ΝΗΜΑΤΑ (THREADS)

Λειτουργικά Συστήματα - Εργαστήριο

Πάνος Παπαδόπουλος

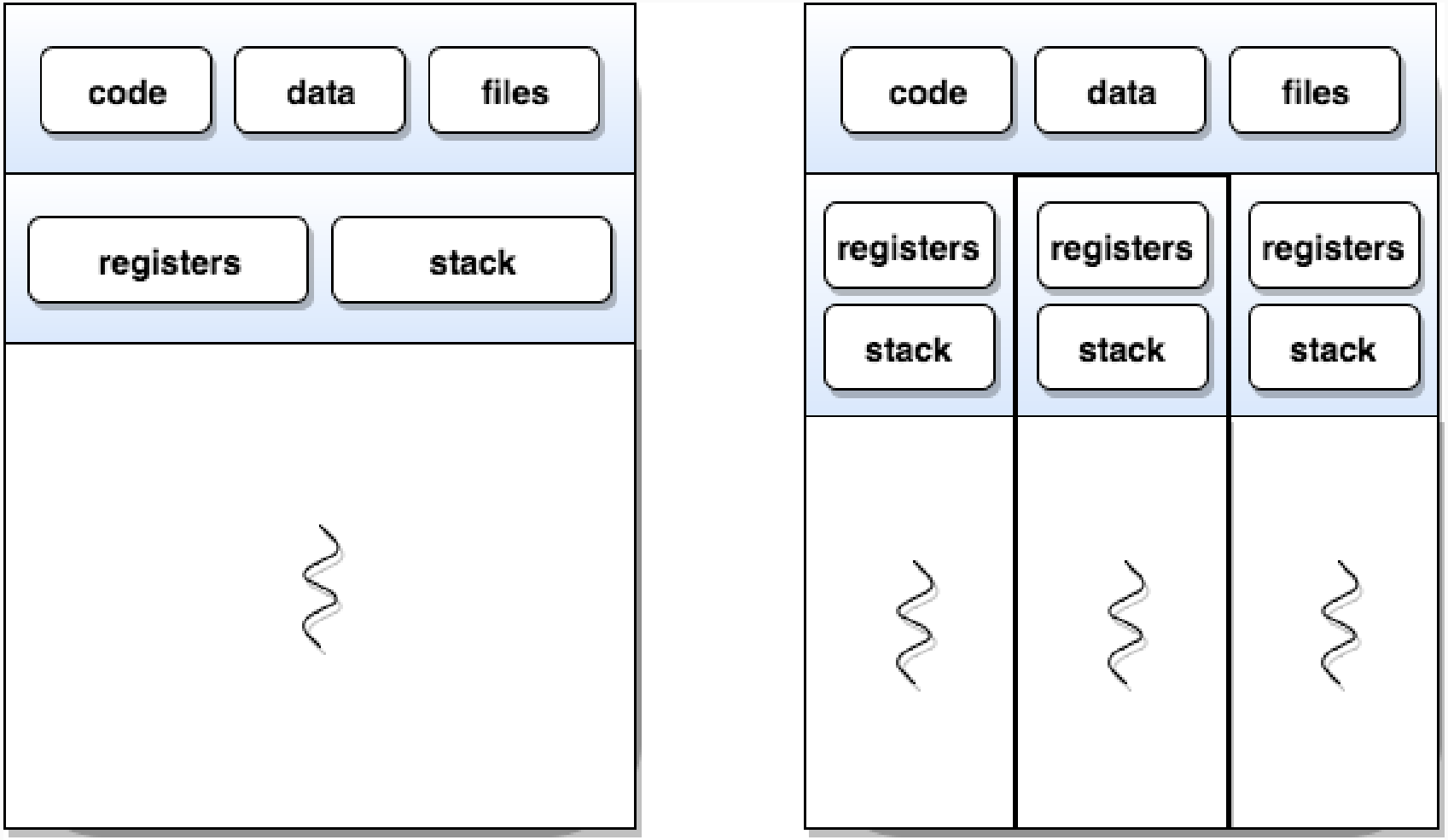
Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική
Πανεπιστήμιο Θεσσαλίας

ΘΕΩΡΙΑ

ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ (1/2)

- Ένα **thread** συχνά θεωρείται ως η μικρότερη μονάδα επεξεργασίας που ένας scheduler διαχειρίζεται.
- Ένα **process** μπορεί να έχει **πολλαπλά threads** τα οποία εκτελούνται ασύγχρονα.
- Η **ασύγχρονη εκτέλεση** δίνει τη δυνατότητα στο κάθε thread να εκτελεί μια εργασία / υπηρεσία **ανεξάρτητα**.
- Όλα τα **threads** έχουν το **ίδιο** αντίγραφο του **heap**.
- Όλα τα **threads** έχουν το **ίδιο** αντίγραφο του **κώδικα**.
- Κάθε **thread** έχει **διαφορετικό stack**.

ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ (2/2)



PROCESSES vs THREADS

Processes	Threads
▷ **Δεν** έχουν κοινή μνήμη.	▷ **Έχουν** κοινή μνήμη (**ίδιου process**).
▷ Ο συγχρονισμός γίνεται από τον **kernel**.	▷ Ο συγχρονισμός γίνεται από το **process** στο οποίο ανήκουν.
▷ Context switching	▷ **Fast** context switching
▷ Δύσκολη (αργή) επικοινωνία.	▷ Γρήγορη επικοινωνία λόγω κοινής μνήμης.

POSIX THREAD (PTHREAD) LIBRARIES

- Επιτρέπει τη δημιουργία νέας ταυτόχρονης ροής διεργασιών.
- Οι παράλληλες τεχνολογίες προγραμματισμού, όπως τα MPI και PVM χρησιμοποιούνται σε ένα κατανεμημένο υπολογιστικό περιβάλλον, ενώ τα threads περιορίζονται σε ένα μόνο υπολογιστή.
- Ένα thread δημιουργείται ορίζοντας μια συνάρτηση και τα ορίσματά της που θα υποβληθούν σε επεξεργασία στο thread.
- `#include <pthread.h>`
- Προσθήκη του **-pthread** κατά το **compile** και το **link**.

ΒΑΣΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

- **pthread_create()**: δημιουργία νέου thread.
- **pthread_cancel()**: τερματισμός ενός thread.
- **pthread_detach()**: αποδέσμευση των πόρων συστήματος ενός thread (το thread μπορεί να τερματίσει χωρίς να περιμένει από τον parent να δεχτεί τον κωδικό εξόδου).
- **pthread_join()**: αναμονή/τερματισμός ενός thread.
- **pthread_exit()**: τερματισμός του thread που την κάλεσε χωρίς επιστροφή του κωδικού εξόδου.
- **pthread_kill()**: αποστολή ενός signal σε ένα thread.
- **pthread_self()**: εύρεση thread id του thread που την κάλεσε.
- **pthread_equal()**: σύγκριση δύο thread id.

ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ THREAD (1/2)

- `#include <pthread.h>` (Δήλωση της `pthread_create()`)
- Prototype: `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
- Δημιουργεί ένα νέο thread.
- Στη παράμετρο **thread** αποθηκεύεται το **id** του thread που δημιουργήθηκε.
- Η παράμετρος **attr** αναπαριστά μια δομή που περιέχει τα **attributes** του thread (αν το attr είναι **NULL** τότε το thread έχει τα **default attributes**).

ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ THREAD (2/2)

- Η παράμετρος **start_routine** αφορά τη **συνάρτηση** που θα εκτελέσει το thread.
- Η παράμετρος **arg** αφορά το **μοναδικό όρισμα** που δέχεται η start_routine.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- **Compile** και **link** με **-pthread**.

ΤΕΡΜΑΤΙΣΜΟΣ THREAD (1/2)

- Ένα thread μπορεί να τερματίσει καλώντας τις **pthread_exit()**, **pthread_join()** και **pthread_cancel()**.
- Αν κάποιο από τα threads ενός process ή αν το κυρίως process καλέσει την **exit()** τότε **όλα τα threads τερματίζουν**.
- Αν ένα thread καλέσει **return** τότε έμεσα καλείται η **pthread_exit()**.
- Όταν ένα thread κάνει **exit**, **δεν** απελευθερώνει τους πόρους του εκτός και αν γίνει **detach**. Η συνάρτηση που χρησιμοποιείται είναι η **pthread_detach()**.

ΤΕΡΜΑΤΙΣΜΟΣ THREAD (2/2)

- Τα **detached threads** απελευθερώνουν τους πόρους τους όταν αυτά κάνουν **exit**.
- Τα threads που δεν είναι detached είναι **joinable** και **δεν απελευθερώνουν** τους πόρους τους εκτός και αν κάποιο άλλο thread καλέσει τη **pthread_join()** για αυτά ή αν όλο το process κάνει **exit**.
- Η **pthread_join()** προκαλεί τον καλούντα να **περιμένει** για το συγκεκριμένο thread να κάνει **exit**(παρόμοια με το waitpid).
- Για να μην έχουμε **memory leaks** όλα τα threads θα πρέπει να καλούν είτε **pthread_detach()** είτε **pthread_join()**.

PTHREAD_EXIT()

- #include <pthread.h> (Δήλωση της **pthread_exit()**)
- Prototype: void **pthread_exit(void *retval)**
- Τερματίζει το thread που την κάλεσε.
- Επιστρέφει τιμή μέσω του **retval**.
- **Compile** και **link** με **-pthread**.

PTHREAD_CANCEL()

- #include <pthread.h> (Δήλωση της **pthread_cancel()**)
- Prototype: int **pthread_cancel**(pthread_t **thread**)
- Αποστολή **request** για cancel σε ένα thread.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- **Compile** και **link** με **-pthread**.

PTHREAD_JOIN()

- `#include <pthread.h>` (Δήλωση της **pthread_join()**)
- Prototype: `int pthread_join(pthread_t thread, void **retval)`
- Περιμένει ένα thread να τερματίσει.
- Το thread ορίζεται από τη παράμετρο **thread**.
- Αν η παράμετρος **retval** δεν είναι **NULL**, τότε αντιγράφεται το **exit status** στη θέση που δείχνει η παράμετρος retval.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- **Compile** και **link** με **-pthread**.

PTHREAD_DETACH()

- `#include <pthread.h>` (Δήλωση της **pthread_detach()**)
- Prototype: `int pthread_detach(pthread_t thread)`
- Μαρκάρει ένα thread που ορίζεται από τη παράμετρο **thread** ως **detached**.
- Όταν ένα **detached thread** τερματίζει, οι πόροι του συστήματος αποδεσμεύονται.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- **Compile** και **link** με **-pthread**.

CODE 1 (1/2)

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

void *print_message(void *arg);

int main (int argc, char *argv[])
{
    int ret;
    pthread_t thread;

    ret = pthread_create(&thread, NULL, print_message, NULL);
    if (ret != 0) {
        printf("pthread_create() error.\n");
        return 1;
    }

    pthread_join(thread, NULL);

    return 0;
}
```

CODE 1 (2/2)

```
21 void *print_message(void *arg)
22 {
23       printf("Hello from thread function.\n");
24
25       return 0;
26 }
```

CODE 2 (1/2)

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

void *print_message(void *arg);

int main (int argc, char *argv[])
{
    int x, ret;
    pthread_t thread;

    x = 10;
    ret = pthread_create(&thread, NULL, print_message, (void*)&x);
    if (ret != 0) {
        printf("pthread_create() error\n");
        return 1;
    }

    pthread_join(thread, NULL);

    return 0;
}
```

CODE 2 (2/2)

```
22 void *print_message(void *arg)
23 {
24     int  *dat;
25
26     dat = (int*)arg;
27
28     printf("Hello from thread function and print %d.\n", *dat);
29
30     return 0;
31 }
```

CODE 3 (1/2)

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

void *print_message(void *arg);

int main (int argc, char *argv[])
{
    int ret;
    pthread_t thread;
    char *msg = "Hello from thread function.";

    ret = pthread_create(&thread, NULL, print_message, (void*)msg);
    if (ret != 0) {
        printf("pthread_create() error\n");
        return 1;
    }

    pthread_join(thread, NULL);

    return 0;
}
```

CODE 3 (2/2)

```
22 void *print_message(void *arg)
23 {
24     char *dat;
25
26     dat = (char*)arg;
27
28     printf("%s\n", dat);
29
30     return 0;
31 }
```

CODE 4 (1/2)

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

typedef struct data {
    int x;
    char y;
    float z;
} data_t;

void *print_message(void *arg);

int main (int argc, char *argv[])
{
    int ret;
    pthread_t thread;
    data_t d;

    d.x = 10;
    d.y = 'A';
    d.z = 5.2;
```

CODE 4 (2/2)

```
21      ret = pthread_create(&thread, NULL, print_message, (void*)&d);
22      if (ret != 0) {
23          printf("pthread_create() error\n");
24          return 1;
25      }
26
27      pthread_join(thread, NULL);
28
29      return 0;
30  }
31
32  void *print_message(void *arg)
33  {
34      data_t *dat;
35
36      dat = (data_t*)arg;
37
38      printf("%d, %c, %f\n", dat->x, dat->y, dat->z);
39
40      return 0;
41  }
```

ΘΕΜΑΤΑ ΣΥΓΧΡΟΝΙΣΜΟΥ (THREAD SYNCHRONIZATION ISSUES)

- Πρόσβαση Κοινής Μεταβλητής (**Shared Variable**)
- Κατάσταση Συναγωνισμού (**Race Condition**)
- Το Πρόβλημα του Κρίσιμου Τμήματος (**Critical Section Problem**)

ΜΗΧΑΝΙΣΜΟΙ ΣΥΓΧΡΟΝΙΣΜΟΥ

- Mutex Locks
- Condition Variables (αναφορικά)
- Read-Write Locks (αναφορικά)

MUTEX LOCKS (1/2)

- Το **mutex** είναι μια **ειδική μεταβλητή** που είναι είτε σε **locked state** είτε σε **unlocked state**.
- Αν ένα mutex είναι **locked**, τότε υπάρχει κάποιο **thread** που έχει **"αποκτήσει/κλειδώσει"** το mutex.
- Αν **κανένα** thread **δεν έχει "αποκτήσει/κλειδώσει"** ένα συγκεκριμένο mutex τότε το mutex είναι **unlocked**.
- Το mutex έχει επίσης και μία **ουρα αναμονής** των threads περιμένουν να το "αποκτήσουν/κλειδώσουν".
- Η **σειρά** με την οποία τα threads που βρίσκονται στην ουρά αναμονής θα "αποκτήσουν/κλειδώσουν" το mutex **αποφασίζεται** με βάση την **thread-scheduling** (Round Robin, Time Sharing κλπ) **πολιτική**.

MUTEX LOCKS (2/2)

- Όταν ένα mutex είναι **ελεύθερο** και ένα thread επιχειρεί να το ”αποκτήσει/κλειδώσει”, τότε το thread ”αποκτάει/κλειδώνει” το mutex **χωρίς να μπλοκάρει**.

- Όταν ένα mutex είναι **ήδη κλειδωμένο από κάποιο thread** και κάποιο άλλο thread επιχειρεί να το ”αποκτήσει/κλειδώσει”, τότε το thread που προσπαθεί να κλειδώσει το mutex **μπλοκάρει**, δηλαδή σταματάει την εκτέλεσή του σε εκείνο το σημείο και μπαίνει στην ουρά αναμονής.

ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ MUTEX (INIT)

- `#include <pthread.h>` (Δήλωση της **pthread_mutex_init()**)
- Prototype: `int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr)`
- Δημιουργεί ένα νέο **mutex** με τα χαρακτηριστικά που περιγράφονται από την παράμετρο **attr**.
- Αν η παράμετρος **attr** είναι **NULL** τότε το mutex θα έχει τα **default** χαρακτηριστικά.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- Προσθήκη του **-pthread** κατά το **compile** και το **link**.

ΚΛΕΙΔΩΜΑ MUTEX (LOCK)

- #include <pthread.h> (Δήλωση της **pthread_mutex_lock()**)
- Prototype: int **pthread_mutex_lock**(pthread_mutex_t ***mutex**)
- Κλειδώνει το **mutex** που καθορίζεται από την παράμετρο mutex.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- Προσθήκη του **-pthread** κατά το **compile** και το **link**.

ΞΕΚΛΕΙΔΩΜΑ MUTEX (UNLOCK)

- `#include <pthread.h>` (Δήλωση της **pthread_mutex_unlock()**)
- Prototype: `int pthread_mutex_unlock(pthread_mutex_t *mutex)`
- Ξεκλειδώνει το **mutex** που καθορίζεται από την παράμετρο mutex.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- Προσθήκη του **-pthread** κατά το **compile** και το **link**.

ΚΑΤΑΣΤΡΟΦΗ MUTEX (DESTROY)

- `#include <pthread.h>` (Δήλωση της `pthread_mutex_destroy()`)
- Prototype: `int pthread_mutex_destroy(pthread_mutex_t * mutex)`
- Καταστρέφει ένα mutex.
- Αν η εκτέλεση **πετύχει** επιστρέφει **0**, αλλιώς επιστρέφει ένα error number.
- Προσθήκη του **-pthread** κατά το **compile** και το **link**.

ΑΣΚΗΣΕΙΣ

ΑΣΚΗΣΕΙΣ (1/3)

1. Να γραφεί πρόγραμμα σε C που θα δημιουργεί μία **συνάρτηση** η οποία θα τυπώνει μια μεταβλητή στην οθόνη. Η μεταβλητή μπορεί να είναι τύπου int , float , double ή char . Η συνάρτηση θα δέχεται **μόνο 2 ορίσματα**. Ένα **generic type pointer (void *)** και ένα **κάτι (enum, int , string...)** που θα προσδιορίζει τον τύπο δεδομένων της μεταβλητής.

2. Να γραφεί πρόγραμμα σε C που θα δημιουργεί **ένα νέο thread**. Το νέο thread θα πρέπει να τυπώνει ένα **μηνύμα** που θα δέχεται από το process που το δημιούργησε.

ΑΣΚΗΣΕΙΣ (2/3)

3. Να γραφεί πρόγραμμα σε C που θα δημιουργεί **ένα νέο thread**. Το νέο thread θα πρέπει να **τυπώνει το άθροισμα 2 αριθμών** που θα δέχεται από το process που το δημιούργησε.

4. Να γραφεί πρόγραμμα σε C που θα δημιουργεί **ένα νέο thread**. Το νέο thread θα πρέπει να **υπολογίζει το άθροισμα 2 αριθμών** που θα δέχεται από το process που το δημιούργησε. Το αποτέλεσμα θα πρέπει να **τυπώνεται** στη οθόνη από το **αρχικό process**.

5. Να γραφεί πρόγραμμα σε C που θα **αυξάνει** μια μεταβλητή **N φορές** χρησιμοποιώντας **N διαφορετικά threads** .

ΑΣΚΗΣΕΙΣ (3/3)

6. Να γραφεί πρόγραμμα σε C που θα υπολογίζει το **μέσο όρο** των στοιχείων ένος δισδιάστατου πίνακα ακεραίων. Θα πρέπει να δημιουργούνται **τόσα threads όσες είναι και οι γραμμές του πίνακα**, το κάθε thread θα υπολογίζει τον μέσο όρο των στοιχείων της γραμμής που του έχει ανατεθεί και αφού ολοκληρώσουν όλα τα threads την εκτέλεσή τους το **κύριο process θα υπολογίζει και εμφανίζει τον τελικό μέσο όρο** των στοιχείων του πίνακα.