Λειτουργικά Συστήματα

Ενότητα: Πολυνηματικός Προγραμματισμός

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.icte.uowm.gr/mdasyg
Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» 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.

Σκοπός

- Η κατανόηση της διαφορετικότητας των διεργασιών και των νημάτων.
- Η κατανόηση της αύξησης της επίδοσης μέσω παραλληλοποίησης των νημάτων και των διεργασιών.
- Η περιγραφή και η χρήση των νημάτων POSIX.

Πολυνηματικός Προγραμματισμός

- Για πολυνηματικό προγραμματισμό με χρήση της γλώσσας C, επιλέγεται στο πλαίσιο του μαθήματος η βιβλιοθήκη των νημάτων Posix Threads (Pthreads).
- Χαρακτηριστικά των Pthreads:
  - Ευρεία χρήση.
  - Ώριμη υλοποίηση και πλατιά υποστήριξη από την κοινότητα.
  - Αποδοτικές υλοποιήσεις στα σύγχρονα λειτουργικά συστήματα.
  - Ικανοποιητικός βαθμός μεταφερσιμότητας.
  - Διαθέσιμη βιβλιογραφία μέσω Διαδικτύου.
  - Εύχρηστη και καλά ορισμένη προγραμματιστική διεπαφή.
- Σε Linux και MacOSX βρίσκονται προεγκατεστημένα.
- Σε Windows προτείνουμε την εγκατάσταση τους με χρήση του πακέτου.
- Pthreads-win32 και των εργαλείων MinGW.
- http://sourceware.org/pthreads-win32, http://www.mingw.org
- Μετά την εγκατάσταση οι εντολές θα είναι διαθέσιμες από την γραμμή εντολών.

Λειτουργικό Σύστημα

- Μέσο διασύνδεσης των χρηστών και των εφαρμογών τους με το υλικό των υπολογιστών.
- Ο χρονοδρομολογητής του λειτουργικού συστήματος αναθέτει διεργασίες στους επεξεργαστές του συστήματος.

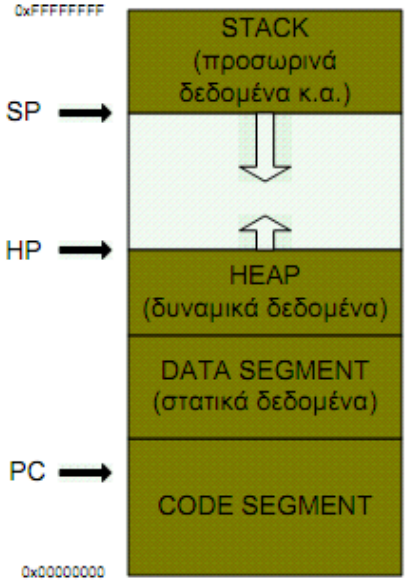
Διεργασίες (1/2)

- Πρόγραμμα = Εκτελέσιμος κώδικας + δεδομένα (στατική οντότητα).
- Διεργασία = Στιγμιότυπο της εκτέλεσης ενός προγράμματος. Συλλογή από δομές δεδομένων που περιγράφουν πλήρως την διαδικασία εκτέλεσης του προγράμματος μια ορισμένη στιγμή (δυναμική οντότητα).

Διεργασίες (2/2)

Η περιγραφή μιας διεργασίας γίνεται μέσω:

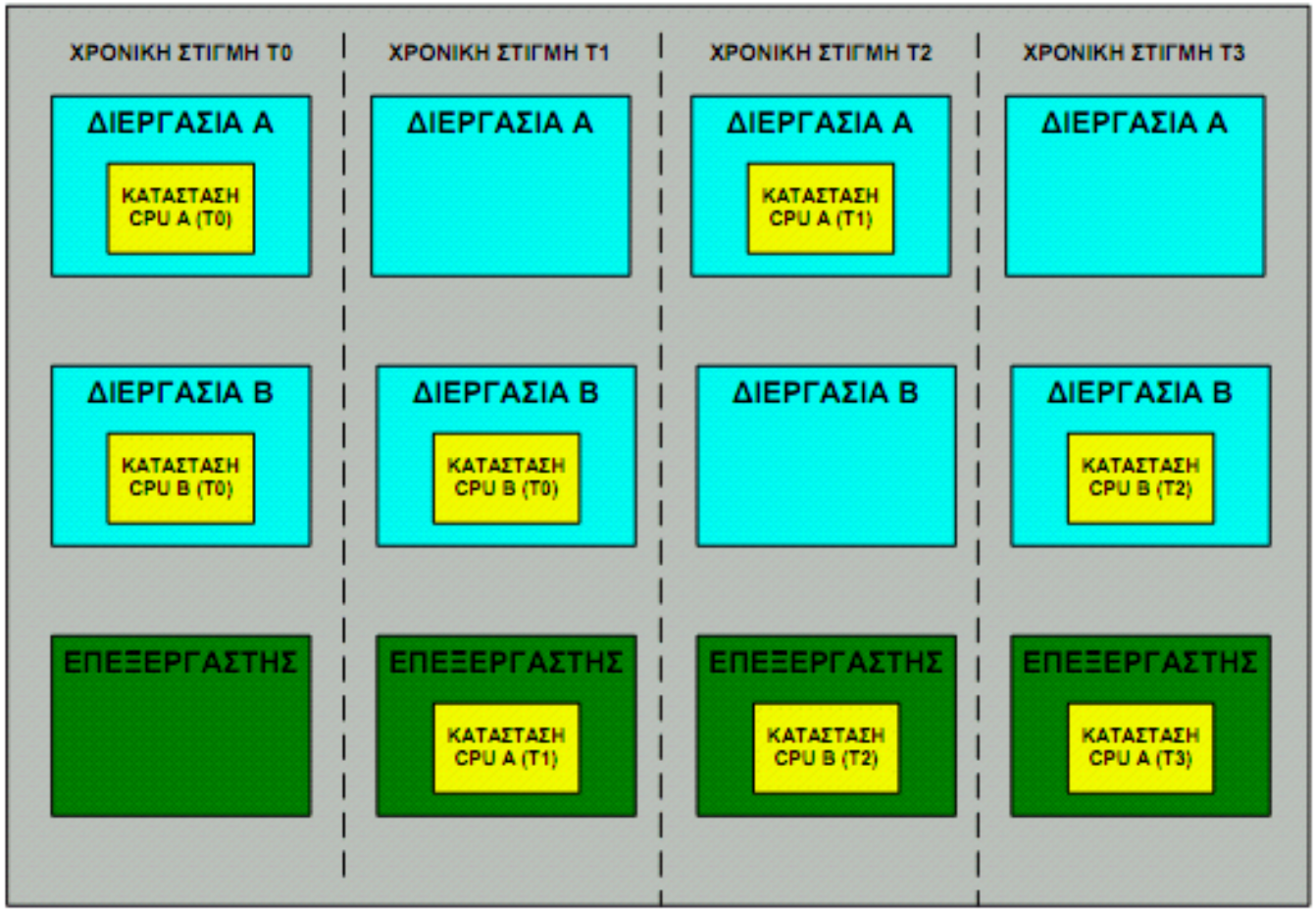
- Του χώρου διευθύνσεων της (address space), δηλαδή τμήματος της κύριας μνήμης που δεσμεύεται για την διεργασία και αποτελείται από:
  - Το τμήμα κώδικα (code segment).
  - Το τμήμα δεδομένων (data segment).
  - Το τμήμα στοίβας (stack segment).
  - Το τμήμα σωρού (Heap).
- Πληροφορίες που διατηρεί το λειτουργικό σύστημα εσωτερικά (κατάσταση διεργασίας, προτεραιότητα, δεσμευμένοι πόροι, στατιστικά).
- Σαν κατάσταση διεργασίας θα αναφέρουμε κάθε στιγμιότυπο κατά στο οποίο τα παραπάνω δεδομένα έχουν συγκεκριμένες τιμές. Ειδικά θα αναφερόμαστε στην:
  - Κατάσταση μνήμης, για να περιγράψουμε την κατάσταση του χώρου διευθύνσεων.
  - Κατάσταση CPU, για να περιγράψουμε τιμές καταχωρητών όπως ο Program Counter (PC), ο Stack Pointer (SP) και άλλοι.



Εκτέλεση Διεργασίας (1/2)

- Για να εκτελεστεί μια διεργασία σε έναν επεξεργαστή πρέπει να φορτωθεί σε αυτόν η κατάσταση CPU της διεργασίας.
- Κατά την εκτέλεση τις διεργασίας η κατάσταση CPU μεταβάλλεται.
- Για να σταματήσει η εκτέλεση μιας διεργασίας και να ξεκινήσει μια άλλη (context switching):
  - Αποθηκεύεται η κατάσταση CPU της τρέχουσας διεργασίας.
  - Φορτώνεται η κατάσταση CPU της προς εκτέλεση διεργασίας.

Εκτέλεση Διεργασίας (2/2)



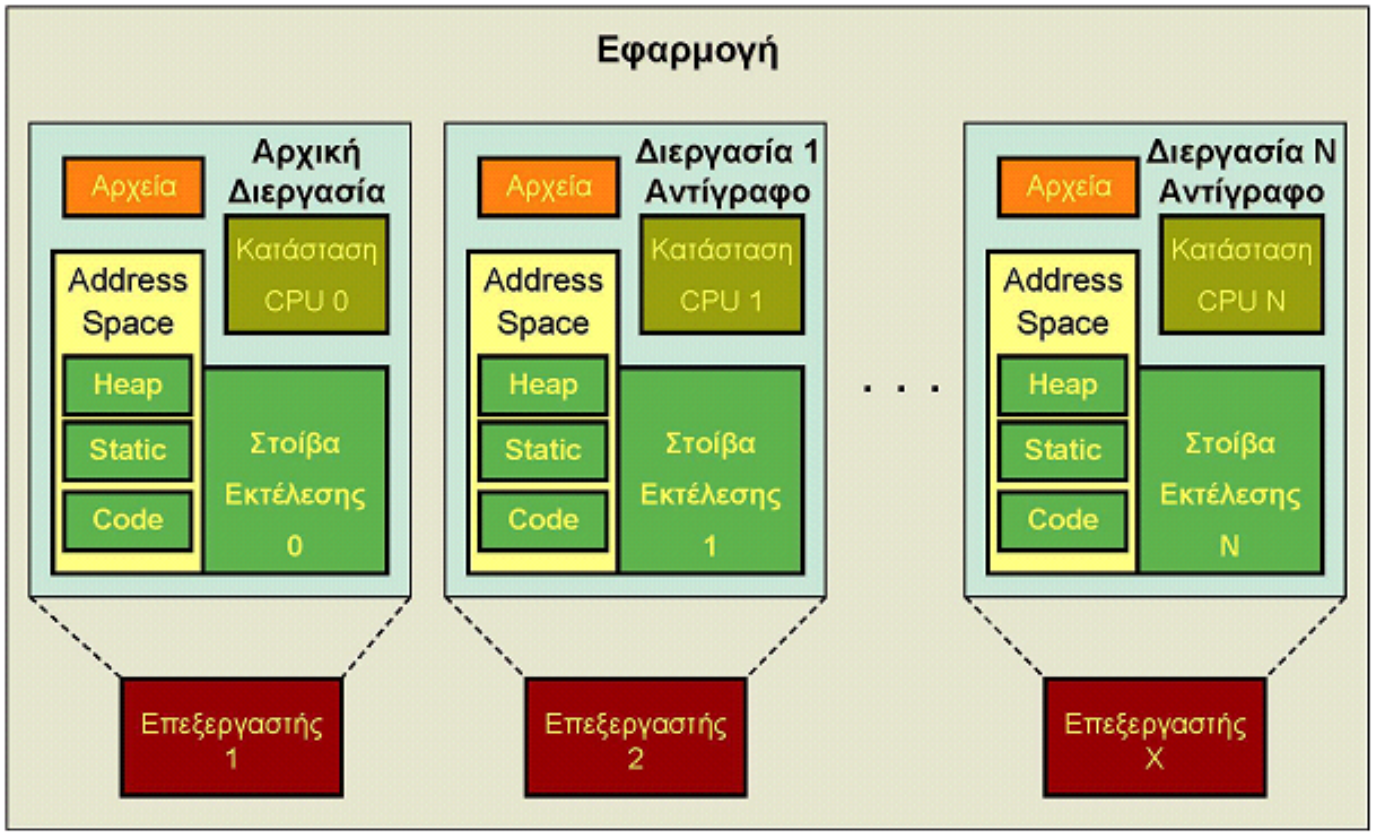
Αξιοποίηση των πολυεπεξεργαστικών

- Κάθε στιγμή σε έναν επεξεργαστή εκτελείται μια διεργασία.
- Θεωρία: “Ο χρόνος μιας εφαρμογής που εκτελείται σε παράλληλο σύστημα, εξαρτάται από το βαθμό παραλληλίας”.
- Οι εφαρμογές καλούνται να εκμεταλλευτούν στο έπακρο την ύπαρξη πολυεπεξεργαστικών συστημάτων.
- Πως μπορεί να γίνει αυτό;

1η Λύση: Πολλαπλές Διεργασίες

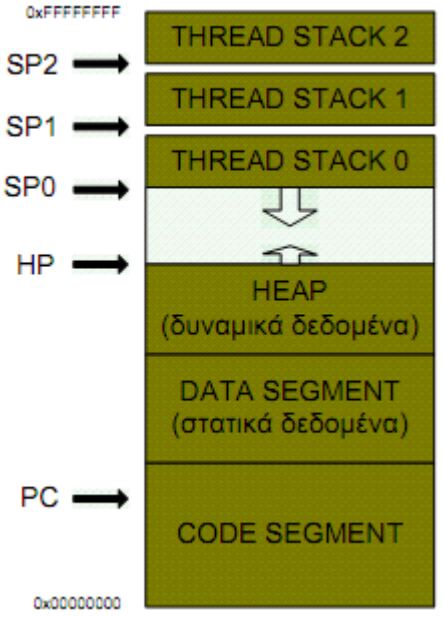
- Η εφαρμογή ξεκινά με την δημιουργία μιας αρχικής διεργασίας (parent process) η οποία δημιουργεί με τη σειρά της διεργασίες-αντίγραφα (child processes).
- Σε κάθε διεργασία-παιδί ο χώρος διευθύνσεων είναι ακριβές αντίγραφο του χώρου διευθύνσεων της αρχικής διεργασίας.
- Κάθε διεργασία-παιδί αναλαμβάνει συγκεκριμένους σκοπούς της εφαρμογής και κατανέμεται σε διαφορετικό επεξεργαστή από τον αλγόριθμο δρομολόγησης, γεγονός που συνεπάγεται την παράλληλη εκτέλεση της εφαρμογής.
- Για την δημιουργία διεργασιών δείτε την κλήση συστήματος fork().

Παράλληλη εκτέλεση διεργασιών

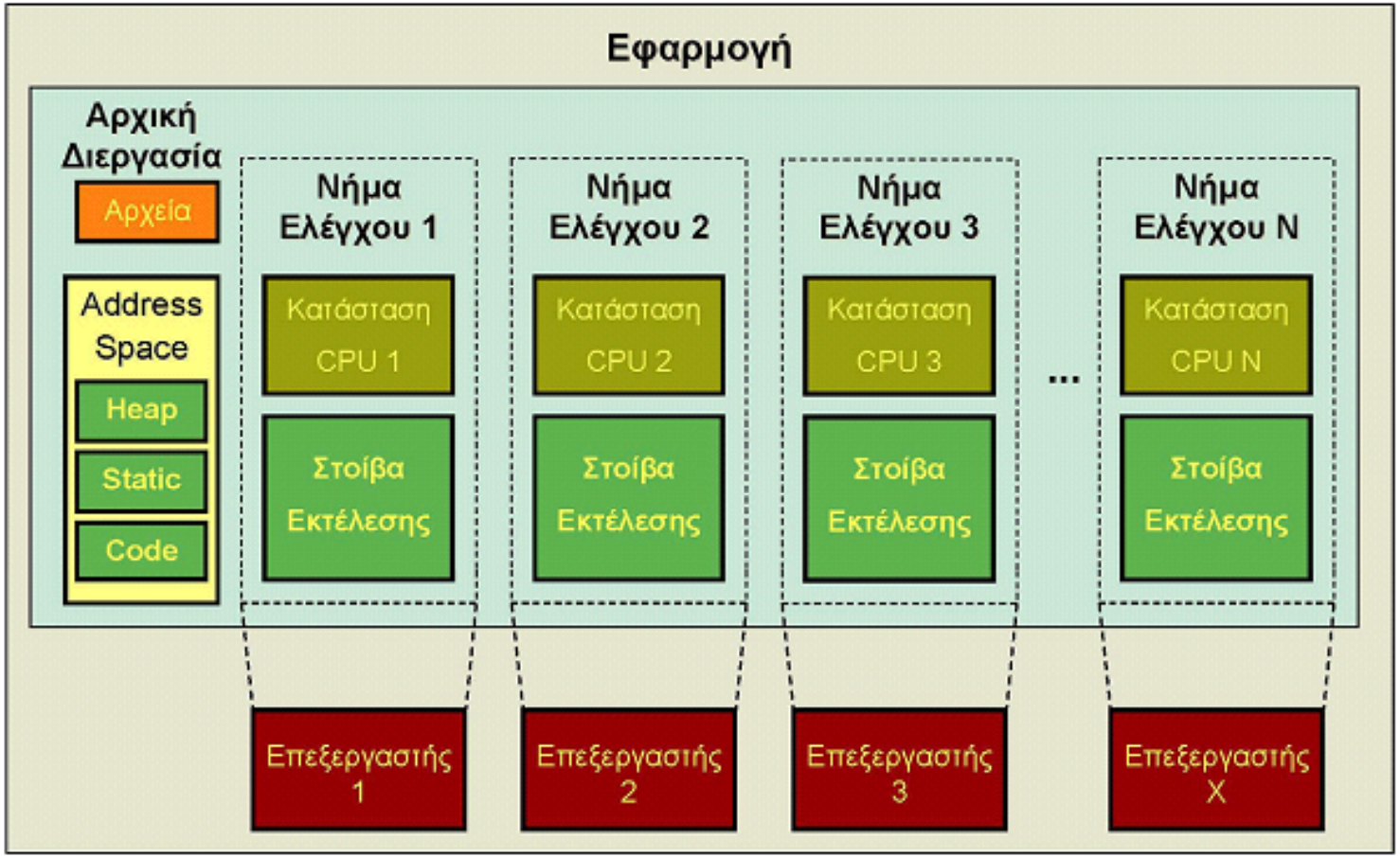


2η Λύση: Νήματα (Threads)

  - Αντιγραφή και διαχωρισμός μόνο των απαραίτητων δεδομένων.
  - Κοινή διαμοίραση των υπόλοιπων.
- Συγκεκριμένα:
  - Μια διεργασία μπορεί να δημιουργήσει πολλαπλά νήματα, τα οποία εκτελούν μια συγκεκριμένη συνάρτηση.
- Κάθε τέτοιο νήμα έχει ξεχωριστή κατάσταση CPU και συνεπώς όλα μπορούν να εκτελεστούν ταυτόχρονα σε διαφορετικούς επεξεργαστές.
- Κάθε νήμα χρησιμοποιεί τον ίδιο χώρο διευθύνσεων, οπότε όλα επενεργούν στα ίδια δυναμικά και στατικά δεδομένα.
- Για την δημιουργία νημάτων δείτε την κλήση συστήματος pthread_create().



Παράλληλη εκτέλεση νημάτων



Σύγκριση διεργασιών - νημάτων

- Χώρος διευθύνσεων.
  - Κάθε διεργασία-παιδί έχει ξεχωριστό χώρο διευθύνσεων.
  - Όλα τα νήματα έχουν κοινό χώρο διευθύνσεων.
  - Τα νήματα είναι πιο βολικά στον προγραμματισμό παράλληλων εφαρμογών καθώς ανταλλάσσουν πληροφορία με απλές λειτουργίες load-store.
  - Η ανταλλαγή πληροφορίας μεταξύ διεργασιών πρέπει να γίνει ρητά (πέρασμα μηνυμάτων, ορισμός κοινής μνήμης).
- Κόστος διαχείρισης.
  - Οι διεργασίες είναι βαριές, η δημιουργία τους και η εναλλαγή τους απαιτούν σημαντικό χρόνο.
  - Τα νήματα είναι πιο ελαφρά, δημιουργούνται και εναλλάσσονται ταχύτερα.
  - Οι παράλληλες εφαρμογές που χρησιμοποιούν νήματα είναι ταχύτερες.
- Μεταφερσιμότητα.
  - Η έννοια της διεργασίας είναι έγκυρη για όλα τα λειτουργικά συστήματα.
  - Η έννοια των νημάτων παρουσιάζει διαφορές από λειτουργικό σε Λειτουργικό.
  - Η υιοθέτηση του προτύπου POSIX για τα νήματα αυξάνει την μεταφερσιμότητα.

Διαχείριση διεργασιών

- Σε κάθε διεργασία που δημιουργείται ανατίθεται ένας μοναδικός αριθμός που λέγεται pid (process id).
  - pid_t pid;
- Μέσω του αναγνωριστικού pid ο προγραμματιστής μπορεί να ξεχωρίσει και να διαχειριστεί τις διάφορες διεργασίες.
- Μια διεργασία μπορεί να μάθει το pid της μέσω της κλήσης.
  - pid_t getpid(void);

Διαχείριση νημάτων

- Σύμφωνα με το πρότυπο POSIX κάθε νήμα που δημιουργείται περιγράφεται από μια δομή τύπου pthread_t:
  - pthread_t thread;
- Μέσω του αναγνωριστικού pthread_t ο προγραμματιστής μπορεί να ξεχωρίσει και να διαχειριστεί τα διάφορα νήματα.
- Ένα νήμα μπορεί να πάρει τη δομή που το περιγράφει με την κλήση.
  - pthread_t pthread_self(void);

Κύκλος ζωής διεργασιών (1/2)

- Δημιουργία:
  - Κλήση συστήματος.
    - **pid_t fork (void);**
- Όταν καλείται η fork() δημιουργείται ένα αντίγραφο της διεργασίας που την κάλεσε.
- Η διεργασία που την καλεί ονομάζεται διεργασία-πατέρας και αυτή που δημιουργείται διεργασία-παιδί.
- Η επεξεργασία συνεχίζεται τόσο στα παιδιά όσο και στον πατέρα, από τις εντολές που ακολουθούν την fork().

Κύκλος ζωής διεργασιών (2/2)

- Για τον ομαλό τερματισμό του προγράμματος η διεργασία - πατέρας θα πρέπει να περιμένει τον τερματισμό των παιδιών:
  - Κλήση συστήματος.
    - **pid_t waitpid(pid_t pid, int *status,int options);**
  - Η διεργασία που καλεί την waitpid() σταματά την εκτέλεση της μέχρι η pid διεργασία που αναμένει να τερματίσει την εκτέλεση της.
  - Τερματισμός των παιδιών.
    - Κλήση **void exit(int);**
  - Η διεργασία που καλεί την exit() τερματίζει την εκτέλεση της.

Παράδειγμα με διεργασίες

```
void * work_function(void *);

int main(int argc, char **argv)
{
        int i=1;
        int pid, status;

        pid = fork();
        if(pid == 0){
                printf("Child\n");
                work_function((void *) i);
                exit(0);
        }

        printf("Father of %d\n", pid);

        waitpid(pid, &status, 0);

        return 0;
}
```

Κύκλος ζωής νημάτων (1/2)

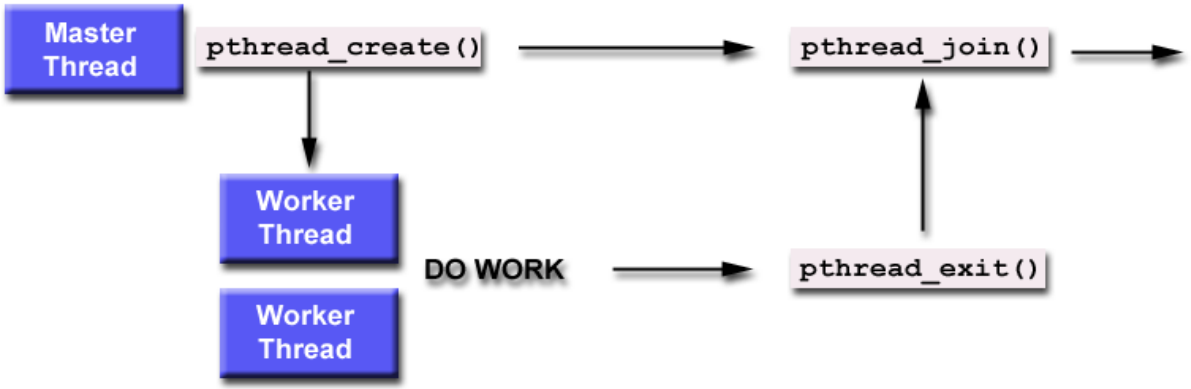
- Δημιουργία.
- Κλήση.
- **int pthread_create (pthread_t * thread, pthread_attr_t *attr, void * (*start_routine) (void *), void * arg);**
- Η κλήση αυτή κατασκευάζει ένα νέο νήμα που θα εκτελέσει τη συνάρτηση start_routine, με όρισμα το arg.
- Η δομή που περιγράφει το νέο νήμα αποθηκεύεται στο όρισμα thread.
- Το νέο νήμα είναι έτοιμο να εκτελεστεί αμέσως μετά τη δημιουργία του.

Κύκλος ζωής νημάτων (2/2)

- Για τον ομαλό τερματισμό του προγράμματος ένα οποιοδήποτε νήμα πρέπει να περιμένει τα υπόλοιπα να τελειώσουν:
  - Κλήση.
  - **int pthread_join(pthread_t thread, void ** ret);**
  - Το νήμα που καλεί την pthread_join( ) σταματά την εκτέλεση του μέχρι το thread νήμα που αναμένει, να τερματίσει την εκτέλεση του. Στο όρισμα ret αποθηκεύεται η τιμή που επιστρέφει η συνάρτηση που εκτελέστηκε από το νήμα.
  - Τερματισμός νημάτων.
  - **void pthread_exit(void *);**

Νήματα και pthread_join() (1/2)

- The pthread_join() subroutine blocks the calling thread until the specified threadid thread terminates.
- The programmer is able to obtain the target thread's termination return status if it was specified in the target thread's call to pthread_exit().
- A joining thread can match one pthread_join() call. It is a logical error to attempt multiple joins on the same thread.
- When a thread is created, one of its attributes defines whether it is joinable or detached. Only threads that are created as joinable can be joined. If a thread is created as detached, it can never be joined.



Νήματα και pthread_join() (2/2)

- To explicitly create a thread as joinable or detached, the attr argument in the pthread_create() routine is used. The typical 4 step process is:
  - Declare a pthread attribute variable of the **pthread_attr_t** data type.
  - Initialize the attribute variable with **pthread_attr_init().**
  - Set the attribute detached status with **pthread_attr_setdetachstate().**
  - When done, free library resources used by the attribute with **pthread_attr_destroy().**
  - The pthread_detach() routine can be used to explicitly detach a thread even though it was created as joinable.

Παράδειγμα με νήματα

```
void * work_function(void *);

int main(int argc, char **argv)
{
        int i = 1;
        pthread_t thread;

        pthread_create(&thread, NULL, workfunction, (void *) i);

        pthread_join(thread, NULL);
        printf("Child ended, exiting…\n");

        return 0;
}
```

Συγχρονισμός μεταξύ νημάτων (1/2)

- Πολύ συχνά, στις διάφορες παράλληλες εφαρμογές δημιουργείται η ανάγκη για συγχρονισμό μεταξύ των διαφόρων νημάτων.
- Για την προσπέλαση από τα διάφορα νήματα κάποιας κοινής μεταβλητής.
- Για να ειδοποιήσει κάποιο νήμα κάποιο άλλο όταν συμβεί κάποιο γεγονός.
  - Νήματα που υλοποιούν ένα pipeline.
  - Νήματα που δουλεύουν με το μοντέλο producer –consumer.
  - Όταν τα νήματα πρέπει να λειτουργήσουν σαν μια ομάδα.
  - Όταν όλα τα νήματα πρέπει να ξεκινήσουν ταυτόχρονα να εκτελέσουν μια εργασία (π.χ. παράλληλο βρόχο).

Προσπέλαση κοινής μεταβλητής χωρίς προστασία

Αποτέλεσμα A=110

ΣΥΜΠΕΡΑΣΜΑ: Χάνονται κάποιες συναλλαγές

Αμοιβαίος αποκλεισμός

- Όταν δύο νήματα ανανεώνουν την ίδια μεταβλητή θα πρέπει να συγχρονίζονται κατά την διαδικασία εγγραφής.
- Αυτό ονομάζεται αμοιβαίος αποκλεισμός.
- Ο αμοιβαίος αποκλεισμός επιτυγχάνεται με την χρήση κλειδιών.
- Ένα κλειδί μπορεί να είναι κλειδωμένο ή ελεύθερο.
- Ένα νήμα πριν ανανεώσει μια κοινή μεταβλητή θα πρέπει να κλειδώσει ένα κλειδί, το οποίο θα ελευθερώσει αμέσως μετά την εγγραφή.

Προσπέλαση κοινής μεταβλητής με χρήση κλειδιού

	Νήμα 1	Νήμα 2	A	B
T0	Διάβασε B=10		A=100	B=10
T1		Διάβασε B=10	A=100	B=10
T2		Πάρε το κλειδί	A=100	B=10
T3	Πάρε το κλειδί	Διάβασε A=100	A=100	B=10
T4	Περίμενε το κλειδί	Πρόσθεσε A=A+B	A=110	B=10
T5	Περίμενε το κλειδί	Άσε το κλειδί	A=110	B=10
T6	Διάβασε A=110		A=110	B=10
T7	Πρόσθεσε A=A+B		A=120	B=10
T8	Άσε το κλειδί		A=120	B=10

ΣΥΜΠΕΡΑΣΜΑ: Δε χάνεται κάποια συναλλαγή. Το A έχει τη σωστή τιμή 120

Μεταβλητές κλειδιά - Δημιουργία

- Το πρότυπο POSIX παρέχει τον τύπο pthread_mutex_t, οποίος ορίζει μια μεταβλητή τύπου κλειδί.
- Οι μεταβλητές αυτού του τύπου θα πρέπει να αρχικοποιηθούν πριν χρησιμοποιηθούν:
  - Στατική αρχικοποίηση:
    - **pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;**
  - Δυναμική αρχικοποίηση χρησιμοποιώντας την κλήση:
    - **int pthread_mutex_init(&mutex, NULL);**
- Ένα αρχικοποιημένο κλειδί αρχικά είναι στην κατάσταση ελεύθερο.

Μεταβλητές κλειδιά - Κλείδωμα

- Ένα νήμα ελέγχου μπορεί να κλειδώσει μια μεταβλητή κλειδί χρησιμοποιώντας την κλήση:
- pthread pthread_mutex_lock (&mutex);
- Μόλις η συνάρτηση αυτή επιστρέψει το κλειδί θα είναι σε κατάσταση “κλειδωμένο”.
- Αν το κλειδί είναι ήδη κλειδωμένο τότε το νήμα που καλεί την παραπάνω κλήση μπλοκάρεται μέχρι το κλειδί να ξαναγίνει “ελεύθερο”.
- Όταν πολλά νήματα καλούν ταυτόχρονα την κλήση αυτή, τότε μόνο ένα από αυτά θα καταφέρει να κλειδώσει το κλειδί.

Μεταβλητές κλειδιά - Απελευθέρωση

- Ένα νήμα ελέγχου μπορεί να απελευθερώσει μια μεταβλητή κλειδί χρησιμοποιώντας την κλήση:
  - **pthread_mutex_unlock (mutex);**
- Μόλις η συνάρτηση αυτή επιστρέψει το κλειδί θα είναι σε κατάσταση “ελεύθερο”.
- Μόλις το νήμα που έχει κλειδώσει το κλειδί καλέσει την παραπάνω συνάρτηση τότε ένα από τα υπόλοιπα νήματα που έχουν μπλοκάρει (περιμένοντας το κλειδί) ξεμπλοκάρεται.
- Και πάλι ένα μόνο από τα μπλοκαρισμένα νήματα θα καταφέρει να ξανακλειδώσει το κλειδί.

Μεταβλητές κλειδιά - Κλείδωμα με έλεγχο

- Ένα νήμα ελέγχου μπορεί να δοκιμάσει να κλειδώσει μια μεταβλητή κλειδιού χρησιμοποιώντας την κλήση:
  - **pthread_mutex_trylock (&mutex);**
- Αν το κλειδί είναι σε κατάσταση ελεύθερο τότε το νήμα που έκανε την κλήση θα κλειδώσει το κλειδί.
- Αν το κλειδί είναι σε κατάσταση κλειδωμένο τότε η κλήση αυτή επιστρέφει έναν ειδικό κωδικό που ενημερώνει τον χρήστη για την κατάσταση.

Μεταβλητές κλειδιά - Καταστροφή

- Για κάθε κλήση pthread_mutex_lock που υπάρχει στο πρόγραμμα μας, θα πρέπει να υπάρχει και η αντίστοιχη κλήση pthread_mutex_unlock για την ίδια μεταβλητή κλειδί.
- Αλλιώς το πρόγραμμα μας μπορεί να ανασταλεί μόνιμα (deadlock).
- Όταν μια μεταβλητή κλειδί δεν χρησιμοποιείται πια, τότε αυτή μπορεί να καταστραφεί με την κλήση:
  - **pthread_mutex_destroy(&mutex);**

Παράδειγμα με μεταβλητές κλειδιά

```
...
pthread_mutex_t mutex =
   PTHREAD_MUTEX_INITIALIZER;
int Global;
...

void work_function(void * arg);
{
        ...

        pthread_mutex_lock(&mutex);
        ...
        // Read/Write Access Global
        // ("Critical Section")
        ...
        pthread_mutex_unlock(&mutex);

        ...
}
```

Συγχρονισμός με χρήση φράγματος (1/2)

- Συγχρονισμός (κυριολεκτικά) των νημάτων σε συγκεκριμένα σημεία του προγράμματος.
- Κατά τον ορισμό ενός φράγματος ορίζεται ο αριθμός των νημάτων που πρόκειται να το χρησιμοποιήσουν.
- Όποτε κάποιο νήμα συναντήσει ένα φράγμα αναστέλλει την εκτέλεση του έως ότου να συναντήσουν το συγκεκριμένο φράγμα όλα τα υπόλοιπα νήματα που έχει οριστεί να το χρησιμοποιήσουν (πλήθος νημάτων).
- Όταν το τελευταίο νήμα συναντήσει το φράγμα, ειδοποιούνται και όλα τα υπόλοιπα να συνεχίσουν την εκτέλεση τους.
- Με αυτό τον τρόπο όλα τα νήματα συγχρονίζονται με το “πιο αργό” νήμα.
- Απλός και σε αρκετούς αλγορίθμους απαραίτητος τρόπος συγχρονισμού.
- Η επίδραση του είναι μικρή όταν το φορτίο είναι ισοκατανεμημένο.

Συγχρονισμός με χρήση φράγματος (2/2)

	Νήμα 1	Νήμα 2	Γύρος
T0	Εκτέλεση Υπολογισμού	Εκτέλεση Υπολογισμού	N
T1	Συνάντηση φράγματος	Εκτέλεση Υπολογισμού	N
T2	Αναμονή στο φράγμα	Εκτέλεση Υπολογισμού	N
T3	Αναμονή στο φράγμα	Εκτέλεση Υπολογισμού	N
T4	Αναμονή στο φράγμα	Συνάντηση φράγματος	-
T5	Έξοδος από το φράγμα	Έξοδος από το φράγμα	-
T6	Εκτέλεση Υπολογισμού	Εκτέλεση Υπολογισμού	N+1
T7			

- Το ταχύτερο σε αυτή τη συγκυρία νήμα 1 συγχρονίζεται με το αργότερο νήμα 2.
- **Παράδειγμα χρήσης:** όταν δεδομένα που παράγονται στο γύρο N είναι απαραίτητα για τον υπολογισμό κατά τον γύρο N+1 κ.ο.κ.

Υλοποίηση φραγμάτων

- Αρχικά δεν προβλεπόταν η υποστήριξη τους.
- Ορίστηκαν σε μεταγενέστερη έκδοση του προτύπου.
- Η υλοποίηση τους στα διάφορα λειτουργικά συστήματα αυτή τη στιγμή είναι προαιρετική.
- Ωστόσο στο linux/BSD διατίθεται πλέον υλοποίηση τους.
- Οι κυριότερες συναρτήσεις για χρήση φραγμάτων είναι:
  - Τύπος:
    - pthread_barrier_t bar;
    - int pthread_barrier_init(&bar, NULL, numOfThreads);
    - int pthread_barrier_wait&bar);
    - int pthread_barrier_destroy(&bar);

Συγχρονισμός μεταξύ νημάτων (2/2)

- Πολύ συχνά, στις διάφορες παράλληλες εφαρμογές δημιουργείται η ανάγκη για συγχρονισμό μεταξύ των διαφόρων νημάτων.
- Για την προσπέλαση από τα διάφορα νήματα κάποιας κοινής μεταβλητής.
- Για να ειδοποιήσει κάποιο νήμα κάποιο άλλο όταν συμβεί κάποιο γεγονός.
- Νήματα που υλοποιούν ένα pipeline.
- Νήματα που δουλεύουν με το μοντέλο producer – consumer.
- Όταν τα νήματα πρέπει να λειτουργήσουν σαν μια ομάδα.
- Όταν όλα τα νήματα πρέπει να ξεκινήσουν ταυτόχρονα να εκτελέσουν μια εργασία (π.χ. παράλληλο βρόχο).

Μεταβλητές υπό συνθήκη (1/2)

- Οι μεταβλητές υπό συνθήκη παρέχουν έναν εναλλακτικό τρόπο για συγχρονισμό μεταξύ των νημάτων.
- Σε αντίθεση με τις μεταβλητές αμοιβαίου αποκλεισμού, η χρήση των μεταβλητών υπό συνθήκη δεν έχει σαν σκοπό να εγγυηθεί την αποκλειστική πρόσβαση ενός νήματος σε δεδομένα (προστασία ενός κρίσιμου τμήματος).
- Οι μεταβλητές υπό συνθήκη αποτελούν ένα μηχανισμό ειδοποίησης μεταξύ των νημάτων.

Μεταβλητές υπό συνθήκη (2/2)

- Για να έχει νόημα η χρήση των μεταβλητών υπό συνθήκη σαν μηχανισμός ειδοποίησης θα πρέπει να συσχετίζεται, από τη μεριά του προγραμματιστή, με τον έλεγχο ορισμένων κοινών δεδομένων.
- Τα κοινά αυτά δεδομένα αποτελούν και τη συνθήκη με την οποία συσχετίζεται η μεταβλητή.
- Χωρίς τις μεταβλητές υπό συνθήκη, ο προγραμματιστής θα έπρεπε συνεχώς να ελέγχει (poll) την τιμή των δεδομένων, καταναλώνοντας πόρους από το σύστημα.
- Αποφεύγεται το συνεχές polling στα δεδομένα, καθώς κάποιο νήμα ειδοποιεί τα ενδιαφερόμενα για την αλλαγή της τιμής των δεδομένων, μέσω μιας μεταβλητής υπό συνθήκη.

Ένα παράδειγμα (1/2)

- Έστω μια κοινή ουρά δεδομένων (queue).
- Ένα νήμα είναι απασχολημένο με την εξαγωγή στοιχείων από την ουρά.
- Όσο η ουρά είναι άδεια το νήμα θα πρέπει να περιμένει χωρίς να εκτελεί χρήσιμη εργασία.
- Στοιχεία στην ουρά τοποθετούν ορισμένα άλλα νήματα σε διάφορες στιγμές κατά την ροή του προγράμματος.

Ένα παράδειγμα (2/2)

- Συνθήκη: Η κατάσταση της κοινής ουράς δεδομένων.
- Γεμάτη ή άδεια. Ο έλεγχος μπορεί να γίνει από την μεταβλητή που αποθηκεύει το μέγεθος της ουράς.
- Το νήμα περιμένει την ειδοποίηση του για την προσθήκη στοιχείου στην ουρά μέσω μιας μεταβλητής υπό συνθήκη.
- Κάποια στιγμή ένα νήμα προσθέτει ένα στοιχείο στην ουρά και ειδοποιεί το νήμα που αναμένει, μέσω της μεταβλητής υπό συνθήκη.
- Κοινά δεδομένα – Κρίσιμα τμήματα:
  - Η μεταβλητή μεγέθους της ουράς.
  - Η κοινή ουρά δεδομένων, μέσω των πράξεων εισαγωγής, εξαγωγής στοιχείων.

Δημιουργία

- Το POSIX παρέχει τον τύπο pthread_cond_t, οποίος ορίζει μια μεταβλητή υπό συνθήκη.
- Και οι μεταβλητές αυτού του τύπου θα πρέπει να αρχικοποιηθούν πριν χρησιμοποιηθούν:
  - Στατική αρχικοποίηση:
    - **pthread_cond_t condition = PTHREAD_COND_INITIALIZER;**
  - Δυναμική αρχικοποίηση χρησιμοποιώντας την κλήση: **int pthread_cond_init(condition);**
- Μια μεταβλητή υπό συνθήκη χρησιμοποιείται πάντα μαζί με μια μεταβλητή τύπου κλειδί.

Αναμονή

- Ένα νήμα ελέγχου μπορεί να περιμένει μέχρι να τεθεί μια μεταβλητή υπό συνθήκη χρησιμοποιώντας την κλήση:
  - **pthread_cond_wait (condition, mutex);**
- Η κλήση αυτή μπλοκάρει το νήμα που την καλεί μέχρι να τεθεί η μεταβλητή υπό συνθήκη.
- Η κλήση αυτή πρέπει να καλείται με την μεταβλητή κλειδί σε κατάσταση κλειδωμένο.
- Κατά την διάρκεια που το νήμα είναι μπλοκαρισμένο η μεταβλητή κλειδί απελευθερώνεται.
- Όταν η κλήση αυτή επιστρέψει το κλειδί είναι πάλι σε κατάσταση κλειδωμένο.

Γιατί χρειάζεται το κλειδί;

- Η συνθήκη, δηλαδή τα δεδομένα προς έλεγχο, είναι κοινά ανάμεσα στα νήματα (πχ το μέγεθος της ουράς).
- Η ασφαλής πρόσβαση στα κοινά δεδομένα εξασφαλίζεται με την χρήση μεταβλητών αμοιβαίου αποκλεισμού.
- Άρα μια μεταβλητή υπό συνθήκη υλοποιεί την ειδοποίηση για την αλλαγή της συνθήκης και μια μεταβλητή αμοιβαίου αποκλεισμού υλοποιεί την ασφαλή/αποκλειστική πρόσβαση στη συνθήκη.
- Επομένως στην κλήση της αναμονής μπορούν να συνδυαστούν και οι δυο χρήσεις των διαφορετικών μεταβλητών, εφόσον ούτως ή άλλως είναι αναγκαίες.

Αφύπνιση

- Ένα νήμα ελέγχου μπορεί να ξυπνήσει ένα νήμα που περιμένει σε κάποια μεταβλητή υπό συνθήκη με την κλήση:
  - **pthread_cond_signal (condition);**
- Ένα νήμα ελέγχου μπορεί να ξυπνήσει όλα τα νήματα που περιμένουν σε κάποια μεταβλητή υπό συνθήκη με την κλήση:
  - **pthread_cond_broadcast (condition);**
- Η κλήση αυτή ενδεχομένως καλείται με το σχετικό κλειδί σε κατάσταση κλειδωμένο. Μετά την κλήση αυτή το κλειδί πρέπει να ελευθερωθεί, έτσι ώστε η αντίστοιχη.
  - **pthread_cond_wait** να λειτουργήσει.

Προϋποθέσεις

- Για να λειτουργήσει σωστά μια μεταβλητή υπό συνθήκη, απαιτείται κατάλληλο κλείδωμα και ξεκλείδωμα της αντίστοιχης μεταβλητής κλειδί.
- Αν κληθεί η pthread_cond_wait χωρίς να έχει κλειδωθεί η μεταβλητή κλειδί, η συμπεριφορά της κλήσης θα είναι απρόβλεπτη.
- Εφόσον έχει δεσμευθεί, αν δεν απελευθερωθεί η μεταβλητή κλειδί μετά την κλήση **pthread_cond_signal**, η αντίστοιχη **pthread_cond_wait** δε θα μπορέσει να ξεμπλοκαριστεί.

Μεταβλητές υπό συνθήκη - Παραδείγματα

```
for(...){
   ...
   pthread_mutex_lock(&mutex);

   count++;
   if(count == N){
        // Wake-up thread 2

        pthread_cond_signal(&condition);
   }

   pthread_mutex_unlock(&mutex);
   ...
   //Do work
}
```

Νήμα 1

```
...
pthread_mutex_lock(&mutex);

while(count<N) {
   pthread_cond_wait(&condition, &mutex);

}
count -= N;
pthread_mutex_unlock(&mutex);
...

//Do work
```

Νήμα 2

Προσοχή στον επανέλεγχο της συνθήκης

- Πάντα πρέπει να γίνεται επανέλεγχος της συνθήκης μετά από το ξεμπλοκάρισμα από την αναμονή σε μια κλήση της.
  - **pthread_cond_wait**
- Αυτό μπορεί να γίνει τοποθετώντας την pthread_cond_wait στο εσωτερικό ενός while loop στο οποίο θα ελέγχετε η συνθήκη.
- Αυτό είναι ιδιαίτερα σημαντικό διότι:
  - Τα νήματα είναι ασύγχρονα. Κάποιο άλλο νήμα μπορεί να έχει προλάβει να μεταβάλει την συνθήκη.
  - Υπάρχει η (σπάνια – αλλά όχι και τόσο σπάνια) περίπτωση των συμπτωματικών ειδοποιήσεων (spurious wakeups). Ειδοποιήσεις οι οποίες προκαλούνται χωρίς να έχει προηγηθεί κάποιο signal ή broadcast της συγκεκριμένης μεταβλητής υπό συνθήκη από άλλο νήμα.

Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης