ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

ΤΟΜΕΑΣ ΟΡΓΑΝΩΣΗΣ & ΔΙΟΙΚΗΣΗΣ

Μεταπτυχιακή Ερευνητική Διπλωματική Εργασία

"Μοντελοποίηση ροών εργασίας και νοητικού φόρτου εργασίας για τον σχεδιασμό συστημάτων ανθρώπου-μηχανής"

από τον **Νικόλαο Γιαννακάκη**

Επιβλέπων: Αναπληρωτής Καθηγητής Κοντογιάννης Θωμάς.

Εξεταστική Επιτροπή:
Ματσατσίνης Νικόλαος
Μουστάκης Βασίλειος

Χανιά, Σεπτέμβριος 2004

Περίληψη

Η μοντελοποίηση ροών εργασίας περιλαμβάνει μια σειρά τεχνικών μεθόδων για τον σχεδιασμό νέων συστημάτων εργασίας ή την βελτιστοποίηση υπαρχόντων συστημάτων. Σε πολλές τεχνικές δεν εξετάζεται αναλυτικά ο ρόλος τους ανθρώπινου παράγοντα, όπως η δυνατότητα εκτέλεσης παράλληλων εργασιών, η αλλαγή προτεραιοτήτων στη σειρά εκτέλεσης, η διακοπή εργασιών και η ενδεχόμενη παράλειψή τους (π.χ. περιορισμοί ανθρώπινης μνήμης). Αλλαγές στη σειρά εκτέλεσης των εργασιών που υπαγορεύονται από τις ανάγκες των εργαζομένων είναι πολύ σημαντικές για την απόδοση του συνολικού συστήματος. Οι αλλαγές αυτές μπορούν να αυξήσουν την απόδοση του συστήματος (π.χ. εάν οι εργασίες γίνονται παράλληλα) ή μπορεί να οδηγήσουν σε ανθρώπινα λάθη (π.χ. εάν αυξηθεί ο νοητικό φόρτος). Είναι ανάγκη λοιπόν να γίνει μία εργονομική ανάλυση εργασίας και να μελετηθεί ο νοητικός φόρτος εργασίας.

Ο σκοπός της ερευνητική αυτής εργασίας είναι να εξετάσει πιθανούς τρόπους μοντελοποίησης νοητικού φόρτου εργασίας, της διαδικασίας καθορισμού προτεραιοτήτων και της διαδικασίας διακοπή και απομνημόνευσης εργασιών προκειμένου να βελτιστοποιηθεί ένα σύστημα ανθρώπου-μηχανής. Ένας δεύτερος στόχος είναι η ανάπτυξη modules σε C++ ώστε να διευκολύνεται η χρήση των νοητικών αυτών διαδικασιών σε ποικίλες πρακτικές εφαρμογές.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Αναπληρωτή Καθηγητή κ. Θωμά Κοντογιάννη για την επίβλεψή του στην ολοκλήρωση αυτής της μεταπτυχιακής εργασίας, καθώς και για την βοήθεια και τις συμβουλές του καθ' όλη τη διάρκεια της εκπόνησής της.

Θα ήθελα επίσης εκ των προτέρων να ευχαριστήσω τον Αναπληρωτή Καθηγητή κ. Μουστάκη Βασίλειο και τον Αναπληρωτή Καθηγητή κ. Ματσατσίνη Νικόλαο για τον χρόνο που θα διαθέσουν για να αναγνώσουν την παρούσα εργασία.

Γιαννακάκης Νίκος

Χανιά
Σεπτέμβρης 2004

Περιεχόμενα

Πίνακας Σχημάτων

Εισαγωγή

Α. Γενικά για την εργασία

Η μοντελοποίηση ροών εργασίας περιλαμβάνει μία σειρά από τεχνικές οι οποίες έχουν σαν στόχο είτε τον σωστό και αποδοτικό σχεδιασμό νέων συστημάτων, είτε την βελτιστοποίηση ως προς την λειτουργικότητα και αποδοτικότητα υπαρχόντων συστημάτων εργασίας Στα συστήματα αυτά κεντρικό ρόλο έχει ο ανθρώπινος παράγοντας. Είτε από την πλευρά του σχεδιαστή του συστήματος είτε από την πλευρά του εργαζόμενου σε αυτό η σημασία του ανθρώπινου παράγοντα είναι καίρια.

Ο ρόλος του σχεδιαστή ενός συστήματος εργασιών περιλαμβάνει κατά κύριο λόγο τη μελέτη και το σχεδιασμό του συστήματος κατά τρόπο ώστε αυτό να είναι όσο το δυνατόν αποδοτικότερό. Ο σχεδιαστής πρέπει να λάβει υπόψιν του όλες τις παραμέτρους του συστήματος όπως τη μεθοδολογία ανάθεσης εργασιών, σε ποια άτομα δηλαδή και πότε ανατίθενται οι εργασίες, αν υπάρχει παραλληλία στην εκτέλεση, αν ο φόρτος των εργαζομένων σε παράλληλες εργασίες είναι υψηλός και μπορεί να οδηγήσει σε σφάλματα ή διακοπή εργασίας. Με τον τρόπο αυτό ο σχεδιαστής μπορεί να ελέγξει σε πρώιμο στάδιο την αποτελεσματικότητα του συστήματος.

Μία από τις σημαντικές παραμέτρους που πρέπει να λάβει υπόψιν του ο σχεδιαστής είναι και ο φόρτος εργασίας. Πάρα πολλές μελέτες, μέθοδοι και μοντέλα έχουν χρησιμοποιηθεί για να προσδιοριστεί αποτελεσματικά ο φόρτος εργασίας. Οι μέθοδοι αυτές κατατάσσονται σε διάφορες κατηγορίες ανάλογα με τον εάν στηρίζονται στην εμπειρία ή εάν βασίζονται στην ανάλυση ενός δεδομένου εργασιακού συστήματος. Μια ευρύτατα αποδεκτή μέθοδος προσδιορισμού του φόρτου εργασίας είναι η θεωρία πολλαπλών πόρων και οι παραλλαγές της.

Η θεωρία πολλαπλών πόρων στηρίζεται στον διαχωρισμό των καναλιών πληροφορίας του ατόμου για να προσδιορίσει τον φόρτο εργασίας. Με βάση την θεωρία αυτή

προσπαθήσαμε να προσεγγίσουμε τον τρόπο λειτουργίας διάφορων δομών εργασιών και τον φόρτο τον οποίο αυτές επιφέρουν στους εργαζόμενους ανάλογα με τον τρόπο με τον οποίο γίνεται η ανάθεση των εργασιών σε αυτούς. Λαμβάνοντας υπόψιν πιθανά σενάρια που μπορεί να εμφανιστούν κατά την λειτουργία του συστήματος, όπως αλλαγή κάποιας εργασίας ή αλλαγή εργαζομένου συγκεκριμένης εργασίας, κατασκευάσθηκε ένα εργασιακό μοντέλο, αρκετά γενικό, στο οποίο είμαστε σε θέση ανά πάσα στιγμή να γνωρίζουμε το νοητικό φόρτο των εργαζομένων και να λαμβάνουμε με βάση αυτόν αποφάσεις για τον σχεδιασμό και τη λειτουργία του συστήματος.

Για την εφαρμογή του μοντέλου χρησιμοποιήθηκαν διαφορετικές περιπτώσεις εργασιακών υποσυστημάτων όπως σειριακή εκτέλεση εργασιών, παράλληλη εκτέλεση εργασιών ή παράλληλη και σειριακή εκτέλεση. Επίσης λαμβάνεται υπόψιν η υπό συνθήκη ανάθεση εργασιών και η εξειδίκευση των εργαζομένων του συστήματος.

Το μειονέκτημα του συγκεκριμένου μοντέλου είναι το ότι δεν είναι φιλικό προς τον χρήστη. Αυτό οφείλεται στο γεγονός ότι ενώ αρχικά είχε σχεδιαστεί σε εξειδικευμένο πρόγραμμα προσομοιώσεων, διαπιστώθηκε ότι το συγκεκριμένο πρόγραμμα έχει περιορισμένες δυνατότητες σε ορισμούς και χρήση δομών δεδομένων. Για το λόγο αυτό χρησιμοποιήθηκε η γλώσσα προγραμματισμού C++ που παρέχει μεγάλες δυνατότητες στο χειρισμό δομών δεδομένων. Μια μελλοντική επέκταση της παρούσας εργασίας μπορεί να είναι και η υλοποίηση της για να καλύπτει ευρύ φάσμα εργασιακών δομών κατά τέτοιο τρόπο ώστε να επιτρέπει στο χρήστη να ελέγχει από μόνος του τις παραμέτρους του συστήματος.

Β. Δομή της εργασίας

Η εργασία αυτή έχει την ακόλουθη δομή:

- Στο $1^{ο}$ κεφάλαιο παρατίθεται ένας γενικός ορισμός του νοητικού φόρτου εργασίας και γίνεται μία βιβλιογραφική ανασκόπηση και παράθεση των προσεγγίσεων, εμπειρικών και αναλυτικών, που χρησιμοποιούνται για τον προσδιορισμό του.
- Στο δεύτερο κεφαλαίο αναλύεται η θεωρία πολλαπλών πόρων και η εφαρμογή της στον προσδιορισμό του νοητικού φόρτου. Γίνεται αναλυτική αναφορά στο τετραδιάστατο μοντέλο πολλαπλών πόρων και στις παραμέτρους του.

- Στο 3^{ο} κεφάλαιο αναλύεται γενικά το παράδειγμα εργασίας που θα μελετηθεί. Στη συνέχεια γίνεται μία ποιοτική ανάλυση του κώδικα σε C++ με αναφορά αρχικά στις κλάσεις που κατασκευάσθηκαν και στην συνέχεια στις συναρτήσεις. Επιπλέον αναλύονται οι αλγόριθμοι δρομολόγησης εργασιών οι οποίοι υλοποιήθηκαν σε κώδικα C++.
- Στο 4^{ο} κεφάλαιο της εργασίας αναλύονται διεξοδικά τα τρία modules που κατασκευάσθηκαν και προσομοιώθηκαν Για το κάθε σύστημα παρατίθενται εικόνες από την προσομοίωση καθώς και τα αποτελέσματα αυτής. Τέλος αναφέρονται και τα συνολικά συγκριτικά αποτελέσματα των τριών συστημάτων.

Κεφάλαιο 1

Φόρτος και νοητικός φόρτος εργασίας – Βιβλιογραφική ανασκόπηση προσεγγίσεων

1. Ορισμός και αρχές του Φόρτου Εργασίας

Ο νοητικός φόρτος σχετίζεται γενικά με εργασίες που αφορούν την επεξεργασία πληροφορίας, αλλά οποιαδήποτε ανθρώπινη δραστηριότητα συμπεριλαμβάνει νοητική επεξεργασία και επομένως νοητικό φόρτο [NF1995]. Η οδήγηση ενός αυτοκινήτου για παράδειγμα προκαλεί ένα συγκεκριμένο επίπεδο νοητικού φόρτου για τον οδηγό. Ο οδηγός πρέπει να ελέγχει το δρόμο, να διατηρεί τον έλεγχο του οχήματος μέσω διατήρησης της σωστής πίεσης στο γκάζι και στα φρένα και να προσέχει τα υπόλοιπα αυτοκίνητα. Όλες αυτές οι διεργασίες απαιτούν νοητική επεξεργασία και αυξάνουν το νοητικό φόρτο του οδηγού. Ο οδηγός του αυτοκινήτου θα απέδιδε αποτελεσματικότερα εφόσον ο νοητικός του φόρτος ελαχιστοποιείται κατά την εκτέλεση της διεργασίας.

Η ελαχιστοποίηση του φόρτου είναι τόσο σημαντική, ώστε να θεωρηθεί ένας πολύ σημαντικός στόχος για τον σωστό σχεδιασμό ενός συστήματος. Για τον λόγο αυτό αν κάποιος σχεδιαστής συστημάτων θέλει να επιτύχει έναν καλό σχεδιασμό, θα πρέπει να κατανοήσει όχι μόνο τις αρχές του φόρτου εργασίας αλλά και το τι σημαίνει κατάλληλος ή βέλτιστος φόρτος.

Το 1991 η Hart έδωσε τον ορισμό του κατάλληλου φόρτου ως μία κατάσταση κατά την οποία ο εργαζόμενος νιώθει άνετα, μπορεί να χειρίζεται τις απαιτήσεις της εργασίας με έξυπνο τρόπο και να διατηρεί μία καλή αποδοτικότητα. Χρησιμοποιεί ένα λειτουργικό ορισμό του φόρτου αντί για καθορισμό των αρχών του και αυτό γιατί πολλοί ερευνητές δεν συμφωνούν σε έναν ορισμό του φόρτου [HSG2001].

Παραδοσιακά, ο φόρτος εργασίας έχει οριστεί σε σχέση με [HBMWCD1993], [MN1998]:

1) Τις επιβεβλημένες απαιτήσεις της εργασίας
2) Το επίπεδο αποδοτικότητας
3) Τη νοητική και σωματική προσπάθεια που καταβάλλεται από τον εργαζόμενο
4) Την αντίληψη του εργαζομένου

Περιγράφοντας με αναλυτικότερο τρόπο τα παραπάνω μπορούμε να πούμε ότι ο φόρτος εργασίας αποτελεί την ποσότητα της εργασίας που ο εργαζόμενος έχει να επιτελέσει ή αναμένεται από αυτόν να ολοκληρώσει. Όμως δεν είναι απόλυτα το συγκεκριμένο ποσό καθώς μπορεί επιπλέον να φανερώνει:

- Το πόσο δύσκολή είναι η εργασία. Η δυσκολία αυτή επηρεάζεται από πολλούς παράγοντες μεταξύ των οποίων είναι το επίπεδο των ικανοτήτων του εργαζομένου.
- Το κατά πόσον το μέγεθος του φόρτου μεταβάλλεται, περιλαμβάνοντας δηλαδή γεμάτες δραστηριότητα και ηρεμότερες περιόδους.
- Το βαθμό στον οποίο ο εργαζόμενος ελέγχει το φόρτο που έχει και τον τρόπο με τον οποίο επιλέγει να ολοκληρώσει την εργασία του.
- Το πόσο εξοικειωμένος είναι ο εργαζόμενος με την εργασία που του έχει ανατεθεί.
- Τη χρονική διάρκεια κατά την οποία ο εργαζόμενος λειτουργεί σε εντατικούς ρυθμούς χωρίς διαλείμματα.

Επιπλέον η φύση του φόρτου εργασίας αλλάζει, ανάλογα με την εργασία, και είναι σημαντικό να υπάρχει γνώση των ειδικών χαρακτηριστικών που παρουσιάζει η κάθε εργασία. Για παράδειγμα κάποιες εργασίες είναι επαναληπτικές, άλλες απαιτούν δοσοληψίες με άλλους ανθρώπους ενώ άλλες απαιτούν από τον εργαζόμενο να αναλαμβάνει με σταθερό ρυθμό καινούργια καθήκοντα στα οποία δεν είναι εξοικειωμένος [LRACBR2002].

Οι σύγχρονοι φυσιολόγοι χρησιμοποιούν ορισμούς παρόμοιους με αυτούς της Hart (1991), οι οποίοι θεωρούν ότι ο φόρτος είναι αποτέλεσμα κάποιου συνδυασμού ενός συγκεκριμένου εργαζομένου και της εργασίας που έχει αναλάβει.

Οποιοδήποτε ορισμό όμως για το φόρτο προτιμήσουμε, ο στόχος του σχεδιαστή ενός συστήματος είναι η βελτιστοποίηση της απόδοσης μέσω ελέγχου και ρύθμισης του φόρτου. Ο σχεδιαστής βελτιστοποιεί το φόρτο μειώνοντάς τον ακριβώς χαμηλότερα

από το επίπεδο στο οποίο επιφέρει αρνητικά αποτελέσματα στην αποδοτικότητα. Ο στόχος αυτός όμως μπορεί να είναι ιδιαίτερα δύσκολος στην επίτευξη του, επειδή κάποιες φορές υπάρχει αναντιστοιχία μεταξύ του φόρτους και της απόδοσης. Αυτή η αναντιστοιχία σημαίνει ότι ο νοητικός φόρτος ενός ατόμου μπορεί να αυξηθεί χωρίς όμως να επιφέρει οποιαδήποτε επίδραση στην απόδοση του συγκεκριμένου ατόμου. Αρκετές θεωρίες που αναπτύχθηκαν για το φόρτο, προσπάθησαν να λάβουν υπόψιν αυτή την αναντιστοιχία και να ορίσουν τη φύση της συσχέτισης φόρτου και απόδοσης. Η κατανόηση αυτών των θεωριών μπορεί να βοηθήσει σημαντικά τον σχεδιαστή ενός συστήματος στην βελτιστοποίηση των συστημάτων που σχεδιάζει καθώς θα μπορεί να ερμηνεύσει τον συσχετισμό φόρτου και απόδοσης στο σύστημά του.

2. Νοητικός Φόρτος Εργασίας

Το γεγονός είναι ότι δεν υπάρχει γενική αποδοχή του ορισμού του νοητικού φόρτου. Πάρα πολλοί ορισμοί έχουν προταθεί και αρκετοί από αυτούς φαίνονται διαισθητικά σωστοί. Παρόλα αυτά, όλοι οι σύγχρονοι ορισμοί αποτυγχάνουν να έχουν ευρεία αποδοχή ή ποιοτική τεκμηρίωση. Οι Gopher και Donchin (1986) υποστήριξαν ότι ο νοητικός φόρτος μπορεί να εκτιμηθεί καλύτερα ως μια «υποθετική έννοια» αντί για μία «παρεμβατική μεταβλητή» [GDDE1986]. Μία τέτοια μεταβλητή αποτελεί μία θεωρητική έννοια η οποία είναι απλά μία ποσότητα που λαμβάνεται μέσω εφαρμογής μίας συγκεκριμένης μεθοδολογίας στις τιμές των εμπειρικών μεταβλητών. Δεν περιλαμβάνει καμία υπόθεση αναφορικά με την ύπαρξη απρόβλεπτων οντοτήτων ή φαινομένων απρόβλεπτων διαδικασιών. Όμως, ο νοητικός φόρτος συμπεριλαμβάνει το επιπρόσθετο χαρακτηριστικό των υποθετικών εννοιών που σχετίζονται με όρους οι οποίοι δεν ανάγονται πλήρως σε εμπειρικούς όρους και αναφέρονται σε διαδικασίες ή οντότητες που δεν είναι άμεσα παρατηρήσιμες [GDKR1989]. Επιπλέον αυτός ο υπερβατικός ορισμός θεωρείται ότι μπορεί να συλληφθεί, να μελετηθεί και να μετρηθεί με τρόπους οι οποίοι θα προωθήσουν την κατανόηση του συστήματος και θα καταστήσουν εφικτή τη χρήση της συγκεκριμένης έννοιας σε πρακτικές δραστηριότητες.

Για να διευκολυνθεί η κατανόηση της έννοιας του νοητικού φόρτου, αναφέρονται συνοπτικά κάποιες γενικές αρχές που τον διέπουν:

- Σε απλή γλώσσα, ο νοητικός φόρτος είναι το ποσό της πνευματική εργασίας ή προσπάθειας που είναι απαραίτητο να καταβληθεί από ένα άτομο ή μία ομάδα ατόμων, για να ολοκληρωθεί μία συγκεκριμένη εργασία μέσα σε ένα προκαθορισμένο χρονικό διάστημα.

- Ο νοητικός φόρτος δεν μπορεί να ανιχνευθεί απευθείας. Μπορεί όμως αυτό να γίνει μέσω της μέτρησης κάποιων άλλων μεταβλητών και παραμέτρων οι οποίες θεωρούνται ότι σχετίζονται ισχυρά με αυτόν. Τέτοιες παράμετροι είναι: η υποκειμενική αξιολόγηση του ατόμου, η αποδοτικότητα του και διάφορα δεδομένα που σχετίζονται με τη φυσιολογία.

- Ο νοητικός φόρτος περιλαμβάνει τόσο στατικές όσο και δυναμικές ιδιότητες, οι οποίες αντικατοπτρίζουν το νοητικό φόρτο κατά μήκος ενός χρονικού διαστήματος ή μία συγκεκριμένη χρονική στιγμή αντίστοιχα.

- Κάθε άτομο έχει περιορισμένες δυνατότητες και περιορισμένους πόρους για την εκτέλεση μιας εργασίας. Ο νοητικός φόρτος περιλαμβάνει την μείωση των πόρων αυτών για την ολοκλήρωση της εργασίας. Ο υψηλός νοητικός φόρτος εξαντλεί ταχύτερα τους πόρους αυτούς σε σχέση με το χαμηλό νοητικό φόρτο. Οι απαιτήσεις για πόρους ενδέχεται να μην ισορροπούν κατά την εκτέλεση ενός καθήκοντος. Κάποιοι πόροι (π.χ. η αντίληψη) μπορεί να παραμένουν ελάχιστα φορτωμένοι ενώ άλλοι πόροι (π.χ. μνήμη) να υπόκεινται σε υπερφόρτωση.

- Ο νοητικός φόρτος είναι μία πολυδιάστατη μεταβλητή. Οι Reid και Nygren (1988) ταξινόμησαν το νοητικό φόρτο σε τρεις διαστάσεις [RGBNTE1988] :
    1. χρονικό φορτίο
    2. φορτίο νοητικής προσπάθειας
    3. φορτίο ψυχολογικής πίεσης

  Οι Hart και Staveland (1988) υπολόγισαν το νοητικό φόρτο από έξι σημαντικές πλευρές [HSGSLE1988]:
    1. πνευματικές απαιτήσεις
    2. ψυχολογικές απαιτήσεις
    3. χρονικές απαιτήσεις
    4. αποδοτικότητα
    5. βαθμός απογοήτευσης
    6. προσπάθεια

Ο νοητικός φόρτος μπορεί ξεκάθαρα να ληφθεί μέσω διαφορετικών διαστάσεων η βαρύτητα των οποίων μεταβάλλεται από περίπτωση σε περίπτωση.

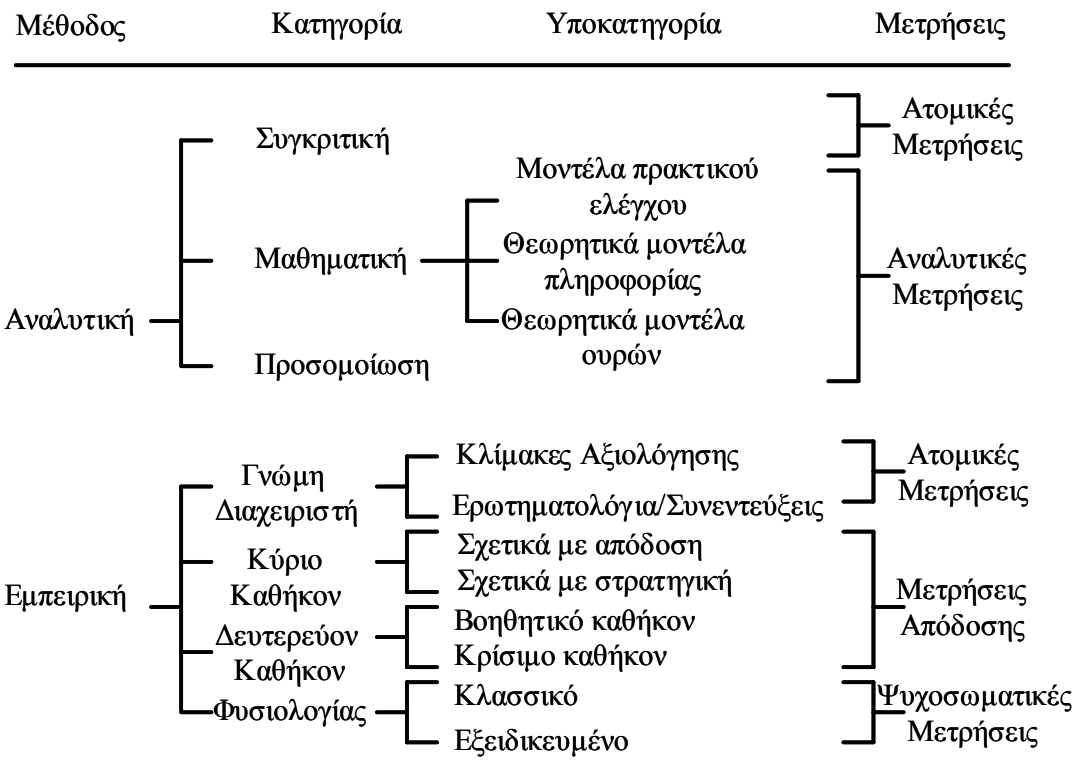
- Ο νοητικός φόρτος μπορεί να επηρεαστεί από πολλούς παράγοντες. Δεν είναι αποκλειστικά μία ιδιότητα του συγκεκριμένου καθήκοντος αλλά επιπλέον του ατόμου και της αλληλεπίδρασης ατόμου-καθήκοντος. Ο Meshkati (1988) κατηγοριοποίησε του παράγοντες που επηρεάζουν το νοητικό φόρτο σε «αιτιατούς» (οι οποίοι απαρτίζονται από τις παραμέτρους του συγκεκριμένου καθήκοντος και περιβάλλοντος, τα λειτουργικά χαρακτηριστικά και τις περιοριστικές μεταβλητές) και «επιδραστικούς» (οι οποίοι περιλαμβάνουν την δυσκολία, μεταβλητές απόκρισης και απόδοσης και μετρήσεις του νοητικού φόρτου) [MN1988]. Έτσι κάθε διάσταση του νοητικού φόρτου μπορεί να επηρεαστεί από κάποιους παράγοντες και του υποπαράγοντές τους. Κανονικά, μπορεί να χρησιμοποιηθεί μια ιεραρχική δομή για να περιγράψει αυτούς του παράγοντες και τους υποπαράγοντες. Για να απλοποιηθεί όμως το συγκεκριμένο μοντέλο λαμβάνονται υπόψιν μόνο οι κυρίαρχοι από αυτούς.

Συνοψίζοντας λοιπόν μπορούμε να πούμε ότι ο νοητικός φόρτος είναι μία έννοια η οποία δεν έχει οριστεί πλήρως. Αυτό που είναι δεδομένο είναι ότι ως έννοια είναι πολύπλευρη αλλά έχει μεγάλη σημασία στην προσπάθεια για επίτευξη υψηλού βαθμού απόδοσης. Καμία από τις ταξινομήσεις που έχουν γίνει δεν μπορεί να αποδώσει την πλήρη πληροφορία που περιλαμβάνει ο νοητικός φόρτος.

Πρόσφατα οι Tsang και Wilson (1997) συνόψισαν τις μετρήσεις του νοητικού φόρτου βασισμένοι σε τέσσερις τύπους [TPSWGF1997]:

- Ατομικές μετρήσεις
- Μετρήσεις απόδοσης
- Ψυχοσωματικές μετρήσεις (μέσω εμπειρικών μεθόδων)
- Αναλυτικές μετρήσεις (μέσω αναλυτικών μεθόδων)

Αυτή η ταξινόμηση έχει γίνει με μεγάλη συνέπεια και φαίνεται στο ακόλουθο σχήμα:



Σχήμα 1.1 Ταξινόμηση τεχνικών μέτρησης νοητικού φόρτου

3. Προσεγγίσεις μέτρησης φόρτου εργασίας

Όλες οι τεχνικές μέτρησης του φόρτου είτε είναι αναλυτικές είτε εμπειρικές μεταβάλλονται ως προς την ευαισθησία, τη διαγνωστικότητα και την διεισδυτικότητα [DDL1991].

- Η ευαισθησία αποτελεί την ικανότητα μιας τεχνικής να διαχωρίσει το φόρτο ο οποίος προκαλείται από μία εργασία από το φόρτο που προκαλεί κάποια άλλη εργασία.
- Η διαγνωστικότητα, αναφέρεται στην ικανότητα της τεχνικής να διαχωρίζει διαφορετικούς τύπους φόρτου εργασίας όπως οπτικό με νοητικό.
- Η διεισδυτικότητα αναφέρεται στο αποτέλεσμα, αν υπάρχει, το οποίο θα έχει η τεχνική μέτρησης του φόρτου στην κύρια εργασία.

3.1 Εμπειρικές προσεγγίσεις

Οι εμπειρικές προσεγγίσεις εξέλαβαν σημαντικής προσοχής και είναι οι πιο οικείες μέθοδοι [ORDEFT1986]. Οι τεχνικές αυτές συγκεντρώνουν δεδομένα, όπως προσωπικές απόψεις, ψυχολογικές μετρήσεις, ψυχοσωματικές μετρήσεις και μετρήσεις απόδοσης από διάφορα άτομα.

3.1.1 Ψυχοφυσιολογικές μετρήσεις

Οι μετρήσεις αυτές του φόρτου μπορούν να περιλαμβάνουν μετρήσεις δραστηριότητας εγκεφάλου με ηλεκτροεγκεφαλογραφήματα (EEG), ενδεχόμενα συσχετισμένα με γεγονότα, μετρήσεις του μαγνητικού πεδίου του εγκεφάλου (MEG), μετρήσεις του εγκεφαλικού μεταβολισμού όπως τομογραφίες εκπομπής ποζιτρονίων (PET), ήλεκτρο-οφθαλμογραφήματα (EOG), κίνησης οφθαλμών και ρυθμός καρδιακών παλμών [KAF1991].

Οι ερευνητές υποδεικνύουν ότι όλες οι παραπάνω ψυχοφυσιολογικές μετρήσεις είναι ευαίσθητες σε κάποιο είδος φόρτου. Οι καρδιακοί παλμοί για παράδειγμα γενικά αυξάνονται με την αύξηση του φόρτου. Έτσι αν ο σχεδιαστής μίας εργασίας αποδώσει στον εργαζόμενο που αναλαμβάνει τη συγκεκριμένη εργασία ένα σύστημα ανίχνευσης νέων στόχων, ο φόρτος του εργαζόμενου αναμένεται να αυξηθεί καθώς το πλήθος των στόχων που θα πρέπει να ανιχνεύει αυξάνεται. Επιπλέον αν ο ρυθμός των καρδιακών παλμών αποτελεί ψυχοφυσιολογική ένδειξη φόρτου, ο ρυθμός αυτός θα πρέπει να αυξάνει καθώς αυξάνεται ο φόρτος.

Δυστυχώς οι μετρήσεις αυτού του τύπου δεν είναι ιδιαίτερα χρήσιμες κατά τον πρώιμο σχεδιασμό ενός συστήματος, καθώς στο στάδιο αυτό δεν μπορεί να υπάρχει φυσική εξομοίωση του συστήματος και ο σχεδιαστής δεν μπορεί να λάβει τέτοιες μετρήσεις κατά την εξομοίωση. Από τη στιγμή όμως που θα δημιουργηθεί το πρότυπο του συστήματος οι μετρήσεις αυτές είναι ιδιαίτερα χρήσιμες για να διαγνωστούν τα πιθανά σημεία στα οποία ο εργαζόμενος στο σύστημα θα αντιμετωπίσει υπερβολικά υψηλά επίπεδα φόρτου.

Ο σχεδιαστής ο οποίος βασίζεται σε ψυχοφυσιολογικού δείκτες υψηλού φόρτου, είναι απαραίτητο να αναγνωρίζει ότι τα άτομα είναι σε θέση κάποιες φορές να διαχειριστούν τον υψηλό φόρτο τον οποίο αντιμετωπίζουν χωρίς αυτό να επηρεάσει

την απόδοσής τους. Με άλλα λόγια οι δείκτες υψηλού φόρτου μερικές φορές δεν συσχετίζονται με την απόδοση του ατόμου. Παρόλα αυτά οι μετρήσεις αυτές μπορούν να βοηθήσουν στην εύρεση δυναμικών περιοχών υψηλού φόρτου που θα μπορούσαν να επηρεάσουν την απόδοση. Στην περίπτωση αυτή μπορούν να χρησιμοποιηθούν άλλες τεχνικές μέτρησης του φόρτου για να διαπιστωθεί οποιαδήποτε μείωση στην απόδοση και το σύστημα μπορεί να αλλάξει αν αυτό κριθεί απαραίτητο πριν αυτό ξεκινήσει.

3.1.2 Μετρήσεις συμπεριφοράς

Μια άλλη εμπειρική τεχνική που μπορεί να χρησιμοποιηθεί από τον σχεδιαστή ενός συστήματος είτε σε συνδυασμό με τις ψυχοφυσιολογικές μετρήσεις είτε αντικαθιστώντας αυτές, είναι οι μετρήσεις συμπεριφοράς κατά την εκτέλεση καθηκόντων. Όταν χρησιμοποιούνται αυτές οι μετρήσεις σαν δείκτες του επιπέδου του φόρτου, συνήθως μελετώνται δύο τύποι μετρήσεων:

- Μετρήσεις κύριου καθήκοντος και
- Μετρήσεις δευτερεύοντος καθήκοντος

Χρησιμοποιώντας τις μετρήσεις του κύριου καθήκοντος αποτιμάται ο φόρτος μέσω εξέτασης κάποιου τμήματος της ικανότητας του εργαζόμενου να εκτελέσει την δεδομένη εργασία όπως η ταχύτητα του ή η ακρίβεια. Για παράδειγμα μπορεί να ζητηθεί από τον εργαζόμενο να εκτελέσει μια εργασία όπως ο έλεγχος ενός πίνακα οργάνων. Μπορεί στην περίπτωση αυτή να μετρηθεί η ακρίβεια και η ταχύτητα με την οποία ο εργαζόμενος πραγματοποιεί τον έλεγχο. Μπορεί να θεωρηθεί στην συνέχεια ότι όσο αυξάνεται ο φόρτος, η απόδοση σε γενικές γραμμές μειώνεται. Η μείωση αυτή της απόδοσης παρουσιάζεται ως αύξηση των λαθών που πραγματοποιεί ο εργαζόμενος ή ως αύξηση του χρόνου με τον οποίο πραγματοποιεί τις κατάλληλες ρυθμίσεις πάνω στον πίνακα ελέγχου. Βέβαια, μείωση της απόδοση δεν εμφανίζεται πάντοτε καθώς το άτομο εφευρίσκει μεθόδους και στρατηγικές που του επιτρέπουν να διατηρεί τα επίπεδα της απόδοσης του καθώς αυξάνεται ο φόρτος. Παρά τους περιορισμού αυτούς όμως, οι μετρήσεις κύριου καθήκοντος συχνά εφαρμόζονται κατά την εξομοίωση και δοκιμή ενός συστήματος.

Οι μεθοδολογίες μέτρησης δευτερεύοντος καθήκοντος επαυξάνουν τις απαιτήσεις του κύριου καθήκοντος. Η τεχνική αυτή εισάγει μία δεύτερη εργασία η οποία εκτελείται

παράλληλα με την κύρια εργασία. Για παράδειγμα μπορεί να αποτιμηθεί η ικανότητα του εργαζομένου να εκτελεί νοητικά μαθηματικούς υπολογισμούς (δευτερεύον καθήκον) καθώς ελέγχει έναν πίνακα οργάνων (κύριο καθήκον). Το ζήτημα στην περίπτωση αυτή είναι ότι το άτομο μπορεί να χρησιμοποιήσει εφεδρικές ικανότητες επεξεργασίας για την πραγματοποίηση του δευτερεύοντος καθήκοντος [EWKD1991]. Η μεθοδολογία αυτή είναι χρήσιμη καθώς μπορεί να χρησιμοποιηθεί για την συλλογή μιας βάσης δεδομένων από καθήκοντα τα οποία αλληλεπιδρούν. Είναι όμως δύσκολο να εφαρμοστεί σε πραγματικές συνθήκες. Επιπλέον το δευτερεύον καθήκον μπορεί να θεωρείται ως ενοχλητικό η βαρετό από τον εργαζόμενο και να πάψει να ενδιαφέρεται για αυτό με αποτέλεσμα να μην γίνεται σωστή αποτίμηση.

3.1.3 Υποκειμενικές μετρήσεις

Η εμπειρική τεχνική που έδωσε το καλύτερο αποτέλεσμα είναι η υποκειμενική μέτρηση του φόρτου [MON1988]. Η υποκειμενική μέτρηση συνήθως πραγματοποιείται μέσω αιτήματος προς τον εργαζόμενο να αποτιμήσει ο ίδιος το επίπεδο του φόρτου. Στην περίπτωση αυτή ο σχεδιαστής του συστήματος θεωρεί ότι όταν ο εργαζόμενος καταναλώνει περισσότερη πνευματική ενέργεια για την πραγματοποίηση της εργασίας, τότε ο εργαζόμενος εμφανίζει ένα αντίστοιχο αίσθημα προσπάθειας το οποίο μπορεί με ακρίβεια να το κρίνει ο ίδιος.

Οι περισσότεροι ερευνητές του νοητικού φόρτου συμφωνούν ότι τα άτομα, βρίσκουν σε γενικές γραμμές εύκολο να δώσουν εκτιμήσεις του φόρτου τους και ότι οι εκτιμήσεις αυτές έχουν αρκετή αξιοπιστία [CWG1996]. Παρόλα αυτά, ο σχεδιαστής ενός συστήματος, πρέπει να γνωρίζει ότι οι εργαζόμενοι ενδέχεται να αναφέρουν ότι αντιμετωπίζουν υψηλό φόρτο πριν την χρονική στιγμή κατά την οποία αυτός ο υψηλός φόρτος πραγματικά επηρεάζει την απόδοσή τους. Έτσι κατά την αποτίμηση του συστήματος πρέπει να λαμβάνονται υπόψιν εκτός από τις υποκειμενικές μετρήσεις φόρτου παράλληλα και μετρήσεις της συνολικής απόδοσης, για να επιβεβαιωθεί ότι ο φόρτος που αναφέρθηκε από τους εργαζόμενους επηρεάζει πραγματικά την απόδοσή τους. Επιπλέον, εκτός από αυτόν τον περιορισμό, δεν υπάρχει συμφωνία μεταξύ των ερευνητών σε ότι αφορά τις πληροφορίες οι οποίες πρέπει να λαμβάνονται από τους εργαζόμενους αλλά και στο πως οι πληροφορίες αυτές μπορούν να χρησιμοποιηθούν στην πρόβλεψη της απόδοσης [TV1996]. Η διαφωνία αυτή οφείλεται στην έλλειψη θεωριών οι οποίες να σχετίζουν επαρκώς το φόρτο με την απόδοση.

3.2 Αναλυτικές προσεγγίσεις

Τα αναλυτικά μοντέλα σχετίζονται με τεχνικές οι οποίες μπορούν να εφαρμοστούν νωρίς στο σχεδιασμό ενός συστήματος. Οι αναλυτικές μέθοδοι είναι σχεδιασμένες κατά τέτοιο τρόπο ώστε να παρέχουν τόσο πρόβλεψη όσο και αποτίμηση με αποτέλεσμα να αποτελούν ελκυστικότερη προσέγγιση. Δυστυχώς όμως οι μέθοδοι αυτές δεν χρησιμοποιούνται τόσο συχνά και δεν έχουν ελεγχθεί σε βαθμό ανάλογο με τις εμπειρικές.

Ο Linton et al (1989) διαχώρισε τον τομέα των αναλυτικών τεχνικών σε πέντε περιγραφικές κατηγορίες [LPD1989]:
1) Σύγκριση
2) Γνώμη ειδικού
3) Μαθηματικά μοντέλα
4) Ανάλυση εργασίας
5) Προσομοίωση με υπολογιστή

Οι πέντε αυτές κατηγορίες μπορούν επιπλέον να διασπαστούν σε δύο μεγάλες ομάδες:

- Αναλυτικές τεχνικές εργασίας
- Τεχνικές προβολής

Τα μαθηματικά μοντέλα, η ανάλυση εργασίας και τα μοντέλα προσομοίωσης ταξινομούνται στην πρώτη από τις δύο ομάδες καθώς χρειάζονται σημαντική προσπάθεια για να αποδώσουν λεπτομερή περιγραφή για το σύστημα. Η σύγκριση και η γνώμη του ειδικού μπορούν να κατηγοριοποιηθούν ως τεχνικές προβολής.

3.2.1 Τεχνικές προβολής

Κάποιες υποκειμενικές τεχνικές έχουν χρησιμοποιηθεί για να προβάλλουν τον νοητικό φόρτο. Αντί να ζητηθεί από τους εργαζόμενους να πραγματοποιήσουν και να αξιολογήσουν την πραγματική εργασία, οι τεχνικές αυτές χρησιμοποιούν τη γνώμη ενός ειδικού για την προβολή δεδομένων από παρόμοια συστήματα ή για να εκτιμήσουν τη δυσκολία της εργασίας με βάση παλαιότερη εμπειρία. Απαιτούν σε γενικές γραμμές αρκετά χαμηλότερο αριθμό εργαζομένων σε σχέση με τις αναλυτικές τεχνικές.

Ένα κύριο παράδειγμα τέτοιας τεχνικής για την ανάλυση του φόρτου είναι η εφαρμογή Pro-SWAT [KG1985]. Η τεχνική αυτή παρέχει μία μέτρηση του υποκειμενικού φόρτου. Ο φόρτος ορίζεται σαν υπέρθεση τριών ορθογωνίων διαστάσεων: χρονικός φόρτος, φόρτος νοητικής καταπόνησης και φόρτος ψυχοσωματικής πίεσης. Κάθε διάσταση αναπαρίσταται από μία κλίμακα αξιολόγησης τριών σημείων. Η τεχνική αυτή μπορεί να εφαρμοστεί από ειδικούς σε πρώιμο στάδιο σχεδιασμού του συστήματος για την αποτίμηση του φόρτου του εργαζομένου.

Εκδοχές τεχνικών προβολής άλλων υποκειμενικών τεχνικών παρουσιάζουν ενδιαφέρον αν και δεν εμφανίζουν την αποτελεσματικότητα της παραπάνω μεθόδου. Οι τεχνικές αυτές περιλαμβάνουν μία πιο ευαίσθητη και διαισθητική διαδικασία η οποία προϋποθέτει εξειδικευμένη εκπαίδευση και εμπειρία στην λειτουργία, τον σχεδιασμό και την ανάλυση ενός συστήματος. Τόσο η επιλογή του ειδικού όσο και το βάθος και ο βαθμός της εμπειρίας του είναι ιδιαίτερα κρίσιμα στοιχεία. Η εύρεση ενός τέτοιου ειδικού είναι πολύ δύσκολη. Γενικότερα η άντληση πληροφοριών για τον φόρτο από ειδικούς πρέπει να γίνεται με ιδιαίτερη προσοχή.

3.2.2 Αναλυτικές τεχνικές εργασίας

- Μοντέλα προσομοίωσης

Τα μοντέλα προσομοίωσης μπορούν βασικά να κατηγοριοποιηθούν ως τεχνικές που ενσωματώνουν τις αρχές της στατιστικής φύσης των εργασιών. Συνδυαζόμενα με την ανάλυση εργασίας, τα μοντέλα προσομοίωσης μπορούν να αποτελέσουν τις καλύτερες και διεξοδικότερες αναλυτικές τεχνικές. Υπάρχει όμως ένα μεγάλο κόστος στην εφαρμογή των μοντέλων αυτών και αυτό είναι ο επιπλέον χρόνος και η προσπάθεια που απαιτούνται για την ανάπτυξη ενός τέτοιου μοντέλου, καθώς αυτό χρειάζεται λεπτομερή περιγραφή των ιδιοτήτων της εργασίας, των πόρων, του χρόνου και των εργαζομένων. Από τη στιγμή όμως που οι παράμετροι αυτές είναι ιδιαίτερα δύσκολο να ληφθούν σε ένα πρώιμο στάδιο του σχεδιασμού ενός συστήματος, η εφαρμογή μεθόδων προσομοίωσης παρεμποδίζεται σημαντικά.

Τα μοντέλα προσομοίωσης ακολουθήθηκαν ενεργά την δεκαετία του 1980. Μοντέλα αναπαράστασης τέτοιου είδους αποτελούν το SIMWAM [KMA1984], τα Siegel-Wolf δικτυακά μοντέλα [MD1985], τα SAINT και Micro-SAINT [CLP1987] και άλλα. Τα περισσότερα από τα μοντέλα αυτά δεν αναπτύχθηκαν με στόχο ειδικά την ανάλυση

του φόρτου εργασίας. Τα χαρακτηριστικά που δίδουν ως έξοδο μπορούν να συγκεντρωθούν και να αναλυθούν για ένα ευρύ φάσμα σκοπών.

- Μαθηματικά μοντέλα

Τα μαθηματικά μοντέλα περιλαμβάνουν θεωρία ελέγχου, θεωρία πληροφορίας, και θεωρία ουρών. Παράδειγμα τέτοιου μοντέλου είναι των Baron και Levinson [BL1977]. Τα μαθηματικά μοντέλα που χρησιμοποιούν θεωρία ελέγχου, θωρούν ότι ο εργαζόμενος λειτουργεί στα όριά του και ότι το κλάσμα της προσοχής αποτελεί ικανοποιητικό ορισμό του φόρτου. Ο εργαζόμενος στα μοντέλα αυτά θεωρείται σαν ένας «σερβομηχανισμός» που προσπαθεί να εξαλείψει τα λάθη ανάδρασης. Είσοδος σε τέτοια μοντέλα μπορεί να είναι, για παράδειγμα, η απόκλιση της τροχιάς μιας πτήσης και έξοδος μπορεί να είναι η αντίδραση του πιλότου πάνω σε κάποια συσκευή αλλαγής πορείας. Για να εκτιμηθεί ο φόρτος του εργαζομένου, μια μεταβλητή του συστήματος όπως ο χρόνος αντίδρασης για την διόρθωση του σφάλματος, πρέπει να οριστεί ως δείκτης του φόρτου. Τα μοντέλα αυτά είναι κατάλληλα για εργασίες οι οποίες μπορούν να μοντελοποιηθούν με ένα γραμμικό δυναμικό σύστημα. Είναι όμως δεδομένο ότι μόνο ένα μικρό υποσύνολο των εργασιών για τις οποίες παρουσιάζει ενδιαφέρον η πρόβλεψη του φόρτου, μπορούν να μοντελοποιηθούν έτσι. Επιπλέον πρέπει να παρέχονται οι παράμετροι του συστήματος με λεπτομέρεια για να εκτελεστούν σωστά και πλήρως τα μοντέλα και οι παράμετροι αυτές τις περισσότερες φορές δεν είναι διαθέσιμες σε πρώιμα στάδια σχεδιασμού.

Τα μαθηματικά μοντέλα που βασίζονται σε θεωρία ουρών της αλληλεπίδρασης ανθρώπου-μηχανής, χαρακτηρίζουν τον εργαζόμενο ως ένα μονοκαναλικό επεξεργαστή που διαμοιράζει τους πόρους σειριακά σε ένα πλήθος εργασιών. Το άτομο αναπαριστάται ως ένας «εξυπηρετητής» που επεξεργάζεται πολλαπλές εργασίες και το ποσοστό χρησιμοποίησης του εξυπηρετητή χρησιμοποιείται ως μέσο μέτρησης του φόρτου. Τα μοντέλα αυτά σε γενικές γραμμές μπορούν να βρουν εφαρμογή σε περιπτώσεις όπου οι χρόνοι απόδοσης είναι κρίσιμοι. Η έμφαση στα μοντέλα αυτά δίνεται περισσότερο στο «πότε τελείται μία εργασία» και όχι στο «πως αυτή εκτελείται». Τα μοντέλα αυτά είναι κατάλληλα για περιπτώσεις πολλαπλών εργασιών στις οποίες ο εργαζόμενος πρέπει να αντιμετωπίσει με επιτυχία τις προτεραιότητες που τίθενται και τις απαιτήσεις σε απόδοση, παράμετροι που διαφέρουν μεταξύ των εργασιών.

Τα μαθηματικά μοντέλα που χρησιμοποιούν θεωρία πληροφορίας ήταν δημοφιλή στη δεκαετία του 1960. Και στην περίπτωση αυτή το κλάσμα της προσοχής αποτελεί την μετρική του φόρτου. Το πρόβλημα με τις μεθόδους αυτές βρίσκεται στο γεγονός ότι είναι ιδιαίτερα δομημένες για να μπορέσουν να εφαρμοστούν σε πραγματικές συνθήκες. Οι μετρήσεις που δίδουν βασίζονται σε καταστατικές πιθανότητες οι οποίες έχουν μεγάλη συνάφεια και είναι ανεξάρτητες από τα άτομα. Στις περισσότερες των εργασιών είναι πρακτικά αδύνατο να βρεθούν αυτές οι πιθανότητες. Ένα από τα πλεονεκτήματα, όμως, των μεθόδων αυτών είναι ότι αποδίδουν το κλάσμα της προσοχής με ένα αρκετά απλό τρόπο πράγμα το οποίο δεν ισχύει για τα υπόλοιπα μαθηματικά μοντέλα.

- Μέθοδοι ανάλυσης εργασίας

Οι μέθοδοι αυτές είναι οι πιο συχνά χρησιμοποιούμενες από όλα τα αναλυτικά εργαλεία για την εκτίμηση του φόρτου σε προκαταρκτικά στάδια σχεδιασμού. Είναι σχετικά εύκολο να κατανοηθούν και να χρησιμοποιηθούν και εμφανίζουν υψηλή ωφελιμότητα. Δεν απαιτείται υπερβολική μαθηματική ή προσομοιωτική εξειδίκευση για να εκτελεστεί η ανάλυση της εργασίας. Τα συγκεκριμένα μοντέλα γενικά εξετάζουν τις απαιτήσεις στην απόδοση του εργαζόμενου ως συνάρτηση του χρόνου σε συσχέτιση με ένα συγκεκριμένο σενάριο αποστολής.

Η βασική διαδικασία ανάλυσης της εργασίας ξεκινά με τον καθορισμό του στόχου του σεναρίου. Στη συνέχεια οι γενικές απαιτήσεις του στόχου (π.χ. αυτές που βρίσκονται σε υψηλό επίπεδο) αναλύονται με συστηματικό τρόπο σε καθήκοντα για τον εργαζόμενο και αυτά στη συνέχεια διασπώνται περαιτέρω σε ακόμα πιο συγκεκριμένα υποκαθήκοντα ή στοιχειώδη καθήκοντα. Αυτά τα στοιχειώδη καθήκοντα μπορούν στη συνέχεια να μεταφραστούν σε συγκεκριμένες ενέργειες του εργαζόμενους οι οποίες είναι πλήρως καθορισμένες. Έτσι το αποτέλεσμα της ανάλυσης είναι μία κατανομή της δραστηριότητας του εργαζόμενου ως συνάρτηση του χρόνου και της φάσης του στόχου.

Μία τεχνική αυτής της κατηγορίας που αναπτύχθηκε πρόσφατα είναι η TAWL από τους Hamilton και Bierbaum [HB1990] και χρησιμοποιείται για να προβλέψει το φόρτο του εργαζομένου χρησιμοποιώντας πληροφορίες από την ανάλυση εργασίας του συστήματος. Το μοντέλο αυτό αναπτύσσεται σε τρία στάδια: ανάλυση εργασίας και φόρτου, κατασκευή μοντέλου και εκτέλεση του.

Στο μοντέλο TAWL ο φόρτος ορίζεται ως η συνολική απαίτηση σε προσοχή που χρησιμοποιεί ο εργαζόμενος για να εκτελεστεί τα καθήκοντά του. Η ανάλυση του φόρτου βασίζεται στην θεωρία MRT (multiple resource theory), στην οποία θα αναφερθούμε εκτενώς στο επόμενο κεφάλαιο αυτής της εργασίας, και παρέχει ανεξάρτητες εκτιμήσεις για το νοητικό, το ψυχοκινητικό και το διαισθητικό τμήμα του φόρτου κάθε εργασίας. Το TAWL υποθέτει ότι κάθε ένα από αυτά τα τμήματα είναι ανεξάρτητα.. Έτσι ο φόρτος εργασίας είναι η απαίτηση που υπερτίθεται σε κάθε ένα από αυτά τα τμήματα φόρτου, από όλες τις εργασίες τις οποίες εκτελεί το άτομο. Αν και η μεθοδολογία αυτή είναι ιδιαίτερα χρήσιμη στην ανάλυση συστημάτων τα οποία απαιτούν παράλληλη ή τυχαία εκτέλεση εργασιών, το κέρδος θα είναι μικρό αν εφαρμοστεί σε περιπτώσεις συστημάτων όπου οι εργασίες χαρακτηρίζονται από απλή σειριακή διαδοχή.

Άλλες μέθοδοι που προτάθηκαν από αυτή την κατηγορία είναι το μοντέλο VACP από τους Aldrich, Szabo και Bierbaum [ASB1989], το TLAP από τους Parks και Boucek [PB1989] και το W/INDEX από τους North και Riley [NR1989]. Και τα τρία αυτά μοντέλα θεωρούν ότι η παράλληλη επεξεργασία πρέπει να λαμβάνεται υπόψιν και χρησιμοποιούν την αρχή των πολλαπλών τμημάτων του φόρτου για να προβλέψουν το βαθμό στον οποίο τα παράλληλα καθήκοντα παρεμβάλλονται μεταξύ τους.

- Άλλες αναλυτικές τεχνικές εργασίας

Κάποια υβριδικά μοντέλα αναπτύχθηκαν τα οποία χρησιμοποιούν τα πλεονεκτήματα των κατηγοριών που προαναφέρθηκαν συνδυάζοντας περισσότερα από ένα μοντέλα. Τα μοντέλα αυτά παρέχουν καλύτερα προγνωστικά αποτελέσματα. Οι Bi και Salvendy [BS1994], ανέπτυξαν ένα νοητικό μοντέλο σε συνδυασμό με ένα λειτουργικό μοντέλο για να προβλέψουν το φόρτο εργασίας στο σχεδιασμό δυναμικών συστημάτων ελέγχου. Το μοντέλο αυτό διαφέρει από τα υπόλοιπα μοντέλα ανάλυσης εργασίας στο γεγονός ότι χρησιμοποιεί μεταβλητές του συστήματος οι οποίες είναι εύκολο να ληφθούν σε πρώιμα στάδια σχεδιασμού όπως η πολυπλοκότητα της εργασίας, η αβεβαιότητα της, το πόσο σφικτό είναι το πρόγραμμα κ.λ.π., σε αντίθεση με όλα τα άλλα μοντέλα το συγκεκριμένο δεν απαιτεί λεπτομερή περιγραφή του σεναρίου και έτσι είναι σχετικά εύκολο στην εφαρμογή. Οι περιορισμοί σε αυτό το μοντέλο είναι ότι δεν σχεδιάστηκε για περιβάλλοντα πολλαπλών εργασιών.

Οι Patel et al. (1999) [PSGK1999], ανέπτυξαν και τεκμηρίωσαν την αναλογία του νόμου του Ohm για τον νοητικό φόρτο βασισμένοι σε ατομικούς παράγοντες και στην ικανότητα του ατόμου να επεξεργάζεται την πληροφορία. Το ρεύμα ορίστηκε ως ο ρυθμός μετάδοσης της πληροφορίας και η τάση αποτελεί τον νοητικό φόρτο για κάθε τμήμα της πληροφορίας στην οποία έχει γίνει επεξεργασία. Η αντίσταση αποτελεί συνάρτηση του είδους της εργασίας και κάποιων ξεχωριστών παραγόντων. Τέτοια μοντέλα όμως θεωρείται ότι εξαρτώνται σε πολύ μεγάλο βαθμό από το σύστημα στο οποίο εφαρμόζονται και είναι δύσκολη η εφαρμογή τους σε συστήματα τα οποία έχουν σχεδιαστεί με διαφορετικό τρόπο.

Κεφάλαιο 2

Θεωρία πολλαπλών πόρων – (Multiple Resource Theory MRT)

Οι προκλήσεις του σύγχρονου κόσμου απαιτούν από εμάς να πραγματοποιούμε όλο και περισσότερα πράγματα, σε σχέση με οποιαδήποτε άλλη εποχή. Δεν είναι απλά απαραίτητο να είμαστε συγκεντρωμένοι στις καθημερινές ανάγκες, αλλά θα πρέπει να είμαστε σε θέση να μπορούμε να κάνουμε πολλά πράγματα ταυτόχρονα και μάλιστα σωστά. Τι είναι αυτό που μας παρέχει την δυνατότητα να πραγματοποιούμε πολλά πράγματα ταυτόχρονα; Πως για παράδειγμα μπορούμε να οδηγούμε ένα αυτοκίνητο διατηρώντας το στη σωστή πορεία, ενώ ταυτόχρονα να συνομιλούμε με τον συνεπιβάτη μας; Όμως με τον ίδιο τρόπο υπάρχουν διεργασίες τις οποίες απλώς δεν μπορούμε να τις πραγματοποιήσουμε ταυτόχρονα χωρίς λάθη, επειδή η προσοχή που απαιτείται για την πραγματοποίηση της μίας διεργασίας έχει ισχυρή αρνητική επίδραση στην άλλη, όπως για παράδειγμα η πραγματοποίηση δύο ταυτόχρονων συζητήσεων.

Θα είναι χρήσιμο κατ' αρχήν να μελετήσουμε την εμπλοκή στην απόδοση σε μία διπλή διεργασία, πως δηλαδή το άτομο κατανέμει τα γνωστικά μέσα στις διαφορετικές διεργασίες και κάτω από ποιες συνθήκες μπορεί να υπάρξει διαμοιρασμός των μέσων αυτών. Για τον λόγο αυτό θα αναλύσουμε την θεωρία πολλαπλών πόρων (multiple resource theory – MRT).

1. Ανασκόπηση

Μία βασική ερώτηση για την κατανόηση του τρόπου με τον οποίο μπορεί να υπάρχει αποδοτικότητα σε δύο παράλληλες εργασίες βρίσκεται στο χώρο της κατανομής των «νοητικών πόρων» και στους περιορισμούς που αυτοί εμπεριέχουν. Αν και το συγκεκριμένο ζήτημα δεν είναι πλήρως ξεκάθαρο με συγκεκριμένους ορισμούς, χρησιμοποιείται η έννοια του «καναλιού», το οποίο περιγράφει την ροη της πληροφορίας δια μέσου των σχετικών σταδίων της επεξεργασίας και χαρακτηρίζεται από κάποια διακριτή αντιληπτική ιδιότητα ως πηγή του. Έτσι ο όρος «χωρητικότητα» χρησιμοποιείται για να περιγράψει ιδιότητες όπως τη χωρητικότητα του καναλιού σε

απόλυτη κρίση, την χωρητικότητα της μνήμης ή το εύρος της χωρητικότητας για την μετάδοση της πληροφορίας κατά μήκους του καναλιού σε bits ανά μονάδα χρόνου. Με τον τρόπο αυτό οι πόροι μπορούν να θεωρηθούν ως bits πληροφορίας που ταξιδεύουν κατά μήκος ενός συγκεκριμένου καναλιού περιοριζόμενοι κυρίως από την χωρητικότητα του καναλιού σε ταχύτητα και μέγεθος.

Παλαιότερα υπήρχε η πεποίθηση ότι υπήρχε μία μοναδική, μη διακριτή, πηγή πόρων και ότι οι πόροι ήταν διαθέσιμοι σε όλες τις αναγκαίες νοητικές διεργασίες. Η ιδέα αυτή ήταν βασισμένη στη θεώρηση ότι η προσοχή προσομοιάζει με την πεπερασμένη επεξεργαστική ικανότητα ενός ηλεκτρονικού υπολογιστή. Δυσκολότερες εργασίες αλλά και εργασίες που προϋποθέτουν μεγαλύτερη αποδοτικότητα απαιτούν περισσότερους πόρους σε σχέση με άλλες λιγότερο απαιτητικές εργασίες. Επιπλέον όσο πιο απαιτητική γίνεται μία διεργασία, απαιτώντας έτσι περισσότερους πόρους, μπορούν να εμφανιστούν φυσικές εκδηλώσεις όπως αύξηση καρδιακών παλμών ή διαστολή της κόρης του ματιού. Αυτά τα γεγονότα δείχνουν ότι οι πόροι επιστρατεύονται για την εκτέλεση μιας απαιτητικής εργασίας.

Τι συμβαίνει όμως όταν υπάρχουν δύο εργασίες οι οποίες πρέπει να πραγματοποιηθούν ταυτόχρονα; Θα μπορούσαμε να θεωρήσουμε ότι αύξηση στην απόδοση της μίας εργασίας συνεπάγεται την μείωση στην απόδοση της δεύτερης εργασίας και αντιστρόφως. Το 1975 οι Norman & Bobrow [NDBD1975] ανέπτυξαν την συνάρτηση αποδοτικότητας-πόρων (performance resource function – PRF) η οποία είναι μία υποθετική συνάρτηση που σχετίζει την ποιότητα της αποδοτικότητας με την προσπάθεια του εργαζομένου. Αν δύο εργασίες πραγματοποιούνται ταυτόχρονα διαχειριζόμενες τους ίδιους πόρους, η αύξηση στη ποιότητα της αποδοτικότητας στη μία από αυτές οδηγεί σε μείωση της αντίστοιχης ποσότητας στην άλλη λόγω καταμερισμού των πόρων.

Σε ότι αφορά την αποδοτικότητα σε μία εργασία, η PRF μπορεί να περιγράφει μία «καλή» περιοχή αποδοτικότητας (οποιαδήποτε προσπάθεια καταβάλλεται οδηγεί σε υψηλό επίπεδο αποδοτικότητας), μια «κακή» περιοχή (οποιαδήποτε προσπάθεια και να καταβληθεί το επίπεδο αποδοτικότητας είναι χαμηλό), μια «ιδανική» στρατηγική (αύξηση της προσπάθειας οδηγεί σε γραμμική αύξηση του επιπέδου αποδοτικότητας) και μία «ευριστική» στρατηγική (γρήγορη αρχική αύξηση της αποδοτικότητας με λίγη προσπάθεια αλλά ασυμπτωτική συμπεριφορά στην συνέχεια). Η ευριστική PRF περιγράφει σε μεγάλο βαθμό την προσπάθεια λήψης αποφάσεων. Σε ότι αφορά τον χρονικό καταμερισμό δύο εργασιών η PRF μπορεί να χρησιμοποιηθεί για να δείξει πως οι πόροι διανέμονται στις δύο εργασίες και πως αυτό επιδρά στην τελική αποδοτικότητα. Ο βαθμός παρεμβολής ανάμεσα στις δύο εργασίες θα εξαρτάται όμως

από τις ξεχωριστές PRFs. Αν δηλαδή μία εργασία είναι είτε τόσο τετριμμένη ώστε να μην απαιτεί καθόλου πόρους, είτε τόσο σύνθετη ώστε ακόμα και με όλους του πόρους να μην μπορεί να πραγματοποιηθεί επαρκώς.

Όσο απλοϊκό και να φαίνεται αυτό το μοντέλο μοναδικών πόρων, υπάρχουν σημαντικότατα εμπόδια με πρώτο ότι δεν μπορεί να εξηγήσει το αποτέλεσμα της εμπλοκής δύο εργασιών. Είναι αναμενόμενο δηλαδή στην περίπτωση που δύο εργασίες γίνονται στο ίδιο επίπεδο απόδοσης, η δυσκολότερη εργασία να απαιτεί περισσότερους πόρους. Παρόλα αυτά, στην πραγματικότητα υπάρχουν πολλά παραδείγματα που δείχνουν ότι αυτό δεν ισχύει πάντοτε. Πράγματι, όπως έδειξαν οι Hunt, Pellegrino και Yee [HPY1989], υπάρχει η ικανότητα συντονισμού της πληροφορίας από πολλαπλές πηγές η οποία είναι ανεξάρτητη από την ικανότητα επεξεργασίας της πληροφορία από μια ξεχωριστή από αυτές τις πηγές. Έτσι μπορεί να υπάρξει βελτίωση στην αποδοτικότητα όταν συμβαίνουν δύο παράλληλες εργασίες σε ότι αφορά την χωρική και λεκτική επεξεργασία. Μάλιστα οι Shah και Miyake (1996) [SPMA1996] παρέχουν με το πείραμά τους βάσιμες αποδείξεις για διαφορετικές γνωστικές πηγές για τις δύο μνήμες εργασίας (μία για τη χωρική σκέψη και η άλλη για την γλωσσική επεξεργασία), περιγράφοντας επιπλέον ότι διεργασίες που διαχωρίζονται σε χωρικές / λεκτικές μπορούν να γίνονται ταυτόχρονα με ελάχιστη εμπλοκή.

Και πάλι όμως μπορούν να υπάρχουν εργασίες χωρικές και λεκτικές οι οποίες δεν μπορούν να γίνουν ταυτόχρονα τόσο καλά. Επιπλέον δύο φαινομενικά παρόμοιες εργασίες μπορούν να αποφέρουν ανόμοια αποτελέσματα όταν πραγματοποιηθούν ταυτόχρονα με κάποια άλλη εργασία. Για παράδειγμα οι Payne et al. (1994) [PPBBAW1994] μελέτησαν την παρεμβολή μεταξύ του επιπέδου κατανόησης του λόγου σε μία εργασία «ακουστικής μνημονικής αναζήτησης» με ταυτόχρονες οπτικές διεργασίες. Οι οπτικές διεργασίες ήταν (α) παρακολούθηση ασταθούς κίνησης (β) χωρική λήψη απόφασης (γ) μαθηματικός συλλογισμός (δ) έλεγχος πιθανοτήτων. Αυτό που βρήκαν ήταν ότι μόνο η χωρική και η μαθηματική διεργασία επηρεαζόταν από χαμηλά επίπεδα κατανόησης του λόγου ενώ οι δύο άλλες διεργασίες όχι. Η παρεμβολή στην περίπτωση αυτή οφείλεται στο ότι η ακουστική διεργασία γίνεται ταυτόχρονα με δύο διεργασίες οι οποίες απαιτούν την επεξεργασία της πληροφορίας αντί απλώς την οπτική διερεύνηση της.

Ο Wickens [WDC1992], απαριθμεί αυτά τα εμπόδια στην «θεωρία πολλαπλών πόρων». Αυτό που προτείνει είναι ότι οι νοητικοί πόροι μπορούν να οριστούν μέσω τριών σχετικά απλών διχοτομημένων διατάσεων:

1) δύο από αυτές σχετίζονται με το επίπεδο στο οποίο βρίσκεται η επεξεργασία (σε πρώιμο ή μέσο στάδιο έναντι προχωρημένου σταδίου επεξεργασίας)
2) δύο άλλες σχετίζονται με τους συγκεκριμένους πόρους που παρέχουν οι κύριες αισθήσεις (ακουστική έναντι οπτικής κωδικοποίησης)
3) δύο τελευταίες σχετίζονται με τον κώδικα επεξεργασίας (λεκτικός έναντι χωρικού).

Αν οποιεσδήποτε δύο εργασίες απαιτούν «ξεχωριστούς πόρους» αντί για «κοινούς πόρους» στις τρεις αυτές διαστάσεις τότε μπορούν να συμβούν τρία πράγματα:

1) μπορεί να υπάρχει πιο αποδοτικός χρονικός καταμερισμός
2) αλλαγές στην δυσκολία μίας διεργασίας δεν θα έχουν ιδιαίτερη επίδραση στην αποδοτικότητα της άλλης
3) τα λειτουργικά χαρακτηριστικά της αποδοτικότητας (performance operating characteristics – POC) τα οποία κατασκευάζονται από δύο ξεχωριστές PRFs θα μοιάζουν με δύο εργασίες περιορισμένες από τα δεδομένα, αφού κάθε εργασία θα χρειάζεται ανεξάρτητους πόρους.

Το βασικό αποτέλεσμα αυτού του μοντέλου είναι ότι σε κάποιες περιπτώσεις μπορούμε να κάνουμε δύο πράγματα ταυτόχρονα με ελάχιστη μείωση της αποδοτικότητας σε κάθε ένα από αυτά. Παρόλα αυτά, αυτό ισχύει μόνο όταν οι δύο εργασίες εξελίσσονται είτε σε διαφορετικά στάδια, αισθήσεις ή κώδικες. Και πάλι όμως υπάρχουν περιπτώσεις όπου συμβαίνουν παρεμβολές ανάμεσα σε δύο εργασίες παρά τις προβλέψεις της MRT. Υπεύθυνοι για αυτό είναι οι διαφορετικοί τρόποι με τους οποίους οι άνθρωποι πραγματοποιούν την χωρική επεξεργασία και οι μηχανισμοί με τους οποίους ουσιαστικά κωδικοποιούμε και επεξεργαζόμαστε χωρικές πληροφορίες.

2. Multiple Resource Theory (MRT)

Η θεωρία αυτή αναφέρεται στην απόδοση σε πολλαπλές εργασίες, η οποία έχει τόσο πρακτικές όσο και θεωρητικές έννοιες. Αυτές οι πρακτικές έννοιες αντικρούονται από προβλέψεις που γίνονται με βάση τη θεωρία, σε ότι αφορά την ικανότητα του ατόμου να αποδώσει σε περιβάλλοντα υψηλού φόρου και πολλαπλών καθηκόντων. Μάλιστα συχνά εκφράζονται μέσω μίας συγκεκριμένης εκδοχής της «θεωρίας πολλαπλών πόρων», το οποίο θεωρούμε ως ένα μοντέλο πολλαπλών πόρων. Η αξία τέτοιων μοντέλων έγκειται στην ικανότητά τους να προβλέπουν σημαντικές λειτουργικές διαφορές στην απόδοση σε περιβάλλον πολλαπλών εργασιών, οι οποίες προκύπτουν από αλλαγές που είναι εύκολο να κωδικοποιηθούν από τον αναλυτή ή τον σχεδιαστή της εργασίας. Σε θεωρητικό επίπεδο, η αξία της «θεωρίας των πολλαπλών πόρων»

έγκειται στην ικανότητα της πρόβλεψης του επίπέδου εμπλοκής ή παρεμβολής δύο παράλληλων εργασιών.

Στις δύο παραπάνω περιπτώσεις ο διαχωρισμός μεταξύ «πολλαπλών» και «πόρων» είναι κρίσιμος. Η έννοια των πόρων συνδυάζει επιπρόσθετα κάτι το οποίο είναι περιορισμένο και κατανεμημένο. Η έννοια των πολλαπλών συνδυάζει παράλληλες, διαχωρισμένες ή σχετικά ανεξάρτητες εργασίες. Τυπικά οι πολλαπλοί πόροι καθορίζουν τον συνδυασμό των δύο αυτών εννοιών, όμως η κάθε έννοια χωριστά έχει ιδιαίτερα σημαντική συνεισφορά στην κατανόηση της απόδοσης σε πολλαπλές εργασίες.

Η θεωρία MRT και οι προβλέψεις της σε ότι αφορά την απόδοση, είναι συχνά στενά συσχετιζόμενη με δύο άλλες έννοιες: την «προσοχή» και τον «φόρτο εργασίας». Έτσι καθίσταται σημαντικό να δοθεί έμφαση στη διάκρισή τους και στο ότι αυτή η θεωρία δεν είναι αποκλειστικά ούτε θεωρία προσοχής ούτε θεωρίας φόρτου. Παρά την στενή σχέση ανάμεσα στην προσοχή και στην απόδοση σε παράλληλα καθήκοντα, οι δύο αυτές έννοιες απέχουν πολύ από το να χαρακτηριστούν συνώνυμες. Επιπλέον η έννοια του φόρτου είναι στενά συνδεδεμένη με τους πόρους και πολύ λιγότερο με την πολλαπλότητα.

2.1 Αρχές

Η αρχή της θεωρίας αυτής μπορεί να εντοπιστεί στην έννοια του περιορισμού που προκύπτει από το μοναδικό ανθρώπινο πληροφοριακό κανάλι, περιορισμός ο οποίος μειώνει την ικανότητα για επιτέλεση δύο εργασιών ταυτόχρονα με τρόπο αποδοτικό όπως αν εκτελούνται ξεχωριστά. Αυτή η άποψη είναι χαρακτηριστική στην ανάλυση εργασιών που απαιτούσαν υψηλή ταχύτητα, και υποστηρίζει ότι ο χρόνος είναι ιδιαίτερα περιορισμένος πόρος και δεν μπορεί να μοιραστεί ανάμεσα στις εργασίες. Ο Moray (1967) σε ένα άρθρο του περιέγραψε ότι ο άνθρωπος κατέχει ένα κεντρικό επεξεργαστή περιορισμένης χωρητικότητας οποίος μπορεί να διαμοιραστεί μέχρι κάποιο βαθμό μεταξύ των εργασιών [MON1967].

Η γενική προσέγγιση όλων αυτών των περιορισμένων σε χωρητικότητα αλλά με ικανότητα καταμερισμού μοντέλων, η οποία αναφέρεται και ως μοντέλο πόρων, είναι ότι οι απαιτήσεις των εργασιών δεν είναι αμετάβλητες. Τέτοια μοντέλα θεωρούν ότι οι νοητικοί πόροι από μία περιορισμένη πηγή, μπορούν να κατανεμηθούν κατάλληλα ώστε να ικανοποιούν τις απαιτήσεις της εργασίας, που ορίζονται από κοινού τόσο από το επίπεδο δυσκολίας όσο και από το επίπεδο της απόδοσης που απαιτείται. Πόροι οι οποίοι δεν χρησιμοποιούνται μπορούν να διατεθούν σε κάποια άλλη

εργασίας. Κατά συνέπεια, αν μία εργασία απαιτεί πολλούς πόρους θα συγκρούεται περισσότερο με κάποια άλλη που εκτελείται παράλληλα. Οι εργασίες που δεν απαιτούν κάποιους πόρους χαρακτηρίζονται και ως αυτόματες, ενώ οι εργασίες που απαιτούν όλους τους πόρους για να επιτευχθεί η μέγιστη απόδοση χαρακτηρίζονται και ως πλήρως περιορισμένες από τους πόρους. Οι εργασίες στις οποίες η μέγιστη απόδοση πετυχαίνεται χρησιμοποιώντας συγκεκριμένους μόνο πόρους χαρακτηρίζονται και ως περιορισμένες από τα δεδομένα [NDBD1975].

Κατά τις δεκαετίες του 60 και του 70 έγιναν πολλά πειράματα που τεκμηρίωσαν την αλληλεπίδραση μεταξύ των απαιτήσεων του κυρίου καθήκοντος και της απόδοσης στο δευτερεύον καθήκον. Πολλά από τα πειράματα αυτά εμβάθυναν για να μπορέσουν να ταυτοποιήσουν λειτουργίες που χαρακτηρίζουν την έννοια της απαίτησης πόρων. Περιλαμβάνουν μετρήσεις όπως υποκειμενική αξιολόγηση, φυσιολογικούς δείκτες και κυρίως κάποια αντικειμενικά χαρακτηριστικά των εργασιών τα οποία επηρεάζουν τις απαιτήσεις σε πόρους. Παραδείγματα τέτοιων χαρακτηριστικών είναι το εύρος ζώνης της πληροφορίας, ο φόρτος της μνήμης, ή το επίπεδο του εργαζόμενου. Η αξία αυτών των μετρήσεων έγκειται στο γεγονός ότι εμποδίζουν τις έννοιες των πόρων και των απαιτήσεων σε πόρους από το να είναι πλήρως κυκλικές στην πρόβλεψη της παρεμβολής πολλαπλών εργασιών. Δηλαδή αποφεύγουν να υποδηλώνουν ότι ή παρεμβολή μίας εργασίας είναι μεγαλύτερη λόγω μεγαλύτερων απαιτήσεων της σε πόρους και ότι η μεγαλύτερη απαίτηση σε πόρους της εργασίας συνεπάγεται μεγαλύτερη παρεμβολή.

Οι ερευνητές γενικά κρατούν μία σιωπηλή στάση στο τι ακριβώς είναι οι πόροι. Το μοντέλο του μοναδικού καναλιού θεωρεί το χρόνο ως περιορισμένο πόρο και μάλιστα ως το μοναδικό περιορισμένο πόρο που έχει σημασία. Υπάρχουν όμως περιπτώσεις στις οποίες η δυσκολία της εργασίας δεν συσχετίζεται άμεσα με τις χρονικές απαιτήσεις. Υπάρχουν δηλαδή παραδείγματα εργασιών που δεσμεύουν τον εργαζόμενο το 100% του χρόνου αλλά εμπλέκονται ελάχιστα με άλλες εργασίες Αν μόνο οι χρονικές απαιτήσεις ήταν υπεύθυνες για την παρεμβολή των εργασιών τότε εργασίες όπως η προηγούμενη θα παρουσίαζε παρεμβολή με άλλες σε επίπεδο ανάλογο με εργασίες που έχουν τεράστιες απαιτήσεις. Κάτι τέτοιο όμως γνωρίζουμε εμπειρικά ότι δεν ισχύει.

2.2 Πολλαπλοί πόροι

Σαν συνέχεια στην ανάπτυξη των μοντέλων πόρων για την παρεμβολή ανάμεσα στις εργασίες, τα στοιχεία συγκλίνουν στο ότι η μεταβλητότητα στην απόδοσή σε δύο παράλληλες εργασίες δεν μπορεί να αποδοθεί μόνο στην δυσκολία της μίας από αυτές

ούτε στην στρατηγική με την οποία γίνεται η κατανομή των πόρων μεταξύ τους. Αντιθέτως, οι διαφορές στις ποιοτικές απαιτήσεις για δομές πληροφοριακής επεξεργασίας οδηγούν σε διαφορές στην αποδοτικότητα του χρονικού καταμερισμού. Ο χρονικός καταμερισμός μεταξύ δύο εργασιών είναι περισσότερο αποδοτικός αν αυτές χρησιμοποιούν διαφορετικές δομές σε σχέση με τη χρησιμοποίηση κοινών δομών [WDC1976].

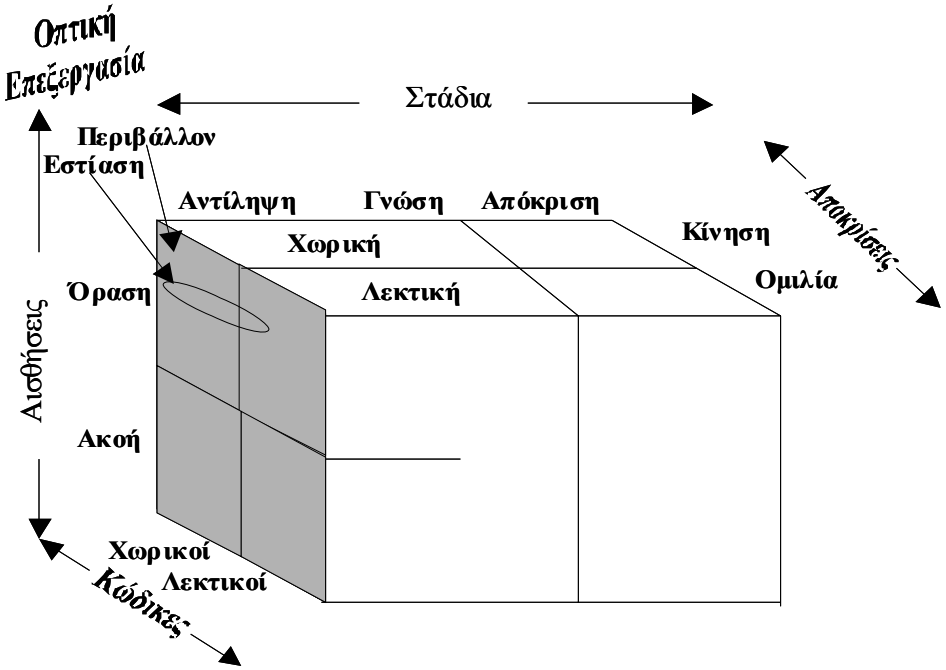
Ένα καταφανές παράδειγμα τέτοιου διαχωρισμού δομών είναι μεταξύ των ματιών (οπτική επεξεργασία) και των αυτιών (ακουστική επεξεργασία). Στις περισσότερες των περιπτώσεων η απόδοση σε δύο παράλληλα καθήκοντα είναι μικρότερη όταν τον χρόνο μοιράζονται δύο οπτικά καθήκοντα σε σχέση με την περίπτωση όπου το ένα από αυτά απαιτεί επεξεργασία ηχητικής πληροφορίας. Αυτό δείχνει ότι τα μάτια και τα αυτιά συμπεριφέρονται σαν πολλαπλοί πόροι επεξεργασίας.

2.3 Τετραδιάστατο μοντέλο πολλαπλών πόρων

Το μοντέλο αυτό[WDC2002] θεωρεί ότι υπάρχουν τέσσερις σημαντικές απόλυτες και διχοτομημένες διαστάσεις που εξηγούν τη διακύμανση στην αποδοτικότητα του χρονικού καταμερισμού. Κάθε διάσταση αποτελείται από δύο διακριτά επίπεδα. Δυο εργασίες οι οποίες απαιτούν ένα επίπεδο μίας συγκεκριμένης διάστασης, εμφανίζουν παρεμβολή σε μεγαλύτερο βαθμό από δύο εργασίες που απαιτούν διαφορετικά επίπεδα της συγκεκριμένης διάστασης. Οι διαστάσεις που απεικονίζονται στο ακόλουθο σχήμα είναι:

- Τα στάδια διεργασίας
- Οι αισθήσεις αντίληψης
- Τα οπτικά κανάλια
- Οι κώδικές διεργασίας

Η εικόνα παρουσιάζει τις τέσσερις διαστάσεις της θεωρίας γραφικά. Κάθε γραμμή στον κύβο χωρίζει τα δύο κατηγορικά επίπεδα της κάθε διάστασης. Η εικόνα δείχνει επιπλέον το πως ο διαχωρισμός μεταξύ λεκτικών και χωρικών κωδίκων διατηρείται σε όλα τα στάδια της επεξεργασίας, τον τρόπο με τον οποίο ο διαχωρισμός μεταξύ ακουστικών και οπτικών διεργασιών ορίζεται στην αντίληψη πληροφοριών και τον τρόπο με τον οποίο ο διαχωρισμός μεταξύ περιφερειακής και εστιακής όρασης εγκλείεται στους οπτικούς πόρους.

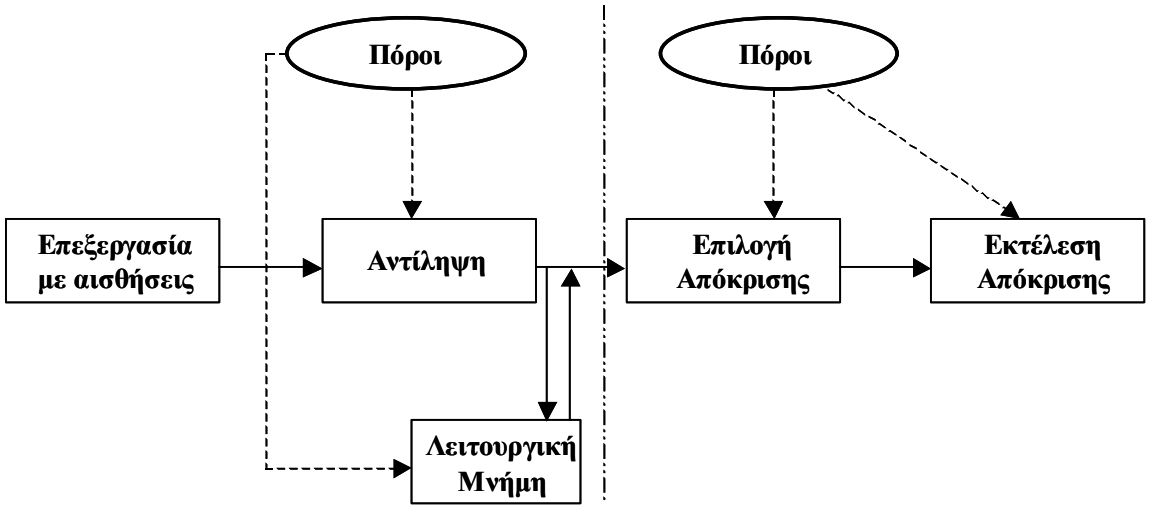


Σχήμα 2.1 Διαστάσεις τετραδιάστατου μοντέλου πολλαπλών πόρων

2.3.1 Στάδια

Οι πόροι που χρησιμοποιούνται σε δραστηριότητες αντίληψης και γνώσης φαίνονται να είναι οι ίδιες και είναι λειτουργικά διαχωρισμένες από εκείνους που είναι θεμελιώδεις για την επιλογή και εκτέλεση της απόκρισης. Απόδειξη για την ύπαρξη αυτής της ανεξαρτησίας παρέχεται όταν η δυσκολία στην απόκριση σε μια εργασία μεταβάλλεται και αυτή η αλλαγή δεν επηρεάζει την απόδοση σε μία παράλληλη εργασία της οποίας οι απαιτήσεις είναι μεγαλύτερες σε πόρους αντίληψης και γνώσης. Και αντιστρόφως όταν αύξηση σε αυτή τη δυσκολία δεν επηρεάζει σημαντικά μία παράλληλη εργασία της οποίας οι απαιτήσεις σχετίζονται κατά κύριο λόγο με την απόκριση.

Η διχοτόμος των σταδίων του μοντέλου προβλέπει επίσης ότι θα υπάρχει ουσιαστική παρεμβολή μεταξύ εργασιών αντίληψης που είναι απαιτητικές σε πόρους και γνωσιακών εργασιών που χρησιμοποιούν τη λειτουργική μνήμη για αποθήκευση η μετασχηματισμό πληροφοριών (Σχήμα 2.2). Παρά το γεγονός ότι οι εργασίες αυτές ορίζουν διαφορετικά στάδια επεξεργασίας της πληροφορίας, υποστηρίζονται από κοινούς πόρους.



Σχήμα 2.2 Δύο πηγές πόρων τροφοδοτούν διαφορετικά στάδια επεξεργασίας πληροφορίας

2.3.2 Αισθήσεις αντίληψης

Είναι φανερό ότι κάποιες φορές μπορεί να γίνει καλύτερος διαχωρισμός της προσοχής μεταξύ του ματιού και του αυτιού από ότι μεταξύ δύο οπτικών ή δύο ακουστικών καναλιών. Αυτό σημαίνει ότι ο χρονικός καταμερισμός διαφορετικών αισθητικών καναλιών (ακουστικού και οπτικού AV) είναι καλύτερος από τον χρονικό καταμερισμό όμοιων αισθητικών καναλιών (ακουστικού-ακουστικού AA ή οπτικού-οπτικού VV).

Το σχετικό πλεονέκτημα του AV χρονικού καταμερισμού ενδέχεται στην πραγματικότητα να μην είναι αποτέλεσμα των διαφορετικών πόρων αντίληψης, αλλά κάποιων περιφερειακών παραγόντων που δημιουργούν το μειονέκτημα στις περιπτώσεις AA και VV. Έτσι εάν δύο ανταγωνιστικά οπτικά κανάλια (VV) απέχουν μεταξύ τους απαιτούν οπτική σάρωση που είναι ένα πρόσθετο κόστος. Αν αντιθέτως είναι πολύ κοντά, προξενούν σύγχυση και επικάλυψη. Σε αντιστοιχία, δύο ακουστικά μηνύματα μπορούν να αλληλεπικαλύπτονται αν καταλαμβάνουν κοντινές ή χρονικά επικαλυπτόμενες συχνότητες. Παρόλα αυτά σε πραγματικές συνθήκες η οπτική σάρωση αποτελεί επαρκή αιτία για τη μείωση της παρεμβολής δύο εργασιών μέσω αποσυμφόρησης ποσότητας πληροφορίας από την όραση στην ακοή [SWL2001].

2.3.3 Οπτικά κανάλια

Εκτός από τον διαχωρισμό μεταξύ οπτικών και ακουστικών αισθήσεων διεργασίας, δύο πλευρές της οπτικής επεξεργασίας, που αναφέρονται ως εστιακή και

περιφερειακή όραση, εμφανίζονται να ορίζουν ξεχωριστούς πόρους με την έννοια ότι [WLWC1992]:

- Υποστηρίζουν αποτελεσματικό χρονικό καταμερισμό
- Χαρακτηρίζονται από ποιοτικά διαφορετικές εγκεφαλικές δομές
- Σχετίζονται με ποιοτικά διαφορετικούς τύπους επεξεργασίας της πληροφορίας.

Η «εστιακή όραση» απαιτείται για υψηλή ανάλυση εικόνας και αναγνώριση προτύπων ενώ η «περιφερειακή όραση» χρησιμοποιείται για προσανατολισμό και προσωπική κίνηση. Όταν για παράδειγμα διασχίσουμε ένα διάδρομο διαβάζοντας ένα βιβλίο με επιτυχία, εκμεταλλευόμαστε την παράλληλη επεξεργασία ή τις δυνατότητες πολλαπλών πόρων της εστιακής και περιφερειακής όρασης. Το ίδιο συμβαίνει και όταν οδηγούμε διατηρώντας το αυτοκίνητο στη σωστή λωρίδα (περιφερειακή όραση) ενώ ταυτόχρονα παρατηρούμε ένα οδικό σήμα ή κοιτάμε τον εσωτερικό καθρέφτη ή αναγνωρίζουμε κάποιο επικίνδυνο εμπόδιο (εστιακή όραση).

2.3.4 Κώδικες διεργασίας

Η διάσταση αυτή ορίζει τον διαχωρισμό μεταξύ αναλογικών / χωρικών διαδικασιών και κατηγορικών / συμβολικών διαδικασιών. Δεδομένα από μελέτες πολλαπλών εργασιών δείχνουν ότι οι χωρικοί και λεκτικοί κώδικές, ανάλογα με το αν λειτουργούν στις αισθήσεις, στην λειτουργική μνήμη ή την απόκριση, εξαρτώνται από χωριστούς πόρους και ο διαχωρισμός αυτός σχετίζεται με τα δύο εγκεφαλικά ημισφαίρια.

Ο διαχωρισμός των λεκτικών και χωρικών πόρων εξηγεί και το γεγονός του σχετικά υψηλού βαθμού αποδοτικότητας με τον οποίο καταμερίζονται χρονικά οι φωνητικές και κινητικές αποκρίσεις θεωρώντας ότι σε γενικές γραμμές οι κινητικές αποκρίσεις είναι χωρικές κατά κύριο λόγο ενώ οι φωνητικές είναι λεκτικές. Μία σημαντική πρακτική εφαρμογή του διαχωρισμού των κωδικών επεξεργασίας είναι η ικανότητα να προβλέψουν πότε θα είναι επικερδές να εφαρμοστεί φωνητικός έναντι κινητικού ελέγχου.

Ο κινητικός έλεγχος έχει αποδειχθεί ότι μπορεί να διαταράξει την απόδοση σε ένα περιβάλλον εργασίας με απαιτήσεις σε χωρική λειτουργική μνήμη. Ο φωνητικός έλεγχος μπορεί να διαταράξει την απόδοση εργασιών με υψηλές λεκτικές απαιτήσεις [WL1988].

2.4 Εφαρμογές του μοντέλου

Οι πιο σημαντικές εφαρμογές του «μοντέλου πολλαπλών πόρων» είναι η πρόβλεψη του επιπέδου της απόδοσης δύο ή περισσότερων εργασιών που διαμοιράζονται τον χρόνο. Με άλλα λόγια χρησιμοποιείται για να κάνει προβλέψεις για το επίπεδο διαταραχής η παρεμβολής δύο εργασιών όταν πρέπει να γίνεται χρονικός καταμερισμός. Το μοντέλο είναι περισσότερο εφαρμόσιμο σε περιβάλλοντα πολλαπλών εργασιών με υψηλές απαιτήσεις. Το μοντέλο μπορεί να εφαρμοστεί είτε άτυπα με διαισθητικό τρόπο είτε τυπικά με υπολογιστικό τρόπο.

Άτυπα, το μοντέλο αυτό μπορεί να εξυπηρετήσει στην καθοδήγηση των σχεδιαστών στην λήψη αποφάσεων όπως το πότε είναι καλύτερο να χρησιμοποιηθεί φωνητικός αντί κινητικού ελέγχου ή να χρησιμοποιηθούν ακουστικές αντί οπτικών οδηγιών ή να χρησιμοποιηθεί χωρική αντί λεκτικής απεικόνισης. Για την εφαρμογή της θεωρίας πολλαπλών πόρων στη λήψη τέτοιων κατηγορικών σχεδιαστικών αποφάσεων, είναι πολύ σημαντικό να λαμβάνονται υπόψιν οι υπόλοιπες περιπτώσεις της εναλλαγής από την μία κατηγορία πόρων στην άλλη.

Πιο τυπικά το μοντέλο μπορεί να αποδοθεί με τρόπο τέτοιο ώστε να υπολογίζει το ποσό της παρεμβολής που προβλέπεται να έχουν δύο εργασίες ως συνάρτηση του ανταγωνισμού που παρουσιάζουν αυτές για τους κοινούς πόρους. Για την υπολογιστική περιγραφή του μοντέλου, είναι απαραίτητο να ληφθεί υπόψιν η επιπρόσθετη τιμή της έννοιας των πολλαπλών πόρων σε σχέση με τα απλούστερα μοντέλα ανάλυσης παρεμβολής εργασιών, με βάση την χρονική ανάλυση των εργασιών. Η απλή χρονική ανάλυση της εμπλοκής των εργασιών ταυτοποιεί τις περιόδους στις οποίες δύο ή περισσότερες εργασίες πρέπει να εκτελεστούν παράλληλα και ορίζει τις περιόδους αυτές ως περίοδοι υπερφόρτωσης. Στην περίπτωση αυτή, ο νοητικός φόρτος μπορεί να οριστεί από τον λόγο:

$$\frac{\textbf{\textit{Απαιτούμενος χρόνος για την πραγματοποίηση των εργασιών}}}{\textbf{\textit{Διαθέσιμος χρόνος}}}$$

Όταν ο λόγος αυτός ξεπερνάει την τιμή 1, σε ένα συγκεκριμένο χρονικό διάστημα, τότε υπάρχει υπερφόρτωση και η απόδοση στην μία εργασίας θα πρέπει να υποστεί καθυστέρηση μέχρι να ολοκληρωθεί η άλλη εργασία.

Ένα τέτοιο μοντέλο είναι κατάλληλο για να προβλέψει ή να εκτιμήσει το επίπεδο του εμφανιζόμενου φόρτου όταν ο παραπάνω λόγος βρίσκεται αρκετά κάτω από τη

μονάδα. Η διαφορά μεταξύ τιμών 0,3 και 0,8 είναι ιδιαίτερα σημαντική για να αποφασιστεί για παράδειγμα ότι ο εργαζόμενος με λόγο 0,3 είναι ικανότερος να αναλάβει μία καινούργια εργασία σε σχέση με τον εργαζόμενο που έχει λόγο 0,8. Επιπλέον όταν όλες οι εργασίες απαιτούν ένα μοναδικό «μη-διανεμητέο» πόρο, ο παραπάνω λόγος έχει πολύ μεγάλη σημασία. Ένας τέτοιος πόρος είναι η φωνή στην ομιλία. Σε πολλές περιπτώσεις και η εστιακή όραση μπορεί να οριστεί ως μη-διανεμητέα και ο φόρτος σχετίζεται στην περίπτωση αυτή με το χρόνο κατά τον οποίο το μάτι σαρώνει την περιοχή ενδιαφέροντος.

Ο βασικός περιορισμός αυτής της χρονικής ανάλυσης είναι ότι θεωρεί όλες τις εργασίες όμοιες στον βαθμό που αφορά την πρόβλεψη της υπερφόρτωσης. Όμως και σύμφωνα με τη θεωρία πολλαπλών πόρων αυτό δεν ισχύει. Κατ' αρχάς οι εργασίες ποικίλουν σε απαιτήσεις πόρων με τρόπους που δεν μπορούν να ερμηνευτούν με το χρόνο. Επιπλέον δύο οποιεσδήποτε εργασίες που διαμοιράζονται το χρόνο μπορούν να ποικίλουν ως προς τις ποιοτικές απαιτήσεις τους σε πόρους, γεγονός το οποίο έχει μεγάλη επίδραση στην αμοιβαία εμπλοκή τους. Για παράδειγμα, η οδήγηση παράλληλα με ανάγνωση κάποιου μηνύματος παράγει μεγάλη εμπλοκή, ενώ η οδήγηση ακούγοντας το ίδιο μήνυμα προκαλεί πολύ μικρότερο φόρτο, παρά το γεγονός ότι και οι δύο περιπτώσεις καταλαμβάνουν το ίδιο ποσό χρόνου στην χρονική ανάλυση.

2.4.1 Υπολογιστικό μοντέλο

Ένα τυπικό υπολογιστικό μοντέλο βασισμένο στη θεωρία πολλαπλών πόρων έχει γενικά τα ακόλουθα χαρακτηριστικά:

- Κάθε εργασία αναπαρίσταται με ένα διάνυσμα των απαιτήσεων σε πόρους, τόσο σε ποιοτικό επίπεδο (είδος πόρων) όσο και σε ποσοτικό επίπεδο (πλήθος πόρων).
- Το ποσό του φόρτου σε κάθε έναν από αυτούς τους πόρους είναι εξαρτώμενο από την εργασία.
- Το μοντέλο υπολογίζει την απώλεια στην απόδοση στη μία ή και στις δύο εργασίες σε σχέση με το επίπεδο απόδοσης όταν αυτές εκτελούνται κατ' αποκλειστικότητα. Χρησιμοποιεί μία μέθοδο που κυρώνει την απόδοση με βάση ότι (α) οι συνολικές απαιτήσεις και των δύο εργασιών είναι υψηλές, (β) και οι δύο εργασίες ανταγωνίζονται για επικαλυπτόμενους πόρους.
- Ο βαθμός στον οποίο η μία ή η άλλη από τις δύο εργασίες χάνει σε απόδοση μπορεί να οριστεί με βάση κάποια στρατηγική. Αν και οι δύο εργασίες έχουν την ίδια προτεραιότητα τότε η μείωση της απόδοσης θα είναι και στις δύο ίση.

Για να ολοκληρωθούν αυτές οι λειτουργίες υπολογιστικά, ένα τυπικό μοντέλο ενσωματώνει τις ακόλουθες συνιστώσες:

- Ένα σκελετό ανάλυσης εργασίας στον οποίο εισάγονται, σε διαφορετικούς πόρους για κάθε εργασία, τα επίπεδα απαιτήσεων της εργασίας. Στην περίπτωση αυτή, η πλευρά των «πολλαπλών» από τους πολλαπλούς πόρους είναι πιο σημαντική από την πλευρά των «πόρων».
- Ένα πίνακα παρεμβολής στον οποίο καθορίζεται η ποσότητα της σύγκρουσης μεταξύ ζευγών πόρων για τις εργασίες. Ο πίνακας αυτός αποτελεί το κέντρο της πλευράς των «πολλαπλών». Εύκολα μπορεί να θεωρηθεί ότι αν δύο εργασίες δεν μπορούν να μοιράζονται κάποιον πόρο, τότε η τιμή της παρεμβολής είναι 1. Στην αντίθετη περίπτωση, όταν δηλαδή δύο εργασίες μοιράζονται τέλεια κάποιον πόρο, η τιμή αυτή είναι 0
- Μία υπολογιστική μέθοδο η οποία τυπικά αποτελείται από δύο συνιστώσες που αντιστοιχούν στις δύο συνιστώσες των πολλαπλών πόρων. Η πρώτη κυρώνει το ζεύγος των εργασιών για τη συνολική απαίτηση σε πόρους. Η δεύτερη κυρώνει το ζεύγος των εργασιών ανάλογα με το βαθμό σύγκρουσης τους στα ζεύγη πόρων.
- Μία τιμή παρεμβολής εργασίας η οποία θα παρέχεται ως έξοδος. Ως άθροισμα των δύο συνιστωσών (απαίτηση σε πόρους και βαθμός εμπλοκής), η τιμή αυτή μπορεί να διανεμηθεί κατάλληλα στη μία η την άλλη εργασία ανάλογα με το σχήμα προτεραιότητας.
- Μία χρονική ανάλυση η οποία μπορεί να εφαρμοστεί σε καταστάσεις στις οποίες συγκεκριμένος συνδυασμός εργασιών είναι χρονικά εξαρτώμενος.

Κεφάλαιο 3

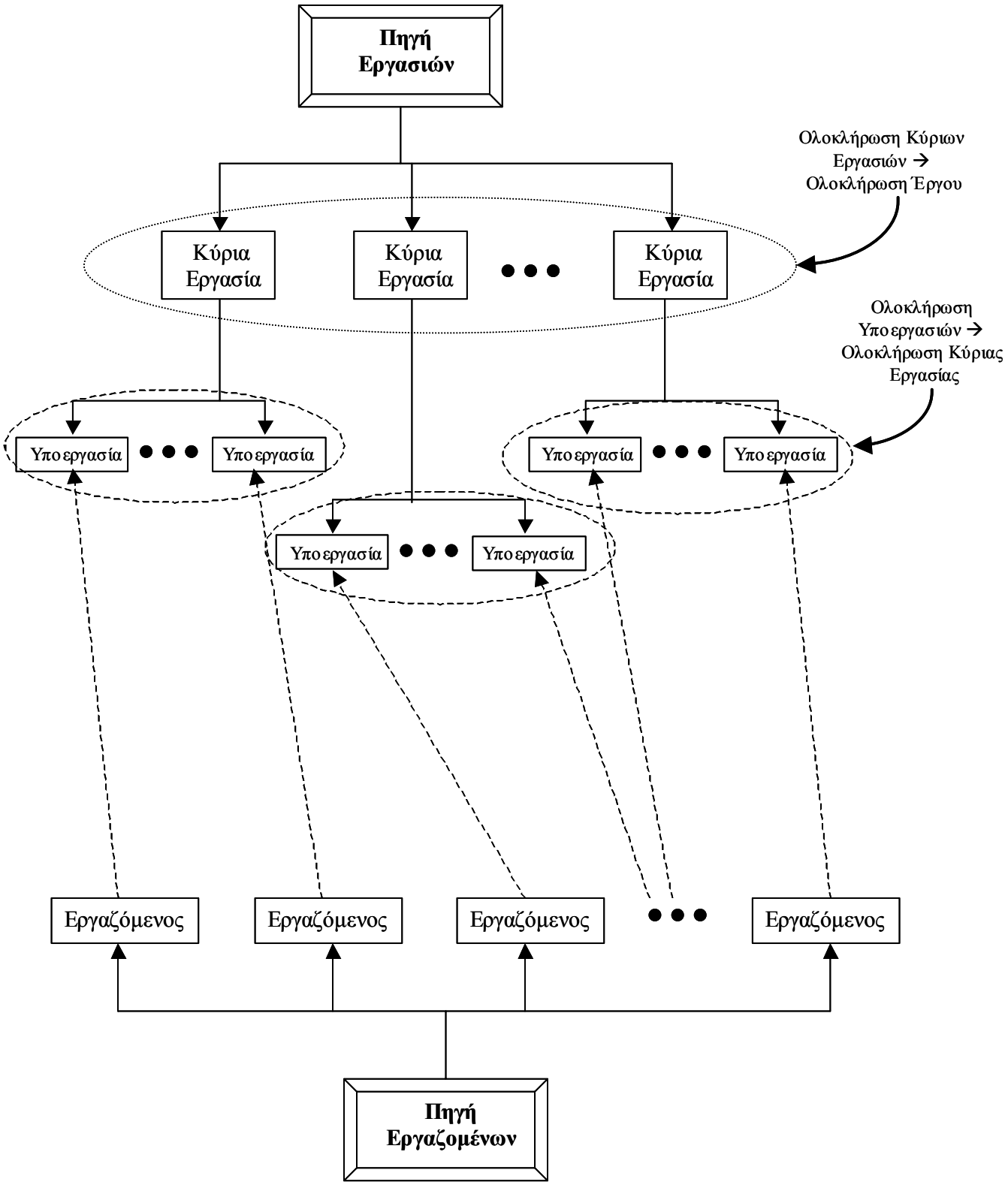
Υλοποίηση της μεθόδου μέσω διακριτής προσομοίωσης ενός βιομηχανικού καθήκοντος

1. Παράδειγμα εργασίας

Στην διπλωματική αυτή θα υλοποιήσουμε την «θεωρία πολλαπλών πόρων» εφαρμόζοντας την σε ένα υποθετικό βιομηχανικό σύστημα όπου υπάρχουν κάποια καθήκοντα τα οποία πρέπει να επιτελεστούν. Η βάση πάνω στην οποία κινούμαστε είναι η ολοκλήρωση των καθηκόντων αυτών με τον καλύτερο δυνατό τρόπο, την καλύτερη δυνατή ακρίβεια και στο συντομότερο δυνατό χρονικό διάστημα.

Θεωρούμε ότι στο σύστημα αυτό υπάρχει μία πηγή η οποία παράγει εργασίες και αποδίδει σε αυτές κάποιες αρχικές ιδιότητες. Οι εργασίες που παράγονται είναι οι «εργασίες στόχοι» που θα πρέπει να εκτελεστούν για να ολοκληρωθεί το συνολικό έργο. Κάθε μία «εργασία στόχος» διαχωρίζεται σε δύο ή περισσότερα υποκαθήκοντα η πλήρης ολοκλήρωση των οποίων σημαίνει και την ολοκλήρωση της εργασίας στόχου. Τα καθήκοντα αυτά αναθέτονται στους εργαζομένους στο σύστημα. Μία αντίστοιχη πηγή παράγει πεπερασμένο αριθμό εργαζομένων με συγκεκριμένες ιδιότητες που καθορίζονται εξ΄ αρχής, για να μεταβληθούν όποτε αλλάξει οτιδήποτε στην κατάστασή τους.

Η μεθοδολογία της ανάθεσης, η οποία αναλύεται σε επόμενο κεφάλαιο, ακολουθεί κάποιους κανόνες ώστε να φέρνει τα καλύτερα δυνατά αποτελέσματα. Στο ακόλουθο σχήμα φαίνεται ο βασικός τρόπος με τον οποίο λειτουργεί το παραπάνω σύστημα στην δημιουργία εργασιών και εργαζομένων και τα βήματα που ακολουθούνται για την υλοποίηση του στόχου που είναι η ολοκλήρωση των εργασιών.



Σχήμα 3.1: Δομή του παραδείγματος εργασίας το οποίο στη συνέχεια θα προσομοιωθεί στη γλώσσα προγραμματισμού C++.

2. Ποιοτική περιγραφή κώδικα C++

2.1 Γενική αναφορά στη γλώσσα C++

Η C++ είναι μία γλώσσα προγραμματισμού γενικής χρήσης, σχεδιασμένη για να κάνει πιο άνετο τον προγραμματισμό. Διαθέτει ευέλικτες και αποδοτικές υπηρεσίες που βοηθούν στον ορισμό νέων τύπων. Υποστηρίζει μία μεγάλη ποικιλία από στιλ προγραμματισμού. Τα βασικά χαρακτηριστικά που παρέχει είναι ο αντικειμενοστραφής προγραμματισμός και οι αφαιρετικοί τύποι δεδομένων. Η έννοια κλειδί όμως της C++ είναι η κλάση (class). Η κλάση ορίζεται από τον χρήστη και παρέχει απόκρυψη δεδομένων, εγγυημένη αρχικοποίηση των τιμών τους και δυναμική χρήση μνήμης.

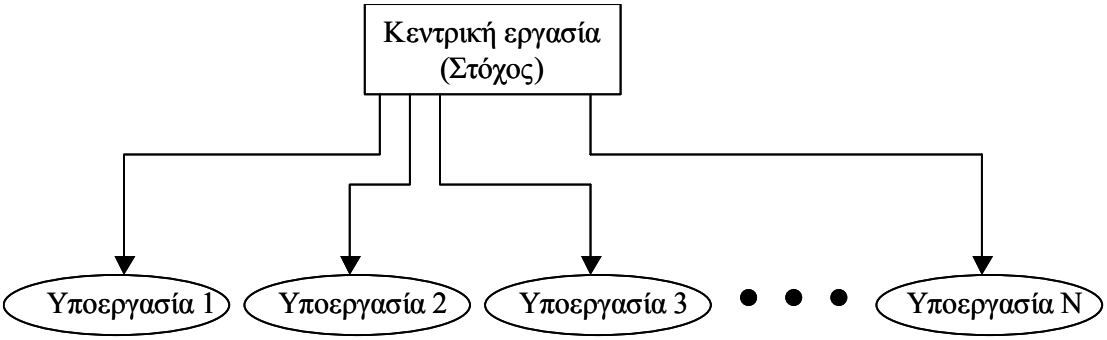
2.2 Κλάσεις

Στην υλοποίηση μας και με βάση τα δεδομένα του προβλήματος που αντιμετωπίζουμε, δύο είναι οι κλάσεις που κατασκευάσθηκαν. Η πρώτη, αναφερόμενη στο παράρτημα Α και ως *MainTask*, είναι η κλάση που αφορά τα καθήκοντα και ο τρόπος με τον οποίο ορίζεται και οι λειτουργίες που παρέχει αναλύονται στην επόμενη ενότητα. Η δεύτερη, αναφερόμενη στο παράρτημα Α ως *Workers*, είναι η κλάση που αφορά τον εργαζόμενο, αυτόν δηλαδή που θα εκτελέσει το συγκεκριμένο καθήκον και ο ορισμός και λειτουργία της αναλύεται στη μεθεπόμενη ενότητα.

2.2.1 Κλάση εργασίας *MainTask*

Η γενική θεώρηση που γίνεται για την κλάση αυτή ξεκινάει από το γεγονός ότι βρισκόμαστε σε ένα σύστημα στο οποίο πρέπει να επιτελεστεί μια εργασία. Η εργασία αυτή αποτελεί τον κεντρικό στόχο και για την ολοκλήρωση της απαιτείται η επιτέλεση κάποιων επιμέρους εργασιών που τη συναποτελούν. Με βάση τη θεώρηση αυτή ένα γενικό διάγραμμα που απεικονίζει την κλάση φαίνεται στο σχήμα 3.2:

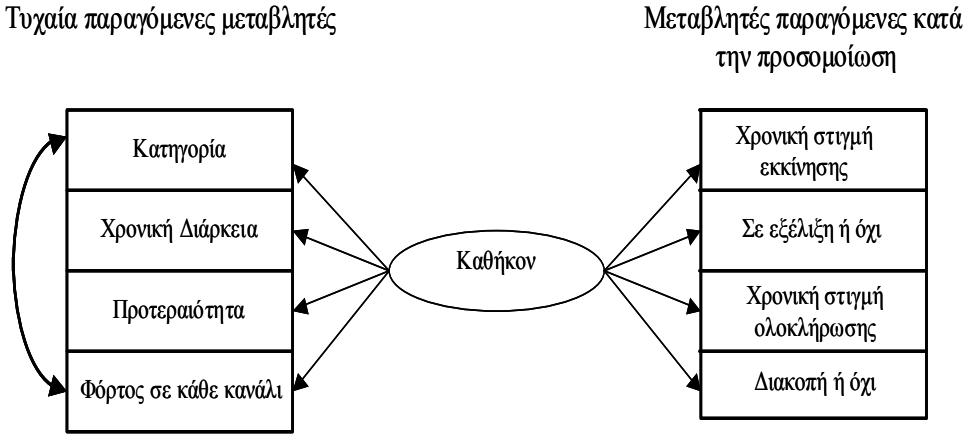
Το πλήθος των υποεργασιών, για να διατηρηθεί η γενικότητα του μοντέλου, παράγεται με τυχαίο τρόπο και απλά έχει θεωρηθεί, για πρακτικούς λόγους και λόγους ελέγχου σωστής λειτουργίας, ότι μπορεί να είναι μέχρι δέκα.



Σχήμα 3.2: Γενικό διάγραμμα της κλάσης *MainTask*

Σε κάθε κεντρική εργασία αποδίδεται κατ΄ αποκλειστικότητα ένας κωδικός *TaskID* που είναι το αναγνωριστικό της και είναι μοναδικός. Με την χρήση του κωδικού αυτού μπορεί να γίνεται αναφορά σε αυτή και στις διάφορες παραμέτρους της. Ένας ακέραιος αριθμός *SubTasks* αποτελεί τον αριθμό των εργασιών της κύρια εργασίας με την θεώρηση ότι $0<SubTasks\leq 10$. Οι δύο αυτές μεταβλητές αποτελούν και τη βάση της κλάση της κύριας εργασίας.

Η «εργασία στόχος» αποτελεί ουσιαστικά το θεωρητικό υπόβαθρο και την βάση πάνω στην οποία στηρίζεται η προσομοίωση. Το πρακτικό τμήμα και αυτό το οποίο παρέχει μετρήσιμα αποτελέσματα είναι οι υποεργασίες τις οποίες αναλαμβάνουν οι εργαζόμενοι. Οι εργασίες αυτές αναγνωρίζονται μέσω ενός μοναδικού κωδικού *SubTid*. Οι κεντρικές πληροφορίες που αφορούν την προσομοίωση αποθηκεύονται σε μεταβλητές που ανήκουν τις υποεργασίες. Μία πρώτη περιγραφή των καθηκόντων αυτών φαίνεται στο σχήμα 3.3 ενώ κάποιες παράμετροι θα αναλυθούν διεξοδικότερα στη συνέχεια.



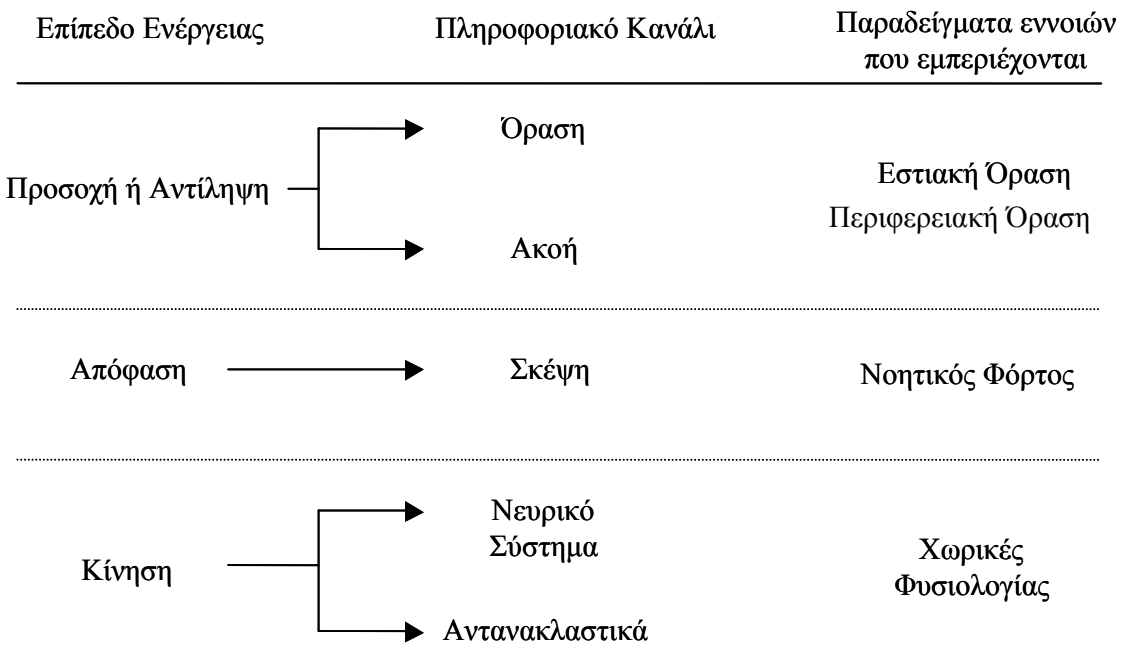
Σχήμα 3.3: Περιγραφή των παραμέτρων των καθηκόντων του συστήματος

- Τυχαία παραγόμενες μεταβλητές

Τυχαία παραγόμενες αναφέρουμε τις μεταβλητές των οποίων η τιμή παράγεται με τυχαίο τρόπο μέσα στο πρόγραμμα. Ο τυχαίος αυτός τρόπος βασίζεται στις γεννήτριες τυχαίων αριθμών της C++ και γίνονται με βάση το ρολόι του συστήματος.

1) Κατηγορία καθήκοντος

Η μεταβλητή αυτή σχετίζεται με τη «θεωρία πολλαπλών πόρων» ως προς τον διαχωρισμό των καθηκόντων με βάση το κανάλι πληροφορίας στο οποίο επιβάλλεται μεγαλύτερη χρήση. Στην περίπτωση μας, ο διαχωρισμός των καναλιών γίνεται σε τρία επίπεδα τα οποία είναι: η προσοχή ή αντίληψη, η απόφαση και η κίνηση. Το πρώτο επίπεδο συμπεριλαμβάνει αισθήσεις όπως η ακοή και η όραση και εμπεριέχει τις έννοιες της εστιακής και περιφερειακής όρασης όπως έχουν αναφερθεί. Το δεύτερο επίπεδο αναφέρεται κυρίως στην έννοια του νοητικού φόρτου ενώ το τρίτο συμπεριλαμβάνει τις χωρικές και φυσιολογικές έννοιες. Σχηματικά η κατηγοριοποίηση αυτή φαίνεται παρακάτω:



Σχήμα 3.4: Κατηγοριοποίηση πληροφοριακών καναλιών σε συνάρτηση με το επίπεδο των ενεργειών και τις έννοιες που εμπεριέχονται

Στο πρόγραμμά μας τα καθήκοντα χωρίζονται σε τέσσερις κατηγορίες ανάλογα με το κανάλι το οποίο επιβαρύνουν περισσότερο. Έτσι η τυχαία τιμή που αποδίδεται μπορεί να έχει τιμές 0,1,2,3 η αντιστοίχηση των οποίων γίνεται με βάση τον ακόλουθο πίνακα:

Τιμή	**Κατηγορία καθήκοντος μέσω επιβάρυνσης που αυτό επιφέρει**
3	Απόφαση
2	Κίνηση
1	Προσοχή ή αντίληψη
0	Γενική επιβάρυνση

2) Χρονική διάρκεια

Για να ελεγχθεί στο επίπεδο του χρόνου η πρόοδος του καθήκοντος, ώστε να είμαστε σε θέση ανάλογα με την πρόοδο αυτή να λάβουμε κατάλληλες αποφάσεις ή να αποτιμήσουμε τα αποτελέσματα, παράγεται μία τυχαία χρονική διάρκεια η οποία ανατίθεται σε κάθε καθήκον.

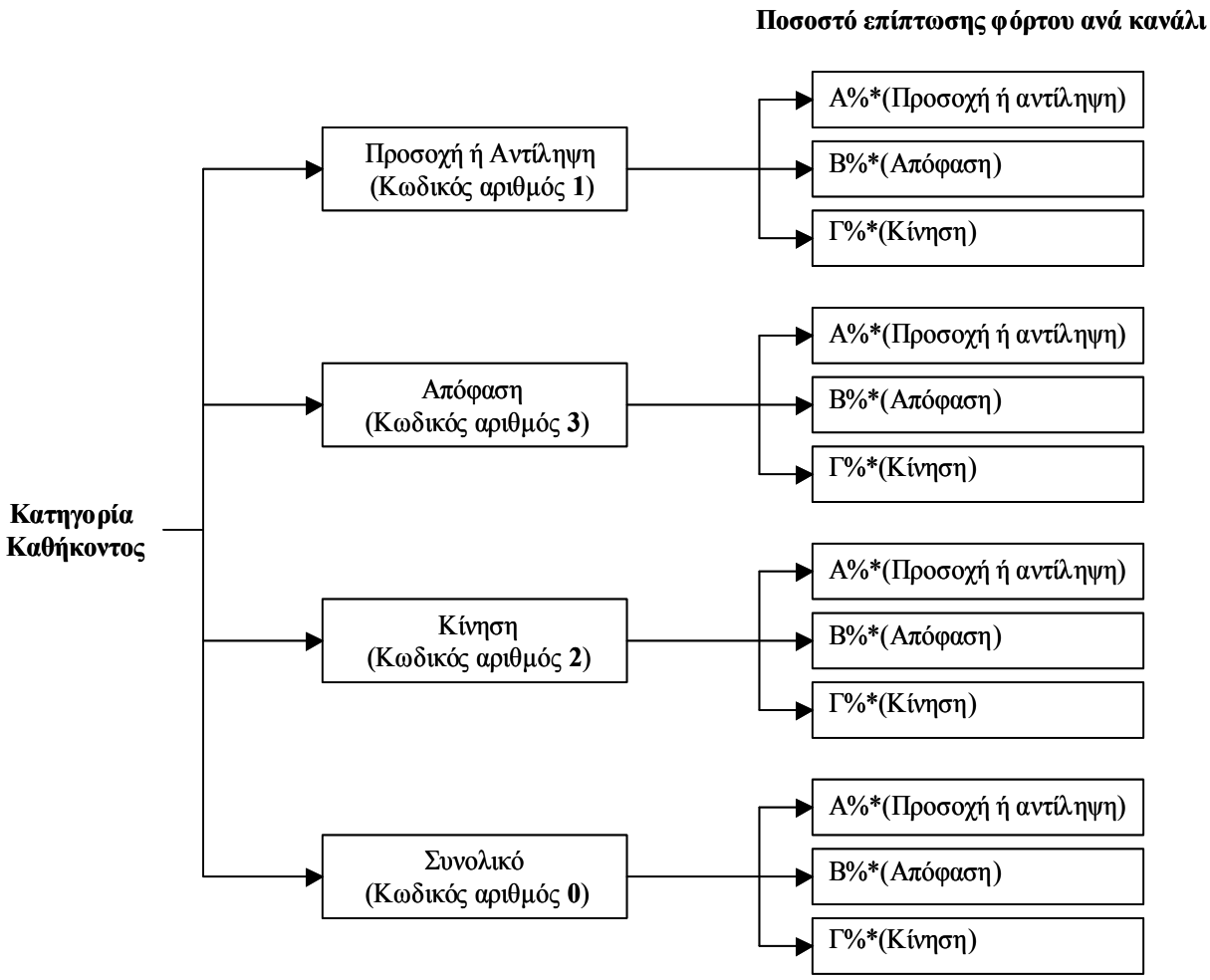
3) Προτεραιότητα

Σε πολλές περιπτώσεις είναι πιθανό, λόγο φόρτου εργασίας ενός εργαζομένου άλλων συνθηκών κάποια από τις εργασίες που βρίσκονται σε εξέλιξη να διακοπεί για να γίνει ανάθεση μίας νέας εργασίας. Στην περίπτωση αυτή, η απόφαση θα ληφθεί μετά από έλεγχο της προτεραιότητας που έχει το συγκεκριμένο καθήκον για την ολοκλήρωση του στόχου. Για την πραγματοποίηση του έλέγχου αυτού χρησιμοποιείται η μεταβλητή της προτεραιότητας. Στην μεταβλητή αυτή αποδίδεται με τυχαίο τρόπο μία τιμή μεταξύ 0 και 3. Όσο η μεγαλύτερη είναι η τιμή αυτή τόσο μεγαλύτερη είναι και η προτεραιότητα της εργασίας.

4) Φόρτος σε κάθε κανάλι

Ο φόρτος που η συγκεκριμένη εργασία θα προσδώσει σε κάθε κανάλι επεξεργασίας παράγεται με τυχαίο τρόπο αλλά σχετίζεται άμεσα με την κατηγορία της εργασίας. Αναλυτικά η ρύθμιση αυτή όπου μπορούμε να δούμε ότι το ποσοστό της επίπτωσης που έχει η κατηγορία του κάθε αναπαρίσταται με τα γράμματα Α,Β,Γ. Οι αριθμοί αυτοί λαμβάνουν τιμές ανά περίπτωση σύμφωνα με τον ακόλουθο πίνακα:

Κατηγορία καθήκοντος	Συγκριτικές τιμές ποσοστών επίπτωσης
Απόφαση	Κυριαρχία Β Β>>Α Β>>Γ
Κίνηση	Κυριαρχία Γ Γ>>Α Γ>>Α
Προσοχή ή αντίληψη	Κυριαρχία Α Α>>Β Α>>Γ
Συνολικό	$A \approx B \approx \Gamma$



Σχήμα 3.5: Φόρτος που προσδίδεται ανά κανάλι με βάση την κατηγορία της εργασίας

- Μεταβλητές παραγόμενες κατά την προσομοίωση

Τις μεταβλητές αυτές τις αναφέρουμε με τον τρόπο αυτό καθώς η τιμή η οποία τους αποδίδεται παράγεται κατά την εκτέλεση του προγράμματος και με βάση τις διαδικασίες που εκτελούνται σε αυτό.

1) Χρονική στιγμή εκκίνησης

Στην μεταβλητή αυτή καταχωρείται η χρονική στιγμή κατά την οποία ο εργαζόμενος στον οποίο έχει ανατεθεί μία συγκεκριμένη εργασία ξεκινάει την υλοποίησή της. Η χρήση της τιμής αυτής μπορεί να είναι είτε για να βρεθεί το στάδιο στο οποίο βρίσκεται η εργασία (ώστε εφόσον κριθεί αναγκαίο, να μελετηθούν πιθανές ενέργειες πάνω σε αυτή) είτε για να γίνει γνωστό το ότι έχει ολοκληρωθεί εφόσον δεν συνέβη κάποιο γεγονός κατά την εξέλιξη της.

2) Σε εξέλιξη ή όχι

Καθ' όλη τη διάρκεια που μία εργασία βρίσκεται σε εξέλιξη, δηλαδή εκτελείται από κάποιον εργαζόμενο, η μεταβλητή αυτή έχει τιμή *true*. Η τιμή αυτή μπορεί να αλλάξει είτε εφόσον η εργασία ολοκληρωθεί είτε εφόσον για κάποιο λόγο διακοπεί η εργασία.

3) Χρονική στιγμή ολοκλήρωσης

Είναι η μεταβλητή στην οποία αποθηκεύεται η χρονική στιγμή ολοκλήρωσης της εργασίας, σε απόλυτο χρόνο, για να γίνει εφικτή η αποτίμηση γεγονότων όπως εάν ολοκληρώθηκε κανονικά ή το κατά πόσο καθυστέρησε η ολοκλήρωση της. Μέσω της μεταβλητής αυτής αποθηκεύονται όλες οι εργασίες που έχουν τελικά ολοκληρωθεί.

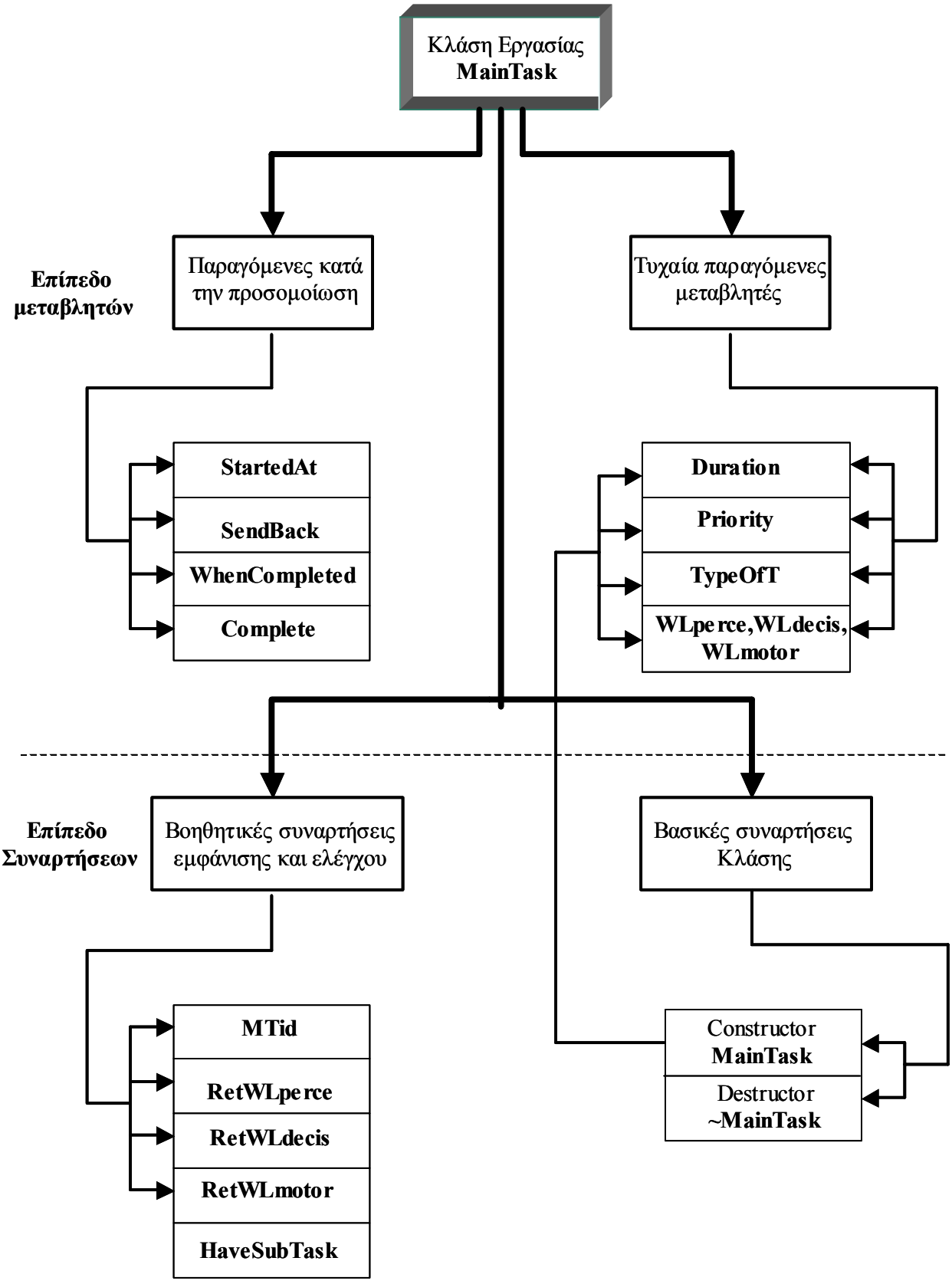
4) Διακοπή ή όχι

Μεταβλητή μέσω της οποίας ελέγχεται αν κάποια εργασία για οποιονδήποτε λόγο έχει διακοπεί. Η αρχική τιμή που αποδίδεται στην μεταβλητή αυτή για κάθε εργασία είναι *false* και αλλάζει μόνο όταν ικανοποιηθούν συγκεκριμένες συνθήκες στο πρόγραμμα.

- Συνάρτηση αρχικοποίησης (constructor)

Στην C++ κάθε κλάση αντιμετωπίζεται ως ξεχωριστό αντικείμενο. Τη στιγμή που θα κατασκευαστεί το αντικείμενο θα πρέπει να γίνουν και οι κατάλληλες αρχικοποιήσεις στις μεταβλητές του. Αυτό πετυχαίνεται με τη συνάρτηση αρχικοποίησης η οποία

είναι μία συνάρτηση που έχει το ίδιο όνομα με την κλάση (στη συγκεκριμένη περίπτωση *MainTask*.



Σχήμα 3.6: Συνολικό διάγραμμα της κλάσης *MainTask*

Στο σχήμα 3.6 φαίνεται συνολικά η κλάση της εργασίας και οι αλληλεπιδράσεις μεταξύ των στοιχείων της. Στο επίπεδο των συναρτήσεων εμφανίζονται οι συναρτήσεις που χρησιμοποιεί η κλάση. Οι πέντε συναρτήσεις, που αναφέρονται ως συναρτήσεις εμφάνισης και ελέγχου χρησιμοποιούνται για τις εξής λειτουργίες:

1)*MTid*
Χρησιμοποιείται για να επιστρέψει τον κωδικό της κύριας εργασίας ώστε να μπορεί το πρόγραμμα να ελέγξει την ύπαρξη αντίστοιχου κωδικού ώστε να το κάνει μοναδικό.

2) *RetWLperce, RetWLdecis, RetWLmotor*
Επιστρέφουν τον τυχαία παραγόμενο φόρτο ανά κανάλι επεξεργασίας.

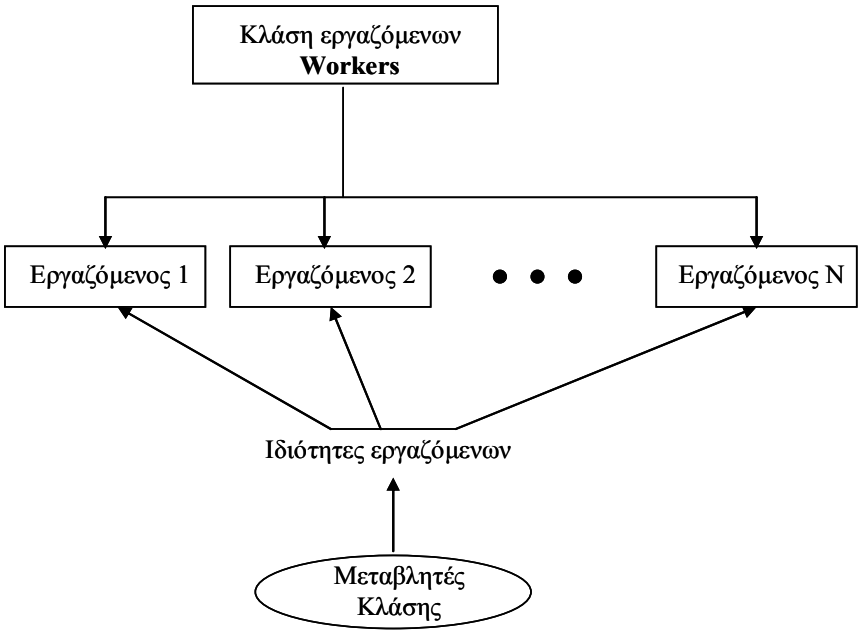
3) *HaveSubTask*
Χρησιμοποιείται για την τυχαία παραγωγή του πλήθους των υποεργασιών που θα συναπαρτίσουν την κύρια εργασία. Ο ελάχιστος αριθμός που μπορεί να δώσει η συνάρτηση αυτή είναι 2.

Στις βασικές συναρτήσεις ελέγχου τις κλάσης περιλαμβάνεται και ο destructor που είναι μία συνάρτηση η οποία καταστρέφει το αντικείμενο μετά τη λήξη της προσομοίωσης για την αποδέσμευση της μνήμης.

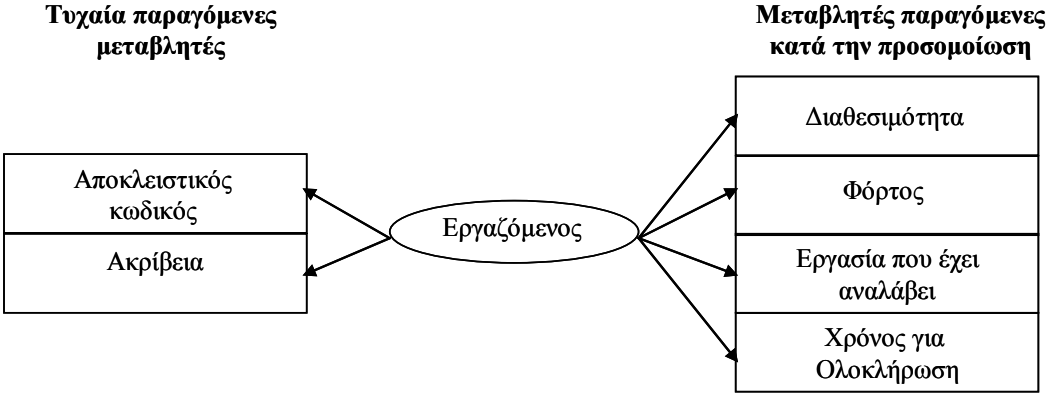
2.2.2 Κλάση εργαζομένων Workers

Η κλάση αυτή εμπεριέχει όλες εκείνες τις παραμέτρους οι οποίες βοηθούν στην προσομοίωση και την παραγωγή των αποτελεσμάτων για το συγκεκριμένο σύστημα. Θεωρούμε γενικά ότι υπάρχει μία «πηγή» από εργαζόμενους με διαφορετικές ιδιότητες ο καθένας. Η κλάση αυτή φαίνεται στο σχήμα 3.7 όπου μέσω των μεταβλητών της κλάσης αντιστοιχίζονται οι ιδιότητες στον κάθε εργαζόμενο.

Οι ιδιότητες του εργαζόμενου διαχωρίζονται και αυτές σε παραγόμενες τυχαία κατά την αρχικοποίηση του προγράμματος και σε παραγόμενες κατά την προσομοίωση. Η απεικόνιση τους φαίνεται στο ακόλουθο σχήμα:



Σχήμα 3.7: Κλάση *Workers* με αντιστοίχηση των ιδιοτήτων τους στις μεταβλητές της κλάσης



Σχήμα 3.8: Περιγραφή των ιδιοτήτων των εργαζομένων του συστήματος

- Τυχαία παραγόμενες μεταβλητές

1) Αποκλειστικός Κωδικός

Κάθε εργαζόμενος έχει ένα μοναδικό κωδικό αναγνώρισης που αναφέρεται ως *WorkerName* στο πρόγραμμα. Ο κωδικός αυτός χρησιμοποιείται για να αναγνωρίζεται ο συγκεκριμένος εργαζόμενος ώστε να είναι εφικτός ο έλεγχος της εργασίας που επιτελεί. Με τον τρόπο αυτό μπορούν να ληφθούν αποφάσεις και να πραγματοποιηθούν ενέργειες οι οποίες να αφορούν μόνο τον συγκεκριμένο εργαζόμενο.

2)Ακρίβεια

Η τιμή της ακρίβειας με την οποία μπορεί κάποιος εργαζόμενος να φέρει σε πέρας ένα καθήκον παράγεται με τυχαίο τρόπο για να διατηρηθεί η γενικότητα. Η ακρίβεια κατηγοριοποιείται ανάλογα με το πληροφοριακό κανάλι που χρησιμοποιεί ο εργαζόμενος και εκφράζεται σαν ένα ποσοστό επί της εκατό. Η τιμή της για κάθε εργαζόμενο αποθηκεύεται σε ένα πίνακα τεσσάρων θέσεων σύμφωνα με τον ακόλουθο πίνακα:

Θέση Πίνακα	Τύπος εργασίας
0	**Προσοχή ή αντίληψη**
1	**Κίνηση**
2	**Απόφαση**
3	**Γενική εργασία**

- Μεταβλητές παραγόμενες κατά την προσομοίωση

1) Διαθεσιμότητα

Η μεταβλητή αυτή υποδηλώνει το αν ένας εργάτης είναι διαθέσιμος και μπορεί έτσι να αναλάβει τη διεκπεραίωση κάποιας εργασίας. Για όλους τους εργάτες του συστήματος η αρχική τιμή αυτής της παραμέτρου είναι *true* και αλλάζει μέσω των διάφορων καταστάσεων που μπορεί να εμφανιστούν κατά τη διάρκεια της προσομοίωσης.

2) Φόρτος

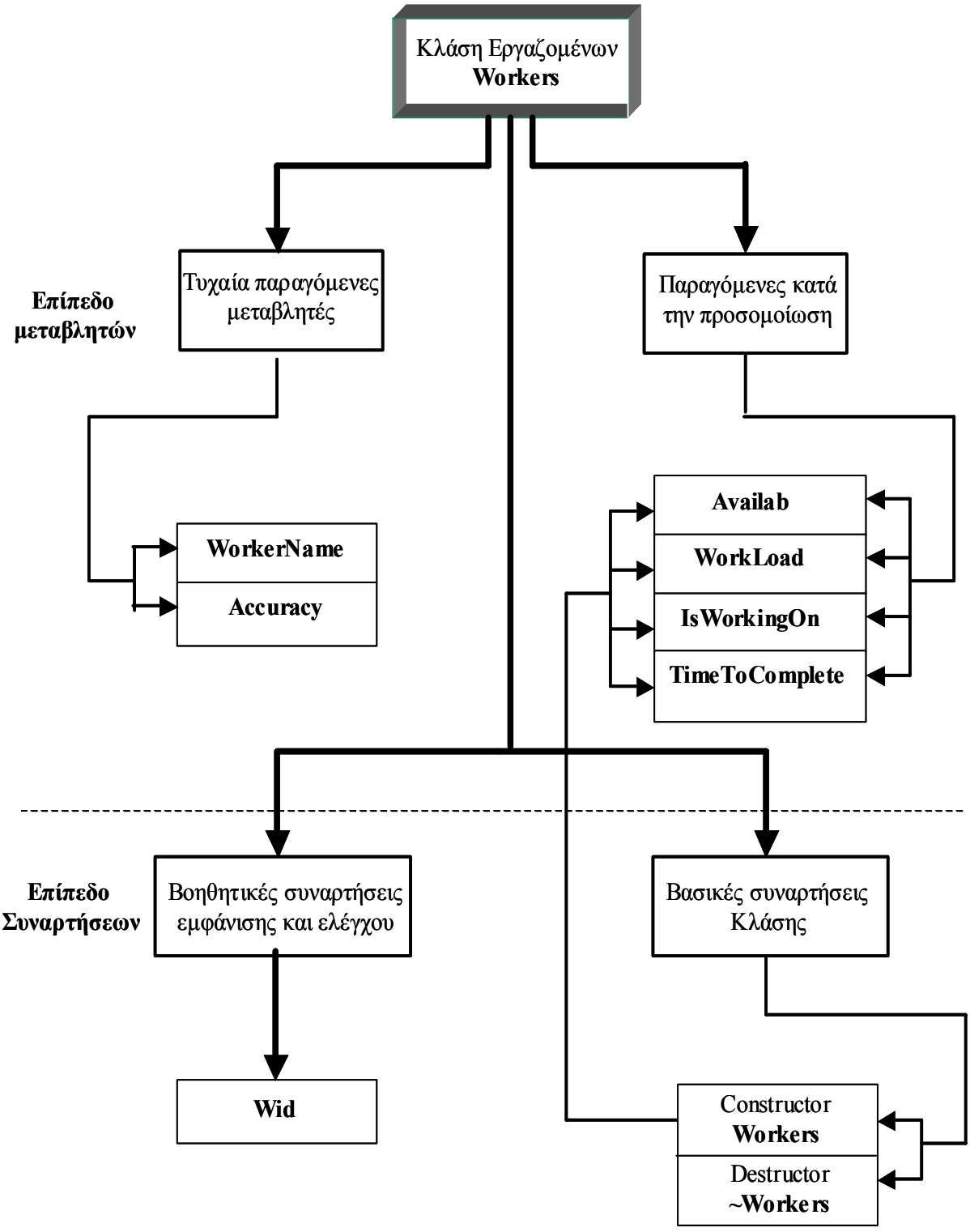
Είναι ο συνολικός φόρτος εργασίας του κάθε εργαζόμενο υπολογιζόμενος κατά τη διάρκεια της προσομοίωσης με βάση τις εργασίες που αυτός έχει αναλάβει. Υπολογίζεται ξεχωριστά για τον κάθε εργαζόμενο και αποτελεί ένα από τα βασικά αποτελέσματα εξόδου του προγράμματος.

3) Εργασία που έχει αναλάβει

Στη μεταβλητή αυτή αποθηκεύεται για τον κάθε εργαζόμενο η εργασία ή οι εργασίες οι οποίες του έχουν ανατεθεί. Η αποθήκευση αυτή γίνεται με βάση τον μοναδικό αναγνωριστικό κωδικό που έχει η κάθε εργασία.

4) Χρόνος για ολοκλήρωση

Εφόσον έχει αναλάβει ο εργαζόμενος ένα καθήκον, και με βάση τη διάρκεια που το χαρακτηρίζει, η μεταβλητή αυτή δίνει μία προσέγγιση του χρόνου στον οποίο πρόκειται να ολοκληρωθεί η εργασία αυτή.



Σχήμα 3.9: Συνολικό διάγραμμα της κλάσης *Workers*

Η κλάση των εργαζομένων περιλαμβάνει δύο κύριες συναρτήσεις. Η πρώτη είναι ο constructor της συνάρτησης στον οποίο πραγματοποιούνται οι αρχικοποιήσεις των απαραίτητων μεταβλητών. Παράγεται ο μοναδικός τυχαίος κωδικός του κάθε εργάτη, αρχικοποιούνται η διαθεσιμότητα στην τιμή *true* και ο φόρτος στην τιμή 0 για να εκλάβουν στη συνέχεια τιμές ανάλογα με τις διαφοροποιήσεις που θα προκύψουν.

Η δεύτερη συνάρτηση είναι αυτή που επιστρέφει το μοναδικό κωδικό του εργαζόμενου και η οποία χρησιμοποιείται κατά κόρον στις συναρτήσεις του προγράμματος για τον προσδιορισμό του εργάτη.

Επίσης στην κλάση περιλαμβάνεται και ο destructor που καταστρέφει τα αντικείμενα που έχουν δημιουργηθεί μετά τη λήξη της προσομοίωσης. Διαγραμματικά η κλάση του εργαζόμενου φαίνεται στο σχήμα 3.9

2.3 Συναρτήσεις προγράμματος

2.3.1 Αλγόριθμοι δρομολόγησης της εργασίας

Πριν περιγράψουμε τις συναρτήσεις και την συγκεκριμένη λειτουργίας τους είναι σημαντικό να αναλυθούν οι γενικές μέθοδοι δρομολόγησης των εργασιών πάνω στις οποίες βασιστήκαμε για την κατασκευή των συναρτήσεων. Έχουν αναπτυχθεί πέντε οι οποίοι βασίζονται στην ακρίβεια, σε αλλαγή εργασίας που βρίσκεται σε αναμονή και σε αλλαγή εργασίας που βρίσκεται σε εξέλιξη.

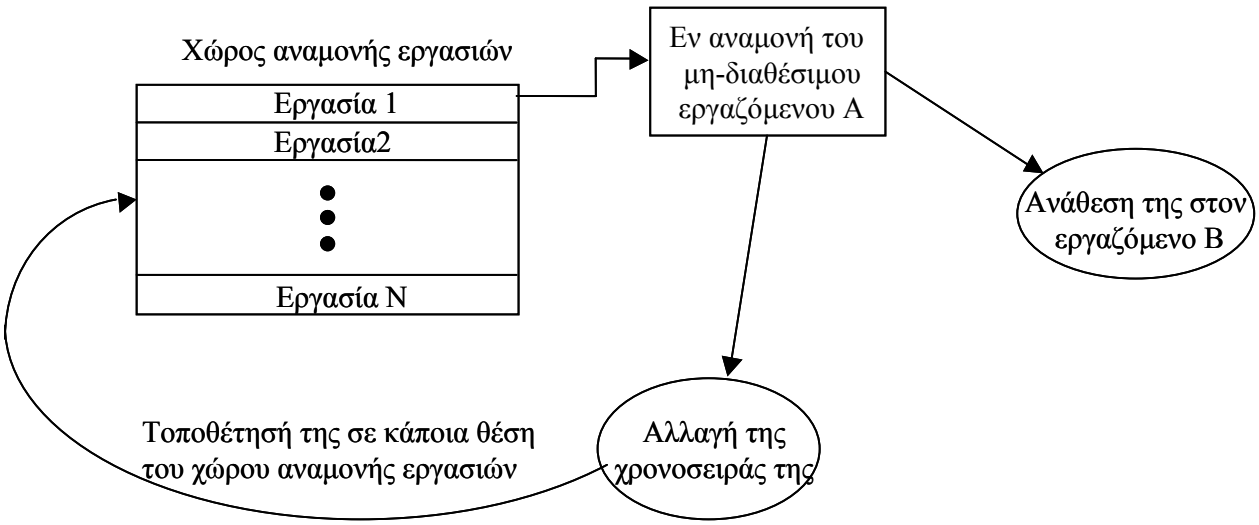
Α. Δρομολόγηση με βάση την ακρίβεια

Είναι η πιο απλή από τις μεθόδους δρομολόγησης που θα αναφέρουμε. Μπορούμε απλά να μειώσουμε την απαιτούμενη ακρίβεια με την οποία θέλουμε να εκπληρωθεί η εργασία και έτσι να αναθέσουμε μία συγκεκριμένη εργασία σε κάποιον εργάτη που την πραγματοποιεί με λιγότερη ακρίβεια. Με μια πρώτη ματιά ο αλγόριθμος αυτός δεν εμφανίζεται διαισθητικά σωστός. Στην πραγματικότητα όμως δεν εφαρμόζεται επιφέροντας αρνητικά αποτελέσματα. Δεν διακόπτεται δηλαδή η εργασία που έχει ξεκινήσει κάποιος εργαζόμενος, που μάλιστα την πραγματοποιεί με καλή ακρίβεια, για να δοθεί σε κάποιον άλλον που θα την πραγματοποιήσει με χαμηλότερη ακρίβεια χωρίς συγκεκριμένο λόγο.

Αιτίες που μπορεί να οδηγήσουν σε τέτοια κίνηση είναι, κατά πρώτο λόγο, ο φόρτος του εργαζομένου. Αν έχει αυξηθεί σημαντικά τότε παρά τη φαινομενικά μεγάλη ακρίβεια μπορεί τελικά να οδηγηθεί σε περισσότερα σφάλματα ή σε μεγαλύτερη καθυστέρηση από κάποιον άλλο εργαζόμενο που με μια πρώτη ματιά εμφανίζεται χειρότερος για τη συγκεκριμένη εργασία. Κατά δεύτερο λόγο, η ανάγκη σε ακρίβεια σε κάποια άλλη εργασία που εκτελείται παράλληλα από τον ίδιο εργαζόμενο μπορεί να οδηγήσει στην διακοπή της πρώτης εργασίας που εκτελεί. Μια άλλη αιτία είναι ότι ο συγκεκριμένος εργαζόμενος μπορεί να είναι πάρα πολύ καλός (την πραγματοποιεί με πολύ μεγάλη ακρίβεια) σε κάποια συγκεκριμένη εργασία η οποία δεν έχει ανατεθεί ακόμη και να θεωρηθεί κατάλληλος για αυτή

Β. Αλλαγή εργασίας σε αναμονή

Ο συγκεκριμένος αλγόριθμος δρομολόγησης περιλαμβάνει δύο υποπεριπτώσεις: αλλαγή του εργαζόμενου που θα την αναλάβει ή επιστροφή της εργασίας στην πηγή.



Σχήμα 3.10: Λειτουργία της δρομολόγησης με βάση το γεγονός ότι η εργασία βρίσκεται σε αναμονή συγκεκριμένου εργαζόμενου

Β1. Αλλαγή του εργαζόμενου

Κατά τον τρόπο αυτό αντί να βρίσκεται η εργασία σε αναμονή του συγκεκριμένου εργαζόμενου που θέλουμε να την αναθέσουμε, ανατίθεται σε κάποιον άλλο εργαζόμενο ο οποίος μπορεί να την φέρει σε πέρας. Η εργασία μπορεί να βρίσκεται

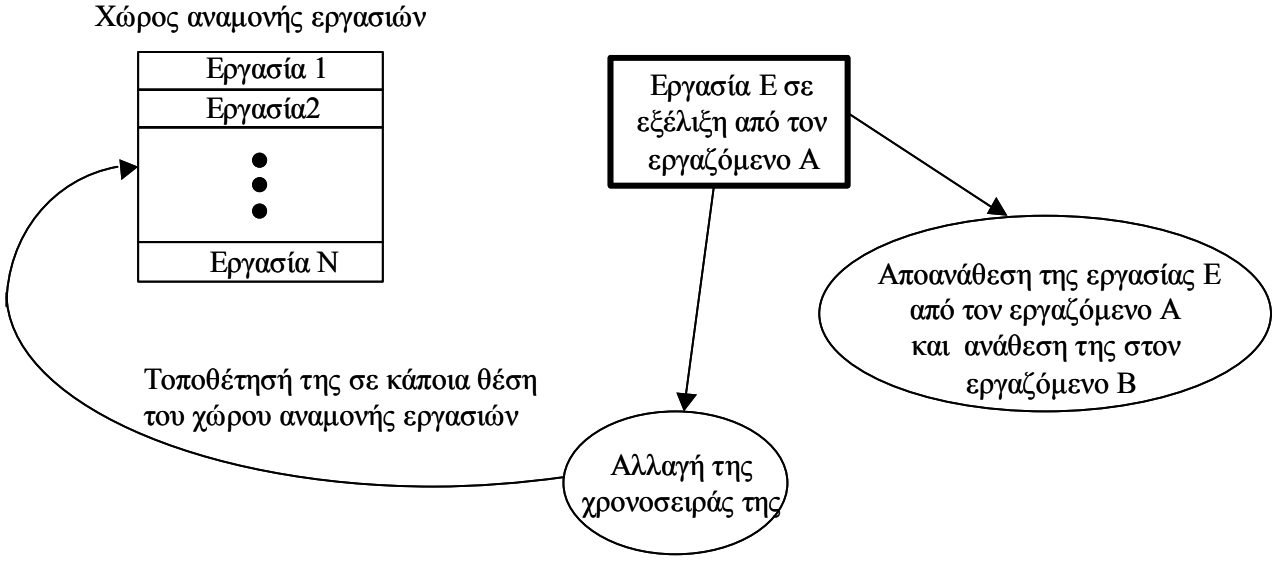
σε αναμονή κάποιου εργαζόμενου είτε για λόγους ακρίβειας είτε για λόγους νοητικού φόρτου του συγκεκριμένου εργαζόμενου. Έτσι η εργασία παύει να βρίσκεται σε αναμονή εφόσον υπάρχει εργαζόμενος στον οποίο ανατίθεται.

Β2. Αλλαγή της χρονοσειράς

Ο δεύτερος τρόπος είναι να επιστρέψει η συγκεκριμένη εργασία που βρίσκεται σε αναμονή σε κάποιο χώρο αποθήκευσης εργασιών και να ξαναβρεθεί στη θέση αναμονής ύστερα από κάποιες χρονικές μονάδες ή με βάση κάποιο άλλο σκεπτικό. (π.χ. τοποθετείται στο τέλος, ή δεύτερη κτλ.). Αυτό μπορεί να συμβεί γιατί ο χρόνος αναμονής της ξεπερνά κάποιο κατώφλι ή επειδή ο εργαζόμενος που πρόκειται να την αναλάβει θα αυξήσει πολύ το φόρτο του ή επειδή δεν υπάρχει κατάλληλος εργαζόμενος για να την αναλάβει.

Γ. Αλλαγή εργασίας σε εξέλιξη

Ο αλγόριθμος αυτός δρομολόγησης εργασίας σε αντιστοιχία με τον προηγούμενο περιλαμβάνει δύο υποπεριπτώσεις: αλλαγή του εργαζόμενου που θα την εκτελεί ή διακοπή της εργασίας και επιστροφή της στον χώρο αναμονής.



Σχήμα 3.11: Λειτουργία της δρομολόγησης εργασίας με βάση το γεγονός ότι η εργασία βρίσκεται ήδη σε εξέλιξη

Γ1. Αλλαγή του εργαζόμενου

Κατά τον πρώτο τρόπο, η εργασία που βρίσκεται σε εξέλιξη και πραγματοποιείται από κάποιον εργαζόμενο ανατίθεται σε κάποιον άλλο που μπορεί να την πραγματοποιήσει ώστε ο συγκεκριμένος εργαζόμενος που αντικαταστάθηκε να αναλάβει κάποια άλλη εργασία. Έτσι για παράδειγμα, αν θεωρηθεί ότι ο εργαζόμενος Α μπορεί να πραγματοποιήσει κάποια εργασία παράλληλα με την εργασία Ε, με μεγάλη ακρίβεια, αλλά αυτό θα αυξήσει κατακόρυφα τον φόρτο ή θα μειώσει πολύ την ακρίβεια του, τότε ανατίθεται σε αυτόν η καινούργια εργασία αλλά ή Ε ανατίθεται σε κάποιον άλλο εργαζόμενο Β που μπορεί να την πραγματοποιήσει με ικανοποιητική ακρίβεια.

Γ2. Αλλαγή της χρονοσειράς

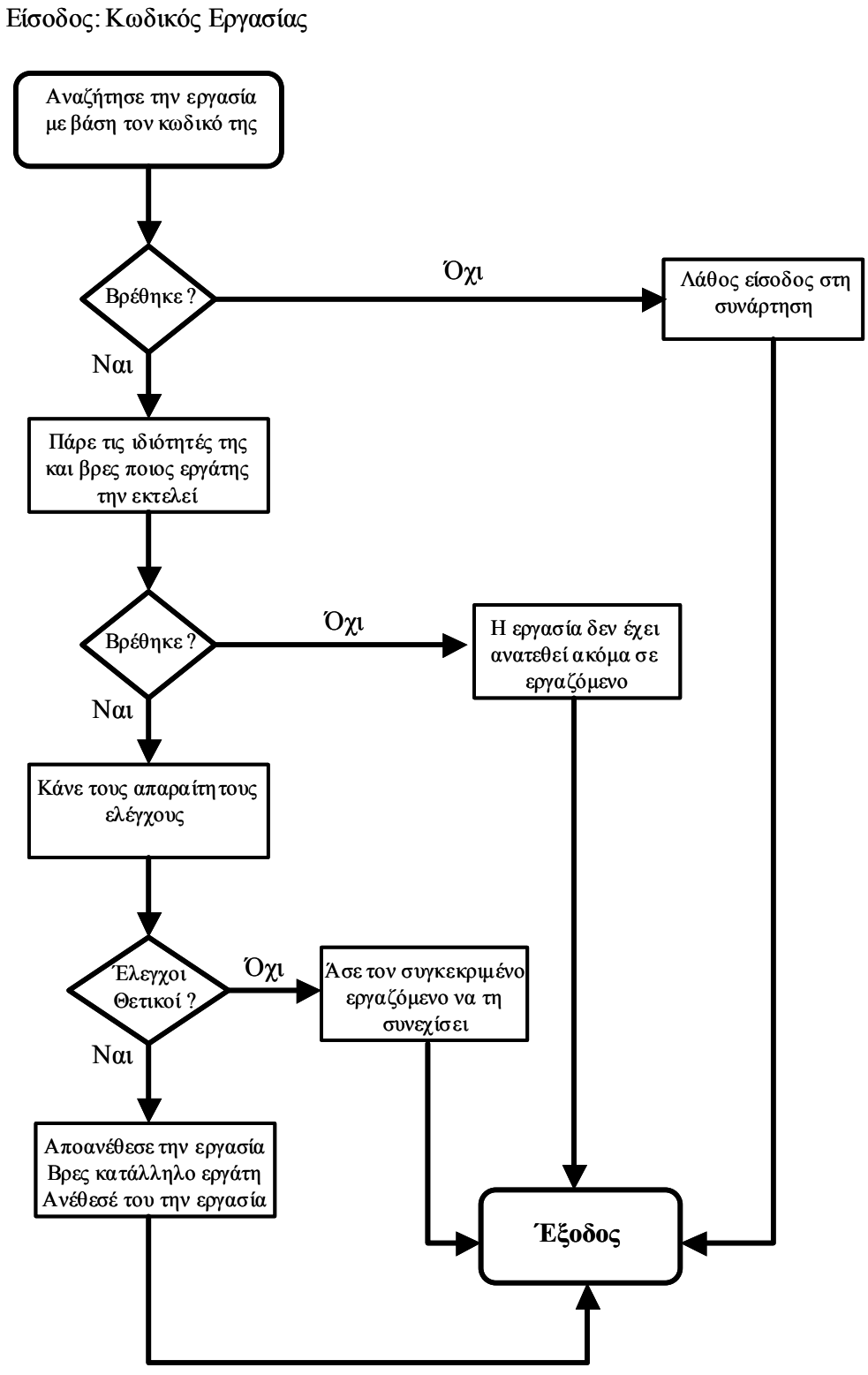
Σύμφωνα με αυτή τη μέθοδο μια εργασία η οποία βρίσκεται σε εξέλιξη, υπό προϋποθέσεις, μπορεί να βρεθεί στον χώρο αναμονής των εργασιών υπό συγκεκριμένες προϋποθέσεις. Οι προϋποθέσεις αυτές είναι κατά βάση: (α) η προς αντικατάσταση εργασία να βρίσκεται σε πρώιμο στάδιο και (β) η εργασία που θα αρχίσει να εκτελείται να έχει μεγάλη προτεραιότητα στην ολοκλήρωση της κύριας εργασίας.

2.3.2 Συναρτήσεις υλοποίησης αλγορίθμων δρομολόγησης

- AssignAccuracyBased

Η συνάρτηση αυτή αποτελεί την υλοποίηση του αλγόριθμου δρομολόγησης εργασίας με βάση την ακρίβεια. Το όρισμα που παίρνει η συνάρτηση αυτή σαν είσοδο είναι ο κωδικός της εργασίας η οποία θέλουμε να ανατεθεί σε κάποιον εργάτη με βάση την απόδοση.

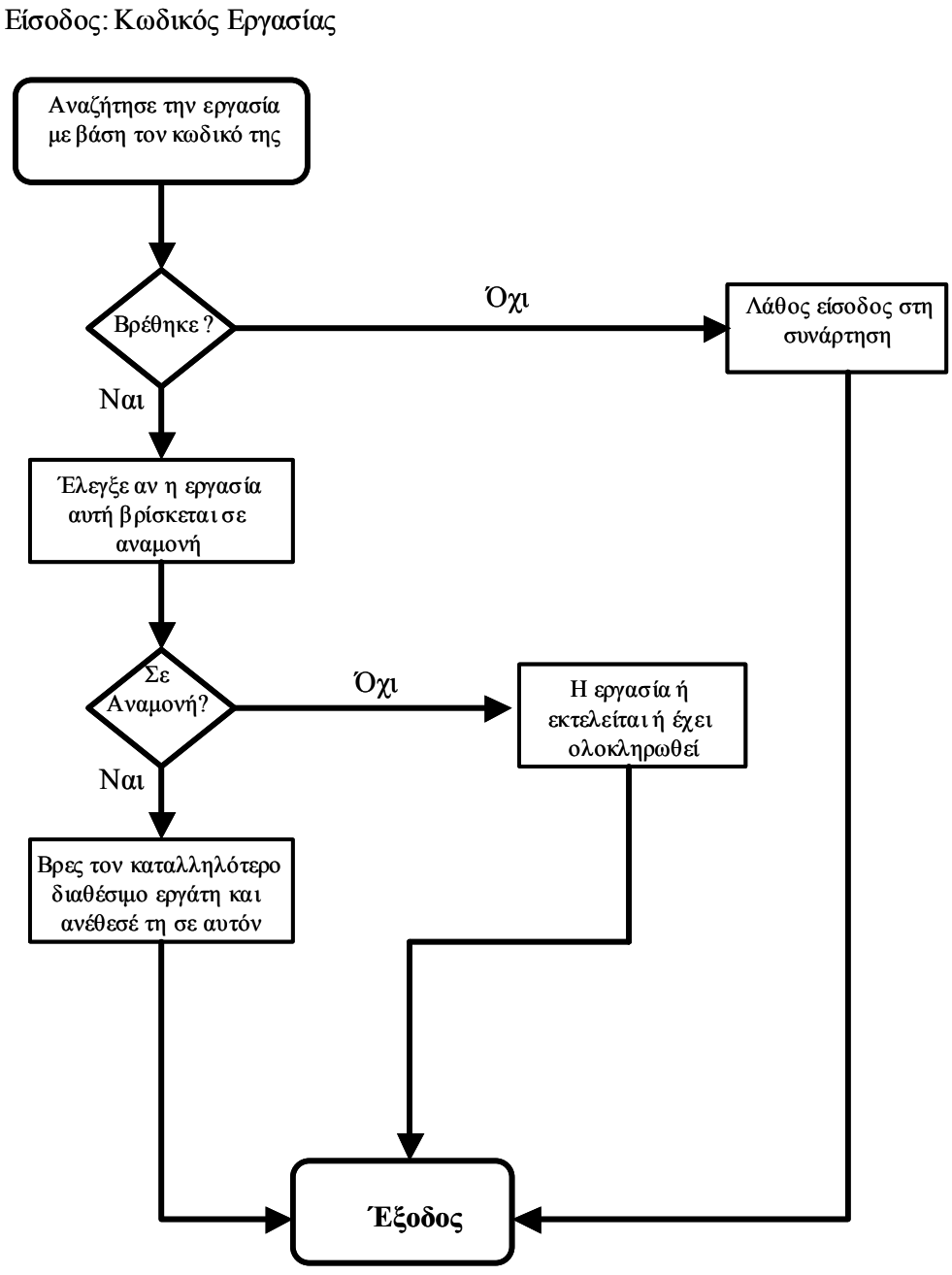
Η λειτουργία της συνάρτησης έχει ως εξής: Αρχικά μέσω του κωδικού της εργασίας βρίσκουμε της ιδιότητες της. Στη συνέχεια βρίσκουμε τον εργαζόμενο στον οποίο έχει ανατεθεί και από αυτόν την ακρίβεια με την οποία την εκτελεί. Από εκεί και πέρα καλείται μία συνάρτηση η οποία αποαναθέτει την εργασία από αυτόν, μία άλλη συνάρτηση βρίσκει τον εργαζόμενο που μπορεί να την εκτελέσει και τέλος καλείται η συνάρτηση η οποία αναθέτει την εργασία στον καινούριο εργαζόμενο. Το διάγραμμα ροής της συνάρτησης αυτής φαίνεται στο σχήμα 3.12:



Σχήμα 3.12: Διάγραμμα ροής της συνάρτησης *AssignAccuracyBased*

- AssignIfTaskWaits

Η συνάρτηση αυτή αποτελεί την υλοποίηση του αλγόριθμου δρομολόγησης εργασίας κατά τον οποίο όταν η εργασία βρίσκεται σε αναμονή κάποιου εργαζόμενου ανατίθεται σε κάποιον άλλον ο όποιος είναι σε θέση να τη φέρει σε πέρας.



Σχήμα 3.13: Διάγραμμα ροής της συνάρτησης *AssignIfTaskWaits*

Η συνάρτηση αυτή λειτουργεί με τον εξής τρόπο Αρχικά χρησιμοποιώντας τον κωδικό της εργασίας βρίσκουμε τις ιδιότητες της και ελέγχουμε την κατάσταση της (μπορεί για παράδειγμα λόγω λάθους εισόδου η συγκεκριμένη εργασία να βρίσκεται ήδη σε εξέλιξη). Στην περίπτωση που επιβεβαιώσουμε ότι η εργασία βρίσκεται σε αναμονή, αναζητούμε τον διαθέσιμο εργάτη με την μεγαλύτερη ακρίβεια για την συγκεκριμένη κατηγορία εργασίας και του ανατίθεται. Το διάγραμμα ροής της συνάρτησης αυτής φαίνεται στο σχήμα 3.13.

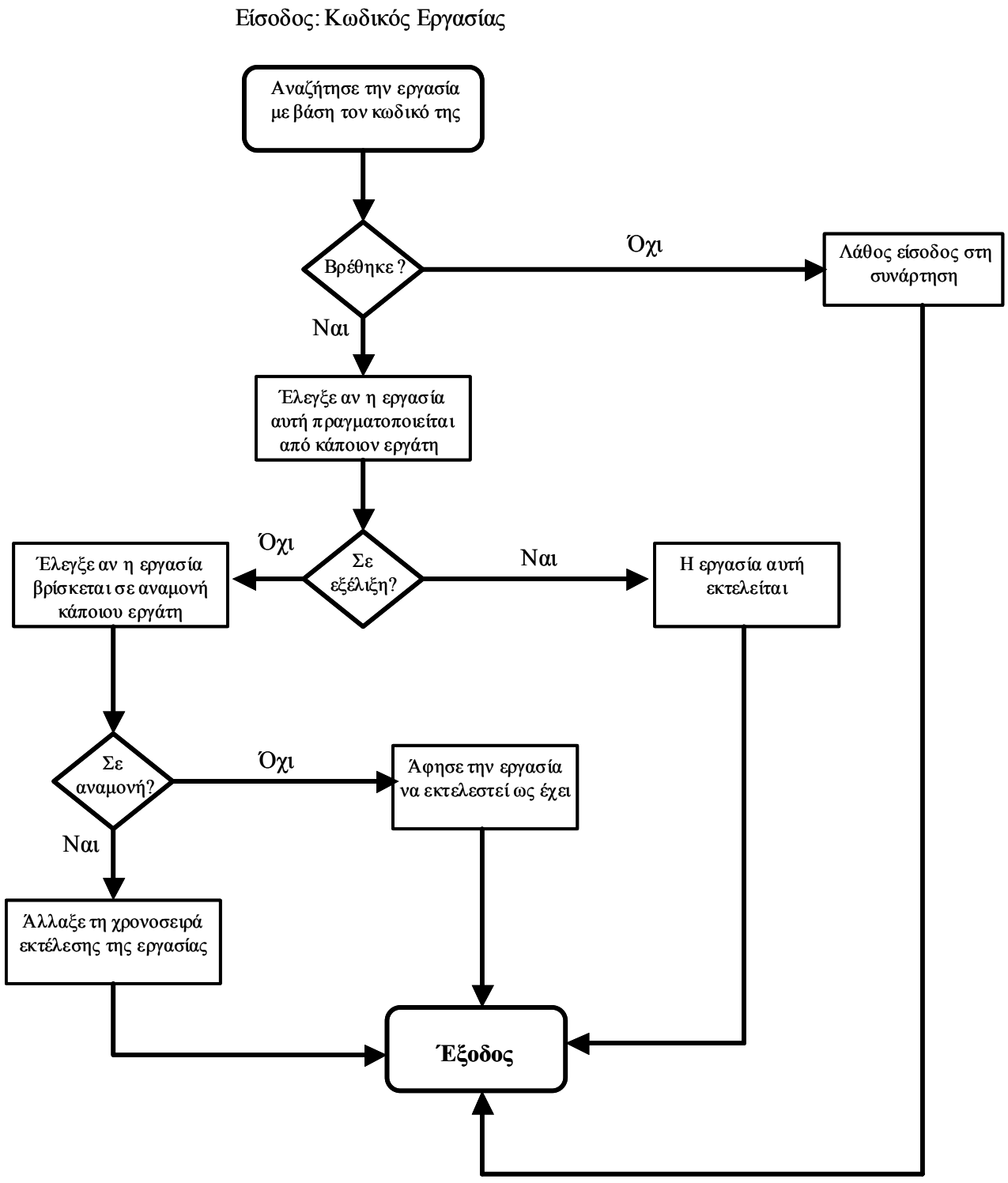
- SendTaskBack

Με τη συνάρτηση αυτή υλοποιούμε στο σύστημα τον αλγόριθμο δρομολόγηση που αλλάζει τη χρονοσειρά εκτέλεσης της μίας εργασίας όταν αυτή βρίσκεται σε αναμονή κάποιου συγκεκριμένου εργαζόμενου. Οι λειτουργίες που πραγματοποιεί η συνάρτηση αυτή είναι οι ακόλουθες: Αρχικά και χρησιμοποιώντας τον κωδικό της εργασίας ελέγχεται η κατάσταση στην οποία αυτή βρίσκεται. Εφόσον επιβεβαιωθεί ότι η εργασία βρίσκεται σε αναμονή εργαζόμενου, τότε ακολουθείται ένα από τα ακόλουθα δύο βήματα: (α) είτε αυτή μετατοπίζεται στο τέλος του χώρου αναμονής εργασιών, (β) είτε τοποθετείται στην δεύτερη θέση για να αποδοθεί μετά από κάποιο χρονικό διάστημα. Το διάγραμμα ροής της φαίνεται στο σχήμα 3.14.

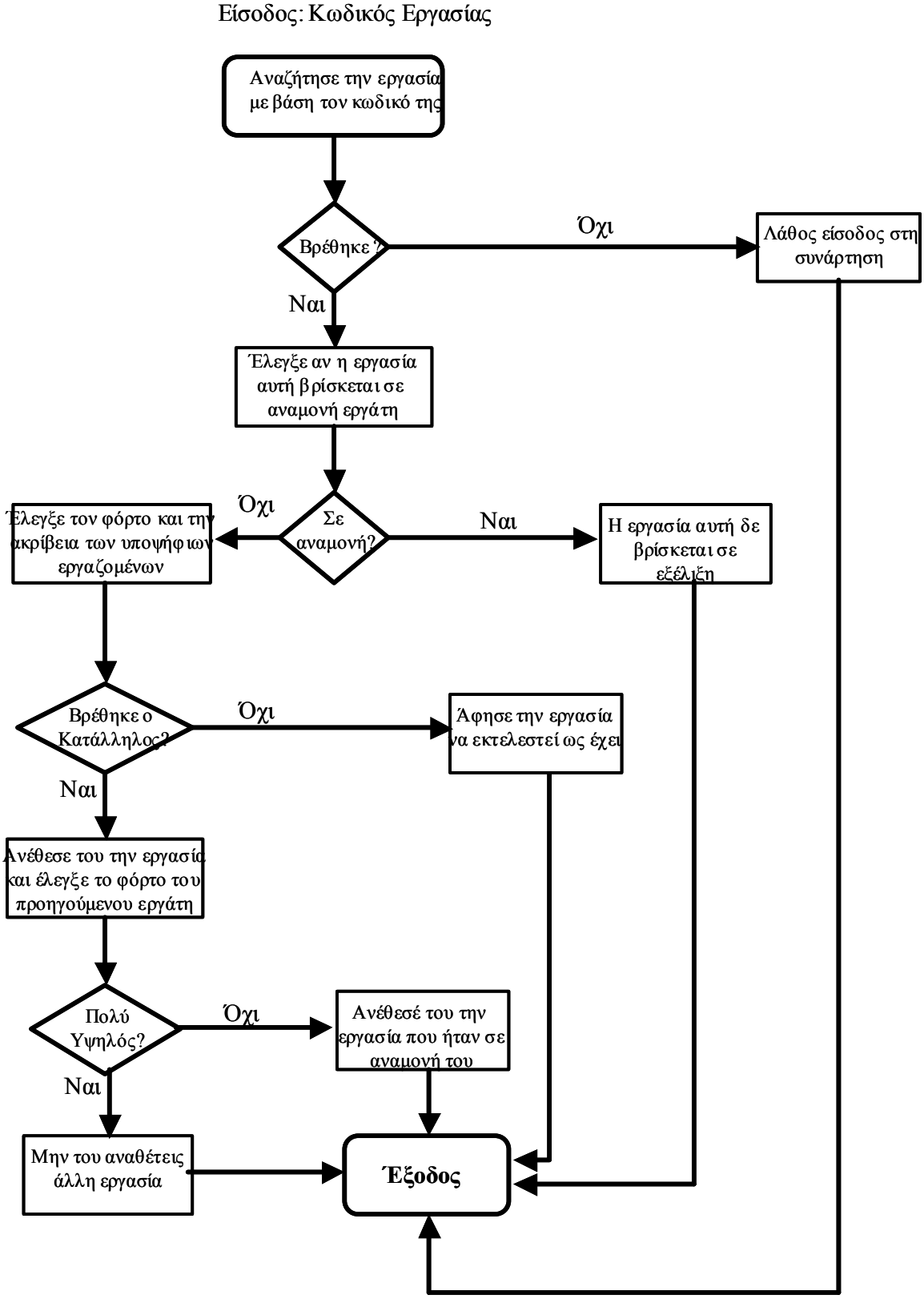
- ChangeWorkerTaskInProgress

Η συνάρτηση αυτή υλοποιεί τον αλγόριθμο δρομολόγησης κατά τον οποίο διακόπτεται η εξέλιξη κάποιας εργασία από τον εργαζόμενο που την πραγματοποιεί και ανατίθεται σε κάποιον άλλο που μπορεί να την πραγματοποιήσει με ικανοποιητική ακρίβεια, ώστε να μπορεί να ανατεθεί κάποια άλλη εργασία στον πρώτο.
Τα βήματα λειτουργίας της συγκεκριμένης συνάρτησης έχουν ως ακολούθως: Κατ΄ αρχάς και με βάση τον κωδικό της. ελέγχεται εάν έχει γίνει κάποιο λάθος και η εργασία αυτή βρίσκεται σε αναμονή. Μετά από τον έλεγχο αυτό βρίσκουμε τον εργαζόμενο που την εκτελεί. Ελέγχεται ο φόρτος των εργαζομένων που μπορούν να την αναλάβουν για να αποδοθεί σε αυτόν που ικανοποιεί τις προϋποθέσεις και συνάμα έχει τον χαμηλότερο φόρτο. Στη συνέχεια η εργασία ανατίθεται σε αυτόν που κρίθηκε καταλληλότερος για τη συνέχισή της και στον εργαζόμενο από τον οποίο πήραμε την εργασία που εκτελούσε ανατίθεται, εφόσον ο φόρτος του δεν θα αυξηθεί

δραματικά, η εργασία στην οποία ήταν ο καταλληλότερος. Το διάγραμμα ροής αυτής της συνάρτησης φαίνεται στο σχήμα 3.15



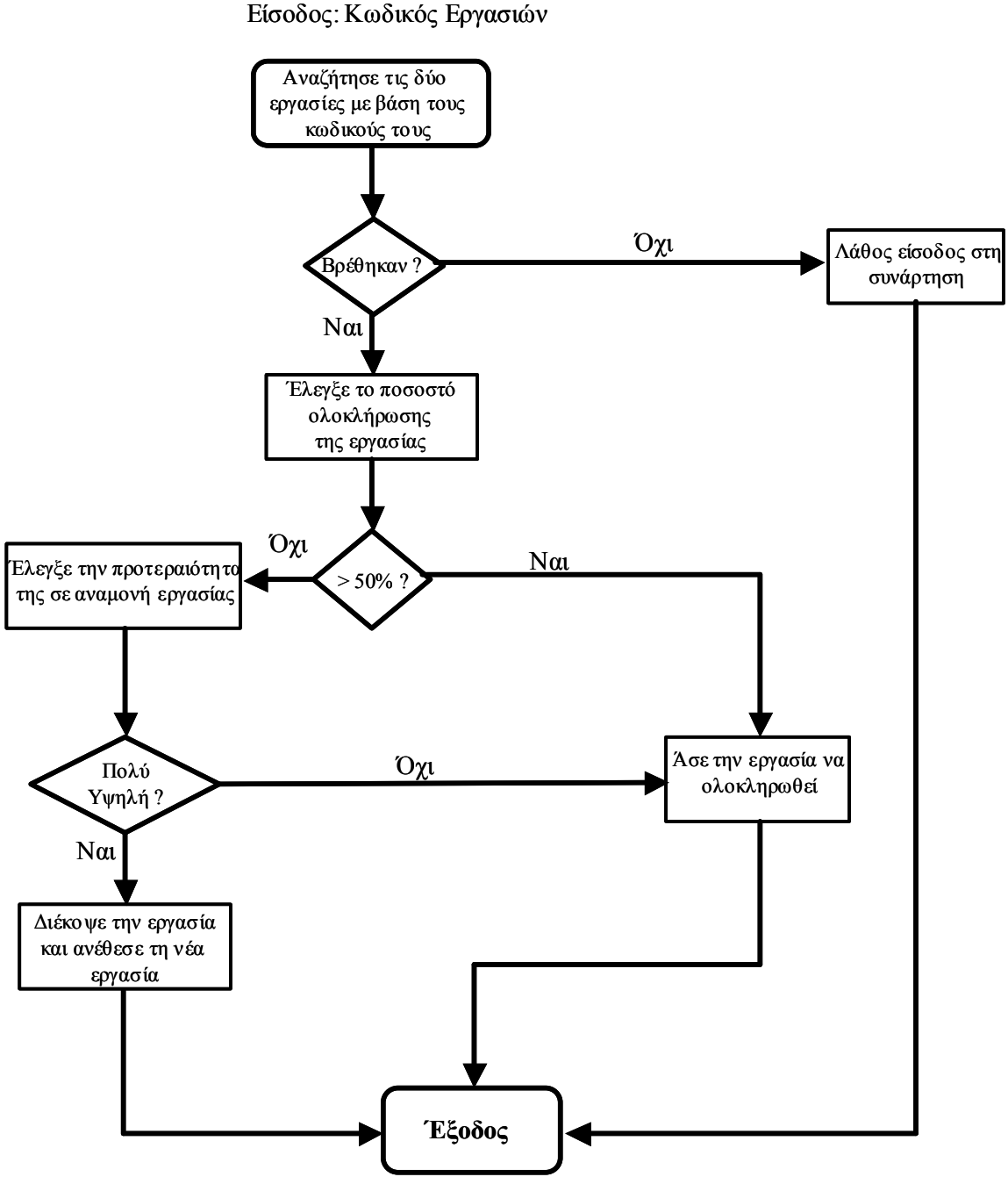
Σχήμα 3.14: Διάγραμμα ροής της συνάρτησης *SendTaskBack*



Σχήμα 3.15: Διάγραμμα ροής της συνάρτησης *ChangeWorkerTaskInProgress*

- SendBackTaskInProgress

Με τη συνάρτηση αυτή υλοποιούμε τον αλγόριθμο δρομολόγησης κατά τον οποίο κάποια εργασία η οποία βρίσκεται σε εξέλιξη διακόπτεται και επιστρέφει πίσω στο χώρο αναμονής εργασιών για να αποδοθεί και να εκτελεστεί κάποια άλλη εργασία που έχει πολύ μεγαλύτερη προτεραιότητα στην ολοκλήρωση της εργασίας στόχου.



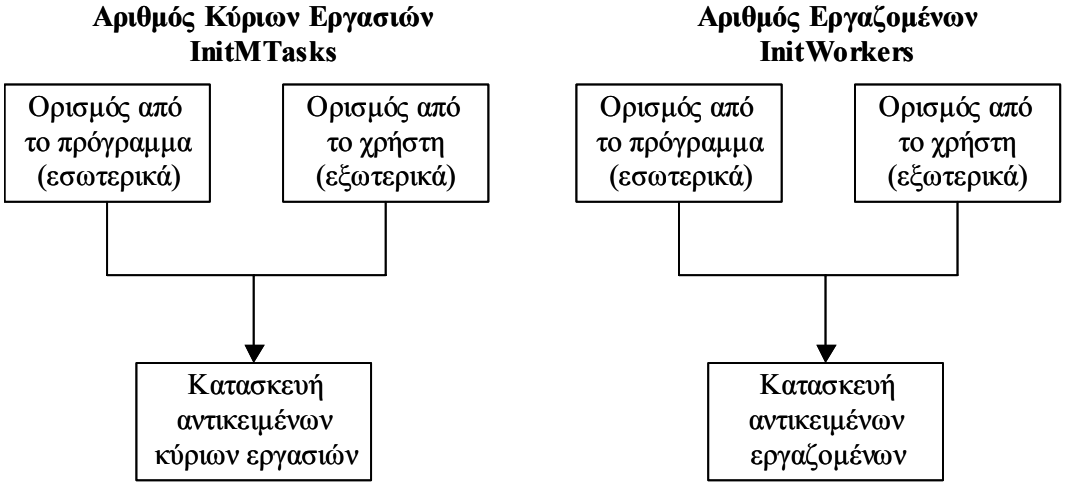
Σχήμα 3.16: Διάγραμμα ροής της συνάρτησης *SendBackTaskInProgress*

Η λειτουργία της συγκεκριμένης συνάρτησης είναι απλή. Κατ΄ αρχάς και με βάση τους κωδικούς των δύο εργασιών (της υπό εξέλιξη εργασίας και αυτής που περιμένει την ανάθεση της) ελέγχεται η κατάσταση τους για τυχόν σφάλματα. Στη συνέχεια γίνεται έλεγχος του σημείου στο οποίο βρίσκεται η εξέλιξη της εργασία που εκτελείται και της προτεραιότητας της εργασίας που αναμένει. Αν η εξέλιξη έχει περάσει το 50% τότε δε γίνεται κάποια ενέργεια και η εργασία αφήνεται να ολοκληρωθεί. Αν το ποσοστό αυτό είναι χαμηλότερο και η προτεραιότητα της σε αναμονή εργασία η υψηλότερη τότε η πρώτη εργασία διακόπτεται και ξεκινά η δεύτερη. Το διάγραμμα ροής αυτής της συνάρτησης φαίνεται στο σχήμα 3.16.

2.3.3 Υπόλοιπες συναρτήσεις

- InitWorkers και InitMTasks

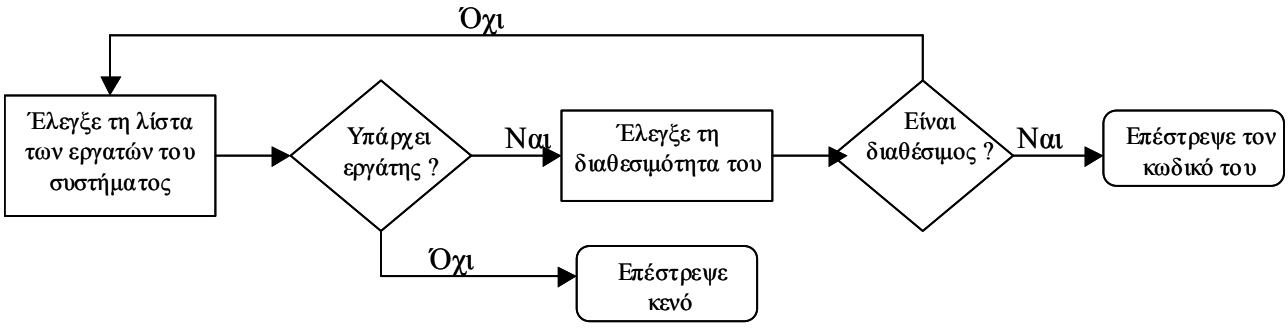
Οι δύο αυτές συναρτήσεις δημιουργούν τα αντικείμενα των «εργασιών στόχων» και των εργαζομένων. Ο αριθμός των αντικειμένων που θα δημιουργηθούν μπορεί να είναι εσωτερικά ορισμένος από το πρόγραμμα ή να δοθεί εξωτερικά από τον χρήστη του προγράμματος.



Σχήμα 3.17: Δημιουργία αντικειμένων από τις συναρτήσεις *InitWorkers* και *InitMTasks*

- AvailableWorker

Η συνάρτηση αυτή ελέγχει τη διαθεσιμότητα των εργαζομένων που υπάρχει στο σύστημα με βάση το γεγονός αν αυτοί έχουν αναλάβει την εκτέλεση ή όχι κάποιας εργασίας. Επιστρέφει τον πρώτο διαθέσιμο εργάτη.



Σχήμα 3.18: Διάγραμμα ροής της συνάρτησης *AvailableWorker*

- AssignTask

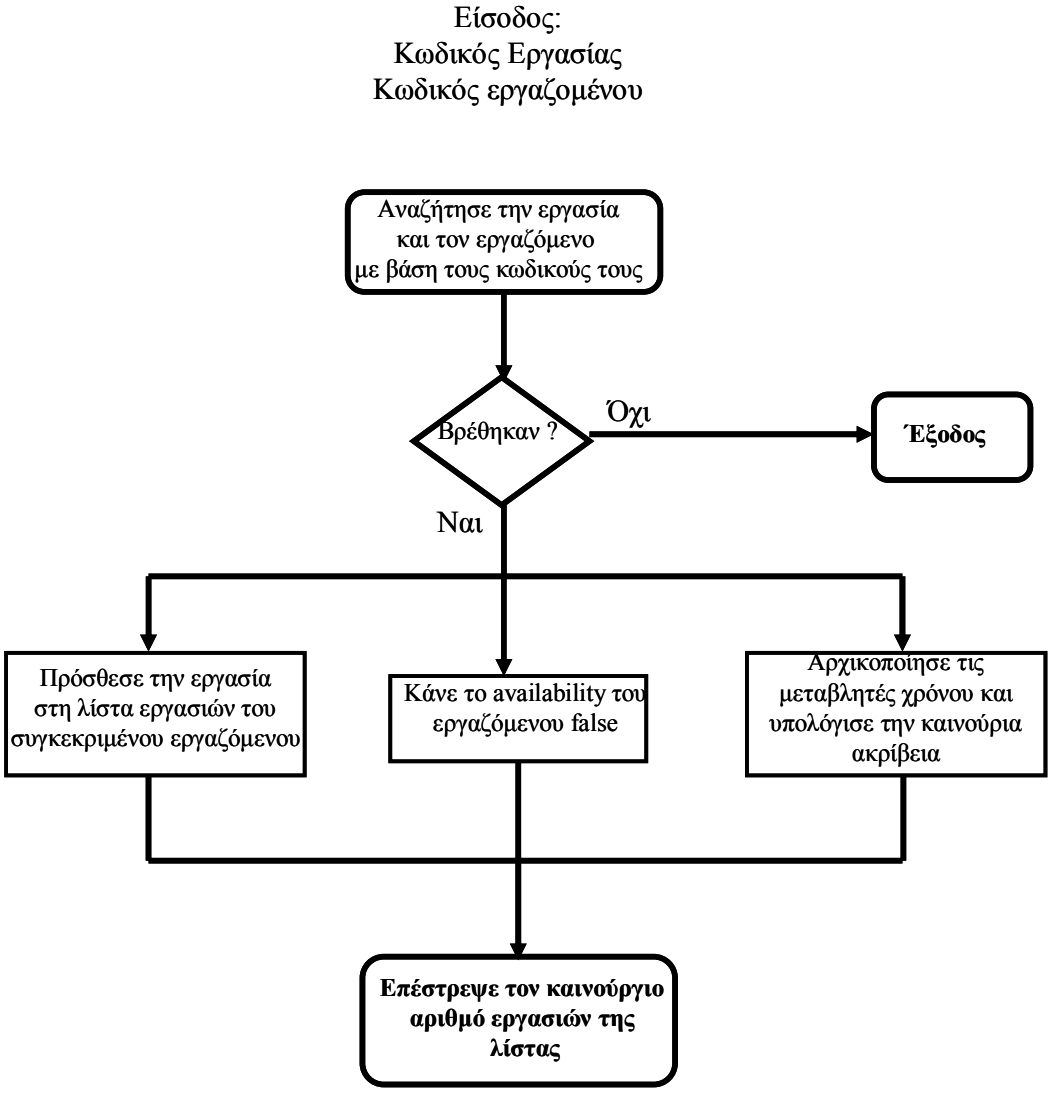
Η συνάρτηση αυτή αποτελεί μια από τις βασικές συναρτήσεις του προγράμματος. Με τη χρήση της συνάρτησης αυτής γίνεται η ανάθεση μίας εργασίας σε κάποιον εργαζόμενο. Η συνάρτηση αυτή έχει δύο ορίσματα σαν εισόδους. Τον κωδικό της εργασίας που θέλουμε να ανατεθεί και τον κωδικό του εργαζόμενου στον οποίο επιθυμούμε να ανατεθεί. Η συνάρτηση αυτή χρησιμοποιείται από τις υπόλοιπες συναρτήσεις δρομολόγησης εργασίας.

Η λειτουργία της συνάρτησης αυτής έχει ως εξής: Αρχικά ελέγχεται η λίστα των εργαζομένων για να επιβεβαιωθεί ότι ο κωδικός του εργαζόμενου που δόθηκε σαν είσοδος είναι σωστός και ότι ο εργαζόμενος αυτός βρίσκεται στην λίστα. Στη συνέχεια η ιδιότητα της διαθεσιμότητάς του λαμβάνει την τιμή *false* σε κάθε περίπτωση αφού είτε εργάζεται ήδη σε κάποια άλλη εργασία είτε όχι μία εργασία πρόκειται να του ανατεθεί. Ακολούθως ελέγχεται η λίστα με τις εργασίες του συγκεκριμένου εργαζόμενου για να βρεθεί η πρώτη κενή θέση σε αυτή.

Εκτός από τον έλεγχο που γίνεται για τον εργαζόμενο πραγματοποιείται έλεγχος και για στον κωδικό της εργασίας. Εφόσον ο κωδικός αυτός είναι σωστός και η εργασία υφίσταται λαμβάνεται από τις ιδιότητες της η κατηγορία στην οποία ανήκει. Η ανάθεση στον εργαζόμενο γίνεται μέσω προσθήκης του κωδικού της εργασίας στην λίστα εργασιών που πραγματοποιεί ο συγκεκριμένος εργαζόμενος Από εκεί και πέρα κάποιες παράμετροι όπως η χρονική στιγμή εκκίνησης της εργασίας λαμβάνουν τιμή με βάση το ρολόι του συστήματος. Εν συνεχεία γίνεται υπολογισμός του φόρτου εργασίας του εργαζομένου μετά την ανάθεση της καινούριας εργασίας. Ο

υπολογισμός του φόρτου γίνεται μέσω της συνάρτηση *CntWorkLoad* η οποία θα αναλυθεί διεξοδικά στη συνέχεια.

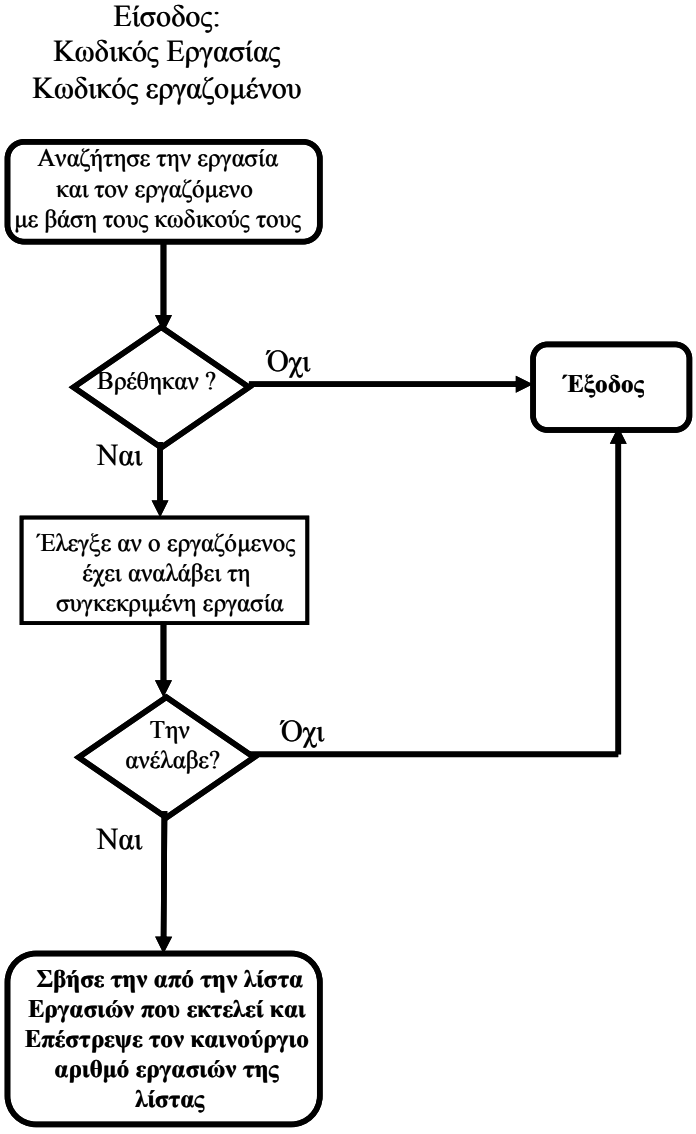
Με τον έλεγχο που γίνεται στην κατηγορία της εργασίας πραγματοποιείται και μία τυπική μείωση στην ακρίβεια του συγκεκριμένου εργαζόμενου καθώς αυτή μπορεί να μειωθεί σε κατηγορίες εργασίας όμοιες με αυτή της εργασίας που έχει αναλάβει. Η μείωση αυτή γίνεται διαισθητικά κατά κάποιο ποσοστό. Τελικώς η συνάρτηση αυτή επιστρέφει έναν αριθμό που αντιστοιχεί στον αριθμό των εργασιών που συνολικά έχει αναλάβει και εκτελεί ο συγκεκριμένος εργαζόμενος. Το διάγραμμα ροής της συνάρτησης αυτής είναι το ακόλουθο:



Σχήμα 3.19: Διάγραμμα ροής της συνάρτησης *AssignTask*

- DeAssignTask

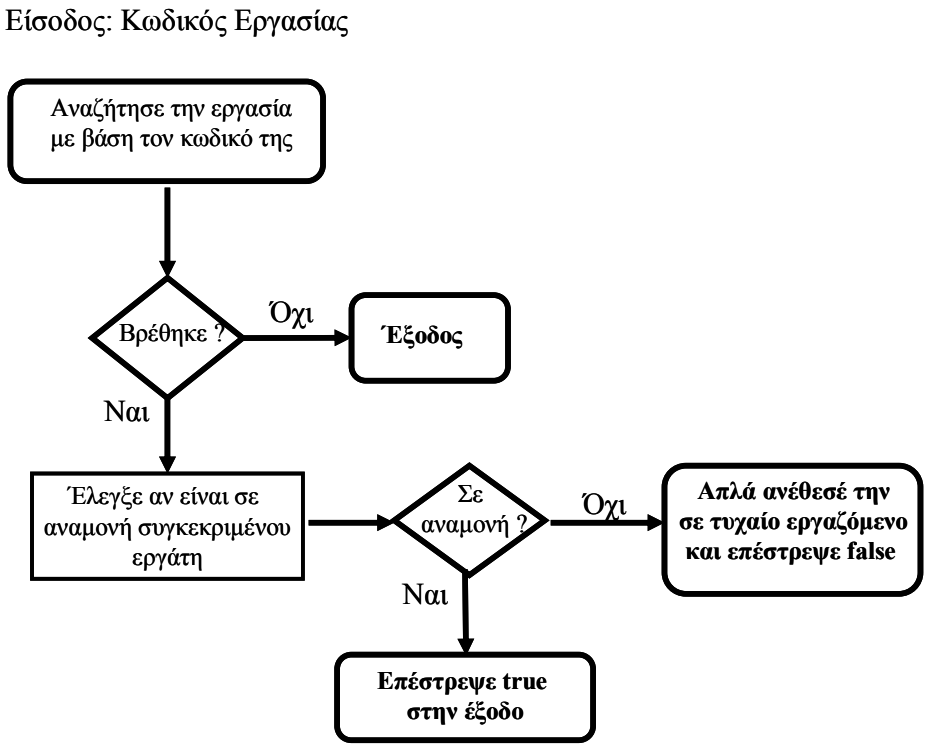
Η συνάρτηση αυτή αποαναθέτει κάποια εργασία από έναν συγκεκριμένο εργαζόμενο. Τα ορίσματα εισόδου της είναι αντίστοιχα με αυτά της *AssignTask* και οι έλεγχοι που πραγματοποιούνται στο αρχικό της στάδιο είναι επίσης ίδιοι. Η διακοπή της εργασίας από τον συγκεκριμένο εργαζόμενο γίνεται μέσω διαγραφής του κωδικού της από τη λίστα των εργασιών που αυτός πραγματοποιεί. Στη συνέχεια υπολογίζεται ο καινούργιος φόρτος του εργαζόμενου χωρίς την εργασία αυτή και η συνάρτηση επιστρέφει τον αριθμό των εργασιών τις οποίες ο συγκεκριμένος εργαζόμενος έχει πλέον στη λίστα του. Το διάγραμμα ροής της συνάρτησης αυτής είναι το ακόλουθο:



Σχήμα 3.20: Διάγραμμα ροής της συνάρτησης *DeAssignTask*

- IfTaskWaits

Η συνάρτηση αυτή είναι βοηθητικό κομμάτι των συναρτήσεων *AssignIfTaskWaits* και *SendTaskBack*. Παίρνει σαν όρισμα εισόδου τον κωδικό της εργασίας που θα ελεγχθεί αν βρίσκεται σε αναμονή και αρχικά γίνεται έλεγχος της ύπαρξης του συγκεκριμένου κωδικού. Στη συνέχεια ελέγχεται αν η εργασία βρίσκεται σε αναμονή κάποιου συγκεκριμένου εργαζόμενου. Αν αυτό δεν ισχύει τότε ανατίθεται σε κάποιον τυχαίο εργαζόμενο που μπορεί να την πραγματοποιήσει και επιστρέφει την τιμή *false*. Σε αντίθετη περίπτωση επιστρέφει *true* (που σημαίνει ότι η εργασία βρίσκεται σε αναμονή) και η τιμή αυτή χρησιμοποιείται από τις δύο συναρτήσεις δρομολόγησης. Το διάγραμμα ροής της είναι το ακόλουθο:



Σχήμα 3.21: Διάγραμμα ροής της συνάρτησης *IfTaskWaits*

- CntWorkLoad

Η συνάρτηση αυτή υπολογίζει ανά πάσα στιγμή τον φόρτο εργασίας του συγκεκριμένου εργαζόμενου του οποίου τον κωδικό παίρνει σαν όρισμα. Στον υπολογισμό του φόρτου λαμβάνεται υπόψιν ο διαχωρισμός που έχει γίνει στα

πληροφοριακά κανάλια. Η σχέση με βάση την οποία υπολογίζεται ο φόρτος είναι η ακόλουθη:

$$W_T = \sum_{t=1}^{n}(L_t) + \sum_{t=1}^{n-1}\sum_{s=t+1}^{n} w_i(a_{t,i} + a_{s,i})$$

όπου:

W_T : Ο φόρτος τη χρονική στιγμή T

i= 1…m είναι οι πηγές του φόρτου (perception, motor, decision)

t,s= 1…n οι εργασίες του εργάτη

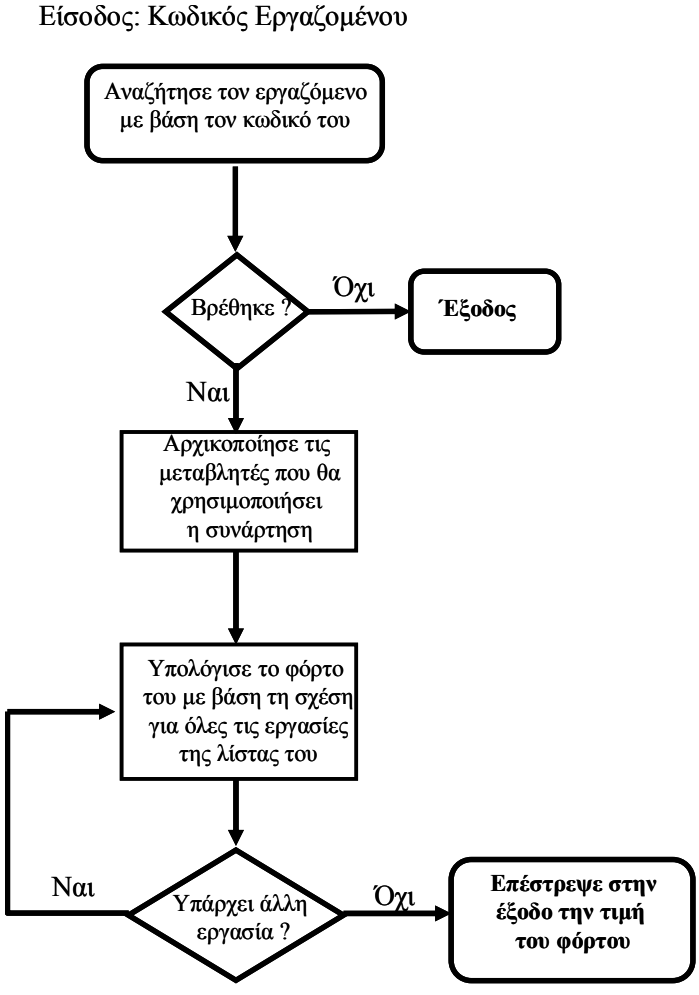
L_t : Ο δεδομένος φόρτος για την εργασία t

$a_{t,i}$: Ο φόρτος στο συγκεκριμένο κανάλι i για το task t

$a_{s,i}$: Ο φόρτος στο συγκεκριμένο κανάλι i για το task s

w_i : το βάρος σύγκρουσης μεταξύ των εργασιών που εκτελούνται ανά κανάλι

Το διάγραμμα ροής για τον υπολογισμό του φόρτου είναι το ακόλουθο:



Σχήμα 3.22: Διάγραμμα ροής της συνάρτησης *CntWorkLoad*

Κεφάλαιο 4

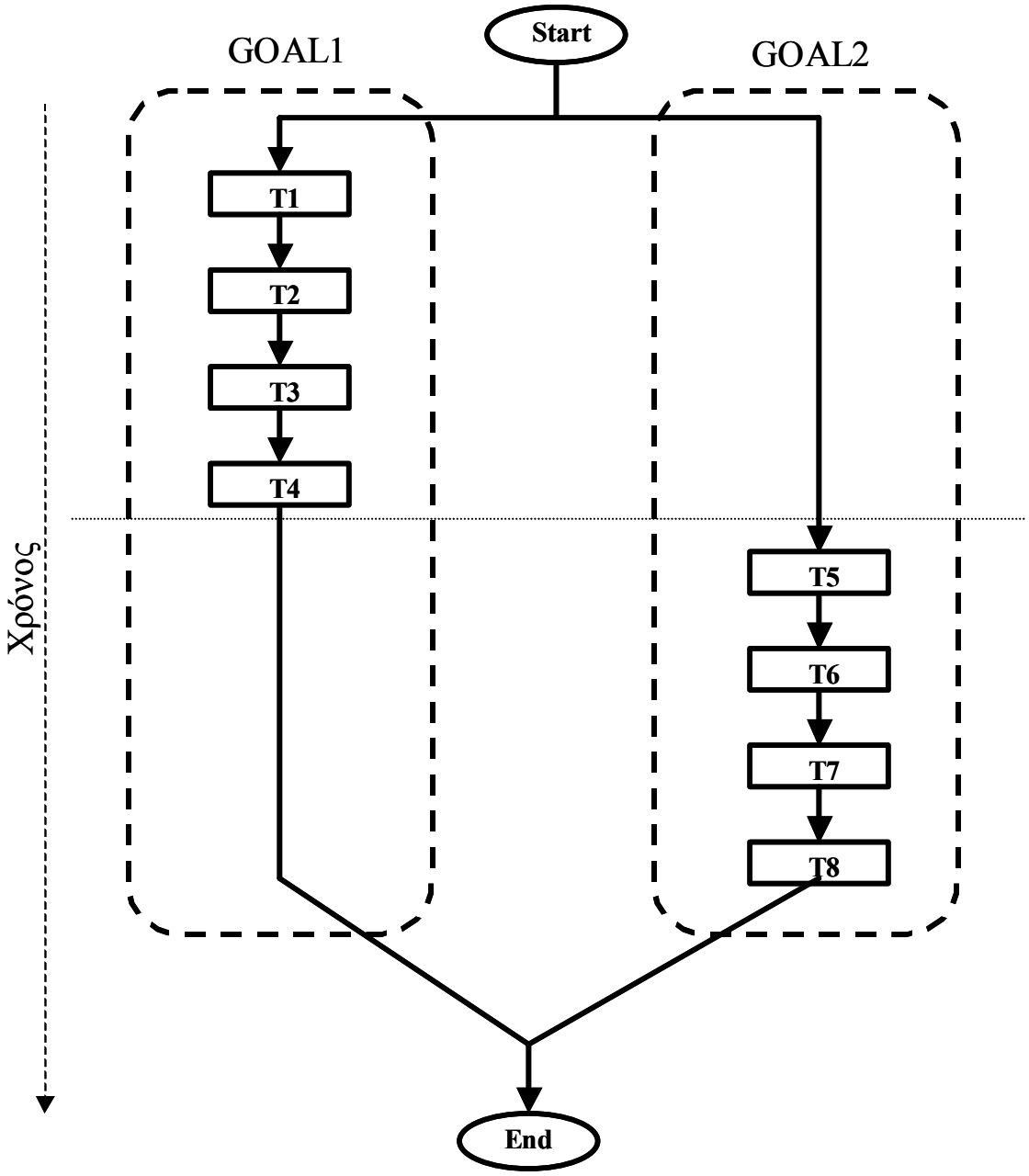
Παραδείγματα εργασίας - (Case Studies)

Στο κεφάλαιο αυτό μελετάται η εφαρμογή του μοντέλου σε διάφορες περιπτώσεις σχημάτων εργασίας. Συγκεκριμένα αναλύονται και προσομοιώνονται τρία πιθανά συστήματα εργασίας. Ένα σειριακό σύστημα όπου για να ξεκινήσει μία εργασία πρέπει να έχει ολοκληρωθεί η προηγούμενη, ένα παράλληλο – σειριακό σύστημα με κάποιες εργασίες να εκτελούνται παράλληλα και κάποιες σειριακά και ένα παράλληλο σύστημα όπου κατ΄ ουσία οι εργασίες εκτελούνται παράλληλα.

1. Σειριακό σύστημα

1.2 Περιγραφή συστήματος

Στο σύστημα αυτό οι εργασίες εκτελούνται σειριακά. Μόλις ολοκληρωθεί κάποια εργασία τότε είναι το σύστημα σε θέση να αποδώσει σε κάποιον εργαζόμενο την επόμενη εργασία. Θεωρούμε ότι η ολοκλήρωση του στόχου απαιτεί την ολοκλήρωση δύο υποστόχων *Goal1* και *Goal2* (Σχήμα 4.1). Ο κάθε υποστόχος αποτελείται από 4 εργασίες οι οποίες πρέπει να ολοκληρωθούν για να ολοκληρωθεί και ο υποστόχος. Οι εργασίες αυτές εκτελούνται σειριακά με απόλυτο τρόπο. Αυτό σημαίνει ότι για να ξεκινήσει κάποια εργασία θα πρέπει να έχει ολοκληρωθεί η προηγούμενή της. Ο πρώτος υποστόχος περιλαμβάνει τις εργασίες *T1*, *T2*, *T3*, *T4* ενώ ο δεύτερος τις *T5*, *T6*, *T7*, *T8*. Λόγω του σειριακού τρόπου εκτέλεσης υπάρχει και ο επιπλέον περιορισμός ότι για να ξεκινήσει ο δεύτερο υποστόχος (*Goal2*) θα πρέπει απαραιτήτως να ολοκληρωθεί ο πρώτος (*Goal1*) δηλαδή να έχει ολοκληρωθεί και η εργασία Τ4.



Σχήμα 4.1: Το σειριακό μοντέλο εργασιών

Οι εργασίες κατηγοριοποιούνται με βάση το κανάλι πληροφορίας το οποίο επιβαρύνουν περισσότερο στις τρεις κατηγορίες που αναφέρθηκαν στον προηγούμενο κεφάλαιο (perception, motor, decision). Για τις ανάγκες τις προσομοίωσης οι αντιστοίχηση στις εργασίες του συστήματος γίνεται με βάση τον επόμενο πίνακα:

Πίνακας 1				
Goal1			***Goal2***	
T1	Motor		**T5**	Motor
T2	Perception		**T6**	Perception
T3	Motor		**T7**	Motor
T4	Decision		**T8**	Decision

Στο σύστημα θεωρούμαι την ύπαρξη δύο εργαζομένων *W1* και *W2* με συγκεκριμένες ιδιότητες. Ο ένας εργαζόμενος είναι ιδιαίτερα εξειδικευμένος σε συγκεκριμένες κατηγορίες εργασιών. Αυτό σημαίνει ότι εμφανίζει πολύ υψηλή ακρίβεια στην πραγματοποίηση συγκεκριμένων εργασιών αλλά πραγματοποιεί με μέτρια ακρίβεια τις υπόλοιπες. Ο άλλος μπορεί να πραγματοποιήσει με σχετικά καλή ακρίβεια όλες τις εργασίες αλλά δεν εμφανίζει σε καμία περίπτωση την εξειδίκευση του πρώτου. Οι δύο εργαζόμενοι ακολουθούν ως προς την εξειδίκευση τους τον παρακάτω πίνακα:

Πίνακας 2				
W1			***W2***	
Κατηγορία Εργασίας	**Ακρίβεια**		**Κατηγορία Εργασίας**	**Ακρίβεια**
Motor	>90%		Motor	60% - 75%
Perception	50% - 60%		Perception	60% - 75%
Decision	50% - 60%		Decision	60% - 75%

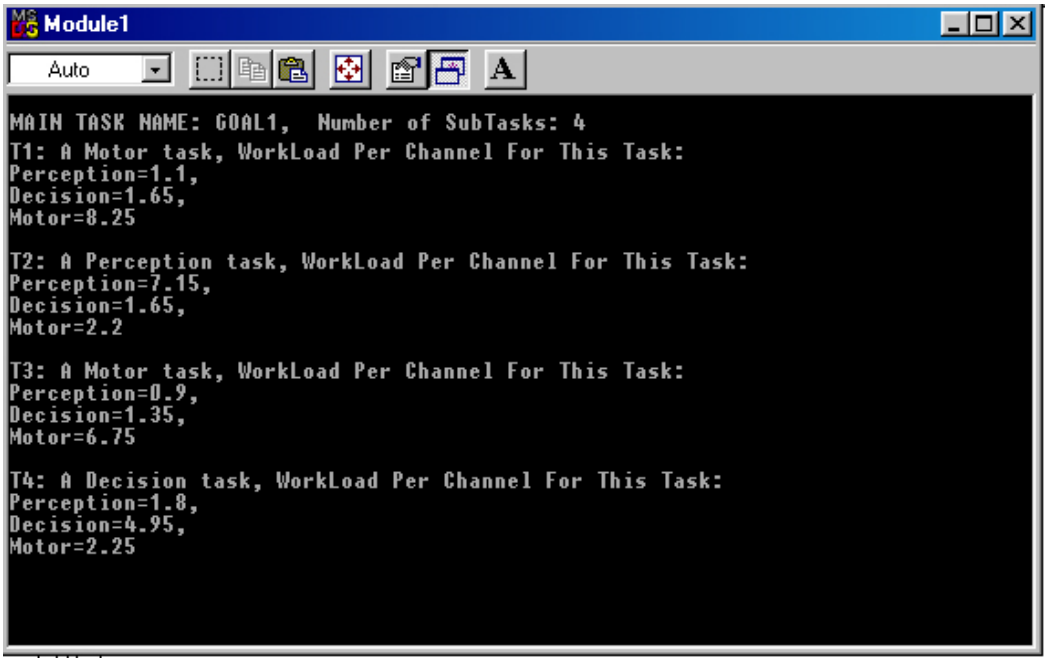
Λόγω της υψηλής ακρίβειας που εμφανίζει ο εργαζόμενος *W1* σε κατηγορίες εργασιών που αφορούν σε κίνηση οποιαδήποτε περίπτωση τέτοιας εργασίας ανατίθεται σε αυτών. Στο σύστημα μας ο εργαζόμενος *W1* θα αναλάβει δηλαδή τις εργασίες *T1*, *T3*, *T5* και *T7* ενώ για τις υπόλοιπες επιλέγεται ο εργαζόμενος *W2* καθώς αυτός θα εμφανίσει ούτως ή άλλως υψηλότερη ακρίβεια.

1.2 Προσομοίωση του σειριακού συστήματος

Το σύστημα αυτό προσομοιώθηκε με βάση μοντέλο που αναφέρθηκε στο προηγούμενο κεφάλαιο. Στις εικόνες που ακολουθούν φαίνονται τα 4 στάδια της εξόδου που μας έδωσε το πρόγραμμα. Τα στάδια αυτά είναι τα ακόλουθα:

- Αρχικοποίηση πρώτου υποστόχου (*Goal1*)

Κατά την εκκίνηση του προγράμματος αρχικοποιούνται οι δύο υποστόχοι για να ακολουθήσει η περαιτέρω εκτέλεση.

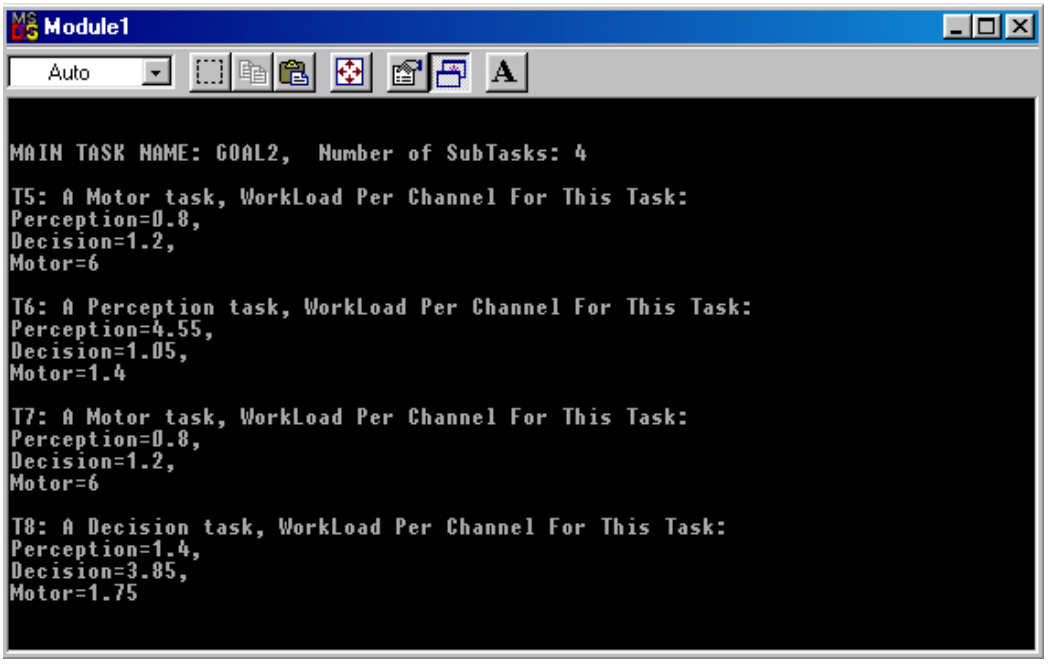


Σχήμα 4.2: Αρχικοποίηση *GOAL1*

Στο σχήμα αυτό φαίνεται ο τρόπος με τον οποίο γίνεται η αρχικοποίηση του πρώτου υποστόχου. Ορίζεται ο αριθμός των εργασιών που περιλαμβάνει σε τέσσερις και από εκεί και πέρα ακολουθεί η αρχικοποίηση των παραμέτρων των εργασιών που αφορούν στο φόρτο εργασίας. Ανάλογα με την κατηγορία της εργασίας δίδεται και μία τιμή για το κάθε κανάλι πληροφορίας. Οι τιμές που αποδίδονται χρησιμοποιούνται για τον υπολογισμό του φόρτου του εργαζόμενου που αναλαμβάνει τη συγκεκριμένη εργασία.

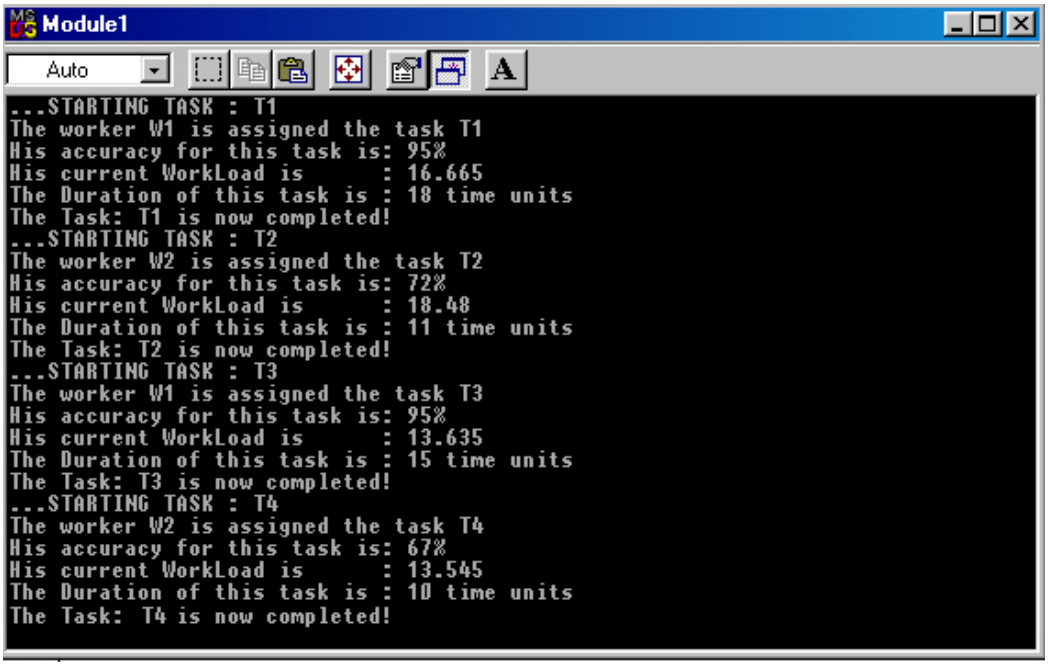
- Αρχικοποίηση δεύτερου υποστόχου (*Goal2*)

Σε αντιστοιχία με την προηγούμενη περίπτωση και με ακριβώς ίδιο τρόπο αρχικοποιείται και ο δεύτερος υποστόχος και η αρχικοποίηση αυτή φαίνεται στο σχήμα 4.3



Σχήμα 4.3: Αρχικοποίηση *GOAL2*

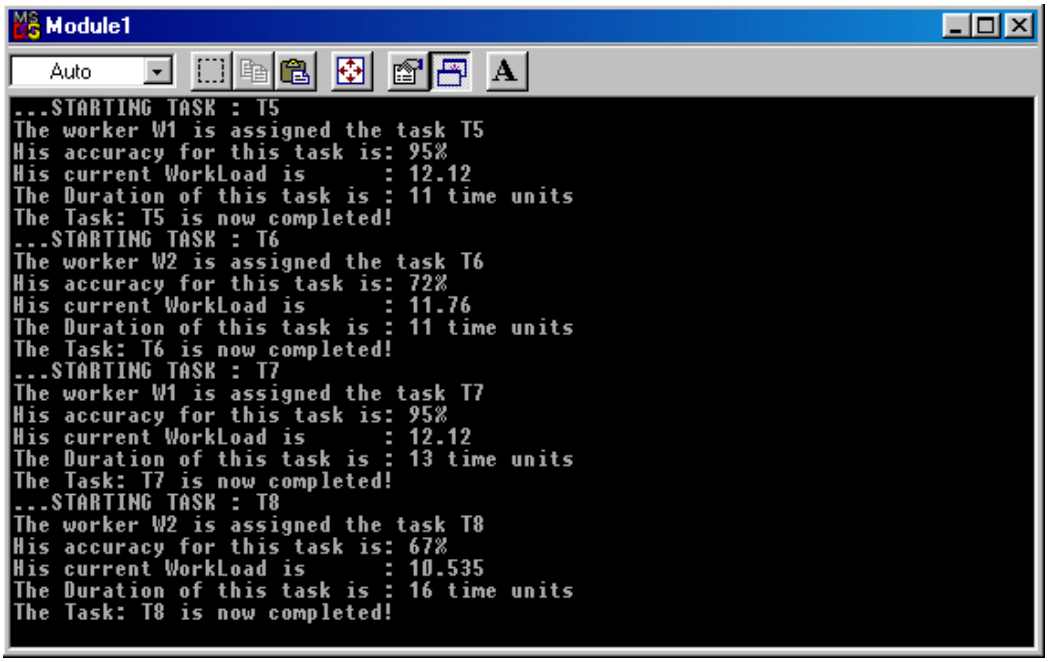
- Υλοποίηση πρώτου υποστόχου (*Goal1*)



Σχήμα 4.4: Υλοποίηση *GOAL1*

Στο σχήμα 4.4 φαίνεται ο τρόπος με τον οποίο λειτουργεί το σύστημα. Ξεκινά αναθέτοντας την εργασία *T1* στον εργαζόμενο *W1*, αφού όπως είπαμε αυτός έχει εξαιρετική ακρίβεια σε εργασίες αυτής της κατηγορίας. Το πρόγραμμα μας παρέχει πληροφορίες για την ακρίβεια του συγκεκριμένου εργαζόμενου και για το φόρτο που η εργασία *T1* του επιφέρει. Εμφανίζεται επίσης και ο τυπικός χρόνος ολοκλήρωσης της εργασίας, δηλαδή η διάρκειά της, καθώς και η χρονική στιγμή στην οποία βρίσκεται το σύστημα. Μόλις ολοκληρώνεται κάποια εργασία, λόγω της σειριακής εκτέλεσης τους, η επόμενη ανατίθεται στον κατάλληλο εργαζόμενο. Έτσι στο σχήμα 4.4 βλέπουμε ότι η εργασία *T2* ανατίθεται στον εργαζόμενο *W2* την χρονική στιγμή 26 (μόλις δηλαδή ολοκληρώθηκε η *T1*) και επιλέγεται να ανατεθεί στον συγκεκριμένο εργαζόμενο καθώς έχει μεγαλύτερη ακρίβεια σε αυτή. Με αντίστοιχο ανατίθενται και οι υπόλοιπες εργασίες μέχρι την ολοκλήρωση του πρώτου υποστόχου που συμβαίνει με την ολοκλήρωση της εργασίας *T4*.

- Υλοποίηση δεύτερου υποστόχου (*Goal2*)

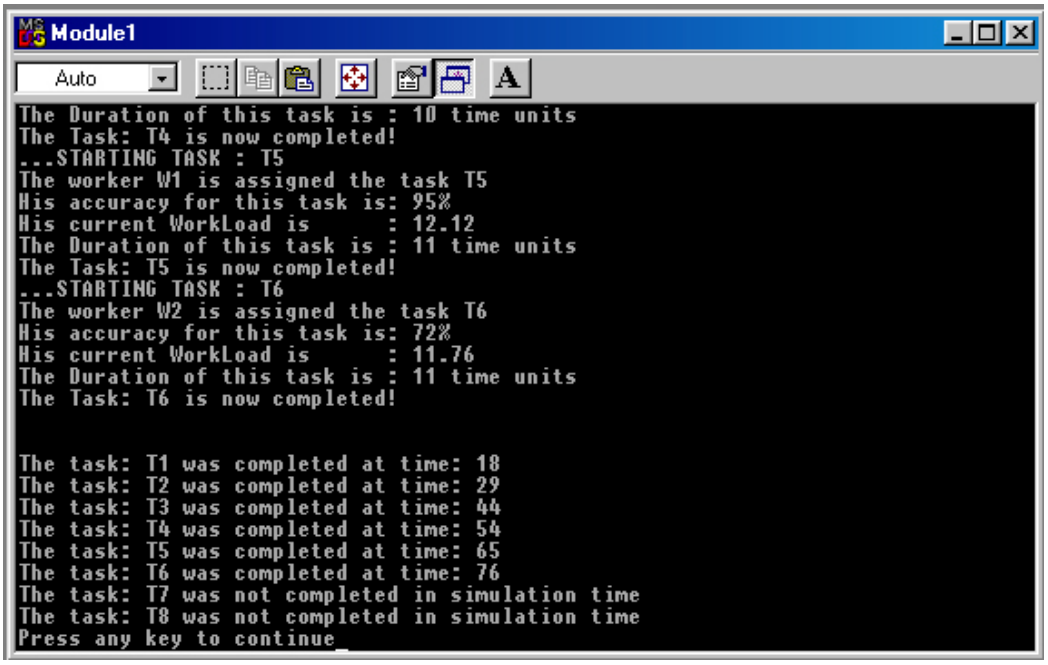


Σχήμα 4.5: Υλοποίηση *GOAL2*

Η υλοποίηση του δεύτερου υποστόχου ξεκινά αμέσως μετά την ολοκλήρωση του πρώτου και γίνεται με ακριβώς τον ίδιο τρόπο. Όταν ολοκληρωθεί η εργασία *T8* ουσιαστικά ολοκληρώνεται η προσομοίωση του συστήματος.

1.4 Αποτελέσματα προσομοίωσης

Στην ουσία το σύστημα αυτό εφαρμόζει υπό συνθήκη ανάθεση εργασιών αφού μόνο εφόσον ολοκληρωθεί μία εργασία ή ένας ολόκληρος υποστόχος μπορεί να ξεκινήσει μία άλλη εργασία ή ένας επόμενος υποστόχος. Η υπό συνθήκη αυτή ανάθεση αποτελεί βάση και για τα υπόλοιπα συστήματα ανάθεσης εργασιών που εξετάζονται στη συνέχεια. Ο χρόνος προσομοίωσης ορίσθηκε αρχικά σε 120 χρονικές μονάδες για την πλήρη ολοκλήρωση της. Σε μία πιθανή μείωση του χρόνου αυτού για παράδειγμα σε 80 χρονικές μονάδες η έξοδος που θα μας έδινε το σύστημα θα ήταν η ακόλουθη:



Σχήμα 4.6: Ολοκλήρωση προγράμματος με χρόνο προσομοίωσης 80 χρονικές μονάδες

Στο σχήμα αυτό βλέπουμε ότι λόγω της σειριακής εκτέλεσης των εργασιών δεν είναι δυνατή η ολοκλήρωση του έργου στον δεδομένο χρόνο. Έτσι μέσα στο χρόνο προσομοίωσης δεν πρόλαβαν να ολοκληρωθούν οι εργασίες *T7* και *T8*. Αυτό είναι και το μεγάλο μειονέκτημα της συγκεκριμένης εργασιακής δομής. Το γεγονός δηλαδή ότι απαιτεί πολύ περισσότερο χρόνο η ολοκλήρωσή της. Στα θετικά της όμως συμπεριλαμβάνεται το γεγονός του μικρού φόρτου εργασίας και της υψηλής ακρίβειας με την οποία εκτελούνται οι εργασίες, αφού λόγω της σειριακής δομής που έχει το σύστημα και της μη παράλληλης εκτέλεσης εργασιών από τους εργαζόμενους, τόσο ο φόρτος εργασίας τους όσο και η ακρίβεια τους δεν μπορεί να μεταβάλλεται.

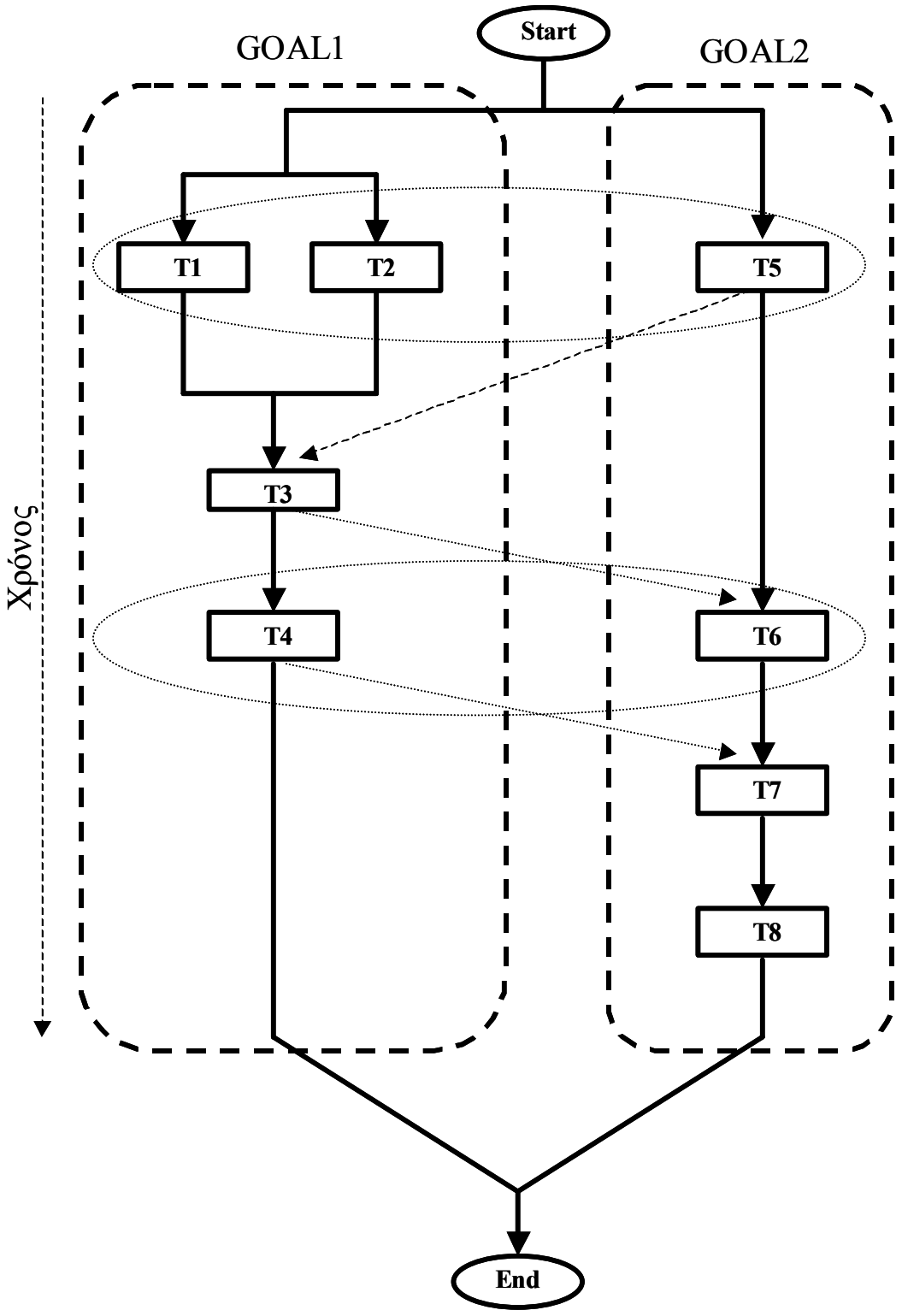
.2. Παράλληλο - Σειριακό σύστημα

2.1 Περιγραφή συστήματος

Στο σύστημα αυτό οι εργασίες εκτελούνται σειριακά και παράλληλα. Μόλις ολοκληρωθεί κάποια εργασία που εκτελείται από μόνη της ή ένας κύκλος εργασιών που πρέπει να εκτελεστεί παράλληλα τότε το σύστημα είναι σε θέση να αποδώσει σε κάποιον εργαζόμενο την επόμενη εργασία ή να αποδώσει σε παραπάνω από έναν εργαζόμενους εργασίες, όταν αυτές πρέπει να εκτελεστούν παράλληλα ή σε έναν εργαζόμενο παραπάνω από μία εργασίες αν απαιτείται αυτές να εκτελεστούν παράλληλα. Θεωρούμε και πάλι ότι η ολοκλήρωση του στόχου απαιτεί την ολοκλήρωση δύο υποστόχων *Goal1* και *Goal2* (Σχήμα 4.7). Ο κάθε υποστόχος αποτελείται από 4 εργασίες (για να υπάρχει σύγκριση με το προηγούμενο σύστημα) οι οποίες πρέπει να ολοκληρωθούν για να ολοκληρωθεί και ο υποστόχος. Και πάλι ο πρώτος υποστόχος *Goal1* περιλαμβάνει τις εργασίες *T1*, *T2*, *T3*, *T4* ενώ ο δεύτερος τις *T5*, *T6*, *T7*, *T8*.

Αυτό που συμβαίνει στο συγκεκριμένο σχήμα εργασιών είναι ότι εργασίες εξαρτώνται από την ολοκλήρωση εργασιών τόσο του υποστόχου στον οποίο ανήκουν όσο και του άλλου υποστόχου. Με τον τρόπο αυτό αυξάνουμε την γενικότητα του συστήματος. Η ανάθεση των εργασιών και πάλι βασίζεται στην εξειδίκευση των εργαζομένων οπότε η εκάστοτε εργασία ανατίθεται στον εργαζόμενο που θα την πραγματοποιήσει με τη μεγαλύτερη ακρίβεια.

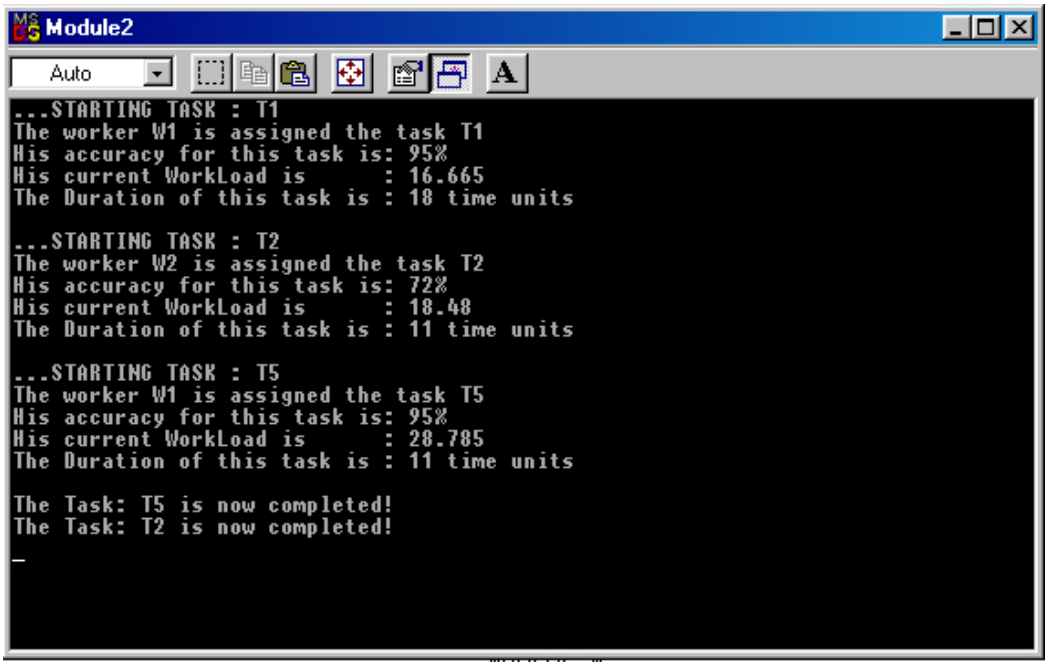
Όπως φαίνεται και στο σχήμα 4.7 η λειτουργία του συστήματος ξεκινάει με την ανάθεση τριών εργασιών που πρέπει να εκτελεστούν παράλληλα (*T1*, *T2* και *T5*). Οι εργασίες αυτές ανατίθενται στους εργαζόμενους *W1* και *W2* με βάση την ακρίβεια τους. Για να ξεκινήσει η εργασία *T3* που ακολουθεί θα πρέπει η ομάδα των τριών αυτών εργασιών να ολοκληρωθεί. Εδώ φαίνεται ότι η *T3* δεν εξαρτάται μόνο από τις *T1*, *T2* που ανήκουν στον ίδιο υποστόχο (*Goal1*) αλλά και από την *T5* που ανήκει στον *Goal2*. Όταν ολοκληρωθεί η εργασία *T3* το σύστημα είναι σε θέση να αναθέσει τις *T4* και *T6* που πρέπει να εκτελεστούν παράλληλα (και εδώ η *T6* εξαρτάται από εργασία άλλου υποστόχου. Ακολούθως εκτελούνται σειριακά οι *T7* και *T8*. Με την ολοκλήρωση της *T8* ολοκληρώνεται και η λειτουργία του συστήματος καθώς έχουν ολοκληρωθεί και οι δύο υποστόχοι.



Σχήμα 4.7: Το παράλληλο - σειριακό μοντέλο εργασιών

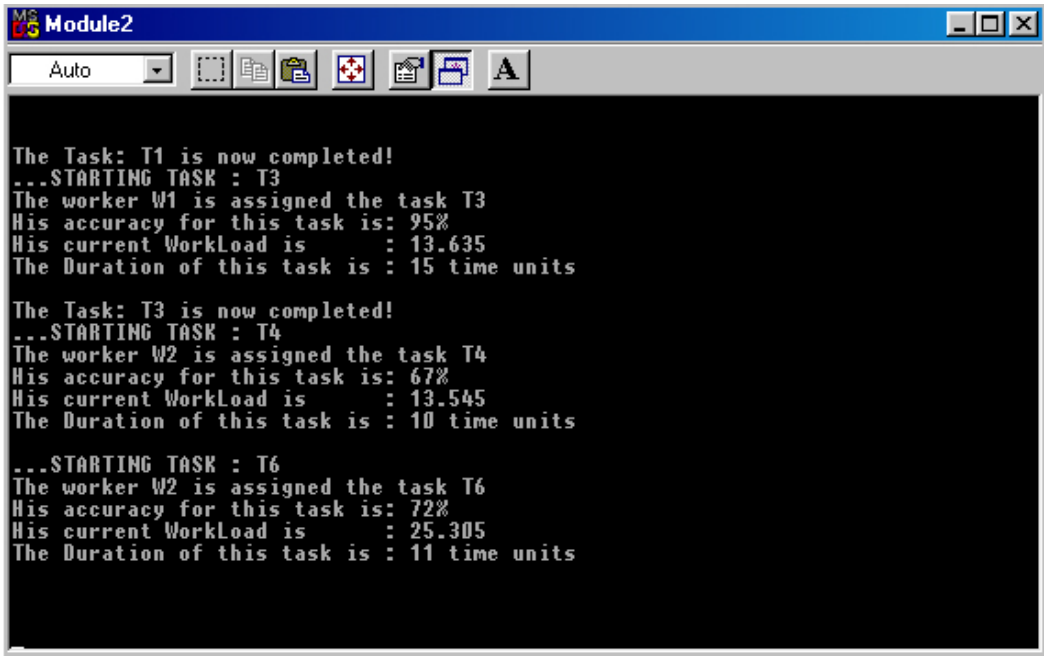
2.2 Προσομοίωση του σειριακού - παράλληλου συστήματος

Η αρχικοποίηση των στόχων είναι ίδια με αυτή του σειριακού συστήματος. Από εκεί και πέρα κατά η εκτέλεση του προγράμματος πραγματοποιείται με βάση την δομή των εργασιών του συγκεκριμένου συστήματος. Έτσι μετά την αρχικοποίηση έχουμε την ακόλουθη διαδοχή γεγονότων:



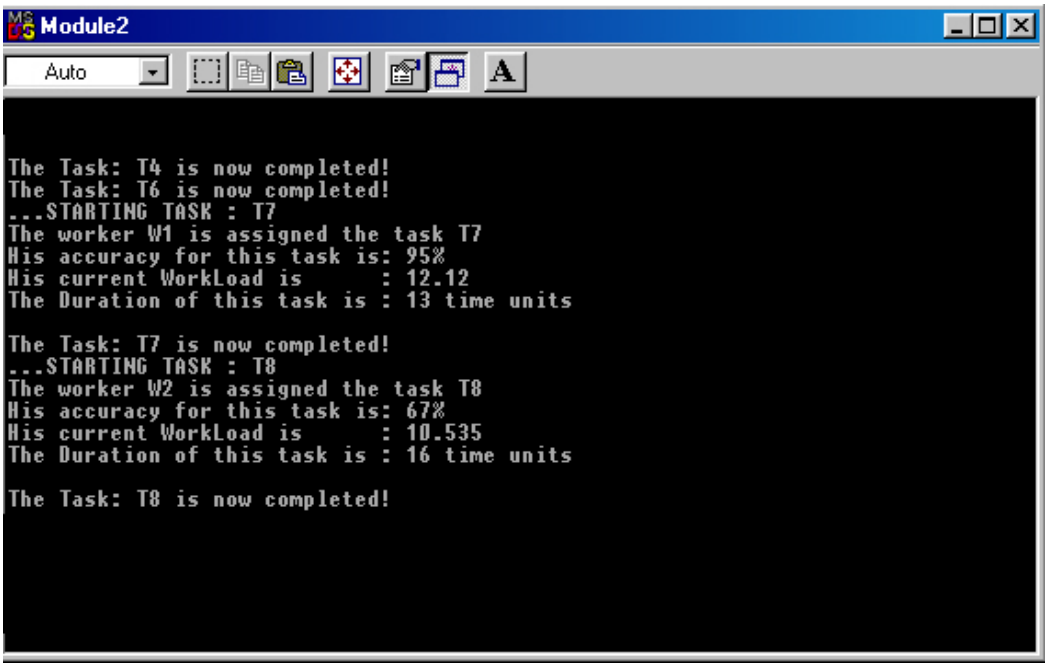
Σχήμα 4.8: Ανάθεση των εργασιών T1,T2, T5 και ολοκλήρωση των T2, T5

Στο σχήμα 4.8 φαίνεται η εκκίνηση του προγράμματος όπου ταυτόχρονα ανατίθενται οι 3 εργασίες (*T1*, *T2* και *T5*) στους εργαζόμενους *W1* και *W2*, με τρόπο ώστε να ανατίθεται πάντα η κάθε εργασία σε αυτόν που θα την πραγματοποιήσει με την μεγαλύτερη ακρίβεια. Με βάση την διάρκεια των εργασιών, όπως φαίνεται από το σχήμα, ολοκληρώνονται πρώτα οι *T2* και *T5*, 11 χρονικές στιγμές μετά την έναρξη της προσομοίωσης. Λόγω της δομής του συγκεκριμένου συστήματος η επόμενη προς ανάθεση εργασία δεν ξεκινάει αν δεν ολοκληρωθεί πρώτα και η εργασία *T1*. Με την ολοκλήρωση της (σχήμα 4.9) ανατίθεται η εργασία *T3* και μόλις ολοκληρωθεί και αυτή έχουμε ταυτόχρονη ανάθεση των δύο επόμενων εργασιών (*T4* και *T6*) οι οποίες θα ολοκληρωθούν με 1 χρονική στιγμή διαφορά (όπως μπορούμε να δούμε στο σχήμα 4.9 από τη διάρκεια τους).



Σχήμα 4.9: Ολοκλήρωση της T1, ανάθεση και ολοκλήρωση της T3 και ανάθεση των T4 και T6

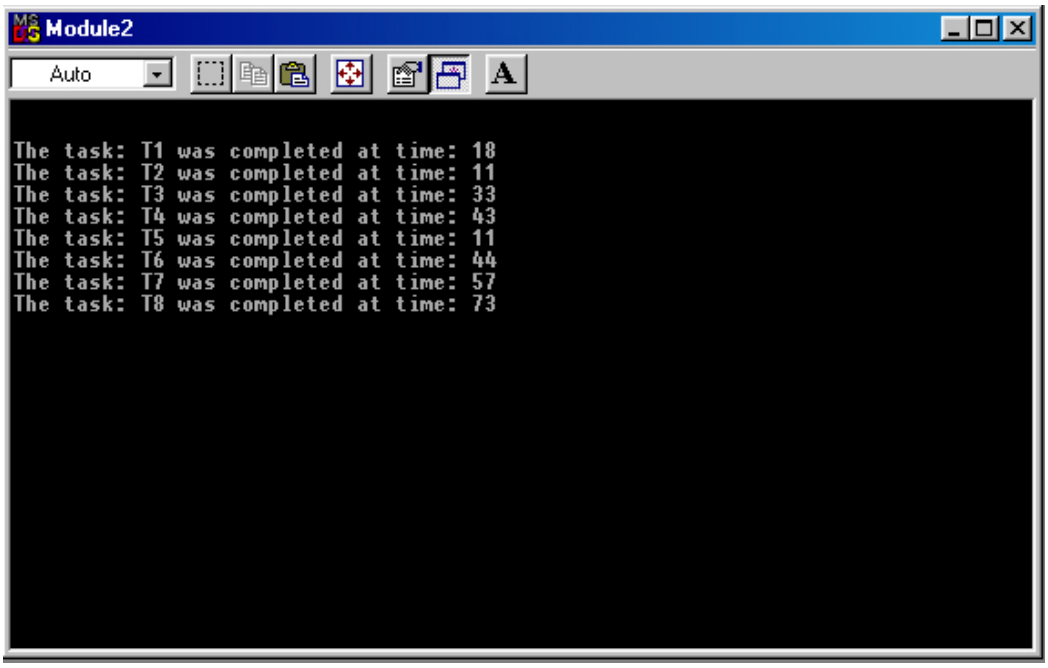
Μόλις ολοκληρωθούν οι *T4* και *T6* ανατίθεται η *T7*, και μόλις ολοκληρωθεί και αυτή η *T8* όπως φαίνεται στο σχήμα 4.10.



Σχήμα 4.10: Ολοκλήρωση των T4, T6 ανάθεση και ολοκλήρωση της T7, ανάθεση και ολοκλήρωση της T8

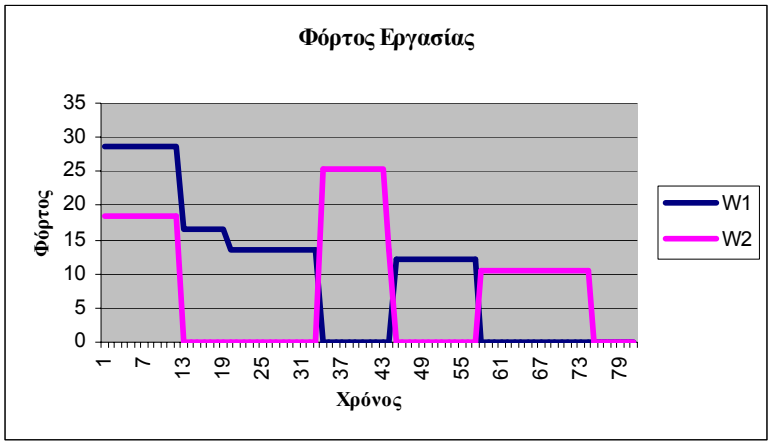
2.3 Αποτελέσματα προσομοίωσης

Το πρόγραμμα μας δείχνει τις χρονικές στιγμές ολοκλήρωσης όλων των εργασιών με χρόνο προσομοίωσης 80 χρονικές μονάδες (σχήμα 4.11):



Σχήμα 4.11: Προσομοίωση 80 χρονικών μονάδων για το παράλληλο σειριακό σύστημα

Το διάγραμμα για τον φόρτο εργασίας του κάθε εργαζόμενου κατά τη διάρκεια λειτουργίας του συστήματος είναι το ακόλουθο:

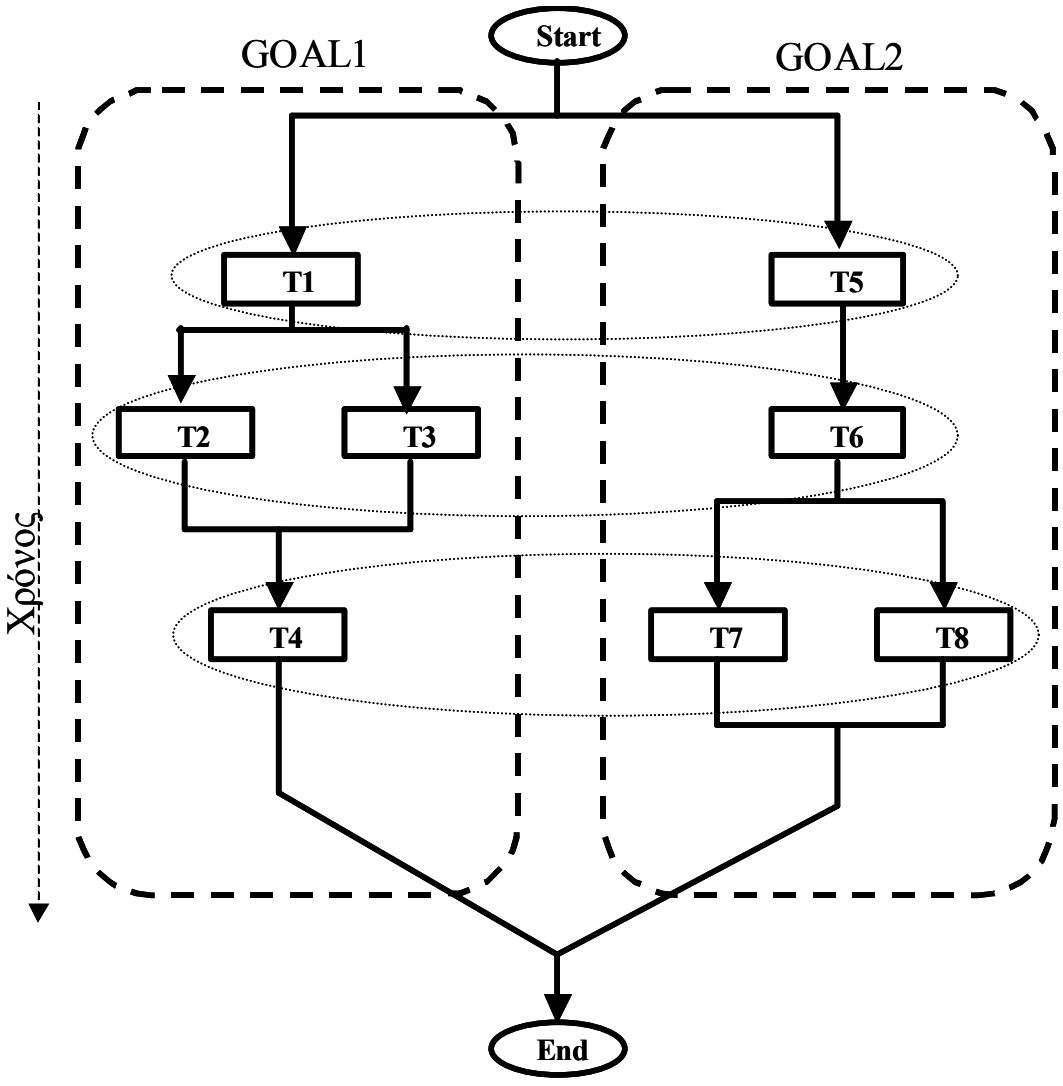


Σχήμα 4.12: Φόρτος εργαζομένων παράλληλου σειριακού συστήματος

3. Παράλληλο σύστημα

3.1 Περιγραφή συστήματος

Στο σύστημα αυτό οι εργασίες εκτελούνται κατά κύριο λόγο παράλληλα. Μόλις ολοκληρωθεί ένας κύκλος εργασιών που πρέπει να εκτελεστεί παράλληλα τότε το σύστημα είναι σε θέση να αποδώσει σε παραπάνω από έναν εργαζόμενους εργασίες, ή σε έναν εργαζόμενο παραπάνω από μία εργασίες. Θεωρούμε και πάλι την ύπαρξη των δύο υποστόχων *Goal1* και *Goal2* για την ολοκλήρωση του στόχου (Σχήμα 4.13). Κάθε υποστόχος αποτελείται πάλι από 4 εργασίες (για λόγους σύγκρισης) ο πρώτος από τις εργασίες *T1*, *T2*, *T3*, *T4* ενώ ο δεύτερος από τις *T5*, *T6*, *T7*, *T8*.

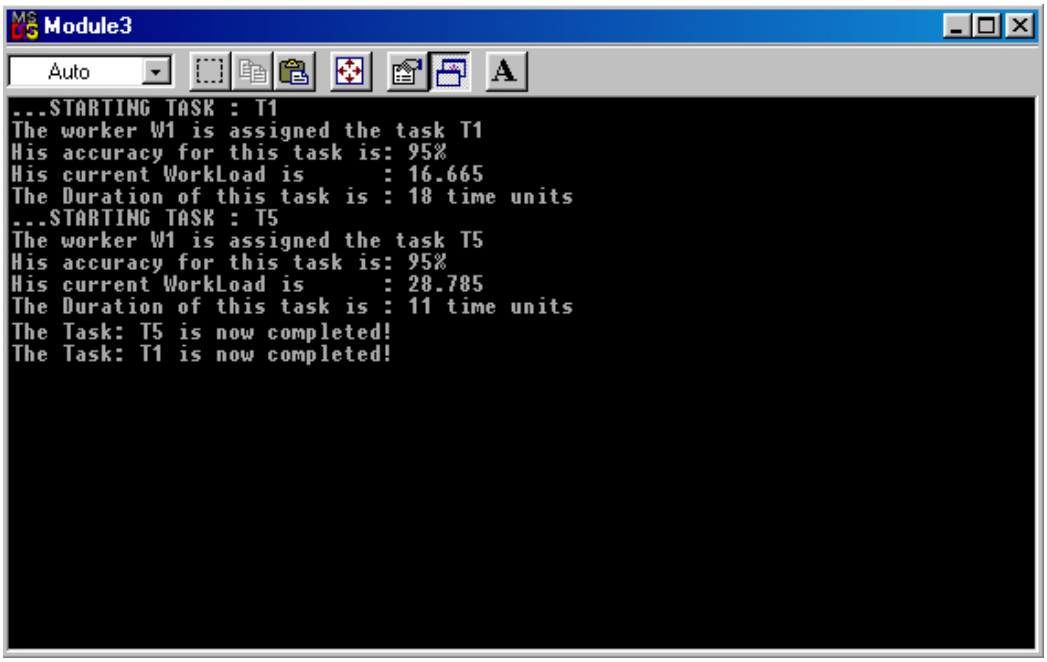


Σχήμα 4.13: Το παράλληλο μοντέλο εργασιών

Στο σχήμα 4.13 βλέπουμε ότι η λειτουργία του συστήματος ξεκινάει με την ανάθεση των εργασιών (*T1* και *T5*). Οι εργασίες αυτές ανατίθενται στον εργαζόμενο *W1* λόγω της ακρίβεια του. Μόλις ολοκληρωθούν οι δύο αυτές εργασίες ανατίθενται οι *T2*, *T3* και *T6*. Μετά την ολοκλήρωση τους ανατίθενται οι *T4*, *T7* και *T8* για να ολοκληρωθεί μετά από αυτές τη λειτουργία του συστήματος.

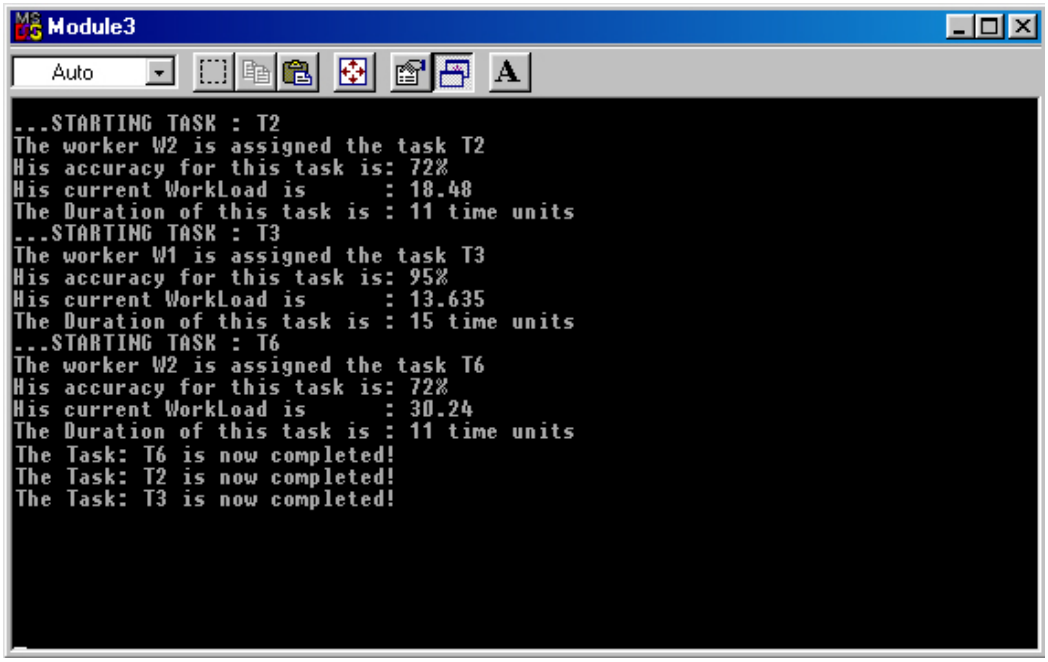
3.2 Προσομοίωση παράλληλου συστήματος

Η λειτουργική εξέλιξη του συγκεκριμένου συστήματος κατά την προσομοίωση φαίνεται στα σχήματα 4.14, 4.15, 4.16.

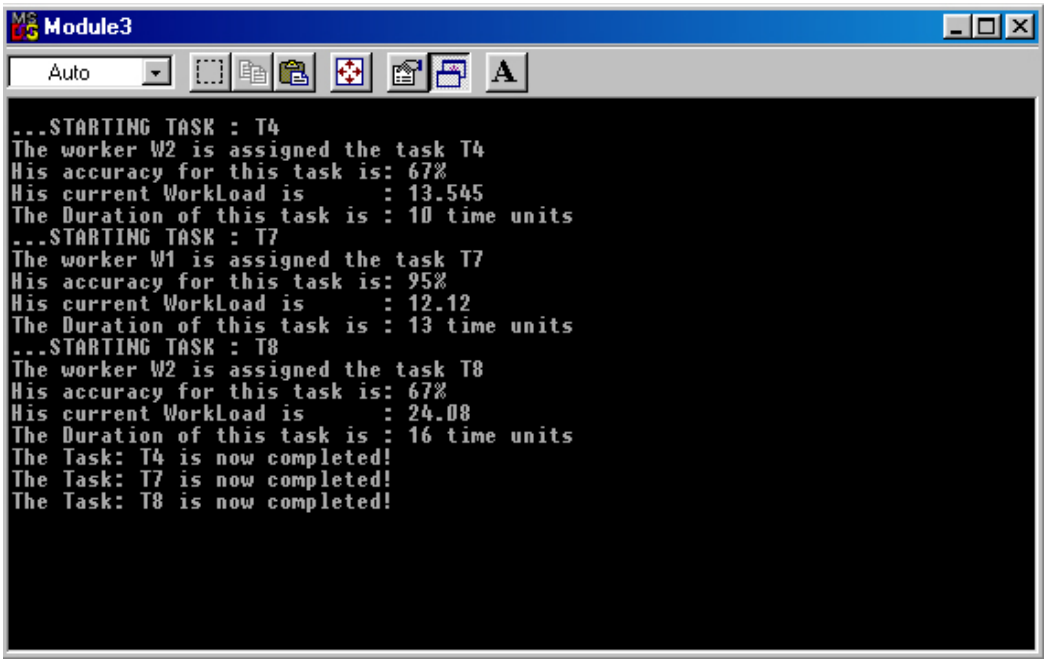


Σχήμα 4.14: Ανάθεση και ολοκλήρωση των T1 και T5

Η προσομοίωση ακολουθεί την ανάλυση που έγινε για την λειτουργία του συστήματος. Στα σχήματα που παρατίθενται δεν μπορεί να φανεί η χρονική διαδοχή των γεγονότων αλλά τα σχήματα παρατίθενται με βάση την χρονική αυτή διαδοχή. Έτσι το σχήμα 4.14 δείχνει την ανάθεση και ολοκλήρωση των δύο πρώτων παράλληλων εργασιών *T1* και *T5*, στο σχήμα 4.15 ανατίθενται και ολοκληρώνονται οι τρεις επόμενες εργασίες *T2*, *T3* και *T6*, ενώ στο σχήμα 4.16 ανατίθενται και ολοκληρώνονται οι τρεις τελευταίες εργασίες *T4*, *T7* και *T8*.



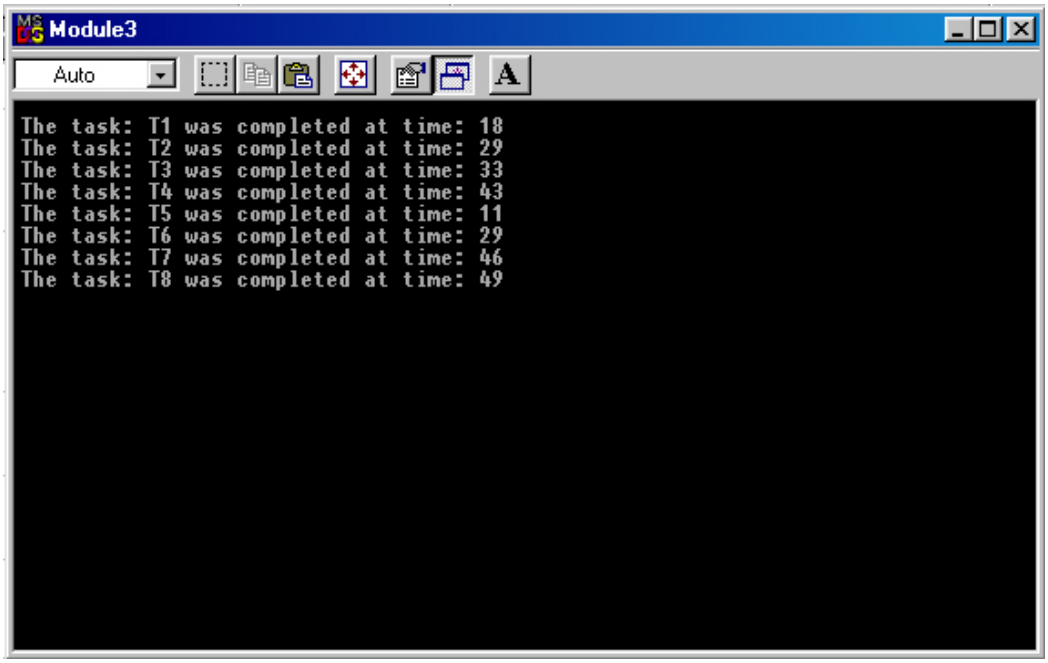
Σχήμα 4.15: Ανάθεση και ολοκλήρωση των T2, T3 και T6



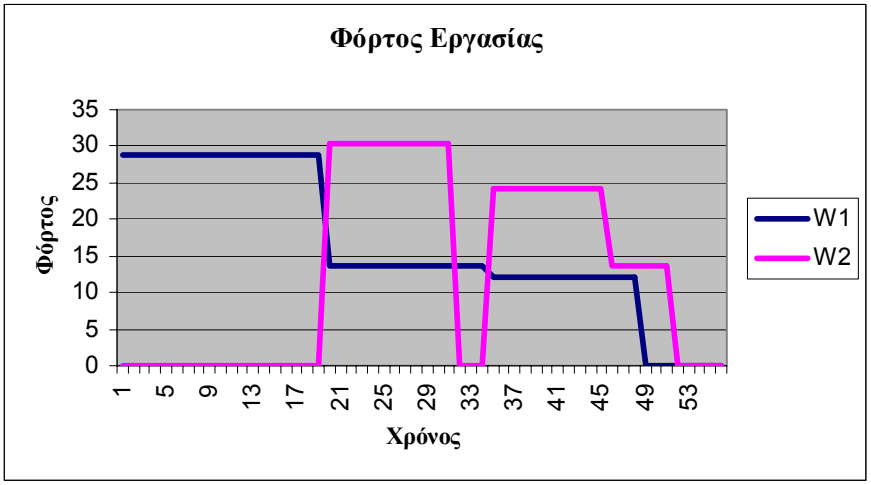
Σχήμα 4.16: Ανάθεση και ολοκλήρωση των T4, T7 και T8

3.3 Αποτελέσματα προσομοίωσης

Μετά την ολοκλήρωση των εργασιών το πρόγραμμα μας δείχνει τις χρονικές στιγμές ολοκλήρωσης αυτών με χρόνο προσομοίωσης 80 χρονικές μονάδες (σχήμα 4.17).



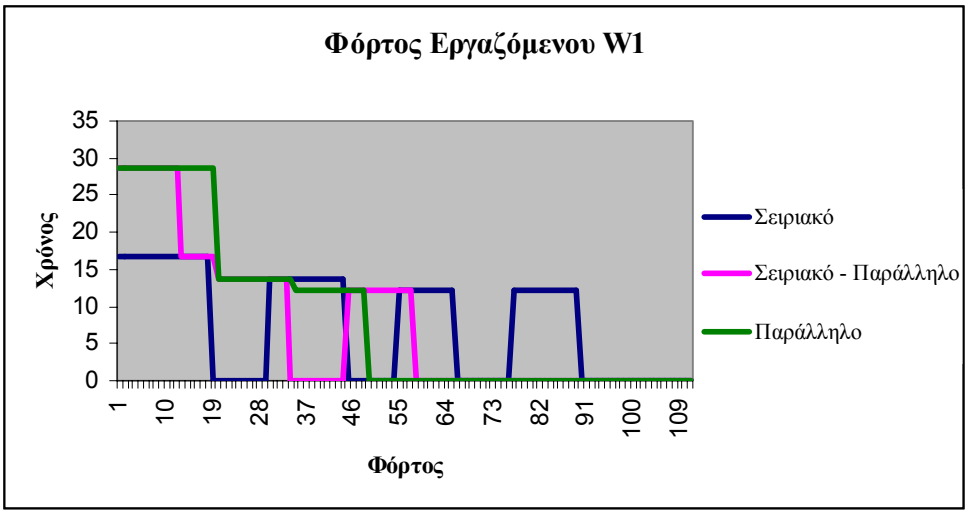
Σχήμα 4.17: Προσομοίωση 80 χρονικών μονάδων για το παράλληλο σύστημα



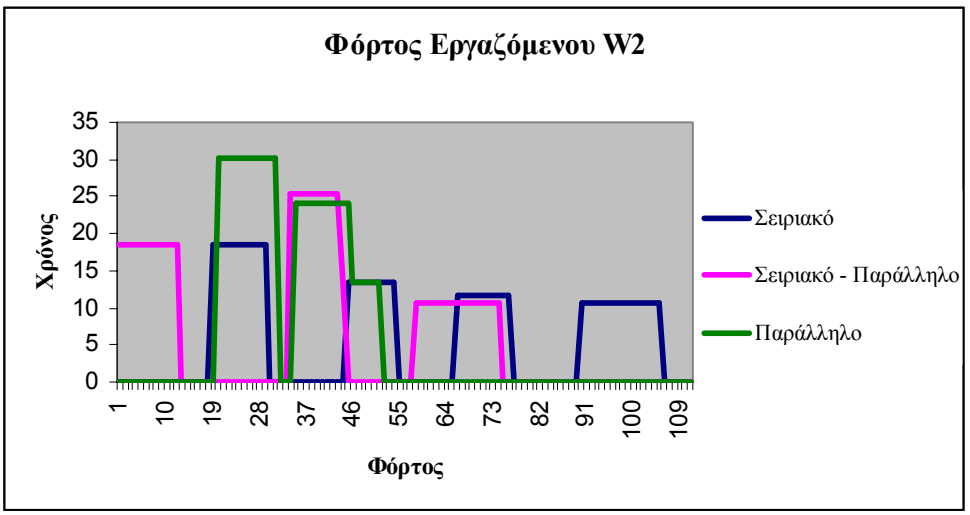
Σχήμα 4.18: Φόρτος εργαζομένων παράλληλου σειριακού συστήματος

4. Σύγκριση μοντέλων και αποτελεσμάτων

Η σύγκριση των τριών μοντέλων ως προς τον φόρτο των δύο εργαζομένων φαίνεται στα σχήματα 4.19 και 4.20.



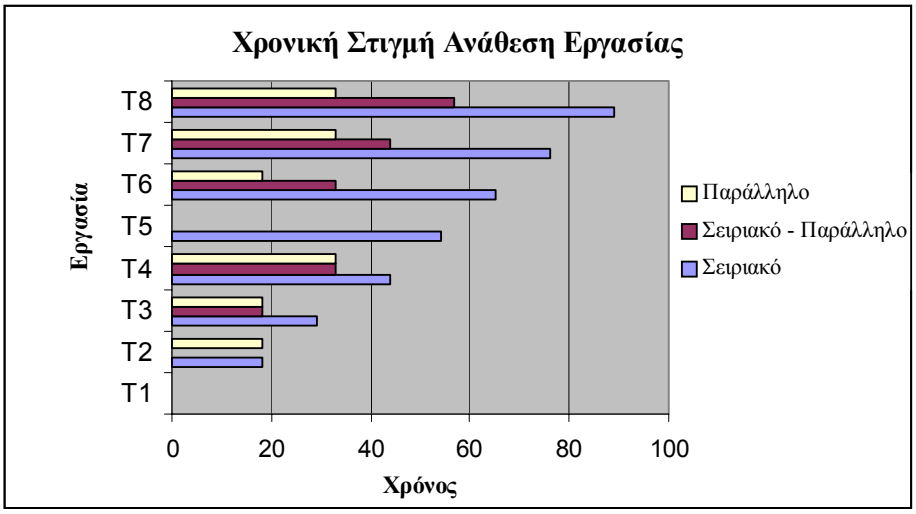
Σχήμα 4.19: Συγκριτικός φόρτος εργαζομένου W1



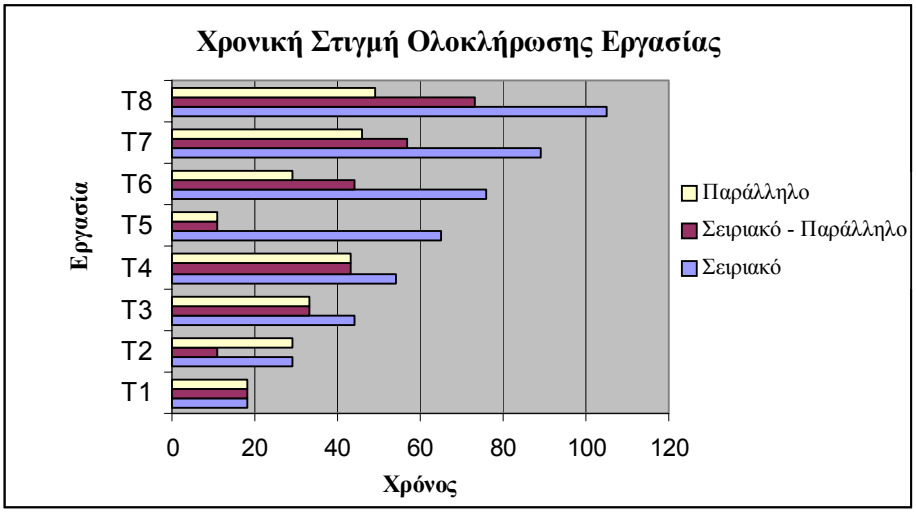
Σχήμα 4.20: Συγκριτικός φόρτος εργαζομένου W2

Από τα δύο αυτά σχήματα βλέπουμε όπως προαναφέραμε ότι στο σειριακό μοντέλο εκτέλεσης εργασιών ο φόρτος του εκάστοτε εργαζόμενου είναι γενικά αρκετά χαμηλότερος (μέγιστη τιμή που εμφανίζει για τον εργαζόμενο *W1* είναι 16,665 ενώ

για τον *W2* είναι 18,48) και για αρκετή μεγάλη περίοδο είναι μηδενικός καθώς ο εργαζόμενος βρίσκεται σε αναμονή της προηγούμενης εργασίας. Αυτό όμως έχει ως συνέπεια την μεγάλη καθυστέρηση τόσο στην ανάθεση των εργασιών στο σύστημα αυτό όσο και στην ολοκλήρωση του. Το γεγονός αυτό φαίνεται καλύτερα στα σχήματα 4.21 και 4.22. Στα δύο άλλα συστήματα παρουσιάζονται και υψηλότερες τιμές φόρτου (στο παράλληλο σύστημα για παράδειγμα η τιμή για τον εργαζόμενο *W2* φτάνει το 30,24 στο χρονικό διάστημα 19-30).



Σχήμα 4.21: Χρονική στιγμή ανάθεσης εργασιών στα τρία συστήματα



Σχήμα 4.22: Χρονική στιγμή ολοκλήρωσης εργασιών στα τρία συστήματα

Αυτό που μπορούμε να δούμε με μία πρώτη ματιά είναι ότι το παράλληλο σύστημα υπερέχει χρονικά τόσο ως προς τον χρόνο που χρειάζεται για να γίνει σε αυτό ανάθεση εργασίας όσο και ως προς το χρόνο ολοκλήρωσης συνολικά των εργασιών και συνεπώς του κάθε υποστόχου. Έτσι βλέπουμε ότι στο παράλληλο σύστημα όλες οι εργασίες έχουν ανατεθεί μέχρι την χρονική στιγμή 33 σε αντίθεση με το σειριακό όπου ή ανάθεση της τελευταίας εργασίας γίνεται την χρονική στιγμή 89 αλλά και με το παράλληλο-σειριακό σύστημα όπου η τελευταία εργασία ανατίθεται τη χρονική στιγμή 57.

Η ταχύτερη ανάθεση των εργασιών έχει τον ανάλογο αντίκτυπο στην χρονική στιγμή ολοκλήρωση τους και κατά συνέπεια στη συνολική ολοκλήρωση του στόχου για το κάθε σύστημα. Έτσι, όπως φαίνεται και στο σχήμα 4.22, στο παράλληλο σύστημα η ολοκλήρωση του στόχου γίνεται τη χρονική στιγμή 49 (οπότε και ολοκληρώνεται η πιο χρονοβόρα εργασία από τις 3 παράλληλες *T4*, *T7*, *T8*, που είναι η *T8*). Στο σειριακό-παράλληλο σύστημα, η ολοκλήρωση του κύριου στόχου γίνεται τη χρονική στιγμή 73 οπότε και ολοκληρώνεται η εργασία *T8*. Στο σειριακό σύστημα έχουμε ολοκλήρωση της λειτουργίας του τη χρονική στιγμή 105, αρκετά αργότερα από τα δύο άλλα συστήματα.

Από όλα τα παραπάνω μπορούμε να συμπεράνουμε ότι σε περίπτωση που οι προδιαγραφές ενός συστήματος απαιτούν την κατά το δυνατόν ταχύτερη ολοκλήρωση του στόχου, η επιλογή ενός μοντέλου παράλληλων εργασιών θα είναι η πιο αρμόζουσα. Βέβαια σε ένα τέτοιο σχήμα υπάρχει πάντα το ενδεχόμενο μεγάλης αύξησης του φόρτου και κατά συνέπεια σφαλμάτων στην εργασία ή καθυστέρηση ολοκλήρωσης. Για το λόγο αυτό η ανάθεση των εργασιών στους εργαζομένους θα πρέπει να γίνεται με ιδιαίτερη προσοχή.

Στην περίπτωση που οι προδιαγραφές ενός συστήματος απαιτούν σχετικά γρήγορη ολοκλήρωση του αλλά παράλληλα υπάρχουν κάποιες εργασίες οι οποίες πρέπει απαραιτήτως να γίνουν σωστά και κατά το δυνατόν με την μεγαλύτερη ακρίβεια, η πιο αρμόζουσα επιλογή συστήματος είναι το παράλληλο-σειριακό. Ως σειριακές εργασίες επιλέγονται εκείνες οι οποίες απαιτείται να πραγματοποιηθούν σωστά και με ακρίβεια ενώ παράλληλα μπορούν να εκτελεστούν όλες οι υπόλοιπες με προσοχή και πάλι στην αύξηση του φόρτου των εργαζομένων.

Αν τέλος αυτό που απαιτείται είναι απολύτως σίγουρη και ακριβής εκτέλεση ενός πλήθους εργασιών, χωρίς να υπάρχουν χρονικοί περιορισμοί τότε το καταλληλότερο σύστημα είναι το σειριακό. Το σύστημα αυτό εγγυάται ότι οι εργασίες θα ανατεθούν στα άτομα που θα τις πραγματοποιήσουν με τη μέγιστη δυνατή ακρίβεια και επιπλέον ανά πάσα στιγμή θα εκτελούν μία και μόνο εργασία ώστε να έχουν χαμηλό φόρτο περιορίζοντας έτσι σημαντικά την πιθανότητα σφαλμάτων.

Παράρτημα

Α. Κλάσεις

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <assert.h>

int Tsk=2;
int Wrk=2;
bool checker=false;
bool checker2=false;

//*********************************************************
//****************** Main Task Declarations ***************
//*********************************************************

class MainTask{
public:
        char* TaskID;
        int SubTasks;
        char* SubTid[10];
        double Duration[10];
        long StartedAt[10];
        bool SendBack[10];
        double WhenCompleted[10];
        int Priority[10];

        MainTask();
        char* MTid() const {return TaskID;}
        int HaveSubTask();
        ~MainTask();

        double RetWLperce(int w) {return WLperce[w];}
        double RetWLdecis(int w) {return WLdecis[w];}
        double RetWLmotor(int w) {return WLmotor[w];}

        char* STaskID;
        double WLperce[10];
        double WLdecis[10];
        double WLmotor[10];
        int TypeOfT[10];
        bool complete[10];
};
```

```
MainTask *MT[1000];

MainTask::MainTask()
{
	int i;
	double selWL;
	char* tot;
	TaskID=new char;
	TaskID[0]='G';
	TaskID[1]='O';
	TaskID[2]='A';
	TaskID[3]='L';

	for (i=0; i<10; i++)
		SubTid[i]='\0';

	if (checker==false){
		TaskID[4]=(char)(48+1);
		TaskID[5]='\0';
		SubTasks=4;


		for (int k=0; k<SubTasks; k++){
			Duration[k]=10+(10*rand())/RAND_MAX;
			complete[k]=false;
			StartedAt[k]=-1;
			SendBack[k]=false;
			WhenCompleted[k]=0;
			Priority[k]=(3*rand())/RAND_MAX;
			STaskID=new char;
			STaskID[0]='T';
			STaskID[1]=48+1+k;
			STaskID[2]='\0';
			TypeOfT[k]=k;
			selWL=9+(3*rand())/RAND_MAX;

		switch(TypeOfT[k]){
		case 1:
			WLperce[k]=0.65*selWL;
			WLdecis[k]=0.15*selWL;
			WLmotor[k]=0.20*selWL;
			tot="Perception task";
			break;
		case 2:
			WLperce[k]=0.10*selWL;
			WLdecis[k]=0.15*selWL;
			WLmotor[k]=0.75*selWL;
			tot="Motor task";
			break;
		case 3:
			WLperce[k]=0.20*selWL;
			WLdecis[k]=0.55*selWL;
			WLmotor[k]=0.25*selWL;
			tot="Decision task";
			break;
```

```
		default:
			WLperce[k]=0.10*selWL;
			WLdecis[k]=0.15*selWL;
			WLmotor[k]=0.75*selWL;
			tot="Motor task";
			break;
		}

		SubTid[k]=STaskID;
		cout<<STaskID;
		cout<<": A "<<tot;
		cout<<", WorkLoad Per Channel For This Task:"<<endl;
		cout<<"Perception="<<WLperce[k]<<", "<<endl;
		cout<<"Decision="<<WLdecis[k]<<", "<<endl;
		cout<<"Motor="<<WLmotor[k]<<" "<<endl;
		cout<<endl;
		}

checker=true;
}

else {
		TaskID[4]=(char)(48+2);
		TaskID[5]='\0';
		SubTasks=4;

		for (int k=0; k<SubTasks; k++){
			Duration[k]=10+(10*rand())/RAND_MAX;
			complete[k]=false;
			StartedAt[k]=-1;
			SendBack[k]=false;
			WhenCompleted[k]=0;
			Priority[k]=(3*rand())/RAND_MAX;
			STaskID=new char;
			STaskID[0]='T';
			STaskID[1]=48+5+k;
			STaskID[2]='\0';
			TypeOfT[k]=k;
			selWL=7+(3*rand())/RAND_MAX;

		switch(TypeOfT[k]){
		case 1:
			WLperce[k]=0.65*selWL;
			WLdecis[k]=0.15*selWL;
			WLmotor[k]=0.20*selWL;
			tot="Perception task";
			break;
		case 2:
			WLperce[k]=0.10*selWL;
			WLdecis[k]=0.15*selWL;
			WLmotor[k]=0.75*selWL;
			tot="Motor task";
			break;
		case 3:
			WLperce[k]=0.20*selWL;
```

```
                        WLdecis[k]=0.55*selWL;
                        WLmotor[k]=0.25*selWL;
                        tot="Decision task";
                        break;
                default:
                        WLperce[k]=0.10*selWL;
                        WLdecis[k]=0.15*selWL;
                        WLmotor[k]=0.75*selWL;
                        tot="Motor task";
                        break;
                }


                SubTid[k]=STaskID;
                cout<<STaskID;
                cout<<": A "<<tot;
                cout<<", WorkLoad Per Channel For This Task:"<<endl;
                cout<<"Perception="<<WLperce[k]<<", "<<endl;
                cout<<"Decision="<<WLdecis[k]<<", "<<endl;
                cout<<"Motor="<<WLmotor[k]<<" "<<endl;
                cout<<endl;
                }
        checker=false;
        }

        TaskID[5]='\0';


}

MainTask::~MainTask(){
        delete &TaskID;

}

//************ End Of Main Task Declarations **********************


//************************************************************
//***************** Worker Declarations *************************
//************************************************************

class Workers
{
public:
        Workers();
        ~Workers();
        char* WorkerName;
        bool Availab;
        double WorkLoad;
        char *IsWorkingOn[10];
        double Accuracy[4];
        double TimeToComplete[10];
        char* Wid() const {return WorkerName;}
};
```

```
Workers *WK[10];

Workers::Workers()
{
	int i;

	if (checker2==false) {
		WorkerName="W1";
		Accuracy[0]=50+(10*rand())/RAND_MAX;
		Accuracy[1]=90+(10*rand())/RAND_MAX;
		Accuracy[2]=50+(10*rand())/RAND_MAX;
		Accuracy[3]=Accuracy[1];
		checker2=true;
	}
	else {
		WorkerName="W2";
		Accuracy[0]=60+(15*rand())/RAND_MAX;
		Accuracy[1]=60+(15*rand())/RAND_MAX;
		Accuracy[2]=60+(15*rand())/RAND_MAX;
		Accuracy[3]=60+(15*rand())/RAND_MAX;
		checker2=false;
	}

	for (i=0; i<10; i++){
		IsWorkingOn[i]='\0';
		TimeToComplete[i]=0;
	}
	Availab=true;
	WorkLoad=0;

}

//****************** End Of Worker Declarations ********************
```

Β. Συναρτήσεις

```
//*********************************************************************
//****************** Begin Of Program Functions ************************
//*********************************************************************


//*************** Initialize Workers And Main Tasks Functions ***************

void InitWorkers(){
        for (int i=0; i<Wrk; i++){
                WK[i]=new Workers;
        }
        WK[2]=NULL;
}

void InitMTasks(){
        char* k;int l;

        for (int i=0; i<Tsk; i++){
                MT[i]=new MainTask;
                k=MT[i]->MTid();
                l=MT[i]->SubTasks;
        }

        MT[2]=NULL;
}


//********** End Of Initialize Workers And Main Tasks Functions ************

//****************** Calculate WorkLoad Function **********************

double CntWorkLoad(char* ww){

        int i,j,k,l,m,n,holdn;
        int hold[10];
        double temp,temp1,temp2,temp3,TWL1,TWL2,TWL3,TWL4,WL;
        double Cml[10];
        double Cpl[10];
        double Cdl[10];
        double PWt,MWt,DWt;
        temp=0;temp1=0;temp2=0;temp3=0;
        TWL1=0;TWL2=0;TWL3=0;TWL4=0;
        WL=0;
        PWt=0.8;MWt=0.5;DWt=0.4;
        i=0;j=0;k=0;l=0;n=0;holdn=0;

        for (m=0; m<10; m++){
                hold[m]=-1;
                Cml[m]=-1;
                Cpl[m]=-1;
                Cdl[m]=-1;
        }
```

```
	while ((WK[i]!=NULL) && (WK[i]->Wid()!=ww)){
		i++;
	}

	while ((WK[i]->IsWorkingOn[j]!='\0') && (MT[k]!=NULL)){

		if (WK[i]->IsWorkingOn[j]=="deAssigned"){
			j++;
		}

		if (WK[i]->IsWorkingOn[j]==MT[k]->SubTid[n]){
			if (MT[k]->complete[n]==false){
				Cml[l]=MT[k]->RetWLmotor(n);
				Cpl[l]=MT[k]->RetWLperce(n);
				Cdl[l]=MT[k]->RetWLdecis(n);
			}
			j++; l++; n=0;
		}
		else {
			n=0;
			while ((WK[i]->IsWorkingOn[j]!=MT[k]->SubTid[n]) && (MT[k]-
>SubTid[n]!='\0') && (WK[i]->IsWorkingOn[j]!='\0') && (MT[k]->complete[n]==true)) {
				n++;
				if (n>=MT[k]->SubTasks){
					k++;
					n=0;
				}
				if (MT[k]->complete[n]==false){
					Cml[l]=MT[k]->RetWLmotor(n);
					Cpl[l]=MT[k]->RetWLperce(n);
					Cdl[l]=MT[k]->RetWLdecis(n);
				}
			}
			holdn=n;
			n=0;
			j++;
			l++;
		}

		temp=0;temp1=0;temp2=0;temp3=0;
		m=0;
		if (Cml[l-1]!=-1){
			TWL1=temp+Cml[l-1]+Cpl[l-1]+Cdl[l-1];
			TWL2=temp1+Cpl[l-1];
			TWL3=temp2+Cml[l-1];
			TWL4=temp3+Cdl[l-1];
			temp=TWL1;
			temp1=TWL2;
			temp2=TWL3;
			temp3=TWL4;
			m++;
		}

		TWL2=PWt*TWL2;
```

```
                TWL3=MWt*TWL3;
                TWL4=DWt*TWL4;

                WL=TWL1+TWL2+TWL3+TWL4;

                if (MT[k]->SubTid[n]=='\0')
                        k++;
        }

        WK[i]->WorkLoad=WL;
        return WL;

}
//************* End of Calculate WorkLoad Function *****************



//******************** Assign A Task Function *********************

int AssignTask(char *ww,char *wt){
        int i=0;
        int j=0;
        int k=0;
        int m=0;
        int TypeOfTask;
        double Acc,cntw,dur;
        time_t now;

        while ((WK[i]!=NULL) && (WK[i]->Wid()!=ww)){
                i++;
        }

        WK[i]->Availab=false;

        while ((j<10) && (WK[i]->IsWorkingOn[j]!='\0')){
                j++;
        }

        while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                m++;
                if (MT[k]->SubTid[m]=='\0'){
                        k++;
                        m=0;
                }
        }

        TypeOfTask=MT[k]->TypeOfT[m];
        WK[i]->IsWorkingOn[j]=MT[k]->SubTid[m];
        MT[k]->StartedAt[m]=time(&now);
        dur=MT[k]->Duration[m];
        cntw=CntWorkLoad(ww);
```

```
	switch(TypeOfTask){
		case 1:
			Acc=WK[i]->Accuracy[0];
			break;
		case 2:
			Acc=WK[i]->Accuracy[1];
			break;
		case 3:
			Acc=WK[i]->Accuracy[2];
			break;
		default:
			Acc=WK[i]->Accuracy[3];
			break;
	}
	cout<<"...STARTING TASK : "<<WK[i]->IsWorkingOn[j]<<endl;
	cout<<"The worker "<<WK[i]->Wid()<<" is assigned the task "<<WK[i]-
>IsWorkingOn[j]<<endl;
	cout<<"His accuracy for this task is: "<<Acc<<"%"<<endl;
	cout<<"His current WorkLoad is      : "<<cntw<<endl;

	/*switch(TypeOfTask){
		case 1:
			Acc=WK[i]->Accuracy[0];
			if (cntw<=10){
				WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-10;
				WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-3;
				WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-3;
				WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-3;
			}
			else if ((cntw>10) && (cntw<=20)) {
				WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-15;
				WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-8;
				WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-8;
				WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-8;
			}
			else if ((cntw>20) && (cntw<=30)) {
				WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-20;
				WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-14;
				WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-14;
				WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-14;
			}
			else if (cntw>30) {
				WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-25;
				WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-18;
				WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-18;
				WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-18;
			}
			break;
		case 2:
			Acc=WK[i]->Accuracy[1];
			if (cntw<=10){
				WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-3;
				WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-10;
				WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-3;
				WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-3;
```

```
		}
		else if ((cntw>10) && (cntw<=20)) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-8;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-15;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-8;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-8;
		}
		else if ((cntw>20) && (cntw<=30)) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-14;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-20;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-14;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-14;
		}
		else if (cntw>30) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-18;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-25;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-18;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-18;
		}
		break;
	case 3:
		Acc=WK[i]->Accuracy[2];
		if (cntw<=10){
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-3;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-3;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-10;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-3;
		}
		else if ((cntw>10) && (cntw<=20)) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-8;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-8;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-15;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-8;
		}
		else if ((cntw>20) && (cntw<=30)) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-14;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-14;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-20;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-14;
		}
		else if (cntw>30) {
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-25;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-25;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-25;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-25;
		}
		break;
	default:
		Acc=WK[i]->Accuracy[3];
		if (cntw<=10){
			WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-4;
			WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-4;
			WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-4;
			WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-4;
		}
```

```
                        else if ((cntw>10) && (cntw<=20)) {
                                WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-8;
                                WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-8;
                                WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-8;
                                WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-8;
                        }
                        else if ((cntw>20) && (cntw<=30)) {
                                WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-14;
                                WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-14;
                                WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-14;
                                WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-14;
                        }
                        else if (cntw>30) {
                                WK[i]->Accuracy[0]=WK[i]->Accuracy[0]-20;
                                WK[i]->Accuracy[1]=WK[i]->Accuracy[1]-20;
                                WK[i]->Accuracy[2]=WK[i]->Accuracy[2]-20;
                                WK[i]->Accuracy[3]=WK[i]->Accuracy[3]-20;
                        }
                        break;
        }*/


        return j+1;
}
//************* End of Assign A Task Function *****************


//***************** De Assign A Task Function ****************

int DeAssignTask(char *ww,char *wt){
        int i=0;
        int j=0;
        int k=0;
        int m=0;
        int n=0;
        while ((WK[i]!=NULL) && (WK[i]->Wid()!=ww)){
                i++;
        }

        WK[i]->Availab=false;

        bool flag=false;
        while ((WK[i]->IsWorkingOn[j]!='\0') && (!flag)){
                if (WK[i]->IsWorkingOn[j]==wt)
                        flag=true;
                else
                        j++;
        }


        while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                m++;
                if (MT[k]->SubTid[m]=='\0'){
                        k++;
                        m=0;
```

```
                }
        }

        if (!flag)
         cout<<"The asked task to deassign: "<<wt<<" is not assigned to this Worker:
        "<<ww<<endl;
        else{
                WK[i]->IsWorkingOn[j]="deAssigned";
                cout<<"The Task "<<wt<<" is not anymore assigned to "<<WK[i]-
>Wid()<<endl;
                cout<<"His current WorkLoad is: "<<CntWorkLoad(ww)<<endl;
        }

        j=0;n=0;
        while  (WK[i]->IsWorkingOn[j]!='\0'){
                if (WK[i]->IsWorkingOn[j]=="deAssigned")
                        n++;
                j++;
        }

        return j-n;
}
//***************** End of De Assign A Task Function ***************


//****************** Find Available Worker Function *****************

char* AvailableWorker(){
        int i=0;
        int j=0;
        while ((WK[i]!=NULL) && (WK[i]->Availab==false)){
                i++;
        }

        if (i>1)
                return "none";
        else
                return WK[i]->Wid();
}
//************* End of Find Available Worker Function ***************


//**************** Accomplish a Main Task Function ******************

void AccompMainTask(MainTask **M, Workers** W){

        int i=0;
        int j=1;
        char *AvW;
        AvW=AvailableWorker();

        AssignTask(AvW,M[i]->SubTid[0]);
        while (M[i]!=NULL){
                if (M[i]->SubTid[j]!='\0'){
                        if (AssignTask(AvW,M[i]->SubTid[j])>2){
```

```
                                AvW=AvailableWorker();
                                j++;
                        }
                        else
                                j++;
                }
                else {
                        i++;
                        j=0;
                }
        }

}
//************* End Of Accomplish a Main Task Function ***************


//************* Assign Task Based on Accuracy Function ***************

void AssignAccuracyBased(char *wt){
        int k=0;
        int m=0;
        int j=0;
        int i=0;
        int TypeOfTask;
        double Acc;
        char *Avw;

        while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                m++;
                if (MT[k]->SubTid[m]=='\0'){
                        k++;
                        m=0;
                }
        }

        TypeOfTask=MT[k]->TypeOfT[m];

        while ((WK[i]!=NULL) && (WK[i]->IsWorkingOn[j]!=wt)){
                j++;
                if (WK[i]->IsWorkingOn[j]=='\0'){
                        i++;
                        j=0;
                }
        }

        switch(TypeOfTask){
                case 1:
                        Acc=WK[i]->Accuracy[0];
                        break;
                case 2:
                        Acc=WK[i]->Accuracy[1];
                        break;
                case 3:
                        Acc=WK[i]->Accuracy[2];
                        break;
```

```
                default:
                        Acc=WK[i]->Accuracy[3];
                        break;
        }

        DeAssignTask(WK[i]->Wid(),WK[i]->IsWorkingOn[j]);
        Avw=AvailableWorker();
        AssignTask(Avw,wt);

}
//************* End Of Assign Task Based on Accuracy Function **************


//********************** Check If Task Waits Function **********************

bool IfTaskWaits(char *wt){
        int i=0;
        int j=0;
        int k=0;
        int m=0;
        int hold=-1;
        int TypeOfTask;
        double Acc;

        while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                m++;
                if (MT[k]->SubTid[m]=='\0'){
                        k++;
                        m=0;
                }
        }

        while ((j<10) && (WK[i]->IsWorkingOn[j]!='\0')){
                j++;
        }

        TypeOfTask=MT[k]->TypeOfT[m];

        while (WK[i]!=NULL){
                switch(TypeOfTask){
                        case 1:
                                Acc=WK[i]->Accuracy[0];
                                break;
                        case 2:
                                Acc=WK[i]->Accuracy[1];
                                break;
                        case 3:
                                Acc=WK[i]->Accuracy[2];
                                break;
                        default:
                                Acc=WK[i]->Accuracy[3];
                                break;
                }
                if (Acc<=85)
                        i++;
```

```
                else {
                        hold=i;
                        i++;
                }
        }

        if ((i>=Wrk) && (hold==-1))
                return true;
        else {
                AssignTask(WK[hold]->Wid(),wt);
                return false;
        }

}
//******************** End of Check If Task Waits Function *****************


//********** Assign Task Based on the fact that it waiting Function *********

int AssignIfTaskWaits(char *wt){
        int i=0;
        int j=0;
        int k=0;
        int m=0;
        int TypeOfTask;
        int hold=0;
        bool flag=false;
        double Acc,temp;

        while ((WK[i]!=NULL) && (!flag)){
                if (WK[i]->IsWorkingOn[j]==wt)
                        flag=true;
                else {
                        if (WK[i]->IsWorkingOn[j]!='\0')
                                j++;
                        else {
                                i++;
                                j=0;
                        }
                }
        }

        if ((i<Wrk))
                cout<<"The Task: "<<wt<<" is in progress by: "<<WK[i]->Wid()<<endl;
        else {
                i=0;
                j=0;
                while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                        m++;
                        if (MT[k]->SubTid[m]=='\0'){
                                k++;
                                m=0;
                        }
                }
```

```
		TypeOfTask=MT[k]->TypeOfT[m];

		if (IfTaskWaits(wt)==false)
			return 1;
		else {
			while (WK[i]!=NULL){
				switch(TypeOfTask){
					case 1:
						Acc=WK[i]->Accuracy[0];
						break;
					case 2:
						Acc=WK[i]->Accuracy[1];
						break;
					case 3:
						Acc=WK[i]->Accuracy[2];
						break;
					default:
						Acc=WK[i]->Accuracy[3];
						break;
				}
				if (i==0)
					temp=Acc;
				if ((i>0) && (temp<Acc)){
					hold=i;
					temp=Acc;
				}
				i++;

			}
		}

		AssignTask(WK[hold]->Wid(),wt);
	}


}
//**** End Of Assign Task Based on the fact that it waiting Function *********


//********* Send Back a Task if it waits instead of assing it Function *******

void SendTaskBack(char *wt){
	int i=0;
	int j=0;
	int k=0;
	int m=0;
	int TypeOfTask;
	int hold=0;
	bool flag=false;
	bool islast=false;
	char *tempT;

	while ((WK[i]!=NULL) && (!flag)){
		if (WK[i]->IsWorkingOn[j]==wt)
			flag=true;
```

```
                else {
                        if (WK[i]->IsWorkingOn[j]!='\0')
                                j++;
                        else {
                                i++;
                                j=0;
                        }
                }
        }

        if ((i<Wrk))
                cout<<"The Task: "<<wt<<" is in progress by: "<<WK[i]->Wid()<<endl;
        else {
                i=0;
                j=0;
                while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
                        m++;
                        if (MT[k]->SubTid[m]=='\0'){
                                k++;
                                m=0;
                        }
                }

                TypeOfTask=MT[k]->TypeOfT[m];

                if (IfTaskWaits(wt)==false)
                        cout<<"this task is not waiting"<<endl;
                else {
                        MT[k]->SendBack[m]=true;
                        if (MT[k]->SubTid[m+1]=='\0')
                                islast=true;
                        else {
                                if (MT[k]->SubTid[m+2]=='\0'){
                                        tempT=MT[k]->SubTid[m];
                                        MT[k]->SubTid[m]=MT[k]->SubTid[m+1];
                                        MT[k]->SubTid[m+1]=tempT;
                                        islast=true;
                                }
                                else {
                                        tempT=MT[k]->SubTid[m];
                                        MT[k]->SubTid[m]=MT[k]->SubTid[m+1];
                                        MT[k]->SubTid[m+1]=tempT;
                                }
                        }
                }
        }
}


//**** End Of Send Back a Task if it waits instead of assing it Function *****
```

```
//************** Change the worker of a Task that is in progress *************

void ChangeWorkerTaskInProgress(char *wt){
	int i=0;
	int j=0;
	int k=0;
	int m=0;
	int p=0;
	int TypeOfTask;
	int hold=0;
	int NofT[15];
	double cntw[15];
	double Wacc[15];
	bool flag=false;
	double temp,tempWL;

	for (p=0; p<Wrk; p++){
		NofT[p]=0;
		cntw[p]=0;
	}

	if (IfTaskWaits(wt)==false)
		cout<<"this task is not waiting"<<endl;
	else {
		while (WK[i]!=NULL){
			if (WK[i]->IsWorkingOn[j]!='\0'){
				NofT[i]=NofT[i]+1;
				j++;
			}
			else {
				i++;
				j=0;
			}
		}

		for (p=0; p<Wrk; p++)
			cntw[p]=CntWorkLoad(WK[p]->Wid());

		while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=wt)){
			m++;
			if (MT[k]->SubTid[m]=='\0'){
				k++;
				m=0;
			}
		}

		TypeOfTask=MT[k]->TypeOfT[m];
		i=0;
		while (WK[i]!=NULL){
			switch(TypeOfTask){
				case 1:
					Wacc[i]=WK[i]->Accuracy[0];
					break;
				case 2:
					Wacc[i]=WK[i]->Accuracy[1];
```

```
                                        break;
                                case 3:
                                        Wacc[i]=WK[i]->Accuracy[2];
                                        break;
                                default:
                                        Wacc[i]=WK[i]->Accuracy[3];
                                        break;
                        }
                        i++;
                }
                temp=Wacc[0];
                for (p=1; p<Wrk; p++){
                        if (temp<Wacc[p]){
                                temp=Wacc[p];
                                hold=p;
                        }
                }

                AssignTask(WK[hold]->Wid(),wt);
                tempWL=CntWorkLoad(WK[hold]->Wid());
                if (tempWL>=(cntw[hold]+0.3*cntw[hold])){
                        AssignAccuracyBased(WK[hold]->IsWorkingOn[0]);
                }

        }


}
//******** End Of Change the worker of a Task that is in progress ************


//********************* Send Back a Task that is in progress *****************

void SendBackTaskInProgress(char *tsendback, char *tassign){
        int i=0;
        int j=0;
        int k=0;
        int m=0;
        int p=0;
        int q=0;
        int hold=0;
        bool flag=false;
        double check;
        time_t now;


        while ((MT[k]!=NULL) && (MT[k]->SubTid[m]!=tsendback)){
                m++;
                if (MT[k]->SubTid[m]=='\0'){
                        k++;
                        m=0;
                }
        }

        while ((MT[p]!=NULL) && (MT[p]->SubTid[q]!=tassign)){
```

```
                q++;
                if (MT[p]->SubTid[q]=='\0'){
                        p++;
                        q=0;
                }
        }

        if (MT[k]->complete[m])
                cout<<"this task: "<<tsendback<<" is completed"<<endl;
        else {
                while ((WK[i]!=NULL) && (WK[i]->IsWorkingOn[j]!='\0')){
                        if (WK[i]->IsWorkingOn[j]!=tsendback){
                                j++;
                        }
                        else {
                                i++;
                                j=0;
                        }
                }

                if (i<Wrk){
                        cout<<"The Task: "<<tsendback<<" is in progress by: "<<WK[i]-
>Wid()<<endl;

                        time(&now);
                        check=difftime(now,MT[k]->StartedAt[m]);
                        if ((check<0.5*MT[k]->Duration[m]) && (MT[p]-
>Priority[q]==3)){
                                DeAssignTask(WK[i]->Wid(),WK[i]->IsWorkingOn[j]);
                                AssignTask(WK[i]->Wid(),MT[p]->SubTid[q]);
                        }
                }
        }


}
//*************** End Of Send Back a Task that is in progress ***************
```

Γ. Simulation

```
void Simulation(){

	srand( (unsigned)time( NULL ) );
	int i=0;
	int j=0;
	int k=0;
	int m=0;
	int x=0;
	int y=0;
	int howm=0;
	int Pwith=0;
	int holdm,holdi,toft,holdm1;
	double dur,SimTime,check1, check2,tmp;
	bool flag=false;
	bool finished=false;
	bool flag1=false;
	bool flag2=true;
	char *AvW;
	long when=0;
	int DurAll[3];
	int a=0;
	int Ldur=0;
	int wk1task=0;
	int wk2task=0;
	bool wk1=false;
	bool wk2=false;
	bool wk1s1=false;
	bool wk2s1=false;
	bool wk1s2=false;
	bool wk2s2=false;
	bool flaggy=false;
	int is1=0;
	int is2=0;
	int isp=0;
	int isp1=0;
	int iso=0;
	int iso1=0;
	int isov1=0;
	int lch=0;
	int mti=0;
	int mti1=0;
	int mtm=0;
	bool as1=false;
	bool as1s1=false;
	bool as1s2=false;
	bool as2=false;
	bool as2s1=false;
	bool as2s2=false;
	int ias1=0;
	int mas1=0;
	int ias1s1=0;
	int mas1s1=0;
	int ias1s2=0;
	int mas1s2=0;
	int ias2=0;
	int mas2=0;
	int ias2s1=0;
```

```
int mas2s1=0;
int ias2s2=0;
int mas2s2=0;
int isfin=0;
int isfin1=0;



DurAll[0]=0;
DurAll[1]=0;
DurAll[2]=0;

SimTime=80;
time_t startT,finishT,start,finish,now;

time(&startT);

while (difftime(finishT,startT)<SimTime){
        if ((difftime(now,startT)>=when) && (!flag)){
                if (!finished){
                        i=0;m=0;
                        flaggy=false;
                        as1=false;
                        as1s1=false;
                        as1s2=false;
                        as2=false;
                        as2s1=false;
                        as2s2=false;

                        wk1task=0;
                        wk2task=0;
                        while (MT[i]->complete[m]==true){
                                m++;
                                if (m==4){
                                        i=1;
                                        m=0;
                                }
                        }

                        if (MT[i]->SubTid[m]=='\0'){
                                        i++;
                                        m=0;
                        }
                        toft=MT[i]->TypeOfT[m];
                        flag=true;
                        if (flag1=true){

                                toft=MT[i]->TypeOfT[m];
                                holdm=m;
        if ((toft==0) || (toft==2)) {
                howm=AssignTask("W1",MT[i]->SubTid[m]);
                as1=true;
                ias1=i;
                mas1=m;
                dur=MT[i]->Duration[m];
                DurAll[a]=dur; a++;
                cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                for (x=0; x<2; x++) {
                        m=holdm;
```

```
                                        if (ias1==1) m=m+4;
                                        if (MT[i]->ParallelWith[m][x]!=0){
                                                Pwith=MT[i]->ParallelWith[m][x];
                                                holdm=m;
                                                m=Pwith-1;
                                                if (Pwith>4){
                                                i=1;
                                                m=Pwith-5;
                                                }
                                        toft=MT[i]->TypeOfT[m];
                                        if ((toft==0) || (toft==2)){
                                                howm=AssignTask("W1",MT[i]->SubTid[m]);
                                                as1s1=true;
                                                ias1s1=i;
                                                mas1s1=m;
                                                dur=MT[i]->Duration[m];
                                                DurAll[a]=dur; a++;
                                                cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                                        }
                                        else {
                                                howm=AssignTask("W2",MT[i]->SubTid[m]);
                                                as1s2=true;
                                                ias1s2=i;
                                                mas1s2=m;
                                                dur=MT[i]->Duration[m];
                                                DurAll[a]=dur; a++;
                                                cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                                        }
                                }
                        }
}
                                else {
                                        howm=AssignTask("W2",MT[i]->SubTid[m]);
                                        as2=true;
                                        ias2=i;
                                        mas2=m;
                                        dur=MT[i]->Duration[m];
                                        DurAll[a]=dur; a++;
                                        cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                                        for (x=0; x<2; x++) {
                                                m=holdm;
                                                if (ias2==1) m=m+4;
                                                if (MT[i]->ParallelWith[m][x]!=0){
                                                Pwith=MT[i]->ParallelWith[m][x];
                                                holdm=m;
                                                m=Pwith-1;
                                                if (Pwith>4){
                                                        i=1;
                                                        m=Pwith-5;
                                                }
                                                toft=MT[i]->TypeOfT[m];
                                                if ((toft==0) || (toft==2)){
                                                        howm=AssignTask("W1",MT[i]->SubTid[m]);
                                                        as2s1=true;
                                                        ias2s1=i;
                                                        mas2s1=m;
                                                        dur=MT[i]->Duration[m];
                                                        DurAll[a]=dur; a++;
                                                cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                                                }
```

```
                        else {
                                howm=AssignTask("W2",MT[i]->SubTid[m]);
                                as2s2=true;
                                ias2s2=i;
                                mas2s2=m;
                                dur=MT[i]->Duration[m];
                                DurAll[a]=dur; a++;
                                cout<<"The Duration of this task is : "<<dur<<" time units"<<endl;
                                }
                        }
                }
        }
}
}
}

        else{

                time(&now);
                if (flag || finished){
                        flag=false;
                        dur=MT[i]->Duration[m];
                        tmp=difftime(now,startT);
                        when=0;

                        holdm=m;
                        holdi=i;
                        m=0;
                        i=0;
                        mti=0;
                        mtm=0;

                        while ((MT[i]!=NULL) && (MT[i]->StartedAt[m]!=-1)){
                                time(&now);

while ((WK[0]->IsWorkingOn[wk1task]!='\0') || (WK[1]->IsWorkingOn[wk2task]!='\0')  && (flaggy))
{

if (as1==true){
        mti=ias1;
        mtm=mas1;
        if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]) {
                wk1=true;
                time(&now);
                check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
                if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
                MT[mti]->complete[mtm]=true;
                WK[0]->IsWorkingOn[wk1task]="deAssigned";
                cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
                MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
                }
        }
else{
        while (WK[0]->IsWorkingOn[wk1task]!='\0') {
                wk1task++;
                if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]){
                        wk1=true;
                        time(&now);
                check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
                if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
```

```
		MT[mti]->complete[mtm]=true;
		WK[0]->IsWorkingOn[wk1task]="deAssigned";
		cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
		MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
		}
}
else wk1=false;
}
wk1task=0;
}
}

if (as1s1==true){
		mti=ias1s1;
		mtm=mas1s1;
		if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]) {
			wk1s1=true;
			time(&now);
			check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);

			if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
				MT[mti]->complete[mtm]=true;
				WK[0]->IsWorkingOn[wk1task]="deAssigned";
			cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
				MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
			}
		}
		else{
			while (WK[0]->IsWorkingOn[wk1task]!='\0') {
				wk1task++;
				if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]){
					wk1s1=true;
					time(&now);
			check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
				if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
					MT[mti]->complete[mtm]=true;
					WK[0]->IsWorkingOn[wk1task]="deAssigned";
			cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
					MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
				}
			}
			else wk1s1=false;
		}
		wk1task=0;
}

}

		if (as1s2==true){
			mti=ias1s2;
			mtm=mas1s2;
			if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
				wk1s2=true;
				time(&now);
			check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
				if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
					MT[mti]->complete[mtm]=true;
					WK[1]->IsWorkingOn[wk2task]="deAssigned";
```

```
		cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
		MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
		}
}

		else{
			while (WK[1]->IsWorkingOn[wk2task]!='\0') {
				wk2task++;
				if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
					wk1s2=true;
					time(&now);
			check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
					if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
						MT[mti]->complete[mtm]=true;
						WK[1]->IsWorkingOn[wk2task]="deAssigned";
			cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
					MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
					}
				}
			else wk1s2=false;
		}
		wk2task=0;
}
}
		if (as2==true){
			mti=ias2;
			mtm=mas2;
			if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
				wk2=true;
				time(&now);
			check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
				if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
					MT[mti]->complete[mtm]=true;
					WK[1]->IsWorkingOn[wk2task]="deAssigned";
			cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
					MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
				}
			}
			else{
				while (WK[1]->IsWorkingOn[wk2task]!='\0') {
					wk2task++;
					if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
						wk2=true;
						time(&now);
			check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
						if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
							MT[mti]->complete[mtm]=true;
							WK[1]->IsWorkingOn[wk2task]="deAssigned";
			cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
							MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
							}
					}
					else wk2=false;
				}
				wk2task=0;
			}
		}
			if (as2s1==true){
				mti=ias2s1;
				mtm=mas2s1;
```

```
if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]) {
        wk2s1=true;
        time(&now);
        check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
        if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
                MT[mti]->complete[mtm]=true;
                WK[0]->IsWorkingOn[wk1task]="deAssigned";
                cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
                MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
        }
}
else{
        while (WK[0]->IsWorkingOn[wk1task]!='\0') {
                wk1task++;
                if (MT[mti]->SubTid[mtm]==WK[0]->IsWorkingOn[wk1task]){
                        wk2s1=true;
                        time(&now);
                check1=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
                        if ((check1<=0) && (MT[mti]->complete[mtm]==false)) {
                                MT[mti]->complete[mtm]=true;
                                WK[0]->IsWorkingOn[wk1task]="deAssigned";
                cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
                                MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
                        }
                }
                else wk2s1=false;
        }
        wk1task=0;
}
}
if (as2s2==true){
        mti=ias2s2;
        mtm=mas2s2;
        if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
                wk2s2=true;
                time(&now);
                check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
                if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
                        MT[mti]->complete[mtm]=true;
                        WK[1]->IsWorkingOn[wk2task]="deAssigned";
                cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
                        MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
                }
        }
        else{
                while (WK[1]->IsWorkingOn[wk2task]!='\0') {
                        wk2task++;
                        if (MT[mti]->SubTid[mtm]==WK[1]->IsWorkingOn[wk2task]) {
                                wk2s2=true;
                                time(&now);
                check2=MT[mti]->Duration[mtm]-difftime(now,MT[mti]->StartedAt[mtm]);
                                if ((check2<=0) && (MT[mti]->complete[mtm]==false)) {
                                        MT[mti]->complete[mtm]=true;
                                        WK[1]->IsWorkingOn[wk2task]="deAssigned";
                cout<<"The Task: "<<MT[mti]->SubTid[mtm]<<" is now completed!"<<endl;
                                        MT[mti]->WhenCompleted[mtm]=difftime(now,startT);
                                }
                        }
```

```
                        else wk2s2=false;
                }
        wk2task=0;
        }
}


        if (MT[mti]->complete[mtm]==true) {
                if (mti==1)
                        lch=mtm+4;
                else lch=mtm;
                if ((MT[mti]->ParallelWith[lch][1]==0) && (MT[mti]->ParallelWith[lch][0]==0)){
                        if (MT[mti]->complete[mtm]==true)
                                flaggy=true;
                        else flaggy=false;
                }

                if ((MT[mti]->ParallelWith[lch][1]==0) && (MT[mti]->ParallelWith[lch][0]!=0)){
                        isp=MT[mti]->ParallelWith[lch][0];
                        isp=isp-1;
                        if (isp>=4){
                                mti=1;
                                isp=isp-4;
                        }
                        if (MT[mti]->complete[isp]==true)
                                flaggy=true;
                        else flaggy=false;
                }

                if ((MT[mti]->ParallelWith[lch][1]!=0) && (MT[mti]->ParallelWith[lch][0]==0)){
                        isp1=MT[mti]->ParallelWith[lch][1];
                        isp1=isp1-1;
                        if (isp>=4){
                                mti=1;
                                isp1=isp1-4;
                        }
                        if (MT[mti]->complete[isp1]==true)
                                flaggy=true;
                        else flaggy=false;
                }

                if ((MT[mti]->ParallelWith[lch][0]!=0) && (MT[mti]->ParallelWith[lch][1]!=0)){
                        iso=MT[mti]->ParallelWith[lch][0];
                        iso1=MT[mti]->ParallelWith[lch][1];
                        iso=iso-1;
                        iso1=iso1-1;
                        if (iso>=4){
                                mti=1;
                                iso=iso-4;
                        }
                        if (iso1>=4){
                                mti1=1;
                                iso1=iso1-4;
                        }

                }

                if ((MT[mti]->complete[iso]==true) && (MT[mti1]->complete[iso1]==true)){
                        flaggy=true;
                        mtm++;
```

```
                        if (mtm==4){
                        mtm=0;
                        mti=1;
                }
        }
        else flaggy=false;

        if (!flaggy){
                mtm++;
                if (mtm==4){
                        mtm=0;
                        mti=1;
                }
        }

}
else  {
        mtm++;
        if (mtm==4){
                if (mti==1)
                        mtm=5;
                else{
                        mtm=0;
                        mti=1;
                }
        }

}

if (flaggy){
        wk1task=5;
        wk2task=5;
        mti=holdi;
        mtm=holdm;
        i=2;
}

}
}
        m=holdm;
        i=holdi;
        while ((MT[isfin]!=NULL) && (MT[isfin]->complete[isfin1]==true)){
                isfin1++;
                if (isfin1==4){
                        isfin++;
                        isfin1=0;
                }
        }

        if (isfin==2)
                finished=true;
        if (!finished){
                holdm1=m;
                m++;
                if (MT[i]->SubTid[m]=='\0'){
                        flag1=true;
                        i++;
                        m=0;
                }
                else
```

```
                        m=holdm1;
        }

                if ( howm>3 )
                        AvW=AvailableWorker();
        }


}


        time(&finishT);

}
i=0;
m=0;

cout<<endl;
cout<<endl;
while (MT[i]!=NULL){
        if (MT[i]->complete[m]){
                cout<<"The task: "<<MT[i]->SubTid[m]<<" was completed at time: "<<MT[i]-
        >WhenCompleted[m]<<endl;
        }
        else
        cout<<"The task: "<<MT[i]->SubTid[m]<<" was not completed in simulation time"<<endl;
                m++;
                if (MT[i]->SubTid[m]=='\0'){
                        i++;
                        m=0;
                }
        }
}
```

Βιβλιογραφία

[ASB1989] Aldrich, T.B., Szabo, S.M., & Bierbaum, C.R. "The development and application of models to predict operator workload during system design". In G.R. McMillan, D. Beevis, E. Salas, M.H. Srub, R. Sutton, G L.V. Breda, Applications of Human Performance Models to System Design (pp 65-80). New York: Plenum. 1989.

[BL1977] Baron, S., & Levinson, W.H. "Display analysis using the optimal control model of the human operator". Human Factors, 19 (pp 437-457). 1977.

[BS1994] Bi, S., & Salvendy, G. "A proposed methodology for prediction of mental workload based on engineering system parameters". Work and Stress: 8, (pp 355-371). 1994.

[CLP1987] Chubb, G.P., Laughery, K.R., & Pritsker, A.B. "Simulating manned systems". In G. Salvendy, Handbook of Human Factors and Ergonomics. New York: Wiley. 1987.

[CWG1996] Cohen, D., Wherry, R.J. Jr., & Glenn, F. "Analysis of workload predictions generated by multiple recourse theory". Aviation, Space and Environmental Medicine, 67(2), (pp 139-145). 1996.

[DDL1991] Damos, D.L. "Multiple Task Performance". Washington, DC: Taylor & Francis. 1991.

[EWKD1991] Eggemeier, F.T, Wilson, G.G., Kramer, A.F., & Damos, D.L. "Workload assessment in multi-task environments". In D.L. Damos. Multiple Task Performance (pp 207-216). Washington, DC: Taylor & Francis. 1991.

[GDDE1986] Gopher, D., & Donchin, E. "Workload: An examination of the concept". In K.R. Boff, L. Kaufman and J. Thomas, Handbook of perception and Human Performance: Volume II. Cognitive Processes and Performance (pp 41-1-41-49). New York: Wiley. 1986.

[GDKR1989] Gopher, D., & Kimchi, R. Engineering psychology. Annual Review of Psycology, 40, 431-455, 1989.

[HB1990] Hamilton, D.B., & Bierbaum, C.R. "Task analysis/workload (TAWL): A methodology for predicting operator workload". In Proceedings of the Human Factors Society 34th Annual Meeting (pp 1117-1121). Santa Monica, CA: Human Factors Society. 1990.

[HBMWCD1993] Huey, B.M., & Wickens, C.D. (Eds.). "Workload in Transition". Washington, DC: National Academy Press, 1993.

[HPY1989] Hunt, E., Pellegrino, J., & Yee, P. "Assessment and modeling of information coordination abilities". In R. Kanfer, P. Ackerman & R. Cudek, Learning and individual differences: Abilities, motivation and methodology. Hillsdale, NJ: Elbraum. 1989.

[HSG2001] Hart, S.G. "Pilots' workload coping strategies". Paper presented at the AIAA/NASA/FAA/HFS Conference on Challenges in Aviation Human Factors, Tysons Corner, VA. January 1991.

[HSGSLE1988] Hart, S.G., & Staveland, L.E. "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research". In P.A. Hancock and N. Meshkati, Human Mental Workload (pp 139-183). Amsterdam: Elsevier Science. 1988.

[KAF1991] Kramer, A.F. "Physiologigal metrics of mental workload: a review of recent progress". In D.L. Damos. Multiple Task Performance (pp 279-328). Washington, DC: Taylor & Francis. 1991.

[KG1985] Kuperman, G.G. "Pro-SWAT applied to advanced helicopter crewstation concepts". In Proceedings of the Human Factors Society 29th Annual Meeting (pp 398-402). Santa Monica, CA: Human Factors Society. 1985.

[KMA1984] Kirkpatrick, M., Malone, T.B., & Andrews P.J. "Development of an interactive microprocessor-based workload evaluation model (SIMWAM)". In Proceedings of the Human Factors Society 228th Annual Meeting. Santa Monica, CA: Human Factors Society. 1984.

[LPD1989] Linton, P.M., Plamondon, B.D., & Dick, A.O. "Operator workload for military system acquisition". In G.R. McMillan, D. Beevis, E. Salas, M.H. Srub, R. Sutton, G L.V. Breda, Applications of Human Performance Models to System Design (pp 21-46). New York: Plenum. 1989.

[LRACBR2002] Lardner, R., Amati, C., Briner, R. "Management Standards for preventing and resolving workload problems causing stress". Final Version, March 2002

[MN1988] Meshkati, N. "Toward development of a cohesive model of workload". In P.A. Hancock and N. Meshkati, Human Mental Workload (pp 305-314). Amsterdam: Elsevier Science. 1988.

[MD1985] Meister, D. "Behavioral Analysis and Measurement Methods". New York: Wiley. 1985.

[MON1988] Moray, N. "Mental workload since 1979". In D.J. Osborne, International Review of Ergonomics (pp 123-150). Taylor & Francis. 1988.

[MON1967] Moray, N. "Where is capacity limited? A survey and a model". Acta Psychologica, 27, (pp 84-92). 1967.

[NDBD1975] Norman, D.A. & Bobrow, D.G. "On data-limited and recourse-limited processes". Cognitive Psychology, 7, (pp 44-64). 1975.

[NF1995] Nachreiner, F. "Standards for ergonomic principles relating to the design of work systems and to mental workload". Applied Ergonomics, 26(4), 259-263, 1995.

[NR1989] North, R.A., & Riley, V.A. "W/INDEX: A predictive model of operator workload". In G.R. McMillan, D. Beevis, E. Salas, M.H. Srub, R. Sutton, G L.V. Breda, Applications of Human Performance Models to System Design (pp 81-90). New York: Plenum. 1989.

[ORDEFT1986]. O'Donnell, R.D., & Eggemeier, F.T. "Workload assessment methodology". In K.R. Boff, L. Kaufman and J. Thomas, Handbook of perception and Human Performance: Volume II. Cognitive Processes and Performance (pp 42-1-42-9). New York: Wiley. 1986.

[PB1989] Parks, D.L., & Boucek, G.P., Jr. In G.R. McMillan, D. Beevis, E. Salas, M.H. Srub, R. Sutton, G L.V. Breda, Applications of Human Performance Models to System Design (pp 81-90). New York: Plenum. 1989.

[PPBBAW1994] Payne, D.G., Peters, L.J., Birkmire, D.P., Bonto, M.A., Anasti, J.S., & Wenger, M.J. In Human Factors, 36, (pp 441-475), 1994.

[PSGK1999] Patel, U.H., Salvendy, G., Geddels, L.A., & Kuczak, T. "An analog of Ohm's law for modeling mental workload". Chinese Journal of Industrial Engineering. In press. 1999.

[RGBNTE1988] Reid, G.B., & Nygren, T.E. "Subjective workload assessment technique: A scaling procedure for measuring mental workload". In P.A. Hancock and N. Meshkati, Human Mental Workload (pp 185-217). Amsterdam: Elsevier Science. 1988.

[SPMA1996] Shah, P., & Miyake, A. "The separability of working memory recourses for spatial thinking and language processing: An individual differences approach". Journals of Experimental Psychology: General, 125, (pp 4-27). 1996.

[SWL2001] Seagull, F.J., Wickens, C.D., & Loeb, R.G. "When is less more? Attention and workload auditory, visual and redundant patient-monitoring conditions". Proceedings of the 45th Annual Meeting of the Human Factors & Ergonomics Society. Santa Monica, CA: Human Factors & Ergonomics Society. 1985.

[TPSWGF1997] Tsang, P.S., & Wilson, G.F. "Mental workload measurements and analysis". In G. Salvendy, Handbook of Human Factors and Ergonomics (2nd Edn.). New York: Wiley. 1997.

[TV1996] Tsang, P.S., & Velazquez, V.L. "Diagnosticity and multidimensional subjective workload ratings". Ergonomics, 39(3) (pp358-381). 1996.

[WDC2002] Wickens, C.D. "Multiple recourses and performance prediction". Theoretical Issues in Ergonomics Science, Vol. 3, No. 2, (pp 159-177). Taylor & Francis. 2002.

[WDC1992] Wickens, C.D. In Engineering psychology and Human Performance 2nd ed., Champagne-Urbana, IL: Harper Collins Publishers Inc. 1992.

[WDC1976] Wickens, C.D. "The effects of divided attention on information processing in tracking". Journal of Experimental Psychology, Human Perception and performance, 2, (pp 1-13). 1976.

[WL1988] Wickens, C.D., & Liu, Y. "Codes and modalities in multiple recourses: a success and qualification". Human Factors, 30, (pp 599-616). 1988.

[WLWC1992] Weinstein, L.F., & Wickens, C.D. "Use of nontranditional flight displays for the reduction of central visual overload in the cockpit". International Journal of Aviation Psychology, 2, (pp 121-142). 1992.