Λειτουργικά Συστήματα

Ενότητα 4: Σήματα. Διαδιεργασιακή Επικοινωνία.

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.icte.uowm.gr/mdasyg
Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ο σκοπός της ενότητας

- Η κατανόηση των σημάτων στα ΛΣ.
- Η παρουσίαση των τεχνικών της διαδιεργασιακής επικοινωνίας.
- Η ανάλυση των συνθηκών συναγωνισμού.
- Το πρόβλημα του παραγωγού-καταναλωτή.

Σήματα (signals) σε διεργασίες

Σήματα στο Unix (Unix Signals)

- Ένα σήμα αντιπροσωπεύει μια ειδοποίηση προς μια συγκεκριμένη διεργασία για κάποιο γεγονός που συνέβη.
- Τα σήματά ονομάζονται και software interrupts.
- Τα σήματα είναι ασύγχρονα (η δημιουργία και αποστολή τους γίνεται χωρίς να υπάρχει προηγούμενη συνεννόηση με τη διεργασία για την οποία προορίζονται).

- Τα σήματα στέλνονται:
  - Από μια διεργασία στην άλλη.
  - Από τον πυρήνα στη διεργασία.

Έννοιες συνδεδεμένες με τα σήματα

- Δημιουργία του σήματος.
- Μετάδοση του σήματος.
- Χειρισμός του Σήματος.

Δημιουργία Σήματος (Signal Generation)

- **Από τον χρήστη (μέσω εντολών):**
  - Η εντολή kill του shell χρησιμοποιείται για να στέλνει σήματα.
  - Ειδικοί χαρακτήρες από το πληκτρολόγιο δημιουργούν σήματα (π.χ. Ctrl-C, ή Delete, που δημιουργούν το σήμα SIGINT).
- **Από διεργασίες/κώδικα:**
  - Κλήση συστήματος kill().
  - Καταστάσεις του υλικού δημιουργούν σήματα, που στέλνονται από τον πυρήνα στη διεργασία (π.χ. αναφορά σε διεύθυνση εκτός του χώρου διευθύνσεων της διεργασίας δημιουργεί το σήμα SIGSEGV).
  - Καταστάσεις λογισμικού δημιουργούν σήματα (π.χ. άφιξη δεδομένων εκτός ζώνης (out-of-band data) σε ένα στόμιο).

Χειρισμός Σημάτων (Signal Handling)

- Τι μπορεί να κάνει μια διεργασία με ένα σήμα;
  - Μπορεί να παρέχει μια συνάρτηση που καλείται οποτεδήποτε στέλνεται ένας συγκεκριμένος τύπος σήματος στην διεργασία.
  - Μπορεί να αγνοήσει το σήμα (κάποιοι τύποι σημάτων «πιάνονται» αυτόματα σε κάθε περίπτωση από το ΛΣ) .
  - Μπορεί να επιτρέψει να συμβεί η προκαθορισμένη λειτουργία για κάθε σήμα.
- **Στις πρώτες εκδόσεις** του Unix τα σήματα μπορούσαν να χαθούν (unreliable), **οι νεότερες εκδόσεις** του Unix όμως, υποστηρίζουν αξιόπιστο (reliable) μηχανισμό σημάτων.

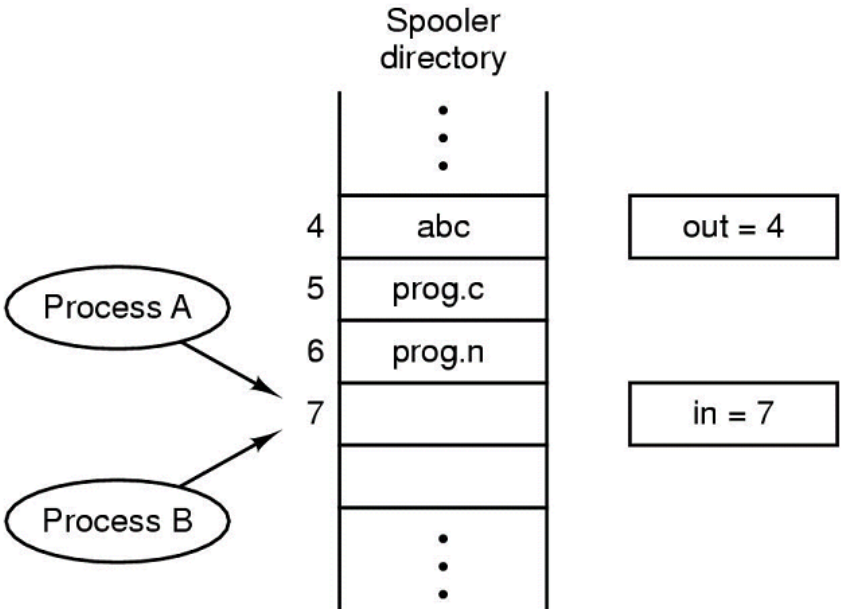
Επικοινωνία μεταξύ των διεργασιών

Διαδιεργασιακή επικοινωνία (1/2)

- Πολύ συχνά μια διεργασία πρέπει να επικοινωνήσει με μια άλλη.
- Το πιο γνωστό παράδειγμα είναι η διασωλήνωση (π.χ. ls -la | grep root | grep Mar).
- Η InterProcess Communication (IPC) μπορεί να γίνει με πολλούς τρόπους.

Ένα παράδειγμα κοινής πρόσβασης

- Έστω δυο διεργασίες Α και Β θέλουν να τοποθετήσουν ένα αρχείο στο print spooler. Έστω είναι κοινές οι out,in.
- Η Β διαβάζει την επόμενη ελεύθερη τιμή και βρίσκει 7, την αποθηκεύει σε μια τοπική μεταβλητή.
- Τότε γίνεται interrupt.
- Εκτελείται η Α, ομοίως και γράφει στη θέση 7 το αρχείο που θέλει.
- Συνεχίζει η Β και γράφει στη θέση 7 το δικό της αρχείο. Χάνεται η εγγραφή της Α.



Τι είναι οι συνθήκες ανταγωνισμού;

- Οι συνθήκες κατά τις οποίες δύο ή περισσότερες διεργασίες διαμοιράζονται την ίδια περιοχή μνήμης και το τελικό αποτέλεσμα εξαρτάται από τη σειρά αλλά και το χρόνο εκτέλεσης.

- Αν υπάρχουν race conditions τότε το σφάλμα παρουσιάζεται τυχαία και μόνο κατά τη διενέργεια παράλληλων ελέγχων.

Διαδιεργασιακή Επικοινωνία (2/2)

- Τρία βασικά ζητήματα:
  - Πώς μια διεργασία θα μεταβιβάσει πληροφορίες σε μια άλλη;
  - Πώς θα εξασφαλιστεί ότι 2 ή περισσότερες διεργασίες δεν εμποδίζουν η μία την άλλη, όταν εκτελούν κρίσιμες ενέργειες;
  - Ποια είναι η κατάλληλη αλληλουχία ενεργειών, όταν υπάρχουν εξαρτήσεις;

Βασικές Έννοιες Συγχρονισμού Διεργασιών

- Ταυτόχρονη πρόσβαση σε διαμοιραζόμενα δεδομένα μπορεί να έχει ως αποτέλεσμα την ασυνέπεια των δεδομένων.
- Η διατήρηση συνέπειας των δεδομένων απαιτεί την ύπαρξη μηχανισμών που να διασφαλίζουν την διαδοχική εκτέλεση των συνεργαζόμενων διεργασιών.
- Η λύση της διαμοιραζόμενες μνήμης στο πρόβλημα του περιορισμένου ενδιάμεσου αποθηκευτικού χώρου (buffer), επιτρέπει την ταυτόχρονη ύπαρξη το πολύ n-1 αντικειμένων σε ένα buffer.

Συνεργαζόμενες Διεργασίες

- Οι **ανεξάρτητες** (Independent) διεργασίες δεν επηρεάζουν άλλες διεργασίες ούτε και επηρεάζονται από την εκτέλεση άλλων διεργασιών.
- Οι **συνεργαζόμενες** (Cooperating) διεργασίες μπορούν να επηρεάσουν ή να επηρεαστούν από την εκτέλεση άλλων διεργασιών.
- Κύριοι λόγοι για τη δημιουργία συνεργαζόμενων διεργασιών:
  - Διαχείριση κοινής πληροφορίας.
  - Επιτάχυνση υπολογισμών.
  - Διαχωρισμός κώδικα σε ξεχωριστά τμήματα.
  - Ευκολία.

Πρόβλημα Παραγωγού - Καταναλωτή

- Παράδειγμα συνεργαζόμενων διεργασιών :

Η διεργασία παραγωγός παράγει την πληροφορία που καταναλώνει η διεργασία καταναλωτής.

- 2 γενικές περιπτώσεις :
  - Μη περιορισμένος ενδιάμεσος χώρος (unbounded buffer).
  - Περιορισμένος ενδιάμεσος χώρος (bounded buffer).

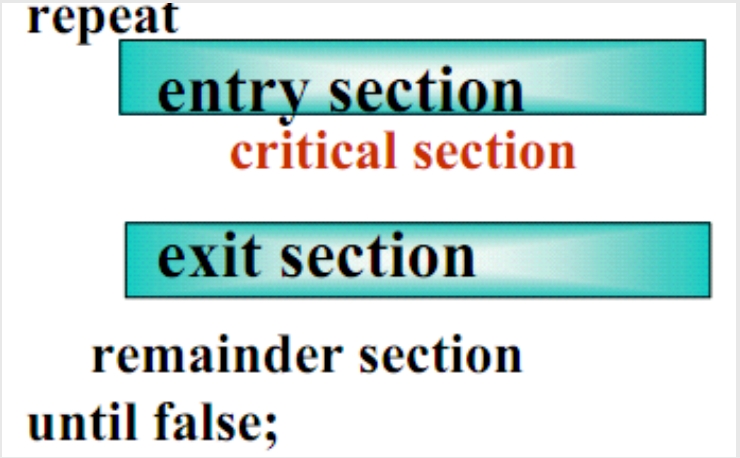
Διαμοιραζόμενα δεδομένα στο πρόβλημα παραγωγού καταναλωτή

```
repeat
…
produce an item in nextp
…
while counter=n do no-op;
buffer[in] := nextp;
in := in+1 mod n;
counter := counter+1;
until false;

type item = …;
var buffer : array[0..n-1]
of item;
in, out : 0..n-1;
counter : 0..n; in := 0; out
:= 0; counter :=0;
```

```
repeat
while counter=0 do no-op;
nextc := buffer[out];
out := out+1 mod n;
counter := counter-1;
…
consume  item in nextc;
…
until false;

Οι εντολές:
counter := counter+1;
counter:=counter-1;
πρέπει να εκτελεστούν
ατομικά
```

Το πρόβλημα του κρίσιμου τμήματος

- **n** διεργασίες ανταγωνίζονται για τη χρήση κάποιων διαμοιραζόμενων δεδομένων.
- Κάθε διεργασία έχει ένα κομμάτι κώδικα, που καλείται κρίσιμο τμήμα, μέσω του οποίου γίνεται η πρόσβαση στα διαμοιραζόμενα δεδομένα.
- ΠΡΟΒΛΗΜΑ: διαβεβαίωση ότι καμία άλλη διεργασία δεν επιτρέπεται να εκτελέσει το κρίσιμο τμήμα της, κατά τη διάρκεια εκτέλεσης του κρίσιμου τμήματος μίας διεργασίας.



Συνθήκες που απαιτούνται για την αποφυγή συνθηκών ανταγωνισμού (1/3)

- Δύο διεργασίες δεν βρίσκονται ποτέ ταυτόχρονα στις κρίσιμες περιοχές τους (mutual exclusion).
- Δεν επιτρέπονται υποθέσεις σε ό,τι αφορά την ταχύτητα ή το πλήθος των επεξεργαστών (no assumptions).
- Διεργασία που δεν βρίσκεται σε κρίσιμο τμήμα δεν επιτρέπεται να μπλοκάρει άλλες διεργασίες (progress).
- Δεν επιτρέπεται μια διεργασία να αναμένει επ' αόριστον να μπει στην κρίσιμη περιοχή της (bounded waiting).

Συνθήκες που απαιτούνται για την αποφυγή συνθηκών ανταγωνισμού (2/3)

- **Mutual exclusion:**

Εάν η διεργασία Pi εκτελεί το κρίσιμο τμήμα της, καμία άλλη διεργασία δεν μπορεί να εκτελέσει το κρίσιμο τμήμα της.

- **Progress:**

Εάν καμία διεργασία δεν εκτελείται στο κρίσιμο τμήμα της και υπάρχουν κάποιες διεργασίες που θέλουν να προχωρήσουν στην εκτέλεση του κρίσιμου τμήματος τους, τότε η επιλογή της διεργασίας που θα προχωρήσει στο κρίσιμο τμήμα της δεν μπορεί να αναβάλλεται επ' αόριστον.

Συνθήκες που απαιτούνται για την αποφυγή συνθηκών ανταγωνισμού (3/3)

- **No assumptions:**

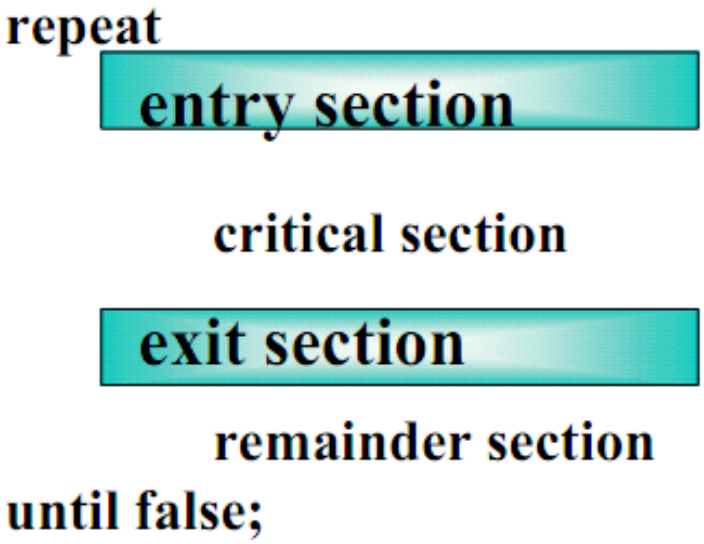
Δεν υπάρχει υπόθεση αναφορικά με τη σχετική ταχύτητα των n διεργασιών.

- **Bounded waiting:**

Πρέπει να υπάρχει όριο στο πόσες φορές επιτρέπεται οι άλλες διεργασίες να προχωρήσουν στο κρίσιμο τμήμα τους, μετά από το αίτημα εισαγωγής στο κρίσιμο τμήμα μίας διεργασίας και πριν από την ικανοποίηση του αιτήματος.

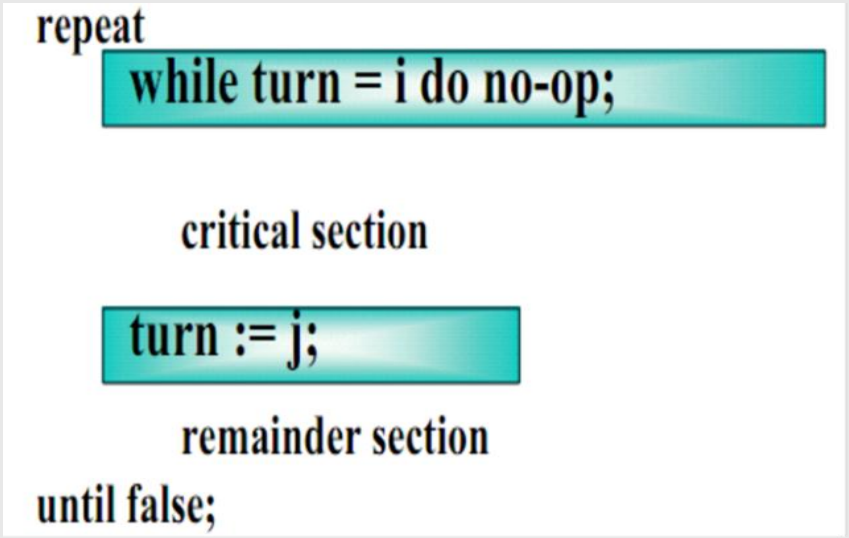
Παράδειγμα….

- Οι διεργασίες πρέπει να διαμοιράζονται κάποιες κοινές μεταβλητές για να συγχρονίζουν τις ενέργειες τους.



Έστω χρησιμοποιούμε μια κοινή μεταβλητή για το συγχρονισμό

- var turn : (0..1); (αρχικά turn = 0).
- turn = i , σημαίνει ότι η Pi μπορεί να μπει στο κρίσιμο τμήμα της.



- **Ικανοποιεί τον αμοιβαίο αποκλεισμό, αλλά όχι την πρόοδο διεργασιών.**

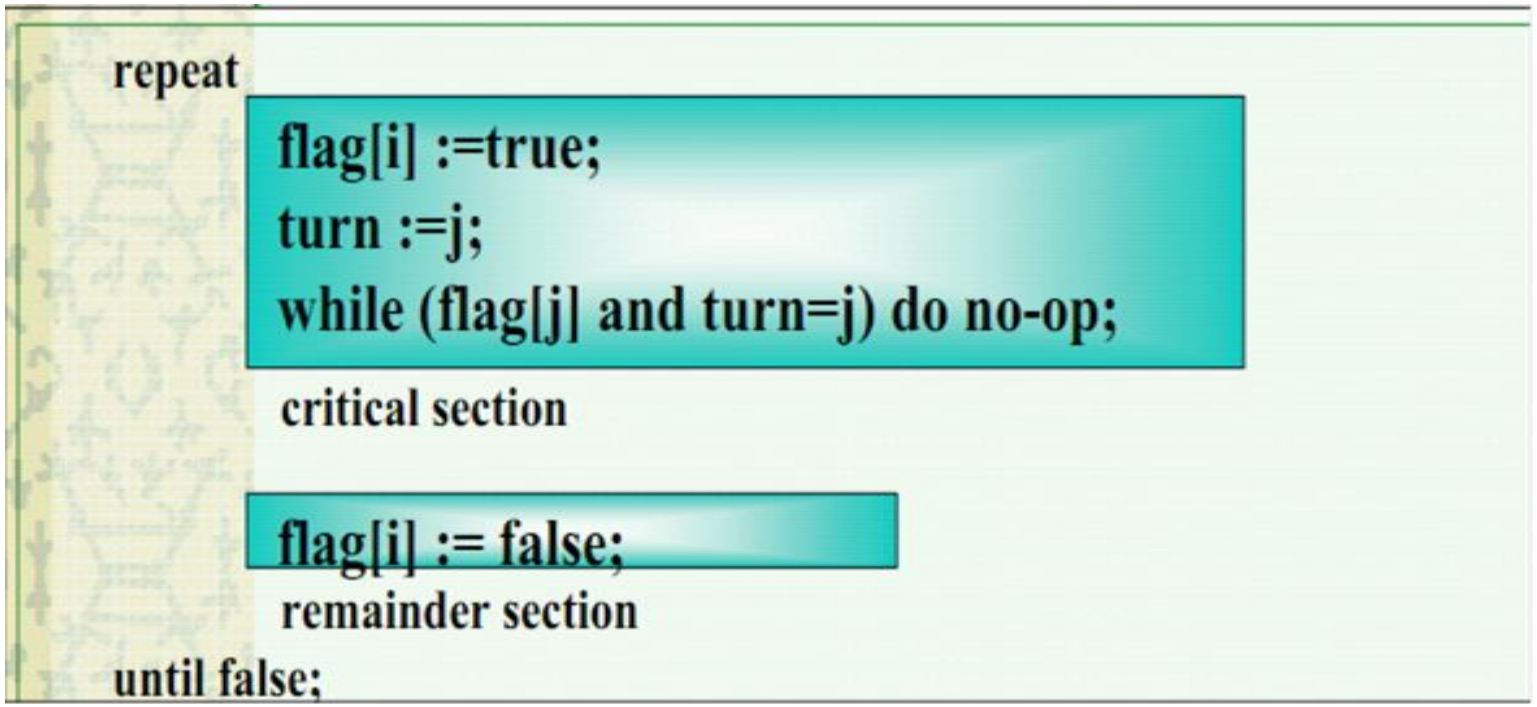
Έστω χρησιμοποιούμε πολλαπλές κοινές μεταβλητές για το συγχρονισμό

- var flag : array[0..1] of boolean;
- αρχικά flag[0] = flag[1] = false;
- flag[i] =true σημαίνει ότι η Pi μπορεί να μπει στο κρίσιμο τμήμα της.

```
repeat
        flag[i] :=true;
        while flag[j]  do no-op;

            critical section

        flag[i] := false;

            remainder section
until false;
```

- **Δεν ικανοποιεί την πρόοδο διεργασιών.**

Έστω συνδυάζουμε τις δυο προηγούμενες λύσεις



- Ικανοποιεί όλες τις προϋποθέσεις.
- Επιλύει το πρόβλημα του κρίσιμου τμήματος για 2 διεργασίες.

Βασικές Έννοιες Συγχρονισμού Διεργασιών..

- Τροποποίηση του κώδικα του προβλήματος παραγωγού-καταναλωτή με προσθήκη μίας μεταβλητής counter που εκκινεί με την τιμή 0 και αυξάνεται κάθε φορά που ένα νέο αντικείμενο προστίθεται στον buffer.

Επικοινωνία μεταξύ Διεργασιών

- **Οι διεργασίες επικοινωνούν μεταξύ (Interprocess Communiation, IPC) τους για να επιτευχθεί:**
  - Ανταλλαγή δεδομένων.
  - Συγχρονισμός επεξεργασίας.
- **Υπάρχουν δύο βασικές προσεγγίσεις επικοινωνίας:**
  - **Χρήση κοινής μνήμης.**

    Κοινή μνήμη (shared memory): οι διεργασίες χρησιμοποιούν κοινές μεταβλητές των οποίων τις τιμές αλλάζουν κατάλληλα.
  - **Ανταλλαγή μηνυμάτων.**

    Ανταλλαγή μηνυμάτων (message passing): οι διεργασίες επικοινωνούν στέλνοντας και λαμβάνοντας μηνύματα.

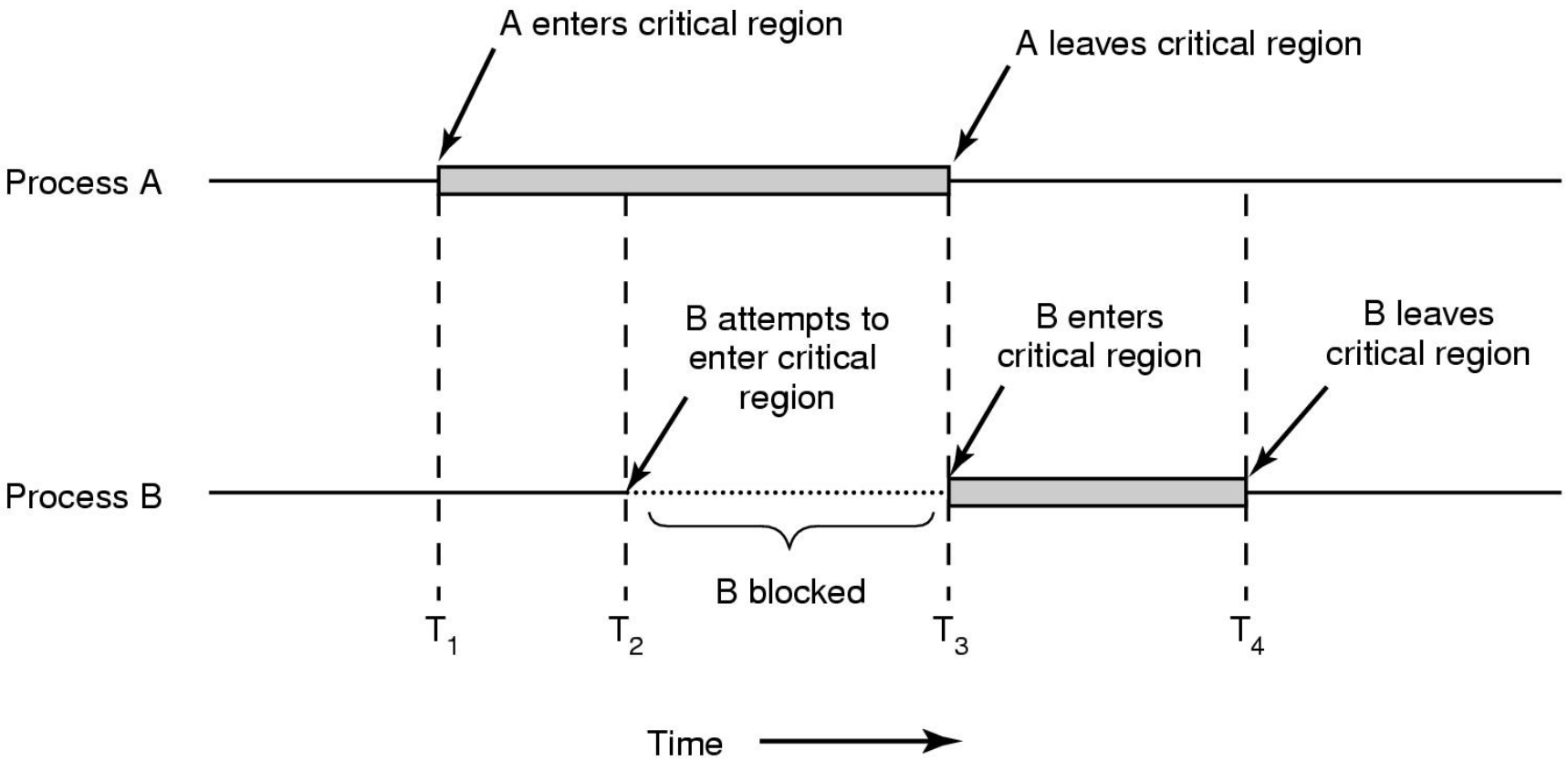
Κοινή μνήμη

- Η πιο γρήγορη μέθοδος για IPC.
- Δε διαμεσολαβεί ο πυρήνας για αυτό το είδος της επικοινωνίας (δεν υπάρχουν κλήσεις συστήματος).
- Απαιτείται ένα είδος συγχρονισμού.
- Η μνήμη είναι κοινή σε μια ή περισσότερες διεργασίες.

Κοινή μνήμη - χρήση

- Ο εξυπηρετητής έχει πρόσβαση σε ένα αντικείμενο κοινής μνήμης χρησιμοποιώντας ένα σημαφόρο.
- Ο εξυπηρετητής διαβάζει από το αρχείο εισόδου.
- Στο αντικείμενο κοινής μνήμης. Το δεύτερο όρισμα προς ανάγνωση, η διεύθυνση του buffer δεδομένων, δείχνει στο αντικείμενο κοινής μνήμης.
- Όταν ολοκληρωθεί η ανάγνωση, ο εξυπηρετητής ειδοποιεί τον πελάτη χρησιμοποιώντας ένα σημαφόρο.
- Ο πελάτης γράφει τα δεδομένα από το αντικείμενο κοινής μνήμης στο αρχείο εξόδου.

Αμοιβαίος αποκλεισμός με χρήση κρίσιμων περιοχών



Αμοιβαίος Αποκλεισμός με «Αναμονή με Απασχόληση»

- **Απενεργοποίηση διακοπών:**

«Μη ελκυστική» διότι μόνον το σύστημα (και όχι οι χρήστες) πρέπει να έχουν τέτοια προνόμια.

- **Μεταβλητές κλειδώματος:**

Η ίδια η μεταβλητή κλειδώματος μπορεί να πέσει θύμα των συνθηκών συναγωνισμού.

- **Αυστηρή εναλλαγή.**
- **Η λύση του Peterson.**
- **Η εντολή TSL/XCHG.**

Αμοιβαίος Αποκλεισμός με αυστηρή εναλλαγή

- **«Αναμονή με Απασχόληση»**

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

A proposed solution to the critical region problem.

(a) Process 0. (b) Process 1.

- Πρόβλημα: Δεν επιτρέπει σε μια διεργασία να εκτελείται ταχύτερα (παραβίαση της 3ης συνθήκης).

Αμοιβαίος Αποκλεισμός με τη λύση του Peterson

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
    int other;                          /* number of the other process */

    other = 1 – process;                /* the opposite of process */
    interested[process] = TRUE;         /* show that you are interested */
    turn = process;                     /* set flag */
    while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
    interested[process] = FALSE;        /* indicate departure from critical region */
}
```

Αμοιβαίος Αποκλεισμός με την ISA εντολή TSL

```
enter_region:
        TSL REGISTER,LOCK       | copy lock to register and set lock to 1
        CMP REGISTER,#0         | was lock zero?
        JNE enter_region        | if it was nonzero, lock was set, so loop
        RET                     | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0            | store a 0 in lock
        RET                     | return to caller
```

- Η εντολή είναι hardware ατομική εντολή και απενεργοποιεί τα interrupt όσο εκτελείται (πιθανόν μειώνει την απόδοση).

Αμοιβαίος Αποκλεισμός με την ISA εντολή XCHG

```
enter_region:
        MOVE REGISTER,#1            | put a 1 in the register
        XCHG REGISTER,LOCK          | swap the contents of the register and lock variable
        CMP REGISTER,#0             | was lock zero?
        JNE enter_region            | if it was non zero, lock was set, so loop
        RET                         | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0                | store a 0 in lock
        RET                         | return to caller
```

Παράδειγμα χρήσης της TSL

```
repeat
        while Test-and-Set(lock) do no-op;
            critical section
        lock := false;
            remainder section
until false;
```

Αμοιβαίος Αποκλεισμός με hardware εντολή

- Ειδικές οδηγίες μηχανήματος (hardware):
  - Πραγματοποιούνται σε ένα μονό κύκλο εντολών.
  - Δεν υπόκειται σε παρεμβολές από άλλες οδηγίες.
  - Ανάγνωση και εγγραφή.
  - Ανάγνωση και δοκιμή.

Ψευδοκώδικας της TSL

```
boolean testset (int i) {
if (i == 0) {
i = 1;
return true;}
else {return false;}
```

Αμοιβαίος Αποκλεισμός με hardware εντολή (προβλήματα)

Απενεργοποίηση διακοπών:

- Η διαδικασία εκτελείται μέχρι να καλέσει μια υπηρεσία του λειτουργικού συστήματος ή μέχρι να διακοπεί.
- Η απενεργοποίηση των διακοπών εγγυάται αμοιβαίο αποκλεισμό.
- Ο επεξεργαστής περιορίζεται στην ικανότητά του να εναλλάσσει προγράμματα.
- Πολυεπεξεργασία: απενεργοποίηση διακοπών σε έναν επεξεργαστή δεν εγγυάται αμοιβαίο αποκλεισμό.

Λήθαργος και αφύπνιση

- Η αναμονή με απασχόληση (busy waiting) δαπανά άσκοπα κύκλους ρολογιού.
- Επίσης, σε περιπτώσεις χρήσης αυστηρών προτεραιοτήτων μπορεί να οδηγήσει σε αδιέξοδα.
- Εισάγονται οι κλήσεις συστήματος sleep και wakeup.

Το πρόβλημα της αντιστροφής προτεραιοτήτων

- Έστω Χ διεργασία με υψηλή προτεραιότητα και Υ με χαμηλή.
- Έστω η Υ είναι ήδη μέσα στην κρίσιμη περιοχή και ενεργοποιείται η Χ για να εκτελεστεί. Ο χρονοδρομολογητής δε θα εκτελέσει ποτέ την Υ και έτσι ποτέ δε θα μπορέσει να φύγει από το κρίσιμο τμήμα, ενώ η Χ ποτέ δε θα μπορέσει να μπει.

Το πρόβλημα της αντιστροφής προτεραιοτήτων στο διάστημα..

- 4ης Ιουλίου 1997, Mars Pathfinder.
- Κοινός Δίαυλος μνήμης .
- Διαδικασία διαχείρισης διαύλου με υψηλή προτεραιότητα (και αμοιβαίο αποκλεισμό).
- Μετεωρολογική διαδικασία με χαμηλή προτεραιότητα.
- Διαδικασία επικοινωνίας με μεσαία προτεραιότητα.
- Σπάνια, η υψηλή προτεραιότητα είχε αποκλειστεί από τη χαμηλή προτεραιότητα και δεν ήταν δυνατή η επικοινωνία.
- Το Watchdog ήταν ενεργοποιημένο και επανεκκινούσε το σύστημα (πολλές φορές).
- Το νέο firmware φορτώθηκε για το VxWorks που επέτρεπε την κληρονομικότητα αμοιβαίας προτεραιότητας από μπλοκαρισμένη διαδικασία.
- Κατά την διάρκεια της δοκιμής στη γη, είχε συμβεί επανεκκίνηση αλλά νόμιζαν πως ήταν φυσιολογικό.

Το πρόβλημα παραγωγού καταναλωτή με sleep/wakeup

```
#define N 100                                   /* number of slots in the buffer */
int count = 0;                                  /* number of items in the buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          item = produce_item( );               /* generate next item */
          if (count == N) sleep( );             /* if buffer is full, go to sleep */
          insert_item(item);                    /* put item in buffer */
          count = count + 1;                    /* increment count of items in buffer */
          if (count == 1) wakeup(consumer);     /* was buffer empty? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          if (count == 0) sleep( );             /* if buffer is empty, got to sleep */
          item = remove_item( );                /* take item out of buffer */
          count = count - 1;                    /* decrement count of items in buffer */
          if (count == N - 1) wakeup(producer); /* was buffer full? */
          consume_item(item);                   /* print item */
     }
}
```

Οδηγεί σε πρόβλημα γιατί γίνεται πρόσβαση στη count χωρίς κλείδωμα. Μπορεί να γίνει interrupt ενώ εκτελείται.

Σηματοφόροι ή Σημαφόροι (Semaphores), 1965 Dijkstra

- Προτείνονται οι εντολές down και up, για να αντικατασταθούν οι sleep και wakeup.
- Χρησιμοποιούνται για να μη χάνονται σήματα αφύπνισης.
- Θεωρούνται ατομικές ενέργειες (αδιαίρετες).
- Όταν εκτελεστεί μια εντολή up, το σύστημα αποφασίζει ποια διεργασία σε λήθαργο θα αφυπνιστεί.

Σηματοφόροι ή Σημαφόροι (Semaphores) (1/3)

- Μια ειδική μεταβλητή που χρησιμοποιείται για σηματοδότηση (signaling).
- Εάν μία διεργασία αναμένει για ένα σήμα (signal), αναστέλλεται (suspended) μέχρι να αποσταλεί αυτό το signal.
- Οι Wait και signal ενέργειες δεν μπορούν να διακοπούν (interrupted).
- Χρησιμοποιείται ουρά για να κρατά τις διεργασίες που αναμένουν στον σημαφόρο.

Σηματοφόροι ή Σημαφόροι (Semaphores) (2/3)

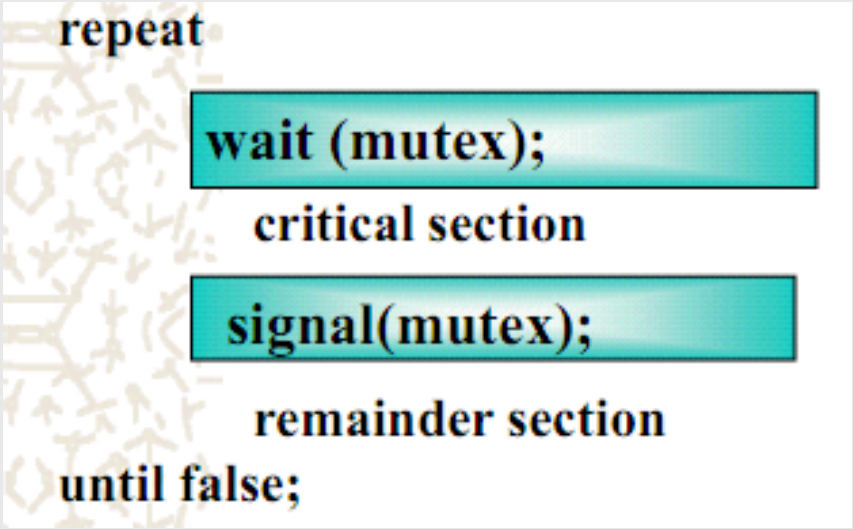
Semaphore : ακέραια μεταβλητή .

- Μπορεί να αρχικοποιηθεί με έναν μη αρνητικό αριθμό.
- Η Wait (ή down) μειώνει την τιμή του σημαφόρου. Αν το S<0 τότε μπλοκάρεται.
- Η Signal (ή up) αυξάνει την τιμή του σημαφόρου.

Χρήση των σημαφόρων

Διαμοιραζόμενη μεταβλητή:

- var mutex : semaphore.
- αρχικά mutex = 1.



Σηματοφόροι ή Σημαφόροι (Semaphores) (3/3)

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item( );                /* generate something to put in buffer */
          down(&empty);                          /* decrement empty count */
          down(&mutex);                          /* enter critical region */
          insert_item(item);                     /* put new item in buffer */
          up(&mutex);                            /* leave critical region */
          up(&full);                             /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                           /* decrement full count */
          down(&mutex);                          /* enter critical region */
          item = remove_item( );                 /* take item from buffer */
          up(&mutex);                            /* leave critical region */
          up(&empty);                            /* increment count of empty slots */
          consume_item(item);                    /* do something with the item */
     }
}
```

- Ο σηματοφόρος mutex είναι δυαδικός και χρησιμεύει στον αμοιβαίο αποκλεισμό
- Οι σηματοφόροι empty και full χρησιμεύουν στο συγχρονισμό
- Επικράτησε στη βιβλιογραφία η ονομασία των δυαδικών σηματοφόρων ως mutex.

mutex

- Σε νεότερα ΛΣ υπάρχουν και εντολές mutex_trylock.

```
mutex_lock:
        TSL REGISTER,MUTEX      | copy mutex to register and set mutex to 1
        CMP REGISTER,#0         | was mutex zero?
        JZE ok                  | if it was zero, mutex was unlocked, so return
        CALL thread_yield       | mutex is busy; schedule another thread
        JMP mutex_lock          | try again
ok:     RET                     | return to caller; critical region entered

mutex_unlock:
        MOVE MUTEX,#0           | store a 0 in mutex
        RET                     | return to caller
```

Mutexes in Pthreads (1/3)

Thread call	Description
Pthread_mutex_init	Create a mutex
Pthread_mutex_destroy	Destroy an existing mutex
Pthread_mutex_lock	Acquire a lock or block
Pthread_mutex_trylock	Acquire a lock or fail
Pthread_mutex_unlock	Release a lock

- **Some of the Pthreads calls relating to mutexes.**

Mutexes in Pthreads (2/3)

Thread call	Description
Pthread_cond_init	Create a condition variable
Pthread_cond_destroy	Destroy a condition variable
Pthread_cond_wait	Block waiting for a signal
Pthread_cond_signal	Signal another thread and wake it up
Pthread_cond_broadcast	Signal multiple threads and wake all of them

- Some of the Pthreads calls relating to condition variables.

Mutexes in Pthreads (3/3)

Using threads to solve the producer-consumer problem.

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000                          /* how many numbers to produce */
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0;                                 /* buffer used between producer and consumer */

void *producer(void *ptr)                       /* produce data */
{       int i;

        for (i= 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);   /* get exclusive access to buffer */
                while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
                buffer = i;                       /* put item in buffer */
                pthread_cond_signal(&condc);      /* wake up consumer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}

void *consumer(void *ptr)                       /* consume data */
{       int i;

        for (i = 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);   /* get exclusive access to buffer */
                while (buffer ==0 ) pthread_cond_wait(&condc, &the_mutex);
                buffer = 0;                       /* take item out of buffer */
                pthread_cond_signal(&condp);      /* wake up producer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}

int main(int argc, char **argv)
{
        pthread_t pro, con;
        pthread_mutex_init(&the_mutex, 0);
        pthread_cond_init(&condc, 0);
        pthread_cond_init(&condp, 0);
        pthread_create(&con, 0, consumer, 0);
        pthread_create(&pro, 0, producer, 0);
        pthread_join(pro, 0);
        pthread_join(con, 0);
        pthread_cond_destroy(&condc);
        pthread_cond_destroy(&condp);
        pthread_mutex_destroy(&the_mutex);
}
```

Semaphores & busy waiting

- Οι semaphores είναι εργαλείο συγχρονισμού που δεν απαιτεί ενεργό αναμονή (busy waiting).
- **Busy Waiting.**
  - Η διεργασία ελέγχει συνεχώς να δει εάν μπορεί να μπει στο κρίσιμο τμήμα της.
  - Η διεργασία δεν μπορεί να κάνει τίποτα το παραγωγικό μέχρι να πάρει «άδεια» για να μπει στο κρίσιμο τμήμα της.

Μηνύματα - Κανάλια Επικοινωνίας

- Δημιουργείται ένα (λογικό) κανάλι (channel) επικοινωνίας μεταξύ των διεργασιών, μέσω του οποίου ανταλλάσσονται τα μηνύματα.
- Το ΛΣ παρέχει τις εξής βασικές πράξεις:
  - open(channel) – δημιουργία/άνοιγμα καναλιού.
  - send(channel, message) – αποστολή μηνύματος.
  - receive(channel, message) – παραλαβή μηνύματος.
  - close(channel) – κλείσιμο καναλιού.
- Συνήθως κάθε κανάλι έχει ένα άκρο (παραλήπτη).
- Ένα κανάλι μπορεί να είναι μονής ή διπλής κατεύθυνσης.

Μηνύματα - Άμεση Επικοινωνία

- Ο παραλήπτης και αποστολέας προσδιορίζονται σαφώς κατά την αποστολή/παραλαβή του μηνύματος.
- Το ΛΣ παρέχει τις εξής βασικές πράξεις:
  - send (P, message) – στείλε μήνυμα στη διεργασία P.
  - receive(Q, message) – λάβε ένα μήνυμα από τη διεργασία Q.
- Τα μηνύματα μπορεί (αλλά δεν πρέπει) να παραδίδονται στον παραλήπτη με την σειρά που στέλνονται από τον αποστολέα.

Μηνύματα - Έμμεση Επικοινωνία

- Τα μηνύματα στέλνονται σε και παραλαμβάνονται από (γραμματο) θυρίδες (mailboxes ή ports).
- Το ΛΣ παρέχει τις εξής βασικές πράξεις:
  - open(mailbox) – δημιουργία/άνοιγμα θυρίδας.
  - send(mailbox, message) – αποστολή μηνύματος.
  - receive(mailbox, message) – παραλαβή μηνύματος.
  - close(mailbox) – κλείσιμο θυρίδας.
- Μια θυρίδα μπορεί να είναι προσβάσιμη από πολλές διεργασίες.
- Συνήθως το μήνυμα που βρίσκεται σε μια θυρίδα το λαμβάνει η διεργασία που «πρώτη» επιχειρεί να παραλάβει ένα μήνυμα.
- Ουσιαστικά επιτρέπεται στο σύστημα να διαλέξει αυθαίρετα τον παραλήπτη. Ο αποστολέας ειδοποιείται για τον παραλήπτη.

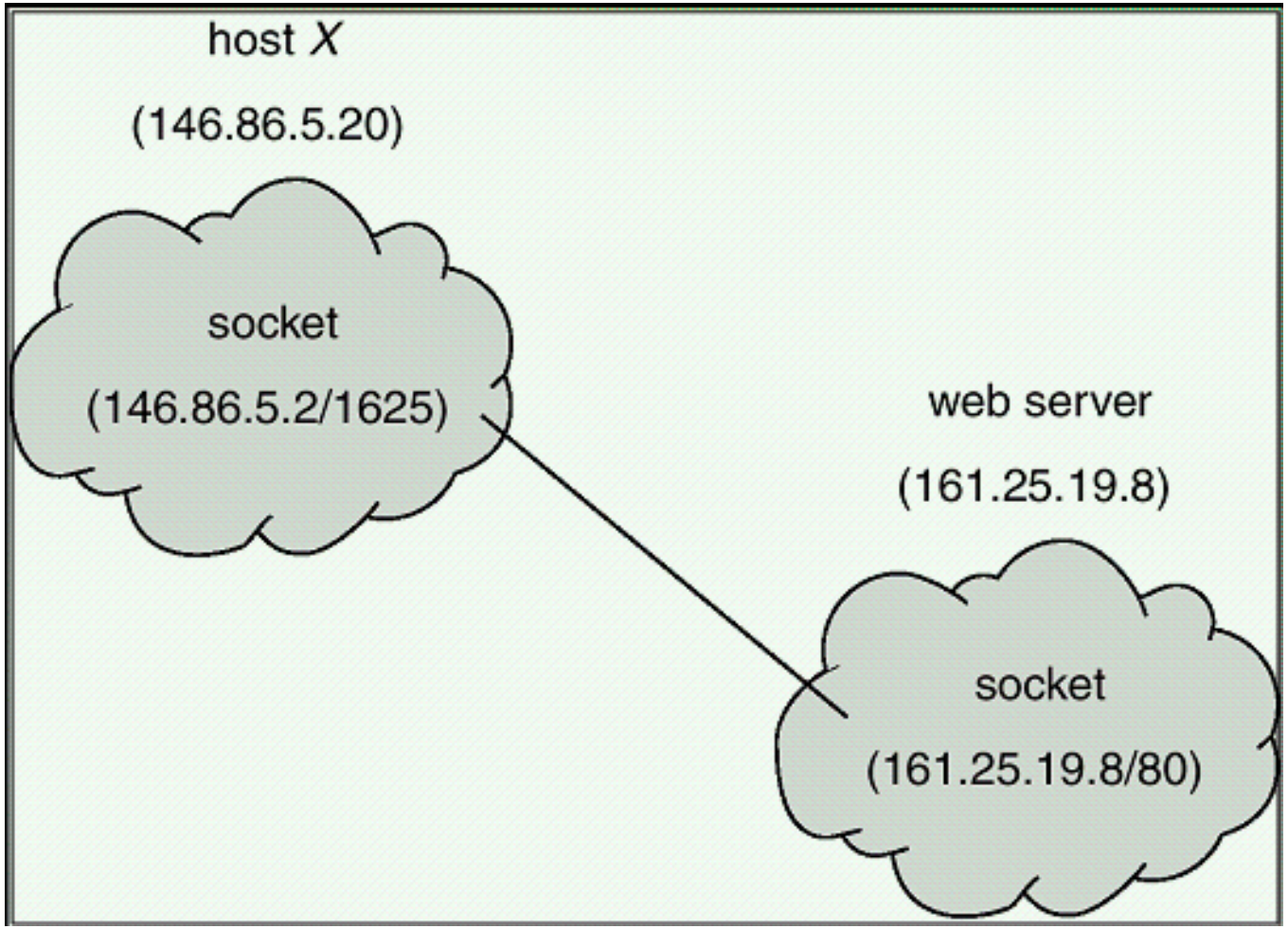
Μεταβίβαση μηνυμάτων στο linux

msgget()	Request message queue id
msgsnd()	relay a message to a particular mailbox
msgrcv()	get a message from a mailbox

Επικοινωνία μέσω δικτύου

- Επικοινωνία ανάμεσα σε διαφορετικούς υπολογιστές γίνεται συνήθως με τα πρωτόκολλα διαδικτύου (Internet protocols, IP).
- Κάθε άκρο μιας σύνδεσης προσδιορίζεται μέσω της διεύθυνσης του υπολογιστή (Internet address) και της θύρας (port) που έχει ανοίξει μια διεργασία.
  - Για παράδειγμα το άκρο 83.212.19.243:1625, αναφέρεται στη θύρα 1625 στον κόμβο με τη διεύθυνση 83.212.19.243.
- Το πιο διαδεδομένο πρωτόκολλο επικοινωνίας του διαδικτύου, το TCP/IP, υποστηρίζει τη δημιουργία συνδέσεων μεταφοράς δεδομένων μεταξύ δύο άκρων.

Επικοινωνία μέσω TCP/IP



Υλοποίηση Μηχανισμών Επικοινωνίας πάνω από Δίκτυο με TCP/IP

- Το TCP/IP είναι η βάση πάνω στην οποία μπορεί να χτιστούν πολλοί διαφορετικοί μηχανισμοί επικοινωνίας διεργασιών.
- Με βάση το TCP/IP μπορούν να υλοποιηθούν και όλοι οι προηγούμενοι μηχανισμοί ανταλλαγής μηνυμάτων.
- Οι αντίστοιχες διασυνδέσεις μεταξύ των διεργασιών υλοποιούνται με συνδέσεις TCP/IP αντί να χρησιμοποιείται μνήμη του τοπικού ΛΣ.
- Η ταχύτητα ανταλλαγής μηνυμάτων μεταξύ των διεργασιών εξαρτάται πλέον από την ταχύτητα μετάδοσης δεδομένων πάνω από το δίκτυο (και την αποδοτικότητα του πρωτοκόλλου TCP/IP) και όχι τόσο από την ταχύτητα αντιγραφής δεδομένων στην μνήμη του ΛΣ.

Τυπικά Ερωτήματα Υλοποίησης

- Υπάρχει «μόνιμη» σύνδεση επικοινωνίας μεταξύ αποστολέα και παραλήπτη;
- Με ποια λογική δημιουργούνται και καταστρέφονται συνδέσεις επικοινωνίας (σε χαμηλό επίπεδο, όπως π.χ. μέσω μνήμης του συστήματος ή δικτυακών συνδέσεων) μεταξύ διεργασιών που ανταλλάσσουν μηνύματα;
- Είναι το μήκος των μηνυμάτων σταθερό και γνωστό εκ των προτέρων, ή μπορεί να είναι μεταβλητό (πως ανακαλύπτεται το μέγεθος μεταβλητών μηνυμάτων την ώρα της εκτέλεσης);
- Τι γίνεται με λάθη, όταν δηλαδή μια από τις δύο διεργασίες τερματιστεί; Πως το αντιλαμβάνεται αυτό η άλλη μεριά;

Συγχρονισμός

- Σύγχρονη επικοινωνία: ο αποστολέας περιμένει μέχρι να λάβει το μήνυμα ο παραλήπτης (blocking send), ο παραλήπτης περιμένει μέχρι να υπάρχει διαθέσιμο μήνυμά προς παραλαβή (blocking receive).
- Ασύγχρονη επικοινωνία: ο αποστολέας συνεχίζει την εκτέλεση ανεξάρτητα από την παραλαβή του μηνύματος (non-blocking send), ο παραλήπτης δεν περιμένει μέχρι να φτάσει κάποιο μήνυμά (non-blocking receive).
- Συνδυασμοί σύγχρονης/ασύγχρονης αποστολής/παραλαβής.

Ενδιάμεση Αποθήκευση (buffering)

- Πόσα μηνύματα μπορούν να αποθηκευτούν προσωρινά, μέχρι να τα ζητήσει ο παραλήπτης;
- Η ουρά των μηνυμάτων μιας σύνδεσης μπορεί να υλοποιηθεί με έναν από τους ακόλουθους τρόπους:
  - **Μηδενική χωρητικότητα – 0 μηνύματα.**

    Ο αποστολέας πρέπει να περιμένει τον παραλήπτη (rendezvous).
  - **Περιορισμένη χωρητικότητα – Ν μηνύματα.**

    Ο αποστολέας περιμένει αν ο σύνδεσμος είναι γεμάτος.
  - **Απεριόριστη χωρητικότητα – απεριόριστο μήκος.**

    Ο αποστολέας δεν περιμένει ποτέ.

Παράδειγμα - Producer/Consumer

- Έστω μια διεργασία (παραγωγός) που επιθυμεί να στέλνει «πακέτα εργασίας» σε μια δεύτερη διεργασία (καταναλωτής).
- Έστω πως η επικοινωνία των διεργασιών πρέπει να υλοποιηθεί με:

  (α) Άμεση ανταλλαγή μηνυμάτων.

  (β) Ασύγχρονο send.

  (γ) Σύγχρονο receive

Παράδειγμα - Consumer

```
typedef struct msg {…} msg;


void consumerCode() {
msg m;
while (1) {
receive(producerID,&m);
processItem(&m);}
}
```

Παράδειγμα - Producer

```
typedef struct msg {…} msg;

void producerCode() {
msg m;
while (1) {
produceItem(&m);
send(consumerID,&m);
}
```

Παράδειγμα

- Έστω πως επιθυμούμε να περιορίσουμε τον αριθμό (έστω Ν) των «πακέτων εργασίας» που βρίσκονται σε αναμονή (έχουν σταλεί από τον παραγωγό χωρίς να τα έχει παραλάβει ο καταναλωτής), αλλά δεν μπορούμε να προσδιορίσουμε την χωρητικότητα του καναλιού επικοινωνίας μεταξύ των δύο διεργασιών.
- Ο συγχρονισμός μπορεί να γίνει βάζοντας τον καταναλωτή να στέλνει αίτηση στον παραγωγό για κάθε «πακέτο εργασίας» που επιτρέπεται να αποθηκευτεί ενδιάμεσα (στο κανάλι επικοινωνίας).
- Στην αρχή πρέπει να σταλούν Ν αιτήσεις, ενώ κάθε φορά που ο καταναλωτής λαμβάνει ένα «πακέτο εργασίας» πρέπει να στέλνει και νέα αίτηση.

Παράδειγμα – Consumer

```
#define N 100
typedef struct msg {…} msg;
typedef int req;
void consumerCode() {
msg m; req ok; int i;

for (i=0; i<N; i++) {send(producerID,&req);}

while (1) {
receive(producerID,&m);
send(producerID,&req);
processItem(&m);}
}
```

Παράδειγμα – Producer

```
typedef struct msg {…} msg;
typedef int req;
void producerCode() {
msg m; req ok;

while (1) {
produceItem(&m);
receive(consumerID,&req);
send(consumerID,&m);
}
```

Μεταβίβαση Μηνυμάτων (Message Passing)

- Ένα μοντέλο IPC, στο οποίο γίνονται δεκτές μόνον οι εντολές:
  - Send(destination,&message).
  - Receive(source,&message).
- Βρίσκουν εφαρμογή σε κατανεμημένα και σε δικτυακά ΛΣ και εισάγουν την έννοια των μηνυμάτων επιβεβαίωσης.
- Ένα από τα πιο γνωστά συστήματα είναι το MPI (Message Passing Interface).

Μεταβίβαση Μηνυμάτων (producer)

```
#define N 100                                   /* number of slots in the buffer */

void producer(void)
{
     int item;
     message m;                                 /* message buffer */

     while (TRUE) {
          item = produce_item( );               /* generate something to put in buffer */
          receive(consumer, &m);                /* wait for an empty to arrive */
          build_message(&m, item);              /* construct a message to send */
          send(consumer, &m);                   /* send item to consumer */
     }
}
```

Μεταβίβαση Μηνυμάτων (consumer)

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

        for (i = 0; i < N; i++) send(producer, &m);  /* send N empties */
        while (TRUE) {
                receive(producer, &m);          /* get message containing item */
                item = extract_item(&m);        /* extract item from message */
                send(producer, &m);             /* send back empty reply */
                consume_item(item);             /* do something with the item */
        }
}
```

Το πρόβλημα των αναγνωστών/συγγραφέων (1/2)

- Οποιοσδήποτε αριθμός αναγνωστών μπορεί να διαβάζει παράλληλα το αρχείο.
- Μόνο ένας συγγραφέας κάθε φορά μπορεί να γράφει στο αρχείο.
- Εάν ένας συγγραφέας γράφει στο αρχείο, κανένας αναγνώστης δεν μπορεί να το διαβάζει.
- Όταν ο συγγραφέας ενημερώσει για τη λήξη της συγγραφής του, επιλέγεται ένας νέος αναγνώστης ή συγγραφέας μέσω ενός scheduler.

Το πρόβλημα των αναγνωστών/συγγραφέων (2/2)

- Διαμοιραζόμενα δεδομένα -

  var mutex, wrt : semaphore (=1);

  readcount : integer (=0);

Διεργασία Συγγραφέα.

```
wait(wrt);
...
writing is performed;
...
signal(wrt);
```

Διεργασία Αναγνώστη.

```
wait(mutex);
readcount := readcount+1;
if readcount=1 then wait(wrt);
signal(mutex);
   ...
reading is performed;
   ...
wait(mutex);
readcount := readcount-1;
if readcount=0 then signal(wrt);
signal(mutex);
```

Άσκηση

- Το παρακάτω ζεύγος διεργασιών διαμοιράζεται μια κοινή μεταβλητή X με αρχική τιμή 5. Οι εντολές εντός κάθε διεργασίας εκτελούνται σειριακά, αλλά οι εντολές της διεργασίας Α είναι δυνατόν να εκτελεσθούν με οποιαδήποτε σειρά ως προς τις εντολές της διεργασίας Β.

[Διεργασία Α]	[Διεργασία Β]
int Y;	int Z;
A1: Y = 2*X;	B1: Z = X+1;
A2: X = Y;	B2: X = Z;

- Πόσες διαφορετικές τιμές της μεταβλητής X μπορούν να προκύψουν όταν ολοκληρωθεί η εκτέλεση και των δύο διεργασιών.

Άσκηση - Απάντηση

Υπάρχουν 4 δυνατές τιμές για το Χ. Ιδού όλοι οι δυνατοί συνδυασμοί της σειράς εκτέλεσης των εντολών των διεργασιών Α και Β:

- Α1 Α2 Β1 Β2: Χ = 11 .
- Α1 Β1 Α2 Β2: Χ = 6 .
- Α1 Β1 Β2 Α2: Χ = 10 .
- Β1 Α1 Β2 Α2: Χ = 10 .
- Β1 Α1 Α2 Β2: Χ = 6 .
- Β1 Β2 Α1 Α2: Χ = 12 .

Τέλος Ενότητας