Κεφάλαιο

Λειτουργικά Συστήματα - Διεργασίες

Σκοπός του κεφαλαίου αυτού είναι να παρουσιάσει τα Λειτουργικά Συστήματα, και να περιγράψει τις πιο βασικές οντότητες του Λειτουργικού Συστήματος, τις διεργασίες.

Όταν ολοκληρώσεις το κεφάλαιο αυτό, θα μπορείς:

- Να εξηγείς τι είναι Λειτουργικό Σύστημα και ποιες είναι οι αρμοδιότητές του.
- Να περιγράφεις το ρόλο της διεργασίας σε ένα Λειτουργικό Σύστημα.
- Να ορίζεις το πρόβλημα του κρίσιμου τμήματος και να προτείνεις τρόπους για τη λύση του.

Μαθήματα

Μάθημα 7.1 Λειτουργικά Συστήματα

Σκοπός του μαθήματος

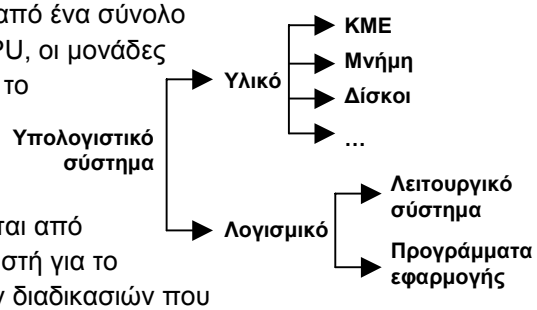
Σκοπός του μαθήματος αυτού είναι να ορίσει τα Λειτουργικά Συστήματα, να παρουσιάσει την ιστορία τους και να περιγράψει τη βασική τους δομή.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς τι είναι Λειτουργικό Σύστημα*
- *Να απαριθμείς τις διάφορες κατηγορίες Λειτουργικών Συστημάτων που έχουν εμφανιστεί*
- *Να περιγράφεις τη βασική δομή ενός Λειτουργικού Συστήματος*

Όπως γνωρίζουμε, κάθε υπολογιστικό σύστημα αποτελείται (α) από ένα σύνολο συσκευών, (όπως η Κεντρική Μονάδα Επεξεργασίας - ΚΜΕ / CPU, οι μονάδες αποθήκευσης όπως μαγνητικοί και οπτικοί δίσκοι, οι εκτυπωτές, το πληκτρολόγιο κλπ), οι οποίες ονομάζονται *υλικό* του υπολογιστή (hardware), και (β) από το *λογισμικό* (software) το οποίο αποτελείται από το *Λειτουργικό Σύστημα* και τα *προγράμματα εφαρμογής*. Τα προγράμματα εφαρμογής γράφονται από τους χρήστες-προγραμματιστές και δίνουν εντολές στον υπολογιστή για το πώς θα χρησιμοποιήσει τις συσκευές για την εκτέλεση διαφόρων διαδικασιών που συνδέονται με το υπολογιστικό σύστημα.



Το Λειτουργικό Σύστημα (ΛΣ) είναι ένα σύνολο προγραμμάτων που λειτουργεί ως σύνδεσμος ανάμεσα στα προγράμματα του χρήστη και το υλικό και καθορίζει τον τρόπο λειτουργίας του υπολογιστικού συστήματος, ελέγχοντας και συντονίζοντας τη χρήση των μονάδων του από τα διάφορα προγράμματα εφαρμογής των χρηστών.

Αν φανταστούμε τον υπολογιστή ως ένα ταξί με τους χρήστες και τα προγράμματά τους να αποτελούν τους επιβάτες, τότε το ΛΣ παίζει το ρόλο του οδηγού, χωρίς τη συνεχή παρουσία του οποίου το ταξί είναι άχρηστο

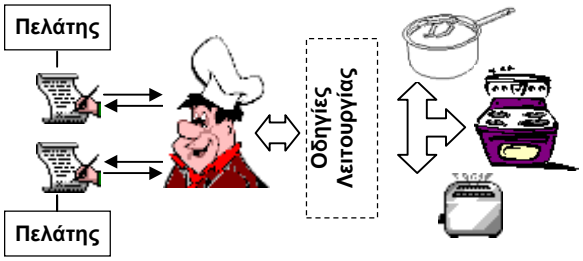
Οι στόχοι ενός ΛΣ είναι:

1. **Η διευκόλυνση των χρηστών**. Τα ΛΣ υπάρχουν επειδή κάνουν πιο εύκολη τη χρήση των υπολογιστικών συστημάτων και δίνουν τη δυνατότητα σε ανθρώπους με μικρές γνώσεις γύρω από τους υπολογιστές να εκτελέσουν πολύπλοκες εργασίες.

2. **Η διευκόλυνση των προγραμματιστών**. Χωρίς ΛΣ κάθε πρόγραμμα έπρεπε π.χ. να ελέγχει τακτικά το πληκτρολόγιο για είσοδο από το χρήστη, να γνωρίζει τις ακριβείς εντολές που πρέπει να στείλει στον εκτυπωτή για να τυπώσει κάτι ή να οργανώνει μόνο του το χώρο αποθήκευσης των δεδομένων του σε ένα σκληρό δίσκο.

3. **Η αποδοτική λειτουργία του υπολογιστικού συστήματος**, δηλαδή η όσο το δυνατόν καλύτερη χρησιμοποίηση του υλικού, ώστε να κατανέμεται καλύτερα το υπολογιστικό φορτίο. Το ΛΣ διαθέτει τη «γενική εικόνα» όλων των προγραμμάτων που πρέπει να εκτελεστούν, όλων των χρηστών του υπολογιστικού συστήματος και των αναγκών τους· έτσι, μπορεί να ρυθμίσει καλύτερα πότε και ποια προγράμματα θα εκτελεστούν κλπ.

Ο ρόλος του ΛΣ μπορεί να περιγραφεί με μια παρομοίωση από την καθημερινή ζωή, η οποία θα χρησιμοποιείται στη συνέχεια, όπου χρειάζεται, για τη διευκρίνιση διαφόρων εννοιών. Ας παρομοιώσουμε λοιπόν ένα υπολογιστικό σύστημα με ένα ζαχαροπλαστείο το οποίο παρασκευάζει γλυκά κατά παραγγελία και σύμφωνα με τις οδηγίες που δίνουν οι πελάτες. Ο υπάλληλος του ζαχαροπλαστείου παίζει το ρόλο της Κεντρικής Μονάδας Επεξεργασίας (ΚΜΕ, CPU), και τα σκεύη του είναι οι υπόλοιπες συσκευές του υπολογιστικού συστήματος. Οι οδηγίες για την παρασκευή των γλυκών είναι τα προγράμματα και οι πελάτες είναι οι χρήστες του υπολογιστικού συστήματος. Οι οδηγίες που ακολουθεί ο υπάλληλος του ζαχαροπλαστείου για την εξυπηρέτηση των πελατών αντιστοιχούν στο ΛΣ του υπολογιστικού συστήματος.



Λειτουργικά Συστήματα Ομαδικής Επεξεργασίας

Ένας πρώτος τρόπος λειτουργίας του ζαχαροπλαστείου είναι οι πελάτες να αναμένουν σε μια ουρά και να δίνουν με τη σειρά στον ζαχαροπλάστη τις οδηγίες τους για την παρασκευή των γλυκών της προτίμησής τους. Μόνο όταν ολοκληρωθεί η εργασία για ένα πελάτη εξυπηρετείται ο επόμενος. Μια καλύτερη λύση είναι να ομαδοποιούνται πελάτες που θέλουν το ίδιο γλυκό και να εξυπηρετούνται διαδοχικά ώστε να γίνεται καλύτερη χρήση των σκευών.

Αυτός ο τρόπος εξυπηρέτησης αντιστοιχεί στα ΛΣ πρώτης γενιάς της δεκαετίας του ’50, τα *Λειτουργικά Συστήματα Ομαδικής Επεξεργασίας* (batch processing). Στα συστήματα αυτά ο χειριστής του υπολογιστικού συστήματος ομαδοποιούσε τα προγράμματα που υπέβαλλαν οι χρήστες· έτσι το ΛΣ δεχόταν μια ομάδα ομοειδών προγραμμάτων (π.χ. προγραμμάτων COBOL), τα επεξεργαζόταν το ένα μετά το άλλο (με τη σειρά εμφανίσεως) και τύπωνε τα αποτελέσματά τους πάλι με την ίδια σειρά. Τα πρώτα αυτά λειτουργικά συστήματα, παρουσίαζαν τα ακόλουθα βασικά μειονεκτήματα:

♦ **Υποαπασχόληση των συσκευών**: Σε ένα ΛΣ ομαδικής επεξεργασίας δεν ήταν δυνατή η ταυτόχρονη εκτέλεση περισσοτέρων του ενός προγραμμάτων. Έπρεπε πρώτα να τελειώσει ένα πρόγραμμα, για να αρχίσει η εκτέλεση του επομένου προγράμματος της ομάδας. Ένα πρόγραμμα όμως δεν μπορεί να απασχολεί ταυτόχρονα όλες τις μονάδες του Η/Υ, οι οποίες κατά συνέπεια υποαπασχολούνταν· για παράδειγμα, η ΚΜΕ απασχολούνταν συνήθως σε ποσοστό μικρότερο του 10%.

Εάν ήταν δυνατό να εκτελούνται πολλά προγράμματα ταυτόχρονα, θα μπορούσε σε μια δεδομένη χρονική στιγμή κάθε ένα από αυτά να απασχολεί και διαφορετική μονάδα. Έτσι θα περιοριζόταν ο ανενεργός χρόνος κάποιων μονάδων του υπολογιστή.

♦ Το χρονικό διάστημα που μεσολαβεί από τη στιγμή που ο προγραμματιστής θα αφήσει στον υπολογιστή το πρόγραμμά του μέχρι τη στιγμή που θα πάρει τα αποτελέσματα, ονομάζεται *χρόνος ανακύκλωσης* (turnaround time) και στα ΛΣ ομαδικής επεξεργασίας το διάστημα αυτό ήταν πολύ μεγάλο. Συνήθως τα αποτελέσματα των προγραμμάτων δεν ήταν διαθέσιμα, παρά μόνο αφού είχε ολοκληρωθεί η εκτέλεση όλης της ομάδας. Έτσι ο χρόνος ανακύκλωσης εξαρτιόταν από το χρόνο εκτέλεσης ολόκληρης της ομάδας, ο οποίος μπορεί να είναι π.χ. 50 ή 100 φορές μεγαλύτερος από το χρόνο εκτέλεσης του συγκεκριμένου προγράμματος.

Λειτουργικά Συστήματα Πολυπρογραμματισμού

> Η εξυπηρέτηση των πελατών του ζαχαροπλαστείου θα ήταν πιο γρήγορη αν ο υπάλληλος είχε οδηγίες από τον ιδιοκτήτη του ζαχαροπλαστείου να παρασκευάζει περισσότερα από ένα γλυκά ταυτοχρόνως. Κατά τη διάρκεια π.χ. του ψησίματος ενός γλυκού έχει τη δυνατότητα να προετοιμάζει στον αναμίκτη (μίξερ) ένα άλλο, εκμεταλλευόμενος το χρόνο του. Όταν όμως ολοκληρωθεί το ψήσιμο του πρώτου θα τον ειδοποιεί π.χ. ένα κουδουνάκι να διακόψει την προετοιμασία του δευτέρου και να συνεχίσει την παρασκευή του πρώτου γλυκού με το επόμενο στάδιό της.

Αυτή την ιδέα οργάνωσης ακολουθούν τα ΛΣ δεύτερης γενιάς (δεκαετίας ’60), τα *Λειτουργικά Συστήματα Πολυπρογραμματισμού* (multiprogramming), στα οποία επιδιώκεται η μείωση του άεργου χρόνου των μονάδων του υπολογιστή και του χρόνου ανακύκλωσης. Η διακοπή μιας λειτουργίας (το κουδουνάκι του ζαχαροπλαστείου) γίνεται με τη δημιουργία ειδικών σημάτων, των *σημάτων διακοπής* (interrupts), τα οποία διακόπτουν την τρέχουσα λειτουργία της ΚΜΕ.

Πολλές φορές το ΛΣ Πολυπρογραμματισμού ορίζεται ως το ΛΣ που επιτρέπει σε περισσότερα από ένα προγράμματα να είναι «φορτωμένα» στη μνήμη τού υπολογιστή και να εκτελούνται συγχρόνως. Φυσικά μόνο ένα πρόγραμμα μπορεί να απασχολεί ανά πάσα στιγμή κάθε μονάδα· π.χ. ένα πρόγραμμα εκτελείται από την ΚΜΕ, ένα άλλο διαβάζει από το δίσκο, ένα τρίτο στέλνει δεδομένα στη σειριακή θύρα εξόδου κλπ.

> Μια πολύ πρακτική υπηρεσία για τους πελάτες του ζαχαροπλαστείου είναι οι τηλεφωνικές παραγγελίες και η κατ’ οίκον διανομή των γλυκών, οι οποίες αντιστοιχούν στην τηλεπεξεργασία που θα δούμε αμέσως μετά.

Παράλληλα με τα ΛΣ Πολυπρογραμματισμού αναπτύχθηκαν και συσκευές οι οποίες έδιναν τη δυνατότητα σε περιφερειακά όπως τηλέτυπα, οθόνες, εκτυπωτές κλπ. να επικοινωνήσουν με ένα υπολογιστή μέσω τηλεπικοινωνιακών γραμμών. Έτσι οι περιφερειακές μονάδες δεν ήταν πλέον απαραίτητο να βρίσκονται στον ίδιο φυσικό χώρο με τον υπολογιστή. Τα ΛΣ που διαχειρίζονταν τέτοια υπολογιστικά συστήματα με «απομακρυσμένα» περιφερειακά ονομάζονται *Λειτουργικά Συστήματα Τηλεπεξεργασίας*. Παρ’ όλα τα πλεονεκτήματα που παρέχουν τα ΛΣ Πολυπρογραμματισμού, παρουσιάζουν και μειονεκτήματα. Ένα απ’ αυτά είναι π.χ. ότι το ίδιο το ΛΣ είναι πλέον ένα πολύπλοκο πρόγραμμα (σε αντίθεση με τα ΛΣ ομαδικής επεξεργασίας που είναι πολύ απλά), το οποίο με τη σειρά του απασχολεί τον

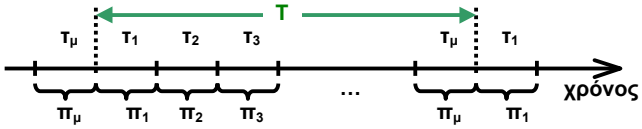
υπολογιστή σε βάρος των προγραμμάτων εφαρμογών, και μάλιστα τόσο πιο πολύ, όσο πιο πολύπλοκο είναι.

Λειτουργικά Συστήματα Καταμερισμού Χρόνου

Τερματικό
Συνδυασμός οθόνης και πληκτρολογίου για κάθε χρήστη ενός μεγάλου υπολογιστή

Πολλά μεγάλα υπολογιστικά συστήματα εξυπηρετούν πολλούς χρήστες, κάθε ένας από τους οποίους έχει στη διάθεσή του ένα τερματικό. Ο χρήστης πληκτρολογεί εντολές στο τερματικό, μέσω του οποίου διαβιβάζονται και εκτελούνται στο υπολογιστικό σύστημα. Για να έχουν όλοι οι χρήστες την εντύπωση ότι εξυπηρετούνται παράλληλα, το υπολογιστικό σύστημα καταμερίζει το χρόνο του, δίνοντας από λίγο και εκ περιτροπής στον καθένα.

> Μια άλλη μέθοδος λειτουργίας του ζαχαροπλαστείου θα ήταν η «ταυτόχρονη» παρασκευή των γλυκών, με καταμερισμό του χρόνου του υπαλλήλου μεταξύ των διαφόρων συνταγών. Κάθε πέντε λεπτά χτυπά ένα ρολόι και ο ζαχαροπλάστης διακόπτει την εργασία του, βάζει στην άκρη το γλυκό που έφτιαχνε και καταπιάνεται με το επόμενο στη σειρά.



Στα υπολογιστικά συστήματα το ρόλο του ρολογιού έχει ένα ειδικό κύκλωμα, ο *χρονιστής* (timer), το οποίο σε τακτά χρονικά διαστήματα δημιουργεί ένα σήμα διακοπής. Ο χρόνος χωρίζεται σε διαστήματα διάρκειας Τ. Αν υπάρχουν μ προγράμματα για εκτέλεση, που συμβολίζονται με $\pi_1,\pi_2,\ldots,\pi_\mu$, η περίοδος Τ διαιρείται σε μ χρονικά διαστήματα $\tau_1,\tau_2,\ldots,\tau_\mu$. Ο χρονιστής δημιουργεί ένα σήμα στην αρχή κάθε περιόδου τ_v, οπότε αρχίζει να εξυπηρετείται από την ΚΜΕ το ν-οστό πρόγραμμα. Αν κάποιο από αυτά δεν ζητά εξυπηρέτηση ή έχει τελειώσει, η σειρά του δίνεται στο επόμενο.

Ένα ΛΣ που οργανώνει τη λειτουργία του υπολογιστή κατ' αυτό τον τρόπο ονομάζεται *Λειτουργικό Σύστημα Καταμερισμού Χρόνου* (time sharing).

Η περίοδος Τ μπορεί να είναι της τάξης των 3 sec· αν υποθέσουμε ότι υπάρχουν 6 προγράμματα για εκτέλεση, στο καθένα (σε κάθε περίοδο) διατίθεται 0,5 sec, χρόνος αρκετά μεγάλος για να καλύψει πολλές φορές ολόκληρη την εκτέλεση ενός προγράμματος.

Εάν κάποιο πρόγραμμα δεν καλύπτεται, περιμένει μέχρι την επόμενη χρονική περίοδο. Μερικές φορές βέβαια ένα πρόγραμμα πρέπει να εκτελεστεί αμέσως, χωρίς καθυστέρηση, επειδή η ταχύτητα απόκρισής του έχει ουσιαστική σημασία. Τότε, σε περιπτώσεις δηλαδή λειτουργίας σε *πραγματικό χρόνο* (real-time operation), το πρόγραμμα έχει προτεραιότητα έναντι των άλλων και εξυπηρετείται πρώτο.

Λειτουργικά Συστήματα 3ης και 4ης γενιάς

Σε πολλά νεότερα ΛΣ, αυτά της *3ης γενιάς*, γίνεται συνδυασμός των παραπάνω μεθόδων, για να υποστηρίζουν ταυτόχρονα ομαδική επεξεργασία και καταμερισμό χρόνου. Η έννοια της ομαδικής επεξεργασίας έχει σήμερα διαφοροποιηθεί και καθορίζεται βασικά από την έλλειψη αλληλεπίδρασης μεταξύ του χρήστη και του προγράμματός του.

Η ανάπτυξη νέων αρχιτεκτονικών για το υλικό, όπως π.χ. παράλληλων υπολογιστών, κατανεμημένων συστημάτων καθώς και δικτύων υπολογιστών, έδωσε την αφορμή για την ανάπτυξη ακόμα πιο εξελιγμένων και πολύπλοκων ΛΣ, αυτών της *4ης γενιάς*. Στα ΛΣ αυτά γίνεται πραγματικά ταυτόχρονη εκτέλεση πολλών προγραμμάτων, αφού κάθε επεξεργαστής μπορεί να ασχολείται με ένα διαφορετικό πρόγραμμα.

Τα συστήματα πολυεπεξεργαστών αντιστοιχούν στην ύπαρξη πολλών ζαχαροπλαστών που μοιράζονται τα ίδια σκεύη· τα κατανεμημένα συστήματα αντιστοιχούν σε πολλούς υπαλλήλους, καθένα με τα δικά του σκεύη και τέλος τα δίκτυα υπολογιστών είναι αντίστοιχα με μια αλυσίδα συνεργαζομένων ζαχαροπλαστείων.

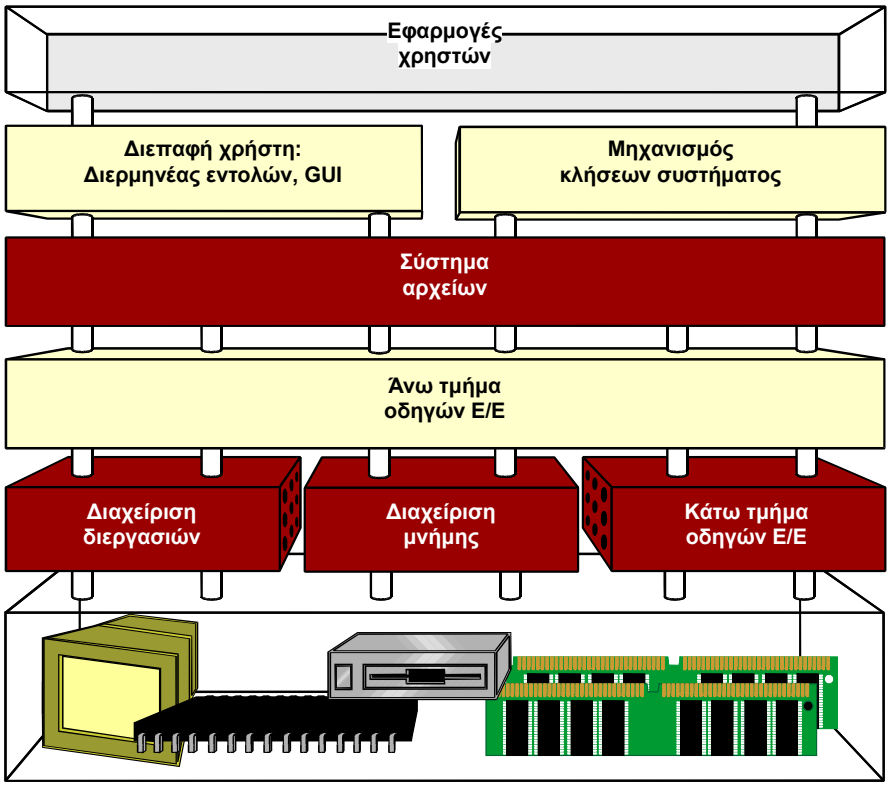
Στα υπολογιστικά συστήματα, το πρόγραμμα του ΛΣ συνήθως πωλείται από τον κατασκευαστή μαζί με το υλικό και είναι καθοριστικής σημασίας για τη λειτουργία του. Είναι δυνατόν ο ίδιος υπολογιστής να χρησιμοποιεί κατά καιρούς διαφορετικά ΛΣ, ανάλογα με τις ανάγκες των χρηστών του, και να έχει τελείως διαφορετική συμπεριφορά και απόδοση. Ακόμα, είναι δυνατόν υπολογιστές με διαφορετικό υλικό να χρησιμοποιούν το ίδιο ΛΣ και να έχουν παρόμοια συμπεριφορά, πιθανώς με διαφορετική απόδοση.

Τα ΛΣ είναι γραμμένα σε συμβολική γλώσσα ή σε συνδυασμό συμβολικής γλώσσας και κάποιας γλώσσας υψηλού επιπέδου (όπως η C).

Δομή ενός Λειτουργικού Συστήματος

Τα περισσότερα ΛΣ, και ιδιαίτερα τα σύγχρονα, είναι οργανωμένα σε *επίπεδα* (layers). Αυτό σημαίνει ότι κατά τη σχεδίασή τους έχουν διαιρεθεί σε τμήματα, και κάθε τμήμα τους επικοινωνεί μόνο με αυτά που βρίσκονται στο αμέσως ανώτερο ή το αμέσως κατώτερο επίπεδο. Όσα τμήματα χρησιμοποιούν απευθείας το υλικό του υπολογιστή, βρίσκονται στο κατώτερο επίπεδο του ΛΣ. Τα υπόλοιπα τμήματα, που βρίσκονται σε ανώτερα επίπεδα, δεν επικοινωνούν καθόλου με το υλικό, αλλά χρησιμοποιούν τα τμήματα που ανήκουν στο αμέσως κατώτερο επίπεδο.

Στο σχήμα φαίνεται ένα παράδειγμα οργάνωσης ΛΣ σε επίπεδα. Η οργάνωση αυτή βέβαια είναι ενδεικτική, γιατί υπάρχουν πολλές παραλλαγές της, αλλά η βασική φιλοσοφία είναι κοινή.



Στο χαμηλότερο επίπεδο ανήκουν τα τμήματα του ΛΣ που διαχειρίζονται:

- ☑ τη μνήμη
- ☑ τα υπό εκτέλεση προγράμματα
- ☑ τις λειτουργίες επικοινωνίας με τις περιφερειακές συσκευές.

Οι λειτουργίες αυτές αποτελούν το κάτω τμήμα των *οδηγών συσκευών E/E* (device drivers), ειδικών τμημάτων του ΛΣ που αναλαμβάνουν την επικοινωνία του ΛΣ με τα περιφερειακά. Στο αμέσως ανώτερο επίπεδο βρίσκεται το άνω τμήμα

των οδηγών Ε/Ε και πάνω από αυτό βρίσκεται το επίπεδο που κάνει τη διαχείριση του συστήματος αρχείων. Τα προγράμματα των χρηστών επικοινωνούν μόνο με το υψηλότερο επίπεδο σε ένα ΛΣ, που αποτελείται από τη *διεπαφή χρήστη* (user interface) και τις κλήσεις συστήματος. Η διεπαφή με το χρήστη μπορεί να γίνεται είτε με εντολές, με το *διερμηνέα εντολών* (command interpreter), ή με μία *διεπαφή χρήστη με χρήση γραφικών* (Graphical User Interface, GUI).

Παρατηρήστε ότι όπως είδαμε γενικά για τον υπολογιστή στο μάθημα 1.1, και το λειτουργικό σύστημα έχει ιεραρχική οργάνωση.

Ανακεφαλαίωση

Το Λειτουργικό Σύστημα είναι ένα σύνολο προγραμμάτων που οργανώνουν και επιβλέπουν τις λειτουργίες του υλικού σε ένα υπολογιστικό σύστημα. Κατά την εξέλιξή τους έχουν εμφανιστεί διάφορες κατηγορίες ΛΣ, όπως τα ΛΣ ομαδικής επεξεργασίας ($1^η$ γενιά), πολυπρογραμματισμού και καταμερισμού χρόνου ($2^η$ γενιά), πολυεπεξεργασίας ($3^η$ και $4^η$ γενιά) κλπ. Τα περισσότερα ΛΣ είναι οργανωμένα σε επίπεδα, με το κατώτερο επίπεδο μόνο να επικοινωνεί απευθείας με το υλικό του υπολογιστή, και τα προγράμματα του χρήστη να χρησιμοποιούν μόνο το ανώτερο επίπεδο.

Γλωσσάριο όρων

Διεπαφή Χρήστη	*User Interface*
Διεπαφή Χρήστη Με Χρήση Γραφικών	*Graphical User Interface - GUI*
Διερμηνέας Εντολών	*Command Interpreter*
Επίπεδο	*Layer*
Καταμερισμός Χρόνου	*Time Sharing*
Κεντρική Μονάδα Επεξεργασίας – ΚΜΕ	*Central Processing Unit - CPU*
Λειτουργία Πραγματικού Χρόνου	*Real-Time Operation*
Λειτουργικό Σύστημα	*Operating System*
Λογισμικό	*Software*
Οδηγός Συσκευής	*Device Driver*
Ομαδική Επεξεργασία	*Batch Processing*
Πολυπρογραμματισμός	*Multiprogramming*
Πρόγραμμα Εφαρμογής	*Application Program*
Σήμα Διακοπής	*Interrupt*
Τηλεπεξεργασία	*Remote Processing*
Υλικό	*Hardware*
Χρονιστής	*Timer*
Χρόνος Ανακύκλωσης	*Turnaround Time*

Ερωτήσεις

? Ποια είναι τα βασικά τμήματα ενός υπολογιστικού συστήματος;

? Ποιο είναι το έργο ενός Λειτουργικού Συστήματος; Τι θα γινόταν αν δεν υπήρχαν ΛΣ;

? Πόσες γενιές ΛΣ έχουν εμφανιστεί, και τι είδους ΛΣ αποτελούν την κάθε γενιά;

? Περίγραψε την οργάνωση ενός ΛΣ σε επίπεδα. Τι πλεονεκτήματα νομίζεις ότι προσφέρει η οργάνωση αυτή;

Μάθημα 7.2 Διεργασίες και Ελαφρές Διεργασίες

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να παρουσιάσει την έννοια της διεργασίας, που είναι πολύ βασική στα ΛΣ, την έννοια της ελαφριάς διεργασίας και να δείξει τις ομοιότητες και τις διαφορές τους.

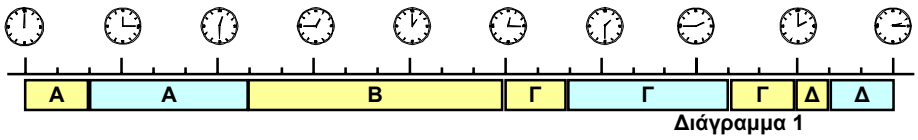
Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να περιγράφεις πώς ένα ΛΣ μπορεί να κατανέμει το χρόνο του μεταξύ πολλών προγραμμάτων*
- *Να εξηγείς τι είναι η διεργασία και σε τι διαφέρει από ένα πρόγραμμα*
- *Να ορίζεις τι είναι η μεταγωγή περιβάλλοντος*
- *Να εξηγείς τι είναι η ελαφρή διεργασία και σε τι διαφέρει από μία διεργασία*
- *Να προσδιορίζεις πότε είναι προτιμότερο να χρησιμοποιούμε τις ελαφρές διεργασίες*

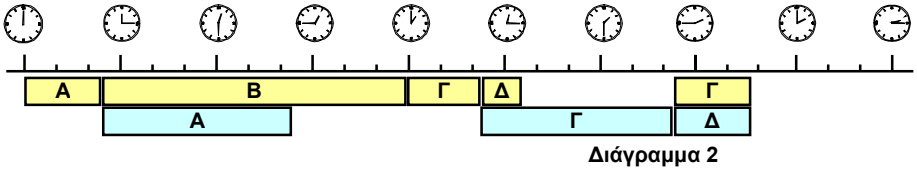
Στους περισσότερους υπολογιστές που έχουν μια μόνο ΚΜΕ, σε κάθε χρονική στιγμή μπορεί να εκτελείται μια μόνο εντολή γλώσσας μηχανής, άρα και ένα μόνο πρόγραμμα. Είναι όμως δυνατόν να κατανέμει η ΚΜΕ το χρόνο της, δίνοντας εκ περιτροπής ένα μικρό ποσοστό του χρόνου της σε κάθε ένα πρόγραμμα έτσι ώστε να δίνει την εντύπωση στους χρήστες ότι τα εκτελεί όλα μαζί.

Ας υποθέσουμε ότι ο ζαχαροπλάστης (η ΚΜΕ) έχει να παρασκευάσει τέσσερα γλυκά, και οι οδηγίες τους αναφέρουν ότι απαιτούν: το Α 35’ (10’ προετοιμασία και 25’ ψήσιμο), το Β 40’, το Γ 45’ (10’ προετοιμασία, 25’ ψήσιμο και 10’ ολοκλήρωση) και το Δ 15’ (5’ προετοιμασία και 10’ ψήσιμο) αντίστοιχα. Αν η εκτέλεση των συνταγών γίνει με τη σειρά παραλαβής τους, χωρίς παράλληλη χρησιμοποίηση του φούρνου, το όλο έργο μπορεί να οργανωθεί στις φάσεις του Διαγράμματος 1.



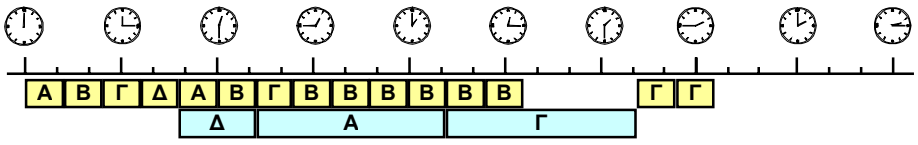
Διάγραμμα 1

Αν η εκτέλεση των συνταγών γίνεται με τη σειρά παραλαβής, αλλά με παράλληλη προετοιμασία και χρήση του φούρνου, έχουμε σημαντική βελτίωση στον απαιτούμενο χρόνο, όπως βλέπουμε στο Διάγραμμα 2.



Διάγραμμα 2

Αν όμως ο ζαχαροπλάστης αφιερώνει 5’ σε κάθε γλυκό, τότε το πρόγραμμα παρασκευής τους μπορεί να οργανωθεί σαν το Διάγραμμα 3.



Διάγραμμα 3

Ο τρόπος αυτός παρασκευής των γλυκών έχει ως αποτέλεσμα τη γρηγορότερη ολοκλήρωση αυτών που παρασκευάζονται εύκολα, ανεξάρτητα από τη σειρά με την οποία δόθηκαν οι συνταγές.

Αντίστοιχα ένα λειτουργικό σύστημα καταμερισμού χρόνου θα μοιράζει το χρόνο της ΚΜΕ σε μικρά στοιχειώδη διαστήματα, τα *κβάντα χρόνου* (time quanta). Σε κάθε κβάντο επιλέγεται ένα από τα προγράμματα και εκτελούνται εντολές από αυτό μέχρι να τελειώσει το χρονικό διάστημα που του διατέθηκε. Όταν ο χρόνος ολοκληρωθεί, επιλέγεται ένα άλλο πρόγραμμα για εκτέλεση κ.ο.κ.

Η διαδικασία αυτή είναι αρκετά πολύπλοκη και επιβαρύνει τις απαιτήσεις του ίδιου του ΛΣ. Το όφελος όμως από αυτή είναι (1) **η αύξηση της απόδοσης**, αφού το υλικό χρησιμοποιείται παράλληλα και (2) **η μείωση του χρόνου ανακύκλωσης**, αφού τα προγράμματα κατά μέσο όρο εξυπηρετούνται πιο γρήγορα.

Η συνεχής εναλλαγή ανάμεσα σε πολλές συνταγές είναι πολύ εύκολο να προκαλέσει σφάλματα. Για να μη γίνονται λοιπόν λάθη, πρέπει ο ζαχαροπλάστης κατά την εναλλαγή από τη μια συνταγή στην άλλη να κάνει ορισμένες ενέργειες όπως:

- Να σημειώνει μέχρι ποιο σημείο της τρέχουσας συνταγής έχει φθάσει, ώστε να μπορεί να συνεχίσει από το σωστό σημείο όταν έλθει ξανά η σειρά της.
- Να τοποθετήσει ό,τι από το γλυκό έχει παρασκευάσει μέχρι εκείνη τη στιγμή σε ένα ορισμένο μέρος, για να μπορεί αργότερα να το ξαναπάρει.
- Αν κάποιο βήμα της τρέχουσας συνταγής δεν είναι δυνατόν να διακοπεί, όπως π.χ. όταν κάτι ψήνεται στο φούρνο, πρέπει να σημειώσει ότι αυτό το βήμα εκκρεμεί. Όταν το ψήσιμο ολοκληρωθεί, θα τοποθετήσει το ψημένο γλυκό μαζί με τα υπόλοιπα υλικά που θα ολοκληρώσουν την παρασκευή του γλυκού και θα βάλει στο φούρνο ό,τι από την επόμενη συνταγή μέχρι τώρα βρισκόταν σε αναμονή.

Όπως στο παράδειγμα του ζαχαροπλαστείου, έτσι και στα υπολογιστικά συστήματα πρέπει να υπάρχει ένα σύνολο πληροφοριών που περιγράφουν την κατάσταση στην οποία βρίσκεται το πρόγραμμα που εκτελείται, για να μπορεί να συνεχιστεί η εκτέλεσή του από το σωστό σημείο την επόμενη φορά που θα έρθει η σειρά του.

Βλέπουμε λοιπόν ότι ένα ΛΣ διαχειρίζεται προγράμματα σε εκτέλεση· μπορεί π.χ. να εκτελεί πολλές φορές το ίδιο πρόγραμμα, μία φορά για κάθε χρήστη. Τα εκτελούμενα προγράμματα αποτελούν ανεξάρτητες οντότητες για το ΛΣ και ονομάζονται *διεργασίες* (processes).

Μια διεργασία είναι ένα πρόγραμμα ή ένα αυτόνομο τμήμα προγράμματος υπό εκτέλεση. Οι όροι *πρόγραμμα* και *διεργασία* διαφοροποιούνται από το γεγονός ότι το πρόγραμμα είναι παθητική οντότητα ενώ η διεργασία είναι ενεργητική.

Η εναλλαγή από τη μια διεργασία στην άλλη ονομάζεται *μεταγωγή περιβάλλοντος* (context switching). Οι πληροφορίες που κρατούνται κατά τη μεταγωγή περιβάλλοντος αφορούν τα ακόλουθα:

- Από ποια εντολή του προγράμματος πρέπει να συνεχισθεί η εκτέλεση της διεργασίας την επόμενη φορά
- Ποια είναι η κατάσταση της ΚΜΕ, ώστε να επαναφερθεί για να συνεχιστεί σωστά η εκτέλεση της διεργασίας
- Ποια είναι τα ενδιάμεσα αποτελέσματα που έχουν υπολογιστεί μέχρι εκείνη τη στιγμή

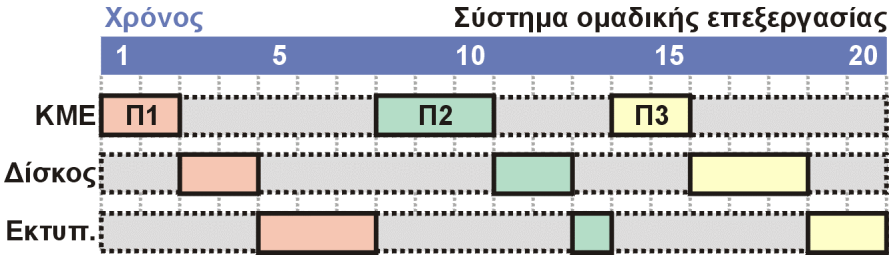
Το ΛΣ καταγράφει αυτές (και άλλες) πληροφορίες για κάθε διεργασία σε μια ειδική περιοχή της μνήμης που ονομάζεται *Σύνολο Ελέγχου Διεργασίας* (Process Control Block), ή αλλιώς *ΣΕΔ* (PCB).

Σύγκριση επίδοσης λειτουργικών συστημάτων

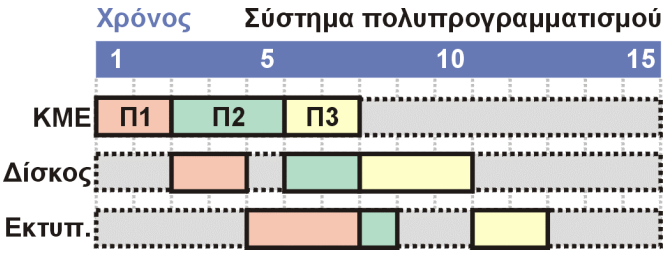
Ένα απλοποιημένο παράδειγμα μπορεί να δείξει με παραστατικό τρόπο τις διαφορές μεταξύ των διαφόρων ΛΣ και τα πλεονεκτήματα του καταμερισμού χρόνου. Σε ένα υπολογιστικό σύστημα δίνονται για εκτέλεση τρία προγράμματα με απαιτήσεις σε χρόνο που φαίνονται στον παράπλευρο πίνακα. Κάνουμε την υπόθεση ότι κάθε διεργασία χρησιμοποιεί κατά σειρά πρώτα την ΚΜΕ, μετά το δίσκο και μετά τον εκτυπωτή.

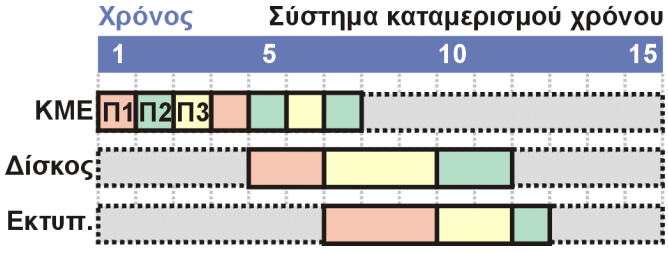
	ΚΜΕ	Δίσκος	Εκτυπωτής	Σύνολο
Πρόγ.1	2	2	3	7
Πρόγ.2	3	2	1	6
Πρόγ.3	2	3	2	7
Σύνολο	7	7	6	20

Το σχεδιάγραμμα εκτέλεσης για ένα ΛΣ ομαδικής επεξεργασίας είναι το εξής:



Για το σύστημα πολυπρογραμματισμού και το σύστημα καταμερισμού χρόνου θα έχουμε:





	Συνολικός χρόνος	Ανενεργός χρόνος	Τέλος της π_1	Τέλος της π_2	Τέλος της π_3
Ομαδική Επεξεργασία	20	40	7	13	20
Πολυπρογραμματισμός	12	16	7	8	12
Καταμερισμός χρόνου	12	16	9	12	11

Τα αποτελέσματα για τα τρια συστήματα συνοψίζονται στον παράπλευρο πίνακα. Παρατηρούμε ότι ο συνολικός χρόνος εκτέλεσης και ο ανενεργός χρόνος για το σύστημα ομαδικής επεξεργασίας είναι πολύ μεγαλύτεροι από ό,τι στα άλλα δυο συστήματα. Το σύστημα καταμερισμού χρόνου έχει άλλο ένα πλεονέκτημα: καμία διεργασία δεν ευνοείται ως προς το πότε θα τερματίσει (ενώ στο σύστημα πολυπρογραμματισμού υπάρχει τέτοια ανισότητα)· μια και όλες οι διεργασίες έχουν περίπου τις ίδιες χρονικές απαιτήσεις, τελειώνουν περίπου στον ίδιο χρόνο.

Οι επιδόσεις της εκτέλεσης των διαφόρων διεργασιών από ένα ΛΣ εξαρτάται από τον τύπο του ΛΣ, αλλά επηρεάζεται και από τις απαιτήσεις των διεργασιών.

Ελαφρές διεργασίες

Κάθε ΛΣ αποτελείται από πολλά επί μέρους προγράμματα, τα οποία εκτελούνται παράλληλα για να εξυπηρετούν τους χρήστες, είναι δηλαδή και αυτό χωρισμένο σε διεργασίες. Το ίδιο μπορεί να γίνει και με τα προγράμματα των χρηστών, να διαιρεθούν δηλαδή σε τμήματα τα οποία εκτελούνται παράλληλα.

> Ο ζαχαροπλάστης (ΚΜΕ) αφιερώνει 5' κάθε φορά στην εκτέλεση μιας συνταγής. Αυτά τα 5' μπορεί να τα μοιράσει σε 5 τμήματα του 1' το καθένα. Σε κάθε τέτοιο τμήμα του 1' μπορεί να ασχοληθεί σε κάθε τμήμα με ένα διαφορετικό στοιχείο του γλυκού το οποίο είναι ανεξάρτητο από τα άλλα. Έτσι μπορεί π.χ. να παρακολουθεί το ψήσιμο της ζύμης στο φούρνο και να ανακατεύει την κρέμα που βράζει.

Όταν όμως τα τμήματα αυτά πρέπει να μοιράζονται διάφορα στοιχεία του προγράμματος, όπως π.χ. μεταβλητές, μπορεί να χρησιμοποιηθεί και ένα εναλλακτικό είδος διεργασιών, οι *ελαφρές διεργασίες* (lightweight processes) ή *νήματα εκτέλεσης* (threads of execution) ή απλούστερα *νήματα* (threads).

Τα νήματα έχουν πολλές ομοιότητες με τις διεργασίες και μια βασική διαφορά: χρησιμοποιούν από κοινού ένα τμήμα μνήμης, στο οποίο έχουν όλα πρόσβαση.

Όπως για κάθε διεργασία, έτσι και για κάθε νήμα το ΛΣ κρατά ένα σύνολο από πληροφορίες σε μια ειδική περιοχή της μνήμης, η οποία αποκαλείται *Σύνολο Ελέγχου Νήματος* (Thread Control Block) ή αλλιώς *ΣΕΝ* (TCB). Το ΣΕΝ κρατά τις απαραίτητες πληροφορίες για κάθε νήμα, αλλά όσες πληροφορίες είναι κοινές μεταξύ τους (π.χ. αυτές για τη μοιραζόμενη μνήμη) καταγράφονται μια φορά μόνο.

> Ο ζαχαροπλάστης πρέπει να θυμάται ή να καταγράφει την πρόοδο για κάθε στάδιο μιας συνταγής που εκτελείται παράλληλα με άλλα στάδια της εκτέλεσής της. Τα σκεύη όμως και τα υλικά που αφορούν τα διάφορα στάδια τα τοποθετεί στο ίδιο μέρος, εκείνο που έχει καθορίσει για τη συγκεκριμένη συνταγή.

Όταν ένα πρόγραμμα διαιρεθεί σε νήματα που εκτελούνται «παράλληλα» (όπως οι διεργασίες), αντί σε ανεξάρτητες διεργασίες, προκαλείται μικρότερη επιβάρυνση στο υπολογιστικό σύστημα, γιατί η μεταγωγή περιβάλλοντος μεταξύ νημάτων είναι πιο γρήγορη από ό,τι μεταξύ διεργασιών. Επιπλέον δεν είναι απαραίτητο πάντα να χρησιμοποιηθούν μηχανισμοί επικοινωνίας από τα νήματα, αφού έχουν για το σκοπό αυτό τη μνήμη που μοιράζονται, κάτι που θα γίνει κατανοητό παρακάτω.

Ένα παράδειγμα προγράμματος που μπορεί να χωριστεί σε νήματα, φαίνεται στο σχήμα. Το πρόγραμμα αυτό υπολογίζει για τις μεταβλητές x και y:

- Με τη ρουτίνα Athroisma, το άθροισμα τους και το αποθηκεύει στη μεταβλητή sum.
- Με τη ρουτίνα Diafora, τη διαφορά τους και την αποθηκεύει στη μεταβλητή diff.
- Με τη ρουτίνα Phliko, το πηλίκο τους και το αποθηκεύει στη μεταβλητή quot.

```
var
x, y: integer;
sum, diff: integer;
quot: real;

procedure Athroisma;
begin
   sum := x + y
end;

procedure Diafora;
begin
   diff := x - y
end;

procedure Phliko;
begin
   quot := x / y
end;
```

Οι τρεις αυτοί υπολογισμοί προσδιορίζουν την τιμή διαφορετικών μεταβλητών και ο ένας δεν εξαρτάται από τον άλλο, έτσι μπορούν να γίνουν παράλληλα.

Αν επιλέξουμε να διαιρέσουμε το πρόγραμμα σε τρεις ταυτόχρονες διεργασίες, και οι τρεις χρειάζονται τις μεταβλητές x και y για να κάνουν τους υπολογισμούς τους. Έτσι η λύση αυτή έχει δυο μειονεκτήματα:

→ Η μνήμη σπαταλάται, γιατί έχουμε τρία διαφορετικά αντίγραφα κάθε μεταβλητής στη μνήμη. Εδώ βέβαια πρόκειται για δυο μεταβλητές μόνο, αλλά αν επρόκειτο για πολλά δεδομένα, η σπατάλη θα ήταν πολύ ουσιαστική.

→ Πρέπει να χρησιμοποιηθεί κάποιος μηχανισμός επικοινωνίας που θα εξασφαλίζει ότι οι μεταβλητές x και y των τριών διεργασιών περιέχουν τα ίδια δεδομένα. Αν π.χ. τα δεδομένα δίνονται από το χρήστη, πρέπει η μια από τις τρεις διεργασίες να τα διαβάσει και να τα στείλει στις άλλες δυο, ή μια άλλη διεργασία να τα ζητήσει από το χρήστη και να τα γράψει σε ένα αρχείο, από όπου θα τα διαβάσουν οι υπόλοιπες. Αυτές οι διαδικασίες και προγραμματιστικό κόπο απαιτούν, και επιβαρύνουν το χρόνο εκτέλεσης των διεργασιών, και απαιτούν επιπλέον πόρους του συστήματος (μηχανισμούς επικοινωνίας ή αρχεία).

Η εναλλακτική λύση που δεν έχει τα παραπάνω μειονεκτήματα είναι να χωρίσουμε το πρόγραμμα σε τρία *νήματα*. Τα νήματα αυτά θα μοιράζονται τη μνήμη που καταλαμβάνουν οι δυο μεταβλητές, και θα τη διαβάζουν για να κάνουν τους υπολογισμούς τους. Μόνο ένα αντίγραφο των μεταβλητών θα κρατείται στη μνήμη -έτσι δε γίνεται σπατάλη- και βέβαια δεν τίθεται ζήτημα επικοινωνίας για ανταλλαγή δεδομένων.

Τα προγράμματα που μπορούν να διαιρεθούν σε τμήματα, κάθε ένα από τα οποία ανατίθεται σε ένα νήμα, ονομάζονται *ταυτόχρονα* (concurrent programs). Ένα τέτοιο πρόγραμμα είναι το προηγούμενο, ή π.χ. το σύστημα κράτησης θέσεων μιας αεροπορικής εταιρίας, όπου κάθε πράκτορας αντιστοιχεί σε ένα νήμα.

Στα επόμενα μαθήματα, ο όρος «Διεργασία» θα χρησιμοποιείται τόσο για τις διεργασίες (που δε χρησιμοποιούν μνήμη από κοινού) όσο και για τα νήματα (τα οποία χρησιμοποιούν μνήμη από κοινού), εκτός και αν αναφέρεται διαφορετικά.

Ανακεφαλαίωση

Ο καταμερισμός του χρόνου της ΚΜΕ μεταξύ πολλών προγραμμάτων δίνει τη δυνατότητα να εκτελούνται αυτά ταυτόχρονα βελτιώνοντας συνολικά τις επιδόσεις του υπολογιστικού συστήματος. Τα προγράμματα που βρίσκονται σε κάποιο στάδιο της εκτέλεσής τους από την ΚΜΕ ονομάζονται διεργασίες. Η διαδικασία εναλλαγής από τη μια διεργασία στην άλλη, δηλαδή η μεταγωγή περιβάλλοντος, απαιτεί την καταγραφή πληροφοριών για τη διεργασία που διακόπτεται ώστε να μπορεί να συνεχιστεί αργότερα σωστά η εκτέλεσή της.

Μια εναλλακτική μορφή οργάνωσης ταυτόχρονων προγραμμάτων, τα οποία διαιρούνται σε τμήματα που μπορούν να εκτελούνται παράλληλα, είναι τα νήματα, μια «ελαφριά» μορφή διεργασιών. Τα νήματα χρησιμοποιούν κοινό τμήμα μνήμης και έτσι δεν απαιτούν μηχανισμούς επικοινωνίας πολλές φορές· επίσης επιβαρύνουν λιγότερο το υπολογιστικό σύστημα.

Γλωσσάριο όρων

Διεργασία	*Process*
Ελαφρή διεργασία	*Lightweight Process*
Κβάντο Χρόνου	*Time Quantum πληθ.-a*
Μεταγωγή Περιβάλλοντος	*Context Switching*
Νήμα	*Thread*
Νήμα Εκτέλεσης	*Thread of Execution*
Σύνολο Ελέγχου Διεργασίας - ΣΕΔ	*Process Control Block - PCB*
Σύνολο Ελέγχου Νήματος - ΣΕΝ	*Thread Control Block - TCB*
Ταυτόχρονο Πρόγραμμα	*Concurrent Program*

Ερωτήσεις

? Ποια είναι τα οφέλη και οι συνέπειες όταν το ΛΣ εναλλάσσει προγράμματα στη χρήση της ΚΜΕ;

? Σε τι διαφέρει η διεργασία από ένα πρόγραμμα; Μπορούν πολλές διεργασίες να αντιστοιχούν στο ίδιο πρόγραμμα;

? Ποια είναι η βασική διαφορά μίας διεργασίας με μία ελαφρή διεργασία;

? Τι είναι η μεταγωγή περιβάλλοντος για μία διεργασία; Είναι η ίδια με τη μεταγωγή περιβάλλοντος για μία ελαφρή διεργασία;

? Πότε συμφέρει να χρησιμοποιήσουμε τις ελαφρές διεργασίες; Πότε νομίζεις ότι πρέπει να τις αποφύγουμε;

Μάθημα 7.3 Απεικόνιση Διεργασιών

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να παρουσιάσει δύο μεθόδους απεικόνισης ταυτόχρονων διεργασιών, μία σχηματική και μία προγραμματιστική.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς τι είναι ο γράφος προβαδίσματος*
- *Να αποφασίζεις πότε ένας γράφος προβαδίσματος παριστάνει διεργασίες που δεν μπορούν να εκτελεστούν*
- *Να συμβολίζεις παράλληλες διεργασίες με τις εντολές parbegin και parend*
- *Να περιγράφεις πώς φτιάχνουμε ένα πρόγραμμα με τις εντολές parbegin και parend από ένα γράφο προβαδίσματος*

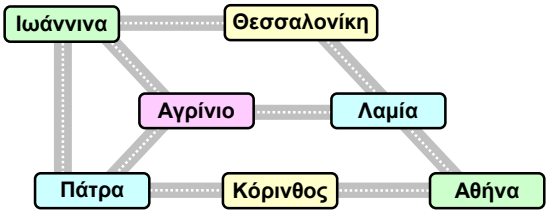
Ένας τρόπος για να αναπαραστήσουμε γραφικά τα ταυτόχρονα προγράμματα είναι να χρησιμοποιήσουμε το *γράφο προβαδίσματος* (precedence graph). Με τη βοήθεια ενός τέτοιου γράφου που αναπαριστά κάποιο πρόγραμμα, μπορούμε να δούμε ποια τμήματα του προγράμματος μπορούν να εκτελεστούν παράλληλα και ποια πρέπει να περιμένουν μέχρι να ολοκληρωθούν κάποια άλλα.

> Η παρασκευή ενός γλυκού αποτελείται από τα εξής στάδια: χτύπημα μαρέγκας, ανακάτεμα των υπόλοιπων υλικών, κόψιμο ξηρών καρπών, ανάμιξη μαρέγκας με τα υπόλοιπα υλικά, ανάμιξη ξηρών καρπών στη ζύμη, ψήσιμο του γλυκού, προετοιμασία γλάσου[*], γλασάρισμα του γλυκού. Κάθε ένα από τα στάδια αυτά μπορεί να αντιστοιχιστεί σε μια διαφορετική διεργασία. Για να οργανώσει όσο πιο αποδοτικά γίνεται ο ζαχαροπλάστης την εργασία του, πρέπει να γνωρίζει ποια στάδια πρέπει να έχουν ολοκληρωθεί προτού να ξεκινήσει ένα άλλο. Για παράδειγμα, δεν μπορεί να γίνει το ψήσιμο του γλυκού αν δεν έχει ετοιμαστεί πρώτα η ζύμη.

Ένας γράφος αποτελείται από *κόμβους* (vertices) και ακμές (edges). Μια ακμή συνδέει δυο κόμβους μεταξύ τους, και μπορεί να έχει ή όχι κατεύθυνση. Στην περίπτωση που η ακμή έχει κατεύθυνση ονομάζεται *κατευθυνόμενη* (directed).

Συνήθως σε ένα γράφο δεν εμφανίζονται μαζί κατευθυνόμενες και μη ακμές· όταν όλες οι ακμές είναι κατευθυνόμενες πρόκειται για ένα *κατευθυνόμενο γράφο* (directed graph), αλλιώς αποκαλείται απλώς γράφος.

[*] Γλάσο: μίγμα συνήθως από βούτυρο, ζάχαρη και αρωματικές ύλες που χρησιμοποιείται για την επικάλυψη των γλυκών και έχει γυαλιστερή όψη.



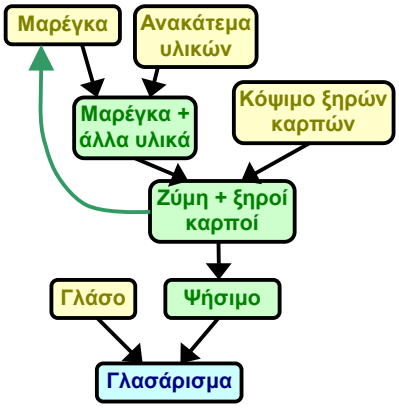
Ένας καλός τρόπος για να φανταστούμε ένα γράφο είναι το οδικό δίκτυο μεταξύ πόλεων. Οι κόμβοι είναι οι πόλεις και οι ακμές είναι οι δρόμοι που συνδέουν ζεύγη πόλεων. Οι μονόδρομοι είναι κατευθυνόμενες ακμές, και οι δρόμοι διπλής κατευθύνσεως είναι μη κατευθυνόμενες ακμές.

Στο σχήμα οι κόμβοι του γράφου είναι οι «Αγρίνιο», «Αθήνα», «Θεσσαλονίκη», «Ιωάννινα», «Κόρινθος», «Λαμία» και «Πάτρα». Όλοι οι δρόμοι του γράφου είναι διπλής κατευθύνσεως, οπότε ο γράφος είναι απλός και κάθε ακμή μπορούμε να τη «διασχίσουμε» και προς τις δυο κατευθύνσεις. Αν είχε κάποιους μονοδρόμους, τότε θα αντικαθιστούσαμε όλες τις απλές ακμές με δυο αντίθετης κατευθύνσεως. Έτσι για το δρόμο Θεσσαλονίκης-Ιωαννίνων, θα είχαμε μια ακμή Θεσσαλονίκη → Ιωάννινα και μια Ιωάννινα → Θεσσαλονίκη.

Σε ένα γράφο προβαδίσματος, κάθε διεργασία παριστάνεται με ένα κόμβο. Αν η διεργασία Δ_x πρέπει να έχει ολοκληρωθεί για να αρχίσει η Δ_y, τότε ο γράφος περιέχει μια κατευθυνόμενη ακμή από τον κόμβο της Δ_x σε αυτόν της Δ_y. Στην εποπτική αναπαράσταση του γράφου έχουμε ένα βέλος από τη Δ_x προς τη Δ_y.

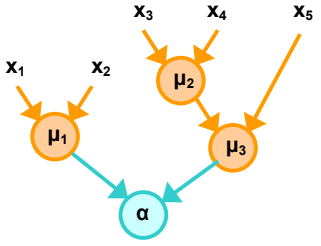
Οι γράφοι προβαδίσματος είναι λοιπόν κατευθυνόμενοι και μια βασική ιδιότητά τους είναι ότι δεν έχουν κύκλους, είναι δηλαδή *ακυκλικοί* (acyclic). Ένας κύκλος είναι μια σειρά από ακμές την οποία μπορούμε να διασχίσουμε από κόμβο σε κόμβο, καταλήγοντας στο σημείο από όπου ξεκινήσαμε. Στο σχήμα του οδικού δικτύου ένας τέτοιος κύκλος είναι ο Κόρινθος → Πάτρα → Αγρίνιο → Λαμία → Αθήνα → Κόρινθος.

Αν ένας γράφος προβαδίσματος έχει κύκλους, τότε δεν μπορεί να βρεθεί καμία δυνατή σειρά εκτέλεσης των διεργασιών.



Ο γράφος προβαδίσματος για το γλυκό που περιγράψαμε δίνεται στο σχήμα. Αν στο γράφο αυτό υπάρχει και μια ακμή ακόμα (η πράσινη γραμμή) από τη διεργασία «Ζύμη+ξηροί καρποί» στη διεργασία «Μαρέγκα», η οποία δημιουργεί ένα κύκλο θα έπρεπε: Η ανάμιξη «Ζύμη+ξηροί καρποί» να γίνει πριν από τη «Μαρέγκα», η οποία πρέπει να γίνει πριν από την ανάμιξη «Μαρέγκα+άλλα υλικά», που πρέπει να γίνει πριν από τη «Ζύμη+ξηροί καρποί». Είναι φανερό ότι δεν είναι δυνατόν να παρασκευαστεί το γλυκό στην περίπτωση αυτή, εξαιτίας του κύκλου στο γράφο προβαδίσματος.

Σε ένα γράφο προβαδίσματος είναι εύκολο να διαπιστώσουμε αν δυο διεργασίες μπορούν να εκτελεστούν ταυτόχρονα, αν δεν είναι δυνατό να μεταβούμε από τον κόμβο της μιας στον κόμβο της άλλης. Φυσικά η κυκλοφορία επιτρέπεται μόνο πάνω στις ακμές και με τη φορά που δείχνουν τα βέλη.



Ένα παράδειγμα ταυτόχρονου προγράμματος σε ένα υπολογιστή είναι η αριθμητική έκφραση $x_1 \times x_2 - (x_3 - x_4) / x_5$. Αν αναλύσουμε την έκφραση σε μερικά αποτελέσματα έχουμε $\mu_1 := x_1 \times x_2$, $\mu_2 := x_3 - x_4$, $\mu_3 := \mu_2 / x_5$, $\alpha := \mu_1 - \mu_3$. Για να υπολογιστεί ένα μερικό αποτέλεσμα, πρέπει να είναι διαθέσιμες όλες οι τιμές που χρησιμοποιεί, π.χ. για να υπολογιστεί το μ_3 χρειάζονται οι τιμές των μ_2 και x_5.

Αν αντιστοιχίσουμε κάθε μερικό αποτέλεσμα σε μια διεργασία, η εξάρτηση των μερικών αποτελεσμάτων από άλλα μας δίνει το γράφο προβαδίσματος για την έκφραση, από όπου φαίνεται ότι π.χ. τα μ_1 και μ_2 ή τα μ_1 και μ_3 μπορούν να υπολογιστούν ταυτόχρονα, γιατί δε συνδέονται με κάποιο μονοπάτι, ενώ τα μ_2 και μ_3 που συνδέονται δεν μπορούν να υπολογιστούν ταυτόχρονα.

Ο γράφος προβαδίσματος είναι ένα πολύ καλό εργαλείο για τη γραφική αναπαράσταση ταυτόχρονων προγραμμάτων. Οι συνηθισμένες όμως γλώσσες προγραμματισμού δεν είναι φτιαγμένες έτσι ώστε να μπορούν να περιγράψουν ταυτόχρονα προγράμματα και επομένως ούτε και γράφους προβαδίσματος. Για να συμβεί αυτό, θα πρέπει οι γλώσσες προγραμματισμού να εμπλουτιστούν κατάλληλα με νέα στοιχεία, όπως θα δούμε στη συνέχεια.

Ο συμβολισμός parbegin … parend

Οι εντολές *parbegin* (**par**allel **begin**) και *parend* (**par**allel **end**) είναι εντολές υψηλού επιπέδου για την περιγραφή ταυτοχρονισμού που προτάθηκαν από τον Dijkstra το 1965.

Σε ένα πρόγραμμα π.χ. Pascal τα **begin** … **end** περικλείουν μια ομάδα εντολών που πρέπει να εκτελεστεί σειριακά. Οι εντολές της ομάδας χωρίζονται μεταξύ τους με ερωτηματικό («;»). Αντίστοιχα, μια ομάδα εντολών που μπορεί να εκτελεστεί παράλληλα περικλείεται από τα **parbegin** … **parend** και οι εντολές χωρίζονται μεταξύ τους με το σύμβολο «||». Οι εντολές αυτές μπορούν να είναι απλές εντολές ή σύνθετες, να είναι δηλαδή με τη σειρά τους μια ομάδα εντολών που περικλείονται από begin…end.

```
begin
   m1 := x1 * x2;
   m2 := x3 - x4;
   m3 := m2 / x5;
   r := m1 - m3
end
```

Σχήμα Α

```
begin
   parbegin
      m1 := x1 * x2 ||
      begin
         m2 := x3 - x4;
         m3 := m2 / x5
      end;
   parend;
   r := m1 - m3
end
```

Σχήμα Β

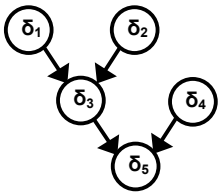
Η εκτέλεση μιας ομάδας παραλλήλων εντολών που περικλείονται από τα parbegin … parend γίνεται ταυτόχρονα: ξεκινούν όλες μαζί και η ομάδα εντολών τελειώνει όταν όλες τους τερματίσουν.

Η παράλληλη υλοποίηση του προγράμματος του Σχήματος Α φαίνεται στο Σχήμα Β.

Οι εντολές parbegin … parend δεν είναι σε θέση να περιγράψουν όλους τους δυνατούς γράφους προβαδίσματος· έχουν όμως το πλεονέκτημα ότι κάνουν το παράλληλο πρόγραμμα πολύ ευανάγνωστο και κατανοητό.

Γράφοι προβαδίσματος και προγράμματα

Πολλοί γράφοι προβαδίσματος μπορούν να αντιστοιχιστούν με περισσότερα από ένα προγράμματα γραμμένα με τις εντολές parbegin-parend. Στο γράφο του σχήματος, η εντολή δ_4 μπορεί να εκτελεστεί ταυτόχρονα με οποιαδήποτε από τις δ_1, δ_2 και δ_3, γιατί δεν υπάρχει μονοπάτι μέσα στο γράφο που να τις συνδέει.Έτσι μπορούν να γραφούν τουλάχιστον *τρία* διαφορετικά παράλληλα προγράμματα τα οποία υπακούουν στους περιορισμούς του γράφου.



Οι τρεις διαφορετικές λύσεις είναι:

1) Η δ_4 εκτελείται παράλληλα με τις δ_1 και δ_2 (οι οποίες μπορούν να εκτελεστούν παράλληλα), πριν από την δ_3. Το πρόγραμμα είναι:

```
parbegin δ1 || δ2 || δ4 parend;
δ3; δ5
```

2) Πρώτα εκτελούνται παράλληλα οι δ_1 και δ_2, και στη συνέχεια η δ_3 εκτελείται παράλληλα με την δ_4.

```
parbegin δ1 || δ2 parend;
parbegin δ3 || δ4 parend;
δ5
```

3) Η δ_1 και η δ_2 εκτελούνται παράλληλα, και μόλις τελειώσουν και οι δυο εκτελείται η δ_3. Η δ_4 εκτελείται παράλληλα με ολόκληρη την ομάδα των δ_1, δ_2, δ_3· ξεκινά μαζί με τις δ_1, δ_2, αλλά η δ_3 ξεκινά ανεξάρτητα από αυτήν. Η δ_5 περιμένει τις δ_3 και δ_4 να τερματίσουν για να ξεκινήσει.

```
parbegin
   begin
      parbegin δ1 || δ2 parend;
      δ3;
   end
||
   δ4
parend;
δ5
```

Από τα τρία προγράμματα μόνο το ένα είναι το «σωστό», δηλαδή αυτό που αντιπροσωπεύει καλύτερα το γράφο προβαδίσματος. Το πρώτο πρόγραμμα έχει ένα πρόβλημα: η διεργασία δ_3 περιμένει και την δ_4 να τελειώσει (εκτός από τις δ_1 και δ_2), ενώ αυτό δεν είναι απαραίτητο. Παρομοίως, στο δεύτερο πρόγραμμα η δ_4 περιμένει τις δ_1 και δ_2 χωρίς να υπάρχει λόγος. Και τα δυο προγράμματα δηλαδή, θέτουν επιπλέον περιορισμούς στην εκτέλεση των διεργασιών από αυτούς που περιγράφει ο γράφος. Αυτό δε συμβαίνει στο τρίτο πρόγραμμα, το οποίο είναι και το σωστό για την προγραμματιστική αναπαράσταση του γράφου προβαδίσματος.

Ανακεφαλαίωση

Ένας τρόπος αναπαράστασης ταυτοχρόνων προγραμμάτων είναι ο γράφος προβαδίσματος, όπου κάθε διεργασία είναι ένας κόμβος. Αν μια διεργασία πρέπει να ολοκληρωθεί προτού μια άλλη ξεκινήσει, ο γράφος περιέχει μια κατευθυνόμενη ακμή από τον πρώτο κόμβο στο δεύτερο. Ο γράφος προβαδίσματος πρέπει να είναι ακυκλικός. Διεργασίες που δε συνδέονται με ένα μονοπάτι μέσα στο γράφο προβαδίσματος μπορούν να εκτελεστούν ταυτόχρονα.

Οι εντολές parbegin και parend είναι ένας πολύ απλός και εύχρηστος συμβολισμός που χρησιμοποιείται για την αναπαράσταση ταυτόχρονων διεργασιών

Γλωσσάριο όρων

Ακμή	*Edge*
Ακυκλικός Γράφος	*Acyclic Graph*
Γράφος Προβαδίσματος	*Precedence Graph*
Κατευθυνόμενη Ακμή	*Directed Edge*
Κατευθυνόμενος Γράφος	*Directed Graph*
Κόμβος	*Vertex πληθ. Vertices*
Μονοπάτι	*Path*

Ερωτήσεις

? Τι είναι ο γράφος;

? Τι συμβολίζει μία ακμή σε ένα γράφο προβαδίσματος;

? Τι συμβαίνει αν ένας γράφος προβαδίσματος περιέχει ένα κύκλο από ακμές;

? Πώς εκτελούνται οι εντολές που περικλείονται από ένα ζεύγος parbegin … parend;

? Η αντιστοιχία προγραμμάτων και γράφων προβαδίσματος είναι 1-1; Γιατί;

Μάθημα 7.4 Κρίσιμα Τμήματα και Αμοιβαίος Αποκλεισμός

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να εξηγήσει την έννοια του κρίσιμου τμήματος σε μία διεργασία και να δείξει τη λύση για ένα απλό πρόβλημα κρίσιμου τμήματος.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς τι είναι κρίσιμο τμήμα μίας διεργασίας*
- *Να ορίζεις τον αμοιβαίο αποκλεισμό*
- *Να περιγράφεις πώς επιτυγχάνουμε αμοιβαίο αποκλεισμό σε ένα απλό πρόβλημα*

Κρίσιμα τμήματα

Αρκετά από τα εργαλεία που χρησιμοποιεί ο ζαχαροπλάστης, όπως π.χ. ο θερμοθάλαμος, μπορούν να χρησιμοποιηθούν για διαφόρες συνταγές. Κατά τη διάρκεια του πενταλέπτου εκτέλεσης μιας συνταγής, ο ζαχαροπλάστης έχει τοποθετήσει στο θερμοθάλαμο λιωμένη σοκολάτα για να τη διατηρήσει ζεστή, αλλά το πεντάλεπτο της συνταγής εκπνέει. Η συνταγή με την οποία ασχολείται για τα επόμενα 5’ ζητά να τοποθετηθεί η ζύμη στο θερμοθάλαμο για να φουσκώσει. Αν ο ζαχαροπλάστης βγάλει τη λιωμένη σοκολάτα για να βάλει τη ζύμη, η σοκολάτα θα κρυώσει, καταστρέφοντας έτσι την εκτέλεση της πρώτης συνταγής.

Οι οδηγίες του ιδιοκτήτη προς το ζαχαροπλάστη (ΚΜΕ), δηλαδή το ΛΣ, πρέπει να μεριμνούν ώστε κάτι τέτοιο να αποφευχθεί.

Σε ένα υπολογιστικό σύστημα, υπάρχουν *μοιραζόμενοι πόροι* (shared resources) -στο παράδειγμά μας ο θερμοθάλαμος- που χρησιμοποιούνται από πολλές διεργασίες και για να διαβαστούν αλλά και για να γραφτούν. Ο τρόπος που γίνονται οι προσπελάσεις σε αυτά έχει πολύ μεγάλη σημασία, γιατί όπως είδαμε και στο παράδειγμα, μια αλλαγή στα δεδομένα από τη μια διεργασία επηρεάζει και τις υπόλοιπες.

Μια τέτοια χαρακτηριστική περίπτωση διεργασιών που μοιράζονται δεδομένα είναι ένα online σύστημα τράπεζας. Η διαδικασία ανάληψης ενός ποσού Α από ένα μηχάνημα ΑΤΜ γίνεται σε πέντε φάσεις:

1. Ρώτα τον πελάτη για το χρηματικό ποσό Α που θέλει
2. Διάβασε το υπόλοιπο Υ του λογαριασμού
3. Υπολόγισε το νέο υπόλοιπο Υ’ μετά την ανάληψη του ποσού Α, Υ’:=Υ-Α
4. Αποθήκευσε το νέο υπόλοιπο
5. Δώσε τα χρήματα και την απόδειξη

Τι θα γίνει αν γίνονται από δυο διαφορετικά μηχανήματα συγχρόνως αναλήψεις για τον ίδιο λογαριασμό; Ας φανταστούμε ότι σε ένα κοινό λογαριασμό το αρχικό υπόλοιπο του λογαριασμού είναι 200.000, ο ένας πελάτης (π.χ. η σύζυγος) ζητά 50.000, ενώ ο άλλος (ο σύζυγος) ζητά 20.000. Επίσης υποθέτουμε ότι η εναλλαγή των διεργασιών γίνεται με τέτοιο ρυθμό ώστε σε κάθε κβάντο χρόνου, το οποίο έχει διάρκεια 1', μια διεργασία εκτελεί μια από τις τέσσερις παραπάνω φάσεις. Το υπόλοιπο του λογαριασμού μετά από τις δυο αναλήψεις θα πρέπει να είναι βέβαια 130.000.

Ώρα	Διεργασία Α	Διεργασία Β	Λογαριασμός
13:01	Ρώτα για το ποσό A=50.000	-	200.000
13:02	-	Ρώτα για το ποσό A=20.000	200.000
13:03	Διάβασε το υπόλοιπο Y=200.000	-	200.000
13:03	-	Διάβασε το υπόλοιπο Y=200.000	200.000
13:04	Y' := Y - A = 200.000-50.000 = *150.000*	-	200.000
13:05	-	Y' := Y - A = 200.000-20.000 = *180.000*	200.000
13:06	Αποθήκευσε το Y'	-	150.000
13:07	-	Αποθήκευσε το Y'	**180.000**

Βλέπουμε λοιπόν ότι το τελικό υπόλοιπο του λογαριασμού δεν είναι σωστό, και μάλιστα εξαρτάται από το ποια διεργασία ξεκίνησε πρώτη: αν είχε ξεκινήσει πρώτη η Β, το υπόλοιπο θα ήταν 150.000. Αν η διεργασία Α ξεκινήσει πρώτη, ο πελάτης που αντιστοιχεί σε αυτή θα λάβει απόδειξη με το σωστό υπόλοιπο για τη συναλλαγή του, ενώ ο πελάτης της διεργασίας Β θα λάβει απόδειξη με λανθασμένο υπόλοιπο.

Το λάθος στο πρόγραμμα προέρχεται από το γεγονός ότι πολλές διεργασίες μοιράζονται δεδομένα. Αν διαβάζουν και γράφουν σε αυτά ανεξέλεγκτα, τότε τα δεδομένα δεν παίρνουν τις σωστές τιμές. Το πρόβλημα αυτό ονομάζεται το πρόβλημα του *κρισίμου τμήματος* (critical section) όπου πρέπει να εξασφαλισθεί η *ακεραιότητα των δεδομένων* (data integrity).

Για να λύσει το πρόβλημα με το θερμοθάλαμο, οι οδηγίες προς το ζαχαροπλάστη (ΛΣ) μπορούν να προβλέπουν δυο λύσεις:

- Παρότι το πεντάλεπτο της πρώτης συνταγής έχει τελειώσει, να συνεχίσει να ασχολείται με αυτή μέχρι να έρθει η στιγμή που χρησιμοποιεί τη σοκολάτα και να απελευθερώσει το θερμοθάλαμο. Τότε θα τη διακόψει και θα προχωρήσει στην επόμενη.
- Να σημειώσει ότι ο θερμοθάλαμος είναι δεσμευμένος από την πρώτη συνταγή. Έτσι αφού η δεύτερη συνταγή χρειάζεται το θερμοθάλαμο για να προχωρήσει, και αυτός δεν είναι ελεύθερος, δεν προχωρεί με αυτή. Τη διακόπτει δηλαδή, πριν ακόμα να εκπνεύσει το πεντάλεπτό της και προχωρεί με κάποια άλλη. Το ίδιο συμβαίνει για όσες συνταγές και όσες φορές χρειάζονται το θερμοθάλαμο, έως ότου αυτός απελευθερωθεί.

Δυο αντίστοιχες λύσεις μπορούν να εφαρμοστούν και στο πρόβλημα του τραπεζικού λογαριασμού:

- Μέχρι να ολοκληρωθεί η συναλλαγή του πρώτου πελάτη δεν ξεκινά αυτή του δεύτερου. *Υλοποίηση*: Αδρανοποιείται το σύστημα διακοπών του χρονιστή (ο οποίος είναι υπεύθυνος να ενημερώνει την ΚΜΕ κάθε φορά που πρέπει να

διακοπεί μια διεργασία για να εκτελεστεί άλλη) μέχρι η πρώτη συναλλαγή να ολοκληρωθεί, οπότε δεν είναι δυνατόν να διακοπεί αυτή για να εκτελεστεί κάποια άλλη.

- Το ΛΣ «σημειώνει» ότι ο λογαριασμός χρησιμοποιείται από την πρώτη διεργασία, και η δεύτερη περιμένει έως ότου ο λογαριασμός «αποσημειωθεί», ελευθερωθεί δηλαδή. *Υλοποίηση*: Με ειδικούς μηχανισμούς, όπως οι σηματοφορείς, που θα εξεταστούν σε επόμενο μάθημα.

Και στις δυο περιπτώσεις, το πρόβλημα της ακεραιότητας των δεδομένων δεν αφορά την πρώτη φάση της διαδικασίας ανάληψης, δηλαδή την εισαγωγή από τον πελάτη του χρηματικού ποσού που επιθυμεί, γιατί τότε τα δεδομένα του λογαριασμού δεν έχουν διαβαστεί ή αλλάξει ακόμα. Αν διακοπεί η διεργασία αμέσως μόλις ρωτήσει τον πελάτη για το ύψος του ποσού, ό,τι αλλαγές γίνουν στο λογαριασμό από κάποια άλλη συναλλαγή δε θα επηρεάσουν αυτήν.

Βλέπουμε λοιπόν ότι μόνο σε ένα τμήμα της κάθε διεργασίας εμφανίζεται κίνδυνος για την ακεραιότητα των δεδομένων. Το τμήμα αυτό του κώδικα ονομάζεται *κρίσιμο τμήμα*.

> Όταν μια διεργασία εκτελεί τον κώδικα του κρίσιμου τμήματός της, καμία άλλη δεν μπορεί να εκτελεί το δικό της κρίσιμο τμήμα, έχουμε δηλαδή το φαινόμενο του *αμοιβαίου αποκλεισμού* (mutual exclusion).

Αν δυο ή περισσότερες διεργασίες είναι ταυτόχρονα έτοιμες να εισέλθουν στο κρίσιμο τμήμα τους, η επιλογή αυτής που τελικά θα το εκτελέσει γίνεται με τυχαίο τρόπο.

Ο Dijkstra (1965) έθεσε μερικούς ακόμα περιορισμούς για τα κρίσιμα τμήματα:

1. Όταν μια διεργασία εκτελεί κώδικα εκτός του κρίσιμου τμήματός της, δεν μπορεί να αποτρέψει άλλες διεργασίες να εκτελέσουν το δικό τους κρίσιμο τμήμα.
2. Όταν δυο ή περισσότερες διεργασίες είναι ταυτόχρονα έτοιμες να εισέλθουν στο κρίσιμο τμήμα τους, η λήψη απόφασης δεν πρέπει να αναβάλλεται επ' άπειρο.
3. Πρέπει να υπάρχει *δικαιοσύνη* (fairness) στον τρόπο επιλογής της διεργασίας που θα εισέλθει στο κρίσιμο τμήμα· όλες οι διεργασίες πρέπει να έχουν την ίδια πιθανότητα να εξυπηρετηθούν μέσα σε κάποιο λογικό χρονικό διάστημα και όχι κάποιες να μονοπωλούν το σύστημα.

Οι δυο πρώτοι περιορισμοί προφυλάσσουν το σύστημα από το *αδιέξοδο* (deadlock) ή το *αμοιβαίο μπλοκάρισμα* (mutual blocking), όπου όλες οι διεργασίες εμποδίζονται να εκτελέσουν το κρίσιμο τμήμα τους ενώ τουλάχιστον μία θα μπορούσε να το κάνει.

Αμοιβαίος αποκλεισμός για δυο διεργασίες: η λύση του Peterson

Η πρώτη λύση που δόθηκε στο πρόβλημα του αμοιβαίου αποκλεισμού για δύο διεργασίες ήταν αυτή του Δανού μαθηματικού T. Dekker, όμως η λύση που έδωσε το 1981 ο J. Peterson είναι πολύ πιο απλή. Υπάρχει μια κοινή μεταβλητή με όνομα π.χ. seira, που δείχνει ποια από τις δυο διεργασίες έχει σειρά να εκτελέσει το κρίσιμο τμήμα της. Αν η μεταβλητή έχει την τιμή 1, τότε είναι σειρά της πρώτης διεργασίας, ενώ αν έχει την τιμή 2 είναι σειρά της δεύτερης. Επίσης υπάρχουν δύο μεταβλητές, μία για κάθε διεργασία, με ονόματα π.χ. eisodos1 και eisodos2. Η μεταβλητή eisodos1 έχει την τιμή true όταν η διεργασία 1 είναι έτοιμη να εκτελέσει ή εκτελεί το κρίσιμο τμήμα της, ενώ σε όλες τις άλλες φάσεις εκτέλεσής της έχει τιμή false.

Διεργασία 1

```
repeat
  A1 := DiabasePoso;
  eisodos1 := true;
  (* εκδήλωση πρόθεσης εισόδου
     στο κρίσιμο τμήμα *)
  seira := 2;

  (* Περίμενε να έρθει η σειρά σου *)
  while eisodos2 and seira <> 1 do
        do_nothing;

  Y := Y - A1; (* κρίσιμο τμήμα *)
  eisodos1 := false;
  DosePoso;
until false;
```

Διεργασία 2

```
repeat
  A2 := DiabasePoso;
  eisodos2 := true;
  (* εκδήλωση πρόθεσης εισόδου
     στο κρίσιμο τμήμα *)
  seira := 1;

  (* Περίμενε να έρθει η σειρά σου *)
  while eisodos1 and seira <> 2 do
           do_nothing;

  Y := Y - A2; (* κρίσιμο τμήμα *)
  eisodos2 := false;
  DosePoso;
until false;
```

Αντίστοιχες τιμές παίρνει και η μεταβλητή eisodos2, για τη διεργασία 2. Οι δύο διεργασίες έχουν τη μορφή που φαίνεται στα σχήματα.

Κάθε διεργασία περιμένει για την άλλη, εφόσον είναι η σειρά της άλλης και εκείνη έχει δηλώσει πρόθεση να εκτελέσει το κρίσιμο τμήμα της. Αν π.χ. είναι η σειρά της διεργασίας 2 αλλά εκείνη δεν είναι έτοιμη να εκτελέσει το κρίσιμο τμήμα της (δηλαδή eisodos2=false), τότε η διεργασία 1 μπαίνει στο δικό της κρίσιμο τμήμα, εμποδίζοντας τη 2 με τη βοήθεια της eisodos1. Αν και οι δυο διεργασίες θέλουν ταυτόχρονα να εκτελέσουν το κρίσιμο τμήμα τους, το εκτελεί πρώτα εκείνη που υποδεικνύεται από τη μεταβλητή seira.

Η λύση αυτή μπορεί να επεκταθεί και για την περίπτωση που έχουμε ν διεργασίες· η λύση αυτή όμως είναι αρκετά πολύπλοκη, ενώ το πρόβλημα λύνεται πολύ πιο απλά με τη βοήθεια των *σηματοφορέων* (semaphores) που περιγράφονται στη συνέχεια.

Ανακεφαλαίωση

Στα ταυτόχρονα προγράμματα η ύπαρξη δεδομένων που χρησιμοποιούν από κοινού πολλές διεργασίες (και τουλάχιστον μια από αυτές γράφει στα δεδομένα), εισάγει το ζήτημα της ακεραιότητας των δεδομένων αυτών. Το τμήμα κώδικα μιας διεργασίας το οποίο διαβάζει ή γράφει χρησιμοποιούν από κοινού χρησιμοποιούμενα δεδομένα ονομάζεται κρίσιμο τμήμα. Από μια ομάδα διεργασιών που χρησιμοποιούν από κοινού δεδομένα μόνο μια μπορεί να εκτελεί ανά πάσα στιγμή το κρίσιμο τμήμα της. Αν για κάποιο λόγο όλες οι διεργασίες της ομάδας εμποδίζονται να εκτελέσουν το κρίσιμο τμήμα τους, τότε έχουμε μια περίπτωση αδιεξόδου.

Γλωσσάριο όρων

Αδιέξοδο	*Deadlock*
Ακεραιότητα Δεδομένων	*Data Integrity*
Αμοιβαίο Μπλοκάρισμα	*Mutual Blocking*
Αμοιβαίος Αποκλεισμός	*Mutual Exclusion*
Δικαιοσύνη	*Fairness*
Κρίσιμο Τμήμα	*Critical Section*
Μοιραζόμενοι Πόροι	*Shared Resources*

Ερωτήσεις

? Τι είναι το κρίσιμο τμήμα; Περίγραψε το κρίσιμο τμήμα στο πρόβλημα των τραπεζικών αναλήψεων.

? Πότε έχουμε αμοιβαίο αποκλεισμό μεταξύ δύο διεργασιών;

? Εξήγησε τη λύση του Peterson για τον αμοιβαίο αποκλεισμό.

Μάθημα 7.5 Σηματοφορείς

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να παρουσιάσει τους σηματοφορείς και τον τρόπο που λύνουν το πρόβλημα του κρίσιμου τμήματος.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να ορίζεις τι είναι σηματοφορέας και οι λειτουργίες P και V*
- *Να περιγράφεις πώς λύνουν οι σηματοφορείς το πρόβλημα του κρίσιμου τμήματος*
- *Να απαριθμείς τις λίστες διεργασιών που κρατά το ΛΣ*
- *Να εξηγείς πώς λύνουμε απλά προβλήματα με σηματοφορείς*

Η λύση για το πρόβλημα του κρίσιμου τμήματος που είδαμε στο προηγούμενο μάθημα είναι σωστή, αλλά έχει δυο μειονεκτήματα:

1. **Προσθέτει πολλές μεταβλητές και επιπλέον εντολές στον κώδικα της διεργασίας κάνοντάς τον πολύπλοκο χωρίς λόγο**. Από τις 8 γραμμές κώδικα της κάθε διεργασίας που δόθηκαν στο προηγούμενο μάθημα, οι 3 μόνο είναι το πραγματικό πρόγραμμα, ενώ οι υπόλοιπες 5 είναι πρόσθετες.
2. **Δεν επεκτείνεται εύκολα για περισσότερες από δυο διεργασίες, όπου είναι ακόμα πιο πολύπλοκη**.

> Ο ανελκυστήρας μιας πολυκατοικίας χρησιμοποιείται με παρόμοιο τρόπο με τα μοιραζόμενα δεδομένα ενός ταυτόχρονου προγράμματος. Ο ένοικος που είναι μέσα στον ανελκυστήρα είναι η διεργασία που εκτελεί το κρίσιμο τμήμα της.
>
> Αν η χρήση του ανελκυστήρα γινόταν με τον τρόπο που περιγράψαμε στο προηγούμενο μάθημα, θα γινόταν μια πολύπλοκη διαδικασία: όποιος ήθελε να τον χρησιμοποιήσει θα δήλωνε αυτή του την πρόθεση, και ο ανελκυστήρας θα εξυπηρετούσε έναν-έναν όλους τους ενοίκους που τον χρειάζονται, σε κάθε όροφο με τη σειρά. Θα υπήρχαν κουμπιά για την εκδήλωση της ανάγκης χρήσης του ανελκυστήρα, για την επιλογή αυτού που έχει την προτεραιότητα, για την ελευθέρωση του ανελκυστήρα κλπ. Με αυτά θα γινόταν η συνεννόηση μεταξύ των ενοίκων κάθε φορά για το ποιος έχει προτεραιότητα, ποιος είναι επόμενος στη σειρά για να εξυπηρετηθεί και ποιος τελικά θα χρησιμοποιήσει τον ανελκυστήρα. Με μια τόσο πολύπλοκη και αργή διαδικασία μάλλον οι περισσότεροι θα χρησιμοποιούσαν το κλιμακοστάσιο!
>
> Τι γίνεται όμως στην πραγματικότητα; Όταν ο ανελκυστήρας χρησιμοποιείται, σε όλους τους ορόφους είναι αναμμένο το φωτάκι με την

ένδειξη «Κατειλημμένος». Όποιος θέλει να χρησιμοποιήσει τον ανελκυστήρα περιμένει μέχρι να σβήσει η φωτεινή ένδειξη και μετά εκδηλώνει την πρόθεσή του πατώντας ένα κουμπί. Όποιος προλάβει και το πατήσει πρώτος, καλεί τον ανελκυστήρα.

Οι λειτουργίες λοιπόν του συστήματος χρήσης του ανελκυστήρα είναι ουσιαστικά δυο:

- Ο ένοικος περιμένει να σβήσει η φωτεινή ένδειξη, πατά το κουμπί και καλεί τον ανελκυστήρα.
- Ο ένοικος βγαίνει από τον ανελκυστήρα και η φωτεινή ένδειξη σβήνει.

Η ιδέα της φωτεινής ένδειξης, που ενεργοποιείται με το πάτημα ενός κουμπιού, μπορεί να μεταφερθεί και στο πρόβλημα του κρίσιμου τμήματος, με τους *σηματοφορείς* (semaphores). Ο σηματοφορέας, που είναι ένας μετρητής με ακέραιες τιμές, αντιστοιχεί στη φωτεινή ένδειξη. Όταν έχει την τιμή 0, τότε η ένδειξη είναι αναμμένη, ενώ όταν έχει την τιμή 1 η ένδειξη είναι σβηστή. Το πάτημα του κουμπιού και η κλήση του ανελκυστήρα προκαλούν μείωση της μεταβλητής κατά 1, ενώ το σβήσιμο της φωτεινής ένδειξης αντιστοιχεί σε αύξηση της μεταβλητής κατά 1.

Λειτουργία P

Λειτουργία V

Ανελκυστήρας	Σηματοφορέας
Περίμενε να σβήσει η φωτεινή ένδειξη. Πάτα το κουμπί.	while s = 0 do_nothing; s := s - 1;
Η φωτεινή ένδειξη σβήνει.	s := s + 1;

Οι σηματοφορείς και οι δυο λειτουργίες τους, με τα ονόματα P και V, παρουσιάστηκαν από το Dijkstra το 1965. Οι δυο λειτουργίες εκτελούνται αδιαίρετα, δηλαδή όταν στη λειτουργία P τελειώσει ο βρόχος «while s = 0 do_nothing», η επόμενη εντολή θα εκτελεστεί αμέσως χωρίς διακοπή από άλλη διεργασία. Αν δυο διεργασίες επιχειρήσουν ταυτόχρονα να εκτελέσουν τη λειτουργία P, τότε θα εκτελεστούν με τυχαία σειρά.

Διεργασία 1

```
repeat
  A1 := DiabasePoso;
     P(sema);
     (* κρίσιμο τμήμα *)
     Y := Y - A1;
     V(sema);
  DosePoso;
until false;
```

Διεργασία 2

```
repeat
  A2 := DiabasePoso;
     P(sema);
     (* κρίσιμο τμήμα *)

     Y := Y - A2;
     V(sema);
  DosePoso;
until false;
```

Το πρόβλημα των αναλήψεων λύνεται τώρα με ένα σηματοφορέα, με όνομα π.χ. «sema», όπως φαίνεται στο σχήμα.

Υλοποίηση των λειτουργιών P και V

Όπως είδαμε, ένας σηματοφορέας είναι πρακτικά ένας μετρητής, τον οποίο μοιράζονται όσες διεργασίες χρησιμοποιούν το σηματοφορέα. Για να γίνονται σωστά οι λειτουργίες στο σηματοφορέα, πρέπει η πρόσβαση στο μοιραζόμενο αυτό μετρητή να γίνεται με ασφάλεια (κρίσιμο τμήμα). Αυτός είναι ο λόγος που οι λειτουργίες P και V πρέπει να εκτελούνται αδιαίρετα.

Σε όλους τους υπολογιστές οι εντολές σε γλώσσα μηχανής, δηλαδή στο στοιχειωδέστερο επίπεδο προγραμματισμού τους, εκτελούνται αδιαίρετα. Έτσι αν οι δυο λειτουργίες υλοποιούνται σαν εντολές γλώσσας μηχανής, τότε ικανοποιείται αυτόματα ο περιορισμός της αδιαίρετης εκτέλεσής τους. Αυτό όμως μπορεί να δημιουργήσει δυο σοβαρά προβλήματα:

- Αν ο σηματοφορέας είναι ήδη κατειλημμένος από μια διεργασία, δηλαδή έχει τιμή 0, η επόμενη που θα εκτελέσει τη λειτουργία P θα αρχίσει να εκτελεί *αδιαίρετα* το βρόχο «while s = 0 do_nothing». Αφού η εκτέλεση είναι αδιαίρετη, όσο ο βρόχος εκτελείται δεν είναι δυνατόν να διακοπεί η διεργασία για να εκτελεστεί κάποια άλλη, συνεπώς και να απελευθερωθεί ο σηματοφορέας από τη διεργασία που τον κατέχει. Ο υπολογιστής τότε βρίσκεται σε κατάσταση αδιεξόδου και εκτελεί επ' άπειρο το βρόχο.

- Ακόμα και αν το προηγούμενο πρόβλημα ξεπεραστεί, κάθε εκτέλεση της λειτουργίας P από μια διεργασία θα συνεπάγεται τη σπατάλη αρκετού υπολογιστικού χρόνου στην εκτέλεση του βρόχου. Οι διεργασίες, σε κάθε κβάντο χρόνου που τους αντιστοιχεί θα εκτελούν το βρόχο, ξοδεύοντας έτσι χρόνο που θα μπορούσε να χρησιμοποιηθεί πιο «παραγωγικά».

Για να ξεπεραστούν τα προβλήματα αυτά, οι λειτουργίες P και V δεν υλοποιούνται στην πραγματικότητα όπως ορίζονται. Για την υλοποίηση των σηματοφορέων χρησιμοποιούνται *λίστες αναμονής* (waiting lists).

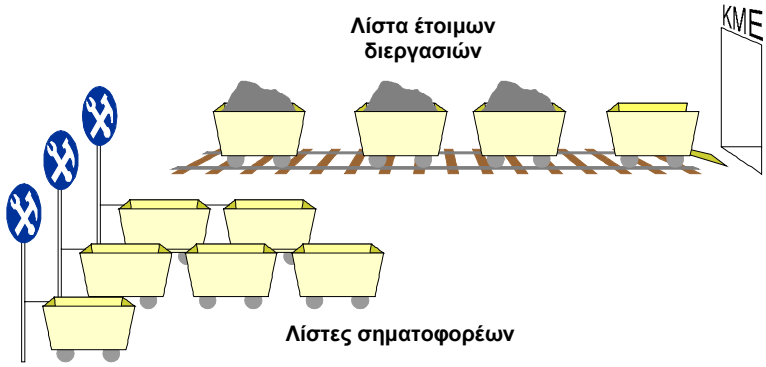
Κάθε σηματοφορέας έχει τη δική του λίστα αναμονής στην οποία περιμένουν οι διεργασίες που χρειάζονται το σηματοφορέα, για όσο χρόνο αυτός είναι κατειλημμένος. Όταν το ΛΣ επιλέγει την επόμενη διεργασία που θα εκτελεστεί, δε λαμβάνει υπόψη αυτές που περιμένουν στη λίστα κάποιου σηματοφορέα, γιατί έτσι και αλλιώς αυτή δεν έχει τίποτα να κάνει μέχρι να ελευθερωθεί ο σηματοφορέας.

Η λειτουργία P λοιπόν γίνεται ως εξής: αν ο σηματοφορέας είναι ελεύθερος, τότε η διεργασία τον καταλαμβάνει. Αν δεν είναι ελεύθερος, η διεργασία καταγράφεται στη λίστα αναμονής του σηματοφορέα, όπου και παραμένει χωρίς να εκτελείται. Στη λειτουργία V, όπου μια διεργασία ελευθερώνει το σηματοφορέα, το ΛΣ εξετάζει τη λίστα αναμονής του. Επιλέγει από αυτή την πρώτη διεργασία και τη διαγράφει από τη λίστα αναμονής, εισάγοντάς τη στη λίστα των διεργασιών που εκτελούνται εκ περιτροπής στην ΚΜΕ. Σύντομα θα έλθει η σειρά της διεργασίας αυτής να εκτελεστεί, και τότε θα καταλάβει το σηματοφορέα ολοκληρώνοντας έτσι τη λειτουργία P.

Η λίστα αναμονής ενός σηματοφορέα οργανώνεται σαν *ουρά* (queue). Σε μια ουρά, όπως π.χ. στο ταμείο μιας τράπεζας, επιλέγεται από την ουρά η διεργασία που περιμένει την περισσότερη ώρα.

Οι λίστες του λειτουργικού συστήματος

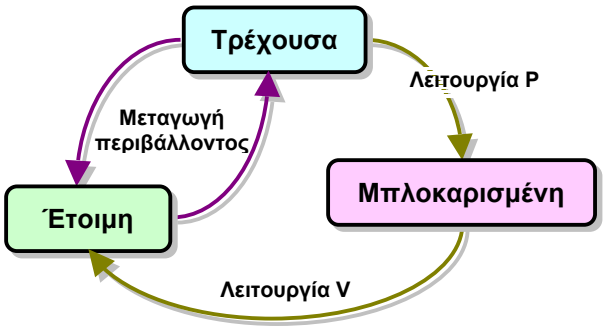
Με τη μέθοδο των λιστών αναμονής, οι διεργασίες χωρίζονται σε κατηγορίες: Για κάθε σηματοφορέα έχουμε την κατηγορία των διεργασιών που περιμένουν γι' αυτόν, και υπάρχει η κατηγορία των διεργασιών που δεν περιμένουν για κανένα σηματοφορέα. Για να μπορεί το ΛΣ να βρίσκει εύκολα και γρήγορα ποιες διεργασίες δεν περιμένουν σε κάποιο σηματοφορέα, κρατά μια ξεχωριστή λίστα με αυτές. Η λίστα αυτή ονομάζεται *λίστα έτοιμων διεργασιών* (ready list, RL)· από αυτήν επιλέγει το ΛΣ την επόμενη διεργασία που θα εκτελεστεί. Συνήθως και η λίστα έτοιμων διεργασιών είναι οργανωμένη σαν ουρά.



Ο τρόπος λειτουργίας της ΚΜΕ είναι παρόμοιος με αυτόν ενός ορυχείου, όπου βαγονάκια μεταφέρουν από τις σήραγγες το μετάλλευμα στο εργοστάσιο για επεξεργασία. Τα βαγονάκια - διεργασίες μπαίνουν ένα-ένα στο εργοστάσιο- ΚΜΕ από μια σιδηροδρομική γραμμή - λίστα έτοιμων διεργασιών για να εξυπηρετηθούν. Όμως κατά τη διάρκεια της λειτουργίας του ορυχείου τα βαγονάκια χρειάζονται συντήρηση· άλλο χρειάζεται λάδωμα στους τροχούς, άλλο επισκευή στα φρένα, και άλλο χρειάζεται επιδιόρθωση στον καταπέλτη. Ανάλογα με την επισκευή που χρειάζονται μπαίνουν στην ουρά του αντίστοιχου συνεργείου-σηματοφορέα, και όταν έρθει η σειρά τους επισκευάζονται. Για όσο

διάστημα βρίσκονται υπό επισκευή δε χρησιμοποιούνται, οπότε βρίσκονται εκτός της γραμμής-λίστας έτοιμων διεργασιών.

Μια διεργασία μπορεί λοιπόν να βρίσκεται στις εξής *καταστάσεις* (process states):



- ☞ εκτελείται από την ΚΜΕ: η διεργασία ονομάζεται *τρέχουσα* (running)
- ☞ περιμένει για εκτέλεση στη λίστα έτοιμων διεργασιών: η διεργασία ονομάζεται *έτοιμη* (ready)
- ☞ περιμένει στην ουρά κάποιου σηματοφορέα: η διεργασία είναι *μπλοκαρισμένη* (blocked)

Στο σχήμα βλέπουμε τον τρόπο που μία διεργασία μπορεί να μεταβεί από μία κατάσταση σε άλλη. Η μετάβαση από την κατάσταση «Τρέχουσα» στην κατάσταση «Μπλοκαρισμένη» με τη λειτουργία P γίνεται μόνο εφόσον ο σηματοφορέας έχει την τιμή 0.

Στη συνέχεια θα συναντήσουμε και άλλες καταστάσεις στις οποίες μπορεί να βρίσκεται μια διεργασία.

Το πρόβλημα των αναγνωστών και συγγραφέων

Στις *βάσεις δεδομένων* (databases) παρουσιάζεται ένα πολύ συνηθισμένο πρόβλημα αμοιβαίου αποκλεισμού διεργασιών, το πρόβλημα των αναγνωστών και συγγραφέων. Μια ομάδα διεργασιών μοιράζεται ένα αρχείο, στο οποίο άλλες διεργασίες θέλουν να γράψουν και να το τροποποιήσουν, ενώ άλλες θέλουν μόνο να διαβάσουν από αυτό. Για να εξασφαλίζεται το ότι τα περιεχόμενα του αρχείου είναι σωστά, πρέπει οι συγγραφείς να έχουν αποκλειστική πρόσβαση στο αρχείο. Όταν γράφει λοιπόν μια διεργασία-συγγραφέας καμία άλλη διεργασία δεν έχει πρόσβαση στο αρχείο. Αντίθετα, πολλές διεργασίες-αναγνώστες μπορούν να διαβάζουν ταυτόχρονα.

Για να δώσουμε μια λύση στο πρόβλημα αυτό πρέπει να αποφασίσουμε ποιοι θα έχουν προτεραιότητα: οι αναγνώστες ή οι συγγραφείς, όταν ταυτόχρονα μια διεργασία θέλει να διαβάσει και μια θέλει να γράψει. Εμείς θα υποθέσουμε εδώ ότι προηγούνται οι αναγνώστες, δηλαδή οι συγγραφείς περιμένουν να ολοκληρωθούν οι λειτουργίες ανάγνωσης.

- Στη συνέχεια πρέπει να αποφασίσουμε ποιους σηματοφορείς και μεταβλητές θα χρησιμοποιήσουμε και ποιο ρόλο θα αναλάβει ο καθένας. Θα εξετάζουμε ένα-ένα τα προβλήματα ακεραιότητας δεδομένων που υπάρχουν και θα προσθέτουμε διαδοχικά εντολές στον κώδικα των διεργασιών. Οι απλούστερες μορφές του κώδικα φαίνονται στο σχήμα. Η διεργασία - συγγραφέας γράφει στο αρχείο myfile, ενώ η διεργασία - αναγνώστης διαβάζει από αυτό.

Συγγραφείς

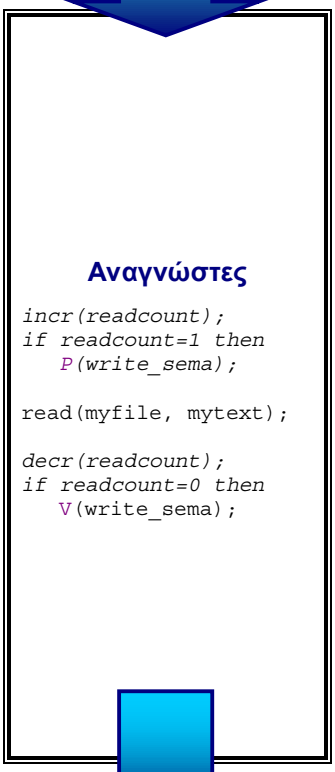
```
write(myfile, "…");
```

Αναγνώστες

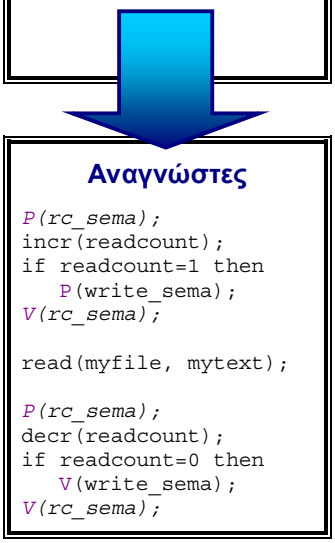
```
read(myfile, mytext);
```

- Οπωσδήποτε πρέπει να χρησιμοποιήσουμε ένα σηματοφορέα ο οποίος θα εξασφαλίζει το ότι μόνο ένας συγγραφέας θα γράφει ανά πάσα στιγμή στο αρχείο, ώστε οι αλλαγές στο αρχείο να γίνονται σωστά. Κάθε συγγραφέας προτού γράψει θα καταλαμβάνει το σηματοφορέα αυτό και όταν τελειώνει θα τον ελευθερώνει. Το όνομα του σηματοφορέα θα είναι write_sema.
  Τώρα κάθε διεργασία - συγγραφέας εκτελεί μία λειτουργία P στο σηματοφορέα, και όταν αυτή ολοκληρωθεί εκτελεί την εγγραφή. Στη συνέχεια απελευθερώνει το σηματοφορέα με τη λειτουργία V για να τον διαθέσει στους υπόλοιπους συγγραφείς.

- Για να βεβαιωθούν οι αναγνώστες ότι δε γράφει κάποιος στο αρχείο πριν διαβάσουν, πρέπει πριν από την ανάγνωση να καταλάβουν το σηματοφορέα write_sema. Όμως, όταν ένας αναγνώστης έχει καταλάβει το σηματοφορέα με τη λειτουργία P, όλοι οι άλλοι που θα εκτελέσουν επίσης την P θα τον περιμένουν. Αυτό είναι κάτι ανεπιθύμητο, μια που πολλοί αναγνώστες μπορούν να διαβάζουν ταυτόχρονα. Έτσι, αρκεί ο *πρώτος* αναγνώστης να καταλάβει το σηματοφορέα αυτό, και οι υπόλοιποι μπορούν πλέον να διαβάζουν άφοβα. Ο τελευταίος αναγνώστης αντίστοιχα πρέπει να ελευθερώσει τον write_sema.
  Για να διαπιστώσει μια διεργασία - αναγνώστης αν είναι η πρώτη, πρέπει σε μια μεταβλητή να κρατείται το πλήθος των αναγνωστών που είναι ενεργοί ανά πάσα στιγμή· στην μεταβλητή θα δώσουμε το όνομα readcount. Αρχικά η μεταβλητή έχει την τιμή 0, κάθε αναγνώστης την αυξάνει πριν από την ανάγνωση, και τη μειώνει μετά από αυτήν. Ο πρώτος αναγνώστης, αφού αυξήσει τη readcount, θα διαπιστώσει ότι έχει την τιμή 1 και θα καταλάβει το σηματοφορέα write_sema. Ο τελευταίος, όταν τη μειώσει, θα διαπιστώσει ότι έχει πάλι τιμή 0 και θα ελευθερώσει το σηματοφορέα write_sema.



- Όμως η προσθήκη που κάναμε στους αναγνώστες έχει ένα πρόβλημα: μπορεί πολλοί αναγνώστες μαζί να αυξάνουν ή να μειώνουν τη μεταβλητή readcount, και μετά να ελέγχουν την τιμή της. Σε μια τέτοια περίπτωση τα αποτελέσματα είναι ανεξέλεγκτα και συνήθως όχι σωστά. Μια που η readcount είναι μοιραζόμενη μεταβλητή, η πρόσβαση σε αυτήν πρέπει να φυλάσσεται από ένα σηματοφορέα, στον οποίο θα δώσουμε το όνομα rc_sema. Οι αναγνώστες θα καταλαμβάνουν τον rc_sema πριν να αυξήσουν την τιμή της readcount και θα τον ελευθερώνουν *αφού* έχουν ελέγξει την τιμή της.



Ανακεφαλαίωση

Οι σηματοφορείς είναι ένας μηχανισμός που λύνει με απλό και λειτουργικό τρόπο το πρόβλημα του κρίσιμου τμήματος. Είναι πρακτικά ακέραιες μεταβλητές, στις οποίες επενεργούν δυο λειτουργίες: η λειτουργία P, που πρέπει να εκτελεστεί πριν από την εκτέλεση του κρίσιμου τμήματος, και η λειτουργία V που πρέπει να εκτελεστεί αμέσως μετά. Η υλοποίηση των σηματοφορέων στα πραγματικά ΛΣ γίνεται με λίστες αναμονής όπου περιμένουν οι διεργασίες για να καταλάβουν το σηματοφορέα. Μια διεργασία περιμένει στην ουρά ενός σηματοφορέα, εκτελείται από την ΚΜΕ ή είναι έτοιμη για εκτέλεση.

Γλωσσάριο όρων

Έτοιμη Διεργασία	*Ready Process*
Κατάσταση Διεργασίας	*Process State*
Λειτουργία P	
Λειτουργία V	
Λίστα Αναμονής	*Waiting List*
Λίστα Έτοιμων Διεργασιών	*Ready List*
Μπλοκαρισμένη Διεργασία	*Blocked Process*
Ουρά	*Queue*
Σηματοφορέας	*Semaphore*
Τρέχουσα Διεργασία	*Running Process*

Ερωτήσεις

? Τι πλεονεκτήματα έχει ένας σηματοφορέας όταν θέλουμε να λύσουμε το πρόβλημα του κρίσιμου τμήματος για πολλές διεργασίες;

? Γιατί υλοποιούμε τους σηματοφορείς με λίστες αναμονής και όχι με τον τρόπο που παρουσιάσαμε στη θεωρία;

? Πόσους αναγνώστες και συγγραφείς μπορούμε να έχουμε στο παράδειγμα;

Τι μάθαμε σε αυτό το κεφάλαιο

- Το Λειτουργικό Σύστημα είναι ένα σύνολο προγραμμάτων που οργανώνουν και επιβλέπουν τις λειτουργίες του υλικού σε ένα υπολογιστικό σύστημα.
- Τα προγράμματα που βρίσκονται σε κάποιο στάδιο της εκτέλεσής τους από την ΚΜΕ ονομάζονται διεργασίες.
- Οι βασικοί τρόποι απεικόνισης διεργασιών είναι ο γράφος προήγησης και οι εντολές parbegin και parend.
- Το τμήμα μιας διεργασίας που χρησιμοποιεί μοιραζόμενα δεδομένα είναι το κρίσιμο τμήμα της. Μόνο μία διεργασία μπορεί να εκτελεί το κρίσιμο τμήμα της κάθε φορά.
- Οι σηματοφορείς είναι μία πολύ απλή και λειτουργική λύση για το πρόβλημα του κρίσιμου τμήματος.

Βιβλιογραφία – Πηγές

📖 Παπακωσταντίνου Γ., Ν. Μπιλάλη, Π. Τσανάκα, Λειτουργικά Συστήματα: Αρχές Λειτουργίας, Συμμετρία, 1989.

📖 Silberschatz A., Peterson J., Galvin P., Operating System Concepts, Addison - Wesley, 1991.