[](http://www.teilar.gr/)

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 8:** Παράλληλοι Αλγόριθμοι.

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)

**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* 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.

[](http://www.edulll.gr/)

# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) αντιληφθεί την ανάγκη της παραλληλίας,

2) υπολογίζει το κόστος και την επιτάχυνση παράλληλων αλγορίθμων,

3) γνωρίζει τους τύπους των παράλληλων υπολογιστών.

# ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368165629)

[ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ. 3](#_Toc368165630)

[ΑΛΓΌΡΙΘΜΟΙ. 3](#_Toc368165631)

[ΣΧΉΜΑΤΑ. 3](#_Toc368165632)

[8. ΠΑΡΆΛΛΗΛΟΙ ΑΛΓΌΡΙΘΜΟΙ. 4](#_Toc368165633)

[8.1 Εισαγωγικός αλγόριθμος. 4](#_Toc368165634)

[8.2 Κόστος παράλληλων αλγόριθμων. 5](#_Toc368165635)

[8.3 Τύποι παράλληλων υπολογιστικών μηχανών. 6](#_Toc368165636)

[8.4 Το δείπνο των φιλοσόφων. 9](#_Toc368165637)

# ΑΛΓΌΡΙΘΜΟΙ.

|  |  |
| --- | --- |
| ***Αλγόριθμος 1****: Σειριακός αλγόριθμος πρόσθεσης 100 αριθμών.* | 4 |
| ***Αλγόριθμος 2****: Παράλληλος αλγόριθμος πρόσθεσης 100 αριθμών.* | 5 |
| ***Αλγόριθμος 3****: Το δείπνο των φιλοσόφων (πρώτη έκδοση).* | 9 |
| ***Αλγόριθμος 4****: Το δείπνο των φιλοσόφων (σωστή έκδοση).* | 10 |

# ΣΧΉΜΑΤΑ.

|  |  |
| --- | --- |
| ***Σχήμα 8.1****: SISD.* | 6 |
| ***Σχήμα 8.2****: MISD.* | 7 |
| ***Σχήμα 8.3****: SIMD.* | 7 |
| ***Σχήμα 8.4****: MIMD.* | 8 |
| ***Σχήμα 8.5****: Τοπολογίες παράλληλων συστημάτων.* | 8 |

# 

# 8. ΠΑΡΆΛΛΗΛΟΙ ΑΛΓΌΡΙΘΜΟΙ.

Όλοι οι αλγόριθμοι που έχουν αναλυθεί ή σχεδιασθεί μέχρι τώρα, αφορούσαν υπολογιστικές μηχανές με ένα επεξεργαστή, και κατά συνέπεια η εκτέλεσή τους ήταν σειριακή. Σε αυτό το κεφάλαιο θα αναπτυχθούν αλγόριθμοι, που θα αφορούν υπολογιστές που διαθέτουν παραπάνω από ένα επεξεργαστές, οι οποίοι μπορούν να δουλεύουν ταυτόχρονα και να μοιράζονται (αν είναι απαραίτητο), κοινές πληροφορίες. Στην καθημερινή ζωή, υπάρχουν πάρα πολλές εφαρμογές, οι οποίες πρέπει να δουλεύουν σε πραγματικό χρόνο, ενώ ταυτόχρονα πρέπει να αναλύουν και επεξεργάζονται τεράστιες ποσότητες δεδομένων, όπως για παράδειγμα ένα σύστημα ελέγχου εναέριας κυκλοφορίας, ή ένα σύστημα εντοπισμού και παρακολούθησης έντονων μετεωρολογικών φαινομένων. Σε τέτοιες περιπτώσεις ακόμη και οι πιο γρήγοροι υπολογιστές (με έναν επεξεργαστή), δεν θα μπορούσαν να ανταπεξέλθουν. Βέβαια, οι άνθρωποι στην καθημερινή ζωή, έχουν εφαρμόσει τέτοια μοντέλα παράλληλης επεξεργασίας. Για παράδειγμα, σε ένα σούπερ μάρκετ οι ταμίες είναι σχεδόν πάντα πάνω από ένας, με αποτέλεσμα να εξυπηρετούν παράλληλα πολλούς πελάτες, ή σε βιομηχανικές μονάδες να υπάρχουν παραπάνω από μία γραμμές παραγωγής, με αποτέλεσμα την δημιουργία παράλληλα πολλών προϊόντων.

## 8.1 Εισαγωγικός αλγόριθμος.

Ας υποτεθεί ότι πρέπει να γίνει αλγόριθμος, ο οποίος θα προσθέτει 100 αριθμούς, οι οποίοι είναι αποθηκευμένοι σε ένα μονοδιάστατο πίνακα, έστω *P*100. Ένας σειριακός αλγόριθμος θα έλυνε αυτό το πρόβλημα, όπως φαίνεται στον *Αλγόριθμο 1* .

***Αλγόριθμος 1****: Σειριακός αλγόριθμος πρόσθεσης 100 αριθμών.*

1: Αριθμός: Σειριακή Πρόσθεση(Πίνακας *P*100).

2: Ακέραιος i,

3: Αθροιστής S,

4: S 🡨 0.

5: Για i από 1 μέχρι 100,

6: S 🡨 S + *P*i,

7: Τέλος Για (i).

8: Επέστρεψε S.

9: Τέλος Αλγόριθμου «Σειριακός Αλγόριθμος Πρόσθεσης 100 Αριθμών».

Είναι προφανές, ότι ο αλγόριθμος χρειάζεται να κάνει μία απόδοση αρχικής τιμής στον αθροιστή, και 100 προσθέσεις μέχρι να ολοκληρωθεί και αποδώσει το αποτέλεσμα, δηλαδή σύνολο 101 στοιχειώδεις πράξεις. Σε περίπτωση όμως, που υπήρχε η δυνατότητα να εκτελούνται ταυτόχρονα δύο προσθέσεις (από δύο επεξεργαστές), ο αλγόριθμος θα μπορούσε να τροποποιηθεί και να διαμορφωθεί, όπως περιγράφεται στον *Αλγόριθμο 2*.

***Αλγόριθμος 2****: Παράλληλος αλγόριθμος πρόσθεσης 100 αριθμών.*

1: Αριθμός: Σειριακή Πρόσθεση(Πίνακας *P*100).

2: Παράλληλη Εκτέλεση (Επεξεργαστές E1, E2),

3: E1 ΚΑΙ E2: Ακέραιος i,

4: E1: Αθροιστής S1, E2: Αθροιστής S2,

5: E1: Αθροιστής S,

6: E1: S1 🡨 0, E2: S2 🡨 0.

7: E1: Για i από 1 μέχρι 50, E2: Για i από 51 μέχρι 100,

8: E1: S1 🡨 S1 + *P*i, E2: S2 🡨 S2 + *P*i,

9: Τέλος Για (i).

10: E1: S 🡨 S1 + S2.

11: E1: Επέστρεψε S.

12: Τέλος Αλγόριθμου «Παράλληλος Αλγόριθμος Πρόσθεσης 100 Αριθμών».

Ο χρόνος που απαιτεί ο παραπάνω παράλληλος αλγόριθμος, είναι για 52 στοιχειώδεις πράξεις, και κατά συνέπεια με μια πρώτη εκτίμηση, είναι σχεδόν δύο φορές ταχύτερος από τον αντίστοιχο σειριακό. Αυτό όμως που δεν συνυπολογίσθηκε, είναι ο χρόνος επικοινωνίας μεταξύ των επεξεργαστών. Δηλαδή, για να μπορέσει ο E1 να εκτελέσει την γραμμή 10 του *Αλγόριθμου 2* πρέπει να έχει την πληροφορία για την τιμή της S2. Επομένως, εάν το κόστος της μεταφοράς αυτής της πληροφορίας, θεωρηθεί όσο και το κόστος μίας στοιχειώδους πράξης, τότε ο πραγματικός λόγος της συμπεριφοράς των δύο αλγορίθμων, είναι 53 / 101. Η προηγούμενη παρατήρηση όμως οδηγεί και στην σκέψη, ότι όσους πιο πολλούς επεξεργαστές διαθέτει ένα σύστημα, τόσο πιο πολύ αυξάνεται το επικοινωνιακό κόστος, και πιθανώς μερικές φορές να καθιστά απαγορευτική την χρήση μεγάλου αριθμού επεξεργαστών, (τι θα γινόταν αν το προηγούμενο πρόβλημα επιλυόταν από 50 επεξεργαστές;).

## 8.2 Κόστος παράλληλων αλγόριθμων.

Για να εκτιμηθεί σωστά ένας παράλληλος αλγόριθμος, πρέπει να συγκριθεί πρώτα από όλα με τον καλύτερο δυνατό αντίστοιχο σειριακό αλγόριθμο. Μια αρκετά καλή ένδειξη του πόσο επαρκής είναι ένας καλός αλγόριθμος, είναι η επιτάχυνση που προσφέρει σε σχέση με τον σειριακό, και αυτό ορίζεται ως εξής:

.

Όπου *α* είναι η επιτάχυνση, το ts δηλώνει τον χειρότερο χρόνο του ταχύτερου σειριακού αλγόριθμου, και τέλος το tp δηλώνει τον χειρότερο χρόνο του ταχύτερου παράλληλου αλγόριθμου.

Επιπλέον, ένας σημαντικός παράγοντας που πρέπει να ληφθεί υπόψη, είναι το πλήθος των συμμετεχόντων επεξεργαστών. Προφανώς, ο μεγαλύτερος αριθμός επεξεργαστών, συνεπάγεται μεγαλύτερο οικονομικό κόστος αγοράς, αλλά και κόστος συντήρησης. Επομένως, το κόστος (*k*) ενός παράλληλου αλγόριθμου αυξάνεται με τον αριθμό των επεξεργαστών (*N*).

Συμπερασματικά, το συνολικό κόστος δίνεται από τον τύπο:

,

και τελικά η απόδοση (*E*) του αλγόριθμου μπορεί να αποδοθεί από τον τύπο:

.

Φυσικά, εκτός από τους παραπάνω ορισμούς του κόστους, μπορούν να προσμετρηθούν και άλλα μεγέθη, όπως η καλωδίωση για την δικτύωση των επεξεργαστών, η συνολική μνήμη, και άλλα.

## 8.3 Τύποι παράλληλων υπολογιστικών μηχανών.

Μία παράλληλη υπολογιστική μηχανή (αλλά και με έναν επεξεργαστή), λειτουργεί εκτελώντας κάποιες οδηγίες επάνω σε κάποια δεδομένα. Ανάλογα με τον τρόπο διαχείρισης των δεδομένων, οι υπολογιστικές μηχανές διακρίνονται στις εξής κατηγορίες:

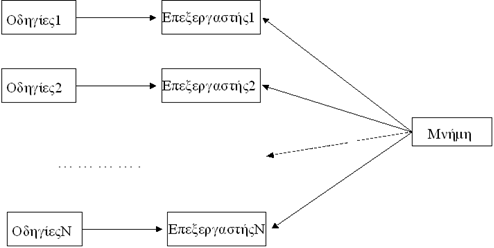
1. SISD: Σειριακή εκτέλεση οδηγιών – Σειριακή διαχείριση δεδομένων, *Σχήμα 8.1* (Single Instruction stream, Single Data stream).
2. MISD: Παράλληλη εκτέλεση οδηγιών – Σειριακή διαχείριση δεδομένων, *Σχήμα* *8.2* (Multiple Instruction stream, Single Data stream).
3. SIMD: Σειριακή εκτέλεση οδηγιών – Παράλληλη διαχείριση δεδομένων, *Σχήμα 8.3* (Single Instruction stream, Multiple Data stream).
4. MIMD: Παράλληλη εκτέλεση οδηγιών – Παράλληλη διαχείριση δεδομένων, *Σχήμα 8.4*  (Multiple Instruction stream, Multiple Data stream).

***SISD***: Σε αυτό το μοντέλο, ένας επεξεργαστής δέχεται μία οδηγία κάθε φορά, η οποία με την σειρά της επενεργεί σε δεδομένα που βρίσκονται καταχωρημένα στην μνήμη.

Μπλόκ διάγραμμα: Ο επεξεργαστής δέχεται τις οδηγίες και τα δεδομένα που υπάρχουν στην μνήμη.

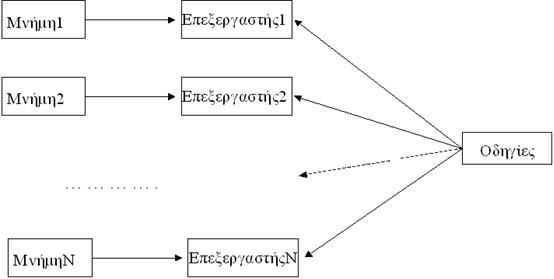
***Σχήμα 8.1****: SISD.*

***MISD***: Πολλοί επεξεργαστές εκτελούν διαφορετικές εργασίες αλλά όλοι μοιράζονται την ίδια μνήμη, και αυτές οι διαφορετικές οδηγίες επενεργούν στα ίδια δεδομένα.



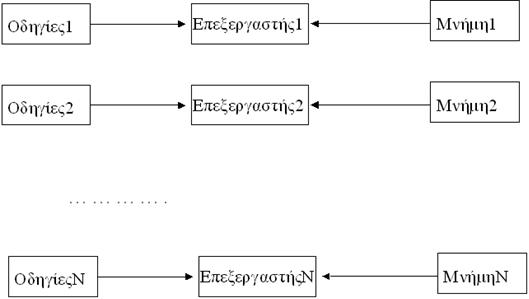
***Σχήμα 8.2*** *: MISD.*

***SIMD***: Αυτός ο τύπος παράλληλης υπολογιστικής μηχανής, λειτουργεί με το να εκτελούν όλοι οι επεξεργαστές τις ίδιες οδηγίες, αλλά αυτές να επιδρούν σε διαφορετικά δεδομένα.



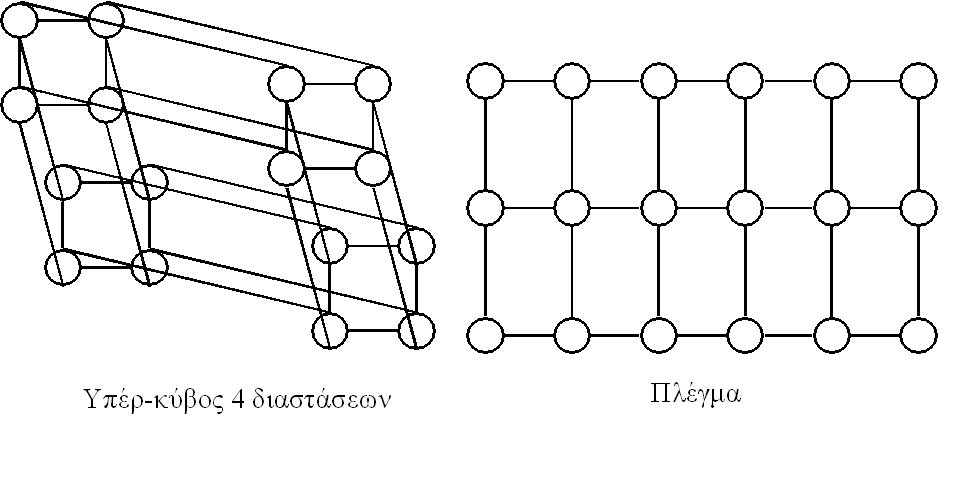
***Σχήμα 8.3*** *: SIMD.*

***MIMD***: Τέλος, αυτό είναι το πιο γενικό και ισχυρό μοντέλο παράλληλης επεξεργασίας, όπου ο κάθε επεξεργαστής λειτουργεί με διαφορετικές οδηγίες σε διαφορετικά δεδομένα. Βέβαια, αυτό είναι και το πιο πολύπλοκο μοντέλο, γιατί οι περισσότεροι αλγόριθμοι πρέπει να επανασχεδιασθούν.



***Σχήμα 8.4*** *: MIMD.*

Τέλος, στην περίπτωση του MIMD μοντέλου, πρέπει να είναι γνωστή η τοπολογία σύνδεσής των επεξεργαστών, έτσι ώστε να αποφεύγονται οι επικοινωνίες μεταξύ των απομακρυσμένων. Με αυτό τον τρόπο, μειώνεται και το κόστος μεταφοράς πληροφοριών, εάν είναι γνωστό ποιοι είναι οι γειτονικοί κόμβοι. Μερικές από τις πιο συνηθισμένες τοπολογίες, φαίνονται στο *Σχήμα 8.5*.



***Σχήμα 8.5****: Τοπολογίες παράλληλων συστημάτων.*

## 8.4 Το δείπνο των φιλοσόφων.

Οι παράλληλοι αλγόριθμοι και κατ’ επέκταση ο παράλληλος προγραμματισμός, δεν είναι ούτε μία εύκολη διαδικασία, αλλά ούτε και χωρίς προβλήματα. Ένα πολύ ενδιαφέρον πρόβλημα που αναδεικνύει όλες τις πιθανές παγίδες που μπορούν να εμφανισθούν, είναι το πρόβλημα «το δείπνο των φιλοσόφων» (Dijkstra): Έστω ότι σε ένα μοναστήρι ζουν πέντε μοναχοί, οι οποίοι είναι ταυτόχρονα και φιλόσοφοι. Ο καθένας τους ασχολείται μόνο με τον στοχασμό, αλλά πότε-πότε πρέπει και να τρώει. Δηλαδή, η ζωή τους είναι ένας ατέλειωτος κύκλος σκέψης και φαγητού. Όσο αναφορά το φαγητό, υπάρχει ένα στρογγυλό τραπέζι όπου στο μέσον υπάρχει μία πιατέλα με μακαρόνια, η οποία ανανεώνεται συνέχεια. Στο τραπέζι υπάρχουν πέντε πιάτα, και ανάμεσα σε κάθε πιάτο υπάρχει ένα πιρούνι, (δηλαδή, σύνολο πάλι πέντε πιρούνια). Ο κάθε μοναχός – φιλόσοφος, όταν θέλει να φάει μπαίνει στην αίθουσα, κάθεται σε μία άδεια θέση, και χρησιμοποιώντας υποχρεωτικά δύο πιρούνια, αυτό-σερβίρεται. Τώρα όμως εδώ, δημιουργείται το πρόβλημα του ότι πρέπει να είναι ελεύθερα και τα δύο πιρούνια, (δεξιά και αριστερά του), γιατί αλλιώς δεν μπορεί να σερβιριστεί. Πάντως, όταν φάει, σηκώνεται και επιστρέφει στο κελί του και στον διαλογισμό του. Το πρόβλημα λοιπόν, έγκειται στο να δημιουργηθεί ένα πρωτόκολλο, για να μπορούν οι φιλόσοφοι να τρώνε, ώστε να μην λιμοκτονήσει ποτέ κανείς τους. Το πρωτόκολλο πρέπει να ικανοποιεί τις ακόλουθες απαιτήσεις:

1. Αμοιβαίος αποκλεισμός: δύο φιλόσοφοι δεν μπορούν να χρησιμοποιούν ταυτόχρονα το ίδιο πιρούνι,
2. Απουσία αδιεξόδου: πρέπει να μπορούν όλοι να φάνε.

***Αλγόριθμος 3****: Το δείπνο των φιλοσόφων (πρώτη έκδοση).*

1: Είσοδος ⁄ Δεδομένα: Δυαδικός Πίνακας fork5 (0 / 1), ακέραιος i.

2: Για i 🡨 1 μέχρι 5 παράλληλα,

3: Εκτέλεση διαδικασίας Φιλόσοφος(i),

4: Τέλος Για (i).

5: Διαδικασία Φιλόσοφος(ακέραιος i).

6: Επανέλαβε για πάντα,

7: Διαλογίσου,

8: Περίμενε(forki), /\* Περίμενε για πιρούνι \*/,

9: Περίμενε(fork(i+1)mod5,  /\* Εξασφάλισε διπλανό πιρούνι \*/,

10: Δείπνησε,

11: Απελευθέρωσε(forki),

12: Απελευθέρωσε(fork(i+1)mod5,

13: Τέλος Επανάληψης.

14: Τέλος Διαδικασίας «Φιλόσοφος».

15: Τέλος Αλγόριθμου «Το δείπνο των φιλοσόφων (πρώτη έκδοση)».

Ο *Αλγόριθμος 3* υλοποιεί μία πολύ απλή ιδέα, και γενικά είναι ικανοποιητικός, αλλά μπορεί να οδηγήσει σε αδιέξοδο. Δηλαδή, ο πίνακας fork, εξασφαλίζει τον αμοιβαίο αποκλεισμό, και η ιδιότητα της ασφάλειας ικανοποιείται γιατί κανείς δεν μπορεί να φάει, εάν δεν εξασφαλίσει πρώτα τα δύο πιρούνια. Το πρόβλημα δημιουργείται μόνο όταν και οι πέντε φιλόσοφοι, μπουν ταυτόχρονα στο δωμάτιο, και καθίσουν όλοι μαζί στο τραπέζι. Τότε, όλοι θα πάρουν από ένα πιρούνι, αλλά δεν θα μπορέσουν ποτέ να εξασφαλίσουν το δεύτερο, και κατά συνέπεια μάλλον θα οδηγηθούν στην λιμοκτονία. Εδώ, αναδεικνύεται το πρόβλημα του τέλειου συγχρονισμού στους παράλληλους αλγόριθμους. Κατά συνέπεια, πρέπει με κάποιο τρόπο να παρακολουθείται το πλήθος των ελεύθερων πιρουνιών, πριν ένας φιλόσοφος προσπαθήσει να μπει στο δωμάτιο για να φάει. Για να αντιμετωπισθεί το παραπάνω πρόβλημα, μία λύση είναι το να επιτρέπεται σε ένα φιλόσοφο να παίρνει πιρούνια μόνο εάν και τα δύο πιρούνια είναι ελεύθερα, και όχι να παίρνει ένα και να περιμένει το δεύτερο, δεσμεύοντας ίσως και έπ’ άπειρον το πρώτο. Και αυτό όμως το σενάριο έχει πρόβλημα. Εάν οι φιλόσοφοι 1 και 3 «συνωμοτούσαν», τότε ο φιλόσοφος 2 δεν θα μπορούσε ποτέ να βρει ταυτόχρονα δύο διαθέσιμα πιρούνια. Επομένως, και αυτή η λύση οδηγεί σε αδιέξοδο. Πρέπει να επισημανθεί, και μάλιστα με πολλή έμφαση το γεγονός ότι στους παράλληλους αλγόριθμους, δεν πρέπει να υπάρχει απολύτως καμία περίπτωση αποκλεισμού ή παραγκωνισμού.

Μία σωστή λύση, θα μπορούσε να βάλλει κάποιους επιπλέον κανόνες ασφάλειας. Δηλαδή, εάν γινόταν έλεγχος σε σχέση με το πόσοι φιλόσοφοι βρίσκονται μέσα στο δωμάτιο. Εάν είναι το πολύ τέσσερις φιλόσοφοι είναι βέβαιο ότι με οποιονδήποτε συνδυασμό, τουλάχιστον ένας θα καταφέρει να φάει. Οπότε, ένας σωστός αλγόριθμος θα μπορούσε να είναι, όπως αυτός που περιγράφεται στον *Αλγόριθμο 4.*

***Αλγόριθμος 4****: Το δείπνο των φιλοσόφων (σωστή έκδοση).*

1: Είσοδος ⁄ Δεδομένα: Δυαδικός Πίνακας fork5 (0/1), ακέραιος i, ακέραιος room=4.

2: Για i 🡨 1 μέχρι 5 παράλληλα,

3: Εκτέλεση διαδικασίας Φιλόσοφος(i),

4: Τέλος Για (i).

5: Διαδικασία Φιλόσοφος( ακέραιος i ).

6: Επανέλαβε για πάντα,

7: Διαλογίσου,

8: Περίμενε(room),

9: Περίμενε(forki), /\* Περίμενε για πιρούνι \*/,

10: Περίμενε(fork(i+1)mod5,  /\* Εξασφάλισε διπλανό πιρούνι \*/,

11: Δείπνησε,

12: Απελευθέρωσε(forki),

13: Απελευθέρωσε(fork(i+1)mod5,

14: Ενημέρωσε(room),

15: Τέλος Επανάληψης.

13: Τέλος Διαδικασίας «Φιλόσοφος».

14: Τέλος Αλγόριθμου «Το δείπνο των φιλοσόφων (σωστή έκδοση)».

Τέλος όγδοης ενότητας.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)[](http://www.edulll.gr/)