

Επικοινωνία Διεργασιών (InterProcess Communication, IPC)

- Τουλάχιστον δύο διεργασίες (ο αποστολέας και ο παραλήπτης)

Σύγχρονη (synchronous)	Ασύγχρονη (asynchronous)
Χρονική σύνδεση	Δε χρειάζεται χρονική σύνδεση
π.χ. τηλέφωνο chat	π.χ. ταχυδρομείο e-mail
Ραντεβού (Rendez vous)	---
---	Ενταμίευση (Buffering)
Χωρητικότητα ενταμιευτή = 0	Περιορισμένη χωρητικότητα ενταμιευτή
	<ul style="list-style-type: none">▪ Ταχύτητα διεργασιών▪ Υπερχείλιση ενταμιευτή
	<ul style="list-style-type: none">▪ Κυκλικοί ενταμιευτές▪ Δεξαμενή ενταμιευτών

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ (Producer-Consumer Problem)

- **Διεργασία παραγωγός**
 - παράγει αντικείμενα και
 - τα τοποθετεί σε έναν κοινό ενταμιευτή (*shared buffer*)
- **Διεργασία καταναλωτής**
 - Παίρνει αντικείμενα από τον κοινό ενταμιευτή και
 - τα χρησιμοποιεί
- π.χ. εκτύπωση, πληκτρολόγηση, ...

ΠΡΟΒΛΗΜΑ:

Πρέπει

- Να εμποδιστεί ο παραγωγός όταν ο ενταμιευτής είναι γεμάτος
- Να εμποδιστεί ο καταναλωτής όταν ο ενταμιευτής είναι άδειος

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

I. ΕΝΤΑΜΙΕΥΤΗΣ ΧΩΡΗΤΙΚΟΤΗΤΑΣ 1

ΛΥΣΗ

- δύο δυαδικοί σηματοφορείς διαθέσιμος και πλήρης
- αρχικά διαθέσιμος =1; πλήρης =0;

```
process παραγωγός;  
{  
  while (true)  
  { παράγαγε α;  
    wait(free);  
    τοποθέτησε α;  
    signal (πλήρης);  
  }  
}
```

```
process καταναλωτής;  
{  
  while (true)  
  { wait(πλήρης);  
    πάρε α;  
    signal(διαθέσιμος);  
    κατανάλωσε α;  
  }  
}
```

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

II. ΚΥΚΛΙΚΟΣ ΕΝΤΑΜΙΕΥΤΗΣ ΧΩΡΗΤΙΚΟΤΗΤΑΣ N

- Ενταμιευτής: `buffer[] B = new buffer[N];`

ΛΥΣΗ

- δύο γενικοί σηματοφορείς
 - `άδειες` (θέσεις)
 - `γεμάτες` (θέσεις)
- δύο μεταβλητές
 - `in`: επόμενη άδεια θέση για τοποθέτηση (παραγωγός)
 - `out`: επόμενη γεμάτη θέση για ανάκτηση (καταναλωτής)
- αρχικά:
- `άδειες = N; γεμάτες = 0; in = 0; out = 0;`

```
process παραγωγός;
{
    while (true)
    { παράγαγε α;
      wait(άδειες);
      B[in]= α;
      in= (in+1) % N;
      signal (γεμάτες);
```

```
    }
}
process καταναλωτής;
{
    while (true)
    { wait(γεμάτες);
      α= B[out];
      out= (out+1) % N;
      signal(άδειες);
```

κατανάλωσε α;

}
}

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

III. ΔΕΞΑΜΕΝΗ N (ΟΜΟΕΙΔΩΝ) ΕΝΤΑΜΙΕΥΤΩΝ

- Δεξαμενή = Λίστα ενταμιευτών (σύνδεση με δείκτες)
- Ένας ενταμιευτής αποσπάται από τη δεξαμενή όταν πρόκειται να χρησιμοποιηθεί
- Κυκλικός ενταμιευτής χωρητικότητας N
~ δεξαμενή N ενταμιευτών χωρητικότητας 1
- **Παραγωγός:** παράγει πλήρεις ενταμιευτές για τον καταναλωτή
- **Καταναλωτής:** αφήνει άδειους ενταμιευτές για τον παραγωγό
- **Παραγωγός:** αποσπά ενταμιευτές (επεξεργασία δεικτών)
- **Καταναλωτής:** επιστρέφει ενταμιευτές (επεξεργασία δεικτών)
- *Πιθανότητα ταυτόχρονης επεξεργασίας ίδιων δεικτών*

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

III. ΔΕΞΑΜΕΝΗ Ν (ΟΜΟΕΙΔΩΝ) ΕΝΤΑΜΙΕΥΤΩΝ

ΛΥΣΗ

- δύο γενικοί σηματοφορείς
 - διαθέσιμοι (ενταμιευτές)
 - πλήρεις (ενταμιευτές)
- ένας επιπλέον δυαδικός σηματοφορέας B για να εξασφαλίσουμε τον αμοιβαίο αποκλεισμό των κρίσιμων περιοχών επεξεργασίας δεικτών
- Απλούστευση: χωρητικότητα ενταμιευτών = 1
- αρχικά διαθέσιμοι = N; πλήρεις = 0; B = 1;

```
process παραγωγός;  
{ while (true)  
  { παράγαγε α;  
    wait(διαθέσιμοι);  
    wait(B);  
    τοποθέτησε α;  
    <επεξ. δείκτες>  
    signal(B);  
    signal(πλήρεις);  
  }
```

```
}  
process καταναλωτής;  
{ while (true)  
  { wait(πλήρεις);  
    wait(B);  
    αφάισε α;  
    <επεξ. δείκτες>  
    signal(B);  
    signal(διαθέσιμοι);  
    κατανάλωσε α;
```

}

}

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

III. ΔΕΞΑΜΕΝΗ N (ΟΜΟΕΙΔΩΝ) ΕΝΤΑΜΙΕΥΤΩΝ

Απόδειξη ορθότητας λύσης

Έστω p το πλήθος των αντικειμένων που έχουν παραχθεί και c το πλήθος των αντικειμένων που έχουν καταναλωθεί. Θα πρέπει να αποδείξουμε ότι

$$N \geq p - c \geq 0.$$

Από το πρόγραμμα του παραγωγού προκύπτει ότι:

$$\#wait(\text{διαθέσιμοι}) \geq p \geq \#signal(\text{πλήρεις}) \quad (1)$$

Από το πρόγραμμα του καταναλωτή:

$$\#wait(\text{πλήρεις}) \geq c \geq \#signal(\text{διαθέσιμοι}) \quad (2)$$

(1) & (2) =>

$$\begin{aligned} \#wait(\text{διαθέσιμοι}) - \#signal(\text{διαθέσιμοι}) &\geq p - c \geq \\ &\#signal(\text{πλήρεις}) - \#wait(\text{πλήρεις}) \end{aligned} \quad (3)$$

Αλλά για όλους τους σηματοφορείς ισχύει:

$$wait(s) \leq \#signal(s) + s_0$$

Άρα

για τον διαθέσιμοι:

$$\#wait(\text{διαθέσιμοι}) - \#signal(\text{διαθέσιμοι}) \leq N \quad (4)$$

για τον πλήρεις:

$$\# \text{signal}(\text{πλήρεις}) - \# \text{wait}(\text{πλήρεις}) \geq 0 \quad (5)$$

$$(3) \ \& \ (4) \ \& \ (5) \Rightarrow \mathbf{N} \geq \mathbf{p} - \mathbf{c} \geq 0$$

ο.ε.δ.

ΠΡΟΒΛΗΜΑ ΠΑΡΑΓΩΓΟΥ-ΚΑΤΑΝΑΛΩΤΗ

ΛΥΣΗ ΚΥΚΛΙΚΟΥ ΕΝΤΑΜΙΕΥΤΗ (πίνακας)

- Οικονομικότερη σε χρόνο
- Η συχνότερα χρησιμοποιούμενη

ΛΥΣΗ ΔΕΞΑΜΕΝΗΣ (δείκτες)

Χρησιμοποιείται όταν:

- Δεν υπάρχουν εντολές ώστε η πρώτη θέση του ενταμιευτή να μπορεί να υπολογιστεί ως επόμενη της τελευταίας
(π.χ. μερικοί δίαυλοι ή ελεγκτές δε διαθέτουν τέτοιες εντολές)
- Υπάρχει μεγάλη χρονική διακύμανση στις ανάγκες επικοινωνίας (π.χ. σ' έναν επεξεργαστή κόμβου δικτύου)
- Τα περιεχόμενα του ενταμιευτή χρειάζεται να μεταφερθούν σε άλλες θέσεις της μνήμης

ΜΕΧΡΙ ΤΩΡΑ

- Επικοινωνία με χρήση
 - κοινών μεταβλητών και δομών δεδομένων και
 - σηματοφορέων
- **ΠΡΟΒΛΗΜΑ**
 - μεγάλη προσοχή στη χρήση τους
 - πιθανότητα λαθών
- **ΙΔΕΑ**

Μηχανισμοί υψηλότερου επιπέδου

Παρακολουθητές ή Γραμματείς (Monitors ή Secretaries)

Προτάθηκαν από τους Hoare και Brinch Hansen.

- Ένας παρακολουθητής (ένα αντικείμενο τύπου monitor) είναι ένας Αφηρημένος Τύπος Δεδομένων (ADT) που περιλαμβάνει:
 - **Τοπικές [στατικές] μεταβλητές/δομές**
 - [Δε χάνουν την τιμή τους κατά την έξοδο από τη διαδικασία στην οποία δηλώνονται (heap vs. stack)]
 - Τυχαίνουν επεξεργασίας MONO από τα υποπρογράμματα υλοποίησης των πράξεων
 - Κάθε κλήση υποπρογράμματος χρησιμοποιεί τις τιμές των μεταβλητών που παρέμειναν από την προηγούμενη κλήση
 - **Πράξεις**
 - Ιδιωτικές διαδικασίες/υποπρογράμματα υλοποίησης πράξεων
 - **Μόνο μία εκτελείται σε μια δεδομένη χρονική στιγμή**
 - **Υλοποιούνται σαν αμοιβαία αποκλειόμενες περιοχές**
 - Με χρήση δυαδικών σηματοφορέων από το μεταγλωττιστή
 - **Αρχικές τιμές**
 - Εντολές που δίνουν αρχικές τιμές στις μεταβλητές/δομές του και που εκτελούνται μια μόνο φορά, όταν το αντικείμενο δηλώνεται/δημιουργείται

Παρακολουθητές ή Γραμματείς

ΛΕΙΤΟΥΡΓΙΑ

- Οι διεργασίες καλούν τις διαδικασίες του παρακολουθητή
- Ο παρακολουθητής υλοποιεί (αυτόματα) τον αμοιβαίο αποκλεισμό γιατί:
 - Μόνο μια διεργασία μπορεί να χρησιμοποιήσει τον παρακολουθητή σε μια χρονική στιγμή, και ΑΡΑ
 - Οι εσωτερικές μεταβλητές και δομές δεν μπορούν να προσπελαστούν συγχρόνως
- Ο παρακολουθητής επιστρέφει τον έλεγχο στις διεργασίες (όταν εξυπηρετήσει τις κλήσεις τους)

ΠΛΕΟΝΕΚΤΗΜΑΤΑ-ΜΕΙΟΝΕΚΤΗΜΑΤΑ

- ανάλογα με τις διαδικασίες στις γλώσσες υψηλού επιπέδου
 - απόκρυψη λεπτομερειών του πόρου
 - αύξηση χρόνου εκτέλεσης

Παρόμοιοι με τις τάξεις της Java, μόνο που οι πράξεις τους εκτελούνται αμοιβαίως αποκλειστικά.

Παρακολουθητές ή Γραμματείς

Παράδειγμα (στοιχειώδες)

```
type monitor bridge // ορισμός
{
    procedure pass
        { pass the bridge }
}
```

Έστω G ένα αντικείμενο που έχει δηλωθεί ως
τύπου bridge;

```
process train1;
{ ...
  G.pass;
  ...
}
```

```
process train2;
{ ...
  G.pass;
  ...
}
```

```
process trainN;
{ ...
  G.pass;
  ...
}
```

Κύριο Πρόγραμμα:

```
{... start (train1); start (train2); ...
```

```
start (trainN); }
```

Παρακολουθητές ή Γραμματείς

ΠΡΟΒΛΗΜΑ

- Ο παρακολουθητής δεν εκτελείται ταυτόχρονα με τις διεργασίες
- Σύγχρονη επικοινωνία!

ΛΥΣΕΙΣ

χαμηλού επιπέδου: επεξεργασία σηματοφορέων από τις διαδικασίες του παρακολουθητή (αυτό θέλαμε να αποφύγουμε αρχικά!)

υψηλού επιπέδου: συνθήκες (conditions)

Παρακολουθητές ή Γραμματείς

Συνθήκες (Conditions)

- Νέος τύπος μεταβλητών
- Πράξεις (έστω ότι εκτελούνται από τη διεργασία P)
 - **cwait(c)**
 - η P εμποδίζεται πάνω στην c και
 - *παύει να χρησιμοποιεί τον παρακολουθητή*
 - **csignal(c)**
 - Αφυπνίζεται μια από τις εμποδισμένες πάνω στην c διεργασίες (ποια? τυχαία ή FIFO)
 - Αλλιώς η `csignal(c)` δεν έχει κανένα αποτέλεσμα
 - Δύο επιλογές
 - Η P εμποδίζεται και αφήνει άμεσα τον παρακολουθητή στην αφυπνισμένη διεργασία (Hoare)
 - Η `csignal(c)` είναι τελευταία εντολή της P του παρακολουθητή (Hansen) ←
 - **cnon_empty(c)**
 - συνάρτηση τύπου `boolean`
 - `true`: αν υπάρχουν διεργασίες εμποδισμένες πάνω στη συνθήκη c
 - `false`: διαφορετικά

Παρακολουθητές ή Γραμματείς

Συνθήκες vs semaphores

- Οι συνθήκες δεν έχουν τιμή/μνήμη (memoryless)
- `cwait(c)`: εμποδίζει πάντα την P
- `csignal(c)`: έχει αποτέλεσμα μόνο αν υπάρχει εμποδισμένη διεργασία πάνω στην c
- Η `cwait(c)` πρέπει να προηγείται της `csignal(c)`

Παρακολουθητές ή Γραμματείς

Πρόβλημα παραγωγού-καταναλωτή (Κυκλικός ενταμιευτής)

```
program παραγωγός_καταναλωτής;
```

```
const N=...;
```

```
type buffer = monitor
```

```
{
```

```
  B: array [0..N-1] of item;
```

```
  in, out, count: 0..N-1;
```

```
  μη_γεμάτος, μη_άδειος: condition;
```

```
procedure πρόσθεσε
```

```
(x: item);
```

```
{ if (count == N)
```

```
  cwait(μη γεμάτος);
```

```
  B[in] = x;
```

```
  in = (in+1) % N;
```

```
  count++;
```

```
  csignal(μη άδειος);
```

```
}
```

```
procedure αφείρεσε
```

```
(var x: item);
```

```
{ if (count == 0)
```

```
  cwait(μη άδειος);
```

```
  x = B[out];
```

```
  out = (out+1) % N;
```

```
  count--;
```

```
  csignal(μη γεμάτος);
```

```
}
```

```
in =0; out =0; count=0; // αρχικές τιμές
```

```
}
```

Παρακολουθητές ή Γραμματείς

Πρόβλημα παραγωγού-καταναλωτή (Κυκλικός ενταμιευτής)

Έστω ότι η μεταβλητή BUF έχει δηλωθεί ως τύπου `buffer`

```
process παραγωγός;                                process καταναλωτής;
α: item;                                           α: item;
{
  while (true)
  { παράγαγε α;
    BUF.πρόσθεσε (α);
  }
}

{ ... start (παραγωγός); ...
  start (καταναλωτής); ... }
```

ΜΕΧΡΙ ΤΩΡΑ

- Επικοινωνία με χρήση
 - **ΣΗΜΑΤΟΦΟΡΕΩΝ**
 - πιθανότητα λαθών
 - **ΠΑΡΑΚΟΛΟΥΘΗΤΩΝ**
 - Πολύ λίγες γλώσσες διαθέτουν το μηχανισμό του παρακολουθητή
 - Πολύ λίγα ΛΣ χρησιμοποιούν παρακολουθητές
 - **ΓΡΑΜΜΑΤΕΙΣ + ΠΑΡΑΚΟΛΟΥΘΗΤΕΣ**
 - Απαιτούν πρόσβαση διεργασιών σε κοινή μνήμη
 - Τι γίνεται σε χαλαρά συνδεδεμένα συστήματα που δε διαθέτουν κοινή μνήμη?

Μοντέλο πελάτη-εξυπηρετητή (Client-Server model)

- **Διεργασία εξυπηρετητής:** προσφέρει υπηρεσίες /εξυπηρετεί αιτήσεις των διεργασιών-πελατών
- **Διεργασίες πελάτες:** ζητούν υπηρεσίες/εξυπηρέτηση από τη διεργασία-εξυπηρετητή

```
process εξυπηρετητής;  
while (true)  
{  περίμενε αίτηση;  
    λάβε αίτηση;  
    εξυπηρέτησε αίτηση;  
}
```

Παραδείγματα υπηρεσιών που προσφέρουν οι εξυπηρετητές

- Ώρα και Ημερομηνία
- Εκτύπωση αρχείων
- Ανάγνωση - Εγγραφή αρχείων (εξυπηρετητής αρχείων)
- ...

Επικοινωνία

- πελάτες-εξυπηρετητής στην ίδια μηχανή
- πελάτες-εξυπηρετητής σε διαφορετικές μηχανές

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

- Μηχανισμός υψηλότερου επιπέδου από τους σηματοφορείς
- Προσφέρεται από πολλές γλώσσες προγραμματισμού
 - Οι διεργασίες μπορεί να εκτελούνται σε διαφορετικούς επεξεργαστές
- δύο πρωτογενείς πράξεις (primitives)
 - **send (διεργασία προορισμού, message) ;**
 - μονοεκπομπή (unicast)
 - στέλνει ένα μήνυμα σε ένα δεδομένο προορισμό
 - παραλλαγές
 - πολυεκπομπή (multicast)
 - `send(MANY, message) ;`
 - πανεκπομπή (broadcast)
 - `send(ALL, message) ;`
 - **receive (διεργασία πηγής, message) ;**
 - *Αν το μήνυμα δεν είναι διαθέσιμο η διεργασία παραλήπτης εμποδίζεται*
 - παραλλαγή
 - δέχεται ένα μήνυμα από οποιονδήποτε προορισμό
 - `receive(ANY, message) ;`
- κλήσεις συστήματος (σαν τους σηματοφορείς)
- όχι ευκολίες της γλώσσας (όπως οι παρακολουθητές)
- παρέχονται συνήθως ως διαδικασίες της βιβλιοθήκης

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

Πρόβλημα παραγωγού-καταναλωτή

```
process παραγωγός;  
{  
  ...  
  while (true)  
  { παράγαγε α;  
    ...  
    send(καταναλωτής, α);  
  }  
  ...  
}
```

```
process καταναλωτής;  
{  
  ...  
  while (true)  
  { receive(παραγωγός, α);  
    ...  
    κατανάλωσε α;  
  }  
  ...  
}
```

- Αν η `send` εκτελεστεί πριν από τη `receive` η διεργασία `producer` εμποδίζεται
- Σύγχρονη επικοινωνία

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

Ασύγχρονη επικοινωνία με χρήση ενταμιευτή μηνυμάτων

- Ενταμιευτής μηνυμάτων = ουρά μηνυμάτων (μια για κάθε διεργασία)
χωρητικότητα καναλιού επικοινωνίας
 - Η διεργασία δ μπορεί να στείλει ένα μήνυμα στη διεργασία δ' αν η ουρά της δ' έχει κενές θέσεις, αλλιώς εμποδίζεται
 - Η διεργασία δ' μπορεί να παραλάβει ένα μήνυμα από την διεργασία δ αν στην ουρά της υπάρχουν μηνύματα από την δ , αλλιώς η δ' εμποδίζεται
 - Υλοποίηση: γραμματοκιβώτια (mailboxes)

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

Ενταμίευση

- Δεξαμενή N ενταμιευτών

▪ ΔΙΑΔΙΚΑΣΙΑ **send**

```
process p1;  
{ ... send(p2, m); ... }
```

- Αν υπάρχει κενός ενταμιευτής, τότε
 - δεσμεύεται
 - αποθηκεύονται σ' αυτόν το m και η ταυτότητα της p1
 - αν η p2 έχει εμποδιστεί περιμένοντας μήνυμα από την p1 ελευθερώνεται
 - η p1 συνεχίζει την εκτέλεσή της
- Αν ΔΕΝ υπάρχει κενός ενταμιευτής, τότε
 - Η p1 εμποδίζεται

▪ ΔΙΑΔΙΚΑΣΙΑ **receive**

```
process p2;  
{ ... receive(p1, m); ... }
```

- Αν έχει σταλεί μήνυμα από την p1, τότε
 - το μήνυμα γίνεται δεκτό (accepted) δηλαδή αντιγράφεται
 - ο ενταμιευτής επιστρέφεται στη δεξαμενή
- Αν ΔΕΝ έχει σταλεί μήνυμα από την p1, τότε
 - Η p2 εμποδίζεται

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

~ ΥΛΟΠΟΙΗΣΗ των send και receive

Προβλήματα:

- Γεμάτος - άδειος ενταμιευτής
- Αμοιβαίος αποκλεισμός επεξεργασίας δεικτών
- Ταυτότητα αποστολέα
- Ουρές μηνυμάτων διεργασιών (μία ανά διεργασία)
- Ειδοποίηση παραλήπτη

buffer

```
{ m: m_type;  
  priority: p_type;  
  timestamp: t_type;  
  sender: id_type;  
  link: ^buffer  
}
```

- **δύο γενικοί σηματοφορείς**
 - διαθέσιμοι (ενταμιευτές)
 - πλήρεις (ενταμιευτές)
- **ένας δυαδικός σηματοφορέας B**
αμοιβαίος αποκλεισμός κρίσιμων περιοχών επεξεργασίας δεικτών
- **ένας δυαδικός σηματοφορέας m2r (message to r)**

ειδοποίηση της διεργασίας παραλήπτη \neq ότι *κάποια* διεργασία έβαλε μήνυμα στην ουρά της

- χωρητικότητα ενταμιευτών = 1

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

~ ΥΛΟΠΟΙΗΣΗ `send` και `receive`

- **αρχικά** διαθέσιμοι = N; πλήρεις = 0; B=1; m2r= 0;

process s

process r

```
procedure send
(r: id_type; m: m_type);
{

wait (διαθέσιμοι);
wait (B);

κτίσε έναν ενταμιευτή b;
πρόσθεσε τον b στην ουρά
                                του παραλήπτη;
 ενημέρωσε τους δείκτες;

signal (B);
```

```
signal (πλήρεις);
signal (m2r);
}

procedure receive
(s: id_type; m: m_type);
temp: ^buffer;
accept: boolean;
{
do
{ wait (πλήρεις);
wait (B);
temp = κεφαλή της
                                ουράς του παραλήπτη;
accept = false;
while (temp<>nil)
    && (!accept)
```

```
{ if (temp.sender ==  
                                     s)  
  { αντίγραψε το μήνυμα  
    m από τον κόμβο temp;  
    ενημέρωσε τους  
                                     δείκτες;  
    accept:= true;}  
temp:= temp^.link;
```

```
}  
signal (B);  
if (!accept)  
    wait(m2r);  
else signal  
    (διαθέσιμοι);  
} while (!accept);  
}
```

Επικοινωνία Διεργασιών με μεταβίβαση μηνυμάτων

Πρόβλημα παραγωγού-καταναλωτή

```
process παραγωγός;
```

```
{
```

```
    παράγαγε α;
```

```
    κτίσε ένα μήνυμα m;
```

```
    send(καταναλωτής, m);
```

```
}
```

```
process καταναλωτής;
```

```
{
```

```
    receive(παραγωγός, m);
```

```
    απόσπασε α από το m;
```

```
    κατανάλωσε α;
```

```
}
```

- Αν το μήνυμα (το α) είμαι μεγάλο τότε πρέπει αντ' αυτού να μεταβιβαστεί η διεύθυνση του (ενταμιευτή του)
 - Στην περίπτωση αυτή:
 - ο αποστολέας δεν μπορεί να χρησιμοποιήσει τον ενταμιευτή μέχρις ότου ο παραλήπτης να καταναλώσει το μήνυμα και πρέπει να εμποδιστεί
 - έτσι, στην περίπτωση αυτή χρειάζεται κι ένας δυαδικός σηματοφορέας!

ΜΕΧΡΙ ΤΩΡΑ

- **Επικοινωνία διεργασιών σε μια μηχανή**
 - Σηματοφορείς
 - Παρακολουθητές
 - Ουρές μηνυμάτων
 - Rendez-vous (Ada)

Διεργασίες σε διαφορετικές μηχανές

- Διαφορετικοί μηχανισμοί send-receive
 - Υποδοχές (sockets) UNIX
 - Κλήσεις απομακρυσμένων διαδικασιών (Remote Procedure Calls, PRCs)
 - Rendez-vous (Ada)