ΗΥ-225: Οργάνωση Υπολογιστών
Άνοιξη 2017

Τμ. Επ. Υπολογιστών


# Σειρά Ασκήσεων 12: Εικονική Μνήμη (Virtual Memory)

Προθεσμία έως Πέμπτη 11 Μαΐου 2017 (βδ. 12.3) ώρα 23:59 (από βδ. 11)

**Βιβλίο** (Ελληνικό, έκδοση 4) - διαβάστε τα εξής:

- Χειρισμός Εξαιρέσεων και Διακοπών στο υλικό ενός επεξεργαστή: §4.9, σελ. 448 - 455 (το μεσαίο κομάτι, σελ. 452-454, μπορείτε να το περάσετε "στα γρήγορα").
- Εικονική Μνήμη: §5.4, σελ. 571 - 602, εκτός από τα εξής ενδιάμεσα κομάτια:
  – σελ. 585-590 (συνδυασμός εικονικής μνήμης, κρυφής μνήμης, και TLB): διαβάστε τις "στα γρήγορα",
  – σελ. 597-599 (λεπτομέρειες γιά TLB miss και page fault exceptions): ξεφυλλίστε τις "στα πεταχτά".
- §5.5, σελ. 602-611 (ένα κοινό πλαίσιο γιά ιεραρχίες μνήμης): χρήσιμο θέμα γιά περαιτέρω κατανόηση, αλλά εκτός ύλης αυτού του μαθήματος.
- Εικονικές Μηχανές (Virtual Machines) - §5.6, σελ. 611-617: μπορείτε να τις περάσετε "στα γρήγορα", αν και είναι σημαντικό θέμα της σημερινής τεχνολογίας που πρέπει να ξέρετε περί τίνος πρόκειται γιά την περαιτέρω σταδιοδρομία σας.

## 12.1 Εικονική Μνήμη: Στόχοι της

Ο πρώτος και βασικότερος σκοπός του μηχανισμού της Εικονικής Μνήμης (Virtual Memory) είναι να δίνει την εντύπωση σε πολλαπλούς "χρήστες" ("προγράμματα") ότι έχουν **"όλη τη μηχανή δική τους"**. Το βασικότερο χαρακτηριστικό του ότι το κάθε πρόγραμμα "νομίζει ότι έχει όλη τη μηχανή δική του" είναι ότι το κάθε πρόγραμμα θεωρεί ότι δικαιούται να χρησιμοποιήσει την οιαδήποτε διεύθυνση μνήμης, σαν να ήταν όλη η μνήμη δική του, και σαν να μην υπήρχαν άλλα προγράμματα που τρέχουν (σχεδόν) ταυτόχρονα. Ο μηχανισμός της Εικονικής Μνήμης κάνει ώστε ο *Χώρος Διευθύνσεων (Address Space)* του κάθε προγράμματος να είναι διαφορετικός και χωριστός από το χώρο διευθύνσεων του άλλου.

Η μονάδα "προγράμματος ή χρήστη" που θεωρεί ότι έχει όλη τη μηχανή δική της ομομάζεται **Διεργασία (Process)** (ή καμιά φορά και "πεδίο προστασίας" - "protection domain"), και αποτελεί "ένα πρόγραμμα". Διαφορετικές διεργασίες μπορεί να ανήκουν σε διαφορετικούς ανθρώπους-χρήστες, οπότε και θέλουμε να προστατεύονται η μία από την άλλη διότι ο κάθε χρήστης μπορεί να θέλει να έχει ιδιωτικές (εμπιστευτικές) πληροφορίες που να μην μπορεί να τις βλέπει ή υποκλέπτει ο άλλος. Μπορεί όμως επίσης διαφορετικές διεργασίες να ανήκουν στον ίδιο άνθρωπο-χρήστη, αλλά και πάλι να θέλουμε να τις προστατέψουμε τη μία από την άλλη, π.χ. γιά σκοπούς modularity (τακτοποίηση των πληροφοριών κατά ομάδες) και debugging: εάν "χαλάσει" η μία διεργασία, να συνεχίσει να δουλεύει η άλλη. Στην ίδια περίπτωση ανήκουν και διαφορετικά προγράμματα, που ο προγραμματιστής τους θεωρούσε ότι το καθένα τρέχει μόνο του, και κατέληξαν να τρέχουν μαζί. Παραπλήσια –αν και διαφορετική από άποψη δικαιωμάτων και προστασίας– είναι και η περίπτωση μιάς "διεργασίας χρήστη" αφ' ενός, και του Λειτουργικού Συστήματος (Operating System) από την άλλη: το λειτουργικό σύστημα είναι σαν μιά άλλη διεργασία, που έχει σαν σκοπό να βοηθά και να συγχρονίζει όλες τις διάφορες διεργασίες των χρηστών. Άλλη περίπτωση είναι δύο διαφορετικές διεργασίες να είναι το ίδιο μεν πρόγραμμα, που όμως τρέχει τρέχει σε διαφορετικά δεδομένα κάθε φορά –π.χ. ξεκινάμε έναν compiler γιά να μεταφράσουμε ένα πρόγραμμα, και πριν τελειώσει ξαναξεκινάμε ένα άλλο "instance" του ίδιου compiler γιά να μεταφράσει ένα άλλο πρόγραμμα.

Η δυνατότητα του επεξεργαστή (εννοούμε τον κάθε ένα πυρήνα, όταν μιλάμε γιά πολυπύρηνους (multicore) επεξεργαστές) να δίνει την εντύπωση ότι τρέχει (σχεδόν)

"ταυτόχρονα" πολλαπλές διεργασίες λέγεται **Πολυπρογραμματισμός (Multi-Programming)**, και στηρίζεται στο ότι οι επεξεργαστές είναι πολύ γρηγορότεροι από τους ανθρώπους: τρέχει πότε τη μία και πότε την άλλη διεργασία, από λίγο την κάθε μία, αλλά τις εναλλάσει τόσο γρήγορα που ο ανθρώπινος χρήστης δεν βλέπει την εναλλαγή και νομίζει ότι αυτές τρέχουν ταυτόχρονα. Γιά να εναλλάσονται μεταξύ τους οι διεργασίες χρησιμοποιείται βασικά ο μηχανισμός των *Εξαιρέσεων/Διακοπών (Exceptions/Interrupts)* – βλ. §4.9 του βιβλίου.

Η εικονική μνήμη, σαν τρόπος να δίνει την εντύπωση στην κάθε διεργασία ότι έχει τη "μηχανή δική της", αποτελεί ειδική περίπτωση του γενικότερου μηχανισμού της **Εικονικοποίησης (Virtualization)** –βλ. §5.6 του βιβλίου. Η γενικότερη αυτή περίπτωση των "εικονικών μηχανών" (virtual machines) δίνει τη δυνατότητα στην κάθε διεργασία όχι μόνο να (νομίζει ότι) έχει τη μηχανή δική της, αλλά επιπλέον και να έχει το δικό της Λειτουργικό Σύστημα, ανεξάρτητο και ενδεχομένως διαφορετικό από το λειτουργικό σύστημα που έχουν οι άλλες διεργασίες - εικονικές μηχανές που τρέχουν ταυτόχρονα πάνω στον ίδιο επεξεργαστή.

Ο **δεύτερος** και **διαφορετικός** σκοπός της εικονικής μνήμης είναι να λειτουργεί σαν ένα επιπλέον επίπεδο της **Ιεραρχίας Μνήμης** μεταξύ κεντρικής μνήμης (DRAM) και μόνιμης (non-volatile - μη πτητικής) μνήμης (σκληρού δίσκου ή μνήμης flash). Με άλλα λόγια, η εικονική μνήμη δίνει τη δυνατότητα στην κάθε διεργασία να "βλέπει" χώρο μνήμης μεγαλύτερο από το κομμάτι της φυσικής μνήμης (DRAM) που όντως της διατίθεται. Παρ' ότι ο σκοπός αυτός είναι εντελώς διαφορετικός από τον πρώτο (πολλαπλές διεργασίες), τυχαίνει όμως οι ίδιοι μηχανισμοί υλικού (hardware) να μπορούν ταυτόχρονα να υλοποιήσουν και τους δύο σκοπούς.

Μιά συνέπεια των δύο προηγουμένων σκοπών είναι το πρόβλημα του **τεμαχισμού (fragmentation)** της μνήμης, το οποίο –και αυτό– το λύνουν οι ίδιοι μηχανισμοί της εικονικής μνήμης. Ο τεμαχισμός της μνήμης προκύπτει από το ότι διάφορες διεργασίες ξεκινάνε και τελειώνουν σε διαφορετικές στιγμές, και κάθε μιά που τελειώνει αφήνει και κάμποσες "τρύπες" από αχρησιμοποιήτη (πλέον) μνήμη εκεί που προηγουμένως αυτή κατείχε και χρησιμοποιούσε μνήμη. Επίσης, όταν πηγαινοέρχονται στο δίσκο (η στην flash) κομάτια μνήμης μιάς διεργασίας που δεν χωράνε στην DRAM, δημιουργούνται "τρύπες" στη DRAM από τα κομάτια που μετακομίζουν στο δίσκο. Οι μηχανισμοί της εικονικής μνήμης δίνουν τη δυνατότητα γιά εκχώρηση μνήμης (π.χ. μέσω malloc()) που μοιάζει να αποτελείται από συνεχόμενες διευθύνσεις ενώ στην πραγματικότητα αποτελείται από κομάτια που βρίσκονται κατεσπαρμένα "εδώ κι εκεί" στη φυσική μνήμη.

## 12.2 Τοποθέτηση και Ανεύρεση Σελίδων: Πίνακες Μετάφρασης

Επειδή οι αστοχίες της DRAM –ειδωμένης σαν βαθμίδα της ιεραρχίας μνήμης– εξυπηρετούνται από το σκληρό δίσκο, και άρα κοστίζουν τραγικά πολύ (εκατομμύρια κύκλους ρολογιού του επεξεργαστή), πρέπει το ποσοστό αστοχίας να γίνει *πάρα πολύ* χαμηλό. Γιά να επιτευχθεί αυτό, πρέπει η τοποθέτηση των σελίδων να είναι πολύ ευέλικτη –δεν μας κάνει ο τρόπος που είναι οργανωμένες οι κρυφές μνήμες, σαν συσχετιστικές μερικών δρόμων, διότι αυτό θα έδινε πολύ μικρή ευελιξία στην επιλογή του "θύματος" που θα αντικατασταθεί γιά να φέρουμε ένα νέο "μπλόκ" λέξεων, με αποτέλεσμα συχνά να διώχνουνται "μπλόκς" που τα χρειαζόμαστε. Έτσι, αντίθετα από τις συνηθισμένες κρυφές μνήμες, η εικονική μνήμη λειτουργεί σαν "πλήρως συσχετιστική", δηλαδή το οιοδήποτε "μπλοκ" λέξεων από το δίσκο μπορεί να τοποθετηθεί οπουδήποτε στην DRAM. Αυτό, φυσικά, θα έκανε την αναζήτηση εξαιρετικά δαπανηρή εάν έπρεπε να αναζητούμε το "μπλόκ" παντού, γι' αυτό η αναζήτηση γίνεται αλλοιώς: με "πίνακες μετάφρασης" που καταγράφουν πού βρίσκεται το κάθε "μπλόκ".

Επειδή η κάθε προσπέλαση στο δίσκο κοστίζει πολύ (χρονικά), θέλουμε να "αποσβέσουμε" αυτό το κόστος κάνοντας πολλή δουλειά κάθε φορά που πηγάινουμε στο δίσκο, εκμεταλλευόμενοι τη χωρική τοπικότητα καθώς και τα χαρακτηριστικά του δίσκου (προσπέλαση συνεχόμενων λέξεων είναι πολύ γρηγορότερη απ' όσο τυχαίων λέξεων). Έτσι, το "μπλόκ" της ιεραρχίας μνήμης μεταξύ DRAM και δίσκου είναι πολύ μεγαλύτερο απ' όσο το μπλόκ (γραμμή) των κρυφών μνημών. Το "μπλόκ" της ιεραρχίας DRAM-δίσκου λέγεται **Σελίδα (Page)** και είναι συνήθως 4 ή περισσότερα KBytes. Ένας δεύτερος βασικός

(μάλλον και βασικότερος) λόγος γιά τον οποίο θέλουμε μεγάλα μπλόκς (σελίδες) στην εικονική μνήμη είναι ότι έτσι διευκολύνεται και επιταχύνεται ο μηχανισμός ανεύρεσης του μπλόκ που ψάχνουμε κάθε φορά, δηλαδή η μετάφραση σελίδων που θα πούμε αμέσως τώρα.

Ο βασικός τρόπος λειτουργίας της εικονικής μνήμης είναι ο εξής. Κάθε διεύθυνση μνήμης που γεννά ο επεξεργαστής –δηλαδή το πρόγραμμα που τρέχει– θεωρείται ως "**εικονική διεύθυνση**", και **μεταφράζεται** σε μιάν άλλη, "**φυσική διεύθυνση**", προτού δοθεί στη μνήμη γιά να επιλεγεί η λέξη την οποία τελικά θα προσπελάσει το πρόγραμμα. Η μετάφραση αυτή είναι η συνάρτηση που "ανευρίσκει" την κάθε λέξη, δηλαδή αυτή που μας λέει πού στη φυσική μνήμη έχει τοποθετηθεί η κάθε εικονική λέξη. Η συνάρτηση αυτή θα ήταν πανάκριβη εάν δούλευε πάνω σε μεμονωμένες λέξεις, αλλά έχει λογικό κόστος επειδή δουλεύει πάνω σε (μεγάλες) σελίδες. Η μέθοδος αυτή αναζήτησης και ανεύρεσης μπλόκς δεν θα δούλευε γιά την κρυφή μνήμη, λόγω του (σχετικά) μικρού μεγέθους των "γραμμών" της κρυφής μνήμης, και της εκεί απαιτούμενης ταχύτητας. Όμως, δουλεύει μιά χαρά γιά την εικονική μνήμη, λόγω του "κόλπου" του TLB και της λειτουργίας του εν παραλλήλω με την κρυφή μνήμη, όπως θα πούμε παρακάτω. Η μετάφραση διευθύνσεων εξυπηρετεί πολλαπλούς σκοπούς ταυτόχρονα:

i. Βρίσκει πού έχουν τοποθετηθεί στη φυσική μνήμη οι σελίδες εικονικής μνήμης που χρησιμοποιούνται περισσότερο επί του παρόντος, ενώ όσες δεν χωράνε στην υπάρχουσα φυσική μνήμη βρίσκονται στο δίσκο –δηλαδή υλοποιεί την ιεραρχία μνήμης μεταξύ DRAM και δίσκου.
ii. Μεταφράζει τις εικονικές διευθύνσεις της κάθε διεργασίας σε διαφορετικές φυσικές διευθύνσεις γιά την κάθε διεργασία, κι έτσι υλοποιεί το διαχωρισμό και την προστασία μεταξύ διεργασιών (εκτός των περιπτώσεων που θέλουμε οι διεργασίες να επικοινωνούν μεταξύ τους μέσω κοινόχρηστης μνήμης (shared memory)).
iii. Ελέγχει ότι η διεργασία που τρέχει έχει δικαίωμα να κάνει την προσπέλαση που ζητά (ανάγνωση/εγγραφή/εκτέλεση) στη διεύθυνση που ζητά, βελτιώνοντας έτσι ακόμη περισσότερο την προστασία των διεργασιών.

Η μετάφραση διευθύνσεων γίνεται απεικονίζοντας ολόκληρες "**σελίδες**" *(pages)* εικονικής μνήμης σε ολόκληρες φυσικές σελίδες. Το μέγεθος της σελίδας είναι αρκετά KBytes σήμερα, και η τάση είναι να μεγαλώνει με τα χρόνια. Γιά να γίνεται η μετάφραση γρήγορα, χρησιμοποιείται συνήθως ένας μικρός κατάλογος ζευγών εικονικής-φυσικής σελίδας γιά τις πιό συχνά χρησιμοποιούμενες σελίδες –ο "**TLB**" *(Translation Lookaside Buffer)*– οργανωμένος σαν μιά μικρή κρυφή μνήμη, συνήθως πλήρως προσεταιριστική –βλέπε σελίδες 583-590. Επειδή μεταφράζονται μόνον τα αριστερά (MS) bits της εικονικής διεύθυνσης, όπως θα πούμε αμέσως παρακάτω, όταν οι σελίδες είναι αρκετά μεγάλες και η κρυφή μνήμη αρκετά μικρή, μπορούμε να πετύχουμε ώστε η προσπέλαση στην κρυφή μνήμη, που ώς γνωστόν ξεκινάει με κάμποσα από τα δεξιά (**LS**) bits της διεύθυνσης, να ξεκινάει με *μόνον αμετάφραστα* bits της διεύθυνσης, κι έτσι το TLB να δουλεύει εν παραλλήλω (ταυτόχρονα) με την κρυφή μνήμη, και άρα ολόκληρος ο μηχανισμός της εικονικής μνήμης να μην προκαλεί καμία επιπλέον καθυστέρηση στο τρέξιμο του επεξεργαστή, όσο οι σελίδες που χρησιμοποιεί ο επεξεργαστής έχουν τη μετάφρασή τους μέσα στο TLB. Οταν μιάν εικονική σελίδα δεν την βρίσκουμε στον TLB, τότε την αναζητάμε στους *Πίνακες Μετάφρασης*, που βρίσκονται στη μνήμη. (Με μεγαλύτερη ακρίβεια, η εν παραλλήλω λειτουργία TLB και level-1 (L1) cache έχει ως εξής: Φροντίζουμε ώστε το μέγεθος της L1 cache (του κάθε "way" της!) να είναι μικρότερο ή ίσο από το μέγεθος της σελίδας. Όταν ισχύει αυτό, τότε τα index bits που χρειάζονται γιά να *ξεκινήσει* η προσπέλαση στην L1 cache προέρχονται όλα από το *αμετάφραστο* κομάτι της διεύθυνσης (από το page offset), δηλαδή από το κομάτι εκείνο της φυσικής διεύθυνσης που το ξέρουμε *αμέσως*, αφού είναι ίσο με το αντίστοιχο κομάτι της εικονικής διεύθυνσης. Με αυτό το κομάτι ξεκινάει η πρόσβαση στην L1 cache. *ταυτόχρονα* με το ξεκίνημα της πρόσβασης στην L1 cache, δίδεται και η εικονική σελίδα στο TLB γιά μετάφραση. Το TLB το κάνουμε να είναι τόσο γρήγορο όσο και η ανάγνωση των tags της L1 cache. Έτσι, με το που τελειώνει η ανάγνωση των tags στην L1 cache (όπου τα tags αυτά είναι φυσικές διευθύνσεις), έχει τελειώσει και η μετάφραση από το TLB (εάν ήταν εύστοχη), κι έτσι τα tags των lines που βρίσκονται στην cache συγκρίνονται με την μεταφρασμένη πλέον (από το TLB) *φυσική* διεύθυνση).

Θεωρήστε το εξής μικρό (εξωπραγματικό σήμερα) σύστημα εικονικής μνήμης, σαν απλό παράδειγμα.

- Οι εικονικές διευθύνσεις έχουν μέγεθος 20 bits (δηλ. είναι πενταψήφιες στο δεκαεξαδικό σύστημα), άρα ο χώρος εικονικών διευθύνσεων είναι 1 MByte ανά διεργασία.
- Μέγεθος σελίδας = 4 KBytes, άρα τα 12 LS bits κάθε διεύθυνσης (3 LS δεκαεξαδικά ψηφία) επιλέγουν 1 byte μέσα στη σελίδα, ενώ τα υπόλοιπα MS bits υποδηλώνουν γιά ποιά σελίδα μιλάμε. Ετσι, η κάθε διεργασία έχει 256 εικονικές σελίδες (1 MByte / 4 KBytes = 256).
- Η φυσική μνήμη είναι 64 KBytes, άρα οι φυσικές διευθύνσεις αποτελούνται από 16 bits (4 δεκαεξαδικά ψηφία), επομένως υπάρχουν 16 φυσικές σελίδες (64 KBytes / 4 KBytes = 16).

Τότε, η μετάφραση μιάς εικονικής διεύθυνσης, π.χ. της FE210, στην αντίστοιχη φυσική γίνεται ως εξής:

- Χωρίζουμε την εικονική διεύθυνση στα 12 LS bits, που υποδηλώνουν το byte μέσα στη σελίδα (εδώ: 210), και στα 8 MS bits, που ορίζουν την εικονική σελίδα (εδώ: FE).
- Τα 12 LS bits (210) δεν χρειάζονται μετάφραση, αφού απεικονίζουμε ολόκληρες εικονικές σελίδες σε ολόκληρες φυσικές σελίδες, και όλες οι σελίδες είναι ευθυγραμμισμένες στα φυσικά όριά τους.
- Τα 8 MS bits, δηλαδή ο αριθμός εικονικής σελίδας (FE), χρησιμοποιούνται σαν index στον πίνακα μετάφρασης (page table) της διεργασίας (process) που τρέχει αυτή τη στιγμή. Ο πίνακας αυτός έχει μέγεθος 256 θέσεις –όσες και οι εικονικές σελίδες ανά διεργασία. Υπάρχει **χωριστός** πίνακας μετάφρασης **γιά κάθε διεργασία**, έτσι ώστε η κάθε διεργασία να μπορεί να έχει τις δικές της, διαφορετικές, φυσικές σελίδες, παρά το γεγονός ότι χρησιμοποιεί ίδιες εικονικές διευθύνσεις με όλες τις άλλες διεργασίες.
- Στη θέση FE του πίνακα μετάφρασης, όπου μας οδήγησαν τα 8 MS bits της εικονικής διεύθυνσης, υπάρχουν πληροφορίες –όπως θα πούμε παρακάτω– γιά να ελέγξουμε αν η εικονική σελίδα FE που θέλουμε υπάρχει στη φυσική μνήμη, και αν έχουμε δικαίωμα να την προσπελάσουμε όπως ζητά η τρέχουσα διεργασία.
- Στη ίδια θέση FE του πίνακα μετάφρασης βρίσκουμε τον αριθμό της φυσικής σελίδας όπου βρίσκεται αυτή τη στιγμή η εικονική σελίδα FE. Στο παράδειγμά μας, υπάρχουν 16 φυσικές σελίδες, άρα ο αριθμός φυσικής σελίδας έχει 4 bits. Έστω ότι βρήκαμε τον αριθμό A σαν αριθμό φυσικής σελίδας. Σε αυτόν κολάμε και τα 12 αμετάφραστα LS bits της εικονικής διεύθυνσης (210), και φτιάχνουμε έτσι τα 16 bits της φυσικής διεύθυνσης: A210, στο παράδειγμά μας.

## 12.3 Προστασία Μνήμης:

**Διαχωρισμός και Προστασία Διεργασιών:** το hardware του επεξεργαστή βρίσκει τον πίνακα μετάφρασης της τρέχουσας διεργασίας από τη (φυσική) διεύθυνση βάσης του πίνακα αυτού, που είναι γραμμένη (από το λειτουργικό σύστημα) σ' έναν ειδικό καταχωρητή του συστήματος διαχείρισης μνήμης –όχι στο κανονικό register file. Όταν ο επεξεργαστής τρέχει σε "**user mode**", δεν επιτρέπεται να γράψει αυτόν τον καταχωρητή, ούτως ώστε να μην μπορεί να υποκριθεί ότι είναι άλλη διεργασία, δηλαδή να μην μπορεί να αποκτήσει πρόσβαση στη μνήμη άλλων διεργασιών.

**Προστασία Λειτουργικού Συστήματος:** Ο παραπάνω ειδικός καταχωρητής που καθορίζει τον τρέχοντα πίνακα μετάφρασης –δηλαδή την τρέχουσα διεργασία– είναι προσπελάσιμος από τον επεξεργαστή μόνον όταν ο επεξεργαστής βρίσκεται σε "**kernel mode**", δηλαδή τρέχει το λειτουργικό σύστημα. Κάθε εξαίρεση (exception) –περιλαμβανόμενου και του καλέσματος συστήματος (system call)– αποθηκεύει την παλιά κατάσταση (user/kernel) στην οποία έτρεχε ο επεξεργαστής, και φέρνει τον επεξεργαστή σε kernel mode. Έτσι, ο trap (exception) handler εκτελείται πάντα σε kernel mode, ενώ ο μόνος τρόπος γιά ένα χρήστη να φέρει τον επεξεργαστή σε kernel mode είναι να προκαλέσει εξαίρεση, εκτελώντας μιάν εντολή system call –κάτι σαν παράνομη εντολή που προκαλεί εξαίρεση, αλλά που το

λειτουργικό σύστημα ξέρει ότι προορίζεται σαν system call και όχι σαν απλή παράνομη εντολή λόγω προγραμματιστικού σφάλματος. Το κάλεσμα συστήματος είναι επίτηδες φτιαγμένο να συμπεριφέρεται σαν εξαίρεση (exception), και όχι σαν απλό κάλεσμα διαδικασίας (εντολή jal), ούτως ώστε η είσοδος στο λειτουργικό σύστημα –που πρέπει να γίνει σε kernel mode– να γίνεται μόνο στην προκαθορισμένη διεύθυνση του trap handler, και όχι σε οιαδήποτε άλλη αυθαίρετη διεύθυνση θα μπορούσε να ζητήσει ένας κακόβουλος χρήστης προκειμένου να παρακάμψει το μέρος εκείνο του λειτουργικού συστήματος που κάνει τους ελέγχους του εάν ο χρήστης έχει δικαίωμα να ζητήσει αυτό που ζητά.

**Παρούσες/Απούσες Σελίδες και Προστασία Σελίδων:** Κάθε θέση του πίνακα μετάφρασης περιέχει:

- Το "**valid bit**", που υποδεικνύει αν η εικονική σελίδα στην οποία αναφερόμαστε είναι παρούσα ή απούσα από τη φυσική μνήμη. Ενδεχόμενη απουσία της εικονικής σελίδας από τη φυσική μνήμη μπορεί να οφείλεται είτε στο ότι η εικονική αυτή σελίδα είναι *παράνομη* (δεν χρησιμοποιείται, δηλαδή το πρόγραμμα δεν είχε εντολές ή δεδομένα εκεί, ούτε ζήτησε να βάλει κάτι μέσω malloc/sbreak), είτε στο ότι είναι νόμιμη μεν αλλά αυτή τη στιγμή βρίσκεται *στο δίσκο* και όχι στη μνήμη.
- Τον **αριθμό της φυσικής σελίδας** (4 bits στο εδώ παράδειγμά μας) όπου βρίσκεται αυτή τη στιγμή η εικονική σελίδα, όταν αυτή είναι παρούσα στη φυσική μνήμη.
- Τα "**bits προστασίας**" (π.χ. 3 bits: "rwx"), που δηλώνουν τι είδους προσπελάσεις επιτρέπεται να κάνει η παρούσα διεργασία στις λέξεις αυτής της σελίδας (read/write/execute).
- Το "**dirty bit**", που δείχνει αν ο επεξεργαστής άλλαξε ή όχι τα περιεχόμενα αυτής της σελίδας από τότε που την διαβάσαμε τελευταία φορά από το δίσκο –επομένως εάν χρειάζεται ή όχι να γράψουμε το (νέο) περιεχόμενό της πίσω στο δίσκο όταν κάποιος άλλος της κάνει "έξωση" (eviction) από τη μνήμη.
- Το "**reference bit**", που το χρησιμοποιεί το λειτουργικό σύστημα γιά να προσεγγίσει τον αλγόριθμο αντικατάστασης "LRU": σε κάθε προσπέλαση στη σελίδα, ο επεξεργαστής θέτει αυτό το bit, ενώ το λειτουργικό σύστημα περιοδικά διαβάζει αυτά τα bits γιά να δεί ποιές σελίδες χρησιμοποιήθηκαν πρόσφατα, και στη συνέχεια τα μηδενίζει.

"Παράνομες" προσπελάσεις εικονικής μνήμης είναι π.χ. οι εξής: *(i)* προσπάθεια εγγραφής σε σελίδα όπου δεν έχω δικαίωμα εγγραφής (π.χ. σελίδα read-only), ή *(ii)* προσπάθεια ανάγνωσης δεδομένων από σελίδα που είναι, π.χ. "execute-only" (δηλ. περιέχει εντολές), ή *(iii)* προσπάθεια ανάγνωσης με σκοπό την εκτέλεση (instruction fetch) από σελίδα που δεν περιέχει εκτελέσιμες εντολές, ή *(iv)* προσπάθεια προσπέλασης οιασδήποτε μορφής σε σελίδα "unallocated", δηλαδή που ούτε ο compiler έβαλε εκεί (στατικά) κάποιο μέρος του προγράμματος (εντολές ή δεδομένα), ούτε η malloc() εκχώρησε εκεί (δυναμικά) χώρο (άδεια χρήσης). Τέτοιες "παράνομες" προσπελάσεις προκαλούν πρόωρο τερματισμό της εκτέλεσης, με το οικείο σε όλους μας μήνυμα "segmentation violation - core dumped".

"Απούσες (αλλά νόμιμες)" σελίδες εικονικής μνήμης είναι εκείνες που ναι μεν το πρόγραμμα έχει νόμιμο δικαιώμα προσπέλασης σε αυτές (δηλ. **δεν** είναι παράνομη η προσπέλαση), αλλά όμως τυχαίνει (είτε ελλείψει επαρκούς μνήμης, είτε επειδή ποτέ μέχρι στιγμής δεν την είχε ζητήσει το πρόγραμμα) η σελίδα αυτή να μην βρίσκεται αυτή τη στιγμή στην (κεντρική) μνήμη (RAM) του υπολογιστή, αλλά στο (σκληρό) δίσκο. Προσπελάσεις σε απούσες σελίδες προκαλούν *προσωρινή* διακοπή της εκτέλεσης του προγράμματος (σφάλμα σελίδας - *page fault* exception/interrupt), ανάληψη από το λειτουργικό σύστημα να προσκομίσει την απούσα σελίδα, και επανεκκίνηση του προγράμματος "σαν να μη συνέβη τίποτα" μετά την προσκόμιση αυτή.
**Διαβάστε γιά τις Εξαιρέσεις/Διακοπές** (Exceptions/Interrupts) από την §4.9 του βιβλίου.
Προσέξτε ιδιαίτερα ότι τα σφάλματα σελίδας πρέπει να τα αναγνωρίζει το υλικό και να τα χειρίζεται **πρίν** ολοκληρωθεί η εκτέλεση της εντολής που τα προκάλεσε: πριν γράψει η εντολή load στον καταχωρητή προορισμού ή η εντολή store στη λέξη προορισμού, μέσα στην –ανύπρακτη!– σελίδα μνήμης. Αλλοιώς, π.χ., δεν μπορούν να επανεκκινηθούν σωστά, μετά την προσκόμιση της απούσας σελίδας από το δίσκο, εντολές όπως `"lw $8, 4($8)"` που προκύπτουν π.χ. από συνηθισμένες εκχωρήσεις όπως: `"p = p->next;"`.

## Άσκηση 12.4: Μονοεπίπεδος Πίνακας Μετάφρασης

**(α)** Γιά το παραπάνω μικρό (εξωπραγματικό σήμερα) παράδειγμα εικονικής μνήμης, κάντε ένα σχηματικό διάγραμμα που να δείχνει τον καταχωρητή που περιέχει τον pointer στον πίνακα μετάφρασης της παρούσας διεργασίας, τον πίνακα μετάφρασης, την εικονική διεύθυνση (20 bits) που γεννά ο επεξεργαστής, τα πεδία από τα οποία αυτή αποτελείται, από που προέρχεται το index στον πίνακα μετάφρασης, τι διαβάζουμε από τη θέση εκείνη του πίνακα, και πώς συνθέτουμε τη φυσική διεύθυνση (16 bits).

**(β)** Έστω ότι, στο παραπάνω απλό παράδειγμά μας, η διεργασία μας έχει τις εξής σελίδες:

- Εικονική σελίδα 00: παράνομη (περιέχει τον NULL pointer).
- Εικονική σελίδα 01: περιέχει κώδικα (r-x), και βρίσκεται στη φυσική σελίδα 4.
- Εικονικές σελίδες 02 έως και 09: περιέχουν κώδικα (r-x), αλλά είναι απούσες από τη φυσική μνήμη.
- Εικονική σελίδα 0A: περιέχει στατικά δεδομένα (rw-), και βρίσκεται στη φυσική σελίδα F, dirty.
- Εικονικές σελίδες 0B έως και BF: παράνομες (unallocated).
- Εικονική σελίδα C0: περιέχει δυναμικά δεδομένα (rw-), και βρίσκεται στη φυσική σελίδα 0, dirty.
- Εικονική σελίδα C1: περιέχει δυναμικά δεδομένα read-only (r--), και βρίσκεται στη φυσική σελίδα 2.
- Εικονικές σελίδες C2 και C3: περιέχουν δυναμικά δεδομένα (rw-), αλλά είναι απούσες από τη φυσική μνήμη.
- Εικονικές σελίδες C4 έως και FD: παράνομες (unallocated).
- Εικονική σελίδα FE: περιέχει δεδομένα στοίβας (rw-), και βρίσκεται στη φυσική σελίδα D, dirty.
- Εικονική σελίδα FF: περιέχει δεδομένα στοίβας (rw-), και βρίσκεται στη φυσική σελίδα 1, clean.

Δείξτε τα περιεχόμενα του πίνακα μετάφρασης της διεργασίας μας, χωρίς τα reference bits αλλά με όλα τα άλλα πεδία του (256 γραμμές επί 4 πεδία ανά γραμμή –επιτρέπεται η χρήση αποσιωποιητικών...).

**(γ)** Ποιές από τις παρακάτω προσπελάσεις στις εικονικές διευθύνσεις που δίδονται προκαλούν σφάλμα σελίδας και **γιατί**; Ποιές από αυτές θα προκαλέσουν τερματισμό του προγράμματος, ως λανθασμένο, και ποιές απλώς θα κάνουν το λειτουργικό σύστημα να φέρει την αιτηθείσα σελίδα από το δίσκο και να επανεκκινήσει το πρόγραμμα; Οι υπόλοιπες, σε ποιά φυσική διεύθυνση μεταφράζονται;

> 01038 (fetch), 0B0F4 (read), C001C (write), 0292C (fetch), 00000 (read), 99F88 (read), FE5D8 (write), FF100 (fetch), C20CC (write), CD0CC (write), C0444 (read), 01FF4 (fetch), C1FFC (write), 008E4 (write), C7700 (read), 01E40 (write).

## Άσκηση 12.5: Πολυεπίπεδοι Πίνακες Μετάφρασης - Οικονομία Χώρου

Οι περισσότερες διεργασίες χρησιμοποιούν σχετικά λίγες από τις εικονικές τους σελίδες, και αυτές συνοθηλευμένες (clustered) σε μερικές "γειτονιές", όπως στην παραπάνω άσκηση 12.4. Εκμεταλλευόμενοι αυτή την ιδιότητα, μπορούμε να μειώσουμε το χώρο μνήμης που καταλαμβάνει ο πίνακας μετάφρασης της κάθε διεργασίας, σπάζοντας τον σε μερικούς μικρότερους πίνακες, οργανωμένους σαν μιά πολυ-επίπεδη ιεραρχία.

Θεωρήστε ότι στο σύστημα μνήμης της άσκησης 12.4 αλλάζουμε τον μοναδικό (μονοεπίπεδο) πίνακα μετάφρασης ανά διεργασία σε διεπίπεδους πίνακες, ώς εξής. Κάθε διεργασία έχει έναν πίνακα πρώτου επιπέδου, μεγέθους 16 θέσεων. Τον πίνακα αυτόν τον βρίσκουμε μέσω του γνωστού pointer που περιέχεται στον ειδικό καταχωρητή που αναφέραμε παραπάνω. Χρησιμοποιούμε τα 4 MS bits της εικονικής διεύθυνσης γιά να επιλέξουμε μία από τις 16 θέσεις αυτού του πίνακα. Κάθε συνδυασμός των 4 αυτών bits, επομένως και κάθε θέση αυτού του πίνακα, αντιστοιχεί σε 16 εικονικές σελίδες. Εαν καμία από αυτές τις 16 σελίδες δεν υπάρχει στη φυσική μνήμη, τότε σημειώνουμε τη θέση αυτή του πίνακα πρώτου επιπέδου σαν άκυρη (valid bit = 0). Αλλοιώς, η θέση αυτή του πίνακα

πρώτου επιπέδου περιέχει έναν pointer σε ένα πίνακα μετάφρασης δευτέρου επιπέδου. Εάν η εικονική μας διεύθυνση μας οδήγησε σε τέτοια θέση στον πίνακα πρώτου επιπέδου, τότε χρησιμοποιούμε τα επόμενα 4 bits της εικονικής διεύθυνσης σαν index στον πίνακα δευτέρου επιπέδου όπου μας οδήγησε ο πίνακας πρώτου επιπέδου. Εκεί, στον πίνακα δευτέρου επιπέδου, βρίσκουμε τα τελικά στοιχεία γιά τη σελίδα που ζητάμε.

**(αβγ)** Κάντε ένα διάγραμμα ανάλογο προς αυτό της άσκησης 12.4(α) γιά το διεπίπεδο σύστημα μετάφρασης αυτής της άσκησης. Στο ίδιο διάγραμμα, δείξτε όλους τους πίνακες δευτέρου επιπέδου που θα υπάρχουν γιά τις σελίδες της άσκησης 12.4(β). Επίσης δείξτε, όλα τα περιεχόμενα όλων των πινάκων μετάφρασης, πρώτου και δευτέρου επιπέδου. Βεβαιωθείτε (χωρίς να δώσετε γραπτά την απάντησή σας) ότι το σύστημα αυτό μεταφράζει τις διευθύνσεις της άσκησης 12.4(γ) το ίδιο όπως και το μονοεπίπεδο σύστημα της άσκησης εκείνης.

**(δ)** Πόσες θέσεις μνήμης καταλαμβάνουν όλοι οι πίνακες μετάφρασης του παρόντος διεπίπεδου συστήματος γιά τη διεργασία μας και γιά τις σελίδες (β); Σε σχέση με το μονοεπίπεδο σύστημα της άσκησης 12.4(β) υπάρχει οικονομία στο χώρο μνήμης που καταλαμβάνεται;

## Άσκηση 12.6: Πολυεπίπεδοι Πίνακες Μετάφρασης σε ένα Ρεαλιστικό Σύστημα

Επαναλάβετε την άσκηση 12.5(α) –που ήταν σαν την 12.4(α) (σχηματικό διάγραμμα πινάκων μετάφρασης)– αυτή τη φορά γιά ένα ρεαλιστικό, σημερινό, σύστημα εικονικής μνήμης:

- Οι εικονικές διευθύνσεις έχουν μέγεθος 64 bits. Όμως, γιά πρακτικούς λόγους, μονο τα 48 LS από αυτά χρησιμοποιούνται (χώρος εικονικών διευθύνσεων 256 TeraBytes ανά διεργασία...).
- Μέγεθος σελίδας 4 KBytes. Επομένως τα 12 LS bits της εικονικής διεύθυνσης δεν χρειάζονται μετάφραση.
- Τρία επίπεδα πινάκων μετάφρασης ανά διεργασία.
- Κάθε πίνακας μετάφρασης έχει μέγεθος 4096 θέσεις, άρα "κοιτάζει" 12 από τα bits του αριθμού εικονικής σελίδας.
- Δυνατότητα επέκτασης έως 256 GBytes φυσικής μνήμης, δηλαδή 38 bits φυσική διεύθυνση, ή 64 M φυσικές σελίδες (26-μπιτος αριθμός φυσικής σελίδας) επί 4 KBytes ανά σελίδα.

## Άσκηση 12.7: TLB, Process ID, και Κοινόχρηστες Σελίδες

Όπως είπαμε και παραπάνω, γιά να γίνεται η μετάφραση γρήγορα, χρησιμοποιείται συνήθως ένας μικρός κατάλογος ζευγών εικονικής-φυσικής σελίδας γιά τις πιό συχνά χρησιμοποιούμενες σελίδες, ο "**TLB**" (Translation Lookaside Buffer), οργανωμένος σαν μιά μικρή κρυφή μνήμη, συνήθως πλήρως προσεταιριστική.

Προκειμένου να μην αναγκαζόμαστε να ακυρώνουμε τα περιεχόμενα του TLB σε κάθε αλλαγή της διεργασίας που τρέχει (context swap), θέλουμε να μπορούμε να έχουμε μέσα στο TLB, ταυτόχρονα, ζευγάρια εικονικής-φυσικής σελίδας πολλών διαφορετικών διεργασιών. Αυτό όμως απαιτεί να μπορούμε να τα ξεχωρίζουμε μεταξύ τους, αφού την κάθε ορισμένη εικονική διεύθυνση ενδέχεται να την χρησιμοποιούν πολλές διεργασίες αλλά γιά διαφορετική πληροφορία και κατά διαφορετικό τρόπο η κάθεμία. Γιά να γίνεται ο διαχωρισμός αυτός, καταγράφουμε τον αριθμό διεργασίας ("**PID**", Process Identifier) μαζί με τον αριθμό εικονικής σελίδας αυτής της διεργασίας σε κάθε θέση (ζευγάρι εικονικής-φυσικής σελίδας) του TLB.

**(α)** Θεωρήστε μιά εικονική μνήμη με μεγέθη όπως της άσκησης 12.4, και θεωρήστε ότι το PID έχει μέγεθος 8 bits (μέχρι 256 ταυτόχρονες διεργασίες). Θεωρήστε ένα TLB μεγέθους 16 θέσεων, με πλήρως προσεταιριστική τοποθέτηση ζευγών (οιοδήποτε ζεύγος μετάφρασης μπορεί να μπεί οπουδήποτε στο TLB). Ποιά πεδία πρέπει να έχει η κάθε θέση αυτού του TLB, και τι μεγέθους το καθένα;

**(β)** Δώστε ένα αριθμητικό παράδειγμα του πλήρους περιεχομένου του TLB όταν αυτό περιέχει ζευγάρια μετάφρασης γιά τις παρακάτω εικονικές σελίδες· αποφασίστε μόνοι σας, αυθαίρετα (σαν να είσασταν το λειτουργικό σύστημα), σε ποιάν φυσική σελίδα θα τοποθετήστε την κάθε εικονική σελίδα που σας ζητείται να ευρίσκεται εν χρήση:

- Την εικονική σελίδα 03 της διεργασίας 3B (περιέχει κώδικα), και την εικονική σελίδα 03 της διεργασίας B4 (περιέχει data r/w), οι οποίες αναφέρονται σε εντελώς διαφορετικές, ανεξάρτητες φυσικές σελίδες.
- Η διεργασία 3B και η διεργασία 3C αποτελούν διαφορετικές ενεργοποιήσεις του ιδίου προγράμματος, π.χ. ο ίδιος web browser τρεγμένος από δύο διαφορετικούς χρήστες. Γιά οικονομία (φυσικής) μνήμης, το λειτουργικό σύστημα κρατά μόνο ένα αντίτυπο του κώδικα αυτής της εφαρμογής στη μνήμη, και κάνει όλες τις διεργασίες που το τρέχουν να "βλέπουν" αυτό το μοναδικό κοινόχρηστο αντίτυπο στη δική της εικονική μνήμη η κάθε μία. Βάλτε λοιπόν στο TLB και την εικονική σελίδα 03 της διεργασίας 3C, που περιέχει τον ίδιο κώδικά με τη σελίδα 03 της 3B.
- Βάλτε την εικονική σελίδα FF της διεργασίας 3B, και την εικονική σελίδα FF της διεργασίας 3C, που περιέχουν δεδομένα στοίβας. Οπως είπαμε, αυτές οι δύο διεργασίες αποτελούν διαφορετικές ενεργοποιήσεις του ιδίου προγράμματος. Παρά το γεγονός ότι ο κώδικας των δύο αυτών διεργασιών είναι κοινός, τα δεδομένα τους όμως είναι διαφορετικά, αφού π.χ. άλλα κάνει ο ένας χρήστης με τον web browser του, και άλλα ο άλλος.
- Οι διεργασίες A2 και A3 συνεργάζονται μεταξύ τους, και επικοινωνούν μέσω κοινόχρηστης μνήμης: Η διεργασία A2 παράγει δεδομένα προς επεξεργασία, και τα γράφει στην εικονική της σελίδα C2 (προστασία -w-). Η διεργασία A3 καταναλώνει τα δεδομένα που παράγει η A2, και τα επεξεργάζεται. Η A3 διαβάζει τα δεδομένα αυτά από την εικονική σελίδα της E3 (προστασία r--), η οποία όμως, μέσω του συστήματος εικονικής μνήμης, αντιστοιχεί στην ίδια φυσική σελίδα στην οποία αντιστοιχεί και η C2 της A2, ώστε να επιτυγχάνεται η συνεργασία παραγωγού-καταναλωτή των δύο διεργασιών. Βάλτε στο TLB σας και αυτές τις δύο εικονικές σελίδες.

**(γ)** Οι διεργασίες 3B και 3C, παραπάνω, είναι προστατευμένες η μία από την άλλη; Μπορεί η μία να διαβάσει τα δεδομένα της άλλης (κλέβοντας έτσι, π.χ., ο ένας χρήστης τον αριθμό πιστωτικής κάρτας που ο άλλος πληκτρολογεί στον browser του γιά μιάν αγορά μέσω διαδικτύου); Μπορεί η μία να αλλοιώσει (γράψει) τα δεδομένα της άλλης (παραπλανόντας έτσι, π.χ., ο ένας χρήστης τον άλλον); Μπορεί η μία να καταστρέψει (γράψει) τον κώδικα της άλλης ("κολλώντας" έτσι, π.χ., ο ένας χρήστης τον άλλον); Πώς εξασφαλίζουμε την επιθυμητή προστασία και ανεξαρτησία μεταξύ αυτών των δύο διεργασιών, ενώ ταυτόχρονα κάνουμε και οικονομία μνήμης κρατώντας ένα μόνο φυσικό αντίτυπο του κώδικα που αυτές τρέχουν;

## Τρόπος Παράδοσης

Παραδώστε τη σειρά αυτή ασκήσεων on-line, σε μορφή **PDF** (μόνον) (μπορεί να είναι κείμενο μηχανογραφημένο ή/και "σκαναρισμένο" χειρόγραφο, αλλά *μόνον* σε μορφή PDF). Παραδώστε μέσω **`turnin ex12@hy225 [directoryName]`** ένα αρχείο ονόματι **`ex12.pdf`** που θα περιέχει τις απαντήσεις σας.

---