ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Λειτουργικά Συστήματα

Ενότητα # 2: Διεργασίες και Νήματα

Διδάσκων: Γεώργιος Ξυλωμένος

Τμήμα: Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Οικονομικό Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

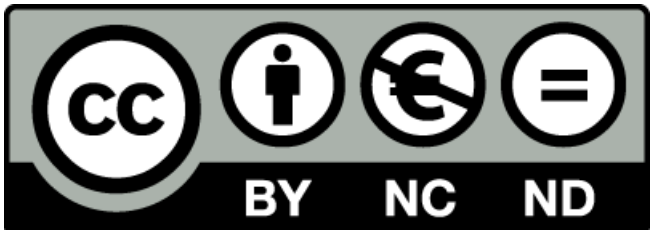
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Οι εικόνες προέρχονται από το βιβλίο «Σύγχρονα Λειτουργικά Συστήματα», A.S. Tanenbaum, 3η έκδοση, 2009, Εκδόσεις Κλειδάριθμος.



Σκοποί ενότητας

- Κατανόηση της έννοιας της διεργασίας και του νήματος εκτέλεσης
- Κατανόηση των διακριτών ρόλων των διεργασιών και των νημάτων
- Εξοικείωση με τους μηχανισμούς διαδιεργασιακής επικοινωνίας
- Κατανόηση των βασικών αλγορίθμων χρονοπρογραμματισμού του επεξεργαστή

Περιεχόμενα ενότητας

- Διεργασίες
- Νήματα
- Διαδιεργασιακή επικοινωνία
- Χρονοπρογραμματισμός
- Κλασικά προβλήματα

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Διεργασίες

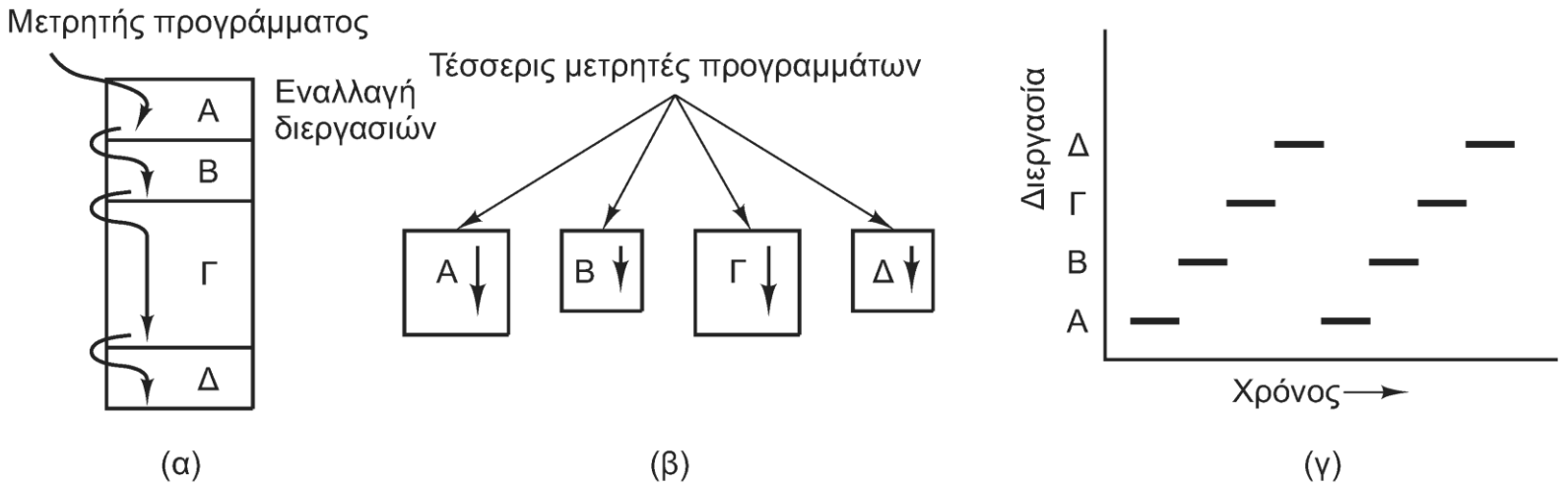
Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Το μοντέλο της διεργασίας (1 από 3)

- Διεργασία (process)
  - Σε κάθε ΛΣ έχουμε πολλές διεργασίες ταυτόχρονα
    - Εξυπηρετητής: πολλές διεργασίες εξυπηρέτησης
    - Πελάτης: φυλλομετρητής, αντι-ιικό, εκτύπωση
  - Ψευδο-ταυτόχρονη εκτέλεση σε μία ΚΜΕ
    - Γρήγορη εναλλαγή διεργασιών
    - Ο προγραμματιστής δεν ασχολείται με τον ταυτοχρονισμό
  - Μοντέλο ακολουθιακής διεργασίας
    - Εκτελούμενο πρόγραμμα: δεδομένα, καταχωρητές, μετρητής
    - Η εναλλαγή διεργασιών ονομάζεται πολυπρογραμματισμός

Το μοντέλο της διεργασίας (2 από 3)



- Μοντέλο διεργασίας
  - Οι διεργασίες φαίνεται ότι εκτελούνται ανεξάρτητα
  - Κάθε ΚΜΕ όμως ασχολείται με μία διεργασία τη φορά
  - Η ταχύτητα εκτέλεσης των διεργασιών είναι απρόβλεπτη
    - Εξαρτάται από τον τρόπο εναλλαγής των διεργασιών
    - Μπορεί να αλλάζει σε κάθε εκτέλεση

Το μοντέλο της διεργασίας (3 από 3)

- Παρενέργειες της ψευδο-ταυτόχρονης εκτέλεσης
  - Δεν κάνουμε ποτέ υποθέσεις για την ταχύτητα εκτέλεσης
    - Το εκτελούμενο πρόγραμμα μπορεί να αλλάξει οποτεδήποτε
- Διάκριση προγράμματος και διεργασίας
  - Το πρόγραμμα είναι μία συνταγή, η διεργασία η εκτέλεση
  - Το ίδιο πρόγραμμα μπορεί να εκτελείται ταυτόχρονα
    - Ο φλοιός εκτελείται από πολλούς χρήστες ταυτόχρονα
- Δημιουργία διεργασιών
  - Σε κλειστά συστήματα οι διεργασίες δημιουργούνται μαζί
  - Γενικά όμως οι διεργασίες δημιουργούνται δυναμικά

Δημιουργία διεργασίας (1 από 2)

- Αρχικοποίηση του συστήματος
  - Διεργασίες προσκηνίου όπως η διεργασία σύνδεσης
  - Διεργασίες παρασκηνίου όπως ο έλεγχος για e-mail
- Δημιουργία διεργασίας από εκτελούμενη διεργασία
  - Καταμερισμός εργασίας (σωληνωτά ή παράλληλα)
- Αίτηση χρήστη για δημιουργία διεργασίας
  - Από το φλοιό ή από γραφική διεπαφή
- Εκκίνηση διεργασίας δέσμης
  - Ξεκινάει όταν έρθει η σειρά της (επιλεγεί προς εκτέλεση)

Δημιουργία διεργασίας (2 από 2)

- Πώς δημιουργούνται οι διεργασίες;
  - Στο UNIX γίνεται με την κλήση fork()
    - Δημιουργία αντιγράφου της εκτελούμενης διεργασίας
    - Ίδια μνήμη, περιβάλλον και ανοιχτά αρχεία
  - Αλλαγή εικόνας μνήμης με την κλήση exec()
    - Μέχρι τότε εκτελείται η αρχική διεργασία
    - Μπορεί να φροντίσει π.χ. για ανακατεύθυνση αρχείων
  - Στα Windows όλα γίνονται με την κλήση CreateProcess
    - Μπορεί να έχει διαφορετικά χαρακτηριστικά ασφάλειας
  - Η νέα διεργασία έχει πάντα ανεξάρτητο χώρο μνήμης

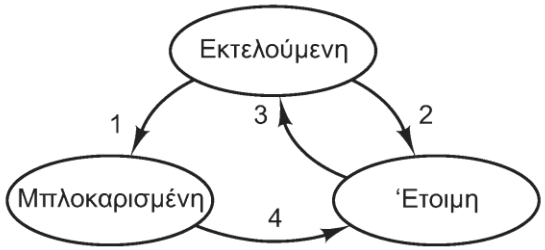
Τερματισμός διεργασίας

- Κανονική έξοδος (εκούσια)
  - Το πρόγραμμα ολοκληρώνει την εκτέλεσή του
  - Exit στο UNIX, ExitProcess στα Windows
- Έξοδος που προκλήθηκε από σφάλμα (εκούσια)
  - Παράδειγμα: οι παράμετροί του είναι λανθασμένες
- Μοιραίο σφάλμα (ακούσια)
  - Παράδειγμα: διαίρεση με το μηδέν, δείκτης εκτός ορίων
- Θανάτωση από άλλη διεργασία (ακούσια)
  - Αποστολή σήματος θανάτωσης από άλλη διεργασία

Ιεραρχίες διεργασιών

- Ομάδα διεργασιών στο UNIX: διεργασία και απόγονοι
  - Μπορούν να σταλούν σήματα σε όλη την ομάδα
  - Κάθε διεργασία αποφασίζει πώς θα τα χειριστεί
- Όλες οι διεργασίες στο UNIX είναι απόγονοι της init
  - Δημιουργεί διεργασίες για κάθε τερματικό
  - Οι διεργασίες αυτές τελικά δημιουργούν έναν φλοιό
  - Ο φλοιός εκτελείται για λογαριασμό του χρήστη
- Στα Windows δεν υπάρχει ιεραρχία διεργασιών
  - Η πατρική διεργασία έχει χειριστήριο προς το παιδί
  - Μπορεί όμως να μεταβιβαστεί και σε άλλες διεργασίες

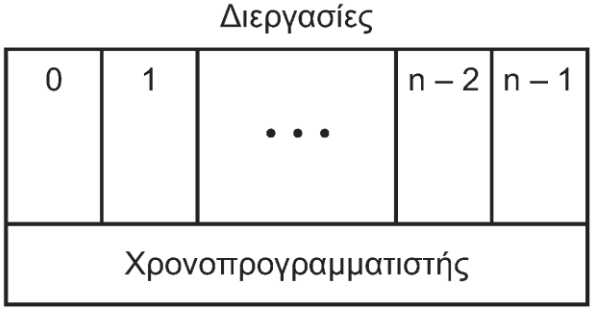
Καταστάσεις διεργασιών (1 από 4)



1. Η διεργασία μπλοκάρεται καθώς περιμένει δεδομένα εισόδου.
2. Ο χρονοπρογραμματιστής επιλέγει άλλη διεργασία.
3. Ο χρονοπρογραμματιστής επιλέγει τη συγκεκριμένη διεργασία.
4. Τα δεδομένα εισόδου είναι διαθέσιμα.

- Ενδογενής εμποδισμός εκτελούμενης διεργασίας
  - Η διεργασία δεν προχωρά μέχρι να συμβεί κάτι
    - Παράδειγμα: είσοδος από συσκευή, σήμα από άλλη διεργασία
- Εξωγενής εμποδισμός εκτελούμενης διεργασίας
  - Η διεργασία σταματά να εκτελείται λόγω του συστήματος
    - Παράδειγμα: εξάντληση κβάντου, προτεραιότητα σε άλλη
- Η μπλοκαρισμένη διεργασία μπορεί να γίνει έτοιμη
- Η έτοιμη διεργασία μπορεί να γίνει εκτελούμενη

Καταστάσεις διεργασιών (2 από 4)



- Λειτουργία συστήματος πολυπρογραμματισμού
  - Το σύστημα εναλλάσσει πολλές διεργασίες
  - Οι διεργασίες μπλοκάρονται όταν περιμένουν γεγονός
    - Από εκτελούμενες γίνονται μπλοκαρισμένες
    - Θα γίνουν έτοιμες μετά από διακοπή ή σήμα
    - Θα εκτελεστούν όταν τις επιλέξει ο χρονοπρογραμματιστής
  - Ο χρονοπρογραμματιστής αποκρύπτει την εναλλαγή
    - Διακοπές, εκκίνηση και διακοπή εκτέλεσης δεν είναι ορατά

Καταστάσεις διεργασιών (3 από 4)

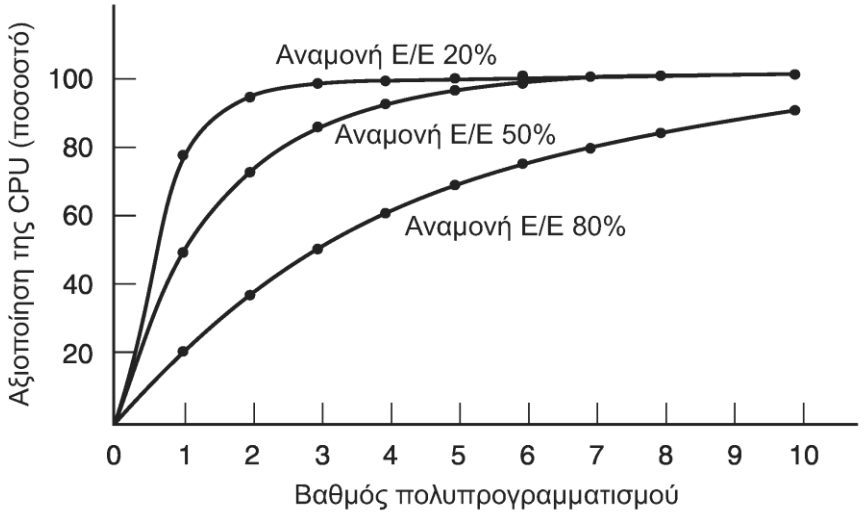
Διαχείριση διεργασιών	Διαχείριση μνήμης	Διαχείριση αρχείων
Καταχωρητές	Δείκτης προς πληροφορίες του τμήματος κώδικα	Βασικός κατάλογος
Μετρητής προγράμματος	Δείκτης προς πληροφορίες του τμήματος δεδομένων	Κατάλογος εργασίας
Δείκτης στοίβας	Δείκτης προς πληροφορίες του τμήματος στοίβας	Περιγραφείς αρχείων
Κατάσταση διεργασίας		Ταυτότητα χρήστη (User ID)
Προτεραιότητα		Ταυτότητα ομάδας (Group ID)
Παράμετροι χρονοπρογραμματισμού		
Ταυτότητα διεργασίας (PID)		
Ταυτότητα γονικής διεργασίας		
Ομάδα διεργασίας		
Σήματα		
Χρόνος εκκίνησης διεργασίας		
Χρόνος χρήσης CPU		
Χρόνος CPU θυγατρικών διεργασιών		
Χρονική στιγμή επόμενης ειδοποίησης		

- Υλοποίηση διεργασιών
  - Πίνακας διεργασιών: μία δομή ανά διεργασία
  - Επιτρέπει την διακοπή και συνέχιση των διεργασιών

Καταστάσεις διεργασιών (4 από 4)

- Υλοποίηση πολυπρογραμματισμού
  - Σε σταθερό σημείο της μνήμης υπάρχει πίνακας διακοπών
  - Ο πίνακας περιέχει ένα διάνυσμα διακοπής ανά διακοπή
  - Όταν συμβαίνει μία διακοπή
    - Αποθηκεύεται μετρητής και κατάσταση επεξεργαστή
    - Ο μετρητής φορτώνεται από το διάνυσμα διακοπής
    - Η διαδικασία εξυπηρέτησης αποθηκεύει την κατάσταση
    - Η διαδικασία εξυπηρέτησης δημιουργεί μια νέα στοίβα
    - Διεκπεραίωση της διακοπής (μπορεί να ελευθερώσει διεργασία)
    - Ο χρονοπρογραμματιστής επιλέγει την επόμενη διεργασία

Μοντελοποίηση πολυπρογραμματισμού



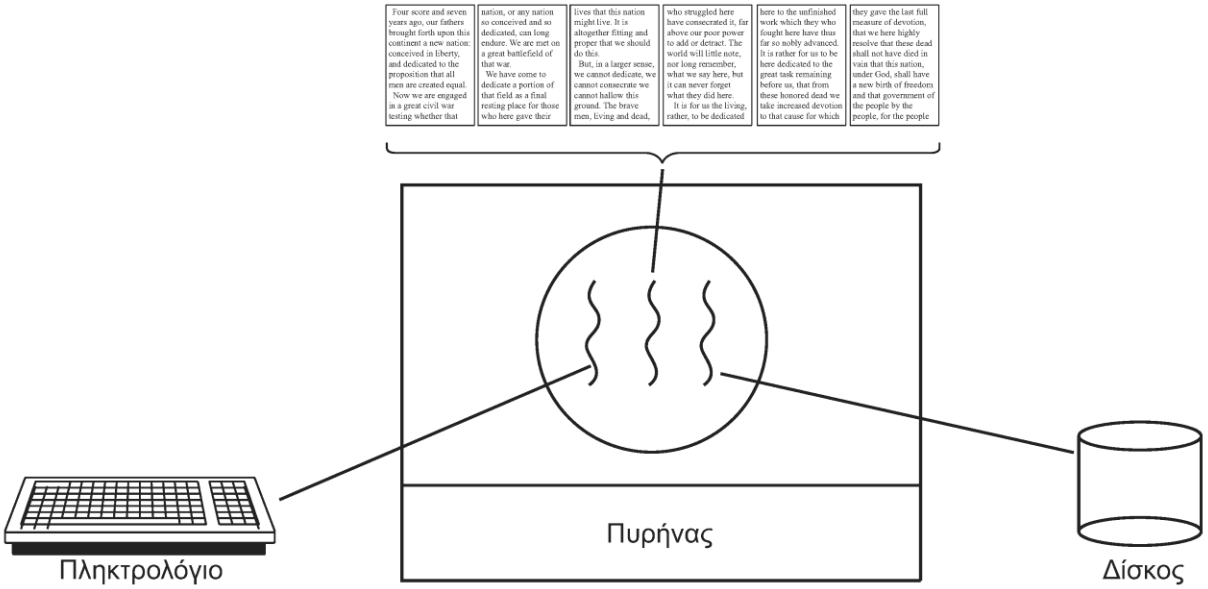
- Μοντελοποίηση του πολυπρογραμματισμού
  - Έστω ότι μία διεργασία καταναλώνει p σε είσοδο/έξοδο
  - Αν έχουμε n διεργασίες ο βαθμός αξιοποίησης είναι $1-p^n$
    - p^n: Πιθανότητα όλες οι διεργασίες να είναι σε αναμονή
  - Όσο μεγαλώνει το p, θέλουμε περισσότερες διεργασίες

Νήματα

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

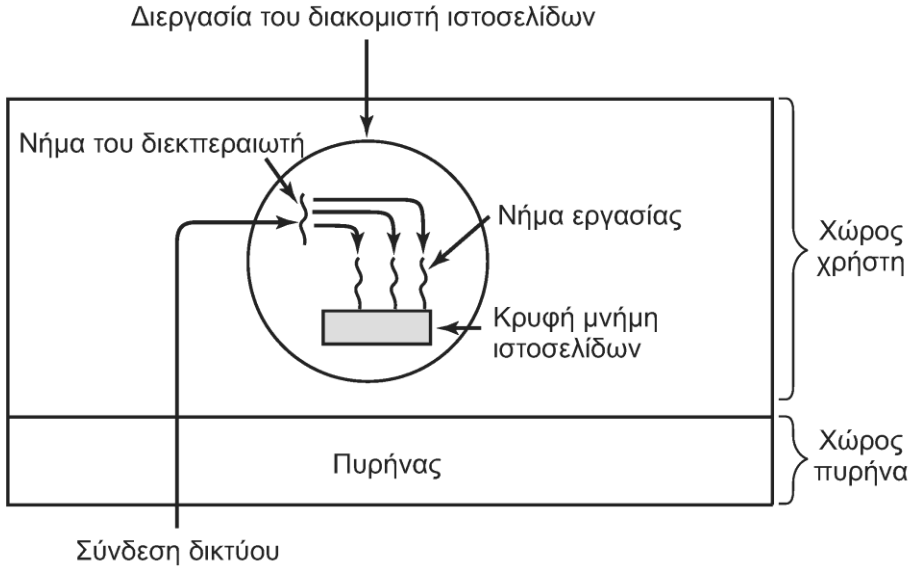
Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Χρήση των νημάτων (1 από 6)



- Νήματα ελέγχου: ροές εκτέλεσης στην ίδια διεργασία
  - Τα νήματα μοιράζονται τη μνήμη της ίδιας διεργασίας
  - Επικοινωνία μέσω της κοινής μνήμης
  - Πιο οικονομική διαχείριση από τις διεργασίες
  - Παράδειγμα: επεξεργαστής κειμένου

Χρήση των νημάτων (2 από 6)



- Οργάνωση νημάτων
  - Παράδειγμα: εξυπηρετητής Ιστού
  - Το νήμα-διεκπεραιωτής δέχεται αιτήσεις από το δίκτυο
  - Τα νήματα-εργάτες εκτελούν τις αιτήσεις
  - Όλα τα νήματα έχουν πρόσβαση στην κρυφή μνήμη

Χρήση των νημάτων (3 από 6)

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b)

- Δομή κώδικα εξυπηρετητή
  - Ο διεκπεραιωτής απλά μεταβιβάζει δουλειές
  - Οι εργάτες εξυπηρετούν μία αίτηση κάθε φορά
  - Πολλοί εργάτες εκτελούνται ταυτόχρονα
  - Ο εμποδισμός κρύβεται από τα νήματα

Χρήση των νημάτων (4 από 6)

- Εναλλακτική λύση: ασύγχρονες κλήσεις
  - Χρησιμοποιεί διεργασίες αντί για νήματα
  - Κάθε κλήση γίνεται ασύγχρονα
    - Οι κλήσεις Ε/Ε δεν εμποδίζουν τη διεργασία
    - Τα αποτελέσματα διακόπτουν τη διεργασία
  - Η διεργασία παρακολουθεί όλες τις κλήσεις
    - Ξεκινάει και σταματάει αιτήσεις
    - Ουσιαστικά προσομοιώνει τα νήματα

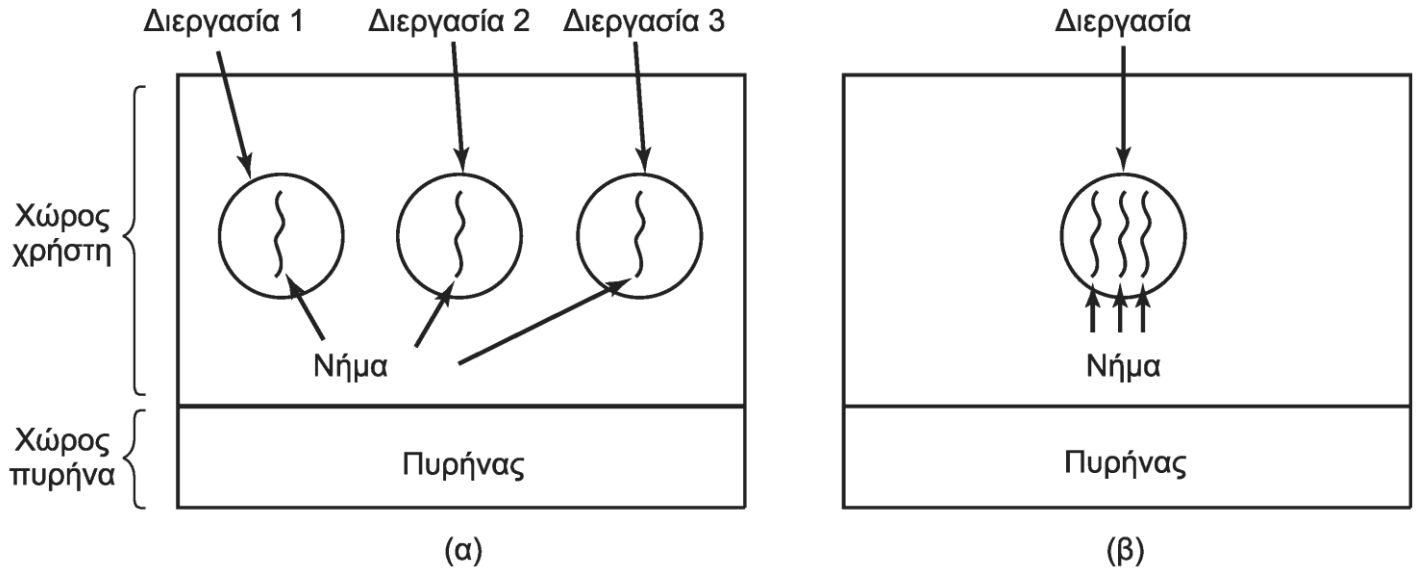
Χρήση των νημάτων (5 από 6)

- Τα νήματα απλοποιούν τον κώδικα
  - Συνδυάζουν πολυπρογραμματισμό με κοινή μνήμη
  - Οι κλήσεις εισόδου/εξόδου οδηγούν σε εμποδισμό
    - Τα υπόλοιπα νήματα συνεχίζουν να εκτελούνται κανονικά
  - Η επικοινωνία μεταξύ των νημάτων είναι γρήγορη
    - Γίνεται μέσω κοινής μνήμης
    - Δεν χρειάζεται παρέμβαση του πυρήνα

Χρήση των νημάτων (6 από 6)

Μοντέλο	Γνωρίσματα
Νήματα	Παράλληλα κλήσεις συστήματος που μπλοκάρουν (ανασταλτικές)
Διεργασία με ένα νήμα	Δεν υπάρχει παραλληλία, κλήσεις συστήματος που μπλοκάρουν (ανασταλτικές)
Μηχανή πεπερασμένων καταστάσεων	Παραλληλία, κλήσεις συστήματος που δεν μπλοκάρουν (μη ανασταλτικές), διακοπές

Κλασικό μοντέλο νημάτων (1 από 5)



- Η διεργασία ομαδοποιεί ένα σύνολο πόρων
  - Κώδικας, δεδομένα, αρχεία, σήματα
- Κάθε νήμα είναι μία ροή εκτέλεσης μέσα στη διεργασία
  - Μετρητής, καταχωρητές, στοίβα
- Τα νήματα μοιράζονται τους πόρους της ίδιας διεργασίας

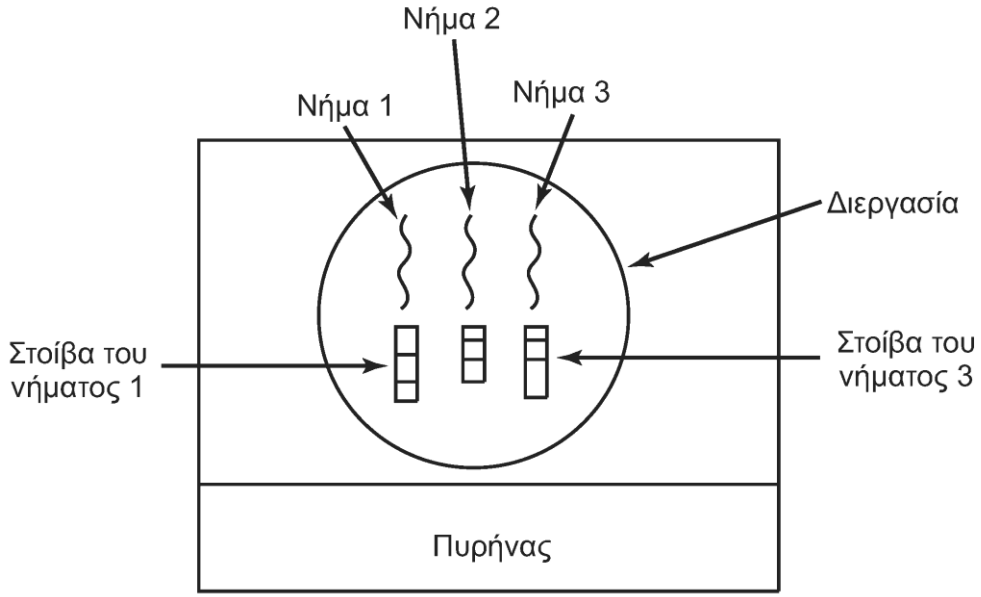
Κλασικό μοντέλο νημάτων (2 από 5)

Στοιχεία ανά διεργασία	Στοιχεία ανά νήμα
Χώρος διευθύνσεων	Μετρητής προγράμματος
Καθολικές μεταβλητές	Καταχωρητές
Ανοιχτά αρχεία	Στοίβα
Θυγατρικές διεργασίες	Κατάσταση
Εκκρεμή σήματα συναγερμού	
Σήματα και χειριστές σημάτων	
Διαχειριστικές πληροφορίες	

Κλασικό μοντέλο νημάτων (3 από 5)

- Τα νήματα δεν προστατεύονται
  - Δεν απομονώνονται όπως οι διεργασίες
  - Οι πόροι είναι κοινοί σε όλα τα νήματα
    - Τα νήματα μπορούν εύκολα να ανταλλάσσουν δεδομένα
    - Μπορούν όμως και να καταστρέφουν το ένα το άλλο
    - Είναι ευθύνη του προγραμματιστή να προσέχει
- Καταστάσεις νημάτων: όπως των διεργασιών
  - Εκτελούμενο, έτοιμο, μπλοκαρισμένο

Κλασικό μοντέλο νημάτων (4 από 5)



- Κάθε νήμα έχει τη δική του στοίβα
  - Η στοίβα περιέχει ένα πλαίσιο για κάθε εκκρεμή κλήση
  - Το πλαίσιο περιέχει τις τοπικές μεταβλητές της κλήσης
  - Κάθε νήμα βρίσκεται σε διαφορετικό σημείο εκτέλεσης

Κλασικό μοντέλο νημάτων (5 από 5)

- Δημιουργία και καταστροφή νημάτων
  - Κάθε διεργασία ξεκινάει με ένα μόνο νήμα
  - Κάθε νήμα μπορεί να δημιουργήσει άλλα
    - Πρέπει να προσδιορίζεται το σημείο εκκίνησης του νέου νήματος
  - Τα νήματα μπορεί να οργανώνονται ιεραρχικά
  - Ο τερματισμός είναι παρόμοιος με των διεργασιών
  - Τα νήματα μπορούν να παραδίδουν την ΚΜΕ οικειοθελώς
- Προβλήματα του μοντέλου των νημάτων
  - Όταν κάνουμε fork αντιγράφονται και όλα τα νήματα;
  - Όταν κλείνει ένα αρχείο κλείνει για όλα τα νήματα;

Νήματα στο POSIX (1 από 3)

Κλήση νήματος	**Περιγραφή**
Pthread_create	Δημιουργία νήματος.
Pthread_exit	Τερματισμός του καλούντος νήματος.
Pthread_join	Αναμονή για έξοδο.
Pthread_yield	Αποδέσμευση της CPU ώστε να μπορεί να ξεκινήσει ένα άλλο νήμα.
Pthread_attr_init	Δημιουργία και απόδοση αρχικών τιμών στη δομή χαρακτηριστικών ενός νήματος.
Pthread_attr_destroy	Διαγραφή της δομής χαρακτηριστικών ενός νήματος.

Νήματα στο POSIX (2 από 3)

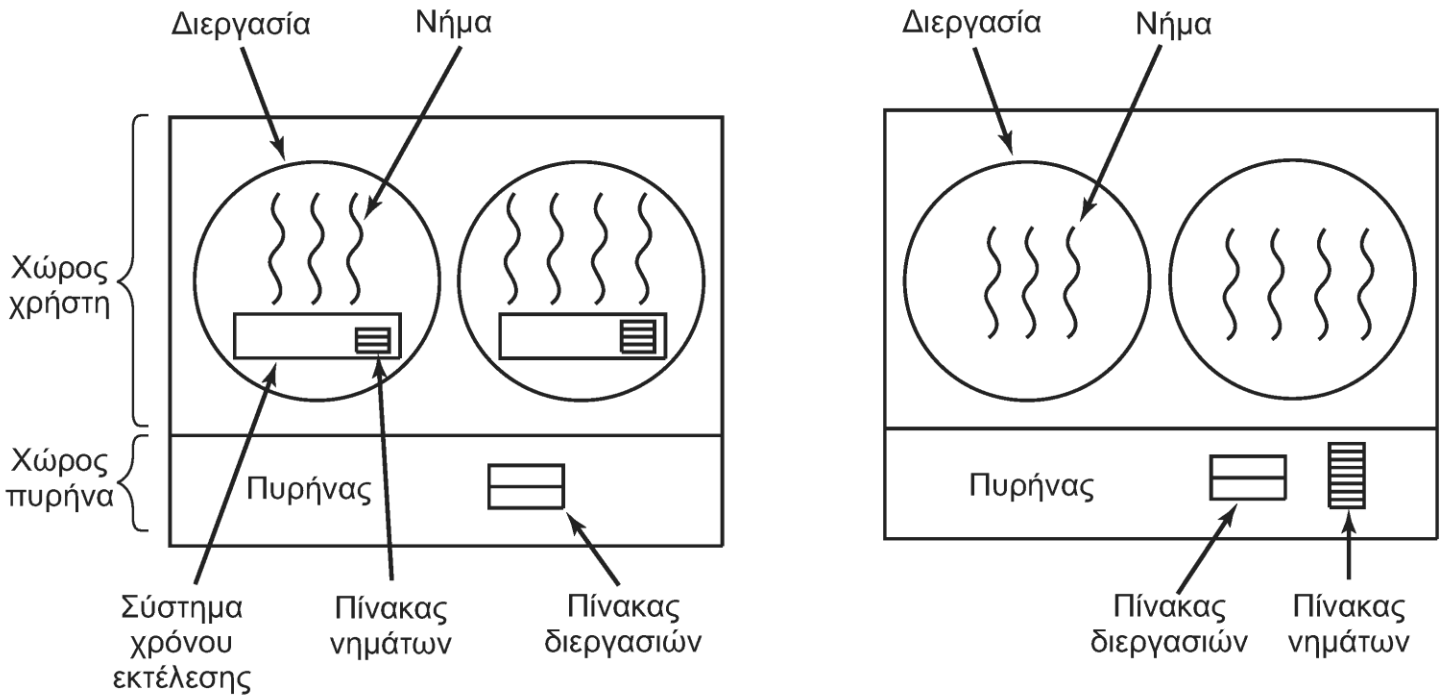
- Νήματα στο POSIX (πακέτο pthreads)
  - Κάθε νήμα περιγράφεται από τυποποιημένη δομή
    - Μεταβιβάζεται κατά τη δημιουργία του νήματος
    - Μπορούμε να αφήσουμε προεπιλεγμένες τιμές
  - Η δημιουργία νήματος επιστρέφει ένα αναγνωριστικό
    - Παρόμοιο με το αναγνωριστικό διεργασίας
    - Ο πατέρας μπορεί να περιμένει τον τερματισμό του παιδιού
  - Το νήμα μπορεί να παραχωρήσει τον επεξεργαστή

Νήματα στο POSIX (3 από 3)

```
#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
      /* Η συνάρτηση αυτή τυπώνει το αναγνωριστικό του νήματος και τερματίζεται. */
      printf("Γεια σου κόσμε. Χαιρετισμούς από το νήμα %d0", tid);
      pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
      /* Το κύριο πρόγραμμα δημιουργεί 10 νήματα και τερματίζεται. */
      pthread_tthreads[NUMBER_OF_THREADS];
      int status, i;

      for(i=0; i < NUMBER_OF_THREADS; i++) {
            printf("Κύριο εδώ. Δημιουργία νήματος %d0", i);
            status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

            if (status != 0) {
                  printf("Ουπς. Η pthread_create επέστρεψε κωδ. σφάλμ. %d0", status);
                  exit(–1);
```

Παράδειγμα χρήσης νημάτων (πακέτο pthreads).

Υλοποίηση νημάτων



- Υλοποίηση νημάτων επιπέδου χρήστη
- Υλοποίηση νημάτων επιπέδου πυρήνα
- Υβριδική υλοποίηση

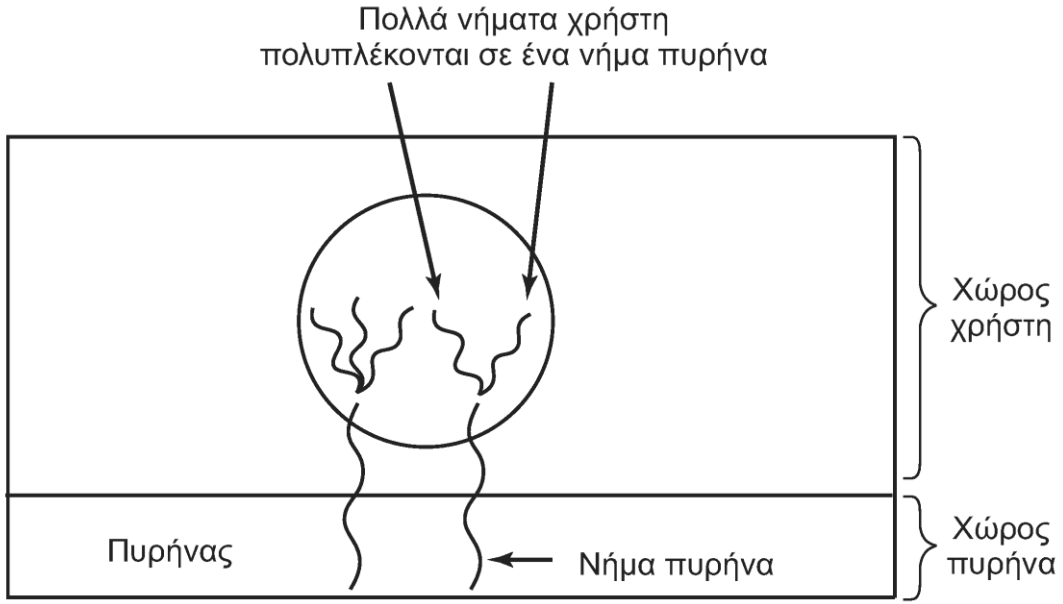
Στο χώρο του χρήστη

- Υλοποίηση επιπέδου χρήστη
  - Τα νήματα υλοποιούνται μέσω μιας βιβλιοθήκης
  - Το λειτουργικό σύστημα δεν εμπλέκεται καθόλου
    - Υλοποίηση σε συστήματα που δεν υποστηρίζουν νήματα
  - Κάθε διεργασία υλοποιεί χρονοπρογρογραμματιστή
  - Το μπλοκάρισμα ενός νήματος μπλοκάρει τη διεργασία
    - Παράκαμψη του προβλήματος με ελεγχόμενες κλήσεις
  - Τα νήματα δεν μπορούν να διακοπούν από το σύστημα
    - Πρέπει να παραδώσουν οικειοθελώς τον έλεγχο
  - Δυστυχώς τα νήματα χρησιμεύουν στο μπλοκάρισμα!

Στο χώρο του πυρήνα

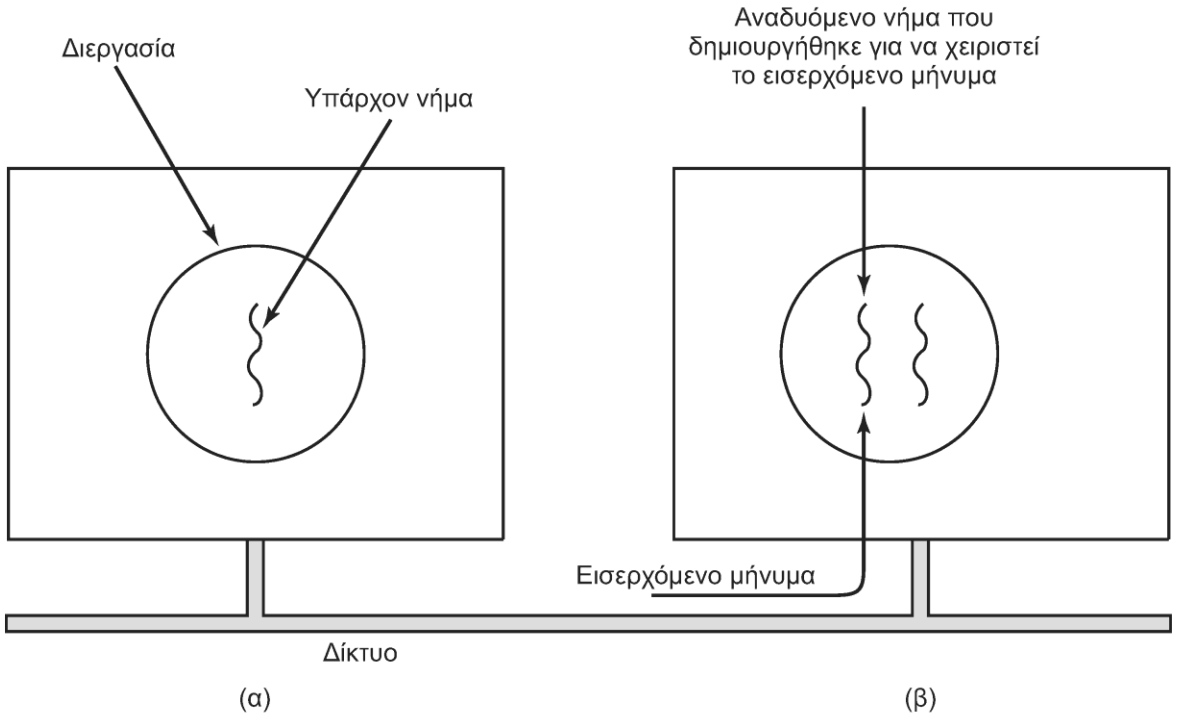
- Υλοποίηση επιπέδου πυρήνα
  - Όλοι οι πίνακες και κλήσεις εμπλέκουν τον πυρήνα
  - Ο πυρήνας αντιμετωπίζει το μπλοκάρισμα
    - Μπορεί να επιλέξει άλλο νήμα της ίδιας διεργασίας
  - Μεγάλο κόστος για τη δημιουργία νέων νημάτων
    - Παράκαμψη με διατήρηση νημάτων σε ανενεργή κατάσταση
  - Πώς λειτουργεί η fork και τα σήματα;
- Υβριδικές υλοποιήσεις
  - Υλοποίηση νημάτων στον πυρήνα
  - Πολλά νήματα χρήστη πάνω σε κάθε νήμα πυρήνα

Ενεργοποιήσεις χρονοπρογραμματιστή



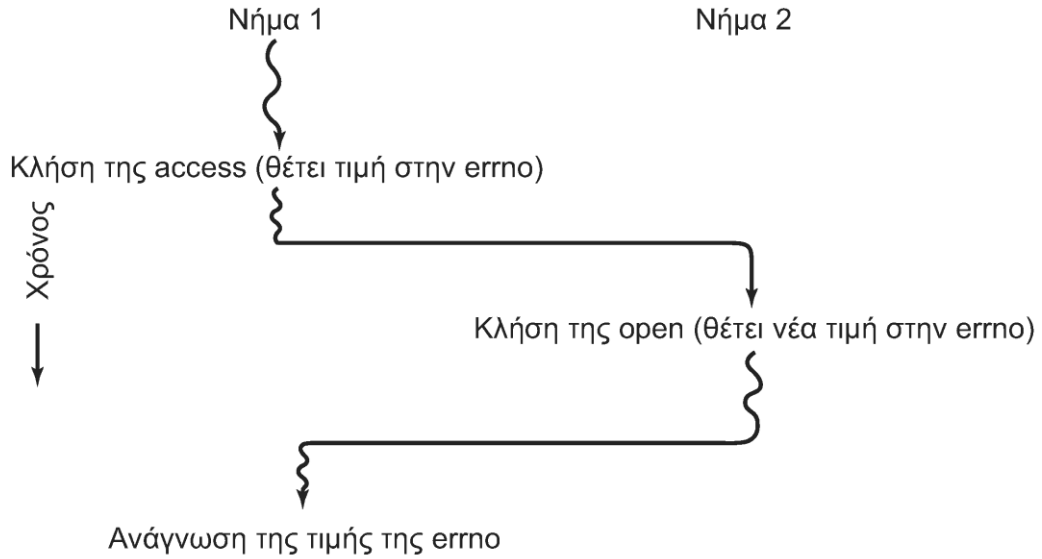
- Κάθε διεργασία έχει μερικά νήματα πυρήνα
  - Όταν κάποιο μπλοκάρεται ο πυρήνας καλεί τον χρήστη
  - Το ίδιο συμβαίνει και όταν ένα νήμα αφυπνίζεται
    - Ουσιαστικά δίνει τον έλεγχο στον χρονοπρογραμματιστή

Αναδυόμενα νήματα



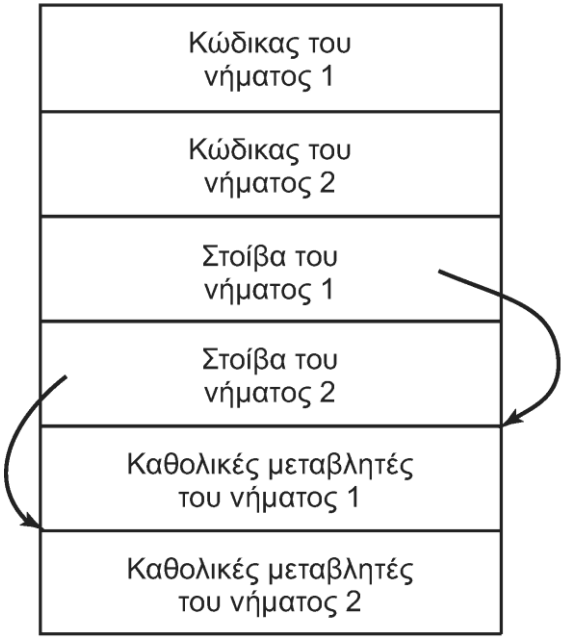
- Αναδυόμενα νήματα
  - Δημιουργούνται όταν έρχεται ένα νέο αίτημα
  - Κατάλληλα για κατανεμημένα και δικτυακά συστήματα

Μετατροπή κώδικα (1 από 3)



- Οι καθολικές μεταβλητές δημιουργούν προβλήματα
  - Παράδειγμα: μεταβλητή errno στο UNIX
  - Μπορεί να διαβάσουμε τιμή από άλλο νήμα
  - Ορισμένες πρέπει να είναι καθολικές σε ένα νήμα!

Μετατροπή κώδικα (2 από 3)



- Ιδιωτικές καθολικές μεταβλητές (ανά νήμα)
  - Δεν υποστηρίζονται από τις γλώσσες προγραμματισμού
  - Υλοποίηση μέσω βιβλιοθήκης νημάτων
  - Ουσιαστικά προσθήκη ενός επιπέδου εμβέλειας

Μετατροπή κώδικα (3 από 3)

- Μη επανεισαγόμενες βιβλιοθήκες
  - Αν κληθούν από δύο νήματα μπορεί να αποτύχουν
  - Συγγραφή νέων βιβλιοθηκών ή χρήση χιτωνίων κλήσεων
- Χειρισμός σημάτων
  - Κάποια σήματα είναι ανά νήμα, κάποια ανά διεργασία
- Διαχείριση στοίβας
  - Πώς χειριζόμαστε τη στοίβα χωρίς γνώση των νημάτων;
- Γενική παρατήρηση: η πολυνημάτωση θέλει δουλειά
  - Τροποποιήσεις σε βιβλιοθήκες και κλήσεις συστήματος

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Διαδιεργασιακή επικοινωνία

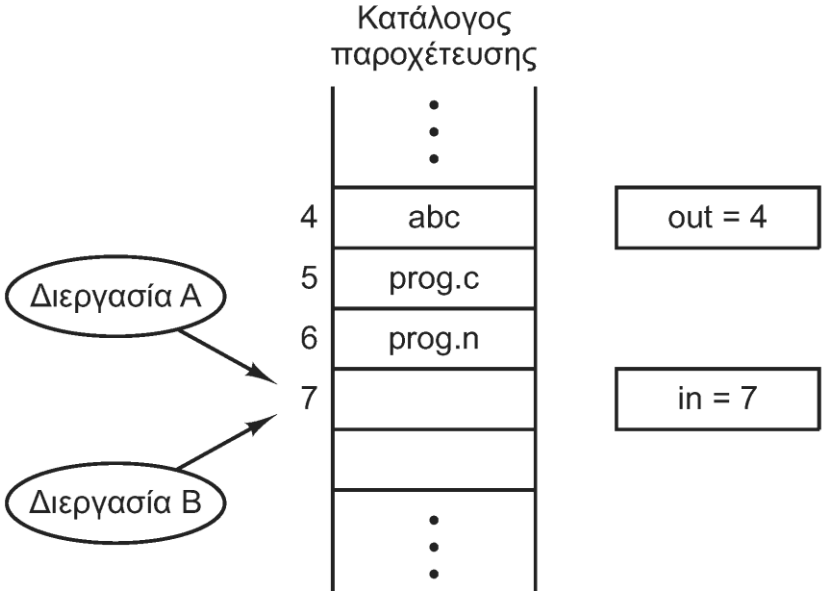
Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Επικοινωνία διεργασιών

- Επικοινωνία διεργασιών
  - Κανονικά οι διεργασίες είναι απομονωμένες
  - Επικοινωνούν μόνο με αυστηρά ελεγχόμενο τρόπο
- Τρεις παραλλαγές επικοινωνίας διεργασιών
  - Μεταβίβαση πληροφοριών μεταξύ τους
    - Παράδειγμα: σωλήνωση στο φλοιό του UNIX
  - Αποφυγή συγκρούσεων για πόρους
    - Παράδειγμα: δέσμευση θέσεων σε σύστημα κρατήσεων
  - Εξαρτήσεις ανάμεσα στις διεργασίες
    - Παράδειγμα: παραγωγή και κατανάλωση δεδομένων

Συνθήκες ανταγωνισμού (1 από 2)



- Συνθήκες ανταγωνισμού (παράδειγμα)
  - Αποστολή αρχείων σε διαχειριστή εκτυπώσεων
  - Αποθήκευση ονομάτων αρχείων προς εκτύπωση σε πίνακα
  - Χρήση δύο μεταβλητών για πρώτο και τελευταίο στοιχείο

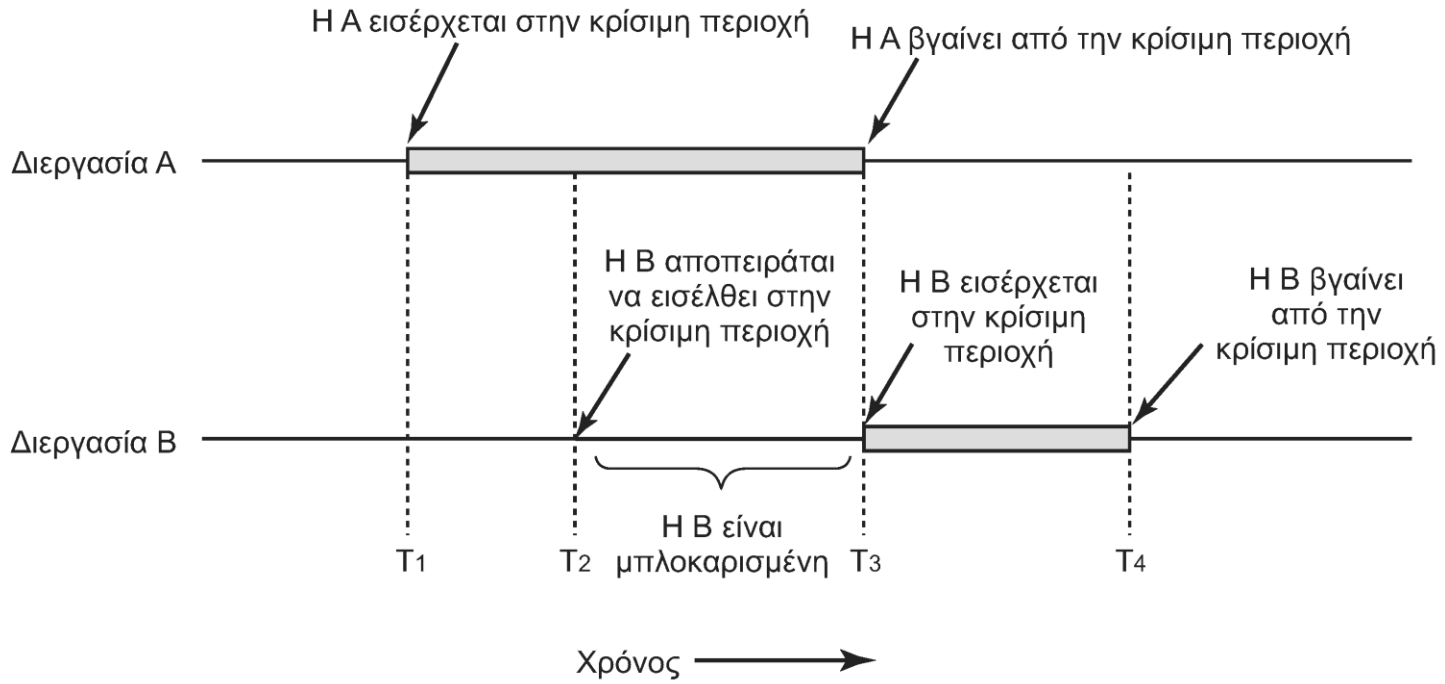
Συνθήκες ανταγωνισμού (2 από 2)

- Έστω ότι οι Α και Β έχουν αρχεία προς εκτύπωση
  - Η Α διαβάζει τη μεταβλητή in=7 και διακόπτεται
  - Η Β διαβάζει και αυτή τη μεταβλητή in=7
  - Η Β αποθηκεύει το αρχείο στη θέση 7
  - Η Β αποθηκεύει την τιμή 8 στην in και διακόπτεται
  - Η Α αποθηκεύει και αυτή το αρχείο στη θέση 7
  - Η Α αποθηκεύει την τιμή 8 στην in
  - Η εκτύπωση της διεργασίας Β έχει εξαφανιστεί!
- Το αποτέλεσμα εξαρτάται από τη σειρά εκτέλεσης
  - Έχουμε συνθήκες ανταγωνισμού (race conditions)

Κρίσιμες περιοχές (1 από 2)

- Αμοιβαίος αποκλεισμός
  - Βασική τεχνική αποφυγής συνθηκών ανταγωνισμού
- Γενική μορφή του αμοιβαίου αποκλεισμού
  - Κάθε διεργασία περιέχει ορισμένες κρίσιμες περιοχές
  - Οι κρίσιμες περιοχές εκτελούνται με αποκλεισμό
  - Γενικότερα έχουμε 4 συνθήκες για αποδοτική λύση
    - Μόνο μία διεργασία μπορεί να είναι στην κρίσιμη περιοχή
    - Δεν βασιζόμαστε σε ταχύτητα εκτέλεσης ή πλήθος ΚΜΕ
    - Διεργασίες εκτός κρίσιμης περιοχής δεν εμποδίζουν τις άλλες
    - Δεν επιτρέπεται οι διεργασίες να περιμένουν επ’ αόριστον

Κρίσιμες περιοχές (2 από 2)



- Παράδειγμα αμοιβαίου αποκλεισμού
  - Η Α εισέρχεται στην κρίσιμη περιοχή
  - Η Β προσπαθεί να εισέλθει αλλά μπλοκάρεται
  - Η Α εξέρχεται και μόνο τότε εισέρχεται η Β

Αναμονή με απασχόληση (1 από 7)

- Αναμονή με απασχόληση
  - Υλοποίηση αμοιβαίου αποκλεισμού με αναμονή
- Απενεργοποίηση διακοπών
  - Απενεργοποίηση κατά την είσοδο στην κρίσιμη περιοχή
  - Ενεργοποίηση κατά την έξοδο από την κρίσιμη περιοχή
  - Είναι αδύνατον να αλλάξει η τρέχουσα διεργασία!
  - Η απενεργοποίηση των διακοπών είναι επικίνδυνη
  - Δεν λειτουργεί σε περιβάλλον πολλών επεξεργαστών
- Μεταβλητές κλειδώματος
  - Η μεταβλητή αλλάζει ανάμεσα σε ανάγνωση και χρήση

Αναμονή με απασχόληση (2 από 7)

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

- Αυστηρή εναλλαγή
  - Χρήση μεταβλητής που δείχνει ποιος έχει σειρά
    - Ονομάζεται και κλείδωμα περιστροφής
    - Η μεταβλητή αλλάζει στο τέλος της κρίσιμης περιοχής
  - Το πρόβλημα είναι ότι παραβιάζονται οι συνθήκες
    - Έστω ότι η διεργασία θέλει να ξαναμπεί στην κρίσιμη περιοχή
    - Πρέπει να περιμένει να μπει πρώτα η άλλη διεργασία

Αναμονή με απασχόληση (3 από 7)

```
#define TRUE    1
#define N       2                       /* το πλήθος των διεργασιών */

int turn;                               /* ποια διεργασία έχει σειρά; */
int interested[N];                      /* όλες οι τιμές αρχικά 0 (FALSE) */

void enter_region(int process);         /* η process παίρνει τις τιμές 0 και 1 */
{
    int other;                          /* ο αριθμός της άλλης διεργασίας */

    other = 1 – process;                /* η άλλη διεργασία */
    interested[process] = TRUE;         /* εκδήλωση ενδιαφέροντος */
    turn = process;                     /* απόδοση τιμής στη σημαία */
    while (turn == process && interested[other] == TRUE) /* εντολή απραξίας */;
}

void leave-region (int process)         /* process: διεργασία που αποχωρεί */
{
    interested[process] = FALSE;        /* γνωστοποίηση εξόδου από κρίσιμη περιοχή */
}
```

Η λύση του Peterson

Αναμονή με απασχόληση (4 από 7)

- Η λύση του Peterson
  - Η διεργασία γράφει στον πίνακα ότι ενδιαφέρεται
  - Επιπλέον θέτει τη μεταβλητή turn
  - Τι γίνεται αν προσπαθήσουν ταυτόχρονα;
    - Και οι δύο εξετάζουν τον πίνακα interested
    - Όποια έγραψε την turn τελευταία θα περιμένει
    - Θα απελευθερωθεί όταν η άλλη βγει από την περιοχή
    - Τότε θα σταματήσει να ενδιαφέρεται

Αναμονή με απασχόληση (5 από 7)

- Η εντολή TSL
  - Απαιτεί μια νέα εντολή υλικού: TSL register, lock
    - Αντιγράφει τη θέση μνήμης lock στον register
    - Αποθηκεύει την τιμή 1 στη θέση μνήμης lock
    - Οι δύο λειτουργίες γίνονται ατομικά
    - Ο δίαυλος κλειδώνεται όσο διαρκεί η εντολή
    - Το κλείδωμα επηρεάζει όλους τους επεξεργαστές
  - Η TSL προσπαθεί να «κλειδώσει» τη θέση lock
    - Αν το πέτυχε, θα το εξετάσει η επόμενη εντολή

Αναμονή με απασχόληση (6 από 7)

```
enter_region:
    TSL REGISTER, LOCK      | Αντιγραφή του κλειδώματος στον καταχωρητή και ανάθεση
                            |    της τιμής 1 στη lock
    CMP REGISTER,#0         | Είχε η lock την τιμή 0;
    JNE enter_region        | Αν η lock είχε μη μηδενική τιμή, της είχε δοθεί μη μηδενική
                            |    τιμή, άρα επανάληψη του βρόχου
    RET                     | Επιστροφή στην καλούσα διεργασία·
                            |    είσοδος στην κρίσιμη περιοχή

leave_region:
    MOVE LOCK,#0            | Αποθήκευση της τιμής 0 στη lock
    RET                     | Επιστροφή στην καλούσα διεργασία
```

- Χρήση της εντολής TSL
  - Για κάθε κρίσιμη περιοχή έχουμε μία θέση μνήμης lock
  - Η TSL διαβάζει και αλλάζει την τιμή της lock
  - Αν η τιμή ήταν 0 προχωράμε, αλλιώς περιμένουμε
  - Στην έξοδο θέτουμε πάλι την τιμή σε 0
  - Το πρόγραμμα μπορεί να διακοπεί μεταξύ δύο εντολών

Αναμονή με απασχόληση (7 από 7)

```
enter_region:
    MOVE REGISTER,#1        | Τιμή 1 στον καταχωρητή
    XCHG REGISTER,LOCK      | Ανταλλαγή περιεχομένων καταχωρητή και μεταβλητής lock
    CMP REGISTER,#0         | Ήταν η lock μηδέν;
    JNE enter_region        | Αν δεν ήταν μηδέν, είχε πάρει μη μηδενική τιμή, άρα βρόχος
    RET                     | Επιστροφή στον καλούντα· είσοδος στην κρίσιμη περιοχή

leave_region:
    MOVE LOCK,#0            | Αποθήκευση της τιμής 0 στη lock
    RET                     | Επιστροφή στην καλούσα διεργασία
```

- Η εντολή XCHG
  - Παραλλαγή της ίδιας ιδέας με την TSL
  - Η XCHG ανταλλάσει τις τιμές των register και lock
    - Η ανταλλαγή γίνεται πάλι ατομικά
  - Πάλι έχουμε δύο πράξεις σε ένα βήμα
    - Ανάγνωση και αλλαγή της θέσης lock
  - Χρησιμοποιείται στους επεξεργαστές x86

Λήθαργος και αφύπνιση (1 από 4)

- Μειονεκτήματα αναμονής με απασχόληση
  - Σπαταλά πόρους του επεξεργαστή κατά την αναμονή
  - Κίνδυνος αντιστροφής προτεραιοτήτων
    - Έστω ότι η Υ έχει υψηλή και η Χ χαμηλή προτεραιότητα
    - Η Χ εκτελείται και μπαίνει στην κρίσιμη περιοχή
    - Εν τω μεταξύ η Υ ξυπνάει και αρχίζει να εκτελείται
    - Η Υ θα περιμένει (για πάντα;) να βγει η Χ από κρίσιμη περιοχή
- Κλήσεις sleep και wakeup
  - Η sleep βάζει την καλούσα διεργασία σε λήθαργο
  - Η wakeup αφυπνίζει μία διεργασία από το λήθαργο

Λήθαργος και αφύπνιση (2 από 4)

- Πρόβλημα παραγωγού-καταναλωτή
  - Δύο διεργασίες μοιράζονται μια περιορισμένη μνήμη
  - Παραγωγός: τοποθετεί αντικείμενα στην περιοχή
    - Δεν μπορεί να προχωρήσει αν δεν υπάρχουν κενές θέσεις
  - Καταναλωτής: αφαιρεί αντικείμενα από την περιοχή
    - Δεν μπορεί να προχωρήσει αν δεν υπάρχουν πλήρεις θέσεις
  - Έστω ότι έχουμε n θέσεις μνήμης και count μη κενές
    - Ο παραγωγός μπλοκάρεται όταν count=n
    - Ο παραγωγός ξυπνά τον καταναλωτή όταν count=1
    - Ο καταναλωτής μπλοκάρεται όταν count=0
    - Ο καταναλωτής ξυπνά τον παραγωγό όταν count=n-1

Λήθαργος και αφύπνιση (3 από 4)

```
#define N 100                                   /* πλήθος θέσεων στην προσωρινή μνήμη */
int count = 0;                                  /* πλήθος στοιχείων στην προσωρινή μνήμη */

void producer(void)
{
    int item;

    while (TRUE) {                              /* επανάληψη επάπειρον */
        item=produce_item();                    /* δημιουργία επόμενου στοιχείου */
        if (count == N) sleep();                /* αν η προσωρινή μνήμη είναι γεμάτη, "ύπνος" */
        insert_item(item);                      /* τοποθέτηση στοιχείου στην προσωρινή μνήμη */
        count = count + 1;                      /* αύξηση μετρητή προσωρινής μνήμης κατά 1 */
        if (count == 1) wakeup(consumer);       /* ήταν η προσωρινή μνήμη άδεια; */
    }
}


void consumer(void)
{
    int item;

    while (TRUE) {                              /* επανάληψη επάπειρον */
        if (count == 0) sleep();                /* αν η προσωρινή μνήμη είναι άδεια, "ύπνος" */
        item = remove_item();                   /* αφαίρεση στοιχείου από προσωρινή μνήμη */
        count = count – 1;                      /* μείωση μετρητή προσωρινής μνήμης κατά 1  */
        if (count == N – 1) wakeup(producer);   /* ήταν η προσωρινή μνήμη γεμάτη; */
        consume_item(item);                     /* εμφάνιση στοιχείου */
    }
}
```

Λήθαργος και αφύπνιση (4 από 4)

- Γιατί η λύση αυτή δεν λειτουργεί;
  - Έστω ότι count=0 (όλες οι θέσεις κενές)
  - Ο καταναλωτής διαβάζει την count αλλά διακόπτεται
  - Ο παραγωγός ξεκινά και τοποθετεί ένα αντικείμενο
  - Ο παραγωγός στέλνει σήμα wakeup στον καταναλωτή
  - Ο καταναλωτής ξεκινά αργότερα και μπλοκάρεται
  - Το σήμα του wakeup του παραγωγού έχει χαθεί!
  - Πρέπει κάπως να θυμόμαστε τα σήματα

Σηματοφόροι (1 από 4)

- Σηματοφόροι
  - Ειδική μορφή ακέραιης μεταβλητής (ιδέα του Dijkstra)
    - Παίρνει θετικές τιμές ή μηδέν
  - Στους σηματοφόρους ορίζονται δύο πράξεις
    - Down: αν η τιμή είναι >0, μειώνεται κατά 1 και συνεχίζουμε
    - Αν όμως είναι 0, τότε η διεργασία μπλοκάρεται
    - Up: αν έχουν μπλοκαριστεί διεργασίες, τότε αφυπνίζεται μία
    - Αλλιώς η τιμή αυξάνεται κατά 1
  - Οι up και down εκτελούνται αδιαίρετα
    - Αλλαγή τιμής, εμποδισμός και αφύπνιση σε ένα βήμα

Σηματοφόροι (2 από 4)

- Υλοποίηση σηματοφόρων
  - Κλήσεις συστήματος που εκτελούνται ατομικά
    - Απενεργοποίηση διακοπών ή εντολές TSL/XCHG
    - Ο εντολές αυτές απαιτούν ελάχιστο χρόνο
- Λύση προβλήματος παραγωγού-καταναλωτή
  - Full: πλήρεις θέσεις, αρχικά 0
  - Empty: κενές θέσεις, αρχικά n
  - Mutex: αμοιβαίος αποκλεισμός, αρχικά 1
    - Δυαδικός σηματοφόρος (0 ή 1)

Σηματοφόροι (3 από 4)

```
#define N 100                   /* αριθμός θέσεων στην προσωρινή μνήμη */
typedef int semaphore;          /* οι σηματοφόροι αποτελούν μια ειδική κατηγορία ακεραίων */
semaphore mutex = 1;            /* ελέγχει την πρόσβαση στην κρίσιμη περιοχή */
semaphore empty = N;            /* πλήθος κενών θέσεων προσωρινής μνήμης */
semaphore full = 0;             /* πλήθος κατειλημμένων θέσεων προσωρινής μνήμης */

void producer(void)
{
    int item;
    while (TRUE) {              /* η σταθερά TRUE ισοδυναμεί με 1 */
        item = produce_item();  /* δημιουργία στοιχείου για προσωρινή μνήμη */
        down(&empty);           /* μείωση μετρητή κενών θέσεων κατά 1 */
        down(&mutex);           /* είσοδος στην κρίσιμη περιοχή */
        insert_item(item);      /* τοποθέτηση νέου στοιχείου στην προσωρινή μνήμη */
        up(&mutex);             /* έξοδος από την κρίσιμη περιοχή */
        up(&full);              /* αύξηση μετρητή κατειλημμένων θέσεων κατά 1*/
    }
}

void consumer(void)
{
    int item;
    while (TRUE) {              /* ατέρμων βρόχος */
        down(&full);            /* μείωση μετρητή κατειλημμένων θέσεων κατά 1 */
        down(&mutex);           /* είσοδος στην κρίσιμη περιοχή */
        item = remove_item();   /* αφαίρεση στοιχείου από προσωρινή μνήμη */
        up(&mutex);             /* έξοδος από την κρίσιμη περιοχή */
        up(&empty);             /* αύξηση μετρητή κενών θέσεων κατά 1 */
        consume_item(item);     /* χρήση του στοιχείου */
    }
}
```

Σηματοφόροι (4 από 4)

- Οι δύο χρήσεις των σηματοφόρων
  - Ο mutex επιτυγχάνει τον αμοιβαίο αποκλεισμό
  - Οι full και empty επιτυγχάνουν το συγχρονισμό
- Απόκρυψη διακοπών
  - Κάθε συσκευή Ε/Ε έχει ένα σηματοφόρο
  - Ο διαχειριστής συσκευής κάνει down
    - Περιμένει να ολοκληρωθεί η λειτουργία Ε/Ε
  - Η διαδικασία εξυπηρέτησης κάνει up
    - Σήμα στον διαχειριστή συσκευής να αφυπνιστεί

Τα mutex (1 από 9)

```
mutex_lock:
    TSL REGISTER,MUTEX    | αντιγραφή mutex στον καταχωρητή και ανάθεση της τιμής 1
    CMP REGISTER,#0       | είχε το mutex την τιμή 0;
    JZE ok                | αν είχε την τιμή 0, το mutex ήταν ξεκλείδωτο, άρα επιστροφή
    CALL thread_yield     | το mutex είναι απασχολημένο, ας χρονοπρογραμματιστεί
                          | κάποιο άλλο νήμα
    JMP mutex_lock        | δοκιμή ξανά αργότερα
ok: RET                   | επιστροφή στο καλούν νήμα, είσοδος στην κρίσιμη περιοχή

mutex_unlock:
    MOVE MUTEX,#0         | αποθήκευση 0 στο mutex
    RET                   | επιστροφή στο καλούν νήμα
```

- Τα mutex
  - Απλή μορφή δυαδικού σηματοφορέα
  - Mutex_lock: κλειδώνει το mutex
    - Αν είναι κλειδωμένο, η διεργασία μπλοκάρεται προσωρινά
    - Όταν αφυπνιστεί δοκιμάζει ξανά (και μπλοκάρεται ξανά)
  - Mutex_unlock: ξεκλειδώνει το mutex

Τα mutex (2 από 9)

- Διαφορά mutex από άλλες λύσεις
  - Το mutex υλοποιείται στο χώρο του χρήστη
  - Στο σηματοφόρο έχουμε ουρά διεργασιών
    - Στο mutex τα νήματα μπλοκάρονται προσωρινά
    - Δοκιμάζουν ξανά όταν έρθει η σειρά τους
  - Στην αναμονή με απασχόληση σπαταλάμε πόρους
    - Στο mutex παραχωρείται ο επεξεργαστής
    - Έχει νόημα σε συνεργατικό περιβάλλουν (νήματα!)

Τα mutex (3 από 9)

- Πώς γίνεται η πρόσβαση στις κοινές δομές;
  - Όταν έχουμε νήματα, όλα βλέπουν την ίδια μνήμη
    - Άρα και τα ίδια mutex ή άλλες μεταβλητές
    - Τα mutex προστατεύουν τις κοινές δομές
    - Αν δεν είναι διαθέσιμες, παραχώρηση επεξεργαστή
  - Όταν έχουμε διεργασίες, έχουν διαφορετική μνήμη
    - Προσπέλαση κοινών δομών μέσω κλήσεων συστήματος
    - Προσπέλαση κοινών δομών μέσω κοινής μνήμης
    - Προσπέλαση κοινών δομών μέσω κοινών αρχείων

Τα mutex (4 από 9)

Κλήση νήματος	**Περιγραφή**
Pthread_mutex_init	Δημιουργία ενός mutex.
Pthread_mutex_destroy	Καταστροφή ενός υπάρχοντος mutex.
Pthread_mutex_lock	Απόκτηση κλειδώματος ή μπλοκάρισμα.
Pthread_mutex_trylock	Απόκτηση κλειδώματος ή αποτυχία.
Pthread_mutex_unlock	Απελευθέρωση κλειδώματος.

Τα mutex (5 από 9)

- Mutex στο πακέτο νημάτων pthreads
  - Δημιουργία και καταστροφή mutex
  - Κλείδωμα (με μπλοκάρισμα ή αποτυχία)
  - Ξεκλείδωμα mutex
- Μεταβλητές συνθήκης
  - Τα mutex δεν επαρκούν για πιο σύνθετες συνθήκες
  - Παράδειγμα: πρόβλημα παραγωγού καταναλωτή
    - Το mutex μπορεί μόνο να κλειδώσει την κοινή μνήμη
    - Δεν αρκεί για ειδικότερες συνθήκες μπλοκαρίσματος

Τα mutex (6 από 9)

Κλήση νήματος	**Περιγραφή**
Pthread_cond_init	Δημιουργία μίας μεταβλητής συνθήκης.
Pthread_cond_destroy	Καταστροφή μίας μεταβλητής συνθήκης.
Pthread_cond_wait	Μπλοκάρισμα και αναμονή σήματος.
Pthread_cond_signal	Σήμα σε άλλο νήμα και αφύπνισή του.
Pthread_cond_broadcast	Σήμα σε πολλά νήματα και αφύπνιση όλων.

Τα mutex (7 από 9)

- Μεταβλητές συνθήκης
  - Μπλοκάρισμα σε συνθήκη μέχρι να εμφανιστεί σήμα
    - Σήμα αφύπνισης για ένα ή για όλα τα νήματα
  - Η μεταβλητή χρησιμοποιείται σε κρίσιμη περιοχή
    - Η διεργασία πρέπει πρώτα να κλειδώσει το mutex
    - Αν μπλοκαριστεί η διεργασία, το mutex ξεκλειδώνεται
    - Μια άλλη διεργασία μπορεί να κλειδώσει τώρα το mutex
    - Όταν αφυπνιστεί το νήμα ελέγχει πάλι τη συνθήκη

Τα mutex (8 από 9)

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000                  /* πλήθος αριθμών που θα παραχθούν */
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0;                         /* προσ. μνήμη μεταξύ παραγωγού και καταναλωτή */

void *producer(void *ptr)               /* παραγωγή δεδομένων */
{    int i;

     for (i= 1; i <= MAX; i++) {
          pthread_mutex_lock(&the_mutex);      /* αποκλ. πρόσβαση στην προσωρινή μνήμη */
          while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
          buffer = i;                          /* τοποθέτηση στοιχείου σε προσωρινή μνήμη  */
          pthread_cond_signal(&condc);         /* αφύπνιση καταναλωτή */
          pthread_mutex_unlock(&the_mutex);    /* πρόσβαση στην προσωρινή μνήμη */
     }
     pthread_exit(0);
}
```

Τα mutex (9 από 9)

```
void *consumer(void *ptr)              /* κατανάλωση δεδομένων*/
{    int i;
     for (i = 1; i <= MAX; i++) {
          pthread_mutex_lock(&the_mutex);      /* αποκλ. πρόσβαση στην προσωρινή μνήμη */
          while (buffer ==0 ) pthread_cond_wait(&condc, &the_mutex);
          buffer = 0;                          /* αφαίρεση από την προσωρινή μνήμη */
          pthread_cond_signal(&condp);         /* αφύπνιση παραγωγού */
          pthread_mutex_unlock(&the_mutex);    /* πρόσβαση στην προσωρινή μνήμη */
     }
     pthread_exit(0);
}

int main(int argc, char **argv)
{
     pthread_t pro, con;
     pthread_mutex_init(&the_mutex, 0);
     pthread_cond_init(&condc, 0);
     pthread_cond_init(&condp, 0);
     pthread_create(&con, 0, consumer, 0);
     pthread_create(&pro, 0, producer, 0);
     pthread_join(pro, 0);
     pthread_join(con, 0);
     pthread_cond_destroy(&condc);
     pthread_cond_destroy(&condp);
     pthread_mutex_destroy(&the_mutex);
}
```

Ελεγκτές (1 από 6)

```
monitor example
        integer i;
        condition c;

        procedure producer( );
        .
        .
        .
        end;

        procedure consumer( );
        .       .       .
        end;
end monitor;
```

- Ελεγκτές
  - Η χρήση των σηματοφόρων και των mutex θέλει προσοχή
    - Λάθος σειρά στις κλήσεις οδηγεί σε αδιέξοδο!
  - Ο ελεγκτής είναι μια αφαίρεση υψηλότερου επιπέδου
    - Είναι δομή της γλώσσας προγραμματισμού, όχι του λειτουργικού

Ελεγκτές (2 από 6)

- Ελεγκτές
  - Ο ελεγκτής αποτελείται από δεδομένα και διαδικασίες
    - Τα δεδομένα προσπελαύνονται μόνο από αυτές τις διαδικασίες
    - Μόνο μία διεργασία μπορεί να είναι ενεργή στον ελεγκτή
  - Υλοποίηση ανάλογα με τον μεταγλωττιστή
    - Παράδειγμα: με σηματοφορείς ή mutex
  - Χρήση μεταβλητών συνθήκης
    - Εντολές signal και wait για χρήση μέσα στον ελεγκτή
    - Τι γίνεται μετά από ένα signal; Ποιος εκτελείται;
      - Λύση Hoare: συνεχίζει η διεργασία που αφυπνίσθηκε
      - Λύση Brinch-Hansen: το signal είναι πάντα η τελευταία εντολή

Ελεγκτές (3 από 6)

```
monitor ProducerConsumer
        condition full, empty;
        integer count;

        procedure insert(item: integer);
        begin
                if count = N then wait(full);
                insert_item(item);
                count := count + 1;
                if count = 1 then signal(empty)
        end;

        function remove: integer;
        begin
                if count = 0 then wait(empty);
                remove = remove_item;
                count := count – 1;
                if count = N – 1 then signal(full)
        end;

        count := 0;
end monitor;
```

```
procedure producer;
begin
        while true do
        begin
                item = produce_item;
                ProducerConsumer.insert(item)
        end
end;

procedure consumer;
begin
        while true do
        begin
                item = ProducerConsumer.remove;
                consume_item(item)
        end
end;
```

Ελεγκτές (4 από 6)

```
public class ProducerConsumer {
    static final int N = 100;                    // σταθερά που δίνει το μέγεθος της προσ. μνήμης
    static producer p = new producer();          // δημιουργία νέου νήματος παραγωγού
    static consumer c = new consumer();          // δημιουργία νέου νήματος καταναλωτή
    static our_monitor mon = new our_monitor();  // δημιουργία νέου ελεγκτή
    public static void main(String args[]) {
        p.start();                               // εκκίνηση νήματος παραγωγού
        c.start();                               // εκκίνηση νήματος καταναλωτή
    }

    static class producer extends Thread {
        public void run() {                      // η μέθοδος run περιέχει τον κώδικα του νήματος
                int item;
                while (true) {                   // ο βρόχος του παραγωγού
                        item = produce_item();
                        mon.insert(item);
                }
        }
        private int produce_item(){ ... }        // παράγει στοιχεία
    }
```

- Παράδειγμα σε Java
  - Η Java υλοποιεί ελεγκτές με μεθόδους synchronized
    - Μία τάξη μπορεί να ορίζει μεθόδους synchronized
    - Όλες οι μέθοδοι αυτές εκτελούνται με αμοιβαίο αποκλεισμό

Ελεγκτές (5 από 6)

```
static class consumer extends Thread {
    public void run() {                          // η μέθοδος run περιέχει τον κώδικα του νήματος
        int item;
        while (true) {                           // ο βρόχος του καταναλωτή
            item = mon.remove();
            consume_item(item);
        }
    }
    private void consume_item(int item) { ... }  // καταναλώνει στοιχεία
}

static class our_monitor {                       // αυτός είναι ο ελεγκτής
    private int buffer[] = new int[N];
    private int count = 0, lo = 0, hi = 0;       // μετρητές και δείκτες πινάκων
    public synchronized void insert(int val) {
        if (count == N) go_to_sleep();           // αν προσ. μνήμη γεμάτη, λήθαργος
        buffer[hi] = val;                        // προσθήκη στοιχείου στην προσ. μνήμη
        hi = (hi + 1) % N;                       // θέση τοποθέτησης επόμενου στοιχείου
        count = count + 1;                       // η προσ. μνήμη έχει τώρα ένα ακόμα στοιχείο
        if (count == 1) notify();                // αν καταναλωτής σε λήθαργο, αφύπνιση
    }
```

- Παραγωγός-καταναλωτής: νήματα της Java
  - Δεν ασχολούνται με αμοιβαίο αποκλεισμό
- Ελεγκτής: τάξη της Java
  - Περιέχει μεθόδους synchronized

Ελεγκτές (6 από 6)

```
        public synchronized int remove() {
                int val;
                if (count == 0) go_to_sleep();          // αν προσ. μνήμη άδεια, λήθαργος
                val = buffer [lo];                      // αφαίρεση στοιχείου από προσ. μνήμη
                lo = (lo + 1) % N;                      // θέση τοποθέτησης επόμενου στοιχείου
                count = count – 1;                      // η προσ. μνήμη έχει τώρα ένα στοιχείο λιγότερο
                if (count == N – 1) notify();           // αν παραγωγός σε λήθαργο, αφύπνιση
                return val;
        }
        private void go_to_sleep() { try{wait(); } catch(InterruptedException exc) {};}
    }
}
```

- Ελεγκτής σε Java
  - Οι διαδικασίες παραγωγού/ καταναλωτή είναι synchronized
  - Δεν υπάρχουν διακριτές μεταβλητές συνθήκης
    - Υπάρχει μία μόνο ανώνυμη συνθήκη για όλο τον ελεγκτή
    - Η εντολή wait() οδηγεί σε μπλοκάρισμα
    - Πρέπει να εκτελείται σε βρόχο γιατί δεν ξέρουμε γιατί ξυπνήσαμε!
    - Η εντολή notify() ξυπνάει ένα άλλο νήμα (τυχαία)

Μεταβίβαση μηνυμάτων (1 από 3)

- Μεταβίβαση μηνυμάτων
  - Οι σηματοφόροι είναι τεχνική χαμηλού επιπέδου
    - Μπορεί όμως να χρησιμοποιηθεί μέσω βιβλιοθήκης
  - Οι ελεγκτές είναι τεχνική υψηλού επιπέδου
    - Απαιτεί όμως υποστήριξη από γλώσσα προγραμματισμού
  - Η μεταβίβαση μηνυμάτων είναι μια πιο γενική τεχνική
    - Και η μόνη κατάλληλη για κατανεμημένα συστήματα
  - Η send(dest, &msg) στέλνει το msg στον dest
  - Η receive(src, &msg) παραλαμβάνει το msg από τον src
    - Μπορεί να προκαλεί μπλοκάρισμα ή να επιστρέφει αμέσως

Μεταβίβαση μηνυμάτων (2 από 3)

```
#define N 100                                   /* πλήθος θέσεων στην προσωρινή μνήμη */

void producer(void)
{
    int item;
    message m;                                  /* προσωρινή μνήμη ενός μηνύματος */

    while (TRUE) {
        item = produce_item();                  /* δημιουργία νέου στοιχείου για την προσ. μνήμη */
        receive(consumer, &m);                  /* αναμονή έλευσης κάποιου νέου μηνύματος */
        build_message(&m, item);                /* κατασκευή μηνύματος που θα σταλεί */
        send(consumer, &m);                     /* αποστολή μηνύματος στον καταναλωτή */
    }
}
```

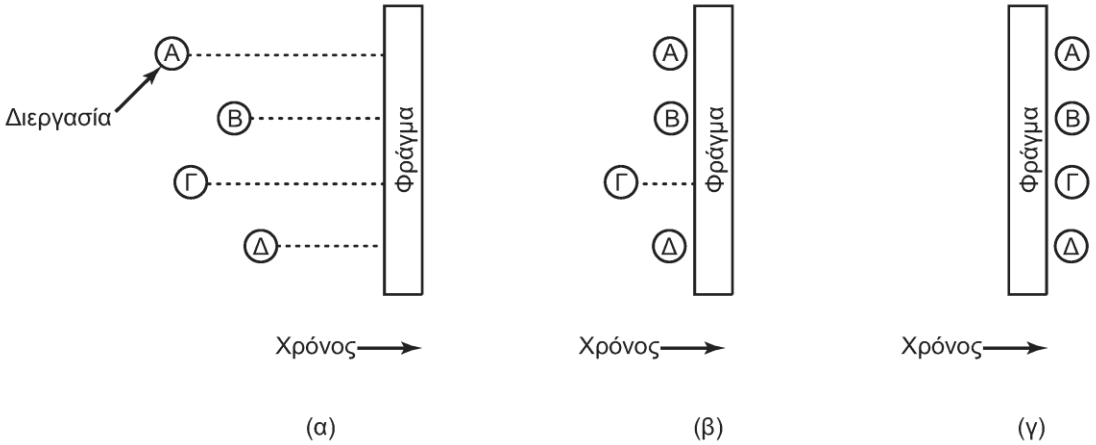
- Παραγωγός-καταναλωτής με μεταβίβαση μηνυμάτων
  - Ο παραγωγός περιμένει κενά μηνύματα
  - Τα γεμίζει και τα στέλνει στον καταναλωτή
  - Ο καταναλωτής στέλνει αρχικά N κενά μηνύματα
  - Μετά περιμένει γεμάτα μηνύματα και τα καταναλώνει

Μεταβίβαση μηνυμάτων (3 από 3)

```
void consumer(void)
{
    int item, i;
    message m;

    for (i = 0; i < N; i++) send(producer, &m);   /* αποστολή N κενών μηνυμάτων */
    while (TRUE) {
        receive(producer, &m);          /* παραλαβή μηνύματος που περιέχει ένα στοιχείο */
        item = extract_item(&m);        /* απόσπαση στοιχείου από το μήνυμα */
        send(producer, &m);             /* επιστροφή κενού μηνύματος */
        consume_item(item);             /* χρήση του στοιχείου */
    }
}
```

- Παραλλαγές μεταβίβασης μηνυμάτων
  - Χρήση γραμματοκιβωτίων επικοινωνίας
    - Παρεμβάλλονται ανάμεσα σε αποστολείς και παραλήπτες
    - Έχουν περιορισμένο χώρο αποθήκευσης μηνυμάτων
  - Χρήση ραντεβού: ο αποστολέας μπλοκάρεται
    - Δεν χρειάζεται προσωρινή μνήμη, τα δύο άκρα συγχρονίζονται

Φράγματα



- Φράγματα
  - Οι διεργασίες διαιρούνται σε φάσεις
    - Στο τέλος κάθε φάσης εκτελούμε μια κλήση barrier
    - Οι διεργασίες μπλοκάρονται μέχρι να φτάσουν όλες εκεί
    - Μόλις συμβεί αυτό, απελευθερώνονται όλες
  - Επιτρέπει συγχρονισμό παράλληλων υπολογισμών

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Χρονοπρογραμματισμός

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

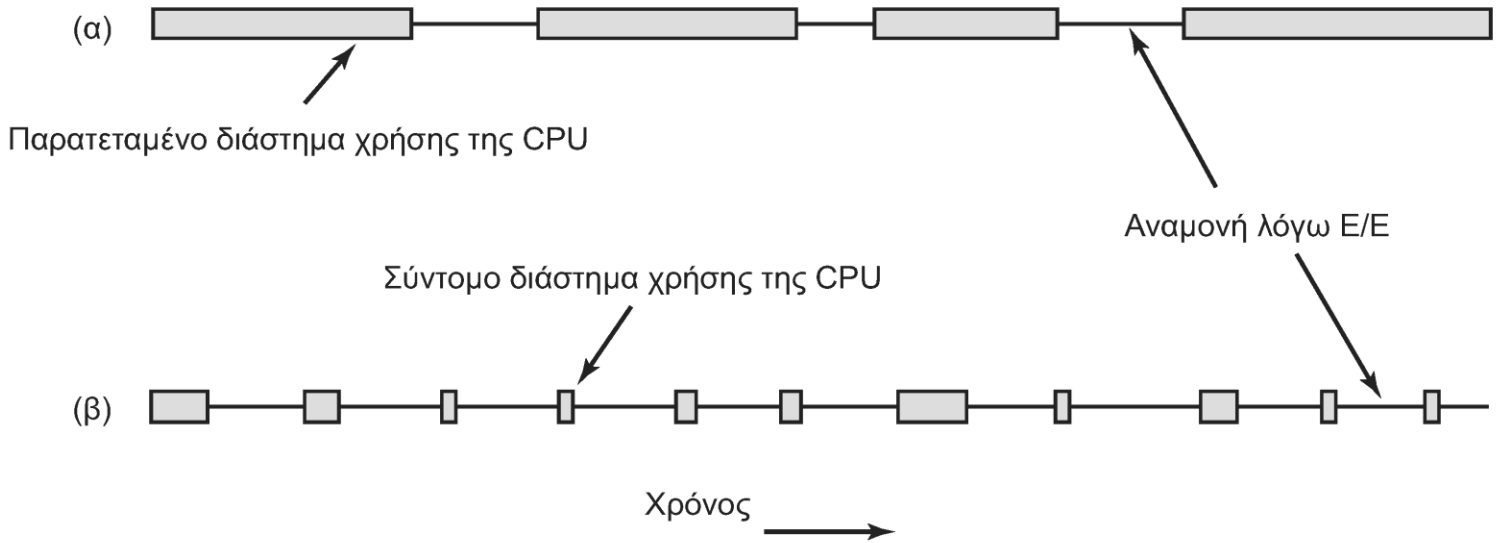
Χρονοπρογραμματισμός (1 από 6)

- Έννοια του χρονοπρογραμματισμού
  - Ποια διεργασία θα εκτελεστεί μετά από κάποια διακοπή;
- Ιστορική εξέλιξη χρονοπρογραμματισμού
  - Στα συστήματα δεσμίδων εκτελούσαμε επόμενη εργασία
  - Στα συστήματα πολυπρογραμματισμού υπάρχει επιλογή
    - Και στα σύγχρονα μικτά συστήματα υπάρχει ζήτημα επιλογής
  - Οι προσωπικοί υπολογιστές δεν δίνουν μεγάλη έμφαση
    - Η ταχύτητα των επεξεργαστών είναι πολύ μεγάλη
    - Ελάχιστες διεργασίες είναι πραγματικά χρονοβόρες
  - Οι εξυπηρετητές προσέχουν τον χρονοπρογραμματισμό

Χρονοπρογραμματισμός (2 από 6)

- Εναλλαγή διεργασιών
  - Η αλλαγή της εκτελούμενης διεργασίας έχει κόστος
    - Αποθήκευση κατάστασης προηγούμενης διεργασίας
    - Φόρτωση κατάστασης νέας διεργασίας
    - Αλλαγή χώρου διευθύνσεων και ακύρωση κρυφής μνήμης
- Συμπεριφορά των διεργασιών
  - Οι διεργασίες εναλλάσσουν Ε/Ε και επεξεργασία
  - Οι διεργασίες χωρίζονται σε δύο κατηγορίες
    - Εξαρτημένες από την ΚΜΕ
    - Εξαρτημένες από είσοδο/έξοδο

Χρονοπρογραμματισμός (3 από 6)



- Συμπεριφορά των διεργασιών
  - Ο σημαντικός παράγοντας είναι η διάρκεια επεξεργασίας
  - Οι επεξεργαστές βελτιώνονται πιο γρήγορα από δίσκους
    - Οι περισσότερες διεργασίας εξαρτώνται από είσοδο/έξοδο
    - Πρέπει να εκτελούνται έτσι ώστε να ξεκινάνε νέα είσοδο/έξοδο

Χρονοπρογραμματισμός (4 από 6)

- Πότε γίνεται ο χρονοπρογραμματισμός;
  - Όταν δημιουργείται / τερματίζεται μια διεργασία
  - Όταν μια διεργασία μπλοκάρεται για κάποιο λόγο
  - Όταν προκύπτει μία διακοπή εισόδου/εξόδου
  - Όταν προκύπτει διακοπή χρονομέτρου (ή n διακοπές)
- Μη προεκτοπιστικοί αλγόριθμοι
  - Η τρέχουσα διεργασία εκτελείται μέχρι να μπλοκαριστεί
- Προεκτοπιστικοί αλγόριθμοι
  - Η τρέχουσα διεργασία εκτελείται μέχρι κάποιο όριο

Χρονοπρογραμματισμός (5 από 6)

- Κατηγορίες αλγορίθμων χρονοπρογραμματισμού
  - Συστήματα δέσμης
    - Μη προεκτοπιστική ή προεκτοπιστική με μεγάλο διάστημα
    - Λιγότερες εναλλαγές διεργασιών για αύξηση επίδοσης
  - Συστήματα αλληλεπίδρασης
    - Υποχρεωτικά προεκτοπιστική για αποφυγή καθυστερήσεων
    - Όλες οι διεργασίες πρέπει να εκτελούνται από λίγο
  - Συστήματα πραγματικού χρόνου
    - Δεν είναι απαραίτητη η προεκτοπιστική εκτέλεση
    - Εκτελούνται μόνο συγκεκριμένες διεργασίες στο σύστημα

Χρονοπρογραμματισμός (6 από 6)

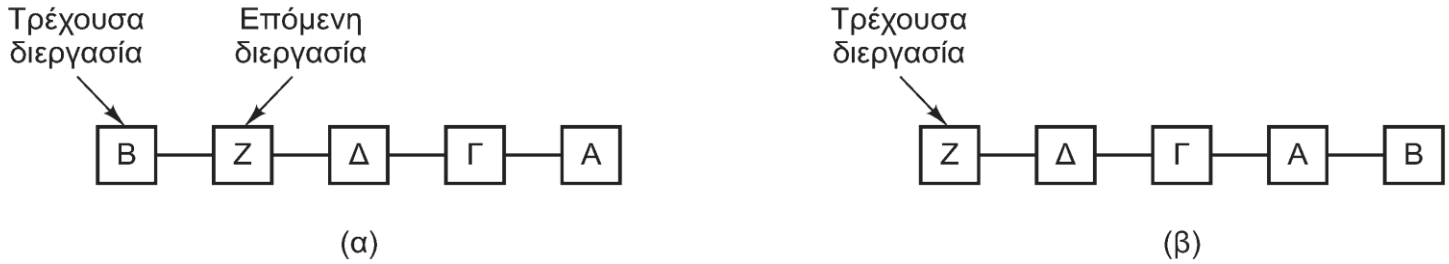
- Στόχοι αλγορίθμων χρονοπρογραμματισμού
  - Δίκαιη κατανομή ΚΜΕ, επιβολή πολιτικής, ισορροπία
  - Συστήματα δέσμης
    - Μεγάλη διεκπεραιωτική ικανότητα και μικρός μέσος χρόνος
    - Υψηλή αξιοποίηση επεξεργαστή
  - Συστήματα αλληλεπίδρασης
    - Χαμηλός χρόνος απόκρισης σε αιτήσεις
    - Τήρηση αναλογιών (οι «απλές» εξυπηρετούνται γρήγορα)
  - Συστήματα πραγματικού χρόνου
    - Τήρηση των προθεσμιών των διεργασιών
    - Προβλεψιμότητα και ομαλότητα χρονοπρογραμματισμού

Συστήματα δέσμης

- Εξυπηρέτηση με βάση τη σειρά άφιξης
  - Οι νέες διεργασίες μπαίνουν στο τέλος μιας ουράς
  - Οι διεργασίες εκτελούνται με τη σειρά της ουράς
  - Οι μπλοκαρισμένες διεργασίες βγαίνουν από την ουρά
    - Όταν απελευθερωθούν μπαίνουν στο τέλος της ουράς
- Εξυπηρέτηση με βάση τη μικρότερη διάρκεια
  - Όλες οι διεργασίες πρέπει να είναι διαθέσιμες μαζί
  - Εκτελούμε πάντα την μικρότερη διεργασία που απομένει
- Εξυπηρέτηση με βάση το μικρότερο υπόλοιπο
  - Κατάλληλη για προεκτοπιστικό χρονοπρογραμματισμό

Συστήματα αλληλεπίδρασης (1 από 6)

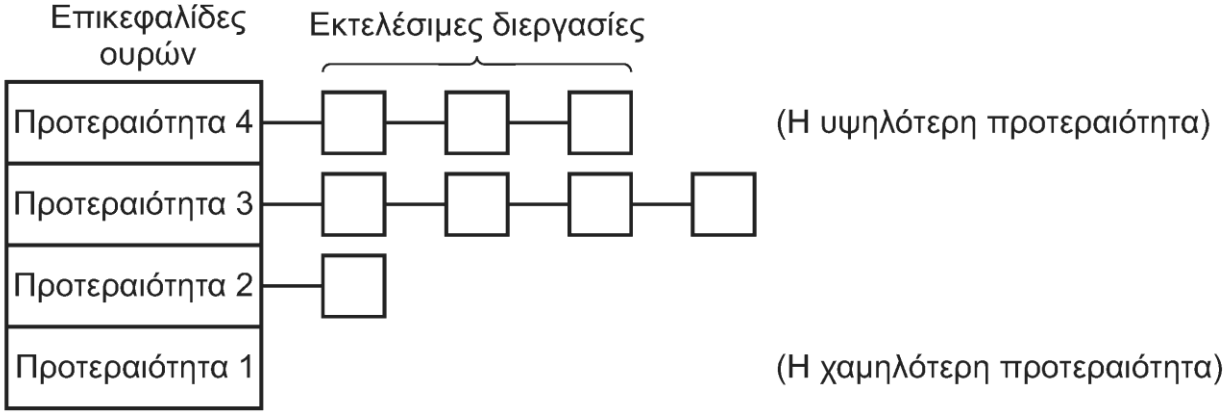
- Χρονοπρογραμματισμός αλληλεπίδρασης
  - Προγραμματισμός εκ περιτροπής
    - Σε κάθε διεργασία εκχωρείται χρονικό διάστημα εκτέλεσης
    - Αν εξαντληθεί το διάστημα αυτό, η διεργασία αλλάζει
    - Η διεργασία αλλάζει κι αν μπλοκαριστεί για κάποιο λόγο
    - Υλοποιείται με κυκλική λίστα



Συστήματα αλληλεπίδρασης (2 από 6)

- Καθορισμός διαστήματος εκτέλεσης (κβάντου)
  - Αρκετά μεγαλύτερο από το χρόνο εναλλαγής διεργασιών
    - Αλλιώς μεγάλο ποσοστό του χρόνου σπαταλιέται σε εναλλαγές
  - Αρκετά μικρό ώστε να έχουμε χαμηλό χρόνο απόκρισης
    - Αλλιώς χρειάζεται πολύς χρόνος για κάθε κύκλο στην ουρά
  - Λίγο μεγαλύτερο από το μέσο χρόνο εκτέλεσης
    - Το διάστημα ανάμεσα σε δύο λειτουργίες εισόδου/εξόδου
- Χρονοπρογραμματισμός με βάση την προτεραιότητα
  - Εκτελείται η έτοιμη με υψηλότερη προτεραιότητα

Συστήματα αλληλεπίδρασης (3 από 6)



- Ομαδοποίηση σε κλάσεις προτεραιοτήτων
  - Εκτέλεση εκ περιτροπής μέσα σε κάθε κλάση
- Ανάθεση προτεραιοτήτων
  - Στατική: ανάλογα με τη σημασία της διεργασίας
    - Ορίζονται προκαταβολικά με εξωτερικά κριτήρια
  - Δυναμική: ανάλογα με τον τρόπο εκτέλεσης
    - Αύξηση προτεραιότητας στις διεργασίες εισόδου/εξόδου

Συστήματα αλληλεπίδρασης (4 από 6)

- Συστήματα πολλαπλών ουρών
  - Οι σταθερές προτεραιότητες έχουν κίνδυνο υποσιτισμού
  - Σταδιακή ελάττωση προτεραιότητας εκτελούμενης
    - Επιτρέπει σε όλες τις διεργασίες να εκτελούνται κάποτε
  - Ανάθεση μεγαλύτερου κβάντου σε μικρές προτεραιότητες
    - Για παράδειγμα, 1 κβάντο στην υψηλή και 2 κβάντα στη χαμηλή
    - Όταν μια διεργασία εξαντλεί το κβάντο, υποβιβάζεται
  - Ανάθεση προτεραιότητας ανάλογα με τη συμπεριφορά
    - Υψηλότερη προτεραιότητα αν χρησιμοποιεί τερματικό
    - Χαμηλότερη προτεραιότητα αν εξαντλεί το κβάντο

Συστήματα αλληλεπίδρασης (5 από 6)

- Εξυπηρέτηση με βάση τη μικρότερη διάρκεια
  - Βασίζεται στον αντίστοιχο αλγόριθμο συστημάτων δέσμης
  - Θεωρούμε κάθε διάστημα εκτέλεσης ως νέα εργασία
  - Πρέπει να μαντέψουμε πόσο θα εκτελεστεί η διεργασία
  - Παράδειγμα: στάθμιση προηγούμενων χρόνων εκτέλεσης
    - $T_n=T_{n-1}/2+T_{n-2}/4+T_{n-3}/8+\ldots$
- Εγγυημένος χρονοπρογραμματισμός
  - Αν έχουμε n χρήστες, ο καθένας παίρνει το 1/n της ΚΜΕ
    - Υπολογίζεται ο λόγος χρόνου εκτέλεσης προς αναλογούντα
    - Εκτελείται η διεργασία με τον μικρότερο τέτοιο λόγο

Συστήματα αλληλεπίδρασης (6 από 6)

- Χρονοπρογραμματισμός με λοταρία
  - Κάθε διεργασία έχει ορισμένους λαχνούς
  - Διαφορετικοί λαχνοί για κάθε πόρο
  - Για κάθε πόρο γίνεται κλήρωση λαχνών
  - Οι πιο σημαντικές διεργασίες έχουν περισσότερους
    - Ο πελάτης μπορεί να παραχωρήσει λαχνούς στον εξυπηρετητή
- Χρονοπρογραμματισμός δίκαιης διανομής
  - Οι πόροι ανατίθενται ανά χρήστη
  - Οι διεργασίες αξιοποιούν μόνο τους πόρους του χρήστη

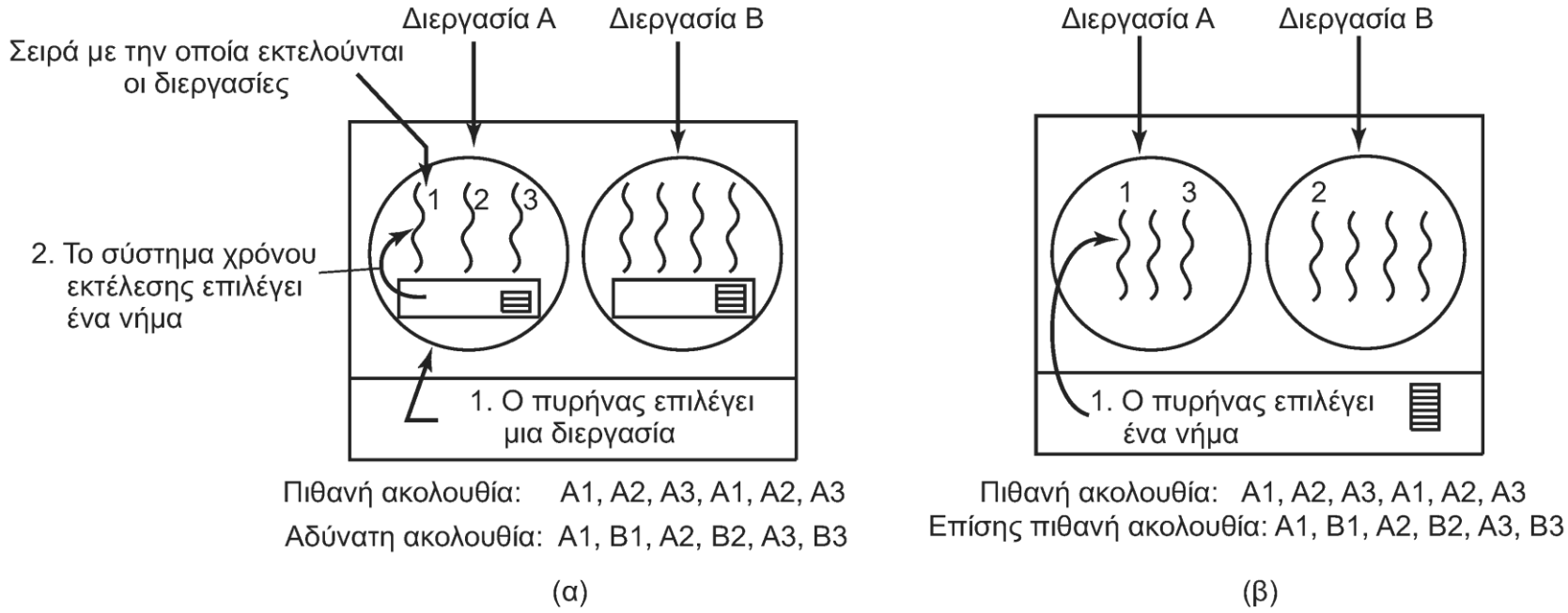
Συστήματα πραγματικού χρόνου

- Χρονοπρογραμματισμός πραγματικού χρόνου
  - Οι διεργασίες πρέπει να εκτελούνται μέσα σε προθεσμίες
  - Περιοδικά: τα ίδια γεγονότα εμφανίζονται περιοδικά
  - Απεριοδικά: τα γεγονότα προκύπτουν απρόβλεπτα
  - Έστω ένα περιοδικό σύστημα
    - Το συμβάν i εμφανίζεται κάθε P_i και απαιτεί χρόνο C_i
    - Το σύστημα είναι χρονοπρογραμματίσιμο αν $\Sigma C_i/P_i <= 1$
    - Αν είναι παραπάνω, είναι αδύνατον να χρονοπρογραμματιστεί
  - Στατικοί αλγόριθμοι: οι αποφάσεις λαμβάνονται αρχικά
  - Δυναμικοί αλγόριθμοι: λήψη αποφάσεων δυναμικά

Πολιτικές και μηχανισμοί

- Πολιτικές και μηχανισμοί χρονοπρογραμματισμού
  - Βασικός παραμετροποιήσιμος μηχανισμός
  - Παράμετροι από το χειριστή ή / και τον χρήστη
  - Ο μηχανισμός υλοποιεί μια επιθυμητή πολιτική
- Παράδειγμα: συστήματα πολλών ουρών
  - Ο μηχανισμός διαλέγει την επόμενη διεργασία
    - Η πρώτη της ανώτερης ουράς
  - Η πολιτική κατατάσσει τις διεργασίες
    - Ανάθεση σε ουρές και κβάντα

Χρονοπρογραμματισμός νημάτων (1 από 2)



- Χρονοπρογραμματισμός νημάτων
  - Σε επίπεδο χρήστη περιορίζονται στο κβάντο διεργασίας
    - Ο ακριβής αλγόριθμος είναι θέμα του πακέτου χρήστη
  - Στο επίπεδο πυρήνα μπορούν να αγνοούνται οι διεργασίες
    - Προκύπτουν διαφορετικές ακολουθίες εκτέλεσης

Χρονοπρογραμματισμός νημάτων (2 από 2)

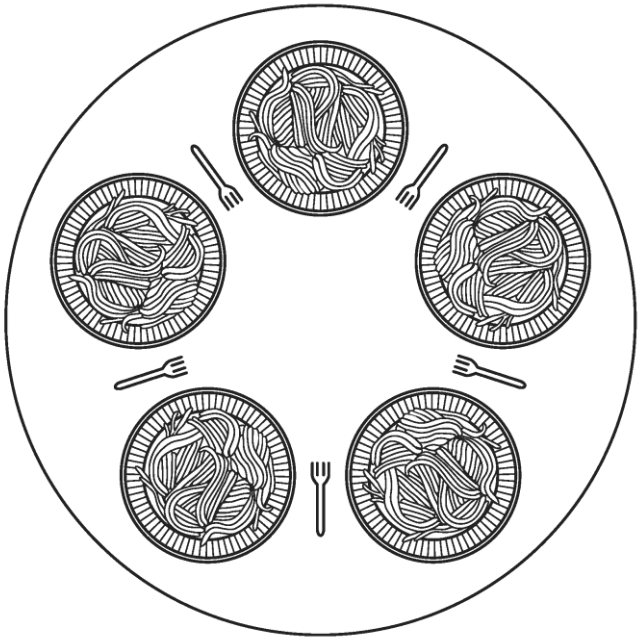
- Νήματα επιπέδου χρήστη
  - Η εναλλαγή έχει γενικά πολύ χαμηλό κόστος
  - Συνήθως εκτέλεση εκ περιτροπής ή με προτεραιότητες
  - Μπορεί να εξαρτάται από την εφαρμογή
  - Παράδειγμα: προτεραιότητες με βάση τις εξαρτήσεις
- Νήματα επιπέδου πυρήνα
  - Ο χρονοπρογραμματιστής είναι πάντα ο ίδιος
  - Η εναλλαγή έχει μεγάλο κόστος
  - Συμφέρει να επιλέξουμε νήματα της ίδιας διεργασίας
    - Αποφεύγουμε την αλλαγή χάρτη μνήμης

Κλασικά προβλήματα

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Το δείπνο των φιλοσόφων (1 από 4)



- Το πρόβλημα του δείπνου των φιλοσόφων
  - Έχουμε η πιάτα και η πιρούνια σε ένα τραπέζι
  - Για να φάει κάποιος χρειάζεται τα δύο γειτονικά πιρούνια
    - Ουσιαστικά χρειαζόμαστε κλείδωμα χωρίς αδιέξοδα

Το δείπνο των φιλοσόφων (2 από 4)

```
#define N 5                     /* το πλήθος των φιλοσόφων */

void philosopher(int i)         /* i: αριθμός φιλοσόφου (από 0 έως 4) */
{
    while (TRUE) {
        think();                /* ο φιλόσοφος σκέπτεται */
        take_fork(i);           /* πάρε το αριστερό πιρούνι */
        take_fork((i+1) % N);   /* πάρε το δεξιό πιρούνι· % είναι ο τελεστής modulo */
                                /* (υπόλοιπο ακέραιας διαίρεσης) */
        eat();                  /* φάε τη μακαρονάδα */
        put_fork(i);            /* απόθεσε το αριστερό πιρούνι στο τραπέζι */
        put_fork((i+1) % N);    /* απόθεσε το δεξιό πιρούνι στο τραπέζι */
    }
}
```

- Το πρόβλημα του δείπνου των φιλοσόφων
  - Αν όλοι πάρουν το αριστερό πιρούνι έχουμε αδιέξοδο
  - Αν κάνουμε πίσω όταν το δεξί δεν είναι διαθέσιμο;
    - Μπορεί να μην πάρουμε ποτέ τα πιρούνια
  - Αν έχουμε τυχαίο διάστημα αναμονής;
    - Πάλι μπορεί να μην πάρουμε ποτέ τα πιρούνια

Το δείπνο των φιλοσόφων (3 από 4)

```
#define N           5           /* πλήθος φιλοσόφων */
#define LEFT        (i+N–1)%N   /* αριθμός αριστερού γείτονα */
#define RIGHT       (i+I)%N     /* αριθμός δεξιού γείτονα */
#define THINKING    0           /* ο φιλόσοφος σκέπτεται */
#define HUNGRY      1           /* ο φιλόσοφος επιχειρεί να πάρει τα πιρούνια */
#define EATING      2           /* ο φιλόσοφος τρώει*/
typedef int semaphore;          /* οι σηματοφόροι είναι ειδική κατηγορία ακεραίων */
int state[N];                   /* πίνακας για την παρακολούθηση της κατάστασης καθενός */
semaphore mutex = 1;            /* αμοιβαίος αποκλεισμός για τις κρίσιμες περιοχές */
semaphore s[N];                 /* ένας σηματοφόρος ανά φιλόσοφο */

void philosopher(int i)         /* i: αριθμός κάθε φιλοσόφου (από 0 έως N – 1) */
{
    while (TRUE) {              /* επάναληψη επάπειρον */
        think();                /* ο φιλόσοφος σκέπτεται */
        take_forks(i);          /* πάρε δύο πιρούνια ή σταμάτα*/
        eat();                  /* φάε τη μακαρονάδα */
        put_forks(i);           /* άφησε και τα δύο πιρούνια στο τραπέζι */
    }
}
```

- Λύση με σηματοφόρους
  - Ο φιλόσοφος μπορεί να φάει αν δεν τρώνε οι γείτονες
  - Κεντρικός σηματοφόρος και πίνακας σηματοφόρων

Το δείπνο των φιλοσόφων (4 από 4)

```
void take_forks(int i)          /* i: αριθμός φιλοσόφου (από 0 έως N – 1) */
{
    down(&mutex);               /* είσοδος στην κρίσιμη περιοχή */
    state[i] = HUNGRY;          /* καταγραφή ότι ο i φιλόσοφος πεινάει */
    test(i);                    /* προσπάθησε να αποκτήσεις δύο πιρούνια */
    up(&mutex);                 /* έξοδος από την κρίσιμη περιοχή */
    down(&s[i]);                /* μπλοκάρισμα αν αποκτήθηκαν τα πιρούνια */
}

void put_forks(int i)           /* i: αριθμός φιλοσόφου (από 0 έως N – 1) */
{
    down(&mutex);               /* είσοδος στην κρίσιμη περιοχή */
    state[i] = THINKING;        /* ο φιλόσοφος έφαγε */
    test(LEFT);                 /* μπορεί να φάει ο αριστερός γείτονας; */
    test(RIGHT);                /* μπορεί να φάει ο δεξιός γείτονας; */
    up(&mutex);                 /* έξοδος από την κρίσιμη περιοχή */
}

void test (int i)               /* i: αριθμός φιλοσόφου (από 0 έως N – 1) */
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        up(&s[i]);
    }
}
```

Πρόβλημα αναγνωστών-γραφέων (1 από 3)

- Το πρόβλημα αναγνωστών-γραφέων
  - Μοντελοποιεί πρόσβαση σε βάσεις δεδομένων
  - Πολλοί μπορούν να διαβάζουν τη βάση
    - Επιτρέπεται ταυτόχρονη ανάγνωση χωρίς όριο
  - Μόνο ένας όμως μπορεί να γράφει στη βάση
    - Όταν η βάση γράφεται, απαγορεύεται η ανάγνωση
    - Δεν επιτρέπεται η ταυτόχρονη εγγραφή
  - Ποιος έχει προτεραιότητα;
    - Περιμένει ο γραφέας ή οι αναγνώστες;

Πρόβλημα αναγνωστών-γραφέων (2 από 3)

```
typedef int semaphore;          /* χρησιμοποιήστε τη φαντασία σας */
semaphore mutex = 1;            /* ελέγχει την πρόσβαση στην 'rc' */
semaphore db = 1;               /* ελέγχει την πρόσβαση στη βάση δεδομένων */
int rc = 0;                     /* πλήθος διεργασιών που διαβάζουν ή θέλουν */
                                /* να διαβάσουν δεδομένα */


void reader(void)
{
    while (TRUE) {              /* επανάληψη επάπειρον */
        down(&mutex);           /* αποκλειστική πρόσβαση στην 'rc' */
        rc = rc + 1;            /* ένας ακόμη αναγνώστης */
        if (rc == 1) down(&db); /* αν είναι ο πρώτος αναγνώστης ... */
        up(&mutex);             /* αποδέσμευση αποκλειστικής πρόσβασης στην 'rc' */
        read_data_base();       /* προσπέλαση δεδομένων */
        down(&mutex);           /* αποκλειστική πρόσβαση στην 'rc' */
        rc = rc – 1;            /* ένας αναγνώστης λιγότερος */
        if (rc == 0) up(&db);   /* αν είναι ο τελευταίος αναγνώστης ... */
        up(&mutex);             /* αποδέσμευση αποκλειστικής πρόσβασης στην 'rc' */
        use_data_read();        /* μη κρίσιμη περιοχή */
    }
}
```

- Το πρόβλημα αναγνωστών-γραφέων
  - Πολλοί αναγνώστες αλλά μόνο ένας γραφέας

Πρόβλημα αναγνωστών-γραφέων (3 από 3)

```
void writer(void)
{
    while (TRUE) {              /* επανάληψη επάπειρον */
        think_up_data();        /* μη κρίσιμη περιοχή */
        down(&db);              /* αποκλειστική πρόσβαση */
        write_data_base();      /* ενημέρωση των δεδομένων */
        up(&db);                /* αποδέσμευση αποκλειστικής πρόσβασης */
    }
}
```

- Λύση με σηματοφόρους
  - Ο db επιτρέπει είτε έναν γραφέα, είτε πολλούς αναγνώστες
    - Αλλάζει τιμή από τον πρώτο/τελευταίο αναγνώστη
  - Οι αναγνώστες συγχρονίζονται μέσω του mutex
    - Προστατεύει τη μεταβλητή rc
  - Ο γραφέας μπορεί να μην εκτελεστεί ποτέ!
    - Εκτός αν οι νέοι αναγνώστες εμποδίζονται όταν περιμένει γραφέας

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Τέλος Ενότητας #2

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα # 2:** Διεργασίες και Νήματα

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής