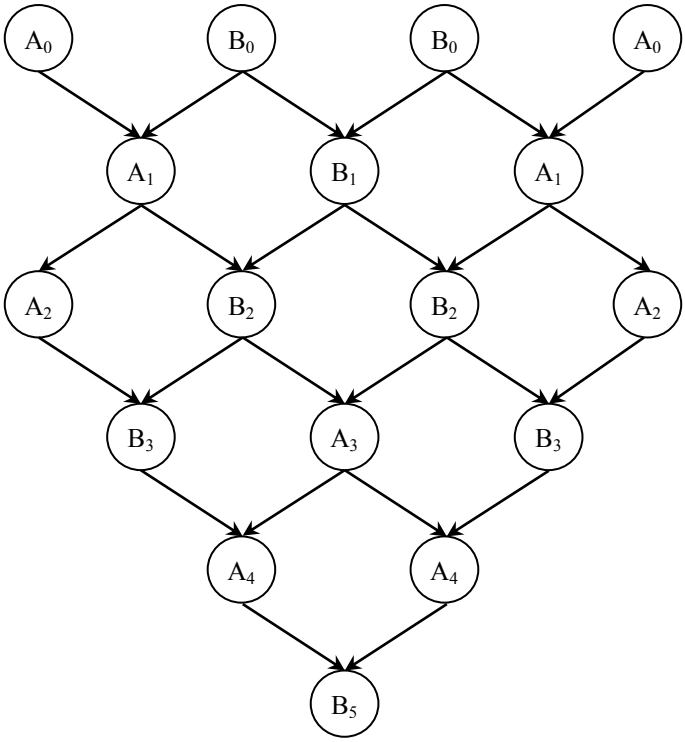
Α.
Στο παρακάτω διάγραμμα εμφανίζεται η εκτέλεση ενός παράλληλου αλγόριθμου που λύνει το ίδιο πρόβλημα με έναν ακολουθιακό αλγόριθμο χωρίς πλεονασμό. Τα A_i και B_i αντιστοιχούν σε ακολουθιακά υποέργα του επιπέδου i ενώ οι ακμές μεταξύ τους αντιστοιχούν σε εξαρτήσεις μεταξύ των διαφόρων υποέργων. Τα υποέργα A_i απαιτούν μία (1) χρονική μονάδα για να εκτελεστούν, ενώ τα υποέργα B_i απαιτούν δύο (2) χρονικές μονάδες για να εκτελεστούν.



I. Εκτελούμε τον αλγόριθμο αυτό σε ένα παράλληλο σύστημα με 4 επεξεργαστές με τον ακόλουθο τρόπο: Τα υποέργα του επιπέδου i αρχίζουν να εκτελούνται μόνο εφόσον έχει ολοκληρωθεί η εκτέλεση όλων των υποέργων του προηγούμενου επιπέδου i-1.
Αγνοώντας τους χρόνους επικοινωνίας μεταξύ των επεξεργαστών, συμπληρώστε τις παρακάτω τιμές:
Ακολουθιακός χρόνος εκτέλεσης : *25*
Παράλληλος χρόνος εκτέλεσης : *11*
Χρονοβελτίωση : *25/11*
Αποδοτικότητα : *25/(11*4)*
Αξιοποίηση : *25/44*

Λύση
Αν τον αλγόριθμο που απεικονίζεται στο σχήμα τον εκτελέσουμε σε μηχανή με έναν επεξεργαστή, κάθε φορά θα εκτελείται και από ένα υποέργο A_i ή B_i μέχρι να ολοκληρωθεί η εκτέλεση όλων των υποέργων, και του B_5. Δηλαδή, για να υπολογίσω τον ακολουθιακό χρόνο εκτέλεσης πρέπει να αθροίσω όλους τους επιμέρους χρόνους των υποέργων, αφού δεν υπάρχουν επιβαρύνσεις.

Άρα,
$T_{ser} = (A_0+B_0+B_0+A_0)_{\text{επίπεδο 0}} + (A_1+B_1+A_1)_{\text{επίπεδο 1}} + \ldots + (B_5)_{\text{επίπεδο 5}} = 25[\text{χρονικές μονάδες}]$

Υπάρχουν όμως εξαρτήσεις μεταξύ των υποέργων που βρίσκονται σε διαφορετικά επίπεδα, αυτό εξάλλου καθορίζει, δηλαδή οι εξαρτήσεις, και τα διαφορετικά επίπεδα του αλγόριθμου όπως απεικονίζονται στο παραπάνω σχήμα. Αρά όταν έχουμε περισσότερους του ενός επεξεργαστές θα μπορεί ναι μεν να εκτελούν ταυτόχρονα διαφορετικά υποέργα αλλά θα πρέπει να έχουν ικανοποιηθεί η εξαρτήσεις του υπο-εκτέλεση έργου πριν αναλάβει κάποιος επεξεργαστής την εκτέλεση του.
Δηλαδή, αν εκτελέσω τον παραπάνω αλγόριθμο σε μηχανή με P=4 επεξεργαστές, θα μπορεί να εκτελούνται τα έργα σε κάθε επίπεδο ταυτόχρονα (παράλληλα) από τους επεξεργαστές του συστήματος. Θα πρέπει όμως, λόγω των εξαρτήσεων, να τελειώνει η εκτέλεση του πιο

χρονοβόρου υποέργου στο επίπεδο που εξελίσσεται η εκτέλεση, πριν αρχίσει η εκτέλεση άλλου υποέργου σε διαφορετικό επίπεδο.

Η παραπάνω εξήγηση, για να διευκολυνθούμε, απεικονίζεται στο ακόλουθο σχήμα.

T →

P_1	A_0		A_1		A_2		B_3	B_3	A_4	B_5	B_5
P_2	B_0	B_0	B_1	B_1	B_2	B_2	A_3		A_4		
P_3	B_0	B_0	A_1		B_2	B_2	B_3	B_3			
P_4	A_0				A_2						

Κάθε θέση του πίνακα αντιστοιχεί σε μία χρονική μονάδα. Όταν εκτελείται έργο A_i χρεώνουμε μία χρονική μονάδα ενώ όταν εκτελείται έργο B_i χρεώνουμε δύο χρονικές μονάδες. Σε κάθε επίπεδο θα πρέπει να ολοκληρώνεται το πιο χρονοβόρο έργο πριν αρχίσει η εκτέλεση υποέργου στο επόμενο επίπεδο. Άρα, όσοι επεξεργαστές στο επίπεδο (i) εκτέλεσαν έργο A_i θα περιμένουν άεργοι, για μία χρονική μονάδα, μέχρι να ολοκληρωθεί η εκτέλεση όλων των υποέργων B_i.Έτσι προκύπτει ο πιο πάνω πίνακας.
Βλέπουμε λοιπόν ότι αν εκτελέσουμε τον αλγόριθμό μας σε μηχανή με τέσσερις επεξεργαστές, θα υπάρχουν χρονικές στιγμές που εκτελούν όλοι οι επεξεργαστές, υπάρχουν όμως και στιγμές που εκτελούν ένας, δύο ή τρεις επεξεργαστές ενώ οι υπόλοιποι, λόγω των εξαρτήσεων όπως εξηγήσαμε πιο πάνω, είναι άεργοι.

Άρα, ο παράλληλος χρόνος είναι ο χρόνος που χρειάστηκε η παράλληλη (P=4) μηχανή μας να εκτελέσει τον παράλληλο αλγόριθμο μέχρι και το τελευταίο του υποέργο, ανεξάρτητα αν όλοι οι επεξεργαστές εκτελούσαν καθ' όλη τη διάρκεια εκτέλεσης του αλγορίθμου.

Άρα,

T_{par}= 11 [χρονικές μονάδες],

Χρονοβελτίωση (speedup), $S=T_{ser}/T_{par}$ = 25/11

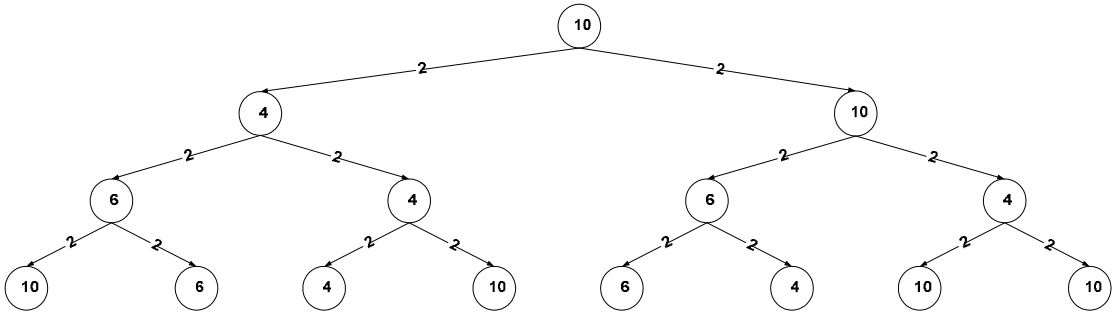
Αποδοτικότητα = $S/ P= (T_{ser}/T_{par})/P= T_{ser}/P*T_{par}$ =25/(4*11)

Αξιοποίηση είναι το δαπανούμενο έργο των επεξεργαστών στη διάρκεια του χρόνου προς το ωφέλιμο έργο στην ίδια χρονική διάρκεια. Αυτό εύκολα βγαίνει από τον πίνακα υπολογίζοντας τις χρονικές στιγμές που πραγματικά εκτελούν οι επεξεργαστές προς το συνολικό χρόνο που διαθέσαμε το σύστημα στην εκτέλεση του συγκεκριμένου αλγορίθμου. Αν δηλαδή μετρήσω όλα τα κουτάκια του πίνακα, όπου δηλαδή υπάρχει Α ή Β (25), προς το σύνολο των χρονικών στιγμών για όλους τους επεξεργαστές (4*11=44).

Αξιοποίηση = 25/44

Β.

Το δυαδικό δέντρο που φαίνεται στο σχήμα, αναπαριστά την παράλληλη έκδοση ενός αλγορίθμου του οποίου η αντίστοιχη ακολουθιακή έκδοση χρειάζεται 380msec για να ολοκληρωθεί. Τα βάρη που φαίνονται στους κόμβους και στις ακμές του δέντρου δηλώνουν υπολογισμό (computation) και επικοινωνία (communication) αντίστοιχα, σε msec. Θεωρώντας ότι έχουμε στη διάθεση μας πολυεπεξεργαστική μηχανή κατανεμημένης μνήμης με τέσσερις επεξεργαστές. Ζητούνται να υπολογιστούν: α) ο παράλληλος χρόνος εκτέλεσης του αλγορίθμου. β) η χρονοβελτίωση. γ) η αποδοτικότητα. δ) η αξιοποίηση. Θεωρείστε επίσης ότι δεν υπάρχουν επιβαρύνσεις άλλες εκτός τη επικοινωνίας, που όπως φαίνεται στο σχήμα είναι πάντα 2msec.

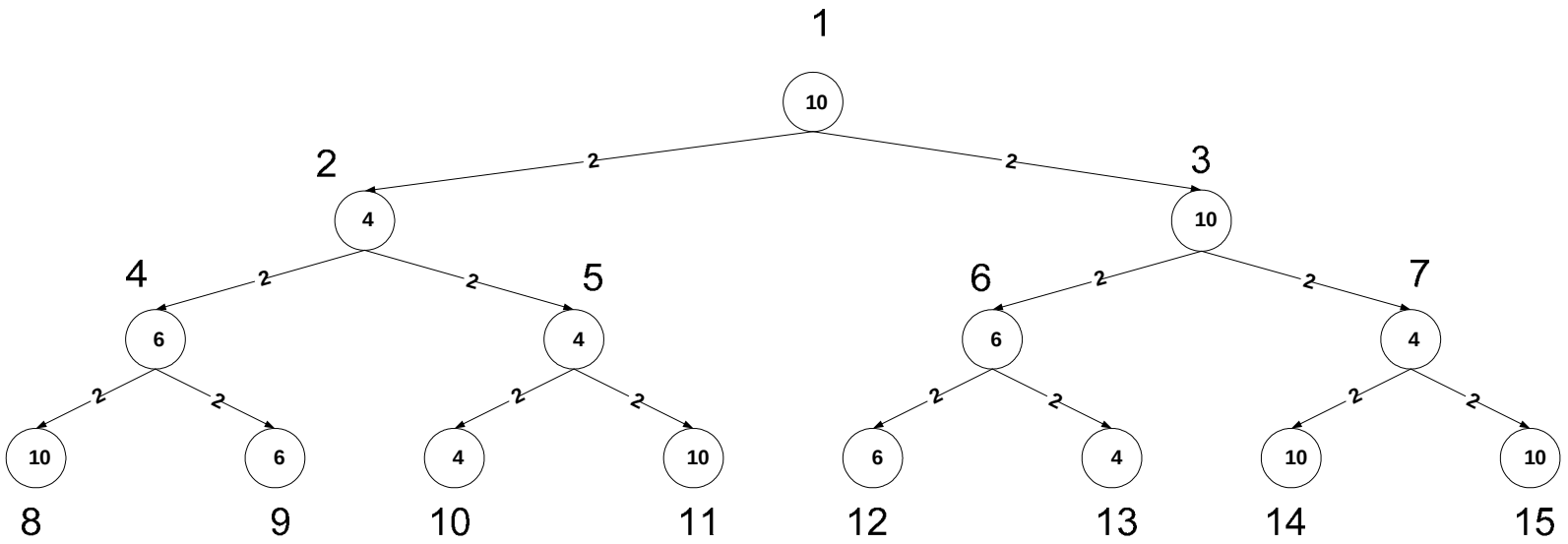


Λύση

Στην άσκηση αυτή έχουμε πάλι να εκτελέσουμε ένα παράλληλο αλγόριθμο για τον οποίο μας δίνονται: τα υποέργα, οι εξαρτήσεις τους, οι επιβαρύνσεις λόγω τις επικοινωνίας και η αρχιτεκτονική της παράλληλης μηχανής. Ζητάμε να υπολογίσουμε τα ίδια μεγέθη όπως και στην προηγούμενη άσκηση.

Για να διευκολυνθούμε φτιάχνουμε ένα πίνακα με πέντε σειρές και πολλές στήλες, περισσότερες από 40, γιατί αν διατρέξουμε τον κλάδο του δέντρου με τα μεγαλύτερα βάρη (χρόνοι εκτέλεσης των υποέργων) θα δούμε ότι ο κλάδος με χρόνο 40, είναι ο αριστερός (υπολογίζοντας του ωφέλιμους χρόνους επεξεργασίας αλλά και τις επιβαρύνσεις). Άρα είναι πιθανό να έχουμε εκτέλεση για τουλάχιστον 40 χρονικές μονάδες. Στην αρχή της πρώτης στήλης του πίνακα βάζουμε Τ για την καταγραφή των χρονικών στιγμών που θα εκτελεί η μηχανή μας. Στη δεύτερη θέση της πρώτης στήλης βάζουμε P1, για να σημειώνουμε τις χρονικές στιγμές που ο P1 θα εκτελεί. Αντίστοιχα στη τρίτη θέση P2 κ.ο.κ..

Στη συνέχεια αριθμούμε τους κόμβους του δένδρου (τα υποέργα) όπως φαίνεται στο επόμενο σχήμα με τους μεγάλους, έντονους αριθμούς. Οι αριθμοί αυτοί δηλώνουν τη σειρά με την οποία θα εκτελεστούν τα υποέργα από τους επεξεργαστές του συστήματος μας στη διάρκεια του χρόνου.



Αρχίζει η εκτέλεση του παράλληλου αλγορίθμου. Ο πρώτος επεξεργαστής P1 αρχίζει να εκτελεί το πρώτο υποέργο (**1**) που βρίσκεται στη κορυφή του δένδρου. Το υποέργο αυτό χρειάζεται δέκα msec για να ολοκληρωθεί. Κατά τη διάρκεια εκτέλεσης του πρώτου υποέργου οι υπόλοιποι τρεις επεξεργαστές του συστήματος μας παραμένουν ανενεργοί. Στον πίνακα που έχουμε ετοιμάσει καταγράφουμε στη σειρά που αντιστοιχεί στον επεξεργαστή P1, τον αριθμό που αντιστοιχεί στο πρώτο υποέργο (**1**) δέκα φορές υποδηλώνοντας τις δέκα χρονικές στιγμές που απαιτεί η εκτέλεση του πρώτου υποέργου. Στη συνέχεια ο επεξεργαστής P1 προωθεί τα αποτελέσματά του στους επεξεργαστές που θα εκτελέσουν τα επόμενα υποέργα (**2)** και (**3)**. Η προώθηση αυτή των αποτελεσμάτων είναι η επικοινωνία η οποία έχει επιβάρυνση 2msec. Έτσι στον πίνακα μας γράφουμε το **C** δύο φορές για να δηλώσουμε το χρόνο επικοινωνίας (**C**ommunication time). Ο P1 με την ίδια επιβάρυνση των 2msec προωθεί το αποτέλεσμα του και στους δύο επεξεργαστές[1] που θα εκτελέσουν τα υποέργα (**2)** και (**3).** Συμβαίνει όμως ο ίδιος επεξεργαστής P1 να συνεχίζει την εκτέλεση του υποέργου (**2**), ενώ ο δεύτερος επεξεργαστής αρχίζει την ίδια στιγμή να εκτελεί το υποέργο (**3**). Με τον ίδιο τρόπο τώρα καταγράφουμε στον πίνακα μας τα αντίστοιχα νούμερα στις αντίστοιχες θέσεις του για όσες χρονικές στιγμές αντιστοιχούν σε κάθε υποέργο που εκτελείται. Μετά την ολοκλήρωση του κάθε υποέργου ακολουθεί η επικοινωνία για την μεταφορά των δεδομένων. Με τον ίδιο τρόπο εκτελούμε όλα τα έργα και καταγράφουμε τις χρονικές στιγμές και τις επικοινωνίες τους στις αντίστοιχες θέσεις του πίνακα μας.

[1] Σωστό θα το ελάμβανα επίσης ακόμα και αν κάποιος χρησιμοποιούσε επιβάρυνση 2msec για επικοινωνία με κάθε υποέργο/ επεξεργαστή.

Προσοχή!, όταν πλέον η εκτέλεση φτάσει στα φύλα του δένδρου μπορεί τα έργα να εκτελούνται το ένα μετά το άλλο, χωρίς όμως να υπάρχει επικοινωνία ούτε άλλη επιβάρυνση. Έτσι στον πίνακα μας φαίνεται η εκτέλεση όλων των υποέργων του αλγορίθμου που απεικονίζεται στο διάγραμμα που δίνει ή άσκηση.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
P1	1	1	1	1	1	1	1	1	1	1	C	C	2	2	2	2	C	C	4	4	4	4	4	4	C	C	8	8	8	8	8	8	8	8	8	8	12	12	12	12	12	12		
P2													3	3	3	3	3	3	3	3	3	3	C	C	6	6	6	6	6	6	C	C	14	14	14	14	14	14	14	14	14	14		
P3																			5	5	5	5	C	C	7	7	7	7	C	C	15	15	15	5	15	15	15	15	15	15	15	10	10	10
P4																									11	11	11	11	11	11	11	11	11	11	9	9	9	9	9	9	13	13	13	13

Ο ακολουθιακός χρόνος μας δίνεται T_{ser}=380msec.

Ο παράλληλος χρόνος είναι T_{par}=44msec

Από τον πίνακα μας, φαίνεται ότι ο πρώτος επεξεργαστής αρχίζει να εκτελεί την πρώτη χρονική στιγμή και το συνολικό έργο ολοκληρώνεται μετά και την εκτέλεση των υποέργων (**10)** και (**13)** αντίστοιχα από τους επεξεργαστές P3 και P4. Η ολοκλήρωση των υποέργων (**10**) και (**13)** γίνεται την $44^{η}$ χρονική στιγμή. Τη στιγμή αυτή ολοκληρώνεται και η εκτέλεση του παράλληλου αλγόριθμου. Άρα, αυτός είναι και ο συνολικός χρόνος εκτέλεσης του αλγορίθμου μας.

Παρατηρούμε επίσης ότι η ταυτόχρονή εκτέλεση δύο επεξεργαστών αρχίζει την $13^{η}$ χρονική στιγμή ενώ από την $24^{η}$ χρονική στιγμή και μέχρι την $42^{η}$ η αξιοποίηση της κατανεμημένης μηχανής μας είναι σχεδόν 100%. Σχεδόν, γιατί βέβαια υπάρχουν και χρόνοι επικοινωνίας μέσα σ' αυτό το χρονικό διάστημα. Με αποτέλεσμα να αφήνουν για συνολικά έξι (6) χρονικές μονάδες άεργους ορισμένους επεξεργαστές.

Σύμφωνα με τα όσα παραθέτουμε στην θεωρεία, την λύση της προηγούμενης άσκησης και την παραπάνω παρατήρηση υπολογίστε την **Αξιοποίηση** και την **Αποδοτικότητα** μόνοι σας.

Χρονοβελτίωση (speedup), $S=T_{ser}/T_{par}$ = 380/44

Παρατηρούμε στο σημείο αυτό ότι η χρονοβελτίωση S>>P, ενώ σύμφωνα με τη θεωρεία πρέπει S<=P, σύμφωνοι. Όμως στο μάθημα το είχα επαναλάβει αρκετές φορές, ότι υπάρχει η πιθανότητα για λίγους κατανεμημένους συνήθως αλγορίθμους να παρουσιάζουν το φαινόμενο αυτό, το οποίο ονομάζουμε superlinear speedup. Στην άσκηση μας, έδωσα πολύ μικρό παράλληλο χρόνο επίτηδες για να κάνω ιδιαίτερα εμφανές το θέμα αυτό. Τέλος, όσοι την ώρα της εξέτασης με ρώτησαν για το θέμα αυτό τους εξήγησα ότι είναι σωστό και μπορεί όντως να συμβεί. Μάλιστα κάθε φορά που έγινε παρόμοιο σχόλιο, και έγινε πολλές φορές, η απάντηση δινόταν φωναχτά για να την ακούν όλοι.-

Προσοχή!, η σειρά εκτέλεσης των υποέργων παίζει σημαντικό ρόλο στον σωστό υπολογισμό του χρόνου παράλληλης εκτέλεσης. Αν δηλαδή εκτελέσουμε τα έργα με διαφορετική σειρά υπάρχει πιθανότητα να εκτελεστούν σε χρόνο που θα είναι μεγαλύτερος του 44msec, αυτό είναι λάθος. Επιλέγουμε την καταλληλότερη σειρά εκτέλεσης των υποέργων, για τον υπολογισμό του παράλληλου χρόνου εκτέλεσης.

Γ.
Σας δίνονται τα παρακάτω τμήματα κώδικα σε *C*:

Νήμα παραγωγός

```
1  while(1){
2     X = 4;
3  }
4
5
6
7
```

Νήματα καταναλωτές

```
1  for(i=0; I < 5; i++){
2    while(x==0){
3      pthread_cond_wait(&ready, &lock);
4    }
5    x = x – 1;
6    printf("Νήμα %d, x = %d", id, x);
7  }
```

Το πρώτο τμήμα εκτελείται από ένα νήμα, το νήμα παραγωγό και το δεύτερο τμήμα από 4 νήματα τα νήματα καταναλωτές. Τα τμήματα κώδικα είναι ημιτελή. Πρέπει να τα συμπληρώσετε ούτως ώστε: Μόλις το νήμα παραγωγός θέτει την τιμή 4 στη μεταβλητή x να ειδοποιεί τα νήματα καταναλωτές και να τα περιμένει να ολοκληρώσουν μια επανάληψη (for loop). Αντίστοιχα μόλις ένα νήμα καταναλωτής ειδοποιείται θα πρέπει να μειώνει την μεταβλητή x, να εκτυπώνει τη νέα τιμή και να περιμένει να ολοκληρώσουν όλα τα νήματα μια επανάληψη. Για την υλοποίηση θα πρέπει να χρησιμοποιήσετε συναρτήσεις για αμοιβαίο αποκλεισμό, ειδοποίηση νημάτων και για αναμονή σε barrier. Υπενθυμίζεται ότι κατά την αναμονή σε μια μεταβλητή υπό συνθήκη το lock παραμένει ελεύθερο, ενώ μετά την επιστροφή από την αναμονή βρίσκεται δεσμευμένο.

Στη συνέχεια σας δίνονται τα πρωτότυπα των συναρτήσεων που θα χρησιμοποιήσετε:

```
int pthread_mutex_lock(pthread_mutex_t *);
int pthread_mutex_unlock(pthread_mutex_t *);
int pthread_cond_broadcast(pthread_cond_t *);
int pthread_barrier_wait(pthread_barrier_t *);
```

Λύση

Νήμα παραγωγός

```
1  while(1){
2    pthread_mutex_lock(&lock);
3    x = 4;
4    pthread_mutex_unlock(&lock);
5    pthread_cond_broadcast(&ready);
6    pthread_barrier_wait(&bar);
7  }
```

Νήματα καταναλωτές

```
1  for(i=0; i < 5; i++){
     pthread_mutex_lock(&lk);
2    While(x==0){
3      pthread_cond_wait(&ready, &lock);
4    }
5    x = x – 1;
6    printf("Νήμα %d, x = %d", id, x);
     pthread_mutex_unlock(&lock);
     pthread_barrier_wait(&bar);
7  }
```

Δ.
Το τέταρτο θέμα είναι ακριβώς ίδιο με αυτό που υπάρχει στις διαφάνειες του μαθήματος.