Λειτουργικά Συστήματα (HY222)

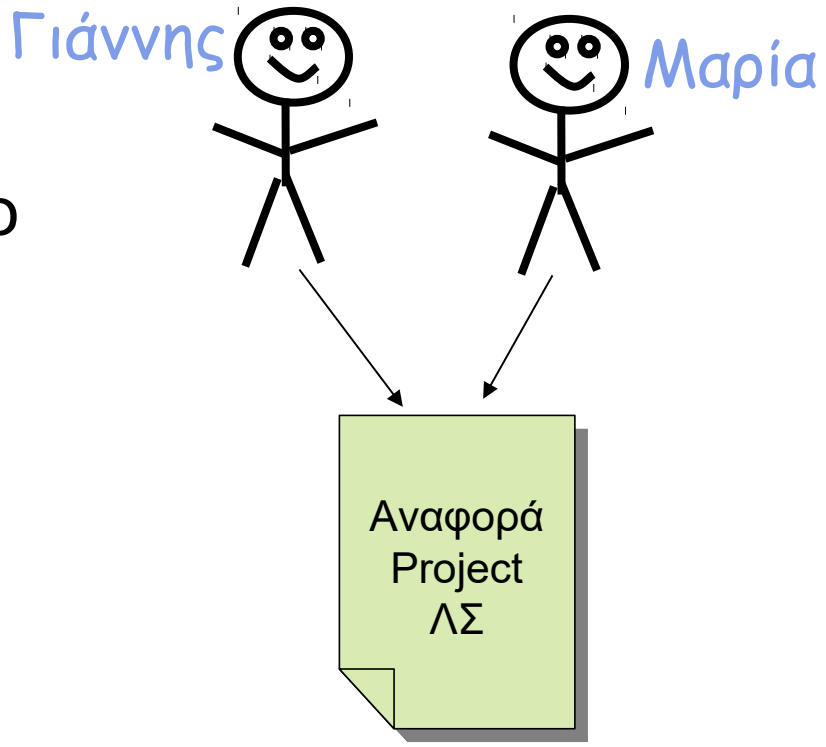
Ταυτοχρονισμός, Συγχρονισμός

Χρήστος Δ. Αντωνόπουλος
16/3/2017

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Πολλαπλές Διεργασίες/Νήματα σε 1 Κοινωνία: Προβλήματα;

- «Κοινωνικές διεργασίες/νήματα»: Διαβάζουν/γράφουν στον ίδιο χώρο.
  - Αποτέλεσμα;
  - Πιθανότατα χάος…
- Πότε να νιώσω ασφάλεια;
  - Απομονωμένες διεργασίες/νήματα
    - Σπάνιο…
  - Αν μόνο διαβάζουν;



Η Πτώχευση της Τράπεζας…

Υπόλοιπο λογαριασμού: 5000.00 €

Χρήστος	Ελένη	Τράπεζα
balance(5000) = account balance		
Ανάληψη 1000 €	balance(5000) = account balance	
balance(4000) = balance-withdrawal	Ανάληψη 2000 €	
	balance(3000) = balance-withdrawal	
	account balance(3000) = balance	
account balance(4000) = balance		

Υπόλοιπο Λογαριασμού : 4000 € ☺☺☺☺☺☺ ☹ ~@#$&^@!#

Ο Τρελός Μετρητής (1/2)

Αρχικά: i = 0; j τοπική μεταβλητή

P1, P2:

```
for (j = 0; j<500; j++)
  i++;
```

i++:

```
read (&i) -> reg
inc(reg)
write reg -> (&i)
```

- Αποτέλεσμα;
- Γιατί;
  - Μα αφού εγώ εκτελώ μια απλή εντολή!

Ο Τρελός Μετρητής (2/2)

i++:
read (&i) -> reg
inc(reg)
write reg -> (&i)

P1	P2
read (&i) (5) -> reg	
inc(reg)	
write reg -> (&i) (6)	
	read (&i) (6) -> reg
	inc (reg)
	write reg-> (&i) (7)
read (&i) (5) -> reg	read (&i) (5) -> reg
inc(reg)	
write reg -> (&i) (6)	inc(reg)
	write reg-> (&i) (6)

- Πρόβλημα συγχρονισμού (race condition)
- Δύσκολο να βρεθεί
  - Άλλοτε εμφανίζεται, άλλοτε όχι…

Λύσεις;

- Αδιαφορούμε…
  - Απλό ☺
  - Λίγες οι περιπτώσεις που δε μας ενδιαφέρει η ακριβής τιμή. ☹
- Δουλεύουμε σε αντίγραφα.
  - Απλό ☺
  - Όταν είναι δυνατό… ☹
- Λύνουμε το πρόβλημα!
  - Και ποιο είναι το πρόβλημα;
    - «Κακές» εναλλαγές διεργασιών

Ατομικότητα

- Κάνε πολλές εντολές να εκτελούνται σαν μία!
  - Μονάδα ατομικότητας: Οι εντολές που πρέπει να εκτελεστούν μαζί.
    - Καλλιτεχνικό ψευδώνυμο: Κρίσιμη περιοχή.
    - Αν δύο νήματα επιχειρήσουν να εκτελέσουν ταυτόχρονα κώδικα στην κρίσιμη περιοχή, το ένα περιμένει μέχρι να βγει το άλλο

P1

```
read (&i) (5) -> reg
inc(reg)
write reg -> (&i) (6)
```

P2

```
read (&i) (6) -> reg
inc (reg)
write reg-> (&i) (7)
```

Αλλά ΠΩΣ; (Μονοεπεξεργαστικά Συστήματα)

- Μονοεπεξεργαστικά συστήματα:
  - Αρκεί να μη διακοπεί το νήμα όταν βρίσκεται στην κρίσιμη περιοχή
    - Έλεγχος από το χρονοδρομολογητή του λειτουργικού

```
while(1) {
        Σταμάτησε το νήμα που εκτελείται
        Αν pc εκτός κρίσιμης περιοχής{
                Σώσε την κατάσταση του προηγούμενου νήματος
                Διάλεξε νέο νήμα
                Φόρτωσε την κατάσταση του νέου νήματος
                Ξεκίνησε το νέο νήμα}
```

    - Δουλεύει; ☺
    - Τι απαιτεί; ☹

Αλλά ΠΩΣ;

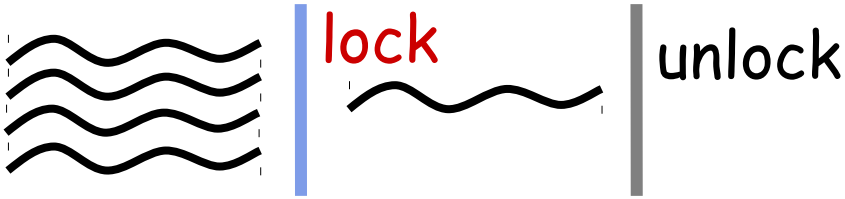
- Μονοεπεξεργαστικά συστήματα:
  - Σταμάτησε τις διακοπές από το ΛΣ

```
disable_interrupts()
i = i + 1
enable_interrupts()
```

  - Απλό ☺
  - Καλό; ☹
    - Τι δουλειά κάνουν οι διακοπές;
    - Αν πέσει σε άπειρο loop;
    - Αν ξεχάσει ο προγραμματιστής να ενεργοποιήσει τις διακοπές;

Αλλά ΠΩΣ; (Πολυεπεξεργαστικά Συστήματα)

- Δουλεύουν οι προηγούμενες λύσεις;
- Πρέπει να εμποδιστούν πολλαπλά νήματα (ίσως σε διαφορετικούς επεξεργαστές) να εκτελέσουν ταυτόχρονα κώδικα στην κρίσιμη περιοχή
  - Ρίξε το μπαλάκι στον κατασκευαστή
    - Να μου φτιάξει ατομική εντολή που να αυξάνει τον counter
    - Και αν θέλεις να αλλάξεις μια εγγραφή σε βάση δεδομένων;
  - Βάλε “πορτιέρη” (lock):
    - Μόνο ένα νήμα περνάει.
    - Τα υπόλοιπα «τρώνε πόρτα» έως ότου βγει…



Πώς το χρησιμοποιώ;

- Δύο εντολές
  - πάρε(lock)
  - άφησε(lock)

```
πάρε (lock)
Προστατευμένος κώδικας
άφησε(lock)
```

- Αμοιβαία αποκλειόμενη πρόσβαση (mutual exclusion)
  - Για τους φίλους mutexes
- Κανόνες:
  - Προστάτευσε κάθε κοινή μεταβλητή με lock
  - Κάθε φορά που θα αγγίξεις κοινή μεταβλητή πάρε το lock πριν την αγγίξεις
  - Αν θέλεις να προσπελάσεις ατομικά πολλές μεταβλητές, πάρε όλα τα locks πριν αρχίσεις και μην τα αφήσεις πριν τελειώσεις

Πώς τα Φτιάχνω; (Μονοεπεξεργαστικό Σύστημα, Σκηνή 1)

- Απλό;

```
πάρε (lock) {
        Όσο (lock == ΠΙΑΣΜΕΝΟ) ;
        lock = ΠΙΑΣΜΕΝΟ;
}

άφησε (lock) {
        lock = ΕΛΕΥΘΕΡΟ;
}
```

- Δουλεύει;

Πώς τα Φτιάχνω; (Μονοεπεξεργαστικό Σύστημα, Σκηνή 2)

- Τα μεγάλα όπλα: Απαγόρευση διακοπής

```
πάρε(lock) {
    απενεργοποίησε_διακοπή();
    όσο (lock == ΠΙΑΣΜΕΝΟ);
    lock = ΠΙΑΣΜΕΝΟ;
    ενεργοποίησε_διακοπή();
}

άφησε(lock) {
  lock = ΕΛΕΥΘΕΡΟ;
}
```

- Δουλεύει;
  - Κι αν το lock βρεθεί όντως πιασμένο;

Πώς τα Φτιάχνω; (Μονοεπεξεργαστικό Σύστημα, Σκηνή 3, Τελική!)

```
πάρε(lock) {
  δικό_μου = ΟΧΙ;
  Όσο(!δικό_μου){
    απενεργοποίησε_διακοπή();
    Αν(lock == ΕΛΕΥΘΕΡΟ) {
      δικό_μου = ΝΑΙ;
      lock = ΠΙΑΣΜΕΝΟ;
    }
    ενεργοποίησε_διακοπή();
  }
}

άφησε(lock) {
  lock = ΕΛΕΥΘΕΡΟ;
}
```

- Δουλεύει;
- Κανένα πρόβλημα (επίδοσης);

Και αν δεν Απενεργοποιούνται οι Διακοπές; (Αλγόριθμος Φούρναρη)

- Κάνε ότι γίνεται στην τράπεζα
  - Πάρε χαρτάκι με αριθμό
  - Περίμενε έως ότου εξυπηρετηθούν όλοι οι προηγούμενοι.
- Κανένα πρόβλημα;
  - Μπορούν 2 να πάρουν το ίδιο εισιτήριο;
    - Στην τράπεζα όχι
    - Στον υπολογιστή;
      - Χρησιμοποίησε και κάτι άλλο για να σπάσεις τις ισοπαλίες (π.χ. ποιος έχει το μικρότερο process ID)

Αλγόριθμος Φούρναρη

- Φάση 1
  - Κοίταξε όλα τα νούμερα που κυκλοφορούν. Βρες το μεγαλύτερο. Πρόσθεσε 1. Αυτό είναι το νούμερό σου.
- Φάση 2
  - Περίμενε έως ότου το δικό σου να είναι το χαμηλότερο νούμερο
    - Αν βρεις κι άλλο ίδιο νούμερο, κοίτα τον αριθμό ταυτότητας ;-)
      - Κερδίζει αυτός που έχει το μικρότερο (ο πιο ηλικιωμένος)

Και σε Πολυεπεξεργαστικά Συστήματα;

- Απενεργοποίηση διακοπής σε όλους τους επεξεργαστές:
  - Δύσκολο
  - Δεν επιτρέπεται
  - Και τι γίνεται με αυτούς που τρέχουν σε διαφορετικούς επεξεργαστές;
- Ρίξε το μπαλάκι στον κατασκευαστή …
  - Ειδικές απλές, ατομικές εντολές
    - Διάβασμα και εγγραφή σε μία κίνηση

π.χ. **atomic_swap** (mem, reg)

Άλλαξε σε μία κίνηση τα περιεχόμενα μιας θέσης μνήμης με αυτά ενός καταχωρητή

Πώς το Φτιάχνω; (ξανά…)

```
πάρε(lock) {
        δικο_μου = ΠΙΑΣΜΕΝΟ;
        Όσο (δικό_μου == ΠΙΑΣΜΕΝΟ)
                atomic_swap (δικο_μου, lock);
}
άφησε(lock) {
        lock = ΕΛΕΥΘΕΡΟ;
}
```

- Δουλεύει;
- Καλλιτεχνικό ψευδώνυμο: spin lock (ενεργής αναμονής), γνωστό και ως test and set lock
  - Είναι καλό (αποδοτικό);

Ενεργή Αναμονή ή Εθελοντική Διακοπή;

- Μειονεκτήματα;
  - Ενεργής αναμονής
  - Εθελοντικής διακοπής
- Πλεονεκτήματα;
  - Ενεργής αναμονής
  - Εθελοντικής διακοπής
- Και τι είναι το καλύτερο;
  - Θα απελευθερωθεί το lock γρήγορα;
    - Ναι: Ενεργή αναμονή
    - Όχι: Εθελοντική διακοπή
  - Και που να ξέρω;
    - Ενεργή αναμονή για 2 φορές το κόστος της εθελοντικής διακοπής
    - Μετά διακοπή
    - Φράζει το κόστος σε 2 φορές το πολύ από το βέλτιστο
    - Εφαρμόζεται και στη ζωή σας…

Απαιτήσεις!!!

- Απαιτήσεις
  - Αμοιβαίος αποκλεισμός
    - Δηλαδή να δουλεύει… Λογικό…
  - Πρόοδος
    - Δε θα πρέπει να σταματούν όλοι. Κάποιος θα πρέπει να προχωράει.
    - Δε θα πρέπει να επιτρέπουμε σε αυτούς που δεν «παίζουν» να μας χαλάνε το «παιχνίδι»
  - Μη απεριόριστη αναμονή
    - Ε, δε θα είμαστε εδώ και για πάντα…
- Ναι, τίποτα άλλο; Ε, καλό θα ήταν…
  - Δικαιοσύνη
  - Αποδοτικότητα (μην πετάξεις τίποτα…)
  - Απλότητα (τώρα μάλιστα…)

Τρώγοντας Έρχεται η Όρεξη… Λίγο Αποδοτικότερα Spin Locks

```
πάρε(lock) {
        δικό_μου = ΠΙΑΣΜΕΝΟ;
        Όσο (δικό_μου == ΠΙΑΣΜΕΝΟ)
                atomic_swap (δικό_μου, lock);
}
άφησε(lock) {
        lock = ΕΛΕΥΘΕΡΟ;
}
```

- Γνωστό και ως test and test and set lock
  - Είναι καλό (αποδοτικό);
  - Το `lock`, πού αποθηκεύεται;
  - Ικανοποιεί τις απαιτήσεις;

```
πάρε(lock) {
   δικό_μου = ΠΙΑΣΜΕΝΟ;
   όσο (δικό_μου == ΠΙΑΣΜΕΝΟ) {
          όσο  (lock == ΠΙΑΣΜΕΝΟ);
          atomic_swap (δικό_μου,lock);
   }
}

άφησε(lock) {
   lock = ΕΛΕΥΘΕΡΟ;
}
```

Κάτι Πιο Ντετερμινιστικό…

```
πάρε(lock) {
   περιμένω[i] = ΝΑΙ;
   δικό_μου = ΠΙΑΣΜΕΝΟ;
   όσο (περιμένω[i] == ΝΑΙ && δικό_μου == ΠΙΑΣΜΕΝΟ) {
          όσο  (περιμένω[i] == ΝΑΙ && lock == ΠΙΑΣΜΕΝΟ);
          atomic_swap (δικό_μου, lock);
  }
   περιμένω [i] = ΟΧΙ;
}
```

```
άφησε(lock) {
   κάνε {
          j = (i+1)%N
   } όσο (j != i && περιμένω[i] == ΟΧΙ);
   Αν (j == i)
          lock = ΕΛΕΥΘΕΡΟ;
   διαφορετικά
          περιμένω[i] = ΟΧΙ;
}
```

Δύσκολο Είναι… Απλούστερες Ιδέες; - Σηματοφόροι

- Τι θέλουμε;
  - Κάτι απλούστερο από τα mutexes (locks).
    - Δε θέλω να ξέρω πώς να το φτιάξω! Το θέλω έτοιμο!
  - Αλλά όχι τόσο απλό που να μην είναι αποδοτικό
- Τι είναι;
  - Μετρητής >=0, τον οποίο μπορείς να αυξήσεις ή να μειώσεις ατομικά.
    - Αν κάποιος πάει να τον κάνει <0, μπλοκάρει
- Λειτουργίες;
  - p()/down()/wait() (proberen)
    - Μείωσε κατά 1 το μετρητή. Αν τον βρεις 0, κοιμήσου (μπλόκαρε) αντί να τον μειώσεις.
  - v()/up()/signal() (verhogen)
    - Αύξησε κατά 1 το μετρητή. Αν πριν ήταν 0, ξύπνα τυχόν υπναράδες
  - init()
    - Κάπως πρέπει να ξεκινήσει κανείς στη ζωή…
- Υλοποίηση
  - Μετρητής
  - Ουρά αναμονής

```
p (mutex)
Προστατευμένος κώδικας
v (mutex)
```

Πώς να Καταστρέψετε το Πρόγραμμά σας με Σηματοφόρους

```
v (mutex)
Προστατευμένος κώδικας
p (mutex)
```

```
p (mutex)
Προστατευμένος κώδικας
```

```
p (mutex)
Προστατευμένος κώδικας
p (mutex)
```

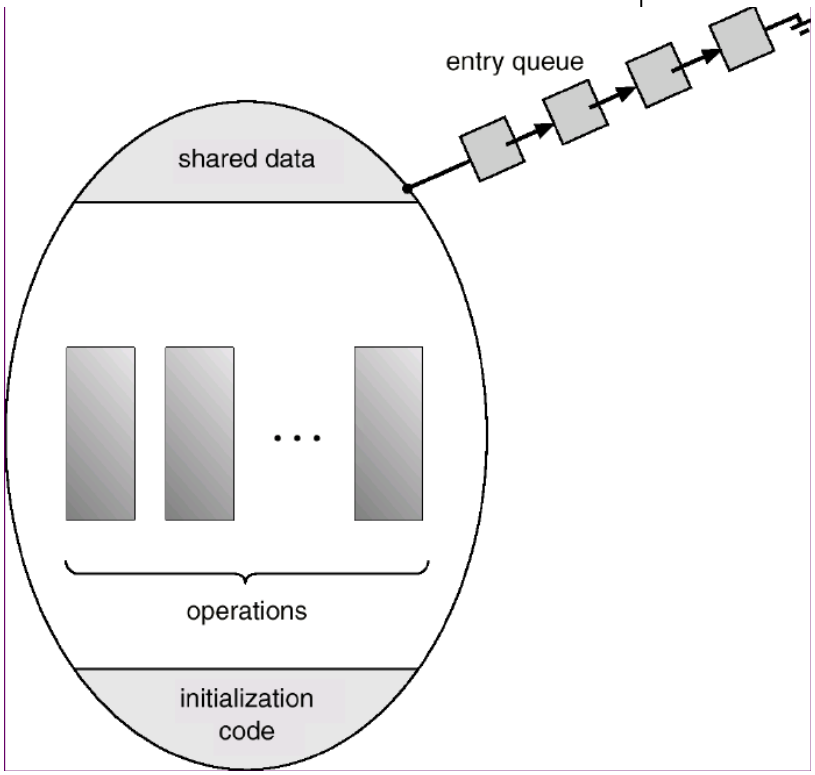
```
Προστατευμένος κώδικας
v (mutex)
```

```
Προστατευμένος κώδικας
```

- Κάτι σε λίγο υψηλότερο επίπεδο;

Monitors (Ελεγκτές)

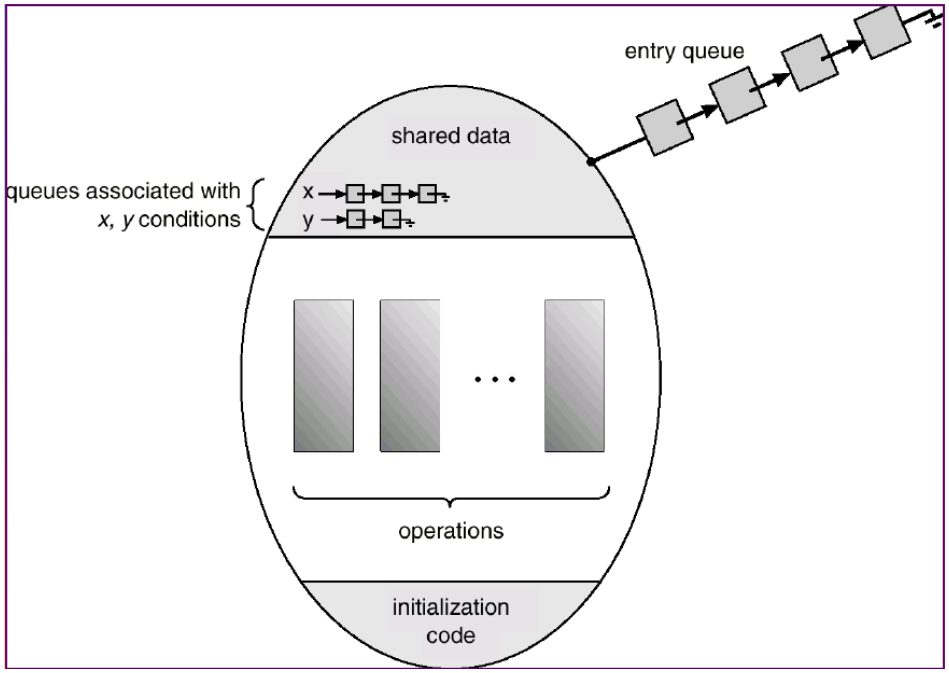
- Ρίξε το μπαλάκι
  - Αυτή τη φορά στο μεταγλωττιστή
- Τι είναι;
  - Κατασκευή σε υψηλού επιπέδου γλώσσα
  - Κάτι σαν τις κλάσεις στις αντικειμενοστρεφείς γλώσσες
- Τι περιέχει;
  - Δεδομένα, συναρτήσεις που μπορούν να τα μεταβάλλουν
  - Όλα προστατευμένα από ένα (υποννοούμενο) lock.




```
monitor Queue {
        int head, tail;   // Διαμοιραζόμενα
                          // δεδομένα
        void enq(val) {}  // Βάλε το val στην
                          // ουρά
        int deq() {}      // Βγάλε κάτι από
                          // την ουρά
}
```

Ελεγκτές: Όλα ή Τίποτα; Μεταβλητές Συνθήκης

- Μόνο μια διεργασία στο monitor
  - Και αν θέλουμε πιο «λεπτό» συγχρονισμό;
    - Μεταβλητές συνθήκης (condition variables)
  - wait (condition)
    - Άφησε το lock του monitor, κοιμήσου, ξαναπάρε το lock του monitor όταν ξυπνήσεις.
  - signal (condition)
    - Ξύπνησε μία υπναρού (αν υπάρχει)
  - Και ποιά θα συνεχίσει;
    - Αυτή που έτρεχε ή αυτή που ξύπνησε;
    - Hoare: Αυτή που ξύπνησε
    - MESA: Αυτή που έτρεχε



Διάσημα Προβλήματα: Πεπερασμένος Buffer

- n θέσεις στον buffer
- Ανεξάρτητοι παραγωγοί – καταναλωτές
  - Παραγωγοί: Περιμένουν αν ο buffer γεμίσει
  - Καταναλωτές: Περιμένουν αν ο buffer είναι άδειος

empty = n; full = 0;

ΠΑΡΑΓΩΓΟΙ

```
do {
    …
    φτιάξε στοιχείο
    …
    p(empty)
    p(mutex)
    βάλε στοιχείο στον buffer
    v(mutex)
    v(full)
    …
} while(1)
```

ΚΑΤΑΝΑΛΩΤΕΣ

```
do {
    …
    p(full)
    p(mutex)
    βγάλε στοιχείο από τον buffer
    v(mutex)
    v(empty)
    …
    κατανάλωσε στοιχείο
    …
} while(1)
```

Διάσημα Προβλήματα: Εγγραφείς / Αναγνώστες

- Κοινά δεδομένα
  - Προσπελαύνονται από αναγνώστες και εγγραφείς
  - Πολλοί αναγνώστες: ΟΚ
  - Πολλοί αναγνώστες και εγγραφέας: Πρόβλημα
  - Πολλοί εγγραφείς: Πρόβλημα
  - Ένας εγγραφέας: ΟΚ
- Ποιος έχει προτεραιότητα;
  - Εγγραφείς;
  - Αναγνώστες;

Διάσημα Προβλήματα: Εγγραφείς / Αναγνώστες (Λύση)

```
mutex=1; wrt_mutex=1;
int readcount = 0;
```

ΕΓΓΡΑΦΕΙΣ

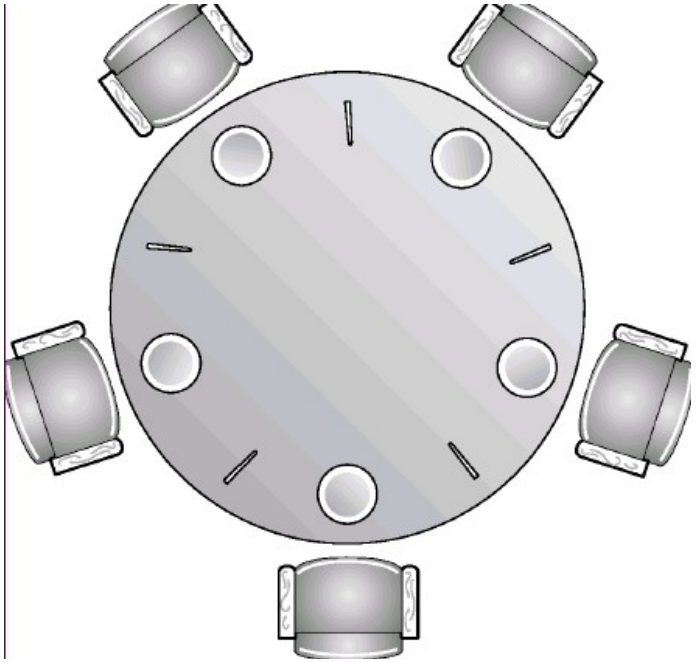
```
p (wrt_mutex)
…
Γράψε
…
v (wrt_mutex)
```

ΑΝΑΓΝΩΣΤΕΣ

```
p (mutex)
readcount++
if (readcount == 1)
   p (wrt_mutex)
v (mutex)
…
Διάβασε
…
p (mutex)
readcount--
if (readcount == 0)
   v (wrt_mutex)
v (mutex)
```

Διάσημα Προβλήματα: Συνδαιτυμόνες Φιλόσοφοι

- Ν κινέζοι φιλόσοφοι
  - Φιλοσοφούν
  - Πεινάνε
  - Τρώνε
    - Τους αρέσει (φυσικά) το ρύζι
    - Το τρώνε (φυσικά) με ξυλάκια
  - Ξαναφιλοσοφούν
    - Σκυλίσια ζωή…
- Μπορούν να φάνε όλοι μαζί;



Διάσημα Προβλήματα: Συνδαιτυμόνες Φιλόσοφοι (Λύση;)

```
chopstick_mutex[N] = {1,1,1…,1}

ΦΙΛΟΣΟΦΟΣ i

do {
   Σκέψου…
   …
   p(chopstick[i])
   p(chopstick[(i+1)%N])
   Φάε…
   …
   v(chopstick[(i+1)%N])
   v(chopstick[i]
}
```

- Σωστό;
  - Παίζουμε;
  - Καθήστε όλοι σε έναν κύκλο.
  - Πιάστε όλοι πρώτα το δεξί και μετά το αριστερό chopstick!
    - Μην κοιτάς το chopstick ΜΟΥ. Ούτε να το σκέφτεσαι!
    - Oops… αδιέξοδο (deadlock)!