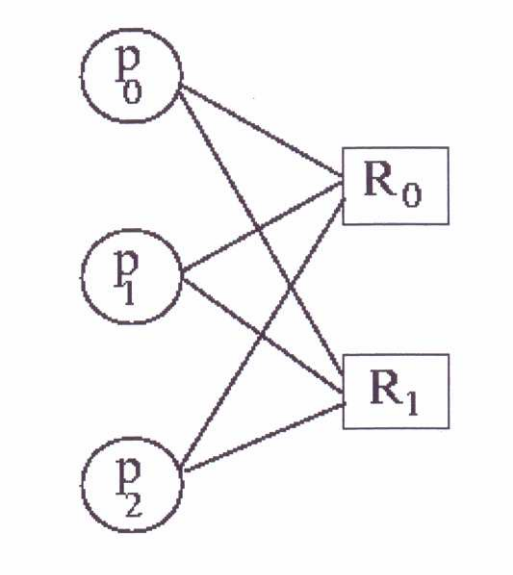
Αμοιβαίος Αποκλεισμός

Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης



- Το σύστημα περιέχει n διεργασίες p_0, …, p_{n-1} και m καταχωρητές R_0, …, R_{m-1}.
- Κάθε διεργασία μοντελοποιείται ως μια μηχανή καταστάσεων (χωρίς όμως inbuf και outbuf πίνακες).
- Κάθε καταχωρητής έχει ένα τύπο που καθορίζει το σύνολο των τιμών που μπορούν να αποθηκευτούν στον καταχωρητή, καθώς και το σύνολο των λειτουργιών που μπορούν να εκτελεστούν ατομικά σε αυτόν.

Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης

Παράδειγμα

Καταχωρητής ανάγνωσης-εγγραφής (read-write register) R με ακέραιες τιμές

Σύνολο τιμών

N (το σύνολο όλων των ακεραίων)

Λειτουργίες

- *read(R):* επιστρέφει την τρέχουσα τιμή του R (χωρίς να προκαλέσει κάποια μεταβολή σε αυτή)
- *write(R, v):* εγγράφει v στον R και επιστρέφει ack

Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης

Καθολικές Καταστάσεις

C = (q_0,…, q_{n-1}, r_0, …, r_{m-1}), όπου:

- q_i είναι η κατάσταση της p_i, και
- r_j είναι η τιμή του καταχωρητή R_j ⇒ η C περιέχει και τις τιμές των καταχωρητών

Γεγονότα

Υπάρχουν μόνο υπολογιστικά γεγονότα. Σε κάθε υπολογιστικό γεγονός:

- η p επιλέγει (βασισμένη στην τρέχουσα κατάστασή της) για προσπέλαση έναν από τους καταχωρητές, έστω τον R
- η p εκτελεί την καθορισμένη λειτουργία στον R
- η κατάσταση της p μεταβάλλεται σύμφωνα με τη συνάρτηση μετάβασής της. Η μετάβαση γίνεται λαμβάνοντας υπόψη την τρέχουσα κατάσταση της p και την τιμή που επιστράφηκε από την λειτουργία που εφαρμόστηκε στον R

Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης – Μετρικά Πολυπλοκότητας

Χωρική Πολυπλοκότητα (αντί για πολυπλοκότητα επικοινωνίας)

Μέγεθος κοινής μνήμης που χρησιμοποιείται για την επίλυση ενός προβλήματος

Χρονική Πολυπλοκότητα

Δεν έχει νόημα! Τι συμβαίνει όταν έχουμε ανταγωνισμό στην πρόσβαση μιας κοινής μεταβλητής (και απαιτείται συγχρονισμός); $\Rightarrow$ συμφόρηση! Σημαντικό θέμα στο οποίο γίνεται έρευνα σήμερα!

Συνήθως, μετράμε τον αριθμό των βημάτων. Ενδιαφερόμαστε για το αν ο αριθμός των βημάτων είναι πεπερασμένος, μη-πεπερασμένος ή φραγμένος.

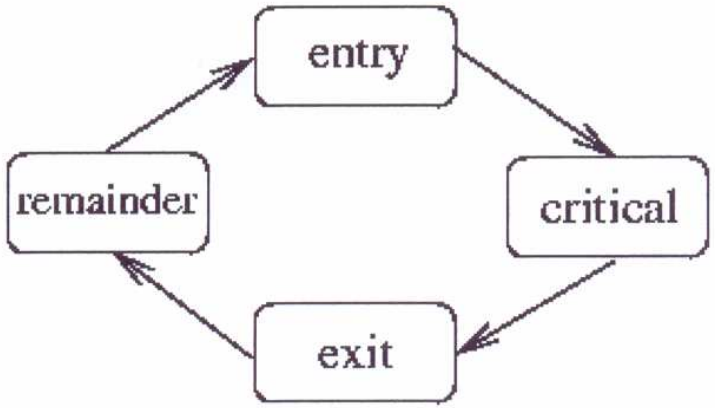
Το Πρόβλημα του Αμοιβαίου Αποκλεισμού

Το πρόβλημα μοντελοποιεί τον διαμοιρασμό ενός μοναδικού, μη-προεκχωρητικού πόρου μεταξύ n χρηστών. Ο πόρος μπορεί να είναι:\

- ο εκτυπωτής
- κάποια άλλη συσκευή εξόδου
- μια βάση δεδομένων ή κάποια δομή δεδομένων
- οτιδήποτε απαιτεί αποκλειστική πρόσβαση προκειμένου να είναι εγγυημένο ότι το αποτέλεσμα θα είναι σωστό

Το Πρόβλημα του Αμοιβαίου Αποκλεισμού

Τμήματα Κώδικα



- Ο χρήστης που την τρέχουσα χρονική στιγμή προσβαίνει τον πόρο βρίσκεται στο **κρίσιμο τμήμα** του.
- Χρήστες που την τρέχουσα χρονική στιγμή δεν ενδιαφέρονται να χρησιμοποιήσουν τον πόρο βρίσκονται στο **μη-κρίσιμο τμήμα** τους.
- Χρήστες που ενδιαφέρονται να χρησιμοποιήσουν τον πόρο και προσπαθούν να αποκτήσουν πρόσβαση σε αυτόν βρίσκονται στο **τμήμα εισόδου** τους.
- Τέλος, χρήστες που μόλις τελείωσαν τη χρήση του πόρου βρίσκονται στο **τμήμα εξόδου** τους (εκτελούν κατάλληλες ενέργειες για να επιτρέψουν σε άλλες διεργασίες να χρησιμοποιήσουν τον πόρο).

Αλγόριθμοι που επιλύουν το πρόβλημα του Αμοιβαίου Αποκλεισμού

Αμοιβαίος Αποκλεισμός

Σε κάθε καθολική κατάσταση οποιαδήποτε εκτέλεσης, το πολύ μια διεργασία βρίσκεται στο κρίσιμο τμήμα της.

Αποφυγή Αδιεξόδου

Αν σε κάποια καθολική κατάσταση (οποιασδήποτε εκτέλεσης ενός αλγορίθμου που επιλύει το πρόβλημα) μια διεργασία βρίσκεται στο τμήμα εισόδου της τότε υπάρχει καθολική κατάσταση σε επόμενο σημείο της εκτέλεσης στην οποία κάποια διεργασία (η ίδια ή άλλη) βρίσκεται στο κρίσιμο τμήμα της.

Αποφυγή Παρατεταμένης Στέρησης

Αν σε κάποια καθολική κατάσταση (οποιασδήποτε εκτέλεσης) μια διεργασία βρίσκεται στο τμήμα εισόδου της τότε υπάρχει καθολική κατάσταση σε επόμενο σημείο της εκτέλεσης στην οποία η ίδια αυτή διεργασία βρίσκεται στο κρίσιμο τμήμα της.

Ποια ιδιότητα από τις δύο τελευταίες είναι πιο ισχυρή;

Αλγόριθμοι που επιλύουν το πρόβλημα του Αμοιβαίου Αποκλεισμού

Υποθέσεις

- Οι μεταβλητές (διαμοιραζόμενες ή μη) που προσπελαύνονται στο τμήμα εισόδου ή στο τμήμα εξόδου δεν προσπελαύνονται σε κανένα από τα άλλα δύο τμήματα.
- Κανένας επεξεργαστής δεν μένει το κρίσιμο τμήμα για πάντα.

Επιτρεπτές Εκτελέσεις

Το τμήμα εξόδου πρέπει να αποτελείται από πεπερασμένο αριθμό βημάτων.

Αλγόριθμοι Αμοιβαίου Αποκλεισμού που χρησιμοποιούν Ισχυρούς Καταχωρητές

Κύρια αποτελέσματα

- Η παροχή ενός και μόνο bit κοινής μνήμης είναι αρκετή για την επίτευξη αμοιβαίου αποκλεισμού και αποφυγής αδιεξόδου.
- Θ(logn) bits απαιτούνται για την επίτευξη αποφυγής παρατεταμένης στέρησης.

Binary Test&Set Registers	**Read-Modify-Write Registers**

Σύνολο Τιμών: {0,1}
 	
Λειτουργίες
boolean **Test&Set**(register T):
value tmp;
tmp = read(T);
write(T,1);
return (tmp);
 	
Reset(T): V = 0;
	Σύνολο Τιμών: οτιδήποτε οποιουδήποτε μεγέθους
	
	Λειτουργίες
	value RMW(register V, function f):
	value tmp;
	tmp = read(V);
	V = f(V);
	return(tmp);
	
	Αυτός ο τύπος καταχωρητή είναι πολύ ισχυρός!!!

Επίτευξη Αμοιβαίου Αποκλεισμού με χρήση ενός Binary Test&Set Καταχωρητή

Έστω Τ ο δυαδικός (binary) Test&Set καταχωρητής, αρχική τιμή 0.

Τμήμα Εισόδου:

```
value t = Test&Set(T);
while (t== 1) t = Test&Set(T);          // wait until Test&Set(T) == 0;
```

<κρίσιμο τμήμα>;

Τμήμα εξόδου:

```
reset(T);
```

<μη-κρίσιμο τμήμα>;

Θεώρημα: Ο παραπάνω αλγόριθμος εγγυάται αμοιβαίο αποκλεισμό και αποφυγή αδιεξόδου χρησιμοποιώντας έναν Binary Test&Set καταχωρητή.

Εγγυάται ο αλγόριθμος αυτός αποφυγή παρατεταμένης στέρησης;

Επίτευξη Αμοιβαίου Αποκλεισμού με χρήση ενός RMW Καταχωρητή

Ο αλγόριθμος χρησιμοποιεί μια κυκλική ουρά μήκους n.

Χρησιμοποιείται επίσης ένας RMW καταχωρητής V που αποτελείται από δύο πεδία V.first και V.last που δείχνουν το πρώτο και το τελευταίο στοιχείο της ουράς αντίστοιχα. Αρχικά, V.first = V.last = 0.

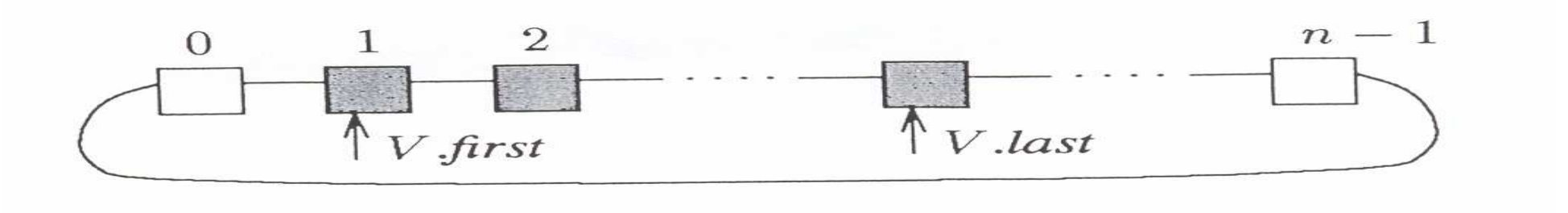
Μια διεργασία p που θέλει να εισέλθει στο κρίσιμο τμήμα της:

- Διαβάζει την τιμή του V σε μια τοπική μεταβλητή position και αυξάνει την τιμή του V.last κατά 1.
- Στη συνέχεια η p διαβάζει επαναληπτικά τον V μέχρι να δει ότι V.first == position.last

Κατά την έξοδο από το κρίσιμο τμήμα, η p αυξάνει το v.first κατά 1.

Θεώρημα: Υπάρχει αλγόριθμος αμοιβαίου αποκλεισμού που εγγυάται αποφυγή παρατεταμένης στέρησης και χρησιμοποιεί έναν RMW καταχωρητή των (2 \ceil{$\log_2$n}) bits.

Επίτευξη Αμοιβαίου Αποκλεισμού με χρήση ενός RMW Καταχωρητή



Κώδικας για κάθε διεργασία p

Αρχικά V = <0,0>;

Τμήμα εισόδου:

position = RMW(V, <V.first, V.last + 1>);
repeat
    queue = RMW(V,V);
until (queue.first = position.last);

<κρίσιμο τμήμα>;

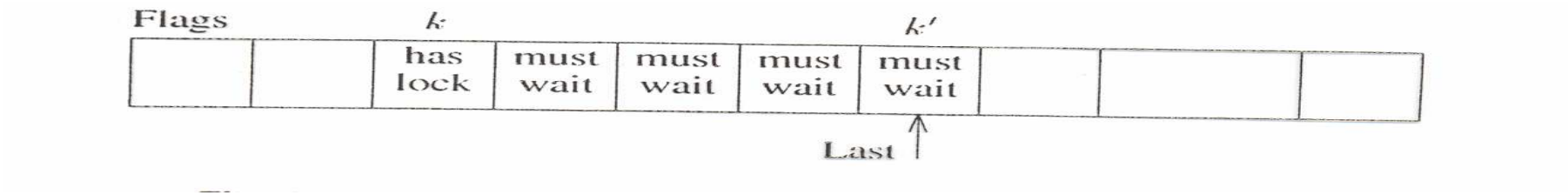
Τμήμα εξόδου:

RMW(V,<V.first+1, V.last>);

<μη-κρίσιμο τμήμα>;

Ποιο είναι το κύριο αρνητικό αυτού του αλγορίθμου;

Επίτευξη Αμοιβαίου Αποκλεισμού με αναμονή σε τοπικές μεταβλητές



Κώδικας για κάθε διεργασία p

Αρχικά Last = 0; Flags[0] = has-lock;
Flags[i] = must-wait, 0 < i < n.

Τμήμα Εισόδου:

my-place = RMW(Last, (Last+1) mod n);
wait until (Flags[my-place] == has-lock);
Flags[my-place] = must-wait;

<κρίσιμο τμήμα>;

Τμήμα εξόδου:

Flags[(my-place + 1) mod n] = has-lock;

<μη-κρίσιμο τμήμα>;

Λήμμα: Σε κάθε εκτέλεση του αλγορίθμου ισχύουν τα εξής για τον πίνακα Flags:

1. το πολύ ένα στοιχείο του έχει την τιμή has-lock,
2. αν κανένα στοιχείο του δεν έχει την τιμή has-lock, κάποια διεργασία βρίσκεται στο κρίσιμο τμήμα της
3. Αν Flags[k] == has-lock τότε ακριβώς (k-Last-1) mod n διεργασίες βρίσκονται στο τμήμα εισόδου και κάθε μια αναμένει εξετάζοντας ένα διαφορετικό στοιχείο του Flags.

Αμοιβαίος Αποκλεισμός χρησιμοποιώντας Read-Write Καταχωρητές

Κύρια Αποτελέσματα

Αλγόριθμοι

1. Ένας αλγόριθμος αμοιβαίου αποκλεισμού για n διεργασίες που εγγυάται αποφυγή παρατεταμένης στέρησης και χρησιμοποιεί O(n) RW καταχωρητές μη-πεπερασμένης χωρητικότητας.
2. Ένας αλγόριθμος αμοιβαίου αποκλεισμού για n διεργασίες που εγγυάται αποφυγή παρατεταμένης στέρησης και χρησιμοποιεί O(n) RW καταχωρητές πεπερασμένης χωρητικότητας.

Αρνητικά Αποτελέσματα

1. Κάθε αλγόριθμος αμοιβαίου αποκλεισμού χρειάζεται n RW καταχωρητές ανεξάρτητα από το μέγεθος τους.

Ο Αλγόριθμος Bakery

for each i, 0 ≤i ≤ n-1:

Choosing[i]: έχει τιμή TRUE όσο η p_i προσπαθεί να επιλέξει αριθμό

Number[i]: ο αριθμός (εισιτήριο) που επιλέγεται από την p_i

Κώδικας για την p_i, 0 ≤i ≤ n-1

Αρχικά, Number[i] = 0, και

Choosing[i] = FALSE, για κάθε i, 0 ≤i ≤ n-1

Τμήμα Εισόδου:

Choosing[i] = TRUE;
Number[i] = max{Number[0], …, Number[n-1]}+1;
Choosing[i] = FALSE;
for j = 0 to n-1, j ≠ i, do
    wait until Choosing[j] == FALSE;
    wait until ((Number[j] == 0) OR ((Number[j], j) > (Number[i], i)));

<κρίσιμο τμήμα>;

Τμήμα Εξόδου: Number[i] = 0;

<μη-κρίσιμο τμήμα>;

Ο Αλγόριθμος Bakery

Λήμμα

Σε κάθε καθολική κατάσταση C οποιασδήποτε εκτέλεσης a του αλγορίθμου, αν η pi βρίσκεται στο κρίσιμο τμήμα της και για κάθε k ≠ i, Number[k] ≠ 0, τότε (Number[k],k) > (Number[i],i).

Απόδειξη (σύντομη περιγραφή)

- Number[i] > 0
- η p_i έχει τελειώσει την εκτέλεση της for
- **Περίπτωση 1**: η p_i βλέπει Number[k] == 0
- **Περίπτωση 2**: η p_i βλέπει (Number[k],k) > (Number[i],i)

Θεώρημα

Ο αλγόριθμος Bakery παρέχει αμοιβαίο αποκλεισμό.

Ο Αλγόριθμος Bakery

Αποφυγή Παρατεταμένης Στέρησης

Θεώρημα: Ο αλγόριθμος Bakery επιτυγχάνει αποφυγή από παρατεταμένη στέρηση.

Απόδειξη (σύντομη περιγραφή): Ας υποθέσουμε ότι υπάρχει τουλάχιστον μια διεργασία που υφίσταται παρατεταμένη στέρηση.

Όλες οι διεργασίες που επιθυμούν να εισέλθουν στο κρίσιμο τμήμα τους θα τελειώσουν την επιλογή αριθμού.

Έστω ότι p_j είναι διεργασία με το μικρότερο ζεύγος (number[j],j) που υφίσταται παρατεταμένη στέρηση.

Αν οποιαδήποτε διεργασία p_k εισέλθει στο κρίσιμο τμήμα της μετά την j, τότε Number[k] > Number[j].

Οποιαδήποτε διεργασία p_k με Number[k] < Number[j] θα εισέλθει στο κρίσιμο τμήμα της και θα εξέλθει αυτού.

Χωρική Πολυπλοκότητα

Ο αριθμός των διαμοιραζόμενων καταχωρητών είναι 2n. Οι n μεταβλητές Choosing είναι δυαδικές, ενώ οι Number αποθηκεύουν μη-πεπερασμένο αριθμό τιμών.

Ένας Αλγόριθμος Αμοιβαίου Αποκλεισμού για 2 Διεργασίες που χρησιμοποιεί πεπερασμένης χωρητικότητας καταχωρητές – Μη-συμμετρική έκδοση

- want[0]: εγγράφεται από τη διεργασία 0 και διαβάζεται από την 1. Αρχική τιμή 0, τίθεται στην τιμή 1 αν η 0 θέλει να εισέλθει στο κρίσιμο τμήμα της
- want[1]: συμμετρική της want[0]

Κώδικας για την p_0:

Τμήμα Εισόδου:

```
want[0] = 1;
wait until (want[1] == 0);
```

<κρίσιμο τμήμα>;

Τμήμα Εξόδου:

```
want[0] = 0;
```

<μη-κρίσιμο τμήμα>;

Κώδικας για την p_1:

Τμήμα Εισόδου:

```
1: want[1] = 0;
   wait until (want[0] == 0);
   want[1] = 1;
   if (want[0] == 1)
      goto line 1;
```

<κρίσιμο τμήμα>;

Τμήμα Εξόδου:

```
want[1] = 0;
```

<μη-κρίσιμο τμήμα>;

- Ο αλγόριθμος επιτυγχάνει αμοιβαίο αποκλεισμό και αποφυγή αδιεξόδου!
- Ο αλγόριθμος είναι μη-συμμετρικός: η p_1 εισέρχεται στο κρίσιμο τμήμα της μόνο αν η p_0 δεν ενδιαφέρεται να εισέλθει!

Ένας Αλγόριθμος Αμοιβαίου Αποκλεισμού για 2 Διεργασίες που χρησιμοποιεί πεπερασμένης χωρητικότητας καταχωρητές – Συμμετρική Έκδοση

Κώδικας για τη διεργασία p_i:

Τμήμα Εισόδου:

```
1: want[i] = 0;
2: wait until ((want[1-i] == 0) OR (priority == i));
3: want[i] = 1;
4: if (priority == 1-i) then {
5:          if  (want[1-i] == 1) then
                    goto line 1; }
6: else wait until (want[1-i] == 0);
```

<κρίσιμο τμήμα>;

Τμήμα Εξόδου:

```
7: priority = 1-i;

8: want[i] = 0;
```

<μη-κρίσιμο τμήμα>;

Ένας Αλγόριθμος Αμοιβαίου Αποκλεισμού για 2 Διεργασίες που χρησιμοποιεί πεπερασμένης χωρητικότητας καταχωρητές – Συμμετρική Έκδοση

Θεώρημα

Ο αλγόριθμος επιτυγχάνει αμοιβαίο αποκλεισμό.

Απόδειξη (sketch): *Γιατί ισχύει αυτό;*

- Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι και οι δύο διεργασίες βρίσκονται στο κρίσιμο τμήμα τους ταυτόχρονα $\Rightarrow$ want[0] = want[1] = 1.
- Ας υποθέσουμε, wlog, ότι η τελευταία εγγραφή της p_0 στην want[0] έπεται της τελευταίας εγγραφής της p_1 στο want[1].
- Διακρίνουμε περιπτώσεις (case analysis) και σε κάθε περίπτωση καταλήγουμε σε άτοπο.

Ένας Αλγόριθμος Αμοιβαίου Αποκλεισμού για 2 Διεργασίες που χρησιμοποιεί πεπερασμένης χωρητικότητας καταχωρητές – Συμμετρική Έκδοση

Θεώρημα

Ο αλγόριθμος εγγυάται αποφυγή αδιεξόδου.

Απόδειξη (sketch)

Ας υποθέσουμε ότι και κάποια/ες διεργασίες εκτελούν το τμήμα εισόδου τους, αλλά καμιά δεν εισέρχεται στο κρίσιμο τμήμα.

Περίπτωση 1: έστω ότι οι δύο διεργασίες είναι για πάντα στο τμήμα εισόδου. Έστω, wlog, ότι priority == 0.

Περίπτωση 2: μόνο μια από τις διεργασίες είναι στο τμήμα εισόδου. Αν κάποια διεργασία βρίσκεται στο κρίσιμο τμήμα δεν μπορεί να παραμείνει εκεί για πάντα.

Θεώρημα

Ο αλγόριθμος εγγυάται αποφυγή παρατεταμένης στέρησης.

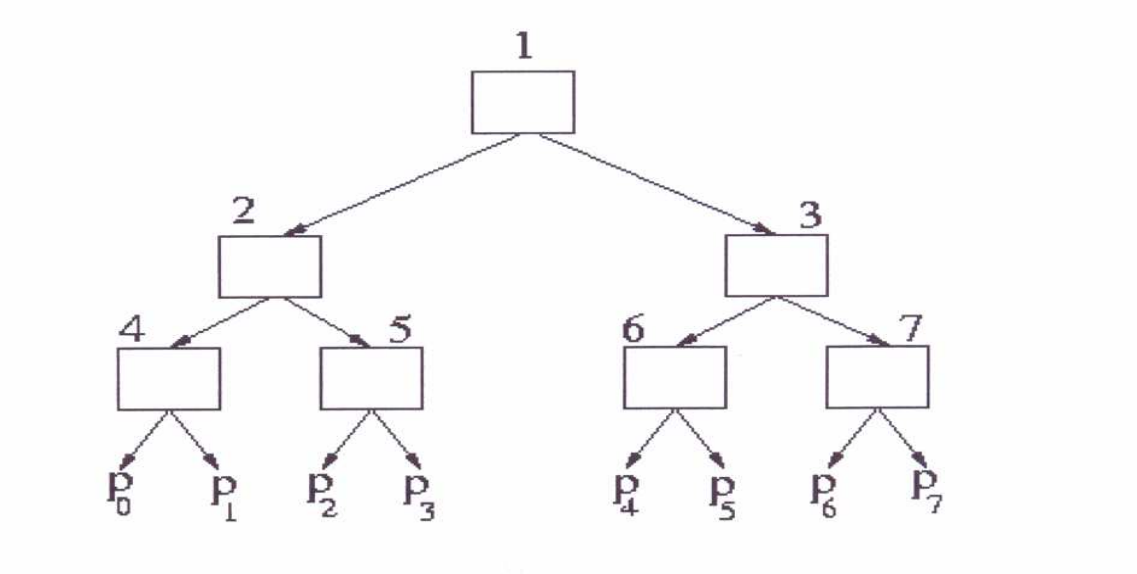
Απόδειξη (sketch)

Ας υποθέσουμε ότι κάποια διεργασία π.χ., η 0, υφίσταται παρατεταμένη στέρηση.

Περίπτωση 1: η διεργασία 1 εκτελεί την εντολή 7 κάποια στιγμή αργότερα. Από τότε και μετά ισχύει ότι priority == 0.

Περίπτωση 2: η διεργασία 1 δεν εκτελεί την εντολή 7.

Ο Αλγόριθμος Tournament για n διεργασίες



- Οι διεργασίες συναγωνίζονται σε ζευγάρια, χρησιμοποιώντας το συμμετρικό αλγόριθμο για 2 διεργασίες.
- Οι διεργασίες σχηματίζουν ένα δένδρο tournament (δηλαδή ένα πλήρες, δυαδικό δένδρο με n φύλλα, ένα για κάθε διεργασία). Κάθε διεργασία ξεκινά από κάποιο συγκεκριμένο φύλλο (εκείνο που της αναλογεί) του δένδρου.
- Σε κάθε επίπεδο του δένδρου, δύο διεργασίες συναγωνίζονται σε κάθε κόμβο. Μόνο οι νικητές συνεχίζουν στο επόμενο επίπεδο. Ο νικητής στον κόμβο ρίζα εισέρχεται στο κρίσιμο τμήμα του.

Ο Αλγόριθμος Tournament για n διεργασίες

```
procedure Node(v: integer, side: 0..1) {

    1: want^v[side] = 0;
    2: wait until ((want^v[1-side] == 0) OR
                   (priority^v == side));
    3: want^v[side] = 1;
    4: if (priority^v == 1-side) then {
    5:          if  (want^v[1-side] == 1) then
                    goto line 1; }
    6: else wait until (want^v[1-side] == 0);
    8: if (v == 1) then

    9:    <κρίσιμο τμήμα>;

    10: else Node(\lfloor v/2 \rfloor, v mod 2)

    11: priority^v = 1-side;
    12: want^v[side] = 0;

}
```

- Η ρίζα αριθμείται με το 1. Τ αριστερό παιδί ενός κόμβου με αριθμό v αριθμείται με τον αριθμό 2v και το δεξιό παιδί με τον 2v+1
- wantv[0], wantv[1], priorityv: μεταβλητές που έχουν συσχετισθεί με τον κόμβο με αριθμό v.
- Ο κόμβος i ξεκινά εκτελώντας την Node(2k+\lfloor i/2 \lceil, i mod 2), όπου k = \ceil{logn} -1.
- Ο αμοιβαίος αποκλεισμός εξασφαλίζεται από το γεγονός ότι ο συμμετρικός αλγόριθμος για 2 διεργασίες τον εγγυάται.
- Το ίδιο ισχύει και για την ιδιότητα αποφυγής παρατεταμένης στέρησης.

Χωρική Πολυπλοκότητα: 3n καταχωρητές

Ο Αλγόριθμος Tournament για n διεργασίες

Προβολή εκτέλεσης του αλγορίθμου tournament σε κάποιο κόμβο v: η ακολουθία βημάτων της εκτέλεσης που πραγματοποιούνται από κλήσης των Node(v,0) και Node(v,1).

Θα δείξουμε ότι:

Για κάθε v, η προβολή κάθε εκτέλεσης του δεντρικού αλγορίθμου στον v αποτελεί εκτέλεση του συμμετρικού αλγορίθμου αμοιβαίου αποκλεισμού για 2 διεργασίες, αν θεωρήσουμε ότι κάθε διεργασία που εκτελεί την Node(v,0) είναι η p_0 και κάθε διεργασία που εκτελεί την Node(v,1) είναι η p_1.

Ο Αλγόριθμος Tournament για n διεργασίες

Πιο φορμαλιστικά:

Έστω ότι $a = C_0\ \varphi_1\ C_1\ \varphi_2\ C_2\ \ldots$ είναι μια εκτέλεση του αλγορίθμου tournament.

Έστω ότι a^v είναι η ακολουθία από καθολικές καταστάσεις και γεγονότα που εναλλάσσονται:

$$D_0\ \pi_1\ D_1\ \pi_2\ D_2\ \ldots$$

που ορίζεται επαγωγικά ως εξής:

Βάση Επαγωγής: D_0 είναι η αρχική καθολική κατάσταση του συμμετρικού αλγορίθμου 2 διεργασιών

Επαγωγική Υπόθεση: Έστω ότι η a^v έχει οριστεί μέχρι και την καθολική κατάσταση D_{i-1}.

Επαγωγικό Βήμα: Έστω ότι $\varphi_j = k$ είναι το i-οστό γεγονός της a που αποτελεί βήμα είτε της Node(v,0) ή της Node(v,1) (έστω, wlog, της Node(v,0)).

Έστω ότι $\pi_i = 0$ και έστω ότι D_i είναι η καθολική κατάσταση στην οποία ισχύουν τα εξής:

- οι καταστάσεις των μεταβλητών είναι ίδιες με τις καταστάσεις των μεταβλητών του κόμβου v στην C_j
- η κατάσταση της p_1 είναι η ίδια με εκείνη στην D_{i-1}
- η κατάσταση της p_0 είναι η ίδια με την κατάσταση της p_k στην C_j εκτός από το ότι το id έχει αντικατασταθεί από το 0.

Ο Αλγόριθμος Tournament για n διεργασίες

Λήμμα

Για κάθε v, η a^v είναι εκτέλεση του συμμετρικού αλγορίθμου για 2 διεργασίες.

Απόδειξη

Οι κώδικες της Node(v,i) και του συμμετρικού αλγορίθμου για 2 διεργασίες είναι ίδιοι.

Αρκεί να αποδείξουμε ότι μόνο μια διεργασία εκτελεί εντολές της Node(v,i) σε κάθε σημείο της εκτέλεσης. Το αποδεικνύουμε με επαγωγή στο επίπεδο του v, ξεκινώντας από τα φύλλα.

Βάση Επαγωγής: Ισχύει εξ κατασκευής.

Επαγωγική Υπόθεση: Θεωρούμε οποιοδήποτε εσωτερικό κόμβο του δένδρου και υποθέτουμε ότι ο ισχυρισμός ισχύει για τους κόμβους παιδιά του.

Επαγωγικό Βήμα: Αποδ/με τον ισχυρισμό για τον v.

- Αν μια διεργασία εκτελεί εντολές π.χ., της Node(v,0), βρίσκεται στο κρίσιμο τμήμα για τον κόμβο που είναι το αριστερό παιδί του v.
- Από επαγωγική υπόθεση και αφού ο συμμετρικός αλγόριθμος για 2 διεργασίες εγγυάται αμοιβαίο αποκλεισμό, συνεπάγεται ότι μόνο μια διεργασία σε κάθε χρονική στιγμή βρίσκεται στο κρίσιμο τμήμα για το αριστερό παιδί του v.
- Αντίστοιχα είναι τα επιχειρήματα για την Node(v,1).

Ο Αλγόριθμος Tournament για n διεργασίες

Λήμμα

Για κάθε v, αν η a είναι επιτρεπτή εκτέλεση, τότε και η a^v είναι επιτρεπτή εκτέλεση.

Απόδειξη

- Αποδεικνύουμε ότι στην a^v δεν υπάρχει κάποια διεργασία που να μένει στο κρίσιμο τμήμα για πάντα.
- Η απόδειξη γίνεται με επαγωγή στο επίπεδο του v, ξεκινώντας από τη ρίζα.

Θεώρημα

Ο αλγόριθμος Tournament επιτυγχάνει αμοιβαίο αποκλεισμό.

Απόδειξη

- Η προβολή κάθε εκτέλεσης στη ρίζα του δένδρου είναι επιτρεπτή εκτέλεση του συμμετρικού αλγορίθμου για 2 διεργασίες.
- Αφού ο αλγόριθμος αυτός εγγυάται αμοιβαίο αποκλεισμό, και ο αλγόριθμος Tournament εγγυάται αμοιβαίο αποκλεισμό.

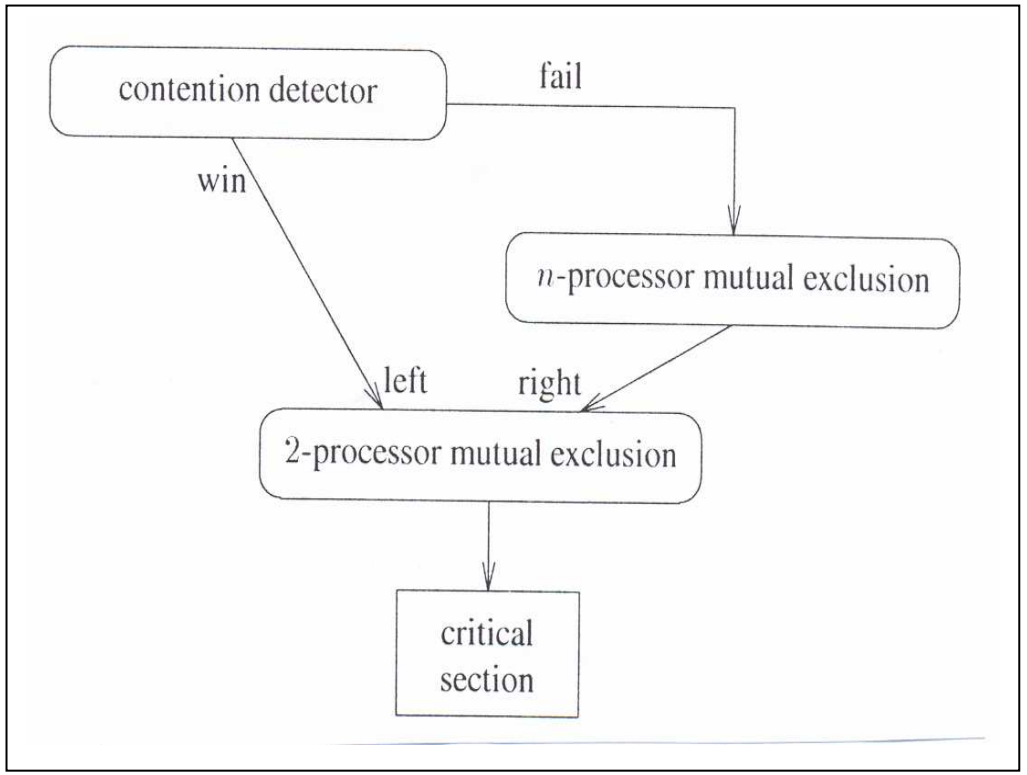
Ένας γρήγορος αλγόριθμος για αμοιβαίο αποκλεισμό

Ανιχνευτής Συμφόρησης

Αρχικά, door = open, race = -1;

race = id;

```
if (door == closed) then return lose;
else  {
      door = closed;
      if (race == id) then return win;
      else return lose;
}
```



Θεώρημα

Σε κάθε επιτρεπτή εκτέλεση του αλγορίθμου:

1. Το πολύ μια διεργασία επιστρέφει win από τον ανιχνευτή συμφόρησης.
2. Αν μια διεργασία p_i εκτελέσει τον ανιχνευτή συμφόρησης μόνη της, δηλαδή καμιά άλλη διεργασία δεν ξεκινά την εκτέλεση του ανιχνευτή συμφόρησης πριν η p_i τερματίσει τη δική της εκτέλεση, τότε η p_i επιστρέφει win.

Κάτω Φράγμα στον Αριθμό των Read/Write Καταχωρητών

Όλοι οι αλγόριθμοι που επιλύουν το πρόβλημα του αμοιβαίου αποκλεισμού χρησιμοποιώντας μόνο read/write καταχωρητές απαιτούν n τέτοιους καταχωρητές.

Αυτό δεν συμβαίνει κατά τύχη!!! Μπορεί να αποδειχθεί ότι το πρόβλημα δεν επιλύεται με λιγότερους από n καταχωρητές. Αυτό ισχύει ακόμη και αν:

1. απαιτείται να ισχύουν μόνο οι ιδιότητες του αμοιβαίου αποκλεισμού και της αποφυγής αδιεξόδου
2. οι καταχωρητές έχουν μη-πεπερασμένο μέγεθος.

Χρήσιμοι Ορισμοί

Μια καθολική κατάσταση λέγεται *ανενεργή* αν όλες οι διεργασίες βρίσκονται στη μη-κρίσιμο τμήμα τους.

Λέμε ότι μια διεργασία j *καλύπτει* μια μεταβλητή x σε μια καθολική κατάσταση C αν στο πρώτο βήμα που θα εκτελέσει η j μετά τη C θα γράψει στη μεταβλητή x.

Για *κάθε* k, $0 \leq k \leq n$, μια *καθολική κατάσταση* είναι *k-προσβάσιμη* από *κάποια* άλλη *καθολική κατάσταση* C' αν υπάρχει τμήμα εκτέλεσης που να ξεκινά από τη C *και* να *καταλήγει* στη C' το οποίο περιλαμβάνει βήματα μόνο των διεργασιών $p_0, \ldots, p_{k-1}$.

Ένα τμήμα εκτέλεσης ονομάζεται p-μόνο, αν μόνο η διεργασία p εκτελεί βήματα σε αυτό. Ονομάζεται S-μόνο αν μόνο διεργασίες στο σύνολο S εκτελούν βήματα σε αυτό.

Ένα τμήμα εκτέλεσης a είναι p-only αν μόνο η διεργασία p εκτελεί βήματα στο a. Το a είναι S-only (όπου S είναι ένα σύνολο διεργασιών) αν μόνο οι διεργασίες που ανήκουν στο S εκτελούν βήματα στο a.

Το *χρονοδιάγραμμα* μιας εκτέλεσης a είναι η ακολουθία από δείκτες διεργασιών που εκτελούν βήματα στην a (με τη σειρά που τα βήματα αυτά συναντώνται στην a).

Ένα χρονοδιάγραμμα καθορίζει μια ακολουθία γεγονότων. Μια καθολική κατάσταση C και ένα χρονοδιάγραμμα σ καθορίζουν με μοναδικό τρόπο ένα τμήμα εκτέλεσης το οποίο συμβολίζουμε με exec(C,σ).

Συμβολίζουμε με $mem(C) = (r_0, \ldots, r_{m-1})$: διάνυσμα τιμών των καταχωρητών στη C

Μια καθολική κατάσταση C είναι *παρόμοια* (similar ή indistinguishable) με μια άλλη καθολική κατάσταση C' ως προς κάποιο σύνολο διεργασιών S αν κάθε διεργασία του S είναι στην ίδια κατάσταση στη C και τη C' και mem(C) = mem(C'). Το γεγονός αυτό το συμβολίσουμε $C \sim^S C'$.

Προφανείς Ισχυρισμοί

Μια διεργασία που εκτελείται μόνη ξεκινώντας από μια ανενεργή καθολική κατάσταση εισέρχεται στο κρίσιμο τμήμα της.

Μια διεργασία που εκτελείται μόνη ξεκινώντας από μια καθολική κατάσταση που είναι παρόμοια μιας ανενεργής εισέρχεται στο κρίσιμο τμήμα της.

Λήμμα 1: Έστω μια καθολική κατάσταση C που είναι παρόμοια μιας ανενεργής καθολικής κατάστασης για κάποια διεργασία j. Έστω ότι a_1 είναι ένα j-only τμήμα εκτέλεσης που ξεκινά από τη C στο οποίο η j εισέρχεται στο κρίσιμο τμήμα της. Τότε, στο a_1, η j πρέπει να εκτελέσει τουλάχιστον μια write εντολή σε κάποιο καταχωρητή.

Απόδειξη:

C': η τελική καθολική κατάσταση του a_1. Τότε:

αν $C = (q_0, \ldots, q_j, \ldots, q_{n-1}, mem(C)) \Rightarrow C' = (q_0, \ldots, q'_j, \ldots, q_{n-1}, mem(C))$

Αν $mem(C) = mem(C') \Rightarrow C \sim^k C', \forall k \neq j$.

Υπάρχει ένα τμήμα εκτέλεσης a_2 (με χρονοδιάγραμμα s_2) που ξεκινά από τη C στο οποίο η διεργασία k εισέρχεται στο κρίσιμο τμήμα της. Ωστόσο, $C \sim^k C'$.

C'': η τελική καθολική κατάσταση της εκτέλεσης $a_1 \cdot a_2$.

$\Rightarrow$ και ο p_j και ο p_k είναι στο κρίσιμο τμήμα στη C''. Άτοπο!!!

Κάτω Φράγμα στον Αριθμό των Read/Write Καταχωρητών

Single-Writer Read/Write Καταχωρητές

Θεώρημα

Έστω ότι ο Α είναι αλγόριθμος που επιλύει το πρόβλημα του αμοιβαίου αποκλεισμού για $n \geq 2$ διεργασίες, χρησιμοποιώντας μόνο single-writer/multe-reader καταχωρητές ανάγνωσης/εγγραφής. Τότε, ο Α χρησιμοποιεί τουλάχιστον n τέτοιους καταχωρητές.

Γενικευμένο Λήμμα (Λήμμα 2)

Έστω μια καθολική κατάσταση C που είναι παρόμοια μιας ανενεργής καθολικής κατάστασης για κάποια διεργασία j. Έστω ότι a_1 είναι ένα j-only τμήμα εκτέλεσης που ξεκινά από τη C στο οποίο η j εισέρχεται στο κρίσιμο τμήμα της. Τότε, στο a_1, η j πρέπει να εκτελέσει τουλάχιστον μια write εντολή σε κάποιο καταχωρητή που δεν καλύπτεται από καμία (άλλη) διεργασία στη C.

Κάτω όριο: 2 διεργασίες και 1 MW καταχωρητής

Θεώρημα 1

Δεν υπάρχει αλγόριθμος που να επιλύει το πρόβλημα του αμοιβαίου αποκλεισμού για δύο διεργασίες χρησιμοποιώντας μόνο έναν διαμοιραζόμενο RW καταχωρητή.

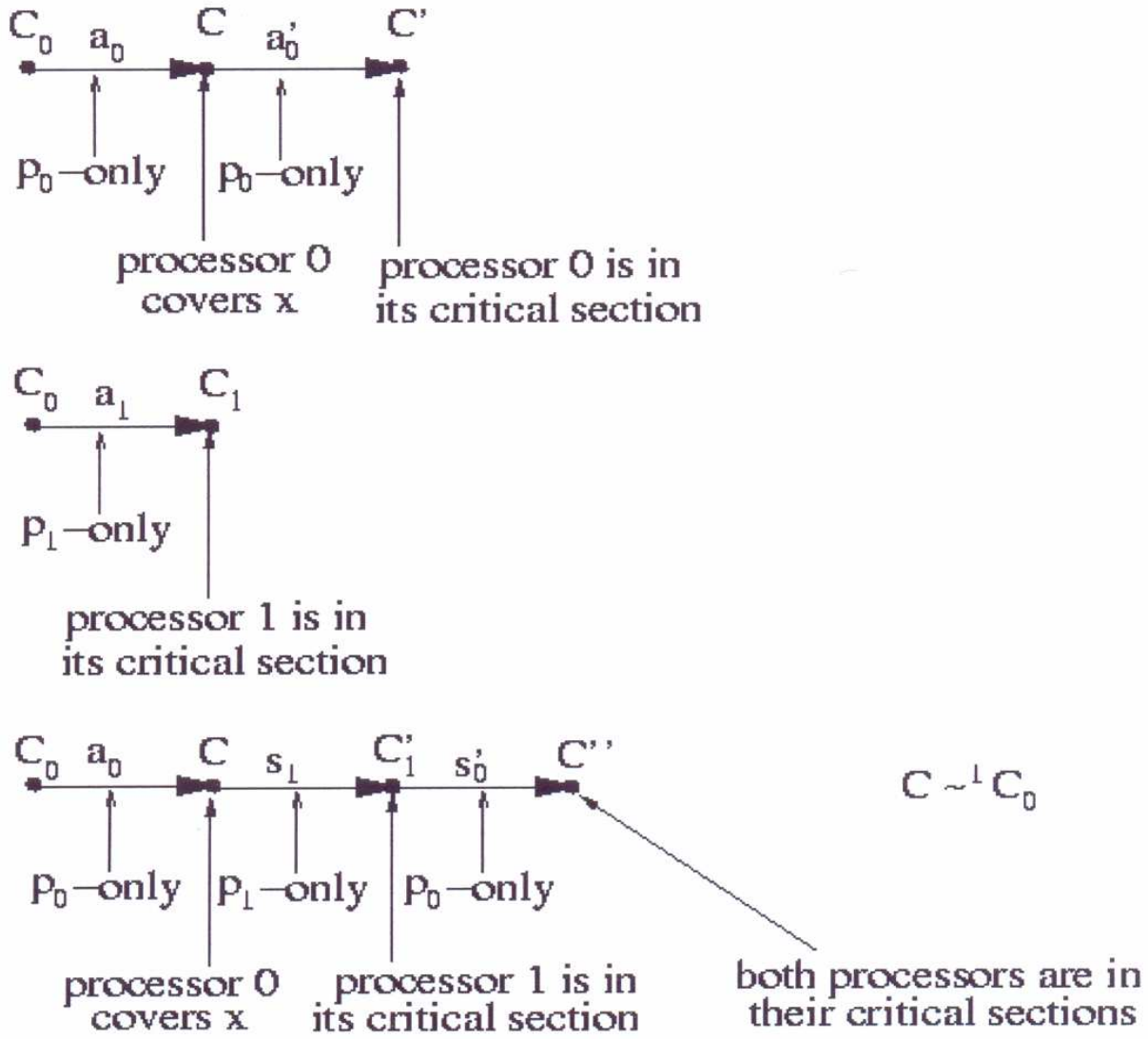
Απόδειξη

Με εις άτοπο απαγωγή.

A: αλγόριθμος

x: ο διαμοιραζόμενος καταχωρητής που χρησιμοποιεί

C_0: αρχική κατάσταση


C0 a0 C a'0 C'
P0–only
P0–only
processor 0 covers x
processor 0 is in its critical section
C0 a1 C1
P1–only
processor 1 is in its critical section
C0 a0 C s1 C'1 s'0 C''
C ~1 C0
P0–only
P1–only
P0–only
processor 0 covers x
processor 1 is in its critical section
both processors are in their critical sections

Κάτω όριο: 3 διεργασίες και 2 MW καταχωρητές

Θεώρημα 2: Δεν υπάρχει αλγόριθμος που να επιλύει το πρόβλημα του αμοιβαίου αποκλεισμού για τρεις διεργασίες χρησιμοποιώντας μόνο δύο διαμοιραζόμενες RW καταχωρητές.

Απόδειξη: Με εις άτοπο απαγωγή.

A: αλγόριθμος

x,y: διαμοιραζόμενοι καταχωρητές

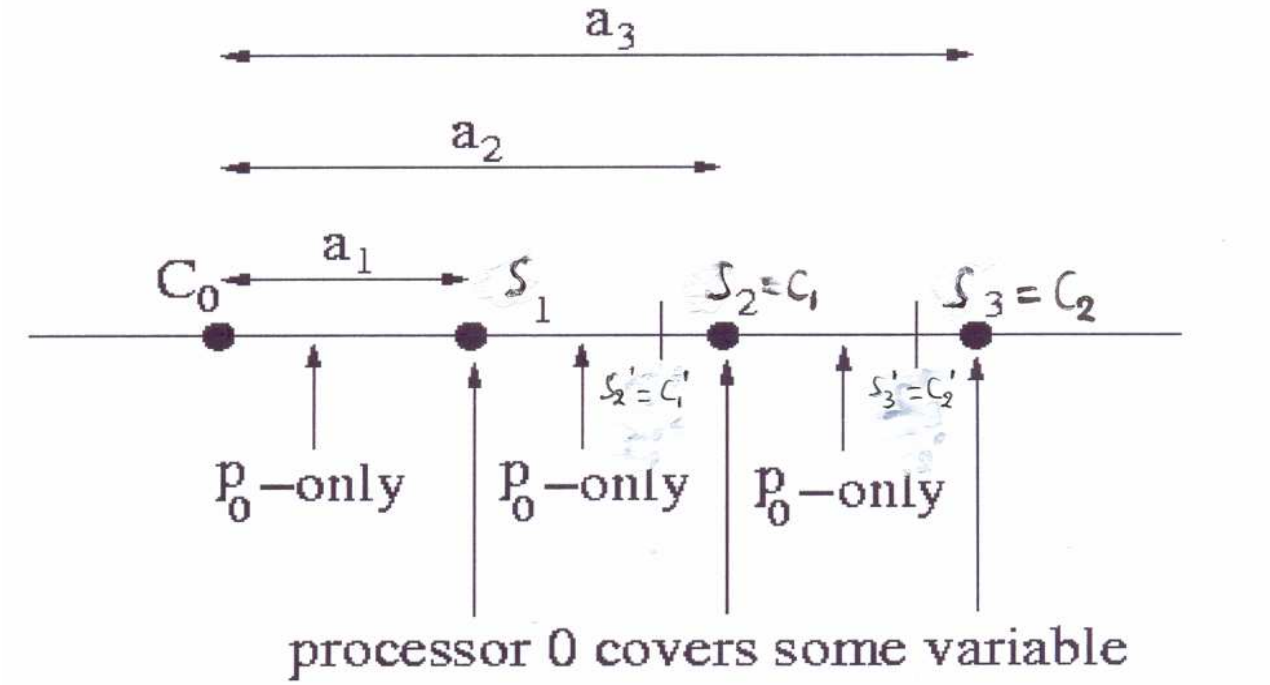
Στρατηγική

1. Δημιουργούμε σενάριο στο οποίο οι διεργασίες p_0 και p_1 εκτελούνται μέχρι που η κάθε μια καλύπτει έναν διαφορετικό καταχωρητή και η καθολική κατάσταση C' που προκύπτει είναι παρόμοια με μια προσβάσιμη ανενεργή καθολική κατάσταση για την p_2. Στη C' και οι δύο καταχωρητές x και y καλύπτονται.
2. Επιτρέπουμε στη p_2 να τρέξει μόνη από τη C' μέχρι να εισέλθει στο κρίσιμο τμήμα.
3. Βάζουμε τις p_0 και p_1 να κάνουν από ένα βήμα ώστε να πανωγράψουν τους καταχωρητές x και y. Αυτό καταστρέφει κάθε ίχνος της εκτέλεσης της p_2.
4. Επιτρέπουμε στις p_0 και p_1 να συνεχίσουν να κάνουν βήματα μέχρι κάποια από αυτές να εισέλθει στο κρίσιμο τμήμα.
5. Τότε, δύο διεργασίες βρίσκονται ταυτόχρονα στο κρίσιμο τμήμα. Άτοπο!!!!

Κάτω όριο: 3 διεργασίες και 2 MW καταχωρητές

Απόδειξη (συνέχεια): Πως μπορούμε να εξαναγκάσουμε τις διεργασίες p_0 και p_1 να καλύπτουν τους καταχωρητές x και y ενώ η διεργασία p_2 νομίζει ότι βρίσκονται στο μη-κρίσιμο τμήμα τους.

Απάντηση



Σε 2 από τις 3 καθολικές καταστάσεις S_1, S_2, S_3, η p_0 καλύπτει τον ίδιο καταχωρητή, π.χ., τον x.

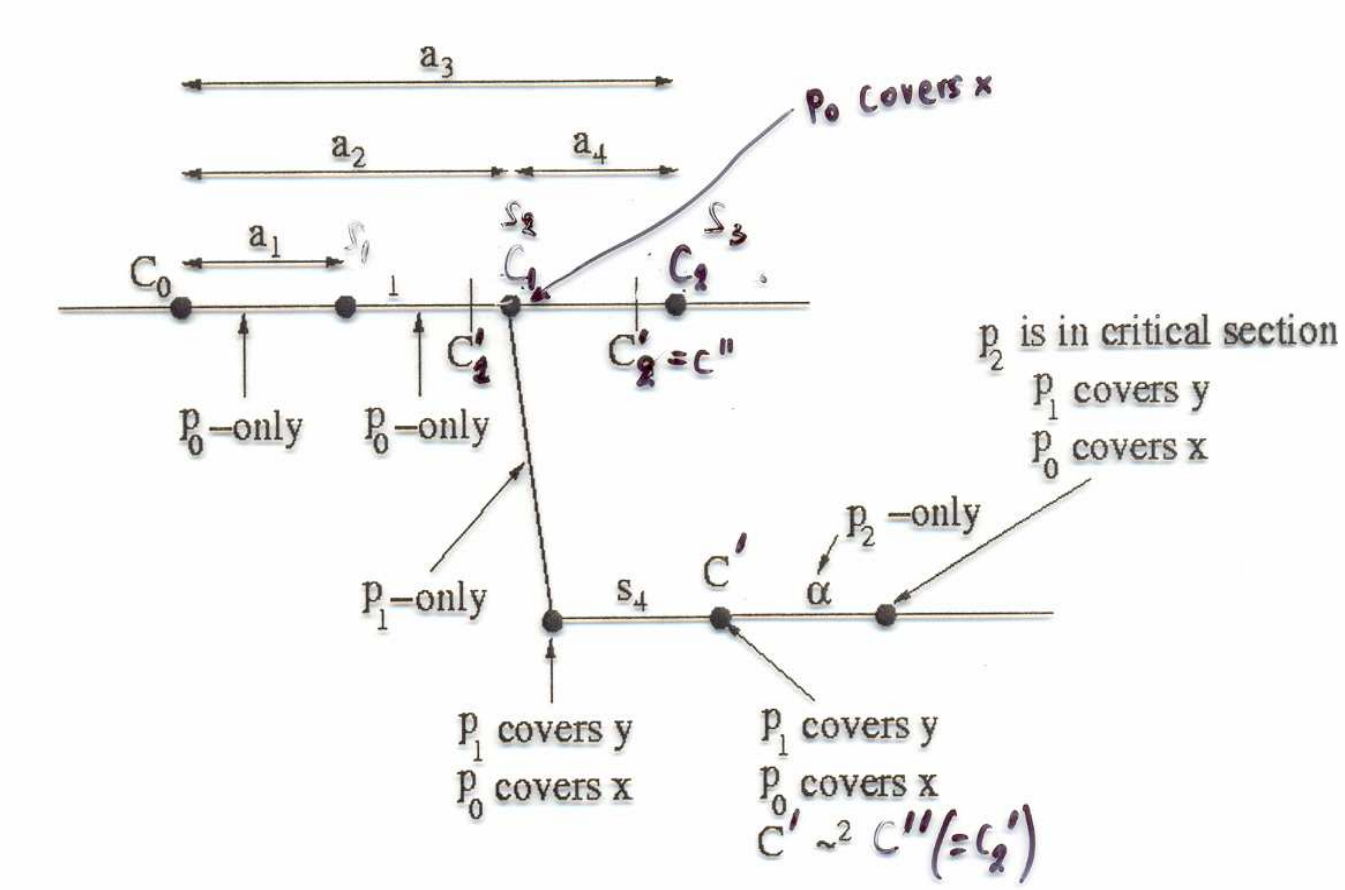
$S_1' = C_0$, S_2' και S_3': ανενεργές καταστάσεις

Έστω ότι $C_1 = S_2$, $C_1' = S_2'$, $C_2 = S_3$, $C_2' = S_3'$.

Αν εκτελέσουμε την p_1 μόνη ξεκινώντας από την S_2 θα εισέλθει στο κρίσιμο τμήμα αφού $S_2 \sim^1 S_2'$.

Στην εκτέλεση αυτή η p_1 γράφει στον y.

Κάτω όριο: 3 διεργασίες και 2 MW καταχωρητές



Από την C' η p_2 μπορεί να εκτελεστεί μόνη μέχρι να εισέλθει στο κρίσιμο τμήμα.

Τότε, οι p_0 και p_1 εκτελούν ένα βήμα η κάθε μια και όλα τα ίχνη της εκτέλεσης της p_2 χάνονται.

Οι p_0, p_1 στη συνέχεια συνεχίζουν μέχρι μια από τις δυο να εισέλθει στο κρίσιμο τμήμα.

Κάτω όριο: Γενική Περίπτωση

Φυσική επέκταση των αποδείξεων των δύο ειδικών περιπτώσεων που μελετήθηκαν παραπάνω.

Θεώρημα 3: Δεν υπάρχει αλγόριθμος που να επιλύει το πρόβλημα του αμοιβαίου αποκλεισμού για $n \geq 2$ διεργασίες χρησιμοποιώντας n-1 ή λιγότερους διαμοιραζόμενες RW καταχωρητές.