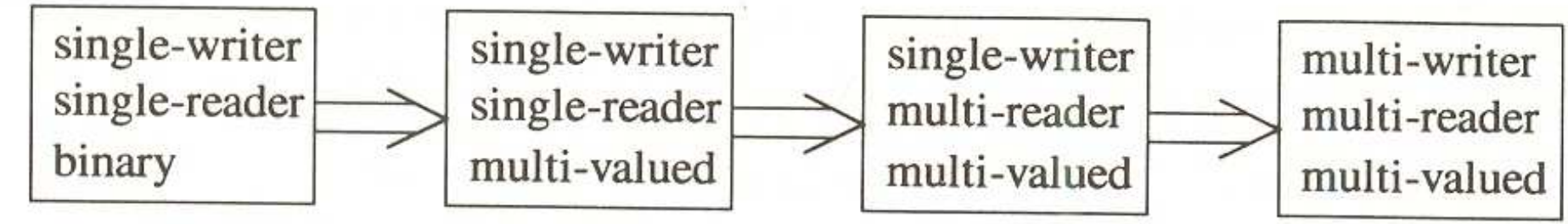
ΥΛΟΠΟΙΗΣΗ ΚΑΤΑΧΩΡΗΤΩΝ ΑΝΑΓΝΩΣΗΣ/ΕΓΓΡΑΦΗΣ

Καταχωρητές που μοιάζουν πιο πολύπλοκοι μπορούν να υλοποιηθούν από απλούστερους καταχωρητές.



Multi-valued from Binary

Βασικό Αντικείμενο: δυαδικός καταχωρητής ο οποίος μπορεί να αναγνωσθεί και να εγγραφεί από μια μόνο διεργασία

Υλοποιούμενο Αντικείμενο: καταχωρητής k-valued ο οποίος μπορεί να αναγνωσθεί και να εγγραφεί από μια μόνο διεργασία

Θεωρούμε ότι οι τιμές αναπαρίστανται σε μοναδιαίο σύστημα και χρησιμοποιούμε έναν πίνακα B[0..k-1] από k binary καταχωρητές. Η τιμή j αναπαρίσταται με ένα 1 στη θέση j του πίνακα και 0 σε όλες τις υπόλοιπες θέσεις.

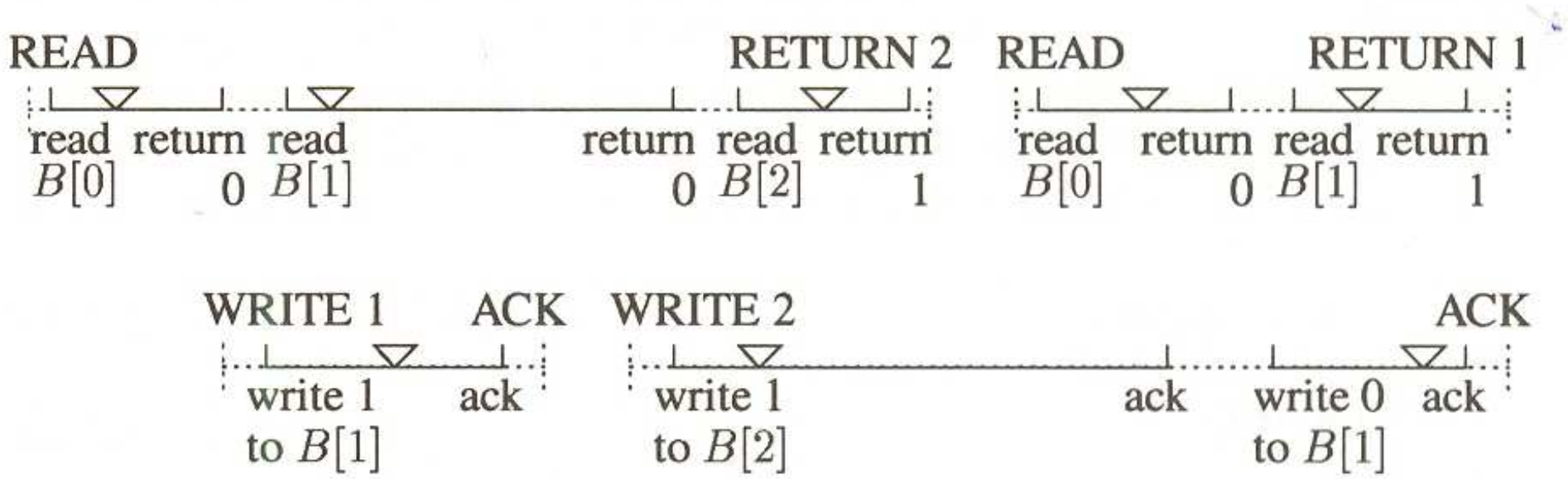
Ένας απλός αλγόριθμος

```
read:
for j = 0 to k-1
      if (B[j] == 1) return j;
```

```
write(v):
B[v] = 1;
for j =0 to k-1, j ≠ v,
      B[j] = 0;
```

Multi-valued from Binary

Αν λειτουργίες read και write εκτελούνται ταυτόχρονα, ο απλός αλγόριθμος δεν είναι σωστός.



Multi-valued from Binary

Ενέργειες επίλυσης του προβλήματος:

1. Μια λειτουργία write θέτει σε 0 μόνο τις θέσεις του πίνακα B που έχουν μικρότερους δείκτες από την τιμή που γράφει η write.
2. Μια λειτουργία read δεν σταματά όταν βρίσκει το πρώτο 1 καθώς εξετάζει τα στοιχεία του B, αλλά βεβαιώνεται ότι εξακολουθούν να υπάρχουν 0 στις θέσεις του B με μικρότερους δείκτες.

read(R):

```
i = 0;
while B[i] == 0 do i = i+1;
up = i; v = i;
for i = up -1 down to 0 do
    if B[i] == 1 then v = i;
return(R,v);
```

write(R,v):

```
B[v] = 1;
for i = v-1 downto 0 do B[i] = 0;
ack(R);
```

Multi-valued from Binary

Linearizability

Έστω α μια εκτέλεση του αλγορίθμου.

Σε κάθε καθολική κατάσταση C, ορίζουμε την τιμή του υλοποιούμενου καταχωρητή στη C να είναι το μικρότερο v τ.ω. B[v] = 1 και B[0] = B[1] = ... = B[v-1] = 0.

Σημείο σειριοποίησης μιας write(v):

Στην πρώτη καθολική κατάσταση στο διάστημα εκτέλεσης της write στην οποία ισχύει ότι B[v] = 1 και B[0] = B[1] = ... = B[v-1] = 0.

Γιατί υπάρχει τέτοια καθολική κατάσταση;

Σημείο σειριοποίησης μιας read(R,v):

Στην πρώτη καθολική κατάσταση μέσα στο διάστημα εκτέλεσης της read στην οποία η τιμή v τοποθετείται στην τοπική μεταβλητή v της διεργασίας (γραμμή 3 ή 5 του κώδικα).

Multi-valued from Binary

Λήμμα 1

1. Στο σημείο που σειριοποιείται μια write(R,v) η τιμή του R είναι v.
2. Η τιμή του R είναι η ίδια σε όλες τις καθολικές καταστάσεις μεταξύ των σημείων σειριοποίησης δυο διαδοχικών λειτουργιών write.

Λήμμα 2

Για κάθε (high-level) λειτουργία ανάγνωσης r που εκτελείται στην α ισχύει ότι η r επιστρέφει την τιμή που έχει γραφτεί στον R από την τελευταία λειτουργία εγγραφής w που σειριοποιείται πριν το σημείο σειριοποίησης της r.

Απόδειξη:

C_r: καθολική κατάσταση στην οποία σειριοποιείται η r

w: τελευταία write που σειριοποιείται πριν από το σημείο σειριοποίησης της r

C_w: καθολική κατάσταση στην οποία σειριοποιείται η w

Multi-valued from Binary

Απόδειξη (συνέχεια):

v: η τιμή που γράφει στον R η w

Λήμμα 1 $\Rightarrow$ η τιμή του R είναι v μεταξύ της C_w και της C_r.

C: καθολική κατάσταση όταν η r τελειώνει την εκτέλεση της while και u η τιμή του i όταν τερματίζει η while.

Ισχύει ότι u $\geq$ v. *Γιατί;*

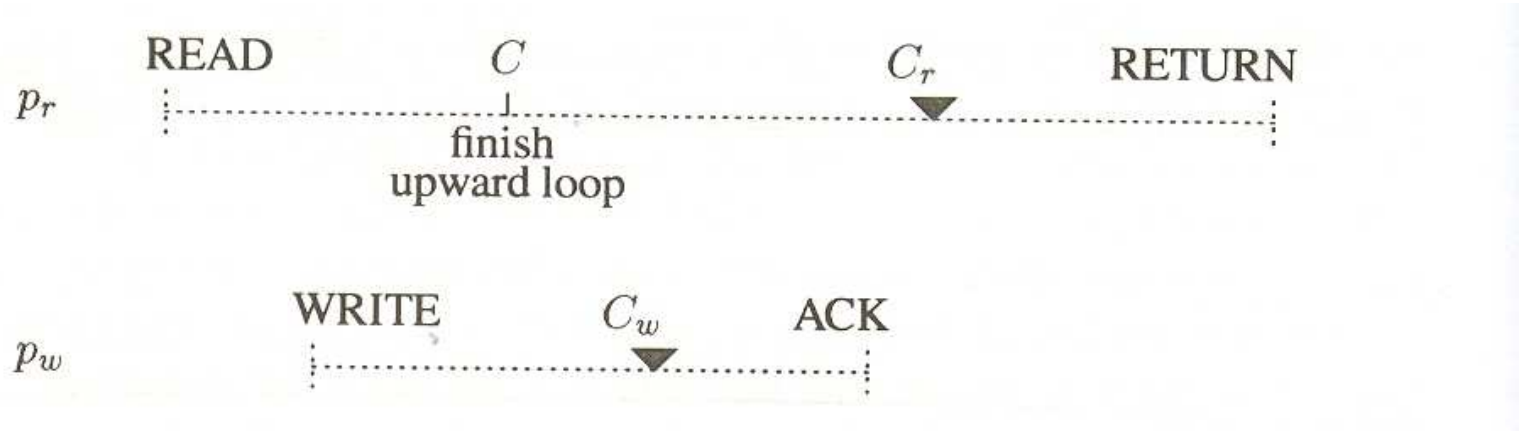
Διακρίνω περιπτώσεις:

1. η C_w προηγείται της C

   Αφού u $\geq$ v, η r διαβάζει την τιμή 1 στο B[v] και 0 σε μικρότερους δείκτες.

Multi-valued from Binary

2. η C_w έπεται της C



Αφού u ≥ v, η r διαβάζει το B[v]. Αν αυτή η ανάγνωση γίνει μετά τη C_w, τότε B[v] = 1 και B[0] = B[1] = B[v-1] = 0. Άρα, η r επιστρέφει v.

Αν αυτή η ανάγνωση γίνει πριν τη C_w, τότε στη C_r, η r διαβάζει κάποιο καταχωρητή που περιέχει 0. Αυτό αντιτίθεται στον ορισμό της C_r.

Θεώρημα: Ο παραπάνω αλγόριθμος είναι μια wait-free υλοποίηση ενός καταχωρητή στον οποίο μπορούν να αποθηκευτούν k τιμές (k-valued) από k δυαδικούς καταχωρητές. Η χρονική πολυπλοκότητα των read και write στην υλοποίηση αυτή είναι O(k).

Multi-Reader from Single-Reader

n: αριθμός των διεργασιών που επιτρέπεται να διαβάσουν τον καταχωρητή

Απλή (αλλά μη-σειριοποιήσιμη) Υλοποίηση

Shared Variables: value Val[1..n]; // πίνακας n θέσεων, μία για κάθε reader

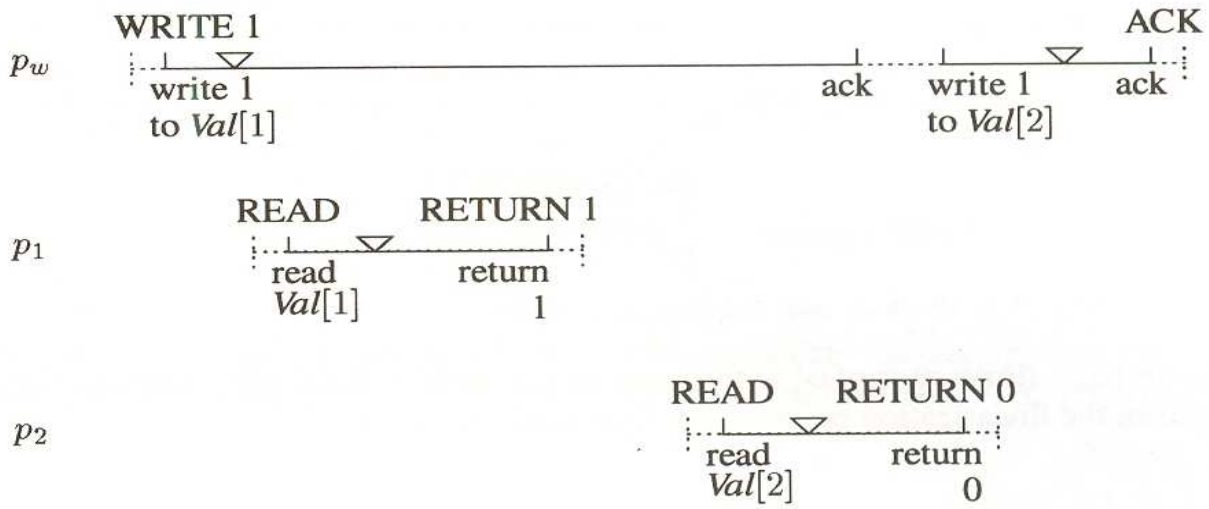
write(v):

for (j=1; j ≤ n; j++) Val[j] = v;

read: // κώδικας για p_i, 1 ≤ i ≤ n

return(Val[i]);

Κακό Σενάριο



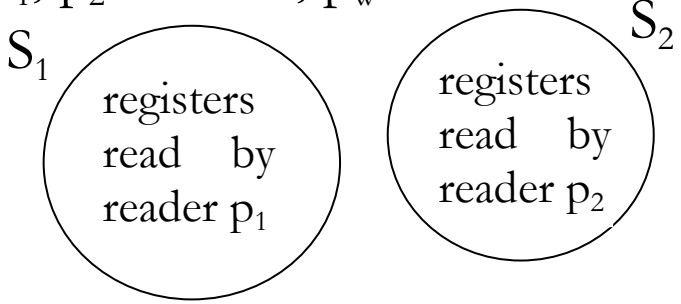
Multi-Reader from Single-Reader

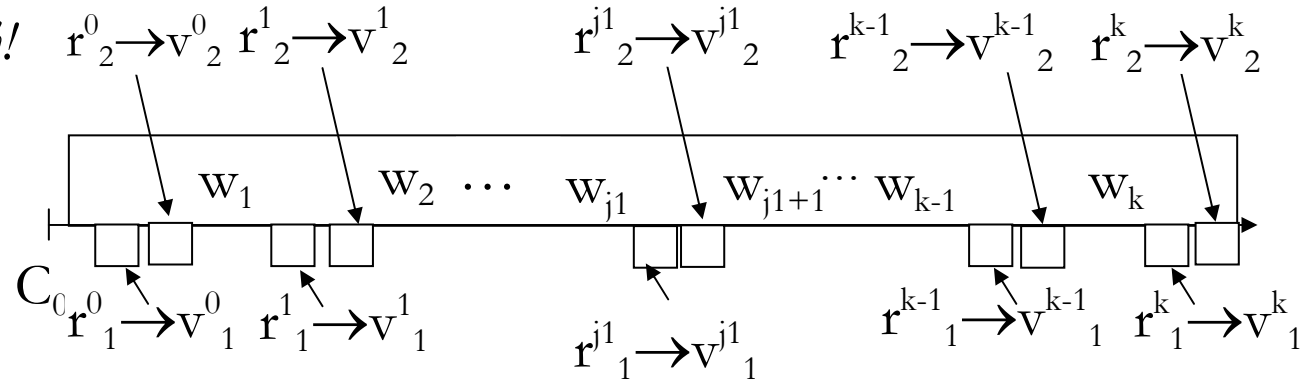
Θεώρημα

Σε κάθε ελεύθερη-αναμονής υλοποίηση ενός single-writer, multi-reader καταχωρητή από single-writer, single-reader καταχωρητές, τουλάχιστον ένας reader πρέπει να γράψει.

Απόδειξη: *Με εις άτοπο απαγωγή!*

p_1, p_2: readers, p_w: writer





Αφού η υλοποίηση είναι σειριοποιήσιμη $\forall\, i \in \{1,2\}: \exists\, j_i,\ 1 \le j_i \le k$, τέτοια ώστε, $v^j_i = 0$ για κάθε $j < j_i$ και $v^j_i = 1$, για κάθε $j \ge j_i$. *Γιατί ισχύει αυτό;*

Ισχύει $j_1 \neq j_2$. Wlog, ας υποθέσουμε ότι $j_1 < j_2$.

Η r^{j1}_1 επιστρέφει 1, ενώ η r^{j1}_2 επιστρέφει 0. Αυτό αντιτίθεται στη σειριοποιήσιμοτητα!!

Multi-Reader from Single-Reader: Ένας Σωστός Αλγόριθμος

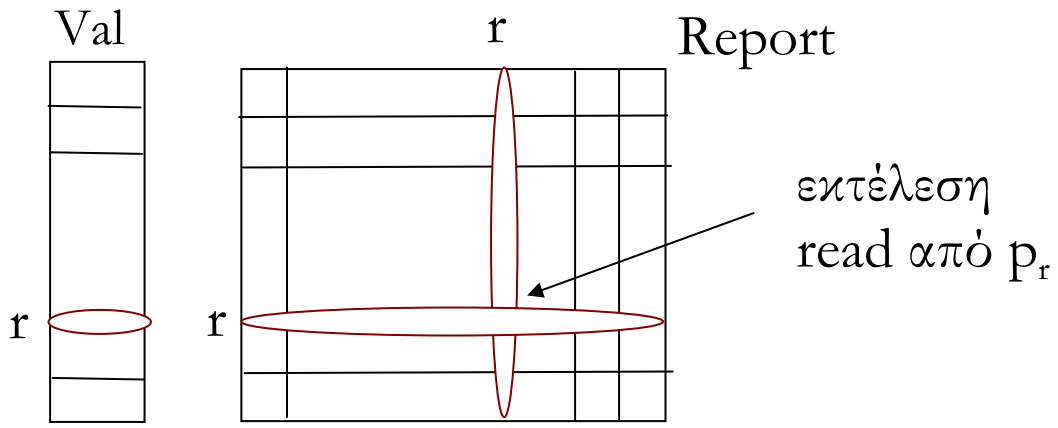
Shared Variables:

<value,seq> Val[i]; // $1 \le i \le n$, τιμή που γράφει ο p_w για κάθε έναν από τους p_r, αρχικά < v_0,0>

<value,seq> Report[i,j]; // $1 \le i,j \le n$, τιμή που επιστρέφει η πιο πρόσφατη read που εκτελείται από // την p_i. Γράφεται από την p_i και διαβάζεται από την p_j. Αρχικά, <v_0,0>

```
read(R): // κώδικας για τη διεργασία p_r, 1 ≤ i ≤ n
<v[0],s[0]> = Val[r];
for i=1 to n do
      <v[i],s[i]> = Report[i,r];
let j be s.t. s[j] = max{s[0], s[1], …, s[n]};
for i=1 to n do Report[r,i] = <v[j],s[j]>;
return (R,v[j]);
```

```
write(R,v): // ένας writer
seq = seq +1;
for i=1 to n do Val[i] = <v,seq>;
ack(R);
```



Multi-Reader from Single-Reader: Ένας Σωστός Αλγόριθμος

Σειριοποιησιμότητα

Θα φτιάξουμε μια σειριακή εκτέλεση π, η οποία (1) θα σέβεται την μερική διάταξη που ορίζεται από τους χρόνους εκκίνησης και περάτωσης κάθε λειτουργίας, και (2) κάθε read επιστρέφει το αποτέλεσμα της τελευταίας write που προηγείται αυτής στη σειριακή εκτέλεση. Λίγο διαφορετική τεχνική από αυτήν που έχουμε δει μέχρι τώρα!!!

Κατασκευή π:

- Πρώτα, τοποθετούμε στην π όλες τις λειτουργίες write σύμφωνα με τη σειρά εκτέλεσής τους από τον εγγραφέα.
- Μια λειτουργία read που επιστρέφει μια τιμή με χρονοσφραγίδα T τοποθετείται στην π ακριβώς πριν τη λειτουργία write που έπεται της write που παρήγαγε το T.

!!! Η συνέπεια προκύπτει άμεσα από τον τρόπο κατασκευής της π.

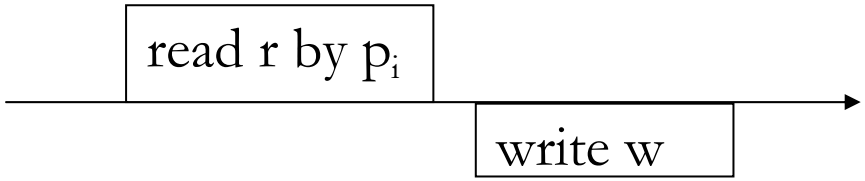
??? Θα δείξουμε ότι η π σέβεται τη μερική διάταξη που ορίζεται από την εκτέλεση των λειτουργιών.

Multi-Reader from Single-Reader: Σειριοποιησιμότητα

Λήμμα: Έστω ότι op_1 και op_2 είναι δύο λειτουργίες που συμβαίνουν σε μια εκτέλεση α, τ.ω. η op_1 τερματίζει την εκτέλεσή της πριν εκκινήσει την εκτέλεση της η op_2. Τότε, η op_1 προηγείται της op_2 στην π.

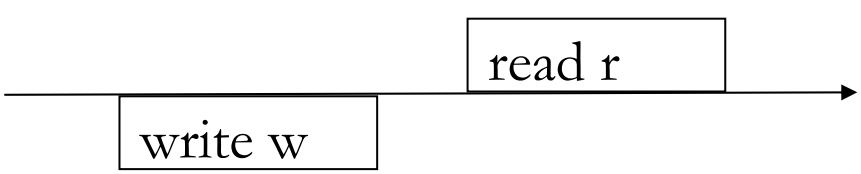
Απόδειξη: Προφανώς, ο ισχυρισμός ισχύει για τις λειτουργίες write.

α.



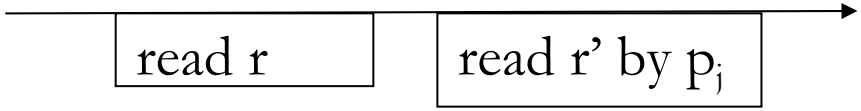
Αν η r είναι μετά τη w στην π, η w' που γράφει T είναι ή η w ή κάποιο μετέπειτα write, και άρα έπεται του r. Άτοπο!!!

β.



Η r διαβάζει από το Val[i] την τιμή που έγραψε η w ή κάποια μετέπειτα write ⇒ η r επιστρέφει τη χρονοσφραγίδα της w ή κάποιας μετέπειτα write. Άρα, η r δεν έχει τοποθετηθεί πριν την w στην π.

γ.



Η p_j διαβάζει στο Report[i] χρονοσφραγίδα ≥ T ⇒ το max της r' επιστρέφει τιμή ≥ T ⇒ η r' δεν θα τοποθετηθεί πριν την r στην π.

Θεώρημα: Ο παραπάνω αλγόριθμος είναι μια ελεύθερη-αναμονής (wait-free) υλοποίηση ενός SW καταχωρητή που μπορεί να διαβάζεται από n διεργασίες. Ο αλγόριθμος χρησιμοποιεί $O(n^2)$ καταχωρητές. Η εκτέλεση κάθε λειτουργίας read ή write απαιτεί O(n) βήματα.

Multi-Writer from Single-Writer

Κύριες Ιδέες

- Η δυσκολία έγκειται στην εύρεση ενός τρόπου συγκρίσης των τιμών και εύρεσης της πιο πρόσφατης. Οι χρονοσφραγίδες είναι διανύσματα, με μια συνιστώσα για κάθε εγγραφέα.
- Η κατασκευή μιας νέας χρονοσφραγίδας χρησιμοποιεί τα διανύσματα όλων των διεργασιών-εγγραφέων.
- Χρησιμοποιούμε την λεξικογραφική διάταξη για να αποφασίσουμε αν μια χρονοσφραγίδα είναι μεγαλύτερη από μια άλλη.
- $p_1, \ldots, p_m$: writers, $p_1, \ldots, p_n$: readers

Διαμοιραζόμενες μεταβλητές που χρησιμοποιούνται:

vector TS[i]: $1 \leq i \leq n$, διανυσματική χρονοσφραγίδα του εγγραφέα pi

<vector, value> Val[i]: $1 \leq i \leq n$, τιμή μαζί με σχετική διανυσματική χρονοσφραγίδα. Γράφεται από τον pi και διαβάζεται από όλους τους αναγνώστες.

Multi-Writer from Single-Writer

Shared Variables:

<value,vector> Val[i]; // $1 \le i \le m$, $< v_0,(0,...,0)>$

vector TS[i]; // $1 \le i \le m$, αρχικά (0,..,0)

read(R): // κώδικας για τη διεργασία p_r, $1 \le i \le n$

```
for i=1 to m do
      <v[i],t[i]> = Val[i];
let j be s.t. t[j] = max{t[1], t[2], …, t[m]};
return (R,v[j]);
```

write(R,v): // writer p_w writes v in R

```
ts = NewTS(w);
val[w] = <v,ts>;
ack(R);
```

procedure NewTS(int w):

```
for i = 1 to m do
      lts[i] = TS[i].[i];
lts[w] = lts[w] + 1;
TS[w] = lts;
return lts;
```

Multi-Writer from Single-Writer

Λήμμα 1: Η λεξικογραφική διάταξη των διανυσματικών χρονοσφραγίδων είναι μια καθολική διάταξη που σέβεται την μερική διάταξη βάσει της οποίας παράγονται οι χρονοσφραγίδες.

Λήμμα 2: Για κάθε i, αν πρώτα κάποια χρονοσφραγίδα VT_1 και αργότερα κάποια χρονοσφραγίδα VT_2 γράφεται στο Val[i], τότε $VT_1 \leq VT_2$.

Σειριοποιησιμότητα

Θα φτιάξουμε μια σειριακή εκτέλεση π, η οποία (1) θα σέβεται την μερική διάταξη που ορίζεται από τους χρόνους εκκίνησης και περάτωσης κάθε λειτουργίας, και (2) κάθε read επιστρέφει το αποτέλεσμα της τελευταίας write που προηγείται αυτής στη σειριακή εκτέλεση.

Multi-Writer from Single-Writer

Κατασκευή π:

- Πρώτα, τοποθετούμε στην π όλες τις λειτουργίες write σύμφωνα με τη λεξικογραφική διάταξη των χρονοσφραγίδων που έχουν συσχετισθεί με τις τιμές που αυτές γράφουν.
- Μια λειτουργία read που επιστρέφει μια τιμή με χρονοσφραγίδα VT τοποθετείται στην π ακριβώς πριν τη λειτουργία write που έπεται της write που παρήγαγε το VT.

!!! Η συνέπεια προκύπτει άμεσα από τον τρόπο κατασκευής της π.

??? Θα δείξουμε ότι η π σέβεται τη μερική διάταξη που ορίζεται από την εκτέλεση των λειτουργιών.

Λήμμα 3: Έστω ότι op_1 και op_2 είναι δύο λειτουργίες σε κάποια εκτέλεση α τ.ω. η op_1 τερματίζει την εκτέλεσή της πριν ξεκινήσει η op_2. Τότε η op_1 προηγείται της op_2 στην α.

Θεώρημα: Ο παραπάνω αλγόριθμος είναι μια ελεύθερη-αναμονής (wait-free) υλοποίηση ενός MW καταχωρητή που μπορεί να εγγράφεται από m διεργασίες και να διαβάζεται από όλες τις n διεργασίες. Η εκτέλεση κάθε λειτουργίας read ή write απαιτεί O(m) βήματα.