Κεφάλαιο 6 Αδιέξοδα

Σε πολλές εφαρμογές, μια διεργασία ζητάει αποκλειστική πρόσβαση σε περισσότερους από ένα πόρους.

Π.χ. 2 Διεργασίες Α, Β θέλουν να αποθηκεύσουν ταυτόχρονα σε CD ένα σκαναρισμένο έγγραφο. Η διεργασία Α ζητάει έλεγχο του scanner και της χορηγείται η άδεια. Η Β ζητάει πρώτα τη μονάδα CD, και το παίρνει. Τώρα η Α ζητάει access στο CD, αλλά δε το παίρνει μέχρι να το free η Β. Χωρίς να το κάνει free το CD η Β, ζητάει το scanner. Σε αυτό το σημείο και οι 2 είναι blocked για πάντα. ΑΔΙΕΞΟΔΟ (deadlock)

6.1 Πόροι

Πόροι (resources): Συσκευές υλικού ή κομμάτια πληροφοριών

6.1.1 Προεκτοπίσιμοι και μη προεκτοπίσιμοι πόροι

2 είδη πόρων:

1) Προεκτοπίσιμος: Ο πόρος που μπορεί να αποσπαστεί από μια διεργασία που τον ελέγχει χωρίς άσχημες παρενέργειες.
Π.χ. Ο Α δεσμεύει printer και λίγο πριν τελειώσει τελειώνει το κβάντο χρόνου της. Η Β έχει τώρα τη μνήμη και θέλει access στον printer που αποτυγχάνει αφού τον έχει η Α. Θα θεωρούσαμε ότι έχουμε deadlock και όμως, θα μπορούσαμε να δώσουμε τη μνήμη του Β στο Α, κάνοντας τη δουλειά του το Α και στη συνέχεια free τον printer. Έτσι δεν έχουμε deadlock.

| Μη προεκτοπίσιμος: πόρος που δε μπορεί να αποσπαστεί από τη διεργασία που την κατέχει, χωρίς να προκληθούν υπολογιστικά σφάλματα.

Χρήση πόρου βήματα
1) Αίτηση για απόκτηση του πόρου
2) Χρήση του πόρου
3) free του πόρου

Αν ένα process ζητάει ένα πόρο και δε μπορεί να τον πάρει, η διεργασία blockάρεται. ή κάνει request χωρίς να blockάρεται (σε άλλα OS)

6.1.2 Απόκτηση Πόρων

Τα προηγούμενα 3 βιβλία

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
    down(&resource_1); // get resource
    use_resource_1(); // use it
    up(&resource_1); // free it
}
```

Διεργασία θέλει access σε 2 πόρ

```
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);
```

2 resources από 1 proces

```
typedef int semaphore;
emaphore resource_1;
emaphore 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);
}
```

Καλύκες χωρίς deadlock

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

→ Με deadlock

6.2 Εισαγωγή στα αδιέξοδα

Ορισμός αδιεξόδου: Ένα σύνολο διεργασιών βρίσκεται σε αδιέξοδο αν κάθε διεργασία του συνόλου περιμένει ένα συμβάν που μόνο μια άλλη διεργασία μπορεί να προκαλέσει

Αδιέξοδο πόρων: Καμία διεργασία δε μπορεί να εκτελεστεί, να απελευθερώσει κάποιο πόρο, και να αφυπνιστεί. Ο αριθμός των διεργασιών καθώς και των πόρων δεν έχουν καμία σημασία

6.2.1 Συνθήκες εμφάνισης αδιεξόδου πόρων

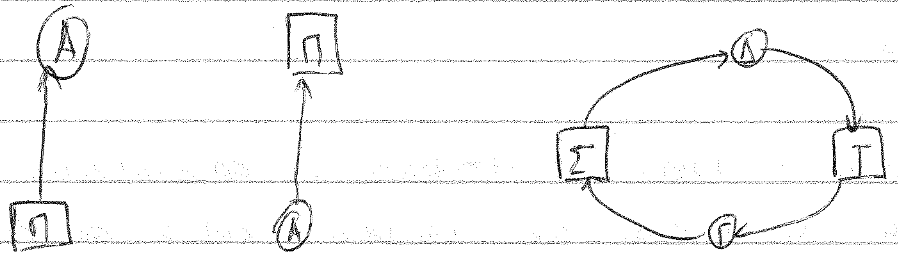
Οδηγία σε deadlock

1) Mutual exclusion. Κάθε πόρος εκχωρημένος σε 1 διεργασία ή διαθέσιμος
2) Hold and wait condition. Οι διεργασίες που δεσμεύουν πόρους να μπορούν να δεσμεύσουν και άλλους
3) Συνθήκη μη προεκχώρησης. Πόροι που έχουν εκχωρηθεί σε μια διεργασία δε μπορούν να αφαιρεθούν με τη βία
4) Circular wait condition. Πρέπει να υπάρχει μια κυκλική αλυσίδα ... 2 ή περισσότερων διεργασιών, κάθε μία από τις οποίες περιμένει ένα πόρο που κατέχει το επόμενο μέλος της αλυσίδας

Πρέπει να ικανοποιούνται και οι 4

6.29 Μοντελοποίηση των αδιεξόδων

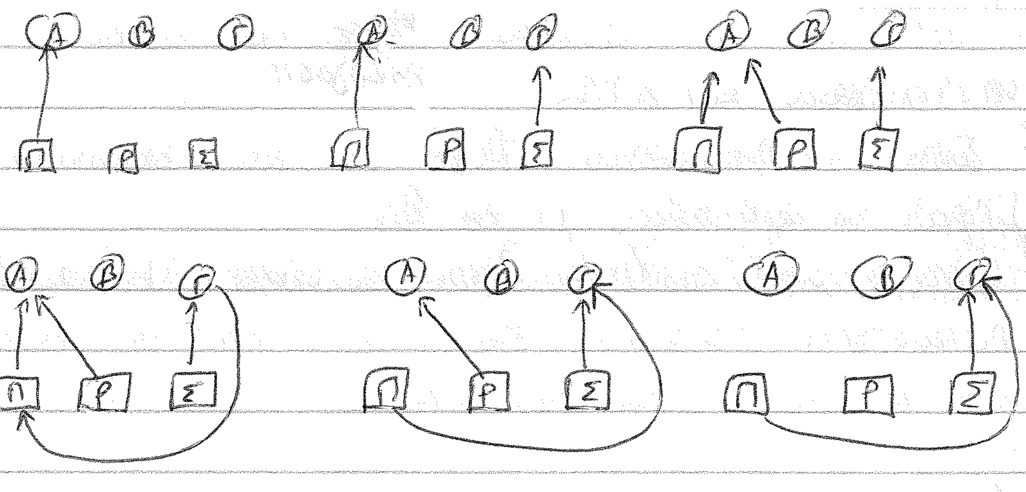
Αναπαράσταση με γράφους με 2 είδη κόμβων : 1) διεργασία (κύκλος) 2) πόρος (τετράγωνο)



η διεργασία Α έχει τον πόρο Π — η διεργασία Α στέλνει αίτηση (block) — deadlock

Το OS δεν είναι αναγκασμένο να εκτελέσει τις διεργασίες με κάποια σειρά. Έτσι μπορεί να αποφευχθεί ένα αδιέξοδο αν μας συμφέρει

1) Α ζητάει Π
2) Γ ζητάει Σ
3) Α ζητάει Ρ
4) Γ ζητάει Π
5) Α free Π
6) Α free Ρ
7) no deadlock



4 Λύσεις

1) Ignore the problem
2) detection & recovery
3) dynamic avoidance με κατανομή των πόρων
4) Prevention με άρνηση 1 από τις 4 συνθήκες που είναι αναγκαίες για τη δημιουργία deadlock

6.3 Αλγόριθμος θρασύκαρδου

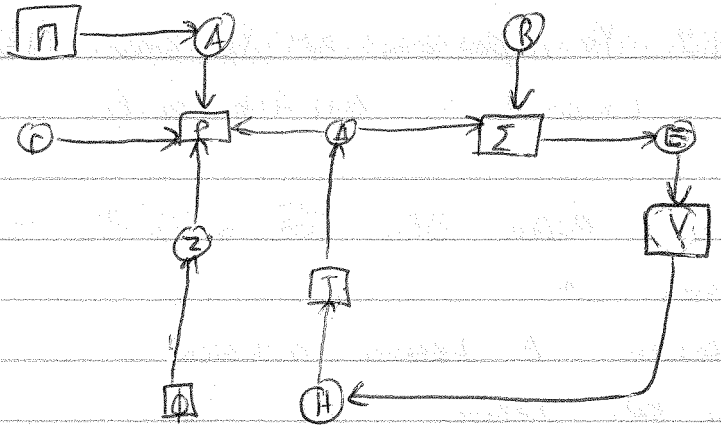
Αγνοούμε το πρόβλημα εντελώς. Αν γίνονται deadlocks κάθε 5 χρόνια και κάθε φορά αναιρέθει το σύστημα λόγω υλικού, compiler, κ.α., κανείς δε θα δώσει χρήματα και χρόνο για την εξάλειψη των deadlock

6.4 Εντοπισμός και ανάκαμψη από αδιέξοδα

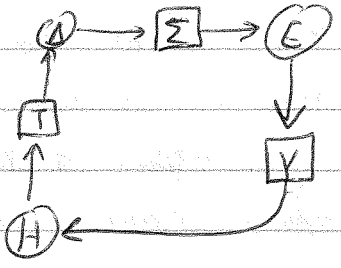
Το σύστημα δε προσπαθεί να αποτρέψει τα deadlocks. Τα αφήνει να εμφανιστούν, τα εντοπίζει και τα λύνει.

6.4.1 Εντοπισμός αδιεξόδου όταν υπάρχει ένας πόρος από κάθε είδος

Έχουμε 1 μόνο πόρο από κάθε είδος
Εμφάνιση κύκλου = deadlock
πχ 1) Η Α έχει Π, ζητάει Ρ 2) Β ζητάει Σ και Γ ζητάει Ρ
4) Δ κατέχει Τ και ζητάει Ρ, Σ 5) Ε κατέχει τον Σ, ζητάει Υ
6) Ζ κατέχει τον Φ, ζητάει Ρ 7) Η κατέχει Υ, ζητάει Τ



έχει deadlock εδώ



Αλγόριθμος εύρεσης deadlock σε μεγάλα συστήματα

1) Για κάθε κόμβο Κ του γράφου, εκτελεί 5 βήματα με τον Κ ως αρχικό κόμβο
2) Δίνει αρχική τιμή στη λίστα Λ ως κενή λίστα και χαρακτηρίζει όλα τα τόξα ως ανεξέταστα
3) Προσθέτει τον συγκεκριμένο κόμβο στο τέλος της Λ και ελέγχει αν ο κόμβος εμφανίζεται 2 φορές. Αν ναι, ο Λ έχει κύκλο και τερματίζεται.
4) Από τον συγκεκριμένο κόμβο, ελέγχει αν υπάρχουν εξερχόμενα ανεξέταστα τόξα. Αν ναι → 5 αλλιώς → 6
5) Διαλέγει στην τύχη ένα ανεξέταστο τόξο και το ανεξέτασε. Το ακολουθεί σε νέο τρέχοντα κόμβο και πάει στο 6
6) Αν ο κόμβος είναι ο αρχικός ο κύκλος τερματίζεται. Διαφορετικά deadlock. Το αφαιρεί και επιστρέφει στον προηγούμενο κόμβο (πριν από αυτόν - τρέχων -), του κάνει current και → 3

Δομή δεδομένων Λ - λίστα κόμβων - καθώς και η λίστα τόξων

6.4.2 Εντοπισμός αδιεξόδων όταν υπάρχουν πολλοί πόροι από κάθε είδος

Η κατηγορία πόρων, Υ1 πόροι στην 1η, Υ2 πόροι στη 2η, Υκ στην κ (1≤κ≤η) το Υ λέγεται διάνυσμα υπαρχόντων πόρων. Εκφράζει το συνολικό αριθμό αντιτύπων από κάθε πόρο.

Έστω Θ το διάνυσμα των διαθέσιμων πόρων όπου Θκ ο αριθμός των αντιτύπων του πόρου κ διαθέσιμα την current time.

Η Μητρώα: Τ μητρώο τρέχουσας κατανομής, Α μητρώο αιτήσεων
- Τ: πόσα αντίγραφα υπέρχουν συνολικά από κάθε κατηγορία πόρων Τκ
- Τκλ αριθμός αντιτύπων του πόρου λ locked από τη διεργασία κ
- Ακλ αριθμός αιτήσεων του πόρου λ από τη διεργασία Δκ

$$\sum_{k=1}^{n} T_{kl} + \Theta_l = Y_l$$ πλήθος αντιτύπων στην κατηγορία αυτή

Υπάρχοντες πόροι (Y_1, Y_2, Y_3, Y_4)

Μητρώο τρέχουσας κατανομής

$$\begin{bmatrix} T_{11} & T_{12} & T_{13} & \dots & T_{1\mu} \\ T_{21} & & & & \\ \vdots & & & & \\ T_{\nu 1} & T_{\nu 2} & T_{\nu 3} & \dots & T_{\nu\mu} \end{bmatrix}$$

πόσα αντίγραφα κάθε πόρου δεσμεύθηκαν από τη διεργασία ν

Διαθέσιμοι πόροι $(\Theta_1, \Theta_2, \Theta_3, \Theta_4)$

Μητρώο αιτήσεων

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1\mu} \\ A_{21} & & & & \\ & & & & \\ A_{\nu 1} & A_{\nu 2} & A_{\nu 3} & & A_{\nu\mu} \end{bmatrix}$$

δείχνει τους πόρους που χρειάζεται η διεργασία ν

Αρχικά, όλες οι διεργασίες ασημείωτες. Σταδιακά μειώνεται ο αριθμός, δηλαδή δεν είναι σε deadlock. Όσες μένουν ασημείωτες στο τέλος, είναι σε deadlock.

Worst case scenario

1) Ψάξε ασημείωτη εργασία Δκ, για την οποία η κ-οστή γραμμή του μητρώου Α είναι μικρότερη ή ίση από την κ-οστή γραμμή του μητρώου Θ
2) Αν βρεθεί πρόσθεσε την κ-οστή γραμμή του Τ στην κ-οστή γραμμή του Θ, σημείωσε την διεργασία και goto 1
3) Αν δεν υπάρχει, ο αλγόριθμος τερματίζει

Αν υπάρχουν ασημείωτες → deadlock

$V = (4, 2, 3, 1)$ $\quad \Theta = (2, 1, 0, 0)$

$$T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \qquad A = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

α έχει η κάθε διεργασία α θέλει ακόμη

Αρχικά μόνο η 3η μπορεί να φύγει και κάνει free $\Theta(2\ 2\ 3\ 0)$
Μετά φεύγει η 2η και κάνει free $\Theta(4\ 2\ 2\ 1)$
" " η 1η " " " " $(4\ 2\ 3\ 1)$

Ø deadlock

6.4.3 Ανάκαμψη από deadlock

Ανάκαμψη μέσω προεκτόπισης

Πολλές φορές μπορούμε να αποσπάσουμε ένα πόρο από μια διεργασία και να το δώσουμε σε άλλη διεργασία. Μερικές φορές χρειάζεται και η ανθρώπινη επέμβαση.
Η δυνατότητα εκτόπισης ενός πόρου από μια διεργασία, εκχώρησή σε μια άλλη και επιστροφή του πόρου στην αρχική διεργασία εξαρτάται σε μεγάλο βαθμό από τη φύση του πόρου. Η ανάκαμψη με αυτό τον τρόπο είναι δύσκολη ή αδύνατη (χωρίς το καταλάβει).

Ανάκαμψη μέσω αναδρομής (rollback)

Οι σχεδιαστές και ελεγκτές γνωρίζουν ότι μπορεί να συμβούν deadlocks και φροντίζουν να υπάρχουν περιοδικά σημεία ελέγχου των διεργασιών. Η ύπαρξη τέτοιων σημείων ελέγχου σε μια διεργασία σημαίνει ότι το state της γράφεται σε ένα αρχείο ώστε να μπορεί να επανεκκινηθεί αργότερα. Το σημείο ελέγχου δεν περιέχει μόνο την εικόνα μνήμης αλλά και το state των πόρων. Τα νέα σημεία ελέγχου δεν αντικαθιστούν τα παλιά αλλά γράφονται σε νέα αρχεία.
deadlock → ευκολο να εντοπιστούν οι πόροι που χρειάζονται.
Για να γίνει ανάκαμψη → recover ενός state από το παρελθόν μιας διεργασίας που κατείχε κρίσιμο πόρο. Ο πόρος αυτός εκχωρείται σε deadlocked διεργασία και η ίδια blockάρεται.

Ανάκαμψη μέσω εξάλειψης διεργασιών

Απόδεσμευση πόρων → kill process (τα κύκλα)
Εναλλακτικά επιλέγεται ένα process εκτός κύκλου να δώσει τους πόρους που χρειαζόμασε με σκοπό αυτοί να ελευθερωθούν.
Προτιμότερο είναι να τερματίζονται διεργασίες που να μπορούν να ξαναρχίσουν χωρίς πρόβλημα.

6.5 Αποφυγή deadlock

Όταν μια διεργασία ζητάει πόρους, τους ζητάει όλους μαζί. Στα περισσότερα συστήματα, οι αιτήσεις γίνονται μια-μια. Το σύστημα πρέπει να αποφασίζει πότε είναι safe η εκχώρηση και πότε όχι.

6.5.2 Ασφαλείς και Ανασφαλείς καταστάσεις

Safe state: αν υπάρχει κάποια σειρά χρονοπρογραμματισμού σύμφωνα με την οποία όλες οι διεργασίες καταφέρνουν να ολοκληρωθούν ακόμη και αν όλες ζητήσουν τον μέγιστο αριθμό πόρων που χρειάζονται.

κατάσταση

A	3	9
B	2	4
Γ	2	7

free 3

A	3	9
B	4	4
Γ	2	7

free 2

A	3	9
B	0	-
Γ	2	7

free 5

A	3	9
B	0	-
Γ	7	7

free 0

A	3	9
B	0	-
Γ	0	-

free 7

πχ 9

A	3	9
B	2	4
Γ	2	7

(α)

free 3

A	4	9
B	2	4
Γ	2	7

(β)

free 2

A	4	9
B	4	4
Γ	2	7

(γ)

free 0

A	4	9
B	-	-
Γ	2	7

(δ)

free 4

Σύστημα καλάει, δε έπρεπε (α) → (β) ανασφαλής κατάσταση.

6.5.3 Αλγόριθμοι τραπεζίτη για ένα μοναδικό πόρο

4 πελάτες, κι μονάδες ο καθένας, ο τραπεζίτης έχει 10 μονάδες.

(α)

A	0	6
B	0	5
Γ	0	4
Δ	0	7

free 10

(β)

A	1	6
B	1	5
Γ	2	4
Δ	4	7

free 2

(γ)

A	1	6
B	2	5
Γ	2	4
Δ	4	7

free 1

(β) safe state

Αν μένει Γ → Δ όλα καλά

Αν γίνει όμως από (β) → (γ) ανασφαλές state

6.5.4 Ο αλγόριθμος τραπεζίτη για πολλαπλούς πόρους

A	3	0	1	1
B	0	1	0	0
Γ	1	1	1	0
Δ	1	1	0	1
E	0	0	0	0

Πόροι που έχουν εκχωρηθεί

A	1	1	0	0
B	0	1	1	2
Γ	3	1	0	0
Δ	0	0	1	0
E	2	1	1	0

Πόροι που χρειάζονται ακόμη

Υ = 6 3 4 9
Κ = 5 3 2 2
Θ = 10 8 8

Αλγόριθμος ασφάλειας

1) αναζήτηση στο δεξιό μητρώο μιας γραμμής Μ, στην οποία οι επιπλέον απαιτήσεις είναι πάντα μικρότερες ή ίσες από το Θ. Αν δεν υπάρχει τέτοια γραμμή → deadlock

2) Υπόθεση ότι η διεργασία της επιλεγμένης γραμμής ζητάει όλους τους πόρους που χρειάζεται και τερματίζεται. Σήμανση ως ολοκληρωμένη και απελευθέρωση των πόρων στο Θ

3) Επανάληψη 1,2 μέχρι να σημανθούν όλες ως ολοκληρωμένες είτε να μην υπάρχει καμία διεργασία που να μπορεί να ικανοποιηθεί

6.6.1 Προσβολή της συνθήκης αμοιβαίου αποκλεισμού

Αποφυγή εκχώρησης ενός πόρου όταν αυτό δεν είναι απαραίτητο, εξασφάλιση ότι λιγότερες διεργασίες θα διεκδικήσουν αυτόν τον πόρο

Δηλαδή η διεργασία παίρνει πόρο μόνη. Αν γίνει αυτό → χάος

6.6.2 Προσβολή συνθήκης δέσμευσης και αναμονής

Εκχώρηση όλων των πόρων πριν να ξεκινήσει καθε process.

Αν υπάρχουν → ok αλλιώς wait

Πρόβλημα → Δε ξέρουν από πριν πόσους πόρους θέλουν

6.6.3 Προσβολή της συνθήκης μη προεκτόπισης

Αν μια διεργασία κατέχει τον printer και βρίσκεται στη μέση της εκτύπωσης η βίαιη απόσπαση του printer είναι πολύπλοκη - αδύνατη υπόθεση

Λύση μετατροπή πόρων σε εικονικούς. Μεγάλοι δίσκοι → μικρό πρόβλημα

Δε μπορούν όλοι οι πόροι να μετατραπούν σε εικονικούς.

6.6.4 Προσβολή της συνθήκης κυκλικής αναμονής

1 διεργασία / 1 πόρο κάθε στιγμή. 2ος πόρος → free (1ο)

Λύση 2 οι διεργασίες μπορούν να ζητήσουν πόρους που θέλουν, αλλά οι αιτήσεις με τη σειρά αρίθμησης

6.7 Άλλα θέματα

Αμοιβαίος Αποκλεισμός	Παροχέτευση στα πάντα
Δέσμευση και αναμονή	Να ζητούνται όλοι οι πόροι στην αρχή
Μη προεκτόπιση	Αφαίρεση πόρων
Κυκλική αναμονή	Αριθμητική διάταξη των πόρων

Μέθοδος αποφυγής αδιεξόδων

6.7.1 Κλείδωμα σε 2 φάσεις

1η φάση: Η διεργασία προσπαθεί να κλειδώσει όλες τις εγγραφές που χρειάζεται, μία κάθε φορά. Αν επιτύχει → φάση 2η όπου ενημερώνει τις εγγραφές και απενεργοποιεί το κλείδωμα

Αν στη 1η φάση κάποια εγγραφή locked → unlock goto φάση 1. (Μοιάζει με δέσμευση πόρων στην αρχή)

Δεν είναι πάντα εφαρμόσιμο. Στα συστήματα real time και process control δεν είναι αποδεκτό να τερματίζεται μια διεργασία όταν είναι στο μέσο κρίσιμης της περιοχής. Δουλεύει καλά όταν ο προγραμματιστής το έχει ρυθμίσει με τέτοιο τρόπο έτσι ώστε όταν διακοπεί η λειτουργία του στη διάρκεια της 1ης, να συνεχίσει από την αρχή

6.7 Αδιέξοδο επικοινωνίας

Αδιέξοδο επικοινωνίας: Μια ομάδα διεργασιών, που είναι μπλοκαρισμένες περιμένοντας ένα συμβάν για το οποίο θα προκληθεί από κάποια διεργασία

Λύση: Timeout timer. Όταν στέλνεται ένα μήνυμα και περιμένει απάντηση, ξεκινάει ο timer. Αν λήξει πριν την απάντηση, ο αποστολέας στέλνει ξανά

2 είδη PC: PC υπηρεσίας & routers
- Host είναι το PC χρήστη, PC εταιρείας ή ο server
- Οι υπολογιστές υπηρεσίας εξυπηρετούν τους χρήστες.
Ο router μεταφέρει πακέτα από src → dest

6.7.3 Ενεργό αδιέξοδο

Livelock: 2 διεργασίες A,B κάθε μία χρησιμοποιεί το κρίσιμο χρόνο της χωρίς πρόοδο ούτε blockάρεται

P_A
enter_region(&resource_1);
enter_region(&resource_2);

P_B
enter_region(&resource_2);
enter_region(&resource_1);

6.7.4 Starvation

Η μη εξυπηρέτηση διεργασιών για μεγάλο χρονικό διάστημα χωρίς να είναι blocked

Λύση: first come first served
Μεγαλύτερο wait time → εξυπηρέτηση χρήσιμα