ΚΕΦΑΛΑΙΟ 1
Εισαγωγή στα λειτουργικά συστήματα

1.1. Γενικά

Σήμερα γνωρίζουμε ότι οι χρήστες των Ηλεκτρονικών Υπολογιστών (Η/Υ) γίνονται ολοένα και περισσότεροι σ' ολόκληρο τον κόσμο. Οι προσωπικοί Η/Υ δεν απευθύνονται μόνο στον επιστημονικό κόσμο της πληροφορικής, αλλά και σε εκατομμύρια χρήστες από διάφορους επιστημονικούς και μη χώρους.

Έτσι λοιπόν, μετά τη δεκαετία του 1980, δημιουργήθηκε η ανάγκη εκσυγχρονισμού των προγραμμάτων που έλεγχαν τη λειτουργία των Η/Υ (των λειτουργικών συστημάτων) και προσαρμογής τους στις ανάγκες του απλού πολίτη. Οπότε, διαμορφώθηκαν τα σύγχρονα λειτουργικά συστήματα, που είναι κυρίως μετεξέλιξη κάποιων παλιών εκδόσεων με νέες αρχές και φιλοσοφία, έτσι ώστε να είναι εύχρηστα και φιλικά απέναντι στον οποιονδήποτε. Παρόλα αυτά όμως, κάποια λειτουργικά συστήματα που δε χαρακτηρίζονται για τη φιλικότητά τους, αλλά αντίθετα για την αποτελεσματικότητά τους, όπως το λειτουργικό σύστημα Unix, διατήρησαν τη μεγάλη τους απήχηση κυρίως ανάμεσα στους ειδικούς του χώρου της πληροφορικής (Παπασταύρου, 1992).

Από τις αρχές της δεκαετίας του 1990, βρισκόμαστε μπροστά στις πιο συναρπαστικές εξελίξεις στο χώρο της πληροφορικής. Η αρχιτεκτονική των Η/Υ έχει αλλάξει. Η παλιά καλή αρχιτεκτονική τύπου SISC άρχισε να δίνει τη θέση της στην αρχιτεκτονική RISC τόσο στους Η/Υ της IBM, όσο και της Apple. Ηδη παράγονται επιτραπέζιοι ηλεκτρονικοί υπολογιστές που εκτελούν πάνω από **εκατό** εκατομμύρια οδηγίες το δευτερόλεπτο. Όμως το εντυπωσιακότερο όλων είναι ότι καθώς οι υπολογιστές γίνονται ολοένα ισχυρότεροι, οι τιμές τους γίνονται ολοένα χαμηλότερες. Πρόκειται λοιπόν για ένα επίτευγμα όχι μόνο της τεχνολογίας των ολοκληρωμένων κυκλωμάτων, αλλά και της επιστήμης ανάπτυξης λογισμικού, γιατί

παράλληλα με τις νέες ταχύτερες φθηνότερες και ελαφρύτερες συσκευές αναπτύχθηκαν και τα κατάλληλα προγράμματα (λογισμικό) ελέγχου τους.

1.1.1. Ιστορική αναδρομή

Οι πρώτοι υπολογιστές της δεκαετίας του 1940 δε διέθεταν κανενός είδους λειτουργικό σύστημα. Τα προγράμματα γράφονταν και εισάγονταν στον Η/Υ σε σειρές, αποτελούμενες από ψηφία 1 ή 0 (τα λεγόμενα bit) με τη χρήση μηχανικών διακοπτών. Αργότερα, χρησιμοποιήθηκαν οι **διάτρητες κάρτες** για την εισαγωγή εντολών σε γλώσσα χαμηλού επιπέδου (assembly).

Εξάλλου, στις αρχές της δεκαετίας του 1950 κατασκευάστηκε στα εργαστήρια της General Motors το **πρώτο λειτουργικό σύστημα** για τον υπολογιστή IBM 701.

Ετσι, όλα γενικά τα συστήματα αυτής της δεκαετίας ήταν ενός χρήστη και μιας εργασίας. Όμως, τα πρώτα συστήματα **ταυτόχρονης εκτέλεσης πολλών εργασιών** κατασκευάστηκαν στη δεκαετία του 1960. Το έτος 1964 η IBM κυκλοφόρησε τη σειρά υπολογιστών 360, που χρησιμοποιούσε ως λειτουργικό σύστημα το OS/360.

Αμέσως, μετά την εξέλιξη του IBM 360 σε IBM 370, άρχισαν να αναπτύσσονται τα **συστήματα πολλών χρηστών** και παράλληλα τα **συστήματα πραγματικού χρόνου**, που χαρακτηρίζονται από την άμεση μετάδοση των δεδομένων σε όλους τους συνδεδεμένους χρήστες. Παραδείγματα τέτοιων λειτουργικών συστημάτων είναι το CTSS το TSS και το CP/CMS.

Εξάλλου, ένα χαρακτηριστικό παράδειγμα συστήματος εξυπηρέτησης πολλών χρηστών ήταν το Multics, το οποίο όμως κρίθηκε πολύπλοκο για τον τελικό χρήστη. Οπότε, στη δεκαετία του 1970 έγινε προσπάθεια από τους Thomson και Ritchie, να ξαναγραφτεί το εν λόγω σύστημα με το όνομα Unix σε έναν υπολογιστή DEC.

Όμως όσο παράξενο κι αν φαίνεται, είναι γεγονός ότι η **εμπειρία** που είχε ο Thomson γράφοντας τον κώδικα του παιγνιδιού Space Travel τον βοήθησε στην ολοκλήρωση του Unix. Για να γραφτεί το Unix ο Thomson ανέπτυξε τη γλώσσα C. Ετσι λοιπόν, το Unix ήταν το πρώτο λειτουργικό σύστημα που δε γράφτηκε σε γλώσσα assembly. Σήμερα, υπάρχουν πολλές εκδόσεις του, όπως το Linux, το SCO, το INTERACTIVE, οι οποίες φυσικά είναι πολύ ανώτερες από τις πρώτες εκδόσεις. Το **μεγάλο πλεονέκτημα** του Unix είναι ότι μπορεί να χρησιμοποιηθεί σε διαφορετικούς τύπους υπολογιστών.

Τα λειτουργικά συστήματα μέχρι και τη δεκαετία του 1970, ήταν τεράστια από πλευράς κώδικα, είχαν πολλά λάθη και ήταν πολύ δύσκολη η συντήρησή τους. Οπότε, για να βρεθεί και να διορθωθεί ένα μικρό λάθος

έπρεπε να δαπανηθούν ατέλειωτες ώρες εργασίας και τεράστια ποσά. Δηλαδή, με λίγα λόγια αυτά δεν ήταν αξιόπιστα λειτουργικά συστήματα. Έτσι, στις αρχές της δεκαετίας του 1970, άρχισαν να αυξάνονται οι επικοινωνίες μεταξύ συστημάτων υπολογιστών και μάλιστα ύστερα από την υιοθέτηση του πρωτοκόλλου επικοινωνίας TCP/IP. Παράλληλα, προέκυψαν και προβλήματα ασφάλειας και κωδικοποίησης των δεδομένων.

Το έτος 1980 είναι η εποχή των προσωπικών υπολογιστών, το δε Αύγουστο του έτους 1981 κυκλοφορεί η πρώτη έκδοση 1.0 του MS-DOS που είναι γραμμένο σε γλώσσα assembly και αποτελείται από 4000 γραμμές κώδικα. Το έτος 1983 κυκλοφορεί η έκδοση 2.0 του MS-DOS για τον πρώτο προσωπικό υπολογιστή της IBM, που τρέχει κάνοντας χρήση του σκληρού δίσκου τον PC XT.

Το έτος 1984 κυκλοφορεί η έκδοση 3.0 του MS-DOS για τον IBM PC AT που χρησιμοποιεί τον επεξεργαστή 80286. Έτσι, το έτος 1989 έφτασε η σειρά της έκδοσης 4.0 του MS-DOS που παρέχει πολύ υποτυπώδεις δυνατότητες ταυτόχρονης εξυπηρέτησης πολλών χρηστών και πολλών προγραμμάτων. Όταν όμως κυκλοφόρησε ο επεξεργαστής 80386, τότε η IBM και η Microsoft συνεργάστηκαν για την κατασκευή ενός λειτουργικού συστήματος που θα μπορούσε να εκμεταλλευτεί τις δυνατότητες του 80386. Έτσι λοιπόν προέκυψε, το OS/2 το οποίο υποστηρίζει την παράλληλη εκτέλεση πολλών προγραμμάτων σε ένα περιβάλλον ενός χρήστη.

1.1.2. Η έννοια του λειτουργικού συστήματος

Είναι γνωστό ότι κατά περιόδους έχουν δοθεί διάφοροι ορισμοί που αφορούν στην έννοια του **λειτουργικού συστήματος**, οι οποίοι όμως θεωρήθηκαν ως **επαρκείς** ή **ανεπαρκείς** ανάλογα με τα επιστημονικά δεδομένα της εποχής. Για παράδειγμα, το έτος 1960 θα έλεγαν ότι το λειτουργικό σύστημα είναι το βασικό πρόγραμμα που ελέγχει τη λειτουργία των μονάδων ενός υπολογιστικού συστήματος. Με τον **όρο μονάδες** ενός υπολογιστικού συστήματος ορίζονται όλα τα τμήματα που απαρτίζουν ένα τέτοιο σύστημα, όπως είναι η κεντρική μονάδα επεξεργασίας, η μνήμη, οι συσκευές εισόδου-εξόδου, το σύστημα διαδρόμων, αλλά και οι περιφερειακές συσκευές όπως είναι ο εκτυπωτής, το scaner κ.λπ. Ο γενικά αποδεκτός ορισμός του λειτουργικού συστήματος θα μπορούσε να είναι ο εξής:

"Λειτουργικό σύστημα είναι ένα σύνολο από προγράμματα που ρυθμίζουν αποτελεσματικά τη διαχείριση της κεντρικής μονάδας επεξεργασίας, του μηχανισμού αποθήκευσης, των συσκευών εισόδου εξόδου και των συσκευών επικοινωνίας, έτσι ώστε να επιτυγχάνεται η υψηλότερη δυνατή απόδοση του υπολογιστικού συστήματος" (Deitel, 1990).

Το λειτουργικό σύστημα ρυθμίζει τον τρόπο αλληλεπίδρασης του

υπολογιστή με το χρήστη, ταυτόχρονα όμως επιτρέπει σε πολλούς χρήστες να μοιράζονται με αποτελεσματικότητα και ασφάλεια τις μονάδες του υπολογιστικού συστήματος, καθώς και προγράμματα λογισμικό και δεδομένα. Τέλος, παρέχει τη δυνατότητα διόρθωσης λαθών που θα μπορούσαν να αποβούν μοιραία για ένα υπολογιστικό σύστημα.

1.1.3. Είδη λειτουργικών συστημάτων

Είναι γνωστό ότι στις ημέρες μας υπάρχει μια τεράστια πληθώρα από υπολογιστικά συστήματα. Οπότε, κάθε είδος υπολογιστικού συστήματος συνοδεύεται κι' από το αντίστοιχο λειτουργικό σύστημα.

Έτσι, έχουμε υπολογιστικά συστήματα ενός χρήστη (single user), όπου γίνεται χρήση ενός μικρού ηλεκτρονικού υπολογιστή (PC) με φθηνό σχετικά επεξεργαστή (CPU). Υπάρχουν όμως και συστήματα πολλαπλών χρηστών (multi user), όπου ένας **κεντρικός υπολογιστής** εξυπηρετεί συνήθως πολλές θέσεις εργασίας, οι οποίες ονομάζονται τερματικά. Η όλη επεξεργασία των δεδομένων γίνεται στον κεντρικό υπολογιστή και τα τερματικά χρησιμεύουν μόνο για τη μεταφορά των στοιχείων από και προς αυτόν.

Εξάλλου, υπάρχουν τα **τοπικά δίκτυα** υπολογιστών (Local Area Networks) γνωστά και με τα αρχικά **L.A.N**, όπου πολλοί μικροί υπολογιστές βρίσκονται σχετικά κοντά ο ένας με τον άλλον και είναι συνδεδεμένοι για να μπορούν να επικοινωνούν μεταξύ τους. Επίσης, υπάρχουν τα **δίκτυα ευρείας περιοχής** (Wide Area Networks), για τα οποία χρησιμοποιούνται τα αρχικά **W.A.N**, στα οποία οι υπολογιστές που είναι συνδεδεμένοι μεταξύ τους βρίσκονται σε μεγάλη απόσταση ο ένας από τον άλλο. Τέλος, υπάρχουν οι **υπερυπολογιστές** που χρησιμοποιούν δεκάδες επεξεργαστές και έχουν δική τους ξεχωριστή αρχιτεκτονική, καθώς και τα **συστήματα πραγματικού χρόνου**.

Αυτά τα πολλά είδη συστημάτων (όχι μόνο σε δυνατότητες και σε ταχύτητα εκτέλεσης, αλλά και σε κατηγορίες εφαρμογών) απαιτούν τη χρήση διαφορετικών προγραμμάτων υποστήριξης.

Έτσι λοιπόν για παράδειγμα, ανάλογα με το αν το υπολογιστικό σύστημα που χρησιμοποιείται είναι ενός χρήστη ή πολλών χρηστών, γίνεται παράλληλα χρήση και του αντίστοιχου λειτουργικού συστήματος.

Πέρα από αυτά όμως, υπάρχουν λειτουργικά συστήματα που επιτρέπουν την εκτέλεση μόνο μιας εργασίας κάθε φορά και λέγονται λειτουργικά συστήματα **μοναδικού καθήκοντος** (single tasking) και αυτά που επιτρέπουν την εκτέλεση πολλών εργασιών ταυτόχρονα και λέγονται λειτουργικά συστήματα **πολλαπλών καθηκόντων** (multi tasking), (Milenkovic, 1987).

Ο παρακάτω πίνακας 1 αποτελεί μια καταγραφή των κυριοτέρων μέχρι σήμερα χρησιμοποιούμενων λειτουργικών συστημάτων με τον αντίστοιχο χαρακτηρισμό τους.

Πίνακας 1. *Χρησιμοποιούμενα λειτουργικά συστήματα.*

Όνομα λειτουργικού συστήματος	Χαρακτηρισμός	Χαρακτηρισμός
CP/M	Ενός χρήστη	Μιας εργασίας
DOS	Ενός χρήστη	Μιας εργασίας
Amiga DOS	Ενός χρήστη	Μιας εργασίας
MS-DOS V.3.3	Ενός χρήστη	Μιας εργασίας
MS-DOS V.4.5	Ενός χρήστη	Πολλών εργασιών
OS/2	Ενός χρήστη	Πολλών εργασιών
XENIX	Πολλών χρηστών	Πολλών εργασιών
UNIX	Πολλών χρηστών	Πολλών εργασιών
OS/400	Πολλών χρηστών	Πολλών εργασιών
NOVELL	Πολλών χρηστών	Μιας εργασίας
DOS VSE/SP	Πολλών χρηστών	Πολλών εργασιών

1.1.4. Σύγχρονες τάσεις των λειτουργικών συστημάτων

Η κατανόηση των σύγχρονων τάσεων των λειτουργικών συστημάτων, μας επιβάλλει να κάνουμε μια περιληπτική περιγραφή των κυριότερων εξελίξεων της πληροφορικής μέσα στη δεκαετία του 1990, με σκοπό να καταδείξουμε την πορεία μεταβολής που ακολουθούν τα λειτουργικά συστήματα στην εποχή μας. Άλλωστε, ύστερα από μερικά χρόνια θα είμαστε σε θέση να κάνουμε μια αξιολόγηση και έναν απολογισμό πάνω στον τρόπο αλλαγής της τεχνολογίας και τεχνογνωσίας των υπολογιστικών συστημάτων κατά την τελευταία δεκαετία.

Στην προκείμενη περίπτωση έχει δοθεί μεγάλη έμφαση στα **συστήματα πραγματικού χρόνου** (real time systems), στα **κατανεμημένα συστήματα** (distributed systems) στην **ασφάλεια** (security) στην **παράλληλη διαδικασία υπολογισμού** (parallel computation), καθώς και στα **ανοιχτά συστήματα** (open systems). Τα εν λόγω συστήματα αποτελούν τις σύγχρονες τάσεις εξέλιξης της επιστήμης των Η/Υ.

Πριν από μερικά χρόνια ήταν συνηθισμένο φαινόμενο το 90% από τους υπολογισμούς μιας εταιρείας να γίνονται από δυο **μέχρι τρεις**

κεντρικούς υπολογιστές, που προφανώς εργάζονταν 24 ώρες το εικοσιτετράωρο.

Σήμερα όμως, η επεξεργασία των δεδομένων ενός οργανισμού, γίνεται από πολλούς (ολοένα και αυξανόμενους) **σταθμούς εργασίας**, από τους οποίους οι περισσότεροι είναι ανενεργοί για ένα μεγάλο χρονικό διάστημα. Ετσι λοιπόν, οι υπολογισμοί κατανεμήθηκαν, οπότε και τα συστήματα έγιναν κατανεμημένα συστήματα.

Από τις αρχές ήδη της δεκαετίας του 1990 οδηγούμαστε σε έναν κόσμο που κατευθύνεται από διεθνώς καθιερωμένα πρότυπα επικοινωνίας και υπολογισμών. Ο κόσμος αυτός είναι αναγκασμένος να υιοθετήσει την παγκόσμια αποδεκτή αρχιτεκτονική των ανοιχτών συστημάτων, όπου όλες οι αρχιτεκτονικές είναι ευρέως διαθέσιμες σε οποιοδήποτε μέρος του πλανήτη, παρέχονται δε από πολλές πηγές και στην περίπτωση που δεν είναι δωρεάν διατίθενται σε πολύ χαμηλό κόστος. Κατ' αυτό τον τρόπο λοιπόν υιοθετούνται ολοένα και περισσότερο τα ανοιχτά χαρακτηριστικά των διεθνών επικοινωνιών. Τέτοια είναι το μοντέλο OSI που ήδη χρησιμοποιείται στην Ευρώπη, στις ΗΠΑ, και στην Ιαπωνία, τα ανοιχτά πρότυπα λειτουργικών συστημάτων, όπως το Unix, και τα ανοιχτά πρότυπα αλληλεπίδρασης υπολογιστή-χρήστη, όπως τα X-windows (Silberschatz και άλλοι, 1991).

Ο προσανατολισμός των σχεδιαστών υπολογιστικών συστημάτων μέχρι σήμερα περιορίζονταν στην κατασκευή υπολογιστών με ένα μοναδικό επεξεργαστή. Ο απώτερος στόχος για την επίτευξη υψηλής ταχύτητας ήταν η κατασκευή επεξεργαστών που θα μπορούσαν να εκτελούν όσο το δυνατόν περισσότερες εντολές ανά δευτερόλεπτο. Το πρόβλημα όμως είναι ότι έχουμε φτάσει στα όρια της ισχύος επεξεργασίας των σημερινών συστημάτων που κάνουν χρήση ενός επεξεργαστή. Δηλαδή, έχουμε αρχίσει να πλησιάζουμε τα φυσικά όρια των σύγχρονων ηλεκτρονικών συσκευών.

Μια λύση στο πρόβλημα θα ήταν η χρήση νέων αρχιτεκτονικών, όπου θα χρησιμοποιούνταν πολλοί επεξεργαστές που θα εργάζονταν παράλληλα (ακολουθιακά). Οπότε, σιγά μεν αλλά σταθερά θα οδηγούμαστε στην **παράλληλη διαδικασία** εκτέλεσης υπολογισμών.

Πέρα από αυτά όμως, πολλά προβλήματα απομένουν άλυτα μέχρι, να φθάσουμε πραγματικά στην παράλληλη επεξεργασία. Έτσι, στην προκείμενη περίπτωση διερωτάται κανείς, πως θα καθοριστεί αυτός ο παραλληλισμός; Άραγε θα είναι τόσο εύκολο και προφανές για τον υπολογιστή να ανιχνεύει ποιες εργασίες θα εκτελεστούν παράλληλα και να τις αναθέτει στον κατάλληλο επεξεργαστή; Μήπως όμως θα ήταν καλύτερο αυτό το καθήκον να το αναλάβει ένας προγραμματιστής; Πως θα αποφασίζεται ποιές εργασίες θα τεμαχίζονται σε επιμέρους καθήκοντα

και ποιες επιμέρους εργασίες θα πηγαίνουν σε ποιους επεξεργαστές; Τι θα γίνεται αν κάποια επιμέρους εργασία απασχολεί τον επεξεργαστή που χρειάζεται κάποιος χρήστης του συστήματος; Παρά το γεγονός ότι κάτω από αυτές τις συνθήκες τα προβλήματα που παραμένουν είναι ακόμα πολλά, το μέλλον των συστημάτων υπολογιστών και των λειτουργικών συστημάτων είναι προσανατολισμένο προς την κατεύθυνση του παράλληλου υπολογισμού.

Έτσι λοιπόν στο μέλλον, όλα τα εν λόγω προβλήματα θα λυθούν με τη σχεδίαση νέων λειτουργικών συστημάτων που θα συντονίζουν όλη τη διαδικασία της παράλληλης επεξεργασίας.

Η επανάσταση που ήδη έρχεται είναι η υιοθέτηση συστημάτων αναγνώρισης φωνής που θα φέρει το τέλος στη χρήση των πληκτρολογίων και θα επιτρέπει στους χρήστες να απευθύνονται στους υπολογιστές με προφορικό λόγο. Εξάλλου, πολλά τέτοια συστήματα λογισμικού έχουν ήδη αναπτυχθεί σε διάφορα πανεπιστήμια, όπως στο CUED του Cambridge, στο CNRS-LIMSI της Γαλλίας, στο Kalsruhe University της Γερμανίας, και στο Carnegie Mellon των ΗΠΑ.

Παράλληλα, πολλές εταιρείες έχουν κατασκευάσει το δικό τους λογισμικό στον τομέα της αναγνώρισης φωνής. Για παράδειγμα, η IBM έχει κυκλοφορήσει στην αγορά το "voice type dictation" και η Dragon Systems Inc. το "dragon dictate" με λεξικό 30.000 αγγλικών λέξεων.

Είναι δεδομένο ότι τα μελλοντικά λειτουργικά συστήματα θα είναι προσαρμοσμένα στις σύγχρονες τάσεις που ήδη περιγράφηκαν. Πάντως, είναι γεγονός ότι οι προγραμματιστές που θα γράψουν τον κώδικα των μελλοντικών λειτουργικών συστημάτων, θα πρέπει να λάβουν σοβαρά υπόψη τους τις **διάφορες παραμέτρους** που καθορίζουν τον τρόπο συμπεριφοράς των λειτουργικών συστημάτων.

Οι παράμετροι αυτοί είναι: **Πρώτο** ότι τα οποιαδήποτε συστήματα βασίζονται σε λεπτές ισορροπίες. **Δεύτερο** θα πρέπει να αναζητηθεί αν είναι θεμιτή η παραγωγή συστημάτων που μπορεί να αποτύχουν και να καθοριστεί το επίπεδο αποτυχίας που θεωρείται ανεκτό. **Τρίτο** θα πρέπει να δοθεί έμφαση στην ανάπτυξη συστημάτων, των οποίων η ποιότητα και η ορθή λειτουργία τους θα είναι άμεσα ανιχνεύσιμες.

ΚΕΦΑΛΑΙΟ 2
Δομή σύγχρονων λειτουργικών συστημάτων και διαχείριση της κεντρικής μονάδας επεξεργασίας

2.1. Γενικά

Τα σύγχρονα λειτουργικά συστήματα χωρίζονται σε τρία βασικά τμήματα. Το **πρώτο τμήμα** είναι ο διερμηνέας εντολών (το λεγόμενο shell),το **δεύτερο τμήμα** είναι το σύστημα αρχείων (file system) και το **τρίτο τμήμα** είναι ο πυρήνας του λειτουργικού συστήματος ή αλλιώς KERNEL.

Ο πυρήνας φορτώνεται πρώτος στη μνήμη όταν φορτώνεται το λειτουργικό σύστημα και μένει εκεί συνέχεια. Χωρίς τον πυρήνα δεν είναι δυνατόν να λειτουργήσει ο υπολογιστής. Ο πυρήνας διαχειρίζεται τη μνήμη, και κάνει κυρίως κατανομή της μνήμης στους χρήστες και στα προγράμματα, παράλληλα δε παρέχει προστασία στα περιεχόμενα της μνήμης.

Επίσης, διαχειρίζεται τη λειτουργία των συσκευών εισόδου-εξόδου, αλλά και των διεργασιών σε συστήματα πολυεπεξεργασίας.

Η καλύτερη κατανόηση των σύγχρονων λειτουργικών συστημάτων επιβάλλει να γίνει αναλυτική επεξήγηση της έννοιας της διεργασίας, καθώς και του τρόπου με τον οποίο γίνεται η διαχείριση των διεργασιών.

2.1.1. Η έννοια της διεργασίας

Προτού αναλύσουμε σε βάθος τη βασική δομή και τις αρχές των λειτουργικών συστημάτων, είναι απαραίτητο να εξετάσουμε και να ορίσουμε την έννοια της διεργασίας (proccess), η οποία παίζει τον κυριότερο ρόλο στον τομέα αυτό.

Είναι γνωστό ότι επίσημα δεν υπάρχει γενικά αποδεκτός ορισμός για το τι είναι διεργασία, εντούτοις όμως δε θα ήταν λάθος να ισχυριστεί κανείς

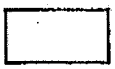
ότι η διεργασία είναι το μικρότερο τμήμα ενός προγράμματος που βρίσκεται σε εκτέλεση.

Για παράδειγμα, η εντολή **ls** στο Unix που χρησιμοποιείται (όπως και η εντολή **dir** στο MS-DOS) για να δείχνει τα περιεχόμενα μιας δισκέτας ή ενός καταλόγου είναι μια διεργασία. Κάθε διεργασία είναι συσχετισμένη με διάφορους παράγοντες, από τους οποίους οι σημαντικότεροι είναι η **κατάστασή** της, ο **κωδικός αριθμός** της διεργασίας (process id) και η **προτεραιότητά** της.

Μια διεργασία μπορεί να είναι σε κατάσταση τρέχουσα (running), έτοιμη (ready), αναμένουσα (blocked), αναστελλόμενη (suspended). Η μετάβασή της από τη μια κατάσταση στην άλλη γίνεται όταν λήξει ο χρόνος εκτέλεσής της ή όταν έλθει η σειρά της.

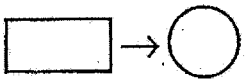
2.1.2. Αναπαράσταση των διεργασιών

Οι Dietel (1990) και Stallings (1992) επεξηγούν τον τρόπο με τον οποίο γίνεται η διαγραμματική αναπαράσταση των διεργασιών και της αναζήτησης από αυτές των μονάδων του συστήματος. Αυτά λέγονται γραφήματα κατανομής μονάδων του συστήματος. Τέτοια γραφήματα είναι τα εξής:

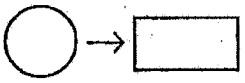


Με το σχήμα αυτό παριστάνεται μια διεργασία.

Με το σχήμα αυτό παριστάνεται μια μονάδα του συστήματος.



Με το σχήμα αυτό παριστάνεται μια διεργασία που απαιτεί μια από τις μονάδες του συστήματος.



Με το σχήμα αυτό παριστάνεται μια μονάδα του συστήματος που έχει ανατεθεί σε μια διεργασία.

Όταν υπάρχουν πολλές διεργασίες που πρέπει να εκτελεστούν, τότε υπάρχει ένα τμήμα του λειτουργικού συστήματος που πρέπει να διαχειρίζεται τη σειρά εκτέλεσής τους και να ρυθμίζει τις προτεραιότητες. Για παράδειγμα, δεν μπορεί μια διεργασία που θέλει 5 min για να εκτελεστεί, να μπλοκάρει όλες τις άλλες διεργασίες, οι οποίες περιμένουν την εκτέλεσή της. Υπάρχει λοιπόν ένα σύστημα διακανονισμού του τρεξίματος των διεργασιών.

2.1.3. Διαχείριση διεργασιών και κεντρικής μονάδας επεξεργασίας

Το κυριότερο πρόβλημα που μπορεί να προκύψει σε ένα σύστημα πολλαπλών διεργασιών είναι το **αδιέξοδο** (deadlock). Ένα σύστημα φθάνει σε ένα τέτοιο αδιέξοδο όταν υπάρχουν δυο ή και περισσότερες διεργασίες, οι οποίες πρέπει να χρησιμοποιήσουν την ίδια μονάδα του συστήματος (για παράδειγμα τον επεξεργαστή), για να ολοκληρωθούν.

Το αποτέλεσμα αυτής της περιπτώσεως είναι ότι κάθε μια διεργασία είναι δυνατόν να περιμένει επ' άπειρον τη μονάδα του συστήματος γιατί αυτή δεν είναι διαθέσιμη. Έτσι, το πρόγραμμα που εκτελείται δε φθάνει ποτέ στο τέλος του.

Η κατανόηση του προβλήματος του αδιεξόδου μας προτρέπει στο να αναλύσουμε τις συνθήκες που οδηγούν σε αδιέξοδο και το χειρισμό τους. Έτσι, η εμπέδοση της ανάλυσης του εν λόγω προβλήματος μας δίνει τη δυνατότητα αποφυγής δημιουργίας αδιεξόδων σε ένα λειτουργικό σύστημα.

(i) Συνθήκες που οδηγούν σε αδιέξοδο

Είναι γνωστό ότι υπάρχουν κάποιες συνθήκες, κάτω από τις οποίες ευνοείται η εμφάνιση αδιεξόδου σε ένα σύστημα. Οι κυριότερες όμως συνθήκες που οδηγούν το λειτουργικό σύστημα σε αδιέξοδο είναι η συνθήκη **του αμοιβαίου αποκλεισμού**, η συνθήκη της **αναμονής**, η συνθήκη της **μη αναγκαστικής εγκατάλειψης** και η συνθήκη της **κυκλικής αναμονής**.

Οι τρεις πρώτες συνθήκες είναι αναγκαίες για την ύπαρξη αδιεξόδου, αλλά όχι όμως και ικανές. Η τέταρτη συνθήκη είναι άμεση συνέπεια των τριών πρώτων. Το πρόβλημα της κυκλικής αναμονής είναι άλυτο και έχει μοιραία κατάληξη για το σύστημα. Ετσι λοιπόν, όταν σε ένα σύστημα πολλαπλών διεργασιών συνυπάρχουν οι εν λόγω τέσσερεις συνθήκες, τότε αυτές σαν μια ενότητα αποτελούν την ικανή και αναγκαία συνθήκη δημιουργίας μη επιλύσιμου αδιεξόδου.

Κάτω από αυτές τις συνθήκες, για την ανάλυση του προβλήματος του αδιεξόδου, επιβάλλεται να γίνει μια περιληπτική περιγραφή των τεσσάρων αυτών καταστάσεων.

(i_α) Η συνθήκη του αμοιβαίου αποκλεισμού.

Η συνθήκη του αμοιβαίου αποκλεισμού είναι μια περίπτωση κατά την οποία, κάθε μια από τις μονάδες του συστήματος (για παράδειγμα εκτυπωτής, ή κεντρική μονάδα επεξεργασίας) είτε είναι απασχολημένη από μια ακριβώς διεργασία, είτε είναι διαθέσιμη. Δηλαδή, τα γεγονότα είναι αμοιβαίως αποκλειόμενα (mutually exclusive), (Hofri, 1990).

(i_β) Η συνθήκη της αναμονής

Η συνθήκη αναμονής είναι η κατάσταση εκείνη με βάση την οποία,

διεργασίες (οι οποίες σε κάποια στιγμή κρατούν απασχολημένες μερικές από τις μονάδες του συστήματος που τους έχουν παραχωρηθεί) απαιτούν να τους δοθούν και νέες μονάδες του συστήματος.

($i_γ$) Η συνθήκη μη αναγκαστικής εγκατάλειψης

Όπως φαίνεται και από το όνομα της συνθήκης αυτής, οι μονάδες του συστήματος οι οποίες έχουν ήδη δοθεί σε κάποιες διεργασίες δεν μπορούν με τη βία να αποδεσμευτούν από αυτές. Επομένως, πρέπει να αφεθούν ελεύθερες από τη διεργασία που τις έχει δεσμεύσει.

($i_δ$) Η συνθήκη της κυκλικής αναμονής

Η συνθήκη της κυκλικής αναμονής, εμφανίζεται όταν υπάρχει μια κυκλική αλυσίδα από δυο ή περισσότερες διεργασίες κάθε μια από τις οποίες κρατά μια ή περισσότερες από τις μονάδες του συστήματος που απαιτούνται από την επόμενη διεργασία της αλυσίδας.

(ii) Χειρισμός των αδιεξόδων

Οι καταστάσεις των αδιεξόδων (όπως και όλα τα προβλήματα) έχουν τις δικές τους πρακτικές αντιμετώπισης και επίλυσης. Όλα εξαρτώνται από την οπτική γωνία θεώρησής τους. Θεωρητικά, μια αδιέξοδη κατάσταση σε ένα υπολογιστικό σύστημα μπορεί να **αγνοηθεί εντελώς**, μπορεί όμως να **ανιχνευθεί και να αντιμετωπιστεί** ή ακόμα καλύτερα μπορεί να **προληφθεί** με την αποφυγή μιας από τις τέσσερις συνθήκες αδιεξόδων που περιγράφηκαν πριν. Το καλύτερο βέβαια θα ήταν μια **δυναμική αποφυγή** του προβλήματος με προσεκτική κατανομή των μονάδων του συστήματος.

Φυσικά, η πρώτη εκδοχή της αγνόησης του προβλήματος, θα ήταν αρκετά επισφαλής και θα μας οδηγούσε σε ασταθή συστήματα. Οι δυο τελευταίες μέθοδοι που αφορούν στην πρόληψη, φαίνεται ότι είναι οι καλύτερες και ασφαλέστερες. Επειδή όμως, πάντα υπάρχει η πιθανότητα να προκύψει ένα αδιέξοδο καλό θα ήταν να χρησιμοποιούνται αλγόριθμοι ανίχνευσης και αντιμετώπισης. Έτσι, επιβάλλεται να αναπτυχθούν τέτοιοι αλγόριθμοι για μια αποτελεσματικότερη αντιμετώπιση των χειρισμών των αδιεξόδων.

($ii_α$) Ανίχνευση και αντιμετώπιση των αδιεξόδων

Η αρχή στην οποία βασίζεται η ανίχνευση και η αντιμετώπιση των αδιεξόδων είναι αυτή βάσει της οποίας το σύστημα **περιοδικά ελέγχει** τον πίνακα των διεργασιών, για να βρεθεί αν υπάρχει κάποια διεργασία, η οποία είναι συνεχώς αναμένουσα (blocked), για περισσότερο από μια χρονική περίοδο (για παράδειγμα, πολλά λεπτά της ώρας). Οπότε, μια τέτοια διεργασία ακυρώνεται. Η ακύρωση της γίνεται με τη χρήση της εντολής kill με τον κωδικό -9 και τον κωδικό αριθμό της διεργασίας **(kill-9 P.I.D)** στο

λειτουργικό σύστημα Unix.

Μια άλλη μέθοδος είναι αυτή με βάση την οποία, το σύστημα δεν κάνει τίποτα άλλο εκτός από το να παρακολουθεί τις απαιτήσεις και τις απελευθερώσεις διεργασιών. Κάθε φορά που μια μονάδα του συστήματος ζητείται ή απελευθερώνεται το γράφημα των μονάδων του συστήματος ενημερώνεται. Υστερα από αυτό, γίνεται ένας έλεγχος για να βρεθεί αν υπάρχει κάποιος κύκλος.

Όταν πράγματι υπάρχει κύκλος που θα οδηγεί σε αδιέξοδο, τότε μια από τις διεργασίες ακυρώνεται (σκοτώνεται kill -9 PID). Εξάλλου, όταν αυτό δεν άρει το αδιέξοδο σκοτώνεται μια άλλη διεργασία και αυτό συνεχίζεται μέχρι να σπάσει ο κύκλος

(ii$_\beta$) Πρόληψη των αδιεξόδων

Όταν κάθε μια από τις 4 συνθήκες που οδηγούν σε αδιέξοδο έχει αποκλειστεί από την αρχή, τότε το αδιέξοδο δεν μπορεί να προκύψει. Επομένως, μπορούν να τεθούν περιορισμοί στις διεργασίες έτσι ώστε να είναι αδύνατον να φτάσουμε σε αδιέξοδο.

Ο πρώτος περιορισμός που μπορεί **θεωρητικά** να τεθεί είναι η άρνηση της συνθήκης του αμοιβαίου αποκλεισμού. Στην πράξη όμως, δεν μπορεί να γίνει άρνηση αυτής της συνθήκης, γιατί θα προκληθούν χειρότερα προβλήματα. Επομένως, θα ήταν φυσικό η λύση να αναζητηθεί στην άρνηση της συνθήκης της αναμονής. Η λειτουργία του συστήματος σε αυτή την περίπτωση διέπεται από τον εξής **κανόνα:**

Κάθε διεργασία πρέπει να απαιτήσει όλες τις απαραίτητες γι' αυτήν μονάδες του συστήματος αμέσως και το σύστημα θα πρέπει να απαντήσει στη βάση της αρχής όλα ή τίποτα. Δηλαδή, πρέπει να παραχωρήσει όλες τις μονάδες ή τίποτα.

Φυσικά όπως κάθε μέθοδος έτσι και αυτή, παρουσιάζει βασικά μειονεκτήματα που δυσχεραίνουν τη λειτουργία του υπολογιστικού συστήματος. Πρώτον υπάρχει σοβαρή **σπατάλη μονάδων** του συστήματος και δεύτερον μπορεί το τελικό αποτέλεσμα να είναι η **επ' άπειρον αναβολή.** Δηλαδή όταν όλες οι μονάδες του συστήματος ζητηθούν (στη βάση "όλα ή τίποτα") και είναι αδύνατον να παραχωρηθούν όλες, τότε το σύστημα θα κολλήσει τελώντας σε αιώνια αναμονή.

Φυσικά όμως υπάρχει και άλλη λύση, όπως είναι αυτή με την οποία μπορεί να γίνει άρνηση της συνθήκης της μη αναγκαστικής εγκατάλειψης με βάση τον παρακάτω **κανόνα:**

Οταν μια διεργασία κρατάει δεσμευμένες κάποιες από τις μονάδες και οι επιπλέον απαιτήσεις της δε γίνονται δεκτές, τότε θα πρέπει να απελευθερώσει όλες τις μονάδες που είχε δεσμεύσει αρχικά και στο εξής

θα είναι απαραίτητο να τις ξαναζητήσει μαζί με τις επιπρόσθετες μονάδες (Stallings, 1992).

Φυσικά και αυτή η θεώρηση των πραγμάτων έχει πολύ σοβαρά μειονεκτήματα. Κατ' αρχήν η διεργασία θα έχει σοβαρές απώλειες όταν απελευθερώσει τις μονάδες που είχε δεσμεύσει αρχικά και υπάρχει πάντα ο κίνδυνος να οδηγηθούμε σε αναβολή επ' άπειρον.

Τέλος, η άρνηση της συνθήκης της κυκλικής αναμονής μπορεί να γίνει με βάση τους παρακάτω δυο κανόνες.

Κανόνας 1ος: Ο κανόνας αυτός ορίζει ότι **κάθε διεργασία δικαιούται** μόνο μια μονάδα του συστήματος κάθε φορά που εκτελείται. Όταν όμως χρειαστεί δεύτερη μονάδα, τότε θα πρέπει ν' απελευθερώσει την πρώτη μονάδα που κατέχει. Ο κανόνας αυτός είναι ανεπαρκής και αν εφαρμόζονταν μόνος του, θα μπορούσε να προκαλέσει μεγάλες καθυστερήσεις (Tanenbaum, 1993).

Κανόνας 2ος: Σύμφωνα με τον κανόνα αυτό **θα πρέπει να υιοθετηθεί** ένα σύστημα αριθμητικής ιεραρχίας των μονάδων. Κάθε μια από τις μονάδες του συστήματος έχει ένα αριθμό (π.χ. 1, 2, 3, 4,....). Όταν για παράδειγμα, μια διεργασία έχει δεσμεύσει τις μονάδες 5, 6, 7, τότε η εν λόγω διεργασία θα μπορεί να απαιτήσει μόνο τις μονάδες που έχουν μεγαλύτερο αριθμό από αυτήν δηλαδή από 8 και πάνω (Tanenbaum, 1993).

Παρόλα αυτά όμως, η λύση της άρνησης της συνθήκης της κυκλικής αναμονής, παρουσιάζει **μειονεκτήματα** που οφείλονται στο γεγονός ότι η προτεινόμενη λύση δεν είναι φιλική στο χρήστη, οδηγεί σε ασταθές περιβάλλον εργασίας και δεν μπορεί να μεταφερθεί εύκολα από το ένα υπολογιστικό σύστημα στο άλλο.

(Iiγ) Αποφυγή αδιεξόδων

Είναι προφανές ότι η εύρεση αλγορίθμων, αλλά και στρατηγικών αποφυγής των αδιεξόδων είναι ζωτικής σημασίας στον τομέα των λειτουργικών συστημάτων. Σύμφωνα με τον Tanenbaum (1990) ο Dijkstra στις αρχές της δεκαετίας του 1970, σχεδίασε έναν αλγόριθμο αποφυγής αδιεξόδων που είναι γνωστός σαν **αλγόριθμος του τραπεζίτη**. Πρόκειται για τον πιο διάσημο αλγόριθμο αποφυγής αδιεξόδων που έχει κατασκευαστεί ποτέ. Ονομάζεται αλγόριθμος του τραπεζίτη, γιατί έχει να κάνει με έναν τραπεζίτη, ο οποίος δίνει δάνεια και εισπράττει πληρωμές.

Φυσικά, στην πράξη γίνεται χρήση του αλγορίθμου από το λειτουργικό σύστημα, το οποίο παίζει και το ρόλο του τραπεζίτη, ενώ το ρόλο των δανειοληπτών μπορούν να παίξουν οι μονάδες του συστήματος.

Γενικά, διακρίνονται δυο μορφές σε οποιαδήποτε λειτουργούντα υπολογιστικά συστήματα. Αυτές είναι η **ασφαλής** και η **ασταθής** μορφή. Ασφαλής μορφή είναι αυτή στην οποία η κατάσταση όλων των μονάδων είναι τέτοια, ώστε όλες οι διεργασίες θα μπορέσουν να εκτελεστούν ως το τέλος. Αντίθετα, στην ασταθή μορφή μπορούμε να οδηγηθούμε ανά πάσα στιγμή σε αδιέξοδο. Ο παρακάτω πίνακας 2 επεξηγεί τα σύμβολα που χρησιμοποιούνται για την περιγραφή τριών καταστάσεων από τις οποίες η **δεύτερη είναι ασταθής** και οι **άλλες δυο είναι σταθερές**. Με τον όρο μονάδες του συστήματος εννοούμε όλες τις υπάρχουσες συσκευές ενός συστήματος υπολογιστών. Για παράδειγμα, μονάδες ενός συστήματος υπολογιστών είναι η κεντρική μονάδα επεξεργασίας, ο σκληρός δίσκος, η μνήμη, ο εκτυπωτής και άλλες πολλές.

Κάθε διεργασία απασχολεί κάποιες από τις μονάδες του συστήματος και ταυτόχρονα ζητάει να της διατεθούν κάποιες άλλες μονάδες επιπρόσθετα. Ο παρακάτω πίνακας 2 περιέχει επίσης την επεξήγηση των συμβόλων που θα χρησιμοποιηθούν στους πίνακες 3,4, και 5 στη σελίδα 18.

Πίνακας 2. *Επεξήγηση των χρησιμοποιούμενων συμβόλων.*

Ο συνολικός αριθμός μονάδων συστήματος	R
Ο συνολικός αριθμός διεργασιών	N
Ο αριθμός μονάδων που κατέχονται από την διεργασία i	LOAN(i)i=1,2,...N
Ο μέγιστος αριθμός μονάδων που απαιτούνται από την διεργασία i	MAX(i) i=1,2,...N
Ο αριθμός μονάδων που ζητάει η διεργασία i	CLAIM(i) i=1,2,...N
Ο αριθμός διαθέσιμων μονάδων	A

Εξάλλου, στους παρακάτω πίνακες 3, 4 και 5, στη σελίδα 18 περιγράφονται **τρεις καταστάσεις**, ως προς τον αριθμό των διεργασιών τους, τον αριθμό των μονάδων που κατέχει κάθε διεργασία, το μέγιστο αριθμό μονάδων που απαιτεί κάθε διεργασία και τον αριθμό των μονάδων που λείπει από κάθε διεργασία. Είναι φανερό ότι η **πρώτη** και η **τρίτη** κατάσταση είναι σταθερές. Στην πρώτη κατάσταση ο συνολικός αριθμός των μονάδων του συστήματος είναι 12. Οι δεσμευμένες μονάδες είναι 10 και οι ελεύθερες μονάδες είναι 2. Αυτές οι δυο μονάδες μπορούν να δοθούν στη δεύτερη διεργασία, η οποία αφού ολοκληρωθεί θα απελευθερώσει και

τις 6 μονάδες που κατέχει, οι οποίες είναι αρκετές για να ολοκληρωθούν οι διεργασίες 1 και 3. Το ίδιο ακριβώς ισχύει και στην κατάσταση 3.

Όμως, στην κατάσταση 2 υπάρχει μόνο μια διαθέσιμη μονάδα, η οποία όπου και αν δοθεί καμιά διεργασία δεν ολοκληρώνεται. Έτσι, η κατάσταση 2 είναι ασταθής και αν κάποια από τις διεργασίες δεν εγκαταλείψει όλες τις δεσμευμένες μονάδες της, τότε θα καταλήξουμε σε αδιέξοδο.

Πίνακας 3. *Κατάσταση 1 - Σταθερή.*

Διεργασία i	loan(i)	max(i)	claim(i)=max(i)-loan(i)
1	1	4	3
2	4	6	2
3	5	8	3
R	12	A	2

Πίνακας 4. *Κατάσταση 2 - Ασταθής.*

Διεργασία i	loan(i)	max(i)	claim(i)=max(i)-loan(i)
1	(2)	4	2
2	4	6	2
3	5	8	3
R	12	A	1

Πίνακας 5. *Κατάσταση 3 - Σταθερή.*

Διεργασία i	loan(i)	max(i)	claim(i)=max(i)-loan(i)
1	1	4	3
2	(5)	6	1
3	5	8	3
R	12	A	1

2.2. Διαχείριση κεντρικής μονάδας επεξεργασίας

Η επιτυχής ολοκλήρωση των διεργασιών ενός συστήματος υπολογιστών, επιτυγχάνεται μόνο διαμέσου της χρήσης της κεντρικής μονάδας επεξεργασίας από τις διεργασίες. Στην περίπτωση αυτή προκύπτει το εξής **σημαντικότατο πρόβλημα**. Όταν σε κάποια χρονική στιγμή περισσότερες από μια διεργασίες βρίσκονται υπό εκτέλεση, τότε φυσικά δεν μπορούν όλες τους να απασχολούν ταυτόχρονα την κεντρική μονάδα επεξεργασίας. Οπότε, τίθεται το ερώτημα. Πως θα επιτευχθεί μια δίκαιη μοιρασιά της κεντρικής μονάδας επεξεργασίας ανάμεσα στις διεργασίες, έτσι ώστε να ολοκληρωθούν όλες οι διεργασίες σε εύλογο χρονικό διάστημα;

Οι σχεδιαστές των λειτουργικών συστημάτων έδωσαν απάντηση σε αυτό το πρόβλημα, αναπτύσσοντας **ολοκληρωμένες μεθόδους** κατανομής της κεντρικής μονάδας επεξεργασίας. Αυτές οι μέθοδοι χωρίζονται σε **τρεις κατηγορίες** που είναι οι εξής: α) μέθοδοι κατανομής υψηλού επιπέδου, β) μέθοδοι κατανομής μεσαίου επιπέδου, και γ) μέθοδοι κατανομής χαμηλού επιπέδου.

α) Στις μεθόδους κατανομής των μονάδων του συστήματος, που ονομάζονται **υψηλού επιπέδου**, ρυθμίζεται ποιες από τις διεργασίες θα ανταγωνιστούν, για να αποκτήσουν τον έλεγχο κάποιων από τις μονάδες του συστήματος.

β) Στις μεθόδους κατανομής των μονάδων του συστήματος που λέγονται **μεσαίου επιπέδου**, ρυθμίζεται ποιες από τις διεργασίες θα ανταγωνιστούν, για τον έλεγχο της κεντρικής μονάδας επεξεργασίας.

γ) Στις μεθόδους κατανομής των μονάδων του συστήματος που λέγονται **χαμηλού επιπέδου**, ρυθμίζεται ποιες από τις διεργασίες και με ποια σειρά θα αποκτήσουν τον έλεγχο της κεντρικής μονάδας επεξεργασίας, όταν αυτή απελευθερωθεί.

2.2.1. Ρύθμιση της προτεραιότητας εκτέλεσης των διεργασιών

Σ' ένα σύστημα πολλών χρηστών ο πυρήνας δεσμεύει τον επεξεργαστή για λογαριασμό μιας διεργασίας και για ένα συγκεκριμένο χρονικό διάστημα που ονομάζεται **quantum**. Μόλις όμως το quantum περάσει, τότε αποδεσμεύεται ο επεξεργαστής από την τρέχουσα διεργασία και παραχωρείται σε μια άλλη. Παράλληλα, ο πυρήνας προγραμματίζει να συνεχίσει αργότερα την εκτέλεση της πρώτης διεργασίας.

Οι **βασικοί στόχοι** της ρύθμισης της προτεραιότητας εκτέλεσης των διεργασιών είναι η ύπαρξη δικαιοσύνης, η αποτελεσματικότητα, η σωστή χρήση και αξιοποίηση του χρόνου του επεξεργαστή, η ελαχιστοποίηση του χρόνου απόκρισης, και ανταπόδοσης, καθώς και η **μεγιστοποίηση** του αριθμού των διεργασιών που θα εκτελεστούν στη μονάδα του χρόνου.

Με τον όρο **δικαιοσύνη** εννοούμε την εξασφάλιση της δίκαιης μοιρασιάς του επεξεργαστή για όλες τις διεργασίες, έτσι ώστε καμιά να μην περιμένει για πάντα. Με τον όρο **αποτελεσματικότητα** εννοούμε την επίτευξη της γρήγορης ολοκλήρωσης όλων των διεργασιών. Σε ιδανικές συνθήκες επιτυγχάνεται συνδυασμός δικαιοσύνης και αποτελεσματικότητας, δηλαδή όλες οι διεργασίες ολοκληρώνονται γρήγορα χωρίς όμως να αδικείται καμιά. Επίσης, ο **χρόνος απόκρισης** είναι ο χρόνος που μεσολαβεί ανάμεσα στην είσοδο μιας εντολής από τον χρήστη και στην έναρξη εκτέλεσης αυτής της εντολής, ενώ ο όρος **χρόνος ανταπόδοσης** είναι ο χρόνος που πρέπει να περιμένουν οι χρήστες για την έκδοση αποτελεσμάτων.

Η **ρύθμιση της προτεραιότητας** εκτέλεσης των διεργασιών γίνεται με τη **χρησιμοποίηση ειδικών αλγορίθμων**, οι οποίοι συμβάλλουν στην αποτελεσματικότερη ανταπόκριση των λειτουργικών συστημάτων στις απαιτήσεις των χρηστών. Αυτοί οι αλγόριθμοι αξιολογούνται με ειδικά για το σκοπό αυτό κριτήρια.

(i) Αλγόριθμοι ρύθμισης προτεραιότητας διεργασιών

Οι εν λόγω αλγόριθμοι αναπτύχθηκαν από τους σχεδιαστές των λειτουργικών συστημάτων, για να ρυθμίσουν την προτεραιότητα απασχόλησης της κεντρικής μονάδας επεξεργασίας.

Οι **αλγόριθμοι ρύθμισης** της προτεραιότητας των διεργασιών χωρίζονται σε **δύο κύριες κατηγορίες**, ανάλογα με το αν οι διεργασίες, α) αναγκάζονται κάποια στιγμή να αφήσουν τον έλεγχο του επεξεργαστή, και β) αν οι διεργασίες διατηρούν τον έλεγχο του επεξεργαστή μέχρι το τέλος.

Οι αλγόριθμοι της πρώτης κατηγορίας λέγονται **αλγόριθμοι αναγκαστικής εγκατάλειψης** και της δεύτερης λέγονται **αλγόριθμοι μη αναγκαστικής εγκατάλειψης**. Ετσι, η κατανόηση εκτέλεσης των διάφορων διεργασιών επιβάλλει την αναφορά και ανάλυση των υπόψη αλγορίθμων, πράγμα που συμβάλλει στην αρτιότερη κατάρτιση του χρήστη υπολογιστικών συστημάτων.

(i_a) Αλγόριθμοι αναγκαστικής εγκατάλειψης

Οι κυριότεροι αλγόριθμοι αναγκαστικής εγκατάλειψης είναι:

α) ο αλγόριθμος της **εξυπηρέτησης εκ περιτροπής** (round robin),
β) ο αλγόριθμος **απλής προτεραιότητας** (simple priority),
γ) ο αλγόριθμος του **ελάχιστου εναπομείναντος χρόνου** (Shortest Remaining Time - **S.R.T.**),
δ) Ο αλγόριθμος των προτεραιοτήτων **S.U.N. UNIX**, (**S.U.N.** είναι τα αρχικά του Stanford University Network)
ε) ο αλγόριθμος της **δίκαιης μοιρασιάς**, (fair share) που χρησιμοποιεί η AT & T, και
στ) ο αλγόριθμος του συστήματος **VAX / VMS**.

Οι εν λόγω αλγόριθμοι αναφέρονται περιληπτικά, πλην του αλγορίθμου της **εξυπηρέτησης εκ περιτροπής** και του αλγορίθμου **απλής προτεραιότητας**, που περιγράφονται αναλυτικά, γιατί είναι οι πλέον διαδεδομένοι και οι πιο συνηθισμένοι αλγόριθμοι υπολογιστικών συστημάτων.

(ia_1) Ο αλγόριθμος της εξυπηρέτησης εκ περιτροπής

Είναι ένας από τους πιο παλιούς, πιο απλούς, πιο δίκαιους αλγόριθμους, σήμερα δε θεωρείται ο πλέον χρησιμοποιούμενος αλγόριθμος. Ο

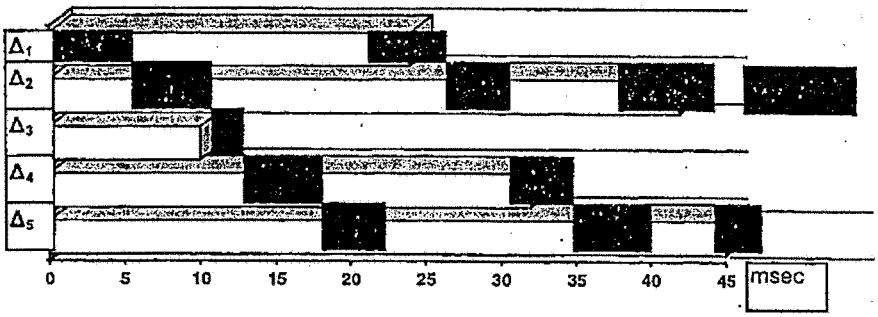
αλγόριθμος της **εξυπηρέτησης εκ περιτροπής** (round robin) δημιουργεί μια ουρά από διεργασίες, οι οποίες είναι όλες σε έτοιμη (ready) κατάσταση. Οποια διεργασία φτάνει πρώτη μπαίνει στην αρχή της ουράς και εκτελείται πρώτη, μόνο όμως για ένα συγκεκριμένο χρονικό διάστημα (quantum). Όταν η διεργασία είναι ακόμα σε τρέχουσα κατάσταση (running) και περάσει χρόνος ίσος με ένα quantum, τότε η διεργασία χάνει αναγκαστικά τον έλεγχο του επεξεργαστή, ο οποίος έλεγχος δίνεται στην αμέσως επόμενη διεργασία της ουράς. Η διεργασία που χάνει τον έλεγχο της κεντρικής μονάδας επεξεργασίας πηγαίνει στο τέλος της ουράς (Deitel, 1990).

Το ενδιαφέρον στον αλγόριθμο αυτό είναι η χρονική διάρκεια του quantum, το οποίο όταν είναι πολύ μικρό, τότε προκαλούνται πάρα πολλές εναλλαγές στη χρήση του επεξεργαστή σε σύντομο χρονικό διάστημα, κάτι που μειώνει την αποτελεσματικότητα. Όταν όμως το quantum είναι πολύ μεγάλο, τότε προκαλείται μεγάλη καθυστέρηση στο σύστημα.

Το παρακάτω γραφικό 2.1 στη σελίδα 22, αναφέρεται σε μια γραφική παράσταση εκτέλεσης του αλγορίθμου της εξυπηρέτησης εκ περιτροπής για πέντε (5) διεργασίες και με **quantum = 5 msec**, όπου 1 msec = 1/1000 sec.

Στην προκειμένη περίπτωση η διεργασία Δ_1 φθάνει τη στιγμή 0 m sec, η διεργασία Δ_2 φθάνει τη στιγμή 2 msec, η διεργασία Δ_3 φθάνει τη στιγμή 3 msec, η διεργασία Δ_4 φθάνει τη στιγμή 4 msec, και η διεργασία Δ_5 φθάνει τη στιγμή 5 msec. Ο απαιτούμενος χρόνος ολοκλήρωσής τους είναι για την Δ_1 10 msec, για την Δ_2 29 msec, για την Δ_3 3 msec, για την Δ_4 7 msec και τέλος για την Δ_5 12 msec.

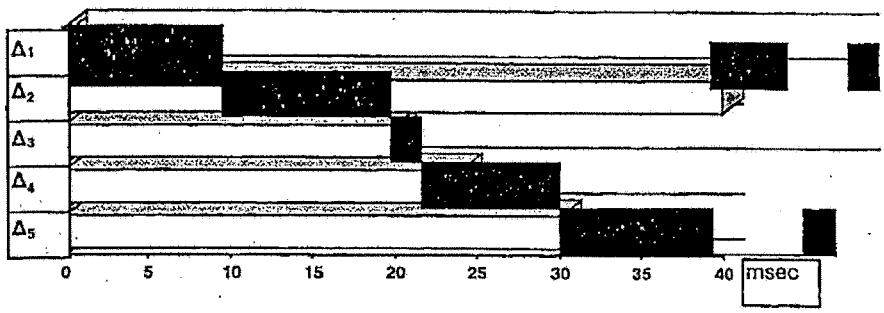
Έτσι, όλες οι εν λόγω διεργασίες εκτελούνται τμηματικά. Για παράδειγμα, η διεργασία Δ_1 αρχίζει να εκτελείται τη στιγμή 0 msec, αλλά τη στιγμή 5 msec διακόπτεται η εκτέλεσή της, χωρίς να έχει ολοκληρωθεί, γιατί έχει περάσει χρόνος ίσος με 1 quantum. Τότε η διεργασία Δ_1 πηγαίνει στο τέλος της ουράς. Τη στιγμή 2 msec έχει ήδη φθάσει η διεργασία Δ_2, η οποία περιμένει να έλθει η σειρά της. Έτσι λοιπόν τη στιγμή 5 msec, αρχίζει να εκτελείται η διεργασία Δ_2. Τη στιγμή 10 msec διακόπτεται και η διεργασία Δ_2 χωρίς να έχει ολοκληρωθεί, γιατί έχει περάσει χρόνος ίσος με 1 quantum. Τότε η διεργασία Δ_2 πηγαίνει στο τέλος της ουράς αμέσως μετά τη διεργασία Δ_1. Έτσι λοιπόν τη στιγμή 10 msec, αρχίζει να εκτελείται η διεργασία Δ_3, η οποία ολοκληρώνεται τη στιγμή 13 msec, δηλαδή πριν λήξει το quantum. Ακολουθεί η εκτέλεση της διεργασίας Δ_4 για χρόνο ίσο με 1 quantum. Η διεργασία Δ_4 διακόπτεται χωρίς να έχει ολοκληρωθεί τη στιγμή 18 msec, γιατί έχει περάσει χρόνος ίσος με 1 quantum. Τότε η διεργασία Δ_4 πηγαίνει στο τέλος της ουράς αμέσως μετά τη διεργασία Δ_2. Ακολουθεί η εκτέλεση της διεργασίας Δ_5 και ο κύκλος εκτέλεσης των διεργασιών συνεχίζεται μέχρι να ολοκληρωθούν όλες.



Γραφικό 2.1. *Γραφική παράσταση εκτέλεσης αλγορίθμου της εξυπηρέτησης εκ περιτροπής με quantum = 5 msec.*

Το παρακάτω γραφικό 2.2 απεικονίζει μια γραφική παράσταση εκτέλεσης του αλγορίθμου της εξυπηρέτησης εκ περιτροπής για πέντε (5) διεργασίες και με **quantum = 10 msec.**

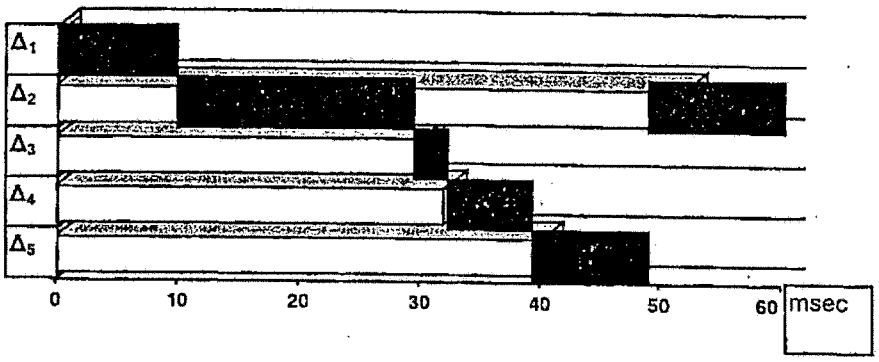
Η διεργασία Δ_1 φθάνει τη στιγμή 0 msec, η διεργασία Δ_2 φθάνει τη στιγμή 1 msec, η διεργασία Δ_3 φθάνει τη στιγμή 2 msec, η διεργασία Δ_4 φθάνει τη στιγμή 3 msec, και η διεργασία Δ_5 φθάνει τη στιγμή 4 msec. Ο απαιτούμενος χρόνος ολοκλήρωσής τους είναι για τη Δ_1 10 msec, για τη Δ_2 29 msec, για τη Δ_3 3 msec, για τη Δ_4 7 msec και τέλος για τη Δ_5 12 msec.



Γραφικό 2.2. *Γραφική παράσταση εκτέλεσης αλγορίθμου της εξυπηρέτησης εκ περιτροπής με quantum = 10 msec.*

Το παρακάτω γραφικό 2.3 παριστάνει μια γραφική παράσταση εκτέλεσης του αλγορίθμου της εξυπηρέτησης εκ περιτροπής για τις ίδιες

πέντε (5) διεργασίες Δ_1, Δ_2, Δ_3, Δ_4, Δ_5 και **quantum = 20 msec.**



Γραφικό 2.3. *Γραφική παράσταση εκτέλεσης του αλγορίθμου της εξυπηρέτησης εκ περιτροπής για quantum = 20 msec.*

(ια$_2$) Ο αλγόριθμος απλής προτεραιότητας

Ο αλγόριθμος απλής προτεραιότητας (simple priority) υιοθετεί την αρχή ότι σε κάθε διεργασία υπάρχει μια προτεραιότητα. Έτσι, από τις διεργασίες που είναι στην ουρά, εκείνη που είναι σε κατάσταση έτοιμη και έχει την υψηλότερη προτεραιότητα επιτρέπεται να εκτελεστεί. Για να εμποδιστεί όμως μια διεργασία με υψηλή προτεραιότητα, ώστε να μην εκτελείται επ' άπειρον, πρέπει η προτεραιότητα της εν λόγω διεργασίας να μειώνεται με κάθε διακοπή του ρολογιού του υπολογιστή. Όταν με αυτή τη μείωση, η προτεραιότητα μιας διεργασίας γίνει χαμηλότερη απ' αυτήν της αμέσως επόμενης, τότε αμέσως συμβαίνει εναλλαγή στην ανάθεση του επεξεργαστή (Deitel, 1990).

(ια$_3$) Ο αλγόριθμος του ελάχιστου εναπομείναντος χρόνου

Με βάση τον αλγόριθμο του **ελάχιστου εναπομείναντος χρόνου** (Shortest Remaining Time - **S.R.T.**), η υψηλότερη προτεραιότητα ανατίθεται στη διεργασία που έχει το μικρότερο υπολειπόμενο χρόνο ολοκλήρωσης. Έτσι λοιπόν, μια διεργασία εκτελείται, μέχρι να βρεθεί μια άλλη διεργασία με μικρότερο εναπομείναντα χρόνο ολοκλήρωσης. Οι διεργασίες μικρής διάρκειας εκτελούνται σχεδόν αμέσως, ενώ οι διεργασίες μεγάλης διάρκειας εκτελούνται τμηματικά. Επομένως, ο μέσος χρόνος αναμονής μιας διεργασίας είναι ανάλογος του χρόνου που απαιτείται για την ολοκλήρωσή της.

Βέβαια στη φάση της σχεδίασης του αλγορίθμου του ελάχιστου

εναπομείναντος χρόνου, προέκυψε το εξής σημαντικότατο πρόβλημα. Όταν μια διεργασία εκτελείται και έχει απομείνει ελάχιστος χρόνος για την ολοκλήρωσή της, τότε θα μπορεί να διακόπτεται από μια άλλη διεργασία με μικρότερο υπόλοιπο χρόνο ολοκλήρωσης.

Εξάλλου, είναι προφανές ότι δεν είναι δυνατόν το λειτουργικό σύστημα να θέσει στην αναμονή μια διεργασία που χρειάζεται ένα δευτερόλεπτο (ή και λιγότερο) για να ολοκληρωθεί.

Για να αντιμετωπιστεί λοιπόν αυτό το πρόβλημα, ορίστηκε ένα χρονικό όριο, έτσι ώστε να μην είναι δυνατή η διακοπή κάθε διεργασίας που χρειάζεται λιγότερο χρόνο από αυτό το χρονικό όριο, που απαιτείται για να ολοκληρωθεί (Deitel, 1990).

(ια$_4$) Ο αλγόριθμος της δίκαιης μοιρασιάς.

Η φιλοσοφία του αλγόριθμου της δίκαιης μοιρασιάς (fair share) είναι ότι οι χρήστες του υπολογιστικού συστήματος χωρίζονται σε ομάδες, ανάλογα με τη θέση τους στον οργανισμό που χρησιμοποιεί το υπολογιστικό σύστημα. Σε κάθε μια από αυτές τις ομάδες γίνεται ανάθεση ενός ποσοστού του διαθέσιμου χρόνου. Επιπλέον, γίνεται ανάθεση μιας προτεραιότητας στις διεργασίες της κάθε ομάδας, με τη χρήση των δύο παρακάτω τύπων.

Προτεραιότητα = (αριθμός χρήσης του επεξεργαστή / 2) + [αριθμός χρήσης του επεξεργαστή / (αριθμός ομάδων Χ ανατεθέν ποσοστό χρόνου)] + (αρχική προτεραιότητα)

Ο υπολογισμός του αριθμού χρήσης του επεξεργαστή γίνεται με τον πιο κάτω τύπο:

Αριθμός χρήσης του επεξεργαστή = (προηγούμενη προτεραιότητα + χρόνος εκτέλεσης) / 2

Πρέπει να σημειωθεί ότι κάθε φορά εκτελείται η διεργασία με τη χαμηλότερη προτεραιότητα (Deitel, 1990).

Παράδειγμα:

Υποθέτουμε ότι υπάρχουν δύο ομάδες που χρησιμοποιούν το υπολογιστικό σύστημα, η **ομάδα 1** και η **ομάδα 2** και σε κάθε μια έχει ανατεθεί το 50% του διαθέσιμου χρόνου. Έστω, ότι η διεργασία Α ανήκει στην ομάδα 1 και η διεργασία Β ανήκει στην ομάδα 2 και ότι τη στιγμή 0 sec οι προτεραιότητες όλων των διεργασιών είναι 0. Επιπρόσθετα δε ότι η διεργασία Α αρχίζει να εκτελείται πρώτη για χρόνο 60 sec.

Επεξήγηση:

Ο αριθμός χρήσης του επεξεργαστή για τη διεργασία Α θα

υπολογίζεται βάσει του τύπου:

Αριθμός χρήσης του επεξεργαστή = (προηγούμενη προτεραιότητα + χρόνος εκτέλεσης) / 2

Έτσι, ο αριθμός χρήσης του επεξεργαστή για τη διεργασία Α θα είναι: Αριθμός χρήσης του επεξεργαστή = (0 + 60) / 2 = 30.

Η προτεραιότητα της διεργασίας Α υπολογίζεται βάσει του τύπου:

Προτεραιότητα = (αριθμός χρήσης του επεξεργαστή / 2 + αριθμός χρήσης του επεξεργαστή) / (αριθμός ομάδων Χ ανατεθέν ποσοστό χρόνου) + (αρχική προτεραιότητα).

Η προτεραιότητα της διεργασίας **Α** θα είναι:

Προτεραιότητα της διεργασίας **Α** = 30 / 2 + 30 / (2 Χ 0.5) + 0 = 45. Παράλληλα, η διεργασία **Β** θα έχει προτεραιότητα 0. Έτσι, τη στιγμή 60 sec θα αρχίσει να εκτελείται η διεργασία Β η οποία έχει τη χαμηλότερη προτεραιότητα για χρόνο ίσο με 60 sec.

Τη στιγμή 120 sec ο αριθμός χρήσης του επεξεργαστή για τη διεργασία **Α** θα υπολογίζεται με βάση τον τύπο:

Αριθμός χρήσης του επεξεργαστή = (προηγούμενη προτεραιότητα + χρόνος εκτέλεσης) / 2

Ο αριθμός χρήσης του επεξεργαστή για τη διεργασία **Α** θα είναι:

(0 + 30) / 2 = 15

Τη στιγμή 120 sec η προτεραιότητα της διεργασίας **Α** υπολογίζεται με βάση τον τύπο:

Προτεραιότητα = (αριθμός χρήσης του επεξεργαστή / 2 + αριθμός χρήσης του επεξεργαστή) / (αριθμός ομάδων Χ ανατεθέν ποσοστό χρόνου) + (αρχική προτεραιότητα).

Έτσι, τη στιγμή 120 sec η προτεραιότητα της διεργασίας **Α** θα γίνει:

Προτεραιότητα της διεργασίας **Α** = 15 / 2 + 15 / (2 Χ 0.5) + 0 = 22.

Εξάλλου, τη στιγμή 120 sec ο αριθμός χρήσης του επεξεργαστή για τη διεργασία **Β** θα υπολογίζεται με βάση τον τύπο:

αριθμός χρήσης του επεξεργαστή = (προηγούμενη προτεραιότητα + χρόνος εκτέλεσης) / 2

Ο αριθμός χρήσης του επεξεργαστή για τη διεργασία **Β** θα είναι:

Αριθμός χρήσης του επεξεργαστή για τη διεργασία
Β = (0 + 60) / 2 = 30

Τέλος, τη στιγμή 120 sec η προτεραιότητα της διεργασίας Β υπολογίζεται με βάση τον τύπο:

Προτεραιότητα = (αριθμός χρήσης του επεξεργαστή / 2 + αριθμός χρήσης του επεξεργαστή) / (αριθμός ομάδων Χ ανατεθέν ποσοστό χρόνου) + (αρχική προτεραιότητα).

Η προτεραιότητα της διεργασίας **Β** τη στιγμή 120 sec θα γίνει:
Προτεραιότητα της διεργασίας **Β** = 30 / 2 + 30 / (2 X 0.5) + 0 = 45.

Τη στιγμή 120 sec λοιπόν, η διεργασία **Α** έχει προτεραιότητα 22 και η διεργασία **Β** έχει προτεραιότητα 45. Επειδή πάντοτε εκτελείται η διεργασία με τη μικρότερη προτεραιότητα, σειρά για εκτέλεση θα έχει η διεργασία **Α**. Η εναλλαγή στην εκτέλεση των διεργασιών Α και Β θα συνεχίζεται μέχρι να ολοκληρωθούν και οι δυο.

(ia_5) Ο αλγόριθμος του λειτουργικού συστήματος S.U.N. Unix

Το λειτουργικό σύστημα S.U.N. Unix πραγματοποιεί χρονοδιαχείριση των διεργασιών αναθέτοντας μια αρχική προτεραιότητα (base priority) σε κάθε μια διεργασία. Η αριθμητική τιμή αυτής της αρχικής προτεραιότητας μπορεί να φθάσει από -20 έως +20 με ενδιάμεση τιμή το μηδέν (0). Οι μεταβολές στην τιμή των αρχικών προτεραιοτήτων υπολογίζονται από το λειτουργικό σύστημα S.U.N. Unix, ανάλογα με τη μεταβολή των συνθηκών του συστήματος και προστίθενται στις αρχικές προτεραιότητες. Έτσι, κάθε διεργασία έχει διαφορετική προτεραιότητα σε κάθε διαφορετική χρονική στιγμή, δηλαδή η προτεραιότητα της κάθε διεργασίας αυξάνεται ή μειώνεται σε σχέση με την εξέλιξη της λειτουργίας του συστήματος. Η διεργασία με την υψηλότερη προτεραιότητα εκτελείται πάντα πρώτη (Tanenbaum, 1993).

Ο αλγόριθμος του λειτουργικού συστήματος S.U.N. Unix, ευνοεί τις διεργασίες που στο πρόσφατο παρελθόν έχουν απασχολήσει πολύ λίγο την κεντρική μονάδα επεξεργασίας. Έτσι λοιπόν, το λειτουργικό σύστημα διατηρεί ένα κατάλογο με τις διεργασίες που έχουν πρόσφατα απασχολήσει τη κεντρική μονάδα επεξεργασίας. Αυτός ο κατάλογος ανανεώνεται κάθε **5 Χ n** δευτερόλεπτα και όλο το προηγούμενο ιστορικό διαγράφεται. Με το σύμβολο **n** παριστάνεται ο μέσος αριθμός διεργασιών που εκτελέστηκαν στο τελευταίο λεπτό.

(ia_6) Ο αλγόριθμος του συστήματος VAX / VMS

Σύμφωνα με τους Kenah και άλλοι (1988), ο αλγόριθμος του συστήματος VAX / VMS χρησιμοποιεί επίσης προτεραιότητες, οι τιμές των

οποίων κυμαίνονται από 0 έως 31, με τη τιμή 31 να αποτελεί την υψηλότερη προτεραιότητα που μπορεί να αποκτήσει μια διεργασία. Οι συνήθεις διεργασίες έχουν προτεραιότητα από 0 έως 15, ενώ οι διεργασίες πραγματικού χρόνου έχουν προτεραιότητα από 16 έως 31.

Οι προτεραιότητες των διεργασιών πραγματικού χρόνου παραμένουν σταθερές και δε μεταβάλλονται. Οι συνήθεις διεργασίες διατηρούν σταθερή την αρχική τους προτεραιότητα αλλά δέχονται ρυθμίσεις στη συνολική τους προτεραιότητα (Tanenbaum, 1993).

(i_β) Αλγόριθμοι μη αναγκαστικής εγκατάλειψης

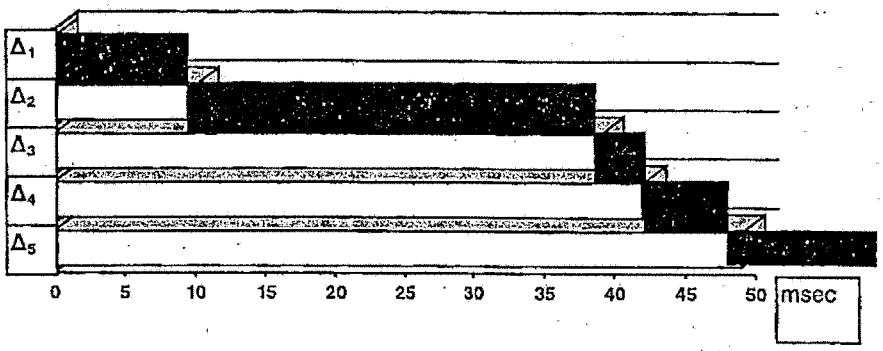
Οι κυριότεροι αλγόριθμοι **μη αναγκαστικής** εγκατάλειψης είναι: α) ο αλγόριθμος "πρώτο εισερχόμενο πρώτο εξερχομενο" (F.I.F.O), και β) ο αλγόριθμος της συντομότερης διεργασίας.

($i\beta_1$) Ο αλγόριθμος "πρώτο εισερχόμενο πρώτο εξερχομενο"

Ο αλγόριθμος "πρώτο εισερχόμενο πρώτο εξερχομενο" (First in First Out-F.I.F.O.), βασίζεται στην αρχή κατά την οποία η πρώτη αφιχθείσα διεργασία εκτελείται και πρώτη. Πρόκειται για τον απλούστερο αλγόριθμο που έχει σχεδιαστεί ποτέ. Οι διεργασίες τοποθετούνται σε μια ουρά ανάλογα με το χρόνο άφιξής τους. Το ιδιαίτερο χαρακτηριστικό του εν λόγω αλγόριθμου είναι ότι όλες οι διεργασίες εκτελούνται μέχρι να ολοκληρωθούν. Ο αλγόριθμος "πρώτο εισερχόμενο πρώτο εξερχομενο", χρησιμοποιείται πολύ σπάνια σήμερα, γιατί έχει πολύ μεγάλο χρόνο αναμονής στην ουρά. Αυτό δημιουργεί σοβαρές καθυστερήσεις, ιδίως όταν σημαντικότατες διεργασίες που πρέπει να εκτελεστούν γρήγορα αναγκάζονται να περιμένουν στην ουρά (Tanenbaum, 1993).

Το παρακάτω γραφικό 2.4 στη σελίδα 28 μας δίνει μια διαγραμματική παράσταση εκτέλεσης ενός αλγόριθμου "πρώτο εισερχόμενο πρώτο εξερχομενο" , για πέντε διεργασίες τις Δ_1,Δ_2,Δ_3,Δ_4,Δ_5.

Στην προκειμένη περίπτωση ,πρώτη φθάνει η Δ_1, δεύτερη η Δ_2, τρίτη η Δ_3, μετά η Δ_4 και τελευταία η Δ_5. Ο απαιτούμενος χρόνος ολοκλήρωσής τους είναι για την Δ_1 10 msec, για την Δ_2 29 msec, για την Δ_3 3 msec, για την Δ_4 7 msec και τέλος για την Δ_5 12 msec. Στο γραφικό 2.4 φαίνεται ότι κάθε διεργασία εκτελείται μέχρι να ολοκληρωθεί, χωρίς να υπάρχουν διακοπές. Είναι προφανές ότι η σειρά εκτέλεσης των διεργασιών είναι ίδια με τη σειρά άφιξής τους. Πρώτη εκτελείται η Δ_1, δεύτερη η Δ_2, τρίτη η Δ_3, μετά η Δ_4 και τελευταία η Δ_5.



Γραφικό 2.4. *Γραφική παράσταση της εκτέλεσης του αλγόριθμου "πρώτο εισερχόμενο πρώτο εξερχομενο".*

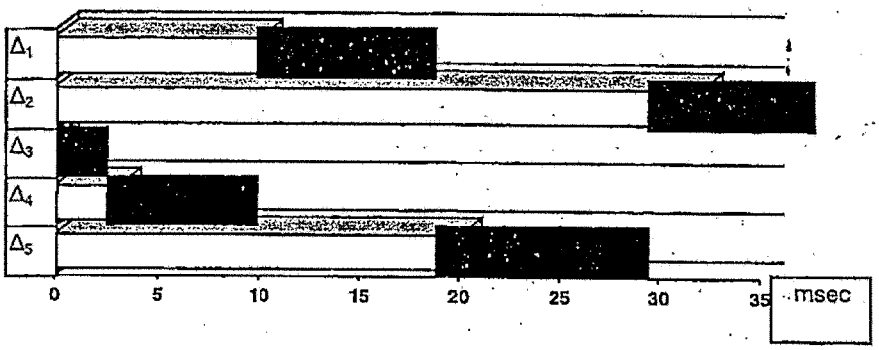
(iβ$_2$) Ο αλγόριθμος της συντομότερης διεργασίας.

Ο αλγόριθμος της συντομότερης διεργασίας (Shortest Job First - **S.J.F.**) βασίζεται στην αρχή κατά την οποία η συντομότερη διεργασία εκτελείται πρώτη. Η κάθε διεργασία που αποκτά τον έλεγχο της κεντρικής μονάδας επεξεργασίας, εκτελείται μέχρι να ολοκληρωθεί, χωρίς να διακοπεί ποτέ. Ο αλγόριθμος της συντομότερης διεργασίας ευνοεί τις σύντομες διεργασίες σε βάρος των μεγάλων διεργασιών που μπορεί να περιμένουν για πολύ μεγάλο χρονικό διάστημα την έναρξη της εκτέλεσής τους (Tanenbaum, 1993). Ο αλγόριθμος της συντομότερης διεργασίας δε κρίνεται ικανοποιητικός για συστήματα πολλαπλών χρηστών, αφού προκαλεί σοβαρές καθυστερήσεις.

Το παρακάτω γραφικό 2.5 στη σελίδα 29, αναφέρεται σε μια διαγραμματική παράσταση εκτέλεσης ενός αλγόριθμου **συντομότερης διεργασίας (S.J.F.)**, για πέντε διεργασίες. Στην προκειμένη περίπτωση και οι 5 διεργασίες φθάνουν τη στιγμή 0 sec και ο απαιτούμενος χρόνος ολοκλήρωσής τους είναι για την Δ_1 10 msec, για την Δ_2 29 msec, για την Δ_3 3 msec, για την Δ_4 7 msec και τέλος για την Δ_5 12 msec.

Στο γραφικό 2.5 φαίνεται ότι πρώτη ξεκινάει την εκτέλεσή της η συντομότερη διεργασία που είναι η Δ_3, μετά η δεύτερη συντομότερη διεργασία που είναι η Δ_4. Ακολουθεί η διεργασία Δ_1, η Δ_5 και τελευταία η διεργασία Δ_2 που είναι η πιο χρονοβόρα από όλες τις άλλες διεργασίες.

Είναι γνωστό ότι από τους αλγόριθμους που περιγράψαμε πιο πάνω, κανένας δεν είναι ο πλέον κατάλληλος για όλες τις περιπτώσεις διαχείρισης της κεντρικής μονάδας επεξεργασίας. Για το λόγο αυτό έχουν θεσπιστεί κάποια κριτήρια αξιολόγησής τους.



Γραφικό 2.5. *Γραφική παράσταση του αλγόριθμου της συντομότερης διεργασίας.*

(ii) Κριτήρια αξιολόγησης των αλγορίθμων

Ο κάθε αλγόριθμος έχει κάποια πλεονεκτήματα, αλλά ταυτόχρονα και μειονεκτήματα. Έτσι, σημασία έχει να γνωρίζει ο σχεδιαστής ενός υπολογιστικού συστήματος σε ποιά από τα χαρακτηριστικά του θα δώσει ιδιαίτερη βαρύτητα, ώστε να είναι σε θέση να διαλέξει τον αλγόριθμο που θα του προσφέρει τις καλύτερες υπηρεσίες.

Με βάση τα όσα αναφέραμε, τα κυριότερα κριτήρια είναι ο **μέσος χρόνος αναμονής**, η **λειτουργικότητα**, ο **λόγος δικαιοσύνης προς αποτελεσματικότητα** και η **μέση ανταπόδοση**.

Οπότε, κάθε ένα από τα κριτήρια αυτά μπορούν να οριστούν και να παρασταθούν μαθηματικά με τους πιο κάτω τύπους:

Μέσος χρόνος αναμονής = (συνολικός χρόνος αναμονής) / (αριθμό διεργασιών)

Λειτουργικότητα = αριθμός εργασιών που ολοκληρώνονται σε 30 msec.

Δικαιοσύνη / Αποτελεσματικότητα = Ο αριθμός των διεργασιών που απασχολούν τον επεξεργαστή σε 30 msec.

Μέση ανταπόδοση = (συνολικός χρόνος αναμονής και εκτέλεσης) / (αριθμό διεργασιών)

Όταν στο παράδειγμα εκτέλεσης αλγόριθμου "πρώτο εισερχόμενο πρώτο εξερχομενο" (**F.I.F.O.**), που περιγράφηκε παραπάνω στο γραφικό 2.4 στη σελίδα 28, υπολογίσουμε τα κριτήρια αξιολόγησης θα προκύψουν τα εξής αποτελέσματα:

Μέσος χρόνος αναμονής = (0 + 10 + 39 + 49) / 5 = 28 msec

Λειτουργικότητα = 1

Δικαιοσύνη / Αποτελεσματικότητα = 2

Μέση ανταπόδοση = (10 + 39 + 42 + 49 + 61) / 5 = 40 msec

Οταν στο παράδειγμα εκτέλεσης αλγόριθμου της **συντομότερης διεργασίας**, που περιγράφηκε στο παραπάνω γραφικό 2.5 στη σελίδα 29 εφαρμόσουμε τα κριτήρια αξιολόγησης θα προκύψουν τα εξής αποτελέσματα:

Μέσος χρόνος αναμονής = (10+32+0+3+20)/5 = 13 msec

Λειτουργικότητα = 3

Δικαιοσύνη / Αποτελεσματικότητα = 4

Μέση ανταπόδοση = (20 + 61 + 3 + 10 + 32) / 5 = 25 msec

Είναι προφανές ότι σε αυτή την περίπτωση ο αλγόριθμος της **συντομότερης διεργασίας** έχει μικρότερο **μέσο χρόνο αναμονής**, υψηλότερη **λειτουργικότητα**, καλύτερο λόγο **δικαιοσύνης / αποτελεσματικότητα**, καθώς και μεγαλύτερη **μέση ανταπόδοση**. Όμως σε κάποια άλλη περίπτωση ο αλγόριθμος "πρώτο εισερχόμενο πρώτο εξερχομενο" (**F.I.F.O.**), ίσως λειτουργούσε καλύτερα. Για το λόγο αυτό δεν μπορούμε να εξάγουμε γενικευμένα συμπεράσματα για την υπεροχή κάποιου αλγορίθμου σε σχέση με τους άλλους, από μια και μόνη περίπτωση.

Όταν στο παράδειγμα εκτέλεσης του αλγόριθμου της εξυπηρέτησης εκ περιτροπής, με quantum = 5 msec που περιγράφηκε παραπάνω στο γραφικό 2.1, εφαρμόσουμε τα κριτήρια αξιολόγησης, τότε θα προκύψουν τα εξής αποτελέσματα:

Μέσος χρόνος αναμονής = (18 + 32 + 10 + 28 + 35) / 5 = 25 msec

Λειτουργικότητα = 2

Δικαιοσύνη / Αποτελεσματικότητα = 6

Μέση ανταπόδοση = 37 msec

Αντίστοιχα, στο παράδειγμα εκτέλεσης του αλγόριθμου της εξυπηρέτησης εκ περιτροπής, με quantum = 10 msec που περιγράφηκε στο παραπάνω γραφικό 2.2 στη σελίδα 22, από την εφαρμογή των κριτηρίων αξιολόγησης θα προκύψουν τα εξής αποτελέσματα:

Μέσος χρόνος αναμονής = 23 msec

Λειτουργικότητα = 3

Δικαιοσύνη / Αποτελεσματικότητα = 4

Μέση ανταπόδοση = 35 msec

Επιπλέον, στο παράδειγμα εκτέλεσης αλγόριθμου της εξυπηρέτησης εκ περιτροπής, με quantum = 20 msec που περιγράφηκε στο παραπάνω γραφικό 2.3 στη σελίδα 23, κάνοντας χρήση των κριτηρίων αξιολόγησης

θα προκύψουν τα εξής αποτελέσματα:

Μέσος χρόνος αναμονής = 27 msec

Λειτουργικότητα = 1

Δικαιοσύνη / Αποτελεσματικότητα = 2

Μέση ανταπόδοση = 39 msec

Κάτω από αυτές τις συνθήκες, είναι φανερό ότι αυξομειώνοντας το quantum μπορούμε να βελτιώσουμε ή να χειροτερέψουμε τα αποτελέσματα που προκύπτουν από τα κριτήρια αξιολόγησης.

Ασκήσεις

1. Συγκρίνετε τους διάφορους αλγόριθμους ως προς τα κριτήρια αξιολόγησής τους και βρείτε ποιός είναι ο πιο δίκαιος και ο πιο αποτελεσματικός αλγόριθμος. Τι αναμένετε να συμβεί όταν το quantum μεγαλώνει;
2. Υποθέτουμε ότι μας δίνονται οι παρακάτω υπό εκτέλεση διεργασίες.

Διεργασία	Δ_1	Δ_2	Δ_3	Δ_4
Χρόνος άφιξης (msec)	0	6	8	11
Χρόνος ολοκλήρωσης (msec)	8	22	4	6

   Ζητείται να σχεδιάσετε πως θα γίνει η χρήση του επεξεργαστή από τις 4 πιο πάνω διεργασίες με quantum = 5 msec και βάσει του αλγορίθμου της εξυπηρέτησης εκ περιτροπής.
3. Χρησιμοποιήστε τα δεδομένα της ασκήσεως 2 και εφαρμόσατε τον αλγόριθμο της συντομότερης εργασίας και τον αλγόριθμο "πρώτο εισερχόμενο πρώτο εξερχομενο" (F.I.F.O.). (Σημειώνεται ότι στην περίπτωση αυτή quantum δε χρησιμοποιείται).