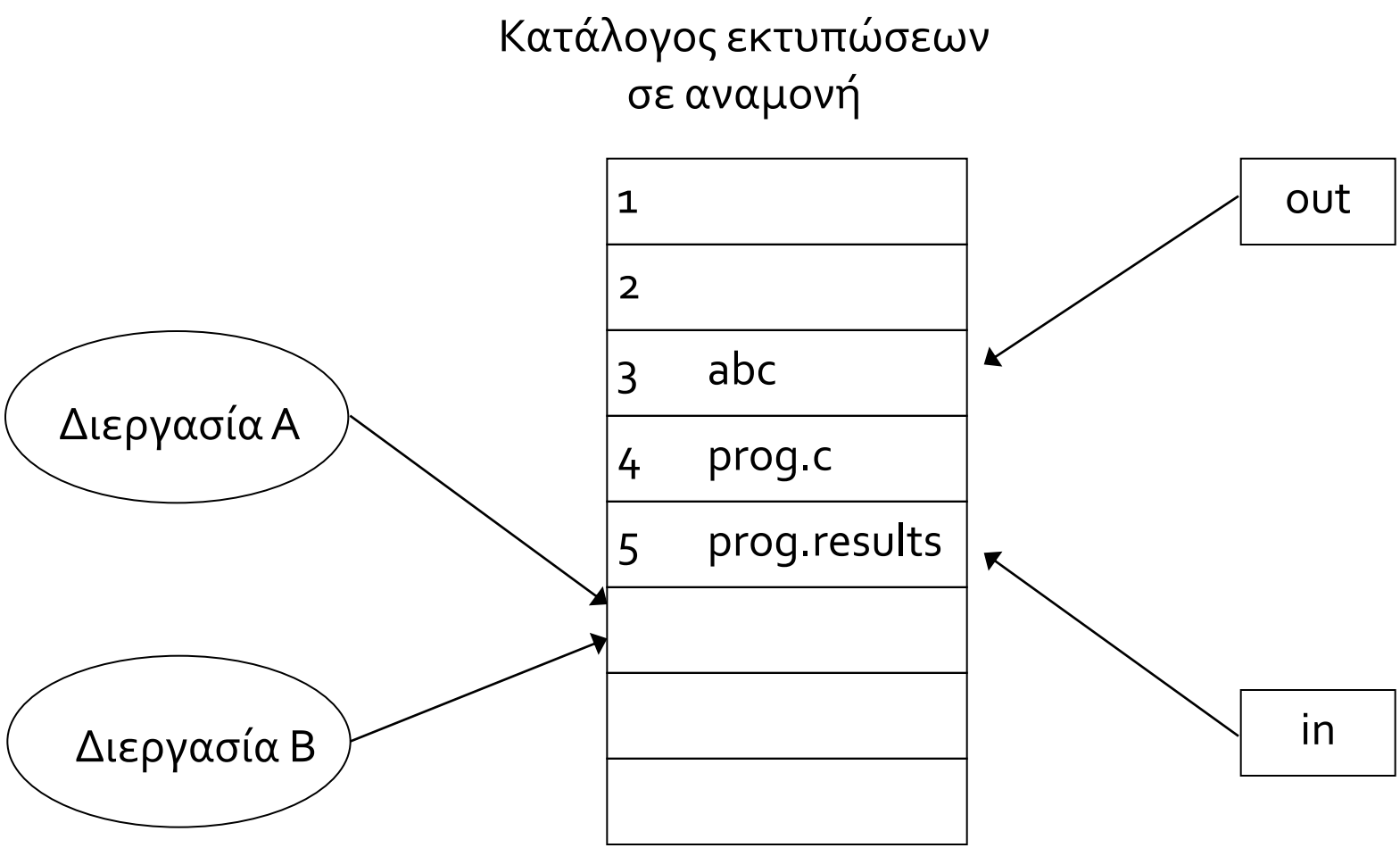
Πανεπιστήμιο Πελοποννήσου
Τμήμα Επιστήμης και Τεχνολογίας Τηλεπικοινωνιών

Λειτουργικά Συστήματα – Προγραμματισμός Συστήματος

Συγχρονισμός Διεργασιών & Αδιέξοδα

Συνθήκες Ανταγωνισμού



Δομή Διεργασιών Ι

```
while (1) {
```

Κανονικός Κώδικας
(δεν τίθεται θέμα συγχρονισμού)

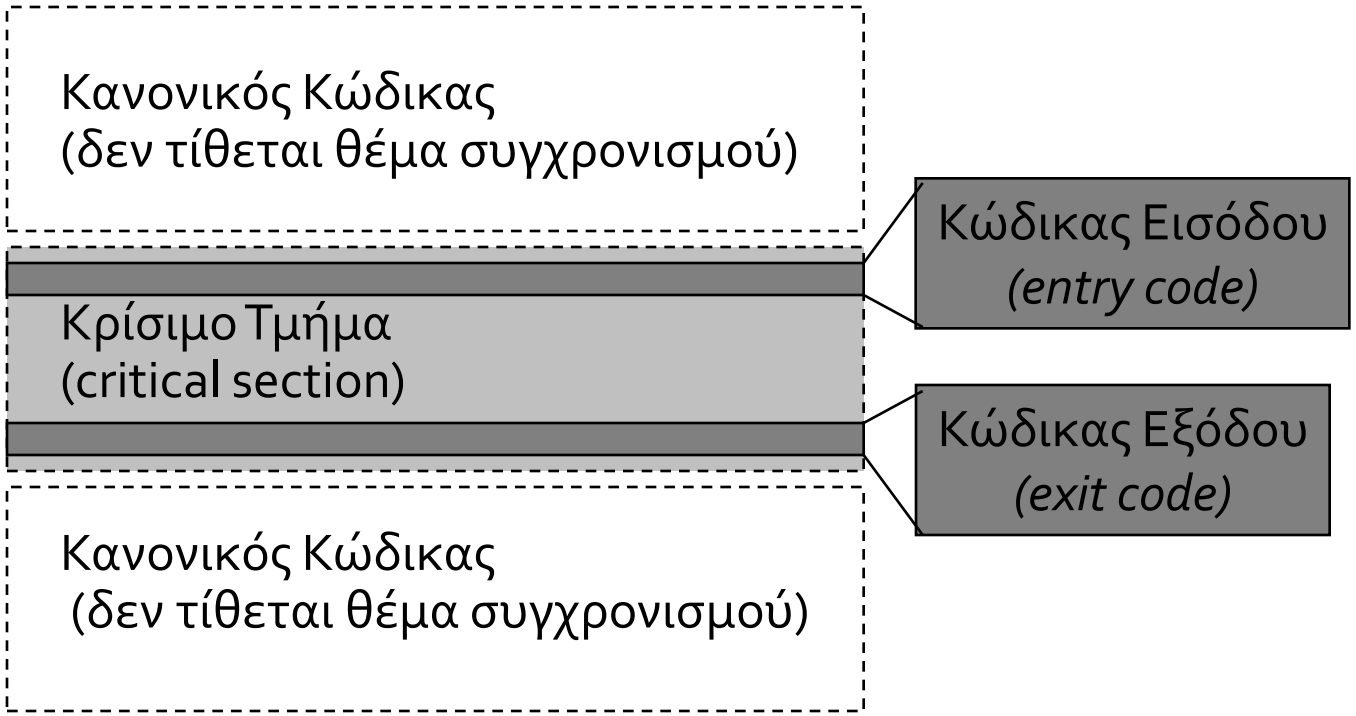
Κρίσιμο Τμήμα
(critical section)

Κανονικός Κώδικας
(δεν τίθεται θέμα συγχρονισμού)

```
}
```

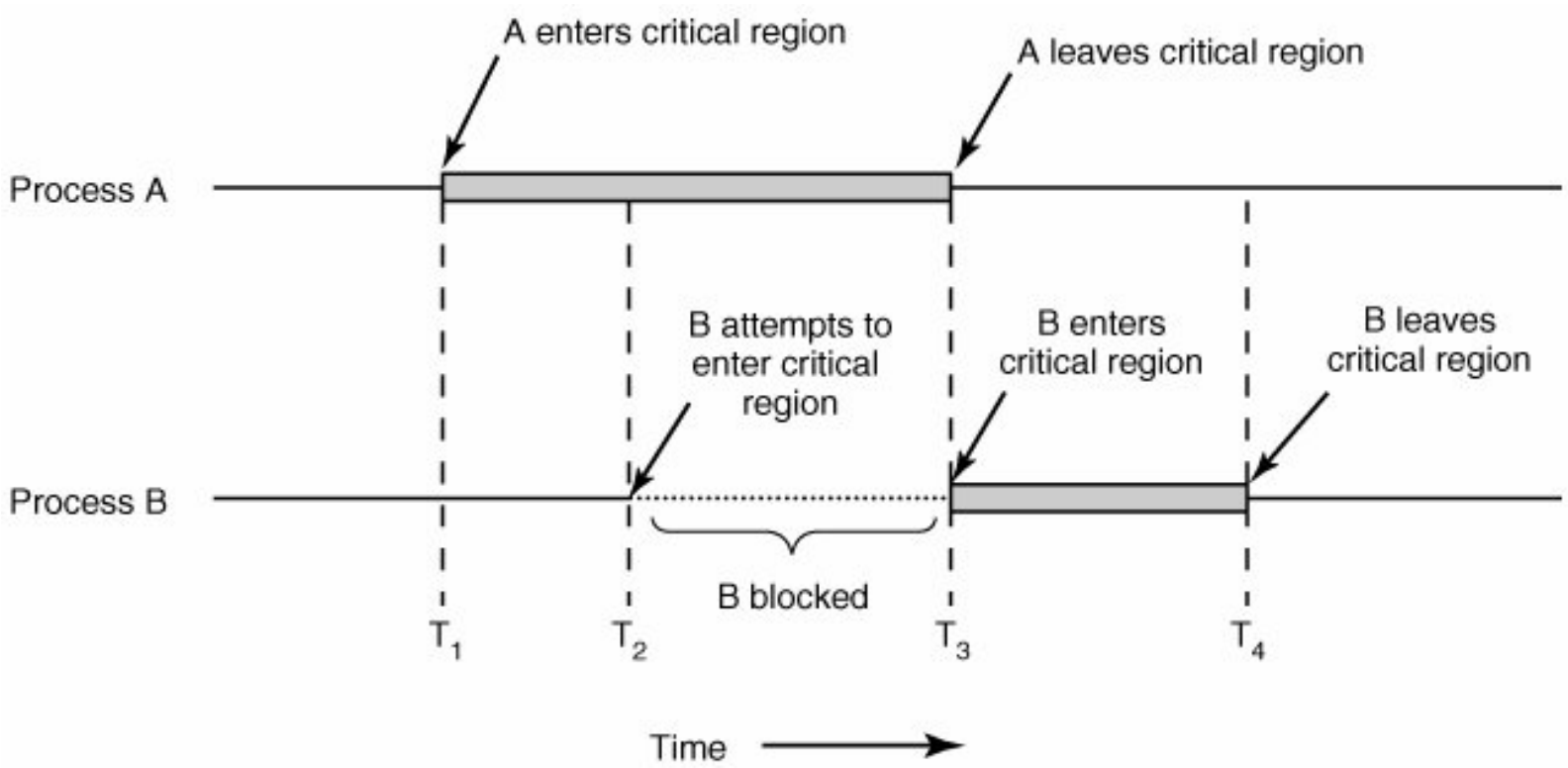
Δομή Διεργασιών ΙΙ

while (1) {



}

Αμοιβαίος Αποκλεισμός με Χρήση Κρίσιμων Περιοχών



Συνθήκες Αμοιβαίου Αποκλεισμού

- Δύο διεργασίες δεν μπορούν να βρίσκονται ταυτόχρονα σε κρίσιμα τμήματα.
- Δεν γίνονται υποθέσεις σχετικά με το πλήθος και την ταχύτητα των επεξεργαστών.
- Διεργασία που δε βρίσκεται σε κρίσιμο τμήμα δεν επιτρέπεται να αναστείλει (μπλοκάρει) άλλη διεργασία.
- Δεν επιτρέπεται να περιμένει επ' αόριστον μία διεργασία για να εισέλθει σε κρίσιμο τμήμα.

Αμοιβαίος Αποκλεισμός

- Απενεργοποίηση διακοπών:
  - Διάθεση της CPU στη διεργασία μέχρι την έξοδο από την κρίσιμη περιοχή
- Μεταβλητές κλειδώματος (lock variables)
  - Κοινόχρηστη μεταβλητή που εκφράζει εάν κάποια διεργασία βρίσκεται στην κρίσιμη περιοχή
  - Για να εισέλθει μία διεργασία στην κρίσιμη περιοχή ελέγχει την τιμή της μεταβλητής
    - αν είναι 0, τη θέτει σε 1 και εισέρχεται στην κρίσιμη περιοχή
    - αν είναι 1, ανακυκλώνει στον βρόχο εξέτασης μέχρι η μεταβλητή να γίνει 0
  - Όταν εξέρχεται, θέτει την μεταβλητή κλειδώματος στην τιμή 0

Αμοιβαίος Αποκλεισμός με Αυστηρή Εναλλαγή

- Οι διεργασίες εισέρχονται με *αυστηρή σειρά* στο κρίσιμο τμήμα
  - η κάθε μία χρίζει την επόμενή της
- Μειονεκτήματα:
  - Πρέπει να είναι γνωστό το συνολικό πλήθος των διεργασιών που *δυνητικά* θα επιθυμούν να εισέλθουν
  - Αν μία διεργασία επιθυμεί να εισέρχεται στο κρίσιμο τμήμα με μεγάλη συχνότητα και η άλλη με μικρή, η πρώτη περιμένει χωρίς λόγο
  - Ο μηχανισμός αναμονής (*ενεργός αναμονή*) δαπανά υπολογιστική ισχύ

```
int turn = 0
```

```
while (TRUE)
{
     while (turn != 0);
     <<critical_section>>
     turn = 1;
     noncritical_region();
}
```

```
while (TRUE)
{
     while (turn != 1);
     <<critical_section>>
     turn = 0;
     noncritical_region();
}
```

Η Λύση του Peterson

```
#define N     2

int turn;
int interested[N];

void enter_region(int process)
{
   int other;
   other = 1 - process;
   interested[process] = TRUE;
   turn = process;
   while (turn == process && interested[other] == TRUE)
}

void leave_region(int process)
{
   interested[process] = FALSE;
}
```

Αλλά…

- Οι προηγούμενες λύσεις σπαταλούν χρόνο της CPU στην αναμονή
- Πρόβλημα αντιστροφής προτεραιοτήτων (priority inversion problem):
  - Μία διεργασία χαμηλής προτεραιότητας έχει εισέλθει στο κρίσιμο τμήμα
  - Μία διεργασία υψηλής προτεραιότητας αφυπνίζεται και προσπαθεί να εισέλθει στο κρίσιμο τμήμα
  - Τίθεται σε ενεργό αναμονή – το λειτουργικό σύστημα της παραχωρεί διαρκώς την ΚΜΕ
  - Η χαμηλής προτεραιότητας διεργασία δεν εξέρχεται ποτέ από το κρίσιμο τμήμα, η υψηλής προτεραιότητας μένει πάντα στην ενεργό αναμονή

Οι εντολές sleep και wakeup

- Ζητούμενο: αντί μία διεργασία να εκτελεί ενεργό αναμονή να αναστέλλεται
  - δε δαπανά κύκλους της CPU
  - επιτρέπει τον χρονοπρογραμματισμό άλλων διεργασιών πιθανώς μικρότερης προτεραιότητας
- Προσθήκη εντολών:
  - **`void sleep(void);`**
    (Η καλούσα διεργασία τίθεται σε αναστολή)
  - **`void wakeup(pid_t pid);`**
    (Αφυπνίζεται η προσδιοριζόμενη διεργασία pid)

Το Πρόβλημα Παραγωγού – Καταναλωτή Ι

- Γνωστό και ως "πρόβλημα περιορισμένης προσωρινής μνήμης" (bounded-buffer problem)
- Κοινή μνήμη μεγέθους N, μετρητής ελεύθερων θέσεων count (o ≤ count ≤ N)
- Διεργασία Παραγωγός:
  - Παράγει πληροφορία
  - Την προσθέτει στην κοινή μνήμη
  - Εάν η κοινή μνήμη είναι γεμάτη, ο παραγωγός "κοιμάται"
- Διεργασία Καταναλωτής:
  - Διαβάζει την πληροφορία από την κοινή μνήμη
  - Εάν η κοινή μνήμη είναι άδεια, ο καταναλωτής "κοιμάται"

Το Πρόβλημα Παραγωγού – Καταναλωτή ΙΙ

- Ο παραγωγός:

```
#define N 100
int count = 0;

void producer(void)
{
     int item;
     while (TRUE)
     {
            item = produce_item();
            if(count == N) sleep();
            insert_item(item);
            count = count + 1;
            if(count == 1) wakeup(consumer);
     }
}
```

Το Πρόβλημα Παραγωγού – Καταναλωτή III

- Ο Καταναλωτής

```
void consumer(void)
{
     int item;

     while (TRUE)
     {
            if (count == 0) sleep();
            item = remove_item();
            count = count - 1;
            if (count == N - 1) wakeup(producer);
            consume_item(item);
     }
}
```

Το… Πρόβλημα με το Πρόβλημα Παραγωγού – Καταναλωτή

- Το πρόβλημα: ο συγχρονισμός Παραγωγού και Καταναλωτή αναφορικά με την πρόσβαση στη μεταβλητή count
- Παράδειγμα:

  Αρχικά… κοινή μνήμη: άδεια (count == 0)
  - Ο Καταναλωτής διαβάζει count == 0
  - Ο Χρονοπρογραμματιστής δίνει τον έλεγχο στον Παραγωγό
  - Ο Παραγωγός τοποθετεί στοιχείο στην κοινή μνήμη και θέτει count = 1
  - Επειδή προηγουμένως count == 0, ο Παραγωγός θεωρεί ότι πρέπει να ξυπνήσει τον Καταναλωτή → wakeup(consumer)

    Ο Καταναλωτής όμως δεν κοιμάται! → δε θα λάβει ποτέ το wakeup
  - Ο Χρονοπρογραμματιστής δίνει πάλι τον έλεγχο στον Καταναλωτή
  - Ο Καταναλωτής ελέγχει την count και τη βρίσκει ίση με 0 (όπως ήταν όταν τη διάβασε)
  - Ο Καταναλωτής πέφτει για ύπνο
  - Ο Παραγωγός συνεχίζει να προσθέτει στοιχεία στην κοινή μνήμη
  - …και όταν count == N: Ο Παραγωγός πέφτει για ύπνο και αυτός!

Σηματοφορείς

- Εισαγωγή από τον Dijkstra το 1965
- Σηματοφορέας: νέος τύπος μεταβλητών που καταμετρούν τα σήματα αφύπνισης
- Λαμβάνει ακέραιες τιμές
- Δυνατές λειτουργίες:
  - up (ή wait ή P)
  - down (ή signal ή V)
  - Οι λειτουργίες up και down είναι "ατομικές" (atomic)
- Ειδική περίπτωση: mutex
  - Δυαδικός σηματοφορέας
  - Αμοιβαίος αποκλεισμός

Η Λειτουργία down

- down(semaphore *s)
  - αν η τρέχουσα τιμή του σηματοφορέα είναι μεγαλύτερη από 0, ο σηματοφορέας μειώνεται κατά 1 και η διεργασία συνεχίζει
  - αλλιώς η διεργασία αναστέλλεται *στο συγκεκριμένο σηματοφορέα* δηλαδή μπλοκάρεται και μπαίνει σε μία ουρά που αναφέρεται στον εν λόγω σηματοφορέα

```
down(semaphore *s)
{
        while (*s <= 0);
        *s--;
}
```

Η Λειτουργία up

- up(semaphore *s)
  - αν έχουν ανασταλεί διεργασίες πάνω στο σηματοφορέα, τότε μία από αυτές αφυπνίζεται και συνεχίζει, ενώ ο σηματοφορέας δεν αλλάζει τιμή
  - αλλιώς η τιμή του σηματοφορέα αυξάνεται κατά 1

```
up(int *s)
{
        *s++;
}
```

Αμοιβαίος Αποκλεισμός με Σηματοφορείς

- Ορίζεται σηματοφόρος
  `semaphore s = 1`

- Σε κάθε διεργασία:

```
while(1)
{
        down(s);
        <<critical_section>>
        up(s);
}
```

Υλοποίηση Σηματοφορέων Ι

```
typedef struct
{
   int sval;
   struct procQ pq;
} semaphore;


void init(semaphore *s, int val)
{
   s->sval=val;
   initQ(s->pq);
}
```

Υλοποίηση Σηματοφορέων ΙΙ

```
void down(semaphore *s)
{
   s->sval--;
   if (s->sval < 0)
   {
        addQ(s->pq, thisProcess());
        suspend();
   }
}

void up(semaphore *s)
{
   s->sval++;
   if (s->sval <= 0)
   {
        resume(rmvQ(s->pq));
   }
}
```

Συγχρονισμός με Σηματοφορείς

Διεργασία P_i

```
…
A
up(flag)
…
```

Διεργασία P_j

```
…
down(flag)
B
…
```

Χρήση του σηματοφόρου flag με αρχική τιμή 0

Εκτέλεση του *B* στην P_j μόνο αφού εκτελεστεί το *A* στην P_i

Παραγωγός – Καταναλωτής με Χρήση Σηματοφορέων Ι

- Προσέγγιση:

`semaphore mutex = 1`

(ελέγχει την πρόσβαση στην κρίσιμη περιοχή)

`semaphore empty = N`

(*πλήθος κενών θέσεων κοινής μνήμης)*

`semaphore full = 0`

(*πλήθος κατειλημμένων θέσεων κοινής μνήμης)*

Παραγωγός – Καταναλωτής με Χρήση Σηματοφορέων ΙΙ

```
void producer(void)
{
   int item;

   while (TRUE)
   {
       item = produce_item();
       down(&empty);
       down(&mutex);
       insert_item(item);
       up(&mutex);
       up(&full);
   }
}
```

Παραγωγός – Καταναλωτής με Χρήση Σηματοφορέων ΙΙΙ

```
void consumer(void)
{
   int item;

   while (TRUE)
   {
       down(&full);
       down(&mutex);
       item = remove_item();
       up(&mutex);
       up(&empty);
       consume_item(item);
   }
}
```

Μεταβίβαση Μηνύματος

- Όλες οι προηγούμενες μέθοδοι προϋποθέτουν την ύπαρξη διαμοιραζόμενης μνήμης - αυτό δεν ισχύει στη γενική περίπτωση
  - π.χ., λειτουργικά συστήματα που δεν παρέχουν τη δυνατότητα αυτή
  - διεργασίες σε διαφορετικούς υπολογιστές
- Ανάγκη για εναλλακτική μέθοδο → μεταβίβαση μηνύματος
- Βασικές λειτουργίες:
  - void send(recipient_t destination, void *message);
  - void receive(sender_t source, void *message);

Παραγωγός – Καταναλωτής Ι

```
void producer(void)
{
   int item;
   message m;

   while (TRUE)
   {
       item = produce_item();
       receive(consumer, &m);
       build_message(&m, item);
       send(consumer, &m);
   }
}
```

Παραγωγός – Καταναλωτής ΙΙ

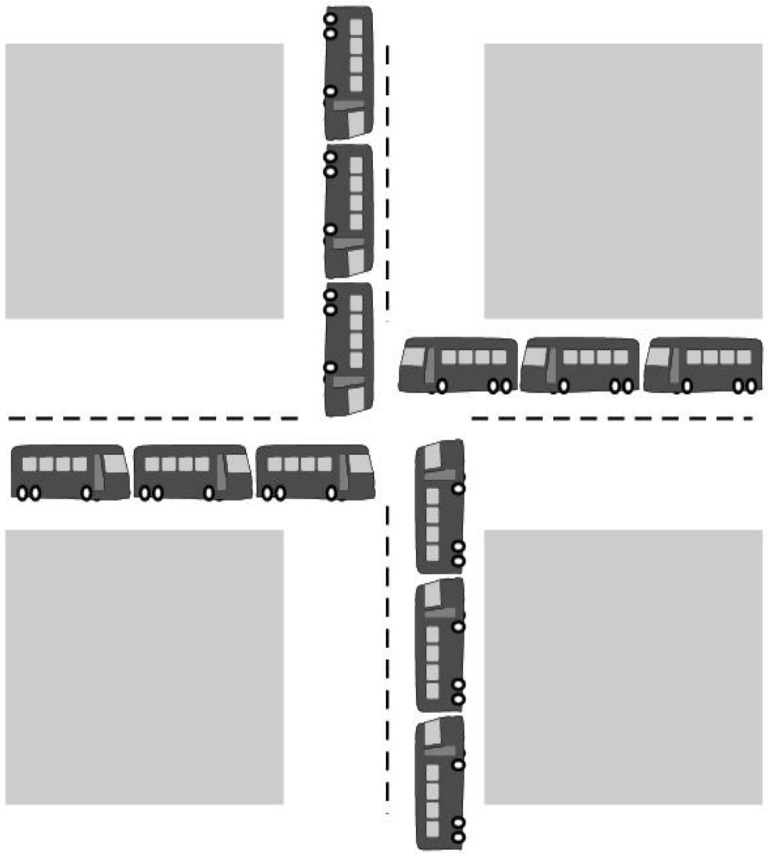
```
void consumer(void)
{
   int item, i;
   message m;

   for (i = 0; i < N; i++) send(producer, &m);
   while (TRUE)
   {
       receive(producer, &m);
       item = extract_item(&m);
       send(producer, &m);
       consume_item(item);
   }
}
```

Το Πρόβλημα των Αδιεξόδων

- Τα υπολογιστικά συστήματα διαθέτουν **πόρους (resources)**
- Οι διεργασίες χρησιμοποιούν τους πόρους
- Συγκεκριμένοι πόροι μπορούν να χρησιμοποιούνται ανά πάσα χρονική στιγμή **από μία μόνο** διεργασία
  - π.χ., εκτυπωτής, αρχείο

→ Αμοιβαίος αποκλεισμός των διεργασιών στη χρήση των πόρων αυτών

- Πολλές διεργασίες μπορεί να θέλουν περισσότερους από έναν πόρους ταυτόχρονα για κάποια εργασία
- Και τελικά, η μία διεργασία να μπλοκάρει την άλλη



Συνθήκες Αδιεξόδου

- Ένα αδιέξοδο μπορεί να συμβεί αν ισχύουν ταυτόχρονα οι εξής συνθήκες:
  - **Αμοιβαίος αποκλεισμός (Mutual exclusion)**: Μόνο μια διεργασία μπορεί να χρησιμοποιεί κάποια οντότητα ενός πόρου κάθε χρονική στιγμή.
  - **Δέσμευση και αναμονή (Hold and wait)**: Μια διεργασία που κατέχει τουλάχιστον έναν πόρο αναμένει να αποκτήσει επιπλέον πόρους που κατέχουν άλλες διεργασίες.
  - **Μη προεκτόπιση (No preemption)**: Ένας πόρος που έχει εκχωρηθεί σε μία διεργασία δεν μπορεί να αφαιρεθεί από αυτή "βίαια". Πρέπει η ίδια η διεργασία να τον αποδεσμεύσει εκούσια.
  - **Κυκλική αναμονή (Circular wait)**: Υπάρχει ένα σύνολο $\{P_0, P_1, \ldots, P_{n-1}\}$ από διεργασίες εν αναμονή, έτσι ώστε η P_0 να περιμένει για έναν πόρο που έχει δεσμεύσει η P_1, η P_1 να περιμένει για έναν πόρο που έχει δεσμεύσει η P_2, ..., και η P_{n-1} να περιμένει για έναν πόρο που έχει δεσμεύσει η P_0.

Περί «Προεκτοπισιμότητας»

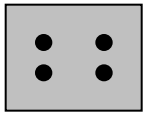
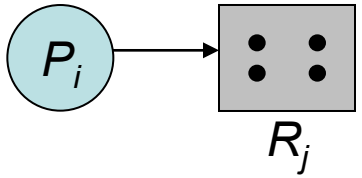
- Οι πόροι μπορεί να είναι:
  - Προεκτοπίσιμοι (preemptable): Μπορούν να αποσπαστούν από κάποια διεργασία χωρίς σοβαρές παρενέργειες
    - π.χ., CPU, μνήμη
  - Μη-προεκτοπίσιμοι (non-preemptable): Η απόσπασή τους από κάποια διεργασία προκαλεί υπολογιστικό σφάλμα
    - π.χ., εκτυπωτής, συσκευή εγγραφής CD
- Τα αδιέξοδα αφορούν μη-προεκτοπίσιμους πόρους

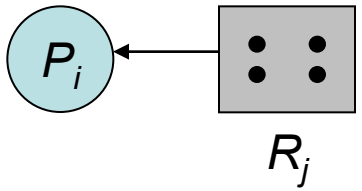
Γράφος Ανάθεσης Πόρων

- Γράφος με σύνολο κόμβων *V* και σύνολο ακμών *E*
- Το σύνολο κόμβων V χωρίζεται σε δύο τύπους:
  - $P = \{P_1, P_2, \ldots, P_n\}$, οι διεργασίες του συστήματος
  - $R = \{R_1, R_2, \ldots, R_m\}$, οι τύποι των πόρων του συστήματος
- Το σύνολο ακμών E χωρίζεται σε δύο τύπους:
  - Ακμή αίτησης: κατευθυνόμενη ακμή $P_i \rightarrow R_j$
  - Ακμή ανάθεσης: κατευθυνόμενη ακμή $R_j \rightarrow P_i$

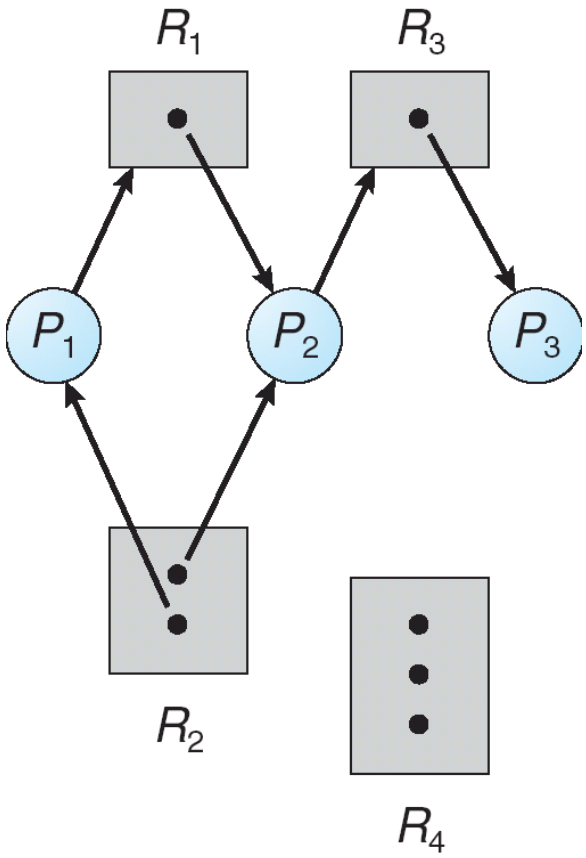
Συμβολισμοί Γράφου Ανάθεσης Πόρων

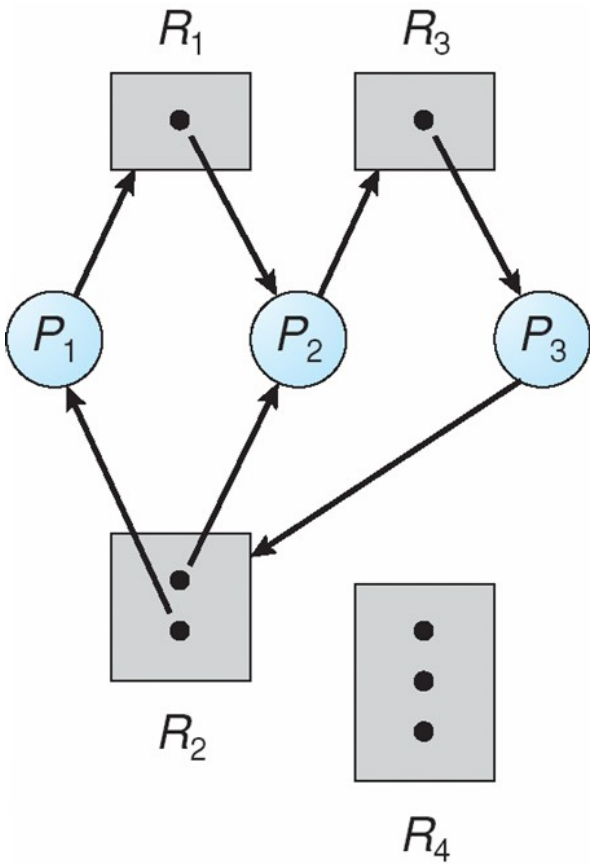
- Διεργασία P_i 

- Πόρος R_j με 4 ίδιες οντότητες 
- Η P_i κάνει αίτηση για μια οντότητα του R_j 

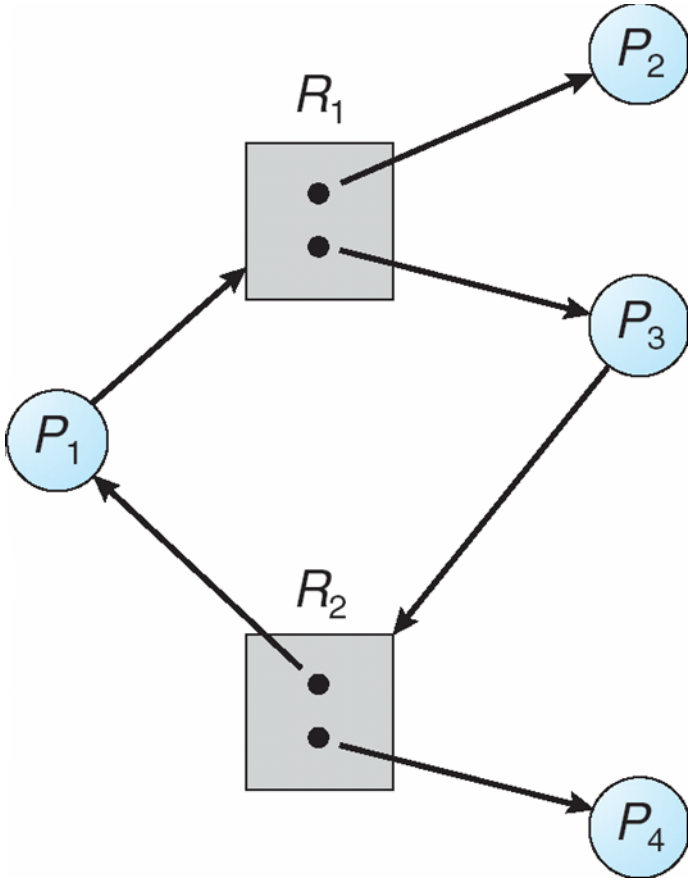
- Η P_i κατέχει μια οντότητα του πόρου R_j 

Παραδείγματα Γράφων Ανάθεσης Πόρων





Γράφος Ανάθεσης Πόρων με Κύκλο αλλά όχι και Αδιέξοδο



Βασικές Παρατηρήσεις

- Αν ένας γράφος δεν περιέχει κύκλους $\Rightarrow$ δεν υπάρχει αδιέξοδο
- Αν ένας γράφος περιέχει κύκλο $\Rightarrow$
  - Αν υπάρχει μόνο μια οντότητα για τον κάθε πόρο $\Rightarrow$ αδιέξοδο
  - Αν υπάρχουν αρκετές οντότητες ανά τύπο πόρου $\Rightarrow$ πιθανότητα εμφάνισης αδιεξόδου (αλλά όχι σίγουρα αδιέξοδο)

Μέθοδοι Χειρισμού Αδιεξόδων

- Αποτροπή: διασφάλιση ότι το σύστημα δεν θα εισέλθει ποτέ σε κατάσταση αδιεξόδου (η ριζική λύση)
- Αποφυγή: προσεκτική κατανομή των πόρων ούτως ώστε το σύστημα να μη φτάσει σε κατάσταση αδιεξόδου
- Εντοπισμός και ανάκαμψη:
  - Ανοχή: το σύστημα μπορεί να εισέλθει σε κατάσταση αδιεξόδου
  - Εντοπισμός: το αδιέξοδο εντοπίζεται
  - Ανάκαμψη: εκτελούνται οι απαραίτητες ενέργειες ούτως ώστε το σύστημα να βγει από το αδιέξοδο
- "Στουρθοκαμηλισμός": προσποίηση ότι τα αδιέξοδα δε συμβαίνουν ποτέ σε ένα σύστημα

  Η προσέγγιση αυτή χρησιμοποιείται στα περισσότερα λειτουργικά συστήματα, συμπεριλαμβανομένων και των UNIX και Windows

Αποτροπή Αδιεξόδου (Deadlock Prevention) [1]

Μέσω της προσβολής (εξαφάνισης) κάποιας από τις τέσσερις συνθήκες

- **Προσβολή συνθήκης αμοιβαίου αποκλεισμού**: άρση της αποκλειστικής εκχώρησης πόρων
  - Δύσκολο! – εξ ορισμού κάποιοι πόροι δεν είναι διαμοιραζόμενοι
- **Προσβολή συνθήκης δέσμευσης και αναμονής**: παροχή εγγυήσεων ότι όταν μια διεργασία κάνει αίτηση για έναν πόρο, δεν κατέχει άλλους πόρους
  - Απαιτεί από τη διεργασία να αιτηθεί και να της ανατεθούν όλοι οι πόροι πριν την έναρξη της εκτέλεσης, ή επιτρέπει στις διεργασίες να κάνουν αιτήσεις μόνο όταν δεν κατέχουν πόρους
  - Δυσκολία: η γνώση όλων των πόρων από την αρχή
  - Μειονεκτήματα: χαμηλή χρησιμοποίηση πόρων, πιθανότητα λιμοκτονίας

Αποτροπή Αδιεξόδου (Deadlock Prevention) [2]

- **Προσβολή συνθήκης μη-προεκτόπισης**: αφαίρεση πόρων από μία διεργασία που βρίσκεται ακόμα στο στάδιο της αίτησης για πόρους
  - Αν μια διεργασία που κατέχει κάποιους πόρους κάνει αίτηση για κάποιον πόρο ο οποίος δεν μπορεί να της ανατεθεί αμέσως, τότε όλοι οι πόροι που κρατούνται από τη διεργασία αποδεσμεύονται
  - Η διεργασία θα ξεκινήσει ξανά μόνο όταν μπορέσει να επανακτήσει όλους τους πόρους της που κατείχε αλλά και αυτούς για τους οποίους κάνει αίτηση
  - Μη ρεαλιστική! (είναι δυνατό να αφαιρέσουμε τον εκτυπωτή ή τη συσκευή εγγραφής CD από μία διεργασία?)
- **Προσβολή συνθήκης κυκλικής αναμονής**: επιβολή μιας συνολικής διάταξης όλων πόρων (δλδ., αρίθμηση των πόρων) και αίτηση για πόρους από τις διεργασίες σε αύξουσα σειρά τύπου των πόρων που χρειάζονται
  - Αποκλείεται η ύπαρξη κύκλων στο γράφο ανάθεσης πόρων
  - Δύσκολη η εφαρμογή στην πράξη, όταν οι πόροι είναι πολλοί (δλδ. πάντα!)

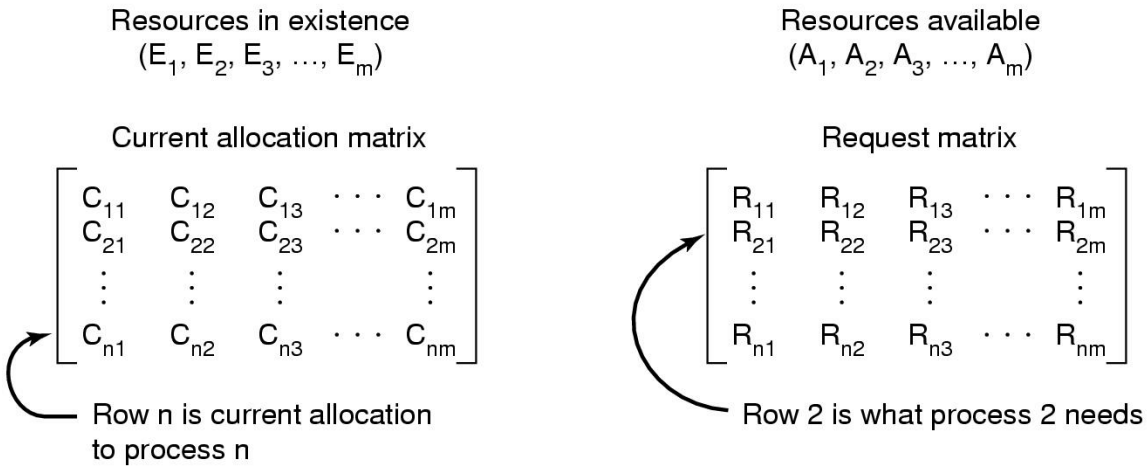
Εντοπισμός και Ανάκαμψη (Detection and Recovery)

- Εντοπισμός αδιεξόδου:
  - Εύκολη υπόθεση όταν υπάρχει μία οντότητα / πόρο
    →Εντοπισμός κύκλου στο Γράφο Ανάθεσης Πόρων
  - Πιο πολύπλοκο όταν έχουμε πολλές οντότητες / πόρο
- Ανάκαμψη από αδιέξοδο:
  - Μέσω προεκτόπισης
    - Απόσπαση πόρου από διεργασία
  - Μέσω ανασκευής (rollback)
    - Ορισμός "σημείων ισορροπίας" των διεργασιών και επιστροφή σε αυτά σε περίπτωση αδιεξόδου

      (επανεκίνηση της διεργασίας από το σημείο αυτό)
  - Μέσω εξάλλειψης διεργασιών
    - Πολύ απλά: σπάσιμο του κύκλου μέσω θανάτωσης κάποιας διεργασίας

Πώς γίνεται ο εντοπισμός αδιεξόδου?

- Δομές δεδομένων:



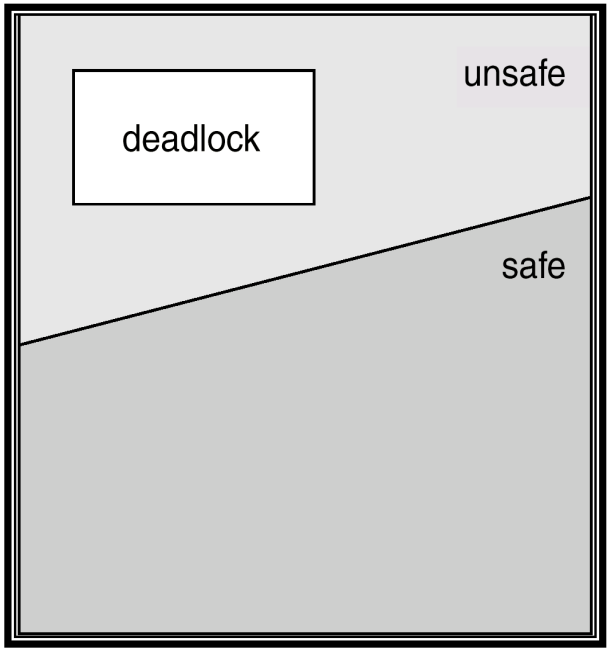
- Λογική αλγορίθμου:

  Βρες εάν υπάρχουν διεργασίες οι οποίες αιτούνται περισσότερους πόρους από όσους υπάρχουν διαθέσιμοι

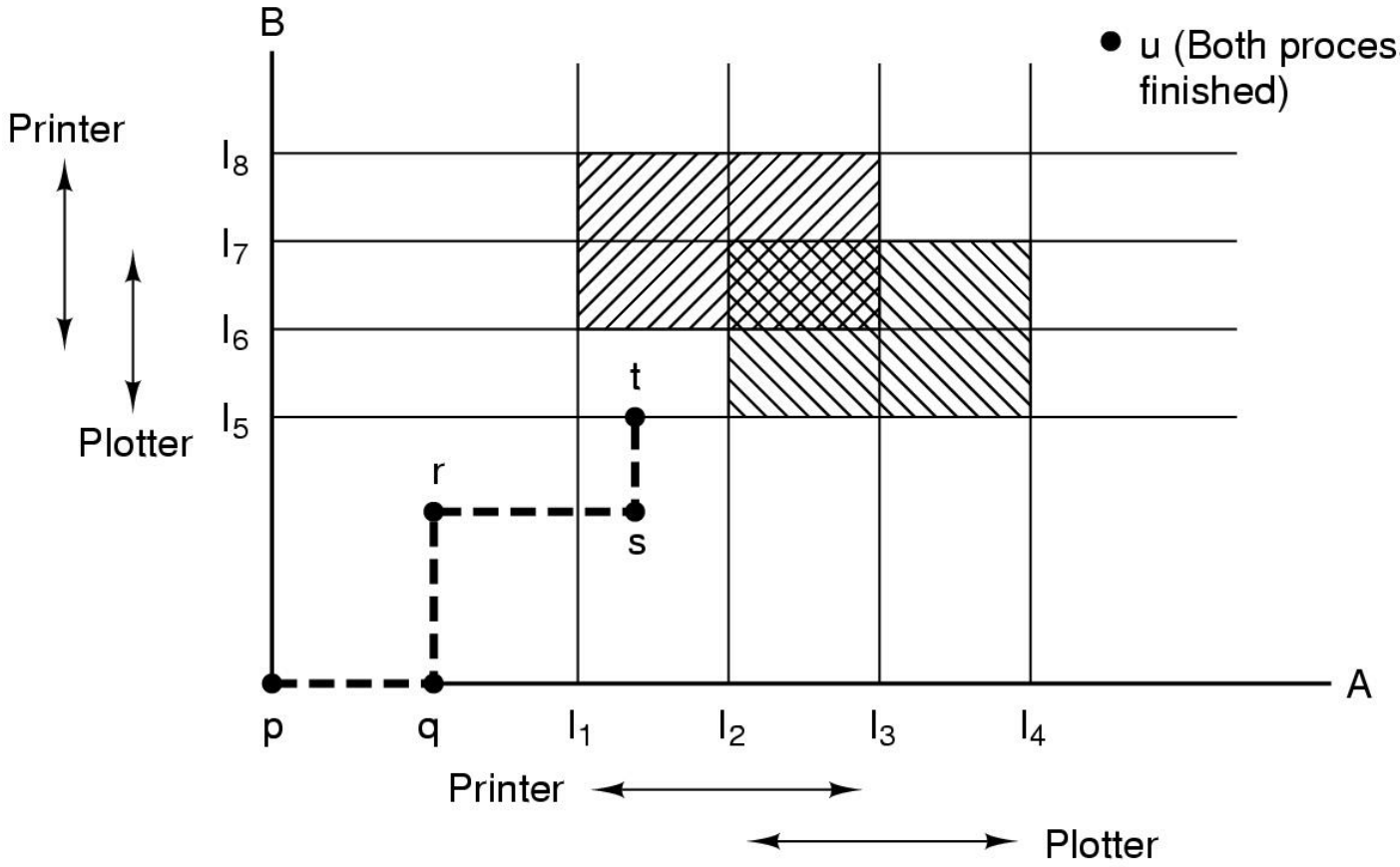
  → Αδιέξοδο

Αποφυγή Αδιεξόδου (Deadlock Avoidance)

- Για να αποφεύγει τις καταστάσεις αδιεξόδου, ένα σύστημα πρέπει να είναι σε θέση να αποφασίζει πότε η εκχώρηση ενός πόρου είναι ασφαλής και μόνο εάν είναι ασφαλής να την πραγματοποιεί
- Ασφαλής κατάσταση (safe state): υπάρχει μια ασφαλής ακολουθία όλων των διεργασιών
- Παρατηρήσεις:
  - Αν ένα σύστημα είναι σε ασφαλή κατάσταση ⇒ δεν υπάρχει αδιέξοδο
  - Αν ένα σύστημα είναι σε μη ασφαλή κατάσταση ⇒ πιθανότητα εμφάνισης αδιεξόδου
  - Αποφυγή αδιεξόδου ⇒ διασφάλιση ότι ένα σύστημα δεν θα εισέλθει ποτέ σε μη ασφαλή κατάσταση
  - Ωστόσο: το ότι το σύστημα θα εισέλθει σε μη ασφαλή κατάσταση δε συνεπάγεται απαραίτητα αδιέξοδο!



Τροχιές Πόρων



Ασφαλείς και Ανασφαλείς Καταστάσεις

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

Ασφαλής κατάσταση

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

Ανασφαλής κατάσταση

Αλγόριθμος του Τραπεζίτη (Banker's Algorithm)

- Λογική του αλγορίθμου:

  διασφάλιση ότι μία τράπεζα δε θα εκχωρούσε ποτέ τα διαθέσιμα μετρητά της με τρόπο τέτοιο που δε θα μπορούσε πλέον να ικανοποιήσει τις ανάγκες των πελατών της
- Ουσιαστικά αποτελεί επέκταση του αλγορίθμου εντοπισμού
  - Χρήση των ίδιων δομών δεδομένων
- Αποτρέπει την είσοδο σε ανασφαλή κατάσταση
- Παράδειγμα (για μοναδικό πόρο):

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

Free: 10

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

Free: 2

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

Free: 1

Αλγόριθμος του Τραπεζίτη για Πολλούς Πόρους

Process	Tape drives	Plotters	Scanners	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	Scanners	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)