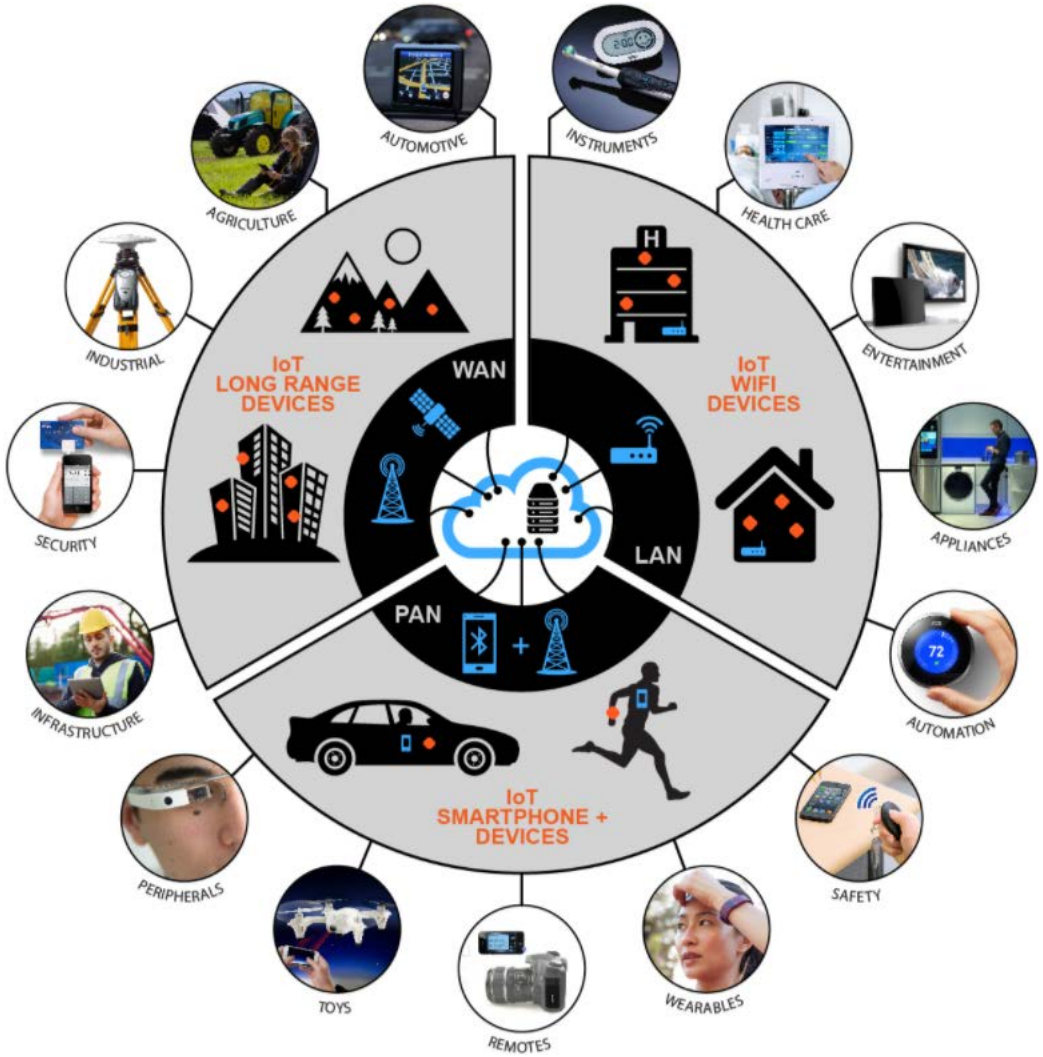
Διαδίκτυο των Πραγμάτων

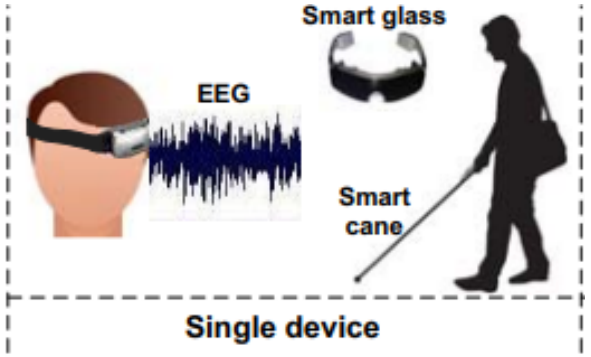
Λειτουργικά Συστήματα για συσκευές IoT

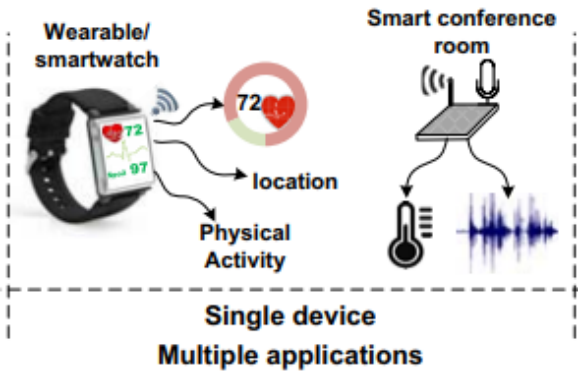
Περιεχόμενα

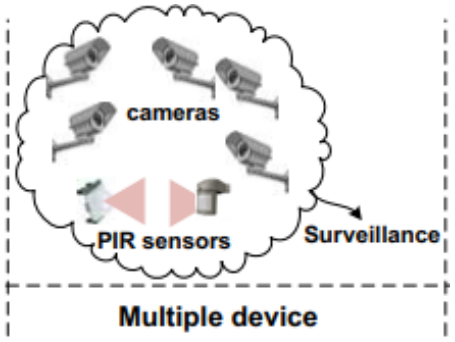
ΕΙΣΑΓΩΓΗ

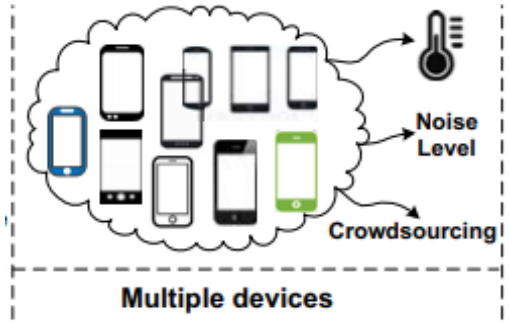
Τοπίο και εφαρμογές IoT







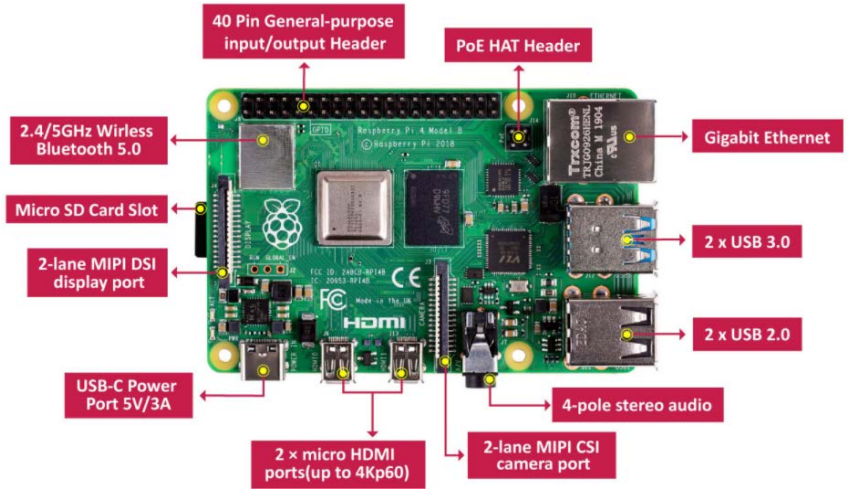




Υλικό (hardware) συσκευών IoT

- Το υλικό μπορεί να ποικίλει πολύ από εφαρμογή σε εφαρμογή. Πολλές συσκευές που χρησιμοποιούνται για άλλες λειτουργίες (π.χ. **smart phones**) μπορούν να παίξουν το ρόλο της συσκευής IoT .
- Οι αυτόνομες συσκευές χωρίζονται σε δύο μεγάλες κατηγορίες:
- **Single board computers** (SBC) - Παραδείγματα: Raspberry Pi, BeagleBone, Qualcomm DragonBoard, UDOO, κλπ.
- **Microcontroller units** (MCU) - Παραδείγματα: Arduino, Arduino-type, Espressif ESP, Particle Electron, κλπ.

συσκευές IoT υψηλών δυνατοτήτων



συσκευές IoT περιορισμένων δυνατοτήτων

Arduino due

Zolertia Re- Mote

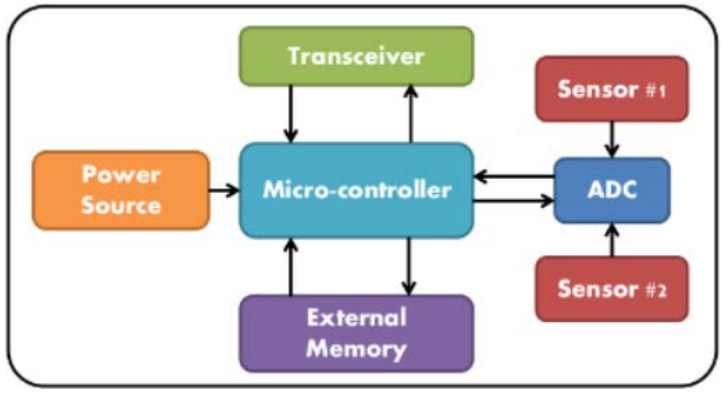
IoT- LAB- M3

Atmel SAM R21

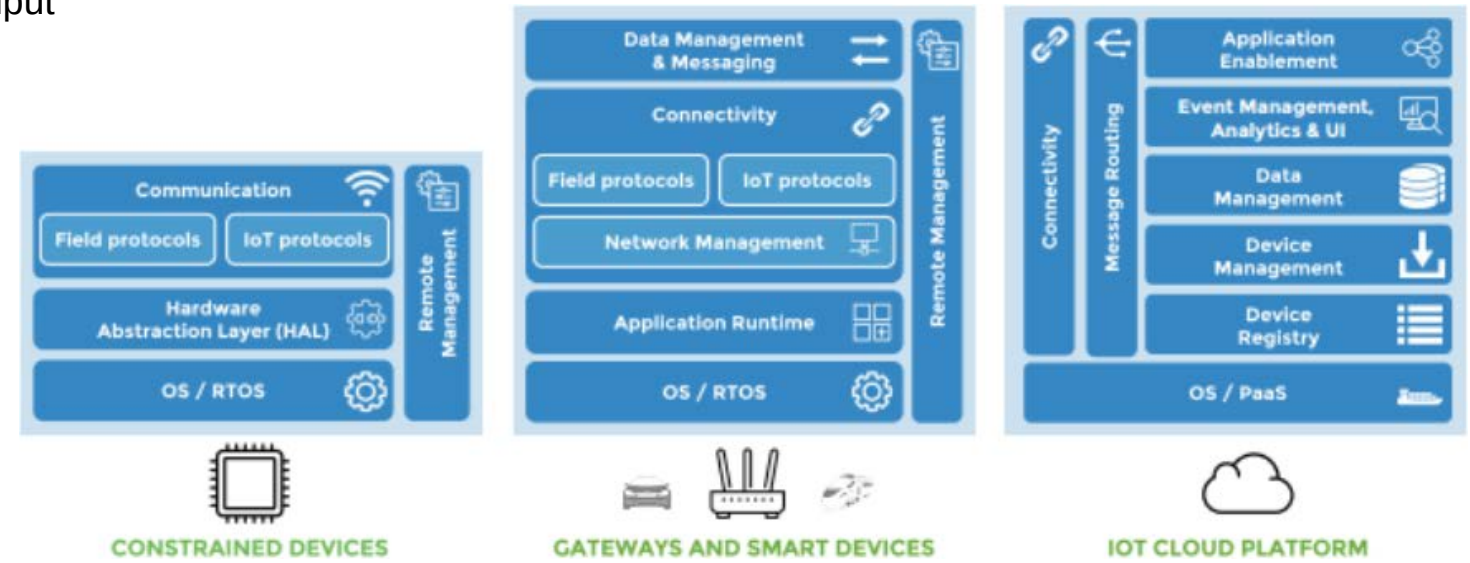
IoT κόμβοι περιορισμένων δυνατοτήτων

Οι περιορισμοί μπορεί να αφορούν:

- Πολυπλοκότητα κώδικα (ROM / Flash), μέγεθος κατάστασης και δεδομένων (RAM)
- Επεξεργαστική ισχύς
- Ασύμμετρα χαρακτηριστικά σύνδεσης
- Διαθέσιμη πηγή ισχύος
- Διεπαφή χρήστη
- Προσβασιμότητα κατά την ανάπτυξη
- Bitrate / Throughput
- Κόστος
- Φυσικό μέγεθος



Βασική αρχιτεκτονική IoT κόμβου περιορισμένων δυνατοτήτων



IoT κόμβοι περιορισμένων δυνατοτήτων

Constrained Nodes (IETF classification)

	Data (RAM)	Code (ROM)
Class 0 (Too constrained)	<< 10 KB	<< 100 KB
Class 1 (Quite constrained)	~ 10 KB	~ 100 KB
Class 2 (Not so constrained)	~ 50 KB	~ 250 KB

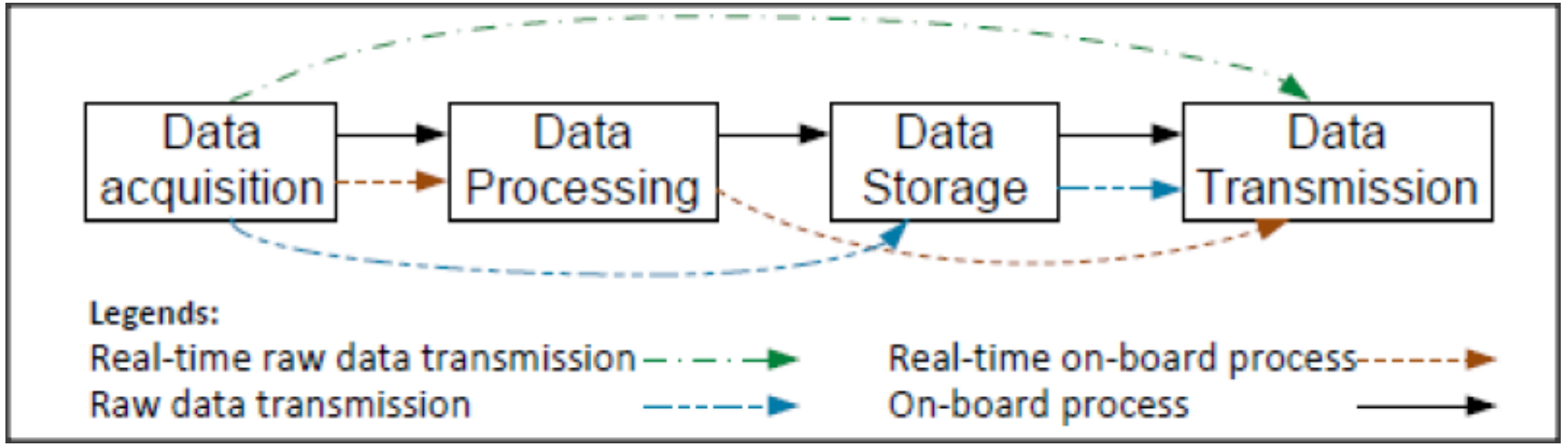
Π.χ. μια εξειδικευμένη συσκευή σε ένα WSN

Π.χ. IEEE802.15.4, BLE, 6LoWPAN, IPv6, CoAP, RPL, DTLS

Π.χ. τυπική στοίβα πρωτοκόλλων όπως HTTP, TLS, TCP

- Νέα λειτουργικά συστήματα από εταιρείες όπως Huawei, ARM και Google για συσκευές IoT
- Εύκολη παραγωγή και συντήρηση κώδικα με χρήση πρωτογενών λειτουργιών λογισμικού και APIs

Γενικά στάδια εφαρμογών IoT

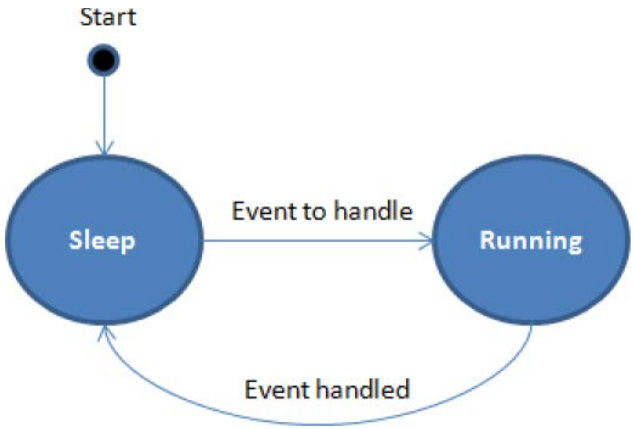


Πηγή: Samie, F., Bauer, L., & Henkel, J. (2016, October). IoT technologies for embedded computing: A survey. In 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS) (pp. 1-10). IEEE.

- Βασικοί πόροι IoT κόμβων με περιορισμούς:
  - υπολογιστική ισχύς
  - μέγεθος μνήμης
  - ενέργεια

Ο ρόλος ενός ΛΣ στο IoT

- Μπορεί να μειώσει τον **χρόνο διάθεσης στην αγορά** (**Time to Market**, **TTM**) μιας εφαρμογής IoT.
- **Διευκολύνει την ανάπτυξη εφαρμογών IoT** ως συλλογή ανεξάρτητων διεργασιών προσφέροντας κατάλληλους μηχανισμούς (π.χ. διαδιεργασιακής επικοινωνίας, συγχρονισμού, διαχείρισης event, κ.λπ.)
- Παρέχει **ευελιξία στην επιλογή υλικού**.
- Συμβάλει στη **μείωση της κατανάλωσης ενέργειας** λειτουργώντας ως σύστημα οδηγούμενο από γεγονότα (event-driven).
- Παρέχει έτοιμες **λύσεις συνδεσιμότητας** (π.χ. Wifi, TCP/IP, Bluetooth, LoRaWan)
- Άλλες **έτοιμες υπηρεσίες**: κρυπτογράφηση, αναβάθμιση OTA (Over The Air), σύστημα αρχείων, κ.ά.



Πιστοποίηση ασφάλειας

SAFERTOS®

Pre-emptive scheduling for:

IEC 61508	ISO 26262
IEC 62304	FDA 510(k)
EN 50128	DO-178B

Certified by TÜV SÜD to IEC 61508-SIL 3

ΑΠΑΙΤΗΣΕΙΣ ΛΣ ΓΙΑ ΣΥΣΚΕΥΕΣ IOT

Απαιτήσεις ΛΣ για συσκευές IoT

- **Μικρό αποτύπωμα μνήμης (RAM, ROM)**
  - Βελτιστοποίηση βιβλιοθηκών (cross-layer), αποδοτικές δομές δεδομένων
  - Ικανοποίηση αντικρουόμενων στόχων (απόδοση, βολικό API, μικρό αποτύπωμα μνήμης)
  - Καλές πρακτικές προγραμματισμού και δομικότητα vs. μικρό αποτύπωμα μνήμης
- **Υποστήριξη ετερογενούς υλικού**
  - Διάφορες αρχιτεκτονικές και οικογένειες MCU: 8-bit (π.χ. Intel 8051/52, Atmel AVR), 16-bit (π.χ. TI MSP430), 32-bit (ARM7, ARM Cortex-M , MIPS32, x86), …
  - Ανισορροπία RAM, ROM
  - Μεγάλη ποικιλία τεχνολογιών επικοινωνίας

Απαιτήσεις ΛΣ για συσκευές IoT

- **Συνδεσιμότητα δικτύου**
  - Μια συσκευή IoT διαθέτει μία (ή περισσότερες) διεπαφές δικτύου
  - Ευρεία γκάμα τεχνολογιών ασύρματης επικοινωνίας χαμηλής ισχύος (π.χ. IEEE 802.15.4, Bluetooth / BLE, DASH7 και EnOcean)
  - Διάφορες ενσύρματες τεχνολογίες (π.χ. PLC, Ethernet ή bus συστήματα).
  - Διασύνδεση στο Διαδίκτυο και end-to-end επικοινωνία με άλλες μηχανές.
  - Ενσωμάτωση στοιβών δικτύου βασισμένων σε πρωτόκολλα IP απευθείας σε συσκευές IoT
  - Πολλαπλές στοίβες δικτύου και συνεχή εξέλιξη της συνδεσιμότητας δικτύου (κατά τα πρότυπα του Linux)

Απαιτήσεις ΛΣ για συσκευές IoT

- **Ενεργειακή απόδοση**
  - Οι συσκευές IoT λειτουργούν με μπαταρίες (π.χ. έξυπνοι μετρητές και συσκευές αυτοματισμού)
  - Ενεργειακή απόδοση λόγω του μεγάλου αριθμού συσκευών IoT παγκοσμίως
  - MCU, ραδιοπομποί, αισθητήρες – έχουν τη δυνατότητα να λειτουργήσουν με ενεργειακά αποδοτικό τρόπο (sleep mode)
  - Παροχή επιλογών εξοικονόμησης ενέργειας προς τα ανώτερα στρώματα
  - Χρήση αυτών των λειτουργιών σε επίπεδο ΛΣ όσο το δυνατόν περισσότερο

Απαιτήσεις ΛΣ για συσκευές IoT

- **Δυνατότητες πραγματικού χρόνου**
  - Η έγκαιρη εκτέλεση εργασιών είναι ζωτικής σημασίας σε διάφορες εφαρμογές IoT π.χ. εφαρμογές υγείας με χρήση BAN, ρομπότ σε περιβάλλοντα βιομηχανικού αυτοματισμού, ad-hoc δίκτυο οχημάτων
  - Οι λειτουργίες του πυρήνα ενός RTOS πρέπει να εκτελούνται σε ντετερμινιστικό χρόνο
  - Παράδειγμα RTOS: ITRON για καταναλωτικές συσκευές

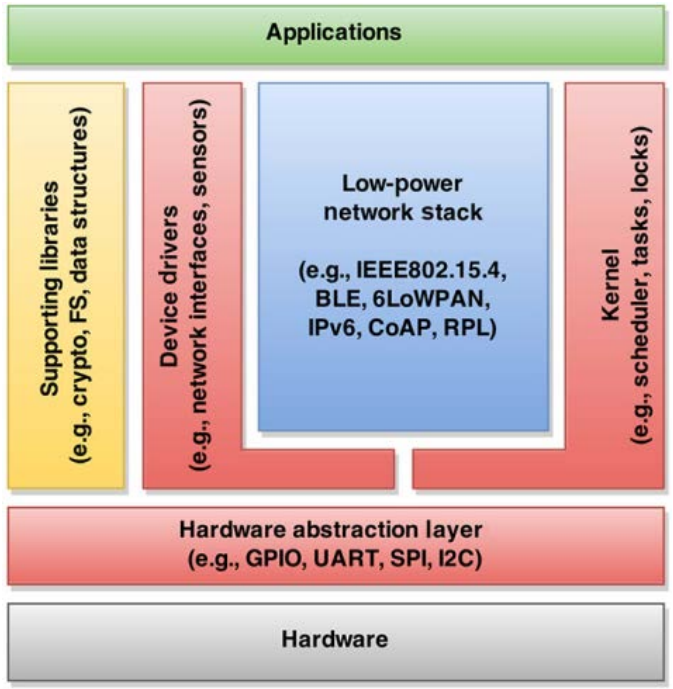
Απαιτήσεις ΛΣ για συσκευές IoT

- **Ασφάλεια**
  - Απαραίτητη για κρίσιμες υποδομές και συστήματα ελέγχου συνδεδεμένα με την ανθρώπινη ζωή
  - Προκλήσεις: ακεραιότητα δεδομένων, έλεγχος ταυτότητας και έλεγχος πρόσβασης σε μέρη της αρχιτεκτονικής IoT
  - Μηχανισμοί: βιβλιοθήκες κρυπτογράφησης και πρωτόκολλα ασφαλείας
  - Ενημερώσεις λογισμικού σε συσκευές IoT που είναι ήδη σε λειτουργία
  - Χρήση λύσεων ανοικτού κώδικα

ΣΧΕΔΙΑΣΤΙΚΕΣ ΕΠΙΛΟΓΕΣ

Σχεδιαστικές επιλογές (1/10)

- **Γενική αρχιτεκτονική και δομικότητα**



Επιλογή μεταξύ του πιο στιβαρού και πιο ευέλικτου μικροπυρήνα ή ενός λιγότερο περίπλοκου και πιο αποτελεσματικού μονολιθικού πυρήνα ή μια υβριδική προσέγγιση

- Exokernel
  - λιγότερες αφαιρέσεις μεταξύ της εφαρμογής και του υλικού
  - αποφυγή συγκρούσεων πόρων
  - έλεγχος επιπέδων πρόσβασης
- Microkernel
  - περισσότερες λειτουργίες στον πυρήνα
  - μινιμαλιστικό σύνολο χαρακτηριστικών
  - λίγη μνήμη, ευελιξία, ευρωστία
- Monolithic
  - όλα τα συστατικά του συστήματος μαζί
  - απλούστερος και πιο αποδοτικός σχεδιασμός
- Hybrid

Σχεδιαστικές επιλογές (2/10)

- **Μοντέλο χρονοδρομολογητή**
  - preemptive schedulers
    - διακόπτει οποιαδήποτε διεργασία (όχι του πυρήνα) οποτεδήποτε για να δρομολογήσει άλλη διεργασία προς εκτέλεση για περιορισμένο χρονικό διάστημα
    - περιοδικό χρονόμετρο, systick → εμποδίζει τη συσκευή IoT να εισέλθει σε βαθύτερη λειτουργία εξοικονόμησης ενέργειας
  - nonpreemptive (or cooperative) schedulers
    - κάθε νήμα είναι υπεύθυνο να παραδώσει τη CPU (δεν μπορεί να διακοπεί από άλλο)

Ένας preemptive scheduler εκχωρεί χρόνο CPU σε κάθε διεργασία, ενώ στο μοντέλο συνεργασίας οι διεργασίες θα πρέπει να παραδώσουν τη CPU από μόνες τους.

Σχεδιαστικές επιλογές (3/10)

- **Κατανομή μνήμης**
  - Η έλλειψη του πόρου απαιτεί εξελιγμένες τεχνικές
  - Στατική κατανομή μνήμης → σύστημα λιγότερο ευέλικτο στις μεταβαλλόμενες απαιτήσεις κατά το χρόνο εκτέλεσης
  - Δυναμική κατανομή μνήμης → περίπλοκος σχεδιασμός συστήματος
    - Υλοποίηση malloc() με μη-ντετερμινιστικό τρόπο, δεν εγγυάται απαιτήσεις πραγματικού χρόνου → ειδικές υλοποιήσεις για ντετερμινιστικές malloc() όπως το TLSF
    - Ανάγκη χειρισμού καταστάσεων out-of-memory
    - Υλοποίηση malloc() σε δομή heap → θέματα κατακερματισμού της μνήμης

Η στατική κατανομή μνήμης εισάγει κάποια επιβάρυνση στη μνήμη και οδηγεί σε λιγότερο ευέλικτα συστήματα, ενώ η δυναμική κατανομή μνήμης οδηγεί σε ένα πιο περίπλοκο σύστημα το οποίο μπορεί να έρχεται σε σύγκρουση με απαιτήσεις πραγματικού χρόνου.

Σχεδιαστικές επιλογές (4/10)

- **Διαχείριση buffer δικτύου**
  - κομμάτια μνήμης, όπως τα πακέτα, πρέπει να διαμοιρασθούν μεταξύ των επιπέδων
  - αντιγραφή μνήμης [memcpy ()] → δαπανηρό από πλευράς πόρων
  - πέρασμα δεικτών μεταξύ των διαφόρων επιπέδων → υπευθυνότητα για την κατανομή μνήμης
    - ανάθεση στα ανώτερα επίπεδα → ανάπτυξη εφαρμογών πιο περίπλοκη και λιγότερο βολική
    - ανάθεση στα κάτω επίπεδα → σύστημα λιγότερο ευέλικτο
    - σχεδιασμός κεντρικού διαχειριστή μνήμης π.χ. TinyOS ή RIOT

Η μνήμη για το χειρισμό πακέτων στη στοίβα δικτύου μπορεί να εκχωρηθεί από κάθε επίπεδο ή να περάσει ως αναφορά μεταξύ των επιπέδων.

Σχεδιαστικές επιλογές (5/10)

- **Μοντέλο προγραμματισμού**
  - καθορίζει πώς ένας προγραμματιστής εφαρμογών μπορεί να αναπτύξει το πρόγραμμα
  - event-driven systems
    - διαδεδομένο μοντέλο σε WSN
    - κάθε εργασία ενεργοποιείται από ένα (εξωτερικό) συμβάν
    - περιλαμβάνει ένα απλό event loop και ένα μοντέλο κοινόχρηστης στοίβας
  - multithreaded systems:
    - κάθε εργασία εκτελείται στο δικό της νήμα
    - επικοινωνία μεταξύ των εργασιών χρησιμοποιώντας ένα API επικοινωνίας μεταξύ διεργασιών (IPC)

Τα event-driven συστήματα μπορεί να είναι πιο αποδοτικά από πλευράς μνήμης ενώ τα multithreading συστήματα διευκολύνουν το σχεδιασμό των εφαρμογών.

Σχεδιαστικές επιλογές (6/10)

- **Γλώσσες προγραμματισμού**
  - Δυο βασικές επιλογές:
    1. μια τυπική γλώσσα προγραμματισμού, ANSI C ή C++
    2. μια γλώσσα ή διάλεκτος εξειδικευμένη για συγκεκριμένο ΛΣ (π.χ. η γλώσσα Swift για το ΛΣ iOS, η γλώσσα NesC για το ΛΣ TinyOS)

Η χρήση τυπικών γλωσσών προγραμματισμού διευκολύνει τη φορητότητα και επιτρέπει τη χρήση γνωστών εργαλείων ανάπτυξης. Μια γλώσσα εξειδικευμένη για συγκεκριμένο ΛΣ μπορεί να αυξήσει την απόδοση και την ασφάλεια του συστήματος.

Σχεδιαστικές επιλογές (7/10)

- **Μοντέλο Driver και Hardware Abstraction Layer**
  - Μια συσκευή IoT συνδέεται με διάφορους sensors/actuators
  - Υλικό με πολλές διεπαφές διασύνδεσης: ADCs/DACs, SPI, I^2C, CAN bus, serial lines, GPIOs
  - Ορισμός μιας ευέλικτης και απλής διεπαφής για τα προγράμματα οδήγησης
  - Hardware Abstraction Layer: παρέχει μια καλά καθορισμένη διεπαφή για υλικό όπως CPU, μνήμη και χειρισμό διακοπών με κύριο όφελος την εύκολη μεταφορά του ΛΣ σε νέες πλατφόρμες υλικού

Ένα καλά σχεδιασμένο επίπεδο αφαίρεσης υλικού και ένα μοντέλο προγράμματος οδήγησης μπορούν να βελτιώσουν σημαντικά το σχεδιασμό του συστήματος, αλλά εισάγουν και κάποια επιβάρυνση - είτε επιπλέον γραμμών κώδικα είτε επιπλέον χρόνου εκτέλεσης.

Σχεδιαστικές επιλογές (8/10)

- **Εργαλεία αποσφαλμάτωσης**
  - Τυπικές αλυσίδες εργαλείων όπως η συλλογή μεταγλωττιστών GNU (GCC) περιλαμβάνουν debugging εργαλεία όπως ο GNU Debugger (GDB).
  - Απαιτείται βοήθεια από το υλικό της συσκευής IoT μέσω κατάλληλων διεπαφών, όπως JTAG και Spy-Bi-Wire
  - Άλλα μέσα όταν δεν παρέχεται τέτοια διασύνδεση:
    - χρήση εντολής printf() για επικοινωνία μέσω μιας σειριακής διεπαφής, π.χ. USART
    - απλός αλγόριθμος που αναβοσβήνει ένα LED
    - περιοδικά διαγνωστικά μηνύματα σε δίκτυα IoT
    - αρχεία log σε εξωτερική flash memory

Η χρήση τυπικών γλωσσών προγραμματισμού επιτρέπει τη χρήση διαθέσιμων debugging εργαλείων, αλλά περιορισμοί στο υλικό ενδέχεται να δημιουργήσουν την ανάγκη για χρήση εναλλακτικών μέσων όπως ένα LED που αναβοσβήνει.

Σχεδιαστικές επιλογές (9/10)

- **Σύνολο χαρακτηριστικών**
  - Διαχωρισμός υπηρεσιών ΛΣ σε λειτουργίες πυρήνα και λειτουργίες υψηλότερου επιπέδου.
  - Στον πυρήνα ανήκουν: scheduler, μοντέλο διεργασιών, μηχανισμός αμοιβαίου αποκλεισμού (mutex), άλλοι μηχανισμοί συγχρονισμού, timers, IPC για multithreading.
  - Υψηλότερα επίπεδα: βιβλιοθήκες συστήματος, shell, logging, κρυπτογραφικές συναρτήσεις, στοίβες δικτύου
  - Add-ons: ενημερώσεις over-the-air, δυναμική φόρτωση και σύνδεση βιβλιοθηκών, lightweight κρυπτογράφηση και αποκρυπτογράφηση

Το τελικό σύνολο χαρακτηριστικών ενός Λειτουργικού Συστήματος για IoT μπορεί να προσδιοριστεί από το μέγεθος του API του.

Σχεδιαστικές επιλογές (10/10)

- **Έλεγχος**
  - Ο έλεγχος και η δοκιμή παίζουν καθοριστικό ρόλο στην επιτυχή ανάπτυξη λειτουργικών συστημάτων IoT
  - Χρειάζεται η εγκατάσταση ενός περιβάλλοντος συνεχούς ολοκλήρωσης (continuous integration)
    - ελέγχους κατασκευής και ολοκλήρωσης συστημάτων και δοκιμές συγκεκριμένων μονάδων
  - Προκλήσεις: κατανεμημένη φύση, βαθιά ενσωμάτωση, περιορισμένοι πόροι
  - Εργαλεία εξομοίωσης υλικού (MSPSim, Emul8)
  - Εξομοιωτές και προσομοιωτές δικτύου (Cooja, ns-2/3)

Η κατανεμημένη φύση και οι περιορισμοί του υλικού καθιστούν τον ενδελεχή έλεγχο ενός ΛΣ για το IoT μια δύσκολη, αλλά κρίσιμη εργασία.

ΜΗ ΤΕΧΝΙΚΑ ΘΕΜΑΤΑ

Μη τεχνικά θέματα

- Ανοικτά Πρότυπα
- Πιστοποίηση
- Τεκμηρίωση
- Ωριμότητα κώδικα
- Άδεια χρήσης
- Διανομή

Ανοικτά Πρότυπα

- Ένα λειτουργικό σύστημα πρέπει να υποστηρίζει τη φορητότητα των εφαρμογών σε διάφορες πλατφόρμες υλικού και αρχιτεκτονικές
- Τυποποιημένα APIs (όπως το POSIX) συμβάλουν στην απλοποίηση της μεταφοράς λογισμικού μεταξύ πολλών λειτουργικών συστημάτων
  - Προσδιορίζουν έναν καθορισμένο τρόπο για μια εφαρμογή να αλληλεπιδρά με το λειτουργικό σύστημα π.χ. σε σχέση με λειτουργίες αρχείων, διαχείριση εργασιών, σήματα και οδηγούς συσκευών
- Η υλοποίηση ενός στάνταρντ API που έχει σχεδιαστεί για ΛΣ γενικού σκοπού όπως το Linux σε συσκευές IoT είναι δύσκολη
- Η φορητότητα μπορεί να ενισχυθεί με υιοθέτηση προτύπων γλωσσών προγραμματισμού όπως ANSI C99 ή C++11
- Τα πρότυπα είναι εξίσου σημαντικά σε επίπεδο δικτύου (π.χ. πρότυπα IETF)

Η χρήση προτύπων βελτιώνει τη φορητότητα και διαλειτουργικότητα.

Πιστοποίηση

- Εφαρμογές με κρίσιμο χαρακτήρα χρειάζονται λειτουργία πραγματικού χρόνου, ανθεκτικότητα και ντετερμινισμό (ιδιότητες που πρέπει να επιδεικνύει το ΛΣ)
- Απαιτείται η πιστοποίηση του ΛΣ μέσω ανεξάρτητων οργανισμών
  - πρότυπο IEC 61508 «Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related System».
  - IoT «IPv6 Ready» του IPv6 Forum
  - πρόγραμμα συμμόρφωσης και πιστοποίησης IPSO Alliance

Η εγκατάσταση κρίσιμων εφαρμογών με έμφαση στην ασφάλεια μπορεί να απαιτεί την πιστοποίηση ολόκληρου του λογισμικού που λειτουργεί σε ένα σύστημα IoT.

Τεκμηρίωση

- Κάθε λογισμικό χρειάζεται μια καλή τεκμηρίωση
- Αυτό ισχύει ακόμη περισσότερο για ένα ΛΣ ως θεμέλιο κάθε άλλου λογισμικού που εκτελείται σε ένα σύστημα IoT
- Χρήσιμο να φωτιστούν τα ενδότερα του ενσωματωμένου λογισμικού, περιορισμοί, συμβιβασμοί
- Καλός δείκτης τεκμηρίωσης: ποσοστό της τεκμηρίωσης ανά γραμμή κώδικα

Η καλύτερη χρήση ενός λειτουργικού συστήματος και η αποδοτικότερη ανάπτυξη εφαρμογών προϋποθέτουν μια πλήρη και κατανοητή τεκμηρίωση.

Ωριμότητα κώδικα

- Η ωριμότητα του λογισμικού είναι δύσκολο να μετρηθεί
- Η πιστοποίηση ως μια ένδειξη ποιότητας, αλλά η πραγματική ευρωστία και ορθότητα ενός συστήματος είναι διαφορετική ιστορία
- Προσεγγιστικοί δείκτες: η ηλικία του έργου, ο αριθμός των contributors και των χρηστών

Η ενδελεχής δοκιμή και η ευρεία χρήση σε εμπορικές εφαρμογές είναι ένας καλύτερος δείκτης για την ωριμότητα ενός ΛΣ από την ηλικία του project ή των πιστοποιήσεων.

Άδεια χρήσης

- Βασικές κατηγορίες αδειών:
  - Nonfree
    - ΛΣ διαθέσιμο μόνο ως δυαδικά δεδομένα ή επιπλέον κόστος για τη λήψη του πηγαίου κώδικα
  - Permissive open source (BSD, MIT ή Apache License)
    - παρέχουν υψηλό βαθμό ελευθερίας
    - πιθανός κατακερματισμός της κοινότητας και της βάσης κώδικα
  - Copyleft (GPL και LGPL)
    - πιο δύσκολα αποδεκτές από ορισμένους κλάδους της βιομηχανίας
    - πιο ολοκληρωμένη κοινότητα και κοινή βάση κώδικα (π.χ. Linux)

Οι άδειες ανοικτού κώδικα - ιδιαίτερα οι copyleft - μπορεί να μην είναι πάντα η πρώτη επιλογή της βιομηχανίας, αλλά έχουν καλές πιθανότητες για δημιουργία κώδικα υψηλότερης ποιότητας και ασφάλειας λόγω του αυξημένου αριθμού συντελεστών και κριτικών.

Διανομή

- Ανάλογα με τον επιλεγμένο τύπο άδειας γίνεται η διανομή και η υποστήριξη του ΛΣ
- Για άδειες ανοικτού κώδικα, το ΛΣ παρέχεται μέσω αποθετηρίων κώδικα όπως Git, Subversion και Mercurial
- Η κοινότητα παρέχει υποστήριξη τύπου best-effort μέσω online forums, open issue trackers, και mailing lists
- Παροχή επαγγελματικών συμβουλών για το λογισμικό και για τα ΛΣ ανοικτού κώδικα

Ο τρόπος διανομής και ο βαθμός υποστήριξης για ένα λειτουργικό σύστημα εξαρτάται σε μεγάλο βαθμό από την άδειά του.

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ IOT

Υποψήφια ΛΣ για το IoT

Κατηγορίες:

- ΛΣ IoT ανοικτού κώδικα
- ΛΣ IoT κλειστού κώδικα
- Άλλες βιβλιοθήκες λογισμικού ή ενδιάμεσο λογισμικό για το IoT.

ΛΣ IOT ανοικτού κώδικα

Contiki

- Αρχική εστίαση σε WSNs και συσκευές memory-constrained 8-bit MCUs
- Τώρα σε 16-bit MCUs και IoT devices based on the ARM 32-bit MCUs
- Event-driven
- Cooperative scheduling approach
- Support for lightweight pseudo-threading
- Υλοποίηση σε C programming language, ορισμένα τμήματα κάνουν χρήση macro-based abstractions (π.χ. Protothreads)
- Layered architecture
- Στοίβες δικτύου
  - uIP, με υποστήριξη για IPv6, 6LoWPAN, RPL και CoAP
  - Rime stack, το οποίο παρέχει ένα σύνολο αφαιρέσεων κατανεμημένου προγραμματισμού

Developer	Adam Dunkels
Working state	Current
Source model	Open source
Initial release	10 March 2003;
Latest release	3.0 / 26 August 2015;
License	BSD
Official website	www.contiki-os.org

RIOT

- Tickless scheduler (Αλλάζει στο idle thread σε κατάσταση idle)
- Σχεδιασμένο με τέτοιο τρόπο ώστε οι λειτουργίες του πυρήνα να δρομολογούνται με χαμηλή ταχύτητα ρολογιού
- Full support for RPL, 6LoWPAN, IPv6, TCP, UDP, etc.
- Support multi threading and Real Time
  - through POSIX
  - Zero latency interrupt handlers
  - minimum context switching times
  - The kernel will never crash because of an error prone device driver

RIOT – The friendly OS for the IoT

OS family	Embedded operating systems
Working state	Current
Source model	Open source
Latest release	2018.04[1] / 11 May 2018; 2 years ago
Repository	github.com/RIOT-OS/RIOT
Platforms	TI MSP430, ARM7, ARM Cortex-M0-M0+-M3-M4, Atmel AVR, MIPS32r2, RISC-V
Kernel type	Microkernel
License	LGPLv2
Official website	riot-os.org

FreeRTOS

- Δημοφιλές RTOS που έχει γίνει port σε πολλές πλατφόρμες
- Preemptive scheduler
- Support for multithreading
- Άδεια MIT, επιτρέπει εμπορική χρήση με εφαρμογές κλειστού κώδικα πηγής
- Χρησιμοποιεί στοίβες δικτύου τρίτων για σύνδεση στο Διαδίκτυο

Developer	Real Time Engineers Ltd.
OS family	Real-time operating systems
Working state	Current
Source model	Open source
Latest release	10.3.1[1] / 2020-02-19
Repository	github.com/FreeRTOS/FreeRTOS
Marketing target	Embedded devices
Platforms	ARM (ARM7, ARM9, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-A), Atmel AVR, AVR32, HCS12, MicroBlaze, Cortus (APS1, APS3, APS3R, APS5, FPF3, FPS6, FPS8), MSP430, PIC, Renesas H8/S, SuperH, RX, x86, 8052, Coldfire, V850, 78K0R, Fujitsu MB91460 series, Fujitsu MB96340 series, Nios II, Cortex-R4, TMS570, RM4x, Espressif ESP32, RISC-V
Kernel type	Microkernel
License	MIT[2]
Official website	www.freertos.org

TinyOS

- Monolithic Kernel
- Υλοποίηση με τη γλώσσα προγραμματισμού nesC
- Χρησιμοποιεί ειδικές γλωσσικές δομές και αφαιρέσεις προγραμματισμού για λιγότερα bugs και βελτίωση της αποδοτικότητας της μνήμης
- Η στοίβα δικτύου BLIP υλοποιεί τη στοίβα 6LoWPAN
- Ο περίπλοκος σχεδιασμός και η customized γλώσσα προγραμματισμού nesC καθιστά δύσκολη την εκμάθηση, και έτσι δεν διαθέτει μια μεγαλύτερη κοινότητα προγραμματιστών
- Δεν υπάρχει διαθέσιμη υποστήριξη τώρα, δεν υπάρχει υποστήριξη για τις πιο πρόσφατες συσκευές IoT

OpenWSN

- Περιλαμβάνει μια στοίβα δικτύου 6TiSCH, έναν βασικό scheduler και ένα board support package (BSP)
- Προσφέρεται σε πολλές πλατφόρμες υλικού IoT
- Ο κώδικας του OpenWSN είναι διαθέσιμος στο GitHub με άδεια BSD.
- Το κύριο σημείο του OpenWSN είναι η στοίβα δικτύου 6TiSCH, συμπεριλαμβανομένης μιας υλοποίησης του MAC IEEE 802.15.4.
- Το OpenWSN αναπτύσσεται από το 2010, από μια εξελισσόμενη, παγκόσμια κοινότητα ανοικτού κώδικα.

nuttX

- Standards compliant, Modular design, Fully preemptible
- Easily extensible to new processor architectures, SoC architecture, or board architectures
- Scheduling: first in, first out (FIFO) and round-robin (RR)
- Tickless operation
- POSIX/ANSI-like task controls, named message queues, counting semaphores, clocks/timers, signals, POSIX Threads (pthreads), environment variables
- Multiple file systems
- Loadable kernel modules
- Memory allocators: (1) standard heap memory allocation, (2) granule allocator, (3) shared memory, and (4) dynamically sized, per-process heaps
- Multiple network interface support; multiple network link layer support
- IPv4, IPv6,Internet protocol suite (TCP/IP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), Internet Group Management Protocol (IGMP) version 2 (client) stacks

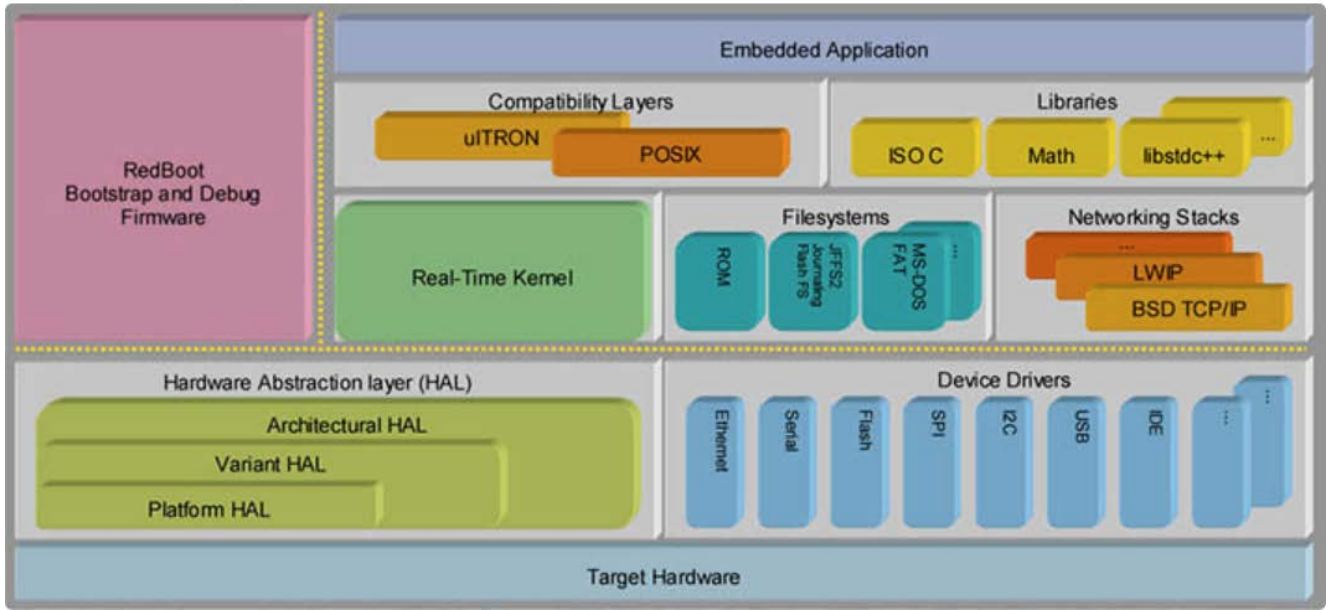
NuttX logo

Developer	Gregory Nutt
Written in	C, C++, assembly
OS family	Real-time operating systems
Working state	Current
Source model	Open source
Initial release	2007; 13 years ago
Latest release	9.1 / July 23, 2020; 3 months ago [1]
Marketing target	Embedded systems
Platforms	ARM, AVR, AVR32, HCS12, LM32, MIPS, RISC-V, SuperH, Xtensa XL6, x86, x86-64, Z80
Kernel type	Real-Time Microkernel
License	Apache License 2.0
Official website	nuttx.apache.org

eCos

eCosPro v4.1
eCosCentric delivers the new v4.1 iteration of the eCosPro Developer's Kit. The release incorporates the latest Eclipse Neon IDE, improvements to the eCosPro Eclipse plug-in and development tools, a variety of run-time enhancements, and integration of the Raspberry Pi as a standard common platform.

eCosPro for Raspberry Pi
eCosCentric ports eCosPro to Raspberry Pi. Delivers deterministic, real-time performance on the Raspberry Pi 3, Pi 2, Pi 1, Pi Zero and Pi Zero Wireless boards, as well as the Pi Compute Modules 1 and 3.

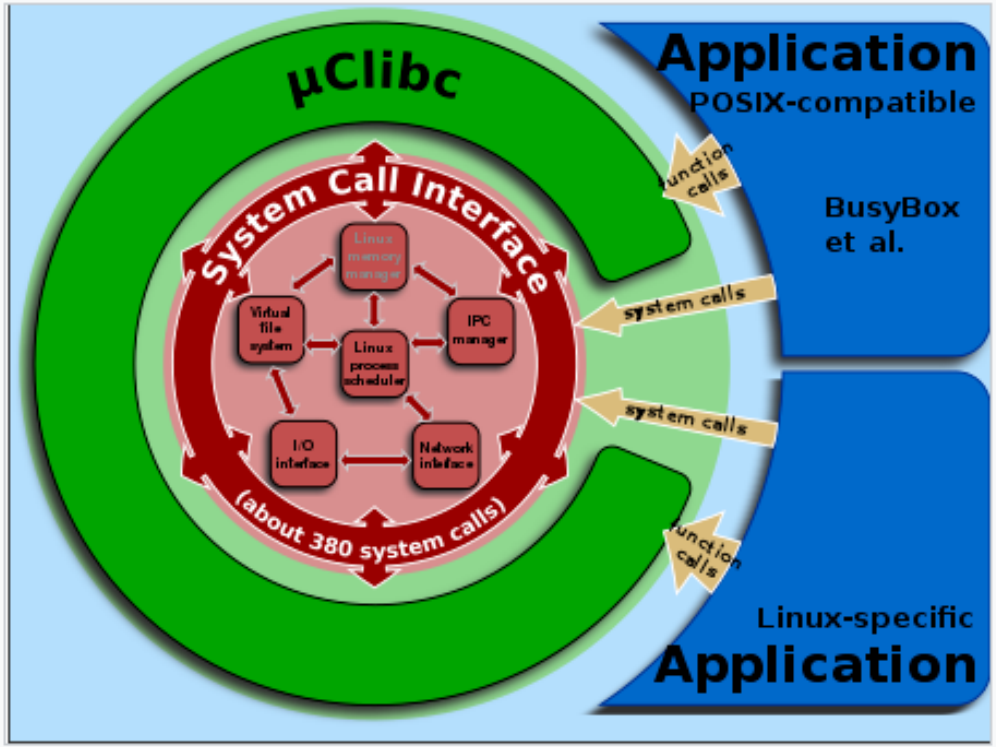


Developer	eCos community, Free Software Foundation
Written in	C, C++, assembly
OS family	Real-time operating systems
Working state	Current
Source model	Open source
Initial release	September 1998; 22 years ago
Latest release	3.0 / March 2009; 11 years ago
Marketing target	Embedded systems
Platforms	ARM (Cortex-A5, Cortex-A7, Cortex-A9, Cortex-A53, Cortex-M3, Cortex-M4, Cortex-M7), CalmRISC, FR-V, Hitachi H8, IA-32, Motorola 68000, Matsushita AM3x, MIPS, NEC V8xx, Nios II, PowerPC, SPARC, and SuperH
License	eCos License: GNU General Public License (with linking exception)[1]
Official website	ecos.sourceware.org

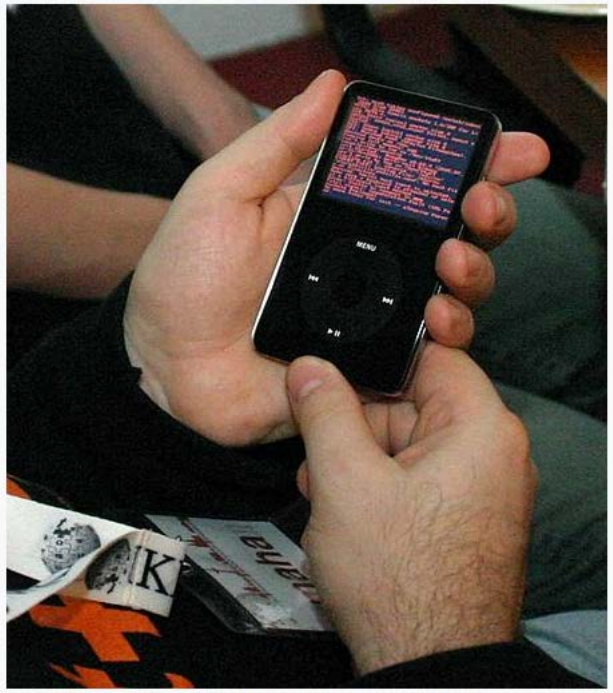
mbedOS

Developer	Collaborative project managed by Arm
Written in	C, C++
Working state	Current
Source model	Open-source
Initial release	September 21, 2009
Repository	github.com/ARMmbed
Marketing target	Microcontrollers, Internet of Things, Wearables
Platforms	32-bit ARM Cortex-M
License	Apache License 2.0
Official website	mbed.com

uClinux



uClibc is a wrapper around the system calls of the Linux kernel and/or µClinux.

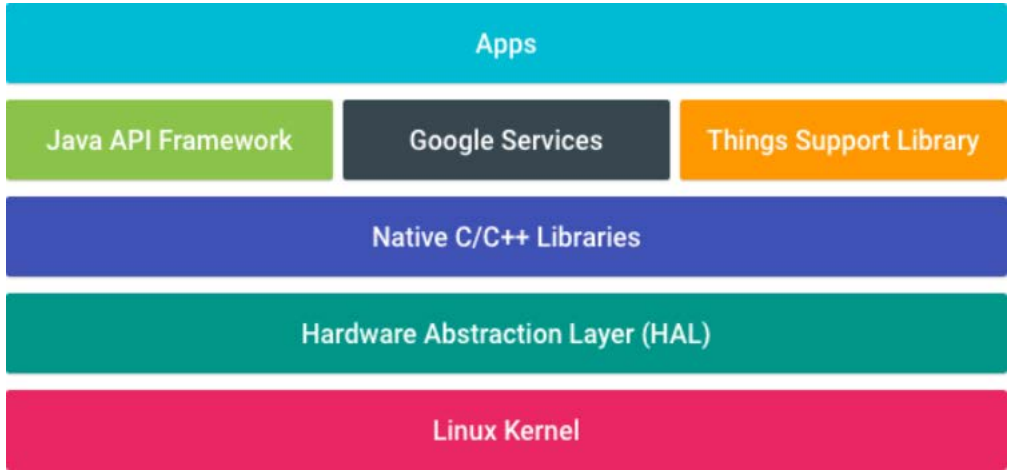


An iPod booting iPodLinux, based on µClinux

OS family	Embedded Linux
Working state	Current
Source model	Open source
Platforms	See below
Kernel type	Linux kernel-fork
Userland	uClinux-dist, uClibc, BusyBox
Official website	uclinux.org at the Wayback Machine (archived 2018-11-13)

Android Things (Brillo)

- Develop using the Android SDK and Android Studio
- Access hardware such as displays and cameras natively through the Android framework
- Connect your apps with Google services
- Integrate additional peripherals through the Peripheral I/O APIs (GPIO, I2C, SPI, UART, PWM)
- Use the Android Things Console to push over-the-air feature and security updates



androidthings

Official website developer.android.com/things/index.html

Επισκόπηση ΛΣ IoT ανοικτού κώδικα

Name	Architecture	Scheduler	Programming model	Targeted device class[a]	Supported MCU families or vendors	Programming languages	License	Network stacks
Contiki	Monolithic	Cooperative	Event-driven, Protothreads	Class 0 + 1	AVR, MSP430, ARM7, ARM Cortex-M, PIC32, 6502	C[b]	BSD	uIP, RIME
RIOT	Microkernel RTOS	Preemptive, tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM Cortex-M, x86	C, C++	LGPLv2	gnrc, OpenWSN, ccn-lite
FreeRTOS	Microkernel RTOS	Preemptive, optional tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM, x86, 8052, Renesas[c]	C	modified GPL[d]	None
TinyOS	Monolithic	Cooperative	Event-driven	Class 0	AVR, MSP430, px27ax	nesC	BSD	BLIP
OpenWSN	Monolithic	Cooperative[e]	Event-driven	Class 0 – 2	MSP430, ARM Cortex-M	C	BSD	OpenWSN
nuttX	Monolithic or microkernel	Preemptive (priority-based or round robin)	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM9, ARM Cortex-M, MIPS32, x86, 8052, Renesas	C	BSD	native
eCos	Monolithic RTOS	Preemptive	Multithreading	Class 1 + 2	ARM, IA-32, Motorola, MIPS ...	C	eCos License[f]	lwIP, BSD
uClinux	Monolithic	Preemptive	Multithreading	>Class 2	Motorola, ARM7, ARM Cortex-M, Atari	C	GPLv2	Linux
ChibiOS/RT	Microkernel	Preemptive	Multithreading	Class 1 + 2	AVR, MSP430, ARM Cortex-M	C	Triple License[g]	None
CoOS	Microkernel RTOS	Preemptive	Multithreading	Class 2	ARM Cortex-M	C	BSD	None
nanoRK	Monlothic (resource kernel)	Preemptive	Multithreading	Class 0	AVR, MSP430,	C	Dual License	None
Nut/OS	Monolithic	Cooperative	Multithreading	Class 0 + 1	AVR, ARM	C	BSD	native

[a] According to [8]: Class 0 devices have <<10 kB RAM and <<100 kB ROM, Class 1 devices have ∼10 kB and ∼100 kB ROM, Class 2 devices have ∼10 kB and ∼100 kB ROM.
[b] with some restrictions
[c] and several other MCUs
[d] added an exception to allow commercial use
[e] only a rudimentary scheduler is provided, support for RIOT and FreeRTOS scheduler
[f] GPL with linking exception

Πηγή: Hahm, O., Baccelli, E., Petersen, H., & Tsiftes, N. (2015). Operating systems for low-end devices in the internet of things: a survey. IEEE Internet of Things Journal, 3(5), 720-734.

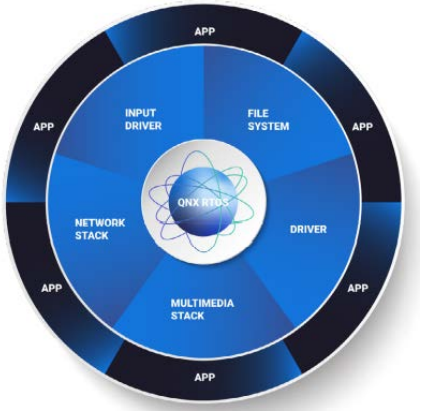
ΛΣ IOT κλειστού κώδικα

ThreadX

- priority-based, preemptive scheduling
  - preemption-threshold
  - priority inheritance
- fast interrupt response
- memory management
- interthread communication
- mutual exclusion
- event notification &event-chaining
- thread synchronization
- efficient timer management & fast software timers
- picokernel design
- small size: minimal size on an ARM architecture processor is about 2 KB

Developer	Microsoft (originally Express Logic)
Written in	C
OS family	Real-time operating system (RTOS)
Working state	Current
Source model	Source-available software
Initial release	1997; 23 years ago
Latest release	v6.1.1_rel[1] / October 16, 2020; 25 days ago
Repository	github.com/azure-rtos/threadx/
Marketing target	Embedded systems, IoT: including sensors, devices, edge routers, gateways
Update method	Re-installation
Package manager	None
Platforms	ARC, ARM, Blackfin, CEVA, C6x, MIPS, NXP, PIC, PowerPC, RISC-V, RX, SH, SHARC, TI, V850, Xtensa, x86, Coldfire, others
Kernel type	Embedded, deterministic, real-time microkernel, picokernel
Default user interface	Embedded UI support (GUIX)
License	Proprietary
Official website	azure.microsoft.com/en-us/services/rtos/

QNX



Microkernel Reliability

With the QNX microkernel architecture, a component failure doesn't bring down other components or the kernel. The failed component is simply shut down and restarted without adversely affecting the rest of the system.

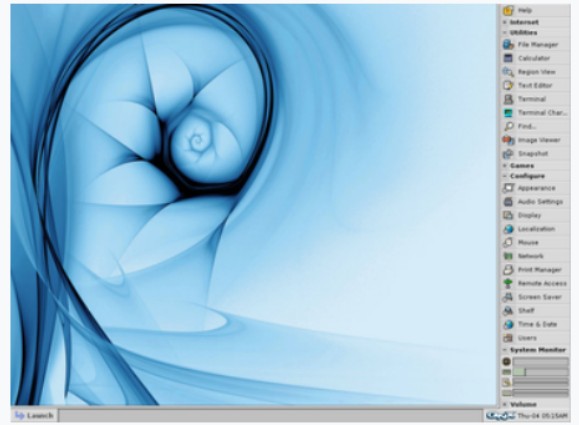
Real-Time Availability

The QNX Neutrino RTOS offers the determinism only a real-time OS can provide. Techniques such as adaptive partitioning guarantee critical processes get the cycles they need to complete their tasks on time, while maintaining the performance your complex embedded systems require.

Comprehensive, Layered Security

With the QNX Neutrino RTOS's layered security features and access to QNX security experts, you can take a holisitic approach to security and implement exactly the security profiles your embedded systems require.

QNX | BlackBerry Subsidiary



The default desktop in QNX 6.4.1

Developer	BlackBerry
OS family	Unix-like
Working state	Current
Source model	Closed source
Initial release	1982; 38 years ago
Latest release	7.1 / July 2020; 4 months ago
Marketing target	Embedded systems
Package manager	Able to use Pkgsrc framework from NetBSD project
Platforms	x86, MIPS, PowerPC, SH-4, ARM, StrongARM, XScale
Kernel type	RTOS (microkernel)
License	Proprietary
Official website	www.qnx.com

VxWorks

- Multitasking kernel with preemptive and round-robin scheduling and fast interrupt response
- Native 64-bit operating system (only one 64-bit architecture supported: x86-64).
- User-mode applications ("Real-Time Processes", or RTP) isolated from other user-mode applications as well as the kernel via memory protection mechanisms.
- Bluetooth, USB, CAN protocols, Firewire IEEE 1394, BLE, L2CAP, Continua stack, health device profile
- Binary, counting, and mutual exclusion semaphores with priority inheritance
- Local and distributed message queues
- File systems: High Reliability File System (HRFS), FAT-based file system (DOSFS), Network File System (NFS), and TFFS
- Dual-mode IPv6 networking stack with IPv6 Ready Logo certification
- Memory protection including real-time processes (RTPs), error detection and reporting, and IPC
- Symbolic debugging

Developer	Wind River (a wholly owned subsidiary of TPG Capital)
OS family	Real-time operating systems
Working state	Current
Initial release	1987; 33 years ago
Latest release	7 / March 2014; 6 years ago
Marketing target	Embedded systems
Platforms	x86 (including Intel Quark), x86-64, MIPS, PowerPC, SH-4, ARM
Kernel type	Monolithic
License	Proprietary
Official website	Windriver.com/products/vxworks

embOS

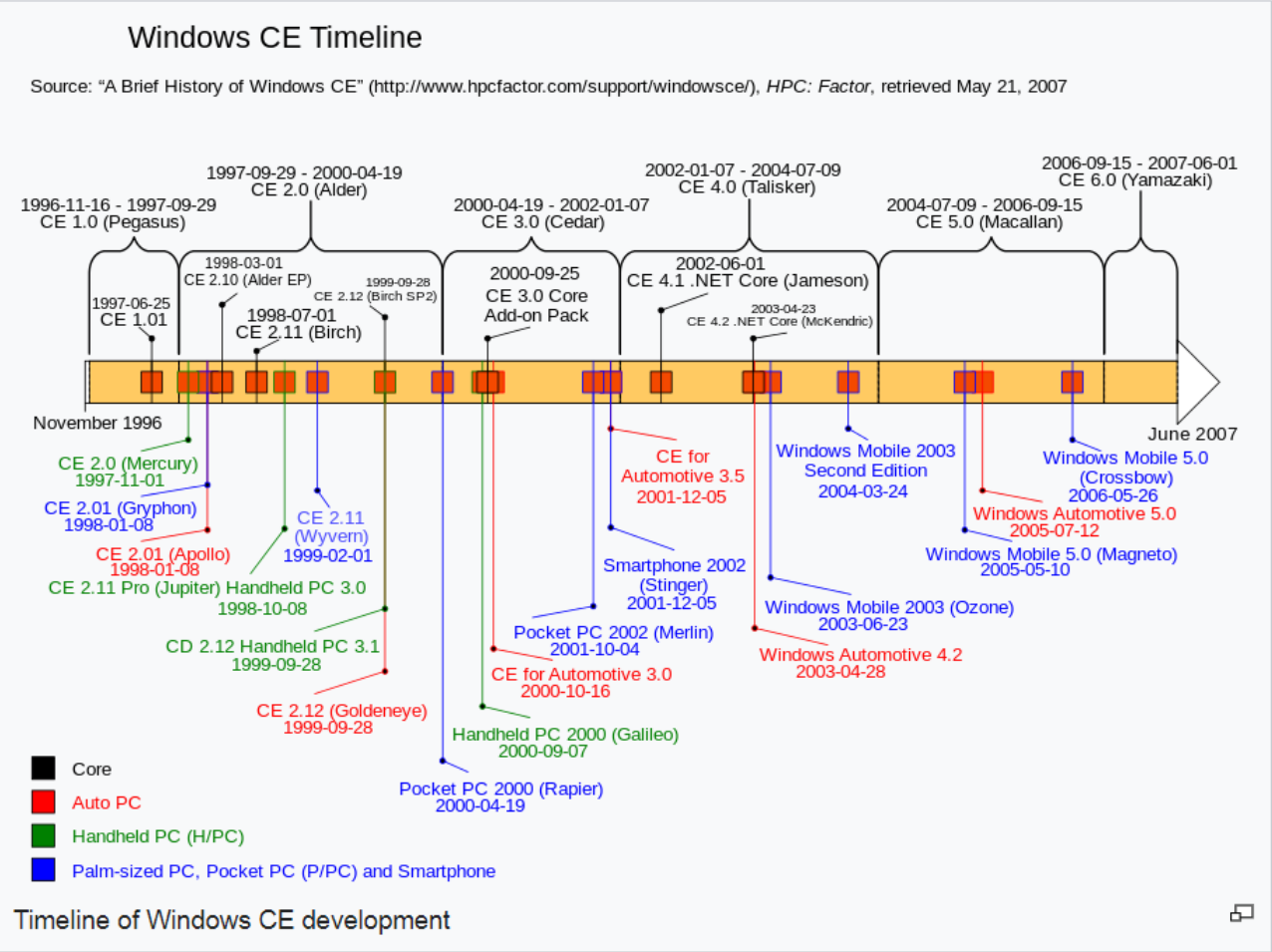
- unlimited amount of tasks (only limited by the amount of available memory)
- preemptive scheduling with up to 2^{32} priorities
- Round Robin with adjustable time-slices for tasks with equal priority
- adjustable time resolution (default is 1ms)
- software timers
- low power and multi-core support
- safe communication among tasks using:
  - task events with up to 32 events per task
  - event objects
  - resource and counting semaphores
  - mailboxes
  - queues
- full interrupt support
- API can be called from assembly, C and C++ code

Nucleus RTOS

- Stable deterministic kernel with a small memory footprint
- Power management APIs for low-power design
- Security – Boot, Data, and Communications including TLS 1.3
- POSIX support for application portability
- A lightweight process model for optimized memory partitioning
- Dynamically load and unload processes for greater modularity of applications
- Multicore Support includes xAMP and SMP support with rpmspg over virtIO and MCAPI for inter-process communication
- File system support for many of the leading format types
- Support for rich UI graphics
- Integrated development tools with Eclipse-Based IDE
- Extensive architecture support including Arm®, RISC-V, MIPS®, and POWER®

Developer	Mentor Graphics Corp., a Siemens Business
Written in	C
OS family	Real-time operating systems
Working state	Current
Source model	Closed source
Initial release	1993; 27 years ago
Latest release	3.x, 2017.02 / May 15, 2017; 3 years ago
Marketing target	Embedded systems, IoT
Available in	English
Platforms	ARM, NXP, MIPS, TI, PowerPC, Altera Nios II, Xilinx MicroBlaze, Renesas SuperH, Infineon, Atmel AT91SAM, others[1]
Kernel type	Real-time monolithic with hybrid support
License	Proprietary
Official website	www.mentor.com /embedded-software /nucleus

Windows CE



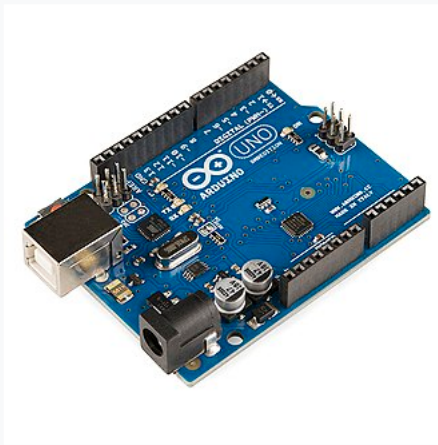
Timeline of Windows CE development

Windows Embedded Compact

Developer	Microsoft
Written in	C[1]
Source model	Closed-source Source-available (through Shared Source Initiative)[2]
Initial release	November 16, 1996; 23 years ago
Latest release	8.0 (Embedded Compact 2013) / June 13, 2013; 7 years ago[3]
Platforms	x86, 32-bit ARM, (SuperH[4] up to 6.0 R2, MIPS and PowerPC were also supported)[5]
Kernel type	Hybrid
License	Commercial proprietary software (volume licensing)
Official website	msdn.microsoft.com/en-ph/embedded/

ΑΛΛΟ ΛΟΓΙΣΜΙΚΟ

Arduino



Arduino Uno SMD R3

Developer	arduino.cc
Manufacturer	Arduino
Type	Single-board microcontroller
Operating system	None
CPU	Atmel AVR (8-bit), ARM Cortex-M0+ (32-bit), ARM Cortex-M3 (32-bit), Intel Quark (x86) (32-bit)
Memory	SRAM
Storage	Flash, EEPROM
Website	www.arduino.cc

Arduino Software IDE

Screenshot of Arduino IDE showing *Blink* program

Developer(s)	Arduino Software
Stable release	1.8.13 / 16 June 2020; 4 months ago[57]
Repository	github.com/arduino/Arduino
Written in	Java, C, C++
Operating system	Windows, macOS, Linux
Platform	IA-32, x86-64, ARM
Type	Integrated development environment
License	LGPL or GPL license
Website	www.arduino.cc/en/Main/Software

Espruino

Official Espruino development boards

Original Espruino

Espruino Pico

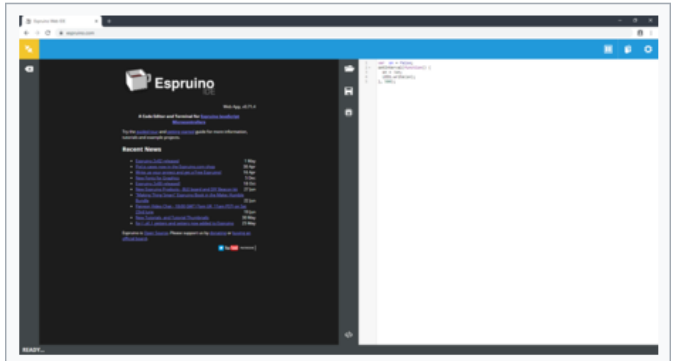
Espruino WiFi

Espruino Puck.js

Espruino Pixl.js

Espruino MDBT42Q Breakout

Espruino Web-Based IDE



Screenshot of Espruino web-based IDE running in Google Chrome

Developer(s)	Gordon Williams
Repository	github.com/espruino/EspruinoWebIDE
License	Apache License 2.0
Website	www.espruino.com/ide/

Node-OS

node-os is a full operating system built on top of the linux kernel

node is the primary runtime - no bash here

node-os uses npm as its primary package manager

open and easy to contribute to - pull request friendly

Βασικά χαρακτηριστικά ΛΣ IoT

Name	Category	MCU w/o MMU	< 32 kB RAM	6LoWPAN	RTOS scheduler	HAL	Energy-efficient MAC layers
Contiki	Event-driven	✓	✓	✓	✘	✓	✓
RIOT	Multithreading	✓	✓	✓	✓	✓	✘[a]
FreeRTOS	RTOS	✓	✓	✘[b]	✓	✘	✘[c]
uClinux	Multithreading	✓	✘	✓	✘	✓	✘
Android	Multithreading	✘	✘	✘	✘	✓	✘
Arduino	Other	✓	✓	✘	✘	✓[d]	✘

(✓) full support, (✘) no support. The table compares the OS for support of MCUs without MMU, MCUs with less than 100 kB of RAM, a 6LoWPAN network stack, a real-time capable scheduler, a hardware abstraction layer (HAL), and energy-efficient MAC layers.

[a] So far only as part of the OpenWSN stack, more implementations are planned.

[b] Available from third parties.

[c] Available from third parties.

[d] Limited portability.

Πηγή: Hahm, O., Baccelli, E., Petersen, H., & Tsiftes, N. (2015). Operating systems for low-end devices in the internet of things: a survey. IEEE Internet of Things Journal, 3(5), 720-734.

Βιβλιογραφία

- Kaur, J., & Reddy, S. R. N. (2018). Operating systems for low-end smart devices: a survey and a proposed solution framework. *International Journal of Information Technology*, *10*(1), 49-58.
- Samie, F., Bauer, L., & Henkel, J. (2016, October). IoT technologies for embedded computing: A survey. In 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS) (pp. 1-10). IEEE
- Hahm, O., Baccelli, E., Petersen, H., & Tsiftes, N. (2015). Operating systems for low-end devices in the internet of things: a survey. *IEEE Internet of Things Journal*, *3*(5), 720-734.