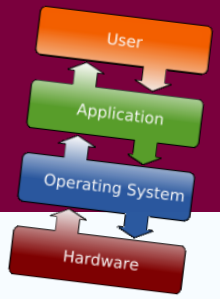


Συνταύτιση:
- Αμοιβαίος Αποκλεισμός
- Συγχρονισμός

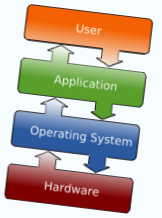
Ανδρέας Λ. Συμεωνίδης

Αν. Καθηγητής
Τμήμα Ηλεκτρολόγων Μηχ/κών
&
Μηχ/κών Υπολογιστών, Α.Π.Θ.

Email: asymeon@eng.auth.gr

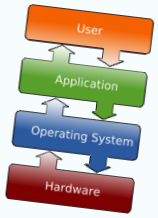
Στόχοι της Δ-5

- Να ορίσει την συνταύτιση και τις σχετικές με αυτήν έννοιες:
  - Αμοιβαίος αποκλεισμός
  - Αδιέξοδα
  - Livelocks
  - Παρατεταμένη στέρηση
- Να αναλύσει τις διαφορετικές τεχνικές (υλικού/λογισμικού) και τις διαφορετικές δομές που χρησιμοποιούνται για την υλοποίηση του αμοιβαίου αποκλεισμού:
  - Σημαφόροι
  - Παρακολουθητές
  - Μηνύματα
- Να παρουσιάσει τις δομές αυτές πάνω σε δυο κλασικά προβλήματα συνταύτισης:
  - Παραγωγού/Καταναλωτή
  - Αναγνωστών/Συγγραφέων



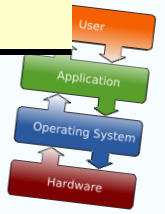
Συνταύτιση

- Πολλαπλές εφαρμογές
- Δομημένες εφαρμογές
- Δομή λειτουργικού συστήματος



Βασικοί Όροι

Ατομική λειτουργία	Μια ακολουθία από εντολές οι οποίες φαίνονται αδιαίρετες. Καμία άλλη διεργασία δεν μπορεί να δει μια ενδιάμεση κατάσταση ή να διακόψει τη λειτουργία.
Κρίσιμο τμήμα	Ένα τμήμα κώδικα μέσα σε μια διεργασία το οποίο απαιτεί πρόσβαση σε διαμοιραζόμενους πόρους και το οποίο δεν πρέπει να εκτελεστεί όσο μια άλλη διεργασία είναι σε ένα αντίστοιχο τμήμα κώδικα
Αδιέξοδο	Μια κατάσταση στην οποία δυο ή περισσότερες διεργασίες δεν μπορούν να συνεχίσουν γιατί κάθε μια από αυτές περιμένουν από την άλλη (άλλες) να κάνουν κάτι
Livelock	Μια κατάσταση στην οποία δυο ή περισσότερες διεργασίες αλλάζουν την κατάστασή τους σε σχέση με κάποια άλλη διεργασία, χωρίς να εκτελούν κάποια χρήσιμη λειτουργία
Αμοιβαίος αποκλεισμός	Η απαίτηση όταν μια διεργασία είναι στο κρίσιμο τμήμα, καμία άλλη δεν είναι σε κρίσιμο τμήμα που ζητά προσπέλαση στους ίδιους διαμοιραζόμενους πόρους
Race condition	Μια κατάσταση στην οποία πολλές διεργασίες ή νήματα διαβάζουν και γράφουν σε διαμοιραζόμενα δεδομένα και το τελικό αποτέλεσμα εξαρτάται από τον σχετικό χρονισμό εκτέλεσής τους.
Παρατεταμένη στέρηση	Μια κατάσταση στην οποία μια διεργασία που μπορεί να εκτελεστεί παραβλέπεται επ' αόριστον από τον χρονοπρογραμματιστή.

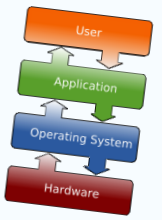


Οι δυσκολίες στην Συνταύτιση

- Ο διαμοιρασμός των γενικών πόρων (global resources)
- Το λειτουργικό σύστημα πρέπει να διαχειρίζεται με βέλτιστο τρόπο τους πόρους
- Δυσκολία στην εύρεση των σφαλμάτων προγραμματισμού

Ένα απλό παράδειγμα

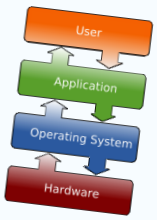
```
void echo()
{
  chin = getchar();
  chout = chin;
  putchar(chout);
}
```



Ένα απλό παράδειγμα (συν.)

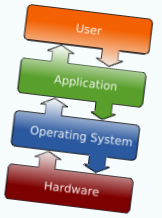
```
Process P1
    …
chin = getchar();
    …
chout = chin;
putchar(chout);
    …
    …
```

```
Process P2
    …
    …
chin = getchar();
chout = chin;
    …
putchar(chout);
    …
```



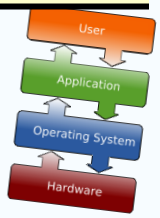
Θέματα του ΛΣ

- Παρακολούθηση των διαφόρων διεργασιών
- Ανάθεση και ελευθέρωση πόρων
  - Χρόνος επεξεργαστή
  - Μνήμη
  - Αρχεία
  - Συσκευές Ε/Ε
- Προστασία δεδομένων και πόρων
- Η έξοδος μιας διεργασίας πρέπει να είναι ανεξάρτητη από την ταχύτητα εκτέλεσης άλλων ταυτόχρονων διεργασιών



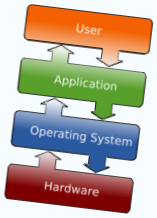
Αλληλεπίδραση Διεργασιών

Βαθμός Γνώσης	Συσχέτιση	Επιρροή της μιας στην άλλη	Ενδεχόμενα προβλήματα ελέγχου
Οι διεργασίες αγνοούν η μια την άλλη	Ανταγωνισμός	- Τα αποτελέσματα μιας διεργασίας είναι ανεξάρτητα των λειτουργιών άλλων διεργασιών - Ο χρονοπρογραμματισμός της διεργασίας μπορεί να επηρεαστεί	- Αμοιβαίος Αποκλεισμός - Αδιέξοδο (ανανεώσιμος πόρος) - Παρατεταμένη στέρηση
Οι διεργασίες γνωρίζουν έμμεσα η μια την άλλη	Συνεργασία μέσω διαμοίρασης	- Τα αποτελέσματα μιας διεργασίας μπορεί να εξαρτώνται από πληροφορίες άλλων διεργασιών - Ο χρονοπρογραμματισμός της διεργασίας μπορεί να επηρεαστεί	- Αμοιβαίος Αποκλεισμός - Αδιέξοδο (ανανεώσιμος πόρος) - Παρατεταμένη στέρηση - Συνοχή δεδομένων
Οι διεργασίες γνωρίζουν άμεσα η μια την άλλη	Συνεργασία μέσω επικοινωνίας	- Τα αποτελέσματα μιας διεργασίας μπορεί να εξαρτώνται από πληροφορίες άλλων διεργασιών - Ο χρονοπρογραμματισμός της διεργασίας μπορεί να επηρεαστεί	- Αδιέξοδο (καταναλώσιμος πόρος) - Παρατεταμένη στέρηση



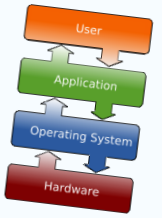
Ανταγωνισμός διεργασιών για πόρους

- Αμοιβαίος αποκλεισμός
  - Κρίσιμα τμήματα
- Αδιέξοδα (Deadlock)
- Παρατεταμένη Στέρηση (Starvation)



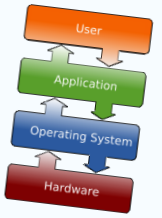
Απαιτήσεις για αμοιβαίο αποκλεισμό

- Μόνο μια διεργασία τη φορά επιτρέπεται στο κρίσιμο τμήμα για έναν πόρο
- Μια διεργασία που σταματά στο μη-κρίσιμο τμήμα, πρέπει να το κάνει χωρίς να εμπλακεί με την εκτέλεση των υπολοίπων διεργασιών
- Όχι αδιέξοδο ή παρατεταμένη στέρηση



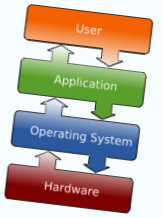
Απαιτήσεις για αμοιβαίο αποκλεισμό (συν.)

- Η πρόσβαση μιας διεργασίας στο κρίσιμο τμήμα δεν πρέπει να καθυστερήσει στην περίπτωση που δεν υπάρχει άλλη διεργασία που να το χρησιμοποιεί.
- Δεν γίνονται καθόλου υποθέσεις σχετικά με τις ταχύτητες των διεργασιών ή τον αριθμό των διεργασιών
- Μια διεργασία παραμένει στο κρίσιμο τμήμα της μόνο για πεπερασμένο χρόνο



Αμοιβαίος αποκλεισμός: Υποστήριξη Υλικού

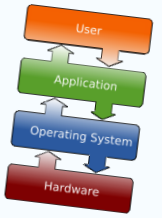
- Απενεργοποίηση διακοπών
  - Μια διεργασία εκτελείται μέχρι να εκκινήσει μια υπηρεσία του ΛΣ ή μέχρι να διακοπεί
  - Η απενεργοποίηση των διακοπών εξασφαλίζει τον αμοιβαίο αποκλεισμό
  - Ο επεξεργαστής περιορίζεται στο να συνδυάζει προγράμματα
  - Δεν λειτουργεί σε πολύ-επεξεργαστικές αρχιτεκτονικές



Αμοιβαίος αποκλεισμός: Υποστήριξη Υλικού (συν.)

- Εντολή Σύγκρισης και Εναλλαγής (Compare & Swap)

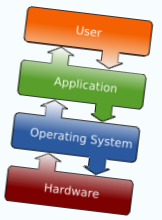
```
int compare_and_swap (int *word, int testval, int newval)
{
  int oldval;
  oldval = *word;
  if (oldval == testval) *word = newval;
  return oldval;
}
```



Αμοιβαίος αποκλεισμός: Υποστήριξη Υλικού (συν.)

- Εντολή ανταλλαγής (Exchange)

```
void exchange (int register, int memory)
{
  int temp;
  temp = memory;
  memory = register;
  register = temp;
}
```



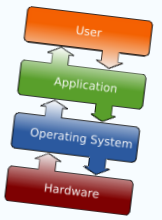
Αμοιβαίος αποκλεισμός

```
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
  while (true) {
    while (compare_and_swap(bolt, 0, 1) == 1)
        /* do nothing */;
    /* critical section */;
    bolt = 0;
    /* remainder */;
  }
}
void main()
{
  bolt = 0;
  parbegin (P(1), P(2), . . . ,P(n));

}
```

Εντολή Σύγκρισης και Εναλλαγής

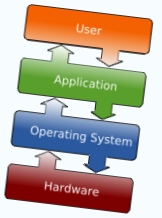
```
/* program mutualexclusion */
int const n = /* number of processes**/;
int bolt;
void P(int i)
{
  int keyi = 1;
  while (true) {
    do exchange (keyi, bolt)
    while (keyi != 0);
    /* critical section */;
    bolt = 0;
    /* remainder */;
  }
}
void main()
{
  bolt = 0;
  parbegin (P(1), P(2), . . ., P(n));
}
```

Εντολή ανταλλαγής



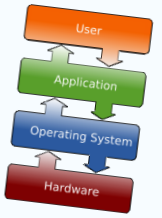
Αμοιβαίος αποκλεισμός: Προσέγγιση Λογισμικού

- Πλεονεκτήματα
  - Μπορεί να εφαρμοστεί σε οποιοδήποτε αριθμό διεργασιών είτε σε έναν ή πολλούς επεξεργαστές που διαμοιράζονται την κύρια μνήμη
  - Είναι απλό και κατά συνέπεια εύκολο να επαληθευτεί
  - Μπορεί να εφαρμοστεί για να υποστηρίξει πολλαπλά κρίσιμα τμήματα



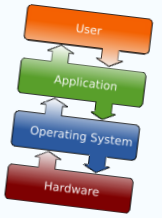
Αμοιβαίος αποκλεισμός: Προσέγγιση Λογισμικού (συν.)

- Μειονεκτήματα
  - Ενεργός Αναμονή καταναλώνει πολύ χρόνο επεξεργαστή
  - Η παρατεταμένη στέρηση είναι πιθανή στην περίπτωση που μια διεργασία αφήνει ένα κρίσιμο τμήμα και αναμένουν περισσότερες της μιας διεργασίες
  - Αδιέξοδο



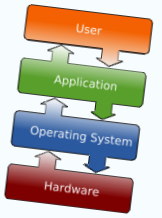
Σημαφόροι (Semaphores)

- Μια ειδική μεταβλητή που χρησιμοποιείται για σηματοδότηση
- Αν μια διεργασία περιμένει για ένα σήμα, αναστέλλεται μέχρι να σταλεί το σήμα
- Ο σημαφόρος είναι μια μεταβλητή με ακέραια τιμή
  - Μπορεί να αρχικοποιηθεί με μια μη-αρνητική τιμή
  - Η λειτουργία αναμονής μειώνει την τιμή του σημαφόρου
  - Η λειτουργία σήματος αυξάνει την τιμή του σημαφόρου



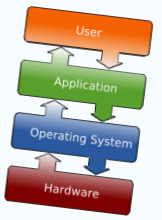
Αρχές των σημαφόρων

```
struct semaphore {
      int count;
      queueType queue;
};
void semWait(semaphore s)
{
      s.count--;
      if (s.count < 0) {
            /* place this process in s.queue */;
            /* block this process */;
      }
}
void semSignal(semaphore s)
{
      s.count++;
      if (s.count <= 0) {
            /* remove a process P from s.queue */;
            /* place process P on ready list */;
      }
}
```



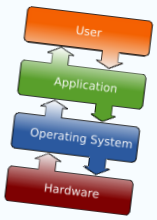
Αρχές των δυαδικών σημαφόρων

```
struct binary_semaphore {
      enum {zero, one} value;
      queueType queue;
};
void semWaitB(binary_semaphore s)
{
      if (s.value == one)
            s.value = zero;
      else {
               /* place this process in s.queue */;
               /* block this process */;
      }
}
void semSignalB(semaphore s)
{
      if (s.queue is empty())
            s.value = one;
      else {
               /* remove a process P from s.queue */;
               /* place process P on ready list */;
      }
}
```

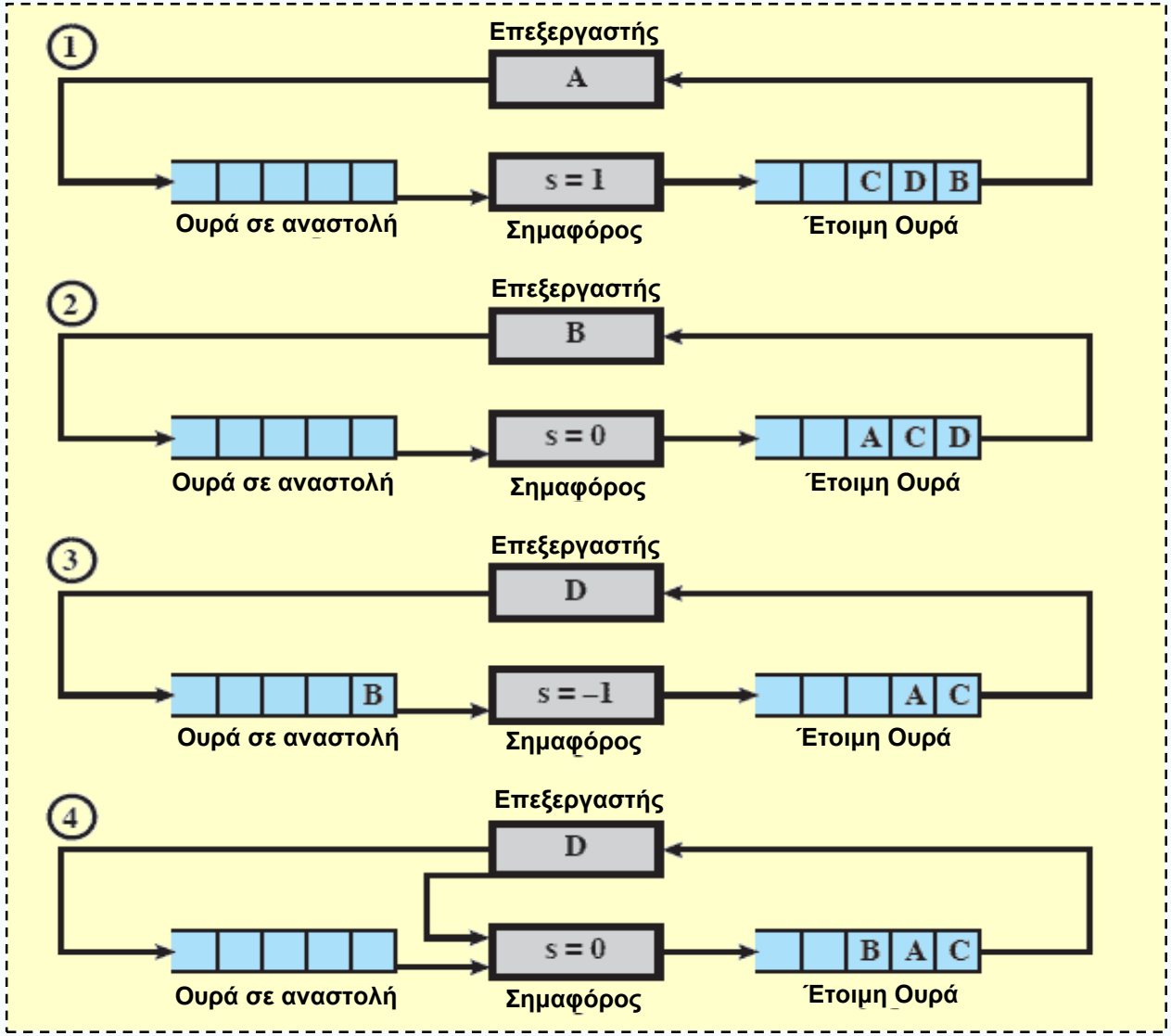


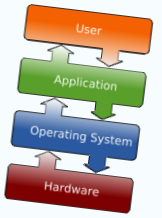
Αμοιβαίος αποκλεισμός χρησιμοποιώντας σημαφόρους

```
/* program mutualexclusion */
const int n = /* number of processes  */;
semaphore s = 1;
void P(int i)
{
     while (true) {
          semWait(s);
          /* critical section   */;
          semSignal(s);
          /* remainder   */;
     }
}
void main()
{
     parbegin (P(1), P(2), . . ., P(n));
}
```

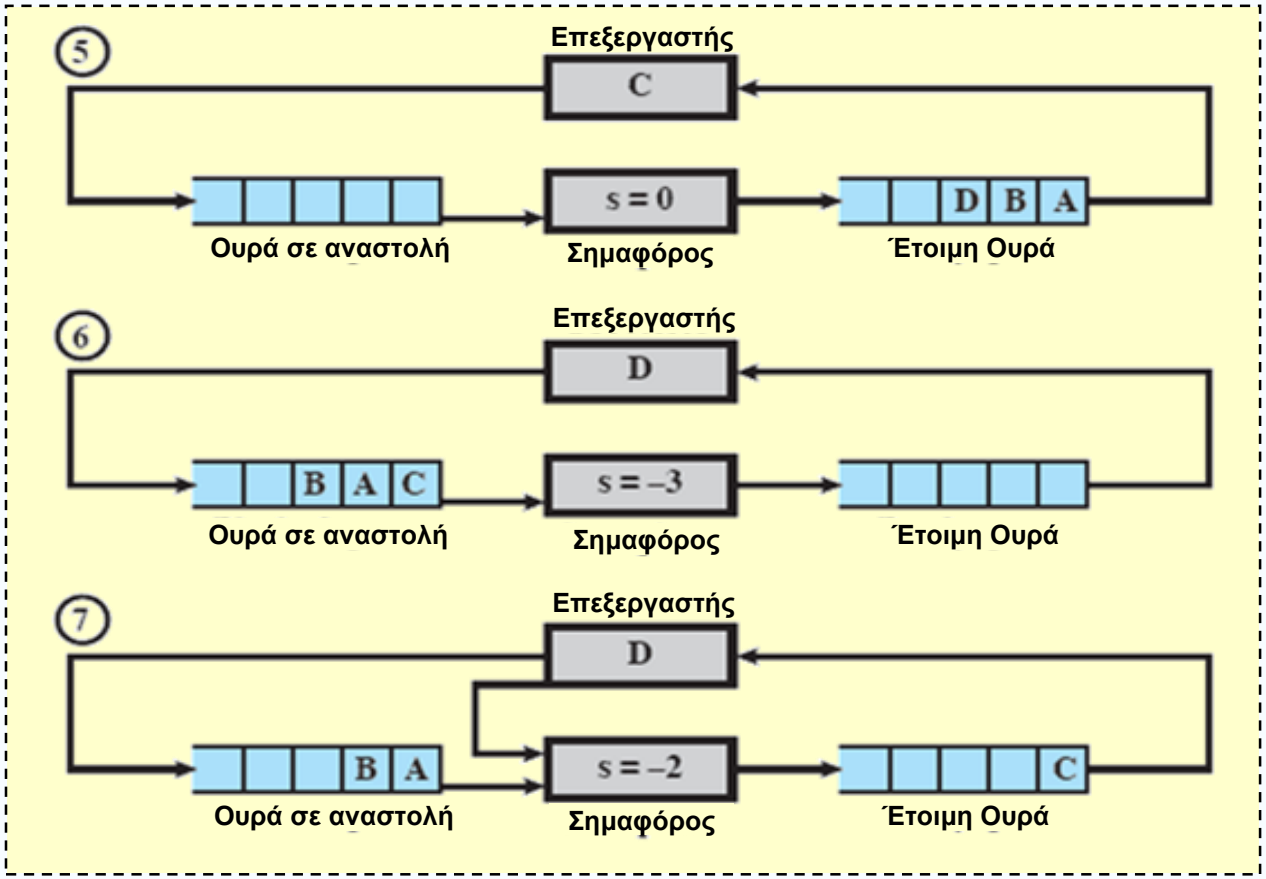


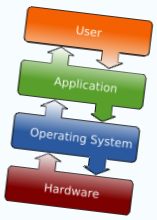
Παράδειγμα ενός μηχανισμού σε σημαφόρους



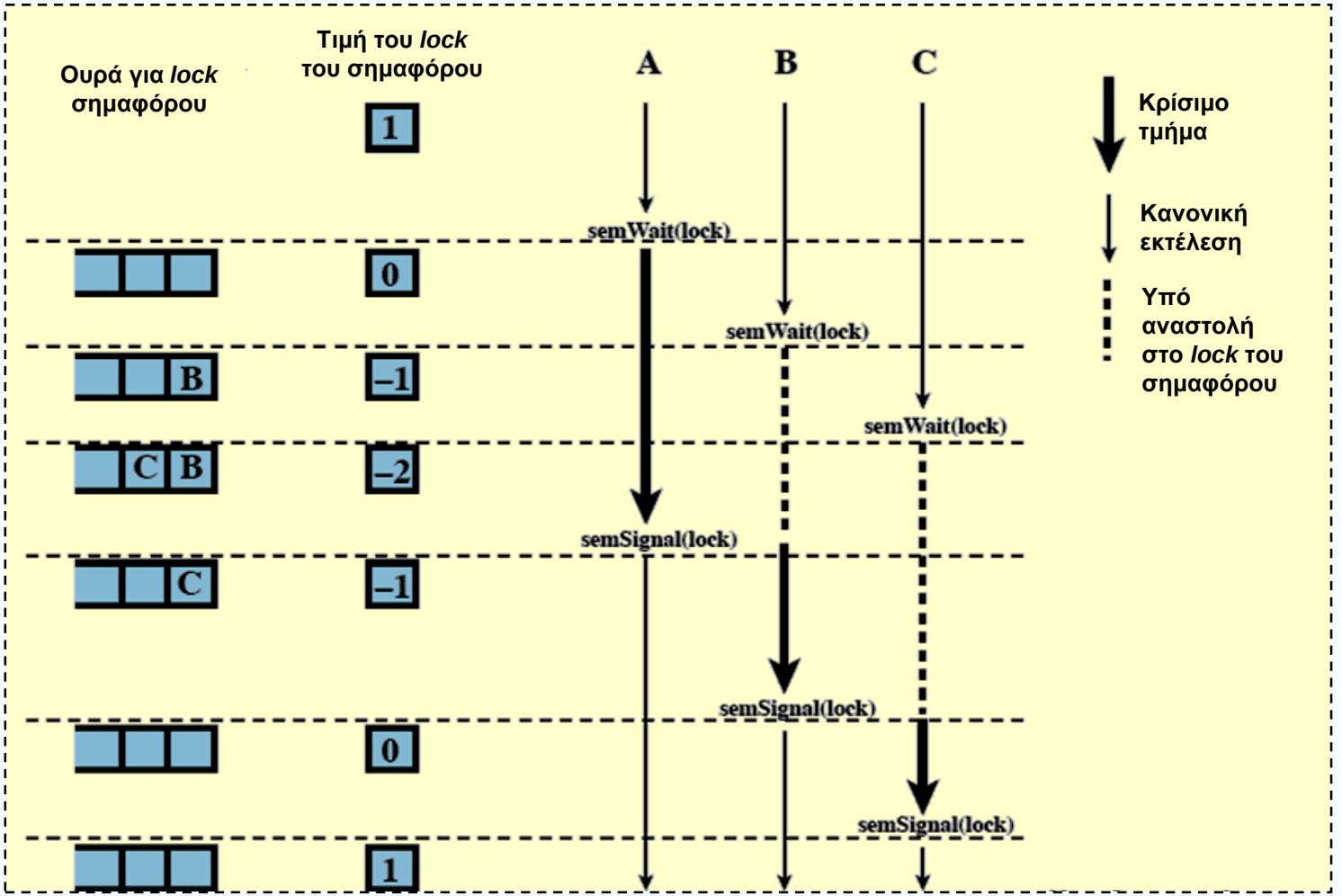


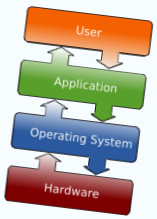
Παράδειγμα ενός μηχανισμού σε σημαφόρους (συν.)





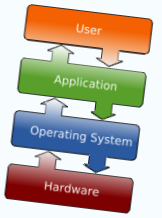
Διεργασίες που χρησιμοποιούν σημαφόρους





Το πρόβλημα Παραγωγού/Καταναλωτή

- Ένας ή περισσότεροι παραγωγοί δημιουργούν δεδομένα και τα αποθηκεύουν σε ένα buffer
- Ένας μοναδικός καταναλωτής παίρνει δεδομένα από το buffer, ένα τη φορά
- Μόνο ένας παραγωγός ή καταναλωτής μπορεί να προσπελαύνει το buffer τη φορά
- Ο παραγωγός δεν μπορεί να προσθέσει δεδομένα σε ένα γεμάτο buffer και ο καταναλωτής δεν μπορεί να πάρει δεδομένα από ένα άδειο buffer

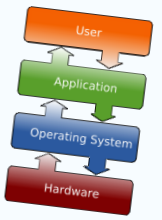


Παραγωγός

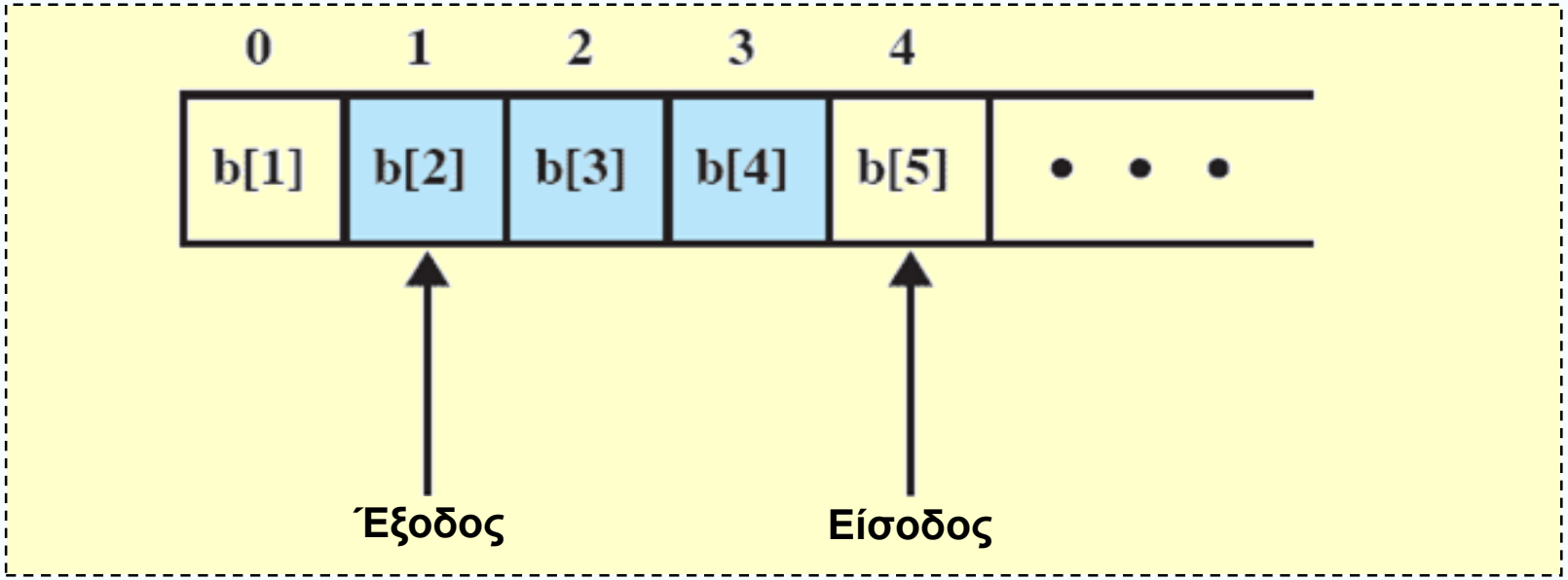
```
while (true) {
   /* produce item v */
   b[in] = v;
   in++;
}
```

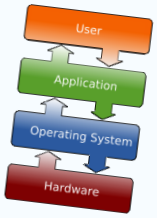
Καταναλωτής

```
while (true) {
   while (in <= out)
        /*do  nothing */;
   w = b[out];
   out++;
   /* consume item w */
}
```



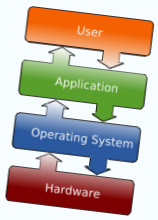
Αποθήκη (Buffer)





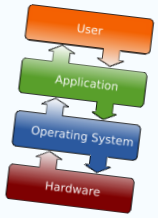
Λάθος λύση

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer()
{
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
     }
}
void consumer()
{
     semWaitB(delay);
     while (true) {
          semWaitB(s);
          take();
          n--;
          semSignalB(s);
          consume();
          if (n==0) semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```



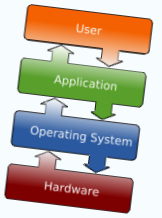
Σωστή λύση

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer()
{
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
     }
}
void consumer()
{
     int m; /* a local variable */
     semWaitB(delay);
     while (true)  {
          semWaitB(s);
          take();
          n--;
          m = n;
          semSignalB(s);
          consume();
          if (m==0) semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```



Σημαφόροι

```
/* program  producerconsumer */
semaphore n = 0, s = 1;
void producer()
{
     while (true) {
          produce();
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```

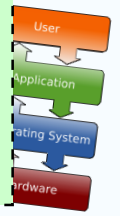


Παραγωγός με κυκλικό buffer

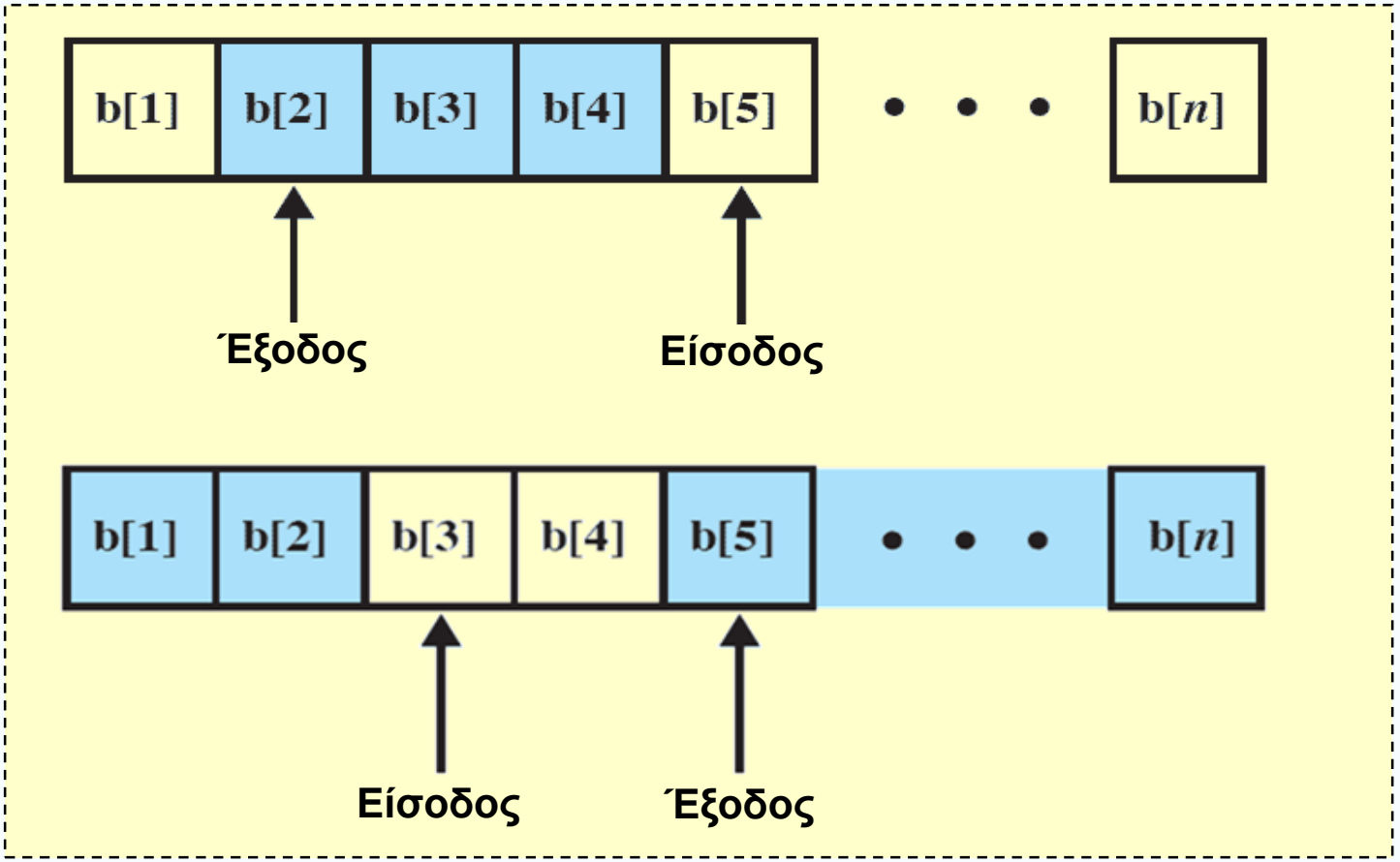
```
while (true) {
   /* produce item v */
   while ((in + 1) % n == out)        /* do nothing */;
   b[in] = v;
   in = (in + 1) % n
}
```

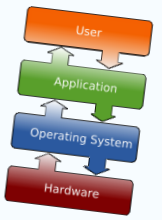
Καταναλωτής με κυκλικό buffer

```
while (true) {
  while (in == out)
       /* do nothing */;
  w = b[out];
  out = (out + 1) % n;  /* consume item w */
}
```



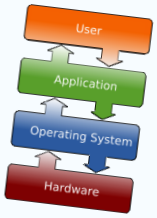
Κυκλικός Buffer





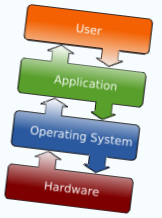
Σημαφόροι

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
void producer()
{
     while (true) {
          produce();
          semWait(e);
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          semSignal(e);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```

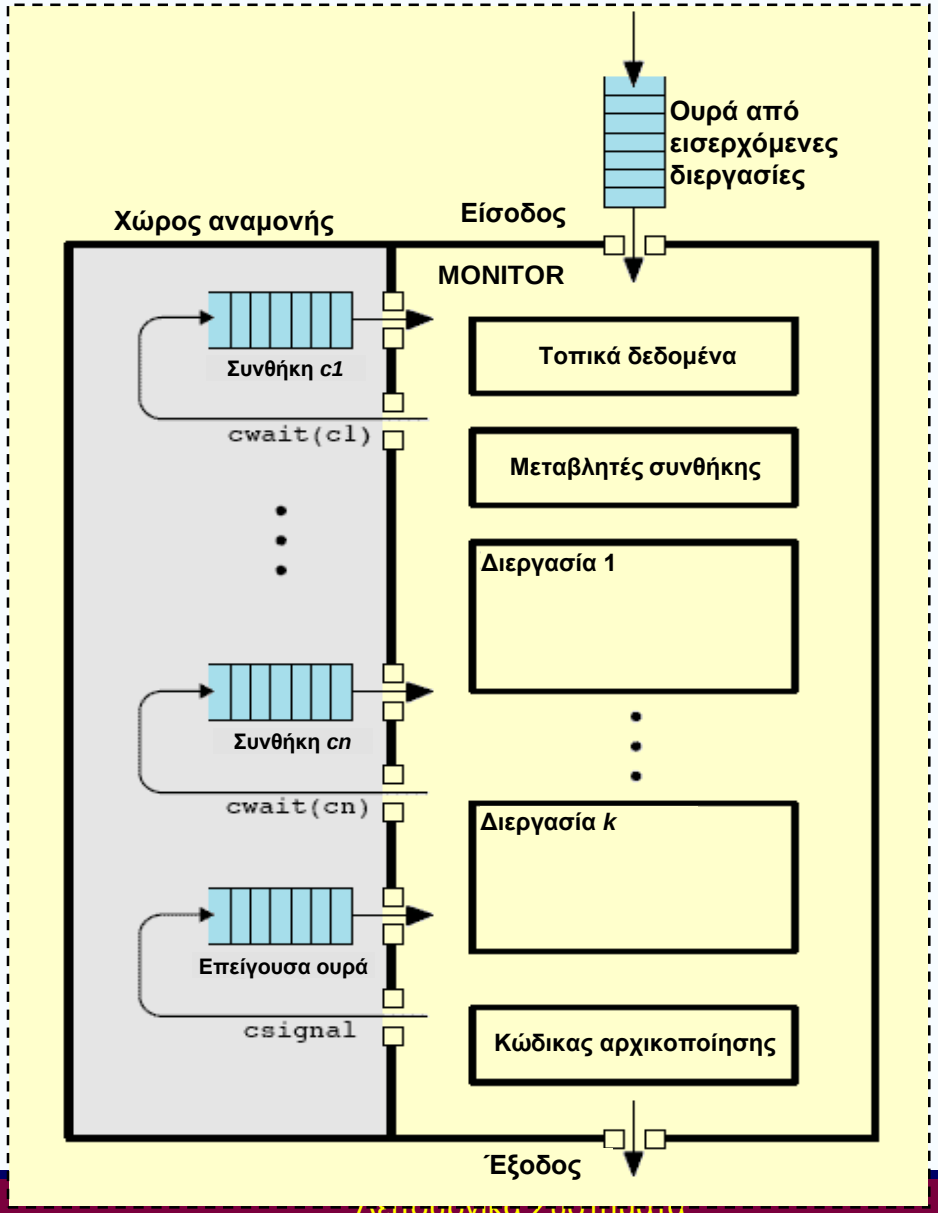


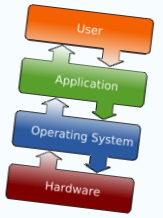
Παρακολουθητές (Monitors)

- Ο παρακολουθητής είναι λογισμικό
- Κύρια χαρακτηριστικά
  - Οι τοπικές μεταβλητές δεδομένων είναι προσπελάσιμες μόνο μέσω του παρακολουθητή
  - Η διεργασία εισάγεται στον παρακολουθητή ενεργοποιώντας μια από τις διαδικασίες του
  - Μόνο μια διεργασία μπορεί να εκτελείται στον παρακολουθητή κάθε φορά



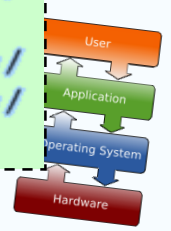
Η δομή του παρακολουθητή





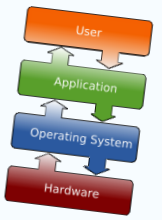
Η λύση χρησιμοποιώντας Παρακολουθητή

```
/* program producerconsumer */
monitor boundedbuffer;
char buffer [N];                                    /* space for N items */
int nextin, nextout;                                /* buffer pointers */
int count;                                          /* number of items in buffer */
cond notfull, notempty;          /* condition variables for synchronization */

void append (char x)
{
     if (count == N) cwait(notfull);       /* buffer is full; avoid overflow */
     buffer[nextin] = x;
     nextin = (nextin + 1) % N;
     count++;
     /* one more item in buffer */
     csignal(notempty);                          /* resume any waiting consumer */
}
void take (char x)
{
     if (count == 0) cwait(notempty);    /* buffer is empty; avoid underflow */
     x = buffer[nextout];
     nextout = (nextout + 1) % N;
     count--;                                       /* one fewer item in buffer */
     csignal(notfull);                           /* resume any waiting producer */
}
{                                                                /* monitor body */
     nextin = 0; nextout = 0; count = 0;          /* buffer initially empty */
}
```



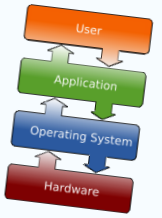
Η λύση χρησιμοποιώντας Παρακολουθητή (συν.)

```
void producer()
{
     char x;
     while (true) {
     produce(x);
     append(x);
     }
}
void consumer()
{
     char x;
     while (true) {
       take(x);
       consume(x);
     }
}
void main()
{
     parbegin (producer, consumer);
}
```



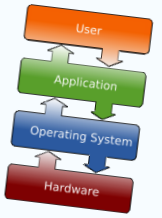
Ένας Παρακολουθητής περιορισμένου Buffer

```
void append (char x)
{
     while(count == N) cwait(notfull);   /* buffer is full; avoid overflow */
     buffer[nextin] = x;
     nextin = (nextin + 1) % N;
     count++;                                  /* one more item in buffer */
     cnotify(notempty);                       /* notify any waiting consumer */
}

void take (char x)
{
     while(count == 0) cwait(notempty); /* buffer is empty; avoid underflow */
     x = buffer[nextout];
     nextout = (nextout + 1) % N;
     count--;                                  /* one fewer item in buffer */
     cnotify(notfull);                        /* notify any waiting producer */
}
```



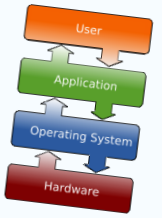
Μεταβίβαση μηνυμάτων

- Επιβάλλουν αμοιβαίο αποκλεισμό
- Ανταλλάσσουν πληροφορίες
  - `send (destination, message)`
  - `receive (source, message)`



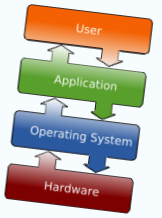
Συγχρονισμός

- Ο αποστολέας και ο παραλήπτης μπορεί ή μπορεί να μην είναι σε αναστολή (να περιμένει για μήνυμα)
- Send σε αναστολή, receive σε αναστολή
  - Τόσο ο αποστολέας όσο και ο παραλήπτης είναι σε αναστολή μέχρι να έρθει μήνυμα
  - Το ονομαζόμενο “rendezvous”



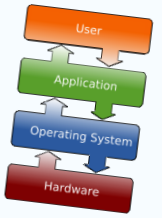
Συγχρονισμός (συν.)

- Send σε μη αναστολή, receive σε αναστολή
  - Ο αποστολέας συνεχίζει
  - Ο παραλήπτης είναι σε αναστολή μέχρι να φτάσει το ζητούμενο μήνυμα
- Send σε μη αναστολή, receive σε μη αναστολή
  - Κανένα από τα δυο μέρη δεν είναι αναγκασμένο να περιμένει



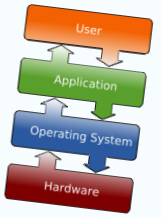
Διευθυνσιοδότηση (Addressing)

- Άμεση διευθυνσιοδότηση (Direct addressing)
  - Η **`Send`** περιέχει έναν συγκεκριμένο προσδιοριστή για τη διεργασία προορισμό
  - Η **`Receive`** θα μπορούσε να γνωρίζει από πριν ποια διεργασία περιμένει το κάθε μήνυμα
  - Η **`Receive`** θα μπορούσε να χρησιμοποιήσει μια παράμετρο αφετηρίας όταν εκτελεστεί η λειτουργία λήψης

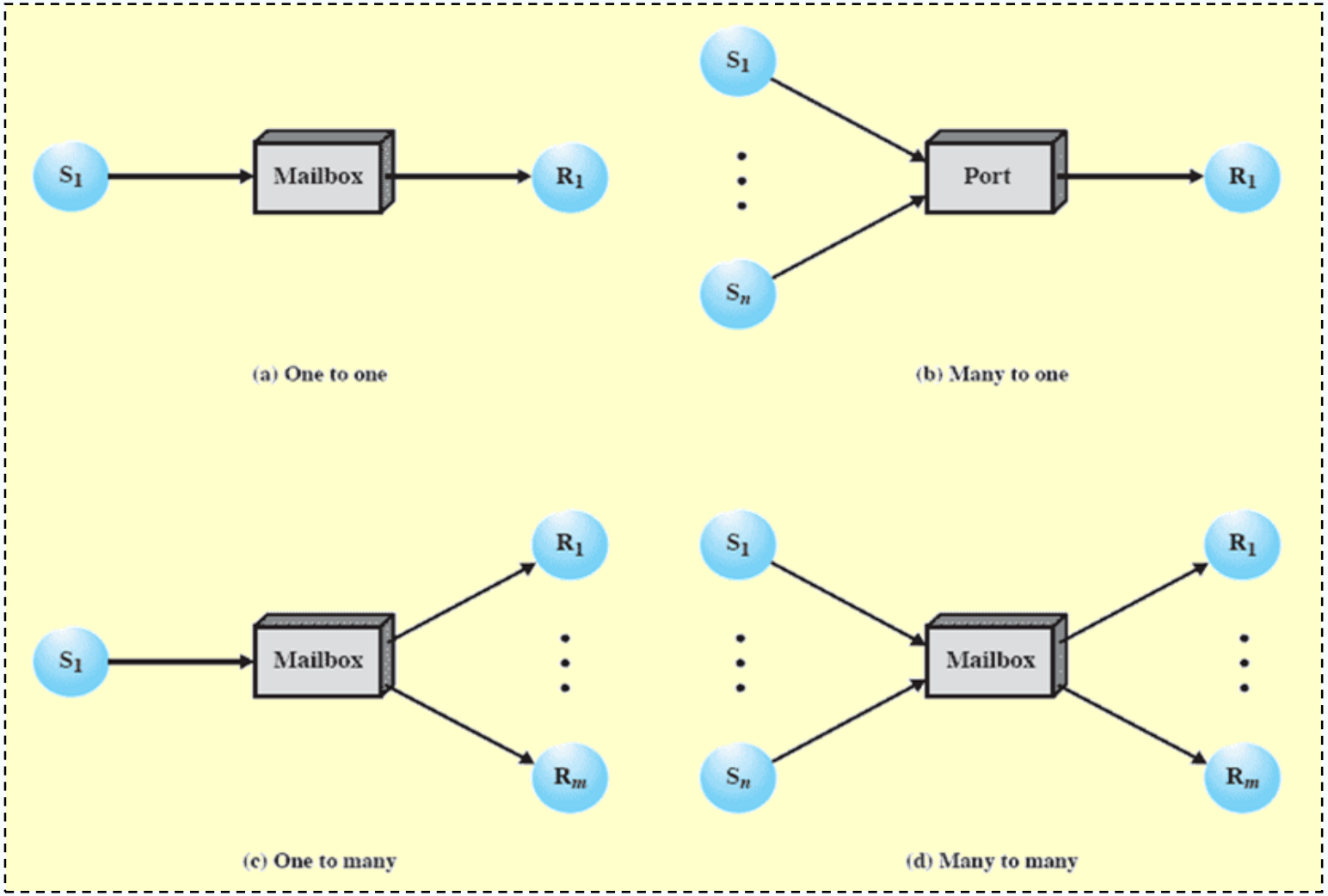


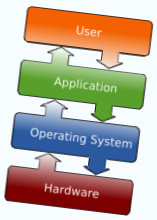
Διευθυνσιοδότηση

- Έμμεση διευθυνσιοδότηση (Indirect addressing)
  - Τα μηνύματα αποστέλλονται σε μια κοινά διαμοιραζόμενη δομή η οποία αποτελείται από ουρές
  - Οι ουρές ονομάζονται γραμματοκιβώτια (mailboxes)
  - Μια διεργασία στέλνει ένα μήνυμα στον mailbox και η άλλη διεργασία παίρνει το μήνυμα από το mailbox

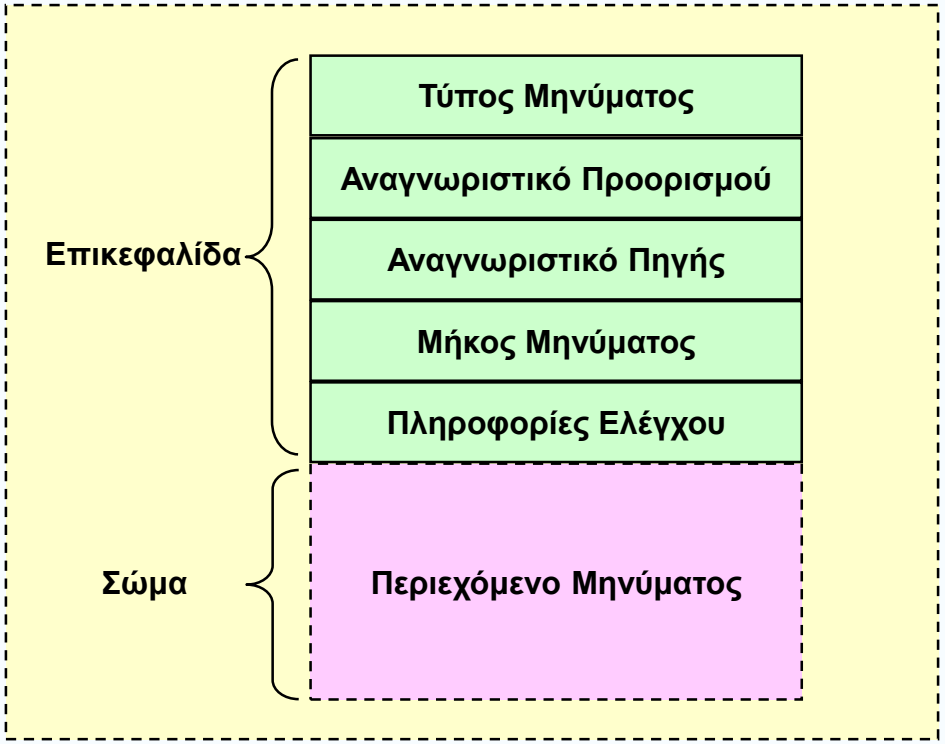


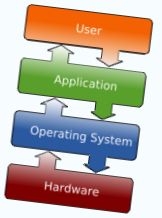
Έμμεση επικοινωνία διεργασιών





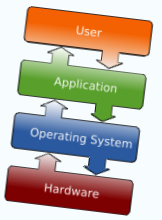
Γενική δομή μηνυμάτων





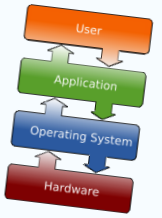
Αμοιβαίος αποκλεισμός χρησιμοποιώντας μηνύματα

```
/* program mutualexclusion */
const int n = /* number of processes  */;
void P(int i)
{
    message msg;
    while (true) {
      receive (box, msg);
      /* critical section   */;
      send (box, msg);
      /* remainder   */;
    }
}
void main()
{
    create mailbox (box);
    send (box, null);
    parbegin (P(1), P(2), . . ., P(n));
}
```



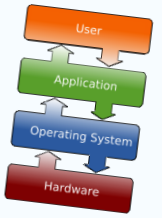
Μηνύματα Παραγωγού/Καταναλωτή

```
const int
    capacity = /* buffering capacity */ ;
    null =/* empty message */ ;
int i;
void producer()
{   message pmsg;
    while (true) {
     receive (mayproduce, pmsg);
     pmsg = produce();
     send (mayconsume, pmsg);
    }
}
void consumer()
{   message cmsg;
    while (true) {
     receive (mayconsume, cmsg);
     consume (cmsg);
     send (mayproduce, null);
    }
}

void main()
{
    create_mailbox (mayproduce);
    create_mailbox (mayconsume);
    for (int i = 1; i <= capacity; i++) send (mayproduce, null);
    parbegin (producer, consumer);
}
```



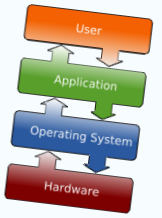
Το πρόβλημα των Αναγνωστών/Συγγραφέων

- Οσοιδήποτε αναγνώστες μπορούν ταυτόχρονα να διαβάσουν ένα αρχείο
- Μόνο ένας συγγραφέας τη φορά μπορεί να γράφει στο αρχείο
- Όσο ένας συγγραφέας γράφει στο αρχείο, κανείς δε μπορεί να το διαβάσει



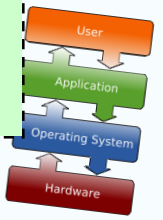
Οι αναγνώστες έχουν προτεραιότητα

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true) {
      semWait (x);
      readcount++;
      if (readcount == 1) semWait (wsem);
      semSignal (x);
      READUNIT();
      semWait (x);
      readcount--;
      if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
 }
void writer()
{
    while (true) {
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
    }
}

void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```



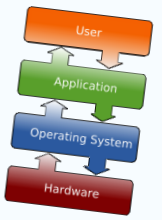
Οι συγγραφείς έχουν προτεραιότητα

```
/* program readersandwriters */
int  readcount, writecount;
semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
    while (true) {
      semWait (z);
           semWait (rsem);
                semWait (x);
                     readcount++;
                     if (readcount == 1) semWait (wsem);
                semSignal (x);
           semSignal (rsem);
      semSignal (z);
      READUNIT();
      semWait (x);
           readcount--;
           if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
}
```



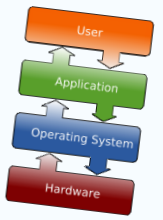
Οι συγγραφείς έχουν προτεραιότητα (συν.)

```
void writer ()
{
    while (true) {
      semWait (y);
           writecount++;
           if (writecount == 1) semWait (rsem);
      semSignal (y);
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
      semWait (y);
           writecount--;
           if (writecount == 0) semSignal (rsem);
      semSignal (y);
    }
}
void main()
{
    readcount = writecount = 0;
    parbegin (reader, writer);
}
```



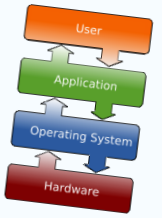
Μεταβίβαση Μηνυμάτων

```
void reader(int i)
{
    message rmsg;
        while (true) {
            rmsg = i;
            send (readrequest, rmsg);
            receive (mbox[i], rmsg);
            READUNIT ();
            rmsg = i;
            send (finished, rmsg);
        }
 }
void writer(int j)
{
    message rmsg;
    while(true) {
        rmsg = j;
        send (writerequest, rmsg);
        receive (mbox[j], rmsg);
        WRITEUNIT ();
        rmsg = j;
        send (finished, rmsg);
    }
}
```



Μεταβίβαση Μηνυμάτων (συν.)

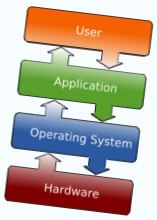
```
void  controller()
{
      while (true)
      {
         if (count > 0) {
            if (!empty (finished)) {
               receive (finished, msg);
               count++;
            }
            else if (!empty (writerequest)) {
               receive (writerequest, msg);
               writer_id = msg.id;
               count = count – 100;
            }
            else if (!empty (readrequest)) {
               receive (readrequest, msg);
               count--;
               send (msg.id, "OK");
            }
         }
         if (count == 0) {
            send (writer id, "OK");
            receive (finished, msg);
            count = 100;
         }
         while (count < 0) {
            receive (finished, msg);
            count++;
         }
      }
}
```



Αμοιβαίος Αποκλεισμός – Άσκηση 1

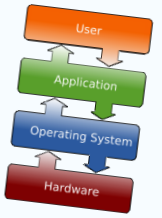
Οι παρακάτω τρεις διεργασίες έχουν πρόσβαση σε 2 διαμοιραζόμενους σημαφόρους :

```
semaphore U = 3;
semaphore V = 0;

[Process 1]      [Process 2]      [Process 3]

L1:wait(U)       L2:wait(V)       L3:wait(V)
   type("C")        type("A")        type("D")
   signal(V)        type("B")        goto L3
   goto L1          signal(V)
                    goto L2
```



Αμοιβαίος Αποκλεισμός – Άσκ 1 (συν.)

- Εντός κάθε διεργασίας οι εντολές εκτελούνται σειριακά, αλλά εντολές από διαφορετικές διεργασίες μπορούν να παρεμβληθούν με οποιαδήποτε σειρά που είναι σύμφωνη με τους περιορισμούς που τίθενται από τους σημαφόρους.
- Για να απαντήσετε τις παρακάτω ερωτήσεις υποθέστε ότι όταν ξεκινά η εκτέλεση οι διεργασίες επιτρέπεται να εκτελούνται μέχρις ότου και οι τρεις εισέλθουν σε κατάσταση wait(), στο σημείο αυτό η εκτέλεση σταματά.

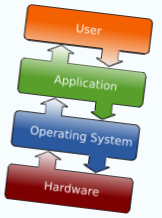


Αμοιβαίος Αποκλεισμός – Άσκ 2

- Το παρακάτω ζεύγος διεργασιών διαμοιράζεται μια κοινή μεταβλητή X:

```
    Process A              Process B
      int Y;                 int Z;
A1:  Y = X*2;          B1:   Z = X+1;
A2:  X = Y;            B2:   X = Z;
```

- Η τιμή της X γίνεται ίση με 5 πριν ξεκινήσει η εκτέλεση οποιασδήποτε διεργασίας. Οι εντολές εντός κάθε διεργασίας εκτελούνται σειριακά, αλλά οι εντολές της διεργασίας A είναι δυνατόν να εκτελεσθούν με οποιαδήποτε σειρά ως προς τις εντολές της διεργασίας B.

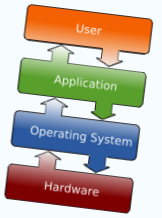


Αμοιβαίος Αποκλεισμός – Άσκ 2 (συν.)

1. Πόσες διαφορετικές τιμές της μεταβλητής X μπορούν να προκύψουν όταν ολοκληρωθεί η εκτέλεση και των δύο διεργασιών;
2. Υποθέστε ότι τα προγράμματα τροποποιούνται για να χρησιμοποιήσουν ένα κοινό σημαφόρο S (αρχική τιμή 1), και X=5.

```
    Process A                    Process B
     int Y;                      int Z;
     wait(S);                    wait(S);
A1:  Y = X*2;            B1:     Z = X+1;
A2:  X = Y;              B2:     X = Z;
     signal(S);                  signal(S);
```

Πόσες διαφορετικές τιμές του X μπορούν να προκύψουν όταν ολοκληρωθεί η εκτέλεση και των δύο διεργασιών;

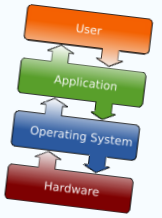


Αμοιβαίος Αποκλεισμός – Άσκ 2 (συν.)

3. Υποθέστε ότι τα προγράμματα τροποποιούνται για να χρησιμοποιήσουν ένα κοινό δυαδικό σημαφόρο T (αρχική τιμή 0), και X=5.

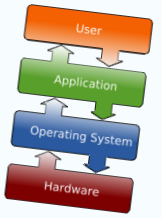
```
          Process A                    Process B
          int Y;                       int Z;
A1:       Y = X*2;          B1:        wait(T);
A2:       X = Y;            B2:        Z = X+1;
          signal(T);                   X = Z;
```

Πόσες διαφορετικές τιμές του X μπορούν να προκύψουν όταν ολοκληρωθεί η εκτέλεση και των δύο διεργασιών;



Βασικά σημεία

- Οι ταυτόχρονες διεργασίες μπορούν να αλληλεπιδρούν με διάφορους τρόπους:
  - Να αγνοούν η μια την άλλη
  - Να γνωρίζουν έμμεσα η μια την ύπαρξη της άλλης
  - Να γνωρίζουν άμεσα η μια την ύπαρξη της άλλης
- Ο αμοιβαίος αποκλεισμός είναι μια συνθήκη κατά την οποία μόνο μια διεργασία, μπορεί να έχει πρόσβαση σε συγκεκριμένο πόρο ή συγκεκριμένη λειτουργία
- Διάφορες υλοποιήσεις υπάρχουν, κάθε μια με πλεονεκτήματα και μειονεκτήματα:
  - Προσεγγίσεις λογισμικού
  - Ειδικού σκοπού εντολές μηχανής
  - Σημαφόροι
  - Παρακολουθητές
  - Μηνύματα



Αναφορές

- “Λειτουργικά Συστήματα – Αρχές Σχεδίασης”, 4η έκδοση, W. Stallings, Εκδόσεις Τζιόλα, 2008.
- “Operating System Concepts”, 7η έκδοση, από Abraham Silberschatz, Peter Galvin και Greg Gagne, Addison-Wesley, 2004.
- “Operating Systems: Design and Implementation”, 3η έκδοση, από Andrew Tanenbaum και Albert Woodhull, Prentice Hall, 2006.