2007

Εργασία στα Λειτουργικά Συστήματα

Θέμα:

Επιλέξτε ένα οποιοδήποτε RTOS μπορείτε να βρείτε (και να προσομειώσετε ή να εκτελέσετε) και να εκτελέσετε σε αυτό μια εργασία που θα τυπώνει το μήνυμα "Hello World from the RTOS". Να παρουσιάσετε τον τρόπο που διαχειρίζεται το Λειτουργικό Σύστημα τη μνήμη, το χρονοπρογραμματισμό, τα νήματα και ποιους τρόπους επιτρέπει για διαδιεργασιακή επικοινωνία. Στα πλαίσια της εργασίας θα πραγματοποιηθεί μελέτη των RTOSs.

Ιωσηφίδης Ιωάννης - 3927,
Μαλακόπουλος Αργύρης - 3954,
Ψυχογιός Νικόλαος - 3878
Δημοκρίτειο Πανεπιστήμιο Θράκης
8/21/2007

Επιβλέπων: Μηνάς Δασυγένης

Περιεχόμενα

1 Πρόλογος

Η εργασία αυτή πραγματοποιήθηκε στα πλαίσια του μαθήματος Λειτουργικά Συστήματα στην Πολυτεχνική Σχολή του Δημοκριτείου Πανεπιστημίου Θράκης, από τους φοιτητές του 5ου έτους Ιωσηφίδη Ιωάννη, Μαλακόπουλο Αργύρη και Ψυχογιό Νικόλαο. Παραδόθηκε πριν την εξεταστική του Σεπτεμβρίου 2007, έχοντας συνεπικουρικό στη γραπτή εξέταση ρόλο. Η τελευταία πραγματοποιήθηκε από το Δρ. Δασυγένη Μηνά, όπως άλλωστε και η διδασκαλία του μαθήματος.

Το θέμα της εργασίας είναι το ακόλουθο: "*Επιλέξτε ένα οποιοδήποτε RTOS μπορείτε να βρείτε (και να προσομειώσετε ή να εκτελέσετε) και να εκτελέσετε σε αυτό μια εργασία που θα τυπώνει το μήνυμα "Hello World from the RTOS". Να παρουσιάσετε τον τρόπο που διαχειρίζεται το Λειτουργικό Σύστημα τη μνήμη, το χρονοπρογραμματισμό, τα νήματα και ποιους τρόπους επιτρέπει για διαδιεργασιακή επικοινωνία. Στα πλαίσια της εργασίας θα πραγματοποιηθεί μελέτη των RTOS* ".

Ο εξοπλισμός που χρησιμοποιήθηκε ήταν παροχή του εργαστηρίου VLSI της Πολυτεχνικής Σχολής, μέσα στο οποίο διεκπεραιώθηκε και η εργασία. Θα πρέπει τέλος να σημειωθεί ότι η εργασία αυτή, ταυτίζεται με μέρος της έρευνας που επιτελεί ο φοιτητής Ιωσηφίδης Ιωάννης για την περάτωση της διπλωματικής του εργασίας γι'αυτό το λόγο θα χρησιμοποιηθεί τροποποιημένη στην παρουσίαση της τελευταίας.

2 Εισαγωγή

Με την ολοένα και μεγαλύτερη ανάπτυξη των ενσωματωμένων συστημάτων, τα λειτουργικά τους συστήματα αποκτούν περισσότερη σημασία, όπως επίσης περισσότερη πολυπλοκότητα. Η ευελιξία και ευκολία χρήσης που προσφέρουν, καθώς και το υψηλότερο επίπεδο αφαίρεσης που παρέχουν τόσο στον προγραμματιστή εφαρμογών, όσο και το χρήστη, τα καθιστά απαραίτητη ανάγκη για ολοένα και περισσότερες συσκευές, αν και πάντα θα υπάρχουν κάποιες οι οποίες μπορούν να λειτουργήσουν χωρίς αυτά, μόνο με μια ελάχιστη διασύνδεση (για παράδειγμα ένας φούρνος μικροκυμάτων). Τα Λειτουργικά Συστήματα Πραγματικού Χρόνου ή Real Time Operating Systems (από εδώ και στο εξής RTOS), συνιστούν τυπικά μια υποκατηγορία των ενσωματωμένων λειτουργικών συστημάτων, αλλά πρακτικά την πλειοψηφία τους. Αυτά έχουν ως πιο βασική παράμετρο το χρόνο απόκρισης του λειτουργικού συστήματος και την προβλεψιμότητα σε κάθε πτυχή του.

Αρχικά, θα γίνει μια συνοπτική περιγραφή των Λειτουργικών Συστημάτων Πραγματικού Χρόνου και όλων εκείνων των χαρακτηριστικών που τα κάνει να ξεχωρίζουν από τα υπόλοιπα Λειτουργικά Συστήματα.

Σαν ένα παράδειγμα RTOS, θα παρουσιαστεί το λειτουργικό σύστημα VDK της Analog Devices το οποία παρέχεται με τους επεξεργαστές της τελευταίας. Αν και δε λειτουργεί με τον παραδοσιακό τρόπο που λειτουργούν τα υπόλοιπα λειτουργικά συστήματα (εγκατάσταση και εκκίνηση από το λειτουργικό σύστημα), παρέχεται σε συνδυασμό με

το περιβάλλον προγραμματισμού VDSP σαν μια απλή λύση στη δημιουργία προγραμμάτων που θα χρησιμοποιούν όλα εκείνα τα χαρακτηριστικά που καθιστούν ένα λειτουργικό σύστημα χρήσιμο (σηματοφόροι, διεργασίες με προτεραιότητες κλπ). Όπως αναφέρει το θέμα της εργασίας, θα παρουσιαστεί ο τρόπος που το VDK διαχειρίζεται τη μνήμη, τα νήματα, το χρονοπρογραμματισμό και περατώνει τη διαδιεργασιακή επικοινωνία.

Τέλος, θα αναφερθούν τα βήματα που ακολουθήθηκαν για τη δημιουργία ενός προγράμματος που τυπώνει το μήνυμα "Hello World from VDK". Σε αυτήν την περίπτωση, το λειτουργικό σύστημα τρέχει είτε σε περιβάλλον προσομείωσης (simulator), είτε απευθείας στον επεξεργαστή Blackfin 561 της εταιρείας Analog Devices.

3 Τα Λειτουργικά Συστήματα Πραγματικού Χρόνου

3.1 Η διαφορά τους από τα λειτουργικά συστήματα γενικού σκοπού

Οι χρονικές απαιτήσεις που έχουν πολλά υπολογιστικά συστήματα επιβάλλουν τη χρήση ιδιαίτερων λειτουργικών συστημάτων, τα Λειτουργικά Συστήματα Πραγματικού Χρόνου (RTOS). Αυτά ωστόσο δεν έχουν διαφορά όσον αφορά τον τύπο των υπηρεσιών που παρέχουν σε σχέση με τα γενικού τύπου λειτουργικά συστήματα (GPOS). Αντίθετα διαφέρουν ως προς τον τρόπο που τις υλοποιούν. Και αυτό γιατί σε όλες τις περιπτώσεις η χειρότερη δυνατή περίπτωση θα πρέπει να έχει καλά κατανοηθεί και περιγραφεί. Στα RTOS απαιτείται απόλυτη προβλεψιμότητα ή αλλιώς ντετερμινισμός.

Στα RTOS απαιτείται απόλυτη προβλεψιμότητα ή αλλιώς ντετερμινισμός

Ο ντετερμινισμός σαν όρος περιγράφει τη θεωρία ότι κάθε πράξη έχει την αιτία της. Δε θα σταθούμε στις φιλοσοφικές και επιστημονικές προεκτάσεις του ζητήματος αλλά θα περιγράψουμε πώς χρησιμοποιείται, καθώς αποτελεί την κύρια διαφορά μεταξύ των RTOS και των GPOS.

Στα GPOS οι υπηρεσίες μπορεί να εισάγουν τυχαίες καθυστερήσεις στις εφαρμογές, το οποίο σημαίνει αργή αποκριτικότητα σε απρόσμενους χρόνους. Οι τυχαίες αυτές καθυστερήσεις είναι απαγορευτικές στα RTOS. Οι υπηρεσίες θα πρέπει να χρησιμοποιούν γνωστά και αναμενόμενα ποσά χρόνου. Για να γίνει αυτό τα ποσά χρόνου θα πρέπει να μπορούν να περιγραφούν με αυστηρά αλγεβρικούς τύπους. Πολλά συστήματα προχωρούν ακόμη παραπέρα με το να θέτουν τους χρόνους αυτούς σταθερούς, κάνοντας το χρονισμό ανεξάρτητο του φορτίου (load-independent timing). Ο αλγεβρικός τύπος είναι σε αυτήν την περίπτωση είναι

$$T_{message} = \text{const.}$$

Έτσι λοιπόν φανερώνεται ότι πραγματικός χρόνος δε σημαίνει όσο το δυνατόν ταχύτερη εξυπηρέτηση ούτε φυσικά χρονισμοί στην κλίμακα των πραγματικών αριθμών, αλλά ικανοποίηση συγκεκριμένων χρονικών προθεσμιών.

Παρακάτω θα αναλυθούν μερικά παραδείγματα του πώς εισάγεται ο ντετερμινισμός σε τρεις βασικές υπηρεσίες των RTOS, στη διαχείριση εργασιών, στη διαδιεργασιακή επικοινωνία και συγχρονισμό και τη δυναμική διαχείριση μνήμης, ώστε να καλυφθούν οι χρονικές προθεσμίες.

3.2 Διαχείριση Εργασιών

Ερχόμαστε τώρα στη διαχείριση εργασιών (task management). Όπως είδαμε, σε κάθε μία από τις βασικές υπηρεσίες θα πρέπει να εφαρμόζεται η ιδέα του ντετερμινισμού, ώστε να παρέχεται προβλεψιμότητα. Πράγματι, αυτή η προβλεψιμότητα υπάρχει και στα δύο σκέλη της διαχείρισης εργασιών, το χρονοπρογραμματισμό και την εναλλαγή εργασιών, τα οποία αποτελούν συμπληρωματικές λειτουργίες.

3.2.1 Ο χρονοπρογραμματισμός

Πριν περιγραφεί ο χρονοπρογραμματισμός, θα πρέπει να αναφερθεί ότι αυτός δεν είναι πάντα δυνατός, δηλαδή δεν είναι κάθε σύστημα πραγματικού χρόνου χρονοπρογραμματίσιμο. Και αυτό εξηγείται ως εξής. Τα συστήματα πραγματικού χρόνου δέχονται ένα σύνολο απο εξωτερικά σήματα, περιοδικά και απεριοδικά. Δυνατότητα χρονοπρογραμματισμού προϋποθέτει δυνατότητα του συστήματος να επεξεργαστεί όλα τα σήματα. Αν έχουμε για παράδειγμα m περιοδικά συμβάντα, το καθένα από τα οποία έχει περίοδο P_i και απαιτεί χρόνο C_i για την ολοκλήρωσή του, το σύστημα είναι χρονοπρογραμματίσιμο όταν

$$\sum_{i=1}^{m}\left(\frac{C_i}{P_i}\right)\leq 1.$$

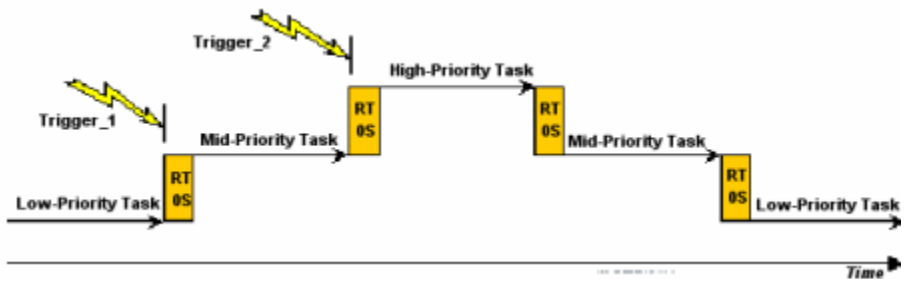
Αφού εξασφαλίζεται η δυνατότητα χρονοπρογραμματισμού, επιλέγεται μια μορφή του. Όπως αναφέρθηκε υπάρχουν αρκετοί τρόποι χρονοπρογραμματισμού. Αυτός που συνηθίζεται να πραγματοποιείται στα RTOS, είναι προεκτοπιστικός και βασισμένος στην προτεραιότητα (priority-based preemptive time scheduling). Ουσιαστικά είναι η συνένωση δύο επιμέρους αλγορίθμων, του προεκτοπιστικού και του βασισμένου στην προτεραιότητα. Θα αναλύσουμε τι σημαίνουν οι δύο αυτές έννοιες.

Το βασισμένο στην προτεραιότητα σκέλος του χρονοπρογραμματιστή

Πρώτα πρώτα, ο χρονοπρογραμματισμός είναι βασισμένος σε σύστημα προτεραιοτήτων. Σε ένα σύστημα πραγματικού χρόνου, το οποίο μπορεί ενώ πραγματοποιεί συγκεκριμένες λειτουργίες, να δεχθεί εξωτερικά ερεθίσματα τα οποία να επιβάλλουν άμεση δέσμευση της CPU από μια επείγουσα λειτουργία, ο καθορισμός προτεραιοτήτων στις εργασίες που επιτελούνται φαίνεται κάτι το εύλογο, στα πλαίσια των χρονικών περιορισμών και απαιτήσεων που αναφέρθηκαν στις αρχικές

παραγράφους. Και πραγματικά, στον αυτόματο πιλότο ενός αεροσκάφους, ή αναμονή ενός εξωτερικού σήματος επικίνδυνης προσέγγισης ενός άλλου αεροσκάφους να τελειώσει λόγου χάρη τη λειτουργία του ένας garbage collector θα είναι καταστροφική, αν όχι ειρωνική. Η προτεραιότητα δεν αναφέρεται μόνο στα εξωτερικά ερεθίσματα. Σε ένα στρατιωτικό υπολογιστικό σύστημα, οι εργασίες που εκτελούνται από στρατηγούς θα έχουν μεγαλύτερη προτεραιότητα από αυτές που εκτελούνται από συνταγματάρχες, αυτές θα έχουν μεγαλύτερη προτεραιότητα από τις αντίστοιχες των λοχαγών, ενώ όλες θα έχουν μεγαλύτερη προτεραιότητα από αυτές ενός απλού στρατιώτη.

Η βασική ιδέα που υλοποιείται λοιπόν, είναι ότι η εργασία με την υψηλότερη προτεραιότητα που πρέπει να τρέχει, θα είναι και αυτή που θα τρέχει. Έτσι αν μια εργασία προτεραιότητας 1 τρέχει και πριν αυτή ολοκληρωθεί, ζητήσει τη χορήγηση της CPU μια εργασία προτεραιότητας 5, η πρώτη εργασία θα παρέχει ευγενικά τη σειρά της. Αν ενώ τρέχει και η δεύτερη εργασία, ζητήσει τη CPU μια τρίτη προτεραιότητας 10, η δεύτερη θα παραχωρήσει επίσης τη θέση της. Το γενικό αυτό μοτίβο φαίνεται στην Εικόνα 1 .



Εικόνα 1: Εναλλαγή των διεργασιών στον χρονοπρογραμματισμό με βάση την προτεραιότητα

Αυτό που πρέπει να γίνει πρώτα είναι να αντιστοιχηθούν με κάποιο τρόπο προτεραιότητες στις εργασίες. Αυτό μπορεί να γίνει στατικά ή δυναμικά. Το προηγούμενο παράδειγμα στο στρατό αποτελεί στατική εκχώρηση προτεραιοτήτων. Η δυναμική εκχώρηση από την άλλη, είναι ένα μέτρο για την επίλυση ενός προβλήματος που μπορεί να προκύψει με τη στατική εκχώρηση. Τι γίνεται αν ένα σύνολο από εργασίες υψηλότερης προτεραιότητας, καταλαμβάνουν διαδοχικά τη CPU; Οι εργασίες που βρίσκονται στο κατώτερο σημείο του συστήματος προτεραιοτήτων, αν δεν υπάρξει κάποια αναπροσαρμογή, θα αργήσουν υπερβολικά να εκτελεστούν, αν εκτελεστούν καθόλου. Αυτό ονομάζεται «λιμοκτονία μέχρι θανάτου» (starvation). Πέρα μάλιστα από το μειονέκτημα ότι η εργασία δεν εκτελείται, μέρος της μνήμης που αντιστοιχεί σε αυτήν καταλαμβάνεται αναίτια.

Η λύση είναι κάποιος αλγόριθμος που ελέγχει εάν μια εργασία αναμένει τη χρήση της CPU για μεγάλο χρονικό διάστημα λόγω της χαμηλής της προτεραιότητας. Ένα παράδειγμα είναι το να ανατίθεται σαν νέα προτεραίοτητα μιας εργασίας ο αριθμός 1/f, όπου f το κλάσμα του χρόνου που χρησιμοποιήθηκε από τη εργασία πριν αυτή μπλοκαριστεί, προς το χρόνο που της αναλογούσε. Αν δηλαδή σε μια εργασία

αναλογούσε ένα κβάντο χρόνου 50 msec και αυτή μπλοκαρίστηκε αφού χρησιμοποίησε μόνο 2 msec, η νέα της προτεραιότητα θα είναι 25. Φυσικά θα πρέπει να υπάρχει ένα σαφές ορισμένο σύστημα προτεραιοτήτων όπου ο αριθμός 25 θα σημαίνει κάτι συγκεκριμένο.

Το βασισμένο στον προεκτοπισμό σκέλος του χρονοπρογραμματιστή
Μιλήσαμε προηγουμένως για κβάντο χρόνου και για δυνατότητα μπλοκαρίσματος μιας εργασίας πριν αυτή ολοκληρώσει τη λειτουργία τους. Αυτά δε συμβαίνουν αυτονόητα σε έναν υπολογιστή. Επίσης τίθεται το ερώτημα τι γίνεται σε περίπτωση που αντιμάχονται εργασίες της ίδιας προτεραιότητας για τη χρήση της CPU. Ο προεκτοπισμός είναι αυτός που δίνει τη λύση.

Πέρα από το εξωτερικό ερέθισμα μιας υψηλής προτεραιότητας διακοπής, ο αλγόριθμος επιλογής του χρονοπρογραμματιστή μπορεί να εφαρμόζεται σε διάφορες περιπτώσεις. Μία από αυτές είναι ο συγχρονισμός του επεξεργαστή με τα σήματα διακοπών, ώστε αυτά να συμβαίνουν ανά καθορισμένα χρονικά διαστήματα. Ο αλγόριθμος αυτός ονομάζεται προεκτοπιστικός και περιλαμβάνει την ικανότητα αυθαίρετης διακοπής μιας εργασίας πριν αυτή ολοκληρωθεί. Σε αυτήν την ιδιότητα βασίζεται και η ικανότητα των εργασιών να παραδίδουν αμέσως τη σκυτάλη σε μια άλλη διεργασία μεγαλύτερης προτεραιότητας.

Αυτό που δηλαδή πραγματοποιείται, είναι η πολύπλεξη των εργασιών στο πεδίο του χρόνου. Δηλαδή καθορίζεται ένα μικρό ποσοστό χρόνου (κβάντο χρόνου), που ονομάζεται χρονοθυρίδα και που αποτελεί το χρόνο στον οποίο επιτρέπεται να εκτελεστεί μια εργασία προτού αυτή διακοπεί από κάποια άλλη. Οι εργασίες εναλλάσονται μεταξύ τους και ολοκληρώνουν ένα μέρος τους κάθε φορά. Ο αλγόριθμος αυτός ονομάζεται αλγόριθμος εκ περιτροπής (round robin). Με αυτόν τον τρόπο επιλύονται οι διαμάχες ανάμεσα στις εργασίες ίδιας προτεραιότητας.

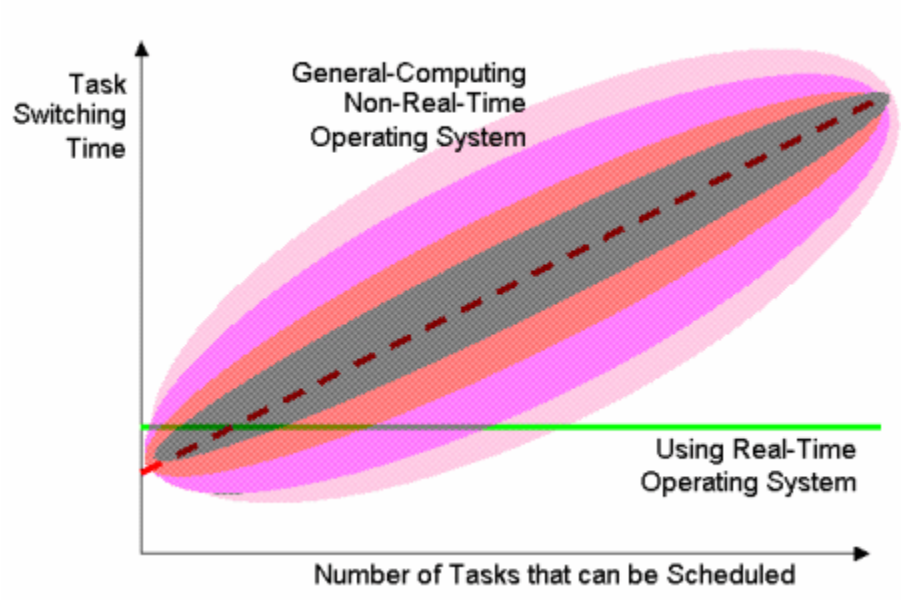
3.2.2 Η εναλλαγή των εργασιών

Ο χρονοπρογραμματιστής αποφασίζει ποια εργασία θα αντικαταστήσει αυτή που μπλοκάρεται. Η διαδικασία της εναλλαγής όμως, που ονομάζεται εναλλαγή εργασιών (task switching) ή εναλλαγή περιθωρίου (context switching), γίνεται από έναν άλλο αλγόριθμο, των διεκπεραιωτή (dispatcher). Η δουλειά του είναι να αποθηκεύει το περιβάλλον της παλιάς εργασίας, να φορτώνει το περιβάλλον της καινούριας εργασίας και να επιτρέπει στην τελευταία να εκτελεστεί. Με τον όρο περιβάλλον εννοούμε το σύνολο των δεδομένων, απαριθμητή προγράμματος, δεδομένα καταχωρητών, δείκτη στοίβας κλπ που αποθηκεύονται σε μια δυναμική λίστα του λειτουργικού συστήματος, τον πίνακα διεργασιών (process table) ή αλλιώς μπλοκ ελέγχου εργασιών (Task Control Block).

Στα λειτουργικά συστήματα γενικού σκοπού ο χρονοπρογραμματιστής ψάχνει στο TCB για να βρει την επόμενη εργασία. Όπως είναι εύλογο, όσο μεγαλύτερος είναι αυτός ο πίνακας, δηλαδή όσο περισσότερες εργασίες βρίσκονται σε αναμονή, τόσο

περισσότερο χρόνο θα κάνει ο χρονοπρογραμματιστής να βρει τη σωστή και να γίνει η εναλλαγή. Αυτό μπορεί να είναι σχετικά ενοχλητικό. Αν για παράδειγμα ο χρόνος εναλλαγής είναι 1 msec ενώ το κβάντο λειτουργίας των διεργασιών είναι 4 msec, αυτό σημαίνει ότι 25% του χρόνου της η CPU το χρησιμοποιεί για διαχειριστικές λειτουργίες. Γι' αυτό το λόγο καθορίζεται συνήθως ένα σχετικά μεγάλο κβάντο χρόνου, όχι όμως αρκετά μεγάλο για να περιμένουν πολύ ώρα σε αναμονή οι υπόλοιπες εργασίες.

Στα συστήματα πραγματικού χρόνου ωστόσο η εξάρτηση του χρόνου εναλλαγής από το πλήθος των εργασιών που βρίσκονται σε αναμονή, δε συνάδει καθόλου με το ντετερμινισμό που έχουν ως απαίτηση. Εδώ αυτό που χρειάζεται είναι να γνωρίζουμε εκ των προτέρων πόση θα είναι η καθυστέρηση αυτή. Έτσι στα λειτουργικά συστήματα πραγματικού χρόνου χρησιμοποιούνται αυξητικά ενημερωμένοι πίνακες στο TCB, που επιτρέπουν το χρονοπρογραμματιστή να αναγνωρίσει την επόμενη εργασία ταχύτατα, σε σταθερό χρόνο. Στην παρακάτω εικόνα σχηματοποιούνται οι δύο προηγούμενες φιλοσοφίες.



Εικόνα 2: Διαφορά των GPOS και RTOS στην εναλλαγή διεργασιών

Στα λειτουργικά γενικού σκοπού, η κόκκινη διακεκομμένη γραμμή περιγράφει τη γραμμική σχέση μεταξύ του πλήθους των εργασιών και του χρόνου εναλλαγής. Για την ακρίβεια ο χρόνος εναλλαγής δεν είναι απαραίτητα πάνω στη γραμμή, αλλά πάνω στις σκιασμένες περιοχές, οι οποίες δείχνουν πιθανότητα απομάκρυνσης από τη γραμμική περιοχή. Στα RTOS ο χρόνος εναλλαγής είναι σταθερός. Αν και για μικρό πλήθος εργασιών τα RTOS φαίνεται να υστερούν για μεγαλύτερο πλήθος το πλεονέκτημά τους είναι αδιαμφισβήτητο.

Έτσι λοιπόν αν κατά τη διάρκεια εκτέλεσης μιας εργασίας, το λειτουργικό σύστημα

δεχθεί ένα εξωτερικό ή εσωτερικό ερέθισμα που δηλώνει την ύπαρξη μιας νέας εργασίας ή απλά τελειώσει το κβάντο χρόνου που αναλογεί στην εκτελούμενη εργασία, πραγματοποιούνται τα παρακάτω βήματα:

- Ελέγχεται αν η εκτελούμενη εργασία δικαιούται να συνεχίσει
- Αποφασίζεται ποια εργασία θα τρέξει αμέσως μετά
- Αποθηκεύεται το περιβάλλον της εργασίας ώστε αυτή να συνεχιστεί αργότερα
- Η εργασία σταματά
- Φορτώνεται η καινούρια εργασία
- Η καινούρια εργασία τρέχει

Να σημειώσουμε ότι όταν προεκτοπίζεται μια εργασία, αυτή δε σταματά απευθείας, αλλά της επιτρέπεται να ολοκληρώσει τη γραμμή κώδικα που εκτελούσε. Όχι όμως κώδικας υψηλού επιπέδου όπως για παράδειγμα μια εντολή printf της C, αλλά κώδικας μηχανής όπως load r0, r1.

Μέχρι τώρα μιλήσαμε για τις εργασίες (tasks) σαν κάτι το ενιαίο. Για την ακρίβεια όμως, αυτές διαιρούνται σε διεργασίες (processes), ρουτίνες εξυπηρέτησης διακοπών (Interrupt Service Routines) και ρουτίνες συστήματος (system routines). Πολλά ενσωματωμένα λειτουργικά συστήματα έχουν μάλιστα τα νήματα (threads) ως εκτελεστικές μονάδες. Οι διεργασίες αναφέρονται στις ρουτίνες που περιέχουν τα προγράμματα εφαρμογών που τρέχουν πάνω στο RTOS. Οι ISRs προκύπτουν από τις συσκευές Ε/Ε και είναι οι ρουτίνες που χειρίζονται τα δεδομένα που κινούνται προς και από αυτές. Οι ρουτίνες συστήματος είναι σε γενικές γραμμές οι διαχειριστικές ρουτίνες που εκτελεί το λειτουργικό σύστημα σε κάθε κλήση συστήματος. Τα νήματα είναι μικρότερου βεληνεκούς εκτελεστικές μονάδες που χρησιμοποιούν το χώρο διευθύνσεων (address space) μιας διεργασίας για να εκτελέσουν συγκεκριμένες λειτουργίες. Η διάκριση αυτή προέρχεται από τα λειτουργικά συστήματα γενικού σκοπού

Στην περίπτωση των διακοπών και στα πλαίσια της προβλεψιμότητας, θα πρέπει να είναι γνωστός ο μέγιστος χρόνος από την εφαρμογή του σήματος διακοπής, μέχρι να αρχίσει να εκτελείται η ρουτίνα εξυπηρέτησης διακοπής. Αυτός ο χρόνος ονομάζεται καθυστέρηση διακοπής (Interrupt Latency) και αποτελεί τη συνισταμένη τριών επιμέρους καθυστερήσεων.

Όταν συμβαίνει μια διακοπή, ο επεξεργαστής τερματίζει την εργασία που τρέχει αφού αυτή ολοκληρώσει την εντολή κώδικα μηχανής που άρχισε. Αυτό συνήθως διαρκεί λιγότερο από ένα παλμό ρολογιού, αλλά σε πιο σύνθετες εντολές μπορεί να διαρκεί παραπάνω. Επίσης αναγνωρίζεται η διακοπή. Αυτό γίνεται από το υλικό και η καθυστέρηση που υπεισέρχεται είναι ελάχιστη. Αν οι διακοπές είναι ενεργοποιημένες, τότε η ISR που σχετίζεται με την εφαρμογή αυτή, καταλαμβάνει τον επεξεργαστή. Εδώ συνισφέρει η τρίτη συνιστώσα της καθυστέρησης, που είναι και πιο δύσκολο να

μετρηθεί. Και αυτό γιατί όταν κάποια εργασία μπει στο κρίσιμο τμήμα της, δηλαδή στο τμήμα όπου γίνεται κοινή προσπέλαση μνήμης, πραγματοποιείται μπλοκάρισμα των διακοπών και ο χρόνος που αυτό διαρκεί εξαρτάται από κάθε εργασία χωριστά. Θα πρέπει δηλαδή να είναι γνωστός και ο μέγιστος χρόνος που κάθε εργασία εκτελεί το κρίσιμο τμήμα της.

3.3 Διαδιεργασιακή Επικοινωνία και Συγχρονισμός

Στο κεφάλαιο για τις υπηρεσίες του πυρήνα αναφέρθηκε ότι οι εργασίες δε δρουν σαν ανεξάρτητες οντότητες αλλά απαιτείται η μεταξύ τους επικοινωνία και συγχρονισμός. Τα λειτουργικά συστήματα πραγματικού χρόνου υλοποιούν τις δύο αυτές λειτουργίες με παρόμοιο τρόπο όπως τα λειτουργικά συστήματα γενικού σκοπού.

Για το συγχρονισμό αρνητικού τύπου, χρησιμοποιούνται σημαφόροι και mutex, ενώ για το συγχρονισμό θετικού τύπου χρησιμοποιούνται σημαίες, σήματα και μηνύματα. Τα τελευταία παίζουν ρόλο και στην επικοινωνία των εργασιών.

Πάλι όμως υπάρχει μια αδυναμία στη μεταβίβαση των μηνυμάτων, που επισύρει τη τροποποίησή της από τα λειτουργικά συστήματα πραγματικού χρόνου ώστε να συμβαδίζει πλήρως με την αρχή του ντετερμινισμού. Εφόσον το μήνυμα είναι ουσιαστικά ένας πίνακας στον οποίο γράφει η εργασία-αποστολέας και από τον οποίο διαβάζει η εργασία-παραλήπτης, η διαδικασία ανάγνωσης του μηνύματος αποτελεί μια σειρά από δύο αντιγραφές. Ο χρόνος όμως των αντιγραφών αυτών αυξάνει όσο αυξάνει το μέγεθος του μηνύματος.

Εικόνα 3: Η επικοινωνία μεταξύ των εργασιών γίνεται με μηνύματα

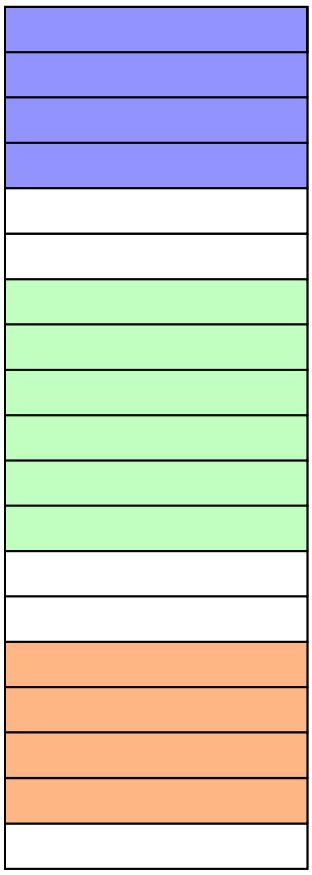
Η λύση είναι να μεταδίδεται στην εφαρμογή παραλήπτη ένας δείκτης στον πίνακα αντί να αντιγράφονται τα δεδομένα από αυτόν. Για την αποφυγή μάλιστα συγκρούσεων στην πρόσβαση του πίνακα, αφαιρείται από την εφαρμογή-αποστολέα το αντίγραφο του δείκτη στον πίνακα. Η λύση αυτή σταθεροποιεί το χρονισμό, αλλά βελτιώνει και την απόδοση.

Τέλος να πούμε πως η συνδυασμένη χρήση σηματοφόρων και προτεραιοτήτων εγείρει ένα σημαντικό για τα RTOS πρόβλημα. Έστω ότι υπάρχουν για παράδειγμα δύο εργασίες, η Α και η Β, με τη Β να έχει υψηλότερη προτεραιότητα από την Α. Αν η Α όμως έχει χρησιμοποιήσει σηματοφόρο για να μπει στο κρίσιμο τμήμα της και η Β την προεκτοπίσει, η Β θα μπλοκαριστεί καθώς δεν θα μπορεί να κάνει προσπέλαση του κοινού πόρου. Έτσι η Α θα ξαναναλάβει τα ηνία του επεξεργαστή μέχρι να βγει από το κρίσιμο τμήμα της. Η Α δηλαδή αποκτά τυπικά μεγαλύτερη προτεραιότητα από τη Β. Το πρόβλημα αυτό ονομάζεται πρόβλημα *αντιστροφής προτεραιοτήτων* (priority inversion).

Η δυναμική εκχώρηση προτεραιοτήτων που αναλύθηκε σε προηγούμενο τμήμα είναι η λύση. Στην εργασία Α εκχωρείται προσωρινά ίση προτεραιότητα με τη Β μέχρι να εκτελέσει το κρίσιμο τμήμα της. Η Β θα εκτελεστεί τότε με τη μικρότερη δυνατή καθυστέρηση, η οποία θα αντιστοιχεί στο χρόνο που θα κάνει η Α για να βγει από το κρίσιμο τμήμα της. Αυτός ο αλγόριθμος ονομάζεται *κληρονομικότητα προτεραιοτήτων* (priority inheritance).

3.4 Δυναμική Διαχείριση Μνήμης

Στις υπηρεσίες διαχείρισης δυναμικής μνήμης δεν θα μπορούσε να απουσιάζει η προβλεψιμότητα. Ας δούμε ένα παράδειγμα όπου οι βασικές υπηρεσίες των λειτουργικών συστημάτων γενικού σκοπού την παραβιάζουν. Έστω μια απλοποιημένη εκδοχή της σωρού η οποία περιέχει 19 θυρίδες (slots). Έστω τώρα ότι η εργασία Α δεσμεύει 4 θυρίδες προτού προεκτοπιστεί από την εργασία Β η οποία με τη σειρά της δεσμεύει 6 θυρίδες που δεν είναι όμως σειριακά επόμενες ως προς αυτές που δέσμευσε η πρώτη. Ομοίως προεκτοπίζεται και η Β από την εργασία Γ που δεσμεύει 5 θυρίδες μνήμης. Η αλληλουχία των γεγονότων αυτών φαίνεται στην Εικόνα 4.

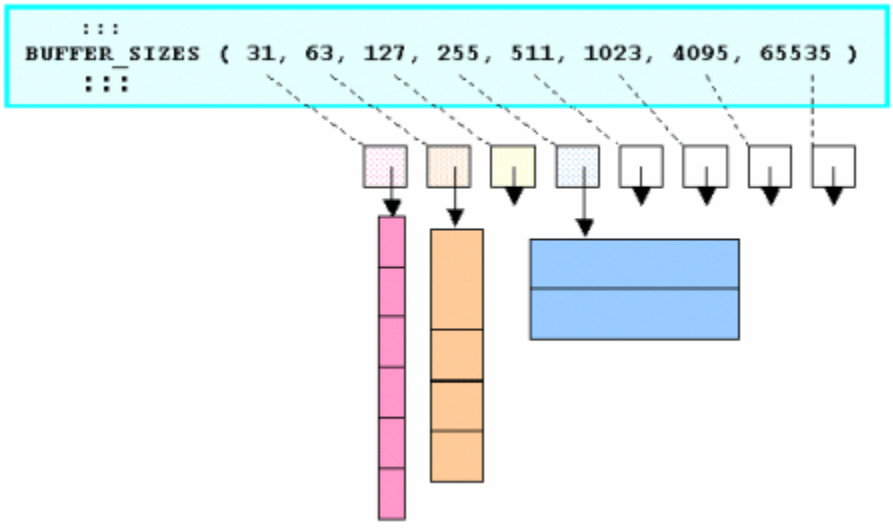


Εικόνα 4: Με μπλε οι 6 χρονοθυρίδες που δεσμεύτηκαν από την εργασία Α, με πράσινο και πορτοκαλί αυτές των εργασιών Β και Γ αντίστοιχα. Οι 4 χρονοθυρίδες που απέμειναν δεν μπορούν να εκχωρηθούν σε μια εργασία που θα τις ζητήσει

Αν τώρα μια εργασία Δ λάβει τη σκυτάλη και επιχειρήσει να δεσμεύσει τέσσερεις θυρίδες από τη μνήμη σωρού, αυτές δεν θα αποδοθούν. Παρόλο που υπάρχουν δεν είναι σειριακές όπως απαιτείται. Η δημιουργία πολλών μικρών περιοχών στη μνήμη, που μπορούν δύσκολα να εκχωρηθούν, ονομάζεται *Κατάτμηση Εξωτερικής Μνήμης* (External Memory Fragmentation). Το πρόβλημα αυτό είναι παρόμοιο με την κατατμηση που υφίστανται δίσκοι στους οποίους το σύστημα αρχείων υλοποιεί τα αρχεία με σειριακή κατανομή. Σε ένα σύστημα όπου οι θυρίδες μνήμης δεσμεύονται και αποδεσμεύονται δυναμικά σε διάφορα μεγέθη το πρόβλημα αυτό μπορεί να είναι σημαντικό. Σε συστήματα όπου δεν υπάρχουν απαιτήσεις πραγματικού χρόνου, αλγόριθμοι όπως ο συλλέκτης σκουπιδιών (garbage collector) επιχειρούν να το αντιμετωπίσουν. Εισάγουν όμως τυχαίες καθυστερήσεις.

Η δημιουργία πολλών μικρών περιοχών στη μνήμη, που μπορούν δύσκολα να εκχωρηθούν, ονομάζεται *Κατάτμηση Εξωτερικής Μνήμης* (External Memory Fragmentation).

Το σενάριο αυτό δεν είναι αποδεκτό στα συστήματα πραγματικού χρόνου. Ένας τρόπος επίλυσης αυτού του προβλήματος είναι η χρήση «πισίνων» μνήμης (memory pools). Αυτές είναι περιοχές που μπορούν να σπάσουν σε επιμέρους τμήματα (buffers ή blocks) συγκεκριμένου πλήθους (από 4 μέχρι 8) και συγκεκριμένου μεγέθους (δυνάμεις του 2). Όταν ένα τμήμα συγκεκριμένου μεγέθους εκχωρείται σε μια εργασία, αφού επιστραφεί κρατάει το μέγεθός του και δεν του επιτρέπεται να σπάσει σε μικρότερα. Έτσι σε επόμενη ζήτηση μνήμης ίδιου μεγέθους αυτό μπορεί να εκχωρηθεί γρήγορα, σε ντετερμινιστικό χρόνο.



***Εικόνα 5**: Μια λίστα από διαθέσιμους buffers για μια "πισίνα¨ μνήμης*

4 Εισαγωγή στο VisualDSP++ Kernel (VDK)

Μαζί με τους επεξεργαστές της, η Analog Devices παρέχει το προγραμματιστικό περιβάλλον VisualDSP++ το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών που εκτελούνται σε αυτούς. Εναλλακτικά οποιοσδήποτε μπορεί να κατεβάσει το περιβάλλον αυτό από την ιστοσελίδα της εταιρίας, www.analog.com, με περιορισμό ωστόσο τη χρήση μόνο σε περιβάλλον εξομοίωσης.

Πέρα από την εγγραφή προγραμμάτων που μπορούν απλά να εκτελεστούν στους επεξεργαστές, το VDSP++ μπορεί να πλαισιώσει τα προγράμματα αυτά με τον πυρήνα VDK. Το VDK όπως αναφέρθηκε και στην εισαγωγή της εργασίας αυτής, δεν είναι ολοκληρωμένο λειτουργικό σύστημα, με την έννοια ότι δεν μπορεί να εγκατασταθεί σε κάποιον επεξεργαστή και μετά αυτός να εκκινεί από εκεί, κάτι που είναι συνήθες με τα Windows και το Linux. Αντίθετα, με τη βοήθεια του VDSP, καθορίζονται τα χαρακτηριστικά του (σηματοφόροι, προτεραιότητες, τύποι νημάτων, νήμα εκκίνησης κλπ) ενώ παράλληλα γράφονται και οι εφαρμογές που θα τρέξουν σε αυτό, ως σύνολο από στιγμιότυπα των τύπων νημάτων. Το αποτέλεσμα φορτώνεται στην εξωτερική μνήμη RAM του επεξεργαστή και εκτελείται.

4.1 Λόγοι χρήσης του VDK

Εάν και το VDK δεν φτάνει την ευελιξία που προσφέρουν άλλες περιπτώσεις RTOS και γενικότερα ενσωματωμένων λειτουργικών συστημάτων, όπως το uClinux και το velOSity που απευθύνονται επίσης στους επεξεργαστές της Analog Devices, υπάρχουν διάφοροι λόγοι για να το προτιμήσει κανείς στο σχεδιασμό του. Για αρχή, η στενή συνεργασία του VDK με το περιβάλλον προγραμματισμού VDSP++ σημαίνει ότι ο επίδοξος προγραμματιστής μπορεί ταχύτητα να αναπτύξει εφαρμογές που χρησιμοποιούν της υπηρεσίες ενός πυρήνα.Αυτό χωρίς να ανησυχεί για την αγορά και εγκατάσταση ενός εμπορικού λειτουργικού συστήματος, ούτε να μπαίνει στην επίπονη διαδικασία της διασταυρούμενης μεταγλώττισης (cross-compiling), που συνοδεύει λύσεις ανοιχτού κώδικα όπως το uClinux.

Το VDK είναι σχεδιασμένο ώστε να παρέχει λύσεις λαμβάνοντας υπόψιν την υποκείμενη αρχιτεκτονική των επεξεργαστών της Analog

Η εξάρτηση του VDK από το VDSP++ συμβάλλει όχι μόνο στην εύκολη αποσφαλμάτωση (debugging) των εφαρμογών, αλλά και στη στενή παρακολούθηση της ροής ελέγχου, της κατάστασης του συστήματας και διαφόρων run-time δεδομένων κατά τη διάρκεια εκτέλεσης των εφαρμογών. Αυτό γιατί, σε αντίθεση με άλλα προγραμματιστικά περιβάλλοντα, το VDSP++ είναι σχεδιασμένο ώστε να παρέχει λύσεις λαμβάνοντας υπόψιν την υποκείμενη αρχιτεκτονική των επεξεργαστών της Analog. Η αποσφαλμάτωση γίνεται με την υποστήριξη τριών διαφορετικών βιβλιοθηκών για το

VDK. Η πρώτη υποστηρίζει πλήρη ενοργάνωση (instrumentation). Σε αυτήν, στατιστικά δεδομένα όπως δέσμευση και αποδέσμευση σημάτων, εναλλαγή διεργασιών (context switching) και αλλαγές στην κατάσταση ενός νήματος σε ένα buffer καταγραφής (history buffer). Μετά από κάθε παύση της εκτέλεσης ο αποσφαλματωτής μπορεί να προσπελάσει τον buffer αυτόν και να παρουσιάσει μια γραφική παρασταση των δεδομένων. Στη δεύτερη βιβλιοθήκη πραγματοποιείται καταγραφή σφαλμάτων (error-checking) ενώ στην τρίτη δεν υποστηρίζεται ούτε instrumentation ούτε error-checking.

Από τα πλεονεκτήματα του VDK δεν θα μπορούσε να παραλειφθεί η επαναχρησιμοποίηση κώδικα που προκύπτει με τη χρήση προτύπων νημάτων (thread templates). Το πρότυπο αποτελεί υλοποίηση ενός αλγορίθμου και καθορίζει πλήρως τη συμπεριφορά και τα δεδομένα που σχετίζονται με όλα τα νήματα αυτού του τύπου (στιγμιότυπα). Καθένα από αυτά, έχει τη δικιά του κατάσταση, το δικό του χώρο στη μνήμη, αγνοώντας πλήρως όλα τα υπόλοιπα στιγμιότυπα του ίδιου προτύπου, εκτός και αν υπάρχει αλληλοεξάρτηση μεταξύ τους. Το ίδιο συμβαίνει στον αντικειμενοστραφή προγραμματισμό όταν για παράδειγμα πολλά στιγμιότυπα μιας κλάσης μπορούν να συνυπάρχουν ανεξάρτητα το ένα από το άλλο. Η τεχνική αυτή, που οφείλεται στη συνεργασία με το VDSP, επιτρέπει την οργανωμένη συγγραφή κώδικα μέσα από τη διαίρεσή του σε πολλές δομικές μονάδες

Τέλος, όπως όλα τα λειτουργικά συστήματα, παρέχει ένα υψηλότερο επίπεδο αφαίρεσης, ώστε τα προγράμματα να αγνοούν την αρχιτεκτονική του επεξεργαστή και να επικεντρώνονται μόνο στις λεπτομέρειες του πραγματικού αλγορίθμου. Πέρα από assembly, το VDK υποστηρίζει τη συγγραφή προγραμμάτων σε C και C++. Μάλιστα εγγυάται μεγάλη φορητότητα στα προγράμματα αυτά μέσω από τη χρήση ενός κοινού API για τους επεξεργαστές της Analog, όχι μόνο της σειράς Blackfin, αλλά και τους SHARC και Tiger.

4.2 Τα νήματα ως βασικές μονάδες εκτέλεσης

Η βασική μονάδα εκτέλεσης του VDK είναι το νήμα (thread), ωστόσο θα χρησιμοποιείται εναλασσόμενα και ο όρος διεργασία για να περιγράψει το ίδιο: Αλληλουχία κώδικα με συγκεκριμένο πίνακα στη μνήμη (run-time context), που αποτελεί ένα μονοπάτι (ροή) εκτέλεσης. Πρακτικά, βοηθούν το χωρισμό μιας εφαρμογής σε μικρότερες οντότητες, καθεμία από τις οποίες επιτελεί ένα μέρος της συνολικής δουλειάς.

Ο ορισμός των νημάτων δε γίνεται μεμονωμένα για το καθένα. Αντίθετα, πρέπει πρώτα να δημιουργηθεί ένας τύπος νήματος (Thread Type), του οποίου στιγμιότυπο θα είναι το επιθυμητό νήμα. Κάθε νήμα, ακόμη και αν αποτελούν στιγμιότυπα του ίδιου τύπου, έχει τη δικιά του στοίβα (stack), κατάσταση, προτεραιότητα, τοπικές μεταβλητές και ταυτότητα (ID). Η τελευταία μπορεί να ληφθεί μέσα από το ίδιο το νήμα με την κλήση συστήματος GetThreadID(). Τα δεδομένα αυτά τοποθετούνται σε ένα χώρο στη σωρό (heap) που δεσμεύεται κατά τη δημιουργία κάθε νήματος. Με λίγα λόγια, τα

στιγμιότυπα ενός τύπου δρουν ανεξάρτητα το ένα από το άλλο, εκτός και αν είναι επιθυμητή οποιαδήποτε αλληλοεξάρτηση, όποτε αυτή καθορίζεται κατάλληλα.

Το μέγεθος της στοίβας των νημάτων, καθορίζεται κατά τον ορισμό του τύπου από τον οποίο προέρχονται και είναι ευθύνη του προγραμματιστή να είναι επαρκώς μεγάλο ώστε να χωράει τα απαραίτητα δεδομένα, αν και υπερχείλιση της στοίβας δεν προκαλεί εξαίρεση (exception), δυσχεραίνοντας την αποσφαλμάτωση.

Κάθε νήμα έχει τη δικιά του προτεραιότητα, ο ρόλος της οποίας θα γίνει πιο ξεκάθαρος στην επόμενη ενότητα, κατά την περιγραφή του χρονοπρογραμματιστή. Ωστόσο θα πρέπει να ειπωθεί ότι πέρα από τον στατικό καθορισμό της προτεραιότητας, κατά τον καθορισμό των τύπων νημάτων, αυτή μπορεί να καθοριστεί με τη χρήση κλήσεων στο API, μέσα από το ίδιο το νήμα ή ακόμη μέσα από άλλο. Οι κλήση αυτή είναι η SetPriority(), η οποία παίρνει σαν ορίσματα την ταυτότητα του νήματος-στόχου και την επιθυμητή προτεραιότητα.

Αν και τα νήματα μπορούν σε C, C++ ή assembly, η επιλογή αυτή είναι διαφανής στο VDK. Το προγραμματιστικό περιβάλλον παράγει καλά ορισμένο κώδικα μηχανής για όλες αυτές τις περιπτώσεις. Τα C++ νήματα αποτελούν παράγωγα της βασικής κλάσης VDK::Thread και έχουν λίγο διαφορετικά ονόματα για τις συναρτήσεις και τις κλήσεις συστήματος. Τα νήματα σε C, μεταγλωττίζονται χωρίς τις επεκτάσεις του μεταγλωττιστή της C++.

Αν και τα νήματα μπορούν να γραφούν σε C, C++ ή assembly, η επιλογή αυτή είναι διαφανής στο VDK

Πολύ βασικό χαρακτηριστικό των τύπων νημάτων, είναι ότι απαιτούν τον καθορισμό και την υλοποίηση πέντε συναρτήσεων, που παίζουν πρωταρχικό ρόλο για τη λειτουργία κάθε νήματος. Αυτές είναι οι run, error, create, init και destroy. Κατά τον ορισμό ενός τύπου νήματος, δημιουργείται από το προγραμματιστικό περιβάλλον ένα ομώνυμο αρχείο πηγαίου κώδικα, στο οποίο οι προαναφερθείσες συναρτήσεις είναι υλοποιημένες ως μηδενικές συναρτήσεις. Είναι στην ευχέρεια του προγραμματιστή η απόδοση της λειτουργικότητας που απαιτείται για κάθε μία από αυτές, όπως αναλύεται παρακάτω.

Run

Η συνάρτηση Run έχει όνομα Run() για νήματα υλοποιημένα σε C++ και RunFunction() για τα αντίστοιχα της C/assembly. Είναι το σημείο αναφοράς για το νήμα, όπου εκτελείται ο κυρίως αλγόριθμος που του δίνει την επιθυμητή λειτουργικότητα, αντίστοιχα με τη συνάρτηση main() στη C. Υλοποιείται σαν ένας ατέρμονος βρόχος while(1) μέσα στον οποίο μπαίνει ο επιθυμητός κώδικας. Η έξοδος από το βρόχο γίνεται με τη γνωστή εντολή break, η οποία συνήθως ακολουθεί μια συνθήκη και σηματοδοτεί τον τερματισμό του νήματος, οπότε αυτό αναμένει την αποδέσμευση των πόρων του. Ο προγραμματιστής ωστόσο έχει την ευχέρεια να αφαιρέσει τον εκ

προεπιλογής βρόχο while για να εκτελεστεί ο κώδικας του νήματος μόνο μια φορά πρωτού τερματιστεί.

Error
Το VDK συμπεριλαμβάνει ένα μηχανισμό διαχείρισης σφαλμάτων, που επιτρέπει τον καθορισμό διαφορετικής συμπεριφοράς για κάθε νήμα, αν και υπάρχει μια προεπιλεγμένη (default) συνάρτηση σφαλμάτων που εγείρει τον πυρήνα σε «κατάσταση πανικού» (kernel panic), καλώντας την ομώνυμη συνάρτηση. Η οριζόμενη από το χρήστη συνάρτηση, έχει όνομα Error() για νήματα σε γλώσσα C++ και ErrorFunction() για νήματα σε C/assembly. Λαμβάνει χώρα όταν τυγχάνει κάποιο σφάλμα σε μια κλήση API, έτσι ώστε να καθορίζεται η δράση του συστήματος ακόμα και σε απρόσμενα ενδεχόμενα, κάτι πολύ επιθυμητό για συστήματα με απαιτήσεις πραγματικού χρόνου.

Create
Η συνάρτηση αυτή είναι αντίστοιχη του constructor στη C++ και επιτρέπει τη δυναμική δημιουργία νημάτων. Ενεργοποιείται όποτε πραγματοποιηθούν οι κλήσεις συστήματος CreateThread() και CreateThreadEx(), μέσα από κάποιο άλλο νήμα ή κατά την έναρξη του συστήματος όταν πρόκειται για νήματα εκκίνησης. Η συνάρτηση αυτή αν και ορίζεται μέσα σε κάθε νήμα, εκτελείται στον πίνακα (context) του καλούντος νήματος. Αναλαμβάνει να δημιουργήσει το νήμα και να βεβαιώσει το ότι όλες οι εκχωρήσεις που ζήτησε το νήμα έγιναν σωστά. Το καινούριο νήμα αποκτάει δικό του πίνακα στη μνήμη μόνον όταν τερματίσουν οι δύο προαναφερθείσες κλήσεις συστήματος και η συνάρτηση create δίνει τα ηνία στη συνάρτηση init.

Init
Η συνάρτηση αυτή υλοποιείται σαν ένας constructor για τα νήματα που είναι γραμμένα σε C++ και σαν συνάρτηση με όνομα InitFunction() για νήματα σε C/assembly. Σκοπός της είναι η αρχικοποίηση του νήματος, παρέχοντας ένα μέρος για την εκχώρηση πόρων συστήματος κατά τη διάρκεια της δυναμικής δημιουργίας του. Οι τοπικές μεταβλητές εκχωρούνται με την εντολή malloc() ή new. Εφόσον καλείται μέσα από τον πίνακα ενός άλλου νήματος, διαδεχόμενη τη συνάρτηση create, δεν μπορεί να ενσωματώσει κλήσεις συστήματος στον κώδικά της.

Destroy
Όταν ένα νήμα τελειώσει την εκτέλεσή του, είτε επειδή έφτασε στο τέλος του κώδικα της συνάρτησης run, είτε μέσα από την κλήση συστήματος DestroyThread(), θα πρέπει να αποδεσμεύσει όλη την ποσότητα μνήμης που του εκχωρήθηκε με τις εντολές malloc/new. Κατ’ αντιστοιχία λοιπόν με τον destructor της C++, χρησιμοποιούνται οι εντολές free ή delete για την αποδέσμευση των πόρων του νήματος.

Επίσης θα πρέπει να αναφερθούν δύο ιδιαίτερα νήματα, τα οποία αναλαμβάνουν καθοριστικές για το σύστημα δουλειές. Το πρώτο είναι το νήμα εκκίνησης (boot thread). Αυτό είναι το πρώτο νήμα που εκτελείται με την έναρξη του συστήματος και

από εκεί είτε δημιουργούνται νέα νήματα στα οποία περνά η εκτέλεση είτε αυτή περνά στα νήματα που έχουν προκαθοριστεί. Μπορεί ακόμη η εκτέλεση να περιορίζεται μόνο στο βρόχο του νήματος εκκίνησης, όταν πρόκειται για ένα απλό πρόγραμμα, που δεν υπάρχει ενδιαφέρον για την ύπαρξη πολλών νημάτων. Στην περίπτωση του παραδείγματος “Hello World”, ακολουθήθηκε αυτή η μέθοδος. Για να δημιουργηθεί ένα νήμα εκκίνησης, χρειάζεται να δημιουργηθεί πρώτα ένα νήμα συγκεκριμένου τύπου, για παράδειγμα BootThreadType και ακολούθως να καθοριστεί ότι το Boot Thread είναι τύπου BootThreadType. Το δεύτερο είναι το ανενεργό νήμα (idle thread), το οποίο είναι το νήμα που εκτελείται όταν κανένα άλλο δεν είναι διαθέσιμο. Το νήμα αυτό έχει την ελάχιστη προτεραιότητα ώστε να μην τύχει και προεκτοπίσει κανένα άλλο. Μεριμνά για την καταστροφή των νημάτων, τα οποία ήρθαν εις πέρας έπειτα από τη χρήση της κλήσης DestroyThread(), με κατάλληλο όρισμα ώστε να μην τερματιστούν άμεσα. Με λίγα λόγια είναι άμεσα υπεύθυνο για την αποδέσμευση των πόρων όταν αυτοί δε χρειάζονται πια.

Τέλος η υποστήριξη από το VDK συστημάτων με παραπάνω από έναν επεξεργαστές, έδωσε έδαφος σε μια ξεχωριστή κατηγορία νημάτων, τα νήματα δρομολόγησης (RThreads). Σκοπός τους είναι η δρομολόγηση των μηνυμάτων που ένα νήμα που εκτελείται σε έναν επεξεργαστή-κόμβο, στέλνει σε ένα νήμα που εκτελείται σε κάποιον άλλον επεξεργαστή-κόμβο. Τα νήματα δρομολόγησης έχουν δύο διαφορετικούς ρόλους, εισόδου (incoming) ή εξόδου (outgoing), οι οποίοι καθορίζονται κατά τη δημιουργία τους. Τα νήματα εξόδου αναλαμβάνουν την εγγραφή μηνυμάτων σε μια φυσικό κανάλι επικοινωνίας ανάμεσα στους επεξεργαστές, με τη χρήση του κατάλληλου οδηγού. Τα νήματα εισόδου αντίστοιχα διαβάζουν από το κανάλι αυτό. Η δρομολόγηση επιτυγχάνεται σαν ένας πίνακας με δύο πεδία. Την ταυτότητα του κόμβου, η οποία είναι σημείο αναφοράς (index) για την επιλογή του νήματος εξόδου, που αποτελεί το δεύτερο πεδίο. Στην καλύτερη περίπτωση κάθε επεξεργαστής είναι φυσικά συνδεδεμένος με όλους τους υπόλοιπους, έχοντας ισάριθμα νήματα εισόδου και εξόδου, ωστέ τα μηνύματα να μην πραγματοποιούν παραπάνω από ένα βήματα (hops) για να βρουν τον προορισμό τους. Στη χειρότερη περίπτωση κάθε επεξεργαστής έχει μόνο ένα νήμα εισόδου και εξόδου, ώστε το σύνολό τους να μπορεί να θεωρηθεί σαν ένας δακτύλιος. Στην πράξη, με την αύξηση του πλήθους των επεξεργαστών αυξάνεται εκθετικά το πλήθος των απαιτούμενων νημάτων. Αυτός είναι ο λόγος που φροντίζεται να διατηρείται μια ενδιάμεση λύση: Κάθε επεξεργαστής έχει νήματα εξόδου για ένα υποσύνολο των υπόλοιπων επεξεργαστών, κερδίζοντας έτσι σε πόρους, αλλά επιβραδύνοντας τη δρομολόγηση με την προσθήκη περισσότερων βημάτων για τα μηνύματα. Έτσι ο πίνακας δρομολόγησης μπορεί να έχει ένα νήμα εξόδου που να αντιστοιχεί σε δύο κόμβους. Και τελικά πέρα από τοπολογία δακτυλίου, μπορεί να επιτευχθούν άλλες τοπολογίες όπως πλέγμα, κύβος και υπερκύβος.

4.3 Ο χρονοπρογραμματισμός στο VDK

Το VDK είναι ένα προεκτοπιστικό και βασισμένο στην προτεραιότητα (pre-emptive multitasking) σύστημα. Αυτό σημαίνει ότι σε όλες τις διεργασίες χορηγούνται προτεραιότητες από το 1 εώς το 30, με το 1 να αντιπροσωπεύει τη μεγαλύτερη δυνατή

προτεραιότητα. Ο προεκτοπισμός σημαίνει ότι ενώ εκτελείται μια διεργασία, μπορεί να μπλοκαριστεί, ώστε να εκτελεστεί είτε μια Διαδικασία Εξυπηρέτησης Διακοπών (ISR) είτε μια διεργασία με μεγαλύτερη προτεραιότητα. Η φιλοσοφία αυτής της σχεδίασης, εξασφαλίζει ότι ανά πάσα στιγμή, η διεργασία με τη μεγαλύτερη προτεραιότητα είναι και αυτή που θα τρέχει.

Πώς υλοποιείται αυτό στην πράξη; Αυτό που συμβαίνει είναι να εκτελείται μια διεργασία μέχρι το τέλος της, εκτός και αν διακοπεί από κάποια διεργασία με μεγαλύτερη προτεραιότητα ή κάποια διακοπή. Όταν επανέλθει η σειρά της, συνεχίζει από το σημείο που σταμάτησε. Αυτό επιτυγχάνεται με τη χρήση μιας FIFO λίστας στην οποία εισάγονται όλες οι διεργασίες που βρίσκονται σε κατάσταση "Ready", ανάλογα με την προτεραιότητά τους και τη σειρά με την οποία μπήκαν στη λίστα αυτή. Με αυτό το σκεπτικό, όταν είναι έτοιμες πολλές διεργασίες της ίδιας προτεραιότητας, εκτελούνται με τη σειρά με την οποία μπήκαν στη λίστα, πρώτα η μία, μετά η άλλη κ.ο.κ. Ωστόσο το VDK επιτρέπει στο χρήστη να ρυθμίσει μια προτεραιότητα, όποιες διεργασίες έχουν την οποία, εκτελούνται με τον αλγόριθμο χρονοπρογραμματισμού Round Robin. Δηλαδή υπάρχει χωρισμός του χρόνου σε χρονοθυρίδες και σε όλες τις διεργασίες μοιράζεται από μία, ώστε αυτές να εναλλάσονται κυκλικά στον επεξεργαστή και να εκτελούνται. Θα πρέπει να σημειωθεί επιπλέον η μέθοδος με την οποία μια διεργασία παραδίδει οικιοθελώς τα ηνία του επεξεργαστή για να εκτελεστεί κάποια άλλη. Αυτή η διαδικασία επιτυγχάνεται μέσα από τη χρήση μιας συγκεκριμένης κλήσης συστήματος, που ουσιαστικά τοποθετεί την τρέχουσα διεργασία στο τέλος της FIFO λίστας. Η μέθοδος αυτή ονομάζεται συνεργιστικός (cooperative) χρονοπρογραμματισμός.

Συνολικά δηλαδή, ο προεκτοπισμός μπορεί να πραγματοποιηθεί από:

- Νήμα υψηλότερης προτεραιότητας
- Αλγόριθμο Round Robin
- Interrupt
- Εκούσια, μέσα από τον κώδικα του ίδιου του νήματος

Υπάρχουν ωστόσο περιπτώσεις όπου ένα νήμα, ανεξαρτήτως προτεραιότητας, εκτελείται απερίσπαστο. Αυτό συμβαίνει όταν το νήμα υλοποιεί μέρος του κώδικα, όπου πραγματοποιείται προσπέλαση κοινών πόρων του συστήματος και γίνεται διαχείριση ολικών (global) οντοτήτων του συστήματος, οπότε μια αλλαγή από κάποια άλλη διεργασία θα είχε αρνητικά αποτελέσματα. Το μέρος αυτό του κώδικα είναι συνηθισμένο να αποκαλείται "κρίσιμη περιοχή" (critical region) και σηματοδοτείται από την απενεργοποίηση των διακοπών και του χρονοπρογραμματιστή. Αυτό πραγματοποιείται με τη βοήθεια συγκεκριμένων κλήσεων του συστήματος που τοποθετούνται πριν από την κρίσιμη περιοχή. Φυσικά με τη χρήση επίσης των συμπληρωματικών κλήσεων συστήματος με το πέρας του κρίσιμου τμήματος, αναλαμβάνεται η επανενεργοποίηση των διακοπών και του χρονοπρογραμματιστή. Έτσι ακόμα και αν μια διεργασία υψηλότερης προτεραιότητας γίνει διαθέσιμη, ή μια

συσκευή Ε/Ε ειδοποιεί για μια ενέργεια, η διεργασία που βρίσκεται στο κρίσιμο τμήμα της συνεχίζει απερίσπαστα την εκτέλεσή της.

Αντίστοιχες με τις κρίσιμες περιοχές είναι οι "απρογραμμάτιστες περιοχές" (unscheduled regions), που αντιστοιχούν σε συγκεκριμένα κομμάτια κώδικα ενός νήματος. Ομοίως κι εδώ, απενεργοποιείται ο χρονοπρογραμματιστής ώστε να αποτραπεί ο προεκτοπισμός από υψηλότερης προτεραιότητας νήματα. Ωστόσο οι διακοπές δεν απενεργοποιούνται και μπορούν αυτές να αντικαταστήσουν τη διεργασία στην εκτέλεση. Οι κρίσιμες και οι απρογραμμάτιστες περιοχές είναι διαπλεκόμενες, δηλαδή μπορεί μια διεργασία να μπει από μια κρίσιμη περιοχή σε μια απρογραμμάτιστη και αντίστροφα.

4.4 Η διαδιεργασιακή επικοινωνία και συγχρονισμός

Πολύ σημαντικό κεφάλαιο στα RTOSs είναι η διαδιεργασιακή επικοινωνία και συγχρονισμός. Τα ζητήματα αυτά προκύπτουν υπό την οπτική γωνία ότι οι διεργασίες ανταγωνίζονται για την εκμετάλλευση των κοινών πόρων του συστήματος (δίσκων, μνημών RAM ή Flash, συσκευών εισόδου/εξόδου), ή συνεργάζονται για να φέρουν εις πέρας έναν κοινό σκοπό. Μάλιστα η κατάσταση γίνεται ακόμα πιο πολύπλοκη, όταν οι διεργασίες προς συγχρονισμό εκτελούνται σε διαφορετικούς επεξεργαστές, κάτι τυπικό για πολυεπεξεργαστικά συστήματα όπως ο Blackfin 561 της εργασίας. Για τους λόγους αυτούς είναι απαραίτητος ένας καλά δομημένος και ορισμένος τρόπος μεταφοράς πληροφοριών ανάμεσα στα νήματα.

Οι διεργασίες ανταγωνίζονται για την εκμετάλλευση των κοινών πόρων του συστήματος (δίσκων, μνημών RAM ή Flash, συσκευών εισόδου/εξόδου), ή συνεργάζονται για να φέρουν εις πέρας έναν κοινό σκοπό

Διάφορες μέθοδοι χρησιμοποιούνται στα Λ.Σ. και στα RTOSs. Οι τέσσερεις τρόποι που χρησιμοποιεί το VDK είναι οι ακόλουθοι:

- Σηματοφόροι (Semaphores)
- Μηνύματα (Messages)
- Γεγονότα και bits γεγονότων (Events and Event bits)
- Σημαίες Συσκευής (Device Flags)

Παρακάτω θα αναλυθεί πώς το VDK υλοποιεί καθέναν από τους παρακάτω μηχανισμούς

Σηματοφόροι

Ο σηματοφόρος είναι ένα σύμβολο (token) που το VDK δημιουργεί είτε στατικά, κατά την έναρξη, είτε δυναμικά, μέσα από τον κώδικα των νημάτων, προκειμένου να αξιοποιηθεί από τα εκτελούμενα νήματα. Σκοπός του είναι ο έλεγχος της πρόσβασης σε έναν κοινό πόρο του συστήματος, επιτρέποντας το συγχρονισμό των νημάτων. Ακόμη, χρησιμοποιείται για να σηματοδοτήσει ένα γεγονός του συστήματος, ή για να προγραμματίσει την περιοδική εκτέλεση των νημάτων.

Το VDK υποστηρίζει τριών ειδών σηματοφόρους: Το σηματοφόρο μέτρησης (counting semaphore) ο οποίος παίρνει σαν τιμή ένα θετικό ακέραιο αριθμό, το δυαδικό σηματοφόρο (binary semaphore) που παίρνει μόνο δύο τιμές, 0 ή 1 και τον περιοδικό σηματοφόρο (periodic semaphore) ο οποίος δημοσιεύεται με περίοδο n.

Πώς γίνεται όμως η χρήση των σηματοφόρων από τα νήματα; Αυτό συμβαίνει μέσα από μια αίτηση για δέσμευση (pending) ενός διαθέσιμου σηματοφόρου μέσα στον κώδικα του νήματος, με διαθέσιμο να εννοούμε εκείνον το σηματοφόρο που έχει μη μηδενική τιμή. Σε περίπτωση που το νήμα δεσμεύσει το σηματοφόρο, συνεχίζει την εκτέλεσή του. Αν πάλι όχι, το νήμα μπλοκάρεται και μπαίνει σε μια λίστα αναμονής μέχρι να δεσμεύσει το σηματοφόρο. Από εδώ και πέρα υπάρχουν δύο περιπτώσεις. Στην πρώτη, το νήμα λαμβάνει έπειτα από λίγο το σηματοφόρο και εισέρχεται στη λίστα με τις «έτοιμες» (Ready) διεργασίες. Θα πρέπει να σημειωθεί πάντως, ότι σε περίπτωση που αποδεσμευθεί ένας σηματοφόρος, το νήμα της λίστας αναμονής που θα τον δεσμεύσει είναι αυτό με τη μεγαλύτερη προτεραιότητα και όχι αυτό που περιμένει τον περισσότερο χρόνο. Στη δεύτερη περίπτωση, ο μέγιστος χρόνος αναμονής (timeout) του νήματος τελειώνει και τα ηνία αναλαμβάνει η συνάρτηση Error του νήματος αυτού. Εν πάσει περιπτώσει, είναι προγραμματιστικά συνετό η δέσμευση ενός σηματοφόρου να μην επιχειρείται στον κώδικα κρίσιμης/απρογραμμάτιστης περιοχής. Σε αυτές τις περιοχές ο χρονοπρογραμματιστής είναι απενεργοποιημένος, έτσι σε περίπτωση που ο σηματοφόρος δε δεσμευθεί λόγω μη διαθεσιμότητας, το τρέχον νήμα θα μπλοκαριστεί ενώ συνάμα κανένα άλλο νήμα δε θα το αντικαταστήσει στην εκτέλεση.

Η αποδέσμευση των σηματοφόρων μπορεί να γίνει σε δύο πεδία, στο πεδίο των νημάτων και στο πεδίο των ρουτινών διακοπής. Στην πρώτη περίπτωση, η αποδέσμευση του σηματοφόρου οδηγεί στην αύξηση της τιμής του κατά 1 και ακολουθούν τρία σενάρια: Σε περίπτωση που κανένα άλλο νήμα δε βρίσκεται στην ουρά αναμονής για σηματοφόρο, η τιμή του τελευταίου απλά αυξάνεται, χωρίς όμως να μπορεί να ξεπεράσει τη μέγιστη τιμή του. Εάν υπάρχουν νήματα αναμονής και ο σηματοφόρος αποδεσμεύτηκε σε περιοχή που ο χρονοπρογραμματιστής είναι ενεργοποιημένος, μπορεί να προεκτοπιστεί από ένα νήμα υψηλότερης προτεραιότητας που, όντας στη λίστα αναμονής, δέσμευσε το δημοσιευμένο σηματοφόρο. Εάν ωστόσο η αποδέσμευση έλαβε χώρα σε απρογραμμάτιστη ή κρίσιμη περιοχή, ακόμα και υψηλότερης προτεραιότητας νήμα να δεσμεύσει το σηματοφόρο, θα αναμένει στην ουρά με τα «έτοιμα» νήματα, μέχρι να επανενεργοποιηθεί ο χρονοπρογραμματιστής.

Στην περίπτωση της αποδέσμευσης σηματοφόρου μέσα από ρουτίνα διακοπής, τα σενάρια είναι παρόμοια, μόνο που σε κάθε περίπτωση η ρουτίνα εκτελείται μέχρι τέλους και αν κάτι προεκτοπίζεται από νήμα υψηλότερης προτεραιότητας, είναι το νήμα που διακόπηκε, αφότου επανέλθει ο έλεγχος σε αυτό και όχι η ρουτίνα διακοπής.

Όλες αυτές οι αλληλεπιδράσεις νημάτων και σηματοφόρων, δημιουργία, καταστροφή, δέσμευση αποδέσμευση γίνεται μέσα από κλήσεις στο API.

Μηνύματα

Ένας άλλος τρόπος για την επικοινωνία μεταξύ των νημάτων, είναι τα μηνύματα. Όπως και οι σηματοφόροι, αναλαμβάνουν το συχρονισμό των νημάτων αλλά και τη μετάδοση δεδομένων. Ωστόσο η ύπαρξή τους καθορίζει μια ξεχωριστή κατηγορία Λ.Σ., τα «προσανατολισμένα σε μηνύματα». Σε αυτό το μοντέλο, οι διεργασίες φροντίζεται να είναι λίγες αλλά σχετικά μεγάλες, με τον καθορισμό καναλιών μεταξύ αυτών για την ανταλλαγή μηνυμάτων. Αυτό είναι αντίθετο από το έτερο μοντέλο, τα «προσανατολισμένα σε διεργασίες» Λ.Σ, όπου οι διεργασίες φροντίζεται να είναι μικρές και ταχύτατα εναλλασόμενες, ώστε η επικοινωνία να γίνεται με κοινά δεδομένα στη μνήμη. Ωστόσο τα μοντέλα αυτά θεωρούνται δυϊκά, δηλαδή μια διεργασία ή υποσύστημα μπορεί να χαρτογραφηθεί από το ένα μοντέλο στο άλλο (Lauer & Needham, 1978).

Πράγματι, το VDK ορίζει μέχρι 15 διαφορετικά κανάλια για την επικοινωνία μεταξύ των διεργασιών, με αύξουσα προτεραιότητα. Για την ενεργοποίση των μηνυμάτων, θα πρέπει ο τύπος νήματος που δημιουργείται να οριστεί σαν message-enabled. Σε αντίθετη περίπτωση, μπορεί μεν να αποστείλει μηνύματα, αλλά δεν μπορεί να παραλάβει. Προαιρετικά, μπορεί να αποσταλεί ένα φορτίο (payload) μαζί με το μήνυμα το οποίο δεν αποτελεί παρά μια αναφορά σε έναν buffer δεδομένων. Ο μέγιστος αριθμός των δυνατών μηνυμάτων ορίζεται κατά την έναρξη του project.

Το νήμα που στέλνει ένα μήνυμα, συνεχίζει κανονικά την εκτέλεσή του εκτός και αν το μήνυμα ενεργοποιήσει ένα νήμα υψηλότερης προτεραιότητας το οποίο θα προεκτοπίσει το αρχικό. Απαραίτητη προϋπόθεση γι’αυτό είναι πάντα ο κώδικας του νήματος που εκτελείται να μη βρίσκεται σε απρογραμμάτιστη ή κρίσιμη περιοχή, όπου ο χρονοπρογραμματιστής είναι απενεργοποιημένος.

Τα μηνύματα που αποστέλλονται σε ένα νήμα, τοποθετούνται σε ουρά αναμονής τύπου FIFO, μία για κάθε κανάλι, από την οποία το νήμα κάνει αίτηση να τα διαβάσει για να συνεχίσει την εκτέλεσή του. Σε περίπτωση που κατά την αίτηση δεν υπάρχει κάποιο κατάλληλο μήνυμα, το αιτούν νήμα μπλοκάρεται μέχρι να φτάσει κάποιο. Όπως και στην περίπτωση των σηματοφόρων, εάν όσο είναι μπλοκαρισμένο το νήμα τελειώσει το timeout, σειρά έχει η συνάρτηση Error του.

Σε αντίθεση μάλιστα με τους σηματοφόρους, τα μηνύματα έχουν κάποιο νήμα που αποτελεί ιδιοκτήτη τους, αν και όχι σταθερό. Αυτό γιατί μέχρι την αποστολή του

μηνύματος, ο ιδιοκτήτης είναι το αποστέλλον νήμα. Όταν όμως το μήνυμα μπει στην ουρά αναμονής, ιδιοκτήτης γίνεται το παραλαμβάνον νήμα. Σε κάθε περίπτωση, το νήμα που κατέχει το μήνυμα, κατέχει και το φορτίο του.

Η καταστροφή των μηνυμάτων μπορεί να γίνει μόνο από το νήμα-ιδιοκτήτη. Πριν ένα νήμα τερματίσει την εκτέλεσή του, οφείλει να καταστρέψει όλα τα μηνύματα που κατέχει, «εκκαθαρίζοντας» επίσης αυτά που βρίσκονται στις ουρές αναμονής. Όλες αυτές οι αλληλεπιδράσεις μεταξύ νημάτων και μηνυμάτων, δημιουργία, αποστολή, αίτηση για παραλαβή, καταστροφή, πραγματοποιούνται μέσω κλήσεων στο API. Σε όλες τις κλήσεις πρέπει να αναφέρεται ως όρισμα κάποιο από τα 15 δυνατά κανάλια.

Σε συστήματα με περισσότερους επεξεργαστές, όπως στην προκειμένη περίπτωση, το VDSP++ από την έκδοση 3.5 και πάνω επιτρέπει την αποστολή μηνυμάτων ανάμεσα σε νήματα που εκτελούνται σε διαφορετικούς επεξεργαστές. Αυτό γίνεται με τον ορισμό των επεξεργαστών σαν κόμβων (nodes), κατά τη δημιουργία του project, καθένας από τους οποίους εκτελεί ένα διαφορετικό στιγμιότυπο του VDK. Όλες οι οντότητες (σηματοφόροι, μηνύματα, νήματα) είναι τότε ιδιωτικές για κάθε κόμβο. Προς το παρόν υποστηρίζονται 32 διαφορετικοί επεξεργαστές-κόμβοι, λόγω μιας επιβάρυνσης (overhead) 5 bit στη μεταβλητή ThreadID που αποτελεί τη χαρακτηριστική ταυτότητα του νήματος. Τα μηνύματα, δρομολογούνται με τη βοήθεια των νημάτων δρομολόγησης, τα οποία αναφέρθηκαν στη σχετική παράγραφο. Συγκεκριμένα κατά τη δημιουργία ενός μηνύματος, εξετάζονται τα 5 bits του ThreadID που αφορά το νήμα-παραλήπτη. Εάν είναι ίδια με αυτά του νήματος αποστολέα βρίσκονται και τα δύο στον ίδιο κόμβο άρα το μήνυμα έχει την ίδια αντιμετώπιση όπως στην περίπτωση μονοεπεξεργαστικού συστήματος. Εάν είναι διαφορετικά, πραγματοποιούνται οι αλγόριθμοι δρομολόγησης των νημάτων δρομολόγησης.

Γεγονότα και Bits Γεγονότων

Τα γεγονότα και τα bits γεγονότων χρησιμεύουν στην επίτρεψη της εκτέλεσης των νημάτων, σε συνάρτηση με την κατάσταση του συστήματος. Τα bits γεγονότων, όταν είναι TRUE, σηματοδοτούν ότι ένα στοιχείο (element) του συστήματος βρίσκεται σε μια συγκεκριμένη κατάσταση. Το γεγονός, είναι ένας τελεστής bool που πραγματοποιείται σε κάποια bits γεγονότων και είναι TRUE όταν αυτό είναι το αποτέλεσμα του τελεστή. Τα νήματα που κάνουν αίτηση για ένα γεγονός, ουσιαστικά ελέγχουν την κατάστασή του. Όταν αυτό είναι TRUE, όλα τα νήματα που το ελέγχουν μπαίνουν στη λίστα με τις έτοιμες διεργασίες. Όταν όμως είναι FALSE, αυτά μπλοκάρονται μέχρι το γεγονός να γίνει TRUE, ή να περάσει ο μέγιστος χρόνος αναμονής (timeout). Η υλοποίηση αυτή μπορεί να αντιστοιχηθεί με ένα βρόχο while όπως ο παρακάτω:

```
while(event==TRUE)
{
continue_exexution
}
```

Σε περίπτωση που ένα bit γεγονότος αλλάξει, ο χρονοπρογραμματιστής υπολογίζει εκ νέου την τιμή όλων των εξαρτώμενων γεγονότων, εκτός και αν η αλλαγή έγινε μέσα από απρογραμμάτιστη ή κρίσιμη περιοχή. Τέλος θα πρέπει να σημειωθεί ότι το πλήθος των bit γεγονότων ισούται με το μέγεθος λέξης του επεξεργαστή μείον 1. Έτσι για το Blackfin η τιμή αυτή είναι 31.

Σημαίες Συσκευών
Όπως οι υπόλοιπες οντότητες, έτσι και οι σημαίες συσκευών αναλαμβάνουν το συγχρονισμό και την επικοινωνία. Ωστόσο έχουν άμεση σχέση με τους οδηγούς συσκευών, πέρα από τα νήματα. Συγκεκριμένα, τα νήματα κάνουν αίτηση για μια σημαία συσκευρή και ο διεκπεραιωτής της αντίστοιχης Ρουτίνας Εξυπηρέτησης Διακοπών (ISR) τη δημοσιεύει, μέσα από αντίστοιχες κλήσεις API.

Τι όμως σημαίνει αναμονή για την ανάγνωση μιας σημαίας; Ουσιαστικά δηλώνει την αναμονή ενός νήματος για τον ορισμό κάποιας συνθήκης σχετικά με μια συσκευή του υλικού. Η αρκετά αφηρημένη πρόταση που διατυπώθηκε προηγουμένως, μπορεί κάλλιστα να αποσαφηνιστεί με ένα παράδειγμα. Θα μπορούσε λοιπόν η μετατροπή μιας σημαίας από FALSE σε TRUE να σηματοδοτεί το ότι ένας νέος buffer δεδομένων μιας συσκευής A/D έγινε διαθέσιμος, ώστε να το γνωρίζουν όλα τα νήματα που έκαναν αίτηση. Η συμπεριφορά αυτή κρύβει μια ειδική μεταχείριση των κλήσεων API που αναλαμβάνουν να δεσμεύσουν και να δημοσιεύσουν σημαίες. Ας χρησιμοποιήσουμε ένα άλλο παράδειγμα για να γίνει αυτό πιο ξεκάθαρο. Έστω ότι ένα νήμα κάνει αίτηση για ανάγνωση μιας σημαίας και αμέσως τυγχάνει μια διακοπή στην οποία η προς ανάγνωση σημαία αλλάζει τιμή. Το νήμα τότε που έκανε την αίτηση θα διαβάσει λανθασμένη τιμή. Η ειδική μεταχείριση που αναφέρθηκε προηγουμένως, δεν είναι παρά η εξασφάλιση ότι όλες οι αιτήσεις για ανάγνωση σημαίας, λαμβάνουν χώρα μέσα σε κρίσιμη περιοχή όπου οι διακοπές είναι απενεργοποιημένες.

Τέλος θα πρέπει να σημειωθεί ότι όταν κάποιο νήμα κάνει αίτηση για κάποια σημαία, μπλοκάρεται αυτόματα. Όταν η σημαία αυτή δημοσιευτεί από τον αντίστοιχο διεκπεραιωτή, όλα τα νήματα τα οποία έκαναν αντίστοιχη αίτηση μεταφέρονται στη λίστα με τις έτοιμες διεργασίες. Αυτή ωστόσο η πράξη είναι μη-ντερμινιστική καθώς δεν υπάρχει μηχανισμός για την πρόβλεψη του πλήθους των νημάτων που θα μεταφερθούν.

4.5 Η διαχείριση μνήμης

Για τη διαχείριση της μνήμης έπρεπε να ληφθεί ειδική μέριμνα ώστε να ικανοποιούνται οι ντετερμινιστικές απαιτήσεις που ενυπάρχουν σε ένα RTOS όπως το VDK. Έτσι χρησιμοποιείται η μέθοδος των πισινών μνήμης (memory pools) όπως αυτή περιγράφηκε στο ανάλογο κεφάλαιο για τα RTOSs. Αυτή επιτρέπει σε αντίθεση με την παραδοσιακή malloc, ντετερμινιστικούς χρόνους αναζήτησης και προστασία ενάντια στην εξωτερική κατάτμηση.

Υλοποιείται ως εξής: Ένα μέρος της σωρού, χωρίζεται σε ισομεγέθη blocks ορίζοντας μια πισίνα. Έτσι μπορούν να οριστούν πολλές πισίνες, καθεμία με διαφορετικό ονομαστικό μέγεθος block. Εάν μάλιστα υποστηρίζονται από την αρχιτεκτονική πολλαπλές σωροί, μπορεί σε καθεμία από αυτές να βρίσκεται διαφορετική πισίνα.

Οι πισίνες ορίζονται είτε στατικά, κατά την έναρξη του project, είτε δυναμικά, μέσα από κατάλληλες κλήσεις στο API που εκτελούνται στον κώδικα των νημάτων. Σε αυτήν την περίπτωση, το μέγεθος και το πλήθος των blocks θα πρέπει να περάσει σαν όρισμα στην κλήση. Ο μέγιστος αριθμός πάντως των πισινών που μπορεί να υποστηριχθεί, όριζεται στην αρχή του project, όταν αυτό χτίζεται.

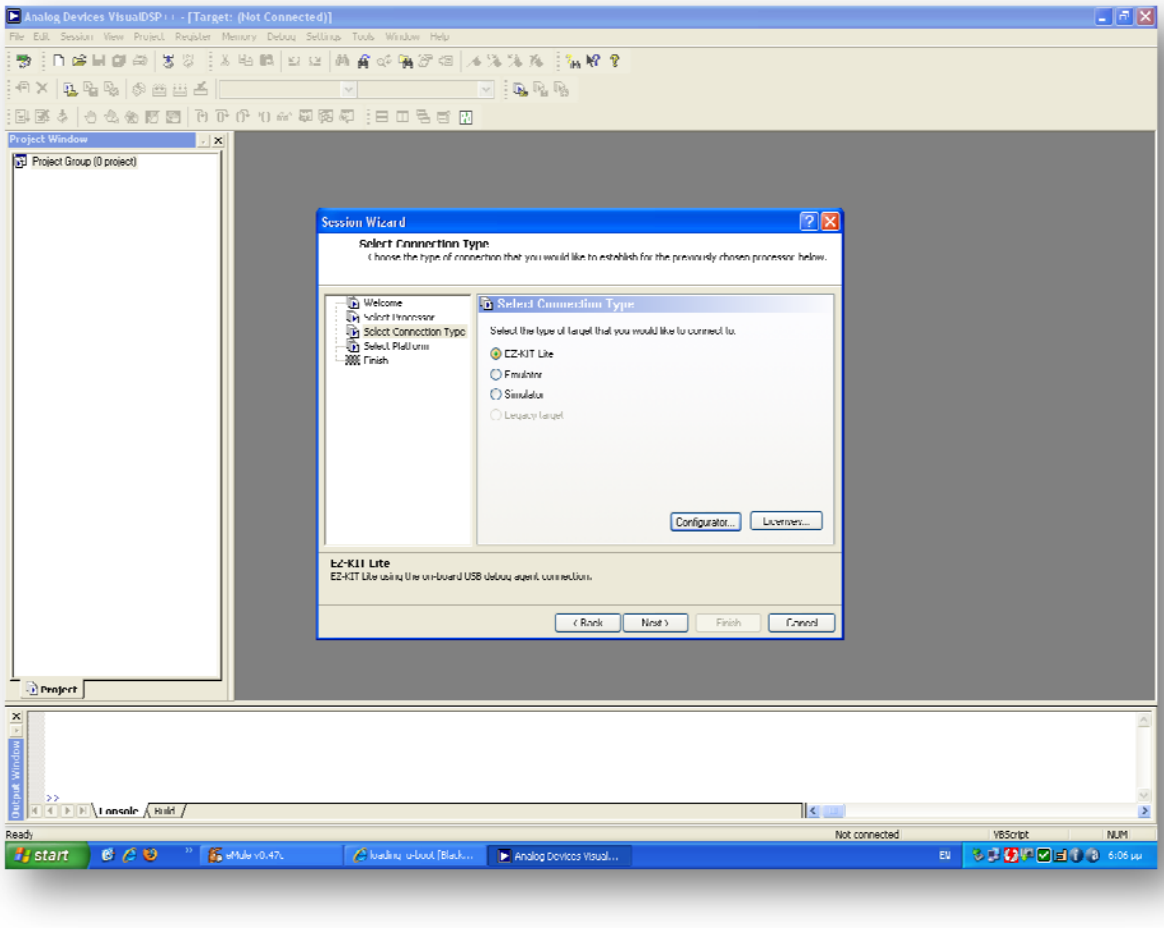
Τέλος θα πρέπει να σημειωθεί ότι η εκχώρηση της μνήμης, έπειτα από επιτυχή κλήση για νέα πισίνα, γίνεται με δύο τρόπους. Κατά τη δημιουργία της πισίνας, οπότε η δεύσμευση και αποδέσμευση των blocks γίνεται σε ντετερμινιστικούς χρόνους, ή κατ'απαίτηση του block, οπότε η χρονική επιβάρυνση μετατοπίζεται από το σκέλος της κατασκευής της πισίνας στο σκέλος της απαίτησης κάθε block.

5 Δημιουργία ενός απλού προγράμματος "Hello World"

Το πρακτικό σκέλος της εργασίας αυτής, αφορά τη δημιουργία ενός προγράμματος που θα τρέχει από το VDK και θα τυπώνει το μήνυμα “Hello World from VDK”. Λόγω διαθεσιμότητας της πλατφόρμας Blackfin 561 EZ-KIT Lite το πρόγραμμα τόσο από μέσα από το περιβάλλον εξοιμοίωσης του VDSP++ όσο και μέσα από τον προαναφερθέντα επεξεργαστή. Η διαδικασία αυτή διαφέρει μόνο σε ένα στάδιο αν και το αποτέλεσμα δεν διαφέρει οπτικά, καθώς και στις δύο περιπτώσεις χρησιμοποιείται η κονσόλα του VDSP++ για την τύπωση του αποτελέσματος. Απλώς στην περίπτωση του Blackfin, το πρόγραμμα μεταφέρεται στη μνήμη του και εκτελείται από εκεί. Ακολουθεί η περιγραφή της συνολικής διαδικασίας.

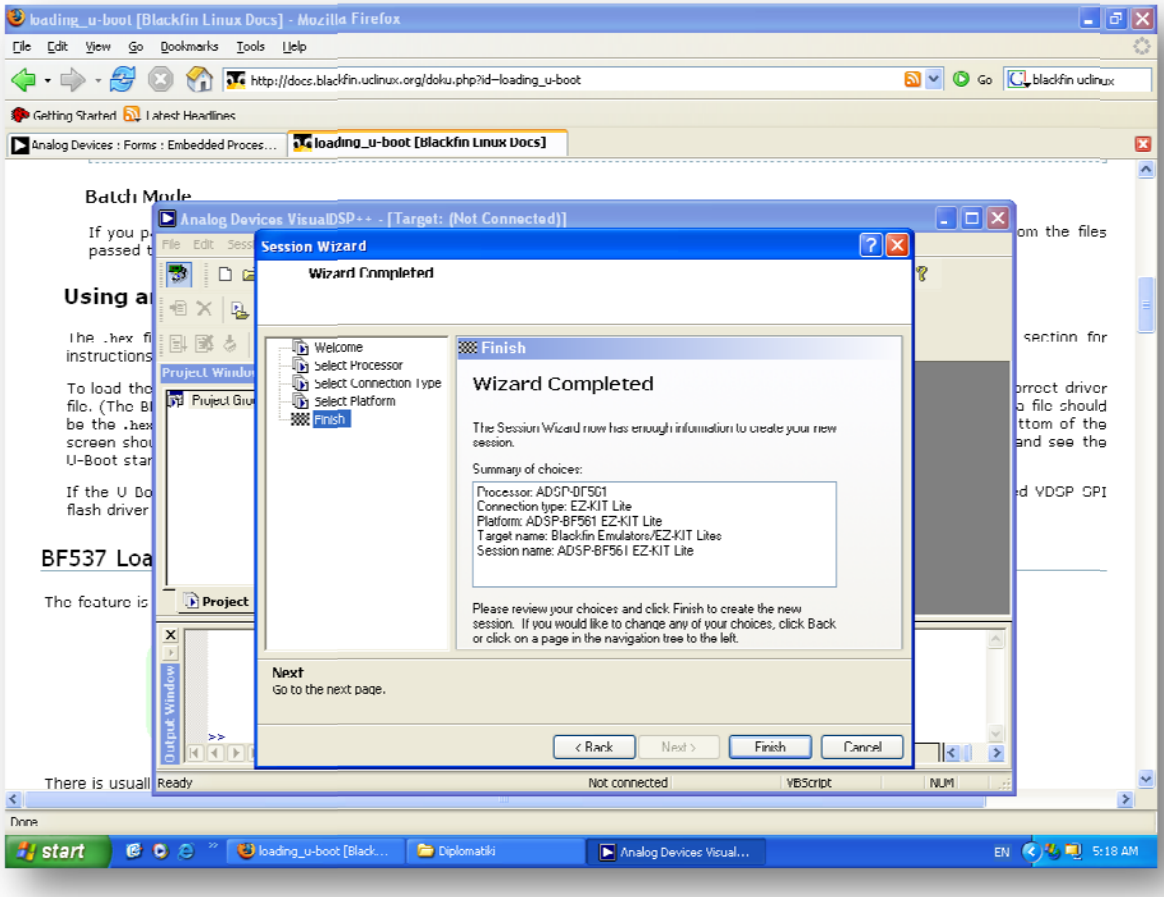
Αρχικά απαιτείται η εγκατάσταση του προγράμματος VDSP++. Αυτό μπορεί να το κατεβάσει ο οποιοσδήποτε από την ιστοσελίδα της εταιρείας (www.analog.com) με άδεια χρήσης 90 ημερών για σκοπούς μόνο εξομοίωσης, είτε από το CD που συνοδεύει τους επεξεργαστές τις εταιρείας, με άδεια για σύνδεση με τον επεξεργαστή. Για τις ανάγκες της εργασίας χρησιμοποιήθηκε η νεότερη έκδοση του VDSP ++ από την ιστοσελίδα, αλλά με τους κωδικούς του CD που συνόδευε τον επεξεργαστή Blackfin-561 το οποίο παρείχε το εργαστήριο VLSI της σχολής.

Επόμενο βήμα είναι ο καθορισμός της συνόδου (session). Από εδώ αποφασίζεται εάν το πρόγραμμα θα εκτελεστεί σε περιβάλλον εξομοίωσης ή από τον επεξεργαστή με σύνδεση. Θα δημιουργηθούν δύο διαφορετικές εκδοχές, μία για εξομοίωση (Simulation) και μία για εκτέλεση από τον επεξεργαστή (EZ-KIT Lite). Πριν από την εκτέλεση του προγράμματος ο χρήστης μπορεί να εναλλάσεται κατά βούληση ανάμεσα στις δύο αυτές συνόδους. Αυτή είναι και η μόνη διαφορά ανάμεσα στις δύο υλοποιήσεις.



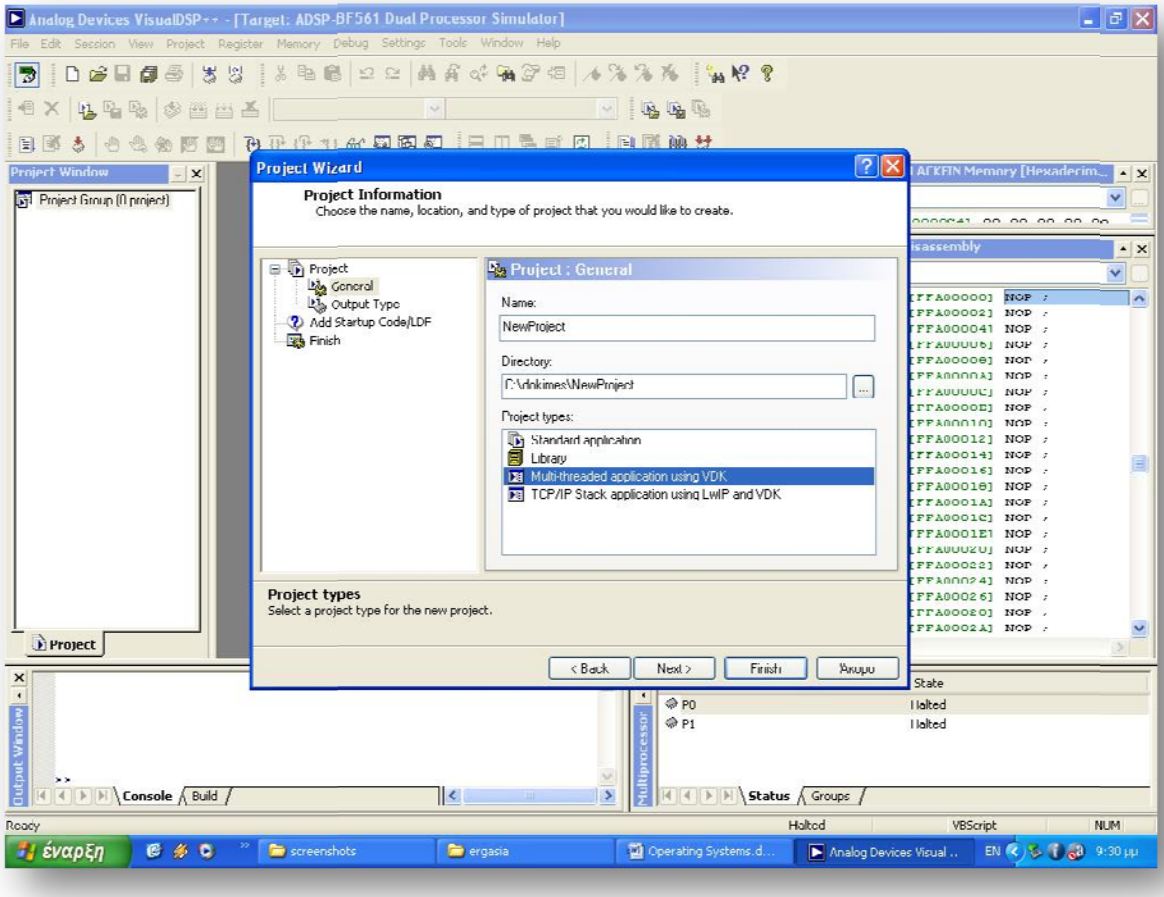
Εικόνα 6: Επιλογή της συνόδου

Ακολουθεί μια σειρά από επιλογές που αναφέρονται στον επεξεργαστή. Από εκεί επιλέγεται ο επεξεργαστής Blackfin 561.



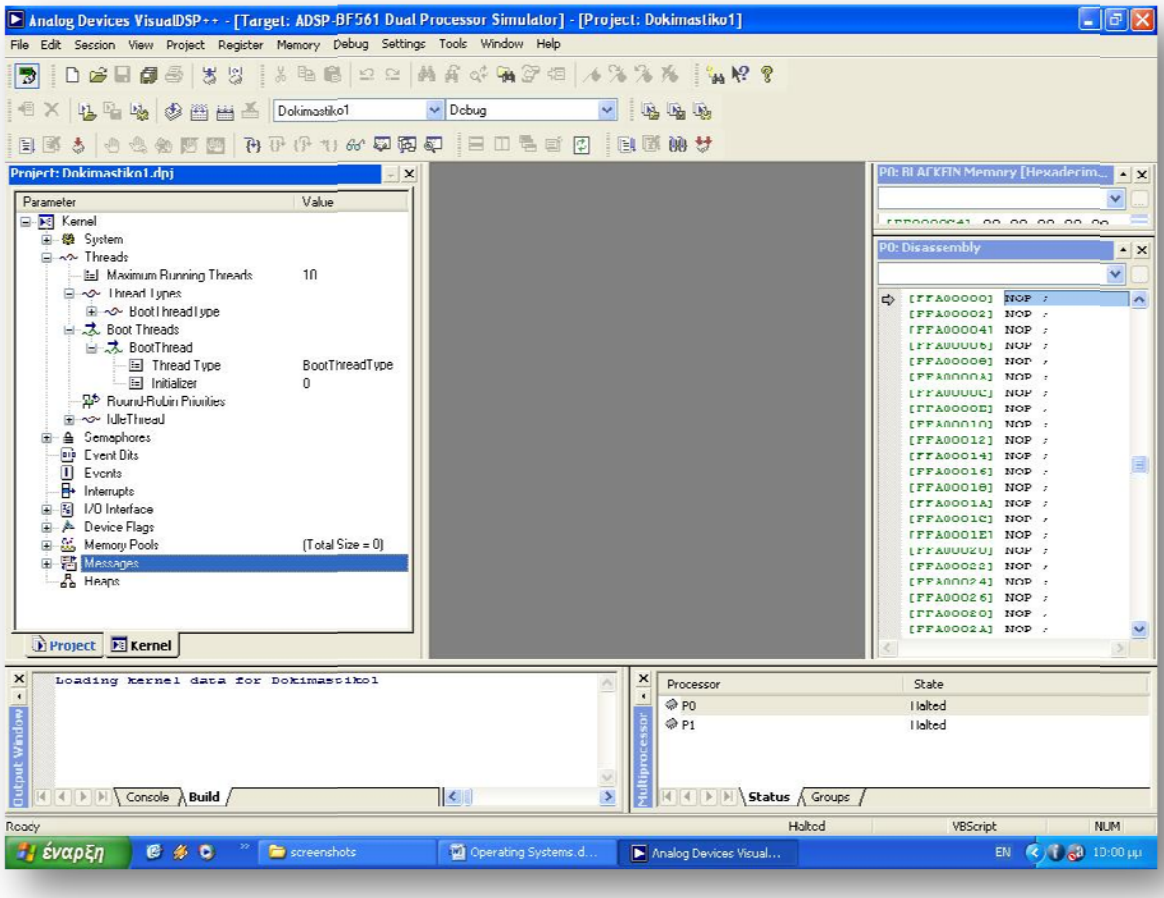
Εικόνα 7: Επιλογή του επεξεργαστή Blackfin 561

Κατά τη δημιουργία ενός καινούριου project, θα πρέπει να επιλεχθεί η δημιουργία εφαρμογής με τον πυρήνα VDK.



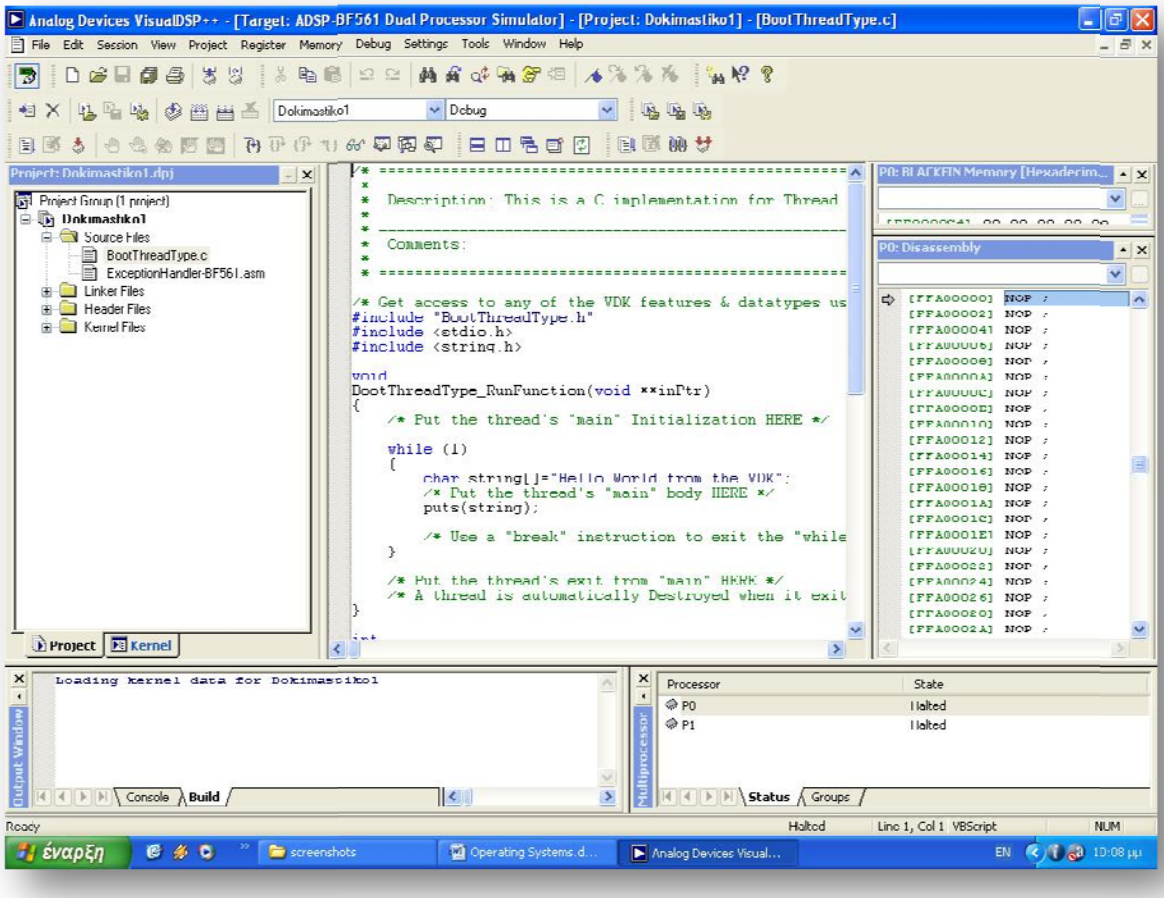
Εικόνα 8: Επιλογή του VDK

Στην καρτέλα kernel που εμφανίζεται, μπορούν να επιλεχθούν όλα τα χαρακτηριστικά του πυρήνα. Σηματοφόροι, προτεραιότητες, διακοπές, νήματα, μηνύματα, όλα αυτά μπορούν να καθοριστούν στατικά από την καρτέλα αυτή, αλλά και δυναμικά μέσα από τον κώδικα των νημάτων. Για τις ανάγκες τις εργασίες, δεν θα πραγματοποιηθεί κάτι πολύπλοκο, αλλά ο κώδικας του Hello World θα ενσωματωθεί στη RunFunction() ενός νήματος εκκίνησης. Το νήμα εκκίνησης δημιουργείται ως εξής. Πρώτα δημιουργείται ένας νέος τύπος νήματος με την ονομασία BootThreadType. Επιλέγεται ως γλώσσα νήματος η C. Ακολούθως, κατασκευάζεται ένα νέο αντικείμενο BootThread, με τύπο BootThreadType. Το νήμα εκκίνησης φαίνεται στην παρακάτω εικόνα.



Εικόνα 9: Δημιουργία ενός νήματος εκκίνησης

Με τη δημιουργία του τύπου BootThreadType, αυτόματα εμφανίζεται στην καρτέλα project το ομώνυμο αρχείο με τον πηγαίο κώδικα.



Εικόνα 10: Το αρχείο με τον πηγαίο κώδικα

Ακολουθεί ο κώδικας του νήματος. Το σημείο που εκτελείται είναι τυπωμένο με πράσινο χρώμα και αντιστοιχεί στη RunFunction του νήματος. Είναι πάντως φανερές και οι άλλες συναρτήσεις, error, create, init και destructor αν και είναι υλοποιημένες ως null συναρτήσεις.

```
/*
==========================================================================
=====
 *
 *    Description: This is a C implementation for Thread BootThreadType
 *
 * --------------------------------------------------------------------
 *    Comments:
 *
 *
==========================================================================
```

```
===*/

/* Get access to any of the VDK features & datatypes used */
#include "BootThreadType.h"
#include <stdio.h>

void
BootThreadType_RunFunction(void **inPtr)
{
	/* Put the thread's "main" Initialization HERE */

	while (1)
	{
		char string[]="Hello World from the VDK";
		/* Put the thread's "main" body HERE */
		puts(string);

		/* Use a "break" instruction to exit the "while (1)" loop */
	}

	/* Put the thread's exit from "main" HERE */
	/* A thread is automatically Destroyed when it exits its run function */
}

int
BootThreadType_ErrorFunction(void **inPtr)
{

	/* TODO - Put this thread's error handling code HERE */

	   /* The default ErrorHandler goes to KernelPanic */

	   VDK_CThread_Error(VDK_GetThreadID());
	   return 0;
}

void
BootThreadType_InitFunction(void **inPtr, VDK_ThreadCreationBlock *pTCB)
{
	/* Put code to be executed when this thread has just been created HERE */

	/* This routine does NOT run in new thread's context.    Any non-static thread
	 *      initialization should be performed at the beginning of "Run()."
	 */
```

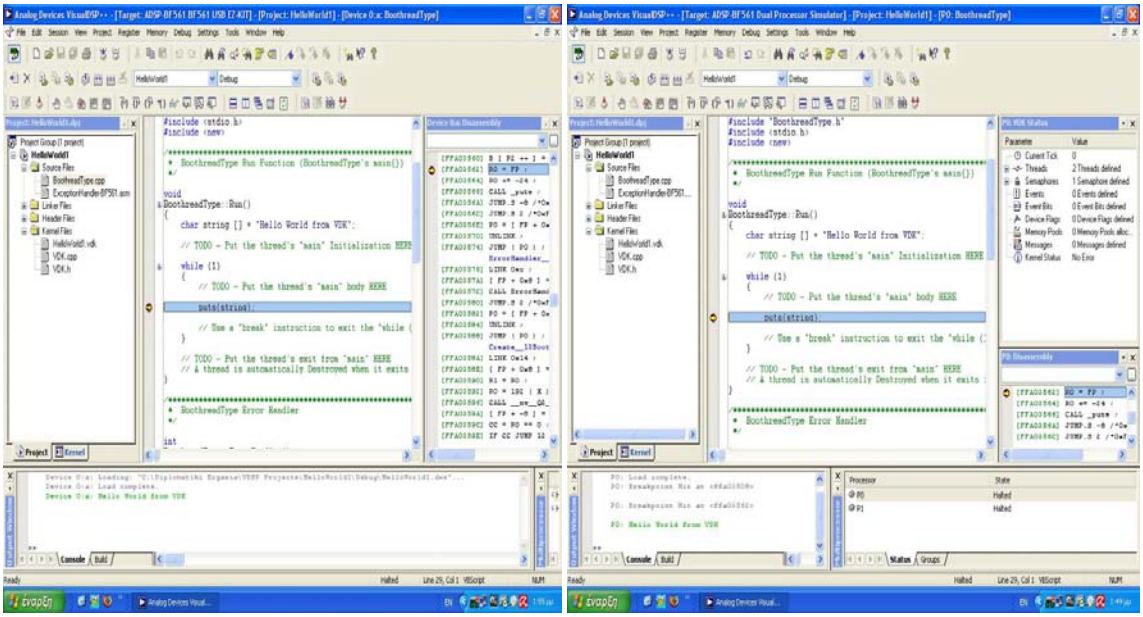
```
}

void
BootThreadType_DestroyFunction(void **inPtr)
{
	/* Put code to be executed when this thread is destroyed HERE */

	/* This routine does NOT run in the thread's context.    Any VDK API calls
	 *	should be performed at the end of "Run()."
	 */
}

/*
**************************************************************************
== */
```

Το αποτέλεσμα της εκτέλεσης του κώδικα αυτού φαίνεται στις δύο εικόνες. Η κονσόλα κάτω αριστερά τυπώνει το ζητούμενο μήνυμα. Η μόνη διαφορά ανάμεσα στις δύο συνόδους, εξομοίωσης και σύνδεσης, φαίνεται στη μπάρα στο ανώτερο σημείο του παραθύρου. Στα αριστερά βρίσκεται η υλοποίηση με απευθείας εκτέλεση στον επεξεργαστή Blackfin, ενώ στα δεξιά η υλοποίηση σε περιβάλλον εξομείωσης.



Εικόνα 11: Το μήνυμα "Hello World from VDK" με εκτέλεση στον επεξεργαστή (αριστερά) και με εκτέλεση σε περιβάλλον εξομοίωσης(δεξιά)

6 Βιβλιογραφία

Lauer, H. C., & Needham, R. M. (1978). On the Duality of Operating System Structures. *Second International Symposium on Operating Systems* (σσ. 3-19). Operating Systems Review.

Li, Q., & Yao, C. (2003). *Real Time Concepts for Embedded Systems.* CMP Books.

Tanenbaum, A. S. (2001). *Modern Operating Systems, 2nd Edition.* Prentice Hall.

Yaghmour, K. (2003). *Building Embedded Linux Systems.* O'Reilly.

7 Internet

http://linuxdevices.com/articles/AT4627965573.html