Λειτουργικά Συστήματα (ΗΥ222)

Διάλεξη 5:
Αδιέξοδα

Χρήστος Δ. Αντωνόπουλος
23/3/2017

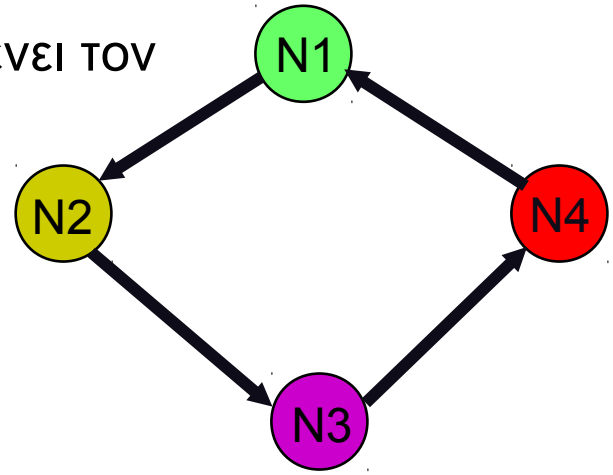
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Συγχρονισμός στον Πραγματικό Κόσμο

- Χρειάζεται οποτεδήποτε > 1 χρήστης πόρου
  - Λίγο πολύ οι ίδιες λύσεις με τον πραγματικό κόσμο: κλείδωμα (στο WC), χρονοπρογραμματισμός (ραντεβού), χρήση πολλαπλών αντιγράφων πόρων (όλοι έχουμε υπολογιστή σπίτι μας)
- Παραδείγματα:
  - Ένας δρόμος, πολλά αυτοκίνητα: φανάρια (συγχρονισμός με χρονοπρογραμματισμό), πολλές λωρίδες («ιδεατά» αντίγραφα – ανταλλαγή μη βέλτιστης χρησιμοποίησης με απλούστερο συγχρονισμό)
  - WC: Κλειδαριά πόρτα (lock), ανδρικές/γυναικείες (δημιουργία αντιγράφων)
  - Συνεργάτες στο project: lock = “πειράζω το sched.c” unlock = “OK, τελείωσα”
  - Παρκάρισμα: Χρήση αντικειμένου (αυτοκίνητο / τσιμεντόλιθος στη θέση), ανάθεση θέσεων parking (πάντα δεσμευμένες: όχι ταυτοχρονισμός = κακή αξιοποίηση), κάρτα μόνιμου κατοίκου (“είδος” δέσμευσης που επιτρέπει ταυτοχρονισμό)

Αδιέξοδο (Deadlock): Η καταστροφή του Παραλληλισμού

- Γραφικά: Κύκλος στο γράφο που δείχνει τις εξαρτήσεις μεταξύ νημάτων
  - Το νήμα Ν1 κρατάει τον πόρο Π1 και περιμένει τον Π2,
  - το Ν2 κρατάει τον Π2 και περιμένει τον Π3,
  - το Ν3 κρατάει τον Π3 και περιμένει τον Π4,
  - το Ν4 κρατάει τον Π4 και περιμένει τον Π1.



- Πρόβλημα:
  - Καμία πρόοδος (όλα είναι τόσο βαρετά)
  - Μπορεί να συμβεί εύκολα
  - Όταν συμβεί, άντε να το καταλάβεις

Παράδειγμα

- Τι μπορεί να γίνει και να έχουμε αδιέξοδο στον κώδικα;

```
/* M????ορά x Euros ??ό ?ο λογ?ρι?μό from ??ον to */
void transfer(account *from, account *to, int x)
   down(from->sema);
   down(to->sema):
   from->balance -= x;
   to->balance += x;
   up(from->sema);
   up(to->sema);
```

Παράδειγμα

- Είναι απαραίτητα τα locks / σηματοφόροι / monitors?
  - Μπορούμε μια χαρά να καταφέρουμε να έχουμε αδιέξοδο και χωρίς αυτά.

```
T1:                          T2:
send n bytes to T2;          while(receive up to 4K of data)
                                 process  data;
while(receive data);             send 4K result to T1
    display data             exit;
exit;
```

  - Buffers 4K
  - Αν γεμίσει ο buffer, μπλοκάρουμε στο send έως ότου ο άλλος πάρει δεδομένα
  - Τι γίνεται για διαφορετικές τιμές του n;

Συνθήκες για Αδιέξοδο

- Περιορισμένη πρόσβαση
  - Ο πόρος μπορεί να χρησιμοποιηθεί από έναν πεπερασμένο αριθμό χρηστών
- Μη διακοπή
  - Αν δοθεί ο πόρος δεν μπορούμε να τον πάρουμε πίσω
- Πολλαπλές ανεξάρτητες αιτήσεις (κρατάω και περιμένω)
  - Δε ζητώ όλους τους πόρους μονομιάς, δεσμεύω όσους μπορώ και περιμένω για τους υπόλοιπους
- Κύκλος στο γράφο αιτήσεων

- Είναι **απαραίτητες και οι 4** συνθήκες!!!

Αντιμετώπιση Αδιεξόδου

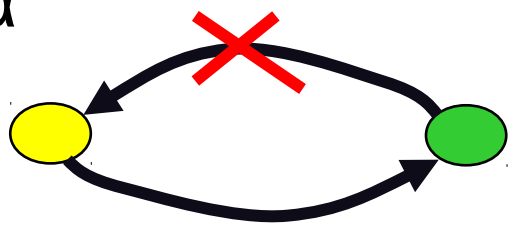
- Δύο βασικές στρατηγικές
  - Των φρονίμων τα παιδιά…
    - Πρόληψη
  - Όποιος δεν έχει μυαλό…
    - Αναγνώριση και διορθωτική δράση

Αποτροπή Αδιεξόδου: Εξαφάνισε 1 από τις 4 συνθήκες

- Περιορισμένη πρόσβαση:
  - Αγόρασε κι άλλους πόρους, κομμάτιασε τους πόρους όσο μπορείς, φτιάξε «ιδεατούς» πόρους ώστε να έχεις “άπειρα” αντίγραφα
- Μη διακοπή:
  - Φτιάξε αντίγραφα ή ιδεατούς πόρους
  - Νήματα: έχουν καθένα ένα δικό του αντίγραφο των καταχωρητών => δε χρειάζεται lock
  - Φυσική μνήμη: «Ιδεατός» κόσμος με την ιδεατή μνήμη: Μπορούμε να πάρουμε ένα τμήμα της φυσικής μνήμης από μια διεργασία και να το δώσουμε σε άλλη!

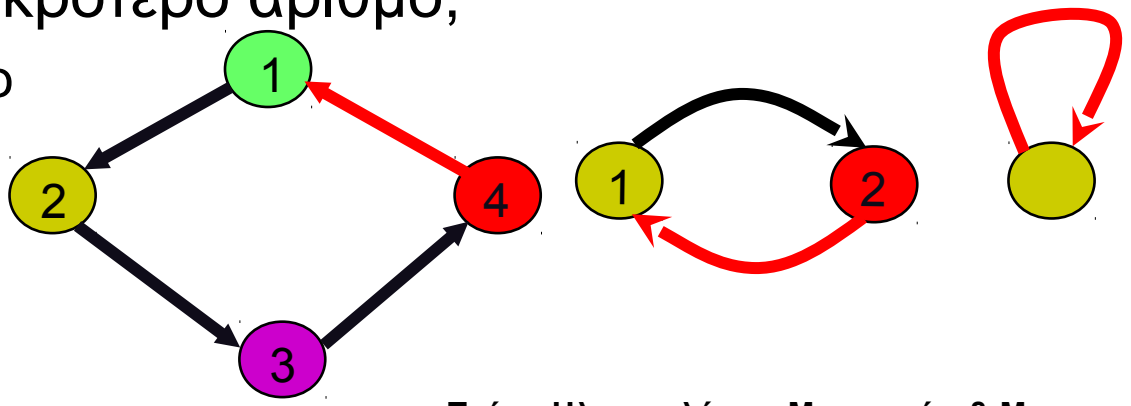
Αποτροπή Αδιεξόδου: Εξαφάνισε 1 από τις 4 συνθήκες

- Κρατάω και περιμένω:
  - Πάρε όλους τους πόρους μονομιάς (όταν είναι διαθέσιμοι)
  - Πρέπει να ξέρεις εξ' αρχής πόσους / ποιους χρειάζεσαι
- Κύκλοι
  - Ένα lock για όλο το σύστημα
    - Κανένα πρόβλημα;
  - Διάταξη των πόρων
    - (please wait…)



Διάταξη Πόρων

- Αρίθμησε (διέταξε τους πόρους): lock#1, lock#2, …
- Δέσμευσέ τους με συγκεκριμένη σειρά
  - Αύξουσα ή φθίνουσα, αλλά παντού και πάντα την ίδια.
- Γιατί δουλεύει;
  - Αριθμήστε όλους τους κόμβους στο γράφο εξαρτήσεων
  - Αν υπάρχει κύκλος δε θα υπάρχει ακμή από μεγαλύτερο σε μικρότερο αριθμό;
    - Ή στον ίδιο κόμβο



Δέσμευση σε 2 φάσεις

- Δέσμευσε όλους τους πόρους,
  - Αν κολλήσεις σε κάποιον αποδέσμευσε όλους και προσπάθησε ξανά
- Πλεονεκτήματα:
  - Δυναμικό, απλό, ευέλικτο
- Μειονεκτήματα:
  - Κόστος σε σχέση με τον αριθμό των πόρων;
  - Μήκος της προστατευμένης περιόχής;
  - Μπορώ να ξέρω τι θα χρειαστώ εκ των προτέρων;

```
print_file:
while(1) {
  while (!try_lock(file));
    if (!try_acquire_printer)
      release_lock(file);
    else
      if (!try_acquire _disk) {
        release_lock(file);
        release_printer();
      }
      else {
        ** do work **
        release all
        break;
      }
}
```

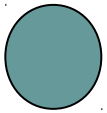
Διάγνωση + Δράση

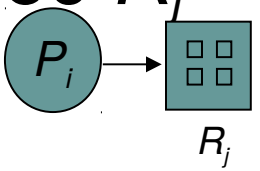
- Τερμάτισε ένα νήμα και απελευθέρωσε τους πόρους. Συνέχισε έως ότου εξαφανιστεί το αδιέξοδο
  - Μειονέκτημα: Σε τι κατάσταση μένει το σύστημα αν σκοτώνω νήματα όπου και όποτε λάχει;
    - Μάλλον δε μπορεί να είναι καλό…
  - Χρήση με στυλ: Πάρε όλους τους πόρους πριν κάνεις οποιαδήποτε αλλαγή στην κατάσταση.
    - Ουδείς αναντικατάστατος (πριν κάνω αλλαγές στην κατάσταση)
    - Σαν τη δέσμευση σε 2 φάσεις, απλά δε χρειάζεται να ασχοληθούμε με την αποδέσμευση των πόρων

Διάγνωση + Δράση

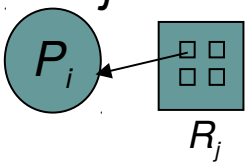
- Πιο “sexy” λύση
  - Αν συμβεί αδιέξοδο, «ακύρωσε» τις ενέργειες που σε οδήγησαν στο αδιέξοδο
    - Η μηχανή του χρόνου…
  - Δεσμεύουμε κανονικά πόρους
  - Κάνουμε αλλαγές στην κατάσταση του προγράμματος χρησιμοποιώντας μόνο αντιστρέψιμες πράξεις
  - Αν τα πράγματα πάνε στραβά σκοτώνουμε το «κακό» νήμα και απελευθερώνουμε τους πόρους του
    - Αλλά ταυτόχρονα αντιστρέφουμε τις αλλαγές που είχε κάνει στην κατάσταση
- Συναλλαγές (transactions)
  - Εύκολες για τον προγραμματιστή
  - Σχετικά δύσκολες στην υλοποίηση:
    - Πώς κρατάω ημερολόγιο των αλλαγών;
    - Πώς φτιάχνω τις αντίστροφες πράξεις;
    - Στο software ή στο hardware;

Πιο Τυπικά: Γράφος Ανάθεσης Πόρων

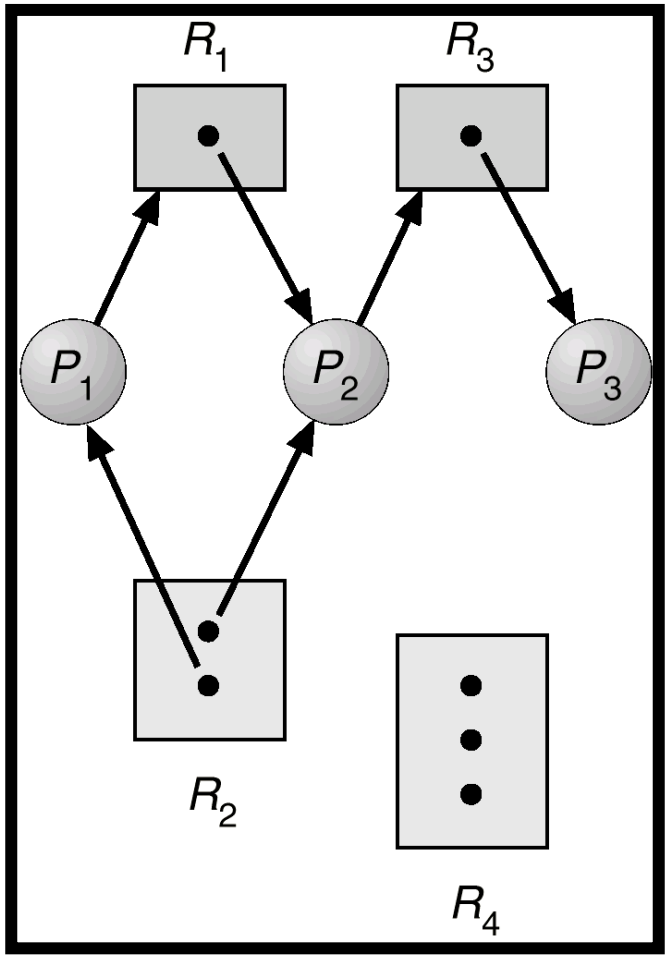
- Διεργασία 
- Πόρος με 4 ίδιες οντότητες 
- Η P_i κάνει αίτηση για μια οντότητα του R_j



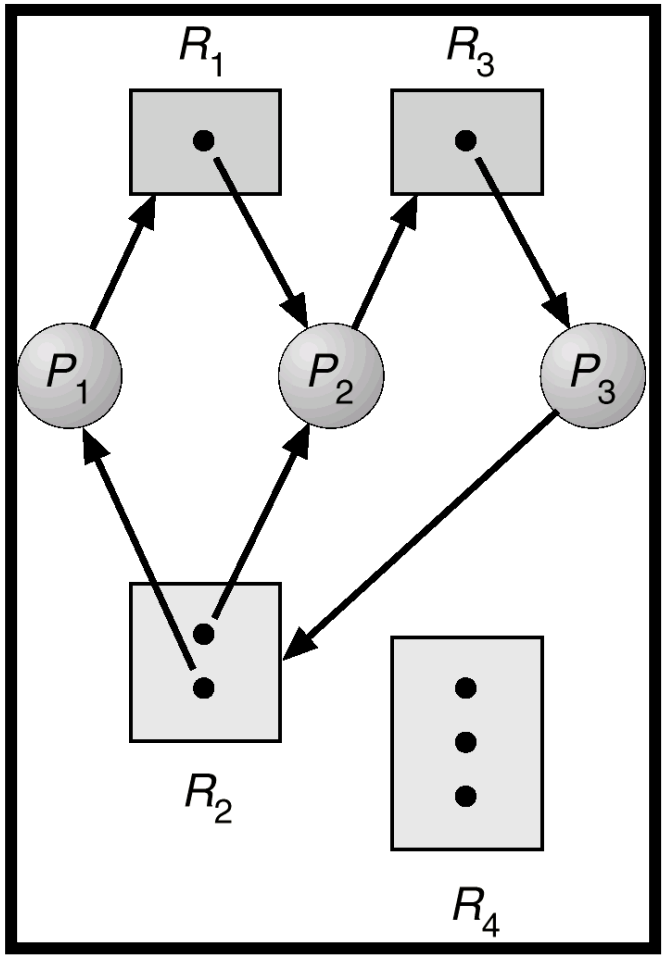
- Η P_i κατέχει μια οντότητα του πόρου R_j



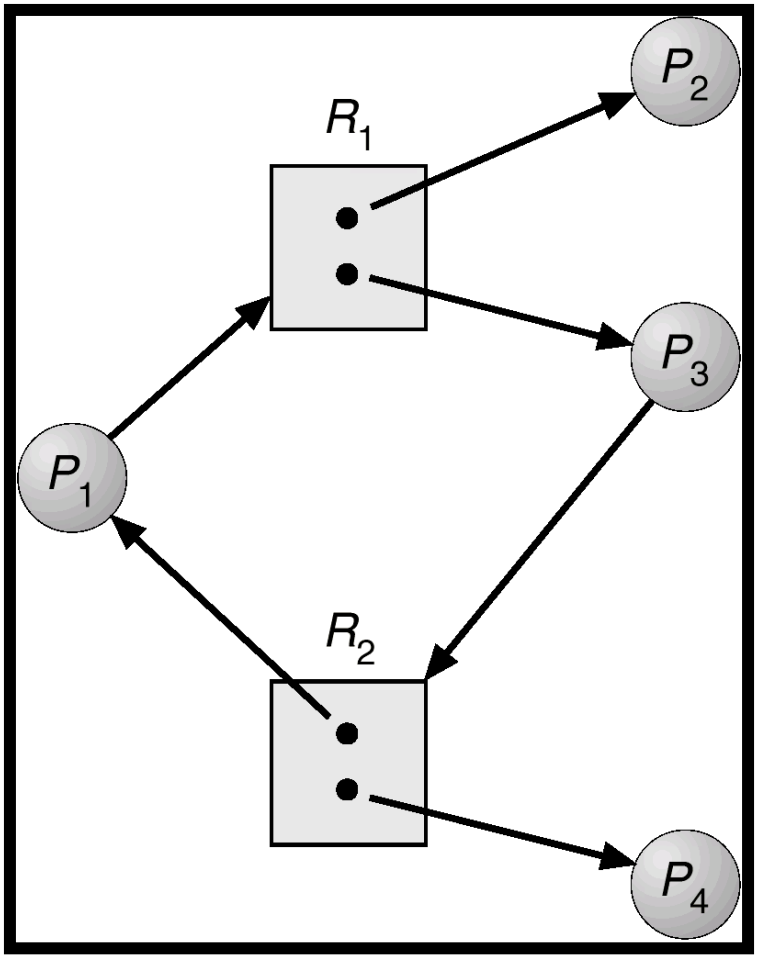
Παράδειγμα Γράφου Ανάθεσης Πόρων



Ανάθεση Πόρων με Αδιέξοδο



Ανάθεση Πόρων Χωρίς Αδιέξοδο



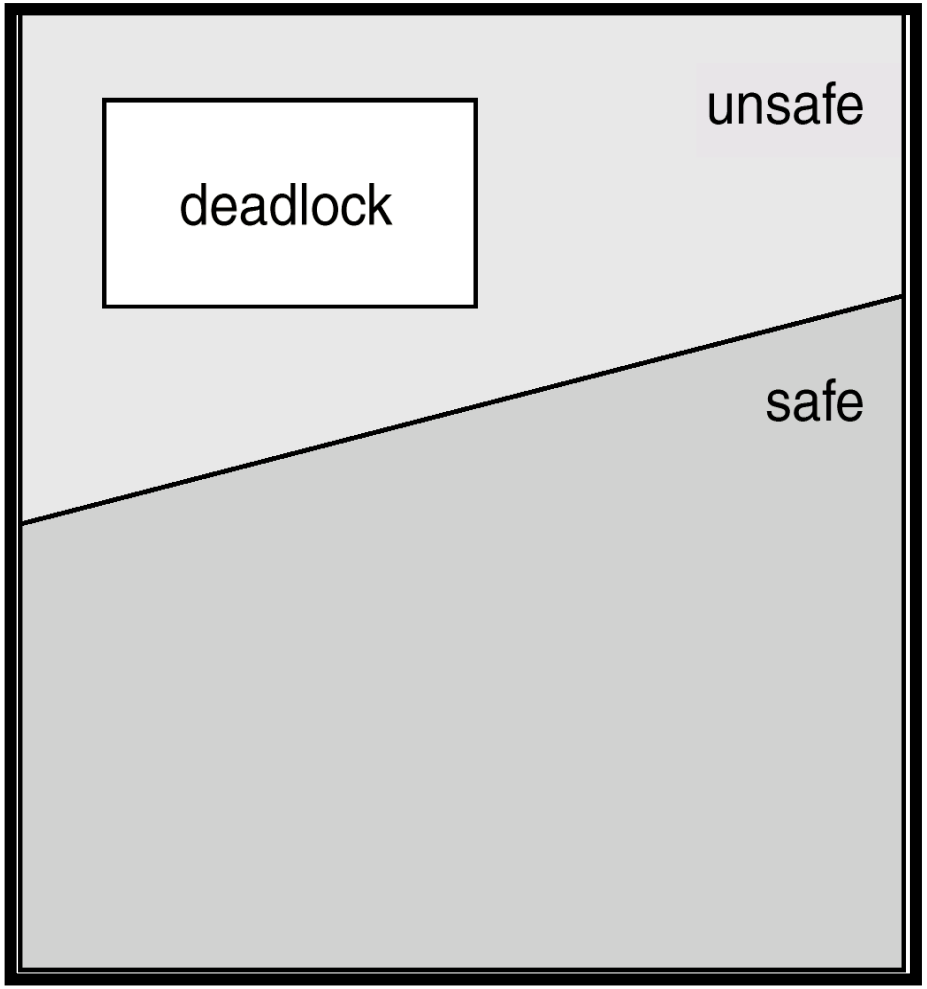
Ασφαλής Κατάσταση

- *Ασφαλής κατάσταση* => υπάρχει μια *ασφαλής ακολουθία* όλων των διεργασιών

- Ασφαλής ακολουθία <P_1, P_2, …, P_n>:

  - Για κάθε P_i, οι πόροι που η P_i μπορεί να αιτηθεί μπορούν να ικανοποιηθούν από τους διαθέσιμους πόρους και τους πόρους που κρατούν όλες οι P_j, με $j<i$

Ασφαλής, Μη ασφαλής, Αδιέξοδο



Εντελώς μεταξύ μας…

- Οι πιο συχνές λύσεις:
  - Εντατικός έλεγχος
    - + Όλοι μπορούν να το κάνουν
    - - Πόσο εντατικός; «Άπειρες» περιπτώσεις
  - ΛΣ: ο δολοφόνος με το signal…
    - Παρακολούθησε το σύστημα: αν γίνει πολύ ήσυχο ενώ δε θα έπρεπε… ξέρεις τι να κάνεις…
    - + Μπορεί να γίνει σε κάποιες εφαρμογές!
    - - Μπορεί να γίνει σε όλες τις εφαρμογές;