ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΠΡΟΗΓΜΕΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ADVANCED COMPUTING SYSTEMS

Τίτλος Διπλωματικής Εργασίας

Συγκριτική Μελέτη Λειτουργικών Συστημάτων Πραγματικού Χρόνου στον Μικροελεγκτή ATmega328p

Γρηγόρης Παπαδόπουλος (ΑΜ: 21030)

Επιβλέπων Καθηγητής: Δρ. Στυλιανός Βουτσινάς

Αθήνα Ιούνιος 2023

Συγκριτική Μελέτη Λειτουργικών Συστημάτων Πραγματικού Χρόνου στον Μικροελεγκτή ATmega328p

Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου και του Εισηγητή

Η μεταπτυχιακή διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική επιτροπή:

Α/α	ΟΝΟΜΑ ΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	**Δρ. Στυλιανός Βουτσινάς**		Stylianos Voutsinas Digitally signed by Stylianos Voutsinas Date: 2023.07.05 12:08:20 +03'00'
2	**Δρ. Ιωάννης Βογιατζής**	**Καθηγητής**	Ioannis Voyiatzis Digitally signed by Ioannis Voyiatzis Date: 2023.07.11 05:55:50 +03'00'
3	**Δρ. Σταύρος Φατούρος**	**Αναπληρωτής Καθηγητής**	Stavros Fatouros Digitally signed by Stavros Fatouros Date: 2023.07.11 11:30:01 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Γρηγόρης Παπαδόπουλος του Ιωάννη, με αριθμό μητρώου 21030 φοιτητής του Προγράμματος Μεταπτυχιακών Σπουδών Προηγμένες Τεχνολογίες Υπολογιστικών Συστημάτων του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

*****_Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 0 μήνες από την κατάθεση και έπειτα από αίτηση μου στη Βιβλιοθήκη και έγκριση του επιβλέποντα καθηγητή._

Ο/Η Δηλών/ούσα

* Ονοματεπώνυμο /Ιδιότητα

Ψηφιακή Υπογραφή Επιβλέποντα

(Υπογραφή)

** Εάν κάποιος επιθυμεί απαγόρευση πρόσβασης στην εργασία για χρονικό διάστημα 6-12 μηνών (embargo), θα πρέπει να υπογράψει ψηφιακά ο/η επιβλέπων/ουσα καθηγητής/τρια, για να γνωστοποιεί ότι είναι ενημερωμένος/η και συναινεί. Οι λόγοι χρονικού αποκλεισμού πρόσβασης περιγράφονται αναλυτικά στις πολιτικές του Ι.Α. (σελ. 6):*

https://www.uniwa.gr/wp-content/uploads/2021/01/%CE%A0%CE%BF%CE%BB%CE%B9%CF%84%CE%B9%CE%BA%CE%B5%CC%81%CF%82_%CE%99%CE%B4%CF%81%CF%85%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%BF%CF%85%CC%81_%CE%91%CF%80%CE%BF%CE%B8%CE%B5%CF%84%CE%B7%CF%81%CE%B9%CC%81%CE%BF%CF%85_final.pdf

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες προς όλους όσους συνέβαλαν στην ολοκλήρωση της διπλωματικής εργασίας μου. Θέλω να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ. Στυλιανό Βουτσινά, για την καθοδήγηση, την υποστήριξη και την αφοσίωσή του κατά τη διάρκεια αυτής της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου για τη συνεχή τους υποστήριξη, την αγάπη τους και την κατανόησή τους καθ' όλη τη διάρκεια αυτής της πορείας μου.

ΠΕΡΙΛΗΨΗ

Η διπλωματική εργασία που παρουσιάζεται ασχολείται με τη συγκριτική μελέτη δύο λειτουργικών συστημάτων πραγματικού χρόνου στον μικροελεγκτή ATmega328p. Αρχικά, εξηγείται η έννοια των λειτουργικών συστημάτων πραγματικού χρόνου και τα πλεονεκτήματα που προσφέρουν σε σχέση με τα λειτουργικά συστήματα γενικού σκοπού. Τα λειτουργικά συστήματα πραγματικού χρόνου είναι σχεδιασμένα για να αντιμετωπίζουν εργασίες που πρέπει να εκτελούνται σε συγκεκριμένους χρόνους, ενώ τα λειτουργικά συστήματα γενικού σκοπού είναι περισσότερο ευέλικτα και δεν δίνουν τόσο μεγάλη έμφαση στην ακρίβεια των χρονικών περιορισμών. Στη συνέχεια, η εργασία παρουσιάζει μετρήσεις από δύο διαφορετικά λειτουργικά συστήματα πραγματικού χρόνου, το FreeRTOS και το ChibiOS. Οι μετρήσεις αξιολογούν την απόδοση των δύο αυτών λειτουργικών συστημάτων ως προς τη χρήση μνήμης και τον χρόνο ολοκλήρωσης των διεργασιών.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Λειτουργικά Συστήματα Πραγματικού Χρόνου, ATmega328p, Σύγκριση Επιδόσεων

ABSTRACT

The presented thesis deals with the comparative study of two real-time operating systems on the ATmega328p microcontroller. Initially, the concept of real-time operating systems and the advantages they offer over general-purpose operating systems are explained. Real-time operating systems are designed to handle tasks that must be performed at specific times, whereas general-purpose operating systems are more flexible and do not place as much emphasis on the accuracy of time constraints. Subsequently, the thesis presents measurements from two different real-time operating systems, FreeRTOS and ChibiOS. The metrics evaluate the performance of these two operating systems in terms of memory usage and process completion time.

KEYWORDS

Real-Time Operating Systems, Arduino Uno, ATmega328p, Performance Comparison

Περιεχόμενα

Κατάλογος Εικόνων

1. Εισαγωγή

Οι σύγχρονοι υπολογιστές αποτελούνται από έναν ή περισσότερους επεξεργαστές, κύρια μνήμη, δίσκους, εκτυπωτές, πληκτρολόγιο, οθόνη, διασυνδέσεις δικτύου και άλλες συσκευές εισόδου και εξόδου. Αυτό αποτελεί ένα πολύπλοκο σύστημα, το οποίο δεν είναι εφικτό για κάθε προγραμματιστή εφαρμογών να κατανοήσει λεπτομερώς πώς συνεργάζονται όλα αυτά τα συστατικά. Επιπλέον, η διαχείριση όλων αυτών των συστατικών και η χρήση τους με τον βέλτιστο τρόπο αποτελεί μία πολύπλοκη υπόθεση. Για αυτόν τον λόγο, οι υπολογιστές είναι εφοδιασμένοι με ένα επίπεδο λογισμικού που ονομάζεται **λειτουργικό σύστημα**, το οποίο αναλαμβάνει τη διαχείριση όλων αυτών των πόρων και παρέχει στα προγράμματα χρήστη ένα καλύτερο και πιο σαφές μοντέλο του υπολογιστή.[1]

Εικόνα 1: Τυπική θέση του λειτουργικού συστήματος σε ένα υπολογιστικό σύστημα [19]

Τα λειτουργικά συστήματα υπάρχουν για περισσότερα από πενήντα χρόνια. Στη διάρκεια αυτό του χρόνου, έχει αναπτυχθεί μια μεγάλη ποικιλία από αυτά όπως:

- Λειτουργικά συστήματα μεγάλων υπολογιστών
- Λειτουργικά συστήματα διακομιστών
- Λειτουργικά συστήματα πολυεπεξεργαστών
- Λειτουργικά συστήματα προσωπικών υπολογιστών
- Λειτουργικά συστήματα υπολογιστών χειρός
- Ενσωματωμένα λειτουργικά συστήματα
- Λειτουργικά συστήματα κόμβων αισθητήρων
- Λειτουργικά συστήματα πραγματικού χρόνου
- Λειτουργικά συστήματα έξυπνων καρτών

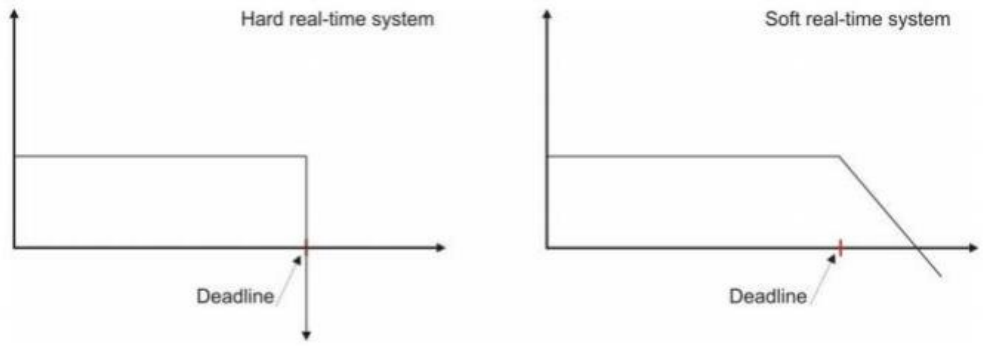
Στην παρούσα διπλωματική εργασία θα αναλυθούν τα λειτουργικά συστήματα πραγματικού χρόνου και θα γίνει σύγκριση του FreeRTOS και ChibiOS όσον αφορά την απόδοση τους πάνω στον μικροελεγκτή ATmega328p.

2. Λειτουργικά Συστήματα Πραγματικού Χρόνου (RTOS)

Τα υπολογιστικά συστήματα πραγματικού χρόνου διαδραματίζουν ζωτικό ρόλο στην κοινωνία μας. Ως λειτουργικό συστήματα πραγματικού χρόνου (RTOS) θεωρείται αυτό στο οποίο ο χρόνος παίζει πρωταρχικό ρόλο. Συνήθως, μια ή περισσότερες φυσικές συσκευές, οι οποίες βρίσκονται έξω από το υπολογιστικό σύστημα, δημιουργούν μια σειρά από ερεθίσματα στα οποία ο υπολογιστής πρέπει να αποκριθεί κατάλληλα μέσα σε σταθερό χρονικό διάστημα. Για παράδειγμα, στα βιομηχανικά συστήματα ελέγχου διεργασιών, υπολογιστές πραγματικού χρόνου πρέπει να συλλέγουν δεδομένα από την παραγωγική διαδικασία και να τα χρησιμοποιούν για να ελέγξουν μηχανές στο εργοστάσιο. Πολύ συχνά υπάρχουν χρονικές προθεσμίες που πρέπει να τηρούνται αυστηρά. Για παράδειγμα, αν ένα αυτοκίνητο έχει μπει στην αλυσίδα παραγωγής, πρέπει να γίνουν συγκεκριμένες ενέργειες σε συγκεκριμένες χρονικές στιγμές. Αν ένα ρομπότ που κάνει συγκολλήσεις ενεργήσει πρόωρα ή καθυστερημένα, το αυτοκίνητο θα καταστραφεί. Άλλα συστήματα πραγματικού χρόνου είναι αυτά που παρακολουθούν τους ασθενείς στις μονάδες εντατικές θεραπείας των νοσοκομείων και οι αυτόματοι πιλότοι των αεροπλάνων. Σε όλες αυτές τις περιπτώσεις, αν η σωστή απόκριση από την πλευρά του συστήματος φτάσει αργά, τα αποτελέσματα είναι συνήθως καταστροφικά όπως αυτά που θα προέκυπταν αν το σύστημα δεν έκανε καμία ενέργεια.[1]

Τα συστήματα πραγματικού χρόνου ανήκουν σε δύο είδη. Στα **αυστηρά συστήματα πραγματικού χρόνου** (hard real-time systems) και στα **ήπια συστήματα πραγματικού χρόνου** (soft real-time systems).

- Στα **αυστηρά συστήματα πραγματικού χρόνου**, οι ενέργειες πρέπει να εκτελούνται ακριβώς κατά τις προκαθορισμένες χρονικές στιγμές ή εντός συγκεκριμένων χρονικών περιθωρίων. Αυτά τα συστήματα χρησιμοποιούνται σε εφαρμογές όπως ο βιομηχανικός έλεγχος διεργασιών, η ηλεκτρονική των αεροσκαφών, η πολεμική βιομηχανία και σε συναφείς τομείς εφαρμογής. Τέτοια συστήματα πρέπει να εξασφαλίζουν την απόλυτη εγγύηση ότι κάθε ενέργεια θα εκτελεστεί σε συγκεκριμένο χρόνο.
- Στα **ήπια συστήματα πραγματικού** χρόνου η ακρίβεια των χρονικών περιορισμών δεν είναι τόσο κρίσιμη και μπορούν να υπάρχουν μικρές καθυστερήσεις στην εκτέλεση των εντολών χωρίς να επηρεαστεί η συνολική λειτουργία του συστήματος. Αυτά τα συστήματα είναι πιο συνηθισμένα στους τομείς της ψυχαγωγίας, των υπολογιστικών παιχνιδιών, των βίντεο και των ήχων σε πραγματικό χρόνο, καθώς και σε εφαρμογές επικοινωνίας και δικτύων.



Εικόνα 2: Οι προθεσμίες στα αυστηρά και στα ήπια λειτουργικά συστήματα πραγματικού χρόνου [18]

Για να επιτευχθεί η συμπεριφορά πραγματικού χρόνου, ο προγραμματισμός διαιρείται σε διάφορες διεργασίες οι οποίες έχουν προβλέψιμη και γνωστή εκ των προτέρων συμπεριφορά. Αυτές οι διεργασίες είναι συνήθως σύντομες και ολοκληρώνονται πλήρως σε πολύ σύντομα χρονικά διαστήματα, συνήθως μικρότερα από ένα δευτερόλεπτο. Όταν εντοπίζεται κάποιο εξωτερικό συμβάν, ο προγραμματιστής επιλέγει την κατάλληλη διεργασία προκειμένου να τηρηθούν όλες οι αντίστοιχες προθεσμίες.
Λόγω της κρισιμότητας της τήρησης αυστηρών προθεσμιών στα συστήματα πραγματικού χρόνου, σε ορισμένες περιπτώσεις το λειτουργικό σύστημα απλώς λειτουργεί ως βιβλιοθήκη που συνδέεται με τα προγράμματα εφαρμογών, όπου όλα είναι στενά συνδεδεμένα και δεν υπάρχει προστασία μεταξύ των διαφορετικών μερών του συστήματος. Το e-Cos αποτελεί ένα παράδειγμα αυτού του τύπου συστήματος πραγματικού χρόνου.[1]

Οι κατηγορίες των συστημάτων υπολογιστών χειρός, ενσωματωμένων συστημάτων και συστημάτων πραγματικού χρόνου έχουν σημαντική επικάλυψη μεταξύ τους. Σχεδόν όλα αυτά τα συστήματα διαθέτουν τουλάχιστον κάποιες λειτουργίες πραγματικού χρόνου. Στα ενσωματωμένα συστήματα και στα συστήματα πραγματικού χρόνου, η εκτέλεση του λογισμικού περιορίζεται στο λογισμικό των κατασκευαστών του συστήματος, με αποτέλεσμα να αυξάνεται η ασφάλεια. Τα συστήματα χειρός και ενσωματωμένα συστήματα κυρίως προορίζονται για καταναλωτές, ενώ τα συστήματα πραγματικού χρόνου κυρίως χρησιμοποιούνται στη βιομηχανία. Όλα αυτά τα συστήματα έχουν αρκετά κοινά στοιχεία.
Τα γεγονότα στα οποία το σύστημα πραγματικού χρόνου είναι υποχρεωμένο να αποκρίνονται, διαιρούνται περαιτέρω σε:

- **περιοδικά**, τα οποία συμβαίνουν σε τακτά χρονικά διαστήματα όπως για παράδειγμα ένας έλεγχος αυτοκινήτων για τους πεζούς κάθε 0,1 δευτερόλεπτο ή ένα σύστημα παρακολούθησης αέρα που παίρνει ένα δείγμα κάθε 10 δευτερόλεπτα.
- **απεριοδικά**, τα οποία προκύπτουν απρόβλεπτα όπως για παράδειγμα ο αερόσακος ενός αυτοκινήτου που πρέπει να αντιδράσει σε μία πρόσκρουση.[11]

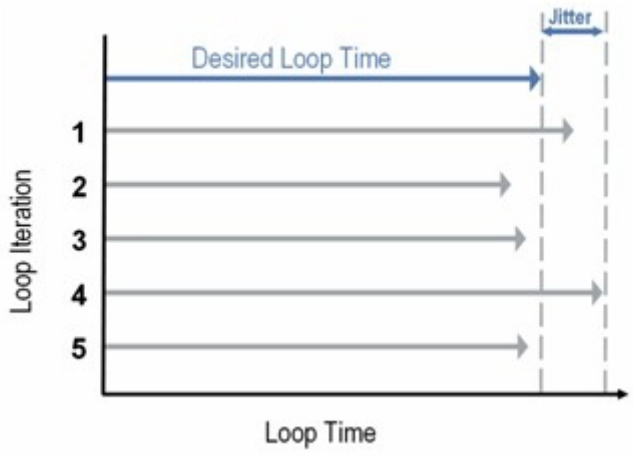
Ένα σύστημα μπορεί να αντιμετωπίζει πολλά διαφορετικά σενάρια και ανάλογα με την πολυπλοκότητα και την διάρκεια κάθε σεναρίου, ενδέχεται να μην είναι δυνατό να χειριστεί όλα τα συμβάντα που δημιουργούνται.

Τα συστήματα πραγματικού χρόνου που πληρούν αυτό το κριτήριο ονομάζονται **χρονοπρογραμματίσιμα**.
Για παράδειγμα, ας θεωρήσουμε ένα ήπιο σύστημα πραγματικού χρόνου στο οποίο υπάρχουν τρία περιοδικά συμβάντα, με περιόδους 100, 200, και 500 msec αντίστοιχα. Αν τα συμβάντα αυτά απαιτούν 50, 30, και 100 msec χρόνο CPU αντίστοιχα (σε κάθε εμφάνισή τους), το σύστημα είναι χρονοπρογραμματίσιμο, επειδή ότι 0,5 + 0,15 + 0,2 < 1. Αν προστεθεί ένα τέταρτο συμβάν με περίοδο 1 δευτερόλεπτο, το σύστημα παραμένει χρονοπρογραμματίσιμο εφόσον το συμβάν δεν απαιτεί περισσότερα από 150 msec χρόνο CPU σε κάθε εμφάνισή του. Στους υπολογισμούς αυτούς υποτίθεται έμμεσα ότι ο επιπλέον χρόνος που καταναλώνεται για να πραγματοποιηθούν οι θεματικές εναλλαγές είναι αρκετά μικρός ώστε να αγνοηθεί.

Ο χρονοπρογραμματισμός των συστημάτων πραγματικού χρόνου χωρίζεται σε δύο κατηγορίες αλγορίθμων, τους στατικούς και τους δυναμικούς. Οι στατικοί αλγόριθμοι λαμβάνουν τις αποφάσεις χρονοπρογραμματισμού πριν την εκκίνηση του συστήματος, χρησιμοποιώντας όλες τις απαραίτητες πληροφορίες και προθεσμίες που είναι διαθέσιμες εκ των προτέρων. Αντίθετα, οι δυναμικοί αλγόριθμοι λαμβάνουν αποφάσεις χρονοπρογραμματισμού κατά τη διάρκεια της λειτουργίας του συστήματος χωρίς να έχουν τις πληροφορίες που μπορούν να χρησιμοποιήσουν και τις προθεσμίες που πρέπει να τηρήσουν. Οι στατικοί αλγόριθμοι χρησιμοποιούνται μόνο όταν είναι διαθέσιμες όλες οι απαραίτητες πληροφορίες και προθεσμίες, ενώ οι δυναμικοί αλγόριθμοι μπορούν να λειτουργήσουν ακόμα και όταν αυτές οι πληροφορίες δεν είναι διαθέσιμες εκ των προτέρων.[1]

Ο χρόνος εκτέλεσης μιας διεργασίας μπορεί να ποικίλει εντός ορισμένων ορίων, ανάλογα με τον υπολογιστικό φόρτο κάθε περιόδου εργασίας. Σε ορισμένες περιπτώσεις, μπορεί να παρουσιαστεί μια **χρονική διαταραχή** ή αλλιώς **διακύμανση** που ονομάζεται **Jitter** και αφορά τη μεταβολή του χρόνου απόκρισης. Επιπλέον, ο χρόνος προετοιμασίας μιας διεργασίας μπορεί να μεταβληθεί μετά την αποδέσμευσή της. Όλες αυτές οι διαταραχές οδηγούν σε μεταβολή του χρόνου ολοκλήρωσης μιας διεργασίας εντός ορισμένων ορίων. Έτσι, ένας επαναλαμβανόμενος βρόχος ελέγχου δεν ολοκληρώνεται πάντα ακριβώς στον προκαθορισμένο χρόνο κάθε περιόδου, αλλά εντός ορισμένων παραμέτρων σφάλματος, γνωστών ως "response time jitter".[8]

Ένα αυστηρό λειτουργικό σύστημα πραγματικού χρόνου έχει χαμηλό Jitter, προσπαθώντας να διασφαλίσει ότι ο χρόνος απόκρισης για κάθε αίτημα παραμένει σταθερός και προβλέψιμος. Αντίθετα, ένα ήπιο λειτουργικό σύστημα πραγματικού χρόνου έχει υψηλότερο Jitter, καθώς επιτρέπει μια μεγαλύτερη μεταβλητότητα στον χρόνο απόκρισης.[6]



Εικόνα 3: Το Jitter (χρονική διαταραχή) σε 5 αιτήματα [8]

2.1 Βασικές προϋποθέσεις Λειτουργικών Συστημάτων Πραγματικού Χρόνου

Το λειτουργικό σύστημα διαχειρίζεται όλους τους διαθέσιμους πόρους του συστήματος, όπως η μνήμη και οι συσκευές εισόδου και εξόδου, για τη βέλτιστη διαχείριση τους. Καθώς οι διεργασίες χρειάζονται πόρους, το λειτουργικό σύστημα εξυπηρετεί τις αιτήσεις τους για τη χρήση αυτών των πόρων, ενώ παράλληλα ανταποκρίνεται σε ειδοποιήσεις για εξωτερικά γεγονότα. Αυτό επιτρέπει στο λειτουργικό σύστημα να ελέγχει και να διασφαλίζει την ασφάλεια και τη σταθερότητα του συστήματος, καθώς και τη βέλτιστη χρήση των πόρων του.[2]

Οι βασικές προϋποθέσεις ώστε ένα λειτουργικό σύστημα να μπορεί να θεωρηθεί ως λειτουργικό σύστημα πραγματικού χρόνου παρατίθενται παρακάτω:

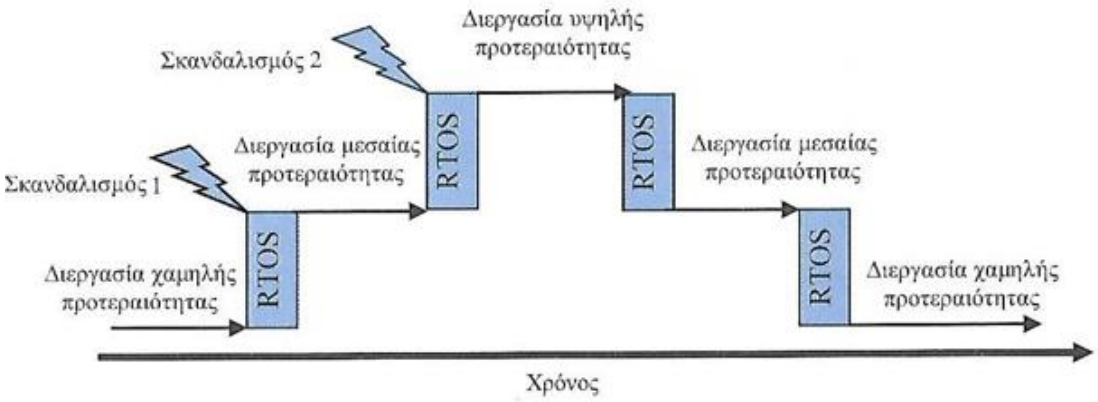
- **Πολυνηματισμός και δυνατότητα προεκτόπισης**

Για να επιτευχθεί η υποστήριξη πολλαπλών εργασιών σε εφαρμογές πραγματικού χρόνου, ένα RTOS πρέπει να είναι πολυνηματικό και προεκτοπιστικό. Αυτό σημαίνει ότι ο χρονοπρογραμματιστής του συστήματος πρέπει να μπορεί να αναστέλλει ένα νήμα σε

οποιοδήποτε σημείο και να παραχωρεί τον επεξεργαστή στο νήμα που έχει την υψηλότερη προτεραιότητα. Επιπλέον, το RTOS πρέπει να μπορεί να διαχειρίζεται τα πολλαπλά επίπεδα διακοπών, έτσι ώστε να μπορεί να διακόψει την εξυπηρέτηση ενός σήματος διακοπής για να μεταβεί στην εξυπηρέτηση ενός άλλου σήματος που θεωρείται πιο επείγον, όχι μόνο σε επίπεδο νημάτων αλλά και σε επίπεδο διακοπών.

- **Προτεραιότητα νημάτων**

Για να εκτοπιστεί μια διεργασία ή ένα νήμα και να αντικατασταθεί από ένα άλλο, το λειτουργικό σύστημα πρέπει να αποφασίσει ποιο από τα νήματα χρειάζεται περισσότερο έναν συγκεκριμένο πόρο, ο οποίος θα δοθεί στο νήμα με την πιο σημαντική προθεσμία. Ιδανικά, αυτή η απόφαση θα πρέπει να λαμβάνεται κατά τη διάρκεια της εκτέλεσης. Στην πράξη, όμως, οι αποφάσεις χρονοπρογραμματισμού δεν λαμβάνονται άμεσα με βάση τις προθεσμίες. Για να μπορέσει το λειτουργικό σύστημα να αντιμετωπίσει τις προθεσμίες, κάθε νήμα καθορίζεται με ένα επίπεδο προτεραιότητας. Οι πληροφορίες που δίνονται για τις προθεσμίες των νημάτων μετατρέπονται σε επίπεδα προτεραιότητας, τα οποία χρησιμοποιούνται από το λειτουργικό σύστημα για τη διανομή των πόρων του μεταξύ των νημάτων, με βάση αυτά τα επίπεδα προτεραιότητας. Παρόλο που αυτή η προσέγγιση μπορεί να οδηγήσει σε σφάλματα, όπως αδιέξοδα, είναι αναγκαία σε RTOS όπου δεν υπάρχει άλλη εφικτή λύση. Στο παρακάτω σχήμα, η άφιξη των σημάτων *σκανδαλισμού 1* και *σκανδαλισμού 2* οδηγεί στη διακοπή των εργασιών που εκτελούνται τη στιγμή της άφιξης του κάθε σήματος, προκειμένου να εκτελεστούν κάποιες εργασίες που έχουν μεγαλύτερη προτεραιότητα.



Εικόνα 4: Εκτέλεση εργασιών σε ένα σύστημα βασισμένο σε προτεραιότητες [2]

- **Προβλέψιμοι μηχανισμοί συγχρονισμού νημάτων**

Η ύπαρξη πολλαπλών νημάτων σε ένα πρόγραμμα απαιτεί τη χρήση μηχανισμών διαδιεργασιακής επικοινωνίας και συγχρονισμού μεταξύ τους. Αυτοί οι μηχανισμοί πρέπει να επιτρέπουν την πρόβλεψη του χρόνου που απαιτείται για την ολοκλήρωση των σχετικών ενεργειών μεταξύ των νημάτων και των διεργασιών. Επιπλέον, είναι απαραίτητο να υποστηρίζεται η δυνατότητα κλειδώματος-άρσης των κοινών πόρων, έτσι ώστε να διασφαλίζεται η ακεραιότητα των δεδομένων και να αποφεύγονται τα σφάλματα που μπορεί να προκληθούν από τον ανταγωνισμό μεταξύ των νημάτων για τους ίδιους πόρους.

- **Κληρονομιά προτεραιότητας**

Όταν χρησιμοποιείται ο χρονοπρογραμματισμός προτεραιοτήτων σε ένα RTOS, είναι σημαντικό να υπάρχει ένας επαρκής αριθμός επιπέδων προτεραιότητας, έτσι ώστε να μπορεί να διαχειριστεί όλες τις διαβαθμίσεις προτεραιότητας και να επιτρέπει στις εφαρμογές με αυστηρές απαιτήσεις προτεραιότητας να εκτελούνται εντός των προθεσμιών τους. Η **αντιστροφή προτεραιότητας** συμβαίνει όταν μια διεργασία υψηλής προτεραιότητας αναστέλλεται από μία διεργασία χαμηλότερης προτεραιότητας, και δεν μπορεί να εκτελεστεί από την κεντρική μονάδα επεξεργασίας. Η τεχνολογία RTOS μπορεί να αποτρέψει την αντιστροφή προτεραιότητας μέσω της μεθόδου της **κληρονομιάς προτεραιότητας**, δίνοντας στη διεργασία με χαμηλή προτεραιότητα την ίδια προτεραιότητα με την διεργασία υψηλότερης προτεραιότητας που έχει ανασταλεί. Στην περίπτωση αυτή, η διεργασία υψηλής προτεραιότητας που εμποδίζεται, μπορεί να ολοκληρωθεί χωρίς να επηρεαστεί από μια διεργασία με μέση προτεραιότητα. Είναι σημαντικό ο σχεδιαστής να διασφαλίζει ότι το RTOS που χρησιμοποιείται αποτρέπει την απεριόριστη αντιστροφή προτεραιότητας.

- **Είδη καθυστερήσεων σε λειτουργικά συστήματα πραγματικού χρόνου**

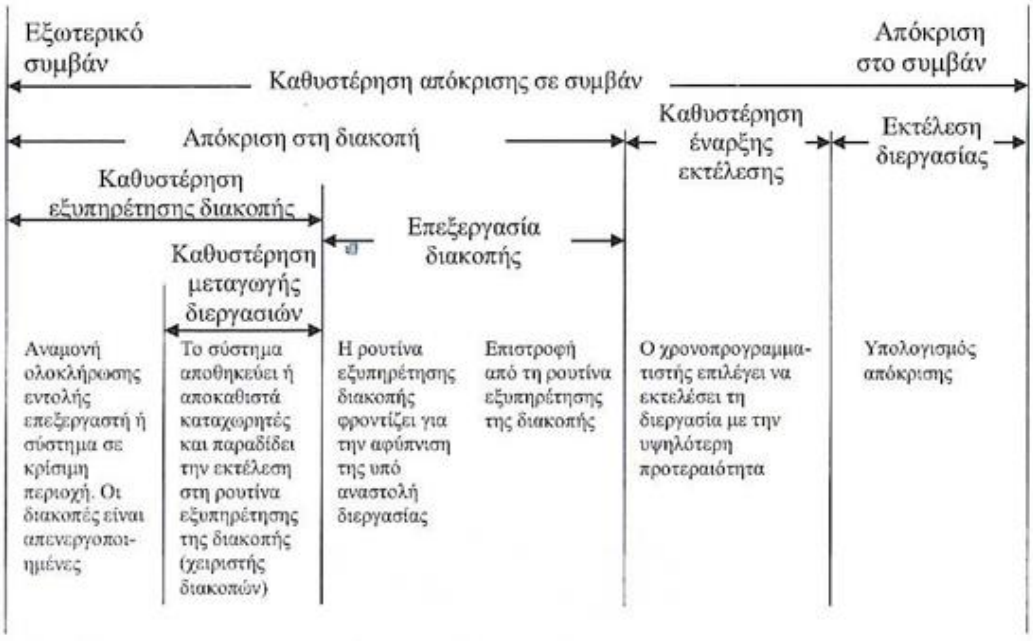
Για να υποστηρίζει εφαρμογές πραγματικού χρόνου, ένα λειτουργικό σύστημα πρέπει να έχει τις κατάλληλες πληροφορίες για τον συγχρονισμό των κλήσεων συστήματός του. Καθοριστικές μετρήσεις της απόκρισής του είναι οι χρονικές καθυστερήσεις, που αντικατοπτρίζουν τον χρόνο που χρειάζεται το σύστημα να ανταποκριθεί σε μια σειρά από περιστατικά. Οι βασικές μετρήσεις απόδοσης περιλαμβάνουν τα εξής:

- **Καθυστέρηση μεταγωγής διεργασιών**: Η καθυστέρηση μεταγωγής διεργασιών είναι ο χρόνος που απαιτείται για την αποθήκευση του περιβάλλοντος μιας εκτελούμενης

διεργασίας (όπως καταχωρητές και κατάσταση) και τη μετάβαση στην εκτέλεση μιας άλλης διεργασίας. Είναι σημαντικό αυτή η καθυστέρηση να είναι όσο το δυνατόν μικρότερη, καθώς μπορεί να επηρεάσει την απόκριση του συστήματος σε περιστάσεις που απαιτούν γρήγορη αλλαγή από μια διεργασία σε μια άλλη.

- **Καθυστέρηση εξυπηρέτησης διακοπής**: Είναι ο χρόνος που απαιτείται από την εμφάνιση ενός σήματος διακοπής μέχρι την εκτέλεση της πρώτης εντολής από τον χειριστή διακοπών ή μέχρι την εκτέλεση της εργασίας εξυπηρέτησης διακοπής, αν ο χειριστής διακοπών είναι ξεχωριστή εργασία. Αυτό είναι ένα μέτρο για το πόσο γρήγορα αντιδρά το σύστημα σε εξωτερικά γεγονότα, όπως μια διακοπή. Είναι σημαντικό αυτός ο χρόνος να είναι όσο το δυνατόν μικρότερος.
- **Καθυστέρηση έναρξης εκτέλεσης**: Είναι ο χρόνος που περνάει από την ολοκλήρωση της τελευταίας εντολής στον χειριστή διακοπών μέχρι την έναρξη της πρώτης εντολής της επόμενης διεργασίας που προγραμματίζεται να εκτελεστεί. Αυτή η καθυστέρηση αντικατοπτρίζει τον χρόνο που απαιτείται για τη μετάβαση του συστήματος από την εξυπηρέτηση μιας διακοπής στην εκτέλεση των εργασιών των χρηστών.
- **Καθυστέρηση απόκρισης σε συμβάν**. Είναι ο χρόνος που περνάει από τη στιγμή που δημιουργείται ένα εξωτερικό συμβάν μέχρι την ολοκλήρωση της εκτέλεσης του σχετικού κώδικα που αναλαμβάνει να χειριστεί το συμβάν και να παράξει την απόκριση σε αυτό.[2]

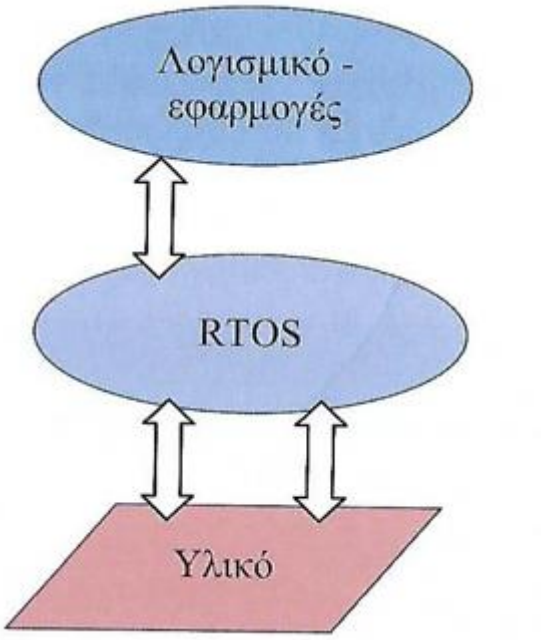
Το σχήμα απεικονίζει τα διάφορα είδη καθυστερήσεων σε ένα σύστημα πραγματικού χρόνου



Εικόνα 5: Τα διάφορα είδη καθυστερήσεων σε ένα σύστημα πραγματικού χρόνου [2]

2.2 Πυρήνες Πραγματικού Χρόνου

Ο πυρήνας είναι η καρδία του λειτουργικού συστήματος και είναι υπεύθυνος για τη διαχείριση των διεργασιών, το σχεδιασμό του προγράμματος χρονοπρογραμματισμού, τη διαδικασία επικοινωνίας μεταξύ των διεργασιών και του υλικού. Στα ενσωματωμένα συστήματα, ο πυρήνας μπορεί να λειτουργήσει ως ένα Real-Time Operating System (RTOS), ενώ στα εμπορικά RTOS, όπως αυτά που χρησιμοποιούνται στα συστήματα ελέγχου εναέριας κυκλοφορίας, απαιτείται η πλήρης λειτουργικότητα ενός γενικού σκοπού λειτουργικού συστήματος.[2]



Εικόνα 6: Ο πυρήνας παρέχει ένα ενδιάμεσο επίπεδο μεταξύ των εφαρμογών και του υλικού [2]

Ο πυρήνας έχει τον ρόλο της διαχείρισης των πόρων του συστήματος, όπως ο επεξεργαστής και η μνήμη, οι οποίοι πρέπει να διαμοιράζονται μεταξύ ανταγωνιζόμενων διεργασιών. Ο πυρήνας πρέπει να διαχειρίζεται αυτήν την διαμοίραση με προσοχή, ώστε να μεγιστοποιηθεί η απόδοση του επεξεργαστή και να εκτελούνται οι διαδικασίες με τον γρηγορότερο δυνατό τρόπο. Ο διαμοιρασμός των πόρων είναι απαραίτητος, καθώς οι πόροι είναι πεπερασμένοι και η διαχείριση του χρόνου είναι κρίσιμη για ένα λειτουργικό σύστημα πραγματικού χρόνου.

Τα λειτουργικά συστήματα πραγματικού χρόνου οφείλουν να παρέχουν τρεις ειδικές λειτουργίες αναφορικά με τις διεργασίες.

- Χρονοπρογραμματισμό
- Εκκίνηση διεργασίας σε επεξεργαστή
- Διαδιεργασιακή επικοινωνία και συγχρονισμό

Ο πυρήνας του λειτουργικού συστήματος είναι το μικρότερο δυνατό τμήμα που μπορεί να παρέχει αυτές τις λειτουργίες. Ο χρονοπρογραμματιστής καθορίζει ποια είναι η επόμενη διεργασία που πρέπει να εκτελεστεί, ενώ ο αποστολέας είναι υπεύθυνος για την προετοιμασία των απαραίτητων διαδικασιών προκειμένου να ξεκινήσει μια διεργασία. Η επικοινωνία μεταξύ διεργασιών και ο συγχρονισμός δίνουν την δυνατότητα στις διεργασίες να συνεργάζονται μεταξύ τους.

Στο παρακάτω σχήμα παρουσιάζονται οι τα διάφορα επίπεδα λειτουργίας ενός λειτουργικού συστήματος και η ταξινόμηση τους. Καθώς προχωράμε προς τα πάνω, από τον νανοπυρήνα προς το πλήρες λειτουργικό σύστημα, βλέπουμε την επιπλέον λειτουργικότητα που παρέχεται και τη σχετική εγγύτητα προς το υλικό σε σχέση με το χρήστη. Ένας **νανοπυρήνας** παρέχει τη διαχείριση απλών νημάτων, καθώς παρέχει μόνο μία από τις τρεις υπηρεσίες που παρέχει ένας πυρήνας, ενώ ένας **μικροπυρήνας** παρέχει επιπλέον τη δυνατότητα χρονοπρογραμματισμού διεργασιών. Ο πυρήνας επίσης παρέχει μηχανισμούς διαχείρισης και συγχρονισμού μεταξύ διεργασιών μέσω σηματοφόρων, γραμματοκιβωτίων και άλλων μεθόδων. Στο **εκτελεστικό επίπεδο πυρήνα πραγματικού χρόνου**, υπάρχουν επιπλέον χαρακτηριστικά όπως μπλοκ ιδιωτικής μνήμης και υπηρεσίες εισόδου-εξόδου. Οι περισσότεροι εμπορικοί πυρήνες πραγματικού χρόνου περιλαμβάνουν αυτά τα επίπεδα. Εκτός από τον πυρήνα, το λειτουργικό σύστημα παρέχει επίσης μια γενικευμένη διεπαφή χρήστη, ασφάλεια και ένα σύστημα διαχείρισης αρχείων.

	Χρήστες
Λειτουργικό σύστημα	Κέλυφος διεπαφής χρήστη
Εκτελεστικά επίπεδα πυρήνα	Υποστήριξη αρχείων και δίσκων
Πυρήνας	Διαδιεργασιακή επικοινωνία & συγχρονισμός
Νανοπυρήνας	Διαχείριση νημάτων
	Υλικό

Εικόνα 7: Ο ρόλος του πυρήνα στο λειτουργικό σύστημα [2]

Οι περισσότεροι πυρήνες πραγματικού χρόνου παρέχουν τις παρακάτω υπηρεσίες:

- **Εξυπηρέτηση στατικών και δυναμικών διεργασιών**: Οι στατικές διεργασίες είναι εκείνες που δημιουργούνται και ξεκινούν κατά την έναρξη μιας εφαρμογής λογισμικού, δεν αλλάζουν κατά τη διάρκεια της εκτέλεσης και παραμένουν διαθέσιμες

για ολόκληρη τη διάρκεια της εφαρμογής. Αντίθετα, οι δυναμικές διεργασίες μπορούν να δημιουργηθούν, να αλλάξουν ή να καταστραφούν κατά τη διάρκεια της εκτέλεσης μιας εφαρμογής. Συνήθως οι δυναμικές διεργασίες χρησιμοποιούνται για μια συγκεκριμένη βραχυπρόθεσμη υπηρεσία.

- **Ουρές και λίστες:** Συνήθως οι συναρτήσεις που παρέχονται για ουρές και λίστες είναι γενικευμένης μορφής και μπορούν να δεχτούν μια μεγάλη ποικιλία διαμορφώσεων. Για τις ουρές, μπορούν να υποστηρίξουν είτε τον αλγόριθμο *πρώτος μπαίνει - πρώτος βγαίνει* (FIFO) είτε τον αλγόριθμο *τελευταίος μπαίνει - πρώτος βγαίνει* (LIFO). Οι λίστες διακρίνονται σε *απλά συνδεδεμένες* και σε *διπλά συνδεδεμένες.*
- **Σηματοφόροι**: Οι σηματοφόροι είναι ένας τρόπος συγχρονισμού που χρησιμοποιείται για να διαχειρίζεται την πρόσβαση σε κοινούς πόρους του συστήματος από ανταγωνιζόμενες διεργασίες. Οι σηματοφόροι χρησιμοποιούνται για να ελέγχουν την πρόσβαση σε έναν πόρο από μια μόνο διεργασία τη φορά. Η χρήση σηματοφόρων εξασφαλίζει ότι οι ανταγωνιζόμενες διεργασίες έχουν πρόσβαση σε έναν κοινό πόρο με συγχρονισμένο τρόπο, χωρίς να υπάρχουν συγκρούσεις ή ανταγωνιστικές καταστάσεις.
- **Γραμματοκιβώτια**: Ένα γραμματοκιβώτιο λειτουργεί ως ένα ενδιάμεσο στοιχείο μεταξύ μιας διεργασίας που αποστέλλει και μιας διεργασίας που παραλαμβάνει, παρέχοντας μια διεπαφή για τη μεταφορά μηνυμάτων. Η αποστολή και η λήψη μηνυμάτων συμπεριφέρονται παρόμοια με τα πραγματικά ταχυδρομεία, όπου μόνο ένας αποδέκτης έχει πρόσβαση στα μηνύματα από το γραμματοκιβώτιο, αλλά πολλοί αποστολείς μπορούν να στείλουν μηνύματα σε αυτό. Τα γραμματοκιβώτια μπορούν να αποτελούνται από πολλές διεργασίες και να λαμβάνουν μηνύματα από πολλές εξωτερικές πηγές, ενώ τα μηνύματα μπορούν να αποσταλούν σε πολλά γραμματοκιβώτια.
- **Σύγχρονη και ασύγχρονη μετάδοση**: Η αποστολή ενός μηνύματος με σύγχρονο τρόπο σημαίνει ότι θα έχουμε αυτόματα αναμονή για επιβεβαίωση λήψης μηνύματος. Τυπικά, η διεργασία που στέλνει το μήνυμα θα περιμένει για σήμα επιβεβαίωσης εντός μιας συγκεκριμένης χρονικής περιόδου και αν δεν λάβει επιβεβαίωση εντός αυτής της περιόδου, τότε θα παραχθεί αναφορά σφάλματος. Είναι στην ευθύνη της εφαρμογής να χειριστεί το σφάλμα με τον κατάλληλο τρόπο. Στην ασύγχρονη μετάδοση ενός μηνύματος, η αποστολή γίνεται χωρίς να αναμένεται η επιβεβαίωση λήψης από τον παραλήπτη. Η αποστολή συνεχίζει αμέσως μετά την αποστολή του μηνύματος, χωρίς να αναμένει την επιβεβαίωση αυτού. Ωστόσο, η διαδικασία επιβεβαίωσης μπορεί να γίνει σε μεταγενέστερο σημείο, με την αποστολή μιας επιβεβαίωσης ή μιας αναφοράς

σφάλματος στην αποστέλλουσα διεργασία, η οποία θα πρέπει να το χειριστεί κατάλληλα.

- **Χρονομετρητές**: Ο πυρήνας διαθέτει χρονομετρητές που βασίζονται σε περιοδικές διακοπές, οι οποίες ενεργοποιούνται με βάση το ρολόι του συστήματος. Η περίοδος του χρονομετρητή μπορεί να προσαρμοστεί στις ανάγκες του συστήματος, αλλά μετά τη ρύθμιση δεν μπορεί να αλλάξει. Ωστόσο, είναι δυνατόν να δημιουργηθούν και άλλοι χρονομετρητές με διαφορετικές περιόδους. Οι διαθέσιμοι χρονομετρητές περιλαμβάνουν συνήθως *χρονομετρητές γενικής χρήσης*, *αντίστροφης μέτρησης* και *χρονομετρητές διανυθέντος χρόνου*. Οι χρονομετρητές γενικής χρήσης είναι χρονομετρητές που συγχρονίζουν μια διεργασία με ένα συμβάν που συμβαίνει μετά από έναν ορισμένο χρόνο. Οι χρονομετρητές αντίστροφης μέτρησης επιτρέπουν σε μια διεργασία να ανασταλεί από μια υπηρεσία πυρήνα για ένα περιορισμένο χρονικό διάστημα. Αυτός ο τύπος χρονομετρητή χρησιμοποιείται κυρίως με τις σύγχρονες μεταδόσεις. Τέλος, οι χρονομετρητές διανυθέντος χρόνου μετρούν το χρόνο που περνάει μεταξύ δύο συμβάντων.
- **Διαχείριση μνήμης**: Ο πυρήνας παρέχει μια μέθοδο για τη διαίρεση της διαθέσιμης μνήμης τυχαίας προσπέλασης (RAM) σε διάφορα τμήματα με διάφορο αριθμό και μέγεθος μπλοκ. Η διαχείριση της μνήμης ρυθμίζεται κατά την εκκίνηση της εφαρμογής και συνήθως δεν αλλάζει. Πολλοί πυρήνες χρησιμοποιούν μια απλά συνδεδεμένη λίστα για τη διαχείριση της μνήμης. Κατά τη διαδικασία εκχώρησης ενός μπλοκ σε μια διεργασία, το μπλοκ αφαιρείται από τις λίστες που καταγράφουν τη διαθέσιμη μνήμη και συνδέεται με τη διεργασία. Αντίθετα, κατά την ελευθέρωση ενός μπλοκ από μια διεργασία, το μπλοκ προστίθεται ξανά στην καταγραφή της διαθέσιμης μνήμης.

Ανεξαρτήτως του λειτουργικού συστήματος που χρησιμοποιείται, ο βασικός στόχος είναι να εξασφαλιστεί η ικανοποίηση των απαιτήσεων πραγματικού χρόνου και η παροχή ενός πολυδιεργασιακού περιβάλλοντος που είναι εύρωστο και ευέλικτο.

Σε ένα πολυδιεργασιακό περιβάλλον, ο πυρήνας πρέπει να παρέχει μια συστηματική μέθοδο ελέγχου μετάβασης από τη μία διεργασία στην άλλη και να χρησιμοποιεί αποτελεσματικά τους διαθέσιμους πόρους. Απαιτείται από τον πυρήνα να παρακολουθεί τους πόρους που χρησιμοποιούνται και την κατάσταση εκτέλεσης κάθε διεργασίας πριν από τη μετάβαση σε μια άλλη διεργασία, που συνήθως αναφέρεται ως **μεταγωγή πλαισίου**. Η επίτευξη της μεταγωγής πλαισίου σε σωστό χρόνο είναι ζωτικής σημασίας για τη σωστή λειτουργία του συστήματος, καθώς η καθυστέρηση της μετάβασης από μια διεργασία σε μια άλλη μπορεί να οδηγήσει σε προβλήματα όπως απώλεια προθεσμιών και αποτυχία του

συστήματος. Για να αποφευχθούν αυτά τα προβλήματα, κάθε διεργασία πρέπει να συνδεθεί με μια προτεραιότητα.[2]

2.3 Κατηγορίες Λειτουργικών Συστημάτων Πραγματικού Χρόνου

- **Συστήματα καθοδηγούμενα από σήματα διακοπής**

Στα συστήματα που λειτουργούν με σήματα διακοπής, το κύριο πρόγραμμα περιλαμβάνει μια εντολή που δημιουργεί μια αναμονή για την επόμενη διακοπή, δημιουργώντας έναν βρόχο αναμονής. Οι διάφορες διεργασίες στο σύστημα προγραμματίζονται μέσω σημάτων διακοπής που εκτελούνται από τις **ρουτίνες εξυπηρέτησης διακοπών** (interrupt service routines), ενώ η εκκίνηση των εργασιών που σχετίζονται με τα σήματα διακοπής γίνεται από τις ίδιες τις ρουτίνες εξυπηρέτησης. Όταν υπάρχει ανάγκη για διακοπές σε επίπεδο υλικού, μια εξωτερική συσκευή όπως ένα ρολόι αποστέλλει σήματα διακοπής σε έναν ελεγκτή διακοπών. Ο ελεγκτής αναλαμβάνει να επεξεργαστεί τα σήματα διακοπής και να παράγει τα κατάλληλα σήματα, λαμβάνοντας υπόψη τη σειρά άφιξης και την προτεραιότητα των ενεργοποιημένων σημάτων. Αν ο υπολογιστής υποστηρίζει πολλαπλά επίπεδα διακοπών, τότε ο ελεγκτής αναλαμβάνει τη διαδικασία της αποστολής των σημάτων στα κατάλληλα επίπεδα διακοπών. Σε περίπτωση που διατίθεται μόνο ένα επίπεδο διακοπών, η ρουτίνα διαχείρισης διακοπών πρέπει να ελέγχει τα διαθέσιμα δεδομένα από τον ελεγκτή διακοπών και να αντιμετωπίζει τις εκκρεμείς διακοπές με βάση τη σειρά προτεραιότητας τους. Ορισμένοι επεξεργαστές μπορούν να υλοποιήσουν τη διαδικασία αυτή σε μικροκώδικα, απαλλάσσοντας έτσι τον σχεδιαστή του λειτουργικού συστήματος από αυτό το καθήκον.[2]

- **Προεκτοπιστικά συστήματα προτεραιοτήτων**

Όταν μια διεργασία υψηλότερης προτεραιότητας διακόπτει μια διεργασία χαμηλότερης προτεραιότητας για να εξυπηρετηθεί πρώτη, λέμε ότι η πρώτη **προεκτοπίζει** τη δεύτερη. Τα συστήματα που χρησιμοποιούν αυτή τη μέθοδο αντί για τον απλό χρονοπρογραμματισμό εκ περιτροπής ή την εξυπηρέτηση των διεργασιών με βάση τη σειρά που έχουν φτάσει, λέγονται **συστήματα προεκτόπισης προτεραιοτήτων**. Οι προτεραιότητες που δίνονται σε κάθε διακοπή βασίζονται στη σημασία της εκτέλεσης της διεργασίας που συνδέεται με τη διακοπή. Για παράδειγμα, ένα σύστημα ελέγχου ενός πυρηνικού σταθμού ενέργειας θα πρέπει να σχεδιαστεί ως προεκτοπιστικό σύστημα προτεραιοτήτων. Αυτό σημαίνει ότι η διεργασία ψύξης του πυρηνικού αντιδραστήρα, η οποία έχει υψηλή προτεραιότητα, πρέπει να διακόπτει

οποιαδήποτε άλλη διεργασία στο σύστημα όταν είναι έτοιμη να τρέξει. Αυτό συμβαίνει όταν παρατηρηθεί ότι η θερμοκρασία στον αντιδραστήρα έχει ξεπεράσει κάποιο καθορισμένο όριο και στέλνεται ένα σήμα διακοπής στο σύστημα.

- **Συστήματα εξυπηρέτησης εκ περιτροπής**

Αυτή η μέθοδος χρησιμοποιείται όταν οι διεργασίες έχουν ίδια προτεραιότητα. Σε κάθε διεργασία ανατίθεται ένα μικρό χρονικό διάστημα, γνωστό ως χρονοθυρίδα ή κβάντο χρόνου, κατά το οποίο μπορούν να εκτελέσουν μια μερίδα της εργασίας τους πριν διακοπούν από μια άλλη διεργασία. Οι διεργασίες εναλλάσσονται μεταξύ τους και ολοκληρώνουν ένα τμήμα της εργασίας τους κάθε φορά. Αυτός ο αλγόριθμος ονομάζεται "round robin" ή "εκ περιτροπής". Με αυτόν τον τρόπο, επιλύονται οι συγκρούσεις μεταξύ διεργασιών με την ίδια προτεραιότητα.[17]

- **Υβριδικά συστήματα**

Τα υβριδικά συστήματα περιλαμβάνουν διακοπές που μπορεί να συμβαίνουν είτε με σταθερό ρυθμό (περιοδικές) είτε ασυνήθιστα (σποραδικές). Οι σποραδικές διακοπές μπορούν να είναι χρήσιμες για την αντιμετώπιση ενός σημαντικού γεγονότος που απαιτεί άμεση εξυπηρέτηση και είναι, συνεπώς, υψηλής προτεραιότητας. Αυτός ο τύπος συστήματος είναι συνηθισμένος στις ενσωματωμένες εφαρμογές.
Ένας διαφορετικός τύπος υβριδικού συστήματος που χρησιμοποιείται στα εμπορικά λειτουργικά συστήματα είναι ο συνδυασμός διαφορετικών συστημάτων χρονοπρογραμματισμού, όπως είναι η εξυπηρέτηση εκ περιτροπής και η προεκτοπιστική χρονοδρομολόγηση. Σε αυτόν τον τύπο συστημάτων, οι διεργασίες με υψηλότερη προτεραιότητα έχουν πάντα προτεραιότητα έναντι των διεργασιών με χαμηλότερη προτεραιότητα. Αντιθέτως, αν υπάρχουν δύο ή περισσότερες διεργασίες με την ίδια προτεραιότητα που είναι έτοιμες να εκτελεστούν ταυτόχρονα, τότε ακολουθείται το σχήμα εξυπηρέτηση εκ περιτροπής.

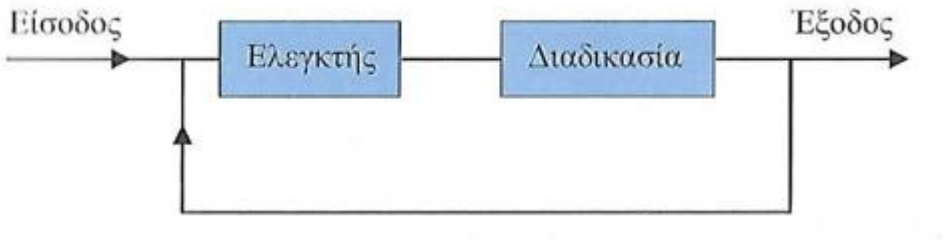
- **Συστήματα προσκηνίου-παρασκηνίου**

Τα συστήματα προσκηνίου-παρασκηνίου αποτελούν μια βελτίωση των συστημάτων που βασίζονται σε διακοπές. Αντί για τον άεργο βρόχο αναμονής για διακοπή, αυτά τα συστήματα περιλαμβάνουν κώδικα που εκτελεί **διεργασίες παρασκηνίου**. Υπάρχει επίσης ένα σύνολο διεργασιών που βασίζονται σε διακοπές ή πραγματικό χρόνο που εκτελούνται στο **προσκήνιο**.

Οι διεργασίες παρασκηνίου μπορούν να διακοπούν από οποιαδήποτε διεργασία προσκηνίου. Αυτό το είδος αρχιτεκτονικής είναι συνηθισμένο στις ενσωματωμένες εφαρμογές.[2]

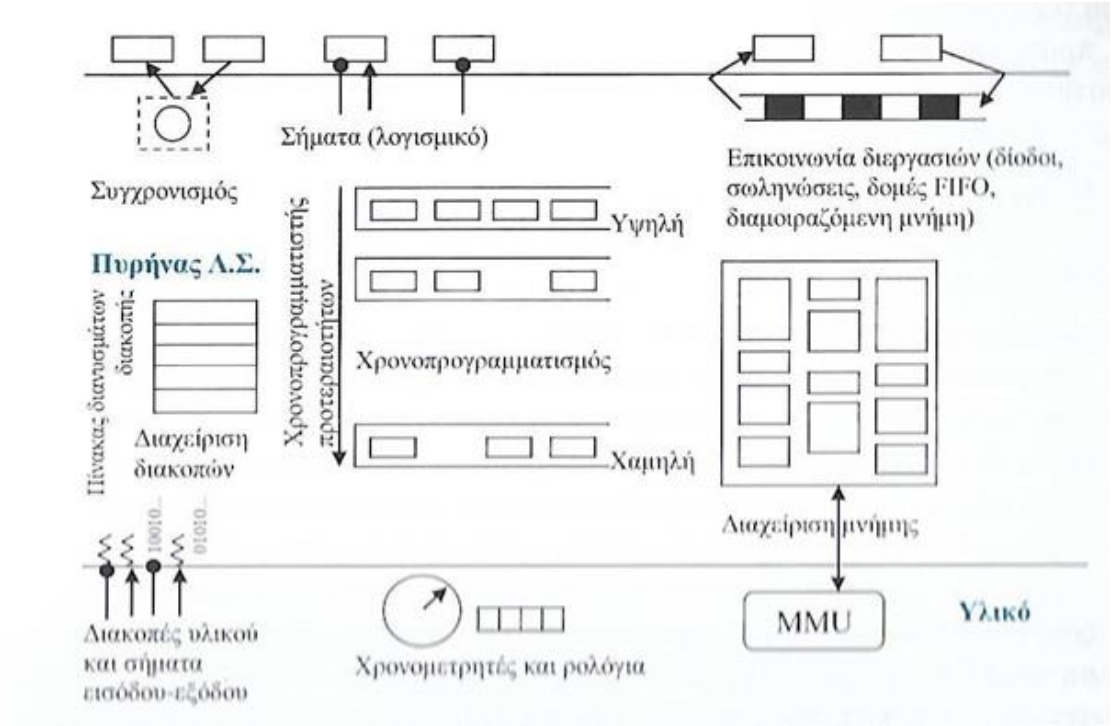
2.4 Σχεδιασμός λογισμικού

Η απλούστερη μορφή ενός συστήματος πραγματικού χρόνου αποτελείται από το υλικό του συστήματος. Αυτό επιτυγχάνεται με τη χρήση απλών συστημάτων ελέγχου με ανάδραση.[2]



Εικόνα 8: Απλό σύστημα ελέγχου με ανάδραση [2]

Ωστόσο, οι σχεδιαστές χρησιμοποιούν λειτουργικά συστήματα πραγματικού χρόνου στο υλικό όπου θα εκτελεστεί η εφαρμογή τους, προκειμένου να δημιουργήσουν, να διατηρήσουν και να υποστηρίξουν το περιβάλλον εκτέλεσης, στο οποίο θα λειτουργήσει η εφαρμογή. Το λειτουργικό σύστημα πραγματικού χρόνου (RTOS) αναλαμβάνει τη διαχείριση όλων των πόρων αποκλειστικά, αποτελεσματικά και αποδοτικά. Καθώς αυξάνεται η πολυπλοκότητα της εφαρμογής πραγματικού χρόνου, γίνεται πιο συμφέρουσα η χρήση ενός εμπορικού RTOS.
Το διάγραμμα παρουσιάζει τη δομή ενός RTOS και τα διάφορα στοιχεία του. Στο διάγραμμα περιλαμβάνονται τα βασικά στοιχεία του RTOS, όπως το υλικό που χρησιμοποιείται, ο πυρήνας του RTOS που εκτελείται πάνω στο υλικό, η εξυπηρέτηση των διεργασιών που εκτελούνται στο σύστημα και οι διακοπές που περιλαμβάνονται στην εφαρμογή πραγματικού χρόνου. Το λειτουργικό σύστημα πρέπει να παρέχει λειτουργίες διαχείρισης εργασιών, όπως χρονοπρογραμματισμό, αποστολή, δημιουργία και ολοκλήρωση εργασιών. Επιπλέον, πρέπει να παρέχει μηχανισμούς συγχρονισμού για τον διαμοιρασμό πόρων μεταξύ των διεργασιών, χειρισμό διακοπών για την εξυπηρέτηση διακοπών υλικού και διαχείριση μνήμης που περιλαμβάνει εικονική μνήμη και δυναμική κατανομή μνήμης. Τέλος, το λειτουργικό σύστημα πρέπει να παρέχει ρολόγια και χρονομετρητές που μπορούν να προγραμματιστούν και διαφορετικούς μηχανισμούς επικοινωνίας μεταξύ των διεργασιών, όπως διόδους, σωληνώσεις, δομές FIFO, διαμοιραζόμενη μνήμη κλπ.[2]



Εικόνα 9: Σχηματικό διάγραμμα δραστηριοτήτων ενός λειτουργικού συστήματος πραγματικού χρόνου [2]

2.5 Υλοποίηση συστημάτων με λειτουργικά συστήματα πραγματικού χρόνου

Η ανάπτυξη λειτουργικών συστημάτων πραγματικού χρόνου δεν βασίζεται μόνο στο λογισμικό των εφαρμογών, αλλά λαμβάνει υπόψη και το υλικό στο οποίο θα εκτελεστεί το σύστημα. Στον σχεδιασμό του υλικού λαμβάνονται υπόψη οι απαιτήσεις του RTOS, και αν το υλικό έχει σχεδιαστεί ειδικά για το RTOS αυτό, τότε ο σχεδιασμός και ο κώδικας του πυρήνα γίνονται πιο απλοί. Συνεπώς, η ανάπτυξη ενός λειτουργικού συστήματος πραγματικού χρόνου πρέπει να λαμβάνει υπόψη και το υλικό, καθώς αυτό επηρεάζει σημαντικά τον σχεδιασμό και την απόδοση του συστήματος. Η στενή σχέση του λογισμικού και του υλικού οδηγεί σε αυξημένη απόδοση και ταχύτητα του συστήματος, ειδικά σε εφαρμογές πραγματικού χρόνου που εκτελούνται σε ειδικά σχεδιασμένο υλικό. Αυτός ο τομέας της αγοράς αναπτύσσεται γρήγορα και συχνά απαιτεί να υπάρχει **δυνατότητα κλιμάκωσης** των πυρήνων του συστήματος.[2]

Για να υλοποιηθεί ένα σύστημα με χρήση ενός RTOS, απαιτείται σωστός υπολογισμός και προγραμματισμός. Ο σχεδιαστής πρέπει να αναλύσει όλες τις πτυχές του συγχρονισμού του συστήματος και να αξιολογήσει εάν το RTOS μπορεί να ικανοποιήσει τις απαιτήσεις του συστήματος βάσει των χρονικών υπολογισμών και της διαίρεσης των εργασιών. Επιπλέον, ο σχεδιαστής πρέπει να αξιολογήσει την προτεραιότητα των εργασιών, τη δυνατότητα πολυεπεξεργασίας και τη συμβατότητα της τεχνολογίας και του προϋπολογισμού με το επιλεγμένο RTOS. Τέλος, πρέπει να ληφθεί υπόψη η υποστήριξη των γλωσσών

προγραμματισμού από το επιλεγμένο RTOS. Παρακάτω παρατίθενται τα πιο σημαντικά στοιχεία που πρέπει να λάβει υπ' όψιν ο σχεδιαστής κατά την υλοποίηση ενός συστήματος RTOS.

- **Χρόνος απόκρισης:** Το χρονικό διάστημα που απαιτείται από ένα σύστημα για να ανταποκριθεί σε μια είσοδο και να παράξει μια έξοδο. Συγκεκριμένα, ο χρόνος απόκρισης του συστήματος πρέπει να είναι μικρότερος από τον ελάχιστο χρόνο μεταξύ διαδοχικών εισόδων, ώστε να εξασφαλιστεί η έγκαιρη ανταπόκριση του συστήματος σε νέες εισόδους. Συνεπώς, ο σχεδιασμός του συστήματος πρέπει να λάβει υπόψη του την απαιτούμενη ταχύτητα της απόκρισης και να εξασφαλίσει ότι η απόδοση του συστήματος είναι ικανοποιητική για τις ανάγκες της εφαρμογής.
- **Διαμέριση εργασιών:** Στον σχεδιασμό συστημάτων πραγματικού χρόνου, σημαντικό βήμα αποτελεί η διαίρεση της εφαρμογής σε εργασίες, όπου κάθε μία αντιστοιχεί σε μία εκτελέσιμη οντότητα. Αυτές οι εργασίες μπορούν να εκτελούνται είτε σε διαφορετικές διεργασίες, είτε σε νήματα, ανάλογα με τις δυνατότητες του υλικού και τις επιλογές του σχεδιαστή. Επίσης, είναι σημαντικό να ισορροπήσει ο σχεδιαστής τον παραλληλισμό και την επικοινωνία, καθώς η διάσπαση μιας εργασίας σε παράλληλα τμήματα μπορεί να αυξήσει τον όγκο των δεδομένων που πρέπει να ανταλλάσσονται ανάμεσα στα διαφορετικά τμήματα, με αποτέλεσμα να εισαχθεί καθυστέρηση. Όταν υπάρχει επικοινωνία ανάμεσα σε διαφορετικά τμήματα της εργασίας, υπάρχει κίνδυνος η καθυστέρηση στην επικοινωνία να καθυστερεί την παράλληλη εκτέλεση των τμημάτων. Έτσι, ο σχεδιαστής πρέπει να βρει τον καλύτερο τρόπο να χωρίσει την εργασία σε τμήματα, ώστε να ελαχιστοποιηθεί ο αριθμός των μεταγωγών πλαισίων και το μέγεθος των δεδομένων που ανταλλάσσονται μεταξύ τους. Οι σχεδιαστές χρησιμοποιούν **κριτήρια συνοχής εργασιών** για να βρουν τον καλύτερο τρόπο να διασπάσουν την εργασία σε παράλληλα τμήματα. Για παράδειγμα, δύο εργασίες δεν μπορούν να εκτελεστούν παράλληλα αν χρησιμοποιούν την ίδια περιοχή μνήμης.
- **Ζητήματα RTOS:** Μετά τον διαχωρισμό των εργασιών, ο σχεδιαστής πρέπει να αξιολογήσει εάν ένα RTOS μπορεί να διαχειριστεί όλες τις εργασίες. Οι βασικοί παράγοντες που πρέπει να εξεταστούν είναι οι περίοδοι χρήσης των χρονομετρητών, οι μέθοδοι επικοινωνίας μεταξύ διεργασιών, η αντιμετώπιση του ανταγωνισμού μεταξύ των διεργασιών και η προστασία της μνήμης.
- **Προτεραιότητα διεργασιών:** Η προτεραιότητα που ανατίθεται σε κάθε διεργασία είναι αναγκαία για την σωστή λειτουργία μιας εφαρμογής. Αυτό συμβαίνει επειδή υψηλής προτεραιότητας διεργασίες έχουν προτεραιότητα στην εκτέλεσή τους και, ανάλογα με την πολυπλοκότητα της εφαρμογής, οι χαμηλότερης προτεραιότητας

διεργασίες μπορεί να μην έχουν αρκετό χρόνο εκτέλεσης. Έτσι, ο σχεδιαστής πρέπει να καθορίσει ποιες εργασίες είναι κρίσιμες και να τους αναθέσει υψηλή προτεραιότητα, ενώ ταυτόχρονα να φροντίσει για τον αρκετό χρόνο εκτέλεσης των χαμηλότερης προτεραιότητας διεργασιών. Με αυτόν τον τρόπο, μπορεί να επιτευχθεί η αποτελεσματική διαχείριση των διεργασιών και η επίτευξη των προθεσμιών της εφαρμογής.

- **Διακοπές:** Κατά τον σχεδιασμό ενός συστήματος που χρησιμοποιεί σήματα διακοπής χωρίς προτεραιότητα, ο σχεδιαστής πρέπει να διασφαλίσει την ελαχιστοποίηση του χρόνου διαχείρισης διακοπών. Για τον σκοπό αυτό, ο χειριστής διακοπών πρέπει να αποθηκεύσει το τρέχον περιβάλλον εκτέλεσης, να δημιουργήσει μια διεργασία που θα εξυπηρετεί τη διακοπή και να επιστρέψει τον έλεγχο στο λειτουργικό σύστημα. Αν και στα συνήθη λειτουργικά συστήματα εκτελούμε τον κώδικα που εξυπηρετεί τη διακοπή μέσα στον πυρήνα αμέσως μόλις λαμβάνεται το σήμα διακοπής, στα λειτουργικά συστήματα πραγματικού χρόνου είναι χρήσιμο να δημιουργούμε μια νέα διεργασία για να ανταποκριθούμε στην άφιξη της διακοπής. Η συγκεκριμένη λύση επιλέγεται διότι όταν μια νέα διεργασία δημιουργείται, τότε αυτή έχει μια συγκεκριμένη προτεραιότητα και προθεσμίες που ορίζονται από τον σχεδιαστή και η εξυπηρέτηση διακοπής της διεργασίας ανταγωνίζεται με άλλες διεργασίες στο σύστημα, υπό τον έλεγχο ενός ενιαίου μηχανισμού χρονοπρογραμματισμού. Η δημιουργία νέων διεργασιών είναι απαραίτητη για να αποφευχθεί η πιθανότητα διακοπής κατά τη διάρκεια της εκτέλεσης μιας διεργασίας υψηλής προτεραιότητας από ένα χαμηλότερης προτεραιότητας γεγονός, το οποίο μπορεί να προκαλέσει παραβίαση των προθεσμιών. Σημαντικό είναι να μην χρησιμοποιούνται σήματα διακοπής χωρίς προτεραιότητα, όταν υπάρχει μια εργασία που δεν μπορεί να προεκτοπιστεί χωρίς την πρόκληση αστοχιών στο σύστημα. Στα συστήματα που χρησιμοποιούν διακοπές, ο σχεδιαστής πρέπει να λάβει υπόψη του τις περιόδους κατά τις οποίες το RTOS μπορεί να ελέγξει για νέες διακοπές και να αναλάβει τις αντίστοιχες ενέργειες. Συγκεκριμένα, κατά την εκτέλεση των εργασιών του συστήματος από το RTOS, οι διακοπές θα απενεργοποιηθούν, και η περίοδος κατά την οποία οι διακοπές θα είναι απενεργοποιημένες θα ονομάζεται **αναμονή διακοπής** του RTOS. Κατά τη διάρκεια αυτής της περιόδου, μπορεί να υπάρξουν καθυστερήσεις ή απώλεια διακοπών. Επομένως, είναι προτιμότερο να χρησιμοποιείται ένα RTOS με μικρή αναμονή διακοπής σε ένα σύστημα όπου η καθυστέρηση ή η απώλεια διακοπών δεν είναι αποδεκτή.
- **Πολυεπεξεργαστικά RTOS:** Στα συστήματα με ενσωματωμένους πολυεπεξεργαστές, υπάρχει ένας επεξεργαστής που ελέγχει κάθε συσκευή στο σύστημα και συνήθως χρησιμοποιούνται RTOS που υποστηρίζουν πολυεπεξεργασία και έχουν ένα

ξεχωριστό στιγμιότυπο του πυρήνα σε κάθε επεξεργαστή. Η δυνατότητα πολυεπεξεργασίας βασίζεται στη δυνατότητα του πυρήνα να ανταλλάσσει δεδομένα μεταξύ των επεξεργαστών. Από την άποψη των εργασιών, σε πολλά RTOS που υποστηρίζουν πολυεπεξεργασία, δεν υπάρχει διαφορά στον τρόπο λειτουργίας αν υπάρχει ένας ή περισσότεροι επεξεργαστές. Το RTOS αποθηκεύει έναν πίνακα στον τοπικό πυρήνα του συστήματος, ο οποίος περιλαμβάνει τις θέσεις κάθε εργασίας στο σύστημα, ώστε να γνωρίζει σε ποιον επεξεργαστή εκτελείται κάθε μία από αυτές. Όταν μια εργασία αποστέλλει ένα μήνυμα σε μια άλλη εργασία, ο τοπικός πυρήνας βρίσκει τη θέση της εργασίας στον πίνακα και κατευθύνει το μήνυμα στον κατάλληλο παραλήπτη. Από την πλευρά της εργασίας, οι ενέργειες εκτελούνται όπως θα έκαναν και σε ένα σύστημα με έναν μοναδικό επεξεργαστή. Με άλλα λόγια, από την πλευρά του χρήστη, η πολυεπεξεργασία στο RTOS φαίνεται να λειτουργεί σαν να είχε έναν μοναδικό επεξεργαστή.

- **Υποστήριξη γλωσσών προγραμματισμού:** Το RTOS πρέπει να διευκολύνει τον προγραμματιστή με τη διαχείριση των διαθέσιμων πόρων με τη μείωση του φόρτου κωδικοποίησης. Μια γλώσσα που υποστηρίζει άμεσα βασικές λειτουργίες συγχρονισμού, όπως οι εντολές SCHEDULE, SIGNAL και WAIT, απλοποιεί τη μετάβαση από το σχεδιασμό στον τελικό κώδικα. Η εντολή SCHEDULE προγραμματίζει μια διεργασία βασιζόμενη σε χρόνο ή σε ένα γεγονός. Οι εντολές SIGNAL και WAIT χρησιμοποιούν έναν σηματοφόρο για να χειριστούν τον συγχρονισμό παράλληλων εργασιών. Λόγω των περιορισμών που ισχύουν στα λειτουργικά συστήματα πραγματικού χρόνου, όπως αναφέρθηκε προηγουμένως, η ανάπτυξη ενός τέτοιου συστήματος είναι πολύ πιο δύσκολη σε σχέση με συνήθη λειτουργικά συστήματα. Επιπλέον, οι προγραμματιστές λογισμικού πραγματικού χρόνου αντιμετωπίζουν πολλά προβλήματα κατά την εύρεση σφαλμάτων ή λαθών. Για την αντιμετώπιση αυτών των προβλημάτων, οι προγραμματιστές χρησιμοποιούν βοηθητικά εργαλεία, όπως εργαλεία αποσφαλμάτωσης παρασκηνίου, λογικούς αναλυτές και προσομοιωτές. Κάθε ένα από αυτά τα εργαλεία έχει γνώση του πως λειτουργεί ο πυρήνας του συστήματος, και αυτό τα καθιστά πολύ χρήσιμα.[2]

2.6 POSIX

Το POSIX είναι μια ομάδα προτύπων συμμόρφωσης για υπηρεσίες λειτουργικών συστημάτων, σχεδιασμένο έτσι ώστε οι εφαρμογές που ακολουθούν αυτά τα πρότυπα να μπορούν να μεταφερθούν εύκολα μεταξύ διαφορετικών λειτουργικών συστημάτων που επίσης ακολουθούν τα ίδια πρότυπα.
Η IEEE έχει καθιερώσει το όνομα POSIX ως συντομογραφία του Portable Operating System Interface, που αναφέρεται στη **διεπαφή λειτουργικών συστημάτων με δυνατότητα μεταφερσιμότητας**. Το <X> στο τέλος του ονόματος δείχνει ότι αυτό το πρότυπο έχει κληρονομήσει στοιχεία από το UNIX, μια τεχνολογία που χρονολογείται από τη δεκαετία του 1970. Η ομάδα του POSIX πρότεινε αρχικά το πρότυπο 1003.1, το οποίο αφορά ένα σύνολο από διαδικασίες βιβλιοθήκης που πρέπει να παρέχει οποιοδήποτε λειτουργικό σύστημα που ακολουθεί το πρότυπο. Αυτές οι διαδικασίες ορίζουν ποιες παράμετροι μεταβιβάζονται σε αυτές, τι κάνουν οι διαδικασίες και ποια αποτελέσματα επιστρέφουν. Το πρότυπο POSIX είναι σχεδιασμένο έτσι ώστε τα προγράμματα εφαρμογών που ακολουθούν αυτό το πρότυπο να είναι εύκολο να μεταφερθούν μεταξύ διαφορετικών λειτουργικών συστημάτων που επίσης ακολουθούν αυτό το πρότυπο. Ο χειρισμός των προγραμμάτων που χρησιμοποιεί ο χρήστης με τον πυρήνα του λειτουργικού συστήματος γίνεται μέσω μιας βιβλιοθήκης και όχι μέσω κλήσεων συστήματος. Επιπλέον, οι πιο πρόσφατες εκδόσεις του POSIX περιλαμβάνουν επεκτάσεις που αφορούν την πραγματικό χρόνο και τον πολυνηματισμό, με αποτέλεσμα να επιτρέπουν στα συμβατά με POSIX συστήματα να χρησιμοποιούνται ευρέως στις εφαρμογές πραγματικού χρόνου.[2]

- **Πρότυπα POSIX**

Το POSIX είναι μια σειρά από πάνω από 30 ανεξάρτητα πρότυπα που καλύπτουν διάφορα θέματα σχετικά με τη λειτουργία των λειτουργικών συστημάτων. Αυτά τα πρότυπα περιλαμβάνουν προδιαγραφές για βασικές εφαρμογές των λειτουργικών συστημάτων, καθώς και προδιαγραφές για τον έλεγχο της συμμόρφωσης ενός λειτουργικού συστήματος σε ένα πρότυπο. Κάποια από τα πρότυπα του POSIX είναι σχετικά με τα συστήματα πραγματικού χρόνου, και θα παρουσιαστούν εν συντομία στη συνέχεια:[2]

- **1003.1a:** Βασικοί ορισμοί. Προσδιορίζει βασικές διεπαφές λειτουργικών συστημάτων, υποστηρίζει απλές διεργασίες και πολλαπλές διεργασίες, σήματα ομάδες χρηστών, συστήματα αρχείων, συσκευές εισόδου-εξόδου.

- **1003.1b:** Επεκτάσεις πραγματικού χρόνου. Παρέχει επεκτάσεις πραγματικού χρόνου, δηλαδή παρέχει συναρτήσεις που χρειάζονται για λειτουργικά συστήματα πραγματικού χρόνου, συμπεριλαμβάνοντας σήματα πραγματικού χρόνου, ασύγχρονες λειτουργίες εισόδου – εξόδου, προτεραιότητες εισόδου – εξόδου, κλείδωμα μνήμης, προστασία μνήμης, διανομή μηνυμάτων, σηματοφόρους και κοινή μνήμη.
- **1003.1c:** Συναρτήσεις νημάτων για υποστήριξη πολλαπλών νημάτων μέσα σε μία διεργασία. Περιλαμβάνει υποστήριξη για έλεγχο νημάτων, συμπεριφορά νημάτων, μεταβλητές αμοιβαίου αποκλεισμού, κληρονομιά προτεραιότητας μέσω των μεταβλητών αμοιβαίου αποκλεισμού, καθώς και μεταβλητές συνθήκης.
- **1003.1d:** Πρόσθετες επεκτάσεις πραγματικού χρόνου**.** Παρέχει υποστήριξη εκτός των άλλων για αποτελεσματική δημιουργία διεργασιών, χρονοπρογραμματισμό σποραδικών διεργασιών από τον εξυπηρετητή, παρακολούθηση χρόνων εκτέλεσης για διεργασίες και νήματα, έλεγχο διακοπών και έλεγχο συσκευών.
- **1003.1j:** Προχωρημένες επεκτάσεις πραγματικού χρόνου**.** Παρέχει ακόμα περισσότερες συναρτήσεις, συμπεριλαμβάνοντας υποστήριξη για συγχρονισμό και κλείδωμα για ανάγνωση – εγγραφή.
- **1003.h:** Περιλαμβάνει προβλέψεις για τήρηση ημερολογίου και ενημερώσεις για σφάλματα και συμβάντα, ρύθμιση της δημιουργίας αρχείου αποτύπωσης μνήμης, τερματισμό λειτουργίας και επανεκκίνηση του υπολογιστικού συστήματος καθώς και διαχείριση της διαμόρφωσης του συστήματος.

➢ **<u>Συμβατότητα POSIX</u>**

Το πρότυπο IEEE POSIX 1003.1b καθορίζει τα πρότυπα συμβατότητας για τις υπηρεσίες των Real-Time Operating Systems (RTOS) και στοχεύει στη δυνατότητα των προγραμματιστών να αναπτύσσουν εφαρμογές που είναι εύκολο να χρησιμοποιηθούν σε διάφορα λειτουργικά συστήματα. Το πρότυπο καλύπτει τις βασικές υπηρεσίες των RTOS, και συγκεκριμένα:

- **Σύγχρονη είσοδος-έξοδος:** Η "σύγχρονη είσοδος-έξοδος" είναι μια συνήθης μέθοδος διαχείρισης των διαδικασιών εισόδου-εξόδου στα λειτουργικά συστήματα. Σε αυτήν τη μέθοδο, όταν μια διεργασία ζητάει μια εργασία εισόδου-εξόδου, η διεργασία τίθεται σε αναστολή μέχρις ότου η εργασία ολοκληρωθεί. Αυτό σημαίνει ότι η διεργασία περιμένει μέχρι να είναι διαθέσιμος ο απαιτούμενος πόρος ή η υπηρεσία εισόδου-εξόδου, και στη συνέχεια η διεργασία επανέρχεται στην εκτέλεσή της. Αυτό εξασφαλίζει ότι η εκτέλεση της διεργασίας είναι συγχρονισμένη με την εκτέλεση των

υπηρεσιών εισόδου-εξόδου και μειώνει τον κίνδυνο σφαλμάτων και αστάθειας στο σύστημα.

- **Ασύγχρονη είσοδος-έξοδος:** Η ασύγχρονη είσοδος-έξοδος αφορά τη διαδικασία εισόδου-εξόδου όπου η διεργασία δεν τίθεται σε αναστολή κατά τη διάρκεια εκτέλεσής της, όταν αιτείται μια διαδικασία εισόδου-εξόδου. Αντίθετα, συνεχίζει την εκτέλεσή της και το λειτουργικό σύστημα ενημερώνει τη διεργασία μέσω ενός σήματος για την ολοκλήρωση της διαδικασίας εισόδου-εξόδου. Αυτό σημαίνει ότι η διεργασία δεν περιμένει να ολοκληρωθεί η διαδικασία εισόδου-εξόδου, αλλά συνεχίζει την εκτέλεσή της και αντιδρά μόνο όταν λαμβάνει το αντίστοιχο σήμα από το λειτουργικό σύστημα. Η ασύγχρονη είσοδος-έξοδος βελτιώνει την απόδοση του συστήματος, αλλά αυξάνει την πολυπλοκότητα του προγραμματισμού.
- **Κλείδωμα μνήμης:** Το κλείδωμα μνήμης είναι μια δυνατότητα του λειτουργικού συστήματος να διατηρεί συγκεκριμένα τμήματα μνήμης μιας διεργασίας στη μνήμη, ώστε να μπορούν να προσπελαύνονται γρήγορα όταν απαιτείται από τη διεργασία. Τα κλειδωμένα τμήματα δεν μπορούν να αντικατασταθούν από άλλες διεργασίες ή να αποθηκευτούν σε συσκευές δευτερεύουσας μνήμης, εκτός αν απελευθερωθούν από τη διεργασία που τα κλείδωσε.
- **Σηματοφόροι:** Οι σηματοφόροι είναι δομές του λειτουργικού συστήματος που χρησιμοποιούνται για τον συγχρονισμό της πρόσβασης σε πόρους από πολλαπλές διεργασίες. Μέσω των σηματοφόρων, οι διαδικασίες μπορούν να επικοινωνούν και να συγχρονίζονται μεταξύ τους, ακόμα και αν εκτελούνται σε διαφορετικούς χώρους διευθύνσεων ή σε διαφορετικά νήματα μέσα στον ίδιο χώρο διευθύνσεων. Οι σηματοφόροι διατίθενται στο POSIX σε δύο τύπους: δυαδικούς (binary) σηματοφόρους που μπορούν να λάβουν μόνο τις τιμές 0 και 1 και σηματοφόρους απαρίθμησης (counting) που μπορούν να λάβουν οποιαδήποτε θετική ακέραια τιμή.
- **Διαμοιραζόμενη μνήμη:** Η διαμοιραζόμενη μνήμη είναι μια δυνατότητα των λειτουργικών συστημάτων που επιτρέπει την πρόσβαση πολλαπλών διεργασιών σε μια κοινή περιοχή της φυσικής μνήμης. Οι διεργασίες μπορούν να αλληλοεπιδρούν και να επικοινωνούν μεταξύ τους χρησιμοποιώντας αυτήν τη διαμοιραζόμενη περιοχή μνήμης. Συνήθως, αυτό επιτυγχάνεται με την απεικόνιση της διαμοιραζόμενης περιοχής μνήμης στον χώρο διευθύνσεων της κάθε διεργασίας, χρησιμοποιώντας τον μηχανισμό εικονικής μνήμης.
- **Χρονοπρογραμματισμός εκτέλεσης:** Ο χρονοπρογραμματισμός εκτέλεσης αναφέρεται στη δυνατότητα να διαμορφωθεί ένα πρόγραμμα που καθορίζει τον τρόπο με τον οποίο οι διάφορες εργασίες θα εκτελούνται στο σύστημα. Αυτό περιλαμβάνει

την εξυπηρέτηση πολλαπλών εργασιών με μέθοδο εκ περιτροπής ή προεκχωρητικό προγραμματισμό βασισμένο στην προτεραιότητα.

- **Χρονομετρητές:** Οι χρονομετρητές βελτιώνουν τη λειτουργικότητα και την ακρίβεια του συστήματος. Για να παρέχει ικανοποιητικές υπηρεσίες πραγματικού χρόνου, ένα σύστημα πρέπει να διαθέτει τουλάχιστον μια συσκευή ρολογιού, γνωστή ως ρολόι συστήματος. Στα συστήματα πραγματικού χρόνου POSIX, αυτό το ρολόι καλείται *ρολόι πραγματικού χρόνου*.
- **Διαδιεργασιακή επικοινωνία:** Η διαδιεργασιακή επικοινωνία είναι ένας μηχανισμός που επιτρέπει στις διαφορετικές εργασίες μιας εφαρμογής να ανταλλάσσουν δεδομένα για την επίτευξη των στόχων της εφαρμογής. Στα RTOS, οι συνήθεις μέθοδοι επικοινωνίας περιλαμβάνουν τα γραμματοκιβώτια και τις ουρές. Οι εργασίες μπορούν να αποστείλουν και να λαμβάνουν μηνύματα σε αυτούς τους μηχανισμούς επικοινωνίας και να συνεργάζονται για την επίτευξη των στόχων της εφαρμογής.
- **Αρχεία πραγματικού χρόνου:** Τα αρχεία πραγματικού χρόνου είναι αρχεία στα οποία ο χρόνος πραγματοποίησης των λειτουργιών ανάγνωσης και εγγραφής είναι αιτιοκρατικός, δηλαδή καθορίζεται από το σύστημα πραγματικού χρόνου. Αυτό επιτυγχάνεται μέσω μηχανισμών που εξασφαλίζουν την εκτέλεση των εντολών ανάγνωσης και εγγραφής στο αρχείο σε προκαθορισμένες χρονικές στιγμές. Αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές πραγματικού χρόνου, όπου οι αποκρίσεις πρέπει να είναι αμεσότερες και πιο αξιόπιστες.
- **Νήματα πραγματικού χρόνου:** Νήματα πραγματικού χρόνου αποτελούν τμήματα μιας εφαρμογής πραγματικού χρόνου που μπορούν να προγραμματιστούν και να εκτελεστούν με συγκεκριμένους περιορισμούς χρόνου. Μπορούν να ενταχθούν σε μία ομάδα εκτελέσιμων νημάτων, που επιτρέπει την εφαρμογή συλλογικών περιορισμών χρόνου. Οι προδιαγραφές του προτύπου καθορίζουν τη διεπαφή προγραμματισμού για τη δημιουργία, έλεγχο και συγχρονισμό των νημάτων, καθώς και τη λειτουργικότητα που πρέπει να προσφέρει μια βιβλιοθήκη νημάτων σε ένα λειτουργικό σύστημα που συμμορφώνεται με το πρότυπο POSIX.

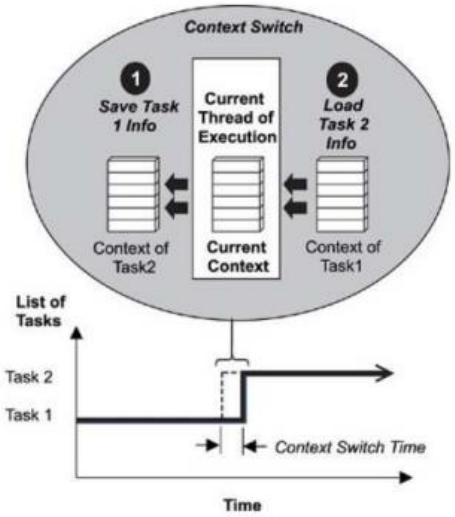
2.7 Η Διαφορά των Λειτουργικών Συστημάτων Πραγματικού Χρόνου σε σχέση με τα Λειτουργικά Συστήματα Γενικού Σκοπού

Οι απαιτήσεις χρόνου έχουν καταδείξει την ανάγκη χρήσης συγκεκριμένων ενσωματωμένων λειτουργικών συστημάτων. Τα λειτουργικά συστήματα πραγματικού χρόνου (RTOS) και τα λειτουργικά συστήματα γενικού σκοπού (GPOS) παρέχουν παρόμοιες υπηρεσίες, αλλά διαφέρουν στον τρόπο υλοποίησής τους. Η κύρια διαφορά ανάμεσά τους είναι ο **ντετερμινισμός** που έχουν τα RTOS, όπου οι υπηρεσίες τους καταναλώνουν μόνο προκαθορισμένο χρόνο και μετά το πέρας του χρόνου αυτού έχουμε σφάλμα.
Στα GPOS, οι υπηρεσίες μπορεί να προκαλέσουν τυχαίες καθυστερήσεις στις εφαρμογές, οπότε η απόκριση σε απρόβλεπτους χρόνους είναι αργή. Αντίθετα, στα RTOS, αυτές οι τυχαίες καθυστερήσεις είναι απαγορευτικές. Οι υπηρεσίες των RTOS πρέπει να χρησιμοποιούν γνωστά και αναμενόμενα χρονικά διαστήματα. Για να γίνει αυτό, τα χρονικά αυτά διαστήματα πρέπει να μπορούν να περιγραφούν με αυστηρά αλγεβρικούς τύπους.[17]
Είναι σημαντικό να σημειωθεί ότι ο όρος "πραγματικός χρόνος" δεν αναφέρεται απαραιτήτως στην ταχύτερη εξυπηρέτηση ή σε φυσικούς χρονισμούς, αλλά στην ικανοποίηση συγκεκριμένων χρονικών προθεσμιών. Ο ντετερμινισμός εισάγεται στα RTOS με τον τρόπο χειρισμού των εργασιών, της διαδικασίας επικοινωνίας και συγχρονισμού και της δυναμικής διαχείρισης μνήμης, προκειμένου να εκπληρωθούν οι χρονικές προθεσμίες.

2.8 Η Εναλλαγή των Διεργασιών

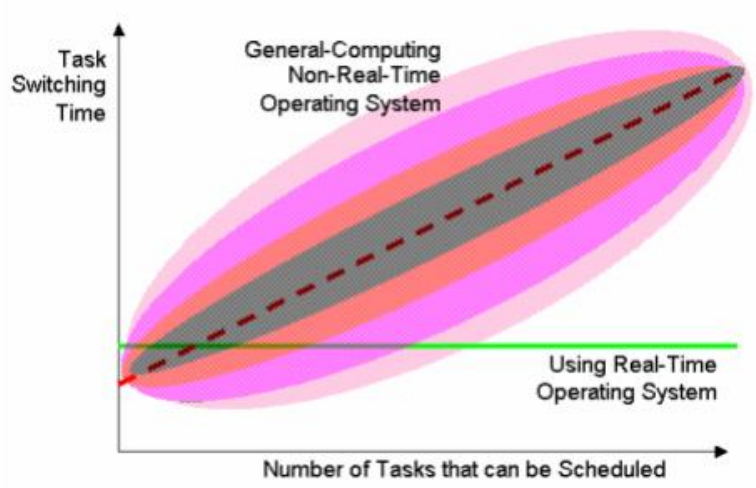
Ο χρονοπρογραμματιστής είναι υπεύθυνος για το ποια διεργασία θα μπλοκαριστεί με σκοπό να εκτελεστεί κάποια άλλη διεργασία. Η διαδικασία αυτή, γνωστή ως **εναλλαγή περιβάλλοντος** ή εναλλαγή περιθωρίου (**context switching**), διαχειρίζεται από έναν διακομιστή (dispatcher) που εκτελεί έναν συγκεκριμένο αλγόριθμο.
Ο ρόλος του διακομιστή είναι να αποθηκεύει το περιβάλλον της προηγούμενης διεργασίας, να φορτώνει το περιβάλλον της νέας διεργασίας και να επιτρέπει στη νέα εργασία να εκτελεστεί. Το περιβάλλον περιλαμβάνει τα δεδομένα, τον καταχωρητή προγράμματος, τα δεδομένα καταχωρητών, το δείκτη στοίβας κ.λπ. Αυτά τα στοιχεία αποθηκεύονται σε μια δυναμική λίστα του λειτουργικού συστήματος, γνωστή ως πίνακας διεργασιών (process table) ή μπλοκ ελέγχου εργασιών (Task Control Block).
Στα λειτουργικά συστήματα γενικού σκοπού, ο χρονοπρογραμματιστής αναζητά στον πίνακα διεργασιών (TCB) για να βρει την επόμενη διεργασία που πρέπει να εκτελεστεί. Όσο μεγαλύτερος είναι ο πίνακας αυτός, δηλαδή όσες περισσότερες διεργασίες βρίσκονται σε αναμονή, τόσο περισσότερο χρόνο απαιτείται για τον χρονοπρογραμματιστή να εντοπίσει την

επόμενη διεργασία και να γίνει η εναλλαγή. Αυτή η καθυστέρηση μπορεί να είναι ενοχλητική στα λειτουργικά συστήματα γενικού σκοπού. Για παράδειγμα, αν ο χρόνος εναλλαγής είναι 1 μsec, ενώ το κβάντο λειτουργίας των διεργασιών είναι 4 μsec, τότε το 25% του χρόνου της CPU χρησιμοποιείται για διαχειριστικές λειτουργίες. Για αυτό το λόγο, συνήθως ορίζεται ένα σχετικά μεγάλο κβάντο χρόνου, αλλά όχι τόσο μεγάλο ώστε οι άλλες διεργασίες να περιμένουν πολλή ώρα σε αναμονή.



Εικόνα 10: Η διαδικασία μετάβασης κατά την εναλλαγή διεργασιών [8]

Στα συστήματα πραγματικού χρόνου, όμως, η εξάρτηση του χρόνου εναλλαγής από τον αριθμό των διεργασιών σε αναμονή δεν είναι αποδεκτή, καθώς αντίκειται στην απαίτηση της πραγματικού χρόνου λειτουργίας. Στα λειτουργικά συστήματα πραγματικού χρόνου, απαιτείται να γνωρίζουμε εκ των προτέρων πόση θα είναι η καθυστέρηση της εναλλαγής. Για τον λόγο αυτό, χρησιμοποιούνται αυξητικά ενημερωμένοι πίνακες στον πίνακα διεργασιών (TCB), οι οποίοι επιτρέπουν στον χρονοπρογραμματιστή να εντοπίσει γρήγορα την επόμενη διεργασία, με σταθερό χρόνο ανεξάρτητα από τον αριθμό των διεργασιών που βρίσκονται σε αναμονή.



Εικόνα 11: Διαφορά των RTOS και GPOS στην εναλλαγή περιβάλλοντος [17]

Στα λειτουργικά συστήματα γενικού σκοπού, η κόκκινη διακεκομμένη γραμμή περιγράφει τη γραμμική σχέση μεταξύ του πλήθους των διεργασιών και του χρόνου εναλλαγής. Συγκεκριμένα, δείχνει ότι όσο αυξάνεται το πλήθος των εργασιών, αυξάνεται επίσης και ο χρόνος που απαιτείται για να γίνει η εναλλαγή μεταξύ τους. Ωστόσο, ο χρόνος εναλλαγής δεν ακολουθεί απαραίτητα ακριβώς τη γραμμή, αλλά μπορεί να έχει αποκλίσεις που αναπαριστούν τις πιθανές αβεβαιότητες και καθυστερήσεις στην εναλλαγή.

Στα λειτουργικά συστήματα πραγματικού χρόνου (RTOS), ο χρόνος εναλλαγής είναι σταθερός και προκαθορισμένος. Αυτό σημαίνει ότι ο καθορισμένος χρόνος που αναλογεί σε κάθε διεργασία πρέπει να τελειώνει και να προχωρά στην επόμενη διεργασία. Παρόλο που τα RTOS μπορεί να φαίνονται λιγότερο αποδοτικά για μικρό αριθμό διεργασιών, το πλεονέκτημά τους είναι εμφανές όταν αυξάνεται το πλήθος των διεργασιών.

Κατά την εκτέλεση μιας διεργασίας, εάν το λειτουργικό σύστημα λάβει ένα εξωτερικό ή εσωτερικό σήμα που υποδηλώνει την ύπαρξη μιας νέας εργασίας ή εάν ολοκληρωθεί το χρονικό κβάντο που αναλογεί στην εκτελούμενη εργασία, ακολουθούνται τα εξής βήματα:

1. Ελέγχεται εάν η εκτελούμενη διεργασία έχει το δικαίωμα να συνεχίσει την εκτέλεσή της.
2. Λαμβάνεται απόφαση σχετικά με την επόμενη διεργασία που θα εκτελεστεί αμέσως μετά.
3. Το περιβάλλον της τρέχουσας διεργασίας αποθηκεύεται, ώστε να μπορεί να συνεχιστεί αργότερα.
4. Η τρέχουσα διεργασία διακόπτεται.
5. Φορτώνεται η νέα διεργασία που πρόκειται να εκτελεστεί.
6. Η νέα διεργασία αρχίζει την εκτέλεσή της.

Με αυτόν τον τρόπο, το λειτουργικό σύστημα εξυπηρετεί την εκτέλεση πολλαπλών διεργασιών με τρόπο που διασφαλίζει την δικαιοσύνη στον χρόνο εκτέλεσής τους και την αποτελεσματική αξιοποίηση των πόρων του συστήματος.[17]

3. Τα Λειτουργικά Σύστηματα Πραγματικού Χρόνου FreeRTOS και ChibiOS

Τόσο το FreeRTOS όσο και το ChibiOS είναι λειτουργικά συστήματα πραγματικού χρόνου (RTOS). Και τα δύο μπορούν να χρησιμοποιηθούν για να διαχειρίζονται τις διεργασίες σε ένα σύστημα που απαιτεί χρονισμό και αξιοπιστία σε πραγματικό χρόνο.
Το FreeRTOS και το ChibiOS μπορούν να είναι χρήσιμα και για συστήματα που απαιτούν χαμηλές απαιτήσεις σε ό, τι αφορά τους πόρους, όπως οι εφαρμογές IoT (διαδίκτυο των αντικειμένων) ή οι εφαρμογές που τρέχουν σε μικροελεγκτές χαμηλής ισχύος.

3.1 Το Λειτουργικό Σύστημα ChibiOS

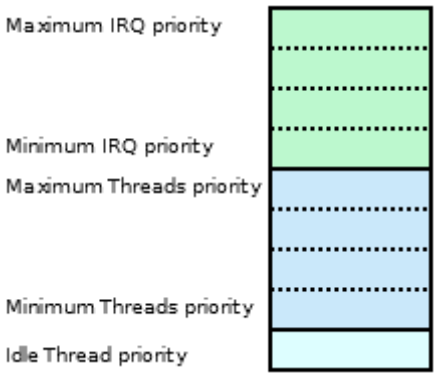
Το ChibiOS/RT είναι ένα μικρό και γρήγορο λειτουργικό σύστημα πραγματικού χρόνου που υποστηρίζει πολλαπλές αρχιτεκτονικές και κυκλοφορεί με συνδυασμό της άδειας GNU έκδοση 3 (GPL3) και της άδειας Apache 2.0. Αναπτύχθηκε από τον Giovanni Di Sirio.
Όλες οι λειτουργίες του, όπως νήματα, σηματοφόροι, κ.λπ., μπορούν να δημιουργηθούν και να διαγραφούν κατά το χρόνο εκτέλεσης. Δεν υπάρχει ανώτατο όριο εκτός από τη διαθέσιμη μνήμη. Για να αυξηθεί η αξιοπιστία του συστήματος, η αρχιτεκτονική του πυρήνα είναι εντελώς στατική, δεν απαιτείται εκχωρητής μνήμης (αλλά διατίθεται προαιρετικά) και δεν υπάρχουν δομές δεδομένων με ανώτερα όρια μεγέθους, όπως στους πίνακες. Οι διεπαφές προγραμματισμού εφαρμογών συστήματος (API) έχουν σχεδιαστεί ώστε να μην έχουν συνθήκες σφάλματος, όπως κωδικούς σφαλμάτων ή εξαιρέσεις.[15]

Εικόνα 12: Το λογότυπο του ChibiOS [20]

- **<u>Προτεραιότητες και Χρονοπρογραμματισμός στο ChibiOS</u>**

Στο μοντέλο προγραμματισμού, οι διεργασίες κατανέμονται σε σταθερά επίπεδα προτεραιότητας. Οι προτεραιότητες διεργασιών είναι συνήθως ένα συνεχές αριθμητικό διάστημα που κυμαίνεται από το ελάχιστο έως το μέγιστο. Το ακριβές αριθμητικό εύρος δεν

έχει σημασία. Συνήθως οι ρουτίνες εξυπηρέτησης διακοπής (ISR) τοποθετούνται σε ένα αριθμητικό διάστημα πάνω από το εύρος προτεραιοτήτων διεργασιών.[14]



Εικόνα 13: Επίπεδα προτεραιότητας [14]

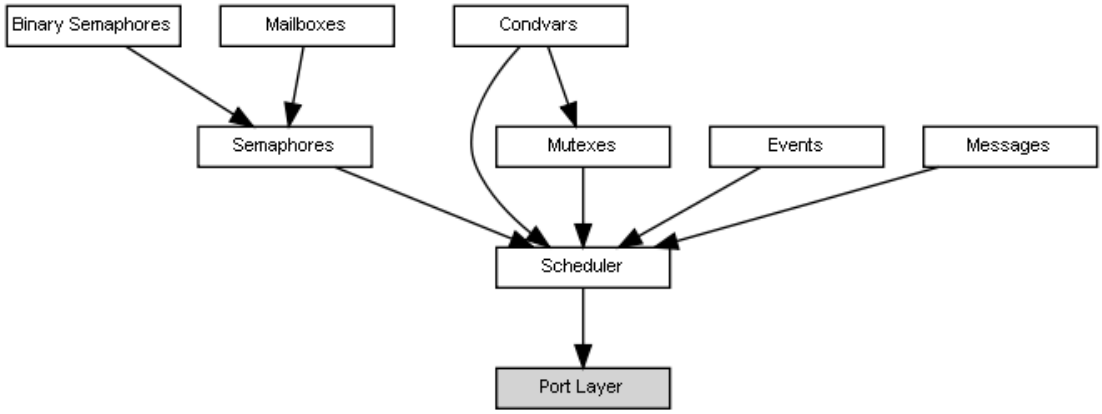
Το χαμηλότερο επίπεδο προτεραιότητας είναι δεσμευμένο για μια διεργασία που συνήθως ονομάζεται "*αδρανής διεργασία*". Είναι η διεργασία που εκτελείται όταν όλες οι άλλες διεργασίες ή τα ISR δεν είναι έτοιμα για εκτέλεση. Ο κανόνας του προγραμματισμού είναι πολύ απλός: **κάθε στιγμή, η διεργασία που εκτελείται είναι η διεργασία με το υψηλότερο επίπεδο προτεραιότητας**. Αυτό ισχύει τόσο για διεργασίες όσο και για ISR.

Το ChibiOS όπως και άλλα λειτουργικά συστήματα πραγματικού χρόνου, χρησιμοποιούν και τον όρο νήματα για τις διεργασίες τους.

```
/* Static variables shared among all threads.*/
uint32_t counter;
static unsigned abc[16];

/* Working area for the thread.*/
static THD_WORKING_AREA(waCounter, 128);

/* Thread function.*/
static THD_FUNCTION(Counter, arg) {
  static uint32_t cows = 0; /* Thread shared variable.*/
  bool condition = true;    /* Thread private variable.*/

  while (condition) {
    /* Thread code.*/
    cows++;
    condition = (bool)(cows < 100);

    /* Small interval.*/
    chThdSleepMilliseconds(100);
  }

  chThdExit((msg_t)cows);
}
```

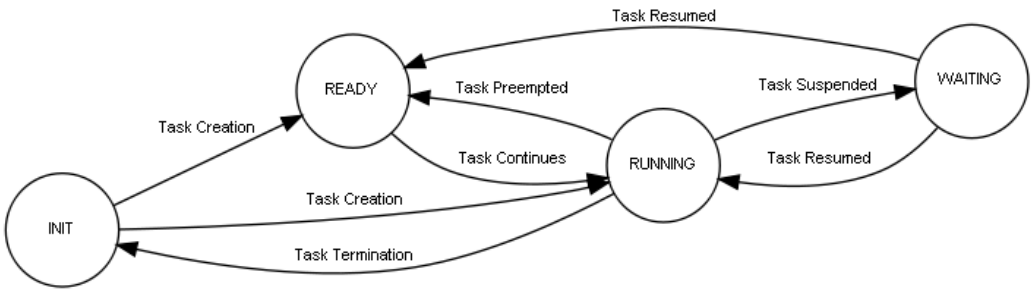
Εικόνα 14: Παράδειγμα διεργασίας στο ChibiOS [9]

Ο πυρήνας του ChibiOS/RT χωρίζεται εσωτερικά σε καλά καθορισμένες μονάδες και έχει ληφθεί μεγάλη προσοχή ώστε τα υποσυστήματα να διατηρούνται καλά απομονωμένα ούτως ώστε αυτό να επιτρέπει την ενεργοποίηση ή απενεργοποίηση κάθε υποσυστήματος από το αρχείο διαμόρφωσης. Η απενεργοποίηση αχρησιμοποίητων υποσυστημάτων επιτρέπει μεγάλη εξοικονόμηση σε κώδικα ή/και χώρο δεδομένων.[9]



Εικόνα 15: Ο πυρήνας του ChibiOS [9]

- **Καταστάσεις Διεργασιών στο ChibiOS**



Εικόνα 16: Οι καταστάσεις διεργασιών στο ChibiOS [14]

Όπως φαίνεται στο σχήμα, οι διεργασίες στο ChibiOS περνούν από διάφορες καταστάσεις οι οποίες είναι:

- **INIT**: Η διεργασία δεν έχει δημιουργηθεί ακόμη.
- **ΕΤΟΙΜΗ/READY**: Η διεργασία είναι κατάλληλη για εκτέλεση αλλά δεν εκτελείται ακόμα.
- **ΕΚΤΕΛΕΣΗ/RUNNING**: Η διεργασία εκτελείται αυτήν τη στιγμή.
- **ΑΝΑΜΟΝΗ/WAITING**: Η διεργασία περιμένει συμβάντα για να συνεχιστεί η εκτέλεση.

3.2 Το Λειτουργικό Σύστημα FreeRTOS

Το FreeRTOS είναι ανεπτυγμένο από τον Richard Barry και την ομάδα του FreeRTOS. Είναι λογισμικό ανοικτού κώδικα και υποστηρίζει πολλές αρχιτεκτονικές. Ο χρονοπρογραμματιστής του υποστηρίζει λειτουργία προεκτροπής ή συνεργατικό χρονοπρογραμματισμό και σχεδιάστηκε έτσι ώστε να είναι μικρός και απλός. Παρέχει ουρές δυαδικούς σηματοφόρους, σηματοφόρους απαρίθμησης καθώς και αναδρομικούς σηματοφόρους.[2]

Εικόνα 17: Το λογότυπο του FreeRTOS [5]

Στο FreeRTOS κάθε νήμα εκτέλεσης ονομάζεται διαδικασία. Οι διαδικασίες υλοποιούνται ως συναρτήσεις της C οι οποίες επιστρέφουν void και σαν όρισμα έχουν ένα δείκτη τύπου void. Μια διαδικασία είναι ένα μικρό πρόγραμμα που έχει ένα σημείο εισόδου, εκτελείται σε ατέρμονα βρόγχο και δεν τερματίζει ποτέ.[11]

```
void ATaskFunction(void *pvParameters) {
  /* Each instance of task will have its own copy
 of variable */
  int iVariableExample = 0;
  /* A task is normally implemented in infinite l
oop */
  for( ;; ) { /* task functionality */ }
  /* Should the code ever break out of the above
loop
  then the task must be deleted. */
```

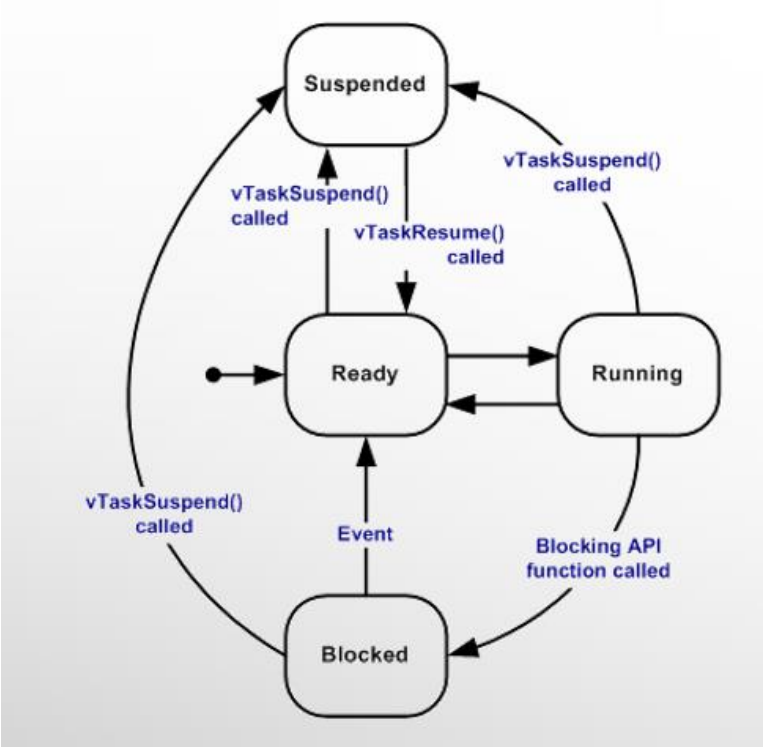
Εικόνα 18: Παράδειγμα διαδικασίας στο FreeRTOS [11]

- **Προτεραιότητες και Χρονοπρογραμματισμός στο FreeRTOS**

Το FreeRTOS χρησιμοποιεί έναν υψηλής προτεραιότητας χρονοπρογραμματιστή εξυπηρέτησης, ο οποίος εξασφαλίζει ότι η διαδικασία με την υψηλότερη προτεραιότητα θα εξυπηρετείται πάντα πρώτη. Αυτό επιτυγχάνεται μέσω της ανάθεσης προτεραιοτήτων σε κάθε διαδικασία και της χρήσης της προτεραιότητας αυτής από τον χρονοπρογραμματιστή για να αποφασίζει ποια διαδικασία θα εκτελεστεί πρώτη. Στις περιπτώσεις όπου υπάρχουν διαδικασίες με ίδια προτεραιότητα, η εξυπηρέτηση γίνεται με τη μέθοδο Round Robin (εξυπηρέτηση εκ περιτροπής).

- **Καταστάσεις Διαδικασιών στο FreeRTOS**

Οι διαδικασίες στο λειτουργικό σύστημα μπορούν να θεωρηθούν ότι καταλαμβάνουν μία από δύο καταστάσεις **Εκτελείται**(running) ή **Δεν εκτελείται** (notRunning). Η εκτελούμενη διαδικασία καταγράφεται από το μπλοκ ελέγχου εργασιών pxCurrentTCB που απλά υποδεικνύει ότι η διαδικασία εκτελείται αυτή τη στιγμή στον επεξεργαστή. Η κατάσταση **Δεν εκτελείται** μπορεί να αναλυθεί σε τρεις υποκαταστάσεις**: Έτοιμή** (ready) , **Σε αναστολή** (suspended) και **Αποκλεισμένη** (blocked).



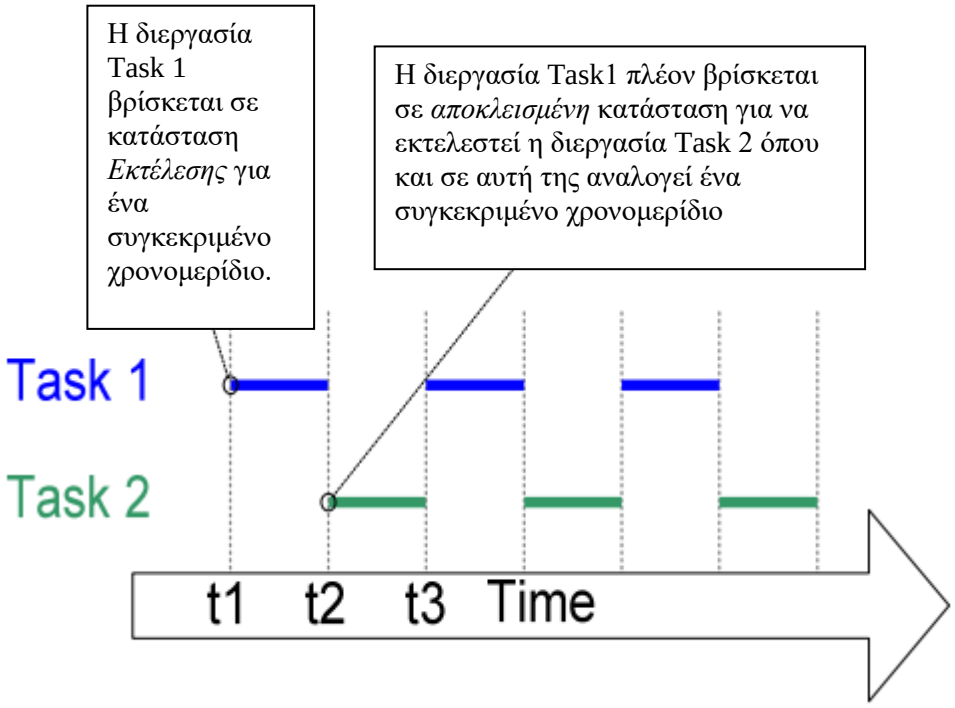
Εικόνα 19: Καταστάσεις διαδικασιών σε λειτουργικό σύστημα πραγματικού χρόνου [11]

Ακολουθεί η περιγραφή των καταστάσεων διαδικασιών του FreeRTOS:

- **Εκτελείται**: Όταν εκτελείται μια διαδικασία, στην πραγματικότητα λέγεται ότι βρίσκεται σε κατάσταση λειτουργίας και χρησιμοποιεί τον επεξεργαστή.
- **Έτοιμη**: Έτοιμες διαδικασίες είναι εκείνες που μπορούν να εκτελεστούν (δεν αποκλείονται ή δεν έχουν ανασταλεί), αλλά δεν εκτελούνται αυτήν τη στιγμή, επειδή μια διαφορετική διαδικασία ίσης ή υψηλότερης προτεραιότητας βρίσκεται ήδη σε κατάσταση εκτέλεσης.
- **Αποκλεισμένη:** Μια διαδικασία βρίσκεται σε αποκλεισμένη κατάσταση, όταν εκείνη την στιγμή αναμένει είτε ένα χρονικό είτε ένα εξωτερικό συμβάν. Για παράδειγμα, εάν μια διαδικασία καλεί την εντολή που αφορά την καθυστέρηση διαδικασίας, θα αποκλειστεί μέχρι να λήξει η περίοδος καθυστέρησης - ένα χρονικό συμβάν. Οι διαδικασίες μπορούν επίσης να αποκλειστούν περιμένοντας για κάποιο γεγονός ουράς ή σηματοφόρου. Οι διαδικασίες στην αποκλεισμένη κατάσταση έχουν πάντα μια χρονική περίοδο, μετά την οποία θα ξεμπλοκαριστούν. Οι αποκλεισμένες διαδικασίες δεν είναι διαθέσιμες για χρονοδρομολόγηση.
- **Σε Αναστολή:** Οι διαδικασίες σε κατάσταση αναστολής δεν είναι επίσης διαθέσιμες για χρονοδρομολόγηση. Οι διαδικασίες θα εισέλθουν ή θα εξέλθουν από την κατάσταση αναστολής μόνο όταν έχουν ρητή εντολή να το κάνουν μέσω των κατάλληλων κλήσεων API.[11]

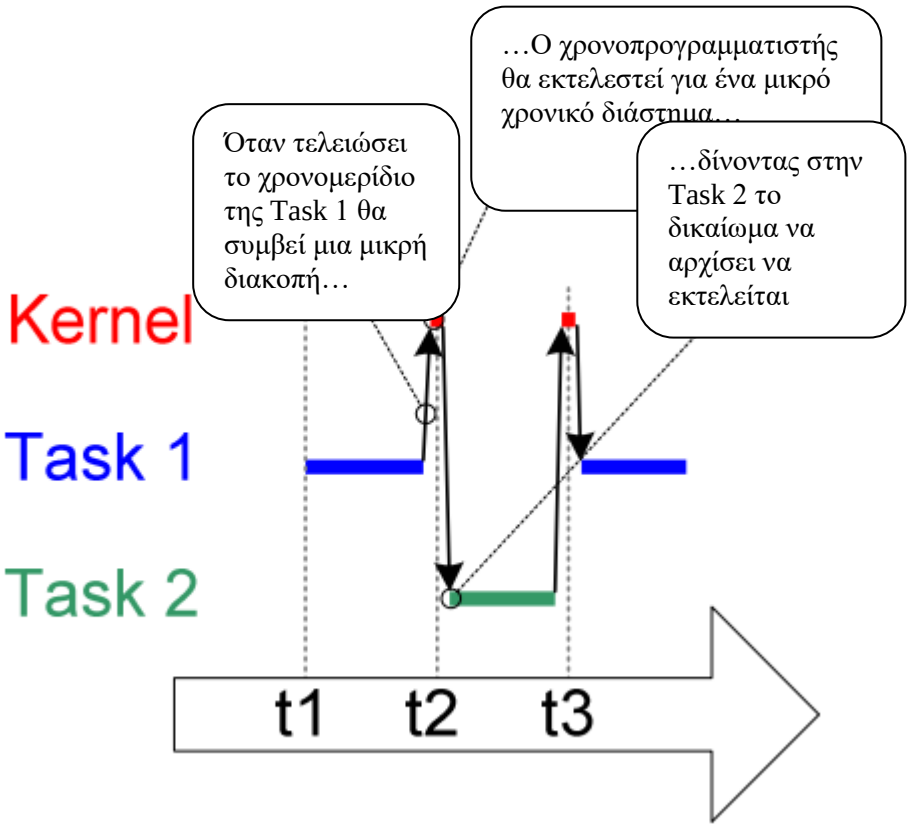
3.3 Οι Διεργασίες στα Λειτουργικά Συστήματα Πραγματικού Χρόνου

Κάθε διεργασία είναι μια ανεξάρτητη μονάδα εργασίας, η οποία εκτελεί μια συγκεκριμένη εργασία στο σύστημα. Κάθε διεργασία έχει τη δική της προτεραιότητα και αναλαμβάνει την εκτέλεση της εργασίας της μόνο όταν είναι η σειρά της σύμφωνα με τον χρονοπρογραμματιστή.



Εικόνα 20: Διεργασίες ίσης προτεραιότητας [16]

Όταν ο χρονοπρογραμματιστής επιλέξει να εκτελέσει την επόμενη διεργασία, υπάρχει μια **περιοδική διακοπή** όπου εκείνη την στιγμή για ένα πολύ μικρό χρονικό διάστημα τρέχει ο ίδιος ο χρονοπρογραμματιστής. Για να γίνει σαφές αυτό, το παρακάτω σχήμα δείχνει το σημείο το οποίο εκτελείται ο χρονοπρογραμματιστής όταν εναλλάσσονται οι διεργασίες.

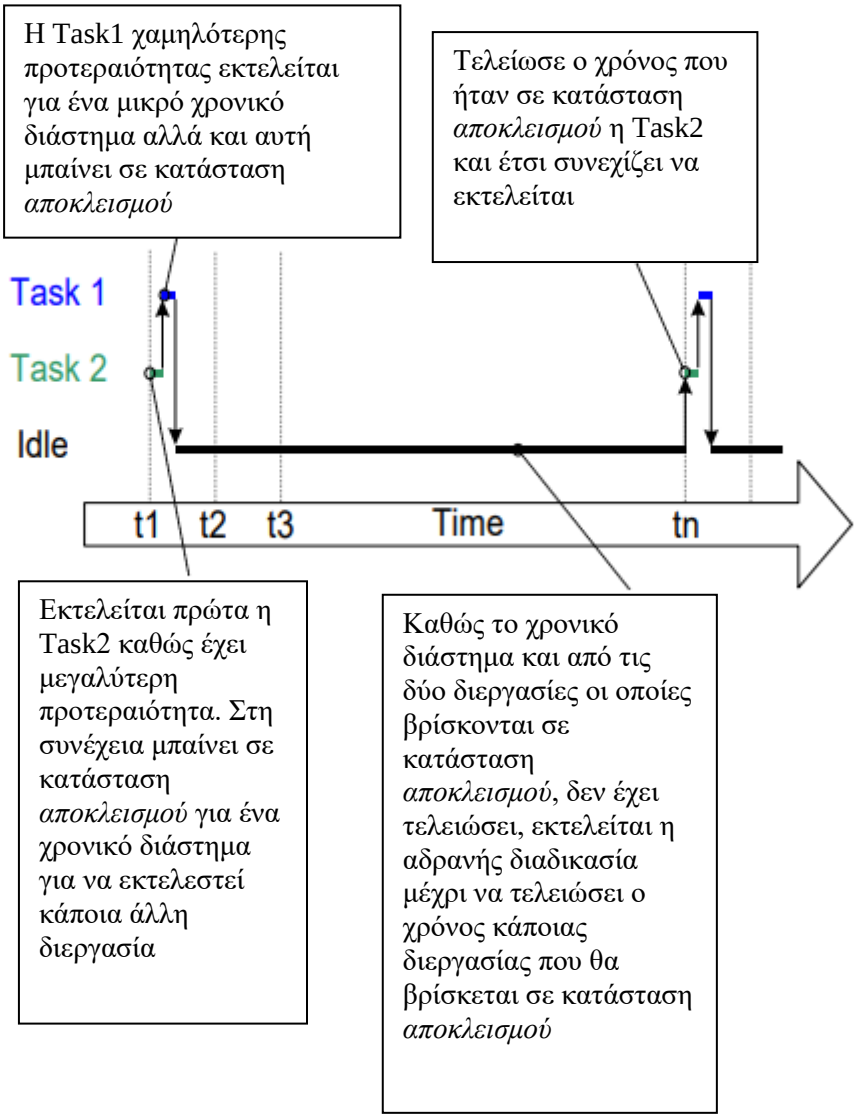


Εικόνα 21: Όταν εκτελείται ο χρονοπρογραμματιστής [16]

- **<u>Αδρανής διεργασία και διεργασίες διαφορετικής προτεραιότητας</u>**

Ο επεξεργαστής χρειάζεται πάντα μια διεργασία για να εκτελεστεί. Πρέπει δηλαδή πάντα να υπάρχει κάποια διεργασία στην κατάσταση *Εκτέλεσης*. Για να διασφαλιστεί αυτό, δημιουργείται αυτόματα μια διεργασία αδράνειας από τον χρονοπρογραμματιστή. Η διεργασία αυτή έχει την χαμηλότερη προτεραιότητα (προτεραιότητα 0) για να διασφαλιστεί ότι ποτέ δεν αποτρέπει υψηλότερης προτεραιότητας άλλης διεργασίας να εισαχθεί στην κατάσταση *Εκτέλεσης*. Η *αδρανής διεργασία* θα εκτελείται όταν δεν υπάρχει άλλη διεργασία υψηλότερης προτεραιότητας σε κατάσταση *Έτοιμη*.

Παρακάτω φαίνεται στο σχήμα να εκτελούνται 2 διεργασίες διαφορετικής προτεραιότητας Task1(Χαμηλή προτεραιότητα), Task2(Υψηλή προτεραιότητα) καθώς και η αδρανής διεργασία (με την χαμηλότερη προτεραιότητα) που εκτελείται όταν δεν υπάρχει κάποια διεργασία για εκτέλεση.

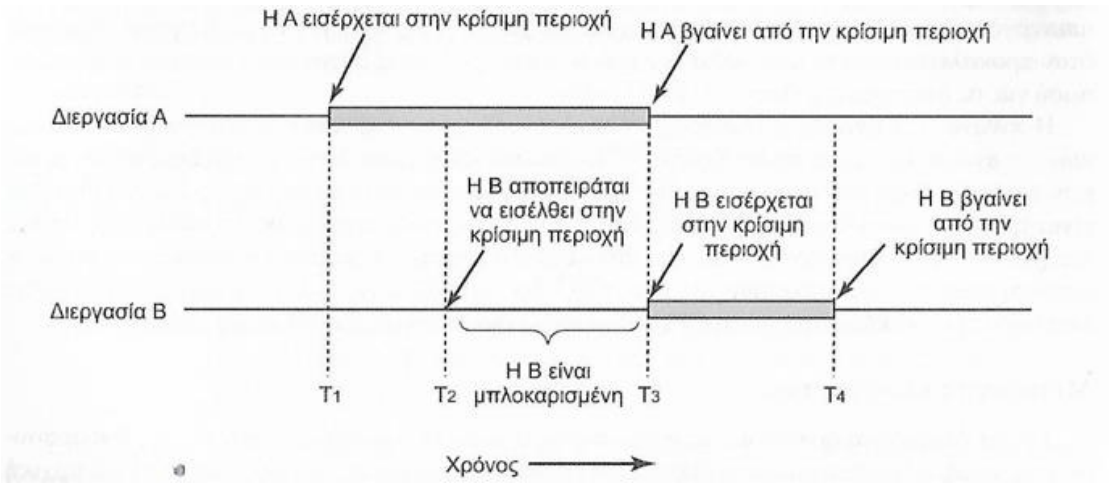


Εικόνα 22: Εκτέλεση αδρανούς διεργασίας και διεργασιών διαφορετικής προτεραιότητας [16]

3.4 Οι Κρίσιμες Περιοχές

Τα λειτουργικά συστήματα πραγματικού χρόνου χρησιμοποιούνται σε πολλές εφαρμογές όπου απαιτείται ακρίβεια και σταθερότητα στην εκτέλεση των διεργασιών. Ωστόσο επειδή συνδέονται στενά με τον παραλληλισμό και τις πολυνηματικές εφαρμογές, όταν προγραμματίζουμε παράλληλες διεργασίες, πρέπει να είμαστε προσεκτικοί για να αποφύγουμε προβλήματα, που αφορούν την εκτέλεση πολλαπλών νημάτων ταυτόχρονα σε έναν κοινό πόρο. Το τμήμα ενός προγράμματος στο οποίο γίνεται προσπέλαση κοινόχρηστης μνήμης ονομάζεται **κρίσιμη περιοχή** ή **κρίσιμο τμήμα**. Αν μπορούσαμε να εξασφαλίσουμε ότι ποτέ δύο ή περισσότερες διεργασίες δε θα βρίσκονται ταυτόχρονα σε κρίσιμες περιοχές, θα αποφεύγουμε τον συναγωνισμό.

Το πιο σημαντικό ζήτημα για την αποφυγή των προβλημάτων της κοινής χρήσης ενός πόρου από πολλές διεργασίες είναι η εύρεση κάποια μεθόδου ώστε να μην επιτρέπεται σε περισσότερες από μία διεργασίες η ανάγνωση ή εγγραφή των κοινών δεδομένων, την ίδια χρονική στιγμή. Με άλλα λογία αυτό που χρειαζόμαστε είναι ο **αμοιβαίος αποκλεισμός**, δηλαδή μία μέθοδος που θα εξασφαλίζει ότι, αν μια διεργασία χρησιμοποιεί μια κοινή μεταβλητή, οι άλλες διεργασίες θα αποκλείονται από την εκτέλεση της ίδιας ενέργειας. [1]



Εικόνα 23: Αμοιβαίος αποκλεισμός με τη χρήση κρίσιμων περιοχών [1]

Υπάρχουν διάφορες προσεγγίσεις για την επίτευξη του αμοιβαίου αποκλεισμού, ώστε στο διάστημα που μια διεργασία ενημερώνει την κοινόχρηστη περιοχή μνήμης, καμία άλλη να μην μπαίνει στο δικό της κρίσιμο τμήμα. Μία λύση είναι η χρησιμοποίηση των **σηματοφόρων.**

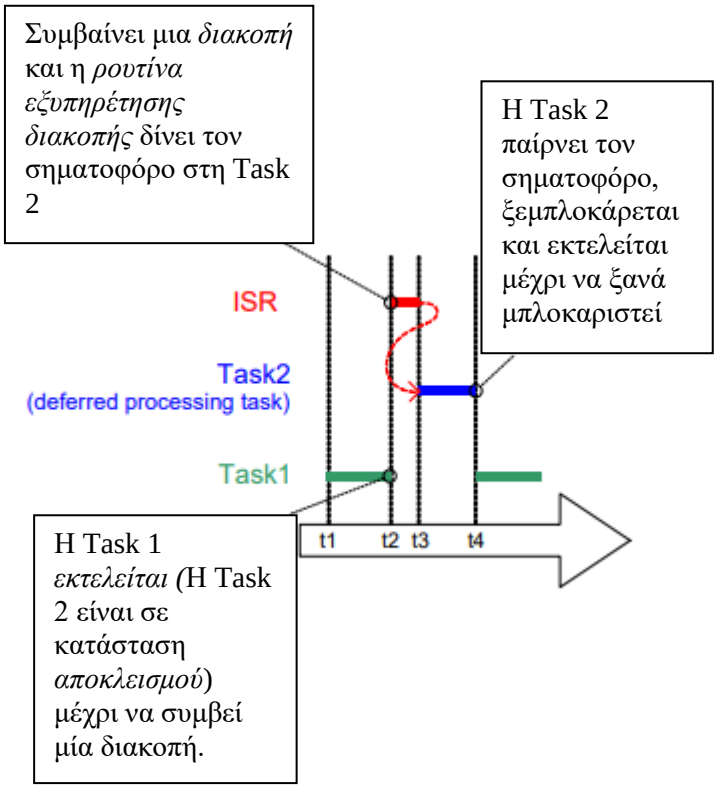
3.5 Σηματοφόροι και Διακοπές

Ο σηματοφόρος είναι ένας μηχανισμός που χρησιμοποιείται από τα περισσότερα λειτουργικά συστήματα. Αντιπροσωπεύει ένα αντικείμενο του λειτουργικού συστήματος το οποίο δεσμεύεται από μία διεργασία Α όταν χρησιμοποιεί έναν κοινόχρηστο πόρο και απελευθερώνεται όταν η διεργασία Α απελευθερώνει τον πόρο. Αν μια διεργασία Β προσπαθεί να αποκτήσει τον ίδιο πόρο ενώ ο σηματοφόρος είναι δεσμευμένος, τότε η διεργασία Β πρέπει να περιμένει μέχρι να απελευθερωθεί ο πόρος. Όταν αυτό συμβεί, η διεργασία Β αποκτά τον σηματοφόρο και οι υπόλοιπες διεργασίες πρέπει να περιμένουν.[8]

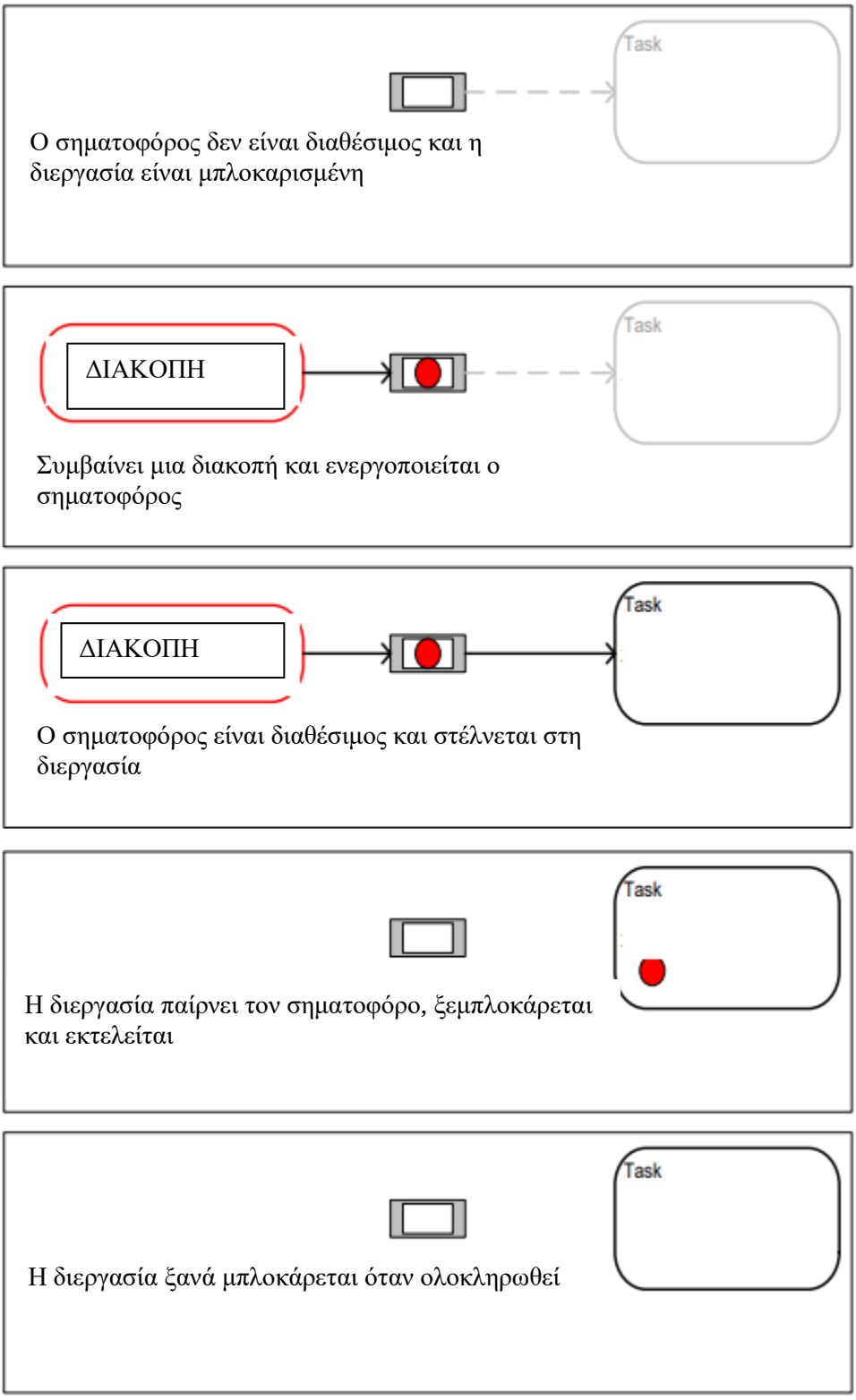
Επιπλέον, οι δυαδικοί σηματοφόροι αποτελούν ένα αποτελεσματικό μέσο συγχρονισμού μεταξύ μιας διακοπής και του χειριστή της. Ο δυαδικός σηματοφόρος λειτουργεί ως μηχανισμός για το ξεμπλοκάρισμα μιας διεργασίας κάθε φορά που εμφανίζεται μια συγκεκριμένη διακοπή, επιτρέποντας την ομαλή εκτέλεση των διαδικασιών.

Η επεξεργασία της διακοπής έχει "ανατεθεί" σε μια διεργασία που ονομάζεται *χειριστής διακοπών*, η οποία αναλαμβάνει την ευθύνη για την αποτελεσματική επίλυση της κρίσιμης κατάστασης. Κατά τη διάρκεια αυτής της διαδικασίας, ο χειριστής βρίσκεται σε κατάσταση *αποκλεισμού* περιμένοντας έναν σηματοφόρο. Σε αυτή την κατάσταση, η διεργασία αναμένει την εκδήλωση του αναμενόμενου συμβάντος, που αντιπροσωπεύει τη διακοπή.

Όταν τελικά συμβεί μια διακοπή, η *ρουτίνα εξυπηρέτησης διακοπής* ISR “δίνει” τον σηματοφόρο, επιτρέποντας έτσι το ξεμπλοκάρισμα της αναμονής και την αποδέσμευση της διεργασίας από την κατάσταση *αποκλεισμένη*. Μέσω αυτής της αμοιβαίας αλληλεπίδρασης μεταξύ του σηματοφόρου και του *χειριστή διακοπών*, επιτυγχάνεται ο αποτελεσματικός συγχρονισμός και η ομαλή εξέλιξη της διακοπής.[11]



Εικόνα 24: Διακοπή μιας διεργασίας για να εκτελεστεί κάποια άλλη με τη χρήση διακοπών [16]



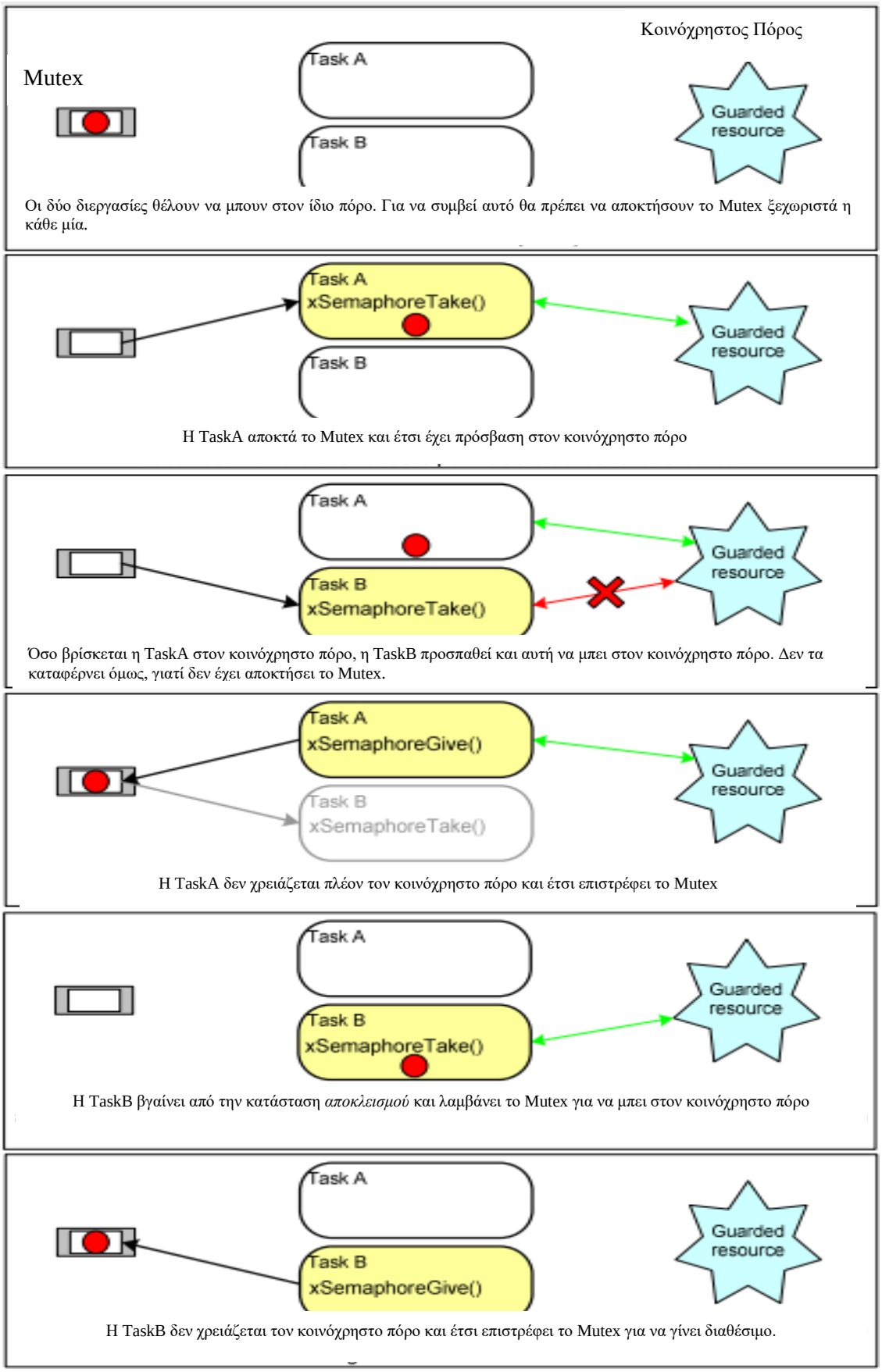
Εικόνα 25: Χρήση Σηματοφόρων [16]

3.6 Mutex

Τα mutex είναι μία απλοποιημένη εκδοχή των δυαδικών σηματοφόρων και χρησιμοποιούνται αποκλειστικά στη διαχείριση του αμοιβαίου αποκλεισμού που εφαρμόζεται σε κάποιο κοινόχρηστο πόρο ή τμήμα κώδικα. Τα Mutex είναι μία μεταβλητή που μπορεί να βρίσκεται σε μία μόνον από δύο καταστάσεις: *Κλειδωμένη* ή *Ξεκλείδωτη*. Χρησιμοποιείται ένας ακέραιος, που όταν έχει την τιμή 0 θεωρείται ξεκλείδωτος και για όλες τις υπόλοιπες τιμές θεωρείται κλειδωμένος.

Όταν μια διεργασία χρειάζεται πρόσβαση στην κρίσιμη περιοχή καλεί το mutex να κλειδώσει. Αν το mutex είναι τη στιγμή αυτή ξεκλείδωτο (που σημαίνει ότι η κρίσιμη περιοχή είναι διαθέσιμη), τότε η διεργασία είναι ελεύθερη να μπει στην κρίσιμη περιοχή.

Αν το mutex είναι ήδη κλειδωμένο, η διεργασία μπλοκάρεται μέχρι τη στιγμή που η άλλη διεργασία που βρίσκεται προς το παρόν στην κρίσιμη περιοχή θα τελειώσει τις εργασίες της και θα καλέσει το Mutex να ξεκλειδώσει.[1].

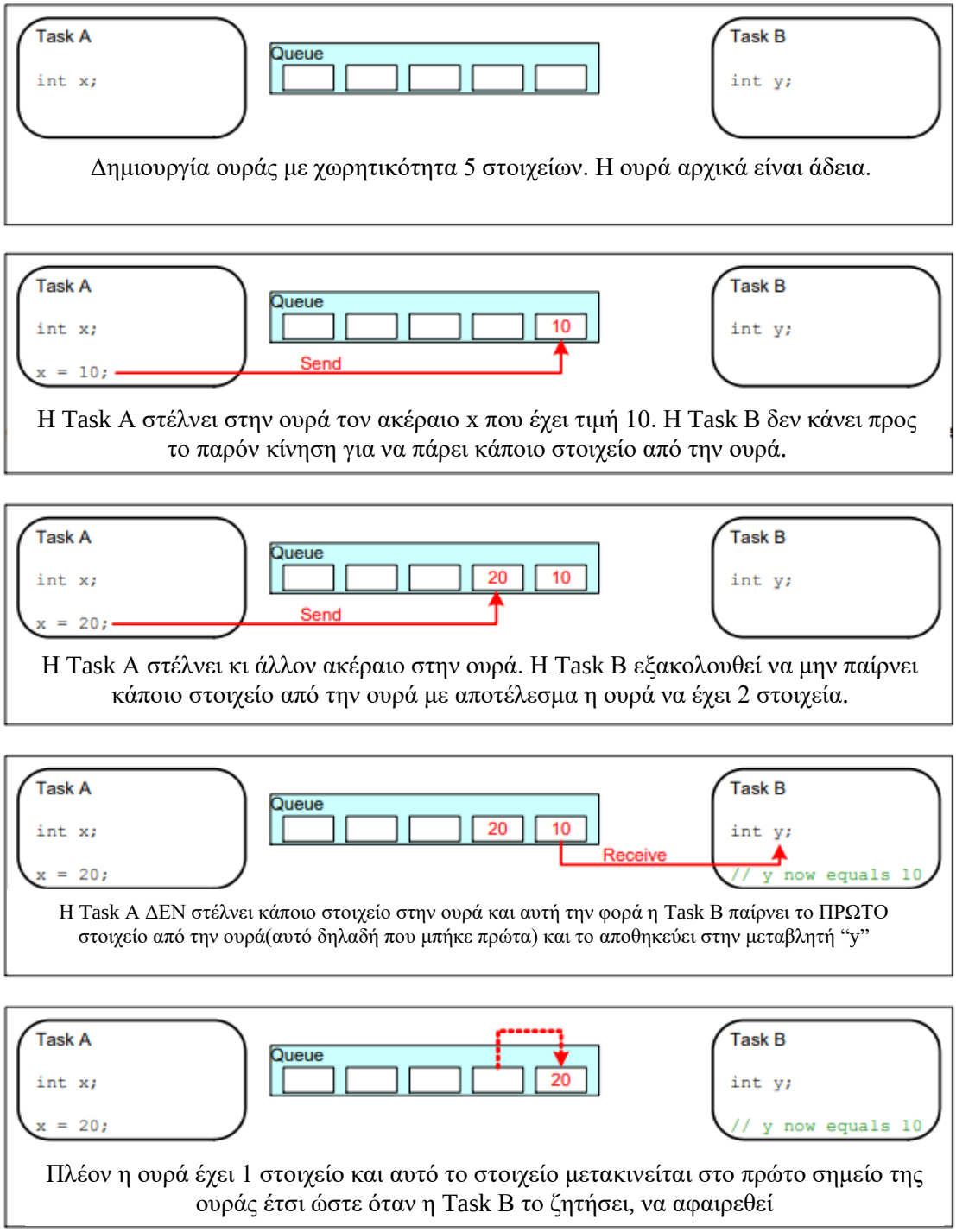


Εικόνα 26: Η λειτουργία των Mutex στον αμοιβαίο αποκλεισμό [16]

3.7 Ουρές

Οι ουρές είναι η κύρια μορφή επικοινωνίας μεταξύ διεργασιών. Μπορούν να χρησιμοποιηθούν για την αποστολή και λήψη μηνυμάτων διαφορετικών διεργασιών. Στις περισσότερες περιπτώσεις χρησιμοποιούνται ως buffer FIFO (First In First Out).

Στο παρακάτω σχήμα φαίνεται ο τρόπος λειτουργίας χρήσης των Ουρών σε ένα λειτουργικό σύστημα πραγματικού χρόνου όπου δύο διεργασίες θέλουν να επικοινωνήσουν μεταξύ τους.



Εικόνα 27: Επικοινωνία δύο διεργασιών με χρήση μιας ουράς [16]

3.8 Αδιέξοδα

Ένα **αδιέξοδο** (deadlock) είναι ένα συνηθισμένο πρόβλημα της πολυδιεργασίας σε συστήματα πραγματικού χρόνου, όπου δύο διεργασίες που μοιράζονται τους ίδιους πόρους, αποτρέπουν η μία την άλλη να έχει πρόσβαση στους κοινούς πόρους, καταλήγοντας τελικά καμία διεργασία να μην μπει στον πόρο. Παρακάτω παρουσιάζεται ένα σενάριο που επεξηγεί πώς δημιουργείται ένα αδιέξοδο:

- Διεργασία 1: Αιτείται τον πόρο Α και τον λαμβάνει.
- Διεργασία 2: Αιτείται τον πόρο Β και τον λαμβάνει.
- Διεργασία 1: Αιτείται τον πόρο Β και μπαίνει στην ουρά αναμονής για την απελευθέρωση του Β.
- Διεργασία 2: Αιτείται τον πόρο Α και μπαίνει στην ουρά αναμονής για την απελευθέρωση του Α.

Τώρα, καμία από τίς δύο διεργασίες δεν μπορεί να προχωρήσει, καθώς η κάθε μία αποτρέπει την άλλη να αποκτήσει πρόσβαση στον πόρο που χρειάζεται. Για να αντιμετωπιστεί αυτή η κατάσταση, το λειτουργικό σύστημα πρέπει να διακόψει μία από τις δύο διεργασίες.[8]

3.9 Οι Βασικές Εντολές στον κώδικα του FreeRTOS και του ChibiOS

Παρακάτω αναφέρονται οι βασικές εντολές στον κώδικα του FreeRTOS [4] και του ChibiOS [10].

	FreeRTOS	ChibiOS
Δημιουργία μιας διεργασίας	xTaskCreate()	chThdCreateStatic()
Δημιουργία σηματοφόρου	xSemaphoreCreateBinary()	chBSemObjectInit()
Δήλωση σηματοφόρου με όνομα xSemaphore	SemaphoreHandle_t xSemaphore	binary_semaphore_t xSemaphore
Δίνει τον σηματοφόρο και η τιμή του γίνεται 1	xSemaphoreGive()	chBSemSignal()
Παίρνει τον σηματοφόρο. Η τιμή του γίνεται 0 και μπλοκάρεται η διεργασία στην οποία βρίσκεται	xSemaphoreTake()	chBSemWait()
Δίνει τον σηματοφόρο από μια διακοπή. Η τιμή του γίνεται 1	xSemaphoreGiveFromISR()	chBSemSignalI()
Δήλωση mutex με όνομα mutex_t	SemaphoreHandle_t	mutex_t
Δημιουργία Mutex	xSemaphoreCreateMutex();	MUTEX_DECL();
Κλειδώνει το mutex	xSemaphoreTake	chMtxLock()
Ξεκλειδώνει το mutex	xSemaphoreGive	chMtxUnlock()
Δημιουργία μιας ουράς/Γραμματοκιβωτίου	xQueueCreate	MAILBOX_DECL()
Στέλνει στην ουρά/γραμματοκιβώτιου ένα στοιχείο που έχει δηλωθεί	xQueueSend()	chMBPostTimeout()

Παίρνει από την ουρά/γραμματοκιβώτιου το στοιχείο που βρίσκεται πρώτο	xQueueReceive()	chMBFetchTimeout()
Η διεργασία βρίσκεται σε αποκλεισμένη κατάσταση για όσο χρόνο έχει δηλωθεί	vTaskDelay()	chThdSleep()
Η διεργασία βρίσκεται σε κατάσταση Αναστολής	vTaskSuspend()	chThdSuspend()
Η διεργασία συνεχίζει από το σημείο που σταμάτησε όταν μπήκε σε κατάσταση Αναστολής	vTaskResume()	chThdResume()
Είσοδος Κρίσιμης Περιοχής	taskENTER_CRITICAL()	chSysLock()
Έξοδος από την Κρίσιμη Περιοχή	taskEXIT_CRITICAL()	chSysUnlock()
Διαγραφή Διεργασίας	vTaskDelete()	chThdExit()

4. Γενικά για τους Μικροελεγκτές

Οι μετρήσεις που αφορούν την απόδοση των δύο λειτουργικών συστημάτων πραγματικού χρόνου θα γίνουν στο Arduino UNO που έχει ενσωματωμένο τον μικροελεγκτή ATmega328p. Επομένως είναι καλό να αναλυθούν κάποια πράγματα που αφορούν τους μικροελεγκτές γενικότερα καθώς και τα χαρακτηριστικά του ATmega328p.

Αν θέλουμε να εξηγήσουμε με απλά λόγια τον ρόλο ενός μικροελεγκτή, μπορούμε να πούμε ότι αποτελεί μια ηλεκτρονική συσκευή που μπορεί να δεχτεί εισόδους σε μορφή ηλεκτρικών σημάτων, να επεξεργαστεί αυτές τις εισόδους μέσω κατάλληλου προγραμματισμού, και να εξάγει σήματα ως αποτέλεσμα της επεξεργασίας. Παραδείγματα:

- Όταν ο αισθητήρας θερμοκρασίας στέλνει ένα ηλεκτρικό σήμα στον μικροελεγκτή, αυτός μεταφράζει το σήμα σε θερμοκρασία. Αν η τιμή της θερμοκρασίας είναι υψηλότερη από ένα καθορισμένο όριο, ο μικροελεγκτής στέλνει ένα θετικό σήμα σε ένα ρελέ, το οποίο κλείνει και ενεργοποιεί έναν ανεμιστήρα. Αντιθέτως, αν η θερμοκρασία είναι χαμηλότερη από ένα δεύτερο όριο, ο μικροελεγκτής στέλνει ένα μηδενικό σήμα, το ρελέ ανοίγει και διακόπτει την τροφοδοσία του ανεμιστήρα, πραγματοποιώντας έτσι τη λειτουργία ενός θερμοστάτη.
- Ένας αισθητήρας απόστασης μεταφέρει μέσω ενός ηλεκτρικού σήματος τις πληροφορίες στον μικροελεγκτή, ο οποίος τις ερμηνεύει και υπολογίζει την απόσταση. Αν η απόσταση είναι μικρότερη από ένα καθορισμένο όριο, ο μικροελεγκτής ενεργοποιεί ένα LED και έναν βομβητή, παρέχοντας οπτική και ακουστική προειδοποίηση. Σε αντίθετη περίπτωση, ο μικροελεγκτής στέλνει μηδενικό σήμα. Αυτό θα μπορούσε να χρησιμοποιηθεί για την εφαρμογή αισθητήρων παρκαρίσματος.

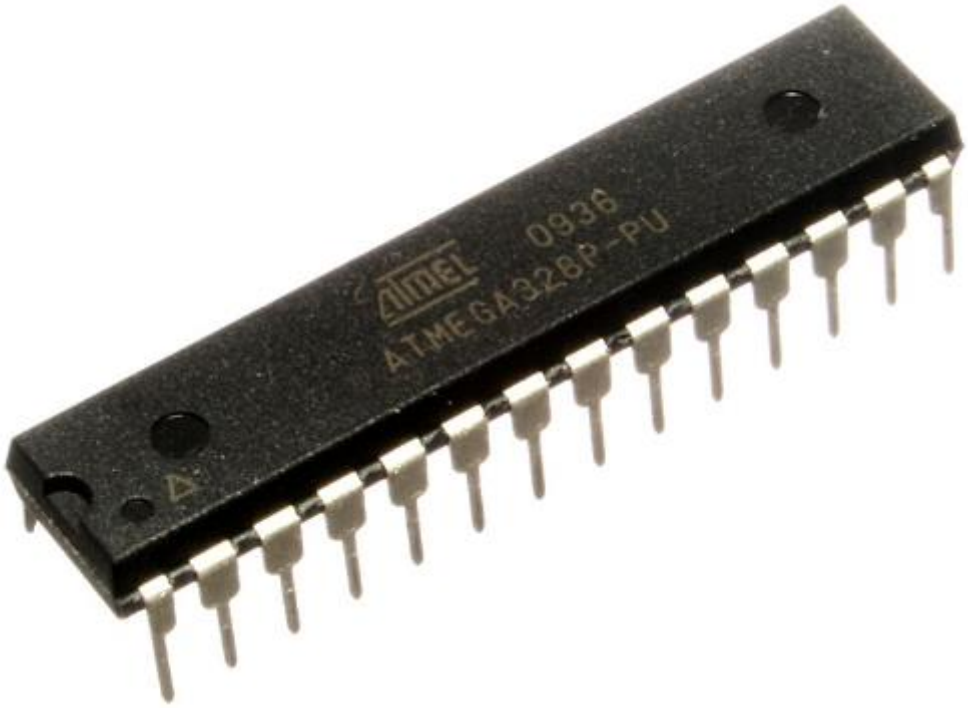
Για να τεθεί σε λειτουργία και να προγραμματιστεί ένας μικροελεγκτής απαιτούνται:

- Ένα κύκλωμα που θα εξασφαλίζει την τροφοδοσία, τον χρονισμό κ.α.
- Μία διάταξη που θα εξασφαλίζει την επικοινωνία μεταξύ Η/Υ και μικροελεγκτή.
- Γνώση της κατάλληλης γλώσσας προγραμματισμού (συνήθως assembly ή C).[3]

4.1 Ο Μικροελεγκτής ATmega328p στο Arduino UNO

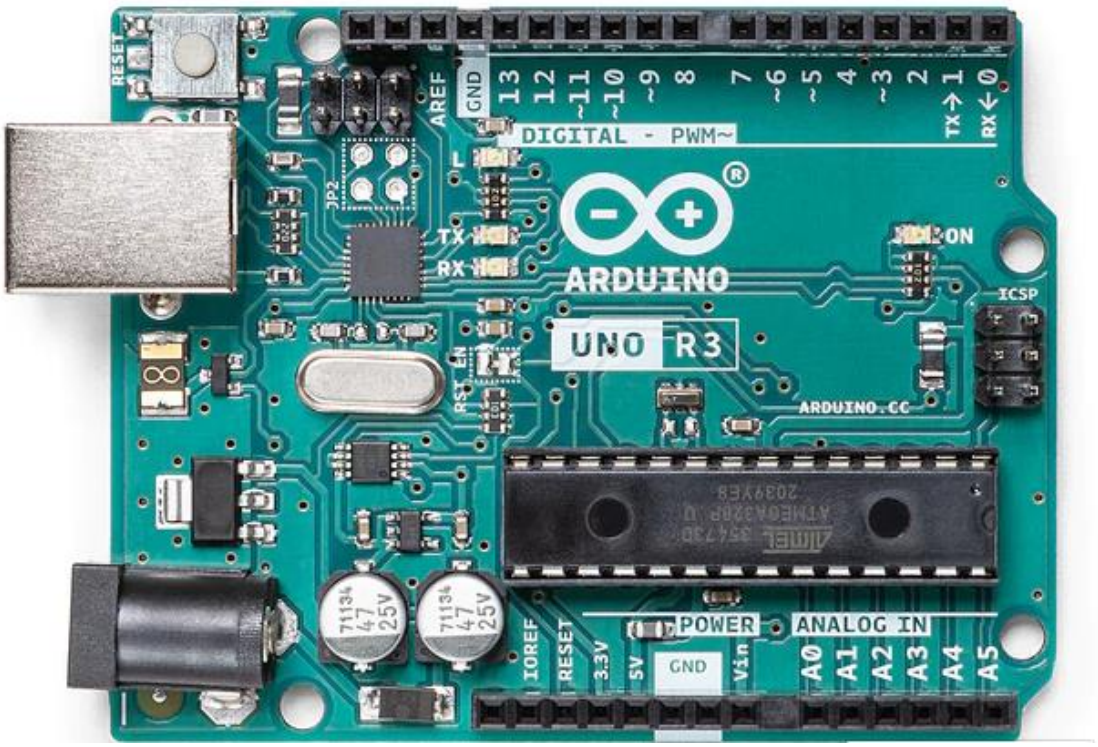
Το Arduino UNO είναι μια ανοικτού κώδικα μητρική πλακέτα που περιλαμβάνει τον μικροελεγκτή **AVR ATmega328p**, το κύκλωμα τροφοδοσίας, το κύκλωμα επικοινωνίας με τον υπολογιστή και τις θύρες εισόδου-εξόδου. Η προγραμματιστική γλώσσα που χρησιμοποιείται είναι η Wiring, η οποία βασίζεται στην γλώσσα C και παρέχει πολλές αυτοματοποιημένες διαδικασίες, καθιστώντας την εύκολη στη χρήση ακόμη και για άτομα χωρίς προηγούμενη εμπειρία στον προγραμματισμό.

Ο μικροελεγκτής **AVR ATmega328p** είναι ένας **8-μπιτος ελεγκτής** στα **16 MHZ** που λειτουργεί στα 5V, και παρέχει 6 αναλογικές εισόδους και 14 ψηφιακές εισόδους – εξόδους.



Εικόνα 28: Ο μικροελεγκτής ATmega238p [13]

Η πλακέτα Arduino συνδέεται στον υπολογιστή μέσω USB. Ο χρήστης δημιουργεί ένα πρόγραμμα με σκοπό να δώσει εντολές στον μικροελεγκτή, και μετά το μεταφορτώνει στην πλακέτα μέσω της σύνδεσης USB. Μόλις ολοκληρωθεί η μεταφόρτωση, η πλακέτα δεν χρειάζεται πλέον σύνδεση με τον υπολογιστή. Έτσι, το καλώδιο USB χρησιμεύει πλέον μόνο για την τροφοδοσία της πλακέτας. Εάν αφαιρέσουμε το καλώδιο USB και τροφοδοτήσουμε την πλακέτα με μια άλλη πηγή ενέργειας, όπως μια μπαταρία ή ένα τροφοδοτικό, η πλακέτα θα μπορεί να λειτουργήσει αυτόνομα χωρίς την υποστήριξη του υπολογιστή.



Εικόνα 29: Η πλατφόρμα Arduino Uno με ενσωματωμένο τον μικροελεγκτή ATmega328p [12]

Αν αφαιρέσουμε τον μικροελεγκτή από την πλακέτα και τον συνδέσουμε με κατάλληλα βοηθητικά κυκλώματα, μπορούμε να τον εφοδιάσουμε με τα απαραίτητα σήματα εισόδου και τροφοδοσίας, έτσι ώστε να μπορεί να λειτουργήσει αυτόνομα. Είναι σημαντικό να σημειωθεί ότι η πλακέτα Arduino δεν αποτελεί απαραίτητα έναν μικροελεγκτή, αλλά είναι το μέσο μέσω του οποίου ο μικροελεγκτής προγραμματίζεται και ελέγχεται.

Ο μικροελεγκτής ATmega328p έχει δύο κύριες κατηγορίες μνήμης:

- Μνήμη φλας (flash memory)
- Μνήμη RAM.

Η **μνήμη φλας** χρησιμοποιείται για την αποθήκευση του προγράμματος και των σταθερών δεδομένων. Η μέγιστη χωρητικότητα της μνήμης flash του ATmega328p είναι 32KB, αλλά ένα μέρος αυτής χρησιμοποιείται από το λειτουργικό σύστημα του Arduino UNO.

Η **μνήμη RAM** χρησιμοποιείται για την αποθήκευση μεταβλητών και για άλλες προσωρινές τιμές κατά τη διάρκεια της εκτέλεσης του προγράμματος. Η μέγιστη χωρητικότητα της μνήμης RAM του μικροελεγκτή είναι 2KB.

Επίσης, το Arduino UNO έχει και μια μικρή **μνήμη EEPROM** (Electrically Erasable Programmable Read-Only Memory), η οποία χρησιμοποιείται για την αποθήκευση δεδομένων που διατηρούνται ακόμα και αν ο μικροελεγκτής είναι απενεργοποιημένος. Η μέγιστη χωρητικότητα της μνήμης EEPROM του είναι 1KB.

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32KB (ATmega328P), 0.5 KB used by bootloade
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz

Εικόνα 30: Χαρακτηριστικά ATmega328p [7]

4.2 Παράδειγματα Εκτέλεσης Λειτουργικού Συστήματος Πραγματικού Χρόνου

Το πρόγραμμα που ακολουθεί είναι γραμμένο σε λειτουργικό σύστημα πραγματικού χρόνου FreeRTOS και δημιουργεί νήματα ή αλλιώς διεργασίες, εκτελώντας παράλληλα διαφορετικές λειτουργίες στο μικροελεγκτή Arduino UNO.

Πιο συγκεκριμένα, το πρόγραμμα δημιουργεί τρεις διεργασίες οι οποίες κάνουν τα εξής:

1. **TaskLed**: Αναβοσβήνει το ενσωματωμένο LED 13 κάθε 1 δευτερόλεπτο,
2. **TaskAnalogRead**: Διαβάζει την αναλογική τιμή από τον ακροδέκτη A0 και την αποθηκεύει στη μεταβλητή AnalogValue
3. **TaskPrint**: Εκτυπώνει την τιμή της μεταβλητής AnalogValue στην σειριακή θύρα κάθε 100μς, καθώς και την τρέχουσα χρονική στιγμή (με την micros() συνάρτηση).

Η συνάρτηση setup() ρυθμίζει τις απαραίτητες ρυθμίσεις και δημιουργεί τις διεργασίες, ενώ η συνάρτηση loop() είναι κενή καθώς δεν χρειάζεται να εκτελεστεί κάποια ενέργεια στο κύριο νήμα του προγράμματος.

```
//Χρησιμοποιείται λειτουργικό σύστημα πραγματικού χρόνου
#include <Arduino_FreeRTOS.h>
int AnalogValue;
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
```

```
  pinMode(A0, INPUT);
//ΔΗΜΙΟΥΡΓΙΑ ΝΗΜΑΤΩΝ/ΔΙΕΡΓΑΣΙΩΝ
  xTaskCreate(TaskLed, "LedOn", 128, NULL, 1, NULL);
  xTaskCreate(TaskAnalogRead, "AnalogRead", 128, NULL, 1, NULL);
  xTaskCreate(TaskPrint, "Print", 128, NULL, 1, NULL);

  vTaskStartScheduler();
}//ΑΝΑΒΟΣΒΗΝΕΙ ΤΟ LED 13 ΚΑΘΕ 1 ΔΕΥΤΕΡΟΛΕΠΤΟ
void TaskLed(void *pvParameters) {
  while (1) {
    digitalWrite(LED_BUILTIN, HIGH);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    digitalWrite(LED_BUILTIN, LOW);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}//ΔΙΑΒΑΖΕΙ ΤΗΝ ΑΝΑΛΟΓΙΚΗ ΤΙΜΗ ΑΠΟ ΤΟΝ ΑΚΡΟΔΕΚΤΗ Α0 ΚΑΙ ΤΗΝ ΑΠΟΘΗΚΕΥΕΙ
void TaskAnalogRead(void *pvParameters) {
  while (1) {
    AnalogValue = analogRead(A0);
    vTaskDelay(1);
  }
}//ΤΥΠΩΝΕΙ ΣΤΗΝ ΟΘΟΝΗ ΤΗΝ ΤΙΜΗ ΤΗΣ ΜΕΤΑΒΛΗΤΗΣ ΚΑΘΕ 100 μς
void TaskPrint(void *pvParameters) {
  while (1) {
    Serial.print("Analog Value---->");
    Serial.println(AnalogValue);
    vTaskDelay(100 / portTICK_PERIOD_MS);
    Serial.print("Time---->");
    Serial.println(micros());
  }
}
void loop() {}
```

```
Έξοδος:
Time---->100284
Analog Value---->394
Time---->199796
Analog Value---->389
Time---->299304
Analog Value---->384
Time---->398812
Analog Value---->379
Time---->498324
Analog Value---->375
Time---->597832
Analog Value---->371
Time---->697352
```

Όπως φαίνεται στην έξοδο, οι χρονικές προθεσμίες έχουν τηρηθεί, καθώς φαίνεται να τυπώνει στην οθόνη κάθε 100 μς την αναλογική τιμή παρόλο που παράλληλα αναβοσβήνει το LED 13 κάθε 1 δευτερόλεπτο.

Στη συνέχεια, μετατρέπουμε το πρόγραμμα για να εκτελεστεί χωρίς λειτουργικό σύστημα πραγματικού χρόνου εκτελώντας τις ίδιες λειτουργίες. Όπως βλέπουμε δεν είναι πλέον πολυνηματικό και δεν τηρεί τις χρονικές προθεσμίες που είχαν τα νήματα/διεργασίες στην προηγούμενη υλοποίηση με το FreeRTOS. Αυτό σημαίνει ότι οι λειτουργίες του προγράμματος θα εκτελούνται σειριακά και η διαδικασία μέτρησης της αναλογικής τιμής μπορεί να επηρεαστεί από τις καθυστερήσεις σε άλλες λειτουργίες.

```
//Δεν χρησιμοποιείται λειτουργικό σύστημα πραγματικού χρόνου
int AnalogValue;
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(A0, INPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
  AnalogValue = analogRead(A0);
  Serial.print("Analog Value---->");
  Serial.println(AnalogValue);
  delay(100);
  Serial.print("Time---->");
  Serial.println(micros());
}
```

```
Έξοδος:
Analg Value---->508
Time---->2100596
Analog Value---->484
Time---->2100596
Analog Value---->407
Time---->4201432
Analog Value---->365
Time---->6302276
```

Όπως φαίνεται στους χρόνους, λόγω της καθυστέρησης 1000 μς που αναβοσβήνει το led, επηρεάζεται τελικά και ο χρόνος της αναλογικής τιμής αφού πλέον δεν τυπώνει κάθε 100 μς την τιμή αυτή.

Ένα πρόγραμμα που δεν είναι πραγματικού χρόνου, δεν σημαίνει απαραίτητα ότι δεν είναι και πολυνηματικό. Τα προγράμματα που ακολουθεί θα αναδείξει το πλεονέκτημα που έχει ένα λειτουργικό σύστημα πραγματικού χρόνου σε σύγκριση με ένα απλό πρόγραμμα που δεν είναι πραγματικού χρόνου. Και στις δύο εκδοχές θα υπάρχουν νήματα που εκτελούνται παράλληλα συγχρονίζοντάς τα ένας δυαδικός σηματοφόρος.

Το πρόγραμμα δημιουργεί τρεις διεργασίες. Οι δύο διεργασίες είναι σε κατάσταση *αποκλεισμού* με διαφορετική προτεραιότητα η κάθε μία ενώ η τρίτη διεργασία εκτελείται 4 φορές και ξεκινάει πρώτη καθώς είναι έτοιμη να εκτελεστεί. Την τέταρτη φορά αυξάνεται ο σηματοφόρος με σκοπό να εκτελεστεί η επόμενη διεργασία. Η διεργασία που θέλουμε να εκτελεστεί μετά τον σηματοφόρο είναι αυτή που έχει υψηλή προτεραιότητα. Στη συνέχεια όταν τελειώσει τη δουλεία της, μπορεί να εκτελεστεί η διεργασία με την χαμηλή προτεραιότητα. Πιο συγκεκριμένα, οι διεργασίες κάνουν το εξής:

1. Thread1: Εκτελείται η διεργασία 4 φορές. Την τέταρτη φορά αυξάνει τον σηματοφόρο για να εκτελεστεί η επόμενη διεργασία.
2. Thread2: Η διεργασία είναι σε κατάσταση *αποκλεισμού* μέχρι να αποκτήσει τον σηματοφόρο. Η διεργασία αυτή έχει χαμηλή προτεραιότητα.
3. Thread3: Η διεργασία είναι και αυτή σε κατάσταση *αποκλεισμού* μέχρι να αποκτήσει τον σηματοφόρο. Η διεργασία αυτή έχει υψηλή προτεραιότητα.

```
//Χρησιμοποιείται λειτουργικό σύστημα πραγματικού χρόνου
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
SemaphoreHandle_t semaphore;

void thread1(void* pvParameters) {
  while (1) {
    for (int i = 0; i <= 3 ; i++) {
      vTaskDelay(1000 / portTICK_PERIOD_MS);
      Serial.println("Thread is running");
      if (i == 3)
        xSemaphoreGive(semaphore);
    }
  }
  vTaskDelay(1000 / portTICK_PERIOD_MS);
}
//Διεργασία χαμηλής προτεραιότητας
void thread2(void* pvParameters) {
  while (1) {
    if (xSemaphoreTake(semaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("Low Priority");
      xSemaphoreGive(semaphore);
    }
```

```
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
//Διεργασία υψηλής προτεραιότητας
void thread3(void* pvParameters) {
  while (1) {
    if (xSemaphoreTake(semaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("High Priority");
      xSemaphoreGive(semaphore);
    }
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
void setup() {
  Serial.begin(9600);
  semaphore = xSemaphoreCreateBinary();
//Επιλέγονται οι προτεραιότητες των διεργασιών
  xTaskCreate(thread1, "Thread 1", 128, NULL, 1, NULL);
  xTaskCreate(thread2, "Thread 2", 128, NULL, 2, NULL);
  xTaskCreate(thread3, "Thread 3", 128, NULL, 3, NULL);

  vTaskStartScheduler();
}
void loop() {}
```

Στην έξοδο κατά την εκτέλεση του προγράμματος οι διεργασίες εκτελούνται με αυτή την σειρά:

```
Έξοδος:
Thread is running
Thread is running
Thread is running
Thread is running
High Priority
Low Priority
```

Βλέπουμε ότι από τότε που αυξηθεί η τιμή του σηματοφόρου μέσω της διεργασίας Thread 1 (Thread is running), αμέσως θα αποκτήσει τον σηματοφόρο η διεργασία Thread 3 με σκοπό να εκτελεστεί πιο γρήγορα καθώς θεωρείται πιο σημαντική. Όταν θα ολοκληρωθεί, τότε και μόνο θα εκτελεστεί η διεργασία Thread 2 που έχει χαμηλή προτεραιότητα.

Τώρα θα εκτελεστεί το πρόγραμμα όταν δεν χρησιμοποιούμε λειτουργικό σύστημα πραγματικού χρόνου.

```
//Δεν χρησιμοποιείται λειτουργικό σύστημα πραγματικού χρόνου
#include <stdio.h>
#include <stdlib.h>
```

```
#include <pthread.h>
#include <semaphore.h>
sem_t semaphore;

void* thread1(void* arg) {
  while (1) {
    for (int i = 0; i <= 3; i++) {
      sleep(1); // Καθυστέρηση 1 δευτερολέπτου
      printf("Thread is running\n");
      if (i == 3) {
        sem_post(&semaphore);
      }
    }
  }
  return NULL;
}
//Διεργασία χαμηλής προτεραιότητας
void* thread2(void* arg) {
  while (1) {
    sem_wait(&semaphore);
    printf("Low Priority\n");
    sem_post(&semaphore);
    sleep(1); // Καθυστέρηση 1 δευτερολέπτου
  }
  return NULL;
}
//Διεργασία υψηλής προτεραιότητας
void* thread3(void* arg) {
  while (1) {
    sem_wait(&semaphore);
    printf("High Priority\n");
    sem_post(&semaphore);
    sleep(1); // Καθυστέρηση 1 δευτερολέπτου
  }
  return NULL;
}

int main() {
  pthread_t t1, t2, t3;
  sem_init(&semaphore, 0, 0);

//Δεν ρυθμίζονται προτεραιότητες
  pthread_create(&t1, NULL, thread1, NULL);
  pthread_create(&t2, NULL, thread2, NULL);
  pthread_create(&t3, NULL, thread3, NULL);

  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
```

```
  pthread_join(t3, NULL);
  sem_destroy(&semaphore);
  return 0;
}
```

Στην έξοδο κατά την εκτέλεση του προγράμματος οι διεργασίες εκτελούνται με αυτή τη σειρά:

```
Έξοδος:
Thread is running
Thread is running
Thread is running
Thread is running
Low Priority
High Priority
```

Βλέπουμε ότι δεν τηρείται η σειρά με την οποία θέλουμε να εκτελεστούν οι διεργασίες. Όταν η διεργασία Thread1 δώσει τον σηματοφόρο, δεν είναι σίγουρο για το ποια θα είναι η επόμενη διεργασία που θα εκτελεστεί. Υπάρχει πιθανότητα να εκτελεστεί η διεργασία με υψηλή προτεραιότητα πρώτα αλλά μπορεί να εκτελεστεί πρώτα η διεργασία με χαμηλή προτεραιότητα. Επομένως η εκτέλεση των διεργασιών σε ένα σύστημα που δεν είναι πραγματικού χρόνου είναι τυχαία με αποτέλεσμα μια σημαντική διεργασία να μην εκτελεστεί στην ώρα της.

5. ΠΡΑΚΤΙΚΟ ΜΕΡΟΣ: Σύγκριση Απόδοσης του FreeRTOS και του ChibiOS στον μικροελεγκτή ATmega328p

Παρακάτω θα γίνει σύγκριση των δύο Λειτουργικών Συστημάτων Πραγματικού Χρόνου, FreeRTOS και ChibiOS, σχετικά με τη μνήμη που απαιτούν (μνήμη Φλας, RAM) και τον χρόνο εκτέλεσης σε μικροδευτερόλεπτα (μς) πάνω στο μικροελεγκτή ATmega328p.

Για την σύγκριση των λειτουργικών συστημάτων αυτών, χρειάστηκαν τα εξής:

- Πλατφόρμα Arduino UNO (με τον μικροελεγκτή ATmega328p) που λειτουργεί στα 16MHZ
- Arduino IDE με εγκατεστημένες τις βιβλιοθήκες FreeRTOS και ChibiOS για την δημιουργία του κώδικα.
- Χρησιμοποίηση της συνάρτησης micros() και βοηθητικών μεταβλητών (startTime, endTime, duration) στα προγράμματα για να γίνουν οι μετρήσεις που αφορούν τον χρόνο.
- Υπολογισμός κατανάλωσης μνήμης Flash και Ram από το πρόγραμμα Arduino IDE. Οι βοηθητικές μεταβλητές που χρειάστηκαν για τον υπολογισμό χρόνου (startTime, endTime, duration), δεν προσμετρήθηκαν για τον υπολογισμό της κατανάλωσης μνήμης.

Οι συγκρίσεις θα πραγματοποιηθούν σε διάφορες περιπτώσεις όπως η εναλλαγή περιβάλλοντος (Context Switch), η επαναφορά διεργασίας μετά από Context Switch, η καθυστέρηση εξυπηρέτηση διακοπής, η καθυστέρηση έναρξης εκτέλεσης μετά από διακοπή, η χρήση των ουρών και η αντιστροφή προτεραιότητας. Στη συνέχεια θα γίνουν συγκρίσεις στο Jitter (χρονικές διαταραχές) όπως επίσης και στον χρόνο ολοκλήρωσης διεργασιών με πολύπλοκες μαθηματικές πράξεις που απαιτούν χρόνο για να ολοκληρωθούν σε κάθε διεργασία.

5.1 Μετρήσεις

Βασικές Μετρήσεις

- **Context Switch (Εναλλαγή Περιβάλλοντος)**

Έχουμε 2 διεργασίες Task1 και Task2. Η Task2 αναλαμβάνει να ανάψει το Led13 ενώ η Task1 αναλαμβάνει να σβήσει το Led13 συγχρονίζοντάς τες ένας σηματοφόρος. Το Led αναβοσβήνει σε πολύ γρήγορο ρυθμό που δεν φαίνεται η διαφορά. Η μέτρηση χρόνου ξεκινάει από την

στιγμή που η Task2 σταματάει την εκτέλεσή της (αποθηκεύει τις πληροφορίες της στο μπλοκ ελέγχου διεργασιών) και σταματάει όταν ξεκινήσει η Task1 την εκτέλεσή της (φορτώνει τις πληροφορίες από το μπλοκ).

```
//FreeRTOS
void Task1(void *pvParameters) {
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    endTime = micros();
    duration = endTime - startTime;
    digitalWrite(13, LOW);
    Serial.println(duration);
  }
}
void Task2(void *pvParameters) {
  while (1) {
    digitalWrite(13, HIGH);
    startTime = micros();
    xSemaphoreGive(xSemaphore);
  }
}
//ChibiOS
static THD_FUNCTION(Task1, arg) {
  while (1) {
    chBSemWait(&xSemaphore);
    endTime=micros();
    duration=endTime-startTime;
    digitalWrite(13, LOW);
    Serial.println(duration);

  }
}
static THD_FUNCTION(Task2, arg) {
  while (1) {
    digitalWrite(13, HIGH);
    startTime=micros();
    chBSemSignal(&xSemaphore);
  }
}
```

- **Επαναφορά Διεργασίας**

Έχουμε 2 διεργασίες Task1 και Task2. Η Task2 αναλαμβάνει να ανάψει το Led13 ενώ η Task1 αναλαμβάνει να σβήσει το Led13 συγχρονίζοντάς τες ένας σηματοφόρος. Το Led αναβοσβήνει σε πολύ γρήγορο ρυθμό που δεν φαίνεται η διαφορά. Η μέτρηση χρόνου ξεκινάει από την στιγμή που η Task2 σταματάει την εκτέλεσή της (αποθηκεύει τις πληροφορίες της στο μπλοκ

ελέγχου διεργασιών), για να εκτελεστεί η Task1, και σταματάει όταν έχει ήδη γίνει η επαναφορά της Task 2 και έχουν επαναφερθεί οι πληροφορίες από το μπλοκ.

```
//FreeRTOS
void Task1(void *pvParameters) {
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    digitalWrite(13, LOW);
  }
}
void Task2(void *pvParameters) {
  while (1) {
    digitalWrite(13, HIGH);
    startTime = micros(); //Σταματάει προσωρινά η εκτέλεση της Task2
    xSemaphoreGive(xSemaphore);
    endTime = micros(); //Επιστρέφει η Task2
    duration = endTime - startTime;
    Serial.println(duration);
  }
}
```

```
//ChibiOS
static THD_FUNCTION(Task1, arg) {
  while (1) {
    chBSemWait(&xSemaphore);
    digitalWrite(13, LOW);
  }
}
static THD_FUNCTION(Task2, arg) {
  while (1) {
    digitalWrite(13, HIGH);
    startTime = micros(); //Σταματάει προσωρινά η εκτέλεση της Task2
    chBSemSignal(&xSemaphore);
    endTime = micros(); //Επιστρέφει η Task2
    duration = endTime - startTime;
    Serial.println(duration);
  }
}
```

- **<u>Καθυστέρηση Εξυπηρέτησης Διακοπής</u>**

Έχουμε μια διεργασία MainTask που ανάβει το Led και μία διεργασία *χειριστής διακοπών* HandlerTask που ενώ έχει υψηλότερη προτεραιότητα δεν εκτελείται διότι είναι σε κατάσταση *Αποκλεισμού* μέχρι να αποκτήσει τον σηματοφόρο. Η διεργασία αυτή αναλαμβάνει να σβήσει το led όταν θα αποκτήσει τον σηματοφόρο. Επίσης έχουμε και την *ρουτίνα εξυπηρέτησης διακοπών* Isr που όταν εκτελεστεί θα δώσει τον σηματοφόρο στην διεργασία HandlerTask. Η

συνάρτηση Isr θα εκτελεστεί όταν συμβεί μια διακοπή κατά την διάρκεια εκτέλεσης του προγράμματος. Η μέτρηση χρόνου ξεκινάει από την δημιουργία του σήματος διακοπής και σταματάει μέχρι να αρχίσει να εκτελείται η διεργασία HandlerTask. Για την ευκολία της μέτρησης, στο πρόγραμμα έχει ρυθμιστεί η διακοπή να συμβαίνει σε ένα συγκεκριμένο χρονικό διάστημα μέσω του ακροδέκτη pin 2 στο Arduino UNO όταν πάρει την τιμή HIGH ενώ θα σταματήσει η διακοπή όταν θα έχει την τιμή LOW.

```
//FreeRTOS
void HandlerTask(void *pvParameters) { //Χειριστής Διακοπών
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    endTime = micros();
    digitalWrite(13, LOW);
    duration = endTime - startTime;
    Serial.println(duration);
  }
}
void MainTask(void *pvParameters) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    startTime = micros();
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
    vTaskDelay(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  xSemaphoreGiveFromISR(xSemaphore, NULL);
  portYIELD_FROM_ISR();
}
```

```
//ChibiOS
static THD_FUNCTION(HandlerTask, arg) { //Χειριστής Διακοπών
  while (1) {
    chBSemWait(&xSemaphore);
    endTime = micros();
    digitalWrite(13, LOW);
    duration = endTime - startTime;
    Serial.println(duration);
  }
}
static THD_FUNCTION(MainTask, arg) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    startTime = micros();
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
```

```
    chThdSleep(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  CH_IRQ_PROLOGUE();
  chSysLockFromISR();
  chBSemSignalI(&xSemaphore);
  chSysUnlockFromISR();
  CH_IRQ_EPILOGUE();
}
```

- **<u>Καθυστέρηση έναρξης εκτέλεσης διεργασίας μετά από διακοπή</u>**

Έχουμε πάλι μια διεργασία MainTask, HandlerTask (Χειριστής Διακοπών) και ρουτίνα εξυπηρέτησης διακοπών. Η μέτρηση χρόνου ξεκινάει από την στιγμή που τελείωσε ο χειριστής διακοπών και σταματάει όταν το πρόγραμμα επανέλθει στο σημείο που είχε συνέβη η διακοπή (δηλαδή η στιγμή που έχει χρονολογηθεί να εκτελεστεί η επόμενη διεργασία MainTask μετά την διακοπή).

```
//FreeRTOS
void HandlerTask(void *pvParameters) { //Χειριστής Διακοπών
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    digitalWrite(13, LOW);
    startTime = micros();
  }
}
void MainTask(void *pvParameters) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
    endTime = micros();
    duration = endTime - startTime;
    Serial.println(duration);
    vTaskDelay(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  xSemaphoreGiveFromISR(xSemaphore, NULL);
  portYIELD_FROM_ISR();
}
```

```
//ChibiOS
static THD_FUNCTION(HandlerTask, arg) { //Χειριστής Διακοπών
  while (1) {
    chBSemWait(&xSemaphore);
```

```
    digitalWrite(13, LOW);
    startTime = micros();
  }
}
static THD_FUNCTION(MainTask, arg) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
    endTime = micros();
    duration = endTime - startTime;
    Serial.println(duration);
    chThdSleep(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  CH_IRQ_PROLOGUE();
  chSysLockFromISR();
  chBSemSignalI(&xSemaphore);
  chSysUnlockFromISR();
  CH_IRQ_EPILOGUE();
}
```

- **Ρουτίνα εξυπηρέτησης διακοπής**

Έχουμε πάλι μια διεργασία MainTask, HandlerTask (Χειριστής Διακοπών) και ρουτίνα εξυπηρέτησης διακοπών. Η μέτρηση χρόνου ξεκινάει από την στιγμή που τελείωσε συνέβη μια διακοπή και τελειώνει στο σημείο που εκτελείται η ρουτίνα εξυπηρέτησης διακοπής(ISR)

```
//FreeRTOS
void HandlerTask(void *pvParameters) { //Χειριστής Διακοπών
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    digitalWrite(13, LOW);
  }
}
void MainTask(void *pvParameters) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    startTime = micros();
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
    vTaskDelay(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  endTime = micros();
  duration = endTime - startTime;
  Serial.println(duration);
```

```
  xSemaphoreGiveFromISR(xSemaphore, NULL);
  portYIELD_FROM_ISR();
}
```

```
//ChibiOS
static THD_FUNCTION(HandlerTask, arg) { //Χειριστής Διακοπών
  while (1) {
    chBSemWait(&xSemaphore);
    digitalWrite(13, LOW);
  }
}
static THD_FUNCTION(MainTask, arg) {
  while (1) {
    digitalWrite(2, LOW);  //ΣΤΑΜΑΤΑΕΙ Η ΔΙΑΚΟΠΗ
    digitalWrite(13, HIGH);
    startTime = micros();
    digitalWrite(2, HIGH);  //ΣΥΜΒΑΙΝΕΙ ΜΙΑ ΔΙΑΚΟΠΗ
    chThdSleep(1);
  }
}
void Isr() { //Ρουτίνα Εξυπηρέτησης Διακοπής
  endTime = micros();
  duration = endTime - startTime;
  Serial.println(duration);
  CH_IRQ_PROLOGUE();
  chSysLockFromISR();
  chBSemSignalI(&xSemaphore);
  chSysUnlockFromISR();
  CH_IRQ_EPILOGUE();
}
```

- **<u>Αφαίρεση δεδομένων από ουρά</u>**

Έχουμε 2 διεργασίες και μία ουρά. Η πρώτη διεργασία αποθηκεύει την ακέραιο της αναλογικής τιμής του ακροδέκτη Α0 σε μία μεταβλητή και τον στέλνει στην ουρά. Η δεύτερη διεργασία παίρνει τον ακέραιο από την ουρά και τον αποθηκεύει σε μία τοπική μεταβλητή της διεργασίας. Η μέτρηση ξεκινάει από την στιγμή που ο ακέραιος μπαίνει στην ουρά και τελειώνει μέχρι και την στιγμή που βγαίνει από την ουρά και αποθηκεύεται στην μεταβλητή.

```
//FreeRTOS
void TaskAnalogRead(void *pvParameters) {
  while (1) {
    int analogValue = analogRead(A0);
    startTime = micros();
    xQueueSend(Queue, &analogValue, portMAX_DELAY);
  }
}
void TaskSerial(void *pvParameters) {
```

```
  int valueQueue;
  while (1) {
    if (xQueueReceive(Queue, &valueQueue, portMAX_DELAY) == pdPASS) {
      endTime = micros();
      duration = endTime - startTime;
      Serial.print("Time--->");
      Serial.println(duration);

      Serial.print("AnalogeValue-->");
      Serial.println(valueQueue);
    }
  }
}
```

```
//ChibiOS
static THD_FUNCTION(TaskAnalogRead, arg) {
  while (1) {
    int analogValue = analogRead(A0);
    startTime = micros();
    chMBPostTimeout(&Queue, (msg_t)analogValue, TIME_INFINITE);
  }
}
static THD_FUNCTION(TaskSerial, arg) {
  int valueQueue;
  while (1) {
    if (chMBFetchTimeout(&Queue, (msg_t *)&valueQueue, TIME_INFINITE)
== MSG_OK) {
      endTime = micros();
      duration = endTime - startTime;
      Serial.print("Time--->");
      Serial.println(duration);

      Serial.print("AnalogeValue-->");
      Serial.println(valueQueue);
    }
  }
}
```

- **<u>Αντιστροφή Προτεραιότητας</u>**

Έχουμε 2 διεργασίες με διαφορετικές προτεραιότητες. Η διεργασία Task2 έχει υψηλή προτεραιότητα και η διεργασία Task1 έχει χαμηλή προτεραιότητα. Ξεκινάει πρώτα η Task2 και μπαίνει σε κατάσταση *αποκλεισμού* και έτσι εκτελείται η Task1. Η διεργασία Task1 κλειδώνει το Mutex για την κρίσιμη περιοχή που είναι η μεταβλητή "data" και μπαίνει και αυτή σε κατάσταση *αποκλεισμού*, και έτσι δεν έχει ξεκλειδώσει ακόμα το mutex. Η Task2 επαναφέρεται, ανάβει το LED και προσπαθεί να κλειδώσει το mutex. Αποτυγχάνει όμως γιατί

η Task1 σταμάτησε την εκτέλεση της ενώ το mutex ήταν ήδη κλειδωμένο χωρίς να το ξεκλειδώσει. Έτσι η Task1 από χαμηλή προτεραιότητα, αποκτά υψηλή προτεραιότητα και έτσι ο έλεγχος επιστρέφει στη Task1 ξεκλειδώνοντας το mutex και στην συνέχεια αποκαθίστανται οι προτεραιότητες. Εφόσον το mutex ξεκλειδώθηκε, η Task 2 επανέρχεται, κλειδώνει με επιτυχία το mutex και σβήνει το LED. Η μέτρηση χρόνου ξεκινάει από την στιγμή που ανάβει το LED πριν προσπαθήσει η Task2 να κλειδώσει το Mutex και σταματάει όταν σβήνει το LED αφού έχει κλειδωθεί το mutex.

```
//FreeRTOS
void Task1(void *pvParameters) {
  while (1) {
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    data = 1;
    vTaskDelay(1);
    xSemaphoreGive(xSemaphore);
  }
}
void Task2(void *pvParameters) {
  while (1) {
    vTaskDelay(1);
    startTime = micros();
    digitalWrite(13, HIGH);
    xSemaphoreTake(xSemaphore, portMAX_DELAY);
    data = 2;
    digitalWrite(13, LOW);
    endTime = micros();
    duration = endTime - startTime;
    Serial.println(duration);
    xSemaphoreGive(xSemaphore);
  }
}
```

```
//ChibiOS
static THD_FUNCTION(Task1, arg) {
  while (1) {
    chMtxLock(&mutex);
    data = 1;
    chThdSleep(1);
    chMtxUnlock(&mutex);
  }
}
static THD_FUNCTION(Task2, arg) {
  while (1) {
    chThdSleep(1);
    startTime = micros();
    digitalWrite(13, HIGH);
    chMtxLock(&mutex);
```

```
    data = 2;
    digitalWrite(13, LOW);
    endTime = micros();
    duration = endTime - startTime;
    Serial.println(duration);
    chMtxUnlock(&mutex);
  }
}
```

Μέτρηση Jitter (Διακύμανση)

Όπως έχει αναφερθεί, το Jitter είναι η μεταβλητότητα του χρόνου που απαιτείται από το Λειτουργικό Σύστημα για να δεχθεί και να επεξεργαστεί ένα αίτημα εφαρμογής. Παρακάτω θα γίνουν μετρήσεις στο FreeRTOS και στο ChibiOS για να φανούν οι διαφορές που έχουν στο Jitter ανά 100 επαναλήψεις. Όσο μικρότερο είναι το Jitter (χαμηλή μεταβλητότητα στον χρόνο απόκρισης) τόσο πιο σταθερό και αξιόπιστο στην απόκριση χρόνου είναι ένα λειτουργικό σύστημα.

```
//FreeRTOS
void Thread1(void* pvParameters) {
  vTaskDelay(1);
  long start = micros();
  while (1) {
    vTaskDelay(1);
    long end = micros();
    long diff = end - start;
    if (diff < min) min = diff;
    if (diff > max) max = diff;
    start = end;
  }
}
```

```
//ChibiOS
static THD_FUNCTION(Thread1, arg) {
  chThdSleep(1);
  long start = micros();
  Serial.println(start);
  while (1) {
    chThdSleep(1);
    long end = micros();
    long diff = end - start;
    if (diff < min) min = diff;
    if (diff > max) max = diff;
    start = end;
  }
}
```

Μέτρηση Ολοκλήρωσης Διεργασιών (Χρονοπρογραμματισμός)

Έχουμε 3 διεργασίες. Οι δύο διεργασίες καταναλώνουν αρκετούς πόρους καθώς εκτελούν πολύπλοκες μαθηματικές πράξεις με πολλές επαναλήψεις. Η τρίτη διεργασία εμφανίζει τον χρόνο ολοκλήρωσης και των δύο διεργασιών.

- Η διεργασία Task1 χρειάζεται περισσότερο χρόνο για να ολοκληρωθεί από την διεργασία Task2 καθώς εκτελεί περισσότερες πράξεις.
- Οι διεργασίες Task1 και Task2 μόλις ολοκληρωθούν διαγράφονται και δεν εκτελούνται ξανά.
- Όποια διεργασία εκτελείται εκείνη την στιγμή φαίνεται και από το Led13 του Arduino UNO. Όταν εκτελείται η διεργασία Task1 είναι αναμμένο το Led και όταν εκτελείται η διεργασία Task2 είναι σβηστό το led.

```
//FreeRTOS
//Διεργασία με τη μεγαλύτερη διάρκεια
void task1(void* pvParameters) {
  vTaskDelay(1);
  Serial.println("Task1 start");
  double x, y, z, w, v;
  for (int i = 0; i <= 10000; i++) {
    digitalWrite(13, HIGH);
    x = sin(i) * cos(i);
    y = sqrt(x);
    z = pow(y, 3);
    w = log(z);
    v = exp(w);
    if (i == 10000) {
      Serial.println(v);
      Serial.println("Task1 end");
      vTaskDelete(NULL);
    }
  }
}
//Διεργασία με τη μικρότερη διάρκεια
void task2(void* pvParameters) {
  Serial.println("Task2 start");
  float x;
  for (int i = 0; i <= 10000; i++) {
    digitalWrite(13, LOW);
    x = sin(i) * cos(i);
    if (i == 10000) {
      Serial.println(x);
      Serial.println("Task2 end");
      vTaskDelete(NULL);
    }
  }
}
```

```
//ChibiOS
//Διεργασία με τη μεγαλύτερη διάρκεια
static THD_FUNCTION(Task1, arg) {
  chThdSleep(1);
  Serial.println("Task1 start");
  double x, y, z, w, v;
  for (int i = 0; i <= 10000; i++) {
    digitalWrite(13, HIGH);
    x = sin(i) * cos(i);
    y = sqrt(x);
    z = pow(y, 3);
    w = log(z);
    v = exp(w);
    if (i == 10000) {
      Serial.println(v);
      Serial.println("Task1 end");
      chThdExit(0);
    }
  }
}
//Διεργασία με τη μικρότερη διάρκεια
static THD_FUNCTION(Task2, arg) {
  Serial.println("Task2 start");
  double x;
  for (int i = 0; i <= 10000; i++) {
    digitalWrite(13, LOW);
    x = sin(i) * cos(i);
    if (i == 10000) {
      Serial.println(x);
      Serial.println("Task2 end");
      chThdExit(0);
    }
  }
}
```

Οι μετρήσεις θα γίνουν σε 2 περιπτώσεις:

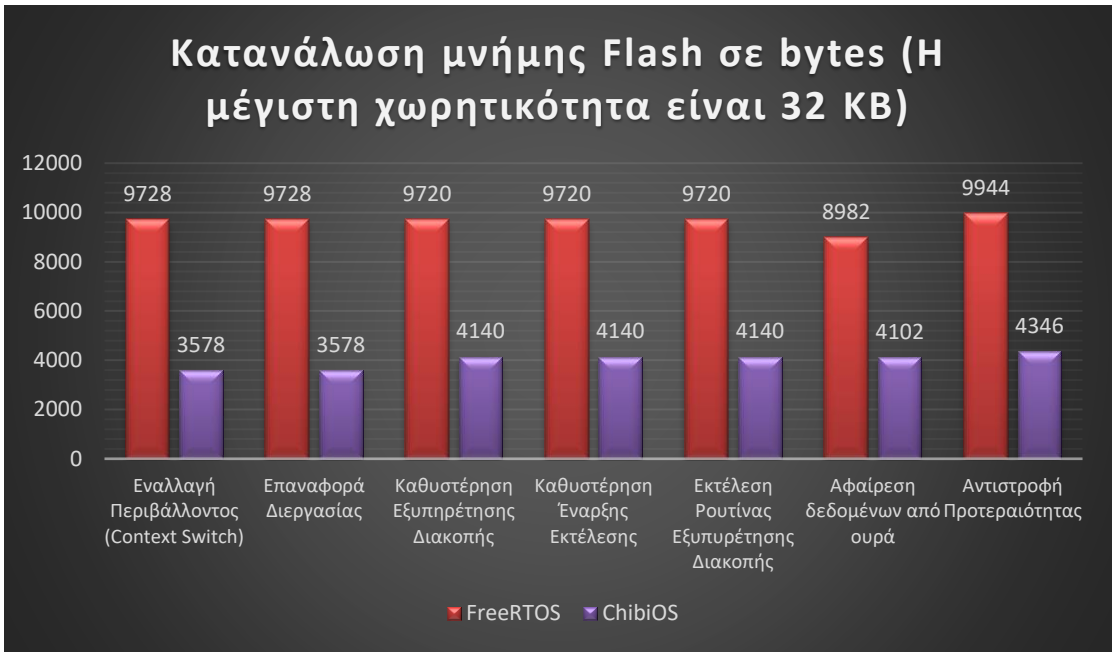
- **<u>Διεργασίες ίσης προτεραιότητας (Round Robin)</u>**: Η Task1 και η Task2 έχουν ίση προτεραιότητα και τα λειτουργικά συστήματα χρησιμοποιούν τον αλγόριθμο Round Robin (χρονοπρογραμματισμό με εξυπηρέτηση εκ περιτροπής). Δηλαδή κάθε διεργασία εκτελείται για ένα κβάντο χρόνου. Μόλις τελειώσει το κβάντο της μίας διεργασίας, σταματάει προσωρινά η εκτέλεσή της και εκτελείται η άλλη διεργασία μέχρι να τελειώσει και το δικό της κβάντο. Αυτή η κατάσταση θα συνεχιστεί μέχρι οι διεργασίες να ολοκληρωθούν. Κατά την εκτέλεση του προγράμματος, βλέπουμε το

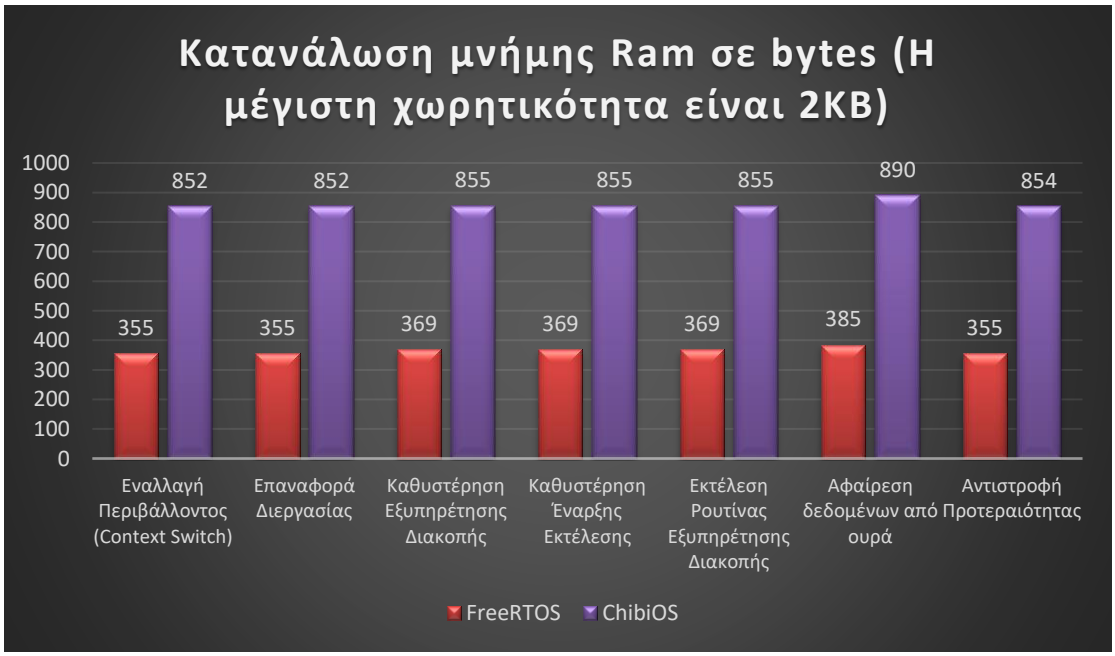
Led 13 να αναβοσβήνει γρήγορα διότι και οι δύο διεργασίες αργούν να ολοκληρωθούν με αποτέλεσμα να εκτελούνται εναλλάξ.

- **Εξυπηρέτηση με βάση τη μικρότερη διάρκεια:** Η διεργασία Task2 έχει μικρότερη διάρκεια ολοκλήρωσης από την διεργασία Task1 επομένως την ρυθμίζουμε να έχει μεγαλύτερη προτεραιότητα έτσι ώστε να ξεκινήσει πρώτη. Το led είναι αρχικά μόνιμα σβηστό καθώς εκτελείται η διεργασία Task2 μέχρι ολοκληρωθεί και στην συνέχεια είναι αναμμένο αφού εκτελείται η διεργασία Task1.

5.2 Αποτελέσματα Μετρήσεων

Αποτελέσματα Βασικών Μετρήσεων

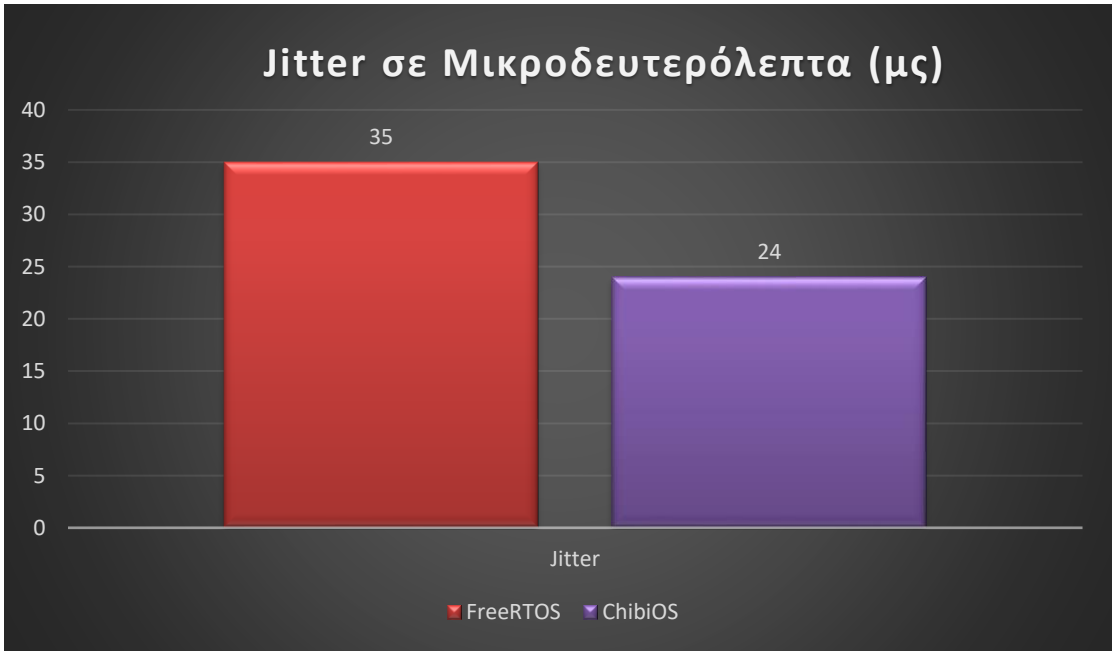




Οι μετρήσεις για τις βασικές λειτουργίες των FreeRTOS και ChibiOS έδειξαν ότι:

- Το FreeRTOS καταναλώνει περισσότερη μνήμη φλας από το ChibiOS
- Το FreeRTOS καταναλώνει λιγότερη μνήμη RAM από το ChibiOS
- Το FreeRTOS είναι πιο αργό στην απόκριση χρόνου από το ChibiOS

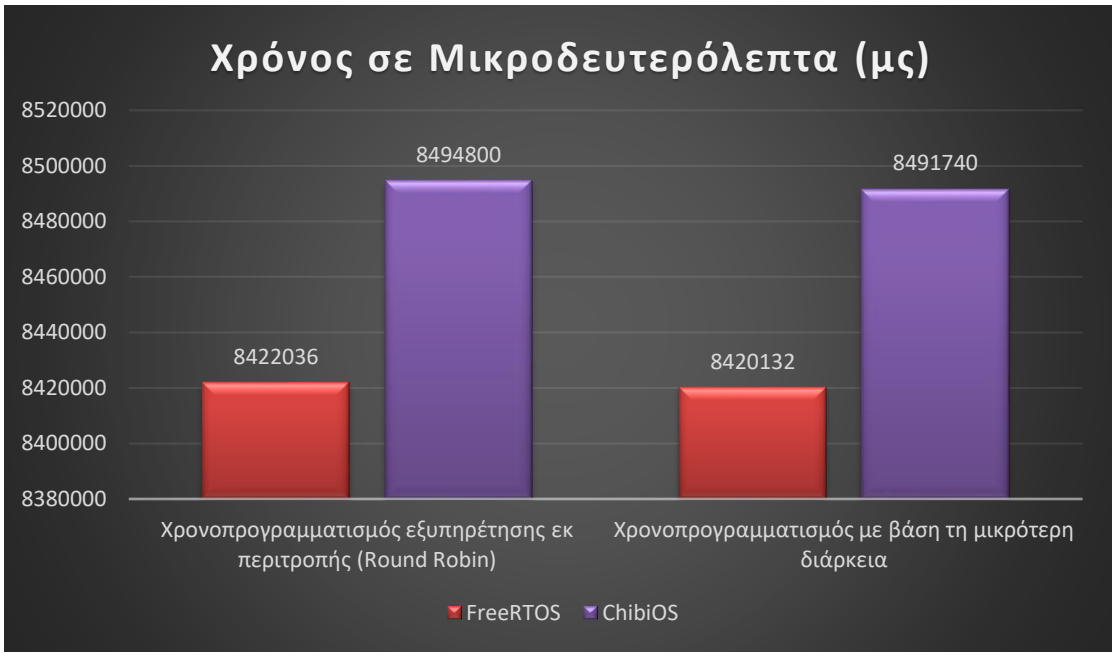
<u>Αποτελέσματα μέτρησης Jitter</u>

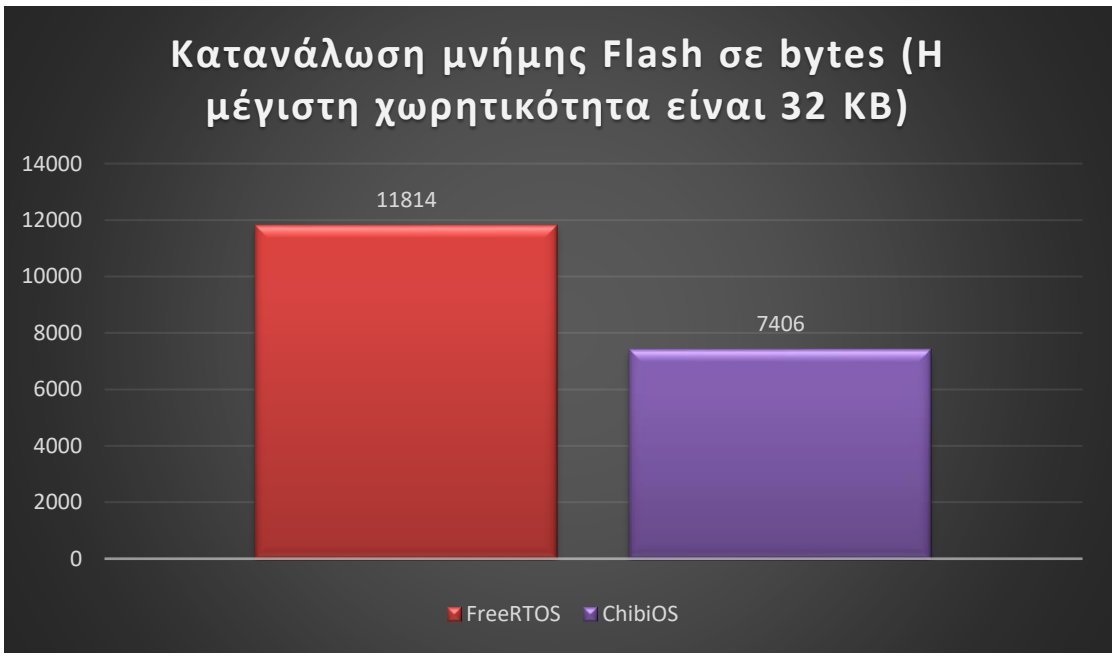


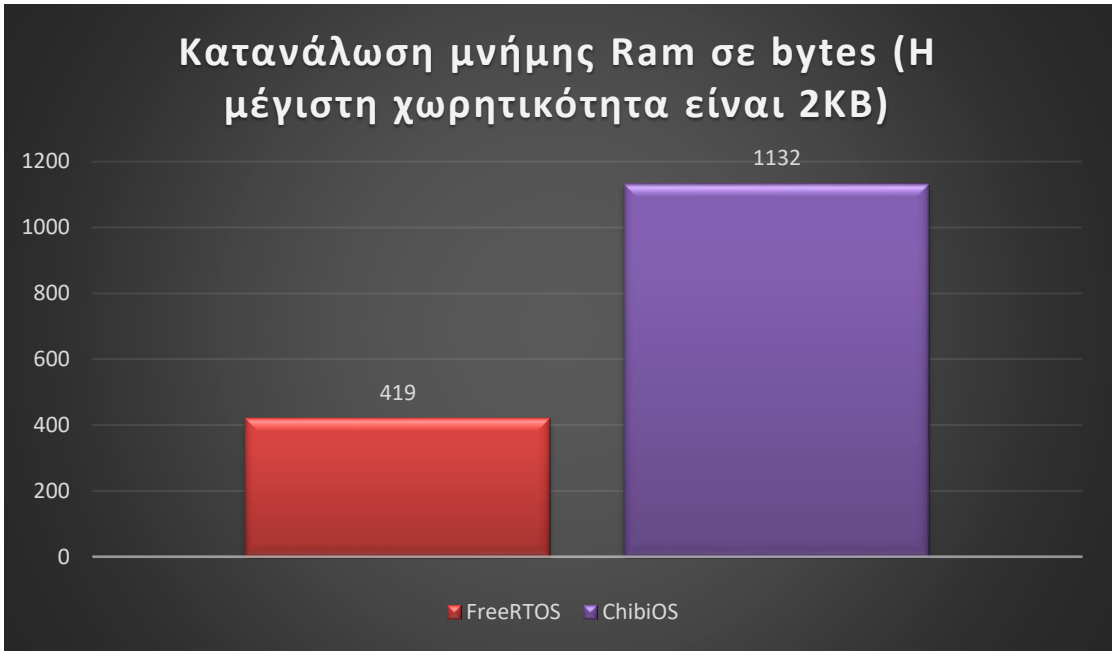
Οι μετρήσεις για το Jitter έδειξαν ότι:

- Το FreeRTOS έχει μεγαλύτερο Jitter από το ChibiOS
- Κατά την διάρκεια εκτέλεσης του προγράμματος, η απόκλιση της τιμής του ελάχιστου και του μέγιστου στο FreeRTOS δεν ήταν σταθερή και προέκυψε ένας μέσος όρος.
- Το Jitter στο ChibiOS ήταν σταθερό σε όλη την διάρκεια εκτέλεσης του προγράμματος και μικρότερο από τον μέσο όρο του FreeRTOS.

<u>Αποτελέσματα μέτρησης ολοκλήρωσης διεργασιών</u>







Οι μετρήσεις που αφορούν την ολοκλήρωση των διεργασιών με πολύπλοκους υπολογισμούς στο ChibiOS και στο FreeRTOS έδειξαν ότι:

- Και οι δύο διεργασίες απαιτούν αρκετό χρόνο για την ολοκλήρωση τους και στα δύο λειτουργικά συστήματα.
- Όταν οι διεργασίες έχουν ίδια προτεραιότητα, το πρόγραμμα αργεί περισσότερο να τελειώσει καθώς εκτελείται με τον αλγόριθμο χρονοπρογραμματισμού εξυπηρέτησης εκ περιτροπής (Round Robin) όπου σε κάθε εναλλαγή από την μία διεργασία στην άλλη απαιτούνται κάποια παραπάνω μικροδευτερόλεπτα.
- Και στις δύο περιπτώσεις (εξυπηρέτησης εκ περιτροπής, εξυπηρέτηση με βάση τη μικρότερή διάρκεια), το FreeRTOS ολοκληρώνει πιο γρήγορα τις διεργασίες αφού φαίνεται πως το ChibiOS είναι πιο αργό στους πολύπλοκους υπολογισμούς που έχουν οι δύο διεργασίες.

Συμπεράσματα

Στο πλαίσιο της διπλωματικής εργασίας, η συγκριτική μελέτη λειτουργικών συστημάτων πραγματικού χρόνου σε μικροελεγκτή ATmega328p, αποκάλυψε ότι το δημοφιλές FreeRTOS έχει καλύτερη απόδοση όσον αφορά την κατανάλωση μνήμης RAM. Αυτό καθιστά το FreeRTOS κατάλληλο για μεγάλες εφαρμογές λόγω χαμηλής κατανάλωσης μνήμης.
Αντίθετα, όσον αφορά την μνήμη Flash το FreeRTOS φαίνεται πως καταναλώνει περισσότερο από το ChibiOS. Το FreeRTOS χρησιμοποιεί πολλές έτοιμες δομές δεδομένων οι οποίες απαιτούν περισσότερη μνήμη Flash για να αποθηκευτούν. Από την άλλη πλευρά, το ChibiOS χρησιμοποιεί μια πιο συμπαγή δομή καταστάσεων καθώς και έναν αποδοτικό μηχανισμό διαχείρισης μνήμης που χρησιμοποιεί λιγότερο Flash, αλλά απαιτεί περισσότερη μνήμη RAM για την εκτέλεση των διεργασιών.

Επιπλέον, όσον αφορά τον χρόνο ολοκλήρωσης των διεργασιών, το ChibiOS είναι πιο αργό αν υπάρχουν πολύπλοκοι υπολογισμοί. Σε τομείς όμως όπως η εναλλαγή περιβάλλοντος, η επαναφορά διεργασίας, η μεταφορά δεδομένων με τη χρήση ουρών, η καθυστέρηση εξυπηρέτησης διακοπής, η αντιστροφή προτεραιότητας και το Jitter, το ChibiOS επιδεικνύει ανώτερες επιδόσεις και είναι κατάλληλο για εφαρμογές που απαιτούν γρήγορη απόκριση σε πραγματικό χρόνο αλλά όμως και για μικρές εφαρμογές λόγω της υψηλής κατανάλωσης μνήμης.

Τέλος, παρόλο που το ChibiOS επιδεικνύει ανώτερες επιδόσεις σε βασικούς τομείς των λειτουργικών συστημάτων όσον αφορά τον χρόνο, αυτό δεν σημαίνει ότι το FreeRTOS είναι κακό σε αυτούς τους τομείς. Και τα δύο λειτουργικά συστήματα μπορούν να εκτελέσουν εργασίες σε πραγματικό χρόνο, απλά έχουν διαφορετικές δυνατότητες και περιορισμούς.

Βιβλιογραφία

1. Σύγχρονα λειτουργικά συστήματα 3η αμερικανική έκδοση, Andrew S. Tanenbaum
2. Συστήματα Πραγματικού Χρόνου, Κωνσταντίνος Βασιλάκης - Δημήτρης Μαρούλης
3. Ο Προγραμματισμός του Μικροελεγκτή AVR ATmega328 με τη Χρήση της Πλατφόρμας Arduino, Δημήτρης Καλοφωλιάς
4. https://www.freertos.org/Documentation/RTOS_book.html
5. https://en.wikipedia.org/wiki/FreeRTOS#/media/File:Logo_freeRTOS.png
6. https://www.ej-eng.org/index.php/ejeng/article/view/2322/1027
7. https://www.researchgate.net/figure/Arduino-Uno-R3-Technical-specifications_tbl3_320678859
8. http://meleththrio.teicm.gr/xmlui/bitstream/handle/123456789/22/Real_time_total_new.pdf;jsessionid=75BB497AC10F7727B3383A783EA6F384?sequence=1
9. https://www.chibios.org/dokuwiki/doku.php?id=chibios:documentation:books:rt:kernel
10. https://www.chibios.org/dokuwiki/doku.php?id=chibios:documentation:start
11. Μικροϋπολογιστές – RTOS, Σημειώσεις (Πανεπιστήμιο Δυτικής Αττικής) - Στέλιος Βουτσινάς
12. https://store.arduino.cc/products/arduino-uno-rev3
13. https://en.wikipedia.org/wiki/ATmega328#/media/File:ATMEGA328P-PU.jpg
14. https://www.chibios.org/dokuwiki/doku.php?id=chibios:documentation:books:rt:embedded
15. https://en.wikipedia.org/wiki/ChibiOS/RT
16. https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf
17. Ενσωματωμένα Λειτουργικά Συστήματα και Λειτουργικά Συστήματα Πραγματικού Χρόνου, Δημήτριος Σούντρης
18. http://eclass.teipir.gr/openeclass/modules/document/file.php/HYS100/IV.%20%CE%A3%CF%85%CF%83%CF%84%CE%AE%CE%BC%CE%B1%CF%84%CE%B1%20%CE%A0%CF%81%CE%B1%CE%B3%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%BF%CF%8D%20%CE%A7%CF%81%CF%8C%CE%BD%CE%BF%CF%85.pdf
19. https://commons.wikimedia.org/wiki/File:Operating_system_placement-el.svg
20. https://osdn.net/projects/chibios/images/