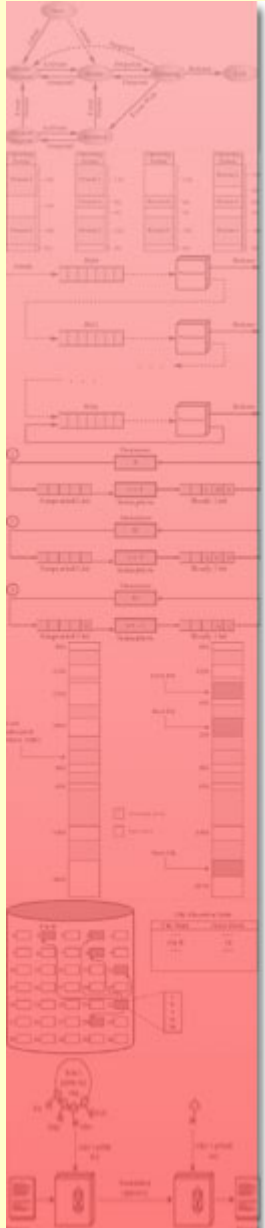


Προϋποθέσεις για Αμοιβαίο Αποκλεισμό

- Μόνο μία διεργασία σε κρίσιμο τμήμα σε κοινό πόρο
- Μία διεργασία που σταματά σε μη κρίσιμο σημείο δεν πρέπει να επιρεάζει τις υπόλοιπες διεργασίες
- Δεν πρέπει να υπάρχει κατάσταση αδιεξόδου ή παρατεταμένης στέρξης για μια διεργασία που θέλει να εισέλθει στο κρίσιμο τμήμα της
- Μια διεργασία δεν πρέπει να καθυστερείτε να εισέλθει στο κρίσιμο τμήμα της αν δεν βρίσκεται κάποια άλλη διεργασία σε κρίσιμο τμήμα
- Δεν πρέπει να γίνονται υποθέσεις για την σχετική ταχύτητα των διεργασιών
- Μια διεργασία εκτελεί κρίσιμο τμήμα για ορισμένο χρονικό διάστημα

Dr. Garmpis Aristogiannis - EPDO

TEI Messolonghi



Συνρουτίνα(Coroutine)

- Σχεδιασμένες να ανταλλάσσουν μεταξύ τους τον έλεγχο της εκτέλεσης
- Δεν επαρκούν για την υποστήριξη ταυτόχρονης επεξεργασίας



Ο Αλγόριθμος του Dekker [DJK65]

- Πρώτη προσπάθεια - Ενεργός Αναμονή (Busy Waiting)
 - Η διεργασία ελέγχει συνεχώς αν μπορεί να εισέλθει στην κρίσιμη περιοχή
 - Η διεργασία δεν μπορεί να κάνει κάτι παραγωγικό μέχρι να της επιτραπεί να εισέλθει στην κρίσιμη περιοχή

P0

```
while (turn != 0) /**/;
```

critical section

Dr. Garmpis Aristogiannis - EPDO

turn = 1;

P1

```
while (turn != 1) /**/;
```

critical section

turn = 0;



Δεύτερη Προσπάθεια

- Κάθε διεργασία ελέγχει την κατάσταση της άλλη αλλά δεν μπορεί να την αλλάξει
- Όταν μια διεργασία θέλει να εισέλθει στην κρίσιμη περιοχή ελέγχει την κατάσταση της άλλης
- Αν καμία άλλη διεργασία δεν είναι σε κρίσιμη περιοχή θέσε την κατάσταση σου για είσοδο στην κρίσιμη περιοχή
- Προβλήματα
 - Η μέθοδος δεν εξασφαλίζει αμοιβαίο αποκλεισμό
 - Κάθε διαδικασία μπορεί να ελέγχει και να εισέρχεται στην κρίσιμη περιοχή ταυτόχρονα

P0

```
while ( flag[1] ) /**/;  
flag[0] = true;
```

critical section

```
flag[0] = false;
```

P1

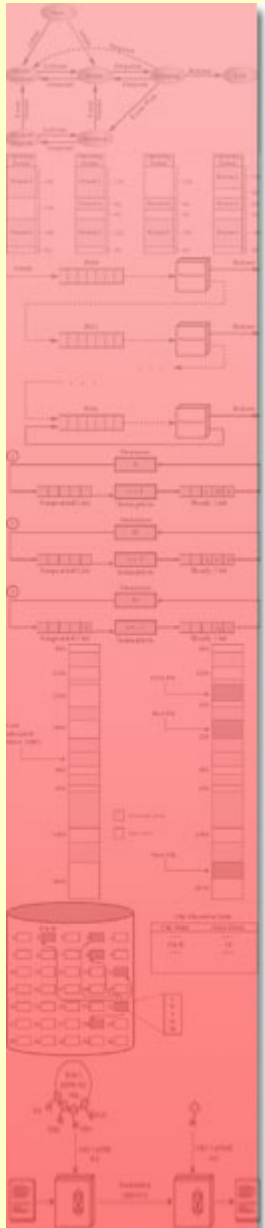
```
while ( flag[0] ) /**/;  
flag[1] = true;
```

critical section

```
flag[1] = false;
```

Dr. Garmpis Anastasios - EPDO

TEI Messolonghi



Τρίτη προσπάθεια

- Θέσε την κατάσταση σου για είσοδο στην κρίσιμη περιοχή πριν τον έλεγχο των άλλων διεργασιών
- Αν κάποια άλλη διεργασία είναι σε κρίσιμη περιοχή όταν τεθεί η μεταβλητή ελέγχου, η διεργασία μπλοκάρεται μέχρι η άλλη διεργασία ελευθερώσει την κρίσιμη περιοχή
- Πρόβλημα
 - Πιθανός αδιέξοδος αν και οι δύο διεργασίες θέσουν την μεταβλητή ελέγχου και εισέλθουν στην κρίσιμη περιοχή, μια και κάθε διεργασία θα περιμένει την άλλη να απελευθερώσει την κρίσιμη περιοχή

P0

```
flag[0] = true;
```

```
while ( flag[1] ) /**/;
```

critical section

```
flag[0] = false;
```

Dr. Garmpris Aristogiannis - EPDO

TEI Messolonghi

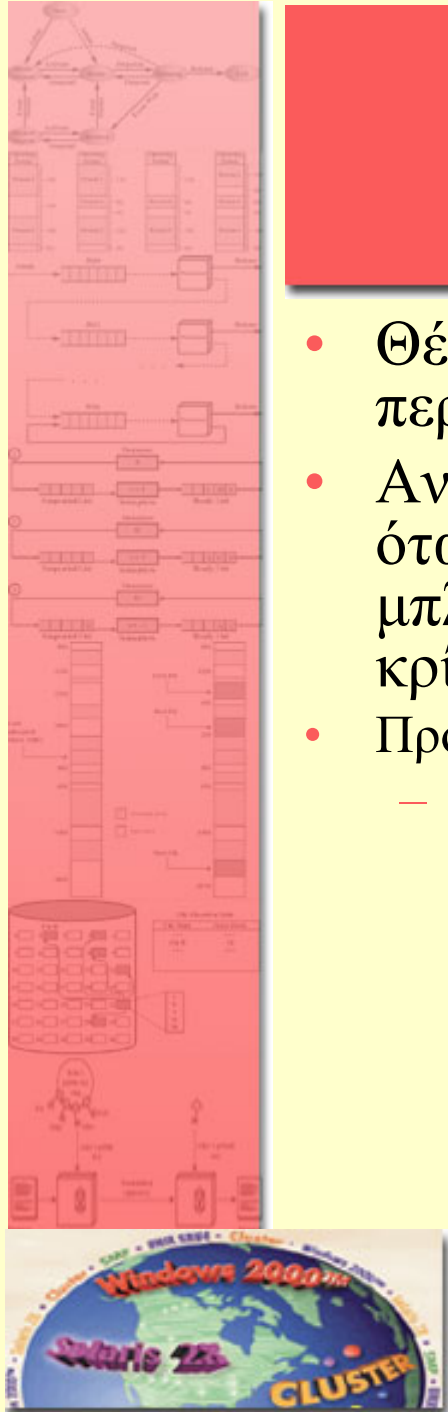
P1

```
flag[1] = true;
```

```
while ( flag[0] ) /**/;
```

critical section

```
flag[1] = false;
```



Τέταρτη προσπάθεια

- Η διεργασία θέτει την μεταβλητή ελέγχου για να δείξει ότι επιθυμεί να εισέλθει στην κρίσιμη περιοχή, αλλά είναι έτοιμη να αναιρέσει
- Ελέγχονται οι άλλες διεργασίες. Αν βρίσκονται σε κρίσιμη περιοχή η μεταβλητή ελέγχου αναιρείται. Επανατίθεται, μέχρι να εισέλθει στην κρίσιμη περιοχή, όταν επιθυμεί να εισέλθει στην κρίσιμη περιοχή
- Πρόβλημα ???
 - Μπορεί να υπάρξουν κύκλοι ενεργοποίησης και απενεργοποίησης των μεταβλητών ελέγχου

P0

```
flag[0] = true;
while ( flag[1] )
{
    flag[0] = false;
    delay
    flag[0] = false;
}
```

Critical section

```
flag[0] = false;
```

P1

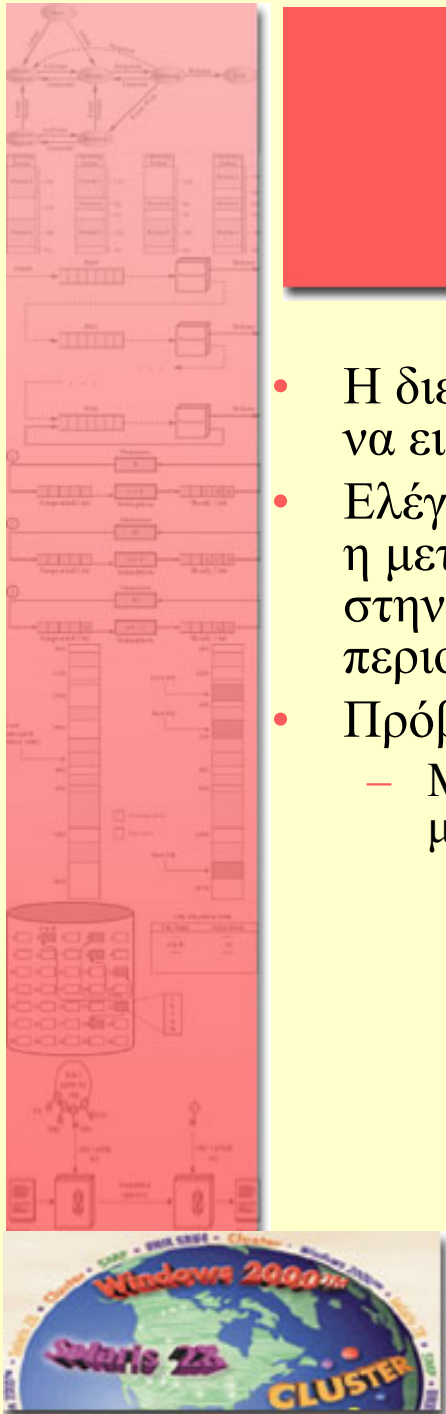
```
flag[1] = true;
while ( flag[0] )
{
    flag[1] = false;
    delay
    flag[1] = false;
}
```

Critical section

```
flag[1] = false;
```

Dr. Vasilios A. Giannis - EPDC

TEI Messolonghi



Σωστές Λύσεις

- Αλγόριθμος Dekker
 - Κάθε διεργασία μπαίνει με την σειρά της στην κρίσιμη περιοχή
 - Αν μια διεργασία θέλει να εισέλθει στην κρίσιμη περιοχή, θέτει την μεταβλητή ελέγχου και μπορεί να περιμένει την σειρά της για να εισέλθει στην κρίσιμη περιοχή.
- Αλγόριθμος Peterson

P0

```
flag[0] = true;
```

```
turn = 1;
```

```
while ( flag[1] &&  
        turn == 1 ) /**/;
```

critical section

```
flag[0] = false;
```

P1

```
flag[1] = true;
```

```
turn = 0;
```

```
while ( flag[0] &&  
        turn == 0 ) /**/;
```

critical section

```
flag[1] = false;
```

Dr. Garmpis Aristogiannis - EPDO

TEI Messolonghi



Αμοιβαίος Αποκλεισμός: Υποστήριξη από το Υλικό

- Απενεργοποίηση διακοπών
 - Μια διεργασία εκτελείται μέχρι να κάνει κλήση υπηρεσίας του Λ.Σ. ή να διακοπεί
 - Η απενεργοποίηση των διακοπών εξασφαλίζει αμοιβαίο αποκλεισμό
 - Ο επεξεργαστής περιορίζεται στην δυνατότητα εναλλαγής διεργασιών
 - Πολυπεξεργασία
 - Δεν εξασφαλίζεται αμοιβαίος αποκλεισμός με την απενεργοποίηση των διακοπών σε ένα επεξεργαστή



Αμοιβαίος Αποκλεισμός: Υποστήριξη από το Υλικό

- Ειδικές Εντολές Μηχανής
 - Εκτελούνται σε έναν κύκλο εντολών
 - Δεν επηρεάζονται από άλλες εντολές
- Διάβασμα και γράψιμο

```
boolean testset (int i) {  
    if (i == 0)  
        {i = 1; return true;}  
    else  
        {return false;}  
}
```



Αμοιβαίος Αποκλεισμός: Υποστήριξη από το Υλικό

- Πλεονεκτήματα
 - Εφαρμόσιμη σε κάθε αριθμό διεργασιών και επεξεργαστών που μοιράζονται κοινή μνήμη
 - Είναι απλή και άρα εύκολα ελέγξιμη
 - Μπορεί να υποστηρίξει πολλαπλές κρίσιμες περιοχές
- Μειονεκτήματα
 - Busy-waiting καταναλώνει υπολογιστικό χρόνο
 - Υπάρχει πιθανότητα παρατεταμένης στέρησης όταν μία διεργασία βγαίνει από την κρίσιμη περιοχή και περισσότερες από μια διεργασίες περιμένουν να εισέλθουν
 - Αδιέξοδο, αν μια διεργασία με χαμηλότερη προτεραιότητα έχει την κρίσιμη περιοχή και μία διεργασία με μεγαλύτερη προτεραιότητα πάρει τον επεξεργαστή και περιμένει να απελευθερωθεί η κρίσιμη περιοχή



Σημαφόροι (Semaphores)

- Ειδικές μεταβλητές που ονομάζονται σημαφόροι χρησιμοποιούνται για σήμανση
- Αν μια διεργασία περιμένει για ένα σήμα, απενεργοποιείται μέχρι να σταλεί το σήμα
- Η αναμονή και η εντολές σήμανσης δεν διακόπτονται
- Χρησιμοποιείται ουρά που κρατά τις διεργασίες που περιμένουν έναν σεμαφόρο
- Είναι μεταβλητές με ακέραια τιμή
 - Μπορεί να αρχικοποιηθεί σε θετική τιμή
 - Η κατάσταση αναμονής μειώνει την τιμή του σεμαφόρου
 - Εντολή σήματος αυξάνουν την τιμή του σεμαφόρου



Το πρόβλημα Παραγωγού - Καταναλωτή

- Ένας ή περισσότεροι παραγωγοί δημιουργούν δεδομένα και τα τοποθετούν σε ένα buffer
- Ένας καταναλωτής παίρνει δεδομένα από τον buffer, ένα τη φορά
- Μόνο ένας παραγωγός ή καταναλωτής μπορεί να έχει πρόσβαση στον buffer κάθε χρονική στιγμή



Το πρόβλημα Παραγωγού - Καταναλωτή

```
producer:  
while (true) {  
/*produce item v */  
  b[in] = v;  
  in++;  
}
```

```
consumer:  
while (true) {  
  while (in <= out)  
    /*do nothing */;  
  w = b[out];  
  out++;  
  /* consume item w */  
}
```



Το πρόβλημα Παραγωγού – Καταναλωτή με κυκλικό buffer

producer:

```
while (true) {  
    /* produce item v */  
    while ((in + 1) % n == out)  
        /* do nothing */;  
    b[in] = v;  
    in = (in + 1) % n  
}
```

consumer:

```
while (true) {  
    while (in == out)  
        /* do nothing */;  
    w = b[out];  
    out = (out + 1) % n;  
    /* consume item w */  
}
```

