ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Λειτουργικά Συστήματα

Διαχείριση Μνήμης

«Δίψα» για Μνήμη!

- ~1990: ένας desktop υπολογιστής είχε περίπου 1MB memory.
  - Τα 4MB ήταν πολυτέλεια!
- Σήμερα: ένα laptop, ακόμα κι ένα κινητό(!) έχει 8 με 16GB memory.
  - Δηλ. περίπου 16,000 φορές μεγαλύτερη μνήμη!!
- Οι εφαρμογές «διψούν» για μνήμη. Γιατί;
  - Περισσότερη λειτουργικότητα
  - Μεγαλύτερη πολυπλοκότητα
  - Λιγότερο ταλαντούχοι προγραμματιστές (?) ☹

«Δίψα» για Μνήμη!

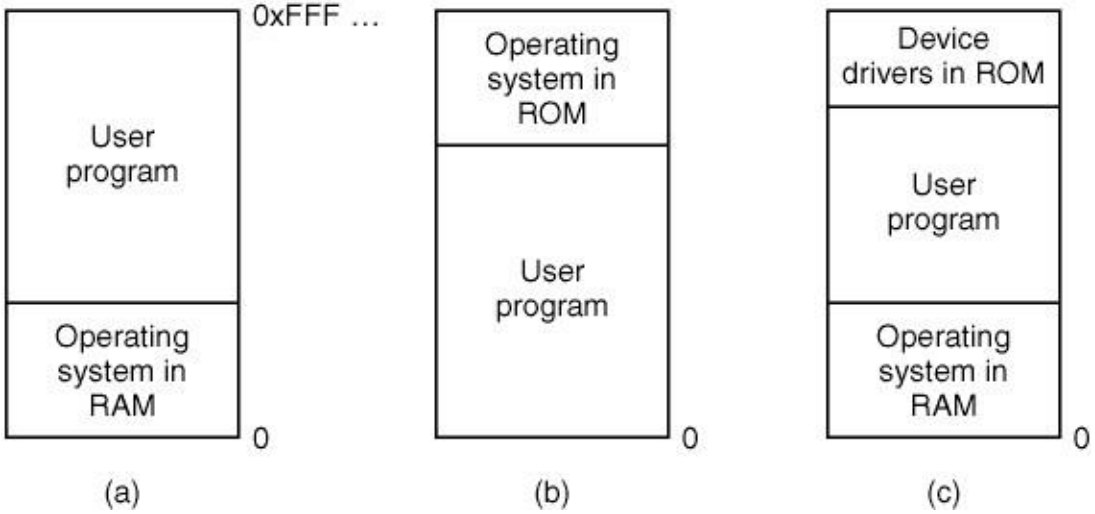
“Nature abhors a vacuum!”

(Η Φύση απεχθάνεται το κενό)

Ίσως να αφορά και στη μνήμη…

Monoprogramming

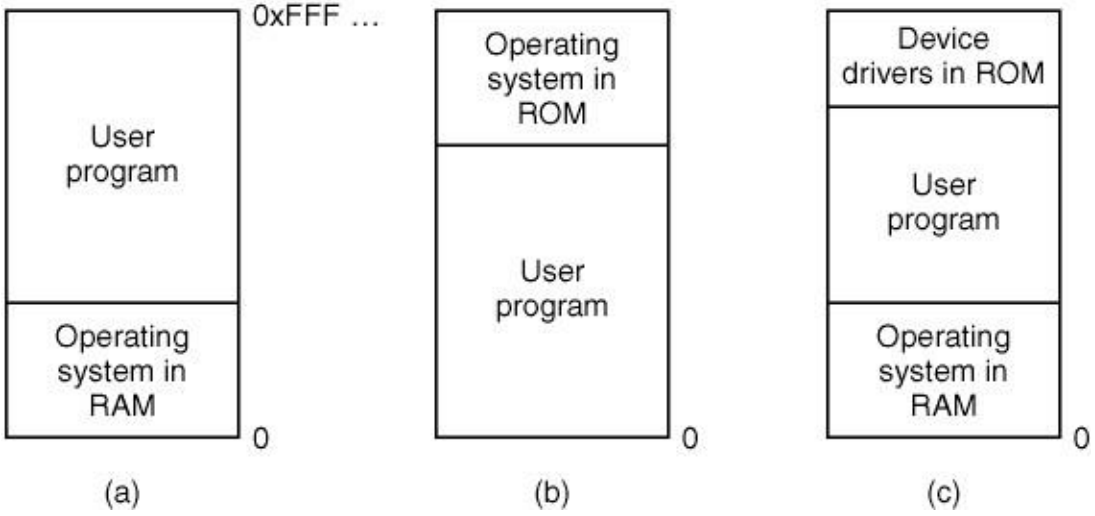
- Μόνο ένα process «φορτωμένο» στη μνήμη κάθε φορά.
- Η ΚΜ χωρίζεται σε 2 μέρη: **ΛΣ**, **user process**.
- Μόλις ο χρήστης ζητήσει να τρέξει ένα πρόγραμμα, το ΛΣ το "φορτώνει" στην ΚΜ και το τρέχει.



Monoprogramming

- Το monoprogramming δεν αρκεί στις πιο πολλές περιπτώσεις:
  - I/O-bound jobs ➔ το CPU υποαπασχολείται ➔ κακή απόδοση.
  - Πολλές εφαρμογές θέλουν τη δυνατότητα δημιουργίας πολλών διεργασιών που τρέχουν "ταυτόχρονα".

- Άρα χρειαζόμαστε multiprogramming
  - Η δουλειά του Memory Manager γίνεται πολύ πιο δύσκολη



Multiprogramming: Ωφέλη

- Έστω
  - ***p****:* η πιθανότητα μια διεργασία να είναι σε I/O-wait σε μια χρονική στιγμή
  - ***N***: το πλήθος των διεργασιών

- Το CPU utilization δίνεται από τη σχέση:
  - CPU utilization = $1 - p^N$
  - ➔ με p = 80% το CPU utilization είναι:

N	1	2	3	4
CPU util.	20%	36%	49%	59%

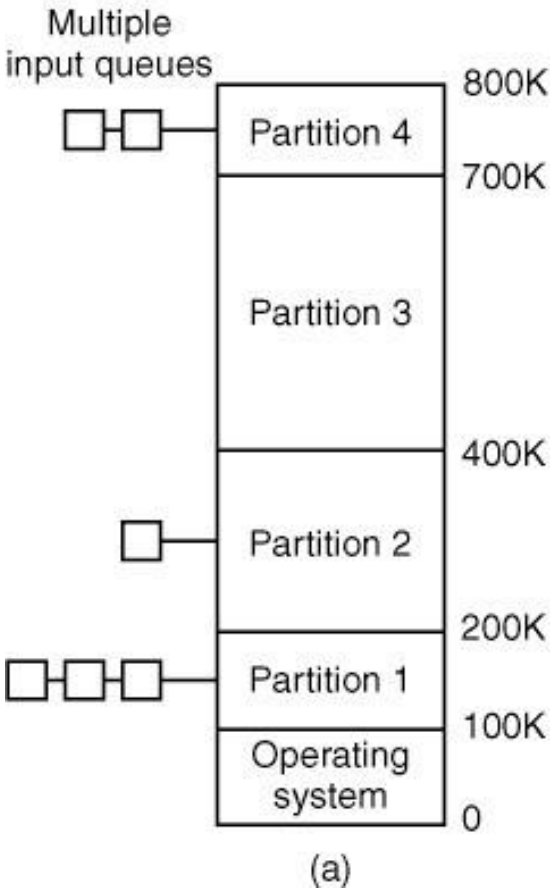
  - ➔ το multiprogramming έχει πολύ ευεργετικές συνέπειες.

Σημείωση:

Το παραπάνω μαθηματικό μοντέλο είναι υπεραπλουστευμένο. Γιατί;

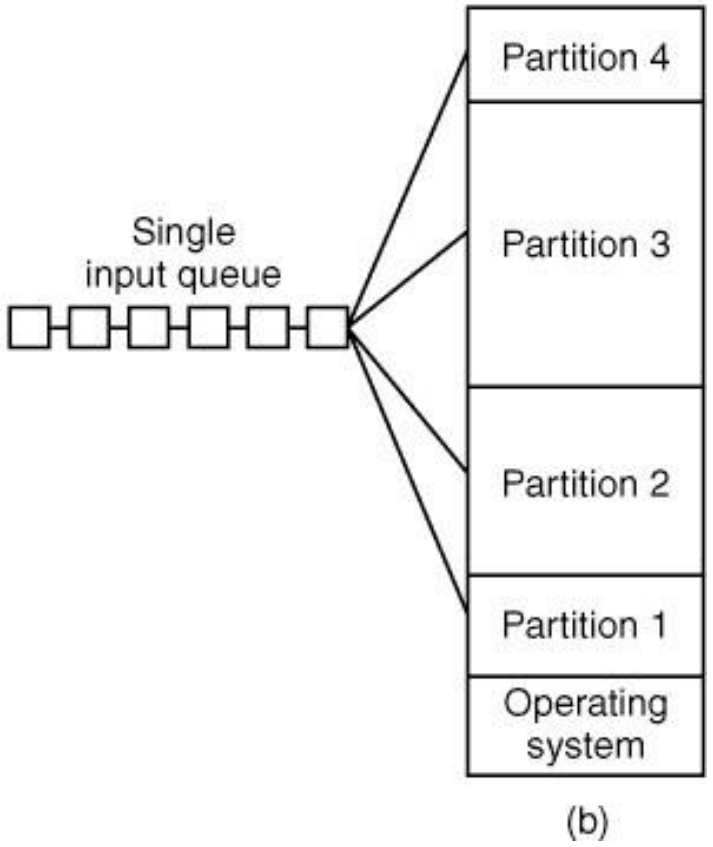
Multiprogr. with fixed partitions

- Ιδέα:
  - Χώρισε τη μνήμη σε **fixed partitions**
  - Φόρτωσε την κάθε process σε κατάλληλου μεγέθους partition
- Συν:
  - Τρέχουμε >1 προγράμματα παράλληλα
- Πλην:
  - Πρέπει να γνωρίζουμε πόση μνήμη θα χρειαστεί κάθε πρόγραμμα
  - Σπατάλη μνήμης
  - Δεν υπάρχει memory isolation / protection
  - Αρκετό queuing time



Multiprogr. with fixed partitions

- Μικρή βελτίωση:
  - Κράτα μία ουρά (queue) για όλα τα partitions.
- Τι κέρδος έχουμε;
  - Λιγότερο queuing time
- Δεν λύνει όλα τα προβλήματα…
  - γνώση μεγέθους εκ των προτέρων
  - σπατάλη μνήμης
  - memory isolation / protection



Multiprogr. with fixed partitions

- **Internal fragmentation** (**Εσωτερικός Κατακερματισμός**)
  - Αν ένα **μικρό process** μπει σε **μεγάλο partition**, τότε αν και υπάρχει πολύς διαθέσιμος χώρος σε αυτό το partition δεν μπορεί να αξιοποιηθεί από άλλη διεργασία. Αυτό ονομάζεται εσωτερικός κατακερματισμός.

- Λύση: Όχι προφανής!
  - Για παράδειγμα αν κάθε φορά ο Memory Manager ψάχνει για το process στην ουρά που ελαχιστοποιεί τον εσωτερικό κατακερματισμό, τότε μικρές διεργασίες θα αδικούνται (λιμοκτονούν).
  - θυμηθείτε ότι οι μικρές και διαδραστικές διεργασίες πρέπει να έχουν υψηλή προτεραιότητα.

- Μια λύση κατά της λιμοκτονίας
  - Σε κάθε μικρό process αντιστοιχούμε μια μεταβλητή, *k*, που αντιπροσωπεύει πόσες φορές ο διαχειριστής *δεν* έδωσε το partition στο process. Μετά από K φορές, το partition δίνεται στο process.
  - Αυτή η τεχνική ονομάζεται γενικά **aging** (**παλαίωση** ή **γήρανση**).

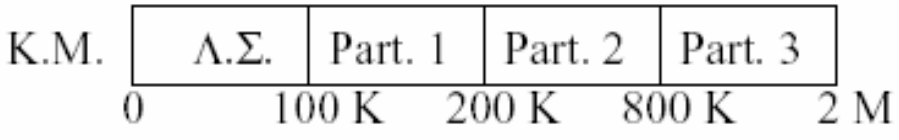
Προβλήματα των Partitions

- Ακόμα και με monoprogramming υπάρχουν 2 partitions.

Λ.Σ.	user program

  - Ο linker και ο loader συνεργάζονται και παράγουν τις διευθύνσεις των εντολών και των δεδομένων του προγράμματος. Ο linker παράγει "λογικές" διευθύνσεις και όχι "φυσικές", - δηλ. η πρώτη διεύθυνση είναι 0, κ.τ.λ.
  - Όμως, στη φυσική διεύθυνση 0 βρίσκονται εντολές ή δεδομένα του Λ.Σ. ➔ τίθεται θέμα προστασίας του Λ.Σ.

- Η λύση εστιάζει στη μετάφραση των «λογικών» διευθύνσεων σε «φυσικές».



- Για παράδειγμα αν το process έχει μπει στο Partition 1, τότε σ' όλες τις διευθύνσεις που παράγονται κατά την εκτέλεση του process, το κατάλληλο υλικό προσθέτει 100Κ.

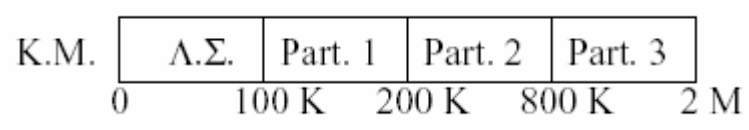
Θεμελιώδη Προβλήματα των Partitions

- Συνήθως η παραπάνω στρατηγική υλοποιείται με τη χρήση ενός ειδικού **base register** (**καταχωρητή βάσης**).
- Στο παραπάνω παράδειγμα, όταν το process φορτωθεί στο Part. 1 και αμέσως πριν του δοθεί η CPU, ο καταχωρητής βάσης παίρνει την τιμή 100 K.
- Αυτό γίνεται από τον kernel του Λ.Σ. (δηλ. οι διεργασίες δεν έχουν πρόσβαση σε αυτόν τον register).
- Έτσι επιτυγχάνεται το επιθυμητό αποτέλεσμα.

Θεμελιώδη Προβλήματα των Partitions

- Ο καταχωρητής βάσης όμως δεν αρκεί!

- Στην περίπτωση πολυπρογραμματισμού, πρέπει το σύστημα να εγγυάται ότι καθένα πρόγραμμα θα προστατεύεται από τα άλλα που είναι στην Κ.Μ.



➔ απαιτείται έλεγχος σχετικά με τις διευθύνσεις που μπορεί να παράγει ένα process.

➔ π.χ., το process στο Part. 1 δεν μπορεί να παράγει διευθύνσεις > 200Κ γιατί τότε θα κάνει ζημιά στο process που τρέχει στο Part. 2.

Θεμελιώδη Προβλήματα των Partitions

- **Limit register**
  - Αυτό το πρόβλημα λύνεται με έναν άλλο ειδικό καταχωρητή, τον **limit register** (**καταχωρητή ορίου**)
  - Ο limit register περιέχει την ανώτατη διεύθυνση του partition στο οποίο τρέχει το process

- Κάθε διεύθυνση που παράγεται, λοιπόν, **προστίθεται στον base register** και το αποτέλεσμα **συγκρίνεται με τον limit register**
  - αν είναι μεγαλύτερο, το process πεθαίνει

- Σημείωση
  - Με base και limit registers τα προγράμματα μπορούν να μετακινηθούν στην Κ.Μ. (δηλαδή ν' ανατεθούν σε διαφορετικά partitions κατά την διάρκεια της εκτέλεσής τους)
  - Αυτό ονομάζεται **relocation** (**επανατοποθέτηση**)

Swapping (Εναλλαγή)

- Τί γίνεται όταν υπάρχουν πιο πολλές διεργασίες από όσες μπορεί να υποστηρίξει η Κ.Μ.;
  - Ορίζεται ένας χώρος στον δίσκο (**swap space**) που φιλοξενεί τις διεργασίες που δεν «χωρούν» στην Κ.Μ.
  - Η μεταφορά διεργασιών μεταξύ Κ.Μ. και swap space στον δίσκο ονομάζεται **swapping** (**εναλλαγή**).

- Παρά ταύτα, η διαχείριση μνήμης με σταθερά partitions είναι καταδικασμένη να μην είναι αρκετά καλή
  - Συνήθως σπαταλάται πολλή μνήμη γιατί οι απαιτήσεις των processes δεν «ταιριάζουν» απόλυτα με τα μεγέθη των partitions
  - Είναι προφανής η ανάγκη να εξετάσουμε τη χρήση partitions μεταβλητού μεγέθους (variable sized partitions)

Μεταβαλλόμενα partitions

- **Βασική ιδέα:**
  - Οι **διευθύνσεις**, ο **αριθμός**, και τα **μεγέθη** των **partitions** **μεταβάλλονται** ανάλογα με τις ανάγκες.
- Κάθε διαμέρισμα αποτελείται από συνεχόμενες διευθύνσεις στην Κ.Μ. και μπορεί να είναι ίσου μεγέθους με τη διεργασία που «φιλοξενεί» (συνήθως λίγο μεγαλύτερο).
- Το σημαντικό αποτέλεσμα είναι ότι τα μεταβαλλόμενα partitions ελαχιστοποιούν τη σπατάλη χώρου στην Κ.Μ.
- Από την άλλη πλευρά, όμως, η ανάθεση μνήμης σε διεργασίες και γενικά το έργο διαχείρισης μνήμης δυσκολεύει.

Μεταβαλλόμενα partitions

- Παράδειγμα:

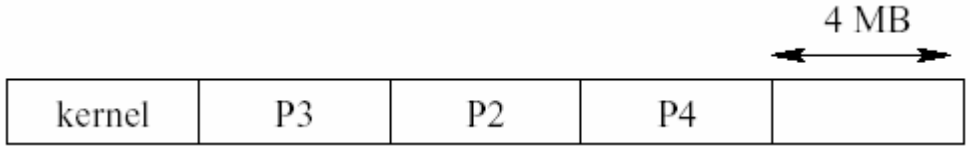
Time 1	Kernel	P1			Μόλις ήρθε το process P1
Time 2	Kernel	P1	P2		Μόλις ήρθε και το process P2
Time 3	Kernel		P2		Έφυγε το process P1
Time 4	Kernel	P3	P2		Ήρθε το process P3
Time 5	Kernel	P3	P2	P4	Ήρθε το process P4

- Βλέπουμε ότι υπάρχουν 2 «άδεια» partitions της Κ.Μ., έστω με μέγεθος 3MB και 1MB αντίστοιχα.
- Αν τώρα έρθει ένα process P5 που χρειάζεται 4MB, δεν μπορεί να γίνει δεκτό, παρότι υπάρχουν 4MB ελεύθερα στο σύνολο.
- Αυτό ονομάζεται **external fragmentation** (**εξωτερικός κατακερματισμός**).

Μεταβαλλόμενα partitions

- Μια λύση στο πρόβλημα του external fragmentation είναι το **storage compaction** (**σύμπτυξη μνήμης**) που συνδέει όλους τους άδειους χώρους της ΚΜ σ' ένα συνεχόμενο διαμέρισμα της Κ.Μ.

- Για παράδειγμα:



- Σπάνια όμως χρησιμοποιείται επειδή απαιτεί χρονοβόρο memory-to-memory copying (CPU time).

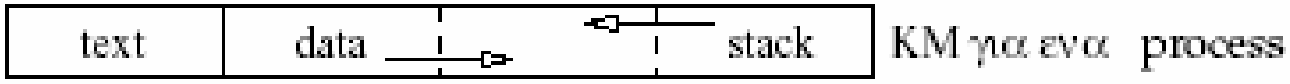
Μεταβαλλόμενα partitions

- Τα μεταβαλλόμενα partitions οπωσδήποτε αυξάνουν την ευελιξία που έχουμε. Όμως, παραμένει ένα ερώτημα: **Πόση μνήμη να αναθέσουμε σε ένα process;**

- Η δυσκολία έγκειται στο ότι συνήθως οι ανάγκες ενός process για μνήμη μεταβάλλονται κατά την εκτέλεσή του.

- Ένα process έχει τρία τμήματα:
  - **text/code:** το object code («εκτελέσιμο») -- δεν μεταβάλλεται
  - **data:** το τμήμα δεδομένων -- μεταβάλλεται [βλ. malloc()]
  - **stack:** η «στοίβα» -- επίσης μεταβάλλεται

  Το data τμήμα μεγαλώνει προς το stack και αντίστροφα.



ΚΜ για ενα process

- Προσέξτε: αν δίναμε ακριβώς όσο χώρο χρειάζεται ένα process όταν εισέρχεται στη μνήμη, τότε θα απαιτείτο mem-to-mem αντιγραφή για να μεταφέρουμε ένα process που μεγαλώνει σε άλλο partition.

Διαχείριση Μνήμης

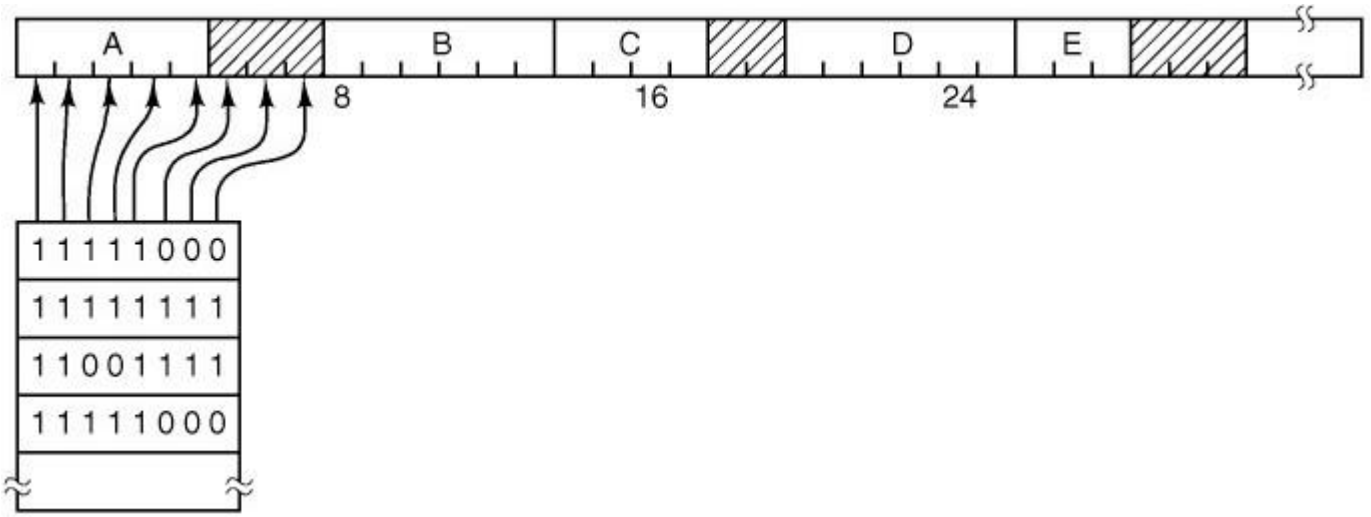
- Τα θεμελιώδη θέματα είναι:
  1. ποια είναι τα άδεια partitions της μνήμης;
  2. ποιο process κατέχει ποιο partition της μνήμης;

- Υπάρχουν διάφορες μέθοδοι Διαχείρισης Μνήμης που δίνουν λύσεις:
  - **Bitmaps:** χάρτες ψηφίων που περιέχουν ένα bit για κάθε partition. Π.χ., αν BM[i]==0 ➔ το partition i είναι ελεύθερο.
  - **Linked Lists:** λίστες των οποίων οι κόμβοι αντιστοιχούν σε partitions.
  - **Buddy Systems:** βασίζεται σε διαφορετικές λίστες, οι κόμβοι των οποίων αντιστοιχούν σε partitions των οποίων το μέγεθος είναι διαφορετικές δυνάμεις του 2.

ΔΜ με Bitmaps

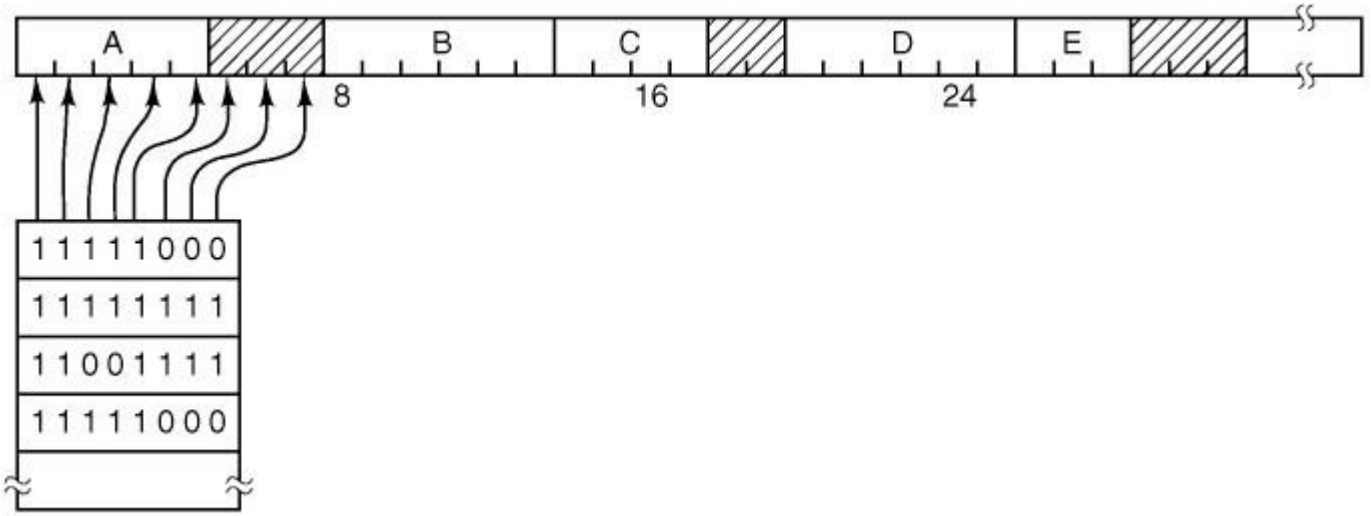
- Η μνήμη χωρίζεται σε **allocation units** (**μονάδες ανάθεσης**) σταθερού μεγέθους λίγων Bytes ή KB. Το μέγεθος των μονάδων ανάθεσης είναι σταθερό.

- Σε κάθε μονάδα ανάθεσης αντιστοιχεί **ένα bit στο bitmap**. Αν η θέση ***x*** είναι 0, τότε η μονάδα ανάθεσης ***x*** δεν χρησιμοποιείται από κανένα process.



ΔΜ με Bitmaps

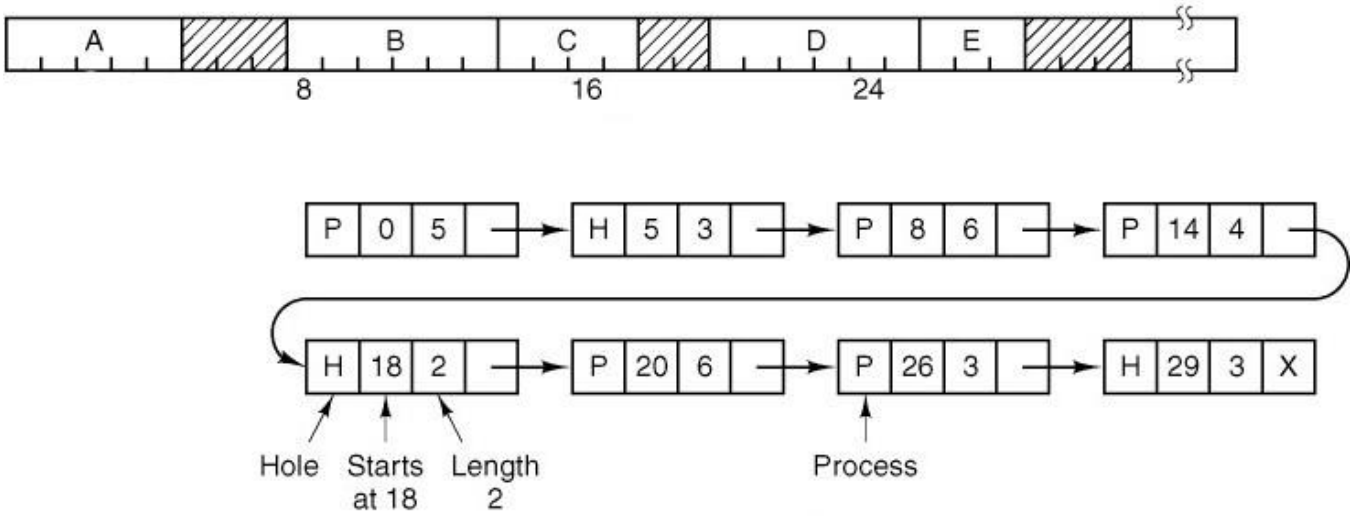
- Ποιο είναι το ιδανικό μέγεθος του allocation unit (μονάδας ανάθεσης);
  - Μικρές μονάδες ανάθεσης → μεγάλο bitmap
  - Μεγάλες μονάδες ανάθεσης → internal fragmentation (η τελευταία μονάδα ανάθεσης δεν θα χρησιμοποιείται όλη)
- +/- της χρήσης Bitmaps
  - **+:** η **απλότητα** τους (ευκολία υλοποίησης).
  - **- : κόστος εύρεσης ελεύθερου partition** -- πρέπει να βρεθούν στο BitMap αρκετά συνεχόμενα 0 - μια χρονοβόρα διαδικασία.



ΔΜ με Linked Lists

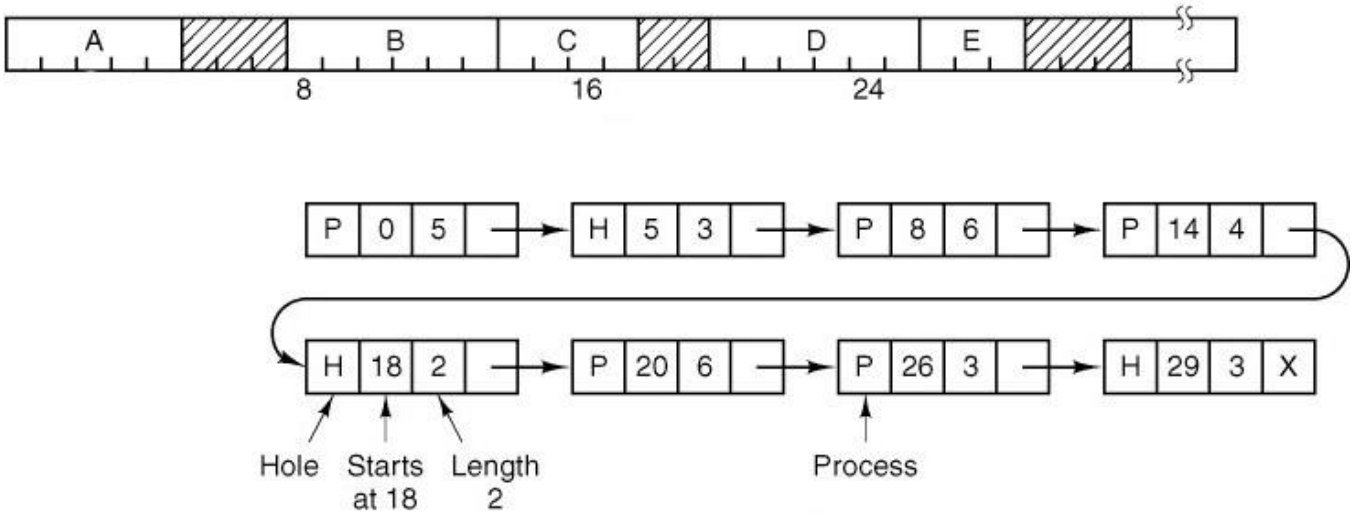
- Στην πιο απλή μορφή τους, κάθε κόμβος αντιστοιχεί σε partition που είτε χρησιμοποιείται από κάποιο process, είτε είναι "ελεύθερο".

- Τα πεδία ενός κόμβου είναι:
  - ένα ψηφίο (διεργασία ή οπή)
  - ένας αριθμός (αρχική διεύθυνση του partition)
  - ένας αριθμός (μέγεθος του partition)
  - ένας δείκτης στον επόμενο κόμβο



ΔΜ με Linked Lists

- Οι κόμβοι της λίστας είναι **ταξινομημένοι** με βάση τη **διεύθυνση του partition** που αντιπροσωπεύουν.
  - Όταν ένα partition ελευθερώνεται (π.χ., το process τερμάτισε ή έγινε swapped out) πρέπει να εξεταστούν ο επόμενος και ο προηγούμενος κόμβος της λίστας για να **συγχωνευθούν οι (πιθανώς) άδειες οπές (holes)**.
  - Δηλαδή ≥ 2 συνεχόμενες οπές να συγχωνευθούν σε μία.
  - Η ταξινόμηση βοηθάει να γίνει αυτό γρήγορα.



Αλγόριθμοι Memory Allocation

- Εκτελούνται κατά τη διάρκεια **δημιουργίας** ή **swap in** ενός process.
- Υπάρχουν διάφορες μέθοδοι
  - **First-Fit**: Βρίσκει την πρώτη οπή (hole) όπου χωράει το process.
  - **Next-Fit:** Σαν το first-fit, μόνο που ξεκινά το ψάξιμο από την προηγούμενη οπή που βρήκε την προηγούμενη φορά που έτρεξε.
  - **Best-Fit:** Βρίσκει την μικρότερη από τις οπές που «χωράνε» τη διεργασία, ώστε να ελαχιστοποιήσει τον εσωτερικό κατακερματισμό.
  - **Worst-Fit:** Βρίσκει την μεγαλύτερη οπή (άρα, προκαλεί τον μεγαλύτερο εσωτερικό κατακερματισμό).

25

Αλγόριθμοι Memory Allocation

- Η **Best-Fit** είναι προφανώς **πιο αργή** μέθοδος από τη **First-Fit** γιατί απαιτεί περισσότερη αναζήτηση.
- Η **Best-Fit** έχει **χαμηλότερο memory utilization** (δηλ. υψηλότερη σπατάλη μνήμης) σε σχέση με τη **First-Fit**. Αυτό εξηγείται διότι η **Best-Fit** έχει την τάση να αφήνει πολύ μικρές «οπές» όπου δεν μπορούν να χωρέσουν άλλες διεργασίες.
- Η βασική ιδέα της **Worst-Fit** είναι ν' αφήνει όσο το δυνατόν μεγαλύτερα partitions προς μελλοντική χρησιμοποίηση. Πειραματικά, όμως, έχει αποδειχθεί ότι η **Worst-Fit** **δεν είναι αποδοτική**.
- Ένας τρόπος να αυξηθεί η επίδοση των παραπάνω αλγορίθμων ανάθεσης είναι να χρησιμοποιούνται 2 διαφορετικές λίστες, μια με διεργασίες και μια με οπές.
  - Αυτό όμως αυξάνει το κόστος συγχώνευσης όταν partition ελευθερωθεί.
- Αν η λίστα των οπών είναι ταξινομημένη με βάση το μέγεθος, τότε οι **Best-Fit** και **First-Fit** είναι περίπου εξίσου αποδοτικές μέθοδοι.

Αλγόριθμοι Memory Allocation

- Ο αλγόριθμος Quick-Fit:
  - Κρατά **πολλές λίστες οπών**
  - Η κάθε μια αντιπροσωπεύει **οπές με διαφορετικά μεγέθη**
  - Έτσι, **η ανάθεση γίνεται γρήγορα** (βρίσκεται πολύ γρήγορα μια οπή ενός συγκεκριμένου μεγέθους)

- Το μειονέκτημά του έγκειται στο ότι, όταν ένα partition ελευθερωθεί, το **κόστος συγχώνευσης οπών** είναι αυξημένο.

Virtual Memory (Εικονική Μνήμη)

- Τί μπορούμε να κάνουμε όταν
  - **Μία διεργασία είναι μεγαλύτερη της Κ.Μ.;** Δηλαδή το σύνολο των μεγεθών των text, data, και stack υπερβαίνει το μέγεθος της Κ.Μ.;

    --- ή ---
  - **Το σύνολο των διεργασιών που τρέχουν** (βλέπε πολυπρογραμματισμό) απαιτούν χώρο μεγαλύτερο της Κ.Μ.;

- Η λύση είναι η τεχνική **εικονικής μνήμης** (**virtual memory**)
  - Το Λ.Σ. κρατά **μερικά κομμάτια** ενός process στη Κ.Μ., ενώ άλλα κομμάτια βρίσκονται στο swap space.
  - Συνήθως λίγα κομμάτια του text, data, και stack είναι στην Κ.Μ. Αυτά τα κομμάτια αφορούν τις **εντολές και τα δεδομένα που χρειάζονται άμεσα** καθώς και αυτά που αναμένεται να χρησιμοποιηθούν στο **εγγύς μέλλον**.
  - Έτσι, αυτό που γίνεται swap in & out είναι **κομμάτια** των text, data, stack τμημάτων ενός process, και **όχι ολόκληρες διεργασίες**.

Paging (Σελιδοποίηση)

- Η έννοια της **σελίδας** (**page**) αναφέρεται στο κομμάτι εκείνο ενός process που γίνεται **swapped-in** & **swapped-out**, δηλαδή είναι η μονάδα εναλλαγής.
  - Οι λειτουργίες αυτές αναφέρονται και ως **page-in** & **page-out**.

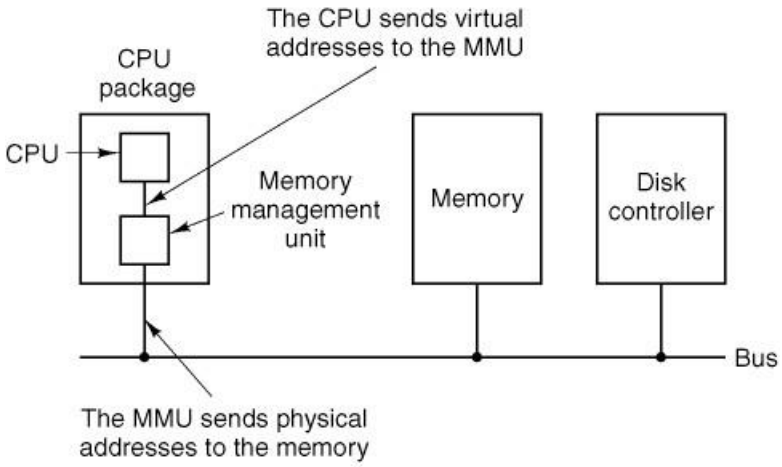
- Το σύνολο των διευθύνσεων μνήμης αποτελεί έναν **χώρο διευθύνσεων** (**address space**).
  - Συνήθως αυτό είναι μεγαλύτερο της Κ.Μ.
  - Δεν υπάρχει μονοσήμαντη αντιστοιχία μεταξύ μιας εικονικής διεύθυνσης (που παράγει ένα πρόγραμμα) και μιας φυσικής διεύθυνσης (physical address). Ακόμα περισσότερο όταν η ΚΜ διαμοιράζεται σε πολλές διεργασίες!

- Συνύπαρξη virtual και physical addresses
  - Τα προγράμματα «μιλούν» με εικονικές διευθύνσεις, δηλ. λειτουργούν στον **χώρο εικονικών διευθύνσεων** (**virtual address space**)
  - Τα memory chips «μιλούν» με φυσικές διευθύνσεις, δηλ. λειτουργούν στον **χώρο φυσικών διευθύνσεων** (**physical address space**)
  - Πριν μία διεύθυνση ενός προγράμματος φτάσει στη μνήμη, πρέπει να μετατραπεί από virtual (εικονική) σε physical (φυσική).

Paging (Σελιδοποίηση)

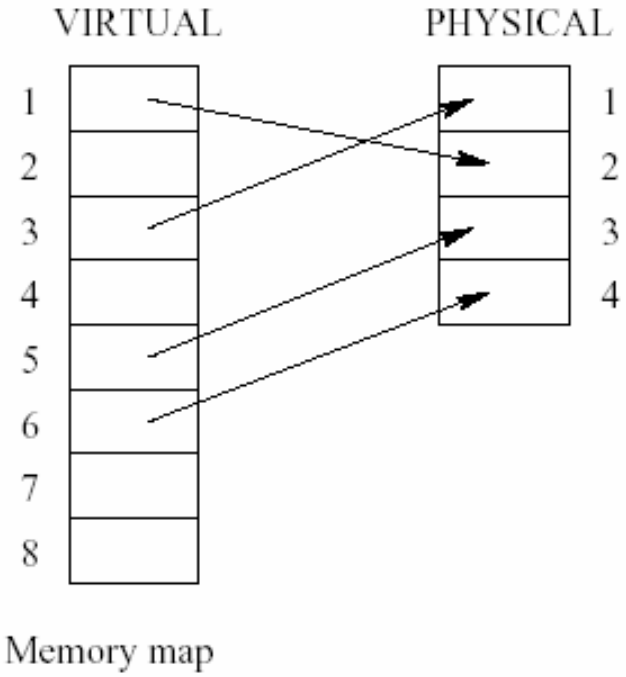
- Χωρίς εικονική μνήμη
  - Ό,τι διεύθυνση παράγεται δίνεται απαράλλαχτη στο memory bus (μετά ίσως από χρήση των καταχωρητών ορίου/βάσης), και φθάνει στο memory chip.

- Με εικονική μνήμη
  - Η παραγόμενη διεύθυνση «μεταφράζεται» σε φυσική πριν δοθεί στο memory bus.
  - Αυτή η μετατροπή γίνεται από ειδικό hardware: **Memory Management Unit** (**MMU**).



Paging (Σελιδοποίηση)

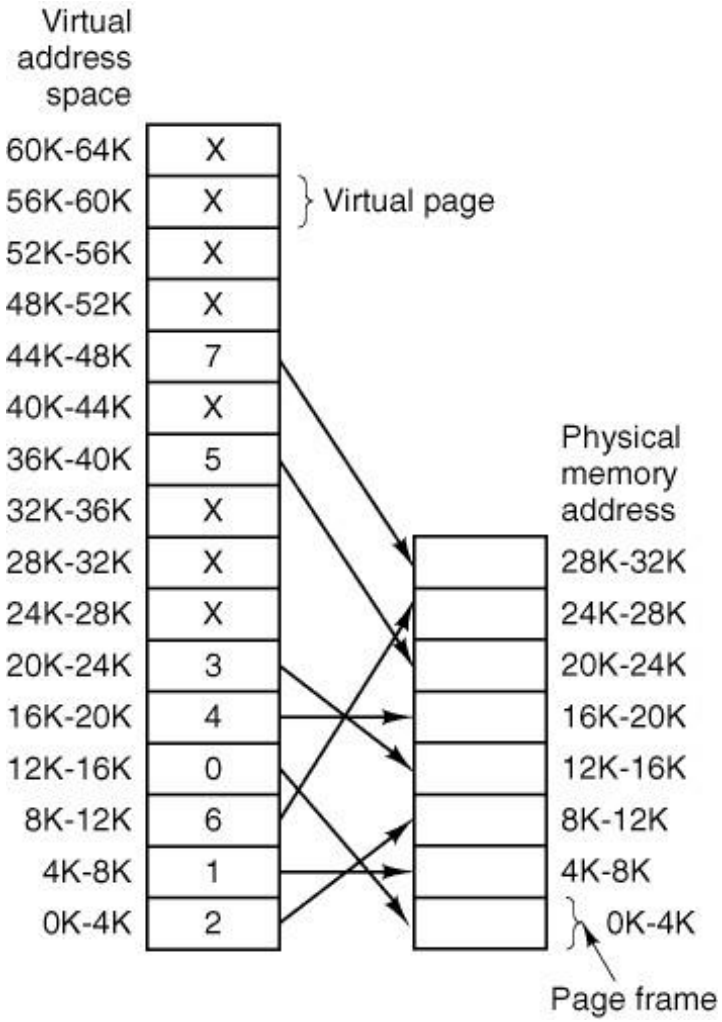
- Το virtual address space διαιρείται σε μονάδες μνήμης που ονομάζονται **pages** (**σελίδες**)
- Αντίστοιχα, η φυσική μνήμη διαιρείται σε **page frames** (**πλαίσια σελίδων**)
- Σελίδες και πλαίσια σελίδων έχουν το ίδιο μέγεθος (συνήθως 512B – 8KB)
- Για να μπορεί να χρησιμοποιηθεί μια σελίδα, αποθηκεύεται σε ένα πλαίσιο σελίδας



Memory map

Paging (Σελιδοποίηση)

- **Παράδειγμα:** Θεωρείστε έναν Η/Υ με
  - Virtual memory **64KB** (16-bit virtual addresses)
  - Physical memory **32KB** (15-bit physical addresses)
  - Page size **4KB**
- Άρα, υπάρχουν
  - **16 pages** στη virtual memory, και
  - **8 page frames** στην κύρια μνήμη
- Το MMU διαχειρίζεται την αντιστοιχία ανάμεσα σε pages και page frames.
  - Για την μετατροπή χρησιμοποιεί ένα **page table** (**πίνακα σελίδων**), που θα δούμε παρακάτω
  - Τα **X** σημαίνουν ότι η σελίδα είναι swapped out, και για να χρησιμοποιηθεί απαιτείται πρώτα page-in.



Page Fault (Σφάλμα Σελίδας)

- Όταν ένα πρόγραμμα προσπαθήσει να προσπελάσει κάποια διεύθυνση που αντιστοιχεί σε σελίδα που δεν βρίσκεται στη μνήμη, έχουμε ένα **page fault** (**σφάλμα σελίδας**).

- Συγκεκριμένα, όταν το MMU βρίσκει ότι η σελίδα που περιέχει τη ζητούμενη διεύθυνση δεν υπάρχει στην Κ.Μ., τότε κάνει ένα trap στο Λ.Σ.
  - Αυτό το trap ονομάζεται **page fault** (**σφάλμα σελίδας**).

- Προκειμένου να εξυπηρετηθεί το memory request του προγράμματος, το Λ.Σ. οφείλει να κάνει page-in για τη συγκεκριμένη σελίδα.
  - Δηλαδή, να τη μεταφέρει από το swap space σε κάποιο page frame της Κ.Μ.

Ενέργειες του Λ.Σ. σε Page Fault

1. Από κάποια kernel data structures για το process που τρέχει, **εντοπίζει τη ζητούμενη σελίδα** (για την οποία προκλήθηκε το page fault) στο swap space.

2. Αν η Κ.Μ. είναι γεμάτη:
   - **Πρέπει να ελευθερώσει ένα page frame** που τώρα καταλαμβάνει άλλη σελίδα.
   - Επιλέγει τη «σελίδα-θύμα» (συνήθως τη λιγότερο χρησιμοποιημένη).
   - Αν η σελίδα-θύμα έχει τροποποιηθεί από το τελευταίο της page-in, τότε την αντιγράφει στη θέση της στο swap space. Γιατί;;;
   - Έτσι έχει δημιουργηθεί ένα ελεύθερο page frame στην Κ.Μ.

3. **Φέρνει τη ζητούμενη σελίδα** σε ένα ελεύθερο page frame.

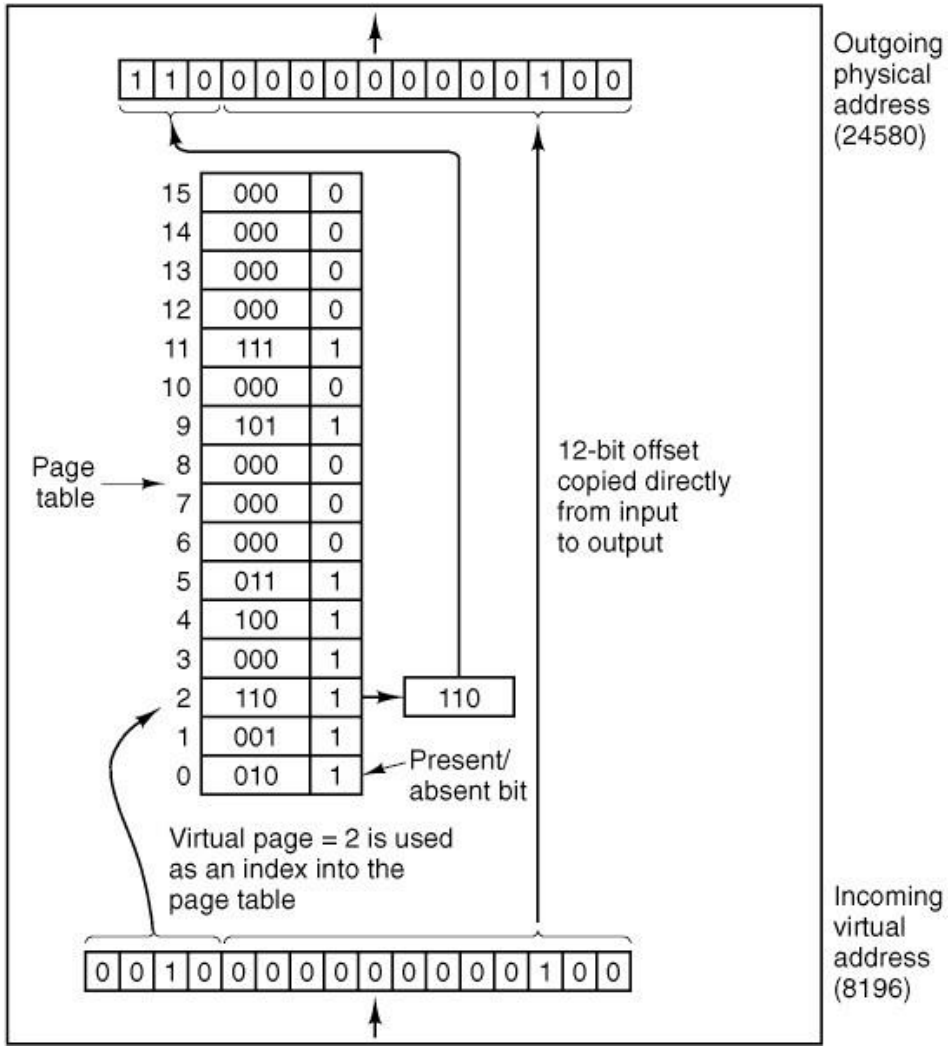
4. Ενημερώνει το **page table** (**πίνακα σελίδων**). Δηλαδή:
   - την εγγραφή που δείχνει στο πλαίσιο που χρησιμοποιήθηκε - δηλ. την εγγραφή της σελίδας «θύμα»: να δείχνει πλέον ότι αυτή η σελίδα δεν είναι χαρτογραφημένη
   - την εγγραφή της ζητούμενης σελίδας, ώστε να δείχνει το κατάλληλο πλαίσιο.

5. Τέλος, **ξανακαλείται η εντολή** που προκάλεσε το page fault.

Page Table (Πίνακας Σελίδων)

- Για να ξέρει το MMU πού βρίσκεται η κάθε σελίδα (δηλ. αν βρίσκεται σε κάποιο page frame της μνήμης και ποιο, ή αν είναι στο swap space), διατηρεί ένα **page table** (**πίνακα σελίδων**), που μετατρέπει page indexes σε page frame indexes.
- Στο παράδειγμά μας, αφού η σελίδα είναι 4KB, απαιτούνται 12 bits για τη διευθυνσιοδότηση *εντός* μιας σελίδας (**offset**).
- Με 64KB virtual address space έχουμε 16 pages, άρα 4 bits για να προσδιορίσουμε ένα page.
- Με 32KB μνήμη, έχουμε 8 page frames, άρα 3 bits για να προσδιορίσουμε το page frame.
- Άρα το page table μεταφράζει 4-bit page indexes σε 3-bit page frame indexes.

	size	address length
Page	4KB	12 bits
Virtual Space	64KB	16 bits
Physical Space	32KB	15 bits



Page Table (Πίνακας Σελίδων)

- Μία εικονική διεύθυνση χωρίζεται σε
  - **Virtual page number** (**εικονικό αριθμό σελίδας**): high-order bits
  - **Offset**: low-order bits
- Ο εικονικός αριθμός σελίδας χρησιμοποιείται σαν δείκτης θέσης στο PT.
- Αν το page βρίσκεται στην κύρια μνήμη, ο αριθμός πλαισίου στην εγγραφή του PT αποτελεί τα σημαντικά ψηφία που αντικαθιστούν τα σημαντικά ψηφία του virtual page number.
- Μόλις προσθέσουμε σε αυτά το offset, παράγεται η σωστή φυσική διεύθυνση -- physical address.

Page Tables (Πίνακες Σελίδων)

- Προκύπτουν όμως 2 σημαντικά προβλήματα:
  - **Μέγεθος**: Ένα page table μπορεί να είναι πολύ μεγάλο.
    - Σημειώστε ότι κάθε process έχει δικό του page table!
  - **Ταχύτητα**: λειτουργία πρέπει να είναι πολύ γρήγορη γιατί χρησιμοποιείται για κάθε διεύθυνση που παράγεται!

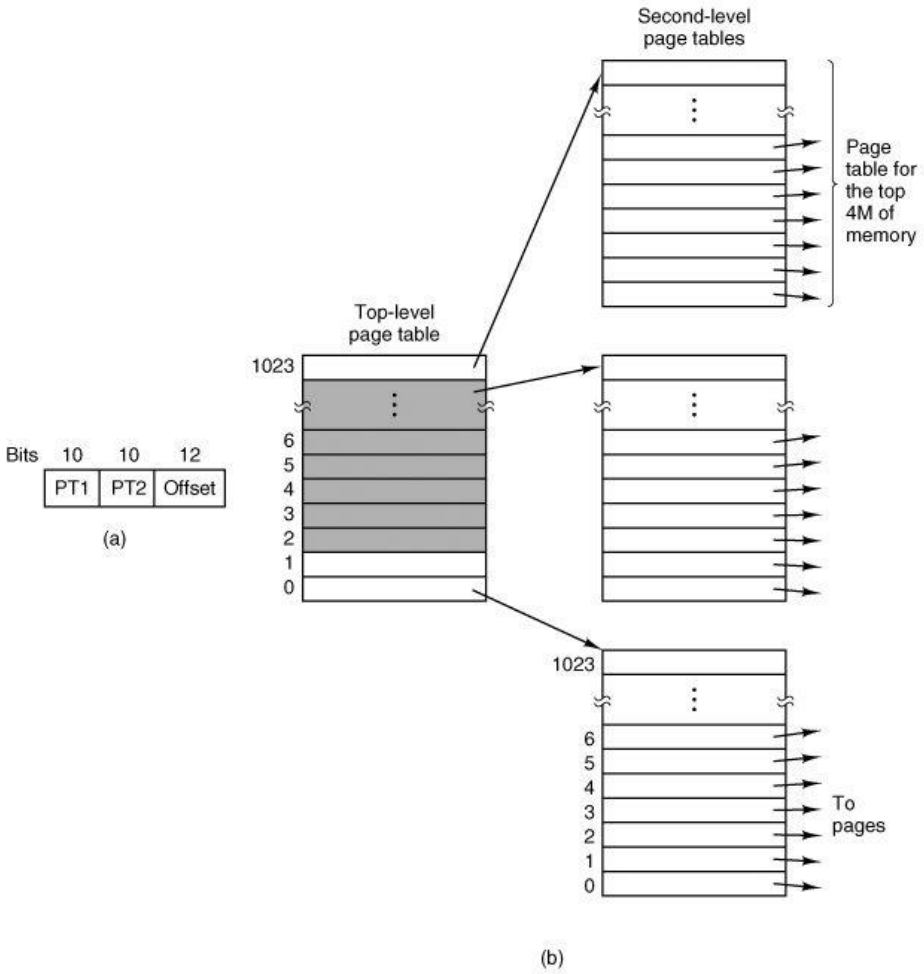
- Αναλογιστείτε ότι σχεδόν όλα τα μοντέρνα συστήματα έχουν ≥32-bit διευθύνσεις
  - άρα ≥ 4GB virtual address space.
  - Αν page size = 4KB, το κάθε PT έχει 1M εγγραφές!

- Επίσης, λόγω του μεγέθους τους, οι πίνακες σελίδων αποθηκεύονται (στην καλύτερη περίπτωση!) στην Κ.Μ. (όχι σε registers)
  - διπλασιάζονται οι προσβάσεις στην ΚΜ που απαιτούνται για να εκτελεστεί μια εντολή…!

- Συνήθως λύσεις άλλοτε βασίζονται σε registers για το mapping ή σε τεχνικές που "μικραίνουν" το μέγεθος των page tables.

Multilevel Page Tables

- Τα **multilevel page tables** (**πολυεπίπεδοι πίνακες σελίδων**) αποσκοπούν στο να τμηματοποιήσουν το page table (που είναι πολύ μεγάλο) έτσι ώστε να απαιτούνται λίγα (και μικρά) κομμάτια του στην Κ.Μ. σε κάθε στιγμή.
- **Παράδειγμα:** 32-bit διευθύνσεις και page table 2 επιπέδων
- Τα bits 0-9 (PT1) χρησιμοποιούνται ως index στο αρχικό PT. Κάθε εγγραφή του, αντί να δείχνει σε κάποιο page, δείχνει σ' ένα page table 2ου επιπέδου στην Κ.Μ.
- Τα bits 10-19 (PT2) χρησιμοποιούνται ως index στο PT 2ου επιπέδου που βρέθηκε στο προηγούμενο βήμα. Αυτό μας οδηγεί στη ζητούμενη σελίδα.
- Τα 12-bits (Offset) χρησιμοποιούνται για να βρεθεί το κατάλληλο byte στο πλαίσιο σελίδας που δείχνει το PT 2ου επιπέδου.



Multilevel Page Tables

- Στο παράδειγμα:
  - Κάθε σελίδα είναι 4KB (αφού το offset είναι 12 bits)
  - Κάθε PT στο 2ο επίπεδο αναπαριστά (χαρτογραφεί) 4MB
  - Το PT στο 1ο επίπεδο αναπαριστά 4GB (32 bits -> 4 GB).

- Έτσι αν τα text, data, και stack segments του process είναι το καθένα <=4MB τότε χρειαζόμαστε να έχουμε στην κύρια μνήμη μόνο 3 PTs του 2ου επιπέδου.

- Αφού το text segment δεν μεγαλώνει, τότε η ανάθεση εικονικών διευθύνσεων στο process μπορεί να γίνει ως εξής:
  - 0-4MB ➔ text
  - 4MB - 8MB ➔ data
  - τα τελευταία 4MB (από τα 4GB) ➔ stack (μεγαλώνει "προς τα κάτω")

- Θα χρειαστούν περισσότερα PTs στην Κ.Μ. μόνο αν το data ή το stack segment μεγαλώσουν πάνω από 4MB το καθένα.

Translation Lookaside Buffer (TLB)

- Αφού τα PTs είναι πολύ μεγάλα για να χωρέσουν σε CPU registers, πώς μπορούμε να μειώσουμε το κόστος πρόσβασης σ' αυτά (στην Κ.Μ.);

- Η απάντηση βασίζεται στη χρήση συσχετιστικής μνήμης (associative memory) που λειτουργεί σαν «κρυφή μνήμη» ή «προσωρινή μνήμη» (cache).

- Η θεμελιώδης παρατήρηση είναι ότι η συμπεριφορά των περισσότερων προγραμμάτων είναι τέτοια ώστε να παρατηρούνται μεγάλοι αριθμοί αναφορών σε λίγες σελίδες (που αλλάζουν αργά με το πέρασμα του χρόνου)
  - π.χ., όλες οι εντολές στην ίδια σελίδα εντολών (βλέπε loops).

- Έτσι χρησιμοποιείται ειδικό υλικό που αποτελείται από καταχωρητές το σύνολο των οποίων καλούνται **TLB** -- **translation lookaside buffer**. Ο αριθμός αυτών των registers είναι μικρός (π.χ., ≤ 64).

Translation Lookaside Buffer (TLB)

- Ο κάθε καταχωρητής έχει την ίδια πληροφορία που υπάρχει σε μια καταχώρηση του page table και επιπλέον έχει και τον αριθμό της εικονικής σελίδας.

- Η MMU εξετάζει πρώτα αν η ζητούμενη εικονική σελίδα βρίσκεται στην κρυφή μνήμη TLB (όλοι οι registers εξετάζονται παράλληλα).
  - Αν ναι, τότε αποφεύγεται η πρόσβαση του PT στη Κ.Μ.
  - Αν όχι, τότε προσπελαύνεται το PT και η κατάλληλη PT εγγραφή αντικαθιστά κάποια άλλη στην TLB.

- Σε μερικές σύγχρονες μηχανές, η διαχείριση σφαλμάτων στο TLB γίνονται από το ΛΣ.

- Προσέξτε ότι:
  - αν δρομολογηθεί άλλη διεργασία (process/context switch), τότε θα πρέπει να "μηδενισθεί" το TLB!
  - Η έννοια και χρήση κρυφής μνήμης είναι καθοριστικής σημασίας για Συστήματα Λογισμικού!

Page Table Entries

- Συνήθως η ακριβής δομή εξαρτάται από τη μηχανή. Αλλά, οι πιο πολλές εγγραφές έχουν τις εξής πληροφορίες:
- **Προστασία -- Protection bits:** ορίζουν αν η σελίδα αυτή μπορεί να γίνει read/write/execute
- **Ενημέρωση -- Modify bit:** δηλώνει αν έχει ενημερωθεί η σελίδα από τότε που ήρθε στην Κ.Μ.
- **Αναφορές -- Reference bits:** δηλώνουν πόσες φορές το τελευταίο χρονικό διάστημα έγινε αναφορά σ' αυτή τη σελίδα.
- **Παρούσα -- Present bit:** δηλώνει αν η σελίδα βρίσκεται στη ΚΜ.
- **Δείκτης πλαισίου -- Page Frame bits:** ορίζουν σε ποιο frame της ΚΜ βρίσκεται αυτή η σελίδα.

Page Table Entries

- Προσέξτε ότι η διεύθυνση της σελίδας στο swap space (δίσκο) δεν βρίσκεται στην εγγραφή του PT...

- **Modify bit:** χρησιμοποιείται όταν αποφασίζεται να εκδιωχθεί μια σελίδα απ' την ΚΜ για να ελευθερωθεί χώρος για μια άλλη σελίδα.
  - Αν το bit αυτό είναι 1, τότε η σελίδα γράφεται στο swap space - αλλιώς δεν χρειάζεται αυτή η εγγραφή στον δίσκο – μια χρονοβόρα εντολή.

- **Reference bit:** Χρησιμοποιείται για ν' αποφασιστεί ποια σελίδα ("θύμα") θα αντικατασταθεί.
  - Το πρόσφατο παρελθόν, οδηγός του μέλλοντος ...

Inverted Page Tables

- Αν ο χώρος εικονικών διευθύνσεων είναι πολύ μεγάλος, (πχ 2^{64}) τα PTs πρέπει να είναι εξαιρετικά μεγάλα (πχ 2^{52} εγγραφές για πλαίσια μεγέθους 4KB).

  ➔ απαιτείται άλλη προσέγγιση!

- Τα **inverted page tables** (**ανεστραμμένοι πίνακες**) είναι μια λύση:
  - Υπάρχει ένα μόνο PT, με μια εγγραφή για κάθε page frame, που δείχνει ποια σελίδα ποιας διεργασίας φιλοξενεί (πχ 2^{18} εγγραφές).
  - Σε κάθε αναφορά μνήμης, δηλαδή σε εικονική σελίδα, χρησιμοποιείται μια συσκευή TLB, όπως πριν.
  - Αν δεν βρεθεί στο TLB, για να βρούμε αν υπάρχει η σελίδα στη μνήμη γρήγορα, χρησιμοποιείται ένα πίνακας κατακερματισμού (hash table)
  - Κάθε φορά που εισέρχεται μια σελίδα στη μνήμη, με βάση τον αριθμό εικονικής σελίδας, εισέρχεται και στον πίνακα κατακερματισμού. Επίσης, πρέπει να ενημερώνεται και το TLB.

Page Replacement

- Όταν σημειώνεται ένα σφάλμα σελίδας τότε το Λ.Σ. πρέπει (μερικές φορές - αν δεν υπάρχει ελεύθερο πλαίσιο) να διαλέξει μια σελίδα (victim) "θύμα" που θα αντικατασταθεί από τη ζητούμενη σελίδα.

- Αν το "θύμα" είναι «βρώμικο» (dirty) (δηλ. έχει ενημερωθεί όσο ήταν στη Κ.Μ.) τότε το Λ.Σ. πρέπει να την ξαναγράψει πάλι στο swap space. Μια σελίδα είναι dirty αν η εγγραφή στο ΠΣ που δείχνει σ' αυτήν έχει το ψηφίο M = 1.

- Προφανώς η απόδοση του συστήματος είναι καλύτερη αν το "θύμα" δεν είναι μια πολυ-χρησιμοποιημένη σελίδα.

- Το μεγάλο πρόβλημα κόστους πηγάζει από την ανάγκη πρόσβασης στον δίσκο για αντικατάσταση σελίδων – μια χρονοβόρα διαδικασία!

Αντικατάσταση Σελίδων

Η βέλτιστη στρατηγική:

- Αν ξέραμε μετά από πόσο χρόνο (π.χ., αριθμό εντολών) η κάθε σελίδα στην ΚΜ θα **επαναπροσπελαστεί**, τότε θα μπορούσαμε να διαλέξουμε σαν θύμα τη σελίδα που δεν θα χρειαστεί για το μεγαλύτερο χρονικό διάστημα.
- Όμως, γενικά σε πραγματικά συστήματα, τέτοιου είδους πληροφορία δεν υπάρχει, ούτε μπορεί να παραχθεί.

- Σε μερικές περιπτώσεις είναι δυνατόν να υποβληθεί ένα πρόγραμμα σε κάποιου είδους «**ανιχνευτή**» («tracer») το οποίο καταχωρεί την συμπεριφορά του προγράμματος αναφορικά με τις προσπελάσεις σε σελίδες.

- Αυτή η πληροφορία χρησιμοποιείται όταν το πρόγραμμα τρέχει πραγματικά και έτσι μπορούμε να υλοποιήσουμε τη βέλτιστη στρατηγική.

Page Replacement Algorithms

Στις επόμενες διαφάνειες θα δούμε τους παρακάτω αλγόριθμους αντικατάστασης σελίδας:

- **Not Recently Used** (**NRU**)
- Αλγόριθμους βασισμένους σε FIFO:
  - **Second Chance**
  - **Clock**
  - **WS-Clock** (πιο μετά)
- **Least Recently Used** (**LRU**)
- **Not Frequently Used** (**NFU**)

Αλγόριθμος Not Recently Used (NRU)

- Συνοπτικά, αυτή η μέθοδος διαλέγει σαν θύμα μια σελίδα που δεν έχει χρησιμοποιηθεί πρόσφατα.

- Πρέπει λοιπόν να υπάρχουν πληροφορίες σχετικά με το πόσο πρόσφατα χρησιμοποιήθηκε η κάθε σελίδα. Στον αλγόριθμο NRU χρησιμοποιούνται τα **R και M ψηφία** που συνήθως υπάρχουν στις εγγραφές των Page Tables:

  - Το R γίνεται 1 όταν η αντίστοιχη σελίδα προσπελαύνεται.

  - Το M γίνεται 1 όταν η αντίστοιχη σελίδα ενημερώνεται.

  - Τις ενημερώσεις των R, M bits τις κάνει συνήθως το hardware.

Αλγόριθμος Not Recently Used (NRU)

- Όταν αρχίζει ένα process τότε όλα τα R, M bits των εγγραφών του PT παίρνουν την τιμή 0.
- Σε κάθε clock interrupt το R bit γίνεται πάλι 0. Έτσι αν εξετάσουμε μια PT εγγραφή και δούμε το R = 1 ➔ ότι υπήρξε αναφορά σ' αυτή τη σελίδα μετά το τελευταίο clock tick.
- Όταν σημειώνεται ένα page fault: Το Λ.Σ. χρησιμοποιεί 4 κατηγορίες σελίδων με βάση τις τιμές των R & M bits.

K1 R = 0 M = 0
K2 R = 0 M = 1 **(υπάρχει αυτή η κατηγορία; ΝΑΙ)**
K3 R = 1 M = 0
K4 R = 1 M = 1

- Η σελίδα-θύμα επιλέγεται με σειρά προτεραιότητας K1 → K2 → K3 → K4
  - Γιατί προτιμάει θύματα από K1 και όχι από K2;
- Ο NRU είναι απλός και έχει καλή απόδοση.

Αλγόριθμοι βασισμένοι σε FIFO: Δεύτερη Ευκαιρία

- Προφανώς η "καθαρόαιμη" FIFO στρατηγική έχει προβλήματα. Γιατί;

Ο Αλγόριθμος Δεύτερης Ευκαιρίας (Second - Chance)

- Βασίζεται στη στρατηγική FIFO: εξετάζει σελίδες αρχίζοντας από την πιο παλιά.
- Αλλά, αντί να αντικαθιστά την πιο παλιά σελίδα, εξετάζει το R bit.
- Αν R==0 τότε την αντικαθιστά.

  Αν R==1 τότε
  - κάνει το bit R = 0
  - η σελίδα τοποθετείται στο τέλος της λίστας (δηλ. "βαφτίζεται" σαν η πιο πρόσφατη σελίδα στη ΚΜ)
  - η αναζήτηση θύματος συνεχίζεται στην επόμενη πιο παλιά σελίδα.

  Αν R==1 για όλες τις σελίδες, τότε second-chance είναι στην ουσία FIFO (Γιατί ;)

Αλγόριθμοι βασισμένοι σε FIFO: Ρολόι

Ο Αλγόριθμος «Ρολόι» (Clock)

- Το σημαντικότερο μειονέκτημα του second-chance είναι ότι συνεχώς ενημερώνει τη λίστα - χρονοβόρα διαδικασία.
- Στον αλγόριθμο **Ρολόι**, οι σελίδες-πλαίσια οργανώνονται σε μια **κυκλική λίστα** που προσομοιώνει ένα ρολόι, δηλαδή υπάρχει ένας **δείκτης** που δείχνει σε μια σελίδα.

- Όταν σημειώνεται ένα page fault:
  - εξετάζεται η σελίδα που δείχνει ο δείκτης:
  - Αν R=0, τότε αυτή είναι το θύμα & ο δείκτης προωθείται να δείχνει στην επόμενη σελίδα.
  - Αν R=1, τότε R ← 0 & ο δείκτης δείχνει στην επόμενη σελίδα και η αναζήτηση συνεχίζεται.

Αλγόριθμος Least-Recently Used (LRU)

Βασική ιδέα:

- σελίδες που προσπελάστηκαν στο πρόσφατο παρελθόν θα επανα-προσπελασθούν και στο εγγύς μέλλον (και αντιστρόφως). Έτσι,
  - το θύμα είναι η σελίδα που δεν έχει χρησιμοποιηθεί για το μεγαλύτερο χρονικό διάστημα.

- Μια προσέγγιση υλοποίησης είναι με μια λίστα όλων των σελίδων στην ΚΜ με την **πιο-πρόσφατα-χρησιμοποιημένη σελίδα στην κορυφή της λίστας** και την LRU σελίδα στο τέλος.

- Απλή μέθοδος, αλλά η λίστα χρειάζεται ενημέρωση σε κάθε αναφορά στη μνήμη (που γίνεται > 1 φορές για κάθε εντολή) ➔ το κόστος είναι απαγορευτικό!

Αλγόριθμος Least-Recently Used (LRU)

- Μια προσέγγιση αποφυγής του παραπάνω κόστους είναι να χρησιμοποιηθεί **ειδικό hardware**.
- Μια μέθοδος χρησιμοποιεί έναν 64-bit counter η τιμή του οποίου αυξάνεται σε κάθε εντολή.
- Επίσης, κάθε εγγραφή του PT έχει 64 bits παραπάνω, έτσι ώστε μετά από κάθε αναφορά σε μια σελίδα της ΚΜ **η τιμή του counter να αποθηκεύεται στην αντίστοιχη εγγραφή** του PT.

➔ **η μέθοδος διαλέγει σαν θύμα τη σελίδα της οποίας η εγγραφή στο PT έχει τη μικρότερη τιμή του counter.**

- Το παραπάνω ειδικό hardware κοστίζει, φυσικά, και μπορεί να μην υπάρχει/προσφέρεται από κάποιο σύστημα. Έτσι πολλοί σχεδιαστές Λ.Σ. καταφεύγουν σε **λύσεις software** με τις οποίες προσπαθούν να προσομοιώσουν την στρατηγική LRU.

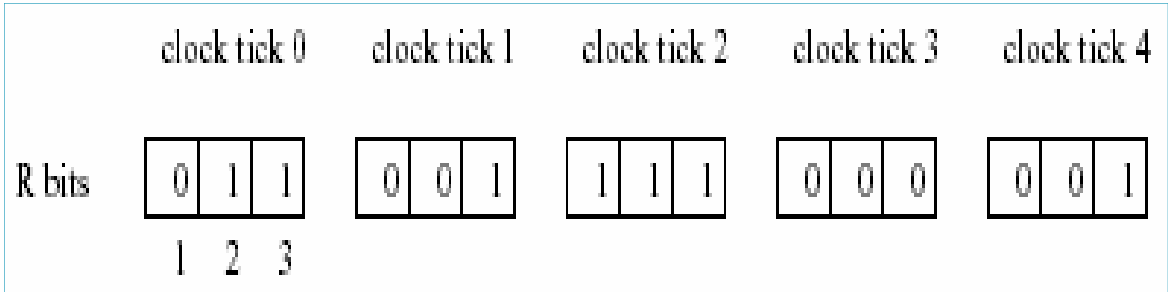
Αλγόριθμος Not-Frequently-Used (NFU)

- Πρόκειται για μια καλή προσομοίωση του LRU.
- **Με κάθε σελίδα συνδέεται και ένας μετρητής** (software counter) -- αρχικά έχει την τιμή 0.
- Με κάθε clock interrupt το Λ.Σ. προσπελαύνει όλες τις σελίδες στην Κ.Μ. Για κάθε σελίδα:
  - ο counter «ολισθαίνει δεξιά» (**shift right)**
  - η τιμή του counter τότε προσαυξάνεται με την τιμή του R (0 ή 1) της ίδιας σελίδας.
  - Το R προστίθεται στο πιο σημαντικό ψηφίο του counter.
- Όταν σημειώνεται ένα σφάλμα σελίδας, **η σελίδα με τη μικρότερη τιμή του μετρητή επιλέγεται σαν θύμα**.

Αλγόριθμος Not-Frequently-Used (NFU)

- Ο παραπάνω αλγόριθμος είναι μια συγκεκριμένη μορφή NFU που ονομάζεται **αλγόριθμος γήρανσης** (**aging**).

  π.χ.



Counters	000000	000000	100000	010000	001000
	100000	010000	101000	010100	001010
	100000	110000	111000	011100	101110

Αλγόριθμος Not-Frequently-Used (NFU)

- Αν μια σελίδα δεν έχει χρησιμοποιηθεί για N ticks
  ➔ τα πρώτα N bits = 0.

Αυτός ο αλγόριθμος διαφέρει κατά 2 τρόπους από τον LRU:

1. Στο παράδειγμα: Αν και η σελίδα 1 είναι το θύμα, είναι πιθανό να είχε χρησιμοποιηθεί μετά τη σελίδα 2 στο clock tick 2.

2. Υπάρχει ένας πεπερασμένος αριθμός (για τον counter) ψηφίων (π.χ., αν όλα τα 6 ψηφία 2 σελίδων είναι 0, τότε διαλέγουμε στην τύχη μεταξύ τους).
   Αυτό όμως δεν δημιουργεί ιδιαίτερα προβλήματα!

Σχεδιασμός Paging Systems

- Όταν αρχίζει ένα process το Λ.Σ., μπορεί να επιλέξει:
  - να μην φέρει καμία σελίδα του στη Κ.Μ. (δηλαδή να αφήσει το σύστημα να φέρει την κάθε σελίδα από τον δίσκο όταν υπάρξει η πρώτη αναφορά σ' αυτήν). Αυτή η στρατηγική ονομάζεται **σελιδοποίηση κατ' απαίτηση** (**on-demand paging**).
  - να φορτώσει κάποιες σελίδες του process από τον δίσκο πριν το process αρχίσει να τρέχει. Αυτή η στρατηγική ονομάζεται **προσελιδοποίηση** (**prepaging**).

- Συνήθως η συμπεριφορά των προγραμμάτων που τρέχουν χαρακτηρίζεται από διάφορες φάσεις. Σε κάθε φάση οι αναφορές μνήμης που γίνονται απ' το πρόγραμμα εντοπίζονται σ' ένα μικρό σύνολο σελίδων του (locality of reference) που είναι ένα πολύ μικρό υποσύνολο του address space του προγράμματος.

Σχεδιασμός Paging Systems

- Αυτό το υποσύνολο των σελίδων, που χρησιμοποιεί ένα πρόγραμμα στη τωρινή του φάση, ονομάζεται **λειτουργικό σύνολο** (**working set**).

- Αν όλο το working set ενός process βρίσκεται στην ΚΜ, τότε το process αυτό θα δημιουργεί **page faults μόνο κατά τη μετάβαση** στην επόμενη φάση.
  - Αυτό είναι και το ζητούμενο!

- Αν το working set είναι πολύ μεγάλο και δεν χωράει στην διαθέσιμη "ελεύθερη" φυσική μνήμη, τότε το πρόγραμμα θα τρέξει πολύ αργά.

Σχεδιασμός Paging Systems

Το φαινόμενο thrashing «αλώνισμα»: Όταν ένα process σπαταλάει ένα μεγάλο μέρος του συνολικού χρόνου σε page faults (π.χ., ένα process έτρεξε για 15 secs, τα 14 από τα οποία ήταν για page faults).

Προσέξτε:

- κάθε εντολή παίρνει χρόνο < 1 μs
- κάθε πρόσβαση στον δίσκο παίρνει χρόνο ~= 10 ms

Το Μοντέλο του Λειτουργικού Συνόλου (Working Set - WS)

Λ.Σ. που εφαρμόζουν αυτό το μοντέλο **πρέπει να**:

- ξέρουν **ποιο είναι το working set** κάθε διεργασίας, και
- πριν την τρέξουν (είτε στην αρχή, είτε μετά από swap in) να έχουν φέρει το WS στην Κ.Μ., (δηλ. εφαρμόζουν **prepaging**).

Σχεδιασμός Paging Systems

- Πώς μπορεί το Λ.Σ. **να ξέρει ποιο είναι το WS**;

- Μία μέθοδος βασίζεται στον αλγόριθμο NFU με **γήρανση** (**aging**) για την αντικατάσταση σελίδων που είδαμε πριν. Το WS ενός process αποτελείται απ' όλες τις **σελίδες των οποίων τα πρώτα N ψηφία των μετρητών δεν είναι 0**. Δηλ. αν στα τελευταία N ticks του ρολογιού έχει σημειωθεί αναφορά στη σελίδα, τότε η σελίδα ανήκει στο WS της διεργασίας.

- Το μοντέλο WS μπορεί να συνδυαστεί με τον αλγόριθμο **clock**:

  Αντί να επιλέγεται ως θύμα η σελίδα στην οποία δείχνει ο δείκτης αν το R = 0:

  - πρώτα εξετάζεται **αν η σελίδα αυτή ανήκει στο WS**.
  - Αν ναι, τότε δεν επιλέγεται ως θύμα, και συνεχίζεται η αναζήτηση.
  - Αλλιώς, επιλέγεται ως θύμα.

  Αυτός ο αλγόριθμος ονομάζεται **WS_Clock**

Τοπική και Καθολική Αντικατάσταση (Global και Local Page Replacement)

- **Θεμελιώδης ερώτηση**: Όταν σημειώνεται ένα σφάλμα σελίδας, και αναζητούμε ένα θύμα:
  - θα περιορίσουμε την έρευνά μας μόνο στις σελίδες της διεργασίας που προκάλεσε το σφάλμα;
  - ή θα εξετάσουμε όλες τις σελίδες της Κ.Μ. ανεξάρτητα από το σε ποια διεργασία ανήκουν;

- Στην πρώτη περίπτωση η στρατηγική ονομάζεται **«τοπική» local** ενώ στη δεύτερη περίπτωση ονομάζεται **«καθολική» global.** Με καθολικούς αλγόριθμους, ο αριθμός των πλαισίων που έχουν δοθεί σε κάθε διεργασία μεταβάλλεται με το χρόνο.
  - Συνήθως προτιμούνται οι καθολικοί αλγόριθμοι διότι συνήθως το μέγεθος του **WS αλλάζει με την πάροδο του χρόνου**.

- Με τοπικούς αλγόριθμους
  - **Αν το WS μικρύνει** τότε σπαταλάται (στην ουσία) χώρος της Κ.Μ., έλλειψη του οποίου δημιουργεί page faults σ' άλλες διεργασίες.
  - Αντίθετα, **αν το WS μεγαλώσει**, μπορεί να συμβεί **thrashing.**

Τοπική και Καθολική Αντικατάσταση (Global και Local Page Replacement)

- Οι καθολικοί αλγόριθμοι δεν έχουν αυτά τα προβλήματα. Όμως πρέπει να αποφασίσουν πόσα πλαίσια σελίδων να δώσουν σε κάθε process.

- Συνήθως, **ανάλογα με το μέγεθος** του process, τού ανατίθεται και ένας αριθμός πλαισίων.
  - Αλλά και αυτό μπορεί να δημιουργήσει **thrashing**.

- Το πρόβλημα μπορεί να λυθεί μετρώντας τη **συχνότητα σφαλμάτων σελίδων** (**page fault frequency**) κάθε process και χρησιμοποιώντας 2 κατώφλια: **High PFF** και **Low PFF.** Όταν μετριέται το PFF ενός process p:

  - Αν PFF(p) > H_PFF τότε το Λ.Σ. δίνει πιο πολλά πλαίσια στο p.

  - Αν PFF(p) < L_PFF τότε το Λ.Σ. μπορεί να αφαιρέσει πλαίσια από το process p.

Μέγεθος Σελίδας

- Μικρές ή μεγάλες σελίδες ;

  Η τελευταία σελίδα κάθε segment (text, data, ή stack) είναι **κατά μέσο όρο η μισή άδεια ➔ μικρές σελίδες είναι καλύτερα**.

- Επίσης όταν μια σελίδα είναι μεγάλη τότε μπορεί το WS του process να χρειάζεται μόνο ένα υποσύνολο της πληροφορίας στη σελίδα και έτσι σπαταλάται μνήμη ➔ **μικρές σελίδες είναι καλύτερα**.

- Από την άλλη μεριά:
  - μικρές σελίδες ➔ μεγάλα PTs ➔ σπαταλάται μνήμη
  - μικρές σελίδες ➔ μη αποδοτικό I/O κατά τη διάρκεια page faults.
- Αυτό συμβαίνει γιατί, όταν οι σελίδες γίνονται swapped in/out, ο μεγαλύτερος χρόνος που απαιτείται οφείλεται στη χρησιμοποίηση του δίσκου για **seek** (δηλ. να τοποθετηθεί η κεφαλή του δίσκου στη σωστή σελίδα) ➔ **μεγάλες σελίδες είναι καλύτερα**!

Ζητήματα Υλοποίησης

ΛΣ και Σελιδοποίηση

- Δημιουργία διεργασίας:
  - Δέσμευση ΚΜ για το PT και αρχικοποίηση
  - Δέσμευση στον δίσκο χώρου για το swap file και αρχικοποίηση (εντολές και δεδομένα προγράμματος).
- Χρονοπρογραμματισμός διεργασίας:
  - Αρχικοποίηση TLB, ενημέρωση για ποιο είναι το τρέχον PT, προσελιδοποίηση(;)
- Σφάλμα σελίδας:
  - Εύρεση στον δίσκο ζητούμενης σελίδας, εύρεση θύματος, page_out(;), page_in,
- Τερματισμός διεργασίας: Απελευθέρωση χώρου για PT, swap file στον δίσκο.

Ζητήματα Υλοποίησης

Πισωγύρισμα Εντολής (Instruction Back-Up)

- Έστω μια εντολή που προκαλεί page fault.
- Το page fault μπορεί να προκλήθηκε όταν έγινε αναφορά στη σελίδα εντολών, ή σε μια από τις σελίδες που περιέχουν τις παραμέτρους--δεδομένα.
- Μόλις το Λ.Σ. φέρει τη ζητούμενη σελίδα, **πρέπει να επαναληφθεί η εντολή** από την αρχή

  ➔ το Λ.Σ. πρέπει να βρει που είναι το 1ο byte της εντολής.

  Συνήθως οι εντολές είναι πολλά bytes (π.χ., 2 bytes για opcode και 2 για κάθε παράμετρο) ➔ δεν είναι εύκολη υπόθεση.
- Μερικά συστήματα είτε κρατούν αυτή τη πληροφορία σε ειδικούς registers, είτε ο μικροκώδικας (microcode) της εντολής τη βάζει (push) στο stack, όπου το Λ.Σ. τη βρίσκει!

Ζητήματα Υλοποίησης

Κλείδωμα Σελίδων (Page Locking)

- Υπάρχει **αλληλεπίδραση** μεταξύ υποσυστήματος **I/O** και **virtual Memory** (Ε/Ε και εικονικής μνήμης).
- Θεωρήστε ένα process, P1, που εκτελεί **read (fd, &buf, …)**. Το Λ.Σ. δίνει την εντολή στον δίσκο να φέρει το ζητούμενο disk block στη σελίδα(ες) Σ1, Σ2,... όπου βρίσκεται ο buf.
- Το P1 μπλοκάρει και το CPU δίνεται στο P2. Το P2 προκαλεί page faults και σαν θύμα επιλέγεται μια από τις σελίδες Σ1, Σ2,... όπου βρίσκεται ο buf της P1 και τη φορτώνει με την επιθυμητή σελίδα της P2.
- Αργότερα ο δίσκος φέρνει την απάντηση από τον δίσκο για την P1 και χρησιμοποιώντας DMA την αποθηκεύει στη διεύθυνση όπου τώρα υπάρχει η σελίδα της P2 ... καταστροφή!!!
- Η λύση στο πρόβλημα είναι να χρησιμοποιηθεί ένα **lock bit** για κάθε πλαίσιο. Όταν το lock bit είναι 1 τότε αυτή η σελίδα δεν μπορεί να επιλεχθεί σαν θύμα από τον page manager.
- Έτσι, αν όλες οι σελίδες που εμπλέκονται σε I/O έχουν το lock bit = 1 τότε το παραπάνω πρόβλημα αποφεύγεται.

Ζητήματα Υλοποίησης

Διαμοιρασμός Σελιδων (Page Sharing)

- Συνήθως >1 διεργασίες τρέχουν το ίδιο πρόγραμμα (δηλ., ίδιο code segment)
- Άλλες φορές, διαφορετικές διεργασίες έχουν shared memory.

➔ συμφέρει οι σελίδες να διαμοιράζονται, αντί να υπάρχει ένα αντίγραφο του κοινού προγράμματος για κάθε διεργασία.

➔ όταν ένα process γίνεται swap out (ή τελειώνει) πρέπει οι σελίδες (text) που χρησιμοποιούνται από άλλες διεργασίες να μην γίνουν swap out (και το swap space να μην ελευθερωθεί)

Ζητήματα Υλοποίησης

Swap Space

- Συνήθως είναι ένα ειδικό partition στον δίσκο.
- Υπάρχει ένα **free list** έτσι ώστε καινούργιες διεργασίες να βρίσκουν swap space.
- Όταν ένα process αρχίζει, το Λ.Σ. τού αναθέτει τόσο swap space όσο είναι και το μέγεθός του.
- Στην εγγραφή του Process Table του Λ.Σ. για το process P υπάρχει **ένα πεδίο που δείχνει στη διεύθυνση στον δίσκο για το process P (swap space)**.
- Έτσι με το virtual page number και αυτή τη διεύθυνση βρίσκουμε πού είναι η κατάλληλη σελίδα στον χώρο εναλλαγής.
- Συνήθως τα segments όπως το stack και το data ενός process έχουν διαφορετικά swap spaces, λόγω κυμαινόμενου μεγέθους.

Ζητήματα Υλοποίησης

Ο Δαίμονας Σελιδοποίησης (paging daemon)

- Σε πολλά συστήματα δαίμονες σελιδοποίησης (ειδικές διεργασίες) **διασφαλίζουν ότι υπάρχουν "αρκετά" ελεύθερα πλαίσια σελίδων στη μνήμη**.
- Περιοδικά, ο δαίμονας σελιδοποίησης (paging daemon) ξυπνάει, ελέγχει αν υπάρχουν αρκετά ελεύθερα πλαίσια.
- Αν ναι, ξανακοιμάται.
- Αν όχι, τρέχει τον αλγόριθμο αντικατάστασης σελίδων, μέχρι αρκετά frames να γίνουν ελεύθερα.

➔ Έτσι το σύστημα έχει καλύτερη απόδοση. Γιατί ;

Χειρισμός Λάθους (Page Fault Handling)

1. Έχουμε παγίδα στον πυρήνα (kernel trap). Ο kernel σώζει PC και διεύθυνση εντολής (1ο byte).
2. Ο kernel σώζει general-purpose regs.
3. Βρίσκει ποια εικονική σελίδα προκάλεσε page fault (μπορεί να χρειαστεί να κάνει parse την εντολή).
4. Αν οι έλεγχοι προστασίας περάσουν, το Λ.Σ. προσπαθεί να βρει ελεύθερο frame (τρέχοντας ίσως τον page replacement αλγόριθμο).
5. Αν το θύμα έχει M=1, τότε ζητάει απ' τον disk controller το γράψιμο της σελίδας και δρομολογεί άλλο process (αφού κλειδώσει τη σελίδα).

Χειρισμός Λάθους (Page Fault Handling)

6. Αφού γραφτεί η σελίδα, ο kernel βρίσκει που στο swap space είναι η ζητούμενη σελίδα και ζητά απ' τον disk controller να την φέρει (και δρομολογεί άλλη διεργασία).

7. Όταν έρθει η σελίδα τα PTs ενημερώνονται και η σελίδα ξεκλειδώνεται.

8. Ο kernel βρίσκει το 1ο byte της εντολής που προκάλεσε το page fault.

9. Το process που προκάλεσε το page fault ξανατρέχει (ο έλεγχος επιστρέφει στην assembly-code ρουτίνα του βήματος 2). Η assembly ρουτίνα επανατοποθετεί τις κατάλληλες τιμές των general-purpose CPU regs και επιστρέφει.

10. Το process τώρα τρέχει σαν το page fault να μην είχε συμβεί.