

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Υλικό από:

Tanenbaum, *Modern Operating Systems, Structured Computer Organization*

Stallings, *Operating Systems: Internals and Design Principles.*

Silberschatz, Galvin and Gange, *Operating Systems Concepts.*

Deitel, Deitel and Choffnes, *Operating Systems*

Λειτουργικά Συστήματα, Γ.Α. Παπαδόπουλος, Πανεπιστήμιο Κύπρου

Λειτουργικά Συστήματα, Κ. Διαμαντάρας, ΤΕΙΘ

Systems Programming in C, A.D. Marshal, University of Cardiff

Σύνθεση

Κ.Γ. Μαργαρίτης, Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

Πόροι (1)

Μονάδες υλικού, συσκευές, δομές δεδομένων που χρησιμοποιούν οι διεργασίες για να εκτελεστούν.

Παραδείγματα πόρων:

- εκτυπωτές
- δίσκοι
- διαμοιραζόμενα αρχεία
- διαμοιραζόμενες μεταβλητές

Οι διεργασίες πρέπει να έχουν πρόσβαση στους απαιτούμενους πόρους σε τακτά χρονικά διαστήματα και με συγκεκριμένη σειρά.

Αν μια διεργασία κρατά τον πόρο A και ζητά τον πόρο B και ταυτόχρονα μια άλλη διεργασία κρατά τον πόρο B και ζητά τον πόρο A έχουμε αδιέξοδο.

Πόροι (2)

Τα αδιέξοδα συμβαίνουν όταν διεργασίες αποκτούν αποκλειστική πρόσβαση σε κάποιο πόρο.

Σε ένα σύστημα πολυπρογραμματισμού η συνολική απαίτηση πόρων από όλες τις συνεξελισσόμενες ενεργές διεργασίες υπερβαίνει κατά πολύ το συνολικό ποσό των διαθέσιμων πόρων.

Προεκτοπίσιμοι (Preemptable) πόροι

το λ.σ. μπορεί να αφαιρέσει το πόρο από τη διεργασία χωρίς προβλήματα (πχ ένα τμήμα μνήμης).

Μη-προεκτοπίσιμοι (Nonpreemptable) πόροι

η αφαίρεσή τους θα προκαλέσει αστοχία ή κατάρρευση (πχ ο εκτυπωτής ή η συσκευή CD).

Πόροι (3)

Επαναχρησιμοποιήσιμοι (Reusable) πόροι:

Χρησιμοποιούνται με ασφάλεια από μια διεργασία σε κάθε χρονική στιγμή και δεν εξαντλούνται από αυτή τη χρήση.

Οι διεργασίες αποκτούν πόρους που θα απελευθερώσουν στη συνέχεια ώστε να χρησιμοποιηθούν από άλλες διεργασίες.

Τέτοιοι πόροι είναι : επεξεργαστές, I/O κανάλια, κύρια και δευτερεύουσα μνήμη, αρχεία, βάσεις δεδομένων, και σηματοφόροι.

Το αδιέξοδο προκύπτει όταν μια διεργασία δεσμεύει ένα πόρο και απαιτεί έναν άλλο.

Η αιτία αδιεξόδων αυτού του είδους είναι συχνά η πολύπλοκη λογική των προγραμμάτων.

Μια στρατηγική αντιμετώπισης είναι η επιβολή περιορισμών που να αφορούν τη σειρά με την οποία ζητούνται οι πόροι.

Πόροι (4)

Καταναλώσιμοι (Consumable) πόροι:

Δημιουργούνται (παράγονται) και καταστρέφονται (καταναλώνονται) από μια διεργασία

Όταν ένας πόρος δεσμεύεται από μια διεργασία παύει να υπάρχει.

Τέτοιοι πόροι : Διακοπές (Interrupts), σήματα (signals), μηνύματα και πληροφορίες σε ενταμιευτές (buffers) I/O

Το αδιέξοδο μπορεί να συμβεί αν ένα μήνυμα που αποστέλλεται από μια διεργασία δεν παραλαμβάνεται από την άλλη

Σπάνιος συνδυασμός γεγονότων μπορεί να οδηγήσει σε αδιέξοδο

Πόροι (3)

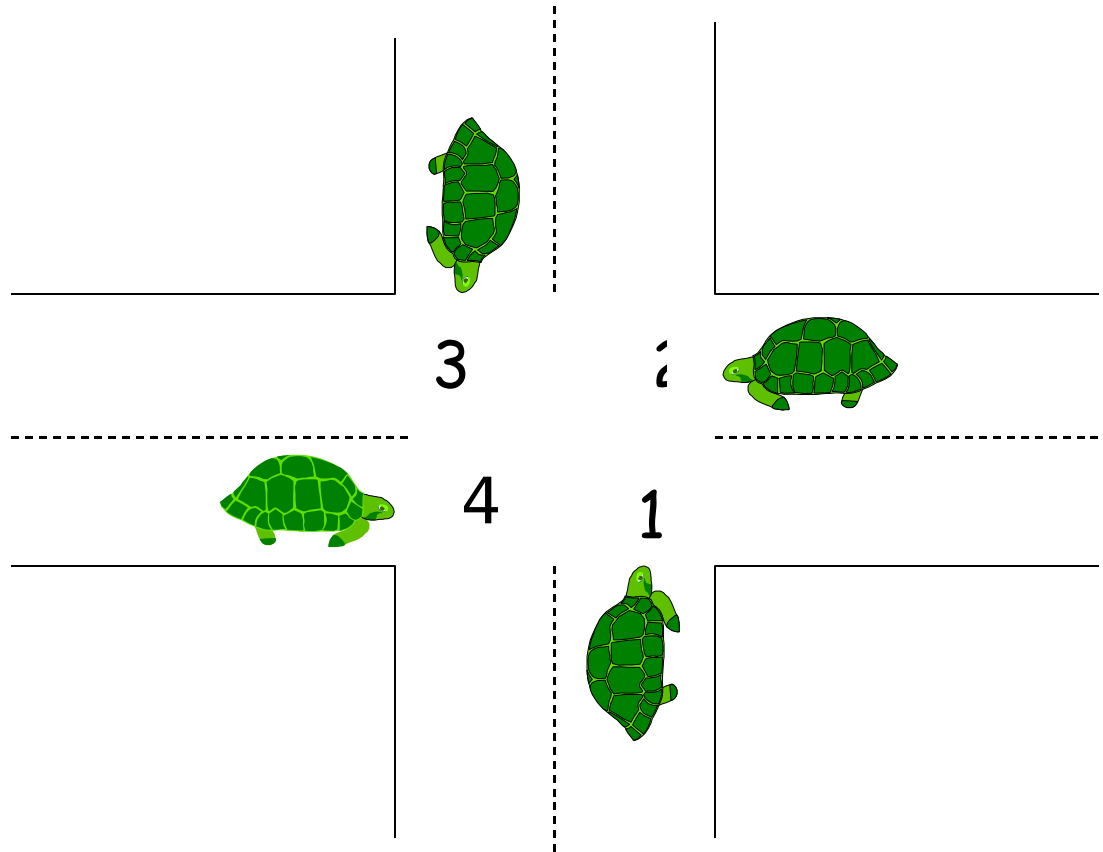
Ακολουθία γεγονότων για την απόκτηση πόρου:

- Αίτηση για απόκτηση πόρου
- Χρήση (κατάληψη) πόρου
- Απελευθέρωση πόρου

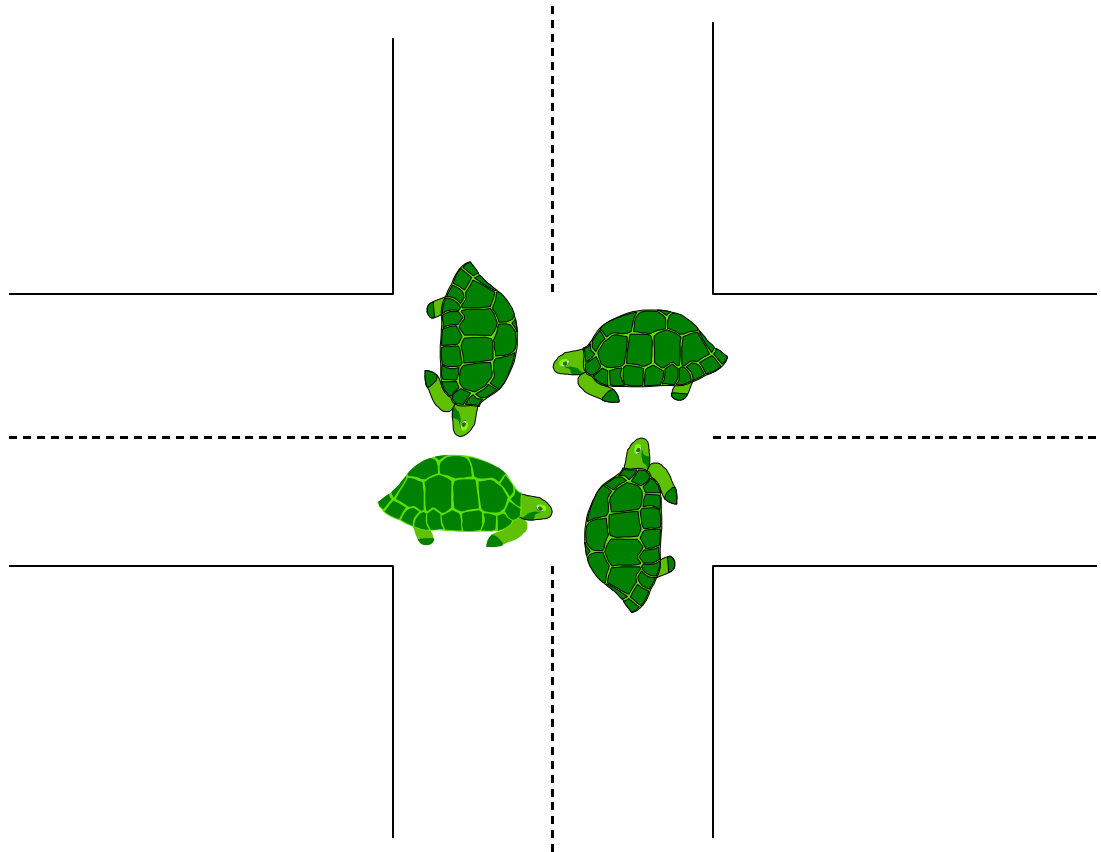
Αν η αίτηση δεν γίνει δεκτή η διεργασία πρέπει να περιμένει:

- Μπορεί να ανασταλεί (ή να μπει σε ενεργό αναμονή)
- Μπορεί να αποτύχει παράγοντας κωδικό σφάλματος (αν υπάρχει πχ περιορισμός πραγματικού χρόνου)

Το αδιέξοδο είναι πιθανό



Αδιέξοδο



Απόκτηση πόρου (1)

```
typedef int semaphore;  
semaphore resource_1;
```

```
void process_A(void) {  
    down(&resource_1);  
    use_resource_1( );  
    up(&resource_1);  
}
```

(a)

```
typedef int semaphore;  
semaphore resource_1;  
semaphore resource_2;
```

```
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

(b)

Χρήση semaphore για τον έλεγχο προσπέλασης σε πόρους.
(a) Ένας πόρος. (b) Δύο πόροι.

Απόκτηση πόρου (2)

```
semaphore resource_1;  
semaphore resource_2;
```

```
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

```
void process_B(void) {  
    down(&resource_2);  
    down(&resource_1);  
    use_both_resources( );  
    up(&resource_1);  
    up(&resource_2);  
}
```

Κώδικας που μπορεί να προκαλέσει αδιέξοδο. Και οι δύο διεργασίες προσπαθούν πρώτα να καταλάβουν τους δύο πόρους εναλλάξ. Αν η κάθε μια καταλάβει από ένα πόρο καμιά δε θα μπορέσει να συνεχίσει.

(b)

Απόκτηση πόρου (3)

```
typedef int semaphore;
    semaphore resource_1;
    semaphore resource_2;

void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}

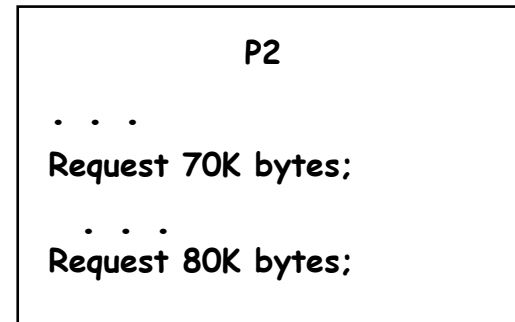
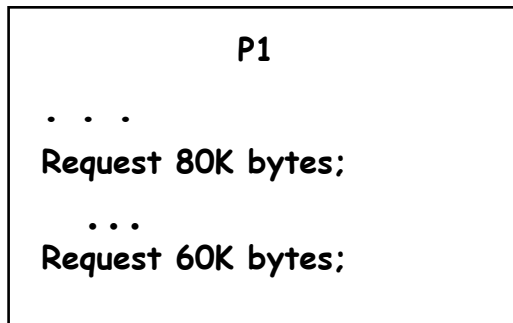
void process_B(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}
```

(a)

Κώδικας που αποφεύγει το αδιέξοδο. Και οι δύο διεργασίες προσπαθούν να καταλάβουν τους δύο πόρους με την ίδια σειρά. Μόνο μια διεργασία θα περάσει στο δεύτερο αίτημα, δηλαδή το πρώτο αίτημα περικλείει το δεύτερο.

Εξάντληση πόρου

Ο διαθέσιμος χώρος για κατανομή στην κύρια μνήμη είναι 200Kbytes, και πραγματοποιείται η παρακάτω σειρά αιτημάτων.



Το αδιέξοδο προκύπτει αν και οι δύο διεργασίες προχωρήσουν στο 2ο αίτημά τους.

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

Εισαγωγή στα αδιέξοδα

Τυπικός ορισμός :

Ένα σύνολο διεργασιών βρίσκεται σε αδιέξοδο αν κάθε διεργασία του συνόλου αναμένει ένα συμβάν που μόνο μια άλλη διεργασία του συνόλου μπορεί να προκαλέσει.

Συνήθως το συμβάν είναι η απελευθέρωση ενός δεσμευμένου πόρου.

Καμμία διεργασία δε μπορεί να ...

- Εκτελεστεί
- Απελευθερώσει πόρους
- Να αφυπνιστεί

Συνθήκες αδιεξόδου

1. Συνθήκη αμοιβαίου αποκλεισμού

Κάθε πόρος είναι: είτε δεσμευμένος από ακριβώς μία διεργασία είτε ελεύθερος.

2. Συνθήκη δέσμευσης και αναμονής

Διεργασίες που ήδη δεσμεύουν πόρους μπορούν να ζητήσουν και νέους πόρους.

3. Συνθήκη μη-προεκτόπισης

Πόροι που έχουν εκχωρηθεί σε διεργασίες δε μπορούν να απελευθερωθούν με εξαναγκασμό, πρέπει να τους ελευθερώσουν οι διεργασίες.

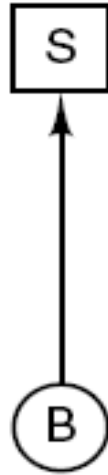
4. Συνθήκη κυκλικής αναμονής

Πρέπει να υπάρχει κύκλος από δύο ή περισσότερες διεργασίες, με τη κάθε μια να αναμένει για πόρο που είναι δεσμευμένος από επόμενο μέλος του κύκλου.

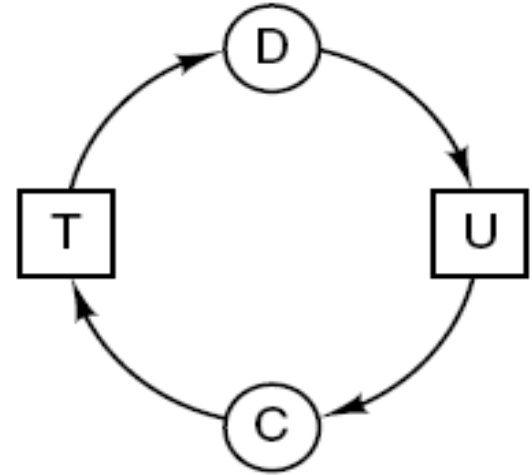
Μοντελοποίηση αδιεξόδου (1)



(a)



(b)



(c)

Γράφοι κατανομής πόρων.

(a) Η διεργασία A κατέχει (δεσμεύει) τον πόρο R.

(b) Η διεργασία B ζητά (αναμένει) τον πόρο S.

(c) **Αδιέξοδο**: Η διεργασία C ζητά τον πόρο T που δεσμεύεται από τη διεργασία D που ζητά τον πόρο U που δεσμεύεται από τη διεργασία C.

Μοντελοποίηση αδιεξόδου (2)

Στρατηγικές αντιμετώπισης αδιεξόδων

- Αγνοούμε το πρόβλημα, ας το λύσει ο χρήστης ή ο διαχειριστής συστήματος (αλγόριθμος στρουθοκαμήλου, μπλε οθόνες, ας πούμε ...).
- Εντοπισμός και ανάκαμψη, αφήνουμε το αδιέξοδο να συμβεί, το ανιχνεύουμε και το επιλύουμε κατά περίπτωση.
- Δυναμική αποφυγή με προσεκτική κατανομή πόρων, έλεγχο και πρόβλεψη κατά την εκτέλεση του λ.σ.
- Πρόληψη, δομική κατάργηση (στο κώδικα του λ.σ.) μιας από τις τέσσερις συνθήκες δημιουργίας αδιεξόδων.

Παράδειγμα (1)

A
Request R
Request S
Release R
Release S

(a)

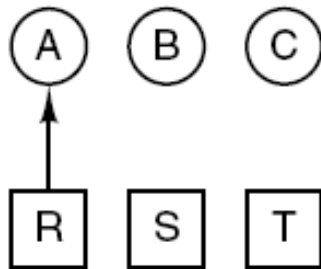
B
Request S
Request T
Release S
Release T

(b)

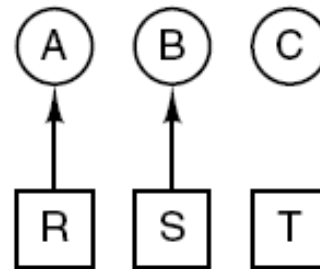
C
Request T
Request R
Release T
Release R

(c)

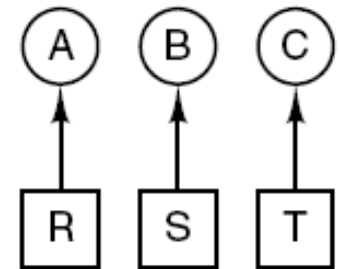
1. A requests R
2. B requests S
3. C requests T



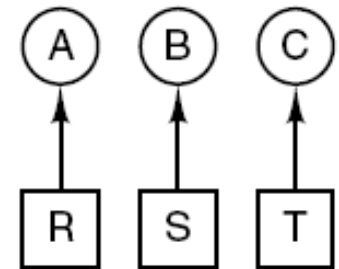
(d)



(e)



(f)



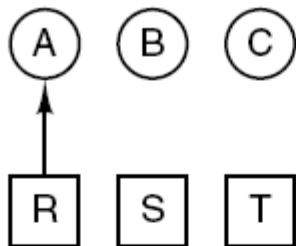
(g)

- (a) – (c): ακολουθίες εντολών στους κώδικες των διεργασιών A, B και C.
(d): σειρά εκτέλεσης εντολών στο σύστημα.
(e) – (g): αναπαράσταση των βημάτων 1-3.

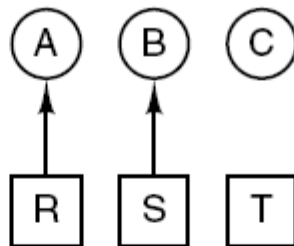
Παράδειγμα (2)

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

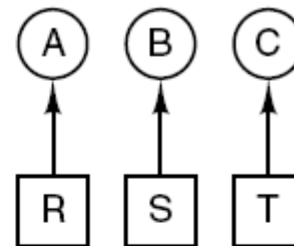
(d)



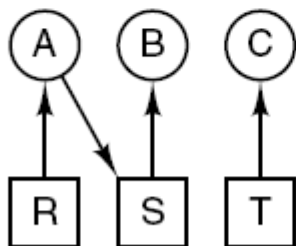
(e)



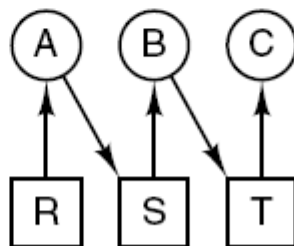
(f)



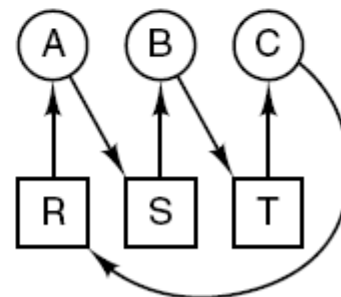
(g)



(h)



(i)



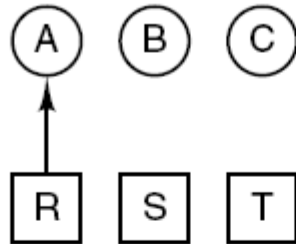
(j)

- (d): σειρά εκτέλεσης εντολών στο σύστημα.
(e) – (j): αναπαράσταση των βημάτων 1-6 και πρόκληση αδιεξόδου.

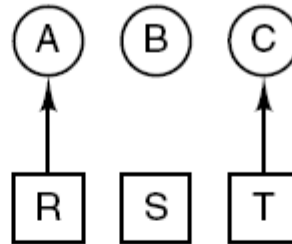
Παράδειγμα (3)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

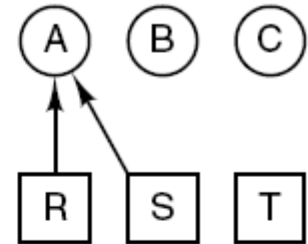
(k)



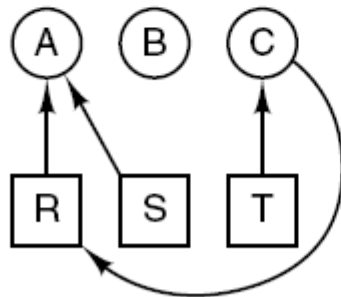
(l)



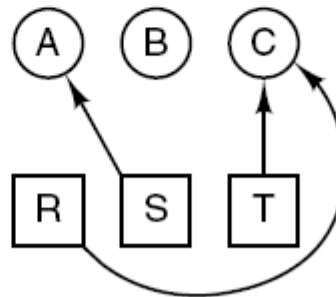
(m)



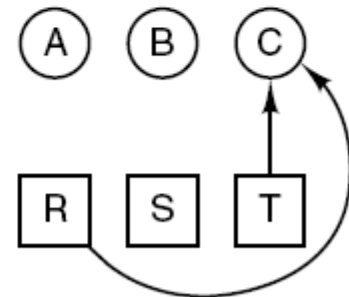
(n)



(o)



(p)



(q)

- (d): εναλλακτική σειρά εκτέλεσης εντολών στο σύστημα.
 (e) – (j): αναπαράσταση των βημάτων 1-6 και αποφυγή αδιεξόδου.

Αλγόριθμος στρουθοκαμήλου

Το πρόβλημα αγνοείται ή θεωρείται ότι τεχνικο-οικονομικά δεν είναι συμφέρουσα η αντιμετώπιση.

Λογική αντιμετώπιση αν

- Τα αδιέξοδα συμβαίνουν πολύ σπάνια
- Το απαιτούμενο κόστος είναι πολύ ψηλό

UNIX και Windows σε μερικές περιπτώσεις προσεγγίζουν το πρόβλημα με αυτό το τρόπο: εξισορρόπηση μεταξύ τεχνικά ορθού και τεχνικά εφικτού.

Πχ \$ ulimit -a

open files, max user processes, pending signals, stack size, ...

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

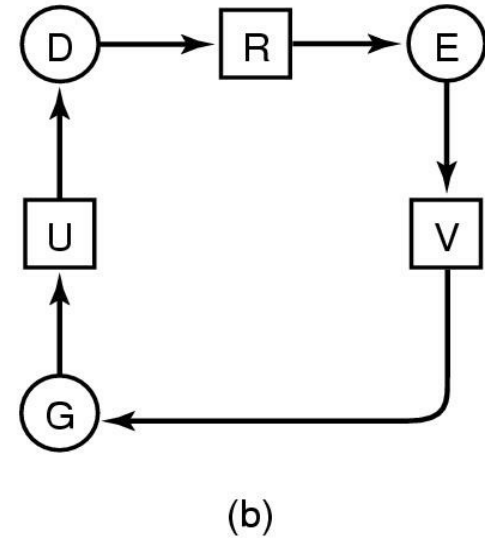
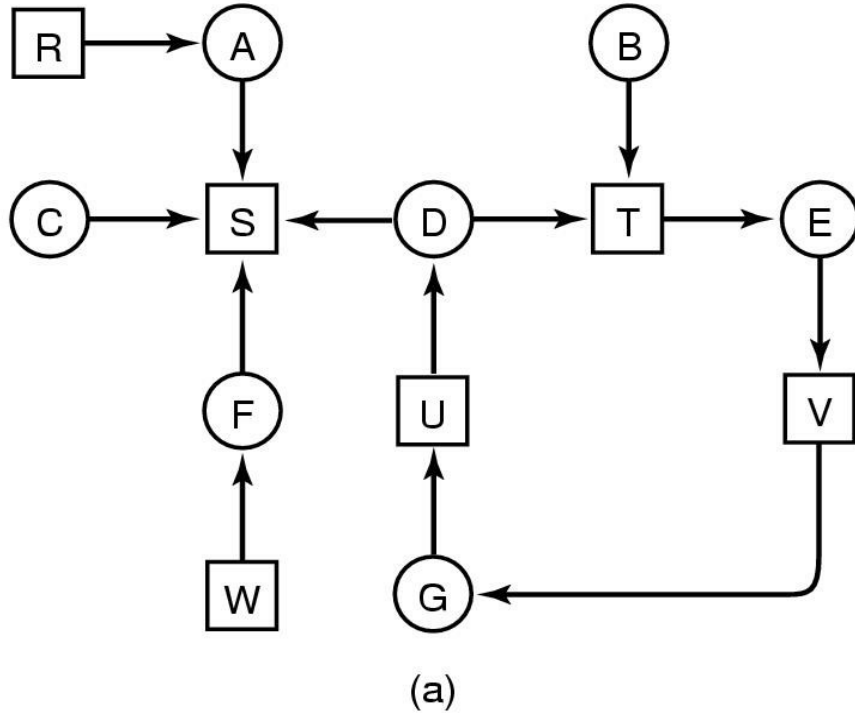
Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

Ανίχνευση αδιεξόδου με ένα πόρο από κάθε είδος (1)



Σημειώστε τις δεσμεύσεις και αιτήματα πόρων από τις διεργασίες (a). Ο κύκλος που δημιουργείται φαίνεται στο (b).

Ανίχνευση αδιεξόδου με ένα πόρο από κάθε είδος (2)

Αλγόριθμος εντοπισμού:

1. Για κάθε κόμβο N στο γράφο, εκτέλεση των επόμενων 5 βημάτων του αλγορίθμου, με σημείο εκκίνησης το N .
2. Αρχικοποίηση της μιας κενής λίστας L , και χαρακτηρισμός όλων των τόξων ως ασημείωτα.
3. Εισαγωγή του τρέχοντα κόμβου στο τέλος της λίστας L , έλεγχος να ο κόμβος εμφανίζεται δύο φορές στη λίστα L . Αν ο κόμβος εμφανίζεται δύο φορές, τερματισμός.

Ανίχνευση αδιεξόδου με ένα πόρο από κάθε είδος (3)

4. Για τον τρέχοντα κόμβο, έλεγχος αν υπάρχουν εξερχόμενα ασημείωτα τόξα. Αν ναι, συνέχεια στο βήμα 5, αλλιώς συνέχεια στο βήμα 6.
5. Επιλογή ενός εξερχόμενου ασημείωτου τόξου και σημείωσή του. Επιλογή του κόμβου που δείχνει το τόξο, συνέχεια στο βήμα 3.
6. Ο κόμβος δεν έχει ασημείωτα εξερχόμενα τόξα. Οπισθοδρόμηση στο προηγούμενο κόμβο και συνέχεια στο βήμα 3. Αν πρόκειται για τον αρχικό κόμβο, ο γράφος δεν περιέχει κύκλους, τερματισμός.

Παράδειγμα

Κόμβος	Λίστα	Βήματα	Τόξο
R	R	1, 2, 3, 4, 5	R->A
A	R, A	3, 4, 5	A -> S
S	R, A, S	3, 4, 6	
A	R, A	3, 4, 6	
R	R	3, 4, 6	
B	B	3, 4, 5	B -> T
T	B, T	3, 4, 5	T -> E
E	B, T, E	3, 4, 5	E -> V
V	B, T, E, V	3, 4, 5	V -> G
G	B, T, E, V, G	3, 4, 5	G -> U
U	B, T, E, V, G, U	3, 4, 5	U -> D
D	B, T, E, V, G, U, D	3, 4, 5	D -> S
S	B, T, E, V, G, U, D, S	3, 4, 6	
D	B, T, E, V, G, U, D	3, 4, 5	D -> T
T	B, T, E, V, G, U, D, T	3, Τερματισμός	

Ανίχνευση αδιεξόδου με πολλούς πόρους από κάθε είδος (1)

Resources in existence
($E_1, E_2, E_3, \dots, E_m$)

Resources available
($A_1, A_2, A_3, \dots, A_m$)

Current allocation matrix

Request matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row n is current allocation
to process n

Row 2 is what process 2 needs

Διάνυσηα υπαρχόντων πόρων E ,
Διάνυσηα διαθέσιμων πόρων A ,
Μητρώο τρέχουσας κατανομής C ,
Μητρώο αιτήσεων R .

n

$$\sum_{i=1}^n C_{ij} + A_j = E_j, \quad j = 1, 2, \dots, m.$$

Ανίχνευση αδιεξόδου με πολλούς πόρους από κάθε είδος (2)

Αλγόριθμος εντοπισμού:

1. Αναζήτηση για μια ασημείωτη διεργασία, P_i , $i=1, 2, \dots, n$ για την οποία η γραμμή i του μητρώου R είναι μικρότερη ή ίση του διανύσματος A , δηλαδή $R_{ij} \leq A_j$, $j = 1, 2, \dots, m$.
2. Αν βρεθεί μια τέτοια διεργασία τότε η γραμμή i του μητρώου C προστίθεται στο διάνυσμα A , δηλαδή $A_j = A_j + C_{ij}$, $j = 1, 2, \dots, m$. Σημειώνει τη διεργασία και πηγαίνει στο βήμα 1.
3. Αν δεν υπάρχει τέτοια διεργασία ο αλγόριθμος τερματίζει.

Παράδειγμα (1)

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Processes P1, P2, P3.

Παράδειγμα (2)

P1	R1 <= A ? [2 0 0 1] <= [2 1 0 0] FALSE	(1)
P2	R2 <= A ? [1 0 1 0] <= [2 1 0 0] FALSE	(1)
P3	R3 <= A ? [2 1 0 0] <= [2 1 0 0] TRUE	(1) 'χωραει'
	Σημείωση P3 προς εκτέλεση	(2) 'εκτελείται'
Εκτέλεση P3	και απελευθέρωση των πόρων της P3	(2) 'αποδεσμεύει'

$$A = A + C3 = [2 \ 1 \ 0 \ 0] + [0 \ 1 \ 2 \ 0] = [2 \ 2 \ 2 \ 0]$$

$$C = \begin{matrix} [0 \ 0 \ 1 \ 0] \\ [2 \ 0 \ 0 \ 1] \\ [0 \ 0 \ 0 \ 0] \end{matrix} \quad R = \begin{matrix} [2 \ 0 \ 0 \ 1] \\ [1 \ 0 \ 1 \ 0] \\ [0 \ 0 \ 0 \ 0] \end{matrix}$$

Παράδειγμα (3)

P1 R1 <= A ? [2 0 0 1] <= [2 2 2 0] FALSE

R2 <= A ? [1 0 1 0] <= [2 2 2 0] TRUE

Σημείωση P2 προς εκτέλεση

(1) 'χωράει'

(2) 'εκτελείται'

Εκτέλεση P2 και απελευθέρωση των πόρων της P2

(2) 'αποδεσμεύει'

$$A = A + C2 = [2 2 2 0] + [2 0 0 1] = [4 2 2 1]$$

$$C = \begin{array}{cc} [0 0 1 0] & [2 0 0 1] \\ [0 0 0 0] & R = [0 0 0 0] \\ [0 0 0 0] & [0 0 0 0] \end{array}$$

Παράδειγμα (4)

P1

$R1 \leq A ? [2\ 0\ 0\ 2] \leq [4\ 2\ 2\ 1]$ TRUE

(1) 'χωράει'

Σημείωση P1 προς εκτέλεση

(2) 'εκτελείται'

Εκτέλεση P1

και απελευθέρωση των πόρων της P1

(2) 'αποδεσμεύει'

$$A = A + C1 = [4\ 2\ 2\ 1] + [0\ 0\ 1\ 0] = [4\ 2\ 3\ 1] = E$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Ανάκαμψη από αδιέξοδο (1)

Ανάκαμψη μέσω προεκτόπισης

- Αφαίρεση πόρου από μια διεργασία και προσωρινή εκχώρηση σε άλλη διεργασία
- Εξαρτάται από τον τύπο του πόρου (προεκτοπίσιμος)

Ανάκαμψη μέσω επιστροφής (rollback)

- Περιοδικός έλεγχος διεργασιών σε ειδικά σημεία ελέγχου (checkpoints)
- Διαδοχικές καταγραφές της εικόνας (image) της διεργασίας, της δέσμευσης πόρων κλπ
- Αν ανιχνευτεί αδιέξοδο η διεργασία διακόπτεται, οι λειτουργίες μέχρι το τελευταίο checkpoint ακυρώνονται (rollback), οι πόροι αποδεσμεύονται
- Μετά τη λύση του αδιεξόδου η διεργασία επανέρχεται στη κατάσταση του τελευταίου checkpoint.

Ανάκαμψη από αδιέξοδο (2)

Ανάκαμψη μέσω εξάλειψης (kill)

- Άτσαλος αλλά απλός τρόπος επίλυσης αδιεξόδων
- Εξάλειψη (kill -9) μιας διεργασίας στο κύκλο του αδιεξόδου, ενώ οι άλλες διεργασίες συνεχίζουν
- Επιλογή διεργασίας που μπορεί να ξεκινήσει από την αρχή χωρίς πρόβλημα

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

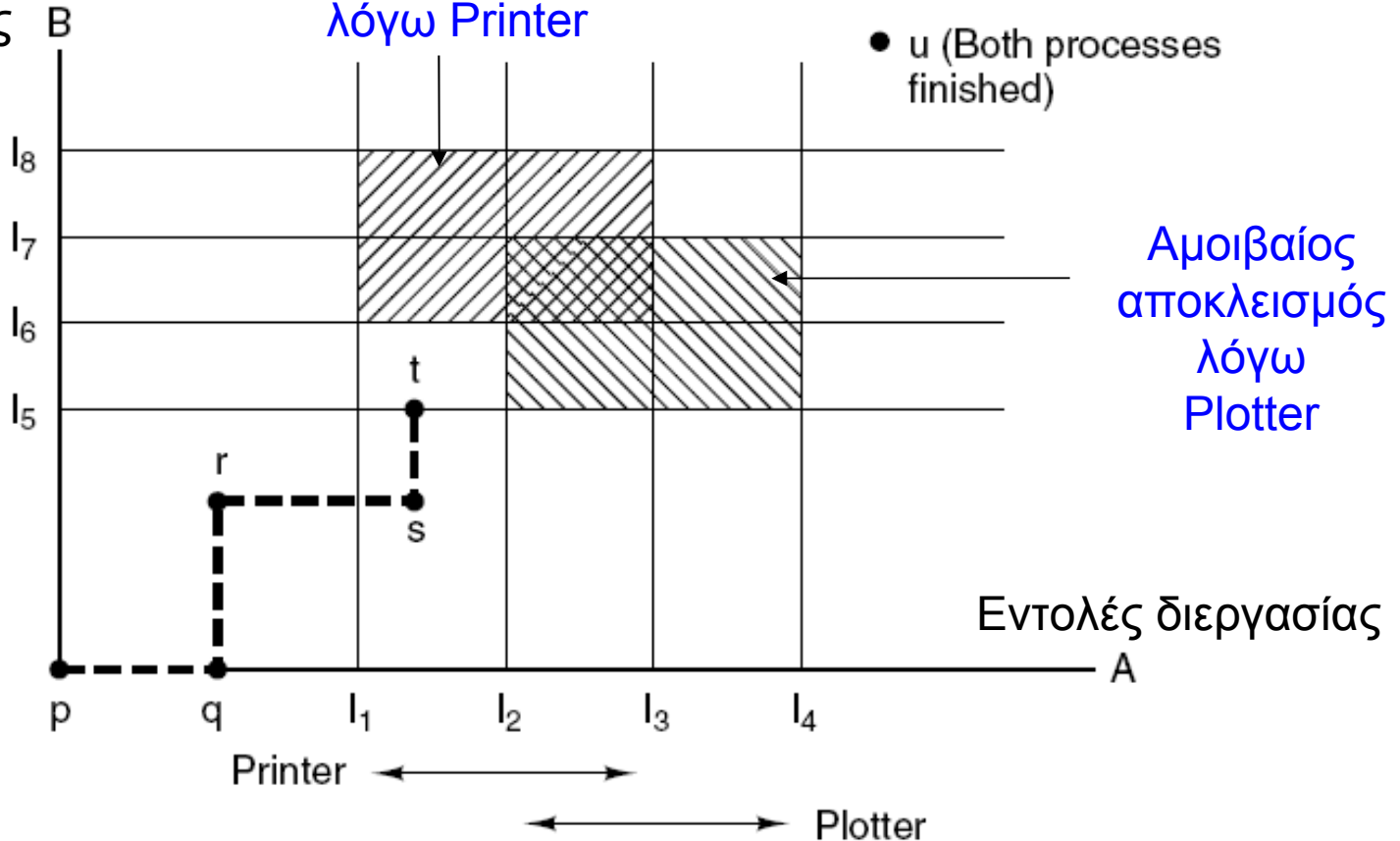
Τροχιές και καταστάσεις διεργασιών (1)

Αμοιβαίος αποκλεισμός
λόγω Printer

Εντολές διεργασίας

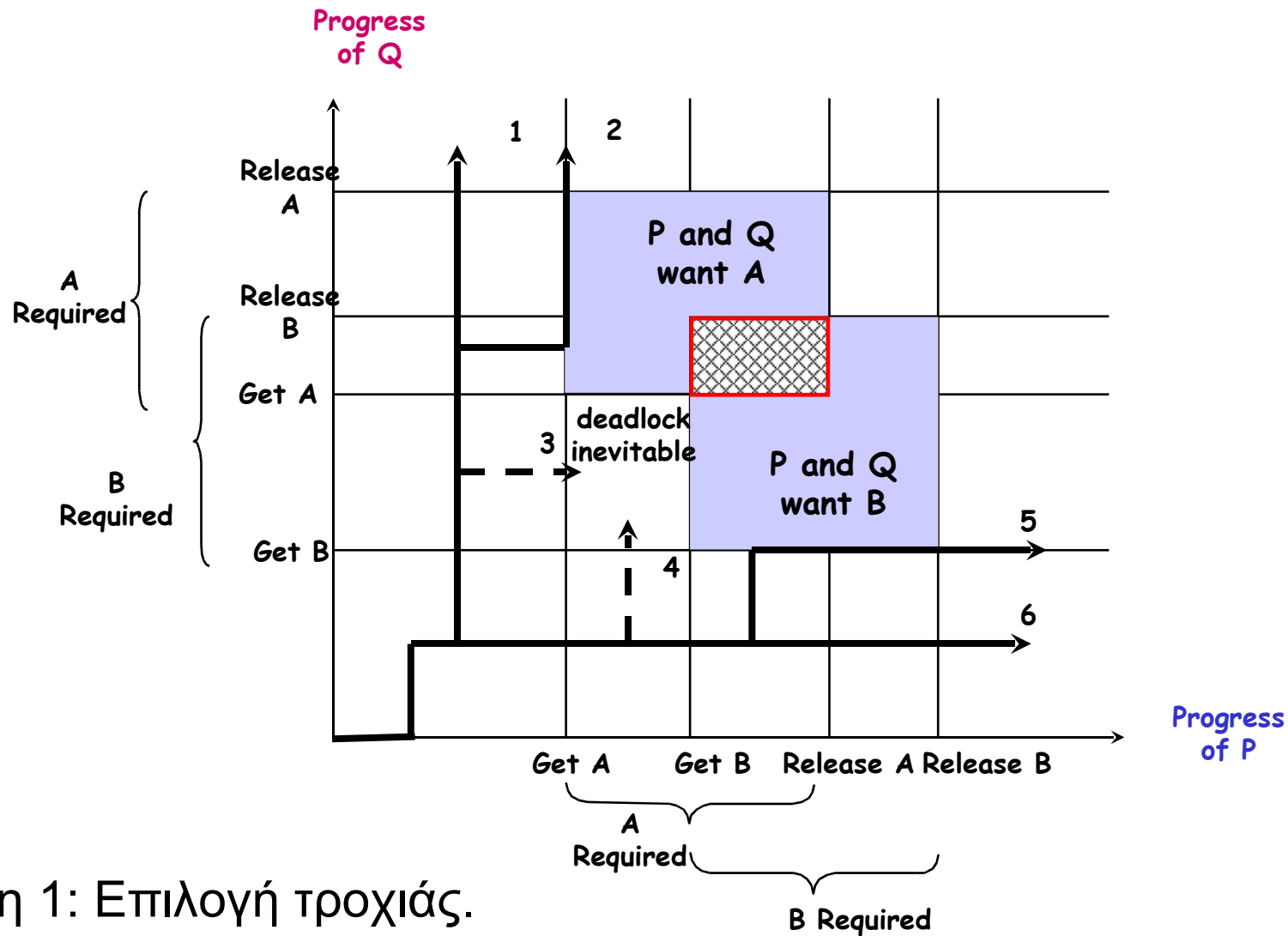
Printer

Plotter



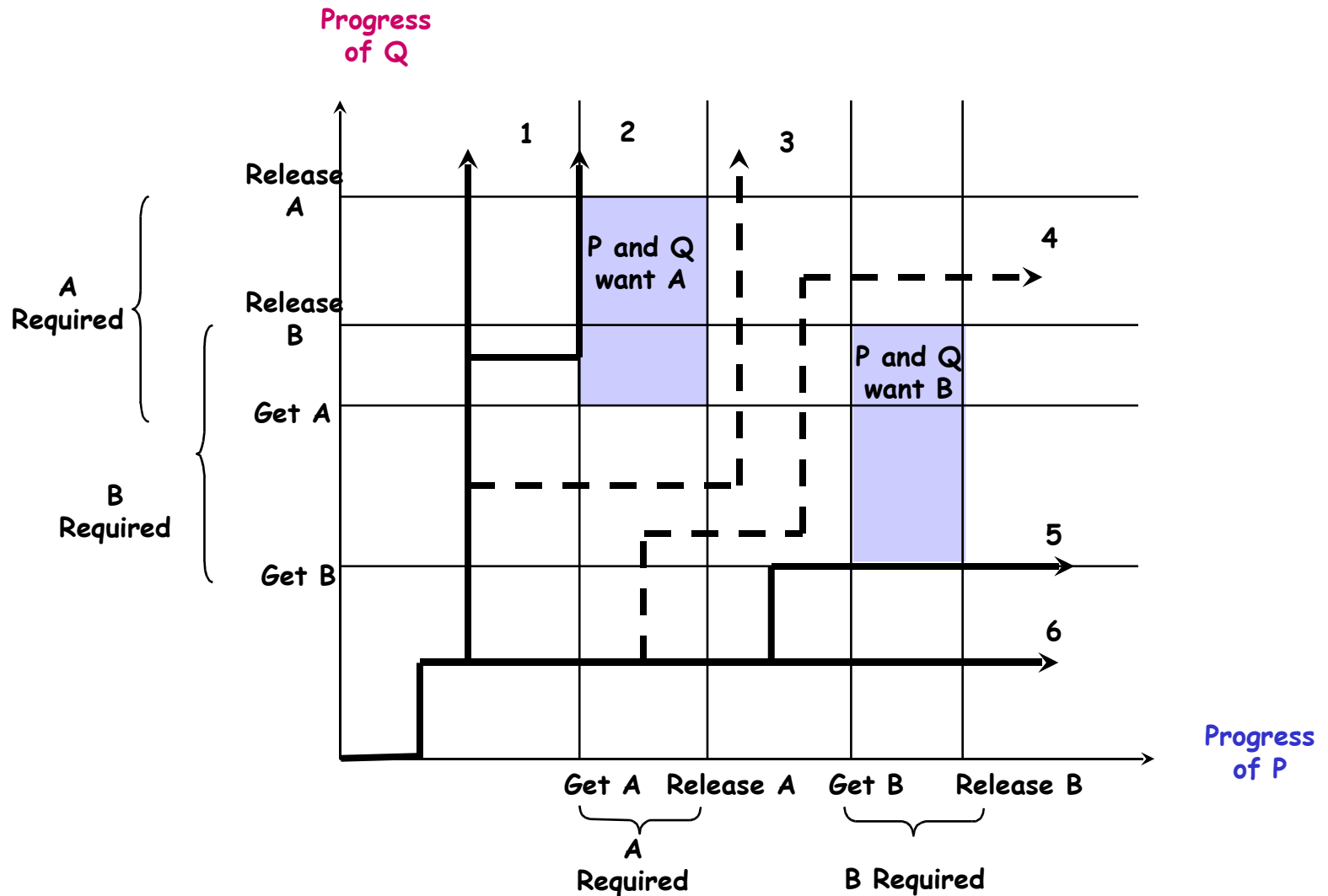
Δύο διεργασίες A, B εκτελούνται εναλλάξ. Η A στο σημείο r-s (I1), ζητά και δεσμεύει τον Printer. Στη συνέχεια η B, στο σημείο s-t (I5) ζητά τον Plotter. Το λ.σ. πρέπει να αναστείλει τη B και να αφήσει πρώτα την A να φθάσει στο σημείο I4.

Τροχιές και καταστάσεις διεργασιών (2)



Λύση 1: Επιλογή τροχιάς.

Τροχιές και καταστάσεις διεργασιών (3)



Λύση 2: Αλλαγή δεσμεύσεων

Ασφαλείς και ανασφαλείς καταστάσεις (1)

Has Max		
A	3	9
B	2	4
C	2	7

Free: 3

(a)

Has Max		
A	3	9
B	4	4
C	2	7

Free: 1

(b)

Has Max		
A	3	9
B	0	-
C	2	7

Free: 5

(c)

Has Max		
A	3	9
B	0	-
C	7	7

Free: 0

(d)

Has Max		
A	3	9
B	0	-
C	0	-

Free: 7

(e)

Τρεις διεργασίες (A, B, C) και ένας πόρος (διαθέσιμος σε 10 αντίγραφα).
Εκκίνηση από τη κατάσταση (a) ελεύθεροι 3 πόροι.
Σειρά εκτέλεσης:

- (b) δέσμευση 2 πόρων και εκτέλεση B
- (c) αποδέσμευση πόρων από B
- (d) δέσμευση 5 πόρων και εκτέλεση C
- (e) αποδέσμευση πόρων από C
- (f) δέσμευση 6 πόρων και εκτέλεση A
- (g) ελεύθεροι 10 πόροι

Ασφαλείς και ανασφαλείς καταστάσεις (2)

Has Max

A	3	9
B	2	4
C	2	7

Free: 3

(a)

Has Max

A	4	9
B	2	4
C	2	7

Free: 2

(b)

Has Max

A	4	9
B	4	4
C	2	7

Free: 0

(c)

Has Max

A	4	9
B	—	—
C	2	7

Free: 4

(d)

Τρεις διεργασίες (A, B, C) και ένας πόρος (διαθέσιμος σε 10 αντίγραφα).

Εκκίνηση από τη κατάσταση (a) ελεύθεροι 3 πόροι.

Σειρά εκτέλεσης:

(b) δέσμευση 1 πόρου από A

(c) δέσμευση 2 πόρων και εκτέλεση B

(d) αποδέσμευση πόρων από B

(e) ελεύθεροι 4 πόροι

Δεν μπορεί να εκτελεστεί ούτε η A ούτε η C.

Αλγόριθμος τραπεζίτη για ένα πόρο

Έστω n διεργασίες, P_i , $i=1, 2, \dots, n$ και 1 πόρος (σε K αντίγραφα).

C_i οι πόροι που κατέχει κάθε διεργασία

R_i οι πόροι που ζητά κάθε διεργασία

A_i οι μέγιστοι πόροι που μπορεί να ζητήσει κάθε διεργασία, $C_i + R_i \leq A_i$

$$\text{Ελεύθεροι πόροι} = K - \sum_{i=1}^n C_i$$

Για κάθε αίτημα A_i πρέπει:

Μετά την εξυπηρέτησή του αιτήματος οι ελεύθεροι πόροι που απομένουν να μπορούν να εξυπηρετήσουν τουλάχιστο μια διεργασία αν αυτή ζητήσει το μέγιστο του υπολοίπου της.

Να υπάρχει P_j , $j=1, 2, \dots, n$ $i \neq j$ ώστε $(\text{Ελεύθεροι πόροι} + R_i) \leq A_j - C_j$

Παράδειγμα

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

(a) Αρχική κατάσταση: 4 διεργασίες, 1 πόρος (10 αντίγραφα), οι πόροι που κατέχουν οι διεργασίες και το μέγιστο αίτημα ανά διεργασία.

Η κατάσταση (b) είναι **Ασφαλής** γιατί η C σίγουρα εξυπηρετείται με τους ελεύθερους 2 πόρους.

Η κατάσταση (c) είναι **Ανασφαλής** γιατί καμμία διεργασία δεν εξυπηρετείται σίγουρα με τον 1 ελεύθερο πόρο που απομένει.

Αλγόριθμος τραπεζίτη για πολλούς πόρους (1)

Process
Tape drives
Plotters
Printers
CD ROMs

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process
Tape drives
Plotters
Printers
CD ROMs

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

$$E = (6342)$$

$$P = (5322)$$

$$A = (1020)$$

Διάνυσα υπαρχόντων πόρων E,
Διάνυσμα κατεχόμενων πόρων P,
Διάνυσμα διαθέσιμων πόρων A,
Μητρώο τρέχουσας κατανομής C,
Μητρώο αιτήσεων R.

$$E_j = P_j + A_j, \quad j = 1, 2, \dots, m$$

n

$$\sum_{i=1}^n C_{ij} = P_j, \quad j = 1, 2, \dots, m.$$

Αλγόριθμος τραπεζίτη για πολλούς πόρους (2)

Έλεγχος ασφαλούς κατάστασης:

1. Αναζήτηση για μια ασημείωτη διεργασία, P_i , $i=1, 2, \dots, n$ για την οποία η γραμμή i του μητρώου R είναι μικρότερη ή ίση του διάνυσματος A , δηλαδή $R_{ij} \leq A_j$, $j = 1, 2, \dots, m$. Αν δεν υπάρχει τέτοια διεργασία είμαστε σε ανασφαλή κατάσταση και το σύστημα θα οδηγηθεί σε αδιέξοδο.
2. Δέσμευση των πόρων της διεργασίας που επιλέχθηκε, εκτέλεση της διεργασίας, σημείωση της διεργασίας ως εκτελεσθείσας. Οι πόροι της προστίθενται στο διάνυσμα A .
3. Επανάληψη των βημάτων 1 και 2 έως να σημειωθούν όλες οι διεργασίες (ασφαλής αρχική κατάσταση) ή να μην υπάρχει επιλογή στο βήμα 1 (ανασφαλής αρχική κατάσταση που οδηγεί σε αδιέξοδο).

Παράδειγμα (1)

- P1 R1 <= A ? [1 1 0 0] <= [1 0 2 0] FALSE (1)
 - P2 R2 <= A ? [0 1 1 2] <= [1 0 2 0] FALSE (1)
 - P3 R3 <= A ? [3 1 0 0] <= [1 0 2 0] FALSE (1)
 - P4 R4 <= A ? [0 0 1 0] <= [1 0 2 0] TRUE (1) 'χωράει'
- Σημείωση P4 προς εκτέλεση (2) 'εκτελείται'
- Εκτέλεση P4 και απελευθέρωση των πόρων της P4 (2) 'αποδεσμεύει'

$$A = A + C4 = [1 0 2 0] + [1 1 0 1] = [2 1 2 1]$$

$$C = \begin{bmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

Παράδειγμα (2)

P1 R1 <= A ? [1 1 0 0] <= [2 1 2 1] TRUE
 Σημείωση P1 προς εκτέλεση

(1) 'χωράει'
(2) 'εκτελείται'

Εκτέλεση P1 και απελευθέρωση των πόρων της P1

(2) 'αποδεσμεύει'

$$A = A + C1 = [2 \ 1 \ 2 \ 1] + [3 \ 0 \ 1 \ 1] = [5 \ 1 \ 3 \ 2]$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

Παράδειγμα (3)

P2 R2 <= A ? [0 1 1 2] <= [5 1 3 2] TRUE

(1) 'χωράει'

Σημείωση P2 προς εκτέλεση

(2) 'εκτελείται'

Εκτέλεση P2 και απελευθέρωση των πόρων της P2

(2) 'αποδεσμεύει'

$$A = A + C2 = [5 1 3 2] + [0 1 0 0] = [5 2 3 2]$$

$$C = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [1 1 1 0] \\ [0 0 0 0] \\ [0 0 0 0] \end{bmatrix} \quad R = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [3 1 0 0] \\ [0 0 0 0] \\ [2 1 1 0] \end{bmatrix}$$

Παράδειγμα (4)

P3 R2 <= A ? [3 1 0 0] <= [5 1 3 2] TRUE

(1) 'χωράει'

Σημείωση P3 προς εκτέλεση

(2) 'εκτελείται'

Εκτέλεση P3 και απελευθέρωση των πόρων της P3

(2) 'αποδεσμεύει'

$$A = A + C3 = [5 2 3 2] + [1 1 1 0] = [6 3 4 2]$$

$$C = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \end{bmatrix} \quad R = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [2 1 1 0] \end{bmatrix}$$

Παράδειγμα (5)

P5 R2 <= A ? [2 1 1 0] <= [6 3 4 2] TRUE

(1) 'χωράει'

Σημείωση P5 προς εκτέλεση

(2) 'εκτελείται'

Εκτέλεση P5 και απελευθέρωση των πόρων της P5

(2) 'αποδεσμεύει'

$$A = A + C5 = [6 3 4 2] + [0 0 0 0] = [6 3 4 2]$$

$$C = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \end{bmatrix} \quad R = \begin{bmatrix} [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \\ [0 0 0 0] \end{bmatrix}$$

Παραδοχές

- Πολλαπλά στιγμιότυπα των πόρων
- Κάθε διεργασία πρέπει εκ των προτέρων να διεκδικεί τη μέγιστη χρήση των πόρων
- Όταν μια διεργασία απαιτεί έναν πόρο ίσως χρειαστεί να περιμένει
- Όταν μια διεργασία λάβει όλους τους πόρους πρέπει να τους επιστρέψει σε πεπερασμένο χρονικό διάστημα.
- Η μέγιστη απαίτηση για πόρους πρέπει να δηλώνεται εκ των προτέρων
- Οι σημαντικές διεργασίες πρέπει να είναι ανεξάρτητες και δεν υπόκεινται σε απαιτήσεις συγχρονισμού
- Πρέπει να υπάρχει ένας σταθερός αριθμός πόρων προς ανάθεση
- Καμιά διεργασία δεν μπορεί να περιέλθει σε κατάσταση εξόδου (exit) ενώ δεσμεύει πόρους
- Συντηρητικός αλγόριθμος

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

Πρόληψη αδιεξόδου

- Προσβολή της συνθήκης αμοιβαίου αποκλεισμού
- Προσβολή της συνθήκης δέσμευσης και ανμονής
- Προσβολή της συνθήκης μη προ-εκτόπισης
- Προσβολή της συνθήκης κυκλικής αναμονής

Προσβολή της συνθήκης αμοιβαίου αποκλεισμού

Μερικές συσκευές (όπως οι εκτυπωτές) μπορούν λειτουργήσουν με παροχέτευση (spooling), δηλαδή σειριοποίηση των αιτήσεων χρήσης του πόρου.

Η μόνη διεργασία που δεσμεύει τον πόρο είναι ο spooler. Έτσι τα πιθανά αδιέξοδα εξαλείφονται αφού ο spooler δεν ζητά άλλους πόρους.

Πρόβλημα:

Δεν είναι δυνατή η σειριοποίηση όλων των αιτημάτων: ακύρωση χωρικής και χρονικής πολυπλεξίας.

Γενικές αρχές:

Αποφεύγουμε την δέσμευση πόρου παρά μόνο αν είναι απολύτως απαραίτητη.

Ελαχιστοποιούμε τους πόρους που μπορεί να δεσμεύσουν ένα πόρο.

Προσβολή της συνθήκης δέσμευσης και αναμονής

Οι διεργασίες επιβάλλεται να ζητούν και να δεσμεύουν εξ' αρχής όλους τους πόρους που χρειάζονται προκειμένου να εκτελεστούν πλήρως. Έτσι εξαλείφεται η αναμονή για πόρους κατά την εκτέλεση.

Προβλήματα:

Εφαρμογή μόνο αν οι πόροι είναι γνωστοί εκ των προτέρων (πχ σε συστήματα δέσμης).

Οι πόροι δεσμεύονται για διάστημα μεγαλύτερο από αυτό που πραγματικά απαιτείται: σπατάλη πόρων, μείωση της χρονικής πολυπλεξίας (concurrency).

Τροποποίηση:

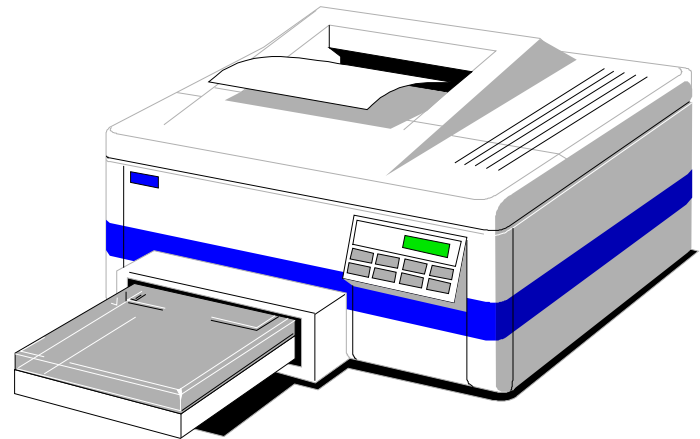
Μια διεργασία που ζητά πόρους υποχρεώνεται να αποδεσμεύσει όλους τους πόρους της και στη συνέχεια ζητά όλους τους πόρους μαζί.

Προσβολή της συνθήκης μη-προεκτόπισης

Περιορισμένη εφαρμογή: απαιτεί αφαίρεση πόρου από διεργασία.

Πχ αφαίρεση εκτυπωτή από διεργασία που έχει ξεκινήσει εκτύπωση !!??

Έχει εφαρμογή μόνο σε προεκτοπίσιμους πόρους (πχ μνήμη).



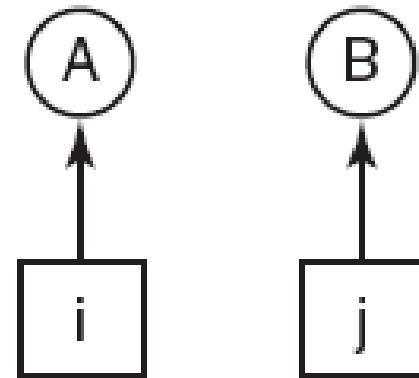
Προσβολή της συνθήκης κυκλικής αναμονής (1)

Εκδοχή 1η: Οι διεργασίες επιβάλλεται να δεσμεύουν μόνο ένα πόρο κάθε φορά. Απαλείφει τους κύκλους **αλλά** δεν είναι γενικά εφαρμόσιμη (πχ μεταφορά αρχείου στο spooler για εκτύπωση).

Εκδοχή 2η: Γενική αρίθμηση πόρων. Οι διεργασίες δεσμεύουν όσους πόρους θέλουν αλλά με τη σειρά αρίθμησης. Έτσι αποτρέπονται οι κύκλοι.

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)



(b)

(a) Αρίθμηση πόρων. (b) Η διεργασία A μπορεί να ζητήσει τον πόρο j αλλά η διεργασία B δεν μπορεί να ζητήσει τον πόρο i.

Προσβολή της συνθήκης κυκλικής αναμονής (2)

Εκδοχή 2η (συνέχεια): Επιβάλλει σειρά στην εκτέλεση των διεργασιών: σε περίπτωση κοινού αιτήματος πρώτες θα εκτελεστούν αυτές που κατέχουν πόρους με μεγαλύτερο αριθμό.

Εκδοχή 3η: Γενική αρίθμηση πόρων. Οι διεργασίες δεσμεύουν όσους πόρους θέλουν αλλά με αριθμό μεγαλύτερο από αυτό που ήδη κατέχουν.

Προβλήματα:

Η γενική αρίθμηση πόρων δεν είναι πάντα εύκολη.

Σύνοψη

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα

Πόροι

Εισαγωγή στα αδιέξοδα

Ανίχνευση και ανάκαμψη από αδιέξοδα

Αποφυγή αδιεξόδου

Πρόληψη αδιεξόδου

Άλλα θέματα

Άλλα θέματα

- Κλείδωμα δύο φάσεων
- Ενεργό αδιέξοδο (Livelock)
- Λιμοκτονία (Starvation)
- Γευματίζοντες φιλόσοφοι (ξανά)

Κλείδωμα δύο φάσεων

Πρώτη φάση

Η διεργασία προσπαθεί να κλειδώσει όλες τις εγγραφές (πόρους) που χρειάζεται, ένα προς ένα.

Αν η απαιτούμενη εγγραφή (πόρος) είναι κλειδωμένη(ος) τότε η διεργασία ξεκινά πάλι από την αρχή.

Στη πρώτη φάση δεν γίνεται καμμία επεξεργασία.

Αν η πρώτη φάση πετύχει τότε μόνο ξεκινά η

Δεύτερη Φάση

Επεξεργασία των εγγραφών (χρήση των πόρων)

Σταδιακή αποδέσμευση (ξεκλείδωμα) των εγγραφών (πόρων)

Σημειώστε την ομοιότητα με τη προσβολή της συνθήκης δέσμευσης και αναμονής.

Ο αλγόριθμος δουλεύει σε προγράμματα που επιτρέπεται η τυχαία διακοπή και επανάληψη από την αρχή της πρώτης φάσης.

Ενεργό αδιέξοδο (1)

Δυο διεργασίες μπορεί να βρεθούν σε αδιέξοδο αναμένοντας η μία την άλλη να εκτελέσει κάποια εργασία.

Μπορεί να συμβεί σε συνθήκες ανταγωνισμού, όπου ο πόρος είναι μια κρίσιμη περιοχή και χρησιμοποιούνται

- Ενεργός αναμονή
- Σηματοφόροι

Συνήθως οφείλεται σε λανθασμένη τοποθέτηση των λειτουργιών αποδέσμευσης μιας κρίσιμης περιοχής (*signal()*, *down()* κλπ).

Ενεργό αδιέξοδο (2)

```
void process_A(void) {  
    enter_region(&resource_1);  
    enter_region(&resource_2);  
    use_both_resources( );  
    leave_region(&resource_2);  
    leave_region(&resource_1);  
}
```

```
void process_B(void) {  
    enter_region(&resource_2);  
    enter_region(&resource_1);  
    use_both_resources( );  
    leave_region(&resource_1);  
    leave_region(&resource_2);  
}
```

Παράδειγμα σφάλματος λόγω κακής τοποθέτησης εντολών ελέγχου κρίσιμης περιοχής.

Λιμοκτονία

Ακραίες περιπτώσεις (worst case scenarios) στην εφαρμογή αλγορίθμων εκχώρησης πόρου (ακόμη και CPU).

Πχ αλγόριθμος χρονοπρογραμματισμού Shortest Job First Pre-emptive (SJFP): δουλεύει καλά όσο ο αριθμός (και ρυθμός εισόδου) των διεργασιών σε ένα σύστημα είναι φραγμένος άνω.

Όμως αν ο αριθμός εργασιών είναι πολύ μεγάλος τότε μια CPU-bound διεργασία μπορεί να αναμένει πολύ ακόμη και αν δεν είναι σε αναστολή.

Λύσεις:

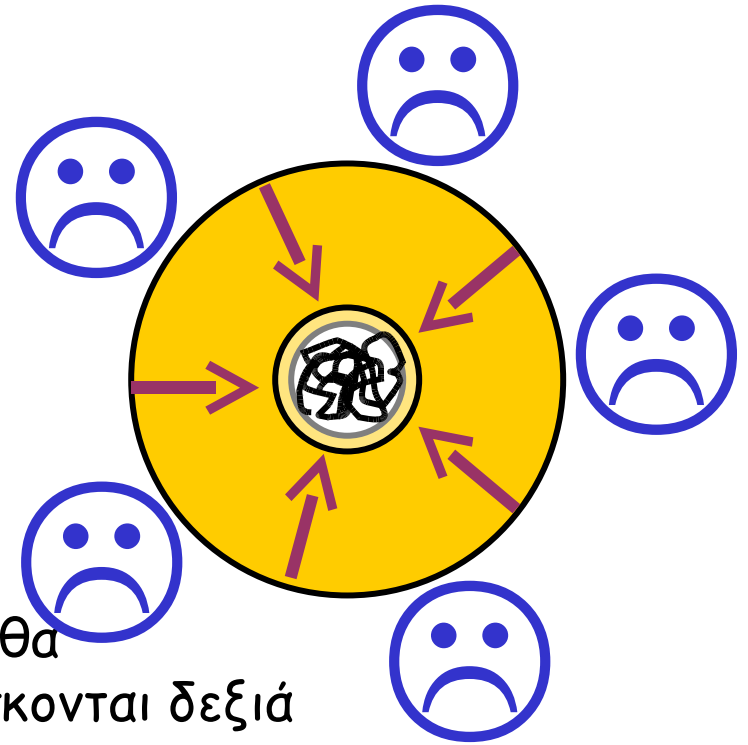
Εφαρμογή χρονικά μεταβαλλόμενων προτεραιοτήτων ή εφαρμογή αλγορίθμου First-come, first-serve (FCFS) μετά από κάποιο χρονικό όριο.

Γευματίζοντες φιλοσόφοι (1)

Πέντε φιλόσοφοι κάθονται γύρω από ένα κυκλικό τραπέζι. Κάθε φιλόσοφος καταναλώνει το χρόνο του διαδοχικά **σκεπτόμενος** και **τρώγοντας**. Στο κέντρο του τραπεζιού υπάρχει ένα μεγάλο πιάτο με spaghetti. Κάθε φιλόσοφος χρειάζεται δύο πηρούνια (forks) για να φάει λίγο spaghetti.

Υπάρχει ένα πηρούνι ανάμεσα σε κάθε ζεύγος φιλοσόφων και όλοι συμφωνούν ότι θα χρησιμοποιούν μόνον τα πηρούνια που βρίσκονται δεξιά και αριστερά από τον καθένα.

Κάθε φιλόσοφος είναι μια διεργασία και κάθε πηρούνι είναι ένας **διαμοιραζόμενος πόρος** με ενέργειες δέσμευσης και απελευθέρωσης. Αν ένας φιλόσοφος πεινάσει, πρέπει πρώτα να πάρει τα πηρούνια δεξιά και αριστερά του δεξιό για να μπορέσει να ξεκινήσει να τρώει.



Γευματίζοντες φιλοσόφοι (2)

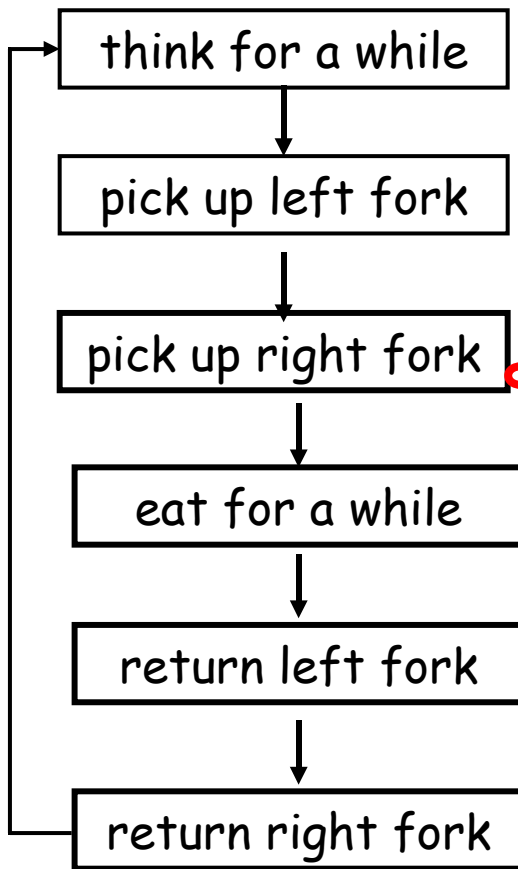
Το πρόβλημα αυτό είναι ένα πρότυπο που χρησιμοποιείται για την αποτίμηση μεθόδων σχετικών με το συγχρονισμό ταυτόχρονων διεργασιών.

Καταδεικνύει τη δυσκολία της εκχώρησης πόρων μεταξύ διεργασιών χωρίς αδιέξοδα και παρατεταμένες στερήσεις.

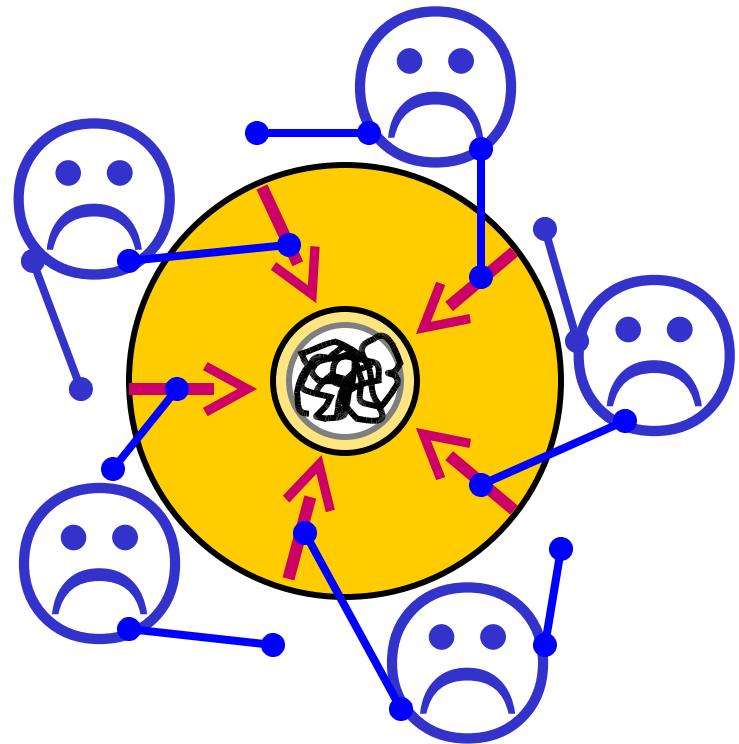
Στόχος είναι η ανάπτυξη ενός πρωτοκόλλου απόκτησης των πηρουινιών που θα εξασφαλίζει

- την απαλλαγή από αδιέξοδα
- τη δικαιοσύνη: κανένας φιλόσοφος δεν πρέπει να υποφέρει από παρατεταμένη στέρηση
- το μέγιστο δυνατό συγχρονισμό

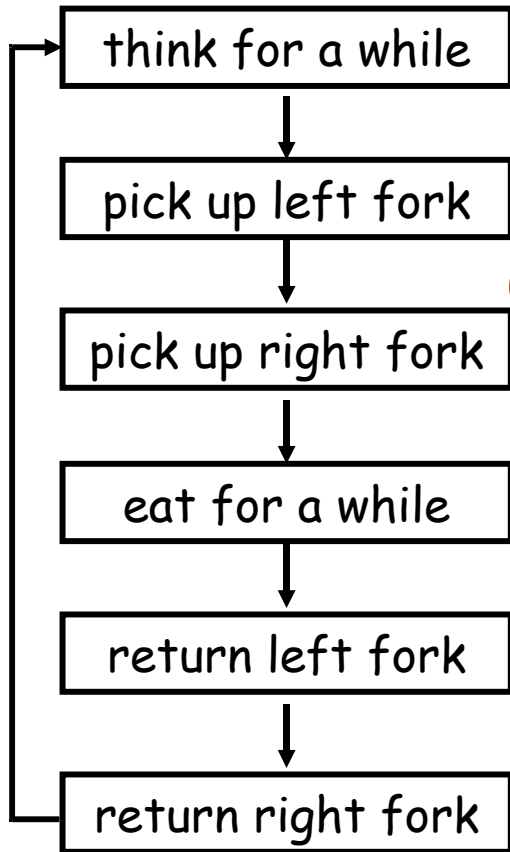
Η προφανής λύση (1)



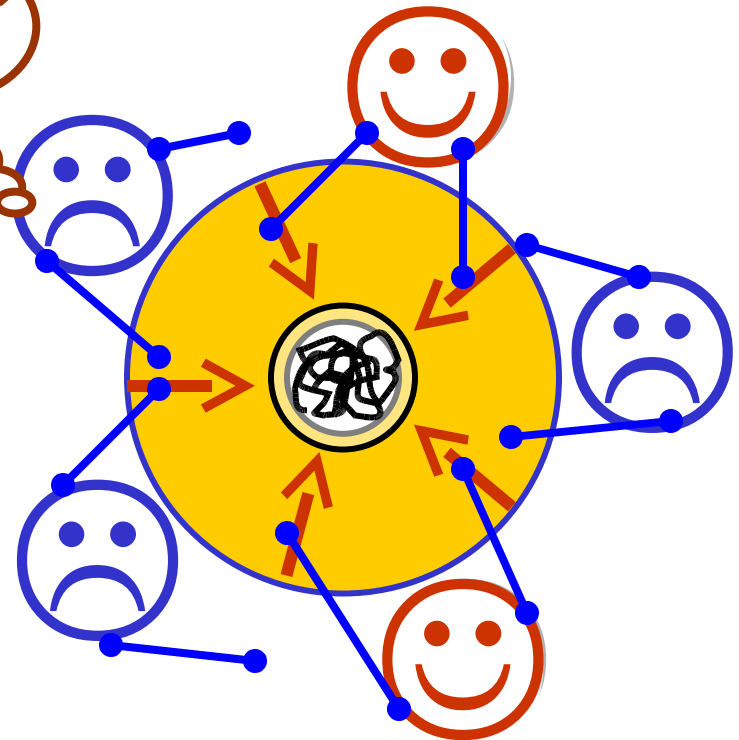
Μπορεί να συμβεί αδιέξοδο...



Η προφανής λύση (2)



Η παρατεταμένη στέρση είναι πρόβλημα...



Λύση

Δεν υπάρχει συμμετρική λύση

Λύσεις

Προσθήκη ενός ακόμη πηρουνιού

Μέγιστος αριθμός 4 φιλοσόφων στο τραπέζι (κυκλική αναμονή)

Να εκτελείται διαφορετική αλληλουχία ενεργειών για τους φιλοσόφους με άρτιο και περιττό αύξοντα αριθμό δηλαδή δημιουργία δύο ομάδων φιλοσόφων. Η μία ομάδα (περιττός a/a) θα αποκτά πρώτα το δεξιό και μετά το αριστερό πηρούνη και η άλλη ομάδα (άρτιος a/a) πρώτα το αριστερό και μετά το δεξιό πηρούνη.

Ένας φιλόσοφος επιτρέπεται να αποκτήσει τα πηρούνια μόνον όταν και τα δύο είναι διαθέσιμα (κρίσιμο τμήμα) - (κατοχή και αναμονή)

Σχεδιασμός του συστήματος έτσι ώστε ένας φιλόσοφος να «κλέψει» ένα πηρούνη που δεν είναι γειτονικό του.

Αλγόριθμος Lehmann-Rabin (non deterministic)

Δες λύσεις στη Διαδιεργασιακή Επικοινωνία.