

Προγραμματισμός Νημάτων POSIX

Η εισαγωγή αυτή είναι βασισμένη στο tutorial "POSIX Threads Programming" του Blaise Barney, Lawrence Livermore Laboratory, USA. Μετάφραση και προσαρμογή: Κ.Γ. Μαργαρίτης και Ι. Παπάκας, Εργαστήριο Παράλληλης Κατανεμημένης Επεξεργασίας, Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας.

Περιεχόμενα

- [Περίληψη](#)
- [Επισκόπηση των Pthreads](#)
 - [Τι είναι ένα Νήμα;](#)
 - [Τι είναι τα Pthreads;](#)
 - [Γιατί Pthreads;](#)
 - [Σχεδιασμός προγραμμάτων με Νήματα](#)
- [Το API των Pthreads](#)
- [Μεταγλώττιση προγραμμάτων με Νήματα](#)
- [Διαχείριση Νημάτων](#)
 - [Δημιουργία και Τερματισμός Νημάτων](#)
 - [Πέρασμα παραμέτρων στα Νήματα](#)
 - [Ένωση και Απόσπαση Νημάτων](#)
 - [Διαχείριση Στοιβάς](#)
 - [Διάφορες Ρουτίνες](#)
- [Μεταβλητές Αμοιβαίου Αποκλεισμού - Mutex](#)
 - [Επισκόπηση Μεταβλητών Mutex](#)
 - [Δημιουργία και Τερματισμός Mutex](#)
 - [Κλείδωμα και Ξεκλείδωμα Mutex](#)
- [Μεταβλητές Συνθηκών](#)
 - [Επισκόπηση Μεταβλητών Συνθηκών](#)
 - [Δημιουργία και Τερματισμός Μεταβλητών Συνθηκών](#)
 - [Αναμονή και Σηματοδότηση στις Μεταβλητές Συνθηκών](#)
- [Συνδυασμός MPI με Pthreads](#)
- [Θέματα που δεν συζητήθηκαν](#)
- [Βιβλιοθήκη Αναφορών Ρουτινών Pthread](#)
- [Αναφορές και Περισσότερες Πληροφορίες](#)
- [Ασκήσεις](#)

Περίληψη

Σε πολυπεξεργαστές με διαμοιραζόμενη μνήμη, όπως τα SMPs και τα muti-core συστήματα, ο παράλληλισμός μπορεί να υλοποιηθεί με τα νήματα. Ιστορικά, οι κατασκευαστές πολυεπεξεργαστών ανέπτυξαν τις δικές τους εκδόσεις νημάτων, προκαλώντας σημαντικά προβλήματα φορητότητας. Το πρότυπο IEEE POSIX 1003.1c αποτελεί μια πρότυπη διεπαφή προγραμματιστή εφαρμογών για τη διαχείριση νημάτων στην γλώσσα C για συστήματα UNIX (υποστηρίζεται και από τις πρόσφατες εκδόσεις των MS-Windows). Οι υλοποιήσεις που οφείλονται σε αυτό τον πρότυπο έχουν το όνομα POSIX Threads ή αλλιώς Pthreads.

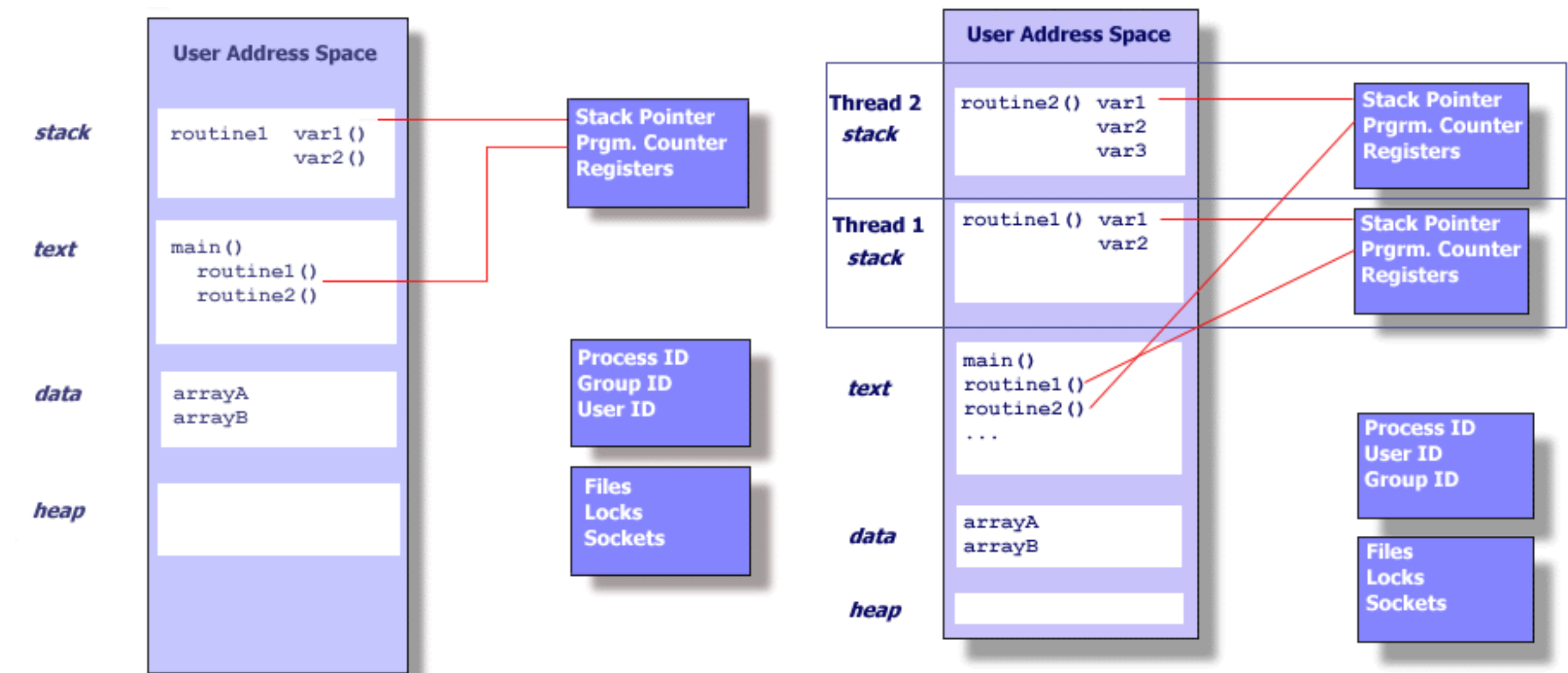
Ο οδηγός αυτός ξεκινά με μια εισαγωγή στις έννοιες, τα κίνητρα, και το σκεπτικ;ο του σχεδιασμού προγραμμάτων με Pthreads. Καθεμία από τις 3 μεγάλες κλάσεις ρουτινών στο API των Pthreads καλύπτεται: Διαχείριση Νημάτων, Μεταβλητές Mutex, και Μεταβλητές Συνθηκών. Παραδείγματα με κώδικα εμφανίζονται σε όλο τον οδηγό για να δείξουν την χρήση των περισσότερων ρουτινών των Pthreads που χρειάζεται ένας προγραμματιστής. Ο οδηγός καταλήγει σε μια συζήτηση για το πως να συνδυαστεί το MPI με τα Pthreads. Επίσης περιλαμβάνονται αρκετά παραδείγματα σε γλώσσα C.

Επισκόπηση των Pthreads

Τι είναι ένα Νήμα;

- Τεχνικά, ένα νήμα ορίζεται σαν μια διακριτή ακολουθία εντολών την οποία μπορεί να εκτελέσει το λειτουργικό σύστημα. Αλλά τι σημαίνει αυτό;
- Για τον προγραμματιστή, νήμα σημαίνει μια συνάρτησης (ρουτίνα) η οποία εκτελείται ανεξάρτητα από (συνδρομικά με) το κυρίως πρόγραμμα (concurrently).
- Για να πάμε ένα βήμα πιο μακριά, ας σκεφτούμε ένα κυρίως πρόγραμμα (a.out) το οποίο περιέχει έναν αριθμό από συναρτήσεις. Μετά ας σκεφτούμε ότι όλες οι συναρτήσεις μπορούν να εκτελεστούν συνδρομικά από το λειτουργικό σύστημα. Αυτό θα ήταν ένα πολυ-νηματικό πρόγραμμα.
- Πως επιτυγχάνεται αυτό;
- Προτού κατανοήσουμε ένα νήμα, πρέπει πρώτα να καταλάβουμε μια διεργασία του UNIX. Μια διεργασία δημιουργείται από το λειτουργικό σύστημα, και για τη διαχείρισή του απαιτούνται ορισμένες πληροφορίες. Οι διεργασίες συνοδεύονται από πληροφορίες σχετικά με το πρόγραμμα εκτελείται, τους πόρους που χρησιμοποιεί και την κατάστασή του, όπως:

- ID διεργασίας, ID χρήστη, και ID ομάδας χρήστη
- Μεταβλητές Περιβάλλοντος
- Τρέχων Κατάλογος.
- Εντολές Προγράμματος (Text)
- Τιμές Καταχωρητών
- Δεδομένα (Data)
- Στοίβα (Stack)
- Σωρός (Heap)
- Περιγραφείς αρχείων (File descriptors)
- Σήματα (Signals)
- Διαμοιραζόμενες Βιβλιοθήκες (Shared Libraries)
- Εργαλεία διαδιεργασιακής επικοινωνίας (όπως ουρές μηνυμάτων, διοχετεύσεις, semaphores, ή διαμοιραζόμενη μνήμη).



ΔΙΕΡΓΑΣΙΑ UNIX

ΝΗΜΑΤΑ ΜΕΣΑ ΣΕ ΜΙΑ ΔΙΕΡΓΑΣΙΑ UNIX

- Τα νήματα χρησιμοποιούν τους πόρους της διεργασίας, αλλά είναι ικανά να χρονο-προγραμματιστούν από το λειτουργικό σύστημα και να εκτελεστούν σαν ανεξάρτητες οντότητες.
- Η ανεξάρτητη ακολουθία εντολών επιτυγχάνεται διότι ένα νήμα έχει τα δικά του:
 - Στοίβα
 - Τιμές Καταχωρητών
 - Ιδιωτικά αλλά και Καθολικά Δεδομένα (του νήματος αλλά και της διεργασίας)
 - Σήματα
 - Ιδιότητες χρονο-προγραμματισμού (όπως πολιτικές ή προτεραιότητα)
- Περιληπτικά, στο περιβάλλον του UNIX ένα νήμα:
 - υπάρχει μέσα σε μια διεργασία και χρησιμοποιεί τις πηγές της διεργασίας
 - Έχει την δικιά του ροή ελέγχου όσο υπάρχει ο πατέρας του και το λειτουργικό το υποστηρίζει
 - Επιλέγει μόνο τις βασικές πηγές της διεργασίας που χρειάζεται για να εκτελεστεί ανεξάρτητα
 - Μπορεί να μοιραστεί τις πηγές της διεργασίας με άλλα νήματα και να συμπεριφερθεί ανεξάρτητα (αλλά και εξαρτημένα)
 - Τερματίζεται εάν ο πατέρας του τερματιστεί - ή κάτι παρόμοιο
 - Είναι "ελαφρύ" διότι το μεγαλύτερο μέρος του έχει επιτευχθεί κατά την δημιουργία του.
- Επειδή τα νήματα των ίδιων διεργασιών μοιράζονται τις πηγές:
 - Οι αλλαγές που πραγματοποιεί ένα νήμα σε αυτές τις πηγές (όπως το κλείσιμο ενός αρχείου) θα φανεί και από τα άλλα νήματα.
 - Δύο δείκτες που έχουν την ίδια τιμή δείχνουν στα ίδια δεδομένα.
 - Ανάγνωση και Εγγραφή στο ίδιο σημείο της μνήμης είναι εφικτό, και έτσι είναι απαραίτητος ο συγχρονισμός από τον προγραμματιστή.

Τι είναι τα Pthreads;

- Ιστορικά, οι κατασκευαστές υλικού υλοποιούσαν τις δικές τους εκδόσεις νημάτων. Αυτές οι υλοποιήσεις διέφεραν σε αρκετά μεγάλο βαθμό μεταξύ τους ώστε να είναι δύσκολο για τον προγραμματιστή να δημιουργήσει φορητές εφαρμογές που χρησιμοποιού νήματα.
- Για να μπορέσουν να εκμεταλλευθούν όλα τα πλεονεκτήματα που προσφέρουν τα νήματα, απαιτούνται μία πρότυπη διεπαφή προγραμματισμού.
 - Για τα συστήματα UNIX, αυτή η διεπαφή προσδιορίστηκε από το πρότυπο IEEE POSIX 1003.1c (1995).
 - Υλοποιήσεις βασισμένες σε αυτό το πρότυπο επίσης λέγονται και νήματα POSIX ή Pthreads.
 - Οι περισσότεροι κατασκευαστές πλέον υποστηρίζουν τα Pthreads μαζί με τα ιδιόκτητα API's.
- Το πρότυπο POSIX συνέχισε να εξελίσσεται και να ενημερώνεται συνεχώς, συμπεριλαμβανομένων των Pthreads. Η τελευταία γνωστή έκδοση είναι η έκδοση IEEE Std 1003.1, 2004.
- Μερικοί χρήσιμοι σύνδεσμοι:
 - POSIX FAQs: www.opengroup.org/austin/papers/posix_faq.html
 - Μεταφόρτωση του προτύπου: www.unix.org/version3/ieee_std.html
- Τα Pthreads ορίζονται σαν ένα σύνολο προγραμματιστικών τύπων δεδομένων και κλήσεων συναρτήσεων στην C, που υλοποιείται με την βιβλιοθήκη pthread.h και μία βιβλιοθήκη νημάτων - αν και αυτή η βιβλιοθήκη μπορεί να είναι μέρος άλλης βιβλιοθήκης, όπως σε μερικές υλοποιήσεις η libc.

Επισκόπηση Pthreads

Γιατί Pthreads;

- Αρχικά ας προσπαθήσουμε να η κατανόησουμε τα πιθανά κερδη από τη χρήση των Pthreads στον προγραμματισμό.
 - Το κόστος δημιουργίας και διαχείρισης μιας διεργασίας, είναι μεγαλύτερο από τα αντίστοιχα ενός νήματος. Η δημιουργία και διαχείριση των νημάτων απαιτεί λιγότερους πόρους από αυτές των διεργασιών.
- Ο επόμενος πίνακας συγκρίνει χρονικά αποτελέσματα για τις υπορουτίνες `fork()` και `pthread_create()`. Οι χρονισμοί αντιστοιχούν σε 50,000 επαναλήψεις, εκτελέστηκαν με το εργαλείο `time`, και οι χρόνοι είναι σε δευτερόλεπτα, δεν χρησιμοποιήθηκαν σημεία βελτιστοποίησης.
- Σημείωση: οι χρόνοι του συστήματος (sys) και του χρήστη (user) συνήθως δεν συμβαδίζουν με τον πραγματικό χρόνο (real), επειδή στα συστήματα αυτά συνήθως εκτελούνται πολλές εργασίες ταυτόχρονα.

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
AMD 2.3 GHz Opteron (16 cpus/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cpus/node)	17.6	2.2	15.7	1.4	0.3	1.3
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

- Όλα τα νήματα μέσα σε μια διεργασία μοιράζονται τον ίδιο χώρο διευθύνσεων. Η δια-νηματική επικοινωνία είναι σε πολλές περιπτώσεις πιο αποδοτική, πιο εύκολη από την δια-διεργασιακής επικοινωνία.
- Εφαρμογές που χρησιμοποιούν νήματα προσφέρουν πιθανά κέρδη απόδοσης και έχουν πρακτικά πλεονεκτήματα εν σχέσει με τις εφαρμογές που χρησιμοποιούν απλές διεργασίες σε πολλούς τομείς:
 - Επικάλυψη χρήσης CPU και I/O: Για παράδειγμα, ένα πρόγραμμα μπορεί να έχει τμήματα όπου θέλει να εκτελέσει μια λειτουργία I/O που διαρκεί πολύ. Ενώ ένα νήμα περιμένει για την διαδικασία αυτή, άλλα νήματα μπορεί να κάνουν σκληρή δουλειά στη CPU.
 - Σχεδιασμός προτεραιότητας/πραγματικού χρόνου: διεργασίες που είναι πιο σημαντικές μπορούν να προγραμματιστούν να υπερκαλύψουν ή να διακόψουν διεργασίες κατώτερης προτεραιότητας.
 - Ασύγχρονη διαχείριση χειριστηρίων: διεργασίες οι οποίες χειρίζονται γεγονότα με απροσδιόριστη συχνότητα και διάρκεια μπορούν να υπερκαλυφθούν. Για παράδειγμα, ένας web server μπορεί ταυτόχρονα να μεταφέρει δεδομένα από προηγούμενα αιτήματα και να διαχειριστεί την άφιξη νέων αιτημάτων.
- Ο πρωταρχικός στόχος-κίνητρο για την χρήση των Pthreads σε μια αρχιτεκτονική SMP είναι η επίτευξη βέλτιστης απόδοσης. Ιδιαίτερα, αν μια εφαρμογή χρησιμοποιεί το MPI για επικοινωνίες με την χρήση κόμβων, υπάρχει η δυνατότητα για πιθανή αύξηση της αποδοτικότητας με την χρήση των Pthreads για την μεταφορά δεδομένων από κόμβο σε κόμβο.
- Για παράδειγμα:
 - Οι βιβλιοθήκες του MPI συνήθως υλοποιούν επικοινωνίες από κόμβο σε κόμβο διαμέσου διαμοιραζόμενης μνήμης, το οποίο συμπεριλαμβάνει τουλάχιστον μία λειτουργία αντιγραφής στην μνήμη (διεργασία σε διεργασία).
 - Για τα Pthreads δεν χρειάζεται καμία λειτουργία αντιγραφής στην μνήμη διότι μοιράζονται τον ίδιο χώρο μνήμης μέσα σε μια διεργασία. Δεν υπάρχει μεταφορά δεδομένων. Γίνεται περισσότερο μια κατάσταση μεταφοράς cache-to-CPU ή memory-to-CPU (χειρότερη περίπτωση). Αυτές οι ταχύτητες είναι πολύ μεγαλύτερες.
 - Μερικές τοπικές συγκρίσεις παρουσιάζονται παρακάτω:

Platform	MPI Shared Memory Bandwidth (GB/sec)	Pthreads Worst Case Memory-to-CPU Bandwidth
----------	---	--

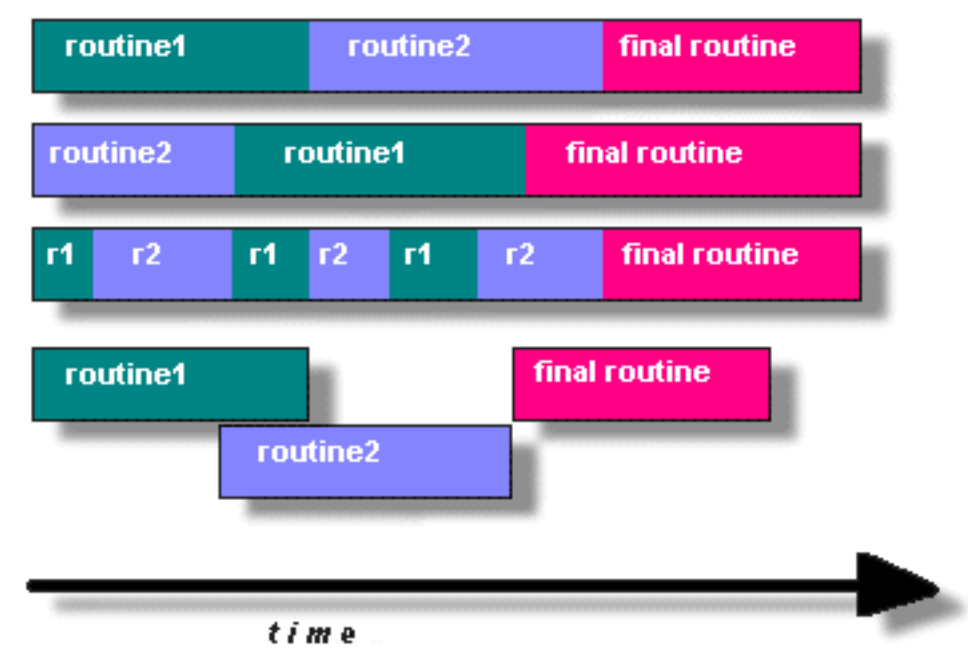
		(GB/sec)
AMD 2.3 GHz Opteron	1.8	5.3
AMD 2.4 GHz Opteron	1.2	5.3
Intel 2.4 GHz Xeon	0.3	4.3
Intel 1.4 GHz Itanium 2	1.8	6.4

Επισκόπηση Pthreads

Σχεδιασμός Προγραμμάτων με Νήματα

► Παράλληλος Προγραμματισμός:

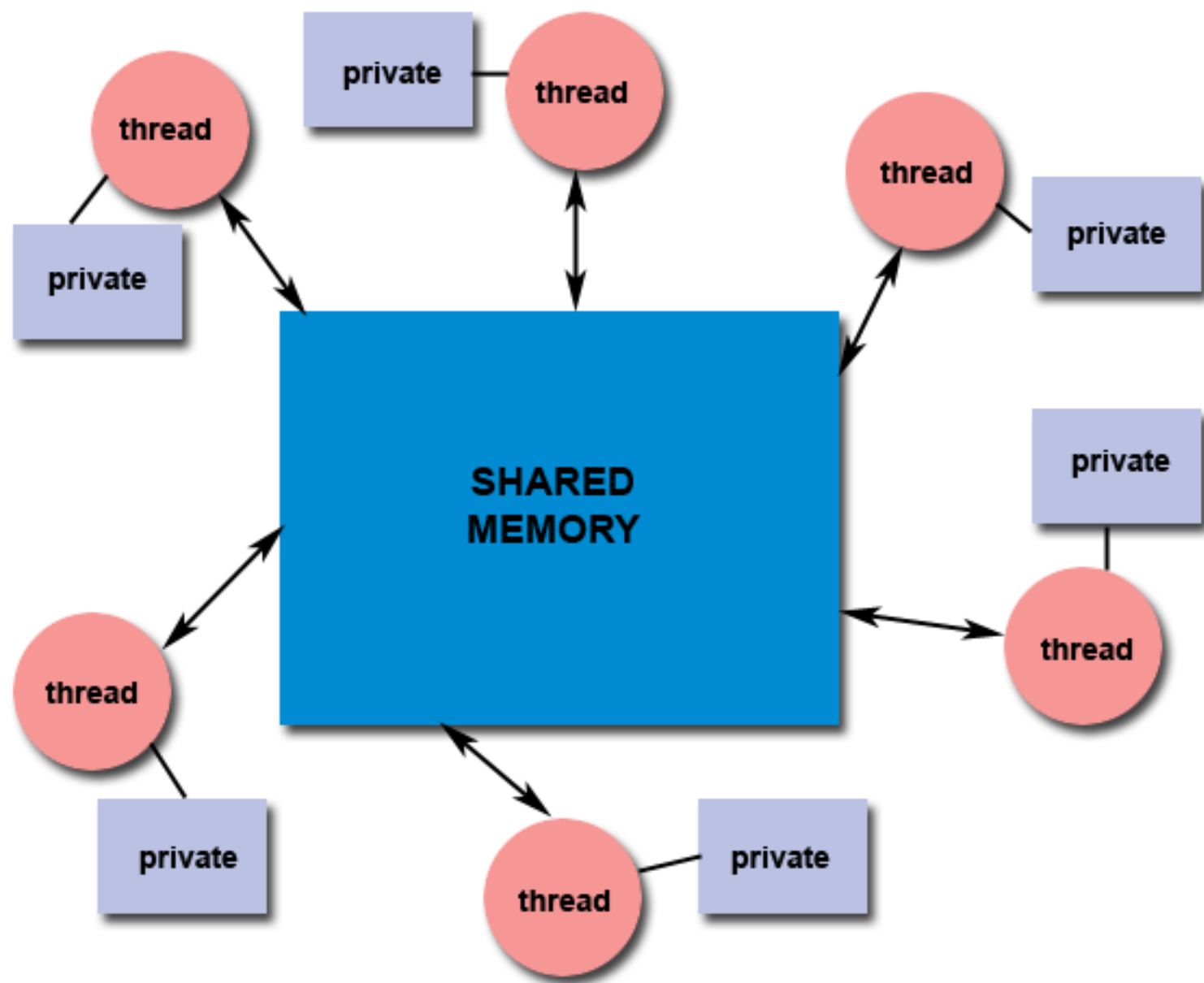
- Στα σύγχρονα πολυεπεξεργαστικά συστήματα, τα Pthreads είναι ιδανικά για τον παράλληλο προγραμματισμό, και οτιδήποτε αντιστοιχεί σε αυτόν γενικά, ισχύει και στα προγράμματα που χρησιμοποιούν Pthreads.
- Υπάρχουν πολλά θέματα προς συζήτηση κατά τον σχεδιασμό παράλληλων προγραμμάτων, όπως:
 - Τι τύπο παράλληλου προγραμματισμού να χρησιμοποιήσω;
 - Πρόβλημα διαμοιρασμού δεδομένων
 - Εξισορρόπηση φορτίου
 - Επικοινωνίες δεδομένων
 - Εξάρτησεις δεδομένων
 - Συγχρονισμός και συνθήκες εκτέλεσης
 - Θέματα διαχείρισης μνήμης
 - Θέματα I/O
 - Πολυπλοκότητα προγράμματος
 - Χρόνος/κόστος/προσπάθειες προγραμματιστή
 - ...
- Η κάλυψη αυτών των θεμάτων είναι πέρα από το πεδίο αυτού του οδηγού, ωστόσο οι αναγνώστες που ενδιαφέρονται μπορούν να ρίξουν μια ματιά στον οδηγό "Εισαγωγή στην Παράλληλη Επεξεργασία".
- Σε γενικές γραμμές, για να μπορέσει ένα πρόγραμμα να εκμεταλλευτεί τα Pthreads, πρέπει να είναι ικανό να οργανωθεί διακριτικά, με ανεξάρτητες διεργασίες οι οποίες μπρούν να εκτελεστούν ταυτόχρονα. Για παράδειγμα, αν οι ρουτίνες routine1 και routine2 μπορούν να αλλάξουν σειρά εκτέλεσης στον πραγματικό χρόνο, είναι υποψήφιες για χρήση νημάτων.



- Τα προγράμματα που έχουν τα ακόλουθα χαρακτηριστικά μπορούν κάλλιστα να χρησιμοποιήσουν Pthreads:
 - Εργασία που μπορεί να εκτελεστεί, ή δεδομένα στα οποία μπορεί να γίνει εργασία, από πολλές διεργασίες ταυτόχρονα
 - Αναστολή λόγω πιθανών αναμονών λόγω I/O
 - Εντατική χρήση CPU σε μερικά τμήματα αλλά όχι σε άλλα
 - Πρέπει να απαντάει σε ασύγχρονα γεγονότα
 - Κάποια εργασία είναι πιο σημαντική από κάποια άλλη (διακοπή λόγω προτεραιότητας)
- Τα Pthreads μπορούν επίσης να χρησιμοποιηθούν σε ακολουθιακές εφαρμογές, για την προσομοίωση παράλληλης εκτέλεσης. Ένα τέλειο παράδειγμα είναι ένας τυπικός web browser, ο οποίος για τους περισσότερους ανθρώπους, τρέχει σε έναν μόνο επεξεργαστή σε ένα σύστημα. Πολλά πράγματα "φαίνεται" να συμβαίνουν ταυτόχρονα.
- Υπάρχουν πολλά κοινά μοντέλα για προγράμματα με νήματα:
 - **Συντονιστής/εργαζόμενοι:** ένα μοναδικό νήμα, ο συντονιστής αναθέτει δουλειά σε άλλα νήματα, τους εργαζόμενους. Τυπικά, ο συντονιστής χειρίζεται όλες τις εισόδους του προγράμματος και διανέμει την δουλειά στα άλλα νήματα. Τουλάχιστον δύο μοντέλα συντονιστή/εργαζόμενων είναι κοινά: στατικοί και δυναμικοί εργαζόμενοι.
 - **Διοχέτευση:** μία εργασία αναλύεται σε μια σειρά από υπο-εργασίες, καθεμία από τις οποίες διαχειρίζεται με την σειρά δεδομένα που διοχετεύονται από τη μια υπο-εργασία στην επόμενη. Οι υπο-εργασίες εκτελούνται συγχρονικά, κάθε μια από διαφορετικό νήμα.
 - **Ομότιμοι εργαζόμενοι:** Παρόμοιο με το μοντέλο συντονιστή/εργαζομένων, αλλά αφού το κυρίως νήμα δημιουργήσει άλλα νήματα, συμμετέχει στην δουλειά.

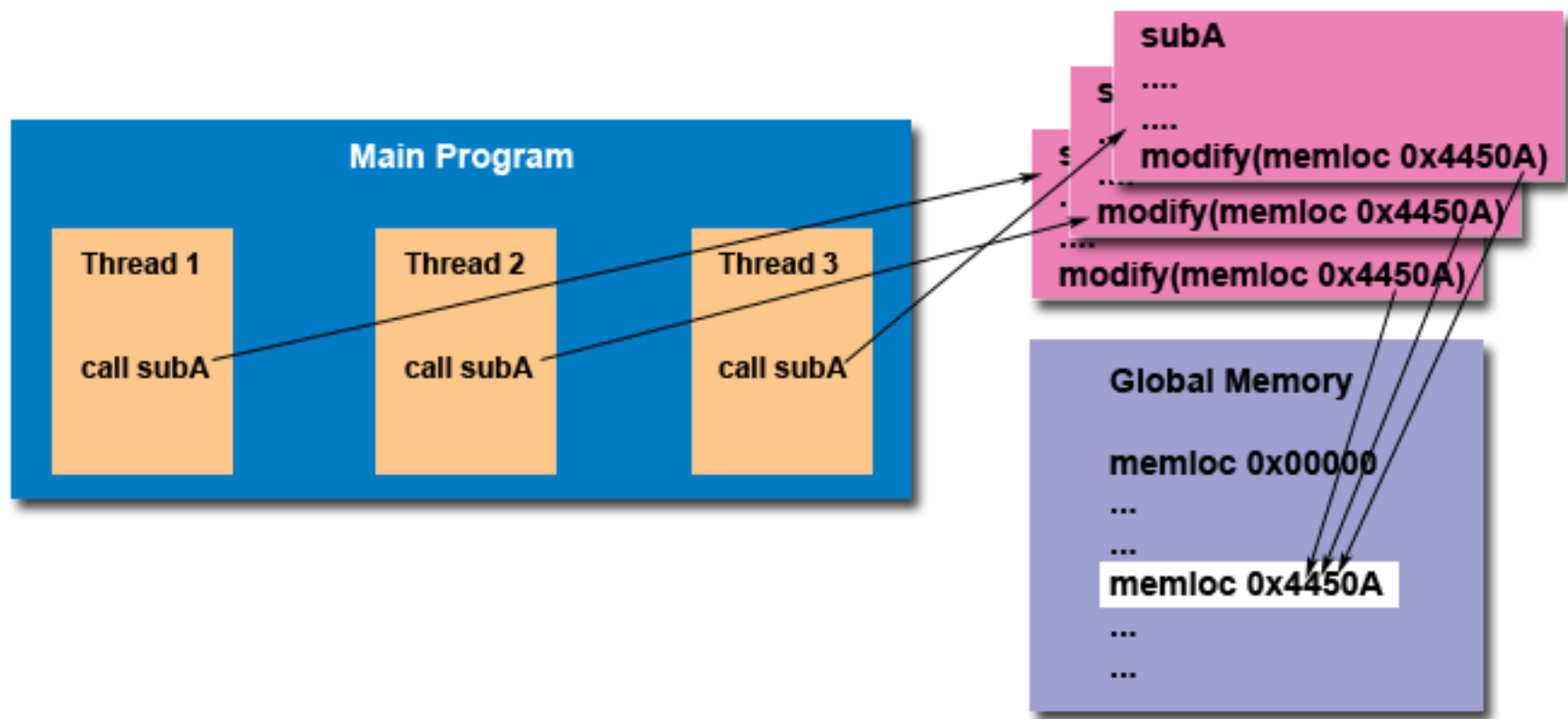
► Μοντέλο Διαμοιραζόμενης Μνήμης:

- Όλα τα νήματα έχουν πρόσβαση στην ίδια, διαμοιραζόμενη μνήμη
- Τα νήματα έχουν επίσης τα δικά τους ιδιωτικά δεδομένα
- Οι προγραμματιστές είναι υπεύθυνοι για την συγχρονισμένη πρόσβαση στα κοινά δεδομένα.



► Ασφάλεια Νημάτων:

- Ασφάλεια νημάτων: περιληπτικά, αναφέρεται στην ικανότητα του προγράμματος να εκτελέσει πολλά νήματα ταυτόχρονα χωρίς τον κίνδυνο αλλοίωσης των δεδομένων στη μνήμη ή την αναστολή από λάθος συνθήκες.
- Για παράδειγμα, υποθέτοντας ότι μια εφαρμογή δημιουργεί πολλά νήματα, καθένα από τα οποία καλεί την ίδια ρουτίνα:
 - Η ρουτίνα αυτή διαβάζει/τροποποιεί μία γενική δομή ή τοποθεσία στην μνήμη.
 - Καθώς το κάθε νήμα καλεί αυτή την ρουτίνα είναι πιθανόν όλα να προσπαθήσουν να αλλάξουν αυτή τη δομή την ίδια στιγμή.
 - Αν η ρουτίνα δεν εφαρμόσει κάποιου είδους συγχρονισμό για να εμποδίσει την καταστροφή δεδομένων, τότε δεν είναι ασφαλές.



- Η επίπτωση των ρουτινών αυτών στους χρήστες είναι ότι αν δεν 100% σίγουρο ότι η χρήση των νημάτων είναι ασφαλής, τότε πρέπει να είναι έτοιμοι να αντιμετωπίσουν τα προβλήματα που θα δημιουργηθούν.
- Σύσταση: Προσοχή στην χρήση βιβλιοθηκών ή άλλων αντικειμένων από το πρόγραμμα τα οποία δεν εξασφαλίζουν την ασφάλεια των νημάτων. Όταν δεν είναι σίγουρο, πρέπει να θεωρηθεί ότι δεν είναι ασφαλή μέχρι αποδείξεως του αντιθέτου. Αυτό μπορεί να συμβεί βάζοντας σε σειρά τις κλήσεις στην ρουτίνα προς παρατήρηση κλπ.

To API των Pthreads

- Το αρχικό API των Pthreads ορίστηκε από το πρότυπο ANSI/IEEE POSIX 1003.1 - 1995. Το πρότυπο POSIX συνέχισε να εξελίσσεται και να υπάρχουν αναθεωρήσεις, συμπεριλαμβάνοντας τα Pthreads. Η τελευταία έκδοση είναι γνωστή ως IEEE Std 1003.1, 2004.
- Αντίγραφα του προτύπου μπορούν να προμηθευτούν από τον IEEE ή να αποκτηθούν δωρεάν από την σελίδα www.unix.org/version3/ieee_std.html.

- Οι ρουτίνες που περιλαμβάνει το API των Pthreads μπορούν να κατανεμηθούν σε 4 μεγάλες ομάδες:
 1. **Διαχείριση Νημάτων:** Ρουτίνες οι οποίες εργάζονται άμεσα με νήματα - δημιουργία, αποσύνδεση, σύνδεση, κλπ. Επίσης περιλαμβάνουν συναρτήσεις για να ορίσουν/εξετάσουν ιδιότητες των νημάτων.
 2. **Mutexes:** Ρουτίνες οι οποίες διαχειρίζονται τον συγχρονισμό, ονομάζονται "mutex", το οποίο είναι συντόμευση του "mutual exclusion" (αμοιβαίος αποκλεισμός). Οι συναρτήσεις mutex προσφέρονται για δημιουργία, καταστροφή, κλείδωμα και ξεκλείδωμα των mutexes. Επίσης περιλαμβάνουν κάποιες συναρτήσεις ιδιοτήτων οι οποίες ορίζουν ή τροποποιούν τις ιδιότητες των mutexes.
 3. **Condition variables:** Ρουτίνες μεταβλητών συνθηκών οι οποίες διαχειρίζονται τις επικοινωνίες μεταξύ νημάτων ο οποίες μοιράζονται mutexes. Βασίζονται στις συνθήκες προγραμματισμού του προγραμματιστή. Αυτή ο ομάδα περιλαμβάνει συναρτήσεις για δημιουργία, καταστροφή, αναμονή και σηματοποίηση βασισμένα σε συγκεκριμένες τιμές μεταβλητών. Επίσης περιλαμβάνουν συναρτήσεις για να ορίσουν/εξετάσουν τις παραμέτρους των μεταβλητών.
 4. **Συγχρονισμός:** Ρουτίνες που διαχειρίζονται τα κλειδώματα και τα φράγματα.
- Συμβάσεις ονομάτων: όλα τα χειριστήρια στην βιβλιοθήκη των νημάτων αρχίζουν με **pthread_**. Μερικά παραδείγματα φαίνονται παρακάτω.

Πρόθεμα Ρουτίνας	Ομάδα Ρουτίνας
pthread_	Νήματα και διάφορες υπορουτίνες
pthread_attr_	Αντικείμενα ιδιοτήτων νημάτων
pthread_mutex_	Mutexes
pthread_mutexattr_	Αντικείμενα ιδιοτήτων mutex.
pthread_cond_	Μεταβλητές συνθηκών (condition variables)
pthread_condattr_	Αντικείμενα ιδιοτήτων μεταβλητών συνθηκών
pthread_key_	Κλειδιά δεδομένων νημάτων (thread-specific data keys)
pthread_rwlock_	Κλειδώματα ανάγνωσης/εγγραφής
pthread_barrier_	Φράγματα συγχρονισμού

- Το σκεπτικό των αδιαφανών αντικειμένων εισχωρεί στον σχεδιασμό του API. Οι βασικές κλήσεις δημιουργούν ή τροποποιούν αυτά τα αντικείμενα - τα αντικείμενα μπορούν να τροποποιηθούν από κλήσεις σε συναρτήσεις ιδιοτήτων, οι οποίες διαχειρίζονται τις αδιαφανείς παραμέτρους.
- Το API των Pthreads περιέχει περίπου 100 ρουτίνες. Αυτό το κείμενο εστιάζει σε ένα υποσύνολο - συγκεκριμένα, αυτές οι οποίες είναι πιο πιθανό να είναι άμεσα χρήσιμες σε νέους προγραμματιστές των Pthreads.
- Για φορητότητα, το αρχείο κεφαλής pthread.h θα πρέπει να περιλαμβάνεται σε κάθε αρχείο που χρησιμοποιεί την βιβλιοθήκη των Pthreads.
- Το υπάρχον πρότυπο του POSIX ορίζεται μόνο για την γλώσσα C.
- Υπάρχουν αρκετά εξαιρετα βιβλία σχετικά με τα Pthreads. Μερικά από αυτά υπάρχουν στις [Αναφορές](#) αυτού του οδηγού.

Μεταγλώττιση προγραμμάτων με Νήματα

- Μερικά παραδείγματα εντολών μεταγλώττισης που χρησιμοποιούνται για κώδικες με Pthreads:

Μεταγλωττιστής / Πλατφόρμα	Εντολή	Περιγραφή
INTEL Linux	icc -pthread	C
	icpc -pthread	C++
GNU Linux	gcc -pthread	GNU C
	g++ -pthread	GNU C++

Διαχείριση Νημάτων

Δημιουργία και Τερματισμός Νημάτων

► Ρουτίνες:

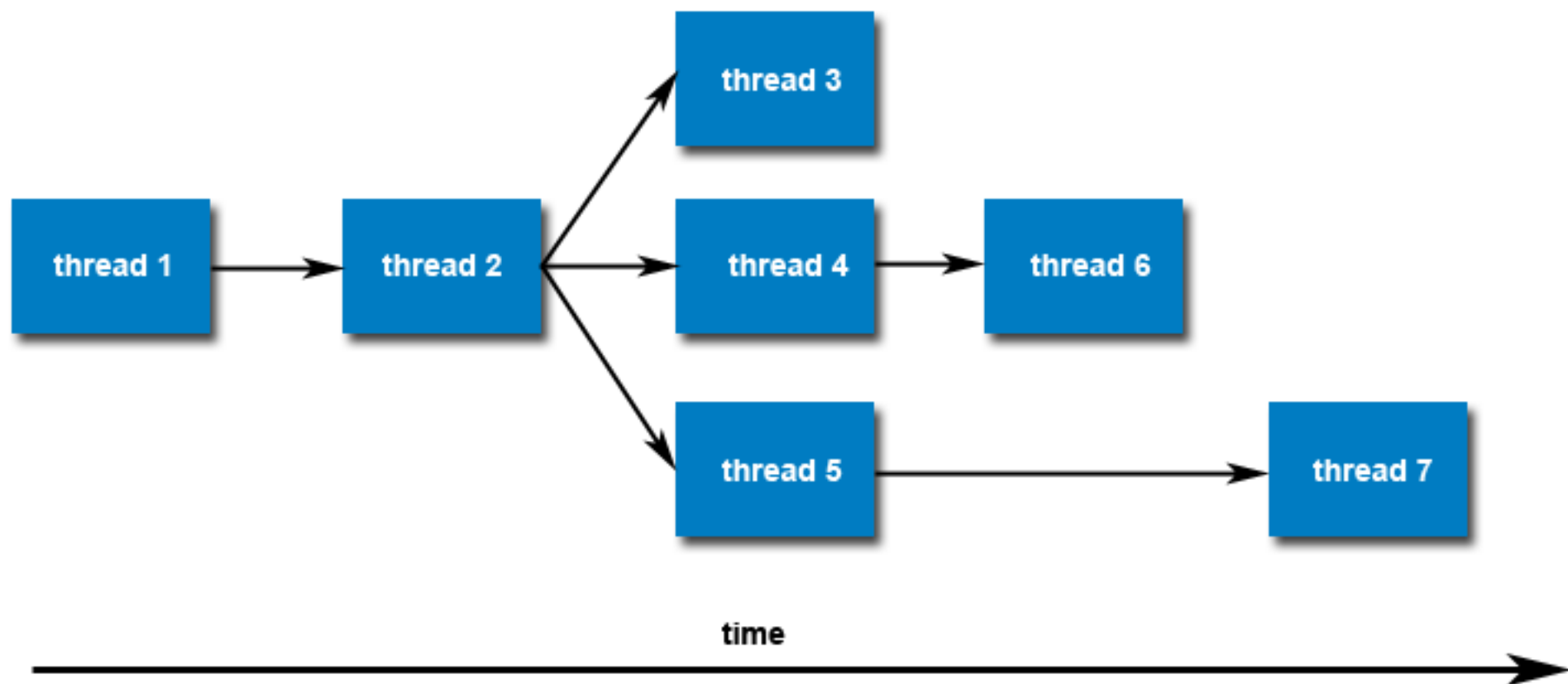
```
pthread_create (thread,attr,start_routine,arg)

pthread_exit (status)

pthread_attr_init (attr)
```

► Δημιουργία Νημάτων:

- Αρχικά, το κύριο πρόγραμμα `-main()` - αποτελείται από ένα μόνο προκαθορισμένο νήμα. Όλα τα υπόλοιπα νήματα πρέπει να δημιουργηθούν ξεχωριστά από τον προγραμματιστή.
- Η συνάρτηση `pthread_create` δημιουργεί ένα καινούριο νήμα και το κάνει εκτελέσιμο. Αυτή η ρουτίνα μπορεί να κληθεί όσες φορές θέλει ο προγραμματιστής μέσα στον κωδικά του.
- Παράμετροι της `pthread_create`:
 - `thread`: Ένα μοναδικό αναγνωριστικό (ID) για το καινούριο νήμα που επιστρέφεται από την υπορουτίνα. Είναι τύπου `pthread_t`.
 - `attr`: Ένα αντικείμενο ιδιοτήτων το οποίο μπορεί να χρησιμοποιηθεί για να ορίσει ιδιότητες των νημάτων. Ένα τέτοιο αντικείμενο είτε δημιουργείται από το χρήστη, πριν τη δημιουργία του νήματος, ή μπορεί να χρησιμοποιηθεί το `NULL` για τις προκαθορισμένες τιμές ιδιοτήτων.
 - `start_routine`: Η παράμετρος `start_routine` είναι η διεύθυνση της ρουτίνας που θα εκτελέσει το νήμα. Είναι δείκτης τύπου `void`.
 - `arg`: Η παράμετρος `arg` είναι η διεύθυνση της δομής των παραμέτρων για τη ρουτίνα `start_routine`. Είναι δείκτης τύπου `void`.
- Ο μέγιστος αριθμός νημάτων που μπορεί να δημιουργηθεί από μια διεργασία, εξαρτάται από την υλοποίηση.
- Εφόσον δημιουργηθούν τα νήματα, μπορούν να δημιουργήσουν άλλα νήματα.



Ερώτηση: Αφού δημιουργηθεί το νήμα, πώς μπορούμε να ξέρουμε πότε θα προγραμματιστεί να τρέξει από το λειτουργικό σύστημα;

► Ιδιότητες Νημάτων:

- Εξ ορισμού, ένα νήμα δημιουργείται με συγκεκριμένες ιδιότητες. Όρισμένες από αυτές τις ιδιότητες μπορούν να τροποποιηθούν από τον προγραμματιστή διαμέσου του αντικειμένου ιδιοτήτων του νήματος.
- Η συνάρτηση `pthread_attr_init` και η `pthread_attr_destroy` χρησιμοποιούνται για να αρχικοποιήσουν/καταστρέψουν το αντικείμενο ιδιοτήτων του νήματος.
- Άλλες ρουτίνες μετά χρησιμοποιούνται για να εξετάσουν/ορίσουν συγκεκριμένες ιδιότητες στο αντικείμενο ιδιοτήτων του νήματος.
- Μερικές από αυτές τις ιδιότητες θα συζητηθούν αργότερα.

► Τερματισμός Νημάτων:

- Υπάρχουν πολλοί τρόποι με τους οποίους μπορεί ένα Pthread να τερματιστεί:
 - Το νήμα επιστρέφει από την αρχική ρουτίνα (η κύρια ρουτίνα για το αρχικό νήμα).
 - Το νήμα καλεί την υπορουτίνα `pthread_exit` (εξετάζεται παρακάτω).
 - Το νήμα ακυρώνεται από ένα άλλο νήμα μέσω της ρουτίνας `pthread_cancel` (δεν καλύπτεται εδώ).
 - Τερματίζει ολόκληρη η διεργασία εξαιτίας κλήσης σε μία από τις υπορουτίνες `exec` ή `exit`.
- Η συνάρτηση `pthread_exit` χρησιμοποιείται για να τερματιστεί ένα νήμα. Τυπικά, η ρουτίνα `pthread_exit()` καλείται αφού ένα νήμα έχει ολοκληρώσει την δουλειά του και δεν χρειάζεται να υπάρχει άλλο.
- Αν η συνάρτηση `main()` τερματίσει πριν τα νήματά της, και τερματίσει με την `pthread_exit()`, τα άλλα νήμα τα θα συνεχίσουν να εκτελούνται. Διαφορετικά, θα τερματιστούν αυτόματα όταν τερματιστεί και η `main()`.
- Ο προγραμματιστής μπορεί να προσδιορίσει μια κατάσταση τερματισμού, η οποία αποθηκεύεται σαν ένας κενός δείκτης για οποιοδήποτε νήμα μπορεί να ενωθεί με το νήμα που το καλεί.
- Καθαρισμός: η ρουτίνα `pthread_exit()` δεν κλείνει αρχεία, όποια αρχεία ανοίχτηκαν μέσα στο νήμα θα μείνουν ανοιχτά μέχρι να τερματιστεί το νήμα.
- Συζήτηση: Στις ρουτίνες που τερματίζουν φυσιολογικά, μπορεί συχνά να κληθεί η `pthread_exit()` και να τερματιστεί - εκτός κ αν, φυσικά, θέλουν να επιστρέψουν έναν κώδικα. Ωστόσο, στην `main()`, υπάρχει ένα σίγουρο πρόβλημα αν η `main()` ολοκληρώσει πριν ολοκληρώσουν τα νήματά της. Αν δεν κληθεί η `pthread_exit()`, όταν ολοκληρώσει η `main()`, η διεργασία (και όλα τα νήματα) θα τερματιστούν. Καλώντας την `pthread_exit()` στην `main()`, η διεργασία και όλα τα νήματά της θα κρατηθούν ζωντανά ακόμα κ αν όλος ο κώδικας της `main()` έχει εκτελεστεί.

Παράδειγμα: Δημιουργία και Τερματισμός Pthread

- Αυτό το απλό παράδειγμα κώδικα δημιουργεί 5 νήματα με την ρουτίνα `pthread_create()`. Κάθε νήμα εκτυπώνει ένα μήνυμα "Hello

World!" , και έπειτα τερματίζει με μια κλήση στην ρουτίνα pthread_exit().

Παράδειγμα - Δημιουργία και Τερματισμός Pthread

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS      5

void *PrintHello(void *threadid)
{
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
    pthread_t  threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0; t<NUM_THREADS; t++){
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Διαχείριση Νημάτων

Πέρασμα παραμέτρων στα Νήματα

- Η ρουτίνα pthread_create() επιτρέπει στον προγραμματιστή να περάσει μία παράμετρο στην ρουτίνα εκκίνησης του νήματος. Σε περιπτώσεις όπου πρέπει να περαστούν πολλές παράμετροι, αυτός ο περιορισμός εύκολα καταρρίπτεται δημιουργώντας μια δομή που περιέχει όλες τις παραμέτρους και μετά περνώντας έναν δείκτη σε αυτή τη δομή κατά την ρουτίνα εκκίνησης του νήματος.
- Όλες οι παράμετροι πρέπει να περαστούν κατ' αναφορά με τον τύπο (void *).



Ερώτηση: πως μπορούμε να περάσουμε με ασφαλή τρόπο δεδομένα σε νέα νήματα, δεδομένης της περιγραφής τους, την εκκίνησή τους και τον χρονοπρογραμματισμό;

Παράδειγμα 1 - Thread Argument Passing

Αυτό το τμήμα κώδικα παρουσιάζει τον τρόπο με τον οποίο μπορεί να περαστεί ένας ακέραιος ως παράμετρος σε κάθε νήμα. Το νήμα που καλεί χρησιμοποιεί μία μοναδική δομή δεδομένων για κάθε νήμα, εξασφαλίζοντας ότι κάθε παράμετρος κάθε νήματος παραμένει ανέπαφη σε όλη τη διάρκεια εκτέλεσης του προγράμματος.

```
long *taskids[NUM_THREADS];

for(t=0; t<NUM_THREADS; t++)
{
    taskids[t] = (long *) malloc(sizeof(long));
    *taskids[t] = t;
    printf("Creating thread %ld\n", t);
    rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
    ...
}
```

Παράδειγμα 2 - Thread Argument Passing

Αυτό το παράδειγμα δείχνει το πέρασμα πολλών παραμέτρων διαμέσου μιας δομής. Κάθε νήμα λαμβάνει ένα μοναδικό στιγμότυπο αυτής της δομής.

```
struct thread_data{
    int  thread_id;
    int  sum;
    char *message;
};

struct thread_data thread_data_array[NUM_THREADS];
```



```
void *PrintHello(void *threadarg)
{
    struct thread_data *my_data;
    ...
    my_data = (struct thread_data *) threadarg;
    taskid = my_data->thread_id;
    sum = my_data->sum;
    hello_msg = my_data->message;
    ...
}

int main (int argc, char *argv[])
{
    ...
    thread_data_array[t].thread_id = t;
    thread_data_array[t].sum = sum;
    thread_data_array[t].message = messages[t];
    rc = pthread_create(&threads[t], NULL, PrintHello,
        (void *) &thread_data_array[t]);
    ...
}
```

SourceOutput

Παράδειγμα 3 - Thread Argument Passing (Incorrect)

Αυτό το παράδειγμα περνάει με λάθος τρόπο παραμέτρους. Περνάει την διεύθυνση της μεταβλητής t, η οποία βρίσκεται σε χώρο της διαμοιραζόμενης μνήμης και είναι ορατή σε όλα τα νήματα. Καθώς εκτελείται η πρώτη επαν'αληψη, αλλάζει η τοποθεσία αυτής της διεύθυνσης, πιθανόν προτού τα άλλα νήματα μπορέσουν να την προσπελάσουν.

```
int rc;
long t;

for(t=0; t<NUM_THREADS; t++)
{
    printf("Creating thread %ld\n", t);
    rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &t);
    ...
}
```

SourceOutput

Διαχείριση Νημάτων

Ένωση και Απόσπαση Νημάτων

► Ρουτίνες:

```
pthread_join (threadid,status)

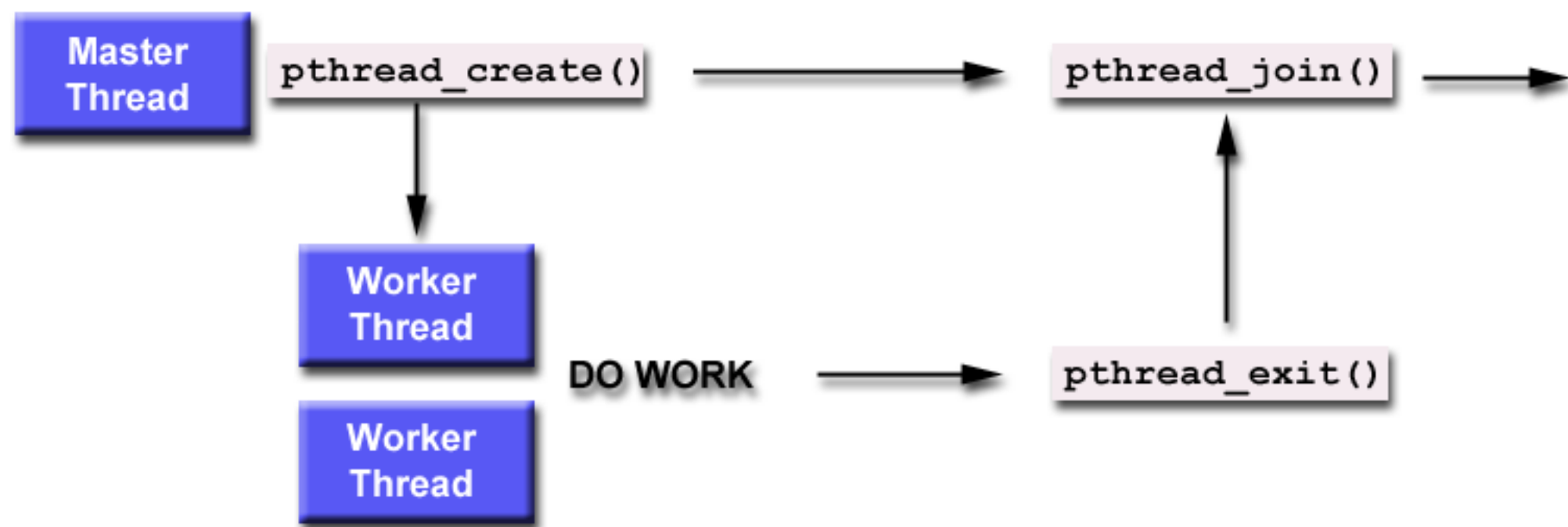
pthread_detach (threadid,status)

pthread_attr_setdetachstate (attr,detachstate)

pthread_attr_getdetachstate (attr,detachstate)
```

► Ένωση:

- Η "ένωση" είναι ένας τρόπος για να επιτευχθεί συγχρονισμός μεταξύ νημάτων. Για παράδειγμα:



- Η ρουτίνα pthread_join() αναστέλει το νήμα συντονιστή μέχρι να τερματιστούν τα νήματα εργαζόμενοι.

- Ο προγραμματιστής μπορεί να αποκτήσει την τελική κατάσταση του νήματος που τερματίζει αν έχει προσδιοριστεί στην κλήση της ρουτίνας pthread_exit().
- Ένα νήμα μπορεί να εκτελέσει μόνο μία ένωση με την κλήση της pthread_join(). Είναι λογικό λάθος η προσπάθεια πολλαπλής ένωσης στο ίδιο νήμα.
- Δύο άλλοι τρόποι συγχρονισμού, mutexes και μεταβλητές συνθηκών, θα συζητηθούν αργότερα.

Ένωση ή Όχι;

- Όταν δημιουργείται ένα νήμα, μία ιδιότητα προσδιορίζει ένα μπορεί να ενωθεί ή όχι. Μόνο τα νήματα που δημιουργούνται αρχικά για ένωση (joinable) μπορούν να ενωθούν. Αν ένα νήμα δημιουργηθεί σαν αποσπασμένο (detached), δεν μπορεί ποτέ να ενωθεί.
- Η τελική έκδοση του προτύπου POSIX προσδιορίζει ότι τα νήματα πρέπει να δημιουργούνται έτοιμα για ένωση (joinable).
- Για να δημιουργηθεί ένα νήμα ικανό για ένωση ή όχι, χρησιμοποιείται η παράμετρος attr στην συνάρτηση pthread_create(). Τα 4 τυπικά βήματα είναι:
 1. Ορισμός μιας μεταβλητής τύπου pthread_attr_t για ένα νήμα.
 2. Αρχικοποίηση της παραμέτρου με την συνάρτηση pthread_attr_init()
 3. Ορισμός της κατάστασης ένωσης της παραμέτρου με την συνάρτηση pthread_attr_setdetachstate()
 4. Μετά το τέλος, γίνεται αποδέσμευση των πηγών των βιβλιοθηκών από την παράμετρο με την συνάρτηση pthread_attr_destroy()

Απόσπαση:

- Η ρουτίνα pthread_detach() μπορεί να χρησιμοποιηθεί για την απόσπαση ενός νήματος ακόμα και αν δημιουργήθηκε ως ικανό να ενωθεί (joinable).
- Δεν υπάρχει ρουτίνα εκ νέου μετατροπής.

Προτάσεις:

- Αν ένα νήμα χρειάζεται ένωση, ας δημιουργηθεί ικανό προς ένωση. Αυτό παρέχει φορητότητα καθώς δεν δημιουργούν όλες οι υλοποιήσεις τα νήματα ικανά προς ένωση αρχικά.
- Αν είναι γνωστό από πριν αν ένα νήμα δεν θα χρειαστεί να ενωθεί ποτέ με κάποιο άλλο νήμα, ας δημιουργηθεί ως ανεξάρτητο. Μερικοί πόροι συστήματος μπορεί να είναι ικανές να αποδεσμευτούν.

Παράδειγμα: Ένωση Pthread



Example Code - Pthread Joining

Αυτό το παράδειγμα παρουσιάζει έναν τρόπο "αναμονής" ολοκλήρωσης νημάτων με την χρήση της ρουτίνας ένωσης των Pthreads. Εφόσον μερικές υλοποιήσεις δεν δημιουργούν τα νήματα ικανά προς ένωση από μόνα τους, τα νήματα ρητά δηλώνονται ικανά προς ένωση ώστε να ενωθούν αργότερα.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS      4

void *BusyWork(void *t)
{
    int i;
    long tid;
    double result=0.0;
    tid = (long)t;
    printf("Thread %ld starting...\n",tid);
    for (i=0; i<1000000; i++)
    {
        result = result + sin(i) * tan(i);
    }
    printf("Thread %ld done. Result = %e\n",tid, result);
    pthread_exit((void*) t);
}

int main (int argc, char *argv[])
{
    pthread_t thread[NUM_THREADS];
    pthread_attr_t attr;
    int rc;
    long t;
    void *status;

    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for(t=0; t<NUM_THREADS; t++) {
        printf("Main: creating thread %ld\n", t);
        rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create()
                is %d\n", rc);
            exit(-1);
        }
    }

    /* Free attribute and wait for the other threads */
    pthread_attr_destroy(&attr);
```

```
for(t=0; t<NUM_THREADS; t++) {
    rc = pthread_join(thread[t], &status);
    if (rc) {
        printf("ERROR; return code from pthread_join()
               is %d\n", rc);
        exit(-1);
    }
    printf("Main: completed join with thread %ld having a status
           of %ld\n",t,(long)status);
}

printf("Main: program completed. Exiting.\n");
pthread_exit(NULL);
}
```

Source

Output

Διαχείριση Νημάτων

Διαχείριση Στοίβας

► Ρουτίνες:

```
pthread_attr_getstacksize (attr, stacksize)

pthread_attr_setstacksize (attr, stacksize)

pthread_attr_getstackaddr (attr, stackaddr)

pthread_attr_setstackaddr (attr, stackaddr)
```

► Αποτροπή προβλημάτων με τις Στοίβες:


- Το πρότυπο POSIX δεν περιορίζει το μέγεθος μιας στοίβας ενός νήματος. Εξαρτάται από την υλοποίηση και ποικίλλει.
- Υπέρβαση του προκαθορισμένου ορίου της στοίβας γίνεται συχνά, με τα συνήθη αποτελέσματα: τερματισμός προγράμματης και/ή αλλιωμένα δεδομένα.
- Ασφαλή και φορητά προγράμματα δεν εξαρτώνται από το μέγεθος της στοίβας, αλλά ρητά καταλαμβάνουν χώρο από πριν για κάθε νήμα με την χρήση της ρουτίνας pthread_attr_setstacksize.
- Οι ρουτίνες pthread_attr_getstackaddr και pthread_attr_setstackaddr μπορούν να χρησιμοποιηθούν από εφαρμογές σε περιβάλλον όπου ο χώρος για τις στοίβες πρέπει να τοποθετηθεί σε συγκεκριμένο χώρο στην μνήμη.

► Μερικά πρακτικά παραδείγματα συστημάτων:

- Το προκαθορισμένο μέγεθος της στοίβας ποικίλει πολύ. Το μέγιστο μέγεθος που μπορεί να έχει επίσης ποικίλει, και μπορεί να εξαρτάται από τον αριθμό των νημάτων ανά κόμβο.

Node Architecture	#CPUs	Memory (GB)	Default Size (bytes)
AMD Opteron	8	16	2,097,152
Intel IA64	4	8	33,554,432
Intel IA32	2	4	2,097,152

Παράδειγμα: Διαχείριση Στοίβας

 **Example Code - Stack Management**

Αυτό το παράδειγμα δείχνει πως να εξεταστεί και να οριστεί το μέγεθος μας στοίβας ενός νήματος.

```
#include <pthread.h>
#include <stdio.h>
#define NTHREADS 4
#define N 1000
#define MEGEXTRA 1000000

pthread_attr_t attr;

void *dowork(void *threadid)
{
    double A[N][N];
    int i,j;
    long tid;
    size_t mystacksize;

    tid = (long)threadid;
    pthread_attr_getstacksize (&attr, &mystacksize);
    printf("Thread %ld: stack size = %li bytes \n", tid, mystacksize);
```



```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        A[i][j] = ((i*j)/3.452) + (N-i);
pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NTHREADS];
    size_t stacksize;
    int rc;
    long t;

    pthread_attr_init(&attr);
    pthread_attr_getstacksize (&attr, &stacksize);
    printf("Default stack size = %li\n", stacksize);
    stacksize = sizeof(double)*N*N+MEGEXTRA;
    printf("Amount of stack needed per thread = %li\n",stacksize);
    pthread_attr_setstacksize (&attr, stacksize);
    printf("Creating threads with stack size = %li bytes\n",stacksize);
    for(t=0; t<NTHREADS; t++){
        rc = pthread_create(&threads[t], &attr, dowork, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    printf("Created %ld threads.\n", t);
    pthread_exit(NULL);
}
```

Διαχείριση Νημάτων

Διάφορες Ρουτίνες

```
pthread_self ()

pthread_equal (thread1,thread2)
```

- Η ρουτίνα pthread_self επιστρέφει το μοναδικό ID του νήματος που δίνει το σύστημα στο νήμα που το καλεί.
- Η συνάρτηση pthread_equal συγκρίνει τα ID 2 νημάτων. Αν διαφέρουν επιστρέφει 0, αλλιώς επιστρέφει μια τιμή διάφορη του μηδενός.
- Να σημειωθεί ότι για τις δύο αυτές ρουτίνες, τα νήματα είναι αδιάφανα και δεν μπορούν εύκολα να παρακολουθηθούν. Επειδή τα ID των νημάτων είναι αδιαφανή αντικείμενα, ο τελεστής στην γλώσσα C == δεν πρέπει να χρησιμοποιείται για την σύγκριση των ID των νημάτων ουτε ενός ID και μιας άλλης τιμής.

```
pthread_once (once_control, init_routine)
```

- Η συνάρτηση pthread_once εκτελεί την ρουτίνα init_routine ακριβώς μια φορά σε κάθε διεργασία. Η πρώτη κλήση σε αυτή τη ρουτίνα από οποιοδήποτε νήμα εκτελεί την ρουτίνα init_routine χωρίς παραμέτρους. Οποιαδήποτε κλήση σε αυτή αργότερα δεν θα έχει κανένα αποτέλεσμα.
- Η ρουτίνα init_routine είναι μια ρουτίνα αρχικοποίησης.
- Η παράμετρος once_control είναι μια δομή ελέγχου συγχρονισμού που απαιτεί την αρχικοποίηση πριν καλέσει την pthread_once. Για παράδειγμα:

```
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

Μεταβλητές Αμοιβαίου Αποκλεισμού - Mutex

Επισκόπηση

- Mutex είναι σύντημηση του "mutual exclusion" (αμοιβαίος αποκλεισμός). Οι μεταβλητές mutex είναι μια από τις βασικές μεθόδους για το συγχρονισμό νημάτων και για τη προστασία διαμοιραζόμενων μεταβλητών όταν υπάρχουν πολλαπλές εγγραφές.
- Μια μεταβλητή mutex λειτουργεί σαν "κλειδωμα" που προστατεύει τη πρόσβαση σε ένα διμοιραζόμενο πόρο (θέση μνήμης). Η βασική ιδέα είναι οτι σε κάθε χρονική στιγμή μόνο ένα νήμα μπορεί να κατέχει -άρα να κλειδώσει- ένα mutex. Αν τα υπολοιπα νήματα θέλουν να καταλάβουν και να κλειδώσουν το ίδιο mutex, πρέπει να περιμένουν μέχρι το νήμα που ήδη το κατέχει να το ξεκλειδώσει, δηλαδή να το απελευθερώσει. Στη συνέχεια και πάλι μόνο ένα από τα νήματα που αναμένουν στο mutex θα το καταλάβει. Έτσι τα νήματα θα καταλάβουν το mutex ένα-ένα.
- Οι μεταβλητές mutex επίσης αποτρέπουν συνθήκες ανταγωνισμού ("race" conditions). Ακολουθεί ένα παράδειγμα τέτοιας συνθήκης:

Thread 1	Thread 2	Balance
Read balance: \$1000		\$1000
	Read balance: \$1000	\$1000

	Deposit \$200	\$1000
Deposit \$200		\$1000
Update balance \$1000+\$200		\$1200
	Update balance \$1000+\$200	\$1200

- Στο παραπάνω παράδειγμα, ένα mutex πρέπει να κλειδώσει τη διαδικασία ανάγνωσης και τροποποίησης της διαμοιραζόμενης μεταβλητής "balance", έτσι ώστε μια συναλλαγή να είναι ενιαία και αδιαίρετη διαδικασία.
- Το σύνολο των εντολών ενός νήματος που εκτελούν την ενημέρωση μιας διαμοιραζόμενης περιοχής μνήμης λέγεται και "κρίσιμη περιοχή" ("critical section"). Κάθε νήμα, ακριβώς πριν εισέλθει στη κρίσιμη περιοχή, πρέπει να καταλάβει ένα mutex. Αν το mutex είναι κατηλειμμένο τότε σημαίνει οτι κάποιο άλλο νήμα έχει εισέλθει στη κρίσιμη περιοχή, άρα θα πρέπει να περιμένει. Μόλις το νήμα καταφέρει να καταλάβει το mutex εισέρχεται στη κρίσιμη περιοχή, εκτελεί την ενημέρωση και βγαίνοντας ελευθερώνει το mutex.
- Μια τυπική χρήση ενός mutex είναι η παρακάτω:
 - Δημιουργία και αρχικοποίηση μιας μεταβλητής mutex
 - Τα νήματα προσπαθούν να καταλάβουν το mutex
 - Ένα νήμα καταλαμβάνει και κλειδώνει το mutex
 - Το νήμα αυτό εκτελεί τις λειτουργίες της κρίσιμης περιοχής
 - Το νήμα ξεκλειδώνει (ελευθερώνει) το mutex
 - Άλλο νήμα εκτελεί τα αντίστοιχα βήματα..
 - Τελικά η μεταβλητή mutex καταστρέφεται
- Όταν πολλά νήματα ανταγωνίζονται για ένα mutex, τα νήματα που αποτυγχάνουν αναστέλλονται σε εκείνο το σημείο εκτέλεσης - υπάρχει και μια μη-ανασταλτική έκδοση της λειτουργίας, μια κλήση "trylock" αντί "lock".
- Ο προγραμματιστής είναι υπεύθυνος για την ορθή χρήση των μεταβλητών mutex. Έστι, για παράδειγμα πρέπει να έλεχει για πιθανά αδιέξοδα (livelock ή starvation), και γενικότερα για την συνεπή χρήση των μεταβλητών mutex.

Μεταβλητές Αμοιβαίου Αποκλεισμού - Mutex

Δημιουργία και Καταστροφή Mutex

► Ρουτίνες:

```
pthread_mutex_init (mutex,attr)

pthread_mutex_destroy (mutex)

pthread_mutexattr_init (attr)

pthread_mutexattr_destroy (attr)
```

► Χρήση:

- Οι μεταβλητές mutex δηλώνονται με τύπο pthread_mutex_t, και πρέπει να αρχικοποιηθούν πριν τη χρήση τους. Υπάρχουν δυο μέθοδοι αρχικοποίησης:
 1. Στατική, κατά τη δήλωση. Για παράδειγμα:

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
```
 2. Δυναμικά, με τη ρουτίνα pthread_mutex_init(). Αυτή η μέθοδος επιτρέπει τον ορισμό επιπλέον ιδιοτήτων στη ματαβλητή mutex, μέσω του αντικειμένου attr.

Το mutex είναι αρχικά ξεκλειδωτο.

- Το αντικείμενο attr ορίζει ιδιότητες του mutex, και πρέπει να είναι το τύπου pthread_mutexattr_t (μπορεί να οριστεί ως NULL για να λάβει προκαθορισμένες τιμές). Το πρότυπο Pthreads ορίζει τρεις ιδιότητες mutex:
 - Πρωτόκολλο (Protocol): Ορίζει το πρωτόκολλο που χρησιμοποιείται για την αποφυγή αναστροφής προτεραιότητας (priority inversion) σε mutex.
 - Οροφή Προτεραιότητας (Prioceiling): Ορίζει την οροφή προτεραιότητας του mutex.
 - Διαμοιρασμός μεταξύ Διεργασιών (Process-Shared): Ορίζει το διαμοιρασμό του mutex μεταξύ διεργασιών.

Οι ιδιότητες των mutexes δεν είναι διαθέσιμες σε όλες τις υλοποιήσεις.

- Η δημιουργία και καταστροφή των αντικειμένων ιδιοτήτων mutexes επιτυγχάνεται με τις ρουτίνες pthread_mutexattr_init() και pthread_mutexattr_destroy() αντίστοιχα.
- Η ρουτίνα pthread_mutex_destroy() χρησιμοποιείται για την καταστροφή του mutex όταν τελειώσει η χρήση του.

Μεταβλητές Αμοιβαίου Αποκλεισμού - Mutex

Κλείδωμα και Ξεκλείδωμα Mutex

► Ρουτίνες:

```
pthread_mutex_lock (mutex)

pthread_mutex_trylock (mutex)

pthread_mutex_unlock (mutex)
```

► Χρήση:

- Η ρουτίνα pthread_mutex_lock() χρησιμοποιείται από ένα νήμα για να κλειδώσει μια συγκεκριμένη μεταβλητή mutex. Αν η μεταβλητή είναι ήδη κλειδωμένη από άλλο νήμα, αυτή η κλήση θα αναστείλει την εκτέλεση του νήματος που καλεί μέχρι να ελευθερωθεί η μεταβλητή.
- Η ρουτίνα pthread_mutex_trylock() προσπαθεί να κλειδώσει μια μεταβλητή mutex. Ωστόσο, αν η μεταβλητή είναι ήδη κλειδωμένη, θα επιστρέψει έναν κωδικό λάθους "απασχολημένη". Αυτή η ρουτίνα μπορεί να είναι χρήσιμη στην αποτροπή αναστολής εκτέλεσης.
- Η ρουτίνα pthread_mutex_unlock() ξεκλειδώνει μια μεταβλητή mutex όταν κληθεί από το νήμα που την έχει ήδη κλειδωμένη (δηλαδή την κατέχει). Η κλήση αυτής της ρουτίνας απαιτείται αφού το νήμα έχει ολοκληρώσει την δουλειά του με τα προστατευμένα δεδομένα έτσι ώστε άλλα νήματα να μπορούν να χρησιμοποιήσουν την mutex. Σφάλμα θα προκύψει εάν:
 - Η μεταβλητή mutex είναι ήδη ελεύθερη
 - Η μεταβλητή mutex ανήκει σε άλλο νήμα
- Δεν υπάρχει κάτι το "μαγικό" με τις μεταβλητές mutex... στην πραγματικότητα είναι σαν μία "συμφωνία κυρίων" μεταξύ των νημάτων. Ο προγραμματιστής είναι υπεύθυνος ώστε κάθε νήμα να κλειδώνει και να ελευθερώνει την κάθε μεταβλητή σωστά. Το επόμενο σενάριο παρουσιάζει ένα λογικό λάθος:

Thread 1	Thread 2	Thread 3
Lock	Lock	
A = 2	A = A+1	A = A*B
Unlock	Unlock	



Ερώτηση: Όταν περισσότερα από ένα νήματα περιμένουν για μια κλειδωμένη μεταβλητή mutex, ποιο νήμα θα την πάρει αφού ελευθερωθεί;

Παράδειγμα: Χρήση Mutex

Example Code - Using Mutexes

Αυτό το παράδειγμα δείχνει την χρήση των μεταβλητών mutex σε ένα πρόγραμμα με νήματα. Τα κύρια δεδομένα είναι διαθέσιμα σε όλα τα νήματα μέσω μιας δομής. Κάθε νήμα δουλεύει σε διαφορετικό τμήμα των δεδομένων. Ο κύριο νήμα περιμένει τα υπόλοιπα νήματα να τελειώσουν τους υπολογισμούς τους και μετά εκτυπώνει το αποτέλεσμα.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

/*
The following structure contains the necessary information
to allow the function "dotprod" to access its input data and
place its output into the structure.
*/

typedef struct
{
    double      *a;
    double      *b;
    double      sum;
    int         vecLen;
} DOTDATA;

/* Define globally accessible variables and a mutex */

#define NUMTHRDS 4
#define VECLN 100
DOTDATA dotstr;
pthread_t callThd[NUMTHRDS];
pthread_mutex_t mutexsum;

/*
The function dotprod is activated when the thread is created.
All input to this routine is obtained from a structure
of type DOTDATA and all output from this function is written into
this structure. The benefit of this approach is apparent for the
multi-threaded program: when a thread is created we pass a single
argument to the activated function - typically this argument
is a thread number. All the other information required by the
function is accessed from the globally accessible structure.
*/

void *dotprod(void *arg)
{
```



```

/* Define and use local variables for convenience */

int i, start, end, len ;
long offset;
double mysum, *x, *y;
offset = (long)arg;

len = dotstr.vecLEN;
start = offset*len;
end   = start + len;
x = dotstr.a;
y = dotstr.b;

/*
Perform the dot product and assign result
to the appropriate variable in the structure.
*/

mysum = 0;
for (i=start; i<end ; i++)
{
    mysum += (x[i] * y[i]);
}

/*
Lock a mutex prior to updating the value in the shared
structure, and unlock it upon updating.
*/
pthread_mutex_lock (&mutexsum);
dotstr.sum += mysum;
pthread_mutex_unlock (&mutexsum);

pthread_exit((void*) 0);
}

/*
The main program creates threads which do all the work and then
print out result upon completion. Before creating the threads,
the input data is created. Since all threads update a shared structure,
we need a mutex for mutual exclusion. The main thread needs to wait for
all threads to complete, it waits for each one of the threads. We specify
a thread attribute value that allow the main thread to join with the
threads it creates. Note also that we free up handles when they are
no longer needed.
*/

int main (int argc, char *argv[])
{
    long i;
    double *a, *b;
    void *status;
    pthread_attr_t attr;

    /* Assign storage and initialize values */
    a = (double*) malloc (NUMTHRDS*VECLen*sizeof(double));
    b = (double*) malloc (NUMTHRDS*VECLen*sizeof(double));

    for (i=0; i<VECLen*NUMTHRDS; i++)
    {
        a[i]=1.0;
        b[i]=a[i];
    }

    dotstr.vecLEN = VECLen;
    dotstr.a = a;
    dotstr.b = b;
    dotstr.sum=0;

    pthread_mutex_init(&mutexsum, NULL);

    /* Create threads to perform the dotproduct */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for(i=0; i<NUMTHRDS; i++)
    {
        /*
        Each thread works on a different set of data.
        The offset is specified by 'i'. The size of
        the data for each thread is indicated by VECLen.
        */
        pthread_create(&callThd[i], &attr, dotprod, (void *)i);
    }

    pthread_attr_destroy(&attr);

    /* Wait on the other threads */
    for(i=0; i<NUMTHRDS; i++)
    {
        pthread_join(callThd[i], &status);
    }
}

```

```
/* After joining, print out the results and cleanup */
printf ("Sum =  %f \n", dotstr.sum);
free (a);
free (b);
pthread_mutex_destroy(&mutexsum);
pthread_exit(NULL);
}
```

 *Serial version*

 *Pthreads version*

Μεταβλητές Συνθηκών

Επισκόπηση

- Οι μεταβλητές συνθηκών προσφέρουν μια επιπρόσθετη μέθοδο συγχρονισμού νημάτων η οποία αποτρέπει τον προγραμματιστή από πιθανά λογικά σφάλματα στη χρήση mutexes. Ενώ οι mutexes υλοποιούν το συγχρονισμό ελέγχοντας την πρόσβαση στα διαμοιραζόμενα δεδομένα, οι μεταβλητές συνθηκών συγχρονίζουν βασισμένες στις τιμές των διαμοιραζόμενων δεδομένων.
- Χωρίς αυτές τις μεταβλητές, πολλαπλά νήματα που προσπαθούν να εισέλθουν σε μια κρίσιμη περιοχή, θα έπρεπε να ελέγχουν συνεχώς το mutex, ακόμη και αν δεν είναι η σειρά τους να εκτελεστούν. Αυτό μπορεί να καταναλώνει πολλούς πόρους καθώς το νήμα θα έπρεπε να είναι συνέχεια απασχολημένο. Μια μεταβλητή συνθήκης είναι ο τρόπος ελέγχου χωρίς αυτό το πρόβλημα.
- Μια μεταβλητή συνθήκης πάντα χρησιμοποιείται συνδυασμό με μία mutex.
- Μία σειρά εκτέλεσης για χρήση μεταβλητών συνθηκών φαίνεται παρακάτω.

Κύριο Νήμα

- Δήλωση και Αρχικοποίηση διαμοιραζόμενης μεταβλητής που απαιτεί συγχρονισμό
- Δήλωση και Αρχικοποίηση μιας μεταβλητής συνθηκών
- Δήλωση και Αρχικοποίηση μιας μεταβλητής mutex
- Δημιουργία νημάτων A και B για να εκτελέσουν την εργασία

Νήμα A

- Εκτέλεση εργασίας μέχρι να εμφανιστεί μια συγκεκριμένη συνθήκη (πχ ένας μετρητής φθάνει σε μια μέγιστη τιμή)
- Κλείδωμα mutex και έλεγχος της τιμής της διαμοιραζόμενης μεταβλητής
- Κλήση της pthread_cond_wait() η οποία ειδοποιεί το νήμα B ότι το νήμα A έχει μπει σε αναστολή και αναμένει σήμα από το νήμα B. Να σημειωθεί ότι η κλήση στην pthread_cond_wait() αυτόματα ξεκλειδώνει το mutex ώστε να μπορέσει να χρησιμοποιηθεί από το νήμα B.
- Όταν το νήμα A δεχτεί σήμα αφύπνισης από το νήμα B, το mutex κλειδώνετα αυτόματα (ατομική λειτουργία)".
- Ρητό ξεκλείδωμα της mutex
- Συνέχεια

Νήμα B

- Εκτέλεση εργασίας
- Κλείδωμα της mutex
- Αλλαγή της τιμής της διαμοιραζόμενης μεταβλητής που περιμένει το νήμα A.
- Έλεγχος της τιμής της διαμοιραζόμενης μεταβλητής από την οποία εξαρτάται η αναμονή του νήματος A. Αν ισχύει η επιθυμητή συνθήκη, το νήμα B ειδοποιεί το νήμα A με τη κλήση της pthread_cond_signal().
- Ξεκλείδωμα της mutex.
- Συνέχεια

Κύριο Νήμα

Συνέχεια

Μεταβλητές Συνθηκών

Δημιουργία και Καταστροφή Μεταβλητών Σηνηκών

► Ρουτίνες:

[pthread_cond_init](#) (condition,attr)

[pthread_cond_destroy](#) (condition)

[pthread_condattr_init](#) (attr)

[pthread_condattr_destroy](#) (attr)

► Χρηση:

- Οι μεταβλητές συνθηκών πρέπει να δηλωθούν με τον τύπο pthread_cond_t, και πρέπει να αρχικοποιηθούν προτού μπορέσουν να χρησιμοποιηθούν. Υπάρχουν 2 τρόποι αρχικοποίησης μιας τέτοιας μεταβλητής:

1. Στατικά, όταν δηλώνεται. Για παράδειγμα:

- ```
pthread_cond_t myconvar = PTHREAD_COND_INITIALIZER;
```
2. Δυναμικά με την ρουτίνα `pthread_cond_init()`. Το ID της μεταβλητής επιστρέφεται στο νήμα που την καλεί μέσω της παραμέτρου *condition*. Αυτή η μέθοδος επιτρέπει τον καθορισμό ιδιοτήτων στις μεταβλητές αυτές, *attr*.
- Το αντικείμενο *attr* χρησιμοποιείται για τον καθορισμό ιδιοτήτων στις μεταβλητές αυτές. Υπάρχει μόνο μία ιδιότητα η οποία ορίζεται από πριν: Διαμοιρασμός μεταξύ Διεργασιών, που επιτρέπει την μεταβλητή να είναι ορατή σε νήματα από άλλες διεργασίες. Το αντικείμενο ιδιοτήτων, αν χρησιμοποιείται, πρέπει να είναι τύπου `pthread_condattr_t` (μπορεί να είναι NULL).
- Να σημειωθεί πως δεν μπορούν όλες οι υλοποιήσεις να προσφέρουν την παράμετρο δμοιρασμό μεταξύ διεργασιών.
- Οι ρουτίνες `pthread_condattr_init()` και `pthread_condattr_destroy()` χρησιμοποιούνται για την δημιουργία και την καταστροφή αντικειμένων παραμέτρων.
  - Η ρουτίνα `pthread_cond_destroy()` χρησιμοποιείται για την αποδέσμευση μιας μεταβλητής που δεν χρειάζεται πλέον.

## Μεταβλητές Συνθηκών

### Αναμονή και Σηματοδότηση στις Μεταβλητές Συνθηκών

#### ► Ρουτίνες:


```
pthread_cond_wait (condition,mutex)

pthread_cond_signal (condition)

pthread_cond_broadcast (condition)
```

#### ► Χρήση:

- Η ρουτίνα `pthread_cond_wait()` αναστέλει το νήμα που την καλεί μέχρι να πραγματοποιηθεί η συγκεκριμένη *συνθήκη*. Αυτή η ρουτίνα πρέπει να κληθεί όσο είναι κλειδωμένη η μεταβλητή *mutex*, και θα την αποδεσμεύσει αυτόματα όσο περιμένει. Αφού ληφθεί το σήμα και ξυπνήσει το νήμα, η μεταβλητή *mutex* θα κλειδωθεί αυτόματα για χρήση από το νήμα. Ο προγραμματιστής είναι υπεύθυνος έπειτα για το ξεκλείδωμα της *mutex* αφού το νήμα τελειώσει μαζί της.
- Η ρουτίνα `pthread_cond_signal()` χρησιμοποιείται για την σηματοδότηση (ή αφύπνιση) άλλου νήματος το οποίο περιμένει την μεταβλητή συνθήκης. Πρέπει να κληθεί αφού κλειδωθεί η μεταβλητή *mutex*, και πρέπει να την ξεκλειδώσει ώστε να μπορεί να ολοκληρωθεί η εκτέλεση της `pthread_cond_wait()`.
- Η ρουτίνα `pthread_cond_broadcast()` πρέπει να χρησιμοποιείται αντί για την `pthread_cond_signal()` αν παραπάνω από ένα νήματα είναι σε αναμονή.
- Είναι λογικό λάθος η κλήση της `pthread_cond_signal()` πριν από την `pthread_cond_wait()`.

 Κατάλληλα κλειδώματα και ξεκλειδώματα των mutex είναι σημαντικά κατά την χρήση αυτών των ρουτινών. Για παράδειγμα:

- Αποτυχία κλειδώματος της mutex πριν την κλήση της `pthread_cond_wait()` μπορεί να προκαλέσει αποτυχία μπλοκαρίσματος.
- Αποτυχία ξεκλειδώματος της mutex μετά την κλήση της `pthread_cond_signal()` μπορεί να μην επιτρέψει την ολοκλήρωση της `pthread_cond_wait()` (αν παραμείνει μπλοκαρισμένη).

### Παράδειγμα: Χρήση Μεταβλητών Συνθηκών

#### Example Code - Using Condition Variables

Αυτό το παράδειγμα παρουσιάζει την χρήση πολλών ρουτινών για μεταβλητές συνθηκών σε Pthread. Η κύρια ρουτίνα δημιουργεί 3 νήματα. Τα 2 κάνουν κάποια δουλειά και ενημερώνουν μια μεταβλητή "count". Η τρίτη περιμένει μέχρι η μεταβλητή να πάρει μία ορισμένη τιμή.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 3
#define TCOUNT 10
#define COUNT_LIMIT 12

int count = 0;
int thread_ids[3] = {0,1,2};
pthread_mutex_t count_mutex;
pthread_cond_t count_threshold_cv;

void *inc_count(void *t)
{
 int i;
 long my_id = (long)t;

 for (i=0; i<TCOUNT; i++) {
 pthread_mutex_lock(&count_mutex);
 count++;

 /*
```



```

 Check the value of count and signal waiting thread when condition is
 reached. Note that this occurs while mutex is locked.
 */
 if (count == COUNT_LIMIT) {
 pthread_cond_signal(&count_threshold_cv);
 printf("inc_count(): thread %ld, count = %d Threshold reached.\n",
 my_id, count);
 }
 printf("inc_count(): thread %ld, count = %d, unlocking mutex\n",
 my_id, count);
 pthread_mutex_unlock(&count_mutex);

 /* Do some "work" so threads can alternate on mutex lock */
 sleep(1);
}
pthread_exit(NULL);
}

void *watch_count(void *t)
{
 long my_id = (long)t;

 printf("Starting watch_count(): thread %ld\n", my_id);

 /*
 Lock mutex and wait for signal. Note that the pthread_cond_wait
 routine will automatically and atomically unlock mutex while it waits.
 Also, note that if COUNT_LIMIT is reached before this routine is run by
 the waiting thread, the loop will be skipped to prevent pthread_cond_wait
 from never returning.
 */
 pthread_mutex_lock(&count_mutex);
 if (count < COUNT_LIMIT) {
 pthread_cond_wait(&count_threshold_cv, &count_mutex);
 printf("watch_count(): thread %ld Condition signal received.\n", my_id);
 count += 125;
 printf("watch_count(): thread %ld count now = %d.\n", my_id, count);
 }
 pthread_mutex_unlock(&count_mutex);
 pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
 int i, rc;
 long t1=1, t2=2, t3=3;
 pthread_t threads[3];
 pthread_attr_t attr;

 /* Initialize mutex and condition variable objects */
 pthread_mutex_init(&count_mutex, NULL);
 pthread_cond_init (&count_threshold_cv, NULL);

 /* For portability, explicitly create threads in a joinable state */
 pthread_attr_init(&attr);
 pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
 pthread_create(&threads[0], &attr, watch_count, (void *)t1);
 pthread_create(&threads[1], &attr, inc_count, (void *)t2);
 pthread_create(&threads[2], &attr, inc_count, (void *)t3);

 /* Wait for all threads to complete */
 for (i=0; i<NUM_THREADS; i++) {
 pthread_join(threads[i], NULL);
 }
 printf ("Main(): Waited on %d threads. Done.\n", NUM_THREADS);

 /* Clean up and exit */
 pthread_attr_destroy(&attr);
 pthread_mutex_destroy(&count_mutex);
 pthread_cond_destroy(&count_threshold_cv);
 pthread_exit(NULL);
}

```



## Συνδυασμός MPI με Pthreads

- Σχεδιασμός:
  - Κάθε διεργασία MPI τυπικά δημιουργεί και διαχειρίζεται N νήματα, όπου το N είναι βέλτιστος ο αριθμός των νημάτων ανά διεργασία MPI.
  - Η εύρεση της καλύτερης τιμής για το N ποικίλλει ανάλογα με την πλατφόρμα και τα χαρακτηριστικά της εφαρμογής.
  - Ο αριθμός επεξεργαστών του κόμβου είναι μια καλή αρχική εκτίμηση του N.
  - Γενικά, μπορεί να υπάρχει πρόβλημα αν πολλά νήματα καλούν συναρτήσεις του MPI. Το πρόγραμμα μπορεί να τερματιστεί ή να συμπεριφερθεί απρόοπτα. Αν πρέπει να γίνουν κλήσεις στο MPI από νήματα, καλύτερα να γίνουν από ένα μόνο νήμα.

- Μεταγλώττιση:
  - Χρήση της κατάλληλης εντολής του MPI ανάλογα με την πλατφόρμα και την γλώσσα υλοποίησης
  - Να είναι σίγουρο ότι γίνεται χρήση των κατάλληλων κεφαλίδων όπως στον παραπάνω πίνακα (-pthread)
  - Το MPICH δεν είναι thread safe
- Ένα παράδειγμα το οποίο χρησιμοποιεί και MPI και Pthreads φαίνεται παρακάτω. Οι εκδόσεις εκτέλεσης σειριακά, με νήματα, με MPI και με MPI και νήματα έχουν μια πιθανή πρόοδο.
  - [Serial](#)
  - [Pthreads only](#)
  - [MPI only](#)
  - [MPI with pthreads](#)
  - [makefile](#)

## Θέματα Που Δεν Καλύφθηκαν

Πολλά χαρακτηριστικά του API των Pthreads δεν συζητήθηκαν σε αυτόν τον οδηγό. Αυτά παρουσιάζονται παρακάτω. Βλέπε το τμήμα [Βιβλιοθήκης Αναφορών Ρουτινών Pthread](#) για περισσότερες πληροφορίες.

- Χρονοπρογραμματισμός Νημάτων
  - Οι υλοποιήσεις διαφέρουν ως το πως τα νήματα είναι προγραμματισμένα να εκτελεστούν. Στις περισσότερες περιπτώσεις, ο προκαθορισμένος μηχανισμός είναι αρκετός.
  - Το API των Pthreads προσφέρει ρουτίνες για τον ρητό χρονοπρογραμματισμό των νημάτων καθώς και τον καθορισμό προτεραιοτήτων οι οποίες μπορεί να υπερκαλύψουν τους προκαθορισμένους μηχανισμούς.
  - Το API δεν απαιτεί συγκεκριμένες υλοποιήσεις για να υποστηρίξει αυτά τα χαρακτηριστικά.
- Κλειδιά: Δεδομένα συγκεκριμένα για Νήματα
  - Καθώς τα νήματα καλούν συγκεκριμένες ρουτίνες, τα τοπικά δεδομένα στην στοίβα του νήματος πηγαινοέρχονται.
  - Για την διατήρηση αυτών των δεδομένων, συνήθως περνάει μία παράμετρος από μία ρουτίνα στην επόμενη, διαφορετικά αποθηκεύονται τα δεδομένα σε καθολικές μεταβλητές που σχετίζονται με το νήμα αυτό.
  - Τα Pthreads παρέχουν έναν άλλον, πιθανώς πιο βολικό, τρόπο για να το καταφέρουν αυτό μέσω των *κλειδιών*.
- Παραμέτρος του Πρωτοκόλλου των Mutex και Διαχείριση Προτεραιοτήτων των Mutex για τον χειρισμό προβλημάτων αλλαγής προτεραιοτήτων.
- Διαμοιρασμός μεταβλητών συνθηκών - μεταξύ των διεργασιών
- Ακύρωση Νημάτων
- Νήματα και Σήματα
- Δομές συγχρονισμού - εμπόδια και κλειδώματα

## Βιβλιοθήκη Αναφορών Ρουτινών Pthread

Για περισσότερη άνεση, παρέχεται η λίστα των ρουτινών των Pthread, συνδεδεμένες με την σελίδα οδηγού τους.

[pthread\\_atfork](#)  
[pthread\\_attr\\_destroy](#)  
[pthread\\_attr\\_getdetachstate](#)  
[pthread\\_attr\\_getguardsize](#)  
[pthread\\_attr\\_getinheritsched](#)  
[pthread\\_attr\\_getschedparam](#)  
[pthread\\_attr\\_getschedpolicy](#)  
[pthread\\_attr\\_getscope](#)  
[pthread\\_attr\\_getstack](#)  
[pthread\\_attr\\_getstackaddr](#)  
[pthread\\_attr\\_getstacksize](#)  
[pthread\\_attr\\_init](#)  
[pthread\\_attr\\_setdetachstate](#)  
[pthread\\_attr\\_setguardsize](#)  
[pthread\\_attr\\_setinheritsched](#)  
[pthread\\_attr\\_setschedparam](#)  
[pthread\\_attr\\_setschedpolicy](#)  
[pthread\\_attr\\_setscope](#)  
[pthread\\_attr\\_setstack](#)  
[pthread\\_attr\\_setstackaddr](#)  
[pthread\\_attr\\_setstacksize](#)  
[pthread\\_barrier\\_destroy](#)  
[pthread\\_barrier\\_init](#)  
[pthread\\_barrier\\_wait](#)  
[pthread\\_barrierattr\\_destroy](#)  
[pthread\\_barrierattr\\_getpshared](#)  
[pthread\\_barrierattr\\_init](#)

[pthread\\_barrierattr\\_setpshared](#)  
[pthread\\_cancel](#)  
[pthread\\_cleanup\\_pop](#)  
[pthread\\_cleanup\\_push](#)  
[pthread\\_cond\\_broadcast](#)  
[pthread\\_cond\\_destroy](#)  
[pthread\\_cond\\_init](#)  
[pthread\\_cond\\_signal](#)  
[pthread\\_cond\\_timedwait](#)  
[pthread\\_cond\\_wait](#)  
[pthread\\_condattr\\_destroy](#)  
[pthread\\_condattr\\_getclock](#)  
[pthread\\_condattr\\_getpshared](#)  
[pthread\\_condattr\\_init](#)  
[pthread\\_condattr\\_setclock](#)  
[pthread\\_condattr\\_setpshared](#)  
[pthread\\_create](#)  
[pthread\\_detach](#)  
[pthread\\_equal](#)  
[pthread\\_exit](#)  
[pthread\\_getconcurrency](#)  
[pthread\\_getcpuclockid](#)  
[pthread\\_getschedparam](#)  
[pthread\\_getspecific](#)  
[pthread\\_join](#)  
[pthread\\_key\\_create](#)  
[pthread\\_key\\_delete](#)  
[pthread\\_kill](#)  
[pthread\\_mutex\\_destroy](#)  
[pthread\\_mutex\\_getprioceiling](#)  
[pthread\\_mutex\\_init](#)  
[pthread\\_mutex\\_lock](#)  
[pthread\\_mutex\\_setprioceiling](#)  
[pthread\\_mutex\\_timedlock](#)  
[pthread\\_mutex\\_trylock](#)  
[pthread\\_mutex\\_unlock](#)  
[pthread\\_mutexattr\\_destroy](#)  
[pthread\\_mutexattr\\_getprioceiling](#)  
[pthread\\_mutexattr\\_getprotocol](#)  
[pthread\\_mutexattr\\_getpshared](#)  
[pthread\\_mutexattr\\_gettype](#)  
[pthread\\_mutexattr\\_init](#)  
[pthread\\_mutexattr\\_setprioceiling](#)  
[pthread\\_mutexattr\\_setprotocol](#)  
[pthread\\_mutexattr\\_setpshared](#)  
[pthread\\_mutexattr\\_settype](#)  
[pthread\\_once](#)  
[pthread\\_rwlock\\_destroy](#)  
[pthread\\_rwlock\\_init](#)  
[pthread\\_rwlock\\_rdlock](#)  
[pthread\\_rwlock\\_timedrdlock](#)  
[pthread\\_rwlock\\_timedwrlock](#)  
[pthread\\_rwlock\\_tryrdlock](#)  
[pthread\\_rwlock\\_trywrlock](#)  
[pthread\\_rwlock\\_unlock](#)  
[pthread\\_rwlock\\_wrlock](#)  
[pthread\\_rwlockattr\\_destroy](#)  
[pthread\\_rwlockattr\\_getpshared](#)  
[pthread\\_rwlockattr\\_init](#)  
[pthread\\_rwlockattr\\_setpshared](#)  
[pthread\\_self](#)  
[pthread\\_setcancelstate](#)  
[pthread\\_setcanceltype](#)  
[pthread\\_setconcurrency](#)  
[pthread\\_setschedparam](#)  
[pthread\\_setschedprio](#)  
[pthread\\_setspecific](#)  
[pthread\\_sigmask](#)  
[pthread\\_spin\\_destroy](#)  
[pthread\\_spin\\_init](#)  
[pthread\\_spin\\_lock](#)  
[pthread\\_spin\\_trylock](#)  
[pthread\\_spin\\_unlock](#)  
[pthread\\_testcancel](#)

---



## Αναφορές και Περισσότερες Πληροφορίες

- Το αρχικό κείμενο είναι του [Blaise Barney](#), lawrence Livermore Laboratory, USA. Η μετάφραση, καθώς και ορισμένες προσαρμογές και τροποποιήσεις οφείλονται στο [Κώστα Μαργαρίτη](#), Πανεπιστήμιο Μακεδονίας.
  - POSIX Standard: [www.unix.org/version3/ieee\\_std.html](http://www.unix.org/version3/ieee_std.html)
  - "Pthreads Programming". B. Nichols et al. O'Reilly and Associates.
  - "Threads Primer". B. Lewis and D. Berg. Prentice Hall
  - "Programming With POSIX Threads". D. Butenhof. Addison Wesley
  - "Programming With Threads". S. Kleiman et al. Prentice Hall
  - Γενικότερο υλικό μπορείτε να βρείτε στο [pdplab.it.uom.gr](http://pdplab.it.uom.gr)
-