ΕΝΟΤΗΤΑ 4

ΑΔΙΕΞΟΔΟ

Περιεχόμενα

1. Αδιέξοδο

- Ένα σύνολο διεργασιών βρίσκεται σε *αδιέξοδο* (deadlock) αν κάθε διεργασία του συνόλου περιμένει ένα γεγονός που μόνο μία άλλη διεργασία του συνόλου μπορεί να προκαλέσει.

- Επειδή όλες οι διεργασίες του συνόλου περιμένουν, καμιά δεν θα προκαλέσει ποτέ κάποιο γεγονός για να ενεργοποιηθεί κάποια άλλη, και έτσι όλες οι διεργασίες θα περιμένουν για πάντα.

- Με τον όρο "γεγονός" συνήθως αναφερόμαστε σε δέσμευση πόρων (resources) δηλαδή μνήμη, περιφερειακές συσκευές, την ΚΜΕ, κλπ.

- Ο αριθμός και είδος των διεργασιών και πόρων που εμπλέκονται σε αδιέξοδο δεν έχουν μεγάλη σημασία. Όμως οι διάφοροι τύποι πόρων αναφορικά με τη δυνατότητα επαναχρησιμοποίησής τους και τον τρόπο δέσμευσης και αποδέσμευσής τους από τις διεργασίες επηρεάζουν τη δημιουργία και αντιμετώπιση του αδιέξοδου.

2. Είδη πόρων

- 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).

- Με βάση τη δυνατότητα επαναχρησιμοποίησης έχουμε δύο είδη πόρων:
  - επαναχρησιμοποιήσιμες (reusable), π.χ. μνήμη, ΚΜΕ, κανάλια εισ/εξ,
  - αναλώσιμες (consumable), π.χ. μηνύματα, διακόπτες, κλπ.

  Και για τις δύο κατηγορίες πόρων μπορεί να υπάρξει αδιέξοδο αλλά στη δεύτερη περίπτωση είναι συνήθως πιο δύσκολο να εντοπιστεί.

2. Είδη πόρων (συνέχεια)

- Παράδειγμα δημιουργίας αδιέξοδου κατά τη χρήση συσκευών:

```
        P1                              P2
repeat                          repeat
        …                               …
        Request(Disk);                  Request(Tape);
        …                               …
        Request(Tape);                  Request(Disk);
        …                               …
        Release(Tape);                  Release(Disk);
        …                               …
        Release(Disk);                  Release(Tape);
        …                               …
forever                         forever
```

- Παράδειγμα δημιουργίας αδιέξοδου κατά τη δέσμευση μνήμης. Υποθέτουμε ότι υπάρχουν διαθέσιμα για δέσμευση 200KB:

```
P1                      P2
…                       …
Request 80K;            Request 70K;
…                       …
Request 60K;            Request 80K;
```

- Παράδειγμα δημιουργίας αδιέξοδου κατά την ανταλλαγή μηνυμάτων (θεωρούμε ότι το `Receive` θέτει τη διεργασία υπό αναστολή αν δεν έχει αφιχθεί το μήνυμα):

```
P1                      P2
…                       …
Receive(P2,M);          Receive(P1,Q);
…                       …
Send(P2,N);             Send(P1,R);
```

3. Συνθήκες αδιέξοδου

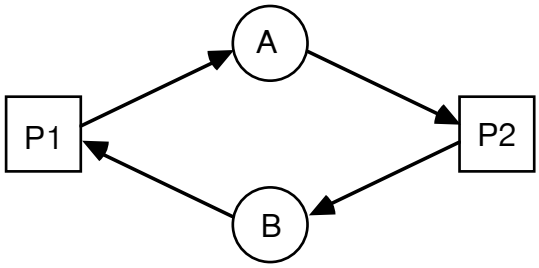
- Οι Coffman *et al.* (1971) έδειξαν ότι πρέπει να ικανοποιούνται τέσσερις συνθήκες για να οδηγηθούμε σε αδιέξοδο:

  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 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, όπως δείχνει το παρακάτω σχήμα.



- Σημειωτέον ότι πρέπει να ικανοποιούνται και οι τέσσερις συνθήκες για να δημιουργηθεί αδιέξοδο.

- 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.

4. Αντιμετώπιση 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.
  Ανίχνευση (detection) ύπαρξης αδιέξοδου και *επανόρθωσης* (recovery).
  Δυναμική *αποφυγή* (avoidance) με προσεκτική κατανομή πόρων.

5. Αγνόηση του αδιέξοδου

- 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 οϊοθετήσει αυτή την πολιτική της αγνόησης του προβλήματος.

6. Πρόληψη του αδιέξοδου

- Η λογική εδώ είναι να εξασφαλίσουμε ότι τουλάχιστον μία από τις τέσσερις συνθήκες δημιουργίας αδιέξοδου δεν μπορεί να ικανοποιηθεί.

- Στη γενική περίπτωση η συνθήκη του αμοιβαίου αποκλεισμού δεν μπορεί να αναιρεθεί διότι πολλοί πόροι (π.χ. ΚΜΕ, εκτυπωτές) απαιτούν αποκλειστική χρήση τους από μία διεργασία.

- Η συνθήκη της 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.

- Η συνθήκη της μη προεκχώρησης είναι επίσης μάλλον αδύνατο να αναιρεθεί στη γενική περίπτωση (π.χ. την ώρα που τυπώνεται ένα αρχείο, ο εκτυπωτής δίνεται σε άλλη διεργασία!). Μπορεί να εφαρμοσθεί μόνο για πόρους όπου είναι εύκολη η αποθήκευση/αλλαγή καταστάσεων (π.χ. ΚΜΕ).

- Η συνθήκη της 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.

7. Ανίχνευση και επανόρθωση αδιέξοδου

- Οι στρατηγικές πρόληψης 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.

- Η ανίχνευση γίνεται βασικά μέσω της υλοποίησης κάποιου από τους αλγόριθμους για εύρεση κύκλων σε κατευθυνόμενους γράφους (directed graphs) και αυτό γιατί η κυκλική αναμονή δημιουργεί κύκλους στα γραφήματα που αναπαριστούν την εκχώρηση των πόρων σε διεργασίες (όπως αυτό της παραγράφου 3).

- Η επανόρθωση του αδιέξοδου μπορεί να γίνει με τους εξής τρόπους:

  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 συντρέχουσων διεργασιών.

  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.

7. Ανίχνευση και επανόρθωση αδιέξοδου (συνέχεια)

- 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 τον 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.

8. Αποφυγή 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.

- Υπάρχουν δύο προσεγγίσεις στο πρόβλημα της αποφυγής αδιέξοδου:
  - Αποφυγή έναρξης εκτέλεσης μίας διεργασίας που μπορεί να προκαλέσει αδιέξοδο.
  - Αποφυγή εκχώρησης ενός πόρου σε κάποια διεργασία αν αυτή η εκχώρηση μπορεί να προκαλέσει αδιέξοδο.

- Η στρατηγική στην πρώτη προσέγγιση είναι ότι επιτρέπεται η έναρξη εκτέλεσης μίας διεργασίας αν το σύνολο των αναγκών της διεργασίας σε πόρους σε συνδυασμό και με τις ανάγκες των υπάρχουσων 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.

8. Αποφυγή αδιέξοδου (συνέχεια)

- Η δεύτερη προσέγγιση, 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.

- 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 σε 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.

- Αν θεωρήσουμε την απλούστερη περίπτωση μίας ομάδας διεργασιών που προσπαθούν να δεσμεύσουν ένα αριθμό πόρων του ιδίου τύπου, τότε η ακόλουθη κατάσταση είναι ασφαλής:

Κωδικός διεργασίας	Ποσότητα πόρων που έχουν δεσμευθεί	Μέγιστη ποσότητα πόρων που μπορεί να δεσμευθούν
1	1	4
2	4	6
3	5	8

Διαθέσιμοι πόροι: 2

- Αν και οι δύο διαθέσιμοι πόροι δοθούν στη διεργασία 2, αυτή μπορεί να ολοκληρώσει την εκτέλεσή της, να αποδεσμεύσει τους 6 πόρους οι οποίοι μπορούν μετά να εκχωρηθούν στις διεργασίες 1 και 3. Έτσι όλες οι διεργασίες θα καταφέρουν τελικά να τερματίσουν.

8. Αποφυγή αδιέξοδου (συνέχεια)

- Αντίθετα η ακόλουθη κατάσταση είναι ανασφαλής:

Κωδικός διεργασίας	Ποσότητα πόρων που έχουν δεσμευθεί	Μέγιστη ποσότητα πόρων που μπορεί να δεσμευθούν
1	8	10
2	2	5
3	1	3

Διαθέσιμοι πόροι: 1

- Σε αυτή την περίπτωση όποια διεργασία και να δεσμεύσει τον τελευταίο διαθέσιμο πόρο θα μπορούσε να προκαλέσει αδιέξοδο αν μία από αυτές θα ήθελε κατόπιν και άλλους πόρους.

- Μία κατάσταση μπορεί από ασφαλής να μετατραπεί σε ανασφαλής. Αν στην περίπτωση της πρώτης κατάστασης, ένας από τους δύο διαθέσιμους πόρους δοθεί στην τρίτη διεργασία, η κατάσταση που προκύπτει είναι ανασφαλής:

Κωδικός διεργασίας	Ποσότητα πόρων που έχουν δεσμευθεί	Μέγιστη ποσότητα πόρων που μπορεί να δεσμευθούν
1	1	4
2	4	6
3	6	8

Διαθέσιμοι πόροι: 1

- Ο λόγος που η νέα κατάσταση είναι ανασφαλής είναι διότι χρειάζονται πλέον τουλάχιστον δύο πόροι για να μπορέσει μία από τις διεργασίες 2 ή 3 να τελειώσει την εκτέλεσή της, οι οποίοι πόροι δεν υπάρχουν.

- Ο αλγόριθμος του τραπεζίτη μπορεί εύκολα να γενικευθεί για πόρους πολλών ειδών με χρήση πινάκων δύο διαστάσεων που απεικονίζουν τη σχέση μεταξύ των διεργασιών, των αναγκών τους σε κάθε είδος πόρου και του διαθέσιμου αριθμού πόρων από κάθε είδος.

8. Αποφυγή αδιέξοδου (συνέχεια)

- Ο αλγόριθμος 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.

  Σε μοντέρνα λειτουργικά συστήματα οι περιορισμοί αυτοί είναι πολύ αυστηροί και έτσι ελάχιστα συστήματα χρησιμοποιούν αυτήν τη μέθοδο.

9. Ολοκληρωμένη πολιτική αδιέξοδου

- Μια και όλες οι προσεγγίσεις αντιμετώπισης του αδιέξοδου έχουν σοβαρά μειονεκτήματα, ίσως το καλύτερο που μπορεί να γίνει είναι να οϊοθετούνται διαφορετικές στρατηγικές για διαφορετικές καταστάσεις. Συγκεκριμένα, οι Silberschatz, Peterson, και Galvin πρότειναν τα 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).

10. Κλασσικά προβλήματα ταυτοχρονίας

- Το πρόβλημα του παραγωγού-καταναλωτή είναι ένα κλασσικό πρόβλημα ταυτοχρονίας που δείχνει μερικά από τα προβλήματα που πρέπει να αντιμετωπισθούν στο συντρέχοντα προγραμματισμό. Δύο άλλα κλασσικά προβλήματα είναι των αναγνωστών και εγγραφέων (readers-writers) και των συνδαιτημόνων φιλοσόφων (dining philosophers).

- Το πρόβλημα των αναγνωστών και εγγραφέων μοντελοποιεί την πρόσβαση σε μία βάση δεδομένων. Υπάρχει μία ομάδα από διεργασίες που προσπαθούν να διαβάσουν ταυτόχρονα μία περιοχή μνήμης και μία ομάδα από διεργασίες που προσπαθούν να γράψουν ταυτόχρονα στην περιοχή αυτή. Μόνο μία διεργασία μπορεί ανά πάσα στιγμή να γράφει αλλά πολλές να διαβάζουν. Την ώρα που μία διεργασία γράφει καμμία άλλη δεν μπορεί να διαβάζει.

- Οι λύσεις που έχουν προταθεί για το συντονισμό των αναγνωστών και εγγραφέων είναι δύο ανάλογα με το αν δίνουν προτεραιότητα στους μεν ή στους δε. Η πιο απλή λύση είναι αυτή που δίνει προτεραιότητα στους αναγνώστες: όσο υπάρχουν αναγνώστες που διαβάζουν τα περιεχόμενα του χώρου μνήμης, όλες οι διεργασίες που θέλουν να γράψουν πρέπει να περιμένουν.

- Αυτό μπορεί να οδηγήσει σε παρατεταμένη στέρηση για τις διεργασίες που θέλουν να γράψουν. Η δεύτερη λύση δίνει προτεραιότητα στους εγγραφείς: αν ένας εγγραφέας έχει ζητήσει να γράψει τότε δεν ικανοποιούνται καινούργιες αιτήσεις για διάβασμα μέχρι να ικανοποιηθεί το αίτημά του. Η λύση αυτή είναι πιο πολύπλοκη γιατί κάνει χρήση περισσότερων δομών ελέγχου ταυτοχρονίας, αλλά είναι και πιο αποδοτική.

- Ακολουθεί η υλοποίηση των δύο λύσεων.

10. Κλασσικά προβλήματα ταυτοχρονίας (συνέχεια)

- ```
  program Readers_Writers1;

  int readcount;
  semaphore x, wsem;

  procedure Reader;
  begin
       while true do
       begin
            wait(x);
            readcount:=readcount+1;
            if readcount=1 then wait(wsem);
            signal(x);
            < διάβασε κοινή περιοχή μνήμης >
            wait(x);
            readcount:=readcount-1;
            if readcount=0 then signal(wsem);
            signal(x);
       end
  end;

  procedure Writer;
  begin
       while true do
       begin
            wait(wsem);
            < γράψε στην κοινή περιοχή μνήμης >
            signal(wsem);
       end
  end;

  begin
       readcount:=0;
       x:=1; wsem:=1;
       parbegin
                 Reader;
                 Writer;
       parend
  end.
  ```
  ```

10. Κλασσικά προβλήματα ταυτοχρονίας (συνέχεια)

- ```
  program Readers_Writers2;

  int readcount, writecount;
  semaphore x, y, z, rsem, wsem;

  procedure Reader;
  begin
       while true do
       begin
            wait(z);
            wait(rsem);
            wait(x);
            readcount:=readcount+1;
            if readcount=1 then wait(wsem);
            signal(x);
            signal(rsem);
            signal(z);
            < διάβασε κοινή περιοχή μνήμης >
            wait(x);
            readcount:=readcount-1;
            if readcount=0 then signal(wsem);
            signal(x);
       end
  end;

  procedure Writer;
  begin
       while true do
       begin
            wait(y);
            writecount:=writecount+1;
            if writecount=1 then wait(rsem);
            signal(y);
            wait(wsem);
            < γράψε στην κοινή περιοχή μνήμης >
            signal(wsem);
            wait(y);
            writecount:=writecount-1;
            if writecount=0 then signal(rsem);
            signal(y);
       end
  end;

  begin
       readcount:=0; writecount:=0;
       x:=1; y:=1; z:=1; rsem:=1; wsem:=1;
       parbegin
                 Reader;
                 Writer;
       parend
  end.
  ```
  ```

10. Κλασσικά προβλήματα ταυτοχρονίας (συνέχεια)

- Το πρόβλημα των συνδαιτημόνων φιλοσόφων αποτελεί κλασσική περίπτωση συγχρονισμού και αποφυγής αδιέξοδου. Προτάθηκε και επιλύθηκε από τον Dijkstra το 1965.

- Η ακόλουθη λύση είναι τόσο λανθασμένη όσο και η σωστότητά της είναι προφανής:

- ```
  program Dining_Philosophers1;

  [0:4] semaphore fork;
  int i, j;

  procedure Philosopher(int i);
  begin
       while true do
       begin
            think();
            wait(fork[i]);
            wait(fork[(i+1) mod 5]);
            eat();
            signal(fork[(i+1) mod 5]);
            signal(fork[i]);
       end
  end;

  begin
       for j:=0 to 4 do fork[j]:=1;
       parbegin
                 Philosopher(0);
                 Philosopher(1); Philosopher(2);
                 Philosopher(3); Philosopher(4);
       parend
  end.
  ```
  ```

10. Κλασσικά προβλήματα ταυτοχρονίας (συνέχεια)

- Το πρόβλημα με την προηγούμενη λύση είναι ότι αν πεινάσουν ταυτόχρονα και οι 5 φιλόσοφοι και πιάσουν το αριστερό τους πηρούνι τότε στην προσπάθειά τους να πιάσουν και το δεξί δημιουργείται αδιέξοδο. Η ακόλουθη λύση δεν δημιουργεί αδιέξοδο γιατί επιτρέποντας μόνο σε 4 φιλόσοφους ταυτόχρονα να γευματίσουν επιτρέπει σε τουλάχιστον ένα να έχει 2 πηρούνια.

-
```
program Dining_Philosophers2;

[0:4] semaphore fork;
semaphore room; /* όχι δυαδικός σημαφόρος */
int i, j;

procedure Philosopher(int i);
begin
     while true do
     begin
          think();
          wait(room);
          wait(fork[i]);
          wait(fork[(i+1) mod 5]);
          eat();
          signal(fork[(i+1) mod 5]);
          signal(fork[i]);
          signal(room);
     end
end;

begin
     for j:=0 to 4 do fork[j]:=1;
     room:=4;
     parbegin
               Philosopher(0);
               Philosopher(1); Philosopher(2);
               Philosopher(3); Philosopher(4);
     parend
end.
```

11. Ταυτοχρονία και αντιμετώπιση αδιέξοδου σε μερικά Λ.Σ.

- Το Unix παρέχει αρκετούς μηχανισμούς για συγχρονισμό διεργασιών και διαδιεργασιακή επικοινωνία (διοχετεύσεις, μηνύματα, σημαφόρους, κλπ.). Ο πιο καινοτομικός είναι οι διοχετεύσεις (pipes) που αποτελούν ουσιαστικά υλοποίηση του μοντέλου του παραγωγού-καταναλωτή σε συνδυασμό με μία επέκταση της ιδέας των συρρουτινών.

- Το OS/2 επίσης παρέχει αρκετούς μηχανισμούς για συγχρονισμό διεργασιών και διαδιεργασιακή επικοινωνία παρεμφερείς με αυτών του Unix. Στην περίπτωση των σημαφόρων, το OS/2 επιτρέπει την αναστολή μίας διεργασίας με τρόπο που να την ενεργοποιήσει όταν συμβεί κάποιο από μία σειρά γεγονότων. Επίσης διαχωρίζονται οι σημαφόροι κύριας μνήμης από τους κοινούς σημαφόρους: οι πρώτοι είναι πιο γρήγοροι αλλά παρέχουν μόνο ένα μέρος της λειτουργικότητας των δεύτερων.

- Το MVS υλοποιεί τον αμοιβαίο αποκλεισμό με χρήση κλειδωμάτων (locks) και χρήση απαρίθμησης για αποφυγή αδιέξοδου.