ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ 2

Πλάνο παρουσίασης

- Διαδικασίες
- Συντονισμός διαδικασιών

Τι είναι μια διαδικασία;

- Μία από τις κεντρικές έννοιες και αφαιρέσεις (abstractions) ενός ΛΣ
- Και όμως δεν υπάρχει αυστηρός ορισμός που να περιγράφει πλήρως την έννοια!
  - Αποτελεί ένα μοντέλο για ένα πρόγραμμα που εκτελείται
  - Το «ζωντάνεμα» ενός κώδικα
  - Μια οντότητα στην οποία προσφέρουν εξυπηρέτηση το ΛΣ και οι επεξεργαστές
  - Ένα ξεχωριστό κομμάτι δραστηριότητας που μπορεί να εναλλαχθεί με κάποιο άλλο παρόμοιο
  - ...

Τι είναι μια διαδικασία;

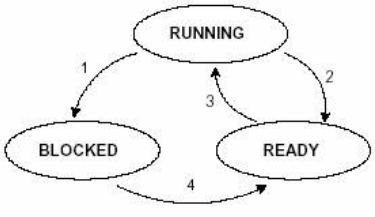
- Οι διαδικασίες εκτελούνται "παράλληλα"
  - Είτε πραγματικά
    - Το υπολογιστικό μας σύστημα διαθέτει περισσότερους επεξεργαστές ή πυρήνες
  - Είτε εικονικά
    - Όσο μια διαδικασία περιμένει από μια περιφερειακή συσκευή να εξυπηρετηθεί (π.χ. σκληρός δίσκος), το ΛΣ "δίνει" (dispatches) τη CPU σε μια άλλη "έτοιμη" (ready) διαδικασία.
    - Όταν η CPU "δίνεται" σε μια διαδικασία, αυτή συνήθως δεν κρατάει τη CPU μέχρι να τελειώσει, ακόμα και όταν η διαδικασία δεν κάνει Είσοδο/Εξοδο (Ε/Ε). Κάθε λίγο (π.χ. 100ms) η CPU δίνεται σε άλλη διαδικασία.
      - time slice ή quantum.

Πολυπρογραμματισμός

- Ανά πάσα χρονική στιγμή ένα ΛΣ μπορεί να έχει περισσότερες διαδικασίες ενεργές
  - Πολυπρογραμματισμός (Multiprogramming)
- Πως τις διαχειρίζεται το ΛΣ;
  - Κάθε διαδικασία βρίσκεται σε συγκεκριμένη κατάσταση κάθε χρονική στιγμή
  - Κάθε διαδικασία αναπαρίσταται με συγκεκριμένο τρόπο σε κάθε ΛΣ

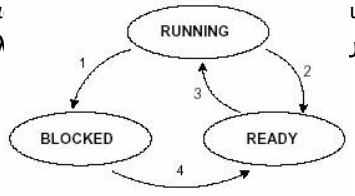
Κατάσταση διαδικασίας

- Κατάσταση μιας διαδικασίας (ως προς την εκτέλεση της)
  - **Τρέχει (Running)**
    - Έχει τη CPU
  - **Έτοιμη (Ready)**
    - Μπορεί να τρέξει, αλλά η CPU έχει δωθεί σε άλλη διαδικασία
  - **Μπλοκαρισμένη (Blocked)**
    - Περιμένει «εξωτερικό» γεγονός (όπως π.χ. δεδομένα από το δίκτυο, ανάγνωση από δίσκο κλπ).



Κατάσταση διαδικασίας

- Σημασία μεταβάσεων
  1. Περιμένει ολοκλήρωση λειτουργίας Εισόδου/Εξόδου
     - Συνήθως έχει γίνει κλήση συστήματος (system call)
  2. CPU δίνεται σε άλλη διαδικασία
     - Τελείωσε το quantum
  3. CPU δίνεται σε αυτή τη διαδικασία
     - Επιλέχθηκε άλλη δια ιστή (scheduler)
  4. Ολοκληρώθηκε η λ J



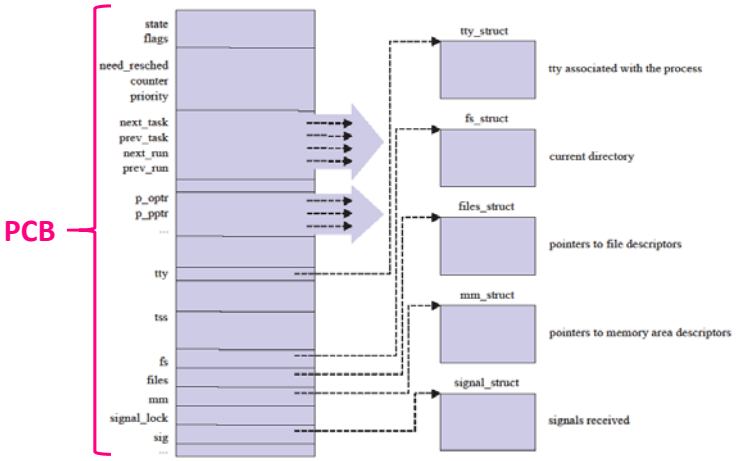
Χρονοδρομολογητής

- Σημαντικότατο τμήμα του ΛΣ
  - Αποφασίζει ποια διαδικασία θα πάρει την CPU
- Πότε όμως εκτελείται;
  - Ανά τακτά χρονικά διαστήματα
    - Διακοπή ρολογιού (clock interrupt)
    - Αντιστοιχεί συνήθως στο quantum
  - Όταν μια διαδικασία εκτελέσει μια λειτουργία Εισόδου/Εξόδου
    - Διαρκούν πολύ, αναλογικά με την ταχύτητα της CPU
- Πως αποφασίζει ποια διαδικασία θα πάρει την CPU;
  - Πληθώρα αλγορίθμων χρονοδρομολόγησης

Αναπαράσταση διαδικασίας

- Κάθε διαδικασία αναπαρίσταται με ένα Μπλοκ Ελέγχου Διαδικασίας (Process Control Block ή PCB)
  - Μετρητής προγράμματος, Δείκτης Στοίβας, καταχωρητές γενικού σκοπού, ...
  - Πίνακες σελίδων (Page Tables) και άλλες πληροφορίες που αφορούν την μνήμη
    - Θα τα δούμε στην συνέχεια πιο αναλυτικά
  - Ταυτότητα διαδικασίας (Process ID)
  - Κατάσταση (Running, Ready, Blocked, …)
  - Πληροφορίες ανοιχτών αρχείων
  - ...
- Τα PCBs όλων των διαδικασιών υπάρχουν σε έναν «Πίνακα Διαδικασιών» (Process Table)
  - Υλοποιείται σαν πίνακας ή διασυνδεδεμένη λίστα

Αναπαράσταση διαδικασίας στο Linux



Διαχείριση Διακοπών (Interrupts)

- Διακοπή (interrupt):
  - Σήμα προς τον επεξεργαστή
    - Συνήθως από άλλη συσκευή υλικού του υπολογιστή
  - Διακοπή διαδικασίας που τρέχει
    - Το ΛΣ αποκτά τον έλεγχο του επεξεργαστή για να χειριστεί την διακοπή
  - Επιτρέπουν την αλληλεπίδραση του υπολογιστή
    - Ποντίκι, πληκτρολόγιο, ...
- Για την διαχείριση απαιτείται συνεργασία υλικού και λογικού (ΛΣ)
  - Οι λεπτομέρειες είναι αρκετά στρυφνές

Διαχείριση Διακοπών (Interrupts)

- Όταν συμβεί μια Διακοπή
  - Σταματάει η εκτέλεση της διαδικασίας που έτρεχε στον επεξεργαστή
    - Πρέπει να διαφυλαχθεί η κατάσταση της διαδικασίας
  - Καθορίζεται η πηγή της διακοπής
    - Συνήθως ένας αριθμός
  - Το ΛΣ διατηρεί το Interrupt Vector
    - Μια σειρά διευθύνσεων (μια για κάθε συσκευή), που "δείχνουν" στην μέθοδο που πρέπει να κληθεί για να εξυπηρετηθεί η διακοπή της κάθε συσκευής
      - Μια μέθοδος για δίσκο, μια για τερματικό, μια για το ρολόι, κ.λ.π.
    - Αυτές οι μέθοδοι λέγονται Interrupt Service Routines (ISR).
  - Καλείται η κατάλληλη μέθοδος
  - Εκτελείται ο χρονοπρογραμματιστής για να επιλέξει την επόμενη διαδικασία που θα πάρει τον επεξεργαστή

Πλάνο παρουσίασης

- Διαδικασίες
- Συντονισμός διαδικασιών

Συντονισμός διαδικασιών

- **Σενάριο:** Είναι πιθανό 2 ή περισσότερες διαδικασίες να χρειαστεί να χρησιμοποιήσουν “ταυτόχρονα” έναν πόρο
  - Συσκευή υλικού
  - Θέση μνήμης
  - ...

Συντονισμός διαδικασιών

- Θυμηθείτε πως “ταυτόχρονα” μπορεί να σημαίνει:
  - Αν το υπολογιστικό σύστημα έχει περισσότερους επεξεργαστές/πυρήνες πραγματικά ταυτόχρονα
  - Διαφορετικά όταν το ΛΣ διακόπτει την εκτέλεση μιας διαδικασίας και δίνει τον επεξεργαστή σε άλλη διαδικασία
    - Η διακοπή μπορεί να γίνει σε μη ασφαλές σημείο

Συνθήκη ανταγωνισμού

- Συνθήκη Ανταγωνισμού
  - Περιγράφει καταστάσεις όπου 2 ή περισσότερες διαδικασίες χρησιμοποιούν τον ίδιο πόρο και το τελικό αποτέλεσμα εξαρτάται από το πότε τρέχει η κάθε διαδικασία.
- Χαρακτηριστική περίπτωση
  - Έλεγχος πρόσβασης σε κοινή μνήμη.

Ανάγκη συντονισμού: απλό παράδειγμα

Διεργασία Α	Διεργασία Β
1. tmpA:=tally;	4. tmpB:=tally;
2. tmpA:=tmpA+1;	5. tmpB:=tmpB+1;
3. tally:=tmpA;	6. tally:=tmpB;

Αρχικά tally=0

Πιθανά σενάρια εκτέλεσης και τιμή της tally μετά την εκτέλεση:

1 2 3 4 5 6 → tally=2

1 2 4 5 3 6 → tally=1

1 2 4 5 6 3 → tally=1

Διεργασίες Α και Β εκτελούνται ταυτόχρονα. tally είναι η κοινή μεταβλητή που ενημερώνεται από τις διεργασίες. Το αποτέλεσμα της εκτέλεσης (τιμή της tally) δεν είναι πάντα το ίδιο. Εξαρτάται από το πως εκτελούνται οι δύο διεργασίες

Ανάγκη συντονισμού: παράδειγμα

- Η σύζυγος και ο σύζυγος κάνουν ταυτόχρονα ανάληψη από τον ίδιο λογαριασμό από διαφορετικά ATM
  - Στο σύστημα της τράπεζας θα τρέξουν δύο διαδικασίες που θα εκτελέσουν τον ίδιο κώδικα

```
ΑΝΑΛΗΨΗ:
  read (υπόλοιπο);
  if υπόλοιπο > ποσό_ανάληψης
      write (υπόλοιπο ← υπόλοιπο –
ποσό_ανάληψης);
```

Τι μπορεί να πάει στραβά;

- **Η σύζυγος**

```
1. read (υπόλοιπο);

3. if υπόλοιπο > ποσό_ανάληψης
4. write (υπόλοιπο ← υπόλοιπο –
     ποσό_ανάληψης);
```

- **Ο σύζυγος**

```
2. read (υπόλοιπο);

5. if υπόλοιπο > ποσό_ανάληψης
6. write (υπόλοιπο ← υπόλοιπο –
     ποσό_ανάληψης);
```

- Και οι δύο διαβάζουν το αρχικό ποσό.
- Και οι δύο ενημερώσεις θα γίνουν με βάση το αρχικό ποσό.
- Η ανάληψη της συζύγου δεν καταχωρείται.

Μοντέλο αποτίμησης εκφράσεων και ρόλος τους στα ζητήματα συντονισμού

- Στα προηγούμενα παραδείγματα, η ανάγνωση και καταχώρηση στην κοινή μεταβλητή (π.χ. tally και υπόλοιπο, ο κοινός πόρος) γίνονται σε διαφορετικές γραμμές/ «φάσεις», για να γίνει πιο κατανοητό πως μπορεί η εκτέλεση να διακοπεί και τί επιπτώσεις μπορεί να επιφέρει, όταν ο κώδικας εκτελεστεί ταυτόχρονα από δύο ή παραπάνω διεργασίες.
- Σε πραγματικά περιβάλλοντα και προγράμματα, η διάκριση μεταξύ της ανάγνωσης και καταχώρησης σε μία κοινή μεταβλητή μπορεί να είναι πιο δυσδιάκριτη, παρόλο που υφίσταται και κατά συνέπεια είναι επιρρεπή σε προβλήματα συντονισμού.

Μοντέλο αποτίμησης εκφρ ς τους στα ζητήματα συντονι

- Παράδειγμα μεθόδου στη Java

```
//count: κοινή, διαμοιραζόμενη μεταβλητή
//Έστω ότι η count έχει τιμή 3 τη στιγμή
//της κλήσης της μεθόδου.

public void increaseSharedVariable() {

    count = count + 5;

}
```

Η αύξηση της κοινής διαμοιραζόμενης μεταβλητής count εκτελείται σε μία «γραμμή» της μεθόδου και δίνει την εντύπωση ότι δεν μπορεί να διακοπεί. Όμως μπορεί να διακοπεί αν δει κανείς τον τρόπο εκτέλεσης της ανάθεσης αυτής. Η εκτέλεση της γραμμής γίνεται σε δύο στάδια:

1) Υπολογισμός του δεξιού μέρους της ανάθεσης (count+5) δίνοντας αποτέλεσμα 8.
2) Ανάθεση του αποτελέσματος 8 στη μεταβλητή count.

Η εκτέλεση του βήματος 1 γίνεται και αυτό σε στάδια:

i) Μεταφορά της τρέχουσας τιμής της count (3) από τη μνήμη σε καταχωρητή (ανάγνωση τιμής)
ii) Αύξηση της τιμής του καταχωρητή προσθέτοντας 5, δίνοντας αποτέλεσμα 8, που διατηρεί ο καταχωρητής.

Αφού ολοκληρωθούν τα i) και ii) ολοκληρώνεται το βήμα 1 και εκτελείται ακολούθως το βήμα 2, όπου γίνεται η ανάθεση της νέας τιμής στην count, μεταφέροντας το αποτέλεσμα (8) από τον καταχωρητή στη θέση μνήμης που αντιστοιχεί στη μεταβλητή count. Αυτό ολοκληρώνει την εκτέλεση της γραμμής. Τα στάδια i) και ii) του βήματος 1 μπορούν να διακοπούν: μπορεί για παράδειγμα να μεταφερθεί η τρέχουσα τιμή count στον καταχωρητή (ανάγνωση), και πριν την αύξηση στο ii), να διακοπεί η εκτέλεση. Όταν συνεχίσει η εκτέλεση, θα συνεχίσει με την τιμή που φορτώθηκε στον καταχωρητή πριν διακοπεί. Κατά συνέπεια, τέτοιες εκφράσεις σε πραγματικά περιβάλλοντα μπορούν να διακοπούν και κατά συνέπεια είναι επιρρεπή σε προβλήματα που οφείλονται στην έλλειψη συγχρονισμού, παρότι μπορεί να μην είναι άμεσα εμφανή.

Παράδειγμα

- Έστω ότι δύο διαδικασίες P1, P2 εκτελούν ταυτόχρονα τον παρακάτω κώδικα με num να είναι μία κοινή, διαμοιραζόμενη μεταβλητή. Ποιες είναι όλες οι δυνατές τιμές που μπορεί να λάβει η μεταβλητή num? Θεωρήστε ότι η αρχικοποίηση της διαμοιραζόμενης μεταβλητής num γίνεται μία φορά πριν την εκτέλεση των δυο διαδικασιών.

```
//διαμοιραζόμενη μεταβλητή num
num : shared integer;
num := 4; //αρχικοποίηση


// κώδικας διαδικασίας
num := num +2
```

Δυνατές τιμές που μπορεί να λάβει η num, μετά την ταυτόχρονη εκτέλεση των δύο διαδικασιών P1, P2 είναι οι εξής:

Τιμή 8: εκτέλεση της P1 και τερματίζει (num=6) και μετά της P2 και τερματίζει - num=8 (ή P2 και μετά P1)
Τιμή 6: έναρξη εκτέλεσης της P1: διαβάζει την τρέχουσα τιμή num στον καταχωρητή (4) και διακόπτεται. Εκτέλεση της P2 η οποία διαβάζει την τρέχουσα τιμή της num (4) προσθέτει 2 και αναθέτει το αποτέλεσμα (6) στην num και τερματίζει . Ακολούθως, συνεχίζει την εκτέλεσή της η P1, η οποία στο τρέχον περιεχόμενο του καταχωρητή (4) προσθέτει 2 και αναθέτει το αποτέλεσμα (6) στη num. Τότε num=6.

Κρίσιμη περιοχή

- Κρίσιμη περιοχή (Critical Section):
  - Το τμήμα του κώδικα στο οποίο προσπελαύνουμε τον κοινό πόρο και που μπορεί να δημιουργήσει το πρόβλημα

Αμοιβαίος αποκλεισμός

- Ένας μηχανισμός που επιτρέπει την πρόσβαση στον κοινό πόρο μόνο από μια διαδικασία ανά πάσα χρονική στιγμή
  - Δηλαδή επιτρέπει μόνο σε μια διαδικασία να εκτελεί την κρίσιμη περιοχή ανά πάσα χρονική στιγμή.
  - Μπορεί να υλοποιηθεί με πολλούς τρόπους
    - Μεταβλητές κλειδώματος (Locks)
    - Ατομικές εντολές
    - ...

Χρήση «μεταβλητών κλειδώματος»

- Για κάθε κοινό πόρο ορίζουμε μια επιπλέον «παραδοσιακή» μεταβλητή που ο σκοπός της είναι να ελέγχει την πρόσβαση διαδικασιών στον πόρο αυτόν
  - Τέτοιες μεταβλητές καλούνται «μεταβλητές κλειδώματος (lock variables)»
- Η μεταβλητή κλειδώματος αναφέρει αν ο πόρος είναι απασχολημένος ή όχι.
  - Μπορεί να είναι τύπου Boolean
  - Μπορεί να ακέραια
- Κάθε διαδικασία που θέλει να χρησιμοποιήσει τον κοινό πόρο (να μπει στην κρίσιμη περιοχή) εξετάζει την μεταβλητή
  - Αν η τιμή της είναι 1, τότε περιμένει μέχρι να γίνει 0.
  - Αν η τιμή είναι 0, τότε την κάνει 1 και εισέρχετε στην κρίσιμη περιοχή

- Είναι σωστή αυτή η λύση;

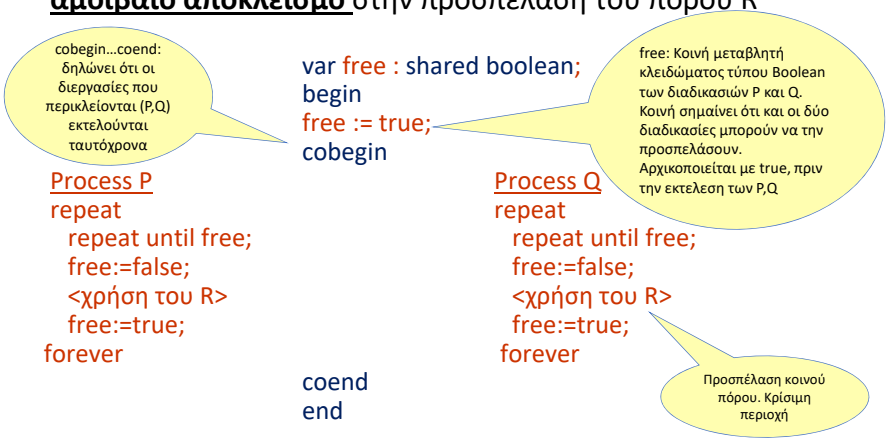
Μεταβλητές κλειδώματος

- Όχι, η λύση δεν είναι σωστή
  - Το πρόβλημα δεν λύθηκε, απλώς “μετατέθηκε” στο επίπεδο της μεταβλητής κλειδώματος

- Τι γίνεται αν 2 διαδικασίες διαβάσουν την τιμή της μεταβλητής κλειδώματος “την ίδια στιγμή”;

Μεταβλητές κλειδώματος: Παράδειγμα

Εξετάστε αν ο παρακάτω αλγόριθμος εξασφαλίζει τον **αμοιβαίο αποκλεισμό** στην προσπέλαση του πόρου R



Μεταβλητές κλειδώματος: Παράδειγμα (απάντηση)

Παρατηρούμε πως

1. Η P και η Q εκτελούν ακριβώς τον ίδιο αλγόριθμο. Χρησιμοποιούν τη κοινή λογική μεταβλητή free που αρχικά είναι αληθής.
2. Κάθε μια διεργασία εξετάζει τη free και αν τη βρει αληθή (στην εσωτερική εντολή repeat) τότε την κάνει ψευδή και χρησιμοποιεί τον πόρο R. Αν η free (και όσο αυτή) είναι ψευδής τότε η διεργασία εκτελεί το βρόγχο repeat until free.
3. Η διεργασία που ολοκληρώνει τη χρήση του πόρου R κάνει τη free ψευδή.
4. Δεν εξασφαλίζεται ο αμοιβαίος αποκλεισμός γιατί είναι δυνατόν και οι δυο διεργασίες να χρησιμοποιήσουν ταυτόχρονα τον πόρο R. Αυτό μπορεί να γίνει ως εξής:

Η P εκτελείται πρώτη και εκτελεί την εντολή repeat until free; και χάνει τη CPU. Στη συνέχεια εκτελείται η Q η οποία εκτελεί επίσης την εντολή repeat until free; και χάνει τη CPU. Στη συνέχεια εκτελείται η P η οποία εκτελεί την εντολή free:=false; και χρησιμοποιεί τον πόρο και σε αυτό το σημείο χάνει τη CPU. Τέλος η Q μπορεί να κάνει το ίδιο δηλ. τελικά και οι δυο διεργασίες χρησιμοποιούν τον πόρο ταυτόχρονα και επομένως *δεν επιτυγχάνεται* αμοιβαίος αποκλεισμός.

Μεταβλητές κλειδώματος

- Για την ορθή επίλυση του προβλήματος με μεταβλητές κλειδώματος, χρειαζόμαστε επιπλέον πληροφορία ανά διαδικασία
  - Λύση Dekker για δύο διαδικασίες
  - Λύση Peterson για δύο διαδικασίες
  - Λύση Dijkstra για περισσότερες διαδικασίες

Ατομικές εντολές

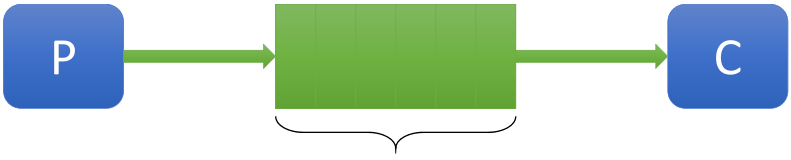
- Καλύτερη λύση είναι να προσφέρει το υλικό υποστήριξη
- Πρόβλημα στο παράδειγμα με την ανάληψη
  - Διάβασμα τιμής, αλλαγή και αποθήκευση νέας τιμής γίνονται σε ανεξάρτητα βήματα
  - Οποιαδήποτε άλλη διαδικασία μπορεί να παρεμβληθεί
- Οι ατομικές εντολές δεν επιτρέπουν την παρεμβολή άλλης διαδικασίας
  - Δεν μπορεί να διακοπεί η εκτέλεσή τους.

Ατομικές εντολές

- Οι ατομικές εντολές υλοποιούν συνήθως απλές πράξεις
  - Όπως για παράδειγμα πρόσθεση/αφαίρεση μιας τιμής
- Για πιο σύνθετες πράξεις
  - Υλοποιούμε μεταβλητές κλειδώματος με χρήση ατομικών εντολών
  - Χρησιμοποιούμε τις μεταβλητές κλειδώματος για αμοιβαίο αποκλεισμό

Συγχρονισμός
Το πρόβλημα του Παραγωγού-Καταναλωτή

- Υπάρχουν 2 διαδικασίες
  - Ο παραγωγός P (Producer) παράγει πληροφορία και την αποθηκεύει σε έναν απομονωτή (buffer)
  - Ο καταναλωτής C (Consumer) προσπελαύνει τον απομονωτή καταναλώνοντας την πληροφορία.
  - Ο απομονωτής δουλεύει ως ουρά (FIFO)



N στοιχεία

Συγχρονισμός
Το πρόβλημα του Παραγωγού-Καταναλωτή

- Προβλήματα
  - Ο απομονωτής αποτελεί κοινή μνήμη μεταξύ P και C.
    - Πρέπει να αποφευχθούν συνθήκες ανταγωνισμού!
- Τι γίνεται όταν ο απομονωτής είναι γεμάτος;
  - Που θα βάλει τα δεδομένα ο P;
- Τι γίνεται όταν ο απομονωτής είναι άδειος;
  - Πως θα πάρει δεδομένα ο C;
- Πέρα από τον αμοιβαίο αποκλεισμό της κοινής μνήμης **απαιτείται επιπλέον και συγχρονισμός (synchronization)**

Συγχρονισμός
Το πρόβλημα του Παραγωγού-Καταναλωτή

- Πρώτη προσπάθεια επίλυσης:
  - Όταν ο απομονωτής είναι γεμάτος ο P “κοιμάται”
    - Τον ξυπνάει ο C μόλις καταναλώσει κάποιο στοιχείο
  - Όταν ο απομονωτής είναι άδειος ο C “κοιμάται”
    - Τον ξυπνάει ο P μόλις παράξει κάποιο στοιχείο
  - Γνωρίζουμε το πλήθος των στοιχείων που μπορεί να αποθηκεύσει ο απομονωτής (έστω N)
  - Ορίζουμε μια μεταβλητή “count” που δείχνει το τρέχων πλήθος στοιχείων στον απομονωτή (πόσες θέσεις είναι γεμάτες)
    - Αρχικά count = 0

Πρώτη προσπ

Υπάρχει πιθανότητα ταυτόχρονης ενημέρωσης της μεταβλητής count →
Πιθανόν λάθος αποτέλεσμα!
Έστω η εξής κατάσταση: N=10, count =3 (ούτε ο παραγωγός, ούτε ο καταναλωτής κάνουν sleep())
1) Παραγωγός: εκτελεί put_item() και διακόπτεται (τότε buffer έχει 4 στοιχεία, count=3)
2) Καταναλωτής: εκτεκελεί remove_item() (τότε το buffer έχει 3, count =3) και διακόπτεται
3) Παραγωγός: πάει να εκτλέσει το count = count + 1. Εκτελεί το δεξί τμήμα count+1 της ανάθεσης – που έχει αποτέλεσμα 4 - αλλά δεν το καταχωρεί στην count γιατί διακόπτεται.
4) Καταναλωτής: Εκτελεί το count=count-1. Η count τώρα έχει τιμή 2.
5) Παραγωγός: Συνεχίζει το βήμα 3). Αναθέτει το αποτέλεσμα 4 στη μεταβλητή count.
Συνεπώς η count θα έχει τιμή 4 ενώ στο buffer θα υπάρχουν 3 στοιχεία.
Η λύση παρουσιάζει και άλλα προβλήματα: τί γίνεται όταν count=0 και πριν την εκτέλεση του sleep() από τον καταναλωτή, ο παραγωγός εισάγει νέο στοιχείο;

Παραγωγός **τής**

```
while (1) {                          ) {
  produce                            nt == 0)
  if (count                          leep();
        sleep(                       ove_item();
  put_item();                        count = count - 1;
  count  = count +                   if (count == N - 1)
  if (count == 1)                          wakeup(producer);
        wakeup(consumer);            consume_item();
}                                    }
```


Σημαφόροι (1)

- Τύπος μεταβλητής
  - Όπως και οι μεταβλητές κλειδώματος
  - Διατηρεί ακέραια τιμή >= 0
- Αρχικοποιείται με οποιαδήποτε ακέραια τιμή
  - Αν αρχικοποιηθεί με τιμή = 1
    - Δυαδική σημαφόρος
  - Αν αρχικοποιηθεί με τιμή > 1
    - Σημαφόρος μετρητής

Σημαφόροι (2)

Η βασική ιδέα της χρήσης τους:

Δυο ή περισσότερες διεργασίες μπορούν να συγχρονιστούν με τη χρήση (ανταλλαγή) απλών σημάτων: μια διεργασία σταματά σε μια συγκεκριμένη θέση και ξεκινά μόνο όταν λάβει ένα σήμα.

Ένας σημαφόρος είναι μια ειδική μεταβλητή που ανήκει και ελέγχεται από το ΛΣ και δημιουργεί τα σήματα συγχρονισμού.

Τα «εργαλεία» με τα οποία μπορούμε να τους χρησιμοποιήσουμε :
η διαχείριση ενός σημαφόρου s γίνεται μέσω των εντολών (λειτουργιών) wait(s) ή P(s) και signal(s) ή V(s):

- **wait(s) ή P(s) ή down(s)**: παραλαμβάνεται ένα σήμα (από τη διεργασία που εκτελεί τη wait) μέσω του σημαφόρου s
- **signal(s) ή V(s) ή up(s)**: η διεργασία που εκτελεί το signal στέλνει ένα σήμα μέσω του σημαφόρου s

Σημαφόροι (3)

- Σε μια σημαφόρο “s” μπορούν να γίνουν οι ακόλουθες πράξεις επί αυτής
  - **Πράξη DOWN(s) [ή WAIT(s) ή P(s)]**
    - Εκτέλεσε **ατομικά**

```
if (s == 0)
   sleep();
else
   s = s – 1;
```

  - **Πράξη UP(s) [ή SIGNAL(s) ή V(s)]**
    - Εκτέλεσε **ατομικά**

```
s = s + 1;
if (s == 1)
   wakeup(a waiting process);
```

Σημαφόροι (4)

Οι σημαφόροι ως δυνατότητα και οι εντολές διαχείρισής τους (δηλ. οι wait και signal) προσφέρονται από το σύνολο των σύγχρονων λειτουργικών συστημάτων.

Οι πράξεις **wait/P και signal/V <u>εκτελούνται ατομικά</u>**

Η εκτέλεσή τους δεν μπορεί να «διακοπεί». Εμφανίζονται στο υπόλοιπο σύστημα ώς μια ενιαία αδιάκοπη πράξη. Αν εκτελείται η πράξη wait ή signal σε μία σημαφόρο S, καμία άλλη διεργασία δεν έχει πρόσβαση στη σημαφόρο S μέχρις ότου τερματίσει η πράξη.

Οι λειτουργίες wait και signal μπορούν να ενσωματωθούν στον κώδικα των διεργασιών που πρέπει να συγχρονίσουν.

Κάθε σημαφόρος πρέπει να έχει μια αρχική τιμή.

Το ΛΣ έχει την ευθύνη της εκτέλεσης των σημαφόρων.

Τα προγράμματα των χρηστών έχουν την ευθύνη της σωστής χρήσης των λειτουργιών των σημαφόρων και της υλοποίησης των σωστών κανόνων συγχρονισμού ανάλογα με τις κάθε φορά απαιτήσεις.

Σημαφόροι (5)

Παράδειγμα χρήσης σημαφόρων: αμοιβαίος αποκλεισμός δύο διαδικασιών P και Q

```
var s : semaphore;
begin
s :=1;
cobegin
```

s: δυαδική σημαφόρος που αρχικοποιείται με την τιμή 1

```
Process P
repeat
  wait (s) [or P(s)]
  <χρήση του R>
  signal (s); [or V(s)]
forever
```

```
Process Q
repeat
  wait (s);
  <χρήση του R>
  signal (s);
forever
```

```
coend
end
```

```
if (s==0)
   sleep();
else
   s = s - 1
```

```
s= s + 1;
if ( s == 1)
wakeup(sleeping)
```

Εξασφαλίζεται αμοιβαίος αποκλεισμός των διαδικασιών P και Q.
Ο πόρος R χρησιμοποιείται το πολύ από μία διαδικασία

Δεύτερη προσπάθεια επίλυσης παραγωγού-καταναλωτή: σημαφόροι

- Χρησιμοποιούμε δύο σημαφόρους
  - emptyCount για να μετράμε πόσες άδειες θέσεις έχει ο απομονωτής
    - Αρχικά emptyCount = N
  - fillCount για να μετράμε πόσες θέσεις είναι γεμάτες στον απομονωτή
    - Αρχικά fillCount = 0

Δεύτερη προσπάθεια επίλυσης με τη χρήση σημαφόρων

Παραγωγός

```
while (1) {
  produce_item();
  DOWN(emptyCount);
  put_item();
  UP(fillCount);
}
```

Καταναλωτής

```
while (1) {
  DOWN(fillCount);
  remove_item();
  UP(emptyCount);
  consume_item();
}
```

Η παραπάνω λύση δουλεύει σωστά όταν έχουμε 1 παραγωγό και 1 καταναλωτή

Δεύτερη προσπάθεια επίλυσης

- Φανταστείτε πως έχουμε 2 παραγωγούς
  - Καθένας εκτελεί το DOWN(emptyCount)
  - Πρώτος παραγωγός τοποθετεί το στοιχείο που έχει παραγάγει στην θέση “k” στον απομονωτή
    - Δεύτερος παραγωγός πρέπει να πάει στην θέση “k+1”
    - Πρέπει να υπάρχει μια μεταβλητή που το υποδηλώνει
      - Η οποία ενημερώνεται προφανώς στην put_item()
    - Αν την προσπελαύνουν ταυτόχρονα έχουμε πάλι κίνδυνο για λάθος αποτελέσματα
    - Πρέπει να προστατεύσουμε την put_item() ώστε πάντα ένας μόνο παραγωγός να την εκτελεί
      - Εισάγουμε άλλη μια σημαφόρο mutex = 1

Τρίτη προσπάθεια επίλυσης σε περίπτωση πολλαπλών παραγωγών/καταναλωτών

Παραγωγός

```
while (1) {
  produce_item();
  DOWN(emptyCount);
  DOWN(mutex)
  put_item();
  UP(mutex);
  UP(fillCount);
}
```

Καταναλωτής

```
while (1) {
  DOWN(fillCount);
  DOWN(mutex);
  remove_item();
  UP(mutex);
  UP(emptyCount);
  consume_item();
}
```

Τρίτη προσπάθεια επίλυσης

- Αυτή η λύση δουλεύει σωστά!

- Τι θα συνέβαινε αν:
  - Ο παραγωγός άλλαζε τη σειρά των 2 DOWN;
  - Ο καταναλωτής άλλαζε τη σειρά των 2 DOWN;

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- Σημαφόροι χρήσιμες σε γενικότερα προβλήματα συγχρονισμού
  - Όχι μόνο για την επίτευξη αμοιβαίου αποκλεισμού
  - Παράδειγμα: δύο διεργασίες οι οποίες πρέπει να εκτελεστούν με συγκεκριμένη σειρά ώστε να υπολογιστεί/εμφανιστεί ένα συγκεκριμένο αποτέλεσμα

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- Παράδειγμα
  - Υποθέστε πως έχουμε δυο παράλληλες διεργασίες που η μια (P) γράφει στην οθόνη τη λέξη "ΤΙΚ" και η δεύτερη (Q) τη λέξη "ΤΑΚ" όπως φαίνεται παρακάτω. Πως πρέπει να αρχικοποιηθούν οι σημαφόροι s1, s2 και πως πρέπει να τοποθετηθούν οι πράξεις wait()/P() και signal()/V(), ώστε η ταυτόχρονη εκτέλεση των P και Q να εμφανίσει πάντα: "ΤΙΚ" "ΤΑΚ" "ΤΙΚ" "ΤΑΚ"… ?

```
                        var s1, s2 : semaphore;
                        begin

                        cobegin
Process P                                   Process Q
repeat                                      repeat

  print "ΤΙΚ";                                print "ΤΑΚ";

forever                                     forever
                        coend
                        end
```


Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- Το πρόβλημα αυτό δεν είναι πρόβλημα αμοιβαίου αποκλεισμού
  - Είναι πρόβλημα γενικότερου συγχρονισμού των διαδικασιών P και Q
  - Πρέπει να εξασφαλιστεί ότι θα τρέξει πρώτα η P (εμφανίζοντας "ΤΙΚ") και μετά η Q (εμφανίζοντας "ΤΑΚ")

```
                        var s1, s2 : semaphore;
                        begin
                        s1 = 1, s2 = 0
                        cobegin
Process P                                   Process Q
repeat                                      repeat
  wait(s1);                                   wait(s2);
  print "ΤΙΚ";                                print "ΤΑΚ";
  signal(s2);                                 signal(s1);
forever                                     forever
                        coend
                        end
```

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- Σύντομη εξήγηση:
  - Σημαφόροι αρχικοποιούνται: s1=1, s2=0
  - Επειδή αρχικά s2=0, η διαδικασία Q δεν θα εκτελεστεί πρώτη (βλ. wait(s2)). Η Q αναμένει.
  - Επειδή αρχικά s1=1, η διαδικασία P θα εκτελεστεί πρώτη (βλ. wait(s1) που θα κάνει s1=0). Μόλις τυπωθεί "ΤΙΚ" η P εκτελεί signal(s2). Ακολούθως, η P αναμένει στο wait(s1) (αφού τώρα s1=0) λόγω της επανάληψης.
  - Η εκτέλεση signal(s2) από την P, ξυπνάει την Q που εμφανίζει "ΤΑΚ" και εκτελεί το signal(s1) που θέτει s1=1.
  - Το signal(s1) από την Q ξυπνάει την P (wait(s1) αφού η s1 έλαβε τιμή 1 από την Q).
  - Η διαδικασία συνεχίζεται με την ίδια σειρά...

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

Ένα υποκατάστημα τράπεζας που λειτουργεί σε μία περιοχή της Ελλάδας έχει Ν πελάτες και έναν αριθμό υπαλλήλων. Το υποκατάστημα της τράπεζας διαθέτει και μία Αυτόματη Ταμειακή Μηχανή (ΑΤΜ) απ'όπου οι πελάτες της τράπεζας μπορούν να κάνουν αναλήψεις με μία ειδική κάρτα που δίνεται σε όλους τους πελάτες. Το ΑΤΜ του συγκεκριμένου υποκαταστήματος επιτρέπει μόνο ανάληψη χρημάτων και επιτρέπει τη χρήση του μόνο από τους πελάτες του υποκαταστήματος. Το ΑΤΜ γεμίζει με συγκεκριμένο μέγιστο ποσό χρημάτων Μ. Όταν το ΑΤΜ αδειάσει από τις αναλήψεις πελατών, ένας συγκεκριμένος υπάλληλος του υποκαταστήματος δέχεται μία ηλεκτρονική ενημέρωση ώστε να ξαναγεμίσει το ΑΤΜ με το μέγιστο ποσό χρημάτων Μ. Κατά το διάστημα που ο υπάλληλος γεμίζει το ΑΤΜ με χρήματα, το ΑΤΜ εμφανίζει ένα μήνυμα στην οθόνη και τίθεται εκτός λειτουργίας για τους πελάτες, δηλαδή δεν τους επιτρέπει να κάνουν αναλήψεις. Αφού ο υπάλληλος γεμίσει με χρήματα το ΑΤΜ, το κάνει και πάλι διαθέσιμο προς χρήση από τους πελάτες.

Επιπλέον θεωρήστε τα εξής: 1) αρχικά το ΑΤΜ είναι γεμάτο με χρήματα 2) το μέγιστο ποσό Μ με το οποίο γεμίζει το ΑΤΜ είναι πάντα ακέραιο πολλαπλάσιο του μέγιστου επιτρεπόμενου ημερησίου ποσού ανάληψης από τους πελάτες και 3) κάθε πελάτης, όταν κάνει μία ανάληψη, αιτείται το μέγιστο επιτρεπόμενο ημερήσιο ποσό.

Είναι το παραπάνω πρόβλημα, πρόβλημα συντονισμού/συγχρονισμού διαδικασιών; Αν ναι, ποιος είναι ο κοινός πόρος και ποιες διαδικασίες υπάρχουν;

Συγγράψτε κώδικα για τους πελάτες και τον υπάλληλο του υποκαταστήματος με σκοπό την επίλυση του παραπάνω προβλήματος. Η λύση θα πρέπει να βασίζεται στη χρήση σημαφόρων.

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- ΑΠΑΝΤΗΣΗ:
  - Το πρόβλημα του υποκαταστήματος ανάγεται στο πρόβλημα παραγωγού-καταναλωτή. Οι διαδικασίες και ο πόρος/απομονωτής είναι τα εξής:
    - **Παραγωγός:** ο υπάλληλος (εκτελεί τις εξής πράξεις: αναμένει μήνυμα ότι το ΑΤΜ έχει αδειάσει, όταν λάβει το μήνυμα απενεργοποιεί το ΑΤΜ και το γεμίζει, το κάνει διαθέσιμο στους πελάτες μόλις ολοκληροθεί η διαδικασία)
    - **Καταναλωτής:** Ν πελάτες τράπεζας (επισκέπτεται το ΑΤΜ και αναμένει αν είναι εκτός λειτουργίας, αν μετά την ανάληψη το ΑΤΜ αδειάσει το ΑΤΜ τίθεται εκτός λειτουργίας και ενημερώνεται ο υπάλληλος, αν δεν αδειάσει τότε το ΑΤΜ γίνεται διαθέσιμο για τον επόμενο πελάτη
    - **Πόρος/Απομονωτής:** ΑΤΜ και χρήματα εντός αυτού

Σημαφόροι σε γενικά προβλήματα συγχρονισμού

- ΑΠΑΝΤΗΣΗ: Κώδικας υπαλλήλου και πελατών για την επίλυση του προβλήματος.

Μεταβλητές: empty: semaphore, machine: semaphore, amount: integer

Αρχικοποίηση μεταβλητών:

```
empty := 0 /* ΑΤΜ όχι άδειο */
machine := 1 /* ΑΤΜ διαθέσιμο */
amount := M /*Ποσό στο ΑΤΜ */
```

Κώδικας υπάλληλου:

```
while (true) do {
   wait(empty); /*Αναμονή να αδειάσει το ΑΤΜ*/
   print("Please wait");
   amount := M;
   signal(machine); /* ΑΤΜ διαθέσιμο στους πελάτες*/
}
```

Κώδικας Πελάτη (κάθ' έναν απ'αυτούς):

```
while (true) do {
   wait(machine); /*Αναμονή διαθεσιμότητας ΑΤΜ*/
   amount := amount – withdraw;
   if (amount ==0)
      signal(empty); /*ενημέρωσε υπάλληλο*/
   else
      signal(machine); /* ΑΤΜ διαθέσιμο*/
}
```

Αδιέξοδα

- Αδιέξοδο
  - Μία κατάσταση κατά την οποία δύο ή παραπάνω διεργασίες που εκτελούνται ταυτόχρονα περιμένει ή μία την περάτωση της άλλης δίχως όμως καμία να περατώνεται.
  - Ένα σύνολο από διαδικασίες που δημιουργούν μια κυκλική αλυσίδα
    - Κάθε διαδικασία στην αλυσίδα δεν μπορεί να προχωρήσει και περιμένει για κάποιο γεγονός που μπορεί να προκληθεί μόνο από κάποιο άλλο μέλος της αλυσίδας.
- Τα γεγονότα για τα οποία περιμένουν οι διαδικασίες είναι η απελευθέρωση κάποιου πόρου.

Αδιέξοδα

- Για να χρησιμοποιήσουν κάποιο πόρο οι διαδικασίες πρέπει να ακολουθήσουν το εξής πρωτόκολλο.
  - Να ζητήσουν τους πόρους
    - Αν δεν είναι διαθέσιμοι (δηλ. κάποια άλλη διαδικασία τους χρησιμοποιεί) τότε η διαδικασία που το ζητάει μπλοκάρει.
  - Να κλειδώσουν και να χρησιμοποιήσουν τους πόρους, αν είναι ελεύθεροι
  - Να απελευθερώσουν τους πόρους όταν τελειώσουν.

Αδιέξοδο: Παράδειγμα

Εξετάστε αν ο παρακάτω αλγόριθμος παρουσιάζει αδιέξοδο

```
var Pturn, Qturn : shared boolean;
begin
Pturn :=false;  Qturn :=false;

cobegin
Process P                          Process Q
repeat                             repeat
   Pturn :=true;                   Qturn := true;
   repeat until (not Qturn);          repeat until (not Pturn);
   <χρήση του R>                      <χρήση του R>
   Pturn :=false;                     Qturn := false;
forever                            forever
coend
end
```


Αδιέξοδο: απάντηση

Ο αλγόριθμος χρησιμοποιεί δυο κοινές μεταβλητές P_{turn}, Q_{turn} που η κάθε μια υποδηλώνει τη σειρά εκτέλεσης της αντίστοιχης διαδικασίας, όταν η κάθε μια τους είναι αληθής. Όταν η P_{turn} είναι αληθής δεν μπορεί να χρησιμοποιήσει τον πόρο η Q (εξαιτίας του εσωτερικού repeat). Για το ίδιο λόγο, όταν η Q_{turn} είναι αληθής δεν μπορεί να χρησιμοποιήσει τον πόρο η P.

Τι γίνεται όμως όταν προσπαθήσουν και οι δυο διαδικασίες να χρησιμοποιήσουν τον πόρο; Υποθέστε πως η P εκτελεί την εντολή P_{turn} :=true και χάνει τη CPU. Στη συνέχεια η Q εκτελεί την εντολή Q_{turn} := true. Από εκεί και πέρα όποια από τις δυο συνεχίσει θα εκτελεί επ' αόριστο την εσωτερική repeat. Δηλ. και οι δυο διεργασίες ουσιαστικά θα «κολλήσουν» χωρίς να υπάρχει η δυνατότητα να συνεχίσουν την εκτέλεση της επόμενης εντολής.

Το φαινόμενο αυτό όπως αναφέρθηκε στη διαφάνεια 46 της παρουσίασης ονομάζεται αδιέξοδο και η εμφάνισή του δείχνει πως ο αλγόριθμος δεν εξασφαλίζει αμοιβαίο αποκλεισμό.

Αδιέξοδα με σημαφόρους: παράδειγμα

- Πως πρέπει να αρχικοποιηθεί η σημαφόρος s στον παρακάτω κώδικα για να προκύψει αδιέξοδο;

```
var s : semaphore;
begin
s = ???
cobegin
```

```
Process P
repeat
  wait (s)
  <χρήση του R>
  signal (s);
forever
```

```
Process Q
repeat
  wait (s);
  <χρήση του R>
  signal (s);
forever
```

```
coend
end
```


Αδιέξοδα με σημαφόρους: παράδειγμα

```
var s : semaphore;
begin
s = 0;
cobegin
```

Με τιμή αρχικοποίησης 0 της σημαφόρου s, και οι δύο διαδικασίες θα «κολλήσουν» στο wait(s)

```
Process P
repeat
  wait (s)
  <χρήση του R>
  signal (s);
forever
```

```
Process Q
repeat
  wait (s);
  <χρήση του R>
  signal (s);
forever
```

```
coend
end
```

Αναγκαίες Συνθήκες για εμφάνιση Αδιεξόδου

- Αμοιβαίος αποκλεισμός
  - Μόνο μια διαδικασία μπορεί να χρησιμοποιεί έναν πόρο
- Κατοχή και αναμονή (hold & wait)
  - Οι διαδικασίες που συμμετέχουν στο αδιέξοδο πρέπει και να κατέχουν κάποιο πόρο αλλά και να περιμένουν για κάποιο πόρο
- Μη προεκτοπισμός (No preemption)
  - Μόνο η κατέχουσα διαδικασία μπορεί να απελευθερώσει τον πόρο - δηλαδή ο πόρος δεν μπορεί να αφαιρεθεί από τη διαδικασία "δια της βίας"
- Κυκλική αναμονή (circular wait)
  - Ν διεργασίες, όπου κάθε διεργασία περιμένει για έναν πόρο που τον κατέχει η επόμενη διεργασία στην αλυσίδα

Μοντελοποίηση αδιεξόδων

- Δημιουργείται ένας κατευθυνόμενος γράφος.
- Κόμβοι του γράφου είναι οι διεργασίες και οι πόροι.
- Η ακμή P ➔ R σημαίνει ότι η διεργασία P περιμένει για τον πόρο R.
- Η ακμή R ➔ P σημαίνει ότι η διεργασία P κατέχει τον πόρο R.
- Στο σύστημα υπάρχει deadlock **εάν και μόνο εάν** ο κατευθυνόμενος γράφος περιέχει ένα **κύκλο**!

Λύσεις

- Υπάρχουν 4 στρατηγικές για την αντιμετώπιση του προβλήματος
  - Στρουθοκαμηλισμός
    - Μην κάνεις τίποτα
- Αποφυγή αδιεξόδου
  - Προσέχεις πότε δίνονται οι πόροι στις διαδικασίες
- Ανίχνευση και ανάνηψη
  - Χρήση του γράφου
- Πρόληψη
  - Σιγουρεύει ότι μια από τις 4 αναγκαίες συνθήκες δεν μπορεί να ισχύσει

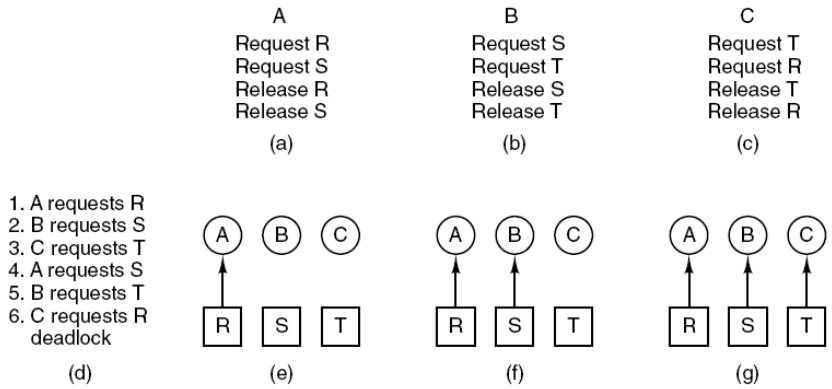
Στρουθοκαμηλισμός

- Η αντιμετώπιση του προβλήματος των αδιεξόδων κοστίζει ακριβά
  - Πολλά ΛΣ επιλέγουν να μην αντιμετωπίζουν καθόλου το πρόβλημα
    - Ούτε καν ανιχνεύουν την εμφάνιση αδιεξόδου
- Καθιστούν υπεύθυνες τις εφαρμογές για τη λύση του προβλήματος.

Πρόληψη αδιεξόδων

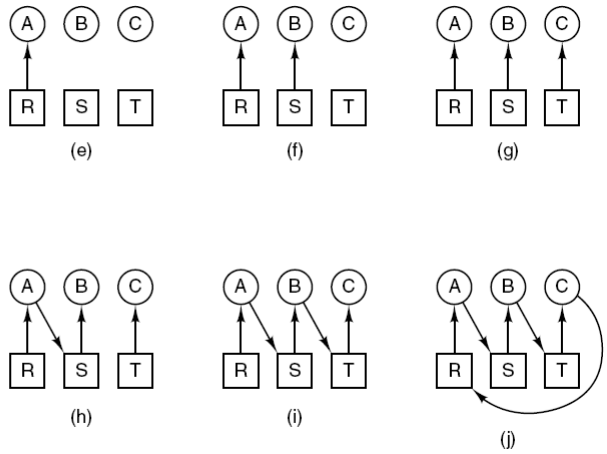
- Αν δεν ισχύει μια από τις 4 συνθήκες που προκαλούν αδιέξοδο, τότε δεν μπορεί να εμφανιστεί αδιέξοδο
- Συνήθως επιλέγεται η συνθήκη της κυκλικής αναμονής
  - Σε όλους τους πόρους δίνεται ένας αύξοντας αριθμός
  - Όλες οι διαδικασίες είναι υποχρεωμένες να ζητούν πόρους με την σειρά που αντιστοιχεί στους αριθμούς τους
    - Δεν μπορεί να προκύψει κυκλική αλυσίδα

Παράδειγμα (1/2)



Παράδειγμα (2/2)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock



Επίλυση αδιεξόδου

- Αν ορίσουμε ότι οι πόροι δεσμεύονται με την σειρά R, S, T
  - Τότε η διαδικασία C θα έπρεπε να δεσμεύσει πρώτα τον πόρο R και μετά τον πόρο T
  - Δεν προκύπτει αδιέξοδο

Υλικό βασισμένο στην
ΠΛΣ60, ΕΑΠ