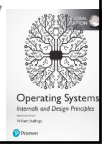
ΕΠΛ222: Λειτουργικά Συστήματα

(μετάφραση στα ελληνικά των διαφανειών του βιβλίου Operating Systems: Internals and Design Principles, 9/E, William Stallings)

Ενότητα 5 (Κεφάλαιο 6) Αδιέξοδο και Παρατεταμένη Στέρηση

Οι διαφάνειες αυτές έχουν συμπληρωματικό και επεξηγηματικό χαρακτήρα και σε καμία περίπτωση δεν υποκαθιστούν το βιβλίο

Γιώργος Α. Παπαδόπουλος
Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου



1

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

2

2

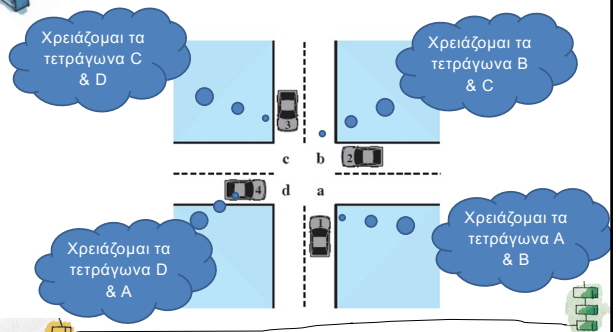
Αδιέξοδο

- Ένα σύνολο διεργασιών βρίσκεται σε αδιέξοδο (deadlock) αν κάθε διεργασία του συνόλου περιμένει ένα συμβάν που μόνο μία άλλη διεργασία του συνόλου μπορεί να προκαλέσει.
- Επειδή όλες οι διεργασίες του συνόλου περιμένουν αυτό το συμβάν, καμία δεν θα το προκαλέσει για να ενεργοποιηθεί κάποια άλλη διεργασία, και έτσι όλες οι διεργασίες θα περιμένουν για πάντα.
- Με τον όρο "συμβάν" συνήθως αναφερόμαστε στην προσπάθεια δέσμευσης πόρων (resources) δηλαδή μνήμη, περιφερειακές συσκευές, την ΚΜΕ, κλπ.
- Στη γενική περίπτωση, δεν υπάρχει αποδοτική λύση στο πρόβλημα αυτό.

3

3

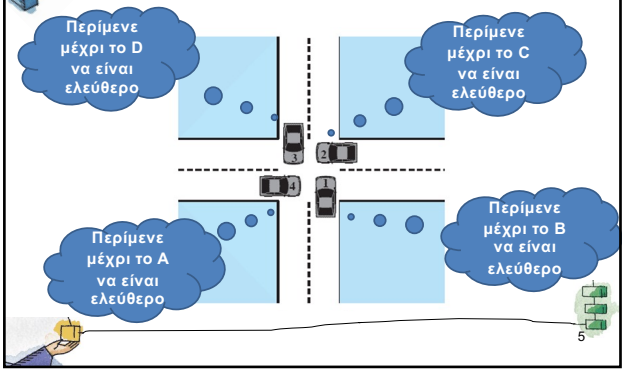
Πιθανό αδιέξοδο



4

4

Πραγματικό αδιέξοδο



5

5

Σενάριο με δύο διεργασίες P και Q

- Η κάθε μία από τις δύο διεργασίες χρειάζεται αποκλειστική πρόσβαση στους πόρους A και B για κάποιο χρονικό διάστημα.

Process P	Process Q
• • •	• • •
Get A	Get B
• • •	• • •
Get B	Get A
• • •	• • •
Release A	Release B
• • •	• • •
Release B	Release A
• • •	• • •

6

6

Διάγραμμα πιθανής εκτέλεσης των διεργασιών

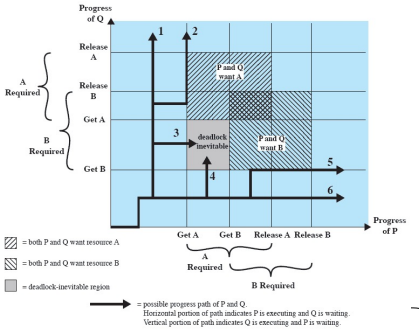


Figure 6.2 Example of Deadlock

7

Εναλλακτική λογική

- Ας υποθέσουμε ότι η διεργασία P δεν χρειάζεται την ίδια χρονική στιγμή και τους δύο πόρους, οπότε ο κώδικας διαφοροποιείται όπως φαίνεται παραδίπλα.

Process P	Process Q
• • •	• • •
Get A	Get B
• • •	• • •
Release A	Get A
• • •	• • •
Get B	Release B
• • •	• • •
Release B	Release A
• • •	• • •

8

Διάγραμμα πιθανής εκτέλεσης των διεργασιών

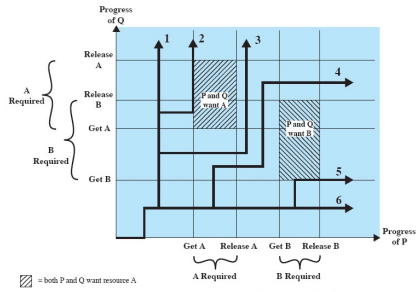


Figure 6.3 Example of No Deadlock [BACO03]

9

Είδη πόρων — 1

- Με βάση τον τρόπο δέσμευσης και αποδέσμευσής τους έχουμε δύο είδη πόρων, όπου μόνο στη δεύτερη κατηγορία δημιουργείται αδιέξοδο:
  - Τους προεκχωρήσιμους (preemptable) πόρους που μπορούν να αποδεσμευθούν από μία διεργασία χωρίς παρενέργειες, λ.χ. η μνήμη.
  - Τους μη προεκχωρήσιμους (nonpreemptable) πόρους που δεν μπορούν να αποδεσμευθούν από μία διεργασία χωρίς παρενέργειες, λ.χ. μία συσκευή (όπως ο εκτυπωτής κατά τη διάρκεια χρήσης του).

10

Είδη πόρων — 2

- Με βάση τη δυνατότητα επαναχρησιμοποίησής τους, έχουμε δύο είδη πόρων:
  - Επαναχρησιμοποιήσιμοι (reusable), π.χ. ΚΜΕ, κανάλια Ε/Ε, κύρια και περιφερειακή μνήμη, συσκευές και δομές δεδομένων όπως αρχεία, βάσεις δεδομένων και σημαφόροι.
  - Αναλώσιμοι (consumable), π.χ. μηνύματα, διακόπτες, σήματα, πληροφορίες σε προσωρινές θέσεις μνήμης, κλπ.
- Και για τις δύο κατηγορίες πόρων μπορεί να υπάρξει αδιέξοδο αλλά στη δεύτερη περίπτωση είναι συνήθως πιο δύσκολο να εντοπιστεί.

11

Παράδειγμα δημιουργίας αδιέξοδου κατά τη χρήση συσκευών

- Θεωρούμε δύο διεργασίες που συναγωνίζονται για αποκλειστική πρόσβαση σε ένα δίσκο D and μία συσκευή ανάγνωσης ταινιών T.
- Δημιουργείται αδιέξοδο αν η κάθε μία από τις διεργασίες δεσμεύει έναν από τους πόρους και μετά ζητεί και τον άλλο.

12

Κώδικας για το προηγούμενο σενάριο

Process P		Process Q	
Step	Action	Step	Action
p_0	Request (D)	q_0	Request (T)
p_1	Lock (D)	q_1	Lock (T)
p_2	Request (T)	q_2	Request (D)
p_3	Lock (T)	q_3	Lock (D)
p_4	Perform function	q_4	Perform function
p_5	Unlock (D)	q_5	Unlock (T)
p_6	Unlock (T)	q_6	Unlock (D)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

13

13

Παράδειγμα δημιουργίας αδιέξοδου κατά τη δέσμευση μνήμης

- Υποθέτουμε ότι η διαθέσιμη μνήμη είναι 200 Kbytes και δύο διεργασίες ζητούν μνήμη ως ακολούθως:

```
P1
...
Request 80 Kbytes;
...
Request 60 Kbytes;
```

```
P2
...
Request 70 Kbytes;
...
Request 80 Kbytes;
```

- Αν και οι δύο διεργασίες προσπαθήσουν να ικανοποιήσουν τη δεύτερη αίτησή τους, τότε θα δημιουργηθεί αδιέξοδο.

14

14

Παράδειγμα δημιουργίας αδιέξοδου με εμπλοκή αναλώσιμων πόρων

- Θεωρούμε δύο διεργασίες που εμπλέκονται στην ανταλλαγή μηνυμάτων (η εντολή `Receive` θέτει τη διεργασία υπό αναστολή αν δεν έχει αφιχθεί το μήνυμα).

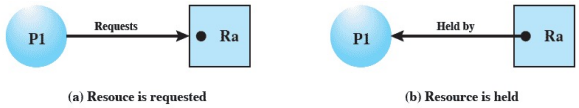
P1	P2
...	...
Receive (P2);	Receive (P1);
...	...
Send (P2, M1);	Send (P1, M2);

15

15

Γράφος εκχώρησης πόρων

- Ο γράφος εκχώρησης πόρων (resource allocation graph) είναι ένας κατευθυνόμενος γράφος που δείχνει την κατάσταση ζήτησης και εκχώρησης πόρων.



(a) Resouce is requested (b) Resource is held

16

16

Συνθήκες για **πιθανή** δημιουργία αδιέξοδου

- Η συνθήκη αμοιβαίου αποκλεισμού.
  - Κάθε πόρος είτε είναι δεσμευμένος από μία μόνο διεργασία είτε είναι διαθέσιμος.
- Η συνθήκη δέσμευσης και αναμονής (hold and wait).
  - Διεργασίες που δεσμεύουν πόρους που τους εκχωρήθηκαν νωρίτερα μπορούν να ζητούν και νέους.
- Η συνθήκη της μη προεκχώρησης.
  - Πόροι που έχουν εκχωρηθεί σε μία διεργασία μπορούν να απομακρυνθούν από τον έλεγχό της μόνο αν τους αποδεσμεύσει αυτή η ίδια.

17

17

Για να δημιουργηθεί πράγματι αδιέξοδο, χρειάζονται …

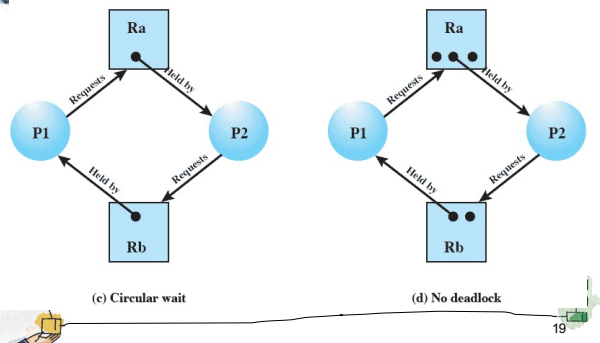
Όλες οι προηγούμενες συνθήκες, και:

- Η συνθήκη της κυκλικής αναμονής (circular wait).
  - Πρέπει να υπάρχει μία κυκλική αλυσίδα δύο ή περισσοτέρων διεργασιών, κάθε μία από τις οποίες περιμένει έναν πόρο που είναι δεσμευμένος από το επόμενο μέλος της αλυσίδας.
- Οι πρώτες τρεις συνθήκες είναι αναγκαίες αλλά όχι και ικανές για τη δημιουργία αδιέξοδου.
- Η τέταρτη συνθήκη είναι πιθανό αποτέλεσμα της ύπαρξης των τριών πρώτων. Η σημαντική διαφορά μεταξύ των τριών πρώτων και της τέταρτης συνθήκης είναι ότι οι τρεις πρώτες έχουν να κάνουν με αποφάσεις πολιτικής στη διαχείριση των διεργασιών ενώ η τέταρτη αποτελεί πιθανή εξέλιξη με βάση τον τρόπο αίτησης δέσμευσης, χρήσης και αποδέσμευσης των πόρων.

18

18

Παραδείγματα γράφων εκχώρησης πόρων — 1



19

Παραδείγματα γράφων εκχώρησης πόρων — 2

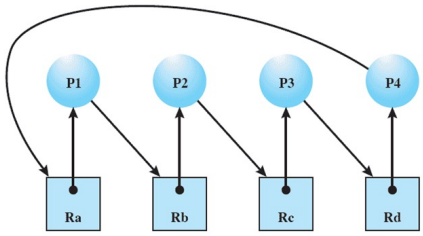


Figure 6.6 Resource Allocation Graph for Figure 6.1b

20

Αντιμετώπιση του αδιέξοδου

- Σε γενικές γραμμές υπάρχουν τέσσερις στρατηγικές αντιμετώπισης του προβλήματος του αδιέξοδου:
- Απλή αγνόηση του προβλήματος.
- Πρόληψη (prevention), με συστηματική αναίρεση μίας από τις τέσσερις αναγκαίες συνθήκες.
- Δυναμική αποφυγή (avoidance) με προσεκτική κατανομή πόρων.
- Ανίχνευση (detection) ύπαρξης αδιέξοδου και επανόρθωση (recovery).

21

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

22

Αγνόηση του προβλήματος

- Ή άλλως "ο αλγόριθμος της στρουθοκαμήλου". Απλά δεν κάνουμε τίποτα.
- Όχι και τόσο λανθασμένη προσέγγιση όσο φαίνεται εκ πρώτης όψης. Η λογική εδώ είναι ότι συνήθως το αδιέξοδο παρουσιάζεται τόσο σπάνια που είναι πιθανότερο μέσα σε μία συγκεκριμένη χρονική περίοδο το σύστημα να καταρρεύσει (crash) από ένα μηχανικό λάθος παρά από τη δημιουργία ενός αδιέξοδου.
- Οι περιορισμοί που πρέπει να επιβληθούν για την αντιμετώπιση του αδιέξοδου με κάποιον από τους άλλους τρεις τρόπους είναι τόσο σοβαροί και αρκετοί που οι περισσότεροι χρήστες θα προτιμούσαν να αντιμετώπιζαν μία μικρή πιθανότητα δημιουργίας αδιέξοδου μία φορά κάθε τόσο παρά να είναι υποχρεωμένοι να ακολουθούν συνεχώς περιοριστικούς κανόνες του τύπου μόνο ένα αρχείο ανοικτό ή μία διεργασία ενεργός ανά πάσα στιγμή.
- Το δίλημμα μεταξύ απουσίας περιορισμών και ορθότητας από τη μία πλευρά και επιβολής αυστηρών περιορισμών από την άλλη δεν είναι εύκολο να επιλυθεί και δεν είναι τυχαίο ότι μοντέρνα λειτουργικά συστήματα όπως το Unix έχουν υιοθετήσει αυτή την πολιτική της αγνόησης του προβλήματος.

23

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

24

Στρατηγική πρόληψης αδιέξοδου

- Βασίζεται στο σχεδιασμό του συστήματος με τέτοιο τρόπο που να μην υπάρχει πιθανότητα να δημιουργηθεί αδιέξοδο.
- Υπάρχουν δύο κυρίως τρόποι να γίνει αυτό:
  - Έμμεσος: αποφυγή ικανοποίησης και των τριών αναγκαίων συνθηκών την ίδια χρονική στιγμή.
  - Άμεσος: Αποφυγή κυκλικής αναμονής.

Συνθήκη του αμοιβαίου αποκλεισμού

- Στη γενική περίπτωση δεν μπορεί να αναιρεθεί διότι πολλοί πόροι (π.χ. ΚΜΕ, συσκευές) απαιτούν αποκλειστική χρήση τους από μία διεργασία.
- Ακόμα και πόροι όπως αρχεία, που δυνατόν να επιτρέπουν πρόσβαση από πολλαπλές διεργασίες για ανάγνωση, αν είναι να γίνει τροποποίηση των περιεχομένων τους, η πρόσβαση πρέπει να γίνει μέσω αμοιβαίου αποκλεισμού.

Συνθήκη δέσμευσης και αναμονής

- Μπορεί να αναιρεθεί με την εφαρμογή του περιορισμού ότι όλοι οι πόροι που χρειάζεται μία διεργασία θα πρέπει να ζητηθούν πριν αρχίσει η εκτέλεσή της.
- Αν οι πόροι είναι διαθέσιμοι, η διεργασία τούς δεσμεύει και εκτελείται, αλλιώς αν ένας ή περισσότεροι δεν είναι διαθέσιμοι, η διεργασία δεν δεσμεύει κανένα και περιμένει για μία χρονική περίοδο πριν ξαναδοκιμάσει.
- Η μέθοδος αυτή έχει τα εξής προβλήματα:
  - Δεν είναι απαραίτητο μία διεργασία να γνωρίζει από την αρχή όλες τις ανάγκες της σε πόρους.
  - Οι πόροι που εκχωρούνται σε μία διεργασία μένουν δεσμευμένοι από αυτήν περισσότερο χρόνο από όσο πραγματικά τους χρειάζεται και έτσι δεν είναι διαθέσιμοι στις υπόλοιπες διεργασίες.
  - Αν και θα μπορούσε να εκτελεσθεί ένα μέρος μίας διεργασίας με λιγότερους πόρους, εν τούτοις παραμένει ανενεργό περιμένοντας τη δέσμευση όλων των πόρων που θα χρειασθεί συνολικά η διεργασία.

Συνθήκη μη προεκχώρησης

- Μπορεί να αναιρεθεί σε κάποιες περιπτώσεις (νοουμένου ότι οι διεργασίες που εμπλέκονται έχουν την ίδια προτεραιότητα):
  - Αν μία διεργασία που δεσμεύει κάποιους πόρους ζητήσει και άλλον αλλά το Λ.Σ. δεν μπορεί να της τον δώσει, τότε η διεργασία πρέπει να αποδεσμεύσει όλους τους πόρους της και σε κάποια μελλοντική στιγμή να τους ξαναζητήσει μαζί με τον νέο πόρο.
  - Εναλλακτικά, αν μία διεργασία ζητήσει έναν πόρο που τον κατέχει άλλη διεργασία, το Λ.Σ. δύναται να αποσπάσει τον πόρο αυτό από την άλλη διεργασία και να τον εκχωρήσει στη διεργασία που τον ζήτησε.
- Μπορεί να εφαρμοσθεί μόνο για πόρους όπου είναι εύκολη η αποθήκευση/αλλαγή καταστάσεων (π.χ. ΚΜΕ) αλλά όχι λ.χ. για συσκευές Ε/Ε (π.χ. την ώρα που τυπώνεται ένα αρχείο, ο εκτυπωτής δίνεται σε άλλη διεργασία!).

Συνθήκη κυκλικής αναμονής

- Μπορεί να αναιρεθεί με την απαρίθμηση όλων των πόρων και την επιβολή του περιορισμού ότι οι πόροι εκχωρούνται στις διεργασίες με αριθμητική σειρά (ας πούμε από μικρότερο σε μεγαλύτερο).
- Είναι αδύνατο επομένως η διεργασία P1 να ζητάει κατά σειρά τους πόρους R3 και R4 και η διεργασία P2 τους πόρους R4 και R3, διότι δεν μπορούμε να έχουμε R4>R3. Έτσι αποφεύγονται κύκλοι.
- Υποφέρει από τα ίδια προβλήματα με τις προηγούμενες τεχνικές σε σχέση με την μη αποδοτική χρήση των πόρων του συστήματος.

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

Διαφορά μεταξύ πρόληψης και αποφυγής αδιέξοδου

- Η διαφορά μεταξύ πρόληψης και αποφυγής αδιέξοδου είναι ότι στην πρώτη περίπτωση ο στόχος είναι η παρεμπόδιση ικανοποίησης μίας τουλάχιστον από τις συνθήκες που το προκαλούν μέσω της επιβολής περιορισμών στις εκχωρήσεις πόρων ενώ στη δεύτερη περίπτωση επιτρέπεται η ικανοποίηση των πρώτων τριών συνθηκών αλλά εφ' όσον δεν οδηγούν στην ικανοποίηση της τέταρτης.
- Ο έλεγχος γίνεται δυναμικά, κάθε φορά που εκχωρείται ένας νέος πόρος, αντίθετα με τις μεθόδους στην περίπτωση της πρόληψης που είναι στατικοί.
- Επιτρέπει μεγαλύτερο βαθμό ταυτοχρονισμού από την στρατηγική της πρόληψης.
- Χρειάζεται όμως γνώση των μελλοντικών αναγκών των διεργασιών σε πόρους.

Δύο τεχνικές αποφυγής αδιέξοδου

- Αποφυγή έναρξης εκτέλεσης μίας διεργασίας που οι ανάγκες της σε πόρους μπορεί να προκαλέσει αδιέξοδο.
- Αποφυγή σταδιακής εκχώρησης ενός πόρου σε κάποια διεργασία αν αυτή η εκχώρηση μπορεί να προκαλέσει αδιέξοδο.

Αποφυγή έναρξης εκτέλεσης μίας διεργασίας

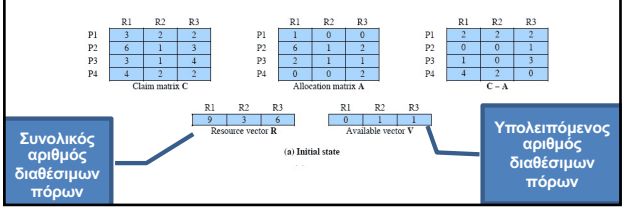
- Επιτρέπεται η έναρξη εκτέλεσης μίας διεργασίας αν το σύνολο των αναγκών της διεργασίας σε πόρους σε συνδυασμό και με τις ανάγκες των υπαρχουσών διεργασιών δεν μπορεί να προκαλέσει αδιέξοδο.
- Η στρατηγική αυτή κάθε άλλο παρά είναι βέλτιστη διότι υποθέτει ότι όλες οι εν' ενεργεία διεργασίες θα ζητήσουν ταυτόχρονα όλους τους πόρους που χρειάζονται.

Αποφυγή σταδιακής εκχώρησης πόρων

- Γνωστή και σαν αλγόριθμος του τραπεζίτη (banker's algorithm), προτάθηκε από τον Dijkstra το 1965. Ο αλγόριθμος του τραπεζίτη εξετάζει κάθε αίτηση για εκχώρηση ενός πόρου σε κάποια διεργασία και την ικανοποιεί μόνο αν αυτό οδηγεί σε ασφαλή κατάσταση.
- Η κατάσταση ενός συστήματος είναι η τρέχουσα εκχώρηση των πόρων του συστήματος στις διεργασίες που εκτελούνται σε αυτό.
- Μία κατάσταση είναι ασφαλής αν το σύστημα δεν βρίσκεται σε αδιέξοδο και υπάρχει τρόπος να ικανοποιηθούν όλες οι εκκρεμείς αιτήσεις με την εκτέλεση όλων των διεργασιών με κάποια σειρά.
- Στην αντίθετη περίπτωση η κατάσταση είναι ανασφαλής.
- Πρέπει να τονισθεί ότι μία ανασφαλής κατάσταση δεν αποτελεί αδιέξοδο *per se* αλλά απλά υποδεικνύει ότι *μπορεί* να οδηγήσει σε αδιέξοδο και επομένως πρέπει να αποφευχθεί.

Παράδειγμα εφαρμογής του αλγόριθμου του τραπεζίτη

- Ένα σύστημα αποτελείται από τέσσερεις διεργασίες και τρεις κατηγορίες πόρων.
- Είναι η κατάσταση, όπως φαίνεται από τους πίνακες, ασφαλής;



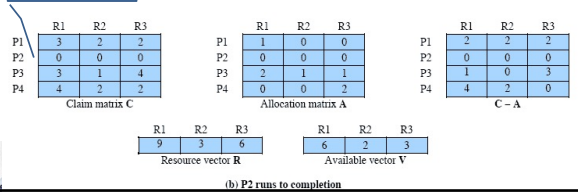
Καθορισμός ασφαλούς κατάστασης

- Με άλλα λόγια, μπορεί κάποια διεργασία να ολοκληρώσει την εκτέλεσή της και να αποδεσμεύσει κατόπιν όλους τους πόρους της για να χρησιμοποιηθούν από άλλες διεργασίες;
- Αυτό σημαίνει ότι η διαφορά μεταξύ της μέγιστης ανάγκης σε πόρους από την ποσότητα των πόρων που έχει δεσμεύσει κάποια διεργασία, πρέπει να είναι μικρότερη ή ίση με τον αριθμό των διαθέσιμων πόρων:
  - $C_{ij} - A_{ij} \leq V_j$, για όλα τα j
- Αυτό δεν είναι δυνατόν για την P1,
  - γιατί έχει μόνο 1 μονάδα από πόρους R1 και χρειάζεται 2 ακόμα μονάδες από πόρους R1, 2 μονάδες από πόρους R2 και 2 μονάδες από πόρους R3.

Εκχώρηση πόρων στην P2

- Αν εκχωρήσουμε μία μονάδα από πόρους R3 στη διεργασία P2, τότε αυτή έχει το μέγιστο αριθμό πόρων που μπορεί να χρειασθεί και μπορεί να ολοκληρώσει την εκτέλεσή της και να αποδεσμεύσει κατόπιν όλους τους πόρους της για να χρησιμοποιηθούν από άλλες διεργασίες.
- Μετά την P2, μπορεί να ολοκληρώσει την εκτέλεσή της και η P1.

Η P2 έχει ολοκληρώσει την εκτέλεσή της

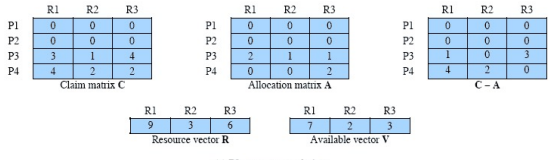


(b) P2 runs to completion

37

Μετά την ολοκλήρωση εκτέλεσης της P1

- Μπορεί να ολοκληρώσει την εκτέλεσή της τώρα και η P3.

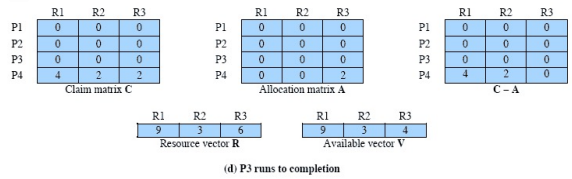


(c) P1 runs to completion

38

38

Μετά την ολοκλήρωση εκτέλεσης της P3



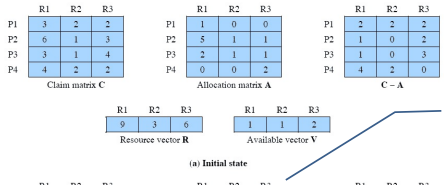
(d) P3 runs to completion

Επομένως η αρχική κατάσταση ήταν ασφαλής.

39

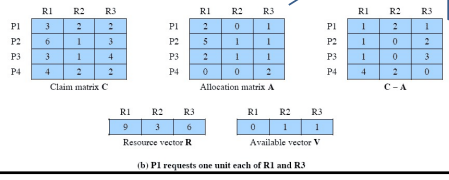
39

Καθορισμός ανασφαλούς κατάστασης



(a) Initial state

Αν η P1 ζητήσει 1 μονάδα πόρων R1 και 1 μονάδα πόρων R3, είναι η κατάσταση ασφαλής;



(b) P1 requests one unit each of R1 and R3

40

40

Εκχώρηση πόρων στην P1

- Η κατάσταση δεν είναι πλέον ασφαλής, γιατί η κάθε διεργασία θα χρειασθεί τουλάχιστον 1 ακόμα μονάδα πόρων R1 και δεν υπάρχουν διαθέσιμοι άλλοι τέτοιοι πόροι.
- Επομένως η αίτηση της P1 πρέπει να απορριφτεί και η διεργασία να τεθεί υπό αναστολή.
- Σημειωτέον ότι η κατάσταση δεν έχει φτάσει σε αδιέξοδο αλλά υπάρχει η *πιθανότητα* να καταλήξει σε αδιέξοδο.
- Θα μπορούσε φυσικά η P1 να αποδεσμεύσει τις μονάδες των πόρων R1 και R3 που κατέχει πριν τους ξαναζητήσει και σε αυτή την περίπτωση δεν έχουμε αδιέξοδο.
- Επομένως ο αλγόριθμος του τραπεζίτη δεν προβλέπει με σιγουριά το αδιέξοδο, αλλά αντιλαμβάνεται πότε δημιουργείται η πιθανότητα αυτό να συμβεί και δεν επιτρέπει το σύστημα να φτάσει σε μία τέτοια κατάσταση.

41

41

Στρατηγική αποφυγής αδιέξοδου

- Όταν μία διεργασία ζητήσει μία ομάδα από πόρους:
  - Θεώρησε ότι οι πόροι εκχωρήθηκαν στη διεργασία.
  - Ενημέρωσε την κατάσταση του συστήματος.
- Κατόπιν διαπίστωσε αν η νέα κατάσταση είναι ασφαλής:
  - Αν ναι, τότε ικανοποίησε την αίτηση της διεργασίας.
  - Αν όχι, τότε θέσε τη διεργασία υπό αναστολή μέχρις ότου είναι ασφαλές να ικανοποιηθεί η αίτησή της.

42

42

Λογική αποφυγής αδιέξοδου: δομές δεδομένων και αλγόριθμος εκχώρησης πόρων

```
struct state {                          /* global data structures */
   int resource[m];
   int available[m];
   int claim[n][m];
   int alloc[n][m];
}

/* resource allocation algorithm */

if (alloc [i,*] + request [*] > claim [i,*])
   < error >;                           /* total request > claim */
else if (request [*] > available [*])
   < suspend process >;
else {                                  /* simulate alloc */
      < define newstate by:
      alloc [i,*] = alloc [i,*] + request [*];
      available [*] = available [*] - request [*] >;
}

if (safe (newstate))
   < carry out allocation >;
else {
       < restore original state >;
       < suspend process >;
}
```


43

Λογική αποφυγής αδιέξοδου: έλεγχος για ασφαλή κατάσταση

```
boolean safe (state S)
{
 int currentavail[m];
 process rest[<number of processes>];
 currentavail = available;
 rest = {all processes};
 possible = true;
 while (possible) {
    <find a process Pk in rest such that
    claim [k,*] – alloc [k,*] <= currentavail;>
    if (found) { /* simulate execution of Pk */
       currentavail = currentavail + alloc [k,*];
       rest = rest - {Pk};
    }
    else possible = false;
 }
 return (rest == null);
}
```


44

Προτερήματα και μειονεκτήματα του αλγόριθμου του τραπεζίτη

- Είναι λιγότερο περιοριστικός από τις μεθόδους πρόληψης (δεν χρειάζεται καταφυγή σε αφαίρεση πόρων από μία διεργασία ή ακόμα και πρόωρο τερματισμό της) αλλά έχει και αυτός μειονεκτήματα:
  - Ο μέγιστος αριθμός πόρων που μία διεργασία τυχόν θα χρειασθεί πρέπει να είναι γνωστός από την αρχή.
  - Ο αριθμός των διαθέσιμων πόρων που υπάρχουν στο σύστημα πρέπει να είναι σταθερός.
  - Οι διεργασίες πρέπει να είναι ανεξάρτητες μεταξύ τους, ώστε η σειρά εκτέλεσής τους να μην δημιουργεί προβλήματα συγχρονισμού.
  - Οι διεργασίες πρέπει να αποδεσμεύσουν τους πόρους που κατέχουν πριν τερματίσουν.
- Σε μοντέρνα λειτουργικά συστήματα οι περιορισμοί αυτοί είναι πολύ αυστηροί και έτσι ελάχιστα συστήματα χρησιμοποιούν αυτήν τη μέθοδο.

45

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

46

Ανίχνευση αδιέξοδου

- Οι στρατηγικές πρόληψης ή αποφυγής του αδιέξοδου είναι πολύ συντηρητικές διότι επιβάλλουν περιορισμούς στον αριθμό και τρόπο εκχώρησης πόρων σε διεργασίες.
- Οι στρατηγικές της ανίχνευσης και επανόρθωσης δεν επιβάλλουν περιορισμούς αλλά περιοδικά ανιχνεύουν το σύστημα για να εντοπίσουν αδιέξοδο και αν χρειασθεί να κάνουν επανόρθωση.
- Η ανίχνευση μπορεί να γίνεται κάθε φορά που μία διεργασία ζητεί κάποιον πόρο αλλά σημειωτέον ότι η ανίχνευση σπαταλά χρόνο της ΚΜΕ.

47

Ένας απλός αλγόριθμος ανίχνευσης

- Χρησιμοποιούμε και πάλι έναν πίνακα καταγραφής των πόρων που εκχωρήθηκαν σε διεργασίες και ένα διάνυσμα για τους διαθέσιμους πόρους.
- Επίσης, ένα πίνακα αιτήσεων ***Q***:
  - Όπου ***Qij*** συμβολίζει το ποσόν των μονάδων από πόρο τύπου ***j*** που ζητεί η διεργασία ***i***.
- Στην αρχή καμία διεργασία δεν έχει περιέλθει σε αδιέξοδο.

48

Βήματα του αλγόριθμου — 1

1. Μάρκαρε κάθε διεργασία που έχει μία σειρά από 0 στον πίνακα καταγραφής των πόρων που εκχωρήθηκαν σε αυτή (μία διεργασία στην οποία δεν έχει εκχωρηθεί κανένας πόρος, δεν είναι δυνατόν να εμπλέκεται σε αδιέξοδο).
2. Δημιούργησε ένα νέο διάνυσμα **W** ίσο με το διάνυσμα για τους διαθέσιμους πόρους.
3. Βρες ένα δείκτη ***i*** έτσι ώστε η διεργασία ***i*** να μην είναι σημειωμένη και η ***i***th γραμμή του ***Q*** να είναι μικρότερη ή ίση με ***W***:
   - Δηλαδή $Q_{ik} \leq W_k$ για $1 \leq k \leq m$.
   - Αν δεν υπάρχει τέτοια γραμμή, σταμάτα.

49

49

Βήματα του αλγόριθμου — 2

4. Αν υπάρχει τέτοια γραμμή:
   - Μάρκαρε τη διεργασία ***i*** και πρόσθεσε την αντίστοιχη γραμμή από τον πίνακα καταγραφής πόρων στο διάνυσμα ***W***.
   - Δηλαδή, θέσε $W_k = W_k + A_{ik}$, για $1 \leq k \leq m$
5. Επέστρεψε στο βήμα 3.

- Αν στο τέλος υπάρχουν αμαρκάριστες διεργασίες, τότε υπάρχει αδιέξοδο στο οποίο έχουν περιέλθει οι διεργασίες αυτές που είναι αμαρκάριστες.

50

50

Η φιλοσοφία του αλγόριθμου

- Βρίσκει μία διεργασία της οποίας οι ανάγκες σε δέσμευση πόρων μπορούν να ικανοποιηθούν με τους διαθέσιμους πόρους.
- Θεωρεί ότι η διεργασία έχει δεσμεύσει αυτούς τους πόρους, ολοκληρώνει την εκτέλεσή της και μετά τους αποδεσμεύει.
- Ο αλγόριθμος μετά επαναλαμβάνει το ίδιο με άλλη διεργασία.
- Αν στο τέλος υπάρχουν διεργασίες για τις οποίες δεν είναι δυνατόν το ανωτέρω σενάριο, τότε αυτές έχουν περιέλθει σε αδιέξοδο.
- Σημειωτέον ότι ο αλγόριθμος αυτός δεν προλαμβάνει το αδιέξοδο, απλά το ανιχνεύει αν έχει δημιουργηθεί.

51

51

Σενάριο ανίχνευσης αδιέξοδου

Request matrix Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Allocation matrix A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Resource vector

R1	R2	R3	R4	R5
2	1	1	2	1

Available vector

R1	R2	R3	R4	R5
0	0	0	0	1

Figure 6.10 Example for Deadlock Detection

52

52

Εφαρμογή του αλγόριθμου στο προηγούμενο σενάριο

- Μάρκαρε τη διεργασία ***P4*** γιατί δεν έχει πόρους υπό την κατοχή της.
- Θέσε $W = (0,0,0,0,1)$.
- Οι αιτήσεις της ***P3*** είναι λιγότερες ή ίσες με ***W***, επομένως μάρκαρε την ***P3*** και θέσε:
  - $W = W + (0,0,0,1,0) = (0,0,0,1,1)$.
- Καμία άλλη μη μαρκαρισμένη διεργασία δεν έχει γραμμή στο ***Q*** που να είναι μικρότερη ή ίση με ***W***, επομένως ο αλγόριθμος τερματίζεται.
- Ο αλγόριθμος καταλήγει στο συμπέρασμα ότι οι διεργασίες ***P1*** και ***P2*** που είναι αμαρκάριστες έχουν περιέλθει σε αδιέξοδο.

53

53

Στρατηγικές επανόρθωσης — 1

- Απλά, εξάλειψη όλων των διεργασιών εμπλεκόμενων σε αδιέξοδο.
- Οπισθοδρόμηση σε κάποιο παλαιότερο σημείο όπου δεν υπήρχε αδιέξοδο και επανέναρξη εκτέλεσης των διεργασιών από εκείνο το σημείο.
  - Απαιτεί την περιοδική αποθήκευση της κατάστασης του συστήματος σε κάποια σημεία ελέγχου (checkpoints).
  - Η πιθανότητα επανεμφάνισης του αδιέξοδου υπάρχει αλλά ελαχιστοποιείται λόγω της μη προκαθορισμένης συμπεριφοράς των συντρεχουσών διεργασιών.

54

54

Στρατηγικές επανόρθωσης — 2

- Σταδιακή εξάλειψη των διεργασιών που εμπλέκονται σε αδιέξοδο. Η σειρά εξάλειψης καθορίζεται από τη λογική της ελαχιστοποίησης του κόστους επανεκτέλεσης μίας διεργασίας. Μετά την εξάλειψη μίας διεργασίας ο αλγόριθμος ανίχνευσης εκτελείται ξανά για να διαπιστωθεί αν το αδιέξοδο έχει εκλείψει.
- Σταδιακή προεκχώρηση πόρων σε διεργασίες που βρίσκονται σε αδιέξοδο με την απομάκρυνσή τους από άλλες διεργασίες. Η διεργασία που έχασε ένα πόρο πρέπει να επιστρέψει στην κατάσταση που ήταν πριν τη δέσμευση του πόρου αυτού. Και εδώ πρέπει να ορισθούν τα κριτήρια με βάση τα οποία επιλέγεται η διεργασία που θα χάσει ή θα πάρει πόρους.

55

Στρατηγικές επανόρθωσης — 3

- Για τους δύο τελευταίους τρόπους επανόρθωσης του αδιέξοδου, η ιδανική διεργασία που πρέπει να επιλεγεί για να της αφαιρεθούν πόροι ή ακόμα και για να τερματισθεί η εκτέλεσή της πρέπει να έχει:
  - καταναλώσει το λιγότερο χρόνο σε χρήση της ΚΜΕ,
  - τον περισσότερο χρόνο για αποπεράτωσή της,
  - παραγάγει τη λιγότερη ποσότητα αποτελεσμάτων,
  - δεσμεύσει τους λιγότερους σε αριθμό πόρους,
  - τη μικρότερη προτεραιότητα.

56

Πλεονεκτήματα και μειονεκτήματα της κάθε προσέγγισης

Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates online handling	•Inherent preemption losses

57

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

58

Ολοκληρωμένη πολιτική αντιμετώπισης του αδιέξοδου

- Μια και όλες οι προσεγγίσεις αντιμετώπισης του αδιέξοδου έχουν σοβαρά μειονεκτήματα, ίσως το καλύτερο που μπορεί να γίνει είναι να υιοθετούνται διαφορετικές στρατηγικές για διαφορετικές καταστάσεις. Μία τέτοια προσέγγιση είναι η εξής:
  - Ομαδοποίηση των πόρων σε κατηγορίες.
  - Χρήση απαρίθμησης μεταξύ των κατηγοριών για πρόληψη αδιέξοδου.
  - Χρήση της καλύτερης προσέγγισης για κάθε κατηγορία πόρων.
- Οι κατηγορίες πόρων που μπορούν να υπάρξουν (με την αντίστοιχη καλύτερη προσέγγιση) είναι οι εξής:
  - Περιφερειακή μνήμη (πρόληψη υπό την προϋπόθεση ότι το μέγιστο ποσό μνήμης που θα χρειασθεί είναι γνωστό).
  - Αρχεία, συσκευές ανάγνωσης ταινιών και άλλα είδη πόρων των οποίων η χρήση μπορεί να είναι γνωστή εκ των προτέρων (αποφυγή).
  - Κύρια μνήμη (πρόληψη με τη μέθοδο της προεκχώρησης).
  - Κανάλια Ε/Ε (πρόληψη με τη μέθοδο της απαρίθμησης).

59

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris και Windows.

60

Η φύση του προβλήματος

- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων αποτελεί κλασσική περίπτωση συγχρονισμού και αποφυγής αδιέξοδου. Προτάθηκε και επιλύθηκε από τον Dijkstra το 1965.
- Οι φιλόσοφοι σκέφτονται και με μη προκαθορισμένη σειρά και συχνότητα επιχειρούν να φάνε.
- Για το σκοπό αυτό χρειάζονται πρόσβαση σε δύο πιρούνια (το δεξί τους και το αριστερό τους).
- Δύο φιλόσοφοι δεν μπορούν να χρησιμοποιήσουν ταυτόχρονα το ίδιο πιρούνι (αμοιβαίος αποκλεισμός).
- Όλοι οι φιλόσοφοι πρέπει να μπορούν να φάνε κάποια στιγμή (αποφυγή αδιέξοδου ή (κυριολεκτικά!) παρατεταμένης στέρησης).

61

61

Οι συνδαιτυμόνες φιλόσοφοι: Σενάριο

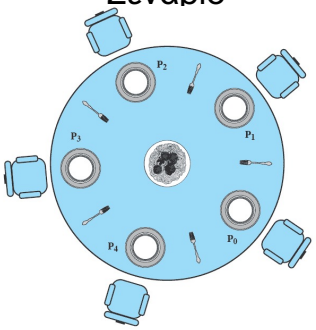


Figure 6.11 Dining Arrangement for Philosophers

62

62

Μία πρώτη λύση με σημαφόρους

```
void philosopher (int i)
{
 while (1) {
    think();
    semWait(fork[i]);
    semWait(fork [(i+1) mod 5]);
    eat();
    semSignal(fork [(i+1) mod 5]);
    semSignal(fork[i]);
 }
}
```

```
semaphore fork[5] = {1,1,1,1,1};
int i;

void main()
{
 parbegin
    philosopher(0);
    philosopher(1);
    philosopher(2);
    philosopher(3);
    philosopher(4);
 parend
}
```

- Ο κάθε φιλόσοφος πρώτα πιάνει το αριστερό του πιρούνι και μετά το δεξί.
- Αφού τελειώσει να τρώει, αφήνει τα πιρούνια κάτω.
- Η λύση μπορεί να οδηγήσει σε αδιέξοδο αν τύχει και οι πέντε φιλόσοφοι την ίδια στιγμή να πιάσουν το αριστερό τους πιρούνι.
- Τότε θα δημιουργηθεί μία κατάσταση όπου κανένας δεν θα μπορεί να πιάσει το δεξί του πιρούνι και επομένως να είναι σε θέση να φάει.

63

63

Αποφυγή του αδιέξοδου

```
void philosopher (int i)
{
 while (1) {
    think();
    semWait(room);
    semWait(fork[i]);
    semWait(fork [(i+1) mod 5]);
    eat();
    semSignal(fork [(i+1) mod 5]);
    semSignal(fork[i]);
    semSignal(room);
 }
}
```

```
semaphore fork[5] = {1,1,1,1,1};
          room = 4;
int i;

void main()
{
 parbegin
    philosopher(0);
    philosopher(1);
    philosopher(2);
    philosopher(3);
    philosopher(4);
 parend
}
```

- Με τη χρήση ενός γενικού σημαφόρου με τιμή μία μικρότερη από τον αριθμό των φιλοσόφων, τουλάχιστον ένας φιλόσοφος θα μπορέσει να έχει πρόσβαση και στα δύο του πιρούνια και επομένως να φάει.
- Η λύση αυτή δεν υποφέρει από αδιέξοδο και αν ο σημαφόρος `room` είναι ισχυρός δεν υποφέρει και από παρατεταμένη στέρηση.

64

64

Λύση με παρακολουθητή — 1

```
monitor dining_controller;
{
 cond ForkReady[5]; /* condition variable for synchronization */
 boolean fork[5] = {true}; /* availability status of each fork */

 void get_forks(int pid) /* pid is the philosopher id number */
 {
  int left = pid;
  int right = (++pid) % 5;

  /*grant the left fork*/
  if (!fork(left)
     cwait(ForkReady[left]); /* queue on condition variable */
  fork(left) = false;

  /*grant the right fork*/
  if (!fork(right)
     cwait(ForkReady(right); /* queue on condition variable */
  fork(right) = false:
 }
```

65

65

Λύση με παρακολουθητή — 2

```
void release_forks(int pid)
{
 int left = pid;
 int right = (++pid) % 5;

 /*release the left fork*/
 if (empty(ForkReady[left]) /*no one is waiting for this fork */
    fork(left) = true;
 else /* awaken a process waiting on this fork */
     csignal(ForkReady[left]);

 /*release the right fork*/
 if (empty(ForkReady[right]) /*no one is waiting for this fork */
    fork(right) = true;
 else /* awaken a process waiting on this fork */
      csignal(ForkReady[right]);
 }
}
/* end of monitor code */
```

66

66

Λύση με παρακολουθητή — 3

```
/* main program */

void philosopher(i) /* the five philosopher clients */
{
 while (1)
 {
  think();
  dining_controller.get_forks(i); /* client requests two forks */
  eat();
  dining_controller.release_forks(i); /* client releases forks */
 }
}

void main()
{
 parbegin
    philosopher(0);
    philosopher(1);
    philosopher(2);
    philosopher(3);
    philosopher(4);
 parend
}
```


67

Επεξήγηση της λύσης

- Χρησιμοποιούνται πέντε μεταβλητές συνθήκης, μία για κάθε πιρούνι, για το συντονισμό της πρόσβασης σε αυτά από τους φιλοσόφους.
- Επιπλέον υπάρχει και ένας αντίστοιχος αριθμός μεταβλητών boolean που δηλώνει αν ένα πιρούνι είναι διαθέσιμο ή όχι.
- Η συνάρτηση `get_forks` χρησιμοποιείται από ένα φιλόσοφο για να εκτελέσει τη διαδικασία δέσμευσης των πιρουνιών που χρειάζεται για να φάει.
- Αν κάποιο πιρούνι δεν είναι διαθέσιμο, ο φιλόσοφος περιμένει στην ουρά της αντίστοιχης μεταβλητής συνθήκης.
- Συμμετρικά, η συνάρτηση `release_forks` χρησιμοποιείται από ένα φιλόσοφο για να αποδεσμεύσει τα πιρούνια που κατέχει.
- Η λύση είναι παρόμοια με την πρώτη λύση με σημαφόρους, αλλά δεν αντιμετωπίζει πρόβλημα αδιέξοδου. Γιατί;

68

Περιεχόμενα

- Βασικές αρχές δημιουργίας αδιέξοδου.
- Αντιμετώπιση του αδιέξοδου:
  - Αγνόηση.
  - Πρόληψη.
  - Αποφυγή.
  - Ανίχνευση και επανόρθωση.
  - Μία ολοκληρωμένη στρατηγική αντιμετώπισης του αδιέξοδου.
- Το πρόβλημα των συνδαιτυμόνων φιλοσόφων.
- Μηχανισμοί ταυτοχρονισμού στα Λ.Σ. UNIX, Linux, Solaris, Windows και Android.

69

Μηχανισμοί ταυτοχρονισμού στο UNIX

- Μερικοί από τους μηχανισμούς επικοινωνίας και συντονισμού μεταξύ των διεργασιών στο UNIX είναι οι ακόλουθοι:
  - Ενδοαγωγοί (pipes).
  - Μηνύματα.
  - Κοινή μνήμη.
  - Σημαφόροι.
  - Σήματα.

70

Ενδοαγωγοί

- Είναι κυκλικής μορφής προσωρινή μνήμη που επιτρέπει σε δύο διεργασίες να επικοινωνήσουν μεταξύ τους με το μοντέλο του παραγωγού-καταναλωτή, κάνοντας χρήση μίας ουράς τύπου πρώτο-εισερχόμενο-πρώτο-εξερχόμενο.
- Είναι δύο ειδών:
  - Επώνυμοι: μόνο άμεσα σχετιζόμενες μεταξύ τους διεργασίες μπορούν να τους χρησιμοποιήσουν.
  - Ανώνυμοι: οποιαδήποτε διεργασία μπορεί να τους χρησιμοποιήσει.

71

Μηνύματα

- Είναι μία ομάδα από bytes συγκεκριμένου τύπου.
- Το UNIX παρέχει στις διεργασίες τις εντολές του συστήματος `msgsnd` και `msgrcv` για την ανταλλαγή μηνυμάτων.
- Μία ουρά μηνυμάτων συσχετίζεται με κάθε διεργασία που λειτουργεί ως γραμματοκιβώτιο.

72

Κοινή μνήμη

- Είναι ένας κοινός χώρος ιδεατής μνήμης που διαμοιράζεται από πολλαπλές διεργασίες.
- Η πρόσβαση στο χώρο αυτό από τις διεργασίες γίνεται είτε μόνο για γράψιμο είτε μόνο για διάβασμα, κάτι που καθορίζεται ξεχωριστά για κάθε διεργασία.
- Οι διεργασίες πρέπει οι ίδιες να υλοποιήσουν τον αμοιβαίο αποκλεισμό.

Σημαφόροι

- Το SVR4 χρησιμοποιεί μία γενικευμένη μορφή των εντολών `semWait` και `semSignal`.
- Με κάθε σημαφόρο υπάρχει μία ουρά από διεργασίες υπό αναστολή στο σημαφόρο αυτό.

Σήματα

- Είναι ένας μηχανισμός λογισμικού που ενημερώνει μία διεργασία για την ύπαρξη ενός γεγονότος.
- Είναι παρόμοιος με ένα διακόπτη αλλά δεν υπόκειται σε προτεραιότητες, δηλαδή αν υπάρχουν περισσότερα από ένα σήματα για μία διεργασία, η διεργασία θα τα δει με τυχαία σειρά.
- Τα σήματα στέλνονται μεταξύ των διεργασιών ή δημιουργούνται από τον πυρήνα.
- Το κάθε σήμα αναπαρίσταται από ένα bit (σήματα του ιδίου τύπου δεν μπορούν να συνυπάρχουν) και η αποστολή του σε μία διεργασία επιτελείται με την ενημέρωση κάποιου πεδίου στον πίνακα διεργασιών.
- Μία διεργασία επεξεργάζεται ένα σήμα που της έχει σταλεί είτε μόλις ενεργοποιηθεί (αν βρίσκεται υπό αναστολή) είτε μόλις ολοκληρώσει την εκτέλεση μίας εντολής.
- Σαν αποτέλεσμα μπορεί να τερματίσει την εκτέλεσή της, να προβεί σε οποιαδήποτε άλλη ενέργεια ή ακόμα και να αγνοήσει το σήμα.

Τα σήματα στο UNIX SVR4

Value	Name	Description
01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work
02	SIGINT	Interrupt
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump
04	SIGILL	Illegal instruction
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing
06	SIGIOT	IOT instruction
07	SIGEMT	EMT instruction
08	SIGFPE	Floating-point exception
09	SIGKILL	Kill; terminate process
10	SIGBUS	Bus error
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space
12	SIGSYS	Bad argument to system call
13	SIGPIPE	Write on a pipe that has no readers attached to it
14	SIGALRM	Alarm clock; issued when a process wishes to receive a signal after a period of time
15	SIGTERM	Software termination
16	SIGUSR1	User-defined signal 1
17	SIGUSR2	User-defined signal 2
18	SIGCHLD	Death of a child
19	SIGPWR	Power failure

Μηχανισμοί ταυτοχρονισμού στο Linux

- Επιπλέον των μηχανισμών που έχει το κλασσικό UNIX, υπάρχουν επίσης:
  - Ατομικές εντολές.
  - Περιστρεφόμενες κλειδαριές (spinlocks).
  - Σημαφόροι (κατά τι διαφορετικοί από το SVR4).
  - Φράγματα (barriers).

Ατομικές εντολές

- Εκτελούνται χωρίς διακοπή ή παρεμβολή από οποιονδήποτε.
- Είναι δύο ειδών:
  - Αυτές που εκτελούνται σε ακέραιους αριθμούς.
  - Αυτές που εκτελούνται σε bits.

Ατομικές εντολές για ακέραιους στο Linux

Table 6.3 Linux Atomic Operations

Atomic Integer Operations	
ATOMIC_INIT (int i)	At declaration: initialize an atomic_t to i
int atomic_read(atomic_t *v)	Read integer value of v
void atomic_set(atomic_t *v, int i)	Set the value of v to integer i
void atomic_add(int i, atomic_t *v)	Add i to v
void atomic_sub(int i, atomic_t *v)	Subtract i from v
void atomic_inc(atomic_t *v)	Add 1 to v
void atomic_dec(atomic_t *v)	Subtract 1 from v
int atomic_sub_and_test(int i, atomic_t *v)	Subtract i from v; return 1 if the result is zero; return 0 otherwise
int atomic_add_negative(int i, atomic_t *v)	Add i to v; return 1 if the result is negative; return 0 otherwise (used for implementing semaphores)
int atomic_dec_and_test(atomic_t *v)	Subtract 1 from v; return 1 if the result is zero; return 0 otherwise
int atomic_inc_and_test(atomic_t *v)	Add 1 to v; return 1 if the result is zero; return 0 otherwise

79

79

Ατομικές εντολές για bits στο Linux

Atomic Bitmap Operations	
void set_bit(int nr, void *addr)	Set bit nr in the bitmap pointed to by addr
void clear_bit(int nr, void *addr)	Clear bit nr in the bitmap pointed to by addr
void change_bit(int nr, void *addr)	Invert bit nr in the bitmap pointed to by addr
int test_and_set_bit(int nr, void *addr)	Set bit nr in the bitmap pointed to by addr; return the old bit value
int test_and_clear_bit(int nr, void *addr)	Clear bit nr in the bitmap pointed to by addr; return the old bit value
int test_and_change_bit(int nr, void *addr)	Invert bit nr in the bitmap pointed to by addr; return the old bit value
int test_bit(int nr, void *addr)	Return the value of bit nr in the bitmap pointed to by addr

80

80

Περιστρεφόμενες κλειδαριές

- Μόνο ένα νήμα μπορεί ανά πάσα στιγμή να έχει τον έλεγχο μίας περιστρεφόμενης κλειδαριάς.
- Οποιοδήποτε άλλο νήμα προσπαθήσει να έχει πρόσβαση σε μία κλειδαριά που είναι ήδη δεσμευμένη θα περιστρέφεται μέχρις ότου η κλειδαριά είναι διαθέσιμη.
- Στην ουσία, η κλειδαριά είναι μία θέση μνήμης, τα περιεχόμενα της οποίας ελέγχονται από μία διεργασία η οποία θέλει να εισέλθει στο κρίσιμο τμήμα της.
  - Αν η τιμή της θέσης μνήμης είναι 0, η διεργασία την κάνει 1 και εισέρχεται στο κρίσιμο τμήμα της.
  - Αν η τιμή δεν είναι 0, η διεργασία συνέχεια ελέγχει τη θέση μνήμης μέχρις ότου αυτή μηδενιστεί.
- Ο μηχανισμός αυτός είναι εύκολος στην υλοποίηση αλλά υποφέρει από το κόστος της ενεργούς αναμονής και είναι κατάλληλος μόνο για περιπτώσεις που εκτιμάται ότι ο χρόνος αναμονής μίας διεργασίας είναι μικρός (όχι περισσότερο από δύο εναλλαγές διεργασιών).

81

81

Κλειδαριές στο Linux

void spin_lock(spinlock_t *lock)	Acquires the specified lock, spinning if needed until it is available
void spin_lock_irq(spinlock_t *lock)	Like spin_lock, but also disables interrupts on the local processor
void spin_lock_irqsave(spinlock_t *lock, unsigned long flags)	Like spin_lock_irq, but also saves the current interrupt state in flags
void spin_lock_bh(spinlock_t *lock)	Like spin_lock, but also disables the execution of all bottom halves
void spin_unlock(spinlock_t *lock)	Releases given lock
void spin_unlock_irq(spinlock_t *lock)	Releases given lock and enables local interrupts
void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags)	Releases given lock and restores local interrupts to given previous state
void spin_unlock_bh(spinlock_t *lock)	Releases given lock and enables bottom halves
void spin_lock_init(spinlock_t *lock)	Initializes given spinlock
int spin_trylock(spinlock_t *lock)	Tries to acquire specified lock; returns nonzero if lock is currently held and zero otherwise
int spin_is_locked(spinlock_t *lock)	Returns nonzero if lock is currently held and zero otherwise

82

82

Σημαφόροι

- Παρόμοιοι με αυτούς του SVR4, αλλά η υλοποίησή τους σε επίπεδο πυρήνα γίνεται με τρόπο που εξυπηρετεί τον πυρήνα και επιτρέπει στον κώδικά του να καλεί σημαφόρους πυρήνα.
- Αυτοί οι σημαφόροι δεν είναι διαθέσιμοι στο επίπεδο του χρήστη, υλοποιούνται ως συναρτήσεις του πυρήνα και επομένως είναι πιο αποδοτικοί από τους σημαφόρους σε επίπεδο χρήστη.
- Το Linux παρέχει τρεις τύπους σημαφόρων σε επίπεδο πυρήνα:
  - Δυαδικούς.
  - Γενικούς.
  - Αναγνώστες-Εγγραφείς.

83

83

Σημαφόροι στο Linux

Traditional Semaphores	
void sema_init(struct semaphore *sem, int count)	Initializes the dynamically created semaphore to the given count
void init_MUTEX(struct semaphore *sem)	Initializes the dynamically created semaphore with a count of 1 (initially unlocked)
void init_MUTEX_LOCKED(struct semaphore *sem)	Initializes the dynamically created semaphore with a count of 0 (initially locked)
void down(struct semaphore *sem)	Attempts to acquire the given semaphore, entering uninterruptible sleep if semaphore is unavailable
int down_interruptible(struct semaphore *sem)	Attempts to acquire the given semaphore, entering interruptible sleep if semaphore is unavailable; returns -EINTR value if a signal other than the result of an up operation is received.
int down_trylock(struct semaphore *sem)	Attempts to acquire the given semaphore, and returns a nonzero value if semaphore is unavailable
void up(struct semaphore *sem)	Releases the given semaphore
Reader-Writer Semaphores	
void init_rwsem(struct rw_semaphore, *rwsem)	Initalizes the dynamically created semaphore with a count of 1
void down_read(struct rw_semaphore, *rwsem)	Down operation for readers
void up_read(struct rw_semaphore, *rwsem)	Up operation for readers
void down_write(struct rw_semaphore, *rwsem)	Down operation for writers
void up_write(struct rw_semaphore, *rwsem)	Up operation for writers

84

84

Φράγματα

- Σκοπός τους είναι να επιβάλουν μία συγκεκριμένη σειρά στην εκτέλεση μίας ομάδας εντολών.

Table 6.6 Linux Memory Barrier Operations

`rmb()`	Prevents loads from being reordered across the barrier
`wmb()`	Prevents stores from being reordered across the barrier
`mb()`	Prevents loads and stores from being reordered across the barrier
`Barrier()`	Prevents the compiler from reordering loads or stores across the barrier
`smp_rmb()`	On SMP, provides a `rmb()` and on UP provides a `barrier()`
`smp_wmb()`	On SMP, provides a `wmb()` and on UP provides a `barrier()`
`smp_mb()`	On SMP, provides a `mb()` and on UP provides a `barrier()`

SMP = symmetric multiprocessor
UP = uniprocessor

85

Μηχανισμοί ταυτοχρονισμού στο Solaris

- Επιπλέον των μηχανισμών που έχει το SVR4, υπάρχουν επίσης:
  - Κλειδαριές για αμοιβαίο αποκλεισμό (mutex).
  - Σημαφόροι.
  - Κλειδαριές τύπου πολλαπλοί αναγνώστες – ένας εγγραφέας.
  - Μεταβλητές συνθήκης.

86

86

Δομές δεδομένων για ταυτοχρονισμό στο Solaris

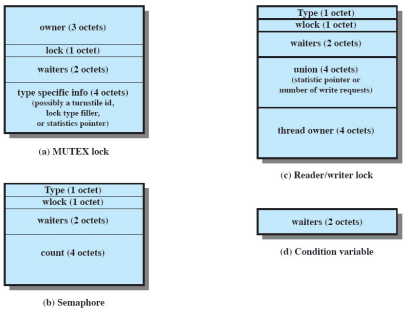


Figure 6.15 Solaris Synchronization Data Structures

87

87

Κλειδαριές για αμοιβαίο αποκλεισμό

- Μια κλειδαριά τύπου mutex χρησιμοποιείται για να υλοποιήσει τον αμοιβαίο αποκλεισμό στην πρόσβαση από πολλαπλές διεργασίες σε κάποιον πόρο.
- Μόνο η διεργασία που έχει κλειδώσει μία κλειδαριά mutex μπορεί να τη ξεκλειδώσει.

88

88

Σημαφόροι και κλειδαριές αναγνωστών-εγγραφέων

- Το Solaris υποστηρίζει γενικούς σημαφόρους.
- Υποστηρίζει επίσης κλειδαριές τύπου αναγνωστών-εγγραφέων που επιτρέπουν σε πολλαπλές διεργασίες να έχουν ταυτόχρονη πρόσβαση για διάβασμα στον ίδιο πόρο αλλά και σε μία διεργασία να έχει αποκλειστική πρόσβαση σε αυτόν τον πόρο για γράψιμο.

89

89

Μεταβλητές συνθήκης

- Μία μεταβλητή συνθήκης χρησιμοποιείται για την αναμονή μίας διεργασίας υπό αναστολή στην ουρά της μεταβλητής αυτής μέχρις ότου ικανοποιηθεί η συνθήκη της μεταβλητής.
- Οι μεταβλητές συνθήκης χρησιμοποιούνται σε συνάρτηση με κλειδαριές τύπου mutex.

90

90

Μηχανισμοί ταυτοχρονισμού στα Windows

- Τα Windows παρέχουν μηχανισμούς ταυτοχρονισμού των νημάτων οι οποίοι είναι ενσωματωμένοι στην αρχιτεκτονική τους:
  - Αντικείμενα διευθυντικής διεκπεραίωσης (executive dispatcher objects).
  - Κρίσιμα τμήματα.
  - Κλειδαριές τύπου αναγνωστών-εγγραφέων.
  - Μεταβλητές συνθήκης.

91

91

Συναρτήσεις `wait`

- Οι συναρτήσεις `wait` είναι σημαντικές στην υλοποίηση των αντικειμένων διευθυντικής διεκπεραίωσης.
- Επιτρέπουν σε ένα νήμα που εκτελεί μία συνάρτηση `wait` να αναστείλει την εκτέλεσή του μέχρις ότου κάποια κριτήρια σχετιζόμενα με αυτή τη συνάρτηση έχουν ικανοποιηθεί.
- Όταν ένα νήμα εκτελεί μία συνάρτηση `wait`, αν τα κριτήρια σχετιζόμενα με αυτή τη συνάρτηση δεν έχουν ικανοποιηθεί, το νήμα τίθεται υπό αναστολή.
- Όσο βρίσκεται υπό αναστολή δεν κάνει χρήση της ΚΜΕ.

92

92

Αντικείμενα διευθυντικής διεκπεραίωσης

Object Type	Definition	Set to Signaled State When	Effect on Waiting Threads
Notification Event	An announcement that a system event has occurred	Thread sets the event	All released
Synchronization event	An announcement that a system event has occurred	Thread sets the event	One thread released
Mutex	A mechanism that provides mutual exclusion capabilities; equivalent to a binary semaphore	Owning thread or other thread releases the mutex	One thread released
Semaphore	A counter that regulates the number of threads that can use a resource	Semaphore count drops to zero	All released
Waitable timer	A counter that records the passage of time	Set time arrives or time interval expires	All released
File	An instance of an opened file or I/O device	I/O operation completes	All released
Process	A program invocation, including the address space and resources required to run the program	Last thread terminates	All released
Thread	An executable entity within a process	Thread terminates	All released

Note: Shaded rows correspond to objects that exist for the sole purpose of synchronization.

93

93

Κρίσιμα τμήματα

- Είναι ένας παρόμοιος μηχανισμός με αυτόν των κλειδαριών τύπου mutex.
  - Με τη διαφορά ότι ένα κρίσιμο τμήμα μπορεί να χρησιμοποιηθεί μόνο από νήματα που ανήκουν στην ίδια διεργασία.
- Είναι πιο αποδοτικός μηχανισμός από άλλους και χρησιμοποιεί ένα ανεπτυγμένο αλγόριθμο υλοποίησης αμοιβαίου αποκλεισμού που καλύπτει και την περίπτωση συστημάτων με πολλαπλούς επεξεργαστές.

94

94

Κλειδαριές τύπου αναγνωστών-εγγραφέων

- Τα Windows Vista υποστηρίζουν ένα τέτοιο μηχανισμό.
- Κάνει χρήση μόνο μίας θέσης μνήμης και είναι εύκολος στην υλοποίηση.

95

95

Μεταβλητές συνθήκης

- Τα Windows Vista υποστηρίζουν επίσης μεταβλητές συνθήκης.
- Μία μεταβλητή συνθήκης δηλώνεται με την εντολή `CONDITION_VARIABLE` και αρχικοποιείται με την εντολή `InitializeConditionvariable`.
- Μπορούν να χρησιμοποιηθούν σε συνάρτηση με κρίσιμα τμήματα ή κλειδαριές αναγνωστών-εγγραφέων.
- Ανάλογα με την περίπτωση, οι εντολές που χρησιμοποιούνται για να θέσουν υπό αναστολή και να ενεργοποιήσουν αντίστοιχα μία διεργασία είναι:
  - `SleepConditionVariableCS`.
  - `SleepConditionVariableSRW`.
  - `WakeConditionVariableCS`.
  - `WakeConditionVariableSRW`.

96

96

Σύγκριση Windows με Linux — 1

Windows	Linux
Common synchronization primitives, such as semaphores, mutexes, spinlocks, timers, based on an underlying wait/signal mechanism	Common synchronization primitives, such as semaphores, mutexes, spinlocks, timers, based on an underlying sleep/wakeup mechanism
Many kernel objects are also dispatcher objects, meaning that threads can synchronize with them using a common event mechanism, available at user-mode. Process and thread termination are events, I/O completion is an event	
Threads can wait on multiple dispatcher objects at the same time	Processes can use the select() system call to wait on I/O from up to 64 file descriptors
User-mode reader/writer locks and condition variables are supported	User-mode reader/writer locks and condition variables are supported
Many hardware atomic operations, such as atomic increment/decrement, and compare-and-swap, are supported	Many hardware atomic operations, such as atomic increment/decrement, and compare-and-swap, are supported
A non-locking atomic LIFO queue, called an SLIST, is supported using compare-and-swap; widely used in the OS and also available to user programs	

97

Σύγκριση Windows με Linux — 2

Windows	Linux
A large variety of synchronization mechanisms exist within the kernel to improve scalability. Many are based on simple compare-and-swap mechanisms, such as push-locks and fast references of objects	
Named pipes, and sockets support remote procedure calls (RPCs), as does an efficient Local Procedure Call mechanism (ALPC), used within a local system. ALPC is used heavily for communicating between clients and local services	Named pipes, and sockets support remote procedure calls (RPCs)
Asynchronous Procedure Calls (APCs) are used heavily within the kernel to get threads to act upon themselves (e.g. termination and I/O completion use APCs since these operations are easier to implement in the context of a thread rather than cross-thread). APCs are also available for user-mode, but user-mode APCs are only delivered when a user-mode thread blocks in the kernel	Unix supports a general signal mechanism for communication between processes. Signals are modeled on hardware interrupts and can be delivered at any time that they are not blocked by the receiving process; like with hardware interrupts, signal semantics are complicated by multi-threading
Hardware support for deferring interrupt processing until the interrupt level has dropped is provided by the Deferred Procedure Call (DPC) control object	Uses tasklets to defer interrupt processing until the interrupt level has dropped

98

Επικοινωνία των διεργασιών στο Android

- Το Android προσθέτει στον πυρήνα μία νέα δυνατότητα που ονομάζεται σύνδεση (binder).
  - Είναι ένας μηχανισμός RPC που είναι αποδοτικός στη χρήση επεξεργαστή και μνήμης.
  - Χρησιμοποιείται επίσης για να συντονίζει την επικοινωνία μεταξύ δύο διεργασιών, η κάθε μία από τις οποίες δύναται να εκτελείται σε διαφορετική ιδεατή μηχανή.
  - Μία διεργασία που θέλει να επικοινωνήσει με άλλη διεργασία, στέλνει το μήνυμα στο binder το οποίο προωθεί το μήνυμα στην άλλη διεργασία και με τον ίδιο τρόπο λαμβάνεται πίσω η απάντηση της δεύτερης διεργασίας προς την πρώτη.
- Η υλοποίηση του binder γίνεται μέσω της συνάρτησης του συστήματος `ioctl` που είναι μία γενικής χρήσης ρουτίνα για επικοινωνία Ε/Ε με διάφορες συσκευές.

99

Λειτουργία του binder

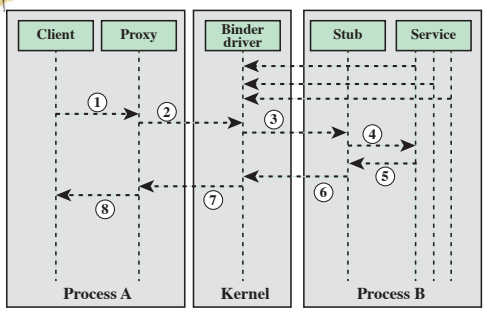


Figure 6.16 Binder Operation

100

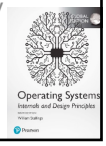
ΕΠΛ222: Λειτουργικά Συστήματα

(μετάφραση στα ελληνικά των διαφανειών του βιβλίου Operating Systems: Internals and Design Principles, 9/E, William Stallings)

Τέλος Ενότητας 5

Οι διαφάνειες αυτές έχουν συμπληρωματικό και επεξηγηματικό χαρακτήρα και σε καμία περίπτωση δεν υποκαθιστούν το βιβλίο

Γιώργος Α. Παπαδόπουλος
Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου



101