

Άσκηση 2

Συγχρονισμός

Λειτουργικά Συστήματα - 6ο εξάμηνο
ακαδημαϊκό έτος 2024 - 2025
Εργαστήριο Υπολογιστικών Συστημάτων



Σύνοψη

- Δύο προβλήματα συγχρονισμού
- Χρήση νημάτων: Υλοποιήσεις με **POSIX Threads**
- Μηχανισμοί συγχρονισμού:
 - Συγχρονισμός Διεργασιών/Νημάτων (Process Synchronization)
 - **POSIX Mutexes και Spinlocks**
 - POSIX Semaphores
 - POSIX Condition Variables
 - Συγχρονισμός σε κοινά δεδομένα (Data Synchronization)
 - GCC atomic operations
- Ο συγχρονισμός διεργασιών βασίζεται στο συγχρονισμό σε κοινά δεδομένα και συχνά περιλαμβάνει τη συνδρομή του Λειτουργικού Συστήματος.

Σύνοψη

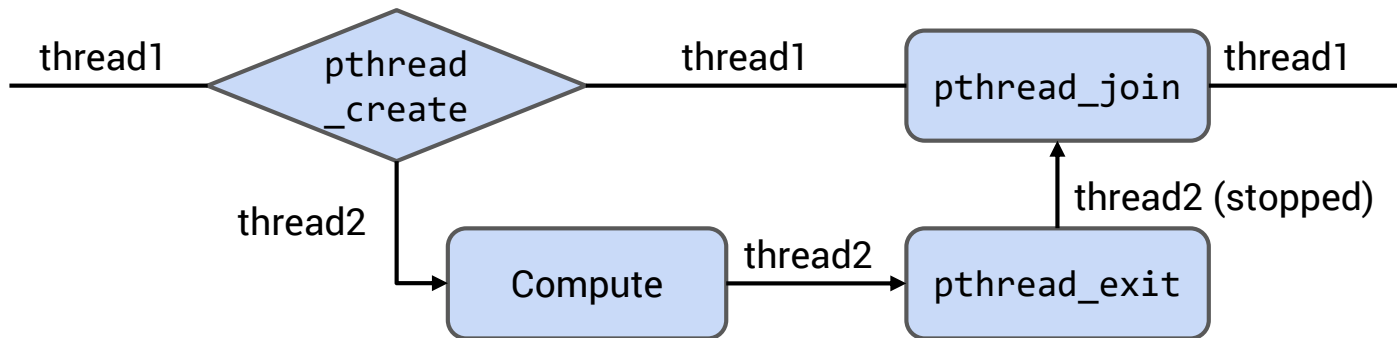
- Ζητούμενο 1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - `simplesync.c`
 - Με POSIX Mutexes (ή spinlocks) και GCC atomic operations
- Ζητούμενο 2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - Συγχρονισμός νημάτων για παράλληλο υπολογισμό
 - Με POSIX semaphores και conditional variables

Σύνοψη

- Δύο προβλήματα συγχρονισμού
- **Χρήση νημάτων: Υλοποιήσεις με POSIX Threads**
- Μηχανισμοί συγχρονισμού:
 - Συγχρονισμός Διεργασιών/Νημάτων (Process Synchronization)
 - POSIX Mutexes και Spinlocks
 - POSIX Semaphores
 - POSIX Condition Variables
 - Συγχρονισμός σε κοινά δεδομένα (Data Synchronization)
 - GCC atomic operations

Δημιουργία Νημάτων στα POSIX Threads

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



Σύνοψη

- Δύο προβλήματα συγχρονισμού
- Χρήση νημάτων: Υλοποιήσεις με POSIX Threads
- **Μηχανισμοί συγχρονισμού:**
 - **Συγχρονισμός Διεργασιών/Νημάτων (Process Synchronization)**
 - POSIX Mutexes και Spinlocks
 - POSIX Semaphores
 - POSIX Condition Variables
 - **Συγχρονισμός σε κοινά δεδομένα (Data Synchronization)**
 - GCC atomic operations

Μηχανισμοί: POSIX

<u>Threads</u> <pthread.h>	pthread_create()	pthread_join()	pthread_exit()
<u>Mutexes</u> <pthread.h>	pthread_mutex_init()	pthread_mutex_lock()	pthread_mutex_unlock()
<u>Spinlocks</u> <pthread.h>	pthread_spin_init()	pthread_spin_lock()	pthread_spin_unlock()
<u>Semaphores</u> <semaphore.h>	sem_init()	sem_wait()	sem_post()
<u>Condition Variables</u> <pthread.h>	pthread_cond_init()	pthread_cond_wait()	pthread_cond_signal() pthread_cond_broadcast()

Περισσότερες πληροφορίες στα man pages:

man -a sem_post

man 7 sem_overview

Μηχανισμοί: GCC Atomic operations

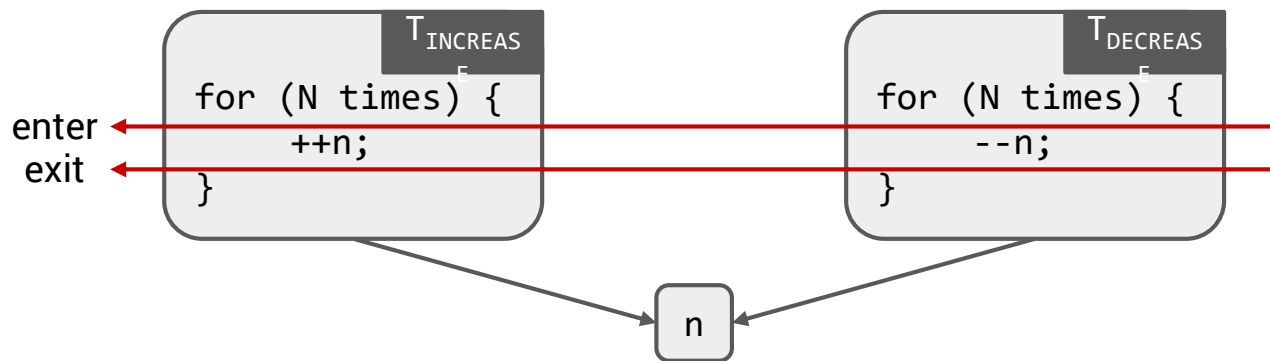
- [GCC Atomic operations](#): Ειδικές εντολές (built-ins) / συναρτήσεις για ατομική εκτέλεση σύνθετων εντολών
- `__sync_add_and_fetch()`, `__sync_sub_and_fetch()`

Σύνοψη

- **Ζητούμενο 1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)**
 - `simplesync.c`
 - Με POSIX Mutexes (ή spinlocks) και GCC atomic operations
- Ζητούμενο 2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - Συγχρονισμός νημάτων για παράλληλο υπολογισμό
 - Με POSIX semaphores και conditional variables

Ζητούμενο 1: Συγχρονισμός σε υπάρχοντα κώδικα

- Δύο νήματα: T_{INCREASE} , T_{DECREASE}
- Αυξάνουν/μειώνουν το **κοινό** n , N φορές αντίστοιχα
- Αρχική τιμή $n=0$. Ζητείται σχήμα συγχρονισμού ώστε **το n να παραμείνει 0**



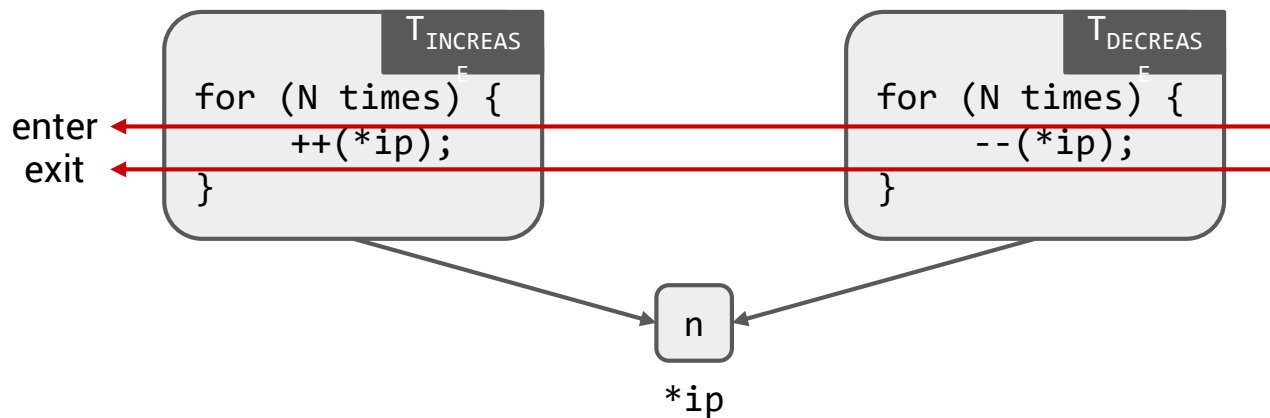
Ζητούμενο 1: Συγχρονισμός στο **simplesync.c**

α) POSIX mutexes

- Κώδικας **μόνο** στα σημεία “enter” και “exit”
- Κατάλληλα αρχικοποιημένα mutexes
- Χωρίς αλλαγή του κώδικα που πειράζει τη μεταβλητή

β) GCC atomic operations

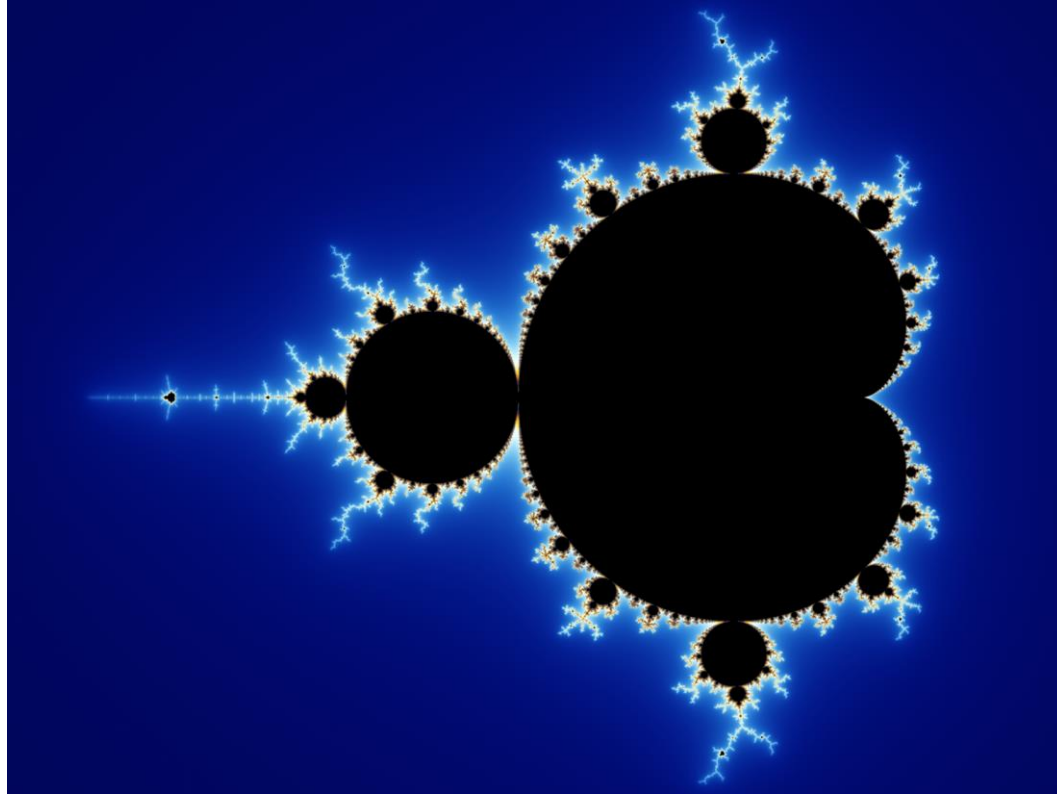
- Αλλαγή του τρόπου πρόσβασης στη μεταβλητή
- Απαιτείται πλέον κώδικας στα “enter” και “exit”;



Σύνοψη

- Ζητούμενο 1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - `simplesync.c`
 - Με POSIX Mutexes (ή spinlocks) και GCC atomic operations
- Ζητούμενο 2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - Συγχρονισμός νημάτων για παράλληλο υπολογισμό
 - Με POSIX semaphores και conditional variables

Ζητούμενο 2: Παραλληλοποίηση Mandelbrot set

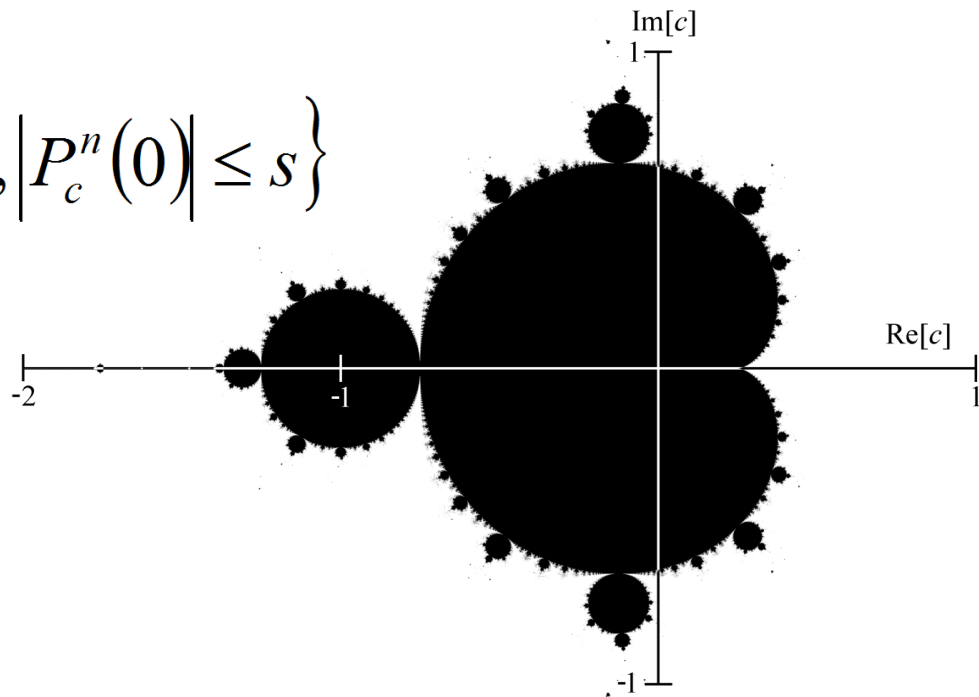


The Mandelbrot set: Ορισμός

$$P_c : C \rightarrow C$$

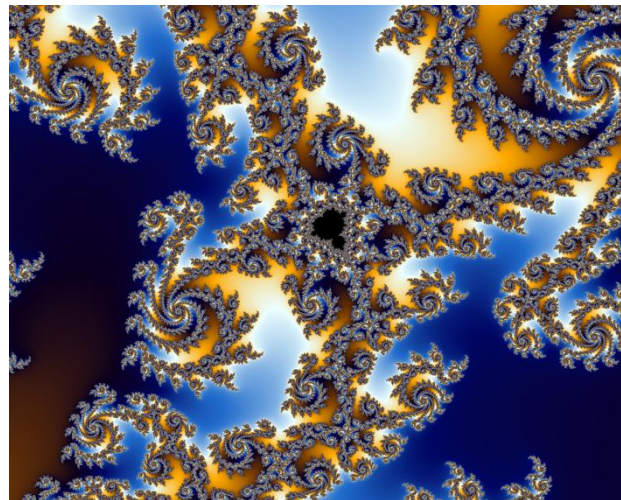
$$P_c : z \rightarrow z^2 + c$$

$$M = \left\{ c \in C : \exists s \in R, \forall n \in N, \left| P_c^n(0) \right| \leq s \right\}$$



The Mandelbrot set: Σχεδίαση

- Για κάθε σημείο c μιας περιοχής του μιγαδικού επιπέδου
 - Επαναληπτικός υπολογισμός του $z_{n+1}=z_n^2+c$, $z_0=0$, μέχρι να ξεφύγει το $|z_n|>2$
 - Κάθε pixel χρωματίζεται ανάλογα με τον αριθμό των επαναλήψεων που χρειάστηκαν ή n_{\max}
- Υπάρχουν κι άλλοι αλγόριθμοι



The Mandelbrot set: Κώδικας

- Σας δίνεται κώδικας (`mandel.c`) που ζωγραφίζει εικόνες από το σύνολο Mandelbrot
 - Στο τερματικό, με χρωματιστούς χαρακτήρες
 - Κάθε εικόνα είναι πλάτους `x_chars` και ύψους `y_chars`
- Η σχεδίαση γίνεται επαναληπτικά, για κάθε γραμμή
- Συναρτήσεις:
 - `compute_and_output_mandel_line(fd, line)`
 - `mandel_iterations_at_point(x, y, MAX)`
 - `set_xterm_color(fd, color)`

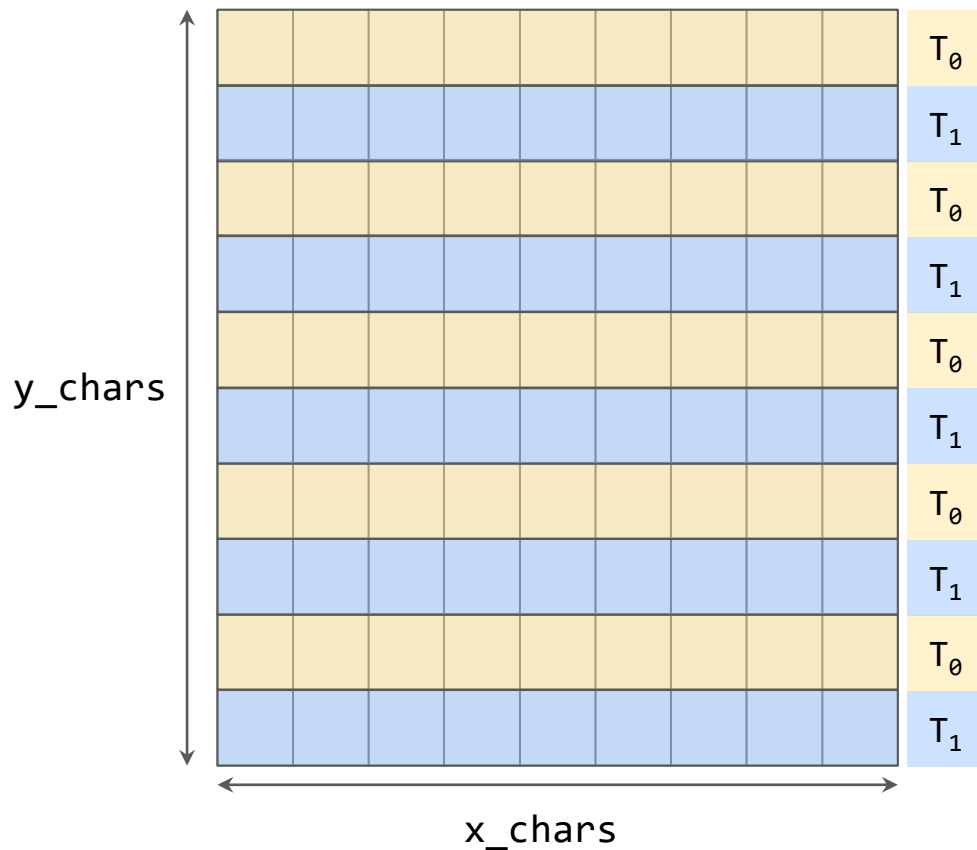
The Mandelbrot set: Παραλληλοποίηση

- Κατανομή του φορτίου ανά γραμμές
- Ξεκινώντας από το πρώτο νήμα, ανάθεση γραμμών με κυκλική επαναφορά

- Νήμα i από N :

$i, i+N, i+2*N, i+3*N, \dots$

Συγχρονισμός;



Condition Variables (σωστή χρήση)

```
pthread_mutex_t lock;  
pthread_cond_t cond;  
int counter = 0;
```

```
/* Operation A */  
pthread_mutex_lock(&lock);  
while (counter < 10)  
    pthread_cond_wait(&cond, &lock);  
... protected critical section ...  
pthread_mutex_unlock(&lock);
```

```
/* Operation B */  
pthread_mutex_lock(&lock);  
counter++;  
pthread_cond_signal(&cond);  
pthread_mutex_unlock(&lock);
```

Condition Variables (λάθος χρήση)

```
pthread_mutex_t lock;  
pthread_cond_t cond;  
int counter = 0;
```

```
/* Operation A */  
pthread_mutex_lock(&lock);  
while (counter < 10)  
    pthread_cond_wait(&cond, &lock);  
... protected critical section ...  
pthread_mutex_unlock(&lock);
```

```
/* Operation B */  
pthread_mutex_lock(&lock);  
counter++;  
if (counter == 10)  
    pthread_cond_signal(&cond);  
pthread_mutex_unlock(&lock);
```

Μπορούμε να κάνουμε signal μόνο 1 φορά αντί 10;
Το signal “ξυπνάει” μόνο μία διεργασία στην “κατάλληλη συνθήκη”
αλλά αυτό είναι λάθος, βλ. the lost wakeur problem και χρήσιμα links
στη συνέχεια

Condition Variables (σωστή χρήση)

```
pthread_mutex_t lock;  
pthread_cond_t cond;  
int counter = 0;
```

```
/* Operation A */  
pthread_mutex_lock(&lock);  
while (counter < 10)  
    pthread_cond_wait(&cond, &lock);  
... protected critical section ...  
pthread_mutex_unlock(&lock);
```

```
/* Operation B */  
pthread_mutex_lock(&lock);  
counter++;  
if (counter == 10)  
    pthread_cond_broadcast(&cond);  
pthread_mutex_unlock(&lock);
```

To broadcast “ξυπνάει” όλες όσες έχουν κάνει wait πάνω στο ίδιο condition variable

Χρήσιμα Links

- [Δημιουργία νημάτων](#)
- [Συγχρονισμός διεργασιών/νημάτων](#)
- [Συγχρονισμός σε κοινά δεδομένα](#)
- [The lost-wakeup problem](#)