




Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών  
Εργαστήριο Υπολογιστικών Συστημάτων

# Διεργασίες και Νήματα (2/2)


Λειτουργικά Συστήματα Υπολογιστών  
6ο Εξάμηνο, 2022-2023

# Νήματα - Σύνοψη



- ◆ Ορισμός νημάτων, σχέση με διεργασίες
- ◆ Πλεονεκτήματα πολυνηματισμού
- ◆ Μοντέλα πολυνηματισμού
  - ➔ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- ◆ Βιβλιοθήκες και APIs πολυνηματισμού
  - ➔ POSIX Threads, Win32, Java Threads

# Νήματα - Σύνοψη

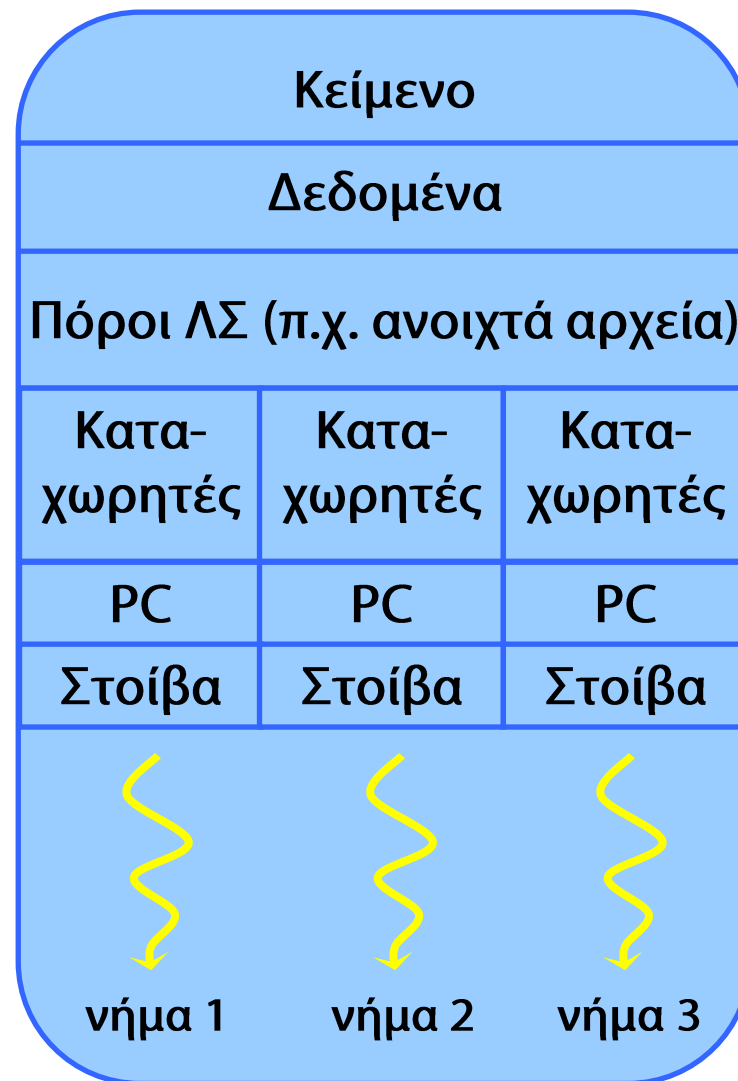
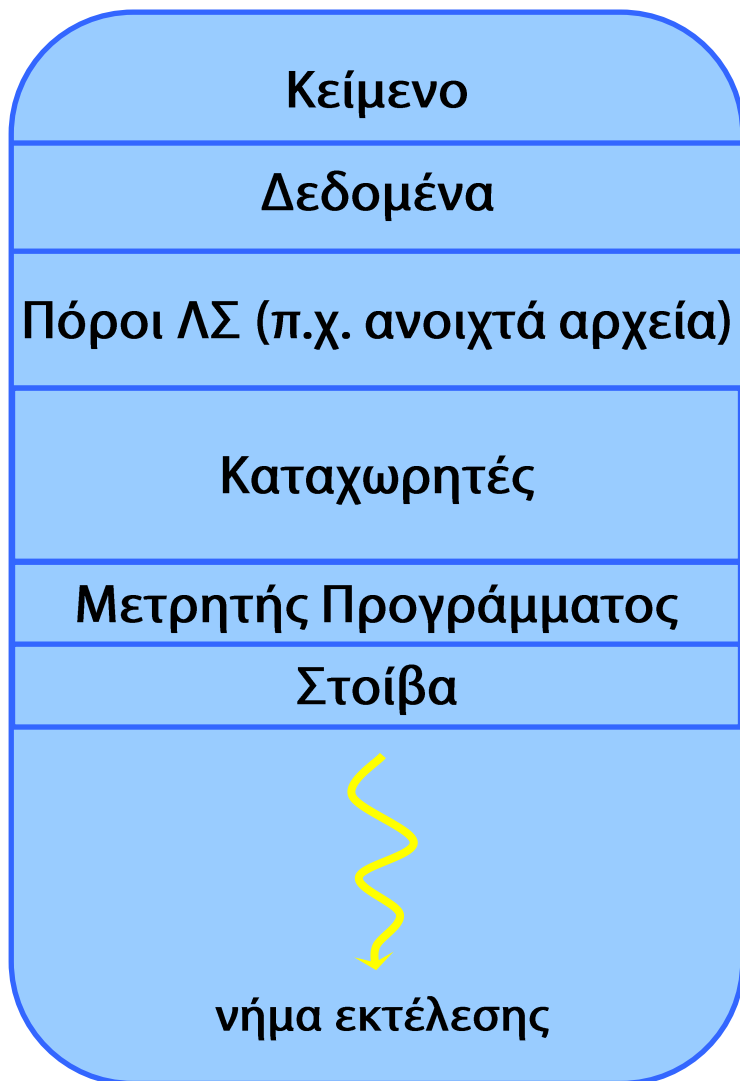


- ◆ Ορισμός νημάτων, σχέση με διεργασίες
- ◆ Πλεονεκτήματα πολυνηματισμού
- ◆ Μοντέλα πολυνηματισμού
  - ➔ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- ◆ Βιβλιοθήκες και APIs πολυνηματισμού
  - ➔ POSIX Threads, Win32, Java Threads


# Νήματα – Threads

- ◆ Νήματα: χωριστές ροές εκτέλεσης μέσα στην ίδια διεργασία
  - ➔ Κατάσταση διεργασίας =  
αρχιτεκτονική κατάσταση + πόροι ΛΣ
  - ➔ Χωριστή αρχιτεκτονική κατάσταση για κάθε νήμα
    - Μετρητής Προγράμματος, Καταχωρητές CPU, Στοίβα
  - ➔ Όλα τα νήματα ζουν μέσα στην ίδια διεργασία
    - Μοιράζονται τους πόρους του ΛΣ
    - Κοινή μνήμη, κοινά ανοιχτά αρχεία, κοινά δικαιώματα
    - Άμεση πρόσβαση σε κοινά δεδομένα

# Πολυνηματική Διεργασία



# Νήματα - Σύνοψη




- ◆ Ορισμός νημάτων, σχέση με διεργασίες
- ◆ Πλεονεκτήματα πολυνηματισμού
- ◆ Μοντέλα πολυνηματισμού
  - ➔ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- ◆ Βιβλιοθήκες και APIs πολυνηματισμού
  - ➔ POSIX Threads, Win32, Java Threads

# Νήματα – Γιατί;

- ◆ Οικονομία πόρων
  - ➔ Ένα νήμα θέλει λιγότερους πόρους ΛΣ από μια διεργασία
- ◆ Υψηλότερη επίδοση
  - ➔ Μικρότερο κόστος δημιουργίας/καταστροφής νημάτων
- ◆ Παράλληλος υπολογισμός
  - ➔ Μικρό κόστος επικοινωνίας ανάμεσα σε νήματα
- ◆ Προσαρμογή στην εφαρμογή (customization)
  - ➔ Χρονοδρομολόγηση νημάτων προσαρμοσμένη στην εφαρμογή
- ◆ Αποκρισιμότητα
  - ➔ Διαφορετικά νήματα κάνουν διαφορετικά πράγματα, π.χ. νήμα αλληλεπίδρασης με χρήστη και νήματα υπολογισμού
- ◆ Ευκολότερος προγραμματισμός / Δομημένη σχεδίαση
  - ➔ Πολυνηματικοί εξυπηρετητές
    - Κάθε νήμα έχει τη δική του κατάσταση → νήμα ανά πελάτη
    - Ασύγχρονη λειτουργία με σύγχρονες κλήσεις συστήματος


# Νήματα - Σύνοψη



- ◆ Ορισμός νημάτων, σχέση με διεργασίες
- ◆ Πλεονεκτήματα πολυνηματισμού
- ◆ Μοντέλα πολυνηματισμού
  - ➔ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- ◆ Βιβλιοθήκες και APIs πολυνηματισμού
  - ➔ POSIX Threads, Win32, Java Threads

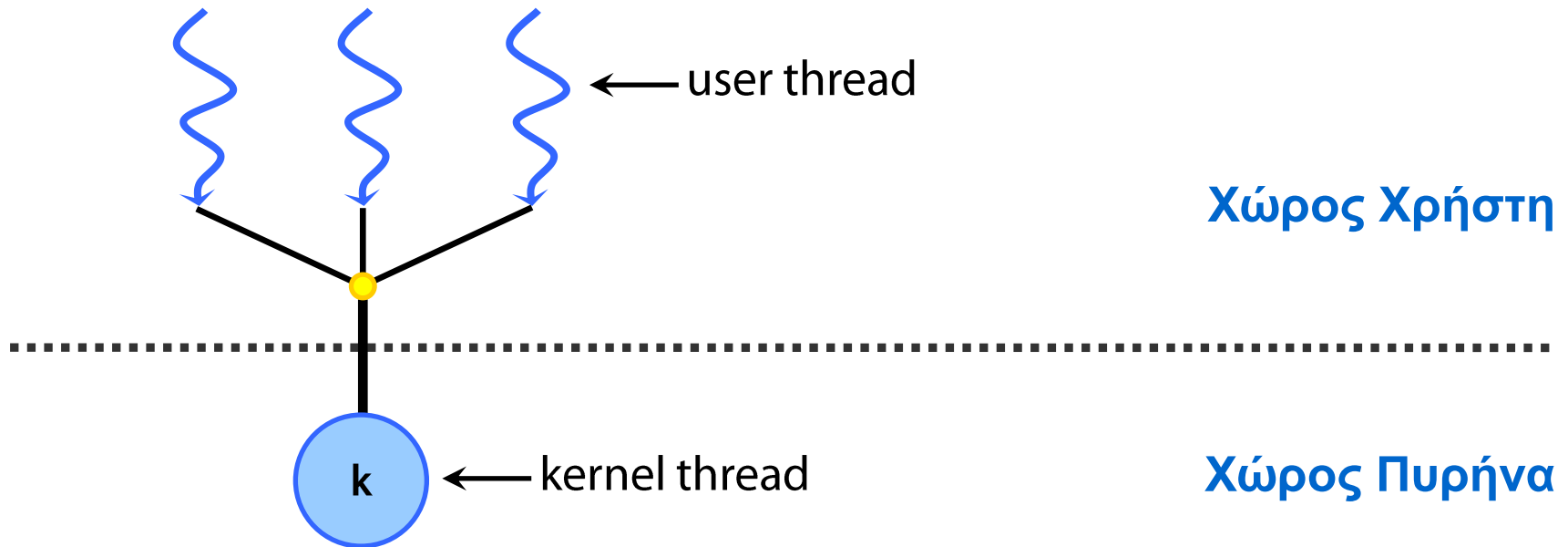


# Μοντέλα Πολυνηματισμού



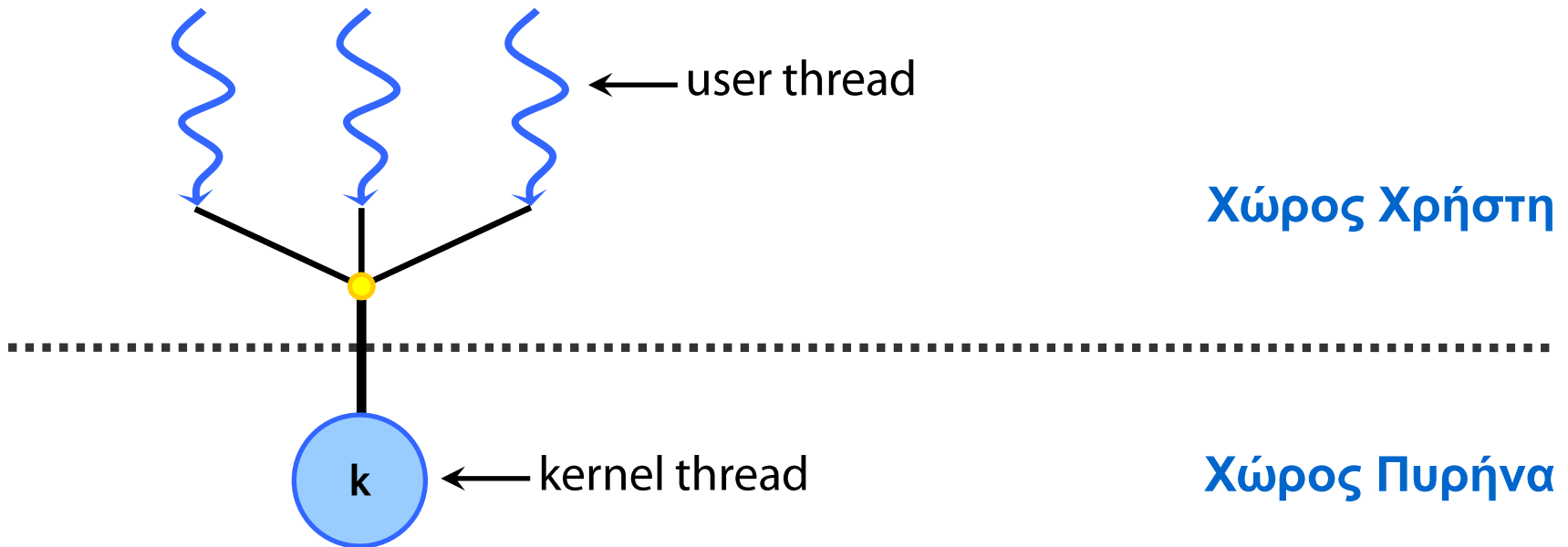
- ◆ Προγραμματιστικά μοντέλα και APIs για πολυνηματισμό
  - POSIX Threads, Win32 API, Java threads, OpenMP
- ◆ Πώς υλοποιείται η υποστήριξη νημάτων;
  - ➔ Στο χώρο πυρήνα, στο χώρο χρήστη, με συνεργασία και των δύο;
  - ➔ Νήματα χώρου χρήστη – νήματα χώρου πυρήνα
    - Ο χρονοδρομολογητής του ΛΣ βλέπει μόνο νήματα πυρήνα
  - ➔ Πλεονεκτήματα / μειονεκτήματα της κάθε προσέγγισης;

# N:1 - Νήματα σε επίπεδο χρήστη (1)



- ◆ Δημιουργία νημάτων σε επίπεδο χρήστη
  - ➔ Με φορητό τρόπο, πχ. `swarcontext()`, `setjmp/longjmp()`
  - ➔ Ο πυρήνας δεν γνωρίζει τίποτε για αυτά
  - ➔ Η χρονοδρομολόγηση γίνεται από κώδικα χρήστη
    - Είτε με διακοπτό (SIGALRM) είτε συνεργατικά, με μη-διακοπτό τρόπο

# N:1 - Νήματα σε επίπεδο χρήστη (2)

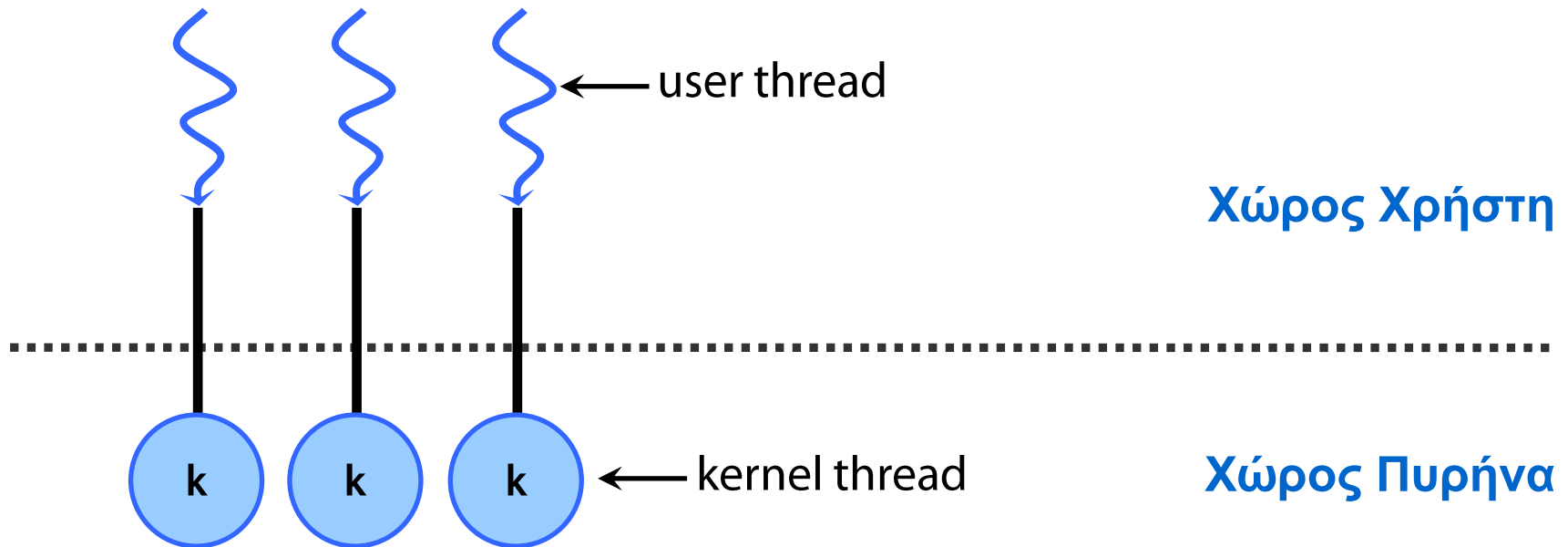


- ◆ Μικρό κόστος διαχείρισης νημάτων
- ◆ Χρονοδρομολόγηση προσαρμοσμένη στην εφαρμογή
- ◆ Τι γίνεται με κλήσεις συστήματος που προκαλούν αναστολή της εκτέλεσης;
  - ➔ Αν ένα νήμα μπλοκάρει σε `read()`, όλη η διεργασία μπλοκάρει

# Παράδειγμα υλοποίησης N:1 - GNU Pth

- ◆ Πολυνηματική βιβλιοθήκη σε επίπεδο χρήστη, με υποστήριξη και του Pthreads API
- ◆ Μη-διακοπτός καταμερισμός χρόνου (non-preemptive multitasking)
- ◆ Δεν χρειάζεται υποστήριξη από τον πυρήνα
  - ➔ Διαχείριση νημάτων με κλήσεις POSIX
    - `makecontext()` / `swapcontext()`, `setjmp()`/`longjmp()`
- ◆ Φορητότητα, μικρό κόστος διαχείρισης νημάτων
- ◆ Προσαρμοσμένη σε πολυνηματικούς εξυπηρετητές, όχι παράλληλο υπολογισμό
- ◆ Προβλήματα: διαχείριση κλήσεων συστήματος, υποστήριξη πολυπεξεργαστών;
  - ➔ “On the other hand, it cannot benefit from the existence of multiprocessors, because for this, kernel support would be needed. In practice, this is no problem, because multiprocessor systems are rare ” [GNU Pth manual]

# 1:1 - Νήματα σε επίπεδο πυρήνα



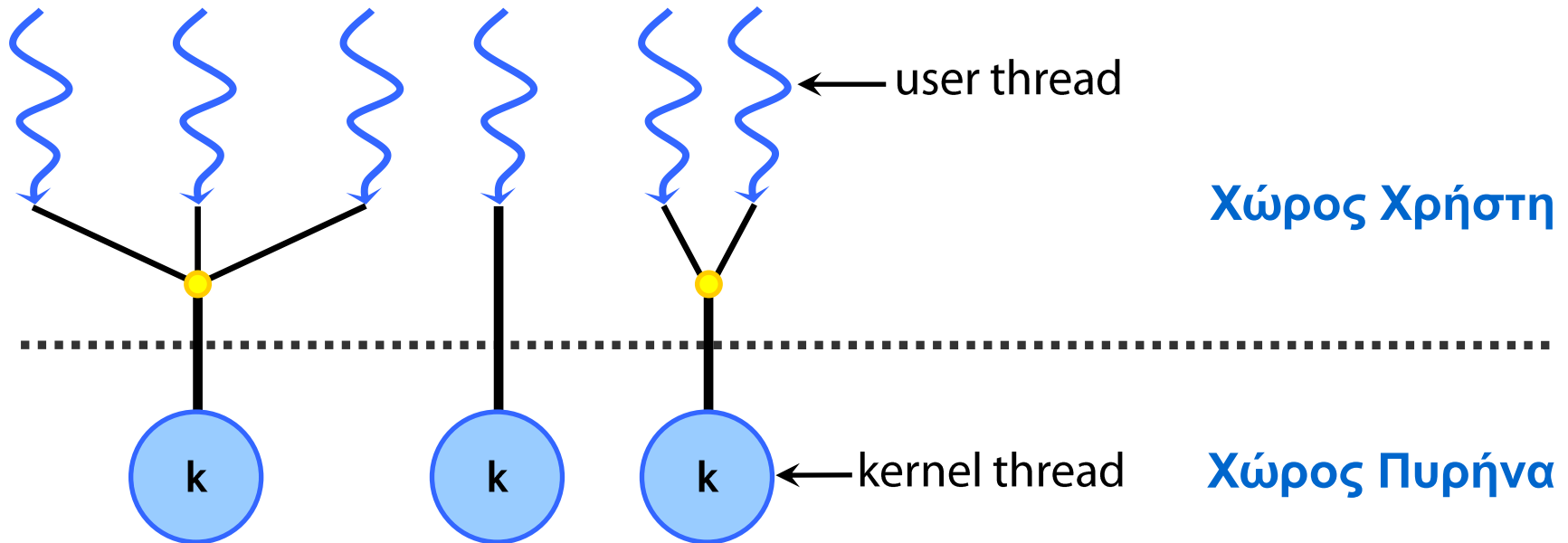
- ◆ Υποστήριξη πολυεπεξεργαστών, χωριστά νήματα σε χωριστούς πυρήνες πολυπύρηνων επεξεργαστών
- ◆ Ακριβότερη διαχείριση νημάτων
- ◆ Απλούστερη υλοποίηση για το χρήστη
- ◆ Linux NPTL, Win32 σε Windows NT/XP/2k, Solaris 9+

# Παράδειγμα υλοποίησης 1:1 – Linux NPTL

## ◆ Linux Native POSIX Threads Library

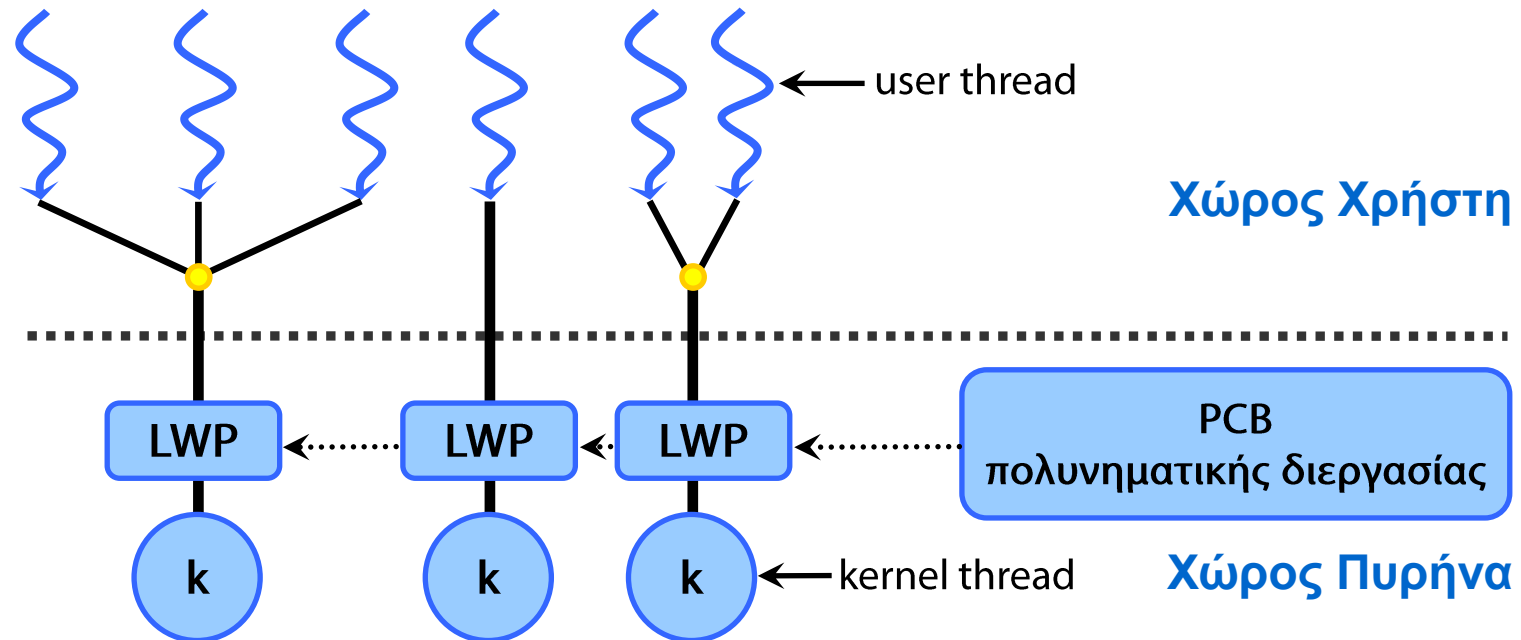
- ➔ Υλοποίηση της POSIX 1003.1c σε επίπεδο πυρήνα
- ➔ Βασισμένη στο μηχανισμό συγχρονισμού *futex* και την κλήση συστήματος `clone()`
- ➔ Διάφορα `clone()` flags: `CLONE_FILES`, `CLONE_SIGHAND`, `CLONE_VM`
- ➔ Η `fork()` υλοποιείται ως `clone()`
- ➔ Ο πυρήνας βλέπει *tasks*, που μοιράζονται σχεδόν τα πάντα (νήματα) ή τίποτε (διεργασίες)

# M:N – Συνδυασμός νημάτων χρήστη/πυρήνα



- ◆ Ένας αριθμός από νήματα χρήστη τρέχει πάνω σε περισσότερα του ενός νήματα πυρήνα
- ◆ Διεπίπεδη χρονοδρομολόγηση
  - ➔ Πολύπλοκη υλοποίηση
- ◆ Solaris έκδοσης < 9


# Παράδειγμα υλοποίησης M:N – Solaris



- ◆ Μια πολυνηματική διεργασία αποτελείται από πολλές ελαφρές διεργασίες – LightWeight Processes (LWPs)
- ◆ Ο χρονοδρομολογητής χώρου χρήστη βλέπει εικονικούς επεξεργαστές
- ◆ Ο χρονοδρομολογητής χώρου πυρήνα βλέπει νήματα πυρήνα
- ◆ Ανάγκη για *urcalls*: Ο πυρήνας ενημερώνει το χώρο χρήστη όταν ένα LWP πρόκειται να μπλοκάρει → εκτέλεση έτοιμου νήματος σε νέο LWP



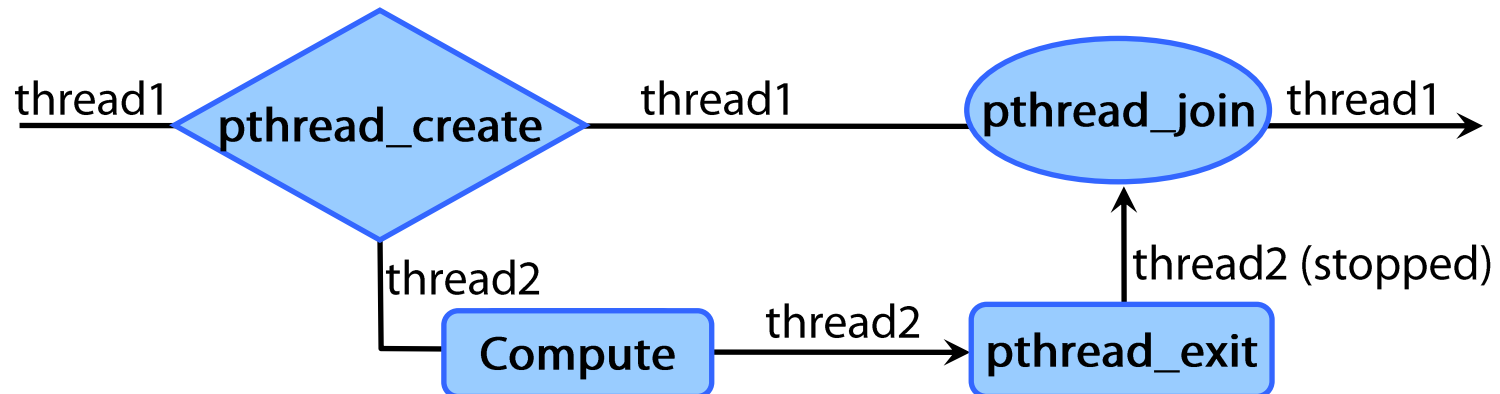
# Νήματα - Σύνοψη



- ◆ Ορισμός νημάτων, σχέση με διεργασίες
- ◆ Πλεονεκτήματα πολυνηματισμού
- ◆ Μοντέλα πολυνηματισμού
  - ➔ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- ◆ Βιβλιοθήκες και APIs πολυνηματισμού
  - ➔ POSIX Threads, Win32, Java Threads

# Δημιουργία νημάτων στα POSIX Threads

- ◆ Δημιουργία με `pthread_create()`
  - ➔ `int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);`
  - ➔ π.χ. `pthread_create(&tid, &attr, thread_fn, arg)`
- ◆ Αναμονή για τερματισμό (`pthread_exit()`) με `pthread_join()`



# Ερωτήσεις;



[goumas@cslab.ece.ntua.gr](mailto:goumas@cslab.ece.ntua.gr)

και στη λίστα:

[OS@lists.cslab.ece.ntua.gr](mailto:OS@lists.cslab.ece.ntua.gr)