keyboard
30	SIGUSR1	terminate process	User defined signal 1
31	SIGUSR2	terminate process	User defined signal 2

Πίνακας 6.1 Το σύνολο των σημάτων ορισμένα στην `<signal.h>` όπως προσδιορίζονται από την *Single UNIX Specification (SUS)*.

6.2.3 Σωληνώσεις

Η παλαιότερη μέθοδος διαδιεργασικής επικοινωνίας στο Unix είναι οι *σωληνώσεις (pipes)*. Πρόκειται για δομές δεδομένων του πυρήνα που επιτρέπουν σε δύο συγγενείς διεργασίες να ανταλλάσουν αμφίδρομα δεδομένα. Εμφανίζονται στον χώρο του χρήστη ως ζεύγη περιγραφέων αρχείων, χωρίς να αντιστοιχούν σε πραγματικά αρχεία, όπου ο ένας περιγραφέας είναι άκρο εγγραφής και ο άλλος άκρο ανάγνωσης. Οι συνήθεις κλήσεις συστήματος για Είσοδο/Εξοδο σε αρχεία (`read()`, `write()`, `close()`) βρίσκουν εφαρμογή και εδώ. Σε κάθε σωλήνωση μπορούν να αντιστοιχούν πολλαπλά ζεύγη περιγραφέων σε διαφορετικές διεργασίες, υλοποιώντας έτσι τη διαδιεργασική επικοινωνία. Αυτή η πολλαπλότητα γίνεται εφικτή μέσω της κατασκευής νέων διεργασιών (μοντέλο `fork()`, όπου η θυγατρική κληρονομεί όλους τους ανοικτούς περιγραφείς της γονικής διεργασίας). Είναι ευθύνη του προγραμματιστή να κλείσει τα άκρα ανάγνωσης ή εγγραφής στις αντίστοιχες διεργασίες ώστε να επιτύχει την επιθυμητή συμπεριφορά (π.χ. μια διεργασία να γράφει και η γονική της να διαβάζει τη σωλήνωση). Το μειονέκτημα είναι ότι με σωληνώσεις μόνο συγγενείς διεργασίες μπορούν να επικοινωνήσουν (γονική με θυγατρική ή αδελφές διεργασίες μεταξύ τους). Μια σωλήνωση δημιουργείται με την κλήση συστήματος `pipe()`. Η ανάγνωση από σωλήνωση με όλα τα άκρα εγγραφής κλειστά, είτε διαβάζει όλα τα δεδομένα που τοποθετήθηκαν στη σωλήνωση προτού κλείσει κάποιο άκρο εγγραφής και δεν αναγνώστηκαν ακόμη, είτε (αν δεν ισχύει αυτή η περίπτωση) η `read()` επιστρέφει αμέσως 0. Η εγγραφή σε σωληνώσεις με όλα τα άκρα ανάγνωσης κλειστά οδηγεί στην αποστολή του σήματος SIGPIPE από τον πυρήνα στην καλούσα διεργασία και στην αποτυχία της κλήσης `write()`. Σε σχέση με τις σωληνώσεις ενδιαφέρον παρουσιάζει η κλήση συστήματος `dup()`, η οποία δέχεται ως όρισμα έναν περιγραφέα αρχείου και τον κλωνοποιεί αναθέτοντας στο αντίγραφο τον πρώτο ελεύθερο αναγνωριστικό αριθμό (έτσι π.χ. μπορεί να αντικατασταθεί με κάποια σωλήνωση η τυπική είσοδος ή η τυπική έξοδος μιας διεργασίας).

Εναλλακτικά, μπορούν να χρησιμοποιηθούν *κατονομασμένες σωληνώσεις (named pipes ή FIFOs)*. Αυτές κατά τη δημιουργία τους αντιστοιχίζονται σε οντότητες του

συστήματος αρχείων, εικονικά αρχεία, ώστε να μπορούν να προσπελαστούν από μη συγγενείς διεργασίες που γνωρίζουν το μονοπάτι του εικονικού αρχείου. Ο χειρισμός τους γίνεται όπως των κοινών αρχείων, οπότε κάθε διεργασία για κάθε FIFO διατηρεί μόνο έναν περιγραφέα αρχείου ο οποίος επιστρέφεται από μια κοινή κλήση `open()`. Το μειονέκτημα είναι ότι ως αποτέλεσμα μια κατονομασμένη σωλήνωση λειτουργεί μονόδρομα, με κάθε διεργασία να μπορεί μόνο να γράφει σε αυτήν ή μόνο να διαβάζει από αυτήν, όπως συμβαίνει πάντα στο POSIX με τα αρχεία. Ένα FIFO δημιουργείται με την κλήση συστήματος `mkfifo()` και η διεργασία που το δημιούργησε πρέπει, αφού το κλείσει με την `close()`, να το διαγράψει από το σύστημα αρχείων με την κλήση συστήματος `unlink()`. Συνήθως πριν από την κλήση της `close()` η διεργασία αυτή διαβάζει δεδομένα από το FIFO, μέσω της κλήσης `read()`, τα οποία έχει γράψει εκεί κάποια άλλη διεργασία που γνώριζε το μονοπάτι του FIFO στο σύστημα αρχείων και το έχει ανοίξει για εγγραφή. Φυσιολογικά, κάθε κλήση `open()` για ανάγνωση μιας κατονομασμένης σωλήνωσης μπλοκάρει την καλούσα διεργασία μέχρι η σωλήνωση να ανοιχτεί από κάποια άλλη διεργασία για εγγραφή (και αντίστροφα). Αυτή η συμπεριφορά μπορεί να τροποποιηθεί με κατάλληλα ορίσματα (μη ανασταλτική `open()`) και σε αυτήν την περίπτωση το άνοιγμα για εγγραφή σε FIFO που δεν έχει ανοιχτεί για ανάγνωση αποτυγχάνει, ενώ το άνοιγμα για ανάγνωση σε FIFO που δεν έχει ανοιχτεί για εγγραφή επιστρέφει αμέσως. Οι αναγνώσεις από το FIFO διατηρούν το μέγεθος των αιτήσεων εγγραφής που προηγήθηκαν, αυτό σημαίνει πως αν μία διεργασία A1 γράψει N Bytes, στη συνέχεια η ίδια γράψει M Bytes και τελικώς μία διεργασία A2 διαβάσει το FIFO, η πρώτη αίτηση ανάγνωσης θα επιστρέψει τα αρχικά N Bytes και όχι τα ολικά N+M Bytes.

Πολλές διεργασίες μπορούν ταυτόχρονα να προσπελαύνουν το ίδιο αρχείο ή FIFO, πιθανώς για να ανταλλάξουν δεδομένα, με αποτέλεσμα να εμφανίζονται συνθήκες συναγωνισμού. Το πρόβλημα επιλύεται με τα *κλειδώματα αρχείων*, δηλαδή εξειδικευμένα *mutex* που παρέχει ο πυρήνας για *αμοιβαίο αποκλεισμό*. Τα κλειδώματα είναι είτε κοινόχρηστα είτε αποκλειστικά (με πολλαπλές διεργασίες να μπορούν να κατέχουν ταυτόχρονα ένα κοινόχρηστο κλείδωμα για το ίδιο αρχείο αλλά όχι ένα αποκλειστικό κλείδωμα, ή παράλληλα ένα κοινόχρηστο και ένα αποκλειστικό κλείδωμα) και αφορούν ένα αρχείο και όχι έναν περιγραφέα αρχείου: αν ο τελευταίος κλωνοποιηθεί, με μια κλήση `dup()` ή `fork()`, δεν υπάρχουν πλέον δύο κλειδώματα

αλλά δύο αναφορές στο ίδιο κλείδωμα. Τα κλειδώματα αρχείων είναι προσπελάσιμα μέσω της εξειδικευμένης κλήσης συστήματος `flock()` ή στο πλαίσιο της `fcntl()`, μιας γενικής χρήσης κλήσης χειρισμού αρχείων. Όταν μια διεργασία ανοίγει ένα αρχείο για ανάγνωση ζητά ένα κοινόχρηστο κλείδωμα, ενώ όταν το ανοίγει για εγγραφή ζητά ένα αποκλειστικό κλείδωμα. Στην τελευταία περίπτωση, αν το κλείδωμα του αρχείου το κατέχει κάποια άλλη διεργασία τότε η καλούσα μπλοκάρει μέχρι το κλείδωμα να ελευθερωθεί.

6.2.4 Σημαφόρος

Ο *σημαφόρος* ή *σηματοφόρος* ή *σηματοφορέας (semaphore)* είναι μια προγραμματιστική δομή δεδομένων, κύρια χρήση της οποίας είναι ο συγχρονισμός ταυτόχρονα εκτελούμενων διεργασιών ενός προγράμματος, ή ακόμα και ο συγχρονισμός διαφορετικών προγραμμάτων.

Το πρώτο μεγάλο βήμα για την αντιμετώπιση των προβλημάτων των παράλληλων διεργασιών σημειώθηκε το 1965, στη διατριβή του Dijkstra. Τον Dijkstra απασχόλησε η σχεδίαση του Λ.Σ. ως συλλογή συνεργαζόμενων σειριακών διεργασιών, καθώς και η ανάπτυξη αποδοτικών και αξιόπιστων μηχανισμών για την υποστήριξη της συνεργασίας. Οι μηχανισμοί αυτοί μπορούν εύκολα να χρησιμοποιηθούν από διεργασίες χρήστη, στην περίπτωση που οι μηχανισμοί διατίθενται από τον επεξεργαστή και το Λ.Σ.

Ο θεμελιώδης κανόνας είναι ο εξής: Δύο ή περισσότερες διεργασίες μπορούν να συνεργάζονται μέσω απλών σημάτων, τέτοιας μορφής ώστε μια διεργασία να μπορεί να εξαναγκαστεί να σταματήσει σε συγκεκριμένη θέση, μέχρι να λάβει συγκεκριμένο σήμα. Οποιαδήποτε απαίτηση πολύπλοκου συντονισμού μπορεί να ικανοποιηθεί από την κατάλληλη δομή σημάτων. Για τη σηματοδοσία χρησιμοποιούνται ειδικές μεταβλητές, οι σημαφόροι. Για τη μετάδοση σήματος διαμέσου του σημαφόρου `s`, μια διεργασία εκτελεί την πρωτογενή λειτουργία `semSignal (s)`. Για την παραλαβή σήματος διαμέσου του σημαφόρου `s`, μια διεργασία εκτελεί την πρωτογενή λειτουργία `semWait (s)`. Αν το αντίστοιχο σήμα δεν έχει μεταδοθεί ακόμα, η διεργασία αναστέλλεται μέχρι να λάβει χώρα η μετάδοση.

Προκειμένου να επιτευχθεί το επιθυμητό αποτέλεσμα, ο σημαφόρος μπορεί να θεωρηθεί μεταβλητή που διαθέτει μία ακέραιη τιμή, στην οποία ορίζονται τρεις μόνο λειτουργίες:

1. Ο σημαφόρος μπορεί να αρχικοποιηθεί, λαμβάνοντας μη αρνητική ακέραιη τιμή.

2. Η λειτουργία `semWait` μειώνει την τιμή του σημαφόρου. Αν η τιμή του σημαφόρου γίνει αρνητική, τότε μπλοκάρεται η διεργασία που εκτελεί τη `semWait`. Διαφορετικά, η διεργασία συνεχίζει την εκτέλεσή της.

3. Η λειτουργία `semSignal` αυξάνει την τιμή του σημαφόρου. Αν η τιμή που προκύπτει είναι μικρότερη ή ίση με το μηδέν, τότε ξεμπλοκάρεται μια διεργασία (αν υπάρχει) που έχει μπλοκαριστεί από μια λειτουργία `semWait`.

Εκτός από αυτές τις τρεις λειτουργίες δεν μπορεί κανείς να ελέγξει ή να χειριστεί τους σημαφόρους. Οι λειτουργίες αυτές εξηγούνται ως ακολούθως. Στην αρχή ο σημαφόρος έχει μηδενική ή θετική τιμή. Αν η τιμή είναι θετική, τότε η τιμή αυτή ισούται με το πλήθος των διεργασιών που μπορούν να εκδώσουν αίτηση αναμονής και αμέσως να συνεχίσουν να εκτελούνται. Αν η τιμή είναι μηδενική, είτε εξαιτίας της αρχικοποίησης είτε επειδή το πλήθος των διεργασιών που έχει εκδώσει αίτηση αναμονής είναι ίσο με την αρχική τιμή του σημαφόρου, τότε μπλοκάρεται η επόμενη διεργασία που εκδίδει αίτηση αναμονής και η τιμή του σημαφόρου γίνεται αρνητική. Κάθε επακόλουθη αναμονή οδηγεί την τιμή του σημαφόρου σε πιο αρνητικά επίπεδα. Η αρνητική τιμή ισούται με το πλήθος των διεργασιών που αναμένουν να ξεμπλοκαριστούν.

Εν συνεχεία, σημειώνονται τρεις ενδιαφέρουσες συνέπειες που προκύπτουν από τον ορισμό του σημαφόρου:

- Σε γενικές γραμμές, κανείς δεν μπορεί να γνωρίζει αν μια διεργασία θα μπλοκαριστεί ή όχι, πριν αυτή μειώσει ένα σημαφόρο.

- Αφότου μια διεργασία αυξήσει κάποιο σημαφόρο και ξυπνήσει κάποια άλλη διεργασία και οι δύο διεργασίες θα συνεχίσουν να εκτελούνται ταυτοχρονικά. Κανείς δεν μπορεί να γνωρίζει ποια από τις δύο ή ακόμα και αν μία από τις δύο θα συνεχίσει αμέσως σε ένα σύστημα μονού επεξεργαστή.

- Όταν αποστέλλεται σήμα σε ένα σημαφόρο, κανείς δε γνωρίζει απαραίτητα αν μια άλλη διεργασία βρίσκεται σε αναμονή και επομένως το πλήθος των διεργασιών που ξεμπλοκάρονται μπορεί να είναι είτε μηδέν είτε ένα.

Επιπρόσθετα, ορίζεται μια περιορισμένη έκδοση, γνωστή ως *δυαδικός σημαφόρος (binary semaphore)*. Ο δυαδικός σημαφόρος μπορεί να λάβει μόνο τις τιμές 0 και 1 και να προσδιοριστεί από τις ακόλουθες τρεις λειτουργίες:

1. Ο δυαδικός σημαφόρος μπορεί να αρχικοποιηθεί λαμβάνοντας είτε την τιμή 0 είτε την τιμή 1.

2. Η λειτουργία `semWaitB` ελέγχει την τιμή του σημαφόρου. Αν η τιμή είναι ίση με μηδέν, τότε μπλοκάρεται η διεργασία που εκτελεί τη `semWaitB`. Αν η τιμή είναι ίση με ένα, τότε η τιμή τροποποιείται και λαμβάνει την τιμή μηδέν και η διεργασία συνεχίζει να εκτελείται.

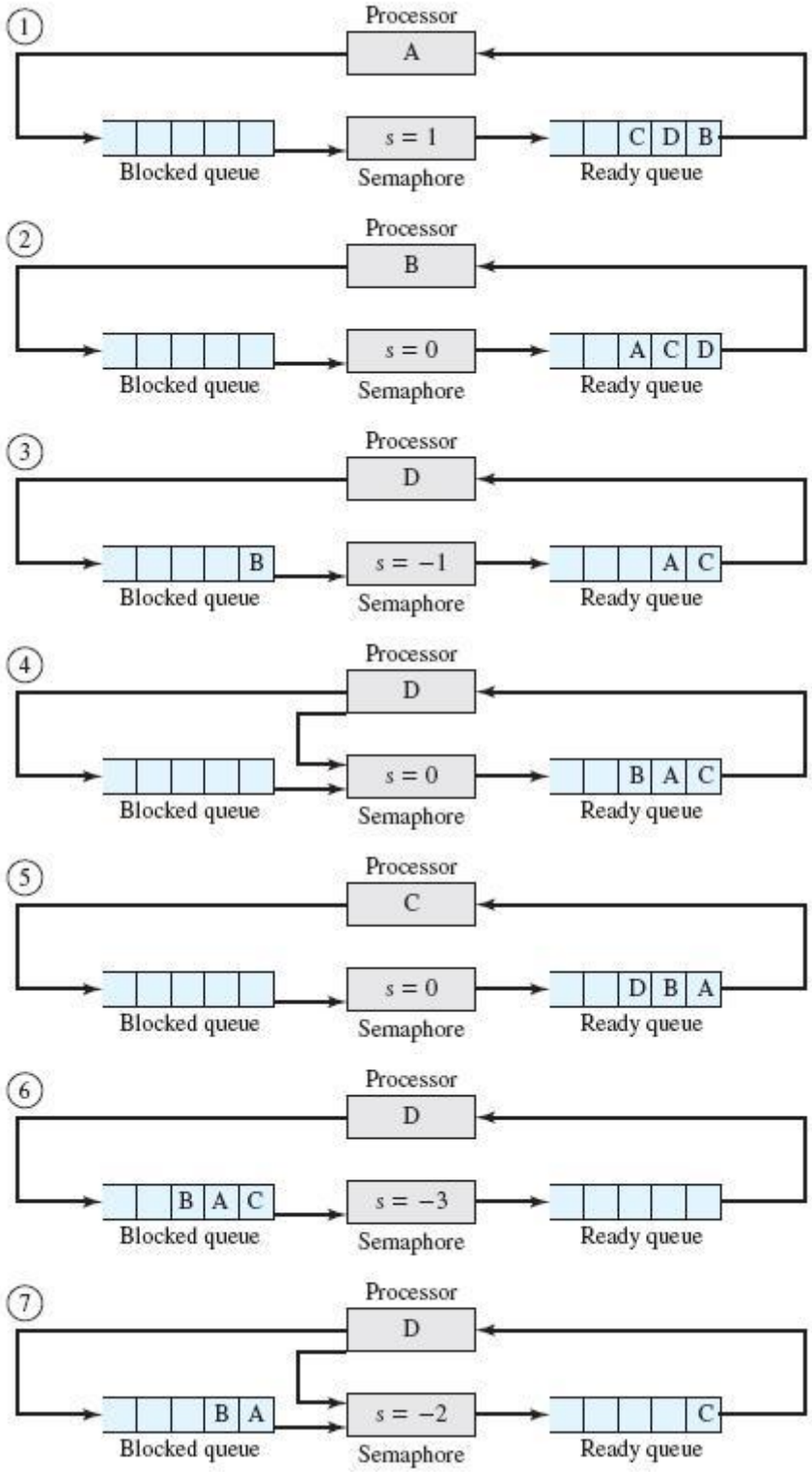
3. Η λειτουργία `semSignalB` κάνει έλεγχο για να διαπιστώσει αν υπάρχουν διεργασίες που είναι μπλοκαρισμένες στο σημαφόρο αυτό (η τιμή του σημαφόρου ισούται με μηδέν). Αν ισχύει αυτό, τότε μια διεργασία που έχει μπλοκαριστεί από μια λειτουργία `semWaitB` ξεμπλοκάρεται. Αν δεν υπάρχουν μπλοκαρισμένες διεργασίες, τότε η τιμή του σημαφόρου λαμβάνει την τιμή ένα.

Κατά κανόνα, είναι ευκολότερη η υλοποίηση του δυαδικού σημαφόρου και μπορεί να αποδειχτεί πως έχει την ίδια εκφραστική δύναμη με το γενικό σημαφόρο. Για να ξεχωρίζουν οι δύο τύποι σημαφόρων, ο μη δυαδικός σημαφόρος συχνά αναφέρεται είτε ως *σημαφόρος μέτρησης (counting semaphore)* είτε ως *γενικός σημαφόρος (general semaphore)*.

Μία έννοια που συνδέεται με το δυαδικό σημαφόρο είναι το *mutex*. Η βασική διαφορά μεταξύ των δύο είναι το γεγονός ότι η διεργασία που κλειδώνει το mutex (θέτει σε αυτό την τιμή 0) πρέπει να είναι και αυτή που θα το ξεκλειδώσει (θα θέσει σε αυτό την τιμή 1). Αντίθετα, είναι δυνατό κάποια διεργασία να κλειδώσει ένα δυαδικό σημαφόρο και κάποια άλλη να είναι αυτή που θα τον ξεκλειδώσει.

Τόσο για τους σημαφόρους μέτρησης όσο και για τους δυαδικούς σημαφόρους, χρησιμοποιείται ουρά για την διατήρηση των διεργασιών που αναμένουν το σημαφόρο. Η ερώτηση που προκύπτει αφορά τη σειρά με την οποία αφαιρούνται οι διεργασίες από μια τέτοια ουρά. Η πιο δίκαιη πολιτική απομάκρυνσης είναι η πρώτο-μέσα-πρώτο-έξω (FIFO): Η διεργασία που έχει μπλοκαριστεί το μεγαλύτερο χρονικό διάστημα απελευθερώνεται πρώτη από την ουρά. Οι σημαφόροι που διαθέτουν στον ορισμό τους την πολιτική αυτή ονομάζονται *ισχυροί σημαφόροι (strong semaphores)*. Οι σημαφόροι, που δεν καθορίζουν τη σειρά με την οποία αφαιρούνται οι διεργασίες από την ουρά, ονομάζονται *ασθενείς σημαφόροι (weak semaphores)*.

Το σχήμα 6.1, αποτελεί παράδειγμα της λειτουργίας των ισχυρών σημαφόρων. Εδώ οι διεργασίες A, B και C εξαρτώνται από ένα αποτέλεσμα της διεργασίας D. Αρχικά, (1) εκτελείται η A, οι B, C και D είναι έτοιμες και ο μετρητής του σημαφόρου είναι ίσος με 1, υποδηλώνοντας ότι ένα από τα αποτελέσματα της D είναι διαθέσιμο. Όταν η A εκδίδει μια εντολή `semWait` σε ένα σημαφόρο `s`, τότε μειώνεται η τιμή του σημαφόρου σε 0 και η A μπορεί να συνεχίσει να εκτελείται. Ακολούθως, επαναπροσχωρεί στην έτοιμη ουρά. Όταν εκτελείται η B (2), τελικά εκδίδει μια εντολή `semWait` και μπλοκάρεται, επιτρέποντας στην D να εκτελεστεί (3). Όταν η D ολοκληρώνει ένα νέο αποτέλεσμα, εκδίδει μια εντολή `semSignal`, η οποία επιτρέπει στην B να μετακινηθεί στην έτοιμη ουρά (4). Η D επαναπροσχωρεί στην έτοιμη ουρά και η C αρχίζει να εκτελείται, αλλά μπλοκάρεται όταν εκδίδει μια εντολή `semWait`. Παρομοίως, οι A και B εκτελούνται και μπλοκάρονται στο σημαφόρο, επιτρέποντας στην D να συνεχίσει την εκτέλεσή της (6). Όταν η D έχει κάποιο αποτέλεσμα, εκδίδει μια `semSignal`, η οποία μεταφέρει τη C στην έτοιμη ουρά. Κατοπινοί κύκλοι της D θα απελευθερώσουν την A και την B από την μπλοκαρισμένη κατάσταση.



Σχήμα 6.1 Παράδειγμα μηχανισμού σημαφόρου.

6.2.5 Αδιέξοδο

Ένα σύνολο διεργασιών βρίσκεται σε *αδιέξοδο* (*deadlock*) αν κάθε διεργασία του συνόλου περιμένει ένα γεγονός που μόνο μια άλλη διεργασία του συνόλου μπορεί να προκαλέσει. Τα αδιέξοδα δημιουργούνται ως αποτέλεσμα της διαχείρισης των πόρων από τις διαδικασίες. Κάθε *πόρος* (*resource*) μπορεί να είναι προεκχωρήσιμος (preemptable) δηλαδή να αποδεσμευτεί από μια διεργασία που τον κατέχει χωρίς παρενέργειες ή μη προεκχωρήσιμος (nonpreemptable). Παραδείγματα της πρώτης κατηγορίας είναι ο δίσκος και η μνήμη. Παραδείγματα της δεύτερης κατηγορίας είναι ο εκτυπωτής και η μονάδα ταινίας. Τυπικά μια διεργασία χρησιμοποιεί έναν πόρο ως εξής:

1. Ζητά τον πόρο από το Λ.Σ. (αίτηση χρήσης).
2. Χρησιμοποιεί τον πόρο.
3. Ενημερώνει το Λ.Σ. ότι δε χρειάζεται άλλο τον πόρο (αποδέσμευση).

Η διαδικασία της αίτησης και η αντιμετώπιση αιτήσεων που δεν μπορούν να καλυφθούν εξαρτώνται από το Λ.Σ.

Οι συνθήκες που πρέπει να ικανοποιούνται για να δημιουργηθεί αδιέξοδο είναι ο *αμοιβαίος αποκλεισμός* (κάθε πόρος είναι δεσμευμένος ή διαθέσιμος), η *δέσμευση και αναμονή* (διεργασίες που δεσμεύουν πόρους μπορούν να ζητούν και νέους), η *μη προεκχώρηση* (μόνο η διεργασία που έχει δεσμεύσει τους πόρους μπορεί να τους αποδεσμεύσει) και η *κυκλική αναμονή* (οι διαδικασίες που ζητούν πόρους πρέπει να σχηματίζουν κύκλο).

Οι παρακάτω στρατηγικές μπορούν να χρησιμοποιηθούν για την αντιμετώπισή τους:

- Αγνόηση του προβλήματος

- Ανίχνευση και επανόρθωση: Η ανίχνευση των αδιεξόδων μπορεί να γίνει με έρευνα του γράφου των αδιεξόδων για κύκλους. Αφού ανιχνευθεί το αδιέξοδο η επανόρθωση μπορεί να γίνει με τους παρακάτω τρόπους:

    1. Προεκχώρηση του πόρου.

2. Οπισθοδρόμηση της διεργασίας σε προηγούμενο σημείο ελέγχου (*checkpoint*).

3. Εξάλειψη κάποιων διεργασιών.

- Δυναμική αποφυγή με προσεκτική κατανομή των πόρων: Η αποφυγή του αδιεξόδου βασίζεται στη συντηρητική ικανοποίηση των αιτήσεων για πόρους με στόχο το σύστημα πάντα να βρίσκεται σε μία ασφαλή κατάσταση κατά την οποία δεν μπορεί να υπάρξει αδιέξοδο. Αιτήσεις που οδηγούν σε μη ασφαλείς καταστάσεις δεν ικανοποιούνται.

- Πρόληψη με αναίρεση των παραπάνω αναγκαίων συνθηκών: Η αναίρεση μιας τουλάχιστον από τις συνθήκες του αδιεξόδου εξασφαλίζει ότι δε θα υπάρχουν αδιέξοδα.

6.3 Συνοχή μνήμης cache σε πολυπύρηνους επεξεργαστές

Η *cache* είναι ένα μικρό και γρήγορο κομμάτι μνήμης που ενσωματώνεται στους επεξεργαστές, με σκοπό να αποθηκεύονται αντίγραφα από τα πιο πρόσφατα δεδομένα που ανακτήθηκαν από την κύρια μνήμη. Η μνήμες cache αποτέλεσαν τον βασικότερο παράγοντα αύξησης της απόδοσης και των ταχυτήτων στους μικροεπεξεργαστές. Χρησιμοποιώντας cache μειώνεται ο μέσος όρος του χρόνου που χρειάζεται ένας επεξεργαστής για πρόσβαση στη μνήμη, αφού πρώτα ελέγχει στην τοπική cache του για τα ζητούμενα δεδομένα και μόνο αν δε βρεθούν εκεί θα ψάξει στην κύρια μνήμη. Τα πράγματα ήταν σχετικά απλά στους μονοπύρηνους επεξεργαστές αφού υπήρχε μόνο ένας επεξεργαστής και μόνο μία cache. Με την δημιουργία όμως των πολυπύρηνων επεξεργαστών, τα πράγματα περιπλέχθηκαν αρκετά.

Το πρόβλημα εστιάζεται στην σωστή διάδοση και αποθήκευση των ορθών δεδομένων στις μνήμες των επεξεργαστών. Για παράδειγμα αν δυο επεξεργαστές P1 και P2 φορτώσουν την ίδια μεταβλητή από την κύρια μνήμη στην τοπική τους μνήμη (cache), και ο P1 αλλάξει την τιμή της, τότε οι δυο επεξεργαστές θα έχουν διαφορετική τιμή για την ίδια μεταβλητή. Ως αποτέλεσμα, ο P2 θα εκτελεί πράξεις πάνω σε λανθασμένα ή παλιά δεδομένα και άρα τα αποτελέσματα του θα είναι επίσης

λανθασμένα. Για να είναι ορθό ένα κύκλωμα με δυο ή περισσότερους επεξεργαστές, πρέπει με κάποιο τρόπο ο P1 να ενημερώσει τους υπόλοιπους για την αλλαγή που έκανε πάνω στην μεταβλητή του έτσι ώστε να διατηρηθεί η συνοχή μνήμης, δηλαδή όλοι οι επεξεργαστές να χρησιμοποιούν πάντα την τελευταία έκδοση των δεδομένων.

Για να έχουμε συνοχή μνήμης πρέπει να τηρούνται οι πιο κάτω κανόνες:

- Όταν υπάρχει εγγραφή ενός κομματιού μνήμης X από ένα επεξεργαστή P, η οποία ακολουθείται από μία ανάγνωση στο ίδιο κομμάτι μνήμης X από τον ίδιο επεξεργαστή P, χωρίς να υπάρχει οποιαδήποτε άλλη ενδιάμεση εγγραφή από άλλο επεξεργαστή στο κομμάτι X, τότε το X πρέπει να έχει και να επιστρέφει πάντα την τιμή που έγραψε ο P.

- Όταν υπάρχει ανάγνωση ενός κομματιού μνήμης X από ένα επεξεργαστή P1 η οποία γίνεται μετά την εγγραφή στο X από τον P2 και δεν υπάρχει οποιαδήποτε άλλη ενδιάμεση εγγραφή από άλλο επεξεργαστή, τότε η τιμή που θα διαβάζει ο P1 πρέπει να είναι η τιμή που έχει γράψει ο P2.

- Εγγραφές στο ίδιο κομμάτι μνήμης πρέπει να γίνονται με σειρά και να επιστρέφεται πάντα η τιμή της τελευταίας εγγραφής. Αν δηλαδή το κομμάτι μνήμης X έλαβε δυο τιμές A και B με σειρά, από δυο οποιουσδήποτε επεξεργαστές, τότε κανένας επεξεργαστής δεν μπορεί να διαβάσει το X σαν B και μετά A.

Ένας τρόπος με τον οποίο μπορούμε να επιβάλουμε και να ελέγχουμε την συνοχή της τοπικής μνήμης των επεξεργαστών είναι να συμπεριλάβουμε σε κάθε κομμάτι μνήμης ένα πεδίο, το οποίο θα κωδικοποιεί τα *δικαιώματα (permissions)* όλων των επεξεργαστών στο συγκεκριμένο κομμάτι. Αν δηλαδή ο επεξεργαστής P1 θέλει να κάνει ανάγνωση ή εγγραφή στη μεταβλητή X, πρέπει πρώτα να κοιτάξει στην τοπική του μνήμη και όταν βρει το X, να κοιτάξει αν το X έχει τα κατάλληλα δικαιώματα για ανάγνωση ή εγγραφή. Αν κάποιος επεξεργαστής επιθυμεί να γράψει σε κάποια μεταβλητή X η οποία βρίσκεται και στη μνήμη άλλων επεξεργαστών, τότε πρέπει να ανακτήσει πρώτα δικαίωμα για εγγραφή, αφαιρώντας οποιοδήποτε άλλο δικαίωμα για

ανάγνωση έχουν οι υπόλοιποι επεξεργαστές πάνω στην Χ. Γενικά, πρέπει πάντα να ισχύει η πιο κάτω αμετάβλητη συνθήκη για κάθε κομμάτι μνήμης:

"*Σε οποιαδήποτε χρονική στιγμή, τα δικαιώματα για κάποιο κομμάτι μνήμης πρέπει να επιτρέπουν είτε ανάγνωση από πολλούς, είτε εγγραφή από έναν και μόνο επεξεργαστή*."

Τα δικαιώματα που έχουν τα κομμάτια μνήμης απεικονίζονται με την κατάσταση του κάθε block. Τα περισσότερα πρωτόκολλα συνοχής μνήμης, χρησιμοποιούν τις καταστάσεις που φαίνονται στον Πίνακα 6.2, εξασφαλίζοντας ότι ισχύουν πάντα και οι αντίστοιχες αμετάβλητες συνθήκες.

Κατάσταση	Δικαιώματα	Αμετάβλητη Συνθήκη
Modified (M)	Ανάγνωση και Εγγραφή	Όλες οι άλλες cache είναι σε I ή NP
Exclusive (E)	Ανάγνωση και Εγγραφή	Όλες οι άλλες cache είναι σε I ή NP
Owned (O)	Ανάγνωση και Εγγραφή	Όλες οι άλλες cache είναι σε S, I ή NP
Shared (S)	Ανάγνωση	Καμιά άλλη cache δεν είναι σε M ή E
Invalid (I)	-	-
Not Present (NP)	-	-

Πίνακας 6.2 Καταστάσεις, δικαιώματα και αμετάβλητες συνθήκες σε πρωτόκολλα συνοχής μνήμης.

Για παράδειγμα ένας επεξεργαστής μπορεί να γράψει σε κάποιο κομμάτι μνήμης Χ αν η κατάσταση του κομματιού είναι Μ ή Ε, γιατί το πρωτόκολλο εξασφαλίζει ότι όλα τα αντίγραφα του Χ στις άλλες μνήμες βρίσκονται σε κατάσταση Ι ή NP. Ένας επεξεργαστής μπορεί να διαβάσει στο Χ αν η κατάσταση του είναι μία από {Μ, Ε, Ο, S}. Το πρωτόκολλο εξασφαλίζει ότι οι αμετάβλητες συνθήκες ισχύουν πάντοτε, μέσω ανταλλαγής κατάλληλων μηνυμάτων μεταξύ των επεξεργαστών ανάλογα με την ενέργεια που εκτελείται (ανάγνωση ή εγγραφή) και την κατάσταση στην οποία βρίσκεται το συγκεκριμένο κομμάτι μνήμης.

Σε περίπτωση που κάποιος επεξεργαστής δεν βρει την μεταβλητή που χρειάζεται για να διαβάσει (read miss) στην τοπική του μνήμη ή η κατάσταση της είναι Ι, τότε εκδίδει ένα αίτημα για ανάκτηση της μεταβλητής με δικαίωμα για ανάγνωση. Το πρωτόκολλο είναι υπεύθυνο να βρει την πιο πρόσφατη έκδοση της μεταβλητής, να

αφαιρέσει το δικαίωμα εγγραφής από τον επεξεργαστή που το κατέχει (αφήνοντας τον σε κατάσταση με δικαίωμα για ανάγνωση μόνο) και να τον υποχρεώσει να στείλει τη μεταβλητή στον επεξεργαστή που την χρειάζεται. Αν η μεταβλητή δεν υπάρχει σε κάποια cache ή υπάρχει αλλά δεν είναι σε κατάσταση {M, E, O, S} τότε η μεταβλητή ανακτάται από την μνήμη.

Στην αντίστοιχη περίπτωση που ο επεξεργαστής δεν βρει την μεταβλητή που χρειάζεται για να γράψει, τότε εκδίδει ένα αίτημα για ανάκτηση της μεταβλητής με δικαίωμα για εγγραφή. Το πρωτόκολλο ακολουθεί την ίδια διαδικασία που περιγράφτηκε πιο πάνω για ανάγνωση, αλλά εξασφαλίζει επίσης ότι αφαιρούνται και όλα τα δικαιώματα για ανάγνωση από τους υπόλοιπους επεξεργαστές.

Βιβλιογραφία

Η βιβλιογραφία παρουσιάζεται σε αλφαβητική σειρά κατά το αγγλοσαξονικό αλφάβητο.

[1] Stallings, W. *Λειτουργικά Συστήματα Αρχές Σχεδίασης*, Έκτη Έκδοση. Εκδόσεις Τζιόλα, 2009.

[2] Stallings, W. *Οργάνωση και Αρχιτεκτονική Υπολογιστών*, Έκτη Έκδοση. Εκδόσεις Τζιόλα, 2003.

[3] Χριστοδουλίδης, Α. Χ. *Μοντέλο-Έλεγχος Αλγορίθμου Συνοχής Μνήμης για Πολυπύρηνους Επεξεργαστές με το Εργαλείο SPIN*, Πανεπιστήμιο Κύπρου, Τμήμα Πληροφορικής, 2010.

[4] www.manpagez.com/man/3/Signal/

[5] www.wikipedia.org/