ΕΝΟΤΗΤΑ 6

ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ

Περιεχόμενα

1. Ο ρόλος του διαχειριστή μνήμης

- Το τμήμα του Λ.Σ. που διαχειρίζεται τη μνήμη λέγεται *Διαχειριστής Μνήμης* (Memory Manager). Καθήκον του είναι να παρακολουθεί ποια τμήματα της μνήμης είναι σε χρήση και ποια όχι, να χορηγεί μνήμη σε διεργασίες όποτε τη χρειάζονται και να την επαναχορηγεί σε άλλες όταν οι πρώτες ολοκληρώνουν την αποστολή τους, καθώς και να διαχειρίζεται την εναλλαγή πληροφοριών (swapping) μεταξύ κύριας μνήμης και δίσκου όταν η κύρια μνήμη δεν είναι αρκετή για να εξυπηρετηθούν όλες οι διεργασίες.

- Συγκεκριμένα, η διαχείριση μνήμης απαιτεί την ικανοποίηση των ακόλουθων αναγκών:

  Μετατόπιση (relocation). Αν μία διεργασία χρειασθεί να απομακρυνθεί από την κύρια μνήμη, δεν θα πρέπει να είναι αναγκαίο όταν επανέλθει να αποθηκευθεί στον ίδιο χώρο που καταλάμβανε προηγουμένως. Για να ικανοποιηθεί αυτή η ανάγκη θα πρέπει οι διευθύνσεις μνήμης που χρησιμοποιούν οι εντολές του προγράμματος να μην είναι απόλυτες αλλά σχετικές ως προς το περιεχόμενο κάποιου καταχωρητή.
  Προστασία (protection). Κάποια διεργασία να μην μπορεί να αναφερθεί σε θέση μνήμης που ανήκει σε άλλη διεργασία. Το πρόβλημα γίνεται πιο δύσκολο λόγω της μετατόπισης των διεργασιών στην κύρια μνήμη και ο έλεγχος πρέπει να γίνεται δυναμικά και όχι στατικά (την ώρα της μετάφρασης).
  Διαμοίρασμα (sharing). Περισσότερες από μία διεργασίες θα πρέπει να μπορούν να αναφέρονται στον ίδιο χώρο μνήμης (π.χ. όλες οι διεργασίες ενός προγράμματος θα πρέπει να μπορούν να αναφέρονται στο χώρο μνήμης στον οποίο βρίσκεται αποθηκευμένος ο κώδικας του προγράμματος.
  Λογική οργάνωση (logical organisation). Αν και η μνήμη (κύρια και περιφερειακή) είναι ουσιαστικά ένας μονοδιάστατος πίνακας που αποτελείται από μία σειρά από bytes, εν τούτοις σε λογικό επίπεδο θα πρέπει να υποστηρίζει τη λογική δομή ενός τυπικού προγράμματος: χωρισμός σε ενότητες (modules), κοινή χρήση κάποιων ενοτήτων από προγράμματα, κλπ.
  Φυσική οργάνωση (physical organisation). Μετατόπιση πληροφοριών μεταξύ κύριας και περιφερειακής μνήμης.

2. Διαχείρηση μνήμης όταν τα προγράμματα είναι εξ ολοκλήρου φορτωμένα στην κύρια μνήμη

- Τα συστήματα διαχείρησης μνήμης μπορούν να χωρισθούν σε δύο κατηγορίες: σε αυτά που μετακινούν διεργασίες μεταξύ κύριας μνήμης και δίσκου κατά τη διάρκεια εκτέλεσής τους και αυτά που δεν διαθέτουν τέτοιες τεχνικές. Ξεκινάμε με τη δεύτερη κατηγορία.

2.1 Σταθερά τμήματα μνήμης

- Με εξαίρεση το χώρο της μνήμης που χρειάζεται το Λ.Σ., ο υπόλοιπος χωρίζεται σε σταθερά τμήματα (fixed partitioning). Τα τμήματα αυτά μπορεί να είναι όλα του ιδίου μεγέθους ή ποικίλων μεγεθών.

- Στην περίπτωση του χωρισμού της μνήμης σε ισομεγέθη τμήματα, κάθε πρόγραμμα καταλαμβάνει ένα τέτοιο τμήμα. Στην περίπτωση που κάποιο πρόγραμμα είναι μεγαλύτερο από το μέγεθος των τμημάτων θα πρέπει να χωρισθεί σε *τμήματα επικάλυψης* (overlays) και να μεταφέρεται στο χώρο της κύριας μνήμης που ανήκει στο τμήμα εκείνο του προγράμματος που πρέπει να εκτελεσθεί, επικαλύπτοντας το προηγούμενο τμήμα τού προγράμματος. Εδώ γίνεται προσπάθεια να χωρισθεί ένα πρόγραμμα σε μέρη ανεξάρτητα (κατά το δυνατόν) μεταξύ τους: π.χ. ένα τμήμα για είσοδο δεδομένων, ένα για την επεξεργασία τους και ακόμα ένα για την έξοδο των αποτελεσμάτων. Η δουλειά αυτή πρέπει να γίνει από τον προγραμματιστή και είναι βασανιστική, βαρετή και χρονοβόρα.

- Επίσης, η μέθοδος αυτή οδηγεί σε σπατάλη μνήμης στην (όχι σπάνια) περίπτωση που κάποιο πρόγραμμα είναι μικρότερο από το μέγεθος των τμημάτων. Αυτό το φαινόμενο ονομάζεται *εσωτερικός κατακερματισμός* (internal fragmentation).

2.1 Σταθερά τμήματα μνήμης (συνέχεια)

- Η σπατάλη αυτή της μνήμης μπορεί να μειωθεί αν η κύρια μνήμη χωρισθεί σε τμήματα ποικίλων μεγεθών. Σε αυτή την περίπτωση όμως τίθεται το ερώτημα με ποιο τρόπο διαμοιράζουμε τα διαθέσιμα τμήματα στα προγράμματα. Υπάρχουν δύο εναλλακτικές λύσεις:
  - i) Κάνοντας χρήση πολλαπλών ουρών αναθέτουμε στο κάθε πρόγραμμα το μικρότερο σε μέγεθος τμήμα που ικανοποιεί τις ανάγκες του. Αυτό προϋποθέτει όμως ότι κάποιο πρόγραμμα γνωρίζει εκ των προτέρων τις ανάγκες του σε μνήμη. Επίσης, μπορεί να ελαχιστοποιείται η σπατάλη μνήμης μέσα σε ένα τμήμα αλλά συνολικά υπάρχει ακόμα σπατάλη γιατί μπορεί κάποιο τμήμα να παραμένει αχρησιμοποίητο αν δεν υπάρχει κάποιο πρόγραμμα αρκετά μεγάλο για να δικαιολογείται η ανάθεσή του σε αυτό.
  - ii) Κάνοντας χρήση μίας ουράς, αναθέτουμε σε κάποιο πρόγραμμα το μικρότερο διαθέσιμο τμήμα. Αν δεν υπάρχει διαθέσιμο τμήμα, κάποιο από τα προγράμματα που βρίσκονται στην κύρια μνήμη θα πρέπει να μεταφερθεί στο δίσκο (κατά προτίμηση κάποιο που βρίσκεται σε τμήμα του μεγέθους που χρειάζεται το νέο πρόγραμμα για να τρέξει, αλλά μπορεί να ληφθούν υπ' όψη εδώ και άλλα κριτήρια όπως προτεραιότητες, κατάσταση διεργασιών, κλπ.).

- Η μέθοδος του διαμοιράσματος της μνήμης σε σταθερά τμήματα, αν και σχετικά απλή σε υλοποίηση, έχει μερικά σημαντικά μειονεκτήματα:
  - Ο αριθμός των εν ενεργεία διεργασιών στο σύστημα περιορίζεται στον αριθμό των τμημάτων.
  - Επειδή στα περισσότερα συστήματα οι ανάγκες σε μνήμη μίας τυπικής εργασίας δεν μπορούν να προκαθορισθούν, ο χωρισμός της μνήμης σε τμήματα διαφορετικού μεγέθους δεν μπορεί να γίνει αποτελεσματικά και σπαταλάται σημαντική ποσότητα μνήμης.

- Η χρήση της μεθόδου αυτής δεν συναντάται πλέον στις μέρες μας. Ιστορικά, ένα από τα πιο επιτυχημένα Λ.Σ. που χρησιμοποιούσαν αυτή τη μέθοδο είναι το OS/MFT (Multiprogramming with a Fixed number of Tasks) της IBM.

2.2 Μεταβλητά τμήματα μνήμης

- Μερικά από τα προβλήματα της χρήσης σταθερών τμημάτων επιλύονται με τη χρήση μεταβλητών τμημάτων. Εδώ, σε κάθε διεργασία εκχωρείται η ποσότητα της μνήμης που χρειάζεται, εφ' όσον βεβαίως είναι διαθέσιμη. Κάποια στιγμή, κάποια από τις διεργασίες που βρίσκονται στην κύρια μνήμη θα αποδεσμεύσει τη μνήμη που χρησιμοποιούσε (είτε γιατί τερμάτισε είτε γιατί θα μεταφερθεί στο δίσκο). Τον χώρο που απελευθερώνεται μπορεί να καταλάβει μία *ή περισσότερες* διεργασίες ανάλογα με τις ανάγκες τους. Αντίθετα με την περίπτωση των σταθερών τμημάτων, εδώ ο αριθμός, η θέση και το μέγεθος των τμημάτων καθώς επίσης και ο αριθμός των διεργασιών στην κύρια μνήμη είναι μεταβλητός.

- Το πλεονέκτημα της μεθόδου αυτής είναι ότι δεν εγκλωβιζόμαστε σε σταθερό αριθμό τμημάτων και αυξάνεται έτσι η απόδοση της μνήμης. Από την άλλη πλευρά όμως ο μηχανισμός δέσμευσης και αποδέσμευσης της μνήμης και ο μηχανισμός παρακολούθησης των μεταβολών που λαμβάνουν χώρα γίνεται πιο πολύπλοκος.

- Ένα άλλο πρόβλημα που δημιουργείται είναι ότι κάποια στιγμή θα υπάρχουν στη μνήμη πολλά μικρά κενά τμήματα τα οποία αν και συνδυασμένα θα μπορούσαν να ικανοποιήσουν τις ανάγκες κάποιας διεργασίας, από μόνα τους είναι άχρηστα. Αυτό το φαινόμενο ονομάζεται *εξωτερικός κατακερματισμός* (external fragmentation). Ο Knuth υπολόγισε ότι σε γενικές γραμμές τα κενά h θα είναι κατά μέσο όρο τα μισά από ότι οι διεργασίες n (δηλαδή h=n/2). Το αποτέλεσμα αυτό είναι γνωστό σαν *ο κανόνας του 50%* (fifty percent rule) και βασίζεται στην παρατήρηση ότι εφαπτόμενα κενά συνενώνονται σε ένα.

- Μία πιθανή λύση θα ήταν η περιοδική μετακίνηση από το Λ.Σ. όλων των τμημάτων έτσι ώστε να βρίσκονται σε συνεχόμενες θέσεις και επομένως οι ελεύθεροι χώροι να είναι και αυτοί μαζί σαν ένα τμήμα. Αυτή η μέθοδος ονομάζεται *συμπίεση μνήμης* (compaction) αλλά χρησιμοποιείται σπάνια γιατί είναι πολύ δαπανηρή σε χρόνο ΚΜΕ. Π.χ. σε μία μηχανή με 1 MB μνήμη και με δυνατότητα μεταφοράς 1 byte/msec (δηλαδή 1 MB / sec) θα χρειαζόταν ένα δευτερόλεπτο για τη συμπίεση όλης της μνήμης.

2.3 Αλγόριθμοι τοποθέτησης

- Για την επιλογή του κατάλληλου τμήματος μνήμης για τις ανάγκες μίας διεργασίας χρησιμοποιείται ένας από τους ακόλουθους *αλγόριθμους τοποθέτησης* (placement algorithms). Ο κοινός παρανομαστής για όλους είναι η προσπάθεια εύρεσης ενός τμήματος μνήμης ίσου ή μεγαλύτερου αυτού που κάποια διεργασία έχει ανάγκη με τρόπο που να ελαχιστοποιεί τη σπατάλη μνήμης αλλά και τη συχνότητα χρήσης τεχνικών συμπίεσης.

- Ο αλγόριθμος της *πρώτης τοποθέτησης* (first-fit) σαρώνει τη μνήμη και διαλέγει το πρώτο τμήμα που θα βρει και είναι κατάλληλο για τις ανάγκες του προγράμματος. Αν το τμήμα αυτό είναι μεγαλύτερο από όσο χρειάζεται το διασπά σε δύο κομμάτια. Είναι ο απλούστερος από τους αλγόριθμους, γρήγορος στην εκτέλεση και παραδόξως έχει συνήθως καλά αποτελέσματα.

- Μία παραλλαγή του προηγούμενου αλγόριθμου είναι αυτός της *επόμενης τοποθέτησης* (next-fit) όπου η αναζήτηση ενός κατάλληλου τμήματος ξεκινάει από το σημείο που βρέθηκε το τελευταίο κατάλληλο τμήμα αντί από την αρχή της μνήμης. Μελέτες προσομοίωσης από τον Bays έδειξαν ότι ο αλγόριθμος αυτός έχει ελαφρώς χειρότερα αποτελέσματα από αυτόν της πρώτης τοποθέτησης. Τείνει να χορηγεί μνήμη από το μεγαλύτερο διαθέσιμο κομμάτι που βρίσκεται προς το τέλος της μνήμης (υψηλότερες διευθύνσεις). Επομένως η περιοχή εκείνη της μνήμης σπάει γρήγορα σε μικρά και άχρηστα τμήματα, δημιουργώντας την ανάγκη χρήσης τεχνικών συμπίεσης. Παρεμπιπτόντως, το αντίθετο συμβαίνει με τον προηγούμενο αλγόριθμο ο οποίος δημιουργεί το ίδιο πρόβλημα στις χαμηλές θέσεις μνήμης.

- Ο αλγόριθμος της *καλύτερης τοποθέτησης* (best-fit) προσπαθεί να βρει το τμήμα εκείνο της μνήμης που είναι όσο γίνεται πιο μικρό (αλλά βεβαίως να εξακολουθεί να ικανοποιεί τις ανάγκες του προγράμματος). Αν και φαίνεται λογική αυτή η προσέγγιση γιατί προσπαθεί να ελαχιστοποιήσει τα άχρηστα κενά, εν τούτοις δημιουργεί μεγαλύτερο πρόβλημα γιατί τα κενά που αναπόφευκτα θα δημιουργηθούν είναι πολύ μικρά και άχρηστα ακόμα και για μικρά προγράμματα. Επίσης είναι πιο αργός από τους προηγούμενους λόγω της σάρωσης όλης της μνήμης.

2.3 Αλγόριθμοι τοποθέτησης (συνέχεια)

- Ο αλγόριθμος της *χειρότερης τοποθέτησης* (worst-fit) κάνει ακριβώς το αντίθετο ψάχοντας να βρεί το μεγαλύτερο τμήμα έτσι ώστε όταν διασπασθεί το υπόλοιπο κομμάτι να είναι αρκετά μεγάλο για να παραμείνει χρήσιμο. Προγράμματα προσομοίωσης έδειξαν ότι δεν είναι πολύ αποτελεσματικός.

- Τέλος, ο αλγόριθμος της *γρήγορης τοποθέτησης* (quick-fit) αποθηκεύει τα πιο συνηθισμένα μεγέθη μνήμης που ζητούνται. Αν και η ζήτηση ενός τμήματος προκαθορισμένου μεγέθους είναι πολύ γρήγορη, εν τούτοις ο αλγόριθμος υποφέρει από τα ίδια προβλήματα με τους υπόλοιπους.

2.4 Αλγόριθμοι αντικατάστασης

- Ακόμα και με τη χρήση της τεχνικής της συμπίεσης, θα φτάσει κάποια στιγμή όπου όλη η μνήμη θα είναι δεσμευμένη από διεργασίες που βρίσκονται σε κατάσταση αναστολής. Σε αυτή την περίπτωση το Λ.Σ. είναι υποχρεωμένο να μεταφέρει κάποιες από αυτές τις διεργασίες στην περιφερειακή μνήμη και να τις αντικαταστήσει με άλλες από εκεί, που είναι έτοιμες για εκτέλεση. Οι πολιτικές με βάση τις οποίες το Λ.Σ. αποφασίζει πως θα γίνονται τέτοιου είδους αντικαταστάσεις αποτελούν τους αλγόριθμους αντικατάστασης και θα εξετασθούν σε συνάρτηση με την τεχνική της ιδεατής μνήμης παρακάτω.

2.5 Διαχείρηση Μνήμης με το Σύστημα των Φίλων

- Επινοήθηκε από τους Knuth και Knowlton, σαν μία μέση λύση μεταξύ των τεχνικών των σταθερών και μεταβλητών τμημάτων. Η μνήμη μπορεί να διαιρεθεί σε μπλοκ μεγέθους 2^N, δηλαδή μεγέθους 1, 2, 4, 8, 16 … bytes μέχρι το συνολικό μέγεθός της. Δηλαδή, αν η μνήμη έχει μέγεθος 1 MB, μπορούν να υπάρχουν 21 τέτοια μπλοκ, από 1 byte μέχρι 1 MB. Στην αρχή, υπάρχει μόνο ένα μπλοκ μεγέθους 1 MB.

2.5 Διαχείρηση Μνήμης με το Σύστημα των Φίλων (συνέχεια)

- Αν τώρα η πρώτη αίτηση για μνήμη είναι των 100K, τότε το πιο κοντινό στις ανάγκες της αίτησης αυτής μέγεθος μπλοκ είναι 128K. Επομένως, το αρχικό μπλοκ του 1 MB σπάει σε δύο “φίλους” των 512K, ένα από αυτά σπάει περαιτέρω σε δύο άλλους “φίλους” των 128K και ένα από αυτά τα τελευταία χρησιμοποιείται για την αίτηση. Αν η δεύτερη αίτηση έχει ανάγκη σε 256K, παίρνει αμέσως το υπάρχων μπλοκ αυτού του μεγέθους. Η ευελιξία της μεθόδου αυτής βρίσκεται στο γεγονός ότι αν ολοκληρώσουν την εκτέλεσή τους δύο διεργασίες που καταλαμβάνουν γειτονικούς χώρους, τα δύο αντίστοιχα φιλικά μπλοκ συνενώνονται σε ένα, επιτρέποντας έτσι την εξυπηρέτηση επιπλέων αιτήσεων.

2.6 Μηχανισμός μετατόπισης

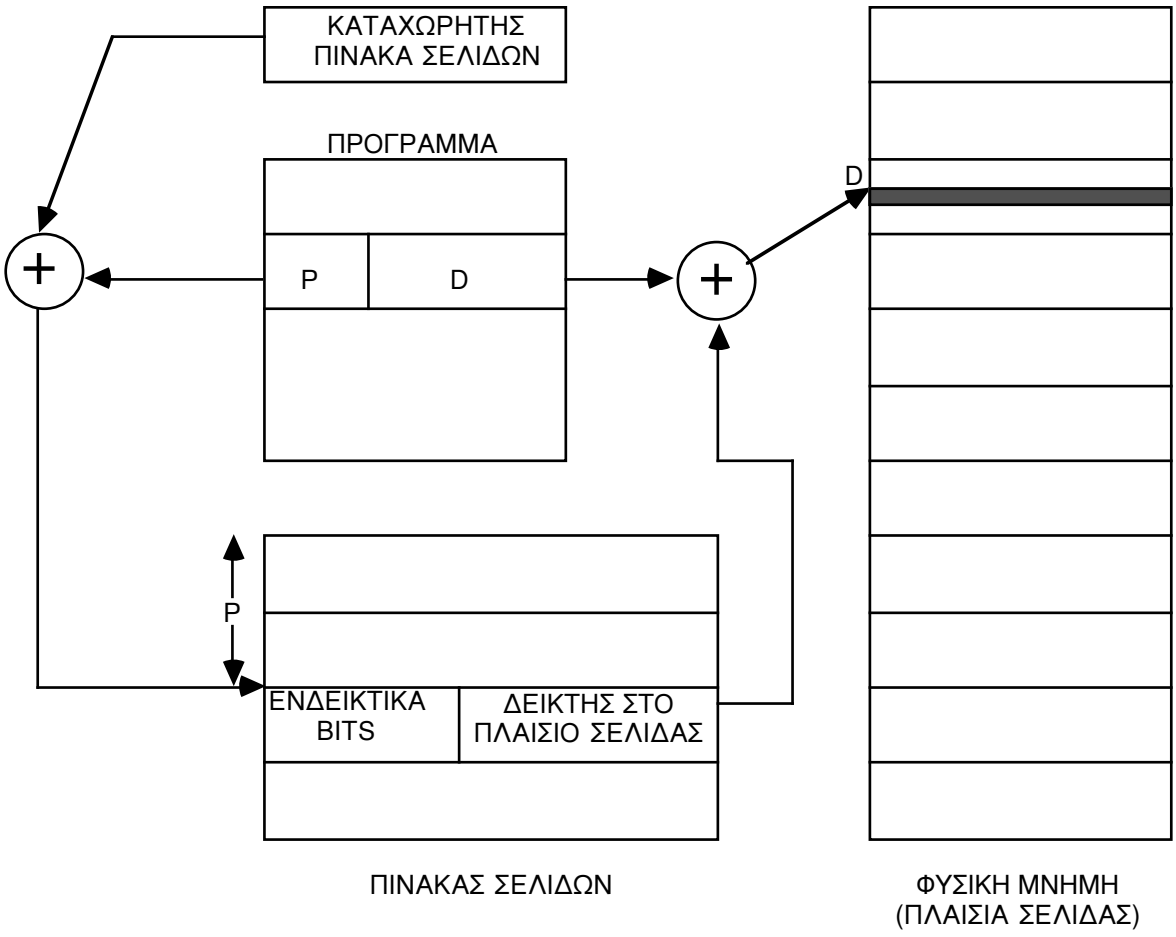
- Για τους περισσότερους από τους μηχανισμούς που εξετάσαμε μέχρι τώρα ισχύει ότι το τμήμα μνήμης που θα καταλάβει μία διεργασία στην κύρια μνήμη δεν είναι κατ’ ανάγκη το ίδιο με αυτό που καταλάμβανε την προηγούμενη φορά που είχε εισέλθει στην κύρια μνήμη. Επίσης, κατά τη συμπίεση της μνήμης, μία διεργασία τυχόν να μετακινηθεί σε άλλο χώρο. Θα πρέπει λοιπόν οι διευθύνσεις μνήμης που χρησιμοποιούν οι εντολές του προγράμματος να αλλάζουν ανάλογα.

- Η τεχνική που συνήθως ακολουθείται (και η οποία ικανοποιεί και τις ανάγκες της προστασίας) είναι να υπάρχουν δύο ειδικοί καταχωρητές, ο *καταχωρητής βάσης* (base register) και ο *καταχωρητής ορίου* (limit, bounds register). Ο πρώτος έχει ως τιμή τη διεύθυνση της αρχής του τμήματος ενώ ο δεύτερος το μέγεθος (ή το τέλος) του τμήματος. Σε κάθε διεύθυνση των εντολών της διεργασίας προστίθεται αυτόματα η τιμή του καταχωρητή βάσης και η καινούργια διεύθυνση ελέγχεται με βάση την τιμή του καταχωρητή ορίου για να διαπιστωθεί αν ανήκει στα επιτρεπτά όρια μνήμης που ελέγχει η διεργασία. Έτσι αν ο καταχωρητής βάσης έχει τιμή 130K και το πρόγραμμα έχει την εντολή CALL 100, τότε αυτή μετατρέπεται σε CALL 130K+100 χωρίς να τροποποιηθεί η ίδια η εντολή του προγράμματος.

2.7 Σελιδοποίηση

- Η κύρια μνήμη χωρίζεται σε ισομεγέθη μικρά κομμάτια που λέγονται *πλαίσια σελίδας* (page frames). Κάθε πρόγραμμα χωρίζεται σε ένα αριθμό από σελίδες (όσες χρειασθούν ανάλογα με το πόσο μεγάλο είναι) οι οποίες έχουν ακριβώς το ίδιο μέγεθος με τα πλαίσια σελίδας της κύριας μνήμης. Όταν ένα πρόγραμμα φορτώνεται στην κύρια μνήμη, όλα τα μέρη του φορτώνονται σε αντίστοιχο αριθμό από πλαίσια.

- Για κάθε πρόγραμμα φορτωμένο στην κύρια μνήμη δημιουργείται ένας πίνακας σελίδων που δείχνει ποια πλαίσια καταλαμβάνονται από το πρόγραμμα. Τα πλαίσια αυτά δεν είναι απαραίτητο να είναι συνεχόμενα. Κάθε εγγραφή του πίνακα αποτελείται από μία ομάδα από bits πληροφοριών (π.χ. προστασίας, κλπ.) και ένα δείκτη που δείχνει τη θέση μνήμης από όπου ξεκινάει η αποθήκευση της σελίδας. Ένας καταχωρητής δείχνει τη θέση μνήμης από όπου ξεκινάει η αποθήκευση του πίνακα.



2.7 Σελιδοποίηση (συνέχεια)

- Κάθε διεύθυνση μνήμης M του προγράμματος είναι ένα ζευγάρι M=(P,D) όπου P είναι ο αριθμός της αντίστοιχης σελίδας και D η μετατόπιση (displacement) μέσα στη σελίδα. Για να αποκωδικοποιηθεί μία διεύθυνση μνήμης βρίσκεται πρώτα ο αντίστοιχος πίνακας με βάση την τιμή του σχετικού καταχωρητή. Η Ροστή εγγραφή παρέχει τον αριθμό του πλαισίου στον οποίο είναι αποθηκευμένη η σελίδα και η πληροφορία D τη συγκεκριμένη θέση μνήμης η οποία αντιστοιχεί στη M.

- Το πλεονέκτημα της σελιδοποίησης είναι ότι μειώνει στο ελάχιστο δυνατόν τον κατακερματισμό της μνήμης: εξωτερικός κατακερματισμός δεν υπάρχει καθόλου ενώ ο εσωτερικός κατακερματισμός περιορίζεται στην τελευταία σελίδα ενός προγράμματος. Αυτή η απλής μορφής σελιδοποίηση μπορεί να θεωρηθεί ότι είναι παραλλαγή της μεθόδου των σταθερών τμημάτων χωρίς τον περιορισμό να είναι τα τμήματα σε συνεχόμενες θέσεις μνήμης.

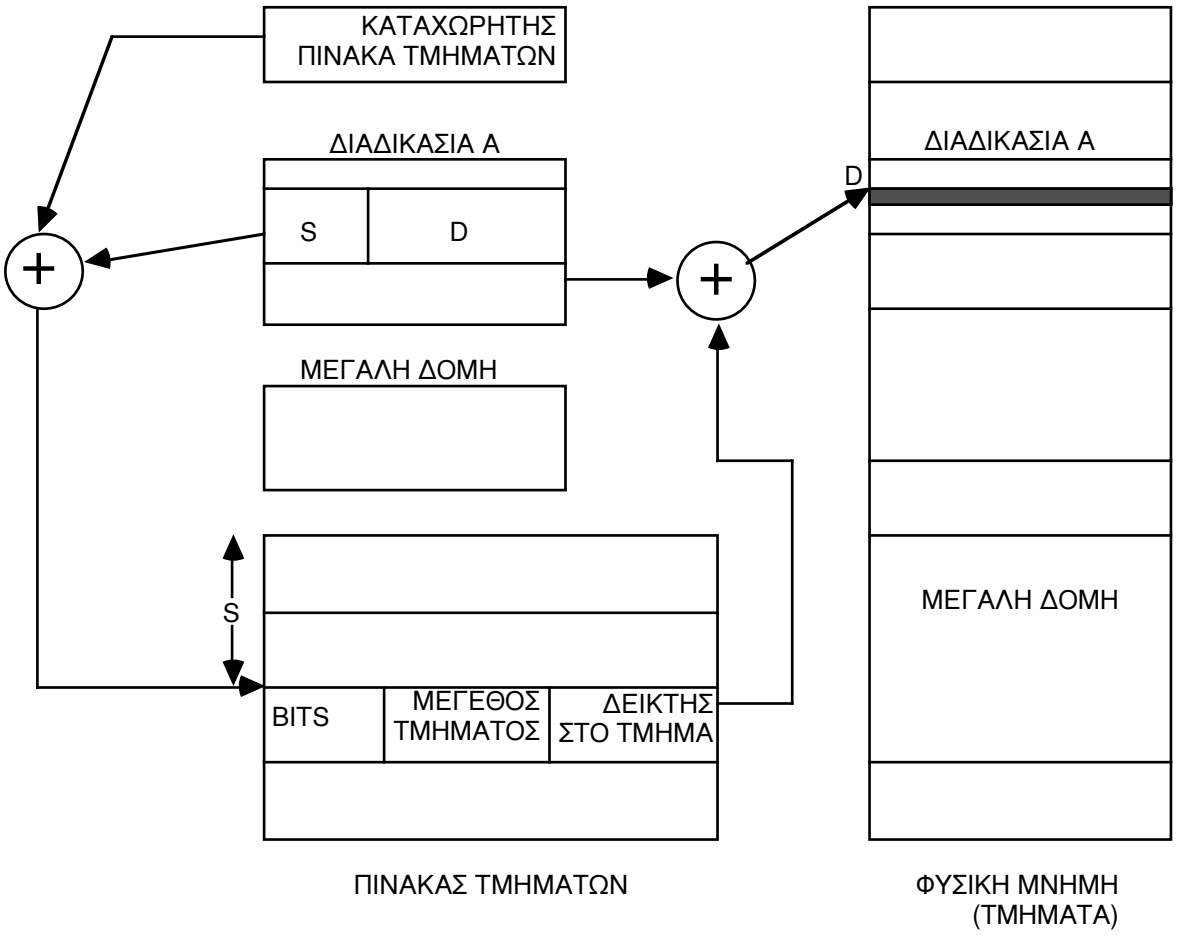
- Όμως για την αποκωδικοποίηση μίας διεύθυνσης μνήμης χρειάζονται δύο προσπελάσεις, μία για τον πίνακα σελίδων και μία για την ίδια τη σελίδα. Το πρόβλημα αυτό μπορεί να επιλυθεί με τη χρήση *συσχετιστικών καταχωρητών* όπως θα δούμε παρακάτω.

- Ένα άλλο ερώτημα είναι ποιο πρέπει να είναι το μέγεθος της σελίδας. Αν είναι σχετικά μικρό ελαττώνει τον εσωτερικό κατακερματισμό της τελευταίας σελίδας ενός προγράμματος αλλά αυξάνει το μέγεθος του πίνακα σελίδων (και κατ' επέκταση τον χρόνο προσπέλασής του και τον χώρο μνήμης που χρειάζεται για αποθήκευση). Αν είναι σχετικά μεγάλος αντιστρέφεται το πρόβλημα. Για να είναι όσο γίνεται πιο εύκολη η αποκωδικοποίηση μίας διεύθυνσης μνήμης το μέγεθος της σελίδας πρέπει να είναι δύναμη του 2 και συνήθως κυμαίνεται μεταξύ 1K και 4K.

2.8 Κατάτμηση

- Αν η σελιδοποίηση μπορεί να θεωρηθεί σαν παραλλαγή της μεθόδου των σταθερών τμημάτων χωρίς τον περιορισμό να είναι τα τμήματα φορτωμένα στη μνήμη σε συνεχόμενες θέσεις, η κατάτμηση μπορεί να θεωρηθεί η αντίστοιχη παραλλαγή της μεθόδου των μεταβλητών τμημάτων μνήμης.

- Οι δομές που δημιουργούνται είναι λίγο πολύ οι ίδιες με αυτές της σελιδοποίησης αλλά ο πίνακας τμημάτων πρέπει να έχει την επί πλέον πληροφορία του μεγέθους του τμήματος. Λόγω του διαφορετικού μεγέθους των τμημάτων δεν υπάρχει εσωτερικός κατακερματισμός αλλά παραμένει το πρόβλημα του εξωτερικού κατακερματισμού σε μικρότερο όμως βαθμό από ότι στη μέθοδο των μεταβλητών τμημάτων μνήμης.



2.8 Κατάτμηση (συνέχεια)

- Η βασική ιδέα στην κατάτμηση είναι ότι ένα πρόγραμμα χωρίζεται συνήθως σε λογικές ενότητες (διαδικασίες, στοίβες, δομημένος προγραμματισμός, συνολική δήλωση των δομών δεδομένων σε συγκεκριμένα σημεία) και κάθε ενότητα αντιστοιχεί σε ένα τμήμα. Μάλιστα δε, είναι ο μεταφραστής αυτός που καθορίζει πως ένα πρόγραμμα θα χωρισθεί σε τμήματα.

- Το μεγάλο πλεονέκτημα είναι ότι μια και ένα τμήμα είναι τώρα μία λογική οντότητα (διαδικασία, δομή δεδομένων, ομάδα από μεταβλητές) γίνεται πιο εύκολος ο έλεγχος και η διαχείρηση των τμημάτων. Π.χ. αν επαναμεταγλωττισθεί μία μόνο διαδικασία ενός προγράμματος, τα τμήματα για το υπόλοιπο πρόγραμμα δεν χρειάζεται να αλλάξουν. Επίσης κάποια τμήματα μπορεί να δηλωθούν μόνο για διάβασμα και κατ' επέκταση να διαμοιράζονται μεταξύ διαφόρων προγραμμάτων. Σημειωτέον ότι ένα πρόγραμμα μπορεί να καταλάβει (μεταβλητού μεγέθους) τμήματα τα οποία να μην είναι σε συνεχόμενες θέσεις στη μνήμη.

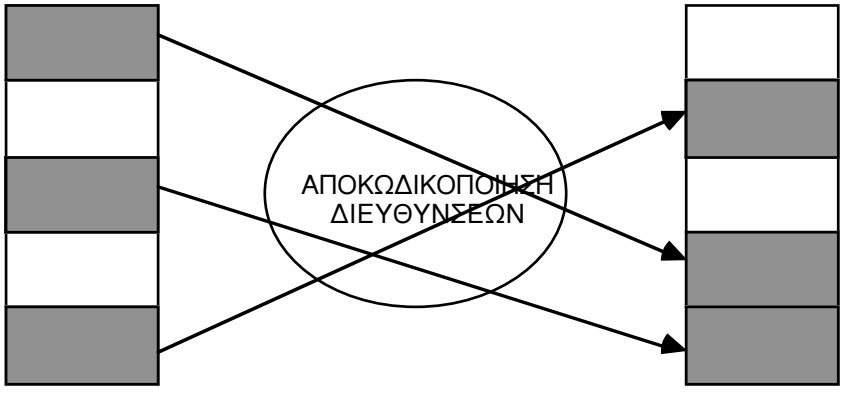
- Όμως ο προγραμματιστής πρέπει να είναι ενήμερος της τεχνικής αυτής και να βοηθάει με την οϊοθέτηση κατάλληλης μεθοδολογίας προγραμματισμού (δόμηση των δεδομένων και εντολών, χρήση διαδικασιών και αρθρωτού προγραμματισμού (modular programming), κλπ.).

3. Ιδεατή μνήμη

- Οι τεχνικές σελιδοποίησης και κατάτμησης που εξετάστηκαν έχουν δύο βασικά χαρακτηριστικά:
  - — Οι αναφορές σε διευθύνσεις μνήμης είναι σχετικές και αποκωδικοποιούνται δυναμικά για να δώσουν τις αντίστοιχες απόλυτες τιμές τους.
  - — Τα μέρη από τα οποία αποτελείται ένα πρόγραμμα (είτε είναι σελίδες είτε είναι τμήματα) δεν χρειάζεται να βρίσκονται στη μνήμη σε συνεχόμενες θέσεις.

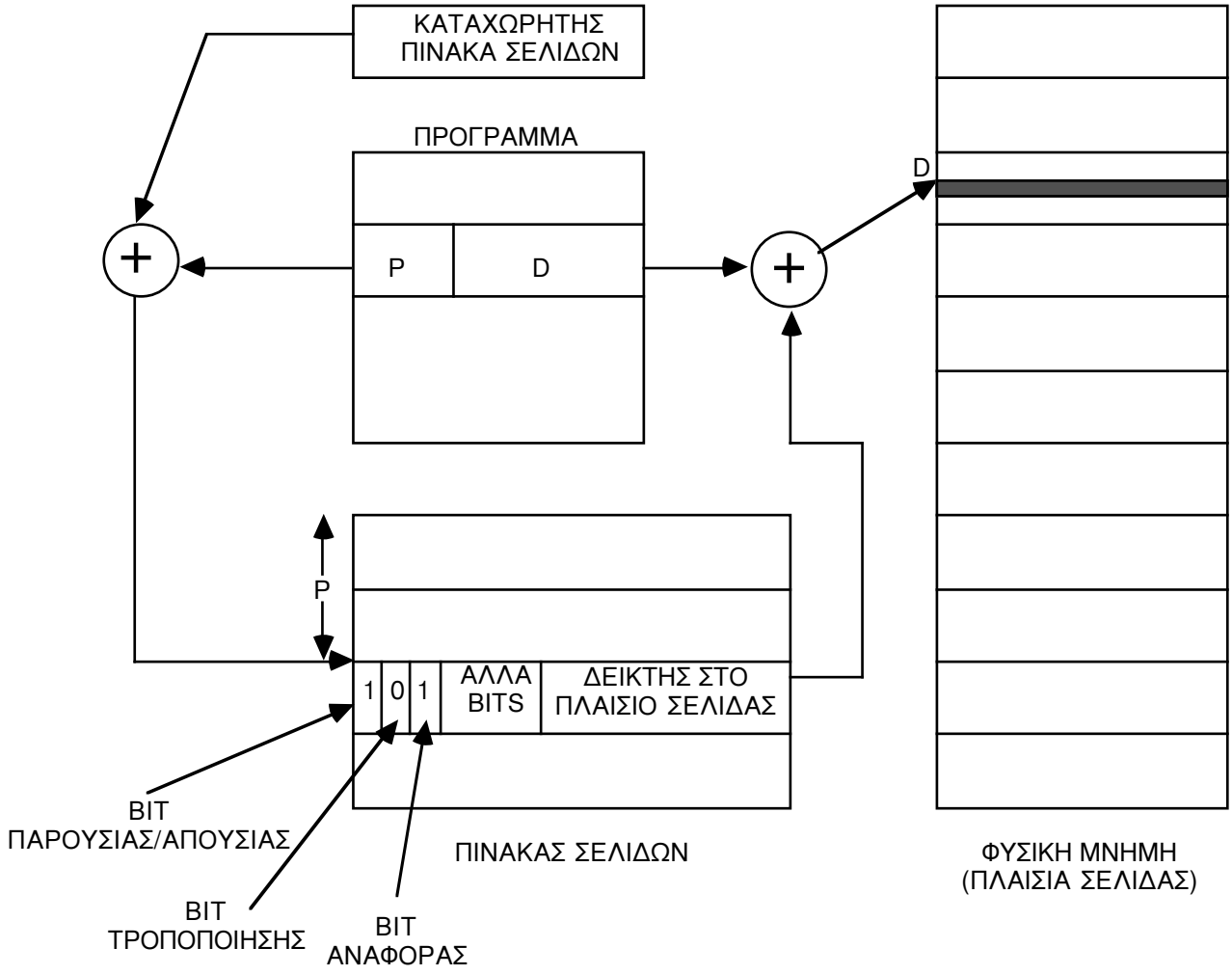
- Η βασική ιδέα που οδήγησε στην ανάπτυξη της *ιδεατής μνήμης* (virtual memory) και ανήκει στον Fotheringham (1961) βασίζεται στην παρατήρηση ότι δεν είναι αναγκαίο όλα τα μέρη ενός προγράμματος να βρίσκονται στην κύρια μνήμη την ίδια στιγμή. Το σημαντικό πλεονέκτημα αυτής της ιδέας είναι ότι μπορεί να χρησιμοποιηθεί η περιφερειακή μνήμη (που είναι πολύ μεγαλύτερης χωρητικότητας από την κύρια αλλά και πιο φθηνή) για την αποθήκευση των περισσοτέρων από τα μέρη ενός προγράμματος που εκτελείται και την εναλλαγή στην κύρια μνήμη μόνο εκείνων των μερών που χρειάζονται ανά πάσα στιγμή. Έτσι η ιδεατή μνήμη ενός προγράμματος μπορεί να είναι μεγαλύτερη από την κύρια μνήμη. Επίσης μια και μόνο ένα μέρος κάθε προγράμματος που εκτελείται χρειάζεται να βρίσκεται στην κύρια μνήμη, αυξάνεται ο βαθμός πολυπρογραμματισμού του συστήματος.

- Η αποτελεσματικότητα της ιδεατής μνήμης βασίζεται στην *τοπικότητα των αναφορών* που κάνει ένα πρόγραμμα κατά χρονικά διαστήματα.



3.1 Σελιδοποίηση με χρήση ιδεατής μνήμης

- Ο μηχανισμός παραμένει λίγο πολύ ο ίδιος άλλα τώρα θα πρέπει να ληφθεί υπ' όψη ότι οι σελίδες εναλλάσονται μεταξύ κύριας και περιφερειακής μνήμης. Έτσι κάθε εγγραφή στον πίνακα σελίδων έχει επιπλέον πληροφορίες ελέγχου:
  - Το *bit παρουσίας/απουσίας* δηλώνει αν η σελίδα βρίσκεται στην κύρια ή στην περιφερειακή μνήμη.
  - Το *bit τροποποίησης* (modified bit) δηλώνει αν η σελίδα έχει υποστεί αλλαγές από τότε που εισήλθε στην κύρια μνήμη.
  - Το *bit αναφοράς* (referenced bit) δηλώνει αν η σελίδα έχει χρησιμοποιηθεί από το πρόγραμμα για γράψιμο ή για διάβασμα.
  - Τέλος, υπάρχουν άλλα bits ελέγχου όπως *bits προστασίας*, κλπ.



3.1 Σελιδοποίηση με χρήση ιδεατής μνήμης (συνέχεια)

- Η αποκωδικοποίηση μίας διεύθυνσης γίνεται με τον ίδιο τρόπο όπως και στην περίπτωση της απλής σελιδοποίησης. Όμως υπάρχει περίπτωση η σελίδα στην οποία γίνεται αναφορά να μην βρίσκεται στην κύρια μνήμη, κάτι που θα φανεί από την τιμή του σχετικού bit. Αυτό οδηγεί στη δημιουργία ενός *λάθους σελίδας* (page fault) και το Λ.Σ. μεταφέρει (αντιγράφει) τη σελίδα αυτή από την περιφερειακή στην κύρια μνήμη.

- Για τη μεταφορά αυτή πρέπει να δημιουργηθεί χώρος στην κύρια μνήμη, δηλαδή κάποια από τις ευρισκόμενες στην κύρια μνήμη σελίδες να μεταφερθεί στην περιφερειακή μνήμη. Το ερώτημα που δημιουργείται εδώ είναι ποια σελίδα είναι η πιο κατάλληλη για να φύγει από την κύρια μνήμη. Με άλλα λόγια δεν θέλουμε να βγάλουμε από την κύρια μνήμη μία σελίδα που θα ξαναχρειασθεί αμέσως μετά γιατί αυτό μπορεί να οδηγήσει στο φαινόμενο της συνεχούς εναλλαγής σελίδων από/προς την κύρια μνήμη με *απότομη πτώση της απόδοσης* του συστήματος (thrashing). Εδώ γίνεται χρήση του bit αναφοράς για την επιλογή της κατάλληλης σελίδας που πρέπει να μεταφερθεί στην περιφερειακή μνήμη.

- Αν η σελίδα που έχει επιλεγεί για απομάκρυνση από την κύρια μνήμη έχει υποστεί αλλαγές από την τελευταία φορά που διαβάσθηκε (κάτι που θα φανεί από τη τιμή του σχετικού bit), τότε θα πρέπει να ενημερωθεί το αντίγραφό της στο δίσκο. Αλλιώς, απλά σβήνεται από την κύρια μνήμη.

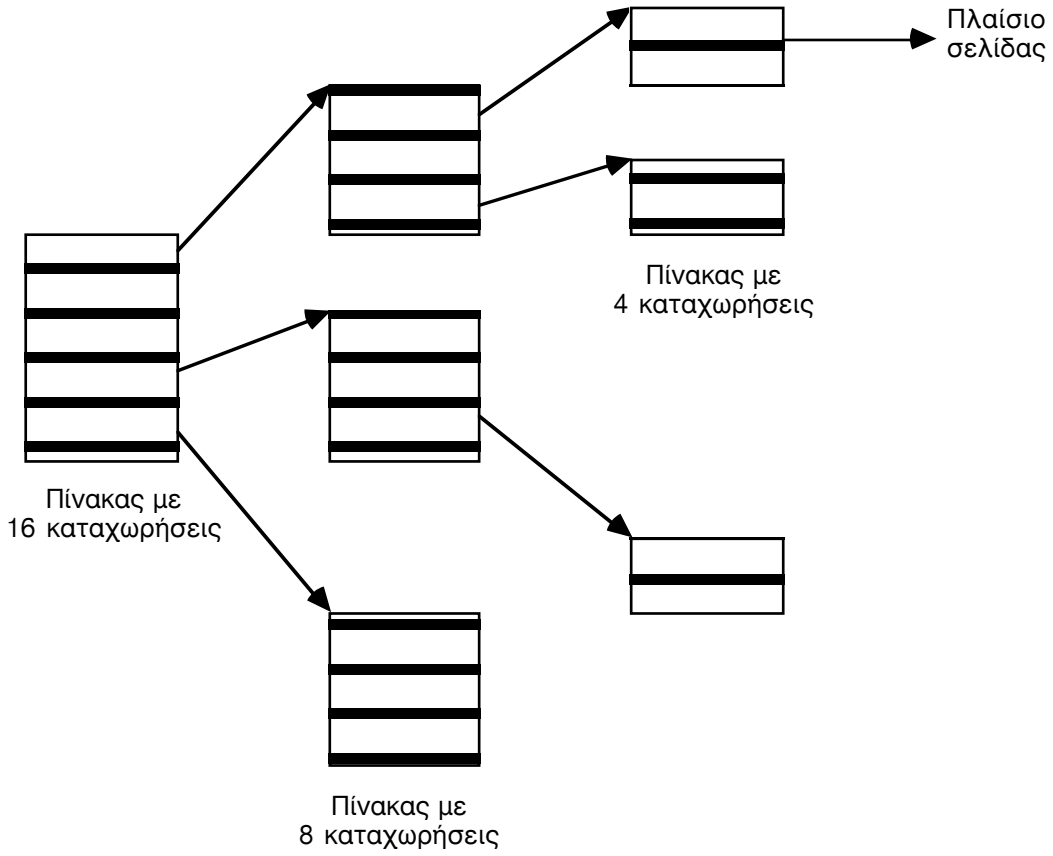
- Στα bits ελέγχου περιλαμβάνονται τα *bits προστασίας* (protection bits) που καθορίζουν το είδος της προσπέλασης που επιτρέπεται από τις διεργασίες στις σελίδες. Συνήθως χρησιμοποιείται ένα μόνο bit με τιμή 0 αν επιτρέπεται και η ανάγνωση και η εγγραφή του και τιμή 1 αν επιτρέπεται μόνο η ανάγνωση. Υπάρχει όμως περίπτωση να υπάρχουν μέχρι και 3 bits (ένα για ανάγνωση, ένα για εγγραφή και ένα για εκτέλεση).

- Στα bits ελέγχου περιλαμβάνονται επίσης και τα *bits κλειδώματος* (lock bits) που καθορίζουν κατά πόσο μία σελίδα μπορεί να απομακρυνθεί από την κύρια μνήμη (ισχύει κυρίως για διαμοιραζόμενες σελίδες).

3.2 Οργάνωση και υλοποίηση του πίνακα σελίδων

- Από τη στιγμή που ο αριθμός των σελίδων δεν περιορίζεται από το μέγεθος της κύριας μνήμης, ο πίνακας σελίδων τείνει να γίνεται μεγάλος και επομένως να τίθεται θέμα γρήγορης προσπέλασής του, μια πράξη η οποία είναι συνεχής (για αποκωδικοποίηση διευθύνσεων και εναλλαγή σελίδων). Το να υπάρχουν πίνακες της τάξης του 1 εκατομμυρίου εγγραφών (δηλαδή 1 εκατομμύριο σελίδες) είναι σύνηθες φαινόμενο. Σημειωτέον ότι κάθε διεργασία έχει το δικό της πίνακα!

- Έτσι ακόμα και ο πίνακας σελίδων πρέπει να χωρισθεί σε σελίδες και το μεγαλύτερο μέρος του να βρίσκεται στην περιφερειακή μνήμη. Αυτό οδηγεί στην οργάνωση του πίνακα σε πολλαπλά (περισσότερα του ενός) επίπεδα και την αποθήκευση μερικών από αυτών στο δίσκο.



3.2 Οργάνωση και υλοποίηση του πίνακα σελίδων (συνέχεια)

- Ένας εναλλακτικός τρόπος οργάνωσης του πίνακα σελίδων είναι να έχουμε μία εγγραφή για κάθε πλαίσιο σελίδας αντί για κάθε ιδεατή σελίδα. Αυτός λέγεται *ανεστραμμένος πίνακας σελίδων* (inverted page table) και κάθε εγγραφή του δηλώνει ποια ιδεατή σελίδα βρίσκεται σε κάθε πλαίσιο, έχει δε το πλεονέκτημα ότι το μέγεθός του είναι σχετικά μικρό (καθορίζεται από το πλήθος των πλαισίων σελίδων που έχει η κύρια μνήμη του συστήματος).

- Τέλος, για την πιο γρήγορη απεικόνιση ιδεατών διευθύνσεων σε φυσικές διευθύνσεις, χρησιμοποιείται συχνά, σε συνδυασμό με τον πίνακα σελίδων, μία μονάδα υλικού που λέγεται *συσχετιστική μνήμη* (associative memory) ή *ενδιάμεση μνήμη μετάφρασης* (translation lookaside buffer). Η μονάδα αυτή αποτελείται από μία ομάδα από καταχωρητές (8 έως 32) οι οποίοι μπορούν να προσπελασθούν παράλληλα. Σε αυτούς τους καταχωρητές αποθηκεύονται οι διευθύνσεις των πλαισίων σελίδας για τις πιο συχνά χρησιμοποιούμενες σελίδες. Όταν γίνεται η αποκωδικοποίηση μίας ιδεατής διεύθυνσης, πρώτα αναζητείται η σχετική πληροφορία στη συσχετιστική μνήμη. Αν δεν βρεθεί εκεί τότε γίνεται αναζήτηση στον πίνακα σελίδων και οι σχετικές πληροφορίες από τη ζητούμενη εγγραφή μεταφέρονται στη συσχετιστική μνήμη στη θέση κάποιας καταχώρησης εκεί η οποία βεβαίως πρέπει να απομακρυνθεί αφού πρώτα ενημερώσει την αντίστοιχη εγγραφή της στον πίνακα σελίδων. Το *ποσοστό επιτυχίας* (hit ratio) στο να βρεθούν οι ζητούμενες πληροφορίες στη συσχετιστική μνήμη παίζει πρωτεύοντα ρόλο στην ταχύτητα αποκωδικοποίησης των ιδεατών διευθύνσεων.

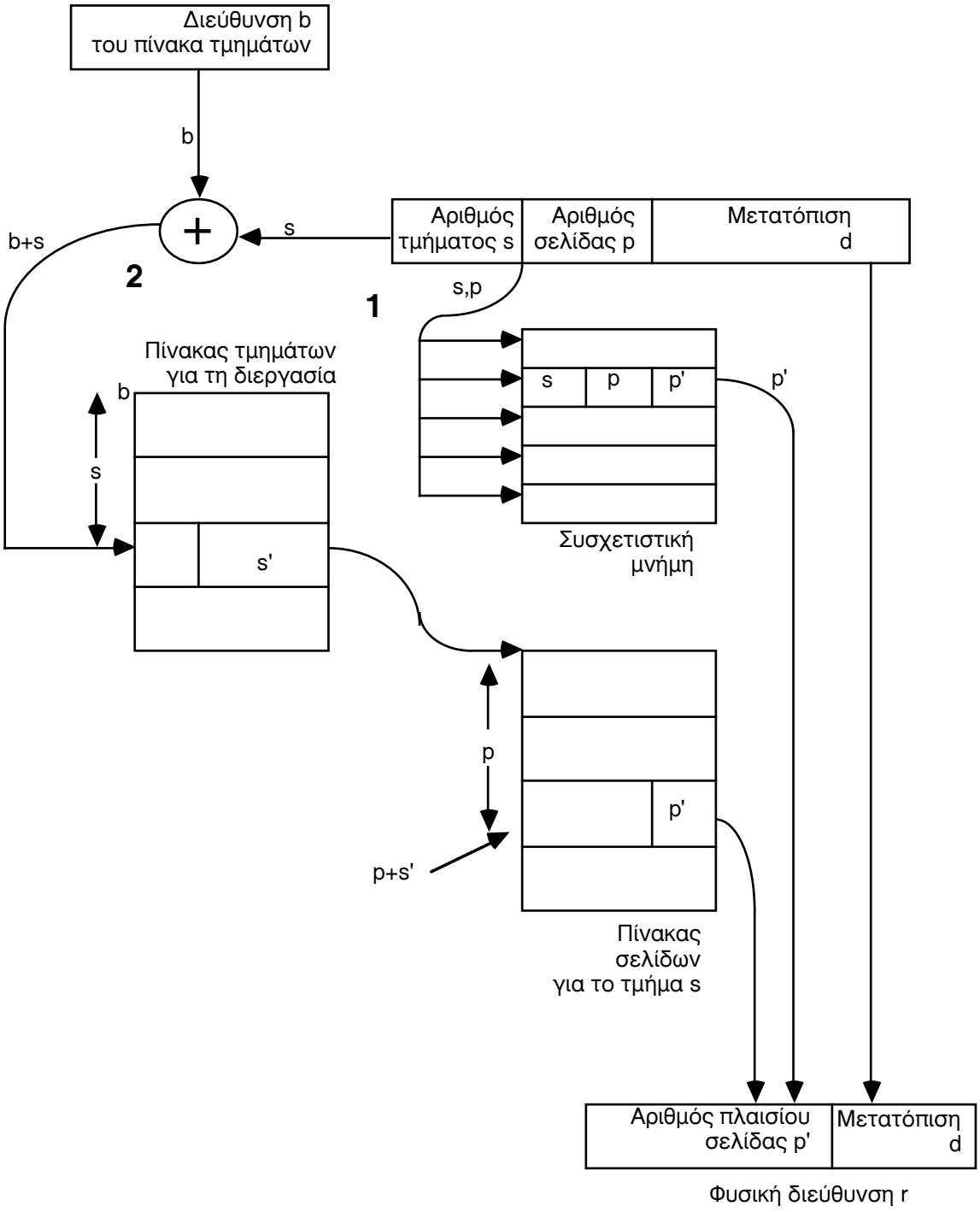
140	1	RW	21
20	1	X	4
110	0	R	56
850	1	RWX	3
1350	0	RW	67
Ιδεατή σελίδα	Τροποποίηση	Προστασία	Πλαίσιο σελίδας

3.3 Κατάτμηση με χρήση ιδεατής μνήμης

- Σε συνδυασμό με τη χρήση ιδεατής μνήμης, η τεχνική της κατάτμησης έχει μερικά πλεονεκτήματα:
  - — Απλοποιεί τη διαχείριση μεγάλων δομών δεδομένων γιατί μία τέτοια δομή μπορεί να έχει το δικό της τμήμα το οποίο να μεγαλώνει δυναμικά στην περιφερειακή μνήμη.
  - — Βοηθάει την ξεχωριστή αλλαγή και μεταγλώττιση των μερών ενός προγράμματος.
  - — Επιτρέπει την κοινή χρήση κάποιων τμημάτων (που αντιστοιχούν λ.χ. σε προγράμματα συστήματος).
  - — Βοηθάει στην ανάπτυξη μηχανισμών προστασίας στη χρήση των τμημάτων μια και έχουν λογική οντότητα. Κάθε τμήμα μπορεί να έχει κάποια bits που να ορίζουν τι δικαιώματα ή σε ποιούς *δακτυλίους προστασίας* (protection rings) ανήκει.

- Συνήθως η χρήση κατάτμησης γίνεται σε συνδυασμό με σελιδοποίηση, δηλαδή ένα πρόγραμμα χωρίζεται σε τμήματα, αλλά τα τμήματα καθ' εαυτά χωρίζονται σε σελίδες. Επίσης μπορεί να γίνει και χρήση συσχετιστικής μνήμης. Κάθε διεύθυνση ιδεατής μνήμης τώρα έχει τη μορφή r=(s,p,d). Οι πιο συχνά χρησιμοποιούμενες σελίδες έχουν εγγραφές στη συσχετιστική μνήμη. Πρώτα γίνεται προσπέλαση στη συσχετιστική μνήμη για να βρεθεί το ζευγάρι (s,p). Αν βρεθεί, τότε προστίθεται στον αριθμό p' του πλαισίου της σελίδας η μετατόπιση d για να υπολογισθεί η φυσική διεύθυνση r. Αλλιώς, η διεύθυνση μνήμης b του πίνακα τμημάτων προστίθεται στον αριθμό s του τμήματος για να βρεθεί η εγγραφή με διεύθυνση b+s στον πίνακα τμημάτων. Από εκεί βρίσκεται η διεύθυνση μνήμης s' του πίνακα σελίδων ο οποίος αντιστοιχεί στο τμήμα s. Η τιμή s'+p δηλώνει τώρα την εγγραφή του πίνακα για τη σελίδα p στην οποία βρίσκεται ο αριθμός p' του πλαισίου της σελίδας.

3.3 Κατάτμηση με χρήση ιδεατής μνήμης (συνέχεια)



4. Στρατηγικές διαχείρισης ιδεατής μνήμης

- Οι στρατηγικές διαχείρισης της ιδεατής μνήμης εστιάζονται στα εξής σημεία:

  Πολιτικές προσκόμισης σελίδας στην κύρια μνήμη (fetch policies).
  Πολιτικές τοποθέτησης σελίδας στην κύρια μνήμη (placement policies).
  Πολιτικές αντικατάστασης σελίδας (replacement policies).
  Διαχείριση των σελίδων που βρίσκονται στην κύρια μνήμη (resident set management).
  Πολιτικές απομάκρυνσης σελίδας από την κύρια μνήμη (cleaning policies).
  Έλεγχος του αριθμού των διεργασιών που βρίσκονται στην κύρια μνήμη (load control). Έχει να κάνει περισσότερο με χρονοπρογραμματισμό και δεν θα ασχοληθούμε περισσότερο εδώ με αυτό.

- Σκοπός όλων αυτών των πολιτικών είναι η μεγιστοποίηση της απόδοσης του συστήματος που σε αυτή την περίπτωση σημαίνει την ελαχιστοποίηση του αριθμού των *σφαλμάτων σελίδας* (page faults) που δημιουργούνται. Και αυτό γιατί η αντιμετώπιση ενός σφάλματος σελίδας είναι μία δαπανηρή σε χρόνο διαδικασία που αναγκάζει το Λ.Σ. μεταξύ άλλων να:
  — αποφασίσει ποια σελίδα πρέπει να απομακρυνθεί από την κύρια μνήμη,
  — να εναλλάξει τη σελίδα αυτή με τη σελίδα που πρέπει να φορτωθεί στην κύρια μνήμη,
  — κατά τη διάρκεια της εναλλαγής αυτής (που είναι σχετικά αργή μια και ουσιαστικά είναι λειτουργίες Ε/Ε) να εναλλάξει την τρέχουσα διεργασία που εκτελείται με άλλη (process switching), κάτι που έχει επιπρόσθετο κόστος.

- Σημαντικό ρόλο στην απόδοση του συστήματος παίζουν επίσης παράγοντες όπως το μέγεθος της κύριας μνήμης, η ταχύτητα προσπέλασης στην κύρια και περιφερειακή μνήμη, ο αριθμός και μέγεθος των διεργασιών που εκτελούνται ταυτόχρονα καθώς επίσης και η συμπεριφορά των εκτελούμενων προγραμμάτων.

4.1 Πολιτικές προσκόμισης σελίδας στην κύρια μνήμη

- Σκοπός τους είναι να αποφασίσουν για το ποιες σελίδες θα φορτωθούν στην κύρια μνήμη. Υπάρχουν δύο προσεγγίσεις:
  - — Μία σελίδα φορτώνεται στην κύρια μνήμη μόνο αν ζητηθεί. Αυτή η στρατηγική λέγεται *απαίτηση σελιδοποίησης* (demand paging) και βασίζεται στη λογική ότι μόλις αρχίσει να εκτελείται μία καινούργια διεργασία δημιουργούνται στην αρχή πολλά σφάλματα σελίδας αλλά μόλις φορτωθούν οι σελίδες που χρειάζονται, αυτά μειώνονται σημαντικά.
  - — Φορτώνεται μία ομάδα από σελίδες πριν ζητηθούν. Αυτή η στρατηγική λέγεται *προσελιδοποίηση* (prepaging) και βασίζεται στο γεγονός ότι λόγω του τρόπου που λειτουργούν οι συσκευές αποθήκευσης, είναι πιο αποδοτικό να μεταφερθεί στην κύρια μνήμη μία ομάδα από σελίδες αποθηκευμένες συνεχόμενα, παρά μία μόνο σελίδα. Το ποιες από αυτές τις σελίδες θα χρησιμοποιηθούν τελικά έχει να κάνει με τη συμπεριφορά του προγράμματος. Συνήθως η προσελιδοποίηση είναι αποτελεσματική όταν το πρόγραμμα παρουσιάζει τοπικότητα στις αναφορές του (locality of reference).

4.2 Πολιτικές τοποθέτησης σελίδας στην κύρια μνήμη

- Σε συστήματα που χρησιμοποιούν σελιδοποίηση το πρόβλημα αυτό δεν υπάρχει γιατί μία σελίδα μπορεί να φορτωθεί σε οποιοδήποτε μέρος της κύριας μνήμης. Μόνο σε συστήματα που χρησιμοποιούν μόνο κατάτμηση πρέπει να χρησιμοποιηθεί κάποιος από τους αλγόριθμους τοποθέτησης.

4.3 Πολιτικές αντικατάστασης σελίδας

- Σκοπός των *πολιτικών αντικατάστασης* (replacement policies) είναι να επιλέξουν την “πιο κατάλληλη” σελίδα που πρέπει να απομακρυνθεί από την κύρια μνήμη για να δημιουργηθεί χώρος για να φορτωθεί κάποια άλλη σελίδα από την περιφερειακή μνήμη. Ο ορισμός της “πιο κατάλληλης” σελίδας δίνεται από την *αρχή της βελτιστοποίησης* (principle of optimality): η σελίδα που πρέπει να αντικατασταθεί είναι αυτή που δεν θα χρειασθεί στο μέλλον για το μεγαλύτερο χρονικό διάστημα. Φυσικά μια και δεν μπορεί να προβλεφθεί το μέλλον οι αλγόριθμοι που ακολουθούν προσπαθούν να εκτιμήσουν κατά προσέγγιση ποια είναι η καλύτερη σελίδα.

- Ο *αλγόριθμος της τυχαίας αντικατάστασης σελίδας* (random page replacement algorithm) απλά διαλέγει τυχαία μία σελίδα. Έχει μικρό κόστος εκτέλεσης αλλά μπορεί κάλλιστα να διαλέξει για αντικατάσταση τη σελίδα που αμέσως μετά θα χρησιμοποιηθεί ξανά. Έτσι χρησιμοποιείται σπάνια.

- Ο *αλγόριθμος της βέλτιστης αντικατάστασης σελίδας* (optimal page replacement algorithm) εξετάζει τις εντολές του εκτελούμενου προγράμματος και διαλέγει τη σελίδα εκείνη στην οποία θεωρεί ότι θα γίνει αναφορά όσο γίνεται πιο μακρυά στο μέλλον. Το πρόβλημα είναι ότι δεν μπορούμε να ξέρουμε τη μελλοντική συμπεριφορά του προγράμματος. Όμως ο αλγόριθμος μπορεί να χρησιμεύσει σαν μέσο σύγκρισης για τους υπόλοιπους.

- Ο *αλγόριθμος αντικατάστασης της σελίδας που δεν χρησιμοποιήθηκε πρόσφατα* (not recently used page replacement algorithm) επιλέγει τη σελίδα εκείνη στην οποία δεν έχει γίνει αναφορά για ένα συγκεκριμένο χρονικό διάστημα (ένα κτύπο του ρολογιού, δηλαδή γύρω στα 20msec). Ο αλγόριθμος αυτός είναι εύκολα κατανοητός, έχει καλές επιδόσεις και υλοποιείται εύκολα με απλή αναφορά στα bits εκείνα του πίνακα σελίδων που δείχνουν αν έχει γίνει αναφορά ή τροποποίηση σε μια σελίδα.

4.3 Πολιτικές αντικατάστασης σελίδας (συνέχεια)

- Ο *αλγόριθμος αντικατάστασης της λιγότερο πρόσφατα χρησιμοποιούμενης σελίδας* (least recently used page replacement algorithm) επιλέγει τη σελίδα εκείνη στην οποία δεν έχει γίνει αναφορά για το μεγαλύτερο χρονικό διάστημα. Με βάση το φαινόμενο της τοπικότητας της αναφοράς, οι πιθανότητες η σελίδα αυτή να χρειασθεί στο εγγύς μέλλον είναι μικρές. Ο αλγόριθμος αυτός έχει καλές επιδόσεις αλλά είναι πολυέξοδος στην υλοποίηση γιατί χρειάζεται να καταγράφεται ο χρόνος που έγινε αναφορά σε μια σελίδα κάθε φορά που χρησιμοποιείται για ανάγνωση ή/και τροποποίηση.

- Ο *αλγόριθμος αντικατάστασης σελίδας πρώτη μέσα πρώτη έξω* (first in first out page replacement algorithm) διαλέγει για αντικατάσταση τη σελίδα εκείνη που έχει μείνει για μεγαλύτερο χρονικό διάστημα στην κύρια μνήμη. Βασίζεται στη λογική ότι οι πιθανότητες η σελίδα αυτή να συνεχίσει να είναι χρήσιμη είναι μικρές. Είναι απλός στην υλοποίηση (χρειάζεται μόνο μία κυκλική ουρά) αλλά η λογική στην οποία βασίζεται δεν είναι συχνά σωστή και έτσι έχει κακή απόδοση.

- Ο *αλγόριθμος αντικατάστασης σελίδων δεύτερης ευκαιρίας* (second chance page replacement algorithm) μπορεί να χαρακτηρισθεί σαν συνδυασμός των δύο τελευταίων. Η σελίδα που βρίσκεται πρώτη για απομάκρυνση της δίνεται μία δεύτερη ευκαιρία με την αλλαγή της τιμής ενός bit από 1 σε 0 και την τοποθέτησή της στο τέλος της ουράς. Αν βρεθεί ξανά στην κορυφή της ουράς και το bit εξακολουθεί να έχει τιμή 0 τότε απομακρύνεται. Αν εν τω μεταξύ έχει χρησιμοποιηθεί, τότε αλλάζει η τιμή του bit σε 1 και δεν αντικαθίσταται. Αν και είναι λογικός αλγόριθμος, οδηγεί σε συνεχή μετακίνηση σελίδων γύρω από τη λίστα και δεν είναι ιδιαίτερα αποδοτικός.

- Ο *αλγόριθμος του ρολογιού για αντικατάσταση σελίδων* (clock page replacement algorithm) είναι μία παραλλαγή του προηγούμενου. Αντί να μετακινούνται οι ίδιες οι σελίδες, υπάρχει ένας δείκτης που ψάχνει να βρεί τη σελίδα εκείνη που έχει τιμή bit 0. Μια και η μετακίνηση του δείκτη είναι λιγότερο δαπανηρή από αυτή των σελίδων, ο αλγόριθμος αυτός είναι πιο αποδοτικός (ο δείκτης συμπεριφέρεται σαν αυτόν ενός ρολογιού).

4.4 Το παράδοξο του Belady

- Αν και διαισθητικά φαίνεται ότι όσα περισσότερα πλαίσια σελίδας βρίσκονται στην κύρια μνήμη τόσο λιγότερα σφάλματα σελίδας θα δημιουργούνται, εν τούτοις με τον αλγόριθμο FIFO ανακαλύφθηκε από τον Belady (1969) ότι μερικές φορές περισσότερα πλαίσια δημιουργούν περισσότερα σφάλματα!

- Στο ακόλουθο παράδειγμα βλέπουμε τη συμπεριφορά 5 ιδεατών σελίδων (0 έως 4) με 3 και 4 πλαίσια σελίδας αντίστοιχα όπου οι αναφορές στις σελίδες γίνονται με την ακόλουθη σειρά: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	

	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	2
			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P

- Παρατηρούμε ότι με 4 πλαίσια έχουμε συνολικά 10 σφάλματα σελίδας ενώ με 3 πλαίσια μόνο 9.

4.5 Διαχείριση των σελίδων που βρίσκονται στην κύρια μνήμη

- Αυτός ο μηχανισμός διαχείρισης της ιδεατής μνήμης εστιάζεται στα εξής σημεία:

  Τρόποι επιμερισμού των πλαισίων σελίδας σε διεργασίες και συγκεκριμένα πόσα πλαίσια σελίδας θα δοθούν σε κάθε διεργασία. Αν δοθούν λίγα πλαίσια, τότε μπορούν να εξυπηρετηθούν περισσότερες διεργασίες αλλά αυξάνει ταυτόχρονα και η συχνότητα δημιουργίας σφάλματος σελίδας.

  Υπάρχουν δύο προσεγγίσεις: αυτή της *σταθερής κατανομής* (fixed allocation) που κρατάει σταθερό τον αριθμό των πλαισίων που χορηγούνται σε μία διεργασία και αυτή της *μεταβλητής κατανομής* (variable allocation) που επιτρέπει δυναμικά την αύξηση του αριθμού των πλαισίων κάποιας διεργασίας, κυρίως αυτών που δημιουργούν πολλά σφάλματα σελίδας.

  Το διαχωρισμό του μηχανισμού αντικατάστασης σελίδας σε *τοπικό* (local replacement policy) όπου εξετάζονται μόνο οι σελίδες εκείνες της διεργασίας που δημιούργησαν το σφάλμα σελίδας και σε *ολικό* (global replacement policy) που εξετάζει όλες τις σελίδες που βρίσκονται στην κύρια μνήμη. Εδώ γίνεται συνήθως συνδυασμός με τον τρόπο επιμερισμού των πλαισίων που οδηγούν σε 3 δυνατούς συνδυασμούς:

  — σταθερή κατανομή με τοπική αντικατάσταση,
  — μεταβλητή κατανομή με ολική αντικατάσταση,
  — μεταβλητή κατανομή με τοπική αντικατάσταση.

- Το πρόβλημα με τη δεύτερη προσέγγιση είναι ότι η σελίδα που θα αντικατασταθεί μπορεί να ανήκει σε οποιαδήποτε διεργασία η οποία τελικά τυχόν να αδικηθεί αν τις αφαιρεθούν από την κύρια μνήμη πολλές σελίδες. Η τρίτη προσέγγιση προσπαθεί να λύσει το πρόβλημα αυτό με περιορισμό της αντικατάστασης μεταξύ των σελίδων που ανήκουν στη διεργασία αλλά περιοδική επανεξέταση της συμπεριφοράς κάθε διεργασίας έτσι ώστε να εκτιμηθεί αν χρειάζεται να γίνει ανακατανομή στον αριθμό των πλαισίων που ανήκουν σε κάθε διεργασία.

4.5 Διαχείριση των σελίδων που βρίσκονται στην κύρια μνήμη (συνέχεια)

- Εδώ γίνεται χρήση του λεγόμενου *μοντέλου συνόλου εργασίας* (working set model) που προσπαθεί να υπολογίσει το μέγεθος της μνήμης που πρέπει να έχει κάθε διεργασία για να εκτελεσθεί χωρίς να δημιουργήσει πολλά σφάλματα σελίδας.

- Συνδυάζεται η χρήση του με αυτή του αλγόριθμου *συχνότητας λάθους σελίδας* (page fault frequency) που ποσπαθεί να εντοπίσει τις διεργασίες εκείνες που δημιουργούν ένα απαράδεκτα υψηλό αριθμό σφαλμάτων σελίδων και ή να τις απομακρύνει προσωρινά από την κύρια μνήμη ή να τις εφοδιάσει με περισσότερα πλαίσια σελίδας.

4.6 Πολιτικές απομάκρυνσης σελίδας από την κύρια μνήμη

- Οι πολιτικές αυτές αποφασίζουν για το πότε μία σελίδα *που έχει υποστεί τροποποίηση* θα γραφεί στην περιφερειακή μνήμη:
  - Η προσέγγιση της *απομάκρυνσης μετά από απαίτηση* (demand cleaning policy) ενημερώνει το αντίγραφο της σελίδας στην περιφερειακή μνήμη μόνο όταν πρέπει να απομακρυνθεί η σελίδα αυτή από την κύρια μνήμη.
  - Η προσέγγιση της *προαπομάκρυνσης* (precleaning policy) ενημερώνει το αντίγραφο μίας σελίδας στην περιφερειακή μνήμη πριν χρειασθεί κάποια διεργασία το αντίστοιχο πλαίσιο. Αυτό επιτρέπει τη μεταφορά σελίδων στην περιφερειακή μνήμη κατά ομάδες αλλά η ενημέρωση των αντίστοιχων αντιγράφων τους μπορεί να καταστεί άχρηστη αν εν τω μεταξύ οι σελίδες αυτές τροποποιηθούν. Η προηγούμενη μέθοδος δεν έχει αυτό το πρόβλημα αλλά η αντιμετώπιση ενός σφάλματος σελίδας παίρνει περισσότερο χρόνο.