Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Sisteme de operare real time

Madalina-Ioana ALEXE

2013

CUPRINS

Capitolul 1. Sisteme de operare real time.Introducere.

Sistemele în timp real sunt folosite pentru conducerea directã, interactivã, a unui proces tehnologic sau a altei aplicatii (de exemplu, un sistem de rezervare de locuri). Necesitatea unor asemenea sisteme se poate usor imagina în cazul analizelor medicale asistate de calculator, a reactiilor chimice sau a unor experiente fizice (de exemplu, cazul acceleratoarelor de particule). De la procesul controlat se transmit cãtre sistemul în timp real parametrii procesului, culesi prin intermediul unor senzori, iar sistemul în timp real transmite cãtre proces deciziile luate. Informatiile despre proces sunt luate în considerare în momentul comunicării lor iar răspunsul sistemului trebuie să fie extrem de rapid (oportun pentru proces), deci timpii de executie ai programelor din sistem trebuie să fie mici.

Sistemele de operare proiectate pentru aplicatii real-time sunt folosite în general pentru computere de tip "embedded" (înglobate în aparate mai mari, precum telefoane mobile, roboti industriali sau echipamente de cercetare ştiintifică).

Sistemele in timp real (TR) sunt definite ca fiind acele sisteme în care corectitudinea depinde nu numai de rezultatul logic al prelucrarii, ci si de momentul la care este disponibil. Principala dimensiune a sistemelor TR o constituie deci timpul. Anumite prelucrari trebuie realizate în limite de timpi predeterminati, procesarile fiind deci supuse constrângerilor temporale.

1.1 Clasificarea sistemelor in timp real

In functie de strictetea constrângerilor, sistemele TR pot fi împartite în doua subclase:

1.1.1 Hard real time systems

- sisteme in timp real critice - hard real-time systems - sistemele pentru care neîndeplinirea unei constrângeri se considera a fi o eroare grava (failure) a sistemului, putând avea urmari catastrofale

1.1.2 Soft real time systems

- sisteme in timp real necritice - soft real-time systems - sistemele pentru care neîndeplinirea oricarei constrângeri poate fi tolerata.

1.1.3 Nivele de abstractizare

Sistemele TR pot fi ilustrate la diferite nivele de abstractizare.La un nivel înalt de abstractizare, un sistem TR apare ca format din trei subsisteme componente.

- Subsistemul controlat reprezinta aplicatia sau mediul care dicteaza cerintele TR.

- Subsistemul de control este întregul echipament de calcul, care este conectat cu mediul controlat printr-un numar de intrari si iesiri, formând interfata aplicatiei. Subsistemul de control

poate cuprinde unul sau mai multe procesoare si resurse. Procesoarele si resursele sunt gestionate de un sistem software denumit sistem de operare TR - SOTR. În mod normal, un sistem TR are o interfata cu un operator uman,

• Subsistemul operator, care initializeaza si monitorizeaza întregul sistem, dând anumite comenzi, mai ales în situatii exceptionale. Sistemele TR ale viitorului se doresc a fi suficient de inteligente, sigure, autonome, pentru o executie independenta de operatorul uman.

Sistemele de operare real-time sunt folosite în principal pentru capacitatea lor de răspuns rapid, şi nu atât pentru volumul total de muncă (calcule) pe care îl pot efectua. Un sistem de operare real-time nu e nevoie să dispună de o viteză (putere) mare de calcul — viteza lui de calcul este influenTată nu numai de viteza de calcul a procesorului pe care rulează, dar şi de algoritmul specializat de planificare precum şi de frecvenŃa ridicată a întreruperii de ceas.

Sistemele de operare în timp real au fost create pentru sistemele fizice de timp real, a căror operare nu depinde doar de rezultatul logic al procesării, ci şi de timpul în care sunt produse rezultatele .De aceea timpul devine o coordonată esenţială, cu impact în toate fazele dezvoltării şi exploatării acestora, de la specificare a întregului sistem şi până la execuţia la nivel de task. Un sistem de timp real este un sistem suficient de rapid pentru a garanta respectarea termenelor de timp, în cazul cel mai defavorabil de operare. Constrângerile de timp se împart în constrângeri “hard” (în sensul de critice, stricte), respectarea termenelor este critică, nerespectarea lor având consecinţe catastrofale şi “soft” (lejere), depăşirea execuţie a funcţiei .

Capitolul 2. Mecanisme de timp real

Cele mai importante caracteristici ale unui sistem de management al timpului într-un sistem de operare sunt:

• Rezoluţia cu care este reprezentat timpul trebuie să fie cât mai fină cu putiinţă.

• Timpul de funcţionare trebuie să fie cât mai mare.

• Pentru sistemele cu resurse limitate, numărul de biţi utilizaţi pentru reprezentarea timpului trebuie să fie mic.

• Timpul de utilizare procesor adăugat de mecanismul de timp trebuie să fie mic.

Într-un sistem de timp real timpul poate fi reprezentat in doua feluri: ca intreg ce reprezintă timpul scurs din momentul resetării sistemului, exprimat în ciclii de tact sau multiplii de cicli de tact daţi de întreruperi de ceas; al doilea tip îl reprezintă o structură mai complexă ce reprezintă timpul exprimat în secunde si nanosecunde(implementat în sisteme de operare de tip POSIX1). Ca mod de implementare o primă variantă este ca nucleulsistemului de operare să menţină un ceas prin software, actualizat de întreruperea de ceas. O altă abordare este cea de a folosi direct valoarea din

registrul unui timer pentru a exprima timpul. Asfel frecvenţa de tact alimentează în mod direct baza de timp, granularitatea de timp minimă cu care poate lucra sitemul fiind egală cufrecvenţa procesor. Această abordare duce la rezoluţii cu cel putin un ordin de marime mai bune decât în primul caz.

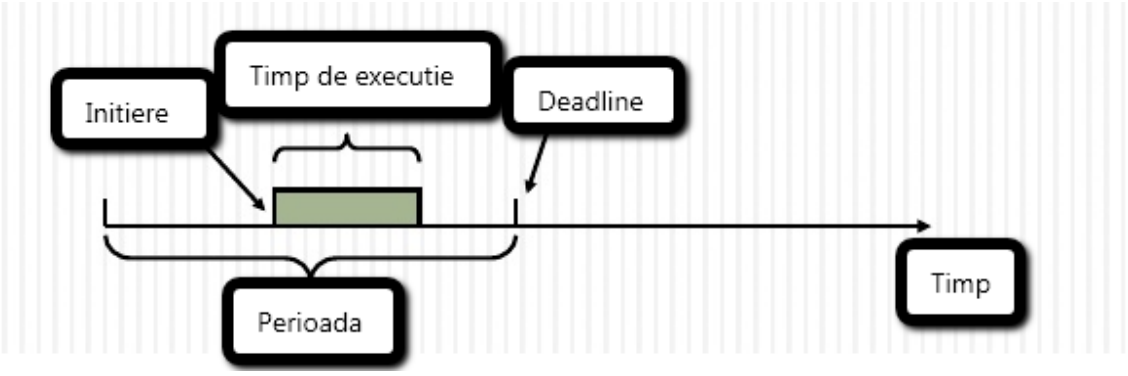
Capitolul 3. Tehnici de planificare și planificatoare

Planificarea taskurilor se referă la găsirea de soluţii fiabile pentru asignarea procesorului, pentru fiecare task în parte, asfel încât să nu existe suprapuneri ale execuţiei lor pe durata operării sistemului . În ceea ce privesc tehnicile de planificare există mai multe direcţii de abordare între care amintim :

• Abordarea bazată pe mecanismul de timp **(Clock-Driven)** - Este o abordare bazată pe algoritmi statici de planificare. Planificarea se repetă la intervale regulate, aceasta făcându-se pe cicli de timp (cicli de planificare). Lansarea planificatorului se face pe întreruperi periodice de ceas.

• Abordarea bazată pe mecanisme de tip coadă **(Round-Robin)** - Este o abordare folosită în mod curent pentru planificarea aplicaţiilor ce partajează timpul. Abordarea bazată pe priorităţi (Priority-Driven) se referă la o clasă largă de algoritmi de planificare ce nu lasă nici o resursă inactivă în mod intenţionat. Decizii de planificare se fac atunci când apar evenimente ca lansarea unei noi instanţe a unui task sau terminarea execuţiei acestuia.

Fiecare proces este format din : timp de executie , perioada si deadline :



Algoritmii de planificare se pot clasifica după numărul de procesoare sistem, în algoritmi monoprocesor și algoritmi multiprocessor; după momentul de generare al planificării, în algoritmi statici (planificare generată offline) și algoritmi dinamici (planificare generată online, în timpul operării sistemului); în funcţie de acceptarea sau nu a întreruperilor în algoritmi preemptivi (se admite întreruperea task-ului curent de către un alt task cu prioritate mai mare) și algoritmi nonpreemptivi (nu se admit întreruperi ale taskurilor) . Mecanismele de planificare implementează tehnicile menționate. Două mari categorii de planificatoare sunt: cele bazate pe priorități (fixe și alocate dinamic) și cele statice.

Capitolul 4. Tipuri de sisteme de operare in timp real

4.1 QNX

4.1.1 Particularitati

QNX este un microkernel RTOS. Un microkernel RTOS este special în felul în care sistemul de operare este construit cu un număr de servicii mici care fiecare se ocupa de o anumită sarcină, aceasta fiind de fapt un set de apeluri de sistem, care sunt disponibile dezvoltatorului. Aceste servicii pun în aplicare comunicarea numai la nivel scăzut in rețea, precum și între procese, intre procesul de planificare, întreruperea manipularii. Alte funcții OS , de exemplu , ca sistemul de fișiere și drivere de dispozitive sunt trimise ca procese ale utilizatorului în afara kernelului. În contrast cele mai multe sisteme de operare folosesc un kernel monolitic. În acest caz, toate funcțiile kernel apartin unui program mai mare. Urmarind design-ul microkernelui în QNX este de retinut ca acesta a dus la un cod mic cu complexitate redusa. Acest lucru ar putea fi un exemplu util atunci când se stabilește cat timp iau apelurile de sistem, sau atunci când se face depanare codului utilizator. Probabil, cel mai mare avantaj, in această abordare este că sistemul de operare este foarte scalabil. Dezvoltatorul este liber să aleagă exact ce componente ar trebui să faca parte din RTOS.

4.1.2 Gestiunea de procese

Fiecare proces în QNX este format din unul sau mai multe thread-uri. Când vine vorba de planificare, QNX planifica fire ci nu procese. T shread-urile sunt programate la nivel global peste sistem, ceea ce înseamnă că acestea sunt tratate la fel indiferent de procesul din care fac parte. Acest lucru este valabil și pentru prioritatea threadu-urilor, mai precis, nu contează carui proces apartine un thread. Când sunt planificate thread-urile, scheduler-ul tine cont de prioritatea tuturor thread-urilor din sistem.

Există 64 de niveluri de priorități, care pot fi atribuite la unui thread, unde 63 este cea mai mare și 0 este cea mai mică prioritate. Nivelul de prioritate 0 este un caz special, în care este plasat procesul de mers în gol al sistemului. Fiecare nivel de prioritate are coada de așteptare proprie în care sunt plasate toate procesele care au acea prioritate. Atunci când thread-urile sunt reprogramate

planificatorul selectează firul cu cea mai mare prioritate, care este gata de a rula și ii da acestuia CPU-ul la dispozitie. Acest lucru se întâmplă ori de câte ori un thread isi modifică statutul sau este preluat de la CPU.

4.1.3 Algoritmi folositi in programarea thread-urilor

QNX suporta diferiti algoritmi pentru programarea thread-urilor: FIFO, Round Robin și programarea sporadică. Toate thread-urile sunt programate independent, ceea ce permite sa fie utilizati diferiti algoritmi pe threaduri dar în același sistem, iar prioritățile și algoritmi pot fi, de asemenea, modificati în timpul de rulare a thread-urilor.

4.1.3.1 FIFO (firstl-in-first-out)

Acest tip de programare este descris destul de mult dupa numele acestuia. Primul thread care este plasat in coada prioritatilor este primul care va selectat pentru rulare.

4.1.3.2 Round Robin Scheduling

Când este utilizat planificatorul Round Robin Scheduling, thread-urile sunt puse in coada sa de prioritati. Ce este interesant in acest caz este ca fiecare thread are cate o perioada de timp la dispozitie in care ar putea rula. Thread-ul care se află în partea din față a cozii este selectat pentru a rula.Dacă firul depășește timpul dat este preluat și a pus in partea din spate a cozii din nou, permițând altor thread-uri să ruleze între timp.

4.1.3.3 Programarea sporadica

Programarea sporadica funcționează un pic diferit. Fiecarui fir ii este atribuita o "prioritate normală" și o "prioritate scăzută", împreună cu un anumit timp care reprezintă timpul in care thread-ului ii este permis ruleze la prioritatea sa normală. Atunci când thread-ul a rulat la prioritate normala pentru întreaga durată a timpului alocat , prioritatea threadului este schimbata la prioritate scazuta. După câteva thread-uri de timp utilizând programarea sporadica, acestea obțin timpul alocat pentru completarea ciclului și isi pot relua activitatea la prioritate normală.

4.2 VxWorks

4.2.1 Particularitati

Sistemul de operare VxWorks se bazeaza pe microkernelul wind. Acest microkernel suporta o multime de trasaturi real-time precum multi-tasking, organizare, comunicare si sincronizare intre taskuri, administrarea memoriei. Toate celelalte functionalitati sunt implementate sub forma unor procese.

4.2.2 Gestiunea de procese . POSIX scheduler

Într-o configurare de bază a VxWorks un singur proces este folosit și atât kernel-ul și sarcinile sunt partajare in acelasi spatiu de adresă, ceea ce permite comutarea rapida de context , rezultand o înaltă performanță. În cele mai multe RTOS thread-urile sunt utilizate pentru a oferi

concurenta. VxWorks utilezeaza sarcinile, care sunt aproape la fel ca și thread-urile. În cazul în care o configurare VxWorks este fără suport pentru procese, atunci toate sarcinile sunt executate în acelasi spațiu de adresa ca și kernel-ul. Acest lucru permite comutarea rapida a contextului, dar pune limite asupra memoriei. Aceasta nu este întotdeauna de dorit si anume să împartă sarcinile de memorie cu kernel-ului. VxWorks poate fi rulat cu sprijinul procesului, și apoi sarcini obțin propriul spațiu de adresă ceea ce le face mai mult egale cu un thread.

VxWorks oferă un set de opțiuni diferite pentru programare, tradiționalul VxWorks scheduler, POSIX scheduler sau un scheduler personalizat. Schedulerul VxWorks nu suportă threaduri în procese, doar in kernel, astfel încât concurența într-un program pentru utilizator se realizează cu task-uri. Când este utilizat POSIX scheduler,VxWorks este capabil de a crea thread-uri, precum și sarcini. Thread-urile și task-urile sunt tratate în mod implicit de aceeași scheduler și sunt programate la nivel global asupra sistemului.

Schedulerul VxWorks utilizeaza programare preventiva bazata pe prioritate. În VxWorks, există 256 de niveluri de prioritate, in care 0 este considerat a fi de cea mai mare prioritate și 255 cea mai mica prioritate. Acest lucru înseamnă că sarcina cu cea mai mare prioritate (număr mic) va fi intotdeauna selectată pentru a rula. Dacă există mai multe sarcini cu aceeași prioritate, prima sarcină va rula până când se eliberează CPU-ul. Acest lucru se poate sau nu se poate întâmpla, în funcție de modul în care sarcina este construita. Pentru a rezolva această problemă un algoritm Round Robin poate fi aplicat pentru a da tuturor sarcinilor o bucata de timp pentru a rula. După ce bucata de timp a fost consumat threadul este preluat și a pus la sfârșitul cozii de așteptare din nou. Programarea este încă bazată pe prioritate, deci, dacă există threaduri cu prioritate mai mare vor fi selectate, indiferent dacă programarea Round Robin este folosita sau nu. Planificatorul POSIX este o unitate de programare mult mai capabila. Aceasta susține si threadurile și sarcinile și este capabil de a programa threaduri la nivel de sistem cat si pe bază de proces. Pot fi folositi diferiti algoritmi de planificare atat la nivel de thread cat si la nivel de task, cei doi algoritmi de planificare disponibili fiind FIFO și Round Robin.

4.3 RTLinux

4.3.1 Particularitati

Exista doua versiuni diferite de RTLinux: RTLinux/Pro si RTLinux/Open. Ne vom concentra asupra versiunii RTLinx/Open.

Exista doua metode de a obtine performante in timp real pentru un sistem Linux:

1. Imbunatatirea preemptiunii kernelului Linux.

2. Adaugarea unui nou nivel software sub kernelul Linx cu control deplin asupra intreruperilor si caracteristicilor importante ale procesorului.

Aceste metode sunt cunoscute ca “imbunatatirea preemptiunii” si “abstractizarea intreruperilor”. Cea din urma metoda este folosita de RTLinux.

RTLinux este un sistem de operare mic si rapid, bazat pe standardul pentru sisteme de operare in timp real POSIX 1003.13.

RTLinux adauga un nivel de virtualizare hardware intre kernelul Linux standard si partea hardware a computerului . Din punctul de vedere al kernelului Linux standard, noul nivel apare ca hardware. RTLinux implementeaza un sistem de operare in timp real complet si previzibil, fara interferente din partea Linuxului care nu lucreaza in timp real. Firele de executie RTLinux sunt executate direct de catre un organizator cu prioritati fixe. Intregul kernel Linux si toate procesele Linux normale sunt administrate de organizatorul RTLinux care lucreaza pe fundal. In acest fel, este posibil sa avem un sistem de operare cu scop general care lucreaza peste un sistem de operare in timp real predictibil.

4.3.2 Managementul proceselor

S-au facut trei modificari kernelului Linux pentru a virtualiza hardware-ul astfel incat RTLinux sa detina intregul control asupra computerului. Nivelul RTLinux are control direct asupra tuturor intreruperilor hardware, intreruperile care nu sunt controlate de fire de executie real-time sunt incredintate nivelului superior Linux. RTLinux detine controlul timerului hardware si implementeaza un timer virtual pentru Linux. Ultima modificare adusa Linuxului a fost sa inlocuirea functiilor care modificau flagurile de intrerupere pentru validare si invalidare (cli si sti) astfel ca Linuxul sa nu poata face o invalidare reala ci doar una virtuala. Aceste modificari sunt complexe si dificile dar nu necesita modificari foarte multe in cod.

RTLinux ofera un mediu de executie mai jos de kernelul Linux. O consecinta este ca firele de executie real-time nu pot folosi serviciile Linux deoarece pot aparea blocaje sau lipsa de consistenta in sistem. Pentru a trece peste aceasta problema, sistemul real-time trebuie sa se imparta in doua nivele: **nivelul real-time hard**, executat peste RTLinux, si **nivelul soft real-time**, executat asemanator proceselor Linux normale. Sunt folosite cateva mecanisme(FIFO, memorie partajata) pentru a se putea face comunicarea intre cele doua nivele.

Folosirea celor doua nivele este o metoda utila de a oferi functionalitate real-time in acelasi timp cu caracteristicile unui sistem de operare de tip desktop. Desface mecanismul kernelului real- time de cel al kernelului general astfel incat fiecare sistem poate fi optimizat independent.

4.3.2.1 Politica de planificare

Exista trei metode de planificare: SCHED_FIFO, SCHED_SPORADIC, SCHED_EDF.

SCHED_FIFO este o planificare cu prioritate fixa in care threadurile cu aceeasi prioritate sunt executate in ordinea FIFO.

SCHED_SPORADIC este o implementare a unui server sporadic folosit pentru a rula actiuni aperiodice. **SCHED_EDF** implementeaza EDF (Earliest Deadline First), o politica de planificare dinamica a prioritatilor. Firele de executie sunt ordonate in functie de prioritate, dar firele cu aceeasi prioritate sunt planificate tinand cont de politica EDF.

4.3.2.2 Domeniul prioritatilor si numarul maxim de fire de executie

Prioritatea poate lua valori in intervalul 0 – 1000000. Nu exista o limitare a numarului de threaduri, dar costul planificarii este proportional cu numarul acestora. Planificatorul actaul este proiectat sa manevreze eficient un numar mic de threaduri (aproximativ 10).

4.3.2.3 Administrarea memoriei

Spatii de adresa protejate

Desi RTLinux este proiectat pentru a rula in procesoare cu MMU, toate firele de executie ale aplicatiilor si kernelul RTLinux ruleaza in acelasi spatiu de adrese. Nu exista o protectie a memoriei intre threaduri si kernel si nici intre threaduri diferite.
Din punctul de vedere al administrarii memoriei, RTLinux este sistemul de operare “oaspete” (guest) al kernelului Linuxului. Kernelul Linuxului detine controlul deplin al memoriei.

Alocarea dinamica a memoriei

RTLinux nu ofera alocare dinamica a memoriei. Principalul argument este ca alocarea dinamica a memoriei nu este predictibila daca este implementata in mod eficient. Scopul referitor la predictibilitate al sistemelor real-time este atins prin prealocarea resurselor pe care firul de executie le va folosi la rulare. Este posibil sa se aloce toata memoria necesara fiecarui thread inainte ca threadul sa fie creat.

Capitolul 5. Comparatie intre QNX si VxWorks

QNX foloseste un microkernel, VxWorks foloseste un nucleu monolitic, acestea fiind două abordări complet diferite atunci când vine vorba de a construi un sistem de operare.

Ambele au argumente pro și contra, microkernelul cu mica sa complexitate, cu mica sa scalabilitate sa poate fi implementat ca un mic RTOS, avand capacitatea de doar cateva sute de kb,putand fi încorporat intr-o aplicatie sau folosit ca un adevart RTOS cu functii avansate.

VxWorks vine într-o arie care poate fi extinsa cu module. Acesta este scalabil, dar nu oferă aceeași libertate pentru dezvoltator ca si QNX. În QNX puteți sterge destul de mult din ceea ce nu aveți nevoie, în VxWorks nu poți. Acest lucru se datorează modului in care cele doua RTOS diferă în strategia lor de kernel.

Concurenta este o caracteristică importantă în cele mai moderne RTOS, acest lucru este realizat de QNX cu procese și thread-uri. În VxWorks poate fi realizat cu procese, thread-uri și sarcini, în funcție de configurație.

QNX programeaza thread-urile dar nu și procesele, programarea devinind consistenta asupra sistemului. Într-o configurare de bază VxWorks nu suportă procesele de utilizator sau thread-uri, ci numai task-uri. Sistemul de operare poate fi configurat cu suport POSIX si suport de proces să acorde sprijin acestuia, dar programarea este încă limitat la FIFO și Round Robin.

Când se utilizeaza schedulerul implicit al VxWorks programarea se face la nivel global, toate sarcinile care rulează folosind același algoritm de planificare.

Când se foloseste schedulerul POSIX este posibil să se programeze fiecare fir de executie sau proces independent. În QNX este posibil să se programeze thread-uri cu algoritmi diferiti independenti unul de cealalalt.

Datorită modului in care VxWorks este construit acesta poate programa sarcinile și thread-urile, fie global, fie pe baza de proces sau pe baza de thread . Acest lucru nu este posibil în QNX, deoarece nu diferențiază un thread de altul si nu contează de ce proces apartine un thread , toate thread-urile din sistem fiind tratate la fel.

VxWorks ramane in urma , atunci când vine vorba de algoritmii de planificare; deoarece in cazul QNX exista trei algoritmi de planificare: FIFO, Round Robin și programare sporadică. Doar două dintre ele coexistă în VxWorks, FIFO și Round Robin. Acesti algoritmi diferiti lucrează în același mod în ambele sisteme, și sunt utilizate împreună cu diferite nivele de prioritizare.

Ambele RTOS aplică programarea preventiva bazata pe prioritizare, ceea ce înseamnă că procesul / thread-ul / task-ul cu cea mai mare prioritate va rula întotdeauna.

Capitolul 6. QNX.VxWorks.CONCLUZII

VxWorks are o strategie de programare mai complexa, este mai puțin scalabil si suporta mai putini algoritmi de planificare decat QNX. VxWorks este un sistem extrem de personalizabil, cu o multime de caracteristici, poate prea multe?

Din moment ce este atât de complex, cu atat de multefuncții si optiuni se pare ca are o curbă de învățare ridicata și cere o mulțime de timp și experiență de la dezvoltator pentru a putea utiliza sistemul. De exemplu, atunci când veți începe de la zero cu un sistem de operare VxWorks vei fi capabil doar de a rula taskuri deoarece are o configuratie primitiva al programării. VxWorks are un mare potențial, dar acesta este în funcție de modulele suplimentare, de exemplu, biblioteca POSIX trebuie să fie inclusa pentru a putea utiliza threaduri.

QNX pe de altă parte, pare a fi mai apropiat de sistem de operare regulat. Datorita acestui aspect acesta este mai simplu de folosit,si poate furniza toate funcționalitatea pe care VxWorks o are. În cazul în care trebuie sa alegem între aceste două sisteme de operare pentru un produs nou, nu contează cu adevărat care din ele este ales numai uitandu-ne la funcționalitatea și fiabilitatea acestora . Ceea ce diferă este cel mai mult sunt aspectele tehnice si ce strategii sunt utilizate pentru a implementa sistemele de operative.

Amândoua au avantajele și dezavantajele lor, care tin si de domeniul lor special de aplicare. Dar in general, VxWorks pare să fi fie cea mai bună combinație de structură de date și algoritmi.

Capitolul 7. Bilbiografie

Andrew S. Tanenbaum - Distributed Operating Systems (Prentice Hall, 1994)

http://www.techonline.com/education-training/fundamentals/4213896/Fundamentals-of--Real-Time-Operating-Systems

http://en.wikipedia.org/wiki/Real-time_operating_system

http://www.engineersgarage.com/articles/rtos-real-time-operating-system

http://gate.upm.ro/os/Curs/curs_sisteme_operare.pdf