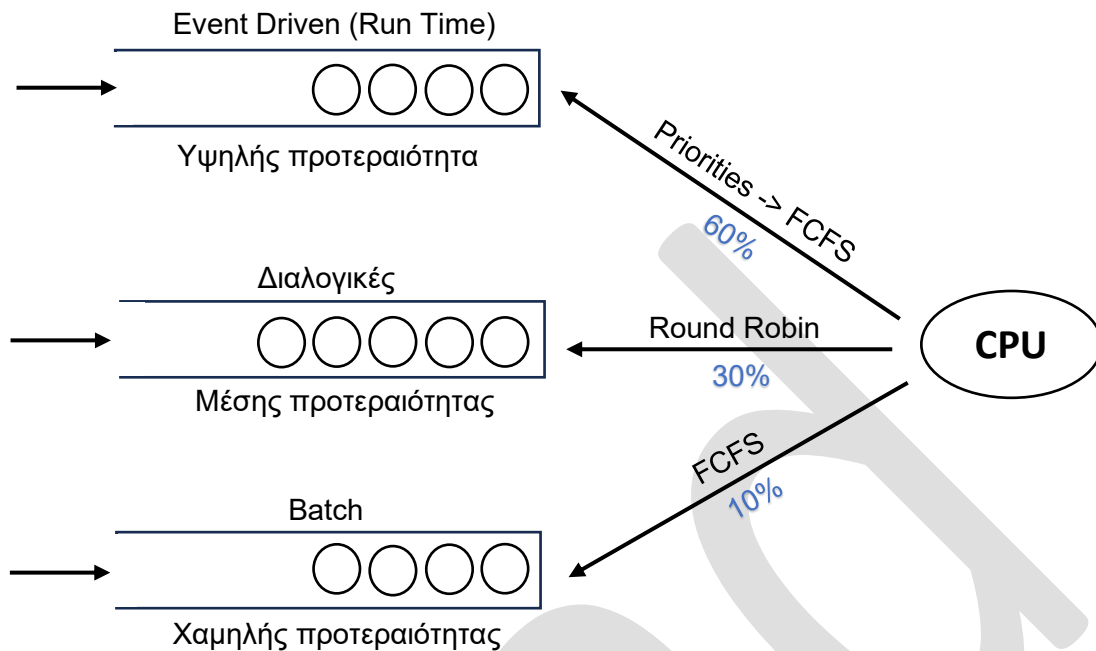


# Λειτουργικά Συστήματα – Τμήμα Β' (Μαξ-Ω)

## Διάλεξη 4

### Ουρές με διεργασίες

Μπορούμε στην κατάσταση READY να έχουμε ουρές με διεργασίες, ώστε να μην βρίσκονται «χύμα».



### Παράδειγμα

Έστω ότι έχουμε 100 κβάντα.

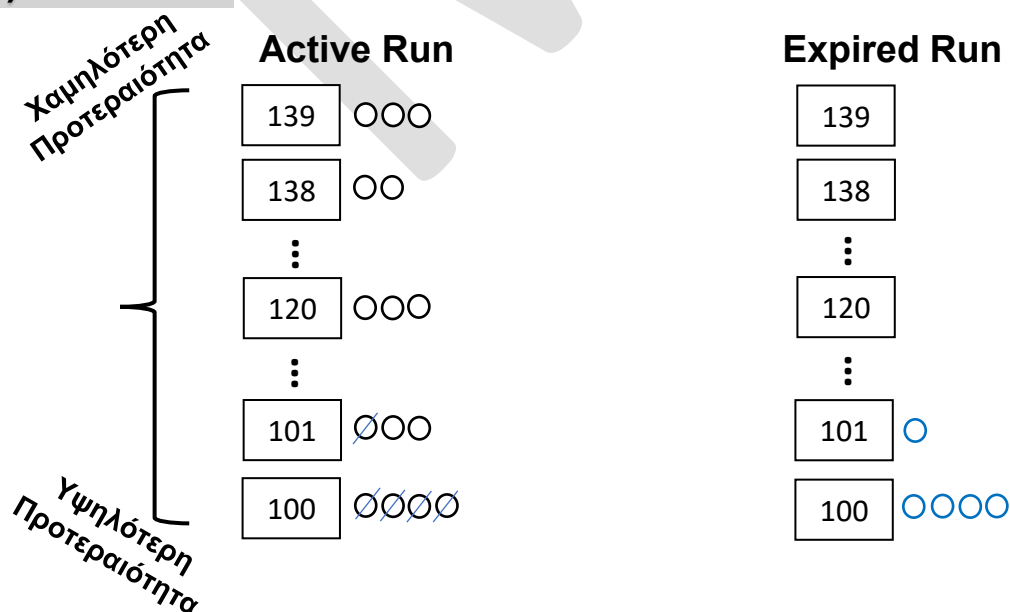
- **Batch**: 10 κβάντα η 1<sup>η</sup> διεργασία

- **Διαλογικές**: 30 κβάντα, 1-1 κβάντο σύμφωνα με τον αλγόριθμο Round Robin (αν για παράδειγμα οι διεργασίες είναι λιγότερες από 30 υπάρχει περίπτωση κάποιες αντί για 1 να παίρνουν και 2 κβάντα)

- **Event driven**: όλα τα κβάντα η 1<sup>η</sup> διεργασία

\* Τα ποσοστά δεν είναι πάντα σταθερά. Αν για παράδειγμα κάποια ουρά είναι άδεια μοιράζονται αντίστοιχα και τα ποσοστά. Π.χ. τελειώνουν οι διαλογικές --> 3 προς 1 στα Batch

### O(1) scheduler



Κάθε διεργασία που παίρνει ένα κβάντο φεύγει από την Active Run ουρά και πηγαίνει στην αντίστοιχη Expired. Κάποια στιγμή θα τελειώσουν όλες οι διεργασίες από την Active πηγαίνοντας στην Expired. Τότε το ΛΣ «σβήνει» την Active Run ουρά κάνοντας την Expired, και αντίστοιχα την Expired την κάνει Active. Αυτή η εναλλαγή θα γίνεται κάθε φορά που τελειώνουν οι διεργασίες της Active ουράς.

## Quantum ανάλογα με το priority

**if priority < 120**

$$q = (140 - \text{priority}) * 20 \text{ ms}$$

**else**

$$q = (140 - \text{priority}) * 5 \text{ ms}$$

### Παράδειγμα

$$105: q = (140 - 105) * 20 = 35 * 20 = 700 \text{ ms}$$

$$130: q = (140 - 130) * 5 = 10 * 5 = 50 \text{ ms}$$

$$120: q = (140 - 120) * 5 = 20 * 5 = 100 \text{ ms}$$

$$119: q = (140 - 119) * 20 = 21 * 20 = 420 \text{ ms}$$

### Παράδειγμα – Άσκηση

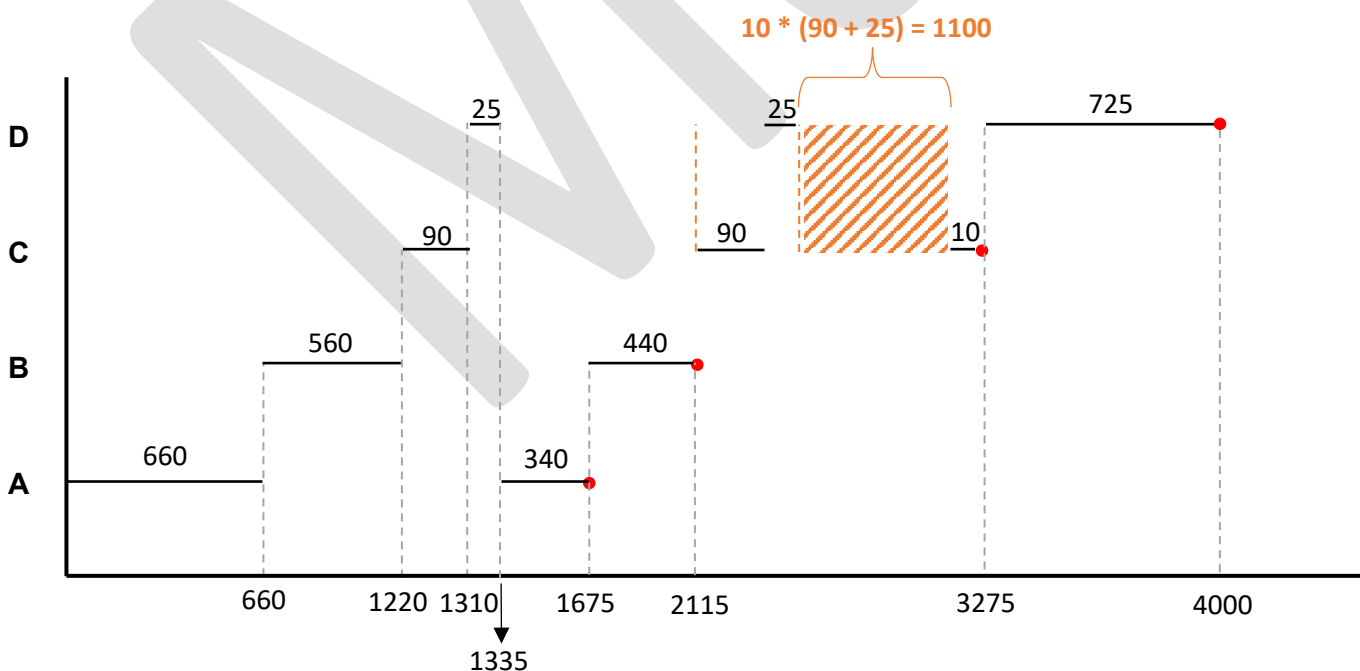
Έχουμε διεργασίες A, B, C, D με αριθμούς προτεραιοτήτων 107, 112, 122 και 135 αντίστοιχα. Όλες έχουν χρόνο εκτέλεσης 1000 ms.

$$q_A = (140 - 107) * 20 = 600 \text{ ms}$$

$$q_B = (140 - 112) * 20 = 560 \text{ ms}$$

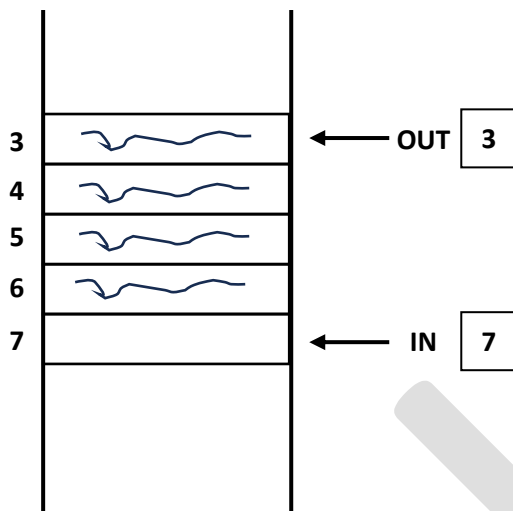
$$q_C = (140 - 122) * 5 = 90 \text{ ms}$$


$$q_D = (140 - 135) * 5 = 25 \text{ ms}$$



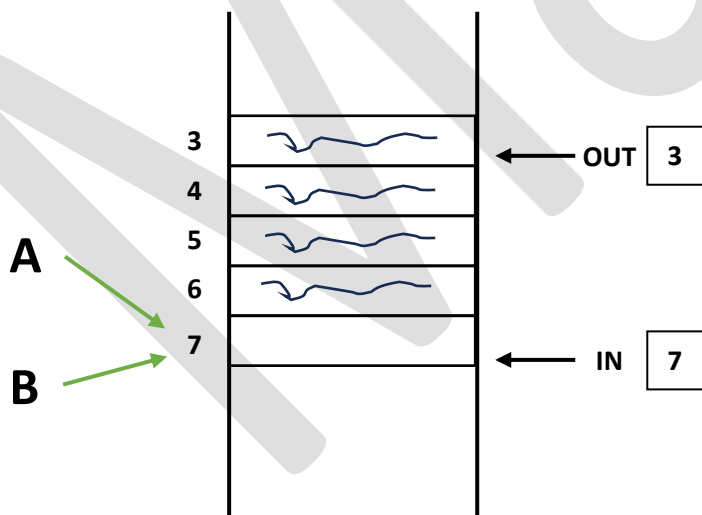
\* Με κόκκινη βουλίτσα σημειώνεται η χρονική στιγμή που τελειώνει η κάθε διεργασία

## Διεργασιακή Επικοινωνία



Ας υποθέσουμε ότι έχουμε ένα πίνακα της παραπάνω μορφής στον οποίο αποθηκεύουμε το όνομα του αρχείου μας. Κάθε διεργασία που θέλει να τυπώσει βάζει ένα αρχείο σε ένα ειδικό κατάλογο και λέει στο ΛΣ να το εκτυπώσει. Έστω όπου  υπάρχει το όνομα κάποιου αρχείου (άρα η θέση 7 είναι κενή). Το ΛΣ όταν τυπώσει τα αρχεία θα τα «σβήσει» μετά από μόνο του οπότε οι διεργασίες δεν θα ξανά ασχοληθούν με αυτά. Το ΛΣ για να γνωρίζει ότι τυπώνει το αρχείο στη θέση 3, αρκεί να έχουμε μια μεταβλητή OUT που να έχει την τιμή 3. Αντίστοιχα, θα πρέπει να έχουμε και μια μεταβλητή IN που θα μας λέει μια καινούρια εκτύπωση σε ποια θέση πρέπει να πάει (η IN δείχνει στη θέση 7, δηλαδή στην επόμενη κενή θέση). Επομένως, μια διεργασία θέλει να τυπώσει θα διαβάσει την τιμή της μεταβλητής IN, να πάει σε αυτή τη θέση να βάλει το όνομα του αρχείου και να αυξήσει την τιμή της μεταβλητής κατά 1.

Ας υποθέσουμε ότι είναι έτοιμες 2 διεργασίες οι A και B.

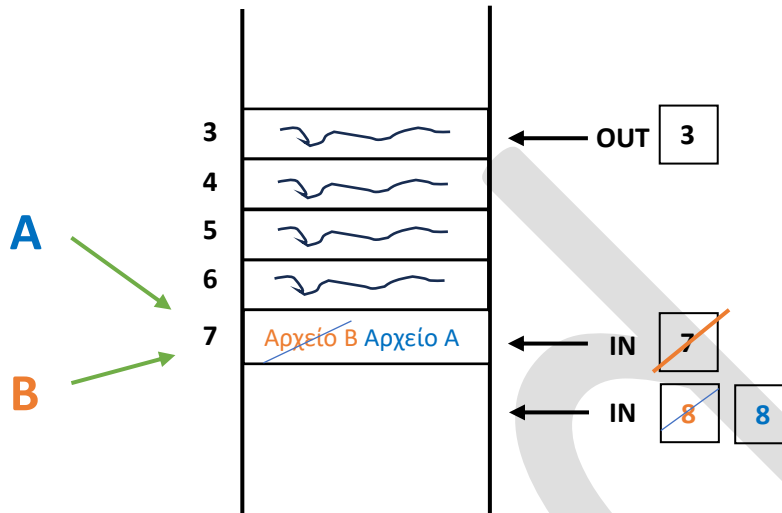


Στο RUN θα είναι μία από τις δύο. Αυτή που είναι πρώτη στο RUN και θέλει να τυπώσει, αυτή θα είναι και η πρώτη που θα φτάσει στη μεταβλητή, και μετά θα έρθει η άλλη.

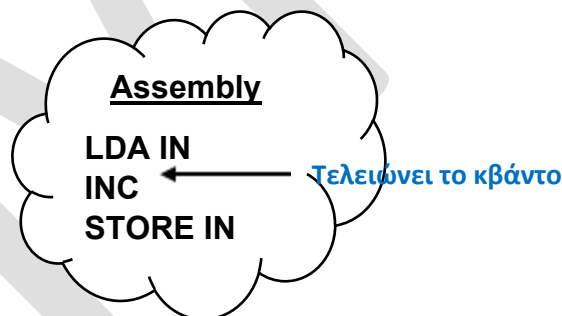
Κάθε διεργασία παίρνει 1 κβάντο. Το κρίσιμο σημείο είναι τι γίνεται όταν τελειώνει το κβάντο μιας διεργασίας. Μια διεργασία μπορεί να διακοπεί οποιαδήποτε χρονική στιγμή.

## Παράδειγμα

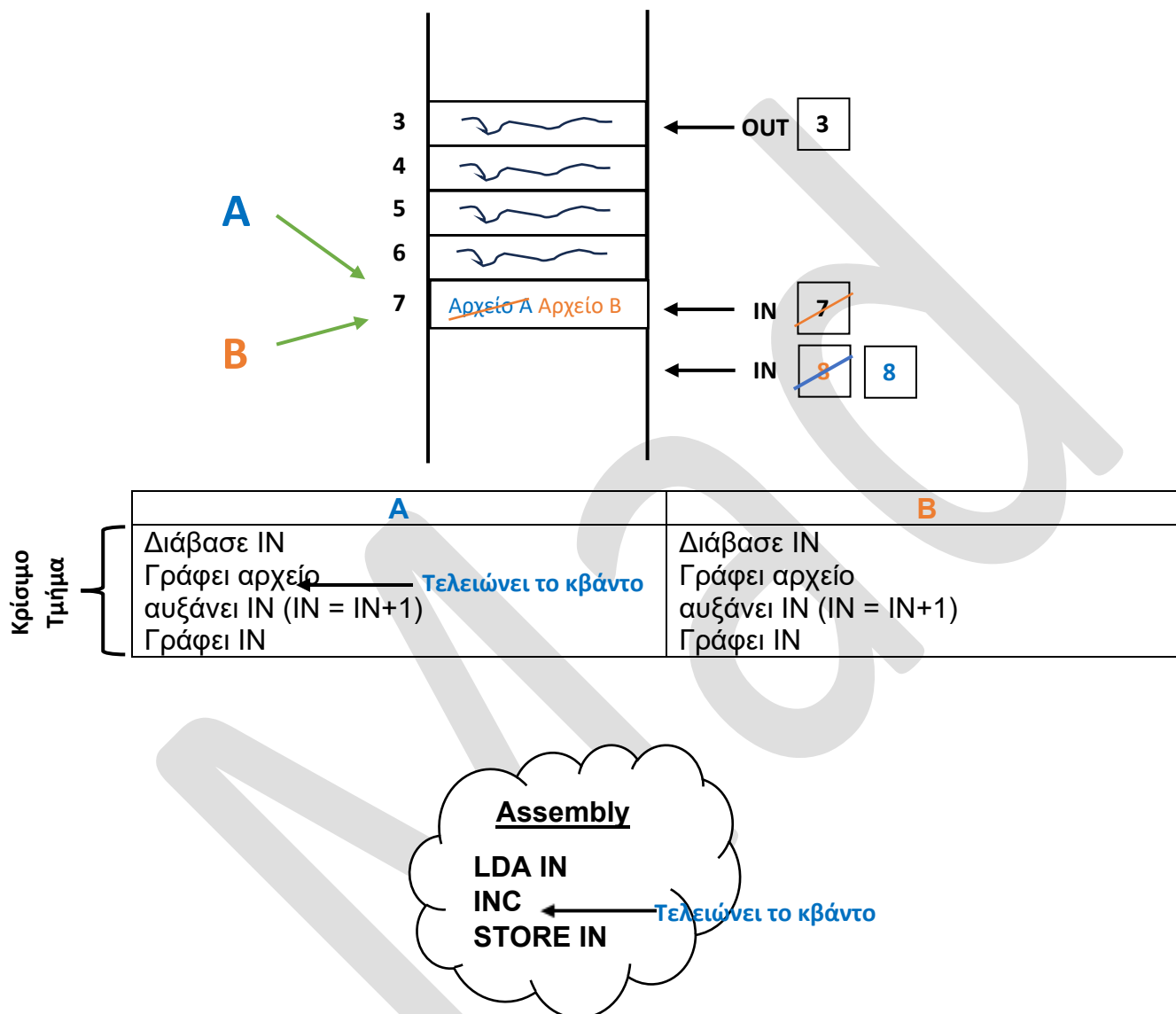
Έρχεται η διεργασία A, διαβάζει την μεταβλητή IN και «βρίσκει» 7. Εκείνη τη στιγμή τελειώνει το κβάντο της. Έρχεται η διεργασία B, διαβάζει την μεταβλητή IN και «βρίσκει» 7. Πάει και βάζει το αρχείο της και κάνει το IN 8. Συνεχίζει τη δουλειά της, μέχρι να τελειώσει το κβάντο της. Αφού έρθει ξανά η σειρά της A, συνεχίζει από εκεί που έμεινε. Πάει δηλαδή στη θέση 7, που είχε διαβάσει πριν τελειώσει το κβάντο και γράφει το αρχείο της. Έπειτα, κάνει το IN 8 και συνεχίζει τη δουλειά της. Παρατηρούμε ότι το αρχείο της B δε θα τυπωθεί ποτέ.



Κρίσιμο Τμήμα		A	B
		Διάβασε IN Γράφει αρχείο αυξάνει IN ( $IN = IN + 1$ ) Γράφει IN	Διάβασε IN Γράφει αρχείο αυξάνει IN ( $IN = IN + 1$ ) Γράφει IN



Ας δούμε και την περίπτωση που το κβάντο τελειώνει μόλις γράψει η διεργασία A το όνομα του αρχείου της. Η διεργασία A διαβάζει το IN (έχει την τιμή 7) και γράφει το αρχείο της στην αντίστοιχη θέση. Σε αυτό το σημείο τελειώνει το κβάντο και «κόβεται». Έρχεται η διεργασία B, διαβάζει το IN (έχει την τιμή 7) και γράφει το αρχείο της στην αντίστοιχη θέση. Έπειτα αυξάνει το IN κατά 1 (έχει δηλαδή πλέον την τιμή 8) και συνεχίζει μέχρι να τελειώσει το κβάντο της. Αφού ξαναέρθει η σειρά της διεργασίας A, συνεχίζει από εκεί που έμεινε. Έχοντας διαβάσει το IN (με τιμή 7) το αυξάνει κατά 1 (έχει δηλαδή πάλι την τιμή 8) και συνεχίζει τη δουλειά της. Παρατηρούμε ότι δε θα τυπωθεί ποτέ το αρχείο της A.



**Κρίσιμο τμήμα** ονομάζουμε το κομμάτι εκτέλεσης μιας διεργασίας κατά την εκτέλεση του οποίου η διεργασία θα είναι καλό να είναι μόνη της, να μη μπει δηλαδή κάποια άλλη εργασία στο αντίστοιχο δικό της κρίσιμο τμήμα. Αυτό ονομάζεται **αμοιβαίος αποκλεισμός** (η μία αποκλείει την άλλη).

## Χαρακτηριστικά καλής λύσης

1. Αμοιβαίος αποκλεισμός
2. Καμία υπόθεση για ταχύτητα ή πλήθος CPU
3. Διεργασία σε μη κρίσιμο τμήμα δεν αναστέλλει άλλες
4. Είσοδος στο κρίσιμο τμήμα σε πεπερασμένο χρόνο
5. Crash recovery (σε μη κρίσιμο σημείο) ~ παρόμοιο με το 3 ~

→ Αν crashάρει, οι υπόλοιπες διεργασίες να τρέχουν κανονικά

## Λύσεις (1<sup>st</sup> part)

### 1. Απενεργοποίηση διακοπών

Μια διεργασία θα μπορούσε όταν μπαίνει στο κρίσιμο τμήμα της να απενεργοποιεί τις διακοπές και όταν βγαίνει να τις ξανά ενεργοποιεί

Πρόβλημα: αν είναι στο Κ.Τ. και απενεργοποιήσει τις διακοπές και εκείνη την ώρα τελειώσει το κβάντο της δεν θα το καταλάβει κανένας, συνεπώς δε θα μπορέσει να τρέξει και καμία άλλη διεργασία. Μόνο όταν βγει από το Κ.Τ. μπορεί να τις ξανά ενεργοποιήσει (Δεν καλύπτει όλα τα χαρακτηριστικά καλής λύσης)

### 2. Μεταβλητές κλειδώματος

Μπορώ να έχω κάποια μεταβλητή στη μνήμη που να υποδηλώνει ότι κάποιος είναι μέσα στο Κ.Τ. του. Μπαίνοντας στο Κ.Τ. μια διεργασία κάνει την μεταβλητή 0 και βγαίνοντας 1. Το πρόβλημα παραμένει παρόμοιο με του 1.

### 3. Αυστηρή Εναλλαγή

Αφορά μόνο 2 διεργασίες και δε λειτουργεί με παραπάνω. Παρατηρούμε infinite loops, όπου οι εντολές θα τρέχουν συνέχεια. Ο λόγος έχουμε infinite loops εδώ πέρα (σε αντίθεση με τα προγράμματα που ήδη γνωρίζουμε) είναι για να ελέγξουμε όλους τους δυνατούς συνδυασμούς, δηλαδή μήπως τύχει και διακοπεί σε κάποιο σημείο του και τρέξει ο βρόγχος της άλλης διεργασίας ο οποίος μπορεί και αυτός να διακοπεί σε οποιοδήποτε σημείο του και να ξανατρέξει η προηγούμενη.

Η πρώτη ελέγχει να δει αν το turn είναι διαφορετικό από το 0 (το turn αποτελεί μια κοινή μεταβλητή που χρησιμοποιείται και από τις δύο διεργασίες. Θα έχει την τιμή 0 όταν μπορεί η μια διεργασία να μπει μέσα και 1 όταν θα μπορεί η άλλη). Η 1<sup>η</sup> διεργασία που ας πούμε έχει το όνομα 0, αν βρει το turn διαφορετικό του 0 τότε περιμένει (το wait μπορεί να είναι κάποια ρουτίνα του ΛΣ το οποίο βάζει τη διεργασία σε μια άλλη κατάσταση αναμονής μέχρι η μεταβλητή turn να αλλάξει τιμή, δηλαδή στη περίπτωση μας μέχρι να γίνει 0). Αν το turn έχει την τιμή 0, τότε περνάει η διεργασία στο Κ.Τ. της. Βγαίνοντας από το Κ.Τ. της το turn κάνει 1 και συνεχίζει στο μη Κ.Τ. κάνοντας άλλες «δουλειές» (π.χ. υπολογισμούς). Αυτό που στοχεύουμε να πετύχουμε είναι όταν η 1<sup>η</sup> διεργασία είναι στο Κ.Τ. της να μη μπορεί να μπει και η 2<sup>η</sup> στο αντίστοιχο δικό της. Αν εξετάσουμε και τις εντολές της 2<sup>ης</sup> διεργασίας θα δούμε ότι είναι παρόμοιες με αυτές της 1<sup>ης</sup> και απλά αλλάζουν οι τιμές της μεταβλητής turn.

Πρόβλημα: έρχεται η 2<sup>η</sup> διεργασία μπαίνει στο Κ.Τ. της, βγαίνει, κάνει το turn 0, μπαίνει στο μη Κ.Τ. της. Γίνεται ο έλεγχος και επειδή το turn είναι διάφορο του 1 μπαίνει σε κατάσταση αναμονής. Μπορεί να ξαναμπει στο Κ.Τ. της; **OXI**. Πρέπει να μπει πρώτα και να βγει η 1<sup>η</sup> διεργασία κάνοντας το turn 1. (δεν είναι καλή γιατί μπορεί να παραβιάσει το 3, 5)

```
while (TRUE) {  
    while (turn<>0) wait  
    critical_section  
    turn=1  
    non_critical_section  
}  
while (TRUE) {  
    while (turn<>1) wait  
    critical_section  
    turn=0  
    non_critical_section  
}
```

Εντολές 1<sup>ης</sup> διεργασίας

Εντολές 2<sup>ης</sup> διεργασίας

## 4. Λύση του Peterson

Λειτουργεί και με παραπάνω από δύο διεργασίες (το παρακάτω πρόγραμμα είναι σχεδιασμένο για 2, αλλά με κάποιες αλλαγές είναι εφικτό και για περισσότερες). Επίσης, δεν προϋποθέτει αυστηρή εναλλαγή, δηλαδή αν μια εργασία «βιάζεται» να μπει στο Κ.Τ. πολλές φορές, μπορεί να μπει όσες φορές θέλει.

Η κάθε διεργασία έχει έναν αριθμό, το process είναι αυτός όπως φαίνεται παρακάτω. Όπως παρατηρούμε η λύση αυτή αποτελείται από δύο ρουτίνες, την enter και την leave. Μια διεργασία που θέλει να μπει στο Κ.Τ. της καλεί την ρουτίνα enter και όταν βγαίνει καλεί την leave. Ο αριθμός της διεργασίας (process) είναι παράμετρος της ρουτίνας enter (π.χ. αν την καλέσει η διεργασία 0, η παράμετρος θα έχει την τιμή 0 οπότε όπου process θα θεωρήσουμε 0).

```
int turn, boolean interested [N]
enter (int process) {
    other=1-process
    interested[process]=TRUE
    turn=process
    while (turn=process and
interested[other]=TRUE) {}
}

leave (int process) {
    interested[process]=FALSE
}
```

\*\*\*\*\* Η συνέχεια στο αρχείο της 5ης διάλεξης \*\*\*\*\*

Χρήσιμα Links:

[https://youtu.be/K6u1A5\\_5zV8?si=uUbs9-bdNBSUhelk](https://youtu.be/K6u1A5_5zV8?si=uUbs9-bdNBSUhelk)