Κεφάλαιο 8

Διεργασίες

- Η έννοια και το μοντέλο της διεργασίας
- Καταστάσεις και Κύκλος Ζωής Διεργασιών
- Συνθήκες Ανταγωνισμού και Διαδιεργασιακή επικοινωνία
- Αμοιβαίος Αποκλεισμός και Κρίσιμα Τμήματα Διεργασιών
- Απλοί Μηχανισμοί Αμοιβαίου Αποκλεισμού
- Ενεργός Αναμονή και Απενεργοποίηση-Αφύπνιση
- Σηματοφορείς
- Χρονοδρομολόγηση Επεξεργαστή

Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με το μοντέλο των διεργασιών το οποίο μας βοηθά στην κατανόηση της λειτουργίας ενός υπολογιστικού συστήματος. Το μοντέλο έμμεσα μας βοηθά να κατανοήσουμε και σχεδιαστικά θέματα που αφορούν την ΚΜΕ. Στην αρχή του κεφαλαίου θα μελετήσουμε την έννοια και το μοντέλο της διεργασίας. Θα γνωρίσουμε βασικούς μηχανισμούς που συντελούν στην υλοποίηση και τη διαχείριση των διεργασιών από το σύστημα και θα εξετάσουμε προβλήματα που δημιουργούνται σε συστήματα που υποστηρίζουν πολλές ταυτόχρονες διεργασίες.

Στη συνέχεια θα μελετήσουμε μηχανισμούς που εγγυώνται την ορθή λειτουργία συστημάτων με πολλές ταυτόχρονες διεργασίες. Θα ασχοληθούμε τόσο με απλούς, όσο και με πιο σύνθετους και ισχυρούς μηχανισμούς επικοινωνίας, συνεργασίας και συγχρονισμού των διεργασιών σε ένα σύστημα. Τέλος, θα δούμε πώς οι μηχανισμοί αυτοί δίνουν επαρκείς λύσεις σε συνηθισμένα προβλήματα διαδιεργασιακής συνεργασίας.

Στο τελευταίο μέρος του κεφαλαίου θα μας απασχολήσει ο τρόπος με τον οποίο πραγματοποιείται η ταυτόχρονη εξυπηρέτηση διεργασιών σε ένα σύστημα πολυπρογραμματισμού. Θα μελετήσουμε αλγόριθμους που ορίζουν τη διαδοχή των διεργασιών στην ΚΜΕ και θα εξετάσουμε τις παραμέτρους των αλγόριθμων αυτών που επηρεάζουν την απόδοση των συστημάτων.

Διδακτικοί στόχοι

Με την ολοκλήρωση του κεφαλαίου, θα μπορείτε να περιγράφετε και να αναλύετε:

- την έννοια της διεργασίας,
- τον τρόπο δημιουργίας και τον κύκλο ζωής της καθώς και τη βασική διαχείριση των διεργασιών από το σύστημα
- θα έχετε αφομοιώσει την ανάγκη της διαδιεργασιακής επικοινωνίας.
- θα μπορείτε να περιγράψετε με τους κυριότερους μηχανισμούς αλληλεπίδρασης, συνεργασίας και συγχρονισμού των διεργασιών οι οποίοι εξασφαλίζουν στο σύστημα τη δυνατότητα να λειτουργεί ορθά και αποδοτικά.

Προερωτήσεις

Στο κεφάλαιο αυτό απαντώνται ερωτήματα όπως:

- ✓ Τι είναι οι διεργασίες και τι είναι το μοντέλο διεργασιών;
- ✓ Ποιες είναι οι καταστάσεις μιας διεργασίας και πώς αυτές μεταβάλλονται;
- ✓ Πώς γίνεται η υλοποίηση και πώς η διαχείριση των διεργασιών;
- ✓ Ποια προβλήματα εμφανίζονται σε συστήματα με πολλές παράλληλες διεργασίες;
- ✓ Πώς λύνονται τα προβλήματα διαχείρισης των πολλών παράλληλων διεργασιών;

8.1 Η έννοια και το μοντέλο της διεργασίας

Τα σύγχρονα συστήματα υπολογιστών είναι σε θέση να εκτελούν ταυτόχρονα ένα μεγάλο αριθμό εργασιών ενός ή περισσότερων χρηστών. Παράλληλα με τις εργασίες των χρηστών, το σύστημα συχνά εκτελεί και εσωτερικές εργασίες διαχείρισης και συγχρονισμού. Ένα τυπικό σύστημα μπορεί στο ίδιο διάστημα να διαβάζει από ένα δίσκο, να εκτελεί κάποιους πολύπλοκους υπολογισμούς, να μεταφέρει δεδομένα μέσω δικτύου και να εκτυπώνει ένα σχέδιο σε κάποιον εκτυπωτή. Σ' ένα **σύστημα καταμερισμού χρόνου** *(time-sharing system)*, η ΚΜΕ διατίθεται διαδοχικά σε διαφορετικά προγράμματα για την εκτέλεση κάποιου μικρού μέρους του καθενός για κάποια χιλιοστά του δευτερολέπτου. Στους χρήστες του συστήματος, η ταχεία αυτή εναλλαγή των προγραμμάτων στην ΚΜΕ δίνει την ψευδαίσθηση της ταυτόχρονης επεξεργασίας. Φυσικά κάθε στιγμή μόνο ένα πρόγραμμα είναι ενεργό σε κάθε ΚΜΕ του συστήματος. Η ψευδαίσθηση της ταυτόχρονης εκτέλεσης προγραμμάτων σε μια ΚΜΕ συχνά αποκαλείται **ψευδοπαραλληλισμός** *(pseudoparallelism)* σε αντίθεση με τον **πραγματικό παραλληλισμό**, όπου στο σύστημα εκτελούνται παράλληλα προγράμματα σε περισσότερες της μίας ΚΜΕ.

Η πολυπλοκότητα των σημερινών συστημάτων μας έχει οδηγήσει στη δημιουργία της έννοιας και του μοντέλου της διεργασίας, με τη βοήθεια της οποίας μπορούμε να μελετήσουμε, να σχεδιάσουμε και να διαχειριστούμε σχετικά εύκολα όλα τα συστήματα υπολογιστών.

Σύμφωνα με το **μοντέλο της διεργασίας** *(process model)*, όλα τα εκτελούμενα προγράμματα στο σύστημα, μαζί και το ίδιο το ΛΣ, αποτελούν ένα σύνολο από **σειριακές διεργασίες** *(sequencial processes)*.

Μια **διεργασία** *(process)* είναι ένα πρόγραμμα σε εκτέλεση.

Σε αντίθεση με το πρόγραμμα, το οποίο είναι κάτι στατικό, η διεργασία είναι μια οντότητα δυναμική και περιλαμβάνει και άλλα στοιχεία εκτός από το πρόγραμμα. Για παράδειγμα, μια συνταγή φαγητού είναι ένα πρόγραμμα. Το εγχειρίδιο των πρώτων βοηθειών είναι ένα άλλο πρόγραμμα. Όταν όμως συγκεντρώσετε τα υλικά της συνταγής και ξεκινήσετε να φτιάχνετε το φαγητό, τότε:

- Η συνταγή είναι το πρόγραμμα, ο αλγόριθμος δηλαδή που θα δώσετε για τη δημιουργία του φαγητού.
- Εσείς είστε εκείνος που θα εκτελέσει τη συνταγή, το πρόγραμμα δηλαδή. Είστε το αντίστοιχο της ΚΜΕ.
- Τα υλικά της συνταγής, τα οποία χρησιμοποιείτε, είναι τα δεδομένα εισόδου (input data).
- Η διαδικασία εκτέλεσης της συνταγής (προγράμματος) από εσάς (ΚΜΕ) με χρήση των υλικών (δεδομένων εισόδου) για την παραγωγή του φαγητού (δεδομένα εξόδου) είναι η διεργασία.
- Το φαγητό που θα φτιάξετε είναι το αποτέλεσμα (η έξοδος) της διεργασίας.

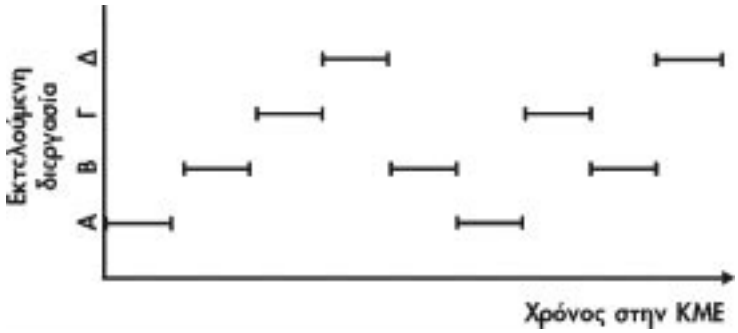
Αν τώρα χρειαστεί να διακόψετε την ενασχόλησή σας με την εκτέλεση της συνταγής, για να ακολουθήσετε τις οδηγίες του εγχειριδίου πρώτων βοηθειών, γιατί καήκατε, τότε:

- Αποθηκεύετε στη μνήμη σας το σημείο στο οποίο είχε φθάσει η εκτέλεση της

συνταγής (η ΚΜΕ φυλάσσει την κατάσταση της διεργασίας που διακόπτεται).

- Αρχίζετε να εκτελείτε το πρόγραμμα "παροχή πρώτων βοηθειών σε περίπτωση ελαφρού εγκαύματος". Πάλι είστε το ανάλογο της ΚΜΕ, το εγχειρίδιο είναι το πρόγραμμα, η αλοιφή εγκαύματος είναι το δεδομένο εισόδου, και το φροντισμένο σας καμένο χέρι είναι η έξοδος της διεργασίας. Έχετε περάσει σε μια άλλη διεργασία.
- Μόλις φροντίσετε το χέρι σας, συνεχίζετε την εκτέλεση της συνταγής, αν δεν έχετε φθάσει πολύ αργά (ευτυχώς στο σύστημα δεν καίγεται ούτε καταστρέφεται καμιά διεργασία από μόνη της).

Όπως και στο φυσικό κόσμο, έτσι και σε ένα σύστημα υπολογιστή, εκτός από περιπτώσεις καταστάσεων ανάγκης, όπου η ενεργή διεργασία διακόπτεται προς εξυπηρέτηση κάποιας άλλης με υψηλότερη προτεραιότητα, η ΚΜΕ ασχολείται διαδοχικά με πολλές διεργασίες που περιμένουν εξυπηρέτηση. Η μέθοδος με την οποία γίνεται η διαδοχή των διεργασιών στην ΚΜΕ ονομάζεται **αλγόριθμος χρονοδρομολόγησης** *(scheduling algorithm)* και η διαδοχή των διεργασιών στην ΚΜΕ ονομάζεται **πολυπρογραμματισμός** *(multiprogramming)*.



Σχήμα 8.1 *Πολυπρογραμματισμός. Η ΚΜΕ διατίθεται διαδοχικά στις διεργασίες του συστήματος, εξυπηρετώντας κάθε μία από αυτές για ορισμένο χρονικό διάστημα.*

Το μοντέλο των διεργασιών, σύμφωνα με το οποίο οτιδήποτε απασχολεί την ΚΜΕ του συστήματος είναι διεργασία, μπορεί να χρησιμοποιηθεί για την ανάλυση, σχεδίαση και διαχείριση κάθε συστήματος υπολογιστή. Το μοντέλο βρίσκει εφαρμογή και σε συστήματα που δεν επιτρέπουν την ταυτόχρονη επεξεργασία διεργασιών, όπως το ΛΣ MS-DOS. Στα συστήματα αυτά υπάρχουν **διακοπές** *(interrupts)* που αναστέλλουν την εκτελούμενη διεργασία, για να εξυπηρετήσουν συμβάντα σημαντικά για το σύστημα, όπως η εγγραφή σε αρχείο (διακοπές δίσκου), η ενημέρωση της οθόνης ή η εκτύπωση. Αν θεωρήσουμε τους εξυπηρετητές των διακοπών αυτών ως διεργασίες (διεργασίες δίσκου, οθόνης, σειριακών θυρών κ.α.), τότε μπορούμε να μελετήσουμε και αυτό το σύστημα με βάση το μοντέλο διεργασιών.

Ελαφριές Διεργασίες - Νήματα

Σε ένα ΛΣ καταμερισμού χρόνου, κάθε στιγμή εκτελούνται πολλά προγράμματα, είτε του ΛΣ είτε των χρηστών. Κάθε πρόγραμμα είναι για το ΛΣ τουλάχιστον μία διεργασία. Μπορεί όμως και ένα πρόγραμμα να εκτελεί εργασίες παράλληλα, όπως το πρόγραμμα κράτησης εισιτηρίων στα γραφεία μιας αεροπορικής εταιρείας.

Στον κεντρικό υπολογιστή της εταιρείας εκτελούνται πολλές εφαρμογές και μια από αυτές είναι το πρόγραμμα κράτησης εισιτηρίων. Η εταιρεία έχει πολλούς πράκτορες που χρησιμοποιούν το πρόγραμμα και κάνουν κρατήσεις ταυτόχρονα.

Το πρόγραμμα κρατήσεων είναι χωρισμένο σε τμήματα που εκτελούνται παράλληλα και το κάθε ένα από αυτά εξυπηρετεί έναν πράκτορα. Το κάθε τμήμα θα μπορούσε να είναι μια διεργασία. Όμως η εναλλαγή των τμημάτων του ίδιου προγράμματος στην ΚΜΕ του υπολογιστή έχει μια ιδιαιτερότητα σε σχέση με την εναλλαγή διαφορετικών προγραμμάτων στην ΚΜΕ:

Τα ανεξάρτητα τμήματα του προγράμματος χρησιμοποιούν όλα μια κοινή περιοχή της μνήμης (π.χ. τις ίδιες μεταβλητές όπως ο αριθμός των ελεύθερων εισιτηρίων), ενώ αυτό δεν συμβαίνει μεταξύ διαφορετικών προγραμμάτων.

Για το λόγο αυτό, οι ενέργειες που απαιτούνται για την εναλλαγή τμημάτων προγράμματος στην ΚΜΕ είναι άλλες και γενικά απλούστερες από τις ενέργειες που πρέπει να κάνει το ΛΣ για εναλλαγή προγραμμάτων (διεργασιών) στην ΚΜΕ.

Για την αποδοτικότερη λειτουργία του ΛΣ, τα τμήματα προγραμμάτων που εκτελούνται παράλληλα συνήθως δεν υλοποιούνται ως διεργασίες, αλλά ως **ελαφριές διεργασίες** *(lightweight processes)* **ή νήματα εκτέλεσης** *(execution threads)* **ή απλά νήματα** *(threads)*.

Τα νήματα έχουν πολλές ομοιότητες με τις διεργασίες αλλά και δύο διαφορές:

- Έχουν όλα πρόσβαση σε μια κοινή περιοχή της μνήμης
- Η διαδικασία εναλλαγής τους στην ΚΜΕ είναι απλούστερη από την αντίστοιχη διαδικασία εναλλαγής διαδικασιών.

Μπορούμε να πουμε γενικά ότι τα διαφορετικά προγράμματα που εκτελούνται παράλληλα σε ένα υπολογιστή είναι διεργασίες και τα ανεξάρτητα τμήματα ενός προγράμματος που εκτελούνται παράλληλα στον υπολογιστή είναι νήματα.

Τα προγράμματα που μπορούν να διαιρεθούν σε νήματα ονομάζονται **ταυτόχρονα προγράμματα** *(concurrent programs)*.

Απεικόνιση διεργασιών

Ενας πολύ καλός τρόπος για να απεικονίσουμε ταυτόχρονα προγράμματα είναι να χρησιμοποιήσουμε το **γράφο προβαδίσματος** *(precedence graph)*. Χρησιμοποιώντας ένα τέτοιο γράφο μπορούμε να αναπαραστήσουμε ένα πρόγραμμα και να δούμε ποια τμήματά του μπορούν να εκτελεσθούν παράλληλα και ποια πρέπει να περιμένουν μέχρι να ολοκληρωθούν κάποια άλλα.

Παράδειγμα 1.

Ας πάρουμε για παράδειγμα την προετοιμασία μιας ποδοσφαιρικής ομάδας για το πρωτάθλημα. Αυτή αποτελείται από τα εξής στάδια: Οι παίκτες ξεκινούν ατομικές προπονήσεις φυσικής κατάστασης. Παράλληλα, η ομάδα επιλέγει νέους παίκτες για μεταγραφή. Αφού αποκτηθούν όσοι επιλεγούν, η ομάδα ξεκινά ομαδικές

προπονήσεις. Ο προπονητής μελετά τις τακτικές παιχνιδιού. Αφού καταλήξει στις καλύτερες, η ομάδα ξεκινά προπονήσεις στις συγκεκριμένες τακτικές. Στη συνέχεια, αφού επιλεγούν ομάδες αντίπαλοι, η ομάδα δίνει φιλικά παιχνίδια, ώστε να είναι έτοιμη για το πρωτάθλημα. Για να οργανώσει ο προπονητής την εργασία του πρέπει να γνωρίζει όλα τα στάδια της προετοιμασίας, όπως επίσης και να ξέρει ποια πρέπει να έχουν ολοκληρωθεί, για να ξεκινήσουν τα επόμενα. Για παράδειγμα, πρέπει να ξέρει οτι πρέπει πρώτα να έχει κάνει όλες τις μεταγραφές πριν ξεκινήσει ομαδικές προπονήσεις.

Ένας γράφος αποτελείται από κόμβους (vertices) και ακμές (edges). Κάθε ακμή συνδέει δύο κόμβους και μπορεί να έχει ή να μην έχει κατεύθυνση. Οταν μια ακμή έχει κατεύθυνση από τον ένα κόμβο στον άλλο, τότε ονομάζεται **κατευθυνόμενη ακμή** *(directed edge)*. Όταν οι ακμές ενός γράφου είναι κατευθυνόμενες, τότε ο γράφος ονομάζεται **κατευθυνόμενος γράφος** *(directed graph)*. Συνήθως σε ένα γράφο δε συναντάμε και κατεθυνόμενες και μη κατευθυνόμενες ακμές.

Παράδειγμα ενός γράφου είναι το οδικό δίκτυο που φαίνεται στο σχήμα που ακολουθεί. (Σχήμα 8.2).

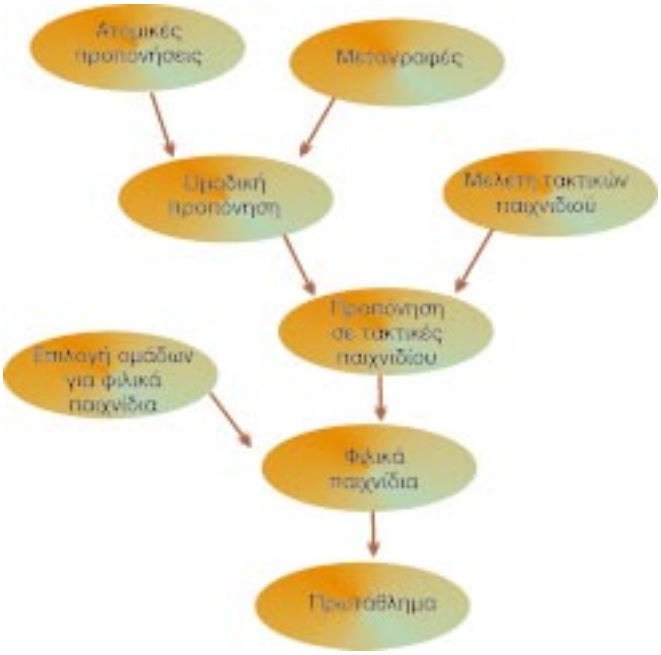
Σχήμα 8.2.

Στο σχήμα, οι πόλεις είναι οι κόμβοι του γράφου και οι δρόμοι είναι οι ακμές. Ολοι οι δρόμοι είναι διπλής κατεύθυνσης. Αν κάποιοι ήταν μονόδρομοι, τότε θα έπρεπε στην απεικόνιση του γράφου να αντικαταστήσουμε όλους τους δρόμους διπλής κατεύθυνσης με δύο μονόδρομους τον καθένα. Στην περίπτωση αυτή, ο γράφος θα ήταν κατευθυνόμενος.

Παρατηρείστε ότι στον γράφο του οδικού δικτύου μπορούμε να κάνουμε κύκλους διασχίζοντάς τον. Μπορούμε δηλαδή να κάνουμε τη διαδρομή **Κόρινθο -> Τρίπολη -> Νάυπλιο -> Κόρινθο** καταλήγοντας έτσι στον κόμβο απόπου ξεκινήσαμε. Ο γράφος αυτός ονομάζεται **κυκλικός**. Ένας γράφος που δεν περιέχει κύκλους ονομάζεται **ακυκλικός** *(acyckic)*.

Παράδειγμα 2.

Ο γράφος που περιγράφει το πρόγραμμα προετοιμασίας της ομάδας μας για το πρωτάθλημα φαίνεται στο Σχήμα 8.3. Παρατηρείστε ότι είναι κατευθυνόμενος και ακυκλικός.



Σχήμα 8.3.

Για να μπορεί ένας γράφος να περιγράψει ένα ταυτόχρονο πρόγραμμα και να ορίσει τη σειρά εκτέλεσης των νημάτων ή των διεργασιών του, αυτός πρέπει να είναι κατευθυνόμενος και ακυκλικός. Μάλιστα, σε ένα γράφο προβαδίσματος, δύο διεργασίες μπορούν να εκτελεσθούν παράλληλα, αν δεν υπάρχει διαδρομή που να συνδέει άμεσα ή έμμεσα τις διεργασίες αυτές.

Αν υπήρχε μια ακμή με κατεύθυνση από την «ομαδική προπόνηση» στις «μεταγραφές», τότε θα έπρεπε η προπόνηση να γίνει μετά τις μεταγραφές, οι οποίες θα πρέπει να γίνουν μετά την ομαδική προπόνηση που πρέπει να γίνει μετά τις μεταγραφές κ.ο.κ. Στην περίπτωση αυτή η προετοιμασία δε θα ολοκληρωνόταν ποτέ.

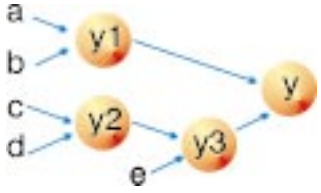
Παράδειγμα 3: Κατασκευή γράφου προβαδίσματος

Εστω το πρόγραμμα που υπολογίζει τη μαθηματική έκφραση Y = a/b + (c – d)/e. Ας την αναλύσουμε σε μερικά αποτελέσματα:

- Y1 = a / b
- Y2 = (c – d)
- Y3 = Y2 / e
- Y = Y1 + Y3

Αντιστοιχούμε τον υπολογισμό καθενός ενδιάμεσου αποτελέσματος σε μια διεργασία. Απεικονίζουμε τις διεργασίες αυτές ως κόμβους σε γράφο. Οταν μια διεργασία απαιτεί την ολοκλήρωση μιας άλλης πρίν από αυτήν, τότε συνδέουμε τις διεργασίες με ακμή. Η φορά της ακμής είναι από τη διεργασία που προηγείται σε αυτή που ακολουθεί.

Ο γράφος που προκύπτει φαίνεται στο Σχήμα 8.4:



Σχήμα 8.4.

Από το γράφο φαίνεται εύκολα οτι οι διεργασίες Υ1 και Υ2 μπορούν να εκτελεσθούν παράλληλα, αφού δεν υπάρχει τρόπος να μεταβούμε από τη μια στην άλλη (και λογικό είναι αφού οι πράξεις υπολογισμού των Υ1 και Υ2 είναι εντελώς ανεξάρτητες). Αντίθετα, ο υπολογισμός (διεργασία) Υ3 πρέπει να εκτελεσθεί αφού ολοκληρωθεί ο υπολογισμός (διεργασία) Υ2. Παρατηρείστε ότι υπάρχει τρόπος μετάβασης από την Υ2 στην Υ3. Επίσης η διεργασία Υ πρέπει να εκτελεσθεί μετά τις Υ1 και Υ2. Προφανώς η Υ εκτελείται μετά και την Υ2, αφού οι δύο κόμβοι συνδέονται (με το μονοπάτι Υ2 -> Υ3 -> Υ)

Οι γράφοι προβαδίσματος βοηθούν ιδιαίτερα στην ανάλυση προγραμμάτων, αλλά δυστυχώς δεν μπορούν να χρησιμοποιηθούν από τις κοινές γλώσσες προγραμματισμού. Σε αυτές χρησιμοποιούμε συνήθως τις εντολές **parbegin** (**par**allel **begin**) και **parend** (**par**allel **end**), που προτάθηκαν το 1965 από τον Dijkstra.

Η χρήση τους είναι εντελώς ανάλογη με τη χρήση των εντολών begin ... end στη γλώσσα pascal. Οπως στην Pascal οι εντολές μεταξύ του begin και του end χωρίζονται με «;» και εκτελούνται σειριακά, έτσι και οι εντολές που βρίσκονται μεταξύ του parbegin και του parend χωρίζονται με το σύμβολο «¦¦» και εκτελούνται παράλληλα. Ολη η ομάδα εντολών μέσα στο **parbegin...parend** ξεκινά να εκτελείται ταυτόχρονα, ενώ η εκτέλεση της ομάδας εντολών τελειώνει όταν τελειώσουν όλες.

Ο συμβολισμός αυτός έχει το πλεονέκτημα οτι κάνει τα προγράμματα ευανάγνωστα και κατανοητά, δυστυχώς όμως δεν μπορεί να περιγράψει όλους τους γράφους προβαδίσματος.

Παράδειγμα 4. Κατασκευή προγράμματος από γράφο προβαδίσματος

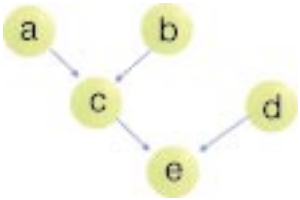
Το πρόγραμμα του παραδείγματος 3 δίνεται στον πίνακα που ακολουθεί γραμμένο με συμβολισμό parbegin...parend. Στην αριστερή στήλη του πίνακα φαίνεται το πρόγραμμα γραμμένο σε pascal που εκτελείται χωρίς παραλληλισμό.

Πίνακας 8.1

begin
Y1 = a / b ;
Y2 = (c – d) ;
Y3 = Y2 / e ;
Y = Y1 + Y3 ;
End
	begin
	parbegin
	Y1 = a / b ¦¦
	begin
	Y2 = (c – d) ;
	Y3 = Y2 / e ;
	End ;
	parend ;
	Y = Y1 + Y3 ;
	End

Παράδειγμα 5. Αντιστοίχιση γράφου και συμβολισμού parbegin...parend

Θα δούμε οτι από ένα γράφο προβαδίσματος μπορούν να προκύψουν πολλά προγράμματα με συμβολισμό parbegin...parend. Τα προγράμματα αυτά διαφέρουν σε πολυπλοκότητα αλλά και στο χρόνο ολοκλήρωσής τους. Ο γράφος που καλούμαστε να υλοποιήσουμε είναι ο ακόλουθος:



Στον πίνακα 8.2 που ακολουθεί δίνονται διαφορετικά προγράμματα (με την ίδια λειτουργία). Για το καθένα σημειώνουμε τα πλεονεκτήματα και τα μειονεκτήματά του.

Πίνακας 8.2

Parbegin a ¦¦ b ¦¦ d parend; c; e;	(+) Απλός κώδικας. (-) Η c περιμένει την ολοκληρωση της d ενώ δεν απαιτείται
Parbegin a ¦¦ b parend; Parbegin c ¦¦ d parend; e;	(+) Σχετικά απλός κώδικας (-) Η d περιμένει την ολοκλήρωση των a και b ενώ δεν απαιτείται
Parbegin Begin Parbegin a ¦¦ b parend ; c end ¦¦ d parend; e;	(-) Σύνθετος κώδικας (+) Καλύτερος παραλληλισμός.

Ο παραλληλισμός που επιτυγχάνεται με το τρίτο πρόγραμμα αφήνεται ως άσκηση. Εξηγείστε τι ακριβώς συμβαίνει και γιατί το τρίτο πρόγραμμα υπερτερεί των άλλων δύο.

8.2 Καταστάσεις και Κύκλος Ζωής Διεργασιών

Σε όλα τα λειτουργικά συστήματα που υποστηρίζουν την έννοια της διεργασίας υπάρχουν διαδικασίες για τη δημιουργία και τη διαχείριση των διεργασιών αυτών.

Σε συστήματα που εκτελούν μόνο μία καθορισμένη εφαρμογή, όλες οι διεργασίες δημιουργούνται αυτόματα με την έναρξη της εκτελούμενης εφαρμογής. Αντίθετα, σε συστήματα με πολλούς χρήστες και πλήθος εφαρμογών, που μπορούν να είναι ενεργές σε κάθε χρονική στιγμή, υπάρχουν συγκεκριμένοι μηχανισμοί δυναμικής δημιουργίας και διαχείρισης διεργασιών.

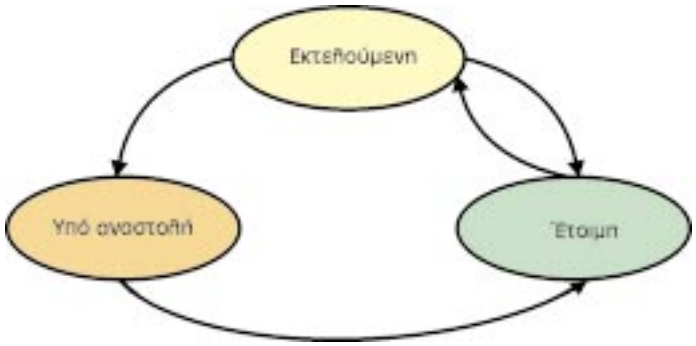
Για παράδειγμα, στο ΛΣ UNIX, μια διεργασία δημιουργεί μια νέα διεργασία ακριβές αντίγραφό της, με χρήση κατάλληλης **κλήσης συστήματος** *(system call)*. Οι δύο διεργασίες, γονέας (η καλούσα) και παιδί (η καλούμενη), εκτελούνται παράλληλα. Στο ΛΣ WINDOWS NT μια διεργασία μπορεί επίσης με κλήση συστήματος να δημιουργήσει μια νέα διεργασία, η εκτέλεση της οποίας πραγματοποιείται ανεξάρτητα από την εκτέλεση της καλούσας διεργασίας.

Παραδείγματα δημιουργίας νέων διεργασιών, είτε από το λειτουργικό είτε από το χρήστη είτε από κάποια άλλη διεργασία, συναντάμε πολλά. Ενδεικτικά είναι τα ακόλουθα:

- Δύο χρήστες κάθονται σε δύο τερματικά ενός συστήματος και αρχίζουν να εργάζονται. Το λειτουργικό σύστημα έχει εκκινήσει τουλάχιστον δύο διεργασίες που αντιστοιχούν στους φλοιούς του ΛΣ με τους οποίους αλληλεπιδρούν οι χρήστες.
- Ένας χρήστης χρησιμοποιεί μια διαλογική εφαρμογή για τη σχεδίαση ενός κτιρίου. Σε κάποιο σημείο και ενώ ο χρήστης σχεδιάζει, η εφαρμογή (που είναι διεργασία) πρέπει να πραγματοποιήσει σημαντικούς υπολογισμούς. Η εφαρμογή-διεργασία δημιουργεί μια νέα διεργασία που εκτελεί τους υπολογισμούς, ενώ η διεργασία σχεδίασης συνεχίζει να είναι ενεργή (για να σχεδιάζει ο χρήστης).

Κάθε στιγμή μια διεργασία μπορεί, ανάλογα με τις συνθήκες που επικρατούν στο σύστημα τη δεδομένη στιγμή, αλλά και με τις απαιτήσεις της διεργασίας σε πόρους συστήματος και σε δεδομένα από άλλες διεργασίες, να βρίσκεται σε μία από τις πιο κάτω καταστάσεις:

1. **Εκτελούμενη** *(running)*. Η διεργασία εκτελείται, απασχολεί δηλαδή πραγματικά την ΚΜΕ.
2. **Έτοιμη** *(runnable, ready)*. Η διεργασία έχει διακοπεί προσωρινά για να γίνει εκτελούμενη κάποια άλλη διεργασία. Μόλις της δοθεί η εντολή από το σύστημα, μπορεί να συνεχίσει την εκτέλεσή της.



Σχήμα 8.5 *Μια διεργασία μπορεί να είναι εκτελούμενη, έτοιμη ή υπό αναστολή. Οι δυνατές μεταβάσεις από κατάσταση σε κατάσταση σημειώνονται με βέλη.*

3. **Υπό αναστολή** *(blocked)*. Η διεργασία έχει διακοπεί προσωρινά επειδή *αδυνατεί* να συνεχίσει την εκτέλεσή της λόγω εξωτερικού (από αυτή) συμβάντος.

Η μεταβολή της κατάστασης μιας διεργασίας πραγματοποιείται με βάση ορισμένους κανόνες και προκαλείται είτε από την ίδια τη διεργασία είτε από το ΛΣ, άμεσα ή έμμεσα (μέσω του χρήστη).

Οι δυνατές μεταβολές είναι:

α. Από εκτελούμενη σε υπό αναστολή. Πραγματοποιείται όταν η διεργασία, ενώ εκτελείται, αποφασίζει ότι δε μπορεί να συνεχίσει. Παράδειγμα είναι η διεργασία που αναμένει είσοδο/έξοδο, πρόσβαση σε μη διαθέσιμη τη συγκεκριμένη στιγμή περιοχή μνήμης ή δεδομένα από άλλη διεργασία.

β. Από υπό αναστολή σε έτοιμη. Το εξωτερικό συμβάν, η αναμονή του οποίου

ανάγκασε τη διεργασία να ανασταλεί πραγματοποιείται. Η διεργασία μπορεί κατά συνέπεια να εκτελεστεί, γίνεται δηλαδή έτοιμη. Αν δεν υπάρχει εκτελούμενη διεργασία στο σύστημα, αμέσως μετά τη μετάβαση 2 πραγματοποιείται η μετάβαση 3.

γ. **Από έτοιμη σε εκτελούμενη.** Πραγματοποιείται με εντολή του ΛΣ και συγκεκριμένα του μέρους του, που ονομάζεται **χρονοδρομολογητής διεργασιών**. Το ΛΣ αποφασίζει, ότι η εκτελούμενη διεργασία έχει καταναλώσει αρκετό χρόνο της ΚΜΕ και είναι καιρός να εκτελεσθεί κάποια άλλη διεργασία. Το σύστημα επιλέγει με κάποιον αλγόριθμο μια έτοιμη διεργασία (έστω την Α) και της δίνει χρόνο επεξεργασίας. Η διεργασία Α έγινε από έτοιμη εκτελούμενη.

δ. **Από εκτελούμενη σε έτοιμη.** Πραγματοποιείται και αυτή με εντολή του *χρονοδρομολογητή διεργασιών.* Το ΛΣ αποφασίζει, ότι η εκτελούμενη διεργασία Α έχει καταναλώσει αρκετό χρόνο της ΚΜΕ, επιλέγει τη νέα διεργασία προς εκτέλεση και μετατρέπει την εκτελούμενη διεργασία Α σε έτοιμη. Με τον τρόπο αυτό, η διεργασία Α περιμένει να συνεχίσει την εκτέλεσή της αργότερα, αφού εξυπηρετηθούν και άλλες διεργασίες.

*Σε ένα σύστημα πολυπρογραμματισμού ονομάζουμε **παράλληλες ή ταυτόχρονες διεργασίες** (parallel processes) το σύνολο των εκτελέσιμων και των εκτελούμενων διεργασιών στις ΚΜΕ του συστήματος*

Η λογική και ο μηχανισμός με τον οποίο κατανέμεται ο χρόνος απασχόλησης της ΚΜΕ στις έτοιμες διεργασίες, αλλά και η επιλογή της έτοιμης διεργασίας, που κάθε φορά γίνεται εκτελούμενη από το χρονοδρομολογητή διεργασιών, είναι σημαντικότατο μέρος του ΛΣ και θα εξεταστεί στη συνέχεια.

Εκτός από τις μεταβάσεις μεταξύ καταστάσεων, σημαντικό είναι να γνωρίζουμε ότι στο μοντέλο των διεργασιών, κάθε στιγμή, μπορούν να συμβούν δύο ακόμη γεγονότα:

- **Δημιουργία μιας νέας διεργασίας.** Το ΛΣ, ένας χρήστης ή μια εκτελούμενη διεργασία μπορούν να δημιουργήσουν μια νέα διεργασία. Η νέα διεργασία θα είναι άμεσα *έτοιμη*.
- **Απομάκρυνση μιας διεργασίας από το σύστημα.** Μια *εκτελούμενη* διεργασία, μόλις ολοκληρωθεί, απομακρύνεται από το σύστημα (αφού δεν απαιτεί πια καμιά επεξεργασία). Στην περίπτωση αυτή έχουμε **ομαλό τερματισμό διεργασίας.** Απομάκρυνση διεργασιών μπορεί να πραγματοποιηθεί και από το χρήστη, το λειτουργικό ή από άλλη διεργασία με κλήση στο ΛΣ, ακόμα και αν η διεργασία δεν έχει ολοκληρωθεί. Στην περίπτωση αυτή έχουμε **μη ομαλό τερματισμό διεργασίας.**

Από τη στιγμή που δημιουργείται μια διεργασία και μέχρι να ολοκληρωθεί ή να τερματισθεί βίαια, ο κύκλος ζωής της αποτελείται από εναλλαγές επεξεργασίας από την ΚΜΕ, που ονομάζονται **εκρήξεις ΚΜΕ** *(CPU bursts)* και από αναμονές για την περάτωση εργασιών Εισόδου/Εξόδου (Ε/Ε) που ονομάζονται **εκρήξεις Ε/Ε** *(I/O bursts)* (Σχήμα 8.6).

Σχήμα 8.6

Οι διεργασίες μπορούν να κατηγοριοποιηθούν ανάλογα με το συνολικό χρόνο που καταναλώνουν σε εκρήξεις της μιας ή της άλλης μορφής. Έτσι, διακρίνονται σε

- **διεργασίες προσανατολισμένες στην ΚΜΕ** *(CPU bound)* όταν απαιτούν κυρίως εκρήξεις ΚΜΕ και σε
- **διεργασίες προσανατολισμένες στην Ε/Ε** *(I/O bound)* όταν απαιτούν κυρίως εκρήξεις Ε/Ε.

Μεταγωγή περιβάλλοντος

Κάθε φορά που το ΛΣ αλλάζει τη διεργασία που εκτελείται στην ΚΜΕ πρέπει να κρατήσει όλες τις πληροφορίες σχετικές με τη διεργασία που απέρχεται από την ΚΜΕ σε κάποιο σημείο, ώστε αυτές να είναι διαθέσιμες για τη μελλοντική συνέχιση της εκτέλεσής της. Για το σκοπό αυτό το ΛΣ τηρεί στη μνήμη μια δομή όπου αποθηκεύει όλες τις πληροφορίες για όλες τις διεργασίες που είναι έτοιμες ή υπό αναστολή. Όταν μια από αυτές γίνει εκτελούμενη, το ΛΣ διαβάζει από τη δομή όλα τα στοιχεία που είναι απαραίτητα για τη συνέχιση εκτέλεσης της διεργασίας. Το σύνολο των πληροφοριών που αποθηκεύονται στη μνήμη για τη διεργασία ονομάζεται **Σύνολο Ελέγχου Διεργασίας – ΣΕΔ** *(Process Control Block – PCB)*. Στην περίπτωση των νημάτων, το σύνολο των πληροφοριών που αποθηκεύονται στη μνήμη ονομάζεται **Σύνολο Ελέγχου Νήματος – ΣΕΝ** *(Thread Control Block – TCB)*.

Η εναλλαγή των διεργασιών και νημάτων στην ΚΜΕ ονομάζεται **μεταγωγή περιβάλλοντος** *(context switching)* και διαφέρει ανάλογα με το, αν εναλλάσσονται στην ΚΜΕ διεργασίες ή νήματα.

Οι πληροφορίες που κρατούνται κατά την μεταγωγή περιβάλλοντος για τη διεργασία που από εκτελέσιμη γίνεται έτοιμη ή υπό αναστολή είναι:

- Η επόμενη εντολή του προγράμματος που θα εκτελέσει η διεργασία όταν ξαναγίνει εκτελούμενη
- Η κατάσταση της ΚΜΕ (δηλαδή τα περιεχόμενα όλων των καταχωρητών της) ώστε αυτή να επαναφερθεί, όταν η διεργασία ξαναγίνει εκτελούμενη
- Όλα τα ενδιάμεσα αποτελέσματα που έχει υπολογίσει η διεργασία, ώστε αυτά να είναι πάλι διαθέσιμα, όταν η διεργασία ξαναγίνει εκτελούμενη

Τη διαδικασία μεταγωγής περιβάλλοντος τη συναντάμε καθημερινά και στη ζωή μας σε απλές ή πολύπλοκες μορφές. Μια απλή περίπτωση είναι, όταν διαβάζουμε ένα βιβλίο και κάποια στιγμή το αφήνουμε για να ασχοληθούμε με κάτι άλλο. Τοποθετούμε το σελιδοδείκτη στη σελίδα που μείναμε, ώστε αργότερα να συνεχίσουμε από τη σελίδα αυτή. Μια πιο σύνθετη περίπτωση είναι σε εστιατόριο με το σερβιτόρο που εξυπηρετεί πελάτες. Έρχεται και παίρνει την παραγγελία μας την οποία γράφει στο μπλοκάκι του. Στη συνέχεια (μεταγωγή περιβάλλοντος), πάει και παίρνει τα πιάτα μιας άλλης παραγγελίας και τα παραδίδει στο τραπέζι. Ύστερα (μεταγωγή περιβάλλοντος), βγάζει την παραγγελία μας από την τσέπη και παραδίδει το χαρτάκι στην κουζίνα. Αργότερα (μεταγωγή περιβάλλοντος), υπολογίζει το λογαριασμό για μια παρέα που φεύγει. Πριν όμως τον δώσει τον ειδοποιούν από την κουζίνα (μεταγωγή περιβάλλοντος), από όπου πηγαίνει και μας φέρνει την παραγγελία μας. Ευτυχώς κατά την τελευταία αλλαγή εργασίας (μεταγωγή περιβάλλοντος) είχε κρατήσει το χαρτάκι του λογαριασμού της παρέας που φεύγει, οπότε τελικά δίνει και σε αυτούς το λογαριασμό.

8.3 Συνθήκες Ανταγωνισμού και Διαδιεργασιακή Επικοινωνία

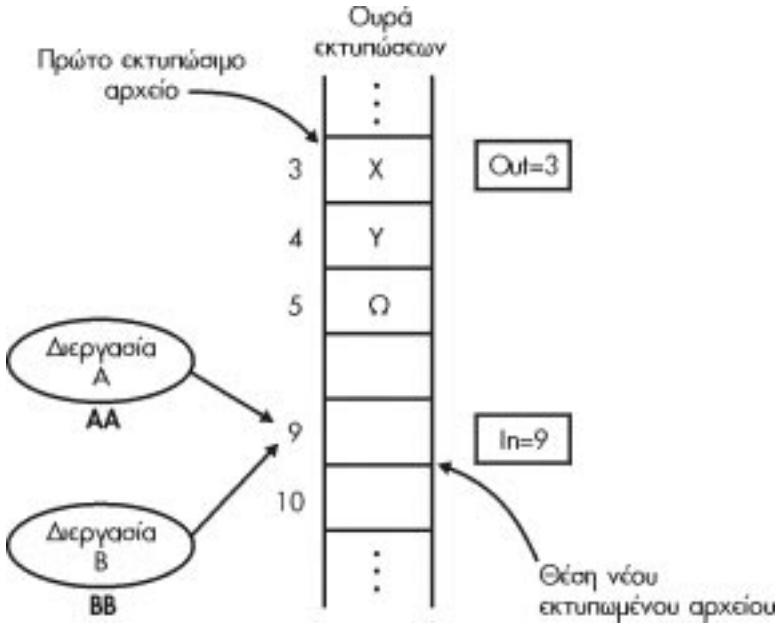
Παράδειγμα

Θεωρείστε ένα σύστημα υπολογιστή στο οποίο είναι εκτελέσιμες πολλές διεργασίες. Δύο από αυτές, η Α και η Β, μόλις γίνουν εκτελούμενες θα εκτυπώσουν ένα αρχείο, η Α το ΑΑ και η Β το ΒΒ. Το σύστημα, για την εξυπηρέτηση των εκτυπώσεων, τηρεί μια ουρά στη μνήμη, στην οποία κάθε διεργασία τοποθετεί το όνομα του προς εκτύπωση αρχείου της. Ο χειρισμός της ουράς γίνεται με δύο μεταβλητές, την out και την in. Η μεταβλητή out περιέχει τη διεύθυνση στην ουρά στην οποία βρίσκεται το πρώτο προς εκτύπωση αρχείο και η in τη διεύθυνση στην οποία θα τοποθετηθεί το νέο προς εκτύπωση αρχείο. Προφανώς, οι θέσεις της ουράς από την out μέχρι και την in-1 περιέχουν ονόματα αρχείων για εκτύπωση.

Παρακολουθείστε μια πιθανή ακολουθία γεγονότων στο σύστημα:

1. Η διεργασία Α γίνεται εκτελούμενη.
2. Η διεργασία Α διαβάζει τη μεταβλητή in για να δει σε ποια θέση της ουράς θα αποθηκεύσει το όνομα του αρχείου ΑΑ, το οποίο θα πρέπει να εκτυπωθεί. Σύμφωνα με την κατάσταση του συστήματος (Σχήμα 8.7), διαβάζει την τιμή 9.
3. Πριν προλάβει η διεργασία Α να αποθηκεύσει το όνομα του αρχείου ΑΑ στην ουρά και πριν ενημερώσει κάποια μεταβλητή, ο χρονοδρομολογητής του συστήματος τη μετατρέπει (για οποιοδήποτε λόγο, όπως αίτηση διακοπής από κάποια συσκευή, λήξη χρόνου εκτέλεσής της, κ.λπ.) σε έτοιμη. Η κατάσταση της Α μαζί με όλη την κατάσταση του συστήματος (και την τιμή 9 για τη μεταβλητή in) φυλάσσεται στο ΣΕΔ.
4. Η διεργασία Β γίνεται εκτελούμενη (επειδή έτσι αποφάσισε ο χρονοδρομολογητής).
5. Η διεργασία Β διαβάζει τη μεταβλητή in, για να δει σε ποια θέση της ουράς θα αποθηκεύσει το όνομα του αρχείου ΒΒ, το οποίο θα πρέπει να εκτυπωθεί. Σύμφωνα με την κατάσταση του συστήματος (Σχήμα 8.7), διαβάζει την τιμή 9.
6. Η διεργασία Β αλλάζει την τιμή της μεταβλητής in σε 9+1=10 και τοποθετεί στη θέση 9 της ουράς το όνομα αρχείου ΒΒ. Στη συνέχεια εκτελεί άλλες εντολές της.
7. Η διεργασία Α γίνεται εκτελούμενη (επειδή έτσι αποφάσισε ο χρονοδρομολογητής).
8. Η διεργασία Α συνεχίζει την εκτέλεσή της από το σημείο διακοπής της. Αυξάνει την τιμή που γνωρίζει για τη μεταβλητή in (είχε διαβάσει in=9), την κάνει 9+1=10 και τοποθετεί στη θέση 9 της ουράς το όνομα του αρχείου ΑΑ. Στη συνέχεια εκτελεί άλλες εργασίες.

Χρονοδρομολογητής *είναι το υποσύστημα του ΛΣ που αλλάζει και ελέγχει την κατάσταση των διεργασιών στο σύστημα.*



Σχήμα 8.7 *Κατάσταση της ουράς εκτυπώσεων στο σύστημα. Δύο διεργασίες προσπελαύνουν ταυτόχρονα μια διαμοιραζόμενη περιοχή μνήμης (μεταβλητή in και ουρά εκτύπωσης)*

Τί έχει συμβεί; Το όνομα του αρχείου ΒΒ έχει χαθεί από την ουρά, αφού στη θέση 9 υπάρχει το όνομα ΑΑ. Η ουρά είναι εσωτερικά συνεπής και δεν παρατηρεί κάποιο σφάλμα. Το επόμενο αρχείο θα τοποθετηθεί στην κενή θέση 10. Απλά το αρχείο ΒΒ δε θα τυπωθεί ποτέ, και κανένας δε θα λάβει μήνυμα για το γεγονός αυτό.

Το πρόβλημα, που ονομάζεται **"πρόβλημα κρίσιμου τμήματος"** δημιουργείται, επειδή στο διάστημα ανάμεσα στην ανάγνωση της διαμοιραζόμενης μεταβλητής in από την Α και στην ενημέρωσή της με τιμή αυξημένη κατά 1, έτσι ώστε να δεσμευτεί η θέση 9, η Α σταματά και η Β πραγματοποιεί την ίδια διαδικασία.

Η Α δεν ενημερώνεται ποτέ για τη νέα τιμή της μεταβλητής in. Το φαινόμενο που περιγράψαμε είναι δυνατό να εμφανιστεί σε κάθε περίπτωση που διεργασίες προσπελαύνουν την ίδια περιοχή μνήμης.

Οι συνθήκες κατά τις οποίες δύο οι περισσότερες διεργασίες διαμοιράζονται την ίδια περιοχή μνήμης και το τελικό αποτέλεσμα των ενεργειών τους εξαρτάται, ενώ δεν πρέπει, από τη σειρά και το χρόνο εκτέλεσής τους ονομάζονται **συνθήκες ανταγωνισμού** *(race conditions)*.

Συνθήκες ανταγωνισμού μπορούν να εμφανισθούν, όταν παράλληλες διεργασίες χρησιμοποιούν κοινούς πόρους. Κοινοί ή **μοιραζόμενοι πόροι** *(shared resources)* είναι όλοι οι πόροι του συστήματος, όπως π.χ. η μνήμη, ό δίσκος κ.λπ., που διατίθενται προς ανάγνωση ή εγγραφή στις διεργασίες.

Είναι προφανές ότι σε ένα σύστημα δεν πρέπει να εμφανίζονται συνθήκες ανταγωνισμού, αφού προκαλούν την εμφάνιση σοβαρών σφαλμάτων. Το χειρότερο αποτέλεσμά τους είναι πως τα προβλήματα που προκαλούν εμφανίζονται τυχαία, αφού εξαρτώνται καθαρά από μια γενικά άγνωστη σειρά στην εκτέλεση των διεργασιών και άγνωστη διάρκεια εκτέλεσης κάθε διεργασίας. Για να το διαπιστώσετε, εξετάστε το παράδειγμά μας, στο οποίο όμως το γεγονός 8 πραγματοποιείται πριν το

γεγονός 5 (η Β διακόπτεται πριν φθάσει στο σημείο να διαβάσει την μεταβλητή In και η Α ξαναγίνεται εκτελούμενη). Δείτε πως και το ΑΑ, και το ΒΒ θα τυπωθούν.

Η ανάγκη να αποφεύγεται η εμφάνιση συνθηκών ανταγωνισμού στο σύστημα είναι ένας από τους σημαντικότερους λόγους για την ύπαρξη στο σύστημα μηχανισμών επικοινωνίας μεταξύ διεργασιών. Η επικοινωνία αυτή, ονομάζεται **διαδιεργασιακή επικοινωνία** (**I**nter**P**rocess **C**ommunication - **IPC**).

Κυριότερος στόχος της είναι να εξασφαλίσει την **ακεραιότητα δεδομένων** *(data integrity)* σε περιπτώσεις όπου παράλληλες διεργασίες χρησιμοποιούν κοινούς πόρους στο σύστημα.

Εκτός από την αντιμετώπιση συνθηκών ανταγωνισμού, η επικοινωνία διεργασιών είναι επιθυμητή και σε άλλες περιπτώσεις, όπως καταστάσεις όπου η έξοδος μιας διεργασίας αποτελεί είσοδο μιας άλλης ή περιπτώσεις που η συνέχιση μιας διεργασίας απαιτεί είσοδο από την έξοδο κάποιας άλλης.

8.4 Αμοιβαίος Αποκλεισμός και Κρίσιμα Τμήματα Διεργασιών

Για να αποφύγουμε την εμφάνιση συνθηκών ανταγωνισμού, πρέπει να εξασφαλίσουμε ότι ποτέ δύο παράλληλες διεργασίες δεν θα επιχειρήσουν ταυτόχρονα πρόσβαση σε διαμοιραζόμενες περιοχές μνήμης ή γενικότερα σε διαμοιραζόμενους πόρους του συστήματος.

Ο αποκλεισμός μιας διεργασίας από κάποια ενέργεια, η οποία ταυτόχρονα επιτελείται από κάποια άλλη διεργασία, ονομάζεται **αμοιβαίος αποκλεισμός** *(mutual exclusion)*.

Στο παράδειγμα της προηγούμενης ενότητας, θα είχαμε αμοιβαίο αποκλεισμό, αν το σύστημα απαγόρευε στη διεργασία Β να διαβάσει την τιμή της μεταβλητής in, προτού η διεργασία Α διαβάσει τη μεταβλητή in και την ενημερώσει με τη νέα τιμή (δεσμεύοντας έτσι τη θέση 9 στην ουρά εκτύπωσης, ώστε η Β να γράψει στη θέση 10 και να μη χαθεί καμμιά εκτύπωση).

Εξετάζοντας τις διεργασίες, διαπιστώνουμε ότι κατά την εκτέλεσή κάθε μιας, αυτή:

- είτε ασχολείται με εσωτερικούς υπολογισμούς χρησιμοποιώντας αποκλειστικά δικούς της πόρους.
- είτε προσπελαύνει και χρησιμοποιεί διαμοιραζόμενες περιοχές μνήμης και διαμοιραζόμενους πόρους του συστήματος.

Στην πρώτη περίπτωση, η διεργασία δεν μπορεί να προκαλέσει ποτέ συνθήκες ανταγωνισμού. Στη δεύτερη περίπτωση όμως, είναι δυνατόν, η διεργασία να δημιουργήσει τις συνθήκες αυτές στο σύστημα.

Κάθε φορά που μια διεργασία εκτελεί μέρος του προγράμματός της, στο οποίο προσπελαύνει διαμοιραζόμενες περιοχές μνήμης και πόρους του συστήματος, λέμε ότι η διεργασία αυτή έχει εισέλθει σε **κρίσιμο τμήμα της** *(critical section)*.

Κατά την εκτέλεσή της, κάθε διεργασία εισέρχεται εναλλάξ σε κρίσιμα και μη κρίσιμα τμήματα της. Αν το σύστημα εξασφαλίσει ότι ποτέ δύο παράλληλες διερ-

γασίες δεν βρίσκονται ταυτόχρονα σε κρίσιμο τμήμα τους, τότε έχει εξασφαλίσει αμοιβαίο αποκλεισμό και συνεπώς έχει αποκλείσει την εμφάνιση συνθηκών ανταγωνισμού.

Η εξασφάλιση εκτέλεσης ενός μόνο κρίσιμου τμήματος κάθε στιγμή στο σύστημα, αποτελεί μια προϋπόθεση για τη σωστή λειτουργία του συστήματος. Δεν είναι η μόνη. Για τη σωστή και αποδοτική επικοινωνία και συνεργασία των παράλληλων διεργασιών ενός συστήματος, πρέπει να ικανοποιούνται οι ακόλουθες τέσσερις συνθήκες (διατυπώθηκαν από το Dijkstra το 1965):

1. Δύο διεργασίες δε βρίσκονται ποτέ ταυτόχρονα στα κρίσιμα τμήματά τους (αμοιβαίος αποκλεισμός).
2. Αν μια διεργασία δε βρίσκεται σε κρίσιμο τμήμα της, τότε δεν επιτρέπεται να αναστείλλει άλλες διεργασίες.
3. Απαγορεύονται οι υποθέσεις σε ότι αφορά στην ταχύτητα και στο πλήθος των επεξεργαστών στο σύστημα.
4. Απαγορεύεται σε κάθε διεργασία να περιμένει επ' αόριστο για να εισέλθει στο κρίσιμο τμήμα της.

Οι συνθήκες 2 και 4 προφυλάσσουν το σύστημα από το **αδιέξοδο** *(deadlock)* ή αλλιώς το **αμοιβαίο μπλοκάρισμα** *(mutual blocking)*, όπου όλες οι διεργασίες εμποδίζουν η μια την άλλη να εκτελέσει το κρίσιμο τμήμα της, ενώ μία από αυτές θα μπορούσε να το εκτελέσει.

Έχοντας ορίσει το **κρίσιμο τμήμα** των διεργασιών, μπορούμε να πούμε ότι:

α. Συνθήκες ανταγωνισμού είναι οι συνθήκες που εμφανίζονται στο σύστημα, όταν δύο παράλληλες διεργασίες ταυτόχρονα εισέρχονται σε κρίσιμο τμήμα τους.

β. Αμοιβαίος αποκλεισμός είναι η απαγόρευση σε μια διεργασία να εισέλθει σε κρίσιμο τμήμα της, όταν κάποια άλλη διεργασία βρίσκεται ήδη σε δικό της κρίσιμο τμήμα.

8.5 Απλοί Μηχανισμοί Αμοιβαίου Αποκλεισμού

Στην παρούσα ενότητα θα εξετάσουμε δύο μηχανισμούς που επιτυγχάνουν τον αμοιβαίο αποκλεισμό ταυτόχρονων διεργασιών σε ένα σύστημα. Εκτός από τους μηχανισμούς της απενεργοποίησης διακοπών και της εντολής **TSL** (*Test and Set Lock* – Δοκίμασε και θέσε κλείδωμα), ιδιαίτερα γνωστή είναι και η λύση του Peterson, η οποία περιγράφεται εκτενώς στη βιβλιογραφία που δίνεται στο τέλος του κεφαλαίου.

8.5.1 Απενεργοποίηση διακοπών

Μια απλή λύση για την επιτυχία αμοιβαίου αποκλεισμού είναι η απενεργοποίηση των διακοπών (interrupts) του συστήματος, κάθε φορά που μια διεργασία εισέρχεται σε κρίσιμο τμήμα της. Ο μηχανισμός αυτός εγγυάται ότι, αν μια διεργασία μπει σε κρίσιμο τμήμα, τότε η εκτέλεση του τμήματος αυτού θα ολοκληρωθεί, αφού το σύστημα δε θα μπορεί να διακόψει την εκτέλεση της διεργασίας. Μόλις ολοκληρωθεί το κρίσιμο τμήμα, οι διακοπές του συστήματος ενεργοποιούνται ξανά.

Η λύση αυτή παρουσιάζει, εκτός του πλεονεκτήματος της απλότητας, σοβαρά μειονεκτήματα:

- Η ενεργοποίηση/απενεργοποίηση των διακοπών του συστήματος είναι ευθύνη των διεργασιών. Ενώ αυτό μπορεί να είναι αποδεκτό για τις διεργασίες του πυρήνα του συστήματος, θεωρείται μη αποδεκτό για τις διεργασίες χρηστών. Ο λόγος είναι ότι, αν μια διεργασία χρήστη δεν ενεργοποιεί/απενεργοποιεί σωστά τις διακοπές του συστήματος, το σύστημα μπορεί να καταρρεύσει, γεγονός που είναι απαράδεκτο.
- Αν το σύστημα περιλαμβάνει περισσότερες της μιας ΜΕ (μονάδες επεξεργασίας), τότε η απενεργοποίηση των διακοπών πραγματοποιείται μόνο στην ΜΕ που εκτελεί τη διεργασία, που εισέρχεται σε κρίσιμο τμήμα. Είναι δυνατόν άλλες διεργασίες να εισέλθουν σε κρίσιμα τμήματα κατά την εκτέλεσή τους σε άλλες ΜΕ του συστήματος. Η απενεργοποίηση των διακοπών όλων των ΜΕ, όταν σε μία από αυτές εισέρχεται διεργασία για εκτέλεση κρίσιμου τμήματος, είναι ένα ιδιαίτερα σύνθετο πρόβλημα, περισσότερο ίσως και από το πρόβλημα του αμοιβαίου αποκλεισμού.

Σαν συμπέρασμα, μπορούμε να πούμε ότι ο μηχανισμός απενεργοποίησης διακοπών, ενώ μπορεί να είναι απλός και συχνά βολικός για την περίπτωση διεργασιών του πυρήνα του ΛΣ, είναι γενικά μη αποδεκτός για χρήση από διεργασίες χρηστών. Για το λόγο αυτό, σπάνια προτιμάται προς υλοποίηση σε σύγχρονα συστήματα.

8.5.2 Η Εντολή TSL

Ο μηχανισμός TSL απαιτεί τη συνδρομή ειδικού υλικού. Το σύστημα παρέχει μια εντολή, την "Έλεγξε και Θέσε Κλείδωμα" (Test and Set Lock), με την οποία η ΚΜΕ διαβάζει το περιεχόμενο μιας θέσης μνήμης και αποθηκεύει στη θέση αυτή μη μηδενική τιμή.

> Η ανάγνωση και η εγγραφή στη μνήμη γίνονται αδιαίρετα, ενώ ταυτόχρονα και όσο διαρκεί η εκτέλεση της εντολής, αποκλείεται και η πρόσβαση οποιουδήποτε άλλου επεξεργαστή στη συγκεκριμένη θέση μνήμης.

Ο συγχρονισμός διεργασιών και ο αμοιβαίος αποκλεισμός επιτυγχάνονται στο σύστημα μέσω μιας διαμοιραζόμενης μεταβλητής flag. Ο μηχανισμός λειτουργεί ως εξής:

- Όταν η μεταβλητή έχει την τιμή 0, τότε καμιά διεργασία δε βρίσκεται σε κρίσιμο τμήμα της. Όταν η μεταβλητή έχει τιμή 1, τότε κάποια διεργασία στο σύστημα βρίσκεται σε κρίσιμο τμήμα της.
- Κάθε φορά που μια διεργασία θέλει να μπει σε κρίσιμο τμήμα, χρησιμοποιεί την εντολή TSL και ελέγχει την τιμή της flag. Αν τη βρει 0, αυτόματα τη θέτει 1 και μπαίνει στο κρίσιμο τμήμα της. Αν τη βρει 1, δεν εισέρχεται στο κρίσιμο τμήμα, παρά επαναλαμβάνει τον έλεγχο.
- Όταν μια διεργασία ολοκληρώσει το κρίσιμο τμήμα της, τότε με μια απλή εντολή θέτει την τιμή της flag σε 0.

Είναι προφανές ότι ποτέ δύο διεργασίες δε θα βρίσκονται ταυτόχρονα σε κρίσιμα τμήματά τους. Η **αδιαίρετη εκτέλεση** ανάγνωσης και εγγραφής από και προς τη flag **και ο αποκλεισμός της μεταβλητής** από όλους τους άλλους επεξεργαστές του συστήματος, που πετυχαίνει η εντολή TSL, εγγυάται, ότι ποτέ δεν πρόκειται μια διεργασία να προσπελάσει τη flag, ενώ αυτή προσπελαύνεται ήδη από κάποια άλλη.

8.6 Ενεργός Αναμονή και Απενεργοποίηση-Αφύπνιση

Ο μηχανισμός TSL που εξετάσαμε στην ενότητα 8.5 έχει μια σοβαρή ατέλεια. Κάθε διεργασία που θέλει να εισέλθει σε κρίσιμο τμήμα της, ελέγχει συνεχώς την τιμή μιας διαμοιραζόμενης μεταβλητής μέχρι να βρει σε αυτήν την τιμή που θα της επιτρέψει να εισέλθει στο κρίσιμο τμήμα της. Όλες οι διεργασίες που αναμένουν, καταναλώνουν χρόνο σε ΜΕ του συστήματος, αφού εκτελούν ένα βρόχο που περιλαμβάνει τον έλεγχο της μεταβλητής. Οι διεργασίες αυτές, όσο περιμένουν δεν παράγουν κανένα έργο.

Ο επαναλαμβανόμενος έλεγχος της διαμοιραζόμενης μεταβλητής ονομάζεται **ενεργός αναμονή** *(busy waiting)* και είναι προφανές ότι σπαταλά χρόνο του επεξεργαστή (εκτελώντας συνεχώς τον έλεγχο της διαμοιραζόμενης μεταβλητής)

Στη συνέχεια, θα εξετάσουμε το μηχανισμό σηματοφορέων, ο οποίος αναγκάζει τις διεργασίες που θέλουν να εισέλθουν σε κρίσιμα τμήματα να **απενεργοποιούνται** αντί να καταναλώνουν χρόνο της ΚΜΕ, **ενώ περιμένουν**.

8.7 Σηματοφορείς

Ο μηχανισμός των σηματοφορέων, αποτελεί έναν από τους πιο αποδοτικούς και πιο διαδεδομένους μηχανισμούς διαδιεργασιακής συνεργασίας. Άλλοι ανάλογοι μηχανισμοί, που χρησιμοποιούνται και οι οποίοι καλύπτονται στην προτεινόμενη βιβλιογραφία, είναι αυτοί των **παρακολουθητών,** των **μετρητών συμβάντων και της ανταλλαγής μηνυμάτων.** Αποδεικνύεται ότι όλοι οι μηχανισμοί αυτοί είναι εντελώς ισοδύναμοι.

Ας εξετάσουμε μια εταιρεία με πολλούς εργαζόμενους, που μοιράζονται δύο τηλεφωνικές γραμμές, μία για τοπικά τηλεφωνήματα και μία για υπεραστικά. Κάθε εργαζόμενος έχει μια τηλεφωνική συσκευή συνδεδεμένη και στις δύο γραμμές. Κάθε συσκευή έχει δύο λαμπάκια, ένα για κάθε γραμμή. Όταν μια γραμμή είναι ελεύθερη, τότε το αντίστοιχο λαμπάκι σε όλες τις συσκευές είναι σβηστό. Όταν είναι κατειλημένη, το αντίστοιχο λαμπάκι είναι αναμμένο σε όλες τις συσκευές. Επιπλέον, το σύστημα έχει ρυθμιστεί, έτσι ώστε κανένας να μην μπορεί να ακούσει συνομιλία άλλου, αν σηκώσει το ακουστικό ενώ η γραμμή είναι κατειλημένη.

Οταν κάποιος θέλει να τηλεφωνήσει, εξετάζει το λαμπάκι της γραμμής που θέλει να δεσμεύσει (ανάλογα με το τηλεφώνημα που πρόκειται να κάνει). Αν το δει αναμμένο, περιμένει εκτελώντας άλλες εργασίες μέχρι το λαμπάκι να σβήσει. Αν το δει σβηστό, σηκώνει το ακουστικό και καλεί τον αριθμό που επιθυμεί. Σε αυτή την περίπτωση, η γραμμή δεσμεύεται και το λαμπάκι της ανάβει σε όλες τις συσκευές. Μόλις τελειώσει το τηλεφώνημά του κατεβάζει το ακουστικό, οπότε η γραμμή ελευθερώνεται και το αντίστοιχο λαμπάκι σβήνει σε όλες τις συσκευές. Η γραμμή είναι διαθέσιμη στον επόμενο που θέλει να τη χρησιμοποιήσει.

Το σύστημα που περιγράψαμε είναι ταυτόσημο με ένα υπολογιστή που χρησιμοποιεί δύο σηματοφορείς για να πετύχει αμοιβαίο αποκλεισμό παράλληλων διεργασιών που επιθυμούν χρήση κοινών πόρων του συστήματος:

- Οι εργαζόμενοι είναι οι διεργασίες
- Οι δύο τηλεφωνικές γραμμές είναι οι μοιραζόμενοι πόροι
- Τα δύο λαμπάκια στις συσκευές είναι οι δύο σηματοφορείς του συστήματος
- Η χρήση μιας τηλεφωνικής γραμμής από έναν εργαζόμενο ισοδυναμεί με την εκτέλεση κρίσιμου τμήματος μιας διεργασίας.

Όπως και στην εταιρεία, έτσι και στον υπολογιστή οι σηματοφορείς έχουν (συνήθως) δύο τιμές:

- Τιμή = 1 (ισοδυναμεί με σβηστό λαμπάκι) που δηλώνει ότι ο μοιραζόμενος πόρος είναι ελεύθερος).
- Τιμή = 0 (ισοδυναμεί με αναμμένο λαμπάκι) που δηλώνει ότι ο μοιραζόμενος πόρος χρησιμοποιείται.

Παρατηρείστε οτι κάθε σηματοφορέας, σε αντιστοιχία με τα λαμπάκια, ελέγχει τη χρήση ενός μόνο κοινού πόρου.

Θα δούμε τώρα το γενικό ορισμό των σηματοφορέων, που προτάθηκε από τον Dijkstra το 1965:

Ο **σηματοφορέας** *(semaphore)* είναι ένας τύπος ακέραιας μεταβλητής που χρησιμοποιείται για την καταμέτρηση των σημάτων απενεργοποίησης/αφύπνισης προς μια διεργασία, ώστε αυτά τα σήματα να μη χάνονται, αλλά να μπορούν να χρησιμοποιούνται μελλοντικά.

Σχήμα 8.8 *Οι διεργασίες σε ένα ΛΣ χρησιμοποιούν τους σηματοφορείς για να επικοινωνούν μεταξύ τους και για να συγχρονίζονται.*

Η τιμή ενός σηματοφορέα είναι:

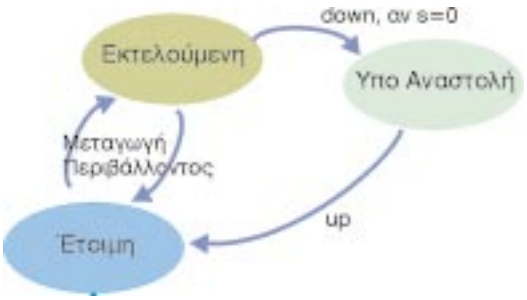
- 0, όταν δεν έχει αποσταλεί κανένα σήμα αφύπνισης σε διεργασία
- σ>0, όταν έχουν αποσταλεί σ το πλήθος, σήματα αφύπνισης σε διεργασίες

Οταν ένας σηματοφορέας οριστεί να παίρνει τιμές 0 και 1, μόνο τότε ονομάζεται **δυαδικός σηματοφορέας** και χρησιμοποιείται για την εξασφάλιση αμοιβαίου αποκλεισμού στο σύστημα. Οι σηματοφορείς που χρησιμοποιούνται για την επικοινωνία διεργασιών παίρνουν και θετικές τιμές μεγαλύτερες του 1.

Σε ένα σύστημα με σηματοφορείς ορίζονται οι κλήσεις συστήματος **down** και **up:**

- Κλήση **down(s)**. Όταν καλείται από τη διεργασία Χ, ελέγχει αν ο σηματοφορέας **s** έχει τιμή μεγαλύτερη του 0. Αν s>0 τότε μειώνει την τιμή του s κατά 1. Αν όμως s=0, τότε η διεργασία Χ απενεργοποιείται (αναστέλλεται).
- Κλήση **up(s)**. Η κλήση αυξάνει την τιμή του σηματοφορέα s κατά 1. Αν υπάρχουν στο σύστημα διεργασίες που έχουν απενεργοποιηθεί μέσω του s - δεν μπόρεσαν να εκτελέσουν την down(s) – το σύστημα επιλέγει μία την οποία και αφυπνίζει (την κάνει έτοιμη).

Είναι προφανές ότι, αν κάποια διεργασία αφυπνιστεί μέσω της κλήσης up(s), τότε αυτή θα εκτελέσει αμέσως την κλήση down(s), αφού εκεί είχε απενεργοποιηθεί. Ο σηματοφορέας s, μετά τις up(s) και down(s) θα έχει πάλι τιμή 0.



Σχήμα 8.9 *Αλλαγή κατάστασης διεργασιών σε σύστημα με σηματοφορείς (s)*

Αν ξαναγυρίσουμε στο παράδειγμα της εταιρείας με τις δύο τηλεφωνικές γραμμές, βλέπουμε οτι:

ο έλεγχος που κάνει ο εργαζόμενος στο λαμπάκι με σκοπό να πάρει τη γραμμή ισοδυναμεί με την κλήση down(s) στο σηματοφορέα της κατάλληλης γραμμής. Αν το λαμπάκι είναι σβηστό (s=1), η κλήση down μειώνει το σηματοφορέα κατά 1 (γίνεται s=0) και ο εργαζόμενος τηλεφωνεί. Ο επόμενος εργαζόμενος βλέπει πλέον το λαμπάκι αναμμένο (s=0) οπότε, όταν προσπαθεί να εκτελεί την down(s), περιμένει.

Η απελευθέρωση της γραμμής από έναν εργαζόμενο, όταν λήξει το τηλεφώνημά του, ισοδυναμεί με τη λειτουργία up(s) στο σηματοφορέα της κατάλληλης γραμμής. Ο σηματοφορέας αυξάνει (γίνεται s=1), οπότε αυτόματα κάποιος άλλος που περίμενε (επειδή είχε δει s=0 δηλαδή λαμπάκι αναμένο) μπορεί να πάρει τη γραμμή (να εκτελέσει δηλαδή τη δική του down(s)).

Κρίσιμο σημείο στο μηχανισμό των σηματοφορέων είναι το γεγονός **ότι και η κλήση down και η κλήση up είναι αδιαίρετες**. Κάθε φορά που μια διεργασία ξεκινά μια κλήση up ή down για ένα σηματοφορέα η κλήση ολοκληρώνεται, πριν δοθεί ο έλεγχος σε άλλη διεργασία. Επιπλέον, σε σύστημα με πολλές μονάδες επεξεργασίας (ΜΕ), **καμιά άλλη παράλληλη διεργασία δεν έχει πρόσβαση στο σηματοφορέα** αυτό. Με άλλα λόγια:

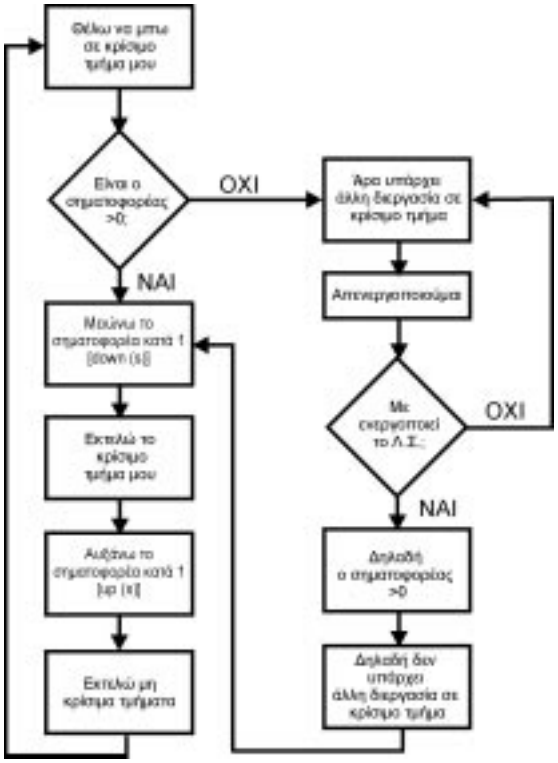
- Όταν εκτελείται η κλήση down(s) ο έλεγχος και μεταβολή της τιμής του s ή η απενεργοποίηση της καλούσας διεργασίας ολοκληρώνονται χωρίς διακοπή. Παράλληλα, κατά την εκτέλεση της κλήσης, καμμία άλλη ΜΕ δεν μπορεί να εκτελέσει λειτουργία down(s) ή up(s).
- Όταν εκτελείται η κλήση up(s), η μεταβολή της τιμής του s και η πιθανή αφύπνιση κάποιας διεργασίας ολοκληρώνονται χωρίς διακοπή. Παράλληλα κατά την εκτέλεση της κλήσης, καμιά άλλη ΜΕ δεν μπορεί να εκτελέσει λειτουργία down(s) ή up(s).

Σε ένα σύστημα με πολλούς κοινόχρηστους πόρους υπάρχουν και πολλοί σηματοφορείς. Το ΛΣ, για να μπορεί να διαχειρίζεται εύκολα τις διεργασίες που περιμένουν κάθε σηματοφορέα, χωρίζει τις διεργασίες που έχουν απενεργοποιηθεί (βρίσκονται

υπό αναστολή) σε τόσες **λίστες αναμονής** όσοι και οι σηματοφορείς. Κάθε διεργασία περιμένει στη λίστα αναμονής του σηματοφορέα που την έχει απενεργοποιήσει. Στο σύστημα υπάρχουν φυσικά και διεργασίες που δεν περιμένουν κανένα σηματοφορέα. Αυτές είναι οι έτοιμες διεργασίες και περιμένουν εξυπηρέτηση από την ΚΜΕ σε ξεχωριστή λίστα, τη **λίστα έτοιμων διεργασιών** *(Ready List – RL)*.

Οι λίστες αναμονής των σηματοφορέων είναι οργανωμένες ως ουρές. Κάθε φορά εξυπηρετείται η διεργασία που περιμένει στη λίστα την περισσότερη ώρα (όπως και στα ταμεία της τράπεζας). Οργάνωση ουράς έχει συνήθως και η λίστα έτοιμων διεργασιών.

Στο σχήμα 8.10 φαίνεται η χρήση των σηματοφορέων από μια διεργασία που εκτελείται και ζητά να εισέλθει σε κρίσιμο τμήμα της. Παρατηρείστε οτι, ανάλογα με την κατάσταση (τιμή) του σηματοφορέα, απενεργοποιείται ή συνεχίζει την εκτέλεσή της. Επίσης, αν η διεργασία έχει απενεργοποιηθεί και ο σηματοφορέας αλλάξει τιμή, τότε το ΛΣ ενεργοποιεί και πάλι τη διεργασία.



Σχήμα 8.10 *Χρήση σηματοφορέα από διεργασία για εκτέλεση κρίσιμου τμήματός της*

Το πρόβλημα συγγραφέων και αναγνωστών

Θα εξετάσουμε τον τρόπο που χρησιμοποιούνται οι σηματοφορείς για να λύσουν ένα από τα κλασσικότερα προβλήματα στην επιστήμη των υπολογιστών, το πρόβλημα συγγραφέων και αναγνωστών. Το πρόβλημα εμφανίζεται με διάφορες μορφές αλλά πιο συνηθισμένο είναι στη μορφή που παίρνει στις βάσεις δεδομένων.

Εκεί, μια ομάδα διεργασιών μοιράζεται ενα αρχείο, στο οποίο κάποιες από τις διεργασίες (συγγραφείς) θέλουν να γράψουν πληροφορίες και να το τροποποιήσουν, ενώ κάποιες άλλες διεργασίες (αναγνώστες) θέλουν να διαβάσουν πληροφορία από αυτό.

Για να εξασφαλίσουμε την ακεραιότητα των δεδομένων, πρέπει κάθε συγγραφέας να έχει αποκλειστική πρόσβαση στο αρχείο. Όταν γράφει δηλαδή μια διεργασία στο αρχείο, καμία άλλη δεν θα πρέπει να μπορεί να το χρησιμοποιήσει ούτε για εγγραφή αλλά ούτε και για ανάγνωση. Στην περίπτωση της ανάγνωσης, αντίθετα, μπορούμε να έχουμε πολλές διεργασίες να διαβάζουν ταυτόχρονα. Σε περίπτωση που την ίδια στιγμή ενεργοποιούνται ένας συγγραφέας και ένας αναγνώστης θα θεωρήσουμε οτι προτεραιότητα έχει ο αναγνώστης. Ο συγγραφέας δηλαδή πρέπει να περιμένει να ολοκληρωθούν όλες οι εργασίες ανάγνωσης. Η παραδοχή αυτή δεν είναι υποχρεωτική, θα μπορούσε να συμβαίνει και το αντίστροφο. Η μόνη διαφορά θα ήταν πως πιο ευνοημένοι θα ήταν τότε οι συγγραφείς και όχι οι αναγνώστες.

Οι απλούστερες μορφές των διεργασιών των συγγραφέων και των αναγνωστών φαίνονται στο σχήμα 8.11.

```
Συγγραφείς
Write(file, text)
Αναγνώστες
Read(file, text)
```

Σχήμα 8.11

Για να εξασφαλίσουμε ότι μόνο ένας συγγραφέας γράφει πάντα στο αρχείο, θα χρησιμοποιήσουμε ένα σηματοφορέα με αρχική τιμή 1. Θα τον ονομάσουμε w. Κάθε φορά που ένας συγγραφέας θέλει να γράψει, θα καταλαμβάνει το σηματοφορέα με την κλήση down(w). Όσο γράφει, όλοι οι άλλοι συγγραφείς, αν προσπαθήσουν να γράψουν, θα βρίσκουν w=0 και θα απενεργοποιούνται στην δική τους κλήση down(w). Το νέο πρόγραμμα των συγγραφέων φαίνεται στο σχήμα 8.12.

```
Συγγραφείς

Down(w)

Write(file, text)

Up(w)
```

Σχήμα 8.12

Πρέπει τώρα να εξασφαλίσουμε οτι, όταν ένας αναγνώστης διαβάζει, κανένας συγγραφέας δεν θα μπορεί να γράψει. Αυτό θα γίνει μέσω του σηματοφορέα w που ελέγχουν οι συγγραφείς πριν γράψουν. Αν κάθε φορά που ενεργοποιείται ενας αναγνώστης δεσμεύει το σηματοφορέα w, τότε οι συγγραφείς θα μπλοκάρονται. Οταν ολοκληρώσει ο αναγνώστης, θα ελευθερώνει το σηματοφορέα με την κλήση up(w) οπότε, αν υπάρχει συγγραφέας που περιμένει, θα ενεργοποιείται. Αν όμως οι αναγνώστες ελέγχουν τον w, τότε πώς θα μπορούν να διαβάζουν πολλοί ταυτόχρονα; Αυτό θα λυθεί με την προσθήκη μιας κοινής μεταβλητής, της readers. Η μεταβλητή έχει αρχική τιμή 0. Κάθε αναγνώστης την αυ-

```
Αναγνώστες
readers := readers + 1

If readers = 1 then
        Down(w)

Read(file, text)

readers := readers – 1

if readers = 0 then
        up(w)
```

Σχήμα 8.13

ξάνει πριν αρχίσει το διάβασμα και τη μειώνει οταν τελειώσει. Κάθε στιγμή η readers, δηλώνει τον αριθμό των αναγνωστών που διαβάζουν. Χρησιμοποιώντας την readers ενας αναγνώστης γνωρίζει αν είναι ο πρώτος ή ο τελευταίος που διαβάζει. Στον κώδικα των αναγνωστών βάζουμε έλεγχο, έτσι ώστε μόνο ο πρώτος να δεσμεύει τον σηματοφορέα w και μόνο ο τελευταίος να τον ελευθερώνει. Ο κώδικας φαίνεται στο σχήμα 8.13.

Κοινή μεταβλητή σημαίνει και ανάγκη σηματοφορέα. Πράγματι η readers μπορεί να μεταβάλλεται ανεξέλεγκτα από πολλούς ταυτόχρονα αναγνώστες. Αυτό μπορεί να δημιουργήσει σοβαρά προβλήματα και κατάρρευση του συστήματος. Η σωστή λύση απαιτεί η πρόσβαση των αναγνωστών στη μεταβλητή να φυλάσσεται με σηματοφορέα (τον r). Ο τελικός κώδικας του αναγνώστη δίνεται στο σχήμα 8.14.

Αναγνώστες

```
Down(r)
readers :=  readers + 1
up(r)

If readers = 1 then
          Down(w)

Read(file, text)

Down(r)
readers :=  readers - 1
up(r)

if readers = 0 then
          up(w)
```

Σχήμα 8.14

8.8 Χρονοδρομολόγηση Επεξεργαστή

Στο παρελθόν, στα περισσότερα συστήματα ήταν δυνατή η εκτέλεση μιας μόνο διεργασίας από το σύστημα υπολογιστή. Η ΚΜΕ αφιερωνόταν στη διεργασία αυτή και μόνο με τη λήξη της ήταν δυνατό το σύστημα ή κάποιος χρήστης να εκκινήσει μια νέα. Αντίθετα, σε κάθε σύγχρονο σύστημα είναι πολύ συνηθισμένο να υπάρχουν κάθε στιγμή αρκετές διεργασίες, είτε του ΛΣ είτε ενός ή περισσότερων χρηστών, οι οποίες είναι εκτελέσιμες. Στις περιπτώσεις όπου υπάρχουν περισσότερες της μιας έτοιμες διεργασίες, πρέπει το σύστημα να αποφασίσει ποια θα εκτελεστεί πρώτη και πώς ο χρόνος της ΚΜΕ θα κατανεμηθεί στις διεργασίες αυτές.

Το υποσύστημα του ΛΣ που καθορίζει με ποιο τρόπο και ποια σειρά θα κατανέμεται ο χρόνος της επεξεργαστικής ισχύος στις εκτελέσιμες διεργασίες ονομάζεται **χρονοδρομολογητής** *(scheduler)*. Ο αλγόριθμος που χρησιμοποιεί ο χρονοδρομολογητής για την εκτέλεση της εργασίας του, ο τρόπος δηλαδή που γίνεται ο χρονοδρομολόγηση, ονομάζεται **αλγόριθμος χρονοδρομολόγησης** *(scheduling algorithm)*.

Επίπεδα Χρονοδρομολόγησης

Σε κάθε σύστημα υπάρχουν δύο επίπεδα χρονοδρομολόγησης και συνεπώς δύο χρονοδρομολογητές (άρα και 2 ίδιοι ή διαφορετικοί αλγόριθμοι χρονοδρομολόγησης)

- **Η μακροχρόνια χρονοδρομολόγηση ή χρονοδρομολόγηση εργασιών** *(long term scheduling ή job scheduling)*. Σε αυτό το επίπεδο, το σύστημα επιλέγει από όλες τις διεργασίες που έχουν υποβληθεί στο σύστημα, ποιες θα έρθουν στη μνήμη (από το μέσο αποθήκευσης που βρίσκονται) και θα εκτελεσθούν. Οι διεργασίες αυτές εισάγονται στη λίστα έτοιμων διεργασιών του συστήματος.
- **Η βραχυχρόνια χρονοδρομολόγηση ή χρονοδρομολόγηση ΚΜΕ** *(short term scheduling ή CPU scheduling)*. Στο επίπεδο αυτό, το σύστημα επιλέγει ποια από τις διεργασίες που έχουν επιλεγεί από τη μακροχρόνια χρονοδρομολόγηση θα εκτελείται κάθε φορά στην ΚΜΕ.

Η βασική διαφορά των δύο διαδικασιών είναι η συχνότητα με την οποία εκτελούνται. Ο μακροχρόνιος χρονοδρομολογητής εκτελείται αυτόματα από το σύστημα σε αραιά χρονικά διαστήματα (π.χ. κάθε 500msec). Ο βραχυχρόνιος χρονοδρομολογητής εκτελείται αυτόματα από το σύστημα πολύ πιο συχνά (π.χ. κάθε 10msec).

Σε πολλά συστήματα υπάρχει και ένα ενδιάμεσο επίπεδο χρονοδρομολόγησης, ο **μεσοπρόθεσμος χρονοδρομολογητής** *(mid-term scheduler)*. Στο επίπεδο αυτό, ο χρονοδρομολογητής επιλέγει από τη λίστα έτοιμων διεργασιών διεργασίες τις οποίες απομακρύνει από τη λίστα και στη θέση τους φέρνει άλλες από τα αποθηκευτικά μέσα του συστήματος.

Αξιολόγηση και τύποι χρονοδρομολόγησης

Ο βασικός στόχος ενός αλγόριθμου χρονοδρομολόγησης είναι να δίνει επεξεργαστική ισχύ σε εκτελέσιμες διεργασίες με τέτοιο τρόπο, ώστε να επιτυγχάνει:

- **Αποδοτικότητα** *(efficiency)*. Οι ΚΜΕ πρέπει να είναι απασχολημένες διεκπεραιώνοντας διεργασίες για το 100% του χρόνου τους.
- **Δικαιοσύνη** *(fairness)*, δηλαδή ο χρόνος των ΚΜΕ να μοιράζεται δίκαια (ανάλογα με τις γενικότερες απαιτήσεις) στις έτοιμες διεργασίες.
- **Χαμηλό χρόνο απόκρισης** *(low response time)*, δηλαδή ελάχιστο χρόνο αναμονής για τους διαλογικούς χρήστες (interactive users).
- **Χαμηλό χρόνο διεκπεραίωσης** *(low turnaround time)*, δηλαδή ελάχιστο χρόνο αναμονής για τους χρήστες ενεργειών δέσμης (batch process users).

Είναι προφανές ότι τα πιο πάνω κριτήρια επιτυχίας, τα οποία δεν είναι και τα μόνα, είναι ανταγωνιστικά. Για παράδειγμα, ελαχιστοποίηση του χρόνου απόκρισης αυτόματα σημαίνει αύξηση του χρόνου διεκπεραίωσης διεργασιών δέσμης και αντίστροφα. Σε ένα ΛΣ, ο αλγόριθμος χρονοδρομολόγησης μπορεί να ρυθμιστεί από το διαχειριστή του συστήματος, ανάλογα με τη φύση των εργασιών στις οποίες θέλει αυτός να δώσει προτεραιότητα στο σύστημα. Οι ρυθμίσεις που μπορούν να γίνουν είναι λιγότερες ή περισσότερες ανάλογα με τον ειδικό αλγόριθμο που κάθε φορά χρησιμοποιείται.

Στη μελέτη της απόδοσης των αλγορίθμων χρονοδρομολόγησης χρησιμοποιούνται συνήθως οι ακόλουθοι ποσοτικοί δείκτες:

- **Ο βαθμός χρησιμοποίησης της ΚΜΕ** *(CPU utilization)*. Πρόκειται για το ποσοστό του χρόνου που η ΚΜΕ είναι απασχολημένη εκτελώντας διεργασίες.Βαθμός χρησιμοποίησης κοντά στο 1 δηλώνει οτι η ΚΜΕ είναι σχεδόν συνέχεια απασχολημένη και ο αλγόριθμος χρονοδρομολόγησης δε σπαταλά χρόνο ΚΜΕ σε αναμονή για εκτέλεση διεργασιών.
- **Η ρυθμαπόδοση** *(throughput)*. Είναι ο αριθμός των διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου. Αν π.χ. 3600 διεργασίες ολοκληρώνονται σε σε 60 λεπτά, η ρυθμαπόδοση του συστήματος είναι 1 διεργασία/sec. Όσο μεγαλύτερη είναι η ρυθμαπόδοση τόσο πιο αποδοτικό είναι το σύστημα.
- **Ο μέσος χρόνος αναμονής** *(mean waiting time).* Ο χρόνος αναμονής είναι ο χρόνος που περιμένει μια διεργασία στην λίστα έτοιμων διεργασιών, αναμένοντας την εκτέλεσή της. Ο μέσος χρόνος αναμονής είναι ο μέσος όρος των χρόνων αναμονής των διεργασιών στο σύστημα. Είναι ο καλύτερος δείκτης «καθυστέρησης» των διεργασιών σε ένα σύστημα με πολλές διεργασίες.
- **Ο μέσος χρόνος απόκρισης** *(mean response time).* Ο χρόνος απόκρισης ορίζεται ως το σύνολο του χρόνου αναμονής της διεργασίας και του χρόνου εκτέλεσής της. Στο χρόνο εκτέλεσης υπολογίζονται μόνο οι χρόνοι των εκρήξεων ΚΜΕ της διεργασίας (δεν μετέχουν οι χρόνοι εκρήξεων Ε/Ε αφού αυτοί είναι πολύ πιο μεγάλοι και διαφοροποιούνται σημαντικά ανάλογα με τα περιφερειακά). Ο μέσος χρόνος απόκρισης είναι ο μέσος όρος των χρόνων απόκρισης των διεργασιών στο σύστημα.

Ανεξάρτητα από τη λογική που υλοποιεί ένας **αλγόριθμος χρονοδρομολόγησης** πρέπει αυτός να λαμβάνει υπόψη του ότι:

- Για το υποσύστημα χρονοδρομολόγησης, κάθε διεργασία είναι απρόβλεπτη.
- Υπάρχουν διεργασίες που σπαταλούν σημαντικό χρόνο της ΚΜΕ αναμένοντας είσοδο/έξοδο κι άλλες που μπορούν για σημαντικό χρόνο να καταναλώνουν το 100% της επεξεργαστικής ισχύος του συστήματος, αν τους δοθεί η ευκαιρία. Ο αλγόριθμος δεν μπορεί να γνωρίζει ποτέ με ακρίβεια πότε μια διεργασία θα ανασταλεί περιμένοντας είσοδο/έξοδο, ούτε πότε θα απενεργοποιηθεί σε κάποιο σηματοφορέα ούτε τέλος, πότε θα τερματίσει.

Για την αντιμετώπιση του φαινομένου της μονοπώλησης των ΚΜΕ, τα περισσότερα ΛΣ χρησιμοποιούν τη διακοπή ρολογιού (timer interrupt) που προκαλεί σε τακτά χρονικά διαστήματα το κύκλωμα χρονισμού του υπολογιστή. Κάθε φορά που εμφανίζεται η διακοπή αυτή, εκτελείται ο πυρήνας του ΛΣ και αποφασίζει, αν η εκτελούμενη στη ΚΜΕ διεργασία θα συνεχιστεί ή αν έχει καταναλώσει αρκετό χρόνο επεξεργασίας, οπότε η διεργασία θα διακοπεί έτσι ώστε να εκτελεστεί άλλη διεργασία.

Η στρατηγική του ελέγχου και της πιθανής αναστολής εκτέλεσης εκτελέσιμων διεργασιών ονομάζεται **χρονοδρομολόγηση προεκχώρησης** *(preemptive scheduling)* και έρχεται σε αντίθεση με τη στρατηγική εκτέλεσης μέχρι τέλους (run to completion) ή αλλιώς **μη προεκχωρητική χρονοδρομολόγηση** *(non-preemptive scheduling)*.

Στη μη προεκχωρητική χρονοδρομολόγηση μια διεργασία διακόπτεται μόνο αν η ίδια το επιτρέψει (για είσοδο/έξοδο ή έλεγχο σηματοφορέα). Διαφορετικά, η διεργασία απασχολεί κατ' αποκλειστικότητα την ΚΜΕ μέχρι να περατωθεί.

Είναι προφανές ότι στα σύγχρονα συστήματα υπολογιστή γενικού σκοπού κατάλληλοι θεωρούνται οι προεκχωρητικοί αλγόριθμοι και μόνο σε αφοσιωμένα συστήματα υπολογιστή ειδικού σκοπού προτιμούνται οι αλγόριθμοι εκτέλεσης μέχρι τέλους.

Χρησιμοποιουν προεκχώρηση τα ΛΣ
- *WINDOWS NT*
- *UNIX*
- *LINUX*

Δεν χρησιμοποιούν προεκχώρηση τα ΛΣ
- *WINDOWS 95*
- *WINDOWS 98*

Οι όροι pre emptive, non preemptive αποδίδονται και ως "διακοπτοί" και "μη διακοπτοί"

Σχήμα 8.15 *Χρονοδρομολόγηση προεκχώρησης. Το ΛΣ σε τακτά χρονικά διαστήματα αποφασίζει αν ο έλεγχος της ΚΜΕ πρέπει να μείνει στην εκτελούμενη διεργασία ή να περάσει σε άλλη έτοιμη.*

Στη συνέχεια του κεφαλαίου θα μελετήσουμε τους βασικότερους προεκχωρητικούς και μη προεκχωρητικούς αλγόριθμους χρονοδρομολόγησης. Οι αλγόριθμοι αυτοί μπορούν να εφαρμοστούν σε όλα τα επίπεδα χρονοδρομολόγησης. Για την κατανόηση των αλγορίμων αλλά και των διαφορών τους, θα μελετήσουμε ένα σύστημα στο οποίο εισέρχονται στη λίστα έτοιμων διεργασιών οι διεργασίες που φαίνονται στον πίνακα 8.3.

Πίνακας 8.3

Διεργασία	Άφιξη (χρονική στιγμή)	Διάρκεια (μονάδες χρόνου)
Α	0	11
Β	5	25
Γ	8	2
Δ	12	10

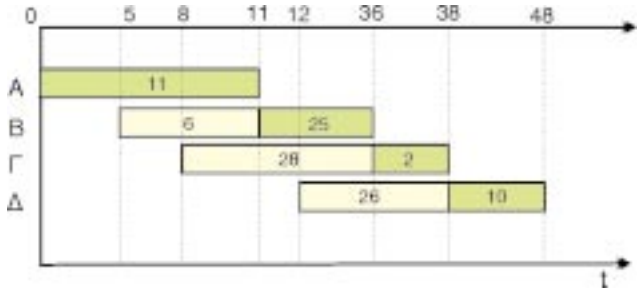
Σε κάθε περίπτωση θα υπολογίσουμε το μέσο χρόνο αναμονής και το μέσο χρόνο απόκρισης του συστήματος.

8.8.1 Μη προεκχωρητική εξυπηρέτηση με βάση τη σειρά άφιξης (Non Preemptive First Come First Serve FCFS)

Είναι ο απλούστερος αλγόριθμος χρονοδρομολόγησης και αυτό είναι ουσιαστικά το μόνο πλεονέκτημά του. Με αυτόν, οι διεργασίες εξυπηρετούνται μέχρι την ολοκλήρωσή τους με τη σειρά που εισήλθαν στη λίστα έτοιμων διεργασιών.

Κάθε φορά που μια διεργασία εισέρχεται στο σύστημα, τοποθετείται στο τέλος της ουράς έτοιμων διεργασιών (όπως συμβαίνει με τους πελάτες μιας τράπεζας). Κάθε φορά που τελειώνει η επεξεργασία μιας διεργασίας, το σύστημα επιλέγει να επεξεργαστεί αυτή που βρίσκεται στην αρχή της ουράς.

Στο σχήμα 8.16 φαίνεται η εκτέλεση των διεργασιών του παραδείγματος (με κίτρινο σημειώνεται ο χρόνος αναμονής και με πράσινο ο χρόνος εκτέλεσης).



Σχήμα 8.16 *Εκτέλεση διεργασιών*

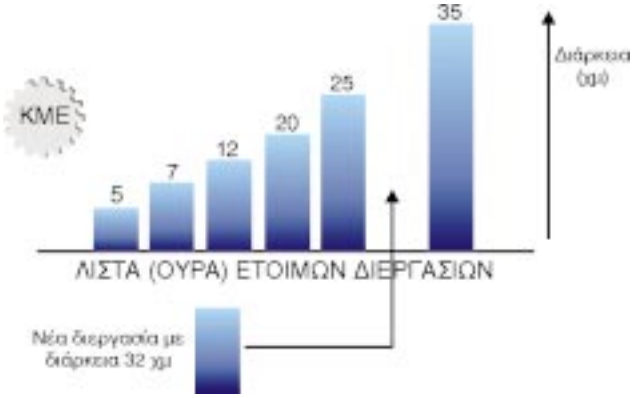
Μέσος χρόνος αναμονής = (0 + 6 + 28 + 26) / 4 = 15 χμ (χρονικές μονάδες)

Μέσος χρόνος απόκρισης = [(0+11) + (6+25) + (28+2) + (26+10)] / 4 = 27 χμ

Γενικά, η απόδοση του αλγορίθμου είναι πολύ χαμηλή σε σχέση με τους μέσους χρόνους αναμονής και απόκρισης αλλά και με όλους τους υπόλοιπους ποσοτικούς δείκτες εκτίμησης απόδοσης αλγορίθμων χρονοδρομολόγησης. Παρατηρείστε ότι η διεργασία Γ με διάρκεια μόλις 2 χμ αναγκάστηκε να περιμένει 28 χμ.

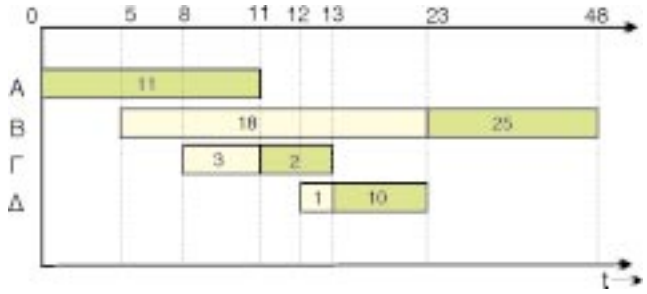
8.8.2 Μη προεκχωρητική εξυπηρέτηση με βάση τη μικρότερη διάρκεια (Non Preemptive Shortest Job First - SJF)

Ο αλγόριθμος εξυπηρέτησης με βάση τη μικρότερη διάρκεια κατατάσσει τις διεργασίες με βάση τη διάρκεια εκτέλεσής τους. Η πιο σύντομη διεργασία τοποθετείται στην αρχή της ουράς έτοιμων διεργασιών και η πιο μεγάλη (χρονικά) στο τέλος της ουράς. Οταν μια διεργασία εισέρχεται στο σύστημα, ο χρονοδρομολογητής την τοποθετεί στην ουρά ανάμεσα στην αμέσως πιο σύντομη και την αμέσως πιο μεγάλη της.



Σχήμα 8.17 *Εισαγωγή διεργασίας με διάρκεια 32 χρονικές μονάδες στη λίστα έτοιμων διεργασιών*

Κάθε φορά που τελειώνει η επεξεργασία μιας διεργασίας, το σύστημα επιλέγει να επεξεργαστεί αυτή που βρίσκεται στην αρχή της ουράς.



Σχήμα 8.18. *Εκτέλεση διεργασιών*

Μέσος χρόνος αναμονής = (0 + 18 + 3 + 1) / 4 = 5,5 χμ

Μέσος χρόνος απόκρισης = [(0+11) + (18+25) + (3+2) + (1+10)] / 4= 17,5 χμ

Παρατηρούμε ότι οι μέσοι χρόνοι αναμονής και απόκρισης είναι πολύ μικρότεροι σε σχέση με τη χρονοδρομολόγηση με βάση τη σειρά άφιξης. Μάλιστα, αποδεικνύεται ότι ο αλγόριθμος δίνει το βέλτιστο μέσο χρόνο αναμονής.

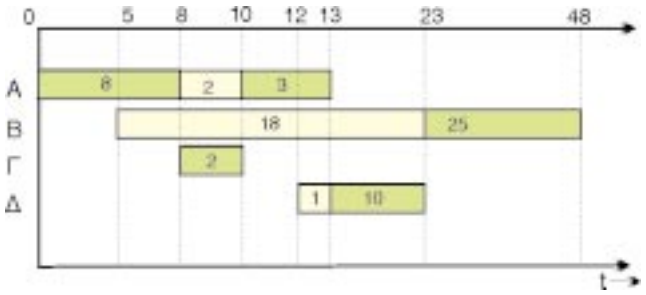
Το μειονέκτημα του αλγορίθμου είναι ότι σε ένα σύστημα είναι πάρα πολύ δύσκολο – έως αδύνατο - να εκτιμηθεί ο χρόνος που απαιτεί για την εκτέλεσή της μια διεργασία. Για το λόγο αυτό, ο αλγόριθμος εφαρμόζεται μόνο σε ειδικά συστήματα και στο επίπεδο της μακροχρόνιας δρομολόγησης. Στα συστήματα αυτά, οι χρήστες, όταν υποβάλουν μια διεργσία στο σύστημα, δηλώνουν τη μέγιστη διάρκεια εκτέλεσής της την οποία χρησιμοποιεί ο μακροχρόνιος δρομολογητής. Αν η διεργασία δεν έχει ολοκληρωθεί στο δηλωμένο χρόνο, συνήθως διακόπτεται. Σε διαλογικά συστήματα ο αλγόριθμος δεν χρησιμοποιείται.

8.8.3 Προεκχωρητική εξυπηρέτηση με βάση τη μικρότερη διάρκεια (Preemptive Shortest Job First - SJF)

Ο αλγόριθμος λειτουργεί όπως ο αντίστοιχος μη προεκχωρητικός με την ακόλουθη διαφορά: κάθε φορά που εισέρχεται στο σύστημα νέα διεργασία, ο χρονοδρομολογητής ελέγχει το χρόνο εκτέλεσης της νέας διεργασίας. Αν αυτός είναι μικρότερος από το χρόνο που απομένει για την ολοκλήρωση της εκτελούμενης διεργασίας, η εκτελλούμενη διεργασία αυτόματα διακόπτεται και τοποθετείται στη λίστα έτοιμων διεργασιών. Η νέα διεργασία γίνεται εκτελούμενη.

Αν ο χρόνος εκτέλεσης της νέας διεργασίας είναι μεγαλύτερος από το χρόνο που απομένει στην εκτελούμενη διεργασία να ολοκληρωθεί, τότε η νέα διεργασία τοποθετείται στην λίστα έτοιμων διεργασιών.

Κάθε φορά που τελειώνει η επεξεργασία μιας διεργασίας, το σύστημα επιλέγει να επεξεργαστεί αυτή που βρίσκεται στην αρχή της ουράς.



Σχήμα 8.19 *Εκτέλεση διεργασιών*

Τη στιγμή 5 μπαίνει στο σύστημα η διεργασία Β. Ο χρόνος εκτέλεσής της είναι 25χμ, μεγαλύτερος από το υπόλοιπο της Α που εκτελείται στην ΚΜΕ (ο χρόνος της Α που απομένει είναι 11-5=6). Συνεπώς, η Α συνεχίζει την εκτέλεσή της.

Τη στιγμή 8 εισέρχεται στο σύστημα η διεργασία Γ με χρόνο εκτέλεσης 2 χρονικές μονάδες. Η Α που εκτελείται στην ΚΜΕ έχει υπόλοιπο 11-8=3 χμ. Συνεπώς, ο χρονοδρομολογητής προωθεί τη Γ στην ΚΜΕ και τοποθετεί την Α στη λίστα διεργασιών. Επειδή η Α έχει υπόλοιπο χρόνου εκτέλεσης 3 χμ, που είναι μικρότερος της Β που είναι ήδη στη λίστα, η Α τοποθετείται στη λίστα μπροστά από τη Β. Ο αλγόριθμος λειτουργεί με παρόμοιο τρόπο και στη συνέχεια.

Μέσος χρόνος αναμονής = (2 + 18 + 0 + 1) / 4 = 5,25 χμ

Μέσος χρόνος απόκρισης = [(2+11) + (18+25) + (0+2) + (1+10)] = 17,25 χμ

Παρατηρούμε ότι οι μέσοι χρόνοι αναμονής και απόκρισης είναι καλύτεροι από όλους του αντίστοιχους χρόνους των μη διακοπτών αλγορίθμων. Ο αλγόριθμος εμφανίζεται ελάχιστα καλύτερος από τον μη-προεκχωρητικό SJF, κάτι που οφείλεται στο συγκεκριμένο παράδειγμα. Σε άλλο παράδειγμα η διαφορά θα μπορούσε να είναι πιο σημαντική.

Σημειώστε ότι στον προεκχωρητικό αλγόριθμο υπάρχει και η επιβάρυνση επεξεργασίας του ίδιου του αλγορίθμου από την ΚΜΕ, που εκτελείται συχνότερα από τον μη-προεκχωρητικό αλγόριθμο (στη μελέτη μας δεν την λαμβάνουμε υπόψη μας). Η επιβάρυνση αυτή πρέπει πάντα να είναι μικρή, ώστε να μην επηρεάζει σημαντικά την επίδοση του αλγορίθμου.

8.8.4 Προεκχωρητική εξυπηρέτηση εκ περιτροπής (Preemptive Round Robin)

Ο αλγόριθμος εξυπηρέτησης εκ περιτροπής είναι ένας από τους παλαιότερους, πιο διαδεδομένους και δικαιότερους αλγόριθμους χρονοδρομολόγησης. Έχει σχεδιαστεί ειδικά για συστήματα καταμερισμού χρόνου, αφού δίνει ένα μικρό χρόνο επεξεργασίας στις έτοιμες διεργασίες, οι οποίες εκτελούνται εκ περιτροπής.

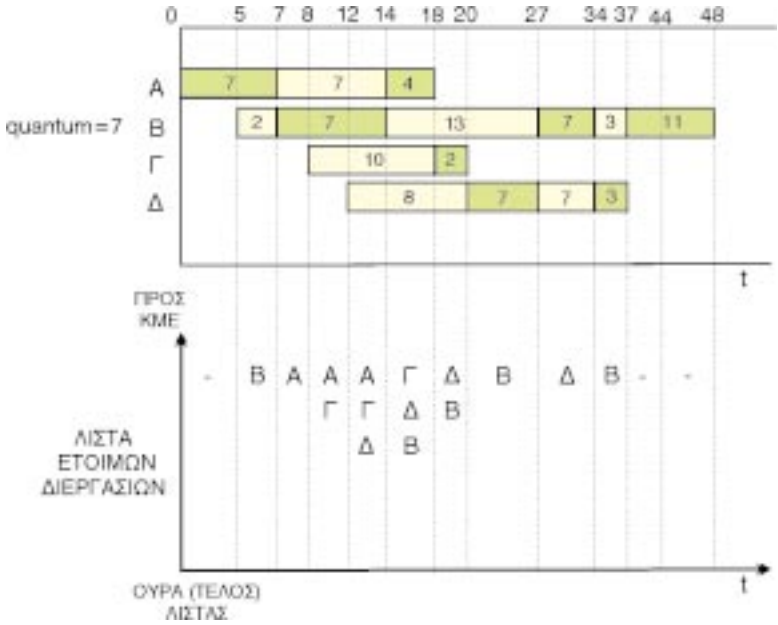
Η λογική του είναι ιδιαίτερα απλή:

- Σε κάθε διεργασία παραχωρείται ορισμένο και πάντα ίσο χρονικό διάστημα - το οποίο ονομάζεται **κβάντο χρόνου** *(time quantum)* - επεξεργασίας, μέσα στο οποίο επιτρέπεται η εκτέλεσή της.
- Με το τέλος του διαστήματος η διεργασία αναστέλεται και τοποθετείται στη λίστα έτοιμων διεργασιών, στο τέλος της ουράς. Η ΚΜΕ δίνεται για το επόμενο

κβάντο χρόνου στην επόμενη έτοιμη διεργασία (που βρίσκεται στην αρχή της λίστας/ουράς)

- Αν μια διεργασία ανασταλεί για οποιοδήποτε λόγο ή ολοκληρωθεί πριν τη λήξη του κβάντο χρόνου της, τότε το σύστημα παρεμβαίνει και δίνει τον έλεγχο σε νέα διεργασία.
- Κάθε φορά που εισέρχεται στο σύστημα μια νέα διεργασία, αυτή τοποθετείται στο τέλος της λίστας έτοιμων διεργασιών.

Η χρονοδρομολόγηση για το παράδειγμά μας φαίνεται στο σχήμα που ακολουθεί. Στο σχήμα φαίνεται για κάθε χρονικό διάστημα και η λίστα των έτοιμων διεργασιών που αναμένουν επεξεργασία.



Σχήμα 8.20 *Εκτέλεση διεργασιών και λίστα έτοιμων διεργασιών*

Παρατηρείστε ότι τη στιγμή 14 ξεκινά πάλι η εκτέλεση της διαδικασίας Α, η οποία τελειώνει πριν το κβάντο χρόνου της. Το ίδιο συμβαίνει και για τη διεργασία Γ που ολοκληρώνεται τη στιγμή 20. Από τη στιγμή 20 και μετά δίνεται ολόκληρο κβάντο στη διεργασία Δ.

Μέσος χρόνος αναμονής = (7 + 18 + 10 + 15) / 4 = 12,5 χμ

Μέσος χρόνος απόκρισης = [(7+11) + (18+25) + (10+2) + (15+10)] = 24,5 χμ

Η υλοποίηση του αλγορίθμου είναι ιδιαίτερα απλή και η μόνη παράμετρος που επηρεάζει την απόδοση του συστήματος είναι το μέγεθος του κβάντου χρόνου.

Αν το κβάντο είναι πολύ μεγάλο, τότε το σύστημα εμφανίζει μεγάλους χρόνους απόκρισης ακόμη και σε απλά διαλογικά αιτήματα (εκφυλίζεται σε μη προεκχωρητικό FCFS). Αν το κβάντο χρόνου είναι πολύ μικρό εμφανίζεται άλλο πρόβλημα, που οφείλεται στο χρόνο που απαιτείται για την αλλαγή της εκτελούμενης διεργασίας. Κάθε φορά που αλλάζει η εκτελούμενη διεργασία, ο χρόνος διαχείρισης χάνεται για το σύστημα. Πολλές εναλλαγές συνεπώς (μικρό κβάντο χρόνου) προκαλούν σπατάλη χρόνου επεξεργασίας.

Αν για παράδειγμα έχουμε χρόνο διαχείρισης 5msec στην εναλλαγή και κβάντο χρόνου 45msec προκύπτει ότι για κάθε 50msec έχουμε επεξεργασία 45msec (90%) και διαχείριση 5sec (σπατάλη 10%).

Αν όμως έχουμε χρόνο διαχείρισης 5msec στην εναλλαγή και κβάντο χρόνου 495msec προκύπτει ότι για κάθε 500msec έχουμε επεξεργασία 495msec (99%) και διαχείριση 5sec (σπατάλη 1%).

Η βέλτιστη λύση για σύστημα με δεδομένο χρόνο αλλαγής εκτελούμενης διεργασίας είναι μια τιμή για το κβάντο χρόνου τέτοια, ώστε να έχουμε χαμηλή αναμονή σε διαλογικές ενέργειες (κβάντο όχι πολύ μεγάλο) και ταυτόχρονα ελάχιστη σπατάλη χρόνου επεξεργασίας (κβάντο πολύ μεγαλύτερο από το χρόνο εναλλαγής διεργασιών). Ενα καλό μέγεθος είναι εκείνο που είναι μεγαλύτερο σε διάρκεια από το 80% των εκρήξεων ΚΜΕ στο σύστημα.

Το βασικότερο πλεονέκτημα του αλγορίθμου είναι οτι ευνοεί την ολοκλήρωση μικρών διεργασιών, οι οποίες δεν αναγκάζονται να έχουν μεγάλους χρόνους αναμονής.

Οι αλγόριθμοι που εξετάσαμε αποτελούν υποπεριπτώσεις ενός γενικού αλγορίθμου χρονοδρομολόγησης: της **Εξυπηρέτησης Προτεραιοτήτων** *(priority scheduling)*. Ο αλγόριθμος αυτός που συναντάται και σε προεκχωρητική και σε μη προεκχωρητική μορφή απόδίδει σε κάθε διεργασία που εισέρχεται στο σύστημα μια προτεραιότητα. Οι διεργασίες στη λίστα έτοιμων διεργασιών βρίσκονται ταξινομημένες με βάση την προτεραιότητα που έχουν. Η διεργασία με τη μεγαλύτερη προτεραιότητα βρίσκεται πάντα στην αρχή της λίστας και είναι η πρώτη που θα εξυπηρετηθεί από την ΚΜΕ. Οι προτεραιότητες μπορούν να είναι είτε στατικές είτε δυναμικές. Σε περίπτωση που ο αλγόριθμος προτεραιοτήτων αποδίδει στατικές προτεραιότητες, κάθε διεργασία παίρνει μια προτεραιότητα, καθώς εισέρχεται στο σύστημα. Οταν γίνεται χρήση δυναμικών προτεραιοτήτων, τότε ο χρονοδρομολογητής μπορεί με κάποια λογική να αλλάζει τις προτεραιότητες των διεργασιών καθώς αυτές βρίσκονται στο σύστημα (π.χ. μια διεργασία μπορεί να παίρνει αυξημένη προτεραιότητα, αν ήδη περιμένει πολλή ώρα να εξυπηρετηθεί.)

Ανάλογα με τα κριτήρια που χρησιμοποιούνται για την απόδοση προτεραιοτήτων αλλάζει σημαντικά και η λειτουργία αλλά και η απόδοση του αλγορίθμου. Οι αλγόριθμοι που εξετάσαμε προκύπτουν από τον αλγόριθμο προτεραιοτήτων ως εξής:

Ο **FCFS**, αν όλες οι διεργασίες παίρνουν την ίδια προτεραιότητα, οπότε πάντα εκτελείται πρώτη αυτή που πρώτη μπαίνει στη λίστα αναμονής.

Ο **SJF**, αν η προτεραιότητα μιας διεργασίας είναι αντιστρόφως ανάλογη του χρόνου ολοκλήρωσής της.

Στον **Round Robin** κάθε φορά που μια διεργσία εισέρχεται στο σύστημα παίρνει τη χαμηλότερη προτεραιότητα. Επίσης, όταν μια διεργασία απομακρύνεται από την ΚΜΕ και γυρνά στην λίστα αναμονής, τότε εισέρχεται στη λίστα έχοντας τη μικρότερη προτεραιότητα.

Στα περισσότερα συστήματα σήμερα, ο αλγόριθμος προτεραιοτήτων εμφανίζεται με πολύ πιο σύνθετες μορφές, που πολλές φορές αποτελούν επεκτάσεις όσων μελετήσαμε σε αυτό το κεφάλαιο.

Ανακεφαλαίωση

Για την ανάλυση, τη μελέτη και την ευκολότερη εσωτερική διαχείριση ενός συστήματος υπολογιστή δημιουργήσαμε το μοντέλο διεργασιών, το οποίο αποτελείται από σειριακές διεργασίες που εκτελούνται παράλληλα. Κάθε διεργασία έχει τη δική της κατάσταση και εναλλάσσεται με τις άλλες στην ΚΜΕ. Η ταχύτατη εναλλαγή διεργασιών στην ΚΜΕ ενός συστήματος καταμερισμού χρόνου δημιουργεί μακροσκοπικά την ψευδαίσθηση του παραλληλισμού.

Σε περιπτώσεις όπου παράλληλες διεργασίες προσπελαύνουν διαμοιραζόμενους σε αυτές πόρους του συστήματος, όπως περιοχές κοινόχρηστης μνήμης, υπάρχει η ανάγκη επικοινωνίας μεταξύ τους. Η αλληλεπίδραση των διεργασιών, όταν ασχολούνται με κοινόχρηστους πόρους, αν δεν είναι σωστά δομημένη, προκαλεί συνθήκες ανταγωνισμού και συχνά οδηγεί το σύστημα σε αδιέξοδα και πιθανό ακόμη και σε κατάρρευση.

Για να αποφύγουμε την εμφάνιση συνθηκών ανταγωνισμού, εισάγουμε την έννοια του κρίσιμου τμήματος διεργασίας. Κρίσιμο τμήμα είναι κάθε τμήμα του προγράμματος της διεργασίας στο οποίο η διεργασία διαχειρίζεται κοινόχρηστους πόρους του συστήματος. Η ορθή λειτουργία του συστήματος απαιτεί μία μόνο διεργασία να βρίσκεται σε κρίσιμο τμήμα της κάθε φορά, κάτι που ονομάζουμε αμοιβαίο αποκλεισμό.

Ο αμοιβαίος αποκλεισμός είναι μια μόνο προϋπόθεση για την ορθή λειτουργία ενός συστήματος. Η συνολική προϋπόθεση είναι να υπάρχει ένα ευρύτερο και καλά δομημένο πλαίσιο από αρχές επικοινωνίας, αλληλεπίδρασης, συνεργασίας και συγχρονισμού διεργασιών στο σύστημα. Με τις αρχές αυτές, οι διεργασίες αλληλεπιδρούν και αλλάζουν καταστάσεις με τέτοιο τρόπο, ώστε να πετυχαίνουν τον αμοιβαίο αποκλεισμό, να αποφεύγουν τα αδιέξοδα και να συγχρονίζονται αρμονικά, οδηγώντας σε ορθή και αποδοτική χρήση του συστήματος.

Πολλές αρχές έχουν προταθεί για τη διαδιεργασιακή επικοινωνία, όπως οι σηματοφορείς , που είναι και μια από τις πιο διαδεδομένες. Θεωρητικά οι αρχές αυτές είναι ισοδύναμες.

Ένα ζήτημα με σοβαρές επιπτώσεις στην ευελιξία και στην απόδοση κάθε συστήματος είναι ο αλγόριθμος με τον οποίο εναλλάσονται οι διεργασίες στην ΚΜΕ, δηλαδή, ο αλγόριθμος χρονοδρομολόγησης. Στα συστήματα υπολογιστών χρησιμοποιούνται, ανάλογα με τις ειδικές απαιτήσεις του καθενός, διάφοροι αλγόριθμοι, όπως αυτοί της εξυπηρέτησης εκ περιτροπής, της εξυπηρέτησης κατά προτεραιότητα και της εξυπηρέτησης με βάση τη διάρκεια. Οι μηχανισμοί αυτοί, που δεν είναι και οι μόνοι, περιλαμβάνουν παραμετροποιήσιμα μεγέθη με τα οποία μπορεί ο διαχειριστής του συστήματος στατικά ή δυναμικά να προσαρμόσει την απόδοση του συστήματος σε ειδικές ανάγκες.

Ερωτήσεις/Δραστηριότητες/Θέματα για συζήτηση

1. Τι είναι η διεργασία; Τι διαφέρει από ένα πρόγραμμα;
2. Τι είναι οι ελαφριές διεργασίες (νήματα). Τι διαφέρουν από τις διεργασίες;
3. Τι είναι ο γράφος προβαδίσματος; Δώστε παράδειγμα.

4. Πώς απεικονίζουμε ταυτόχρονα προγράμματα; Εξηγείστε με παράδειγμα χρησιμοποιώντας τις διαφορετικές μεθόδους απεικόνισης.
5. Περιγράψτε τις καταστάσεις και τον κύκλο ζωής μιας διεργασίας. Δώστε τις δυνατές μεταβάσεις καταστάσεων μιας διεργασίας.
6. Τι ονομάζουμε εκρήξεις ΚΜΕ και τί εκρήξεις Ε/Ε;
7. Τι είναι η μεταγωγή περιβάλλοντος;
8. Τι ονομάζουμε ταυτόχρονες ή παράλληλες διεργασίες σε ένα σύστημα πολυπρογραμματισμού; Είναι πραγματικά παράλληλες;
9. Τι είναι οι συνθήκες ανταγωνισμού;
10. Τι είναι ο αμοιβαίος αποκλεισμός και τί τα κρίσιμα τμήματα μιας διεργασίας; Πόσα κρίσιμα τμήματα μπορεί να έχει μια διεργασία;
11. Τι ονομάζουμε αδιέξοδο;
12. Εξηγείστε γιατί η απενεργοποίηση διακοπών είναι αποδεκτή για κλήσεις συστήματος ενώ είναι μη αποδεκτή για διεργασίες των χρηστών.
13. Τι είναι η ενεργός αναμονή; Σε τι διαφέρει η ενεργός αναμονή από την αναστολή μιας διεργασίας;
14. Εξηγείστε με ένα σχήμα πώς μια διεργασία χρησιμοποιεί ενα σηματοφορέα για να εκτελέσει τα κρίσιμα τμήματά της (που σχετίζονται με το σηματοφορέα αυτό).
15. Ποια είναι τα δύο κύρια χαρακτηριστικά των σηματοφορέων; Εξηγείστε με παράδειγμα γιατί χωρίς τα χαρακτηριστικά αυτά οι σηματοφορείς δε λύνουν τα προβλήματα αμοιβαίου αποκλεισμού και συγχρονισμού διεργασιών.
16. Τι είναι οι λίστες αναμονής διεργασιών του ΛΣ;
17. Τι είναι η χρονοδρομολόγηση, τι ο χρονοδρομολογητής και τι ο αλγόριθμος χρονοδρομολόγησης;
18. Εξηγείστε τα επίπεδα χρονοδρομολόγησης και τους βασικούς δείκτες αξιολόγησης της επίδοσης των αλγορίθμων χρονοδρομολόγησης.
19. Τι ονομάζουμε προεκχωρητική και τι μη-προεκχωρητική χρονοδρομολόγηση;
20. Ποιος είναι ο βασικός λόγος που ο αλγόριθμος εξυπηρέτησης με βάση τη μικρότερη διάρκεια διεργασίας δε χρησιμοποιείται σε συστήματα καταμερισμού χρόνου;
21. Τι διαφέρει η προεκχωρητική από την μη προεκχωρητική εξυπηρέτηση με βάση τη μικρότερη διάρκεια διεργασίας;
22. Τι είναι το κβάντο χρόνου και πώς επηρεάζει την απόδοση του αλγορίθμου εξυπηρέτησης εκ περιτροπής; Δώστε παράδειγμα.

Βιβλιογραφία

1. **Ι. Κάβουρα,** Συστηματα Υπολογιστών ΙΙ Λειτουργικά Συστηματα, 3η εκδοση, Εκδόσεις Κλειδάριθμος, 1995
2. **A. Tanenbaum,** Σύγχρονα Λειτουργικά Συστήματα, Εκδόσεις Παπασωτηρίου, 1993
3. **Silberschatz et al.,** Operating System Concepts, 3rd Ed. Addison-Wesley, 1991

Διευθύνσεις διαδικτύου

http://pandonia.canberra.edu.au/ssw/intro/generic.html
"Generic Operating System Structure", Δομή ενός τυπικού ΛΣ.
http://pandonia.canberra.edu.au/ssw/processes/lecture.html
"System Software Overview", Παρουσίαση ΛΣ από την άποψη των διεργασιών.
http://cs.millersv.edu/cs382.dir/20.html
"The notion of a process", Εξήγηση της έννοιας της διεργασίας.
http://www-dsg.stanford.edu/papers/cachekernel/subsection3_3_2.html
"Interprocess Communication", Παρουσίαση διαδιεργασιακής επικοινωνίας.
http://www.cs.adfa.oz.au/teaching/studinfo/csa2/OSNotes/node12.html
"Interprocess Communication and synchronization", Μηχανισμοί επικοινωνίας και συγχρονισμών διεργασιών.
http://www.cs.adfa.oz.au/teaching/studinfo/csa2/OSNotes/node4.html
"Process Scheduling", Παρουσίαση Μηχανισμών χρονοδρομολόγισης διεργασιών.

Λέξεις κλειδιά

Αδιέξοδο ..... Deadlock
Αμοιβαίος αποκλεισμός ..... Mutual blocking
Διαδιεργασιακή επικοινωνία ..... Interprocess communication
Ενεργός αναμονή ..... Busy waiting
Κρίσιμο τμήμα διεργασίας ..... Process critical section
Σηματοφορέας ..... Semaphore
Συνθήκες ανταγωνισμού ..... Race conditions
Χρονοδρομολόγηση ..... Scheduling