

# Λειτουργικά Συστήματα (ΗΥ321)

## Διάλεξη 3:
## Χρονοδρομολόγηση

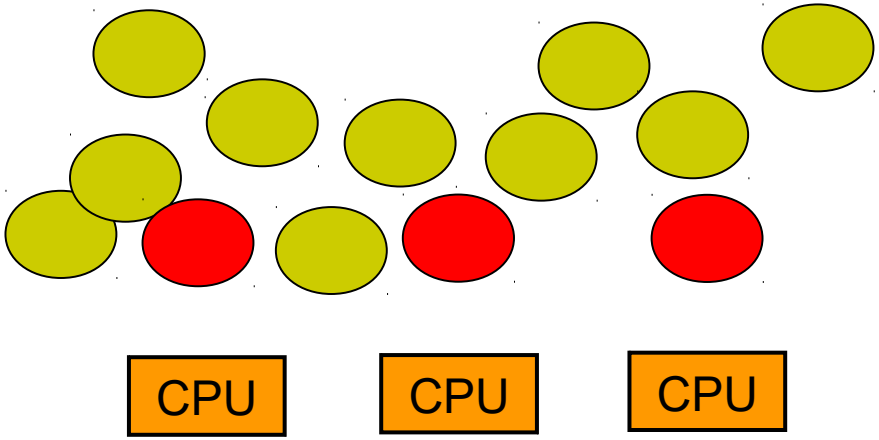
**Χρήστος Δ. Αντωνόπουλος**
**9/3/2017**

**Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας**

# Σε αυτό το Επεισόδιο: Χρονοδρομολόγηση



- Ανάθεσε m διεργασίες σε n επεξεργαστές



# Το Πρόβλημα;



- Ποιος;
  - θα τρέξει στον επόμενο τόνο;
  - θα τρέξει στον συγκεκριμένο επεξεργαστή;
- Απαιτήσεις (μεταξύ άλλων):
  - Ελαχιστοποίηση χρόνου απόκρισης / χρόνου ολοκλήρωσης
    - Χρόνος απόκρισης: Χρόνος από το κλικ του ποντικιού μέχρι το κλείσιμο του παραθύρου
    - Χρόνος ολοκλήρωσης: Ο χρόνος από την υποβολή μιας εργασίας μέχρι την ολοκλήρωσή της
  - Μεγιστοποίηση του ρυθμού παραγωγής αποτελεσμάτων (εργασίες στη μονάδα του χρόνου)
    - Εκμεταλλεύσου όλους τους «πόρους» του συστήματος με τον «καλύτερο δυνατό τρόπο»
    - Ελαχιστοποίησε τους «έμμεσους φόρους» (μεταγωγή περιβάλλοντος, αλλαγή χώρου διευθύνσεων)
  - Δικαιοσύνη
    - Αν και η στοχευμένη αδικία τείνει να βελτιώνει την επίδοση

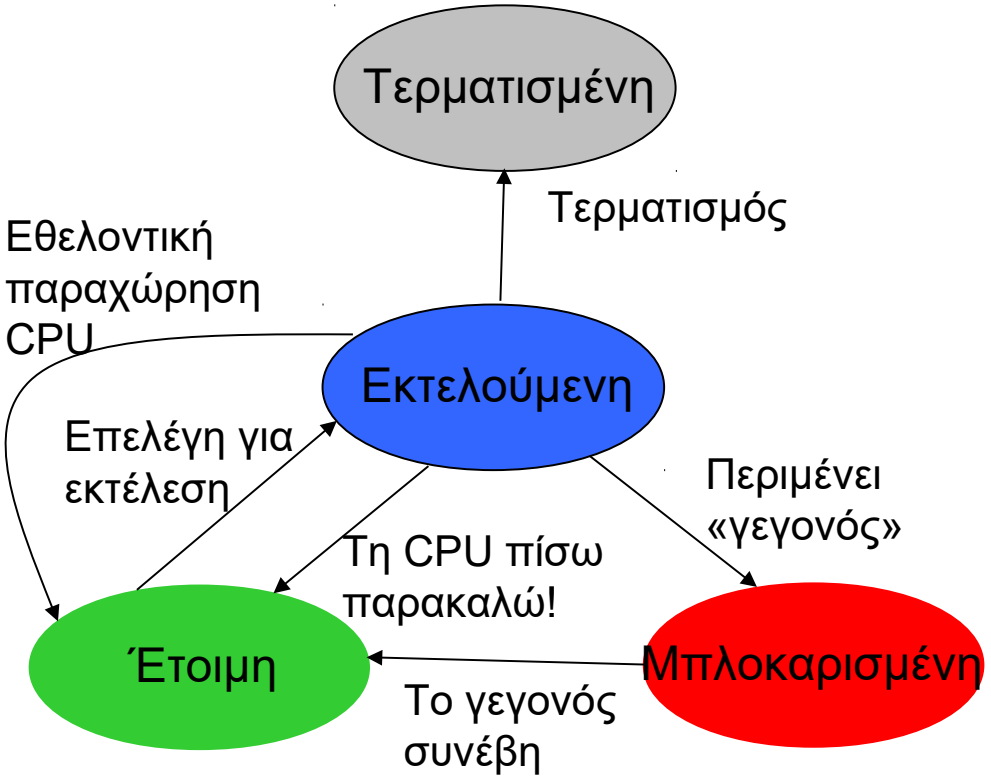
# «Κακές» Περιπτώσεις



- Οι συσκευές I/O μένουν ανενεργές επειδή ο χρονοδρομολογητής δεν ξέρει ότι κάποια διεργασία χρειάζεται I/O
- Ένα τρυκ ευνοεί τις διεργασίες ενός συγκεκριμένου τύπου. Αν μαζευτούν πολλές από αυτές, οι υπόλοιπες θα βαρεθούν να περιμένουν
- Μια αλληλεπιδραστική διεργασία παγιδευμένη πίσω από μια υπολογιστική. Η απόκριση του συστήματος … χάλια
- Η διεργασία Α έχει ένα lock το οποίο χρειάζεται και η διεργασία Β. Η διεργασία Β έχει μεγαλύτερη προτεραιότητα. Η μπορεί να μην Α εκτελεστεί ποτέ…

# Κάθε Πότε;

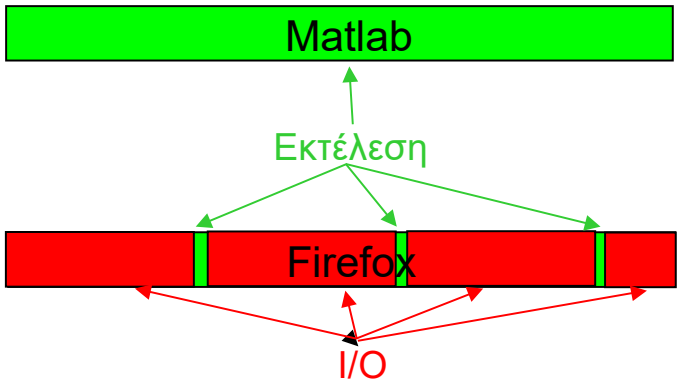
- Μη προεκτοπιστικά συστήματα (non-preemptive) – Η διεργασία εκτελείται έως ότου:
  - Εγκαταλείψει προσωρινά μόνη της τον επεξεργαστή (blocking system call, I/O, …)
  - Πεθάνει
- Προεκτοπιστικά συστήματα (preemptive) – Η διεργασία εκτελείται έως ότου:
  - Συμβεί οτιδήποτε από τα παραπάνω
  - Ολοκληρωθεί κάποιο γεγονός και μια διεργασία γίνει από μπλοκαρισμένη έτοιμη
  - «Χτυπήσει» το ρολόι του συστήματος



# Μια Ματιά Κάτω από την Επιφάνεια



- Πόσες CPUs σε ένα μονοεπεξεργαστικό σύστημα;
  - 1;
  - Και οι συσκευές I/O;
  - Και η κάρτα γραφικών;
  - «Επεξεργαστές» ειδικού σκοπού
    - Προσανατολισμένοι σε συγκεκριμένη εργασία
    - Μπορούμε να τους εκμεταλλευτούμε για «επιτάχυνση» των έργων;



# «Κάστες» Διεργασιών



- Εξαρτώμενες από υπολογισμό


- Εξαρτώμενες από I/O


  - Ιδέα: Όταν μια διεργασία κάνει I/O δώσε τον επεξεργαστή σε άλλη
    - Συνήθως αλληλεπιδραστικές διεργασίες
  - Πρόβλημα: Δεν ξέρουμε εκ των προτέρων ποια θα κάνει I/O και ποια υπολογισμό
- Για ποιες διεργασίες είναι σημαντικός ο χρόνος απόκρισης και για ποιες ο χρόνος ολοκλήρωσης;

# Το Μεγάλο Πρόβλημα της Χρονοδρομολόγησης



- Ποιος είναι ο «καλύτερος» τρόπος να τρέξεις m διεργασίες πάνω σε n επεξεργαστές;
  - Μοιάζει με πρόβλημα πολύπλεξης…
- Προβλήματα; (πόσα θέλετε;)
  - Αντικρουόμενοι στόχοι
    - Ρυθμός παραγωγής αποτελεσμάτων vs. αποκρισιμότητα
    - Αποδοτικότητα vs. δικαιοσύνη
  - Εν οίδα, ότι ουδέν οίδα (ελληνιστί: δεν έχω ιδέα – έλλειψη γνώσης)
    - Τι είναι σημαντικό για το χρήστη;
    - Ποια είναι τα χαρακτηριστικά των διεργασιών;
  - Πραγματικά συστήματα => Πολύπλοκα συστήματα
    - Δεν υπάρχουν ακριβή μαθηματικά μοντέλα και λύσεις
    - Δοκιμάζουμε και … βλέπουμε

# Ας Μιλήσουμε για τους Πόρους



- Τι είναι;
  - Οτιδήποτε χρειάζεται η διεργασία για την εκτέλεσή της (χρόνος CPU, μνήμη, πρόσβαση στο δίσκο…)
- Κατηγορίες;
  - Προεκτοπίσιμοι – μπορούμε να τους πάρουμε πίσω όποτε θέλουμε (π.χ. CPU)
  - Μη προεκτοπίσιμοι – δε μπορούμε να τους πάρουμε πίσω πριν τελειώσει η διεργασία (π.χ. ένα lock)
  - Σαφής διαχωρισμός;

# Και … Πώς Μοιράζουμε τους Πόρους; (1/2)



- Χωρική διαμοίραση (οριζόντια)
  - Αν έχουμε πολλαπλά αντίγραφα ενός πόρου δίνουμε κάθε αντίγραφο σε μια / μια ομάδα διεργασία/ιών
  - Καλή για πόρους που δεν είναι εύκολα προεκτοπίσιμοι
    - Δίσκος, τερματικά, …
  - Η δεν είναι προεκτοπίσιμοι με λογικό κόστος
    - Μνήμη (διαχωρισμός του χώρου διευθύνσεων σε σελίδες αντί της μεταφοράς όλου του χώρου διευθύνσεων από / προς το δίσκο σε κάθε μεταγωγή περιβάλλοντος)

# Και … Πώς Μοιράζουμε τους Πόρους; (2/2)



- Χρονική διαμοίραση (κατακόρυφη)
  - Χωρίσαμε τους πόρους σε ομάδες.
    - Ποιος παίρνει κάθε ομάδα ανά πάσα στιγμή;
    - Για πόσο;
  - Πότε;
    - Όταν οι απαιτήσεις είναι περισσότερες από τους πόρους
      - Π.χ. πολλές διεργασίες ζητούν μία CPU
    - Όταν οι πόροι δε μπορούν να χωριστούν περαιτέρω
      - Π.χ. CPU, κεφαλή δίσκου
    - Όταν οι πόροι είναι προεκτοπίσιμοι με χαμηλό κόστος
      - Π.χ. καταχωρητές που διατηρούν το περιβάλλον μιας διεργασίας στη CPU

# Εξυπηρέτηση με Σειρά Άφιξης (FCFS ή FIFO)

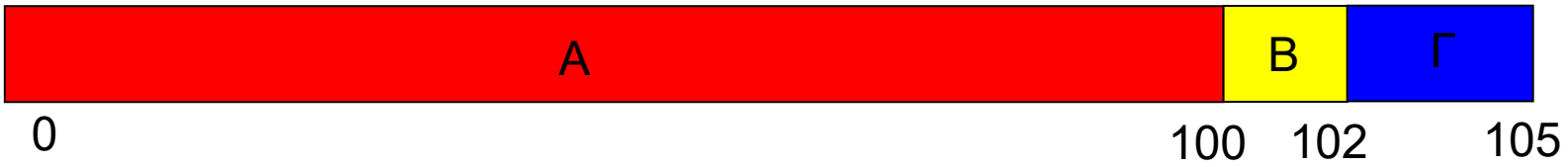


- Εκτέλεσε τα έργα με τη σειρά άφιξής τους
- Μη προεκτοπίσιμα έργα
  - Ένα πρόγραμμα: Εκτέλεσε μέχρι τον τερματισμό του
  - Πολλαπλά προγράμματα: Εκτέλεσε μέχρι να ζητήσει I/O, βάλε το στο τέλος της ουράς μετά την ολοκλήρωση του I/O
- Πλεονεκτήματα:
  - Απλό
- Μειονεκτήματα:
  - Η εικόνα για την απόδοση εξαρτάται από τη σειρά άφιξης
    - Άδικο για τις εργασίες που φτάνουν αργά
    - Κακή περίπτωση: Μια μεγάλη εργασία φτάνει νωρίς

# FCFS: Παράδειγμα



- Α = 100, Β = 2, Γ = 3
- Φτάνουν σχεδόν ταυτόχρονα (η Α λίγο πιο νωρίς από τη Β και η Β λίγο πριν τη Γ)




- Με αυτή τη σειρά;

# Η Αχίλλειος Πτέρνα της FCFS



- Μια υπολογιστικά βαριά διεργασία μπορεί να μπλοκάρει όλες τις άλλες πίσω της (convoy)
  - Για μεγάλες περιόδους υπολογίζει χωρίς I/O
  - Οι αλληλεπιδραστικές εφαρμογές (I/O) δεν έχουν την ευκαιρία να εκτελεστούν
    - Η κι αν εκτελεστούν αυτό θα γίνει για λίγο και θα ξαναβρεθούν μπλοκαρισμένες πίσω από τη βαριά διεργασία

# Δρομολόγηση με Εναλλαγές (Round Robin / RR)

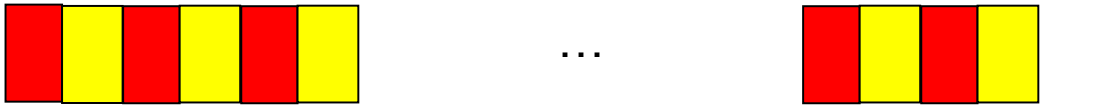


- Μην αφήνεις υπολογιστικά βαριές διεργασίες να μονοπωλούν τον επεξεργαστή. Διάκοψέ τις.
  - Βάλε ένα ρολόι
  - Τρέξε κάθε διεργασία μέχρι να χτυπήσει το ρολόι (για το κβάντο της) ή έως ότου μπλοκάρει (I/O, system call, …)
  - Όταν ξεμπλοκάρει βάλε τη στο τέλος της ουράς.
- Πλεονεκτήματα:
  - Δίκαια
    - Όλα τα συστήματα χρησιμοποιούν RR με τον ένα ή τον άλλο τρόπο (συνήθως σε συνδυασμό με κάτι άλλο)
  - Καλός μέσος χρόνος ολοκλήρωσης για έργα μεταβλητού μεγέθους

# Η Αχίλλειος Πτέρνα της RR



- Τι γίνεται για διεργασίες παρόμοιου μεγέθους;
  - A = 100, B = 100




  - Μέσος χρόνος ολοκλήρωσης;
    - Πώς θα τα πήγαινε εδώ η FCFS;

# RR: Με τι Κβάντο;



- Η μεταγωγή περιβάλλοντος κοστίζει!
  - Άμεσο κόστος +
  - Γραμμάτια για το μέλλον … (cache, TLB)
- Πολύ μεγάλο κβάντο: FCFS
  - Οι εργασίες θα τερματίσουν ή θα μπλοκάρουν πριν να τερματιστεί το κβάντο τους
- Πολύ μικρό κβάντο: Μεγάλη επιβάρυνση
  - Συνεχείς μεταγωγές περιβάλλοντος
- Κατά κανόνα κβάντο ~ 100 msec
  - Κόστος μεταγωγής περιβάλλοντος < 1%

# Χρονοδρομολόγηση με Προτεραιότητες



- Δεν είναι όλες οι διεργασίες ίδιες
  - Ιεράρχισέ τις
- Ανάθεση προτεραιότητας σε κάθε διεργασία
  - Κάθε φορά επιλέγεται από τις έτοιμες διεργασίες αυτή με τη μεγαλύτερη προτεραιότητα
  - Διεργασίες της ίδιας προτεραιότητας δρομολογούνται με RR
  - Στατικές ή δυναμικές προτεραιότητες
    - Ή και τα δύο ταυτόχρονα (Unix)
- Προτεραιότητα ανάλογα με τα χαρακτηριστικά της εργασίας (και τους επιθυμητούς στόχους του scheduler)
  - Καταπολέμηση απεριόριστης αναμονής: Συνέδεσε την προτεραιότητα με το χρόνο από την τελευταία εκτέλεση
  - Αξιοποίηση των συσκευών I/O: Θέσε υψηλή προτεραιότητα στις διεργασίες που εκτελούν πολύ I/O
- Προβλήματα;
  - Μπορούν οι προτεραιότητες να προκαλέσουν αδιέξοδο;

# Αναστροφή Προτεραιότητας & Λύσεις



- Διεργασία Α υψηλότερης προτεραιότητας από τη Β
  - Όμως η Β κρατάει ένα lock το οποίο χρειάζεται η Α
    - Η Α προσπαθεί να πάρει το lock και εκτελεί ενεργή αναμονή. Η Β αδυνατεί να εκτελεστεί
    - Η Α προσπαθεί να πάρει το lock και μπλοκάρει. Όμως στο σύστημα μπαίνει διεργασία Γ μέσης προτεραιότητας. Η Β αδυνατεί να εκτελεστεί
- Χρονοδρομολογητής: Αποφασίζει ποιος θα έχει την ευκαιρία να προχωρήσει
  - Η σημασία μιας διεργασίας πρέπει να αντανακλά και τη σημασία των διεργασιών που εξαρτώνται από αυτή.
    - Λογικό, αλλά όχι απλό…
  - Δυνατότητα παραχώρησης προτεραιότητας
    - Προσφέρεται από κάποια λειτουργικά

# Πρώτα η Συντομότερη (SJF ή STCF)



- Shortest Time to Completion First (ή Shortest Job First)
  - Εκτέλεσε πρώτα τη συντομότερη διεργασία
  - Μπορεί η πολιτική να είναι προεκχωρητική ή όχι
- Παράδειγμα: A=100, B = 2, Γ = 3


  - Πότε συμπεριφέρεται όμοια με την FCFS;
- Αποδεικνύεται ότι είναι βέλτιστη: Βάζοντας τη γρήγορη διεργασία πριν την αργή ευνοούμε τη γρήγορη «περισσότερο» απ' ότι επιβαρύνουμε την αργή
- Πρόβλημα;
  - Και αν έρχονται διαρκώς μικρές διεργασίες;
  - Και που να ξέρω το χρόνο που απομένει;

# Και που να ξέρω…



- Ας το πει ο χρήστης (αν κάνει λάθος σκοτώνω την εργασία)
  - Εφαρμόσιμο;
- Χρησιμοποίησε το παρελθόν…
  - … για να προβλέψεις το μέλλον
    - Π.χ. μια εργασία που εκτελέστηκε για πολύ χρόνο, πιθανότατα θα συνεχίσει να εκτελείται επί μακρόν…
  - Δες την εργασία σαν συνεχείς εναλλαγές υπολογισμού και I/O
    - Αν οι προηγούμενες φάσεις υπολογισμού κράτησαν λίγο μάλλον το ίδιο θα συμβεί και με τις επόμενες
- Αν παρελθόν <> μέλλον;

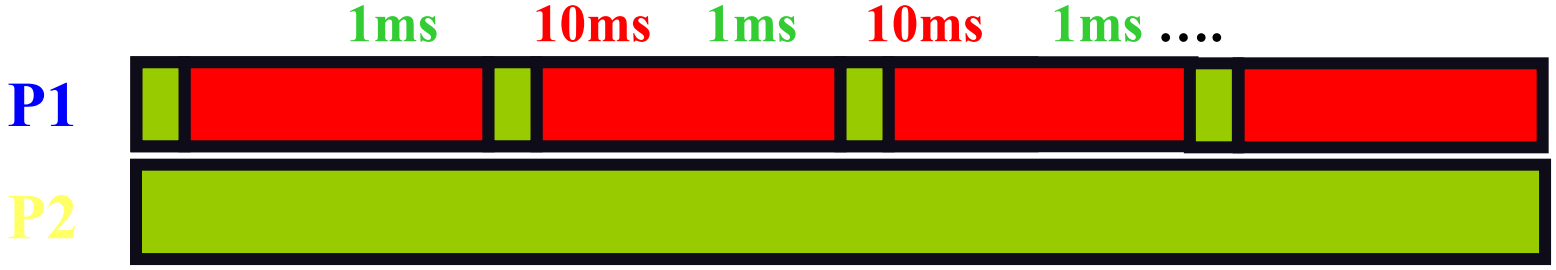
# STCF εκτός CPU;



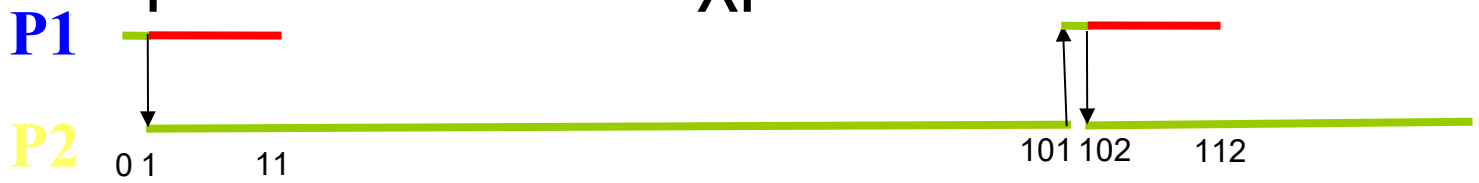
- Δίσκος
  - Εύκολο να προβλεφθεί το μήκος του επόμενου έργου:
    - Χρόνος για να μετακινηθεί η κεφαλή στο σωστό σημείο
    - Χρόνος για τη μεταφορά των δεδομένων
- STCF για δίσκους -> Shortest Seek Time First (SSTF)
  - Εκτέλεσε την ανάγνωση / εγγραφή κοντινότερα στην τρέχουσα θέση του βραχίονα
  - Προεκτοπιστική: Αν εμφανιστεί ανάγνωση / εγγραφή από άλλη διεργασία κοντά στην τρέχουσα θέση της κεφαλής, εκτέλεσε και αυτή

# STCF εναντίον RR

- Διεργασίες $P_1$, $P_2$



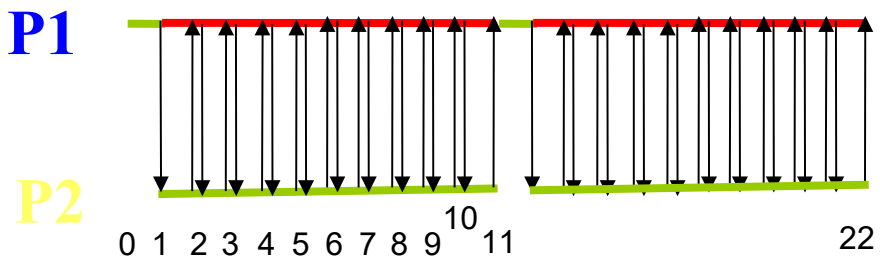

- RR με κβάντο 100 msec: I/O ανενεργό για περίπου 90% του χρόνου…



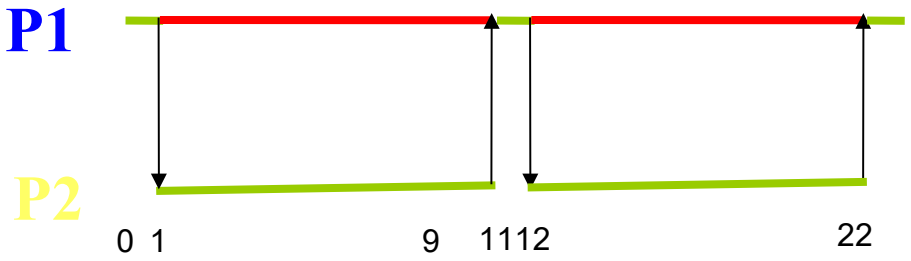
# STCF εναντίον RR



- RR με κβάντο 1 msec: Η $P_2$ θα διακοπτόταν 9/10 φορές χωρίς λόγο



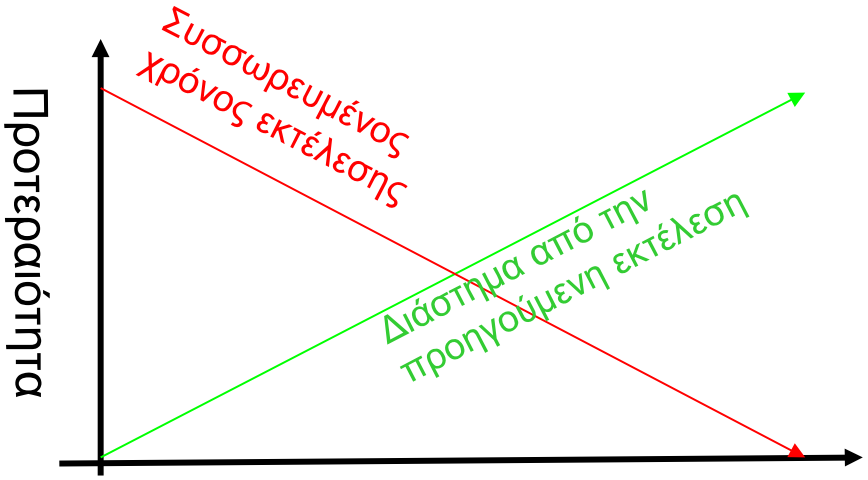

- Τι θα γινόταν με προεκχωρητική STCF?



# Βελτίωση της STCF



- Πρόβλημα;
  - Απεριόριστη αναμονή (πότε;)
- Λύση:
  - Συνέδεσε την «προτεραιότητα» τόσο με το πόσο χρόνο έχει τρέξει η διεργασία συνολικά, όσο και με το διάστημα από την προηγούμενη εκτέλεσή της



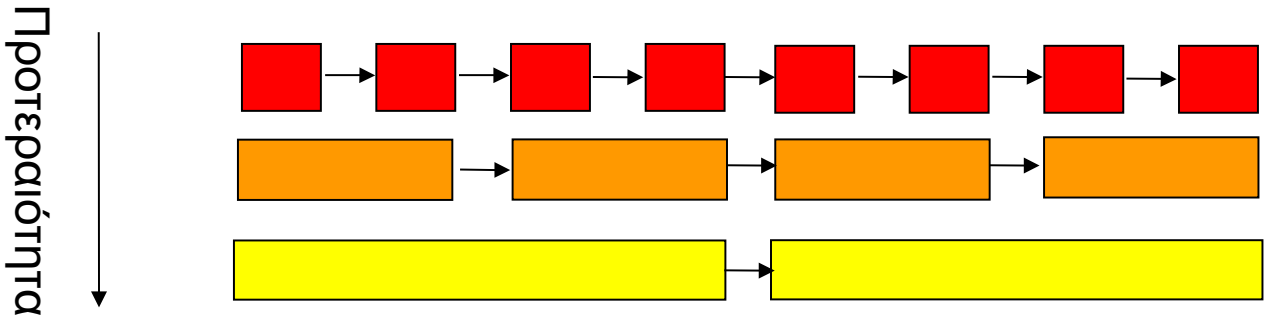
# Η Χρυσή Τομή: Πολυεπίπεδες Ουρές με Ανατροφοδότηση (ή Εκθετικές Ουρές)



- Επιτυγχάνουν καλή απόκριση, χρησιμοποίηση και αποδοτικότητα
  - Αποδοτικότητα: Μεγάλα κβάντα => μικρή επιβάρυνση λόγω μεταγωγών περιβάλλοντος
  - Απόκριση: Γρήγορη εκτέλεση μετά από ξεμπλοκάρισμα
- Ιδέα:
  - Πολλαπλές ουρές, μία για κάθε προτεραιότητα
  - Διαφορετικό κβάντο για κάθε προτεραιότητα
    - Ισχυρή προτεραιότητα => μικρό κβάντο
    - Και αντίστροφα…

# Εκθετικές Ουρές






- Αρχικά (νέα διεργασία):
  - Ισχυρή προτεραιότητα, μικρό κβάντο
  - Όταν διακοπεί πριν τελειώσει το κβάντο, μεταφορά στο τέλος της παρακάτω ουράς
  - Αν χρησιμοποιήσει όλο το κβάντο, μεταφορά σε χαμηλότερη προτεραιότητα
- Προβλήματα:
  - Διαδικασία «γήρανσης»: Μετακίνηση και προς τις ισχυρότερες προτεραιότητες
    - Παλιές συμπεριφορές δε μπορούν να επηρεάζουν επ' αόριστον την αντιμετώπιση της διεργασίας
  - Δε μπορούν οι διεργασίες χαμηλής προτεραιότητας να αντιμετωπίσουν απεριόριστη αναμονή;

# Σύνοψη



- FCFS
  - + Απλή
  - - Μικρά έργα μπορεί να κολλήσουν πίσω από μεγάλα, Κακή εκμετάλλευση του I/O
- RR
  - + Απλή, Δίκαιη, Καλύτερη για μικρές εργασίες
  - - Κακή για διεργασίες ίδιου μεγέθους
- SJF
  - + Βέλτιστη
  - - Δύσκολο να έχεις πληροφορία για τη διάρκεια που απομένει, Άδικη για μεγάλα έργα
- Προτεραιότητες
  - + Πολιτική ανάθεσης ανάλογα με τους στόχους του συστήματος, Δίκαιη (με βάση τους στόχους του συστήματος)
  - - Αντιστροφή προτεραιότητας, Αδιέξοδα
- Πολυεπίπεδες εκθετικές ουρές
  - + Αποδοτικότητα, Απόκριση, Χρησιμοποίηση
  - - Ανάγκη «γήρανσης», Πιθανόν άδικη για μεγάλα έργα

# Χρονοδρομολόγηση Λοταρίας



- Ανέθεσε σε κάθε διεργασία ένα ποσοστό του επεξεργαστικού χρόνου
  - A: 50%, B: 30%, C: 20%
  - Δεν υποστηρίζεται από καμία από τις πολιτικές που εξετάσαμε
- Ιδέα:
  - Δώσε «εισιτήρια» στις διεργασίες
  - Κάθε φορά που τρέχει μια διεργασία, χάνει ένα εισιτήριο
  - Κάθε διεργασία θα τρέξει σε μια «περίοδο» χρόνο ανάλογο με τον αριθμό εισιτηρίων της προς το συνολικό αριθμό εισιτηρίων
  - Όταν τελειώσουν τα εισιτήρια (τέλος περιόδου) ξαναμοίρασε…

# Χρονοδρομολόγηση Λοταρίας



- Χρήση:
  - Δώσε πολλά εισιτήρια σε σημαντικές διεργασίες, λίγα σε λιγότερο σημαντικές
- Παράδειγμα:
  - Προσέγγιση της SJF:
    - Περισσότερα εισιτήρια στις μικρές διεργασίες, λιγότερα στις μεγάλες
    - Αρκεί οι μεγάλες να έχουν 1 εισιτήριο
      - Αποφεύγεται ο κίνδυνος λιμοκτονίας
- Ερωτήματα:
  - Πολυπλοκότητα;
  - Μεταφορά προτεραιότητας;
  - Αν μπλοκάρει κάποια διεργασία;
  - Τι συμβαίνει αν αλλάξει ο αριθμός των διεργασιών;

# Άλλα Ζητήματα (1/2)



- Εργασίες πραγματικού χρόνου
  - Ευαίσθητες στο χρόνο
    - Απώλεια προθεσμίας ισοδυναμεί με λάθος συμπεριφορά
  - Soft (ελαστικές): Απεικόνιση μιας εικόνας σε ακολουθία video κάθε 1/30 του δευτερολέπτου.
  - Hard (ανελαστικές): Απόκριση αυτόματου πιλότου αεροσκάφους
- Ταυτόχρονη χρονοδρομολόγηση πολλών πόρων
  - Επεξεργαστής, μνήμη, εύρος ζώνης προς τη μνήμη, cache, εύρος ζώνης δικτύου

# Άλλα Ζητήματα (2/2)



- Κατανεμημένα συστήματα
  - Σύστημα όχι σε έναν κόμβο
    - Ενίοτε ούτε καν σε ένα δωμάτιο
  - Πώς παρακολουθείς το φορτίο;
  - Αν χρειαστεί να εξισορροπηθεί το φορτίο;
    - Μη αμελητέο κόστος
- Πληροφορία για τη διεργασία:
  - Από την τρέχουσα εκτέλεση
  - Γίνεται να έχουμε πληροφορία από προηγούμενες εκτελέσεις;
    - Π.χ. ένας μεταγλωττιστής συμπεριφέρεται συνήθως με παρόμοιο τρόπο
      - Η Matlab όμως όχι…

# Το Κόστος της Μεταγωγής Περιβάλλοντος

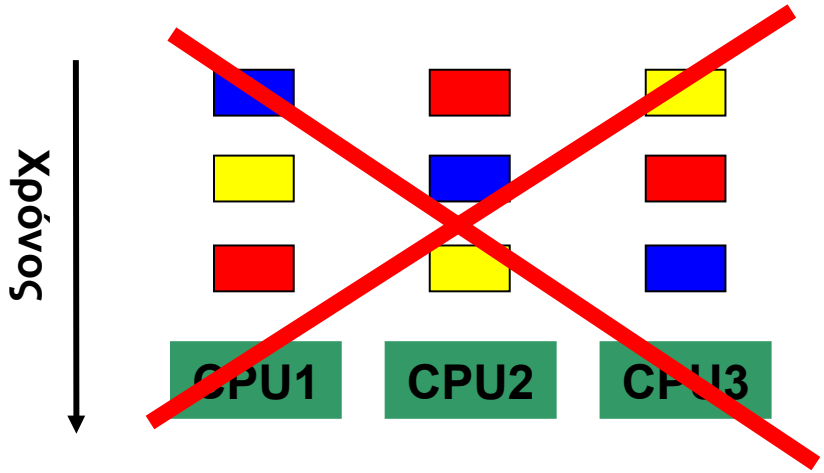


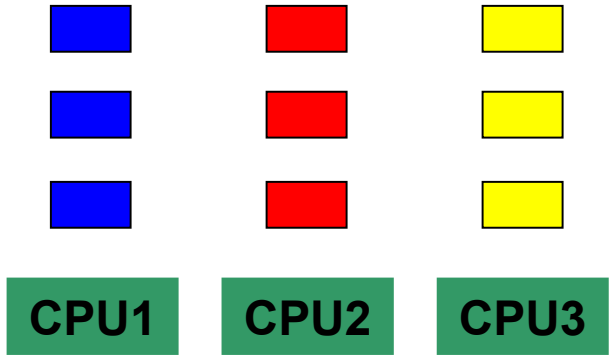
- Μη αμελητέο
  - Άμεσο (registers)
  - Έμμεσο (cache, TLB, …)
    - Ειδικά αν η διεργασία (μνήμη της) έχει πάει στο δίσκο
- Χρονοδρομολόγηση 2 επιπέδων
  - Σε πρώτη φάση διάλεξε και τρέξε μεταξύ των διεργασιών που έχεις στη μνήμη
  - Σε δεύτερη φάση άλλαξε το σύνολο των διεργασιών που έχεις στη μνήμη

# Συσχέτιση Επεξεργαστών – Διεργασιών σε Παράλληλα Συστήματα



- Μεταφορές μεταξύ επεξεργαστών:
  - Καταστροφή συσχέτισης (affinity)
    - Caches, TLB
- Λύση:
  - Προσπάθησε να τρέχεις τη διεργασία στον ίδιο επεξεργαστή
    - Όσο είναι δυνατό…

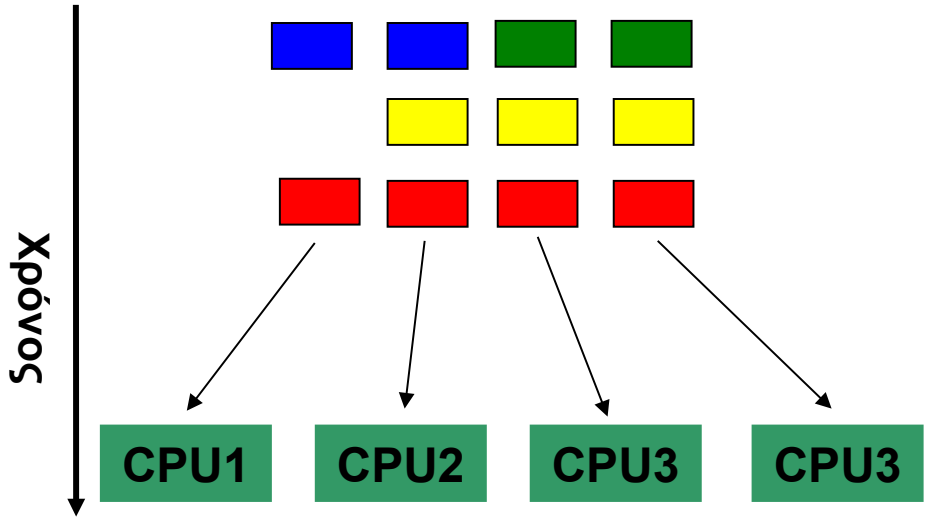


# Χρονοδρομολόγηση Ομάδων (Gang Scheduling) σε Παράλληλα Συστήματα



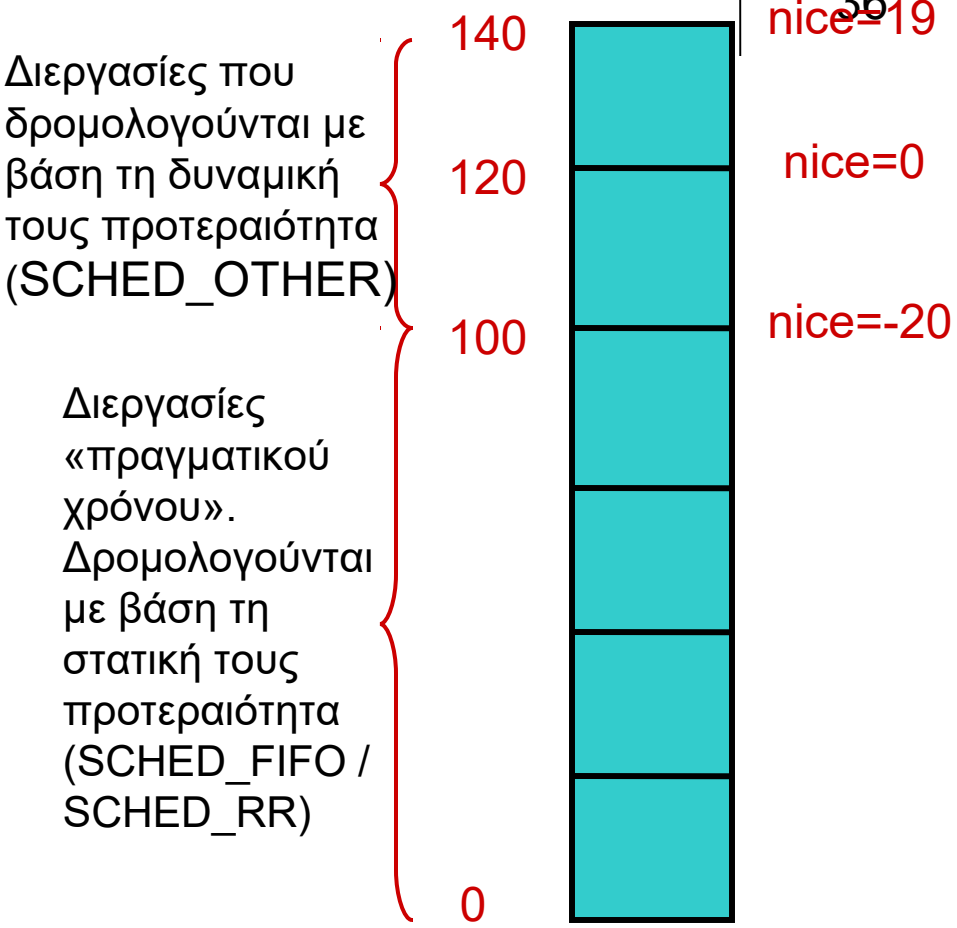
- Ανεξάρτητες διεργασίες
  - Εξισορρόπησε το φορτίο
  - Τρέξε κάθε μία στον επόμενο διαθέσιμο επεξεργαστή
    - Διατηρώντας στο μέτρο του δυνατού τη συσχέτιση διεργασίας-επεξεργαστή
- Εξαρτημένες διεργασίες (π.χ. πολλαπλές διεργασίες της ίδιας εφαρμογής)
  - Ομαδοποίησε τις εξαρτώμενες
  - Δρομολόγησε κάθε ομάδα σαν μία μονάδα (πάντα!)
    - + Αποφυγή αναστροφής προτεραιότητας / αδιεξόδων
    - + Δεν χρειάζονται μεταγωγές περιβάλλοντος για επικοινωνία
    - + Καλύτερη (συνήθως) εκμετάλλευση των cache
    - - Ενδεχομένως να μην εκμεταλλεύεται όλους τους πόρους (επεξεργαστές) …



# Ο Χρονοδρομολογητής του Linux O(1)



- Παραλλαγή των εκθετικών ουρών
  - 140 κλάσεις προτεραιότητας (μικρότερο -> καλύτερο)
  - 2 πολιτικές δρομολόγησης
    - SCHED_FIFO/ SCHED_RR («πραγματικού χρόνου»)
    - SCHED_OTHER (λοιπές)
  - 0-99 διεργασίες «πραγματικού χρόνου»
  - 100-140 λοιπές διεργασίες
    - Δυναμική προτεραιότητα ως στατική προτεραιότητα (τιμή nice) + πλεονέκτημα (ανάλογα με το βαθμό αλληλεπιδραστικότητας της διεργασίας)



# Ο Χρονοδρομολογητής του Linux O(1)- Προτεραιότητες



- Υπολογισμός προτεραιοτήτων στο τέλος κάθε χρονομεριδίου (epoch)
  - δυναμική προτεραιότητα = στατική προτεραιότητα (nice) + πλεονέκτημα (-5 … 5)
    - Το πλεονέκτημα εξαρτάται από το χρόνο που έχει περάσει η διεργασία μπλοκαρισμένη
  - 2 διανύσματα διεργασιών σε κάθε έτοιμη ουρά (runqueue) για κάθε προτεραιότητα
    - Ενεργές και Εξαντλημένες
- Γρήγορη O(1) αναγνώριση της διεργασίας που θα τρέξει στο επόμενο κβάντο

# Ο Χρονοδρομολογητής του Linux O(1)- Υπολογισμός Κβάντου



- Το epoch εξαρτάται από τη στατική προτεραιότητα:

| | | | | | |
|---|---|---|---|---|---|
| Nice | -20 | … | 0 | … | 19 |
| Epoch | 10ms | … | 100ms | … | 200ms |

- Λεπτομέρειες:
  - Οι αλληλεπιδραστικές διεργασίες επανέρχονται στην ενεργή λίστα, ακόμα και μετά το τέλος του epoch τους
    - Εκτός αν υπάρχουν διεργασίες στη μη ενεργή λίστα που «λιμοκτονούν» (δεν εξυπηρετούνται)
  - Διεργασίες ίδιας προτεραιότητας, με μεγάλα epoch, μπορούν να εναλλάσσονται από το χρονοδρομολογητή ακόμα και μέσα στο epoch.

# Ο Χρονοδρομολογητής του Linux O(1)- Προβλήματα



- Προβλήματα με τον χρονοδρομολογητή O(1)
  - Καλή επίδοση σε servers, καλή κλιμακωσιμότητα
  - Προβληματάκια με αλληλεπιδραστικές εφαρμογές...
  - Πολλά heuristics για να εντοπιστούν οι αλληλεπιδραστικές εφαρμογές
  - Περίεργη συμπεριφορά στην αλλαγή διανυσμάτων
    - Απότομες αλλαγές συμπεριφοράς την ώρα της αλλαγής

# Ο Χρονοδρομολογητής του Linux CFS



- Προτοεμφανίστηκε στον 2.6.23, μεταβλήθηκε στον 2.6.24
- “CFS doesn't track sleeping time and doesn't use heuristics to identify interactive tasks—it just makes sure every process gets a fair share of CPU within a set amount of time given the number of runnable processes on the CPU.”
- Εμπνευσμένο από το “Fair Queueing” στην περιοχή των δικτύων
  - Κάθε διεργασία παίρνει το ίδιο ποσοστό πόρων
  - Μοντελοποιεί έναν “ιδανικό επεξεργαστή multitasking” στον οποίο Ν διεργασίες εκτελούνται ταυτόχρονα, λαμβάνοντας καθεμιά 1/Ν του επεξεργαστή
  - Προσπαθεί να δώσει σε κάθε διεργασία ίσο μερίδιο του επεξεργαστή
- Οι προτεραιότητες εκφράζονται με βάρη:
  - Αυξάνοντας την προτεραιότητα μιας διεργασίας κατά 1 οδηγεί πάντα στην ίδια ποσοστιαία αύξηση του ποσοστού χρόνου στη CPU – ανεξαρτήτως της προηγούμενης προτεραιότητας

# Ο Χρονοδρομολογητής του Linux CFS



- Ιδέα: Μέτρησε το ποσό “εικονικού χρόνου” που παίρνει κάθε διεργασία ενώ εκτελείται
  - Πάρε τον πραγματικό χρόνο εκτέλεσης και διαίρεσέ τον με κάποιο βάρος.
  - Χαμηλότερη προτεραιότητα => Ο πραγματικός χρόνος διαιρείται με μεγαλύτερο βάρος
    - Στην πραγματικότητα πολλαπλασιάζεται με το άθροισμα όλων των βαρών δια το βάρος της διεργασίας
  - Φρόντισε ο εικονικός χρόνος να αυξάνει με τον ίδιο ρυθμό σε όλες
- Καθυστέρηση-στόχος : Χρονική περίοδος μετά την οποία όλες οι διεργασίες θα έχουν τρέξει, έστω και λίγο.
  - Κάθε διεργασία έχει ένα κβάντο
  - Ποτέ μικρότερο από ένα ελάχιστο κβάντο
- Red-Black δέντρο για την αποθήκευση όλων των διεργασιών ταξινομημένων με βάση το vruntime τους (εικονικό χρόνο εκτέλεσης)
  - O(log n) για εισαγωγές / διαγραφές
  - Διάλεξε πάντα τη διεργασία με το χαμηλότερο vruntime (αυτή τέρμα αριστερά)

# Ο Χρονοδρομολογητής του Linux CFS - Παραδείγματα



- Έστω Targeted latency = 20ms,
- Ελάχιστο κβάντο = 1ms
- 2 υπολογιστικές διεργασίες με ίδιες προτεραιότητες
  - Αλλαγές κάθε 10ms
- Δύο υπολογιστικές διεργασίες με διαφορά προτεραιότητας 5
  - Μία διεργασία παίρνει 5ms, η άλλη 15ms
- 40 διεργασίες: κάθε μια παίρνει 1ms (πλέον όχι απόλυτα δίκαιος)
- Μία υπολογιστική διεργασία και μία αλληλεπιδραστική με ίδια προτεραιότητα
  - Όσο η αλληλεπιδραστική κοιμάται, η υπολογιστική τρέχει και αυξάνει το vruntime
  - Όταν η αλληλεπιδραστική ξυπνήσει, τρέχει κατευθείαν, αφού είναι πίσω στο vruntime
- Δυνατότητες group scheduling (2.6.24 και μεταγενέστεροι)
  - Δυνατή η ανάθεση ποσοστών της CPU σε ομάδες (π.χ. χρήστες ή process groups)
  - Π.χ. 2 χρήστες, ένας ξεκινάει 1 διεργασία, ο άλλος ξεκινάει 40, καθένας παίρνει 50% του χρόνου του επεξεργαστή

# Ο Χρονοδρομολογητής του Linux – Εξισορρόπηση Φόρτου σε Πολυεπεξεργαστικά Συστήματα



- Κάθε επεξεργαστής έχει τις δικές του runqueues
- Περιοδικά ελέγχεται ο πληθυσμός των runqueues
  - Αν υπάρχουν σημαντικές διαφορές γίνεται εξισορρόπηση
  - Προβλήματα;
    - Καταστροφή συσχέτισης επεξεργαστή – διεργασίας
      - Υπομονή για λίγο…
    - Συγχρονισμός σε πολλαπλές ουρές (πολλαπλά locks)