Τμήμα
Μηχανικών
Πληροφορικής τ.ε.
Τεχνολογικό Εκπαιδευτικό Ίδρυμα
Δυτικής Ελλάδας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το πρόβλημα του αμοιβαίου αποκλεισμού στα λειτουργικά συστήματα

Αντώνιος Μπέλλος

Επιβλέπων καθηγητής: Ταμπακάς Βασίλειος

Αντίρριο,Οκτώβριος 2016

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Αντίρριο, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή
2. Ονοματεπώνυμο, Υπογραφή
3. Ονοματεπώνυμο, Υπογραφή

Ευχαριστίες

Αρχικά θέλω να ευχαριστήσω όλους τους καθηγητές μου για τις πολύτιμη συμβολή τους στην ολοκλήρωση των προπτυχιακών σπουδών μου που ολοκληρώνονται με την παρούσα εργασία. Πολλές ευχαριστίες οφείλω στους γονείς μου που χωρίς την αμέριστη υλική και ηθική τους υποστήριξη θα ήταν αδύνατον να καταφέρω τον σκοπό μου. Επίσης θέλω να ευχαριστήσω τον καθηγητή μου κ. Βασίλειο Ταμπακά για την καθοδήγηση και την επίβλεψη της παρούσας πτυχιακής εργασίας.

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1ο : ΕΙΣΑΓΩΓΗ ΣΤΑ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

1.1 Ιστορική Αναδρομή

Με τον όρο λειτουργικό σύστημα [Α.Δ 3] εννοούμε το σύνολο των προγραμμάτων που διαχειρίζονται τους πόρους του υπολογιστικού συστήματος. Λέγοντας πόρους εννοούμε την μνήμη, τους δίσκους, τις μονάδες εισόδου και τις μονάδες εξόδου. Είναι δηλαδή η σημαντικότερη ομάδα προγραμμάτων, αφού δίχως αυτή δεν θα μπορούμε να λειτουργήσουμε το υπολογιστικό μας σύστημα. Όταν τη δεκαετία του 40 παρουσιάστηκαν οι πρώτοι Η/Υ οι οποίοι βασίζονταν στις λυχνίες κενού δεν είχαν λειτουργικό σύστημα. Το input γινόταν τοποθετώντας διακόπτες ανάλογα με το τι θέλαμε να εισάγουμε σε θέσεις ON ή OFF. Όταν όλα ήταν έτοιμα δινόταν από ένα άλλο διακόπτη η εντολή στον υπολογιστή να διαβάσει το συγκεκριμένο input και ανάλογα εάν δεν υπήρχαν άλλα δεδομένα προς εισαγωγή να βγάλει αποτελέσματα ή να περιμένει και επόμενο input. Η έξοδος και ανάγνωση των αποτελεσμάτων γινόταν σε ένα πίνακα με ενδεικτικές λυχνίες που ανάλογα εάν ήταν ανοιχτές ή σβηστές ο χρήστης έπαιρνε μια συμβολοσειρά με το αποτέλεσμα. Μιλάμε για καθαρό δυαδικό σύστημα και διεπαφή με το υπολογιστικό σύστημα καθαρά στη δική του γλώσσα. Αυτό όμως δεν ήταν δυνατό να συνεχίσει αφού δεν θα είχε καμία εμπορική αξία. Η κάθε εταιρία και ομάδα κατασκευής υπολογιστικού συστήματος είχαν την δική τους προσέγγιση πάνω στο θέμα με αποτέλεσμα μόνο αυτός που σχεδίασε και έφτιαξε το σύστημα να μπορεί να το διαχειριστεί κιόλας. Αυτό καθιστούσε τους υπολογιστές αδιάφορους για εταιρίες που θα ήθελαν να μηχανογραφηθούν αφού δεν θα μπορούν να χειρίζονται τους υπολογιστές οι υπάλληλοι τους αλλά μόνο πολύ εξειδικευμένο προσωπικό από τις κατασκευάστριες εταιρίες. Η πρώτη κίνηση έγινε από την IBM. Στη δεκαετία του ’50 έχουμε τα πρώτα λειτουργικά συστήματα για τα νέα main frame της εταιρίας. Ήδη είχε γίνει πρόοδος στον τρόπο που γίνετε το input των προγραμμάτων ή των δεδομένων με διάτρητες κάρτες ή διάτρητες ταινίες. Στα τέλη της δεκαετίας η IBM είχε παρουσιάσει το πρώτο λειτουργικό της σύστημα για τα μοντέλα IBM 704, 709 και 7090. Ωστόσο η μεγάλη πρόοδος γίνετε την δεκαετία του 60. Η IBM πάλι παρουσιάζει το νέο OS/360 για τα δικά της mainframe και άλλοι

κατασκευαστές ακολουθούν με τα δικά τους συστήματα. Από τα πρώτα εμπορικά υπολογιστικά συστήματα ήταν της Univac, σημερινή Unisys με το λειτουργικό EXEC-8 και αργότερα OS/2200 το υπάρχει μέχρι σήμερα. Την ίδια εποχή αναπτύχθηκε από το MIT τα εργαστήρια Bell και αλλους το λειτουργικό σύστημα Unix πάνω σε υπολογιστές PDP τότε το οποίο έγινε ένα από τα ποιο επιτυχημένα Λ. Σ. για μεγάλα συστήματα στη αρχή αλλά σε κάθε προσωπικό και φορητό υπολογιστή σήμερα. Ήδη όμως είχαν αναπτυχτεί για τα συστήματα αυτά και ανώτερες γλώσσες προγραμματισμού οπού επέτρεπαν το προγραμματισμό των υπολογιστικών συστημάτων με μεγαλύτερη ευκολία και μάλιστα κάποιες είχαν και συγκεκριμένο προσανατολισμό όπως π.χ. η COBOL για τη διαχείριση μεγάλου όγκου δεδομένων.
Ωστόσο στη δεκαετία του 70 και με την έλευση των ολοκληρωμένων κυκλωμάτων άρχισε η παραγωγή μικρών υπολογιστών καθώς και οικιακών υπολογιστών. Εκείνο το λειτουργικό που έγινε επιτυχία τότε ήταν το CP/M που είχε ως στόχο κυρίως συστήματα με τον 8080 της Intel και τους κλώνους ή συμβατούς με αυτόν επεξεργαστές όπως ο Z80. Τα λειτουργικά συστήματα τότε επικοινωνούσαν με το χρήστη μέσω εντολών που έδινε ο χρήστης μέσα από το πληκτρολόγιο. Το αποτέλεσμα μέχρι τις αρχές της δεκαετίας του 70, στα μεγάλα συστήματα και mainframe, ο χρήστης το έβλεπε τυπωμένο πάνω σε μηχανογραφικό χαρτί από ένα εκτυπωτή που υπήρχε μπροστά του σαν κονσόλα. Αργότερα αυτό το σύστημα με τον εκτυπωτή αντικαταστάθηκε με τις οθόνες που γνωρίζουμε, ενώ σε πολλά συστήματα το log του Η/Υ τυπώνονταν σε real time από ένα μικρό εκτυπωτή. Στα μικρά συστήματα τώρα που αναπτύχθηκαν από τα μέσα της δεκαετίας του 70 και ως τα μέσα του 80 λειτουργικά όπως το CP/M ήταν μονόδρομος και η διεπαφή γινόταν μέσω εντολών που πληκτρολογούσε ο χρήστης και το αποτέλεσμα απεικονιζόταν σε μία οθόνη. Η Microsoft είναι από τις πρώτες εταιρίες που παρουσιάζουν Basic για τα συστήματα με CP/M και αργότερα παίρνοντας το CP/M φτιάχνει το MS-DOS το κυρίως λειτουργικό του IBM PC. Τα home συστήματα που αναπτύχθηκαν από τα τέλη της δεκαετίας του 70 είχαν ένα Λ.Σ. ενσωματωμένο σε μια ROM το οποίο ήταν μικρό σε μέγεθος και ο χρήστης χειριζόταν τον υπολογιστή μέσα από ένα περιβάλλον Basic. Η πρώτη εταιρία που ανέπτυξε GUI (Graphics User Interface)ήταν η Xerox με το μίνι σύστημα της το Alto. Πάνω σε αυτό στηρίχτηκε σαν ιδέα η Apple και εγκαταλείποντας τον Apple II παρουσιάζει την Lisa και αμέσως μετά τον Macintosh. Το Mac OS ήταν το πρώτο GUI περιβάλλον για προσωπικούς υπολογιστές και αμέσως μετά την προφανή ευκολία χειρισμού ενός υπολογιστικού συστήματος γίνετε

προσπάθεια να μπει GUI σε νέα αλλά και παλαιότερα συστήματα. Βέβαια υπήρξαν και προσπάθειες από την Commodore με το Geos για τα 8bit συστήματα της καθώς και με το EASE για τους MSX2 χωρίς όμως μεγάλη επιτυχία. Η Microsoft παρουσιάζει τα Windows για τον IBM PC και συμβατούς χωρίς όμως επιτυχία. Ωστόσο οι νέοι 16bit υπολογιστές της Commodore, Atari και Acorn έχουν λειτουργικό σύστημα με GUI. Η Amiga και ο Atari ST γίνονται επιτυχία παγκοσμίως και το GUI της Amiga από τα ποιο αγαπητά. Ο Atari ST καταφέρνει να έχει το δικό του GUI στη ROM καθώς επίσης να βγάλει και έκδοση για τους IBM PC με την ονομασία GEM. Αυτό δινόταν μαζί με τα PC της Atari (PC1, PC2) αλλά και με τα συμβατά της Amstrad (PC1512, PC1640). Η Acorn είχε το δικό της GUI στη σειρά Archimedes όπου στα πρώτα μοντέλα ήταν μέσα στη ROM. Το Unix δεν έμεινε και αυτό απ'έξω αφού αναπτύχτηκε γι'αυτό το Xwindows GUI ενώ υπήρξαν και άλλες προσπάθειες από εταιρίες με μεγάλη επιτυχία όπως το Solaris της SUN. Η επικράτηση όμως των συστημάτων της IBM κατά την δεκαετία του 90 έκαναν το Amiga OS και το GEM να «εξαφανιστούν» από το προσκήνιο και τα Windows της Microsoft να παίρνουν το μεγαλύτερο μερίδιο της αγοράς ακολουθούμενα από το Mac OS της Apple και το Linux. Το Linux αν και ξεκίνησε σαν μια προσπάθεια ενός φοιτητή (Linus Torvalds) από τη Φιλανδία να φτιάξει ένα Unix like λειτουργικό για τον προσωπικό του υπολογιστή σήμερα κατάφερε να είναι ένα από τα ποιο επιτυχημένα λειτουργικά. Σε αυτό στηρίζονται και οι σημερινές εκδόσεις των MacOS και Android. Υπήρξαν βέβαια και λειτουργικά με GUI που αφορούσαν αποκλειστικά φορητές συσκευές. Αυτά ήταν τα Palm OS και το Epoc της Psion όπου σήμερα υπάρχει ακόμα ως Symbian σε κινητά τηλέφωνα, τα οποία λειτουργούν με επεξεργαστές ARM σαν αυτούς που είχε η σειρά Archimedes της Acorn. Σήμερα τα Android, iOS και Windows mobile, προσφέρουν απόλυτη ευκολία στη χρήση υπολογιστικών συστημάτων, αρκετά μικρότερων σε όγκο αλλά και κατά πολύ δυνατότερων σε επεξεργαστική ισχύ από τα πρώτα υπολογιστικά συστήματα που είχαν λειτουργικό σύστημα κάνοντας τον Η/Υ κτήμα του καθενός.

1.2 Λειτουργικό σύστημα

Το λειτουργικό σύστημα είναι ένα πρόγραμμα το οποίο υπηρετεί δύο στόχους. Από τη μια πλευρά, διαχειρίζεται τους πόρους του υπολογιστή και, από την άλλη πλευρά,

λειτουργεί σαν ενδιάμεσο στρώμα ανάμεσα στο υλικό και στον τελικό χρήστη. Με τον τρόπο αυτό διαχειρίζεται την εκτέλεση των προγραμμάτων εφαρμογών αποκρύπτοντας τις τεχνικές ιδιαιτερότητες του υλικού, ενώ παράλληλα παρεμβαίνει καθοριστικά στην απόδοσή του. Σημαντικό ρόλο στην εκτέλεση των προγραμμάτων έχουν οι διαδικασίες (processes). Η διαδικασία σ' ένα λειτουργικό σύστημα μπορεί να οριστεί ως ένα τμήμα κώδικα το οποίο έχει δική του ταυτότητα και εκτελείται. Ο ορισμός αυτός δεν είναι μοναδικός, αλλά μας επιτρέπει να αντιληφθούμε ότι, ενώ ένα πρόγραμμα είναι στατική οντότητα (σύνολο ανενεργών εντολών), μια διαδικασία είναι δυναμική οντότητα (σύνολο εντολών οι οποίες είναι ενεργοποιημένες). Επίσης, θα μας βοηθήσει να εξετάσουμε τον ρόλο ενός λειτουργικού συστήματος σ' έναν υπολογιστή. Μια διαδικασία, εκτός από κώδικα εντολών, μπορεί να περιέχει δεδομένα τα οποία θα χρησιμοποιηθούν από τις εντολές της. Επίσης, μπορεί να δεσμεύσει πόρους του υπολογιστή, όπως χώρο στην κύρια μνήμη ή χρόνο από τον επεξεργαστή ή κάποιο περιφερειακό. Μια διαδικασία βρίσκεται πάντοτε σε μια από τις παρακάτω γενικές καταστάσεις:

Νέα: Η διαδικασία δημιουργείται.

Έτοιμη: Η διαδικασία περιμένει να εκτελεστεί από τον επεξεργαστή.

Τρέχουσα: Οι εντολές της διαδικασίας εκτελούνται από τον επεξεργαστή.

Μπλοκαρισμένη: Η διαδικασία δεν εκτελείται, αλλά περιμένει την ολοκλήρωση κάποιου γεγονότως.

Τερματισμένη: Η διεργασία έχει ολοκληρώσει την εκτέλεσή της και σταματά να υπάρχει.

Αρχικά μια διαδικασία δημιουργείται και στη συνέχεια γίνεται «Έτοιμη» για εκτέλεση από τον επεξεργαστή. Όταν εκτελείται από τον επεξεργαστή, είναι σε κατάσταση «Τρέχουσα». Αν περάσει ο διαθέσιμος γι' αυτή χρόνος, αλλά δεν έχει ολοκληρωθεί, τότε μπορεί να μπει πάλι σε κατάσταση «Έτοιμη». Υπάρχει περίπτωση, όμως, κάποιο γεγονός να τη σταματήσει και να τη θέσει σε κατάσταση «Μπλοκαρισμένη» (π.χ. κάποιο αίτημα από μια συσκευή εισόδου/εξόδου). Όταν το αίτημα αυτό ολοκληρωθεί, η διαδικασία μπαίνει πάλι σε κατάσταση «Έτοιμη». Η ολοκλήρωση της διαδικασίας θα την οδηγήσει στην κατάσταση «Τερματισμένη». Κάθε διαδικασία περιγράφεται από μια αντίστοιχη δομή του λειτουργικού συστήματος, η οποία λέγεται *μπλοκ ελέγχου διαδικασίας* (Process Control Block, PCB). Ένα μπλοκ ελέγχου διαδικασίας μπορεί να περιέχει:

- *δείκτες προς άλλα μπλοκ ελέγχου διαδικασιών,* έτσι ώστε να διατηρείται η σειρά των διαδικασιών,
- την *κατάσταση* στην οποία βρίσκεται η διαδικασία,
- την *τιμή του καταχωρητή,* που ονομάζεται μετρητής προγράμματος και δείχνει στην επόμενη εντολή που πρόκειται να εκτελεστεί,
- τις *τιμές των υπόλοιπων καταχωρητών,* έτσι ώστε να είναι γνωστή η κατάσταση στην οποία βρίσκεται ο επεξεργαστής,
- *πληροφορίες για την κατάσταση της μνήμης* (π.χ. τα όρια της διαθέσιμης μνήμης, τις διαθέσιμες κενές θέσεις μνήμης κ.λπ.),
- *πληροφορίες που αφορούν τη διαχείριση της διαδικασίας* (π.χ. τον χρόνο που μπορεί η διαδικασία να δεσμεύσει τον επεξεργαστή ή τον χρόνο που τον έχει ήδη δεσμεύσει κ.λπ.).

1.3 Υπηρεσίες του λειτουργικού συστήματος

Στη συνέχεια, θα εξετάσουμε μερικές από τις σημαντικότερες *υπηρεσίες* που προσφέρει το *λειτουργικό σύστημα.*

Διαχείριση του επεξεργαστή/διαδικασιών: Παρακολουθεί την κατάσταση στην οποία βρίσκεται ο επεξεργαστής. Αποφασίζει ποια διαδικασία θα δεσμεύσει τον επεξεργαστή και για πόσο χρόνο. Παρακολουθεί τα διάφορα γεγονότα που μπορεί να συμβούν κατά τη διάρκεια της λειτουργίας του επεξεργαστή και διαχειρίζεται τα διάφορα σήματα διακοπών που δημιουργούνται από τον επεξεργαστή και τις περιφερειακές συσκευές. Αναλαμβάνει την αναστολή εκτέλεσης μιας διεργασίας και την επανεκκίνησή της.

Διαχείριση της κύριας μνήμης: Κατανέμει τη μνήμη στις διαδικασίες που τη ζητούν. Παρακολουθεί τον ελεύθερο, αλλά και τον δεσμευμένο χώρο της μνήμης. Επίσης, αποφασίζει πόσο χώρο θα λάβει κάθε διαδικασία, αλλά και για πόσο χρόνο θα τον έχει στην κατοχή της. Δεν επιτρέπει σε διεργασίες να προσπελαύνουν την περιοχή μνήμης όπου βρίσκεται το λειτουργικό σύστημα ή γενικότερα να προσπελαύνουν

ανεξέλεγκτα περιοχές μνήμης που δεν τους ανήκουν. Μετά τον τερματισμό μιας διαδικασίας αποδεσμεύει τον χώρο που είχε καταλάβει.

***Διαχείριση δευτερεύουσας μνήμης*:** Η δευτερεύουσα μνήμη είναι οργανωμένη σύμφωνα με ένα σύστημα καταλόγων και αρχείων. Το λειτουργικό σύστημα είναι υπεύθυνο για την οργάνωση, αλλά και τη διαχείριση αυτού του συστήματος. Κατανέμει χώρο για τα νέα αρχεία, διαγράφει όσα δεν χρειάζονται και αποδεσμεύει τον χώρο που καταλάμβαναν, παρακολουθεί την κατάσταση στην οποία βρίσκονται τα αρχεία και φροντίζει για τη γενική τακτοποίησή τους.

Διαχείριση περιφερειακών: Παρακολουθεί τη λειτουργία των περιφερειακών συσκευών. Από τη μια πλευρά, αναθέτει σε διεργασίες την εξυπηρέτηση αιτημάτων από τα περιφερειακά και, από την άλλη πλευρά, δεσμεύει ή αποδεσμεύει τα περιφερειακά, όταν απαιτείται από κάποιες διεργασίες.

Ασφάλεια: Προστατεύει από τη μη εξουσιοδοτημένη προσπέλαση στους πόρους του υπολογιστή. Αυτό μπορεί να γίνεται σε επίπεδο τελικού χρήστη, όπως με τη χρήση κωδικών ασφαλείας. Οι κωδικοί αυτοί φυλάσσονται και κωδικοποιούνται από το λειτουργικό σύστημα, ώστε η προσπέλαση και η αποκωδικοποίησή τους να μην είναι εύκολη υπόθεση από μη εξουσιοδοτημένα άτομα. Μπορεί όμως να προστατεύει και από επιθέσεις που οφείλονται σε κακόβουλα προγράμματα.

Διαχείριση σφαλμάτων: Παρακολουθεί τα διάφορα σφάλματα που μπορεί να προκύψουν και παράγει κατάλληλα μηνύματα ή και αναφορές γι' αυτά (π.χ. λανθασμένη προσπέλαση μνήμης). Παράλληλα, ακόμα και σε περίπτωση σφάλματος, προσπαθεί να διαφυλάξει τη λειτουργία του υπολογιστή και να αποτρέψει το να σταματήσει απότομα να λειτουργεί.

Έλεγχος της απόδοσης του υλικού: Επιβλέπει την απόδοση λειτουργίας καταγράφοντας τον χρόνο που μεσολαβεί από ένα αρχικό αίτημα εξυπηρέτησης μιας διαδικασίας μέχρι την τελική ικανοποίησή του.

Συντονισμός των διαδικασιών: Φροντίζει ώστε όλες οι διαδικασίες να αποκτούν πρόσβαση στους κοινούς πόρους του συστήματος σε εύλογο χρονικό διάστημα.

1.4 Είδη λειτουργικών συστημάτων

Οι ανάγκες διαχείρισης υπολογιστών και προγραμμάτων δεν είναι μονοσήμαντες. Υπάρχουν πολλοί διαφορετικοί τύποι μηχανημάτων, αλλά και πολλοί διαφορετικοί τρόποι να ικανοποιήσει κανείς τις ανάγκες διαχείρισης των πόρων τους και των προγραμμάτων που εκτελούνται σ' αυτά. Για αυτό τον λόγο έχουν αναπτυχθεί διαφορετικά είδη λειτουργικών συστημάτων. Στην ενότητα αυτή θα ομαδοποιήσουμε τα λειτουργικά συστήματα σε κατηγορίες ανάλογα με τα εξής κριτήρια: α) το πλήθος των χρηστών και των διαδικασιών που υποστηρίζει το λειτουργικό σύστημα, β) το είδος της αλληλεπίδρασης των χρηστών και του λειτουργικού συστήματος και γ) τη θέση στην οποία είναι εγκατεστημένο το λειτουργικό σύστημα. Για κάθε κατηγορία θα παρουσιάσουμε τα διαφορετικά είδη λειτουργικών συστημάτων.

Τα λειτουργικά συστήματα κατηγοριοποιούνται ως εξής σε σχέση με το πλήθος των χρηστών και των διαδικασιών που υποστηρίζουν:

Λειτουργικό σύστημα ενός χρήστη μίας εφαρμογής **(single user single application):** Αυτό το λειτουργικό σύστημα διαχειρίζεται έναν χρήστη και μία μόνο εφαρμογή κάθε φορά. Τέτοιου είδους λειτουργικό σύστημα έχουν σήμερα τα έξυπνα κινητά τηλέφωνα, ενώ η εικόνα που παρουσιάζεται σε ορισμένα από αυτά ότι εκτελούν ταυτόχρονα περισσότερες από μία εφαρμογές είναι πλασματική.

Λειτουργικό σύστημα ενός χρήστη πολλών διαδικασιών **(single user multitasking):** Αυτό το λειτουργικό σύστημα υποστηρίζει έναν χρήστη, αλλά πολλές διαφορετικές διαδικασίες. Οι πολλές διαδικασίες μπορεί να είναι το αποτέλεσμα πολλών διαφορετικών εφαρμογών ή μίας εφαρμογής, η οποία μπορεί να σπάσει σε πολλές διαδικασίες. Παράδειγμα αυτού του λειτουργικού συστήματος είναι το λειτουργικό σύστημα των περισσότερων προσωπικών υπολογιστών. Η έννοια της πολυεπεξεργασίας (multitasking) συνίσταται στο ότι το λειτουργικό σύστημα μπορεί να διαχειριστεί ταυτόχρονα περισσότερες από μία διαδικασίες, χωρίς να χάνει τον έλεγχό τους. Επιπλέον, μπορεί να μεταπηδά από τη μία στην άλλη, χωρίς να γίνεται αντιληπτό στο χρήστη. Με τον τρόπο αυτό μπορούν να εκτελούνται πολλές εφαρμογές στον ίδιο υπολογιστή. Για παράδειγμα, ένας χρήστης μπορεί να δουλεύει με έναν κειμενογράφο, ενώ ταυτόχρονα να ελέγχει τα δεδομένα ενός λογιστικού φύλλου.

Λειτουργικό σύστημα πολλών χρηστών και πολλών διαδικασιών **(multi-user**

multitasking): Ένα λειτουργικό σύστημα αυτής της κατηγορίας μπορεί να διαχειρίζεται πολλούς χρήστες και πολλές διαφορετικές διαδικασίες. Συνήθως, το λειτουργικό σύστημα αυτής της κατηγορίας αφορά υπολογιστές με υψηλές δυνατότητες, δηλαδή υπολογιστές με μεγάλη επεξεργαστική ισχύ και μεγάλη χωρητικότητα μνήμης. Αφορά κυρίως τους μεγάλους υπολογιστές που μπορούν να υποστηρίζουν πλήθος εργασιών με μεγάλες απαιτήσεις και πολλούς χρήστες. Για παράδειγμα, ένας χρήστης μπορεί να εκτελεί ένα πρόγραμμα εξομοίωσης κάποιου πειράματος, ένας άλλος να εκτελεί ένα πρόγραμμα πρόγνωσης του καιρού, ένας άλλος ένα σχεδιαστικό πρόγραμμα με τρισδιάστατα γραφικά κ.τ.λ. Το λειτουργικό σύστημα είναι υπεύθυνο για θέματα κατανομής πόρων, διαχείρισης διαδικασιών, απόδοσης, δικαιωμάτων πρόσβασης, γενικότερα θέματα ασφάλειας και πλήθος άλλα για όλους τους χρήστες.

Σε σχέση με το είδος της αλληλεπίδρασης των χρηστών και του λειτουργικού συστήματος διακρίνονται οι εξής κατηγορίες λειτουργικών συστημάτων:

Λειτουργικό σύστημα ομαδικής επεξεργασίας **(batch operating system):** Στα λειτουργικά συστήματα αυτά κάθε χρήστης προετοιμάζει τα προγράμματά του και τα υποβάλλει όλα μαζί. Από τη στιγμή που έχει γίνει η υποβολή, ο χρήστης δεν μπορεί να έχει κανενός είδους αλληλεπίδραση με το λειτουργικό σύστημα. Περιμένει μόνο να λάβει τα αποτελέσματα. Τα λειτουργικά συστήματα είναι σχετικά εύκολα στην υλοποίησή τους, αλλά παρουσιάζουν μειονεκτήματα, καθώς τα προγράμματα πρέπει να εκτελεστούν με τη σειρά. Ο επεξεργαστής συχνά μένει άεργος. Για παράδειγμα, εάν ένα πρόγραμμα εκτελεί λειτουργίες εισόδου/εξόδου, τότε όλα τα υπόλοιπα απλώς περιμένουν την ολοκλήρωση των αργών λειτουργιών εισόδου/εξόδου. Η έλλειψη αλληλεπίδρασης με τον χρήστη οδηγεί σε μεγάλη αναμονή τους χρήστες μέχρι να πάρουν κάποιο αποτέλεσμα. Δεν εφαρμόζεται κάποια πολιτική προτεραιοτήτων, με αποτέλεσμα η απόδοση του συστήματος να είναι μικρή. Στις μέρες μας τα λειτουργικά συστήματα αυτού του τύπου δεν χρησιμοποιούνται.

Λειτουργικό σύστημα διανομής χρόνου **(time sharing operating system):** Το λειτουργικό σύστημα αυτού του είδους επιτρέπει σε πολλούς χρήστες να αλληλεπιδρούν μαζί του. Κάθε χρήστης χρησιμοποιεί ένα κλάσμα του χρόνου του επεξεργαστή. Έτσι, ο επεξεργαστής μπορεί να εκτελεί πολλές διαδικασίες. Ο χρόνος που διαθέτει σε καθεμία είναι μικρός και η εναλλαγή από τη μια διαδικασία στην άλλη γίνεται γρήγορα. Αυτό δημιουργεί την ψευδαίσθηση στους χρήστες ότι το

λειτουργικό σύστημα τους επιτρέπει να δουλεύουν ταυτόχρονα όλοι μαζί στο ίδιο μηχάνημα. Επιπλέον, οι χρήστες έχουν γρήγορη απόκριση από το σύστημα, αφού, αν μια διαδικασία καθυστερεί να εκτελεστεί, τη θέση της στον επεξεργαστή παίρνει κάποια άλλη διαδικασία, ενώ αυτή περιμένει μέχρι να εκλείψουν οι λόγοι της καθυστέρησής της. Έτσι, μειώνεται ο χρόνος κατά τον οποίο ο επεξεργαστής παραμένει άεργος. Βέβαια, τα συστήματα αυτής της κατηγορίας οφείλουν να αντιμετωπίσουν θέματα ασφάλειας και αξιοπιστίας, καθώς απαιτείται αυστηρός συγχρονισμός των διαδικασιών που είναι ενεργές την ίδια χρονική στιγμή.

Λειτουργικά συστήματα πραγματικού χρόνου **(real time operating systems):** Τα λειτουργικά συστήματα πραγματικού χρόνου πρέπει να αποκρίνονται σε αυστηρούς χρονικούς περιορισμούς. Απόκριση του συστήματος είναι ο χρόνος που μεσολαβεί από τη στιγμή που το σύστημα δέχεται μια αίτηση για εξυπηρέτηση μέχρι να ικανοποιηθεί η αίτηση αυτή. Η χρήση τέτοιων συστημάτων είναι επιβεβλημένη σε περιπτώσεις, για παράδειγμα, επιστημονικών πειραμάτων, ελέγχου της εναέριας κυκλοφορίας ή σε ιατρικά συστήματα ελέγχου.

Τέλος, σε σχέση με τη θέση στην οποία είναι εγκατεστημένο το λειτουργικό σύστημα διακρίνονται οι εξής κατηγορίες:

Τοπικό λειτουργικό σύστημα **(local operating system):** Με τον όρο αυτό αναφερόμαστε στο λειτουργικό σύστημα το οποίο είναι εγκατεστημένο σ' έναν υπολογιστή μόνο και δεν επεκτείνεται η δράση του σε άλλο υπολογιστή.

Κατανεμημένο λειτουργικό σύστημα **(distributed operating system):** Το λειτουργικό σύστημα αυτού του είδους εγκαθίσταται σε πολλούς υπολογιστές, στον καθένα με τη δική του μονάδα επεξεργασίας. Τα κατανεμημένα λειτουργικά συστήματα εξυπηρετούν πολλούς χρήστες, πολλές εφαρμογές και διαχειρίζονται πολλούς επεξεργαστές ταυτόχρονα. Επιπλέον, διαχειρίζονται περισσότερη μνήμη σε σχέση με τα άλλα είδη λειτουργικών συστημάτων. Οι επεξεργαστές επικοινωνούν μεταξύ τους μέσα από διάφορα επικοινωνιακά κανάλια και οι διαδικασίες κατανέμονται στους επεξεργαστές ανάλογα με το ποιος μπορεί να τις διεκπεραιώσει πιο αποδοτικά. Οι χρήστες έχουν την εντύπωση ότι πρόκειται για ένα σύστημα και όχι για πολλά συνδεδεμένα συστήματα. Ακόμα και στην περίπτωση που σε κάποιον επεξεργαστή παρουσιαστεί πρόβλημα, οι υπόλοιποι συνεχίζουν να λειτουργούν κανονικά. Ο καταμερισμός των διαδικασιών είναι καλύτερος και οι υπηρεσίες προς τους χρήστες ταχύτερες. Ωστόσο, είναι συστήματα δυσκολότερα στην υλοποίηση και

στη συντήρηση και απαιτούν γρήγορες γραμμές επικοινωνίας ανάμεσα στους επεξεργαστές.

Δικτυακό λειτουργικό σύστημα **(network operating system):** Αυτά τα λειτουργικά συστήματα είναι εγκατεστημένα σ' έναν διακομιστή (server). Έχουν την ικανότητα να διαχειρίζονται πολλούς χρήστες, εφαρμογές και δικτυακές λειτουργίες. Πρόκειται για σταθερά συστήματα που παρέχουν κεντρική διαχείριση των υπηρεσιών τους. Χαρακτηρίζονται από υψηλή αξιοπιστία και αυξημένες δυνατότητες ασφάλειας. Επιπλέον, επιτρέπουν την απομακρυσμένη πρόσβαση από διαφορετικές τοποθεσίες και από διαφορετικούς τύπους μηχανημάτων. Από την άλλη πλευρά, σε περίπτωση προβλήματος στον διακομιστή, η λειτουργία του συστήματος καταρρέει. Επίσης, χρειάζεται εξειδικευμένο προσωπικό για την αναβάθμιση και τη συντήρησή του. Η διαφορά από τα κατανεμημένα λειτουργικά συστήματα είναι πως δεν δίνουν την εντύπωση ενός ενιαίου συστήματος και κάθε τοπικός ηλεκτρονικός υπολογιστής διατηρεί την αυτονομία του.

Λειτουργικό σύστημα ενεργής σύνδεσης **(online operating system):** Τα λειτουργικά συστήματα αυτής της κατηγορίας ονομάζονται και cloud operating systems (λειτουργικό σύστημα νέφους) ή web operating systems (λειτουργικό σύστημα παγκόσμιου ιστού). Δεν πρόκειται για πραγματικό λειτουργικό σύστημα, αλλά για περιβάλλον εικονικού λειτουργικού συστήματος. Οι όροι αυτοί αναφέρονται ουσιαστικά σε διαδικτυακές υπηρεσίες ενός κατανεμημένου μοντέλου υπολογισμού. Τα συστήματα αυτής της κατηγορίας δημιουργούν ένα εικονικό περιβάλλον εργασίας. Ο χρήστης συνδέεται μέσω διαδικτύου στον διακομιστή που προσφέρει εφαρμογές και τις εκτελεί. Τα δεδομένα αποθηκεύονται στον διακομιστή. Ο υπολογιστής του χρήστη δεν είναι υπεύθυνος για τη διαχείριση των εφαρμογών που εκτελούνται, ούτε για τη διαχείριση του συστήματος αρχείων στο οποίο αποθηκεύονται τα αρχεία. Το μηχάνημα του χρήστη γίνεται το σημείο διεπαφής όπου παρουσιάζονται οι διαδικτυακές εφαρμογές, αλλά στην πραγματικότητα έχει το δικό του ανεξάρτητο λειτουργικό σύστημα.

1.5 Δομή ενός λειτουργικού συστήματος

Στην ενότητα αυτή περιγράφουμε τα βασικά στοιχεία από τα οποία αποτελείται ένα λειτουργικό σύστημα. Το κεντρικό τμήμα ενός λειτουργικού συστήματος λέγεται

πυρήνας (kernel) του λειτουργικού συστήματος. Το τμήμα αυτό είναι υπεύθυνο για την άμεση διαχείριση του υλικού αλλά και για τη διαχείριση των διαδικασιών. Τμήματά του ρυθμίζουν τη σειρά με την οποία θα εκτελεστούν οι διαδικασίες ή την εκτέλεση και τον τερματισμό τους, πόσο χώρο θα καταλάβουν οι διαδικασίες στη μνήμη και την επικοινωνία με τις περιφερειακές συσκευές. Ο χρήστης δεν μπορεί να έχει απευθείας πρόσβαση σ' αυτό το κομμάτι του λειτουργικού συστήματος. Μπορεί, όμως, να έχει απευθείας πρόσβαση μόνο στην περιοχή της μνήμης που του έχει εκχωρηθεί από το λειτουργικό σύστημα. Ένα τμήμα του λειτουργικού συστήματος είναι υπεύθυνο και για τη διαχείριση της κύριας μνήμης του υπολογιστή. Το πώς θα γίνει ο καταμερισμός και η διαχείριση της κύριας μνήμης είναι μια από τις σημαντικές εργασίες για τις οποίες είναι υπεύθυνο το λειτουργικό σύστημα. Ένα άλλο τμήμα του λειτουργικού συστήματος διαχειρίζεται την επικοινωνία με τις περιφερειακές συσκευές. Αναγνωρίζει τις αιτήσεις των περιφερειακών συσκευών για εξυπηρέτηση και αναλαμβάνει να τις διαχειριστεί κατάλληλα. Εκτός από την κύρια μνήμη του υπολογιστή, ένα άλλο τμήμα του λειτουργικού συστήματος διαχειρίζεται τη δευτερεύουσα μνήμη. Διαχειρίζεται δηλαδή τα αρχεία που είναι αποθηκευμένα σε μέσα όπως ο σκληρός δίσκος. Για την ανάγνωση και την εγγραφή πληροφοριών σε αυτά τα μέσα εμπλέκεται ενεργά το λειτουργικό σύστημα. Η ασφάλεια των πληροφοριών και η εξουσιοδοτημένη προσπέλασή τους είναι επίσης θέμα που διαχειρίζεται ένα ξεχωριστό τμήμα του λειτουργικού συστήματος. Μπορούμε να συμπεράνουμε λοιπόν ότι ένα λειτουργικό σύστημα είναι ένα πολύπλοκο λογισμικό, το οποίο αποτελείται από επιμέρους τμήματα καθένα από τα οποία είναι υπεύθυνο για ξεχωριστές υπηρεσίες. Η πρόσβαση του χρήστη στις λειτουργίες του λειτουργικού συστήματος γίνεται μέσα από κάποιον τρόπο διεπαφής. Εάν η επικοινωνία χρήστη και λειτουργικού συστήματος γίνεται γράφοντας εντολές κειμένου, λέμε ότι έχουμε διεπαφή γραμμής εντολής (Command Line Interface, CLI). Αν πάλι η επικοινωνία γίνεται μέσα από κάποιο γραφικό περιβάλλον, τότε λέμε ότι έχουμε γραφικό περιβάλλον διεπαφής (Graphical User Interface, GUI). Και τα δύο είδη διεπαφής δημιουργούν το κέλυφος (shell) του λειτουργικού συστήματος. Επίσης, η επικοινωνία με τον πυρήνα του λειτουργικού συστήματος μπορεί να γίνει με προγραμματιστικό τρόπο, καλώντας συγκεκριμένες υπηρεσίες του μέσα από προγράμματα. Ο τρόπος αυτός λέγεται διεπαφή προγράμματος εφαρμογής (Application Program Interface, API). Σε πολλές περιπτώσεις το μέγεθος ενός λειτουργικού συστήματος είναι τόσο μεγάλο, με αποτέλεσμα να μη βρίσκεται ολόκληρο τοποθετημένο στην κύρια μνήμη

του υπολογιστή, παρά μόνο τα τμήματά του που είναι εντελώς απαραίτητα. Τα υπόλοιπα τμήματα του λειτουργικού συστήματος είναι αποθηκευμένα σε αρχεία στη δευτερεύουσα μνήμη. Αυτά τα αρχεία αποτελούν τις βιβλιοθήκες (libraries) του λειτουργικού συστήματος. Όταν το λειτουργικό σύστημα χρειαστεί τον κώδικα που βρίσκεται σ' αυτά τα αρχεία, τα φορτώνει εκείνη τη στιγμή στην κύρια μνήμη και αργότερα αποδεσμεύει τη μνήμη που κατέλαβαν, όταν πια δεν τα χρειάζεται. Σε άλλες περιπτώσεις, πάλι, οι βιβλιοθήκες εξυπηρετούν τη διεπαφή των προγραμμάτων εφαρμογών με το λειτουργικό σύστημα και χρησιμοποιούνται από προγράμματα εφαρμογών.

ΚΕΦΑΛΑΙΟ 2ο : ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Σε μία εφαρμογή πραγματικού χρόνου,[Α.Δ.7] το υπολογιστικό σύστημα και το υπό έλεγχο περιβάλλον είναι δύο στοιχεία που η συμπεριφορά τους καθορίζεται από

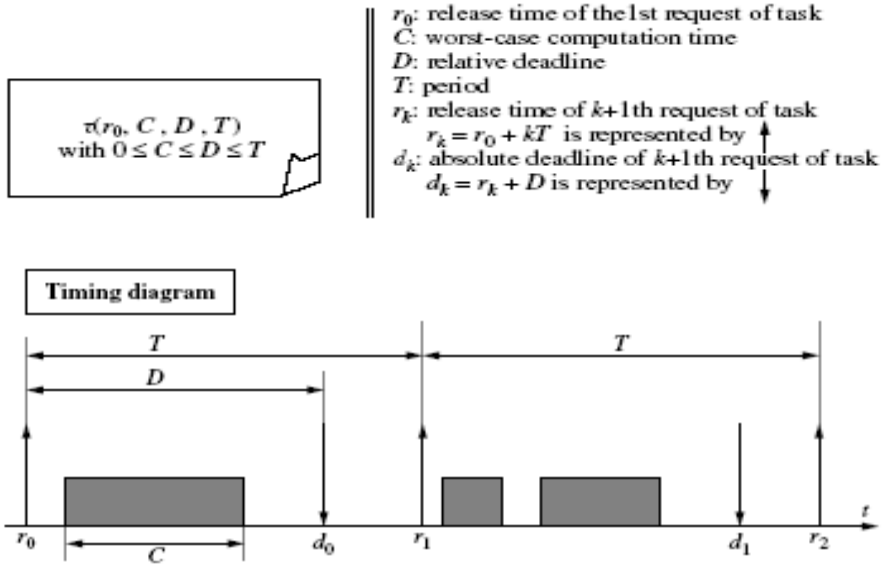
διαφορετικά χρονικά πεδία. Ο χρόνος στο περιβάλλον καθορίζεται από μετρήσεις πολύ συγκεκριμένης διάρκειας *(chronometric time)*. Στο υπολογιστικό σύστημα ο χρόνος καθορίζεται ως μια αλληλουχία συμβάντων *(chronological time)*. Στα συστήματα πραγματικού χρόνου το πρώτιστο μέλημα δεν είναι η υψηλή ακρίβεια ή η υψηλή πιστότητα των δύο προαναφερθέντων μορφών χρόνου (chronometric time - chronological time), αλλά ο ορθός έλεγχος και συγχρονισμός τους. Καθόσον ο χρόνος στο περιβάλλον καθορίζεται από φυσικές διεργασίες, το υπολογιστικό σύστημα πρέπει να προσαρμόσει το ρυθμό των λειτουργιών του σύμφωνα με το «ρολόι» του περιβάλλοντος. Σε κάθε περίπτωση στο πλαίσιο των εφαρμογών πραγματικού χρόνου, οι λειτουργίες του υπολογιστικού συστήματος καλούνται διεργασίες (processes – tasks), ενώ η οργάνωση της εκτέλεσής τους από επεξεργαστές καλείται χρονοδρομολόγηση πραγματικού χρόνου (real – time scheduling). Οι διεργασίες αυτές είναι γενικά σύντομες και ολοκληρώνονται πλήρως σε χρονικό διάστημα μικρότερο από ένα δευτερόλεπτο. Όταν ανιχνεύεται κάποιο εξωτερικό γεγονός που συμβαίνει στο υπό έλεγχο περιβάλλον, η διαδικασία της χρονοδρομολόγησης έγκειται στην επιλογή της κατάλληλης διεργασίας ώστε να τηρηθούν όλες οι αντίστοιχες προθεσμίες και χρονικοί περιορισμοί.

2.1 Παράμετροι Διεργασιών

Οι διεργασίες πραγματικού [Α.Δ.7] χρόνου είναι οι βασικές οντότητες οι οποίες χρονοδρομολογούνται. Οι διεργασίες μπορεί να είναι περιοδικές ή σποραδικές. Επίσης μπορεί να έχουν soft ή hard περιορισμούς. Ένα μοντέλο διεργασίας προσδιορίζεται με χρονικά χαρακτηριστικά που περιλαμβάνουν βασικές και δυναμικές παραμέτρους. Οι βασικές παράμετροι είναι **(Σχήμα 2.1)**:

- *r, χρόνος άφιξης της διεργασίας (task release time)* που προσδιορίζεται από τη χρονική στιγμή που γίνεται αίτηση για εκτέλεση της.
- *C, χρόνος εκτέλεσης της διεργασίας* όταν ο επεξεργαστής εξυπηρετεί μόνον αυτή *(worst-case computation time).*
- *D,* σχετική χρονική προθεσμία εκτέλεσης της διεργασίας που προσδιορίζει τη μέγιστη επιτρεπτή καθυστέρηση ολοκλήρωσής της *(task relative deadline).*

- *T, περίοδος* της διεργασίας μόνο για περιοδικές διεργασίες.
- Όταν η διεργασία έχει hard real-time χρονικούς περιορισμούς τότε η *σχετική* χρονική προθεσμία ολοκλήρωσής της μετατρέπεται σε *απόλυτη χρονική προθεσμία d=r+D.* Υπέρβαση των χρονικών ορίων της απόλυτης προθεσμίας, έχει αποτέλεσμα την αποτυχία τήρησης των χρονικών περιορισμών.



Σχήμα 2.1 Μοντέλο Διεργασίας

Η παράμετρος T δεν υπάρχει σε μια σποραδική διεργασία. Μία περιοδική διεργασία περιγράφεται από όλους τους παραπάνω χρονικούς παράγοντες. Κάθε φορά που μία διεργασία είναι έτοιμη (ready), απευθύνει μια αίτηση εκτέλεσης. Οι διαδοχικοί χρόνοι που απευθύνει αυτήν την αίτηση (request times – arrival times) είναι $r_k = r_0 + kT$, όπου r_0 είναι ο πρώτος χρόνος άφιξης. Οι διαδοχικοί απόλυτοι χρόνοι θα είναι $d_k = r_k + D$. Αν ισχύει D=T, η περιοδική διεργασία έχει σχετική χρονική προθεσμία εκτέλεσης ίση με την περίοδό της. Μια διεργασία είναι καλά δομημένη εάν ισχύει $0 < C \le D \le T$.

Η ποιότητα της χρονοδρομολόγησης εξαρτάται από την ακρίβεια των παραπάνω παραμέτρων και έτσι ο λεπτομερής καθορισμός τους είναι πολύ σημαντικό ζήτημα για το σχεδιασμό των συστημάτων πραγματικού χρόνου. Για την απόλυτα σωστή σχεδίαση ενός συστήματος πραγματικού χρόνου, πρέπει να ληφθούν υπόψη και να προστεθούν στις παραπάνω παραμέτρους και οι χρονικές διάρκειες των διαφόρων λειτουργιών του υπολογιστικού συστήματος οι οποίες είναι αναπόφευκτες,

όπως είναι οι κλήσεις συστήματος (operating systems calls), οι διαδικασίες διακοπών (interrupt processing), κ.α.

Άλλες παράμετροι που προσδιορίζουν τις διεργασίες είναι:

- *u = C/T,* συντελεστής *χρησιμοποίησης (utilization factor)* της διεργασίας. Πρέπει να ισχύει $u \leq 1$
- *ch = C/D,* συντελεστής *φόρτου (load factor)* της διεργασίας. Πρέπει να ισχύει $ch \leq 1$

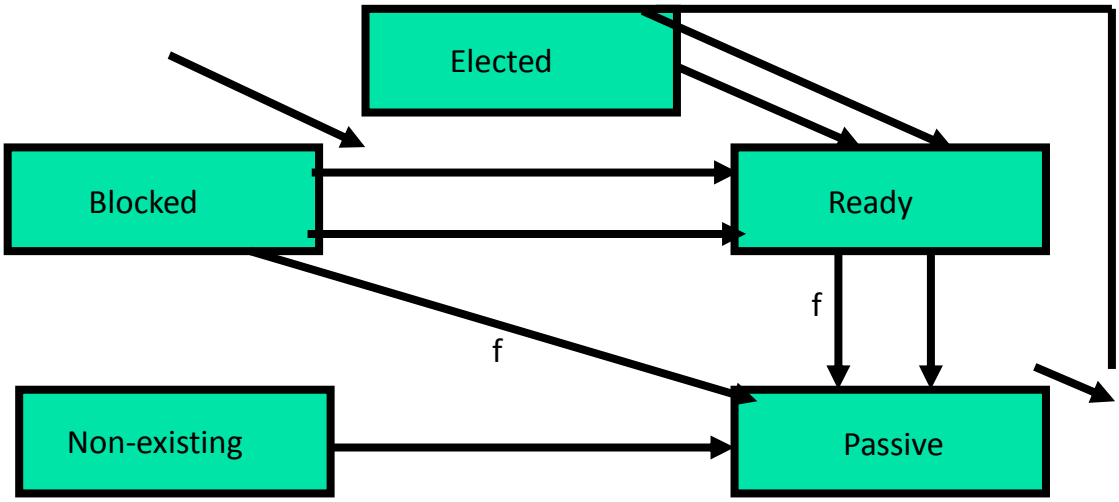
Οι παρακάτω δυναμικές παράμετροι βοηθούν για την παρακολούθηση της εκτέλεσης μιας διεργασίας:

- *s,* χρόνος έναρξης της εκτέλεσης της διεργασίας.
- *e,* χρόνος πέρατος της εκτέλεσης της διεργασίας.
- *D(t) = d-t,* χρόνος που υπολείπεται από το χρονικό σημείο (t) που βρίσκεται η διεργασία μέχρι το χρόνο πέρατος της σχετικής της προθεσμίας. Ισχύει: $0 \leq D(t) \leq D$.
- *C(t),* χρόνος που απομένει από το χρονικό σημείο που βρίσκεται η διεργασία μέχρι το χρονικό σημείο περάτωσής της. Ισχύει: $0 \leq C(t) \leq C$.
- *L = D – C* (nominal laxity), προσδιορίζει το χρονικό περιθώριο που έχει η διεργασία για να ολοκληρώσει τη λειτουργία της εντός των ορίων της προθεσμίας της .
- *L(t) = D(t) – C(t)* (relative laxity), προσδιορίζει το χρονικό περιθώριο που απομένει στη υπό εξέλιξη διεργασία από το σημείο που βρίσκεται μέχρι να ολοκληρώσει τη λειτουργία της εντός της προθεσμίας της.
- *TR = e-r,* χρόνος αντίδρασης της διεργασίας. Με ορθή χρονοδρομολόγηση πρέπει να ισχύει $C \leq TR \leq D$.
- *CH(t) = C(t)/D(t),* συντελεστής φόρτου σε μια υπό εξέλιξη διεργασία. Ισχύει $0 \leq CH(t) \leq C/T$

2.2 Καταστάσεις Διεργασιών

Οι καταστάσεις στις οποίες μπορεί να βρεθεί μια διεργασία είναι οι ακόλουθες **(Σχήμα 2.2):**

- *Elected*: η διεργασία εκτελείται, γεγονός που σημαίνει ότι ο επεξεργαστής έχει διατεθεί αποκλειστικά στη διεργασία εκείνη τη στιγμή. Σε αυτή την περίπτωση οι χρόνοι C(t) και D(t) μειώνονται συνεχώς.
- *Blocked:* η διεργασία έχει μπλοκαριστεί, επειδή δεν μπορεί να συνεχίσει την εκτέλεσή της μέχρι να λάβει χώρα κάποιο εξωτερικό συμβάν (αποδέσμευση πόρου, σήμα συγχρονισμού, κλπ.). Σε αυτή την περίπτωση οι χρόνοι L(t) και D(t) μειώνονται συνεχώς.
- *Ready:* η διεργασία είναι έτοιμη για εκτέλεση. Σε αυτή την περίπτωση οι χρόνοι L(t) και D(t) μειώνονται συνεχώς.
- Passive: δεν υπάρχει πρόσφατη αίτηση για ενεργοποίηση της διεργασίας.
- Non-existing: η διεργασία δεν έχει δημιουργηθεί.



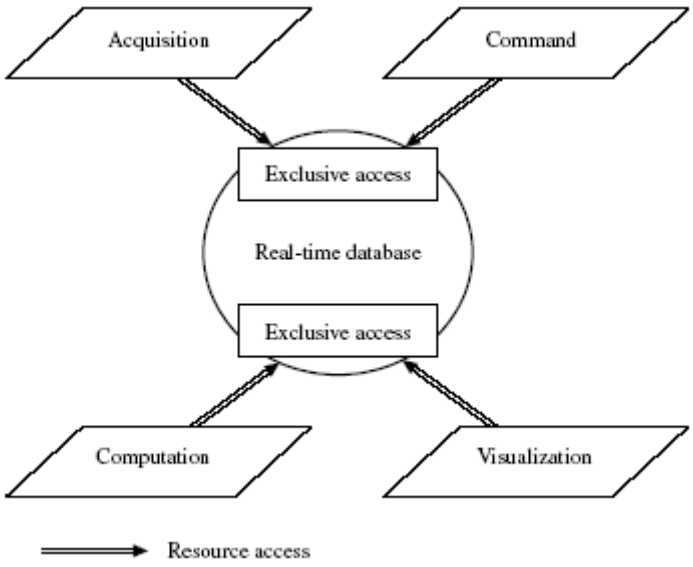
Σχήμα 2.2 Καταστάσεις Διεργασιών

Επιπρόσθετα, εκτός από τις χρονικές παραμέτρους, οι διεργασίες προσδιορίζονται και από άλλα χαρακτηριστικά που είναι τα παρακάτω:

- *Προεκτοπίσιμες ή μη προεκτοπίσιμες (preemptive or non – preemptive) διεργασίες*: μερικές διεργασίες από τη στιγμή που βρίσκονται σε κατάσταση *elected,* δηλαδή εκτελούνται, δεν θα μεταπέσουν σε άλλη κατάσταση πριν από το πέρας της εκτέλεσής τους. Αυτές καλούνται *non – preemptive διεργασίες.* Σε αντίθετη περίπτωση οι διεργασίες που βρίσκονται σε κατάσταση *elected*

και διακόπτεται ξαφνικά και βίαια η εκτέλεσή τους για να μεταβούν στην κατάσταση *ready,* καλούνται *preemptive* διεργασίες. Στην κατάσταση elected επιλέγεται να μεταβεί μια άλλη διεργασία.

- *Αλληλεξάρτηση διεργασιών (dependency of tasks):* οι διεργασίες πρέπει να αλληλεπιδρούν σύμφωνα με ένα κοινά συμφωνημένο τρόπο ή με την εκπομπή ενός μηνύματος ή τέλος με μια σαφή διαδικασία συγχρονισμού. Αυτό προϋποθέτει μια προγενέστερη σχέση μεταξύ των διεργασιών, η οποία θα είναι γνωστή πριν την εκτέλεσή τους και θα υποδεικνύει την προτεραιότητα εκτέλεσής τους. Έτσι θα
- επιτυγχάνεται ευκολότερα ο *αμοιβαίος αποκλεισμός (mutual exclusion)* δηλαδή η εξασφάλιση ότι εάν μια διεργασία χρησιμοποιεί την *κρίσιμη περιοχή (critical section)* ενός πόρου ή ενός αρχείου ή μιας βάσης δεδομένων, οι άλλες διεργασίες αποκλείονται από την εκτέλεση της ίδιας ενέργειας **(Σχήμα 1.10)**. Ο αμοιβαίος αποκλεισμός είναι πολύ σημαντικός παράγοντας στη διαδικασία της ροής των διεργασιών, διότι αποτρέπει την εξαγωγή σφαλμάτων που συμβαίνουν όταν δύο ή περισσότερες διεργασίες έχουν ταυτόχρονη προσπέλαση σε μια κρίσιμη περιοχή.



Σχήμα 2.3 Παράδειγμα Διαμοιρασμού Κρίσιμης Περιοχής από Τέσσερις Διεργασίες.

Υπάρχουν όμως και ανεξάρτητες διεργασίες οι οποίες δεν έχουν σχέσεις αλληλεξάρτησης μεταξύ τους και δεν διαμοιράζονται κρίσιμες περιοχές και έτσι ο χειρισμός τους όσο αφορά στη χρονοδρομολόγηση να είναι διαφορετικός.

- *Μέγιστο χρονικό περιθώριο (maximum jitter):* ο χρόνος που μεσολαβεί από τη χρονική στιγμή της αίτησης μιας διεργασίας μέχρι τη χρονική στιγμή της έναρξης της εκτέλεσής της πρέπει να είναι γνωστός και να κυμαίνεται σε συνήθη όρια. Ο μέγιστος αυτός χρόνος μεταξύ των διεργασιών καλείται maximum jitter.
- *Βαθμός επείγοντος (urgency):* οι χρονικές προθεσμίες της κάθε διεργασίας προσδιορίζουν το βαθμό του επείγοντος όσο αφορά στην παροχή των δεδομένων των διεργασιών. Σε δύο διεργασίες με ισοδύναμη urgency παρέχεται η ίδια χρονική προθεσμία (deadline).
- Σπουδαιότητα (importance): το σύστημα ελέγχου μπορεί να καθυστερήσει ή να παύσει την εκτέλεση ορισμένων διεργασιών όταν αυτές, ξεπερνώντας τις χρονικές προθεσμίες τους, δεν επιφέρουν ζημία στην εφαρμογή. Αντίθετα, άλλες διεργασίες είναι βασικές για την ομαλή εξέλιξη της εφαρμογής και δεν πρέπει να καθυστερήσει καθόλου η εκτέλεσή τους. Μπορεί δύο διεργασίες να έχουν ισοδύναμη urgency (ίδιες deadlines), αλλά να έχουν διαφορετική importance διότι μεταφέρουν διαφορετικής αξίας δεδομένα.

2.3 Αλγόριθμοι Διεργασιών

Σε ένα σύστημα πραγματικού χρόνου,[Α.Δ.7] οι διεργασίες έχουν χρονικούς περιορισμούς και η εκτέλεσή τους πρέπει να πραγματοποιηθεί μέσα σε σαφή χρονικά όρια. Ο αντικειμενικός σκοπός της χρονοδρομολόγησης, είναι να επιτρέψει στις διεργασίες να εκτελεστούν ικανοποιώντας αυτούς τους χρονικούς περιορισμούς. Ο τρόπος χρονοδρομολόγησης πρέπει να προβλέπει ότι όλοι οι χρονικοί περιορισμοί θα εκπληρωθούν, όταν η εφαρμογή διεκπεραιώνεται με κανονικό τρόπο. Όταν συμβεί κάτι ασυνήθιστο ή μια βλάβη στην υπό έλεγχο διαδικασία, πρέπει να είναι σε θέση να ενεργοποιηθούν διεργασίες που θα σημάνουν συναγερμό (alarm tasks) και τότε ο σκοπός της χρονοδρομολόγησης είναι να κρατήσει την όλη διαδικασία ασφαλή έχοντας ανοχή στο επίπεδο της ποιότητας λειτουργίας της εφαρμογής.

Οι διεργασίες μιας εφαρμογής πραγματικού χρόνου μπορεί να προκαλούνται ταυτόχρονα και έχουν τον ίδιο και ταυτόχρονο χρόνο άφιξης ή να προκαλούνται προοδευτικά.

Μερικοί ακόμα χρήσιμοι ορισμοί για τη λειτουργία των διεργασιών, είναι οι παρακάτω:

- Ορίζεται ως *συντελεστής χρήσης* του επεξεργαστή του υπολογιστικού συστήματος *(processor utilization factor)* για ένα πλήθος από n περιοδικές διεργασίες, η έκφραση:

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti}$$

- Κατά παρόμοιο τρόπο ορίζεται και ο *συντελεστή φόρτου* του επεξεργαστή *(processor load factor)* για ένα πλήθος από n περιοδικές διεργασίες, ως η έκφραση:

$$CH = \sum_{i=1}^{n} \frac{Ci}{Di}$$

- *Ανοχή του επεξεργαστή (processor laxity):*Επειδή οι παραπάνω εκφράσεις του processor utilization factor και του processor load factor δεν επαρκούν για να εκτιμήσουν εάν μια διεργασία κατά την εκτέλεσή της θα υπερβεί τις χρονικές προθεσμίες, εισάγεται η έννοια της α*νοχή του επεξεργαστή (processor laxity) LP(t),* που προσδιορίζει το μέγιστο χρόνο που ο επεξεργαστής μπορεί να παραμείνει ανενεργός χωρίς αυτό το γεγονός να κοστίσει στη διεργασία να χάσει κάποια χρονική προθεσμία. Ο χρόνος αυτός ποικίλλει συνεχώς ανάλογα σε πιο χρονικό σημείο εξέλιξης βρίσκεται η διεργασία και σε κάθε περίπτωση είναι $LP(t) \geq 0$. Υπολογίζεται και η *εξαρτώμενη (conditional) laxity LCi(t)* μιας διεργασίας i, όταν έχουν προηγηθεί j διεργασίες ως:

$$LCi(t) = Di - \sum_{i=1}^{n} Cj(t)$$

- *Ανενεργός χρόνος του επεξεργαστή (processor idle time): το χρονικό διάστημα στο οποίο δεν έχει ανατεθεί κάποια διεργασία στον επεξεργαστή.*

Η χρονοδρομολόγηση ενός πλήθους *(set)* από διεργασίες απαιτεί την κατάρτιση ενός αυστηρού πλαισίου με σκοπό την τήρηση των χρονικών περιορισμών, που να λαμβάνει υπόψη:

- την εκτέλεση όλων των διεργασιών όταν το σύστημα λειτουργεί με κανονικό τρόπο.
- Την εκτέλεση τουλάχιστον των διεργασιών που θεωρούνται ότι έχουν τον υψηλότερο βαθμό από άποψης σπουδαιότητας (για παράδειγμα αυτές που θα κρατήσουν το υπολογιστικό σύστημα και την υπό έλεγχο διαδικασία ασφαλή), όταν το σύστημα λειτουργεί με ανώμαλο τρόπο.

Ένας ανώμαλος τρόπος λειτουργίας μπορεί να προέλθει είτε από αστοχία του υλικού *(hardware faults)* είτε από άλλα μη αναμενόμενα συμβάντα.

Οι αλγόριθμοι χρονοδρομολόγησης αναθέτουν την εκτέλεση των διεργασιών στους επεξεργαστές και παρέχουν μια αυστηρά διατεταγμένη λίστα διεργασιών στην οποία καθορίζεται η αλληλουχία και η προτεραιότητα εκτέλεσής τους. Οι κατηγορίες αλγορίθμων που χρησιμοποιούνται είναι οι παρακάτω:

- *Off-line*: ένας αλγόριθμος off-line καταρτίζει μια ολοκληρωμένη λίστα με την αλληλουχία των διεργασιών και τις παραμέτρους που καθορίζουν την κάθε μία. Η σειρά εκτέλεσης των διεργασιών είναι εκ των προτέρων γνωστή και δεν αλλάζει σε καμία περίπτωση. Προφανώς αυτή η προσέγγιση είναι στατική και άκαμπτη. Θεωρεί ότι όλες οι παράμετροι, όπως οι χρόνοι άφιξης των διεργασιών, οι χρόνοι εκτέλεσης, κλπ., είναι σταθεροί και έτσι δεν υπάρχει δυνατότητα προσαρμογής σε μεταβολές του εξωτερικού περιβάλλοντος.
- *On-line*: η κατηγορία αυτή έχει γνώση των παραμέτρων όλων των διεργασιών που έχουν προκαλέσει αίτηση για εκτέλεση και επιτρέπει κάθε χρονική στιγμή την επιλογή της επόμενης διεργασίας. Έτσι ανάλογα με τα εξωτερικά συμβάντα, ο αλγόριθμός αποφασίζει για το ποιά διεργασία θα ακολουθήσει αυτήν που εκτελείται. Αυτή η κατηγορία αλγορίθμων επιλέγεται όταν υπάρχει η ανάγκη για δυναμικό χειρισμό των διεργασιών και όταν υπάρχουν λίγες πληροφορίες για τη συμπεριφορά της υπό έλεγχο διαδικασίας. Χρησιμοποιείται κυρίως για χειρισμό απεριοδικών –

σποραδικών διεργασιών και για συστήματα που θεωρείται σίγουρο ότι θα λειτουργήσουν με ανώμαλο τρόπο.

- *Preemptive*: Σε μια τέτοια κατηγορία αλγορίθμων χρονοδρομολόγησης, η εκτέλεση μιας διεργασίας μπορεί να διακοπεί βίαια πριν τον προβλεπόμενο χρόνο ολοκλήρωσης της όλης ενέργειάς της και να ανατεθεί στον επεξεργαστή μια άλλη διεργασία με μεγαλύτερη προτεραιότητα. Η διεργασία που διακόπτεται μεταπίπτει στην κατάσταση ready και αναμένει για το πότε θα μεταβεί στην κατάσταση elected. Πλεονέκτημα αυτού του τύπου χρονοδρομολόγησης είναι ότι με κατάλληλο χειρισμό των διεργασιών αποφεύγονται οι παραβιάσεις των χρονικών περιορισμών.
- *Non – preemptive*: Σε μια non-preemptive χρονοδρομολόγηση, οι διεργασίες από τη στιγμή που θα αρχίσουν την εκτέλεσή τους θα την ολοκληρώσουν οπωσδήποτε. Με αυτόν τον τρόπο όμως είναι πολύ πιθανό να το σύστημα να μην είναι σε θέση να τηρήσει τις χρονικές προθεσμίες.
- *Best effort – timing fault intolerance*: Η τεχνική αυτή χρησιμοποιείται στα soft συστήματα όταν η εφαρμογή επιτρέπει την ανοχή στην τήρηση των χρονικών περιορισμών.
- *Centralized – distributed* : η χρονοδρομολόγηση τέτοιου είδους χρησιμοποιείται όταν οι παράμετροι όλων των διεργασιών είναι εκ των προτέρων γνωστοί και είναι δυνατό να δομηθούν σε μια κατανεμημένη αρχιτεκτονική. Η τεχνική αυτή χρησιμοποιείται όταν η εφαρμογή αποτελείται από διάφορα επίπεδα και σε κάθε επίπεδο τηρείται μια «τοπική» λογική χρονοδρομολόγησης.

2.4 Χαρακτηριστικά Χρονοδρομολόγησης

Διακρίνονται τα παρακάτω χαρακτηριστικά χρονοδρομολόγησης:

- *Εφικτή χρονοδρομολόγηση (feasible schedule):* ο αλγόριθμος χρονοδρομολόγησης έχει ως αποτέλεσμα ένα χρονοπρογραμματισμό ενός

πλήθους από διεργασίες. Αυτός ο χρονοπρογραμματισμός είναι feasible, εάν όλες οι διεργασίες ικανοποιούν τους χρονικούς τους περιορισμούς.

- *Ικανή (schedulable) χρονοδρομολόγηση:*μία χρονοδρομολόγηση είναι schedulable όταν ο αλγόριθμος χρονοδρομολόγησης είναι ικανός να οδηγήσει σε ένα feasible χρονοπρογραμματισμό.
- *Βέλτιστος αλγόριθμος χρονοδρομολόγησης (optimal scheduling algorithm):* είναι αυτός που παράγει ένα feasible χρονοπρογραμματισμό για κάθε schedulable πλήθος από διεργασίες.
- *Τεστ ικανότητας της χρονοδρομολόγησης (schedulability test):* επιτρέπει τον έλεγχο του κατά πόσο ένα πλήθος από διεργασίες που έχουν υποβληθεί σε ένα αλγόριθμο μπορεί να έχει ως αποτέλεσμα ένα feasible χρονοπρογραμματισμό.
- *Τεστ αποδοχής (acceptance test):*ένας on – line αλγόριθμος επιτρέπει, όπως προαναφέρθηκε, το δυναμικό χρονοπρογραμματισμό. Έτσι μια διεργασία μπορεί να κληθεί για εκτέλεση όταν εντοπισθεί ότι υπάρχει κίνδυνος να μην ικανοποιήσει τις χρονικές δεσμεύσεις της εάν παραμείνει σε κατάσταση άλλη από την elected. Για να συμβεί η κλήση αυτής της διεργασίας πρέπει να υπάρχει μια ρουτίνα που να ελέγχει όλες τις διεργασίες που δεν εκτελούνται και να προκαλεί τη κλήση αυτών που τείνουν να χάσουν τις χρονικές προθεσμίες τους. Αυτή η διαδικασία καλείται acceptance test.
- *Περίοδος χρονοδρομολόγησης (scheduling period):* ο καθορισμός ενός περιοδικού ή σποραδικού πλήθους διεργασιών είναι το στοιχείο που οδηγεί στην ανάλυση των χρονικών απαιτήσεων για αυτές τις διεργασίες. Όταν οι περιοδικές διεργασίες περατώνονται κατά ακαθόριστο χρονικό τρόπο η ανάλυση πρέπει να τείνει στο άπειρο. Σε κάθε περίπτωση που περιλαμβάνει ένα πλήθος περιοδικών διεργασιών, μπορεί να αναλυθεί με έγκυρο τρόπο υπολογίζοντας μια χρονική περίοδός τους ή ψευδοπερίοδο (scheduling period) [2]. Η περίοδος αυτή του πλήθους των διεργασιών, αρχίζει με $\tau(r_{i,o})$, ο χρόνο άφιξης της πρώτης διεργασίας $t = \text{Min}(r_{i,o})$ και τελειώνει σε ένα χρονικό σημείο το οποίο συνεκτιμάται από το σχήμα της περιόδου των περισσοτέρων διεργασιών με κοινά χαρακτηριστικά περιοδικότητας (least common multiple-

LCM), του πρώτου χρόνου άφιξης και των προθεσμιών των απεριοδικών διεργασιών:

$$\text{Max}\{(r_{i,o}), (r_{j,o}+D_{i,})\} + 2.\ \text{LCM}(T_{i,})$$

όπου i είναι μία από τις περιοδικές και j είναι μια από τις σποραδικές διεργασίες.

2.5 Πολιτικές Εκτέλεσης της Χρονοδρομολόγησης

Η εκτέλεση της χρονοδρομολόγησης βασίζεται σε συμβατικές δομές δεδομένων:

- *Election table*: όταν η χρονοπρογραμματισμός είναι σταθερός και γνωστός πριν την έναρξη της εφαρμογής, όπως για παράδειγμα σε μια στατική off-line χρονοδρομολόγηση, αυτός ο προκαθορισμένος χρονοπρογραμματισμός μπορεί να αποθηκευθεί σε ένα *πίνακα διεργασιών (election table)*, που επιτρέπει στο χρονοπρογραμματιστή να γνωρίζει ποια είναι η επόμενη διεργασία προς εκτέλεση.
- *Priority queuing list:* στον on-line χρονοπρογραμματισμό δημιουργείται μια δυναμική αλληλουχία διεργασιών, της οποίας η πρώτη στη σειρά διεργασία είναι αυτή που εκτελείται την κάθε χρονική στιγμή. Η δομή αυτή σύμφωνα με την οποία καταρτίζεται η λίστα της αλληλουχίας των διαδικασιών είναι καλείται Priority queuing list ή priority ordered list .
- *Constant or varying priority:* η προτεραιότητα εκτέλεσης μιας διεργασίας καθορίζεται από ένα πλήθος χρονικών παραμέτρων. Η προτεραιότητα παραμένει σταθερή όταν οι αυτές οι παράμετροι, όπως για παράδειγμα ο χρόνος άφιξης και πέρατος, τα deadlines, κλπ. Η προτεραιότητα αυτή μεταβάλλεται όταν οι υπόψη παράγοντες αλλάζουν κατά τη διάρκεια της εκτέλεσης της διεργασίας.
- *Two – level scheduling:* όταν η χρονοδρομολόγηση γίνεται πολύπλοκη, διαιρείται σε δύο μέρη. Το ένα επεξεργάζεται την πολιτική που θα ακολουθηθεί (υψηλού επιπέδου αποφάσεις, όπως την επιλογή της προτεραιότητας κάποιων κρίσιμων διεργασιών). Το άλλο μέρος επεξεργάζεται

χαμηλότερου επιπέδου μηχανισμούς, όπως την εκτέλεση μιας διεργασίας που έχει αποφασισθεί από το υψηλότερο επίπεδο να εκτελεσθεί.

2.6 Χρονοδρομολόγηση στα Κλασσικά Λειτουργικά Συστήματα

Σε ένα σύστημα πολυπρογραμματισμού, η χρονοδρομολόγηση έχει δύο αντικειμενικούς σκοπούς:

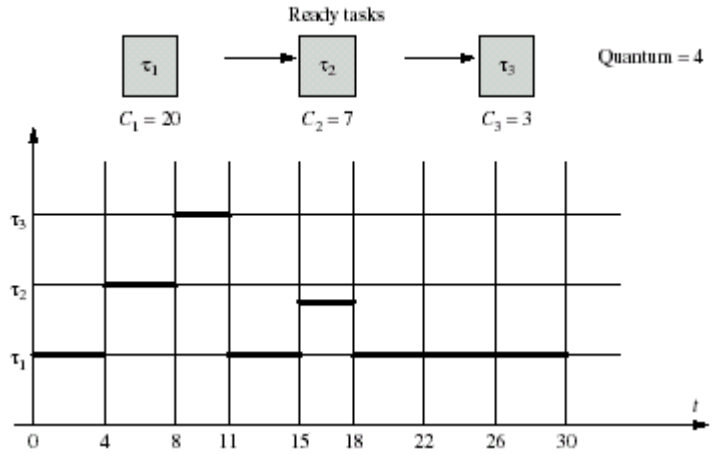
- Να μεγιστοποιήσει τη χρήση του επεξεργαστή, δηλαδή αυξήσει το χρόνο που αυτός είναι ενεργός (active) και να μειώσει το χρόνο που είναι ανενεργός (idle). Θεωρητικά, ο λόγος active/idle του επεξεργαστή ποικίλλει από 0% έως 100%. Πρακτικά ο χρόνος αυτός ποικίλλει από 40% έως 95%.
- Ελαχιστοποιήσει το χρόνο απόκρισης των διεργασιών, δηλαδή το χρόνο μεταξύ της αίτησης για εκτέλεση μέχρι το πέρας αυτής.

Για την επίτευξη των παραπάνω αντικειμενικών σκοπών, χρησιμοποιούνται αλγόριθμοι χρονοπρογραμματισμού, κυριότεροι από τους οποίους είναι [4]:

- *Εξυπηρέτηση με βάση τη σειρά άφιξης (first come first served):*είναι πιθανότατα ο απλούστερος από όλους τους αλγόριθμους χρονοπρογραμματισμού. Ο αλγόριθμος εξασφαλίζει ότι οι διεργασίες καταλαμβάνουν το χώρο της CPU με τη σειρά που το ζήτησαν, ενώ χρησιμοποιεί μια ουρά που περιέχει τις έτοιμες διεργασίες. Είναι ένας μη – προεκτοπιστικός (non – preemptive) αλγόριθμος. Έτσι έχει το μειονέκτημα ότι μια διεργασία που ο χρόνο διάρκειας της εκτέλεσής της είναι μεγάλος, θα κρατήσει για όλον αυτό το χρόνο στην κατοχή του την CPU.
- *Εξυπηρέτηση με βάση τη μικρότερη διάρκεια (shortest job first):*ο αλγόριθμος εξυπηρετεί τις διεργασίες που έχουν το μικρότερο χρόνο εκτέλεσης, χωρίς προεκτόπιση. Απαιτείται να είναι γνωστός εκ των προτέρων ο χρόνος εκτέλεσης των διεργασιών. Μια προεκτοπιστική έκδοση του αλγορίθμου είναι η *εξυπηρέτηση με βάση τη μικρότερη διάρκεια που απομένει (shortest remaining time next),* κατά την οποία μια διεργασία εγκαταλείπει την CPU,

όταν μία άλλη διεργασία που βρίσκεται στην κατάσταση ready έχει μικρότερο χρόνο εκτέλεσης και έτσι αναλαμβάνει αυτή τον έλεγχο της CPU.

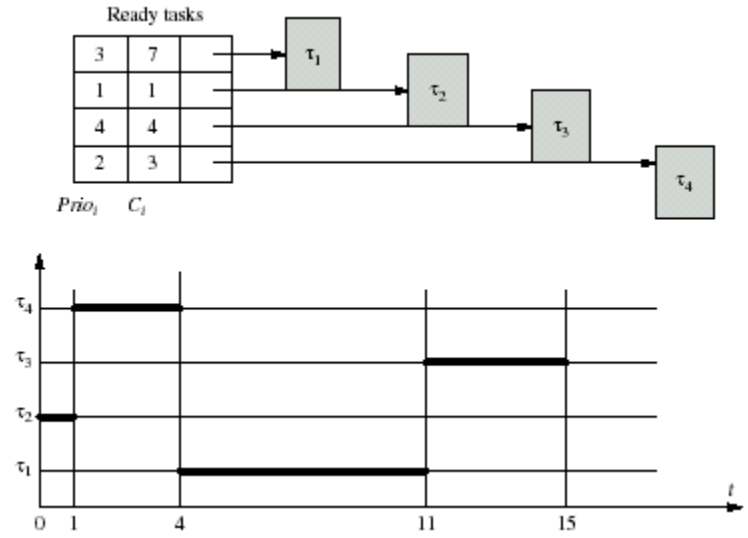
- *Χρονοπρογραμματισμός εκ περιτροπής (round robin):*ο αλγόριθμος εκχωρεί σε κάθε διεργασία ένα χρονικό διάστημα το οποίο ονομάζεται *κβάντο χρόνου (quantum time),*συνήθως μεταξύ από 10ms μέχρι 100ms, μέσα στο οποίο επιτρέπεται η εκτέλεσή της. Αν η διεργασία εξακολουθεί να εκτελείται στη στιγμή που τελειώνει το κβάντο χρόνου της η CPU παραχωρείται υποχρεωτικά σε άλλη διεργασία.



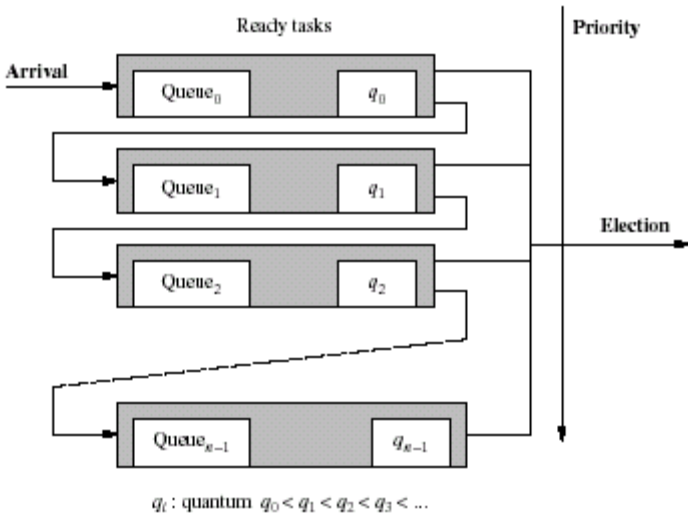
Σχήμα 2.4 Παράδειγμα Χρονοδρομολόγησης Round Robin

- *Σταθερή προτεραιότητα (constant priority):* οι αλγόριθμοι τέτοιας μορφής, εκχωρούν μια σταθερή προτεραιότητα σε κάθε διεργασία και αναθέτουν στη διεργασία που εκτελείται κάθε φορά το μέγιστο βαθμό προτεραιότητας. Οι αλγόριθμοι αυτοί λειτουργούν και με προεκτοπιστικό και με μη – προεκτοπιστικό τρόπο. Το μειονέκτημα που παρατηρείται σε αυτή τη περίπτωση είναι ότι μια διεργασία με χαμηλό βαθμό προτεραιότητας μπορεί να μην εκτελεσθεί ποτέ («λιμοκτονεί»).
- *Προτεραιότητα πολλαπλών επιπέδων (multilevel priority):* οι αλγόριθμοι τέτοιας μορφής, εκχωρούν προτεραιότητες σε λίστες που η κάθε μια περιέχει ένα πλήθος από διεργασίες. Σε μια συγκεκριμένη λίστα όλες οι διεργασίες που περιέχονται σε αυτήν έχουν τον ίδιο βαθμό προτεραιότητας και εξυπηρετούνται πρώτα οι παλιότερες κάθε λίστας με μη – προεκτοπιστικό

τρόπο. Ο χρονοδρομολογητής εξυπηρετεί πρώτα όλες τις διεργασίες της πρώτης λίστας (λίστα με αριθμό 0), όταν αδειάσει αυτή εξυπηρετεί αυτές της λίστας 1 κ.ο.κ. Είναι δυνατό να υπάρχει και κβάντο χρόνου από μία λίστα σε άλλη.



Σχήμα 2.5 Παράδειγμα Χρονοδρομολόγησης Σταθερής Προτεραιότητας



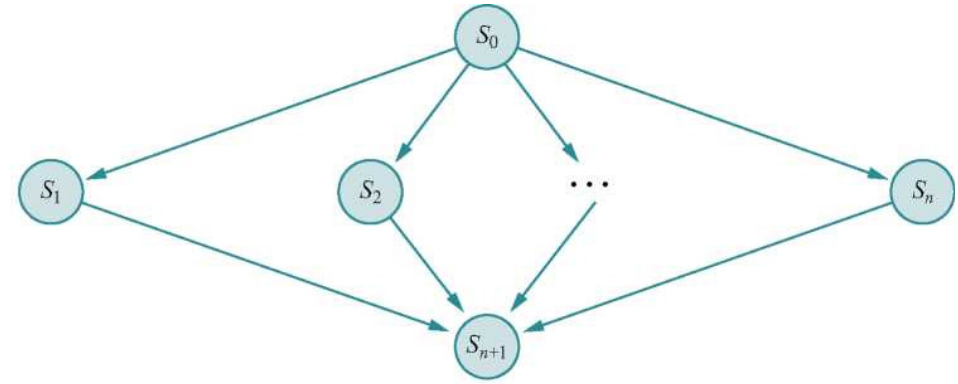
Σχήμα 2.6 Παράδειγμα Χρονοδρομολόγησης με Προτεραιότητα Πολλαπλών Επιπέδων

ΚΕΦΑΛΑΙΟ 3ο : ΑΜΟΙΒΑΙΟΣ ΑΠΟΚΛΕΙΣΜΟΣ ΣΤΑ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Δύο διαδικασίες λέγονται «παράλληλες» (concurrent) όταν οι εκτελέσεις τους επικαλύπτονται χρονικά.[2] Σε περιβάλλον πολυεπεξεργαστών, οι εντολές γλώσσας μηχανής δύο παραλλήλων διαδικασιών μπορεί να επικαλύπτονται πράγματι χρονικά (true parallelism). Αλλά σε περιβάλλον ενός μόνο επεξεργαστή, μια εντολή γλώσσας μηχανής κάποιας διαδικασίας μπορεί μόνο να «παρεμβληθεί» ανάμεσα σε δύο εντολές άλλων διαδικασιών. Εδώ γενικά υποθέτουμε ότι όλες οι δυνατές αναμείξεις των ακολουθιών των εντολών είναι δυνατόν να συμβούν (indeterminism). Έχει παρατηρηθεί ότι σε όλες τις πρακτικές περιπτώσεις μπορούμε να χρησιμοποιούμε την ιδέα του indetermision ως υπόθεση εργασίας. Τα λογικά αποτελέσματα ή προβλήματα φαίνονται να είναι ίδια και στις δύο περιπτώσεις (είτε κάνουμε την παραπάνω υπόθεση εργασίας είτε όχι). Τα προβλήματα αυτά προκαλούνται από την άγνοιά μας για τις σχετικές ταχύτητες των παράλληλων διαδικασιών. Στην περίπτωση όπου υπάρχει μόνο ένας επεξεργαστής στο σύστημα, θα χρησιμοποιούμε και τον όρο «ταυτόχρονα», για να δηλώσουμε ότι κάποιες διαδικασίες εκτελούνται «παράλληλα», με βάση τον πιο πάνω ορισμό των παραλλήλων διαδικασιών.

3.1 Εντολές Παραλληλισμού

Οι διαδικασίες που φτάνουν σε ένα επεξεργαστή για εξυπηρέτηση ακολουθούν μια τυχαία κατανομή που τις περισσότερες φορές δεν είναι προβλέψιμη. Επομένως, η σειρά με την οποία θα εκτελεστούν οι επιμέρους εντολές είναι άγνωστη ή, αλλιώς, μπορεί να εκτελεστούν σε οποιαδήποτε σειρά, αν δεν πάρει περιοριστικά μέτρα το Λειτουργικό Σύστημα. Παραδείγματος χάριν εστω οτι οι κώδικες s_1 ,s_2 ,..., s_n εκτελούνται «παράλληλα» (γενικά, όχι αναγκαστικά με την ίδια ταχύτητα).Το γράφημα προτεραιότητας της σειράς εκτέλεσης έχει ως εξής :



Σχήμα 3.1 : Παράλληλες διεργασίες

Από το σχήμα προκύπτει οτι η σειρά εκτέλεσης των διεργασιών μπορεί να είναι οποιαδήποτε . Για την επίτευξη αποδοτικής συνεργασίας παραλλήλων διεργασιών, απαιτείται να πληρούνται οι κάτωθι προϋποθέσεις:

- Δύο διεργασίες δε βρίσκονται ποτέ ταυτόχρονα στα κρίσιμα τμήματά τους
- Δεν επιτρέπονται υποθέσεις σε ότι αφορά την ταχύτητα ή το πλήθος των επεξεργαστών
- Διεργασία που δε βρίσκεται σε κρίσιμο τμήμα δεν επιτρέπεται να αναστείλει άλλες διεργασίες
- Δεν επιτρέπεται η έπ' αόριστο αναμονή μιας διεργασίας, για να εισέλθει στο κρίσιμο τμήμα της.

3.2 Η ανάγκη συντονισμού

Έστω ένα σύστημα στο οποίο εκτελούνται ταυτόχρονα δύο διαδικασίες κι ας υποθέσουμε ότι οι διαδικασίες αυτές έχουν κοινές μεταβλητές. Όταν μια από τις διαδικασίες αυτές αλλάζει μια ή περισσότερες από τις κοινές μεταβλητές, τότε το αποτέλεσμα που παράγεται από τη δεύτερη διαδικασία επηρεάζεται από τη σχετική ταχύτητα των δύο διαδικασιών.[2] Για παράδειγμα, θεωρούμε ότι οι διαδικασίες P_1, P_2 καταγράφουν τη θέση του cursor όταν αυτή αλλάζει ως αποτέλεσμα της μετακίνησής του με το ποντίκι (P_1) ή με χρήση του πληκτρολογίου (δηλαδή μετά από γράψιμο,

σβήσιμο ή μετακίνηση με τα βελάκια) (P_2). Τέτοιες διαδικασίες χρησιμοποιούνται σε «επεξεργασία κειμένου με αλληλεπίδραση» (interactive text processing). Η καθεμία από τις διαδικασίες P_i (i = 1, 2) εκτελεί τους ακόλουθους κώδικες για τις διαδικασίες P_1 και P_2 [θεωρούμε ότι διαδικασίες P_i (i = 1, 2) αναφέρονται στο ίδιο τερματικό, έστω T]:

BEGIN P_1	BEGIN P_2
Αν αλλάξει η θέση του cursor στο T, τότε	Αν αλλάξει η θέση του cursor στο T,
1. διάβασε τη θέση του cursor στη μεταβλητή χ_1	1. διάβασε τη θέση του cursor στη μεταβλητή χ_2
2 .γράψε την τιμή της χ_1 στη CURSORPOSITION	2. γράψε την τιμή της χ_2 στην CURSORPOSITION
END	END

Εδώ CURSORPOSITION είναι μια μεταβλητή (κοινή για τις δύο διαδικασίες) και αντιστοιχεί στη θέση του cursor στο τερματικό T. Το χ είναι (τοπική) μεταβλητή της διαδικασίας P_i. Έστω ότι μια τρέχουσα τιμή της μεταβλητής CURSORPOSITION είναι, π.χ., 653. Για να προσδιορίσουμε τη θέση του cursor στην οθόνη, πρέπει να ξέρουμε σε πόσες θέσεις, έστω X, μπορεί να μετακινηθεί σε μια γραμμή. Τότε, με το ακέραιο πηλίκο της τιμής της μεταβλητής CURSORPOSITION και του X θα βρούμε τη γραμμή στην οποία βρίσκεται και με το CURSORPOSITION mod X τη θέση του στη γραμμή. Έστω ότι ο cursor μετακινείται με το ποντίκι στη θέση 342, άρα η διαδικασία P_1 θα αλλάξει την τιμή της μεταβλητής CURSORPOSITION σε 342. Παράλληλα, ένα καινούριο γράμμα γράφεται σε ένα κείμενο, άρα η διαδικασία P_1 θα αλλάξει την τιμή της μεταβλητής CURSORPOSITION σε 654. Ας θεωρήσουμε ότι γράφεται το τελευταίο γράμμα σε ένα κείμενο και έπειτα μετακινείται ο cursor στην αρχή της τελευταίας σελίδας (π.χ. για να γίνει μια διόρθωση στη συνέχεια). Με βάση αυτή την περιγραφή, θα περιμέναμε ότι η τελική τιμή της μεταβλητής CURSORPOSITION θα ήταν 342. Έστω η ακόλουθη χρονική σειρά εκτέλεσης των αναμεμειγμένων εντολών των διαδικασιών P_j και P_2:

- P_2: διαβάζει τη θέση του cursor στη χ_2 ($\chi_2 = 654$)
- Εδώ η P_2 χάνει την CPU, γιατί τελειώνει ο στοιχειώδης χρόνος δρομολόγησής της. Η P_1 αρχίζει να τρέχει.
- P_1: διαβάζει τη θέση του cursor στη χ_1 ($\chi_1 = 342$)
- P_1: γράφει την τιμή της χ στην CURSORPOSITION (CURSORPOSITION = 342)
- Εδώ η P_1 χάνει την CPU, γιατί τελειώνει ο στοιχειώδης χρόνος δρομολόγησής της. Η P_2 αρχίζει να τρέχει.
- P_2: γράφει την τιμή της χ_2 στην CURSORPOSITION (CURSORPOSITION = 654)

Το αποτέλεσμα της παραπάνω σειράς γεγονότων είναι να μην καταγραφεί η σωστή τιμή στη μεταβλητή CURSORPOSITION. Το CURSORPOSITION γίνεται τελικά 654 αντί για 342.

Λάθη σαν τα παραπάνω έχουν το εξής χαρακτηριστικό: Επειδή εξαρτώνται από τις σχετικές ταχύτητες των διαδικασιών, γενικά δεν επαναλαμβάνονται / αναπαράγονται (δηλαδή δεν είναι το ίδιο αποτέλεσμα κάθε φορά που τρέχουν παράλληλα οι ίδιες διαδικασίες, άρα είναι μη ντετερμινιστικό το αποτέλεσμα). Αυτό κάνει πιο δύσκολα τα πράγματα, με αποτέλεσμα να μην μπορεί ο προγραμματιστής να τα ανιχνεύσει εύκολα και να τα διορθώσει. Ουσιαστικά, η διόρθωση τέτοιων προβλημάτων είναι σχεδόν αδύνατη.

Βλέπουμε, λοιπόν, ότι, όταν παράλληλες διαδικασίες αλληλεπιδρούν (μέσω κοινών μεταβλητών), πρέπει να ασκηθεί κάποιος έλεγχος στην αλληλεπίδρασή τους. Θα καλούμε «συντονισμό» (συγχρονισμό) γενικά κάθε περιορισμό στη σχετική διάταξη των εντολών (των διαδικασιών που αλληλεπιδρούν) στο χρόνο. Οι κοινές μεταβλητές γενικά παριστάνουν κοινούς «πόρους», όπως, π.χ., φυσικούς πόρους (μνήμη, I/O εργαλεία) ή μηνύματα. Στο προηγούμενο παράδειγμα η CURSORPOSITION ήταν ο «πόρος». Το λάθος έγινε επειδή οι P_1 και P_2 χρησιμοποίησαν το CURSORPOSITION «ταυτόχρονα» (ενώ η P_1 άρχισε να ασχολείται με την κοινή μεταβλητή, η P_2 παρεμβλήθηκε και άλλαξε την τιμή της). Το λάθος δε θα γινόταν αν υπήρχε ο περιορισμός του τύπου «εφόσον μια διαδικασία έχει πρόσβαση σε μία μεταβλητή, οι

άλλες διαδικασίες δεν μπορούν να έχουν προσπέλαση στη μεταβλητή αυτή». Αυτός ο περιορισμός (κανόνας συντονισμού) ονομάζεται «κανόνας αμοιβαίου αποκλεισμού» (mutual exclusion rule).

3.3 Κρίσιμο Τμήμα Διεργασίας και Αμοιβαίος αποκλεισμός

Σε ένα υπολογιστικό σύστημα,[2] υπάρχουν *μοιραζόμενοι πόροι* (shared resources) 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 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. 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). 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.

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.

3.4 Προτεινόμενες Λύσεις στο Πρόβλημα του Αμοιβαίου Αποκλεισμού

Στη παράγραφο αυτή θα μελετήσουμε τρόπους για την επίτευξη αμοιβαίου αποκλεισμού, έτσι ώστε όταν μια διεργασία μεταβάλει τα περιεχόμενα διαμοιραζόμενης περιοχής μνήμης, καμία άλλη δεν εισέρχεται στο κρίσιμο τμήμα της. Ας θεωρήσουμε δύο κυκλικές διαδικασίες (οι διαδικασίες αυτές επαναλαμβάνουν συνέχεια ένα κομμάτι κώδικα) P και Q, που συχνά χρησιμοποιούν τον κοινό πόρο (μεταβλητή) R. Η καθεμία από τις διαδικασίες P και Q χρησιμοποιούν τον R μόνο για περιορισμένο χρόνο (κάθε φορά που έχουν προσπέλαση στον R). Το πρόβλημα που εισάγει ο αμοιβαίος αποκλεισμός είναι ο τρόπος με τον οποίο θα εξασφαλιστεί ο ακόλουθος περιορισμός. Οι διαδικασίες P και Q δεν πρέπει να χρησιμοποιήσουν τον κοινό πόρο R «ταυτόχρονα». Αυτό ορίζει τον κανόνα του αμοιβαίου αποκλεισμού, όπως αναφέραμε σε προηγούμενη ενότητα.

3.4.1 Πρώτη Λύση (Χρήση μεταβλητής κλειδώματος)

Θα δοκιμάσουμε πρώτα να συντονίσουμε τις διαδικασίες P και Q μέσω μιας κοινής μεταβλητής κλειδώματος «free» που δείχνει αν ο πόρος είναι διαθέσιμος.[2] Αν δεν είναι, (free = False), η διαδικασία περιμένει. Αν είναι διαθέσιμος (free = True), η διαδικασία τον μαρκάρει ως μη διαθέσιμο και τον χρησιμοποιεί. Όταν τελειώσει, πάλι μέσω της μεταβλητής free, δηλώνει ότι ο πόρος είναι ελεύθερος.

var *free* : shared boolean ;
begin
free := true ;
cobegin

“P” repeat	**“ Q” repeat**
repeat until *free;*	**repeat until *free;***
***free* := false ;**	***free* := false ;**
[χρήση του *R*]	**[χρήση του *R*]**
free := true ;	**free := true ;**
forever	**forever**

coend

end

Αρχικά η *free* είναι αληθής. Συνεπώς είναι δυνατόν η P και η Q να αναφερθούν «ταυτόχρονα» στη *free* και να την βρουν αληθή. (Αυτό, π.χ., θα γίνει εάν οι P και Q εκτελέσουν το «repeat until *free*» η μία μετά την άλλη). Αν γίνει αυτό, και οι δύο διαδικασίες θα κάνουν τη *free* ψευδή (η μία μετά την άλλη) και θα χρησιμοποιήσουν την R ταυτόχρονα. Δηλαδή η «λύση» δεν έχει μεγάλη αποτελεσματικότητα.

3.4.2 Δεύτερη Λύση (Μέθοδος της Αυστηρής Εναλλαγής)

Με τη χρήση της μεθόδου της Αυστηρής Εναλλαγής [2] κάθε διαδικασία θα παίρνει τον πόρο εναλλάξ, με τη σειρά της. Αντί της *free,* θα χρησιμοποιήσουμε τη μεταβλητή

P_{turn} ,που δείχνει αν είναι η σειρά της P να χρησιμοποιήσει τον R. Αυτό συμβαίνει όταν P_{turn} = True (και είναι False όταν είναι η σειρά της Q) :

```
var Pturn : shared boolean ;
begin
Pturn := true;
cobegin

"P" repeat                          "Q" repeat
repeat until Pturn ;                repeat until (not Pturn) ;
[χρήση του R]                       [χρήση του R]
Pturn := false ;                    Pturn := true ;
forever                             forever

coend

end
```

Στη λύση της αυστηρής εναλλαγής, η μεταβλητή **Pturn** καταγράφει τη διεργασία που έχει δικαίωμα να εισέλθει στο κρίσιμο τμήμα της, ελέγχοντας ή ενημερώνοντας τη μνήμη. Αρχικά, η **Pturn** έχει τιμή true, οπότε η διεργασία P εισέρχεται στο κρίσιμο τμήμα, ενώ η διεργασία Q περιμένει, εκτελώντας διαρκώς ένα βρόχο ελέγχου. Η διεργασία Q βρίσκεται σε ενεργό αναμονή (busy waiting). Όταν η διεργασία P τελειώσει το κρίσιμο τμήμα, μεταβάλλει την **Pturn** σε false, οπότε επιτρέπει στην αντίστοιχη διεργασία να εισέλθει στο δικό της κρίσιμο τμήμα. Η διαδικασία πετυχαίνει αμοιβαίο αποκλεισμό, επειδή η **Pturn** μπορεί να είναι μόνο true (ή μόνο false) σε μια δεδομένη χρονική στιγμή. Αλλά η λύση αυτή αναγκάζει το ΛΣ να παρέχει τον πόρο R στις P και Q εναλλάξ, ανεξάρτητα από τις ανάγκες τους, και γι' αυτό δεν είναι αποδεκτή. Αν, π.χ., είναι η σειρά της P να χρησιμοποιήσει τον R, αλλά δεν τον χρειάζεται ακόμη επειδή έχει να κάνει ένα μεγάλο υπολογισμό, τότε η Q περιμένει χωρίς λόγο, ενώ ο R μένει σε αχρηστία. Ένα άλλο ανεπιθύμητο χαρακτηριστικό της λύσης αυτής είναι η σπατάλη χρόνου που γίνεται στα μη παραγωγικά loops (στα repeat until…), κατάσταση που ονομάζεται πολυάσχολη αναμονή (busy waiting).

3.4.3 Τρίτη Λύση (Μέθοδος της Εκ των προτέρων Δήλωσης Πρόθεσης Χρησιμοποίησης Πόρου)

Μία τρίτη λύση του προβλήματος [2] με στόχο να αποφύγουμε την αυστηρή εναλλαγή, είναι χρησιμοποιώντας δύο boolean μεταβλητές οι οποίες δείχνουν ποιες διαδικασίες έχουν πρόθεση να χρησιμοποιήσουν τον κοινό πόρο. Η λύση αυτή λέει: Πριν χρησιμοποιήσεις τον κοινό πόρο (α), δήλωσε την πρόθεση σου (β)· αν κανείς άλλος δεν έχει δηλώσει όμοια πρόθεση, τότε χρησιμοποίησε τον πόρο και, όταν τελειώσεις, ανακάλεσε την πρόθεση χρήσης.

var Pturn , Qturn : shared boolean ;

begin

Pturn : false ; Qturn := false ;

Cobegin

“P” repeat	**“Q” repeat**
Pturn := true ;	**Qturn := true ;**
repeat until (not Qturn) ;	**repeat until (not Pturn) ;**
[χρήση της R] ;	[χρήση της R];
Pturn := false ;	**Qturn := false ;**
…..	**….**
forever	**forever**

coend

end

Και η λύση όμως της εκ των προτέρων δήλωσης χρησιμοποίησης πόρων παρουσιάζει προβλήματα με σημαντικότερο αυτό του αδιεξόδου. Το πρόβλημα δημιουργείται επειδή κάθε διαδικασία δηλώνει την πρόθεσή της να χρησιμοποιήσει τον πόρο (Pturn = true) και περιμένει την άλλη διαδικασία να ανακαλέσει την αντίστοιχη δήλωσή της. Αυτό μοιάζει με δύο ανθρώπους που επιχειρούν να καλέσουν ο ένας τον άλλον

ταυτόχρονα και, επειδή βρίσκουν τη γραμμή κατειλημμένη, ξανακαλούν συνεχώς. Αν περίμεναν για κάποιο χρονικό διάστημα (τυχαίο και διαφοερτικό για τον καθένα), τότε, κατά πάσα πιθανότητα, το πρόβλημα θα λυνόταν.

3.4.4 Τέταρτη λύση (Η λύση του Dekker)

Το 1965 ο Dekker πρότεινε την παρακάτω λύση στο πρόβλημα :

var outside 1, outside 2 : shared boolean ;
turn : shared 1…2 ;
begin
***outside* 1 : true ; *outside* 2 := true ; *turn* := 1 ;**

cobegin

"P" repeat	**"Q" repeat**
begin	**begin**
repeat	**repeat**
***outside* 1:= false ;**	***outside* 2:= false ;**
repeat	**repeat**
if *outside* 2 then go to enter ;	**If *outside* 1 then go to enter ;**
until *turn* = 2 ;	**until *turn* = 1 ;**
***outside* 1: = true ;**	***outside* 2: = true ;**
repeat until *turn* = 1 ;	**repeat until *turn* = 2 ;**
forever ;	**forever ;**
end	**end**
enter P inside ;	**enter Q inside**
[χρήση του R] ;	[χρήση του R] ;
***turn*: = 2 ;**	***turn*: = 1**
***outside* 1:= true ;**	***outside* 2:= true ;**

P outside ; **Q outside ;**

forever **forever**

coend

end

Η λύση του Dekker [Α.Δ 2] έλυσε το πρόβλημα του αμοιβαίου αποκλεισμού. Η λύση του Dekker λειτουργεί ως εξής: Υπάρχει μια κοινή μεταβλητή με όνομα turn, 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 outside1 και outside2. Η μεταβλητή outside1 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. Αντίστοιχες τιμές παίρνει και η μεταβλητή eisodos2, για τη διεργασία 2. 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 outside2=false), τότε η διεργασία 1 μπαίνει στο δικό της κρίσιμο τμήμα, εμποδίζοντας τη 2 με τη βοήθεια της outside1. Αν και οι δυο διεργασίες θέλουν ταυτόχρονα να εκτελέσουν το κρίσιμο τμήμα τους, το εκτελεί πρώτα εκείνη που υποδεικνύεται από τη μεταβλητή turn.

3.5 Σηματοφορεις

Η λύση του Dekker που αναπτύξαμε στην προηγούμενη παράγραφο είναι σωστή αλλά έχει δύο μειονοκτήματα :

1. Προσθέτει πολλές μεταβλητές και επιπλέον εντολές στον κώδικα της διεργασίας κάνοντάς τον πολύπλοκο χωρίς λόγο.
2. Δεν επεκτείνεται εύκολα για περισσότερες από δυο διεργασίες, όπου είναι ακόμα πιο πολύπλοκη.

Για να εξηγήσουμε το πρόβλημα θα δώσουμε ένα ανάλογο από την καθημερινή ζωή[10]. 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 , 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. Μια τόσο πολύπλοκη και αργή διαδικασία προφανώς θα δημιουργούσε προβλήματα στην λειτουργικότητα του συστήματος. Στην πραγματικότητα αυτό που συμβαίνει είναι 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. Οι δύο σηματοφορείς εκτελούν δύο λειτουργίες :

Λειτουργία P : **while s = 0 do_nothing;**
s := s - 1;

Λειτουργία V: **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.

3.5.1 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.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Παπακωσταντίνου Γ., Ν. Μπιλάλη, Π. Τσανάκα, Λειτουργικά Συστήματα: Αρχές Λειτουργίας, Συμμετρία, 1989.

[2] Σπυράκης Παύλος , Λειτουργικά Συστήματα Ι, Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα 2001.

[3] Andrew S. Tanenbaum, Σύγχρονα Λειτουργικά Συστήματα, Κλειδάριθμος 2009

[4] Stallings, Operating Systems: Internals and Design Principles.

[5] Γ.Α. Παπαδόπουλος, Λειτουργικά Συστήματα, Πανεπιστήμιο Κύπρου

[6] Κ. Διαμαντάρας ,Λειτουργικά Συστήματα, , ΤΕΙΘ

[7] Silberschatz, Galvin and Gange, Operating Systems Concepts.

[8] Whitman M., Mattord H ,*Principles of Information Security*,(2011), 4th Edition

[9] Κάτσικα Σ. Γκρίτζαλη Δ., «Ασφάλεια Πληροφοριακών Συστημάτων», Εκδόσεις Νέων Τεχνολογιών, 2004. ISBN13: 978-960-8105-57-7

[10] Κατσούλας Νικόλαος, Δρ. Όροβας Χρήστος, Παναγιωτίδης Σωτήρης, Λειτουργικά Συστήματα και Ασφάλεια Πληροφοριακών Συστημάτων, Διόφαντος

Αναφορές Διαδυκτύου [Α.Δ.χ]

[1] http://www.icsd.aegean.gr/lecturers/ckaraf/OperSyst/ABS_OperSyst_2up_bw.pdf

[2] http://www2.cs.ucy.ac.cy/~nicolast/courses/lectures/OSprinciples.pdf

[3] https://el.wikipedia.org/wiki/

[4] http://eclass.sch.gr/modules/document/file.php/T489103/%CE%9B.%CE%A3.pdf

[5] http://www.iep.edu.gr/images/school_books/

[6] http://ebooks.edu.gr/modules/ebook/show.php/DSB103/173/1211,4428/

[7] https://eclass.uoa.gr/modules/document/index.php?course=D454