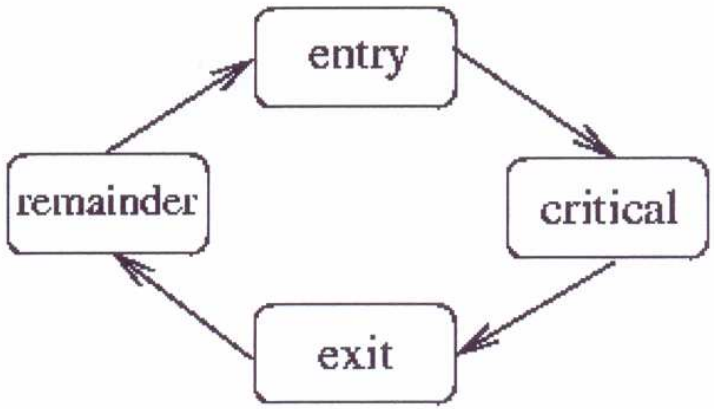
Το Πρόβλημα του Αμοιβαίου Αποκλεισμού

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.

Mutual Exclusion using atomic registers: Basic Topics

- Algorithms for Two Processes
- Tournament Algorithms
- Starvation-free Algorithms
- Tight Space bounds
- A Fast Mutual Exclusion Algorithms

Properties & complexity

- Time complexity
  - Fast
  - Adaptive
- Fairness
  - FIFO, …
- Fault-tolerance
- Local spinning
- Space complexity

Βασικοί Ορισμοί

r-bounded-waiting: A waiting process will be able to enter its critical section before each of the other processes is able to enter its critical section r+1 times.

Bounded Waiting: There exists a positive integer r for which the algorithm is r-bounded waiting. That is, if a given process is in its entry code, then there is a bound on the number of times any other process is able to enter its critical section before the given process does so.

Linear Waiting: 1-bounded waiting

First-In-First-Out: the term is used for 0-bounded waiting -> FIFO guarantees that no beginning process can pass an already waiting process.

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.

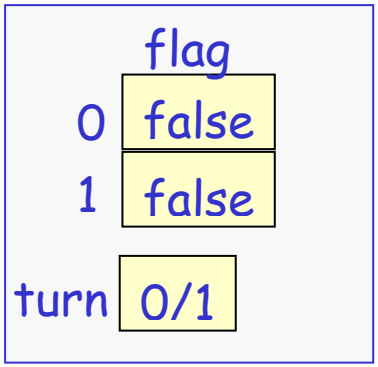
Αρνητικά Αποτελέσματα

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

Section 2.1.1

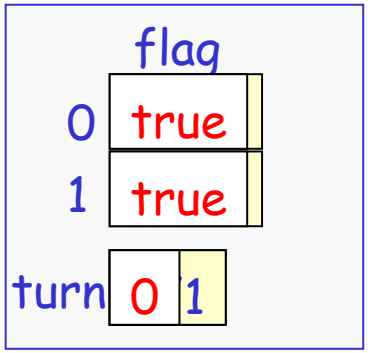
Peterson's algorithm

Thread 0	Thread 1
1. flag[0] = true	1. flag[1] = true
2. turn = 1	2. turn = 0
3. while (flag[1] and turn == 1)	3. while (flag[0] and turn == 0)
4. {noop}	4. {noop}
5. critical section	*5. critical section*
6. flag[0] = false	6. flag[1] = false



A variant of Peterson's algorithm
Is it correct ?

Thread 0	Thread 1
turn = 1	turn = 0
flag[0] = true	flag[1] = true
while (flag[1] and turn = 1) {noop}	while (flag[0] and turn = 0) {noop}
critical section	*critical section*
flag[0] = false	flag[1] = false



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!

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.

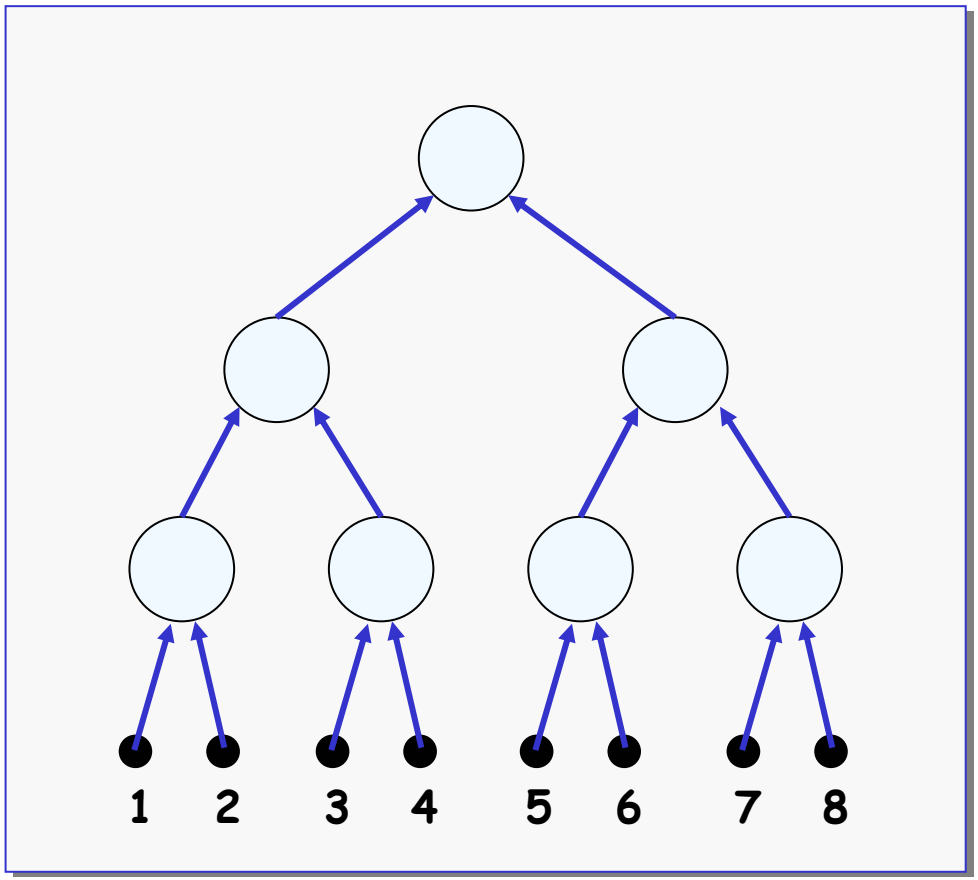
Περίπτωση 2: η διεργασία 1 δεν εκτελεί την εντολή 7.

Solutions for Many Processes

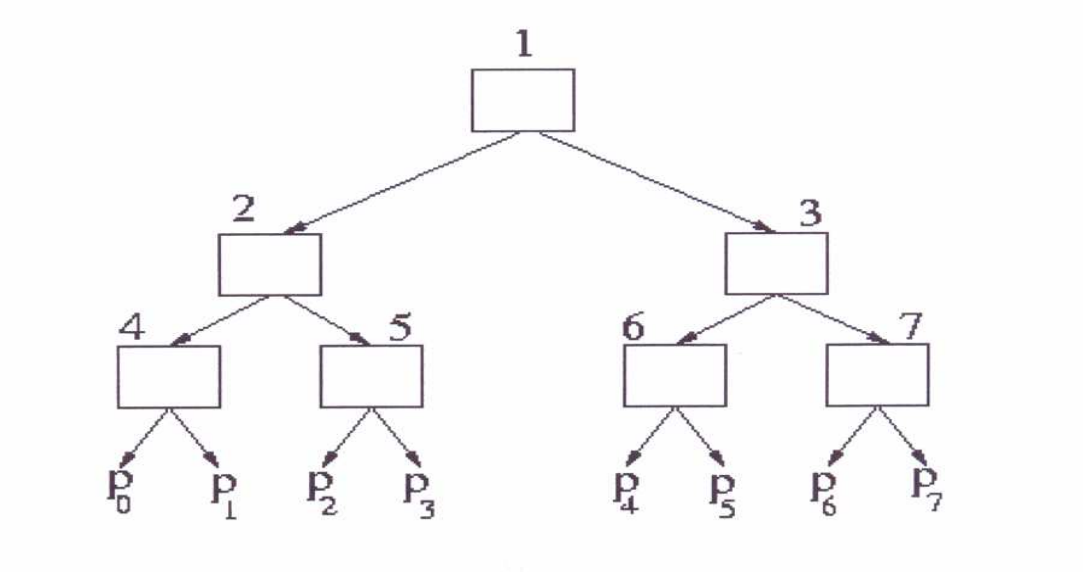
How can we use a two-process algorithm to construct an algorithm for many processes?

Section 2.2

Tournament Algorithms



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

Starvation-free Algorithms

4.1 Basic Definitions (i.e., FIFO)
4.2 The Bakery Algorithm
4.3 The Black-White Bakery Algorithm Algorithm

Ο Αλγόριθμος Bakery

for each i, $0 \leq i \leq n-1$:

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, $0 \leq i \leq n-1$

Αρχικά, Number[i] = 0, και

Choosing[i] = FALSE, για κάθε i, $0 \leq i \leq n-1$

Τμήμα Εισόδου:

Choosing[i] = TRUE;

Number[i] = max{Number[0], …, Number[n-1]}+1;

Choosing[i] = FALSE;

for j = 0 to n-1, $j \neq i$, 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.

Properties of the Bakery Algorithm

- Satisfies mutex & FIFO.
- The size of number[i] is unbounded.
- Safe registers: reads which are concurrent with writes may return arbitrary value.

Properties of the Bakery Algorithm

- Satisfies mutex & FIFO.
- The size of number[i] is unbounded.
- Safe registers: reads which are concurrent with writes may return arbitrary value.

The Black-White Bakery Algorithm

code of process i , $i \in \{1, \ldots, n\}$

```
1. choosing[i] = true
2. mycolor[i] = color
3. number[i] = 1 + max{number[j] | (1 ≤ j ≤ n) ∧ (mycolor[j] = mycolor[i])}
4. choosing[i] = false
5. for j = 0 to n do
6.       await choosing[j] = false
7.       if mycolor[j] = mycolor[i]
8.       then await (number[j] = 0) ∨ (number[j],j) ≥ (number[i],i) ∨
                    (mycolor[j] ≠ mycolor[i])
9.      else await  (number[j] = 0) ∨ (mycolor[i] ≠ color) ∨
                    (mycolor[j] = mycolor[i]) fi od
10. critical section
11. if mycolor[i] = black then color = white else color = black fi
12. number[i] = 0
```

Tight space bounds for mutual exclusion using atomic registers

- A lower bound (Section 2.5.1)
- An Upper Bound (Section 2.5.2)

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.

Κάτω Φράγμα στον Αριθμό των Read/Write Καταχωρητών

Χρήσιμοι Ορισμοί

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 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 $C \sim^S C'$.

Κάτω Φράγμα στον Αριθμό των Read/Write Καταχωρητών

Χρήσιμοι Ορισμοί

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.

Κάτω Φράγμα στον Αριθμό των Read/Write Καταχωρητών

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

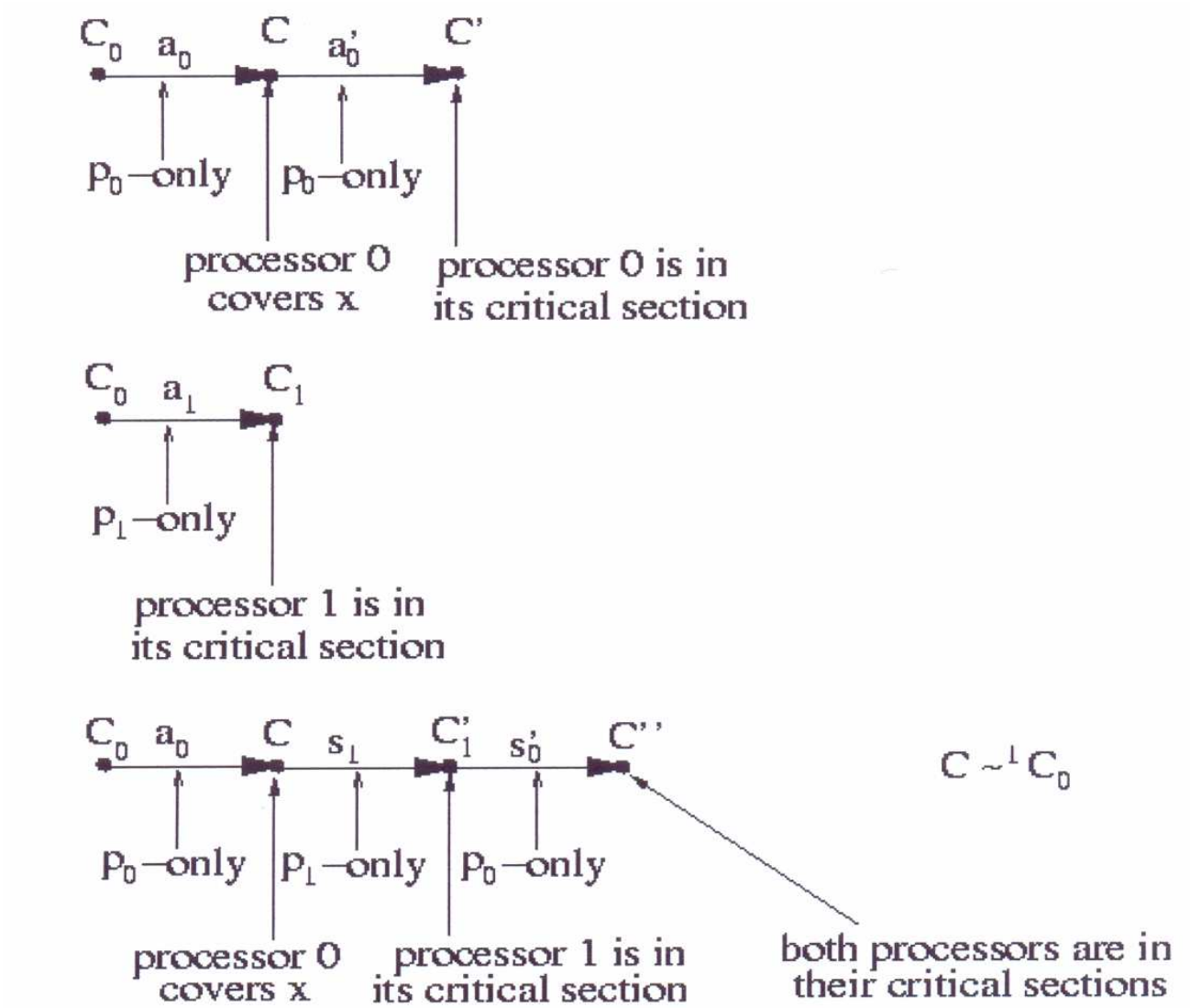
Θεώρημα 1
Δεν υπάρχει αλγόριθμος που να επιλύει το πρόβλημα του αμοιβαίου αποκλεισμού για δύο διεργασίες χρησιμοποιώντας μόνο έναν διαμοιραζόμενο RW καταχωρητή.

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.
Α: αλγόριθμος
x,y: διαμοιραζόμενοι καταχωρητές

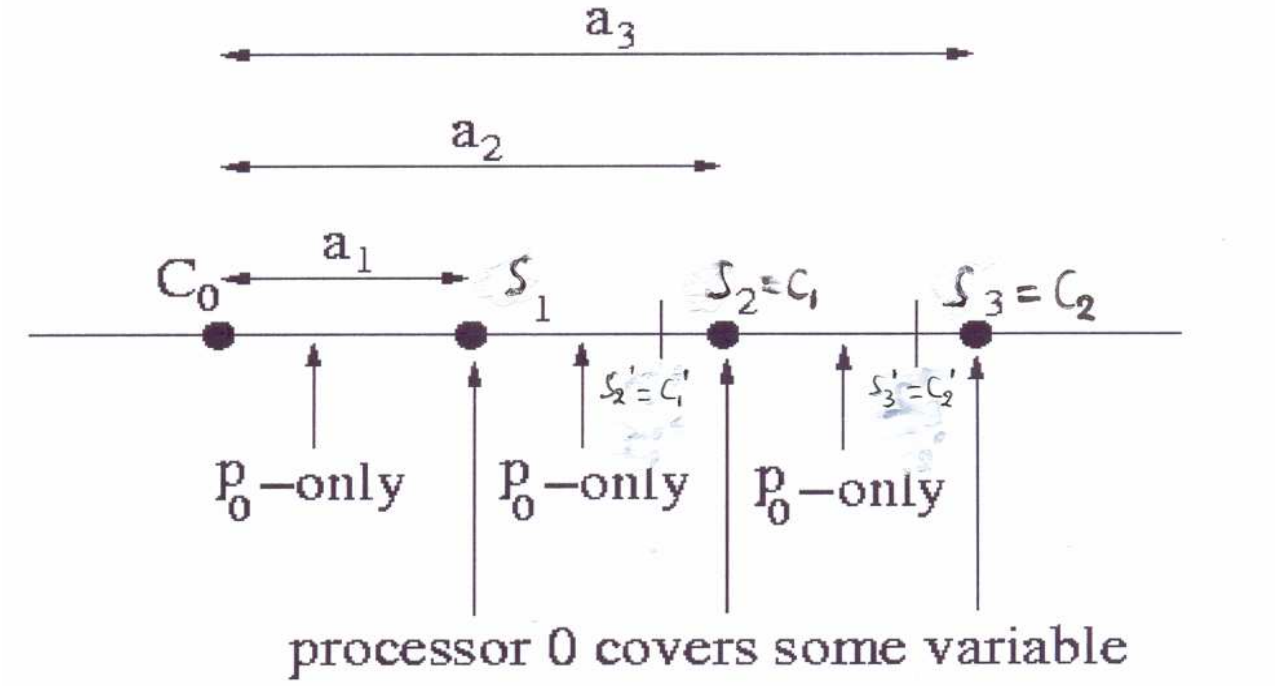
Στρατηγική

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.

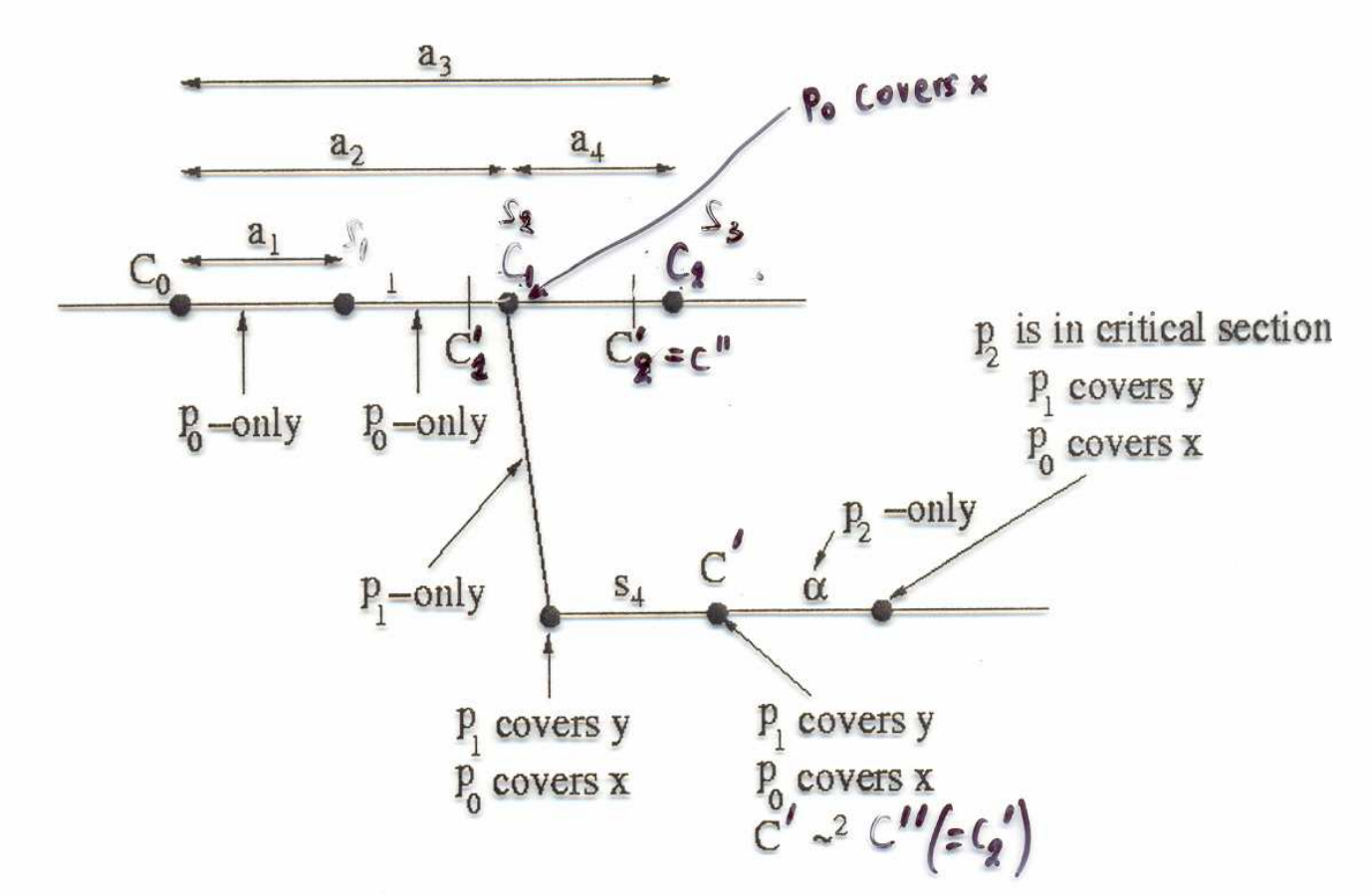
S_1' = C_0, S_2' και S_3': ανενεργές καταστάσεις

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.

Tight Space Bounds

Section 2.5.2

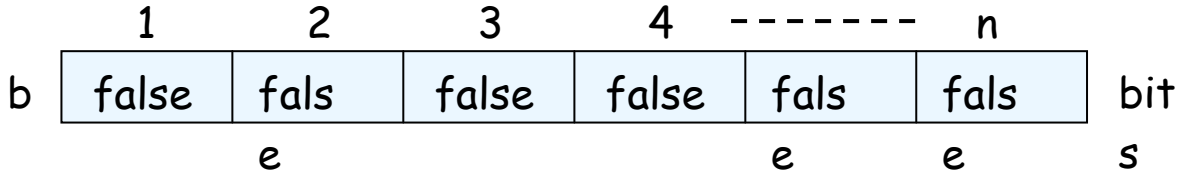
An Upper Bound

Theorem: There is a deadlock-free mutual exclusion algorithm for n processes that uses n shared bits.

The One-Bit Algorithm

code of process i , i ∈ {1 ,..., n}

```
1. repeat
2.         b[i] = true; j = 1;
3.         while (b[i] = true) and (j < i) do
4.         if b[j] = true then b[i] = false; await b[j] =false fi
5.         j = j+1
   od
6. until b[i] = true
7. for j = i+1 to n do await b[j] = false od
8. critical section
9. b[i] = false
```



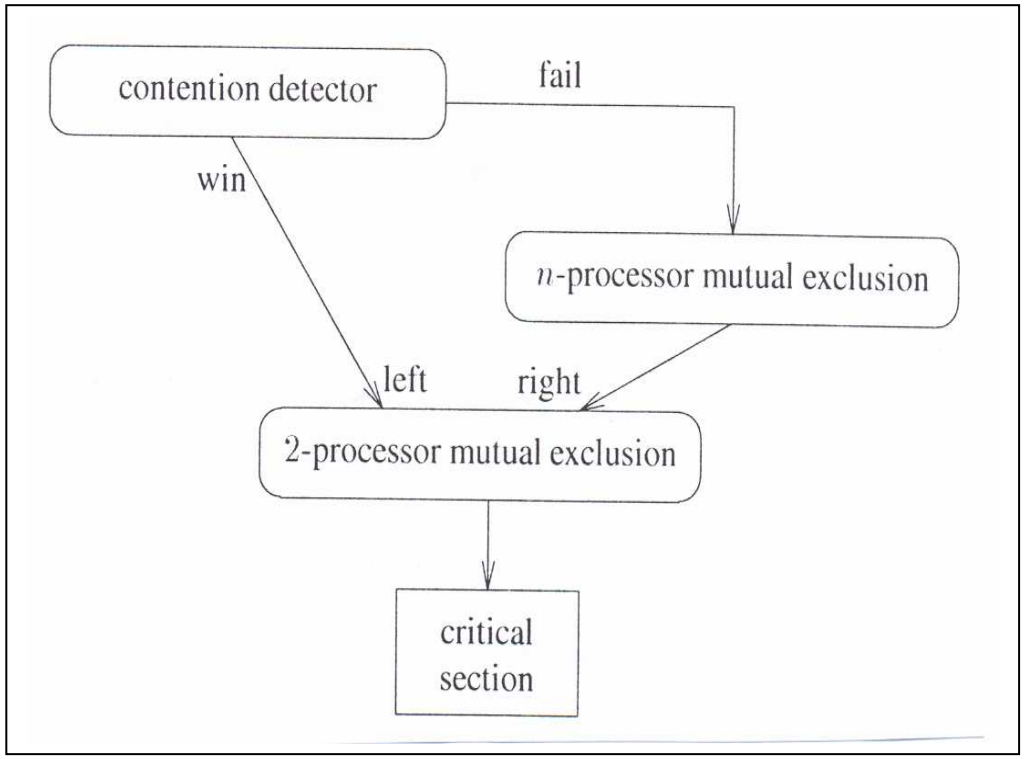
Properties of the One-Bit Algorithm

- Satisfies mutual exclusion and deadlock-freedom
- Starvation is possible
- It is not fast
- It is not symmetric
- It uses only n shared bits and hence it is space optimal

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:

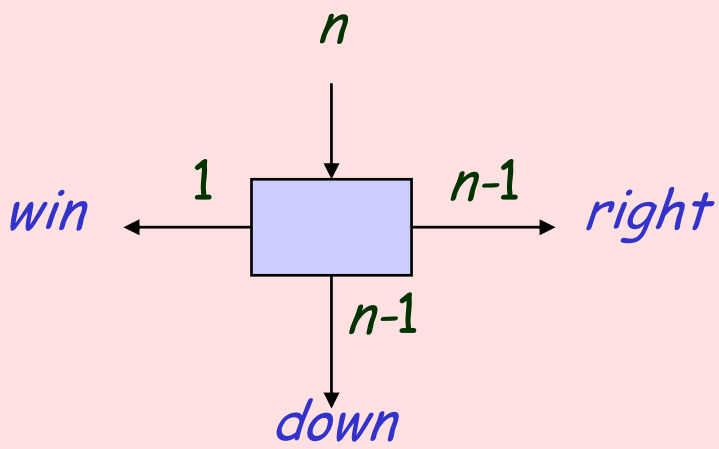
1. Το πολύ μια διεργασία επιστρέφει win από τον ανιχνευτή συμφόρησης.
2. Αν μια διεργασία p_i εκτελέσει τον ανιχνευτή συμφόρησης μόνη της, δηλαδή καμιά άλλη διεργασία δεν ξεκινά την εκτέλεση του ανιχνευτή συμφόρησης πριν η p_i τερματίσει τη δική της εκτέλεση, τότε η p_i επιστρέφει win.

Fast Mutual exclusion Algorithm

- Mutual exclusion and deadlock-freedom
- Starvation of individual processes is possible
- fast access
  - in the absence of contention, only 7 accesses to shared memory are needed
- With contention
  - Even if only 2 processes contend, the winner may need to check all the O(n) shared registers
  - System response time is of order n time units
- n+2 shared registers are used

Splitter

- At most *n*-1 can move *right*
- At most *n*-1 can move *down*
- At most 1 can *win*
- In solo run ➔ 1 win

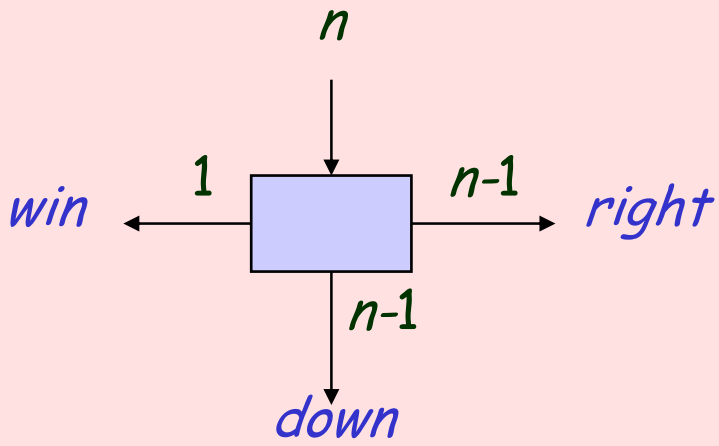


The code of the splitter

```
x = i
if  y = 1 then go right fi
y = 1
if x ≠ i  then go down fi
win
```

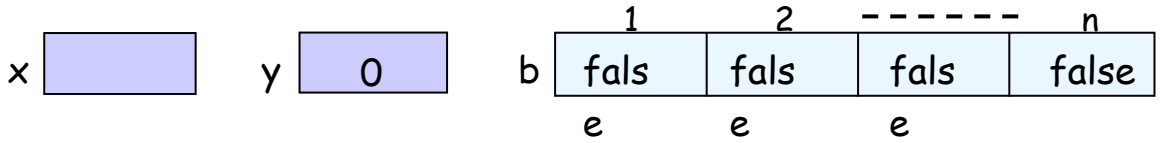
x

y 0



Fast Mutual exclusion Algorithm

code of process i , i ∈ {1 ,..., n}




```
1. start: b[i] = true
2.        x = i
3.        if y ≠ 0 then b[i] = false
4.                      await y = 0
5.                      goto start fi
6.        y = i
7.        if x ≠ i then b[i] = false
8.                      for j = 1 to n do await b[j] = false od
9.                      if y ≠ i then await y = 0
10.                                   goto start fi fi
11.       critical section
12.       y = 0
13.       b[i] = false
```

Schematic for the fast algorithm