Hard Real Time Characteristics of Linux RTAI Based Embedded Systems

Eugen DODIU, Adrian GRAUR, Vasile Georghiță GĂITAN

***Abstract*—A system is considered to be a hard real-time if all the computations are not only correct but their response is provided each time before a fixed deadline. Having a deadline missed for such a system can possibly produce severe material damage or human health injury, therefore adequate testing and analysis should be done.**

This paper evaluates the suitability of a general purpose operating system like Linux with a real time extension. As can further be seen Linux has overall good raw performance that can make it usable for an embedded system with industrial usage. To show the characteristics under full load stress we have done several test cases for different scenarios that will be presented next in the paper.

***Index Terms*—hard real-time/soft real-time, embedded systems, Linux RTAI, real-time scheduling/communication**

I. INTRODUCERE

În acest articol ne propunem să interpretăm rezultatele practice obținute în urma testării unei arhitecturi hardware specifice, bazate pe un sistem de operare cu grad de utilizare ridicat si o extensie de timp real pentru a observa dacă el poate fi sau nu considerat hard real-time.

Sistemele de timp real diferă de cele clasice prin faptul că ele trebuie să ofere raspuns la o problema într-un anumit interval de timp. În sistemele de timp real funcționarea corectă a task-urilor nu se rezumă doar la furnizarea unui răspuns corect ci şi la timpul când aceste rezultate sunt furnizate. Spre deosebire de sistemele de timp real soft în care neîndeplinirea unui deadline nu este o problemă majoră în cadrul sistemelor hard real-time ratarea unei limite de timp impuse poate atrage dupa sine producerea de pagube materiale şi chiar rănirea sau pierderea de vieți omenești [1].

Pentru acest motiv problematica sistemelor de timp real hard trebuie privită cu luare aminte. De cele mai multe ori realizarea unui sistem cu funcționare de timp real hard este o adevarată provocare inginerească deoarece trebuie analizați o multime de factori din sistem ce pot influența semnificativ stabilitatea acestuia. În funcție de domeniul de aplicație se poate opta pentru folosirea unui sistem hard real-time sau soft real-time. Putem da ca exemplu concret mecanismul de detecție a solului dintr-un avion, mecanism care poate să execute bucla principală a task-ului de detecție pană la 40 de ori pe secundă. Acesta necesită un sistem de tip hard real-time deoarece apariția unei erori în cadrul execuției programului , chiar şi o singura dată, poate avea efecte distructive. În funcție de posibilitatea adăugării de noi taskuri în timpul funcționării echipamentelor, sistemele de operare pot fi împărțite în statice sau dinamice.

De obicei sistemele de operare statice intră în gama echipamentelor de tip hard real-time în timp ce sistemele de operare dinamice nu pot fi clasificate în aceeaşi categorie.

Sistemele de operare statice presupun funcționarea cu un număr prestabilit de task-uri încă de la momentul compilării si încarcării programului cu imposibilitatea adăugării de noi task-uri pe parcursul execuției programului. Uzual aceste executabile rulează din memorii flash ceea ce ar face imposibilă adăugarea lor în mod dinamic în procesul de run-time. În cealaltă categorie intră sistemele de operare dinamice care permit adăugarea de noi task-uri foarte uşor chiar în timpul funcționării sistemului nemaifiind necesară oprirea, recompilarea, reprogramarea si repornirea echipamentelor. Din acest punct de vedere se poate observa că cele doua caracteristici viteza de executie şi scalabilitatea execuției task-urilor sunt într-un raport invers proporțional, de aceea alegerea soluției optime cade în sarcina inginerilor proiectanți .

În mod uzual destinația sistemelor de operare precum Windows sau Linux nu este una industrială, ele fiind folosite cu precădere în sistemele de calcul de tip Desktop, stații grafice , stații de calcul, mainframe-uri şi în general în calculatoarele cu putere de calcul mare [3]. Trebuie de făcut distincția că, de obicei, puterea de calcul foarte mare a acestor calculatoare nu presupune în mod neapărat şi execuție real-time. În ultimele perioade inginerii au facut eforturi considerabile pentru a aduce şi aceste sisteme de operare precum Windows şi Linux în gama celor de timp real cu oarecare succese însa funcționarea lor în cadrul dispozitivelor hard real-time stă sub semnul întrebării. În lucrarea de față am încercat efectuarea unor teste practice pe un sistem hardware cu destinație industrială pentru a determina care sunt performanțele sistemului şi daca ele pot fi acceptate în clasa celor hard real-time.

II. PLANIFICAREA ÎN SISTEMELE DE OPERARE DE TIMP REAL

Activitățile în sistemele de timp real sunt îngrădite de obicei de constrângeri care stabilesc foarte clar timpii de început ai unui task , durata de execuție şi timpii de sfârşit. Obținerea unei calitați ridicate a serviciilor într-un astfel de sistem consta în realizarea unei scheme de planificare a task-urilor care să ofere rezulatele aşteptate în orice situație, ori obținerea unui astfel de deziderat nu este tocmai uşor. În partea introductivă a lucrării se amintea de complexitatea proiectarii sistemelor de timp real [4]. Dacă ar fi să motivăm această afirmație putem

spune că problemele deosebite apar în procesul de design al blocurilor software deoarece funcționarea sistemului este strict dependentă de subsistemul de întreruperi care funcționează total asincron şi determină extinderea acestei caracteristici asupra întregului sistem.

A. MODELUL MATEMATIC AL TASK-ULUI DE TIMP REAL

Task-urile sunt entitațile de bază care se execută într-un sistem de operare de timp real. Ele pot fi periodice sau aperiodice şi pot avea sau nu contrângeri de timp real. Principalii parametrii ai unui task sunt:

r – timpul de lansare a unui task (periodic sau trigerat de un eveniment)

C – timpul maxim de execuție a unui task

D – timpul de finalizare cu intârzierea maxim acceptată

T – perioada task-ului (caracteristic doar task-urilor periodice)

Când un task are o constrângere de timp real atunci intervine şi noțiunea de deadline absolut descris de relatia: $d=r+D$. Depăşirea deadline-ului absolut *d* determină o ratare a limitării de timp impuse pentru acel task,iar pentru taskurile care funcționează în sistemele de timp real acest lucru poate duce la efecte adverse severe. În timp ce un task periodic este modelat de toți cei patru parametrii în cazul task-urilor aperiodice parametrul *T* este absent. Pentru task-urile aperiodice timpii de lansare în execuție succesivi vor apărea de forma $r_k = r_0 + kT$ în care r_0 este timpul primei lansări iar r_k este lansarea în execuție cu numărul de ordine $k+1$. Timpii de finalizare absoluți sunt descrişi de egalitățile: $d_k = r_k + D$; dacă $D=T$ atunci timpul de execuție maxim admis este egal cu perioada. *Un task este bine format dacă ecuația* $0<C\leq D \leq T$ *se respectă [5]*. Parametrii principali ai unui task sunt prezentați în Fig. 1.

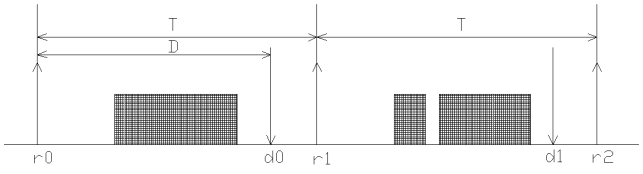

Fig.1. Modelul temporal al task-ului.

Dupa crearea task-urilor acestea vor evolua uzual între stadiile pasiv, pregătit şi în execuție dar toate stările posibile sunt urmatoarele:

în execuție – procesorul este alocat acelui task

blocat – atunci când un task este blocat pe o anumită resursă sau aşteaptă un mesaj

pregatit – task-ul aşteaptă să fie executat

pasiv – task-ul nu are nici o cerere de execuție

ne-existent - task-ul nu este creat încă

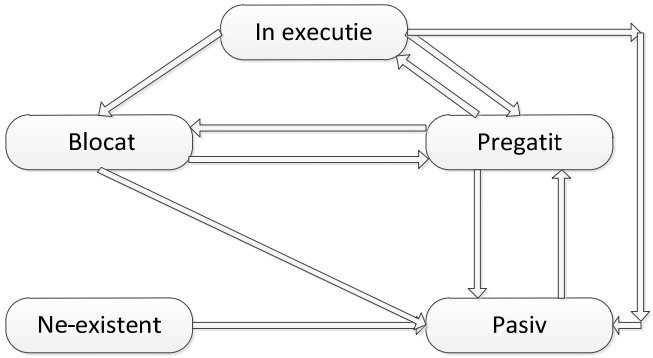

Fig.2. Posibilele stări de existență ale unui task.

Pe baza parametrilor descrişi mai sus se pot defini urmatoarele notiuni:

Factorul de utilizare a procesorului

Pentru un set de *n* task-uri periodice are urmatoarea expresie:

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti} \qquad (1)$$

Factorul de încărcare al procesorului

Pentru un set de n task-uri periodice n aceasta variabila se calculează după următoarea formula:

$$CH = \sum_{i=1}^{n} \frac{Ci}{Di} \qquad (2)$$

Din cauza timpilor limita nici factorul de încărcare (2) nici factorul de utilizare (1) nu sunt suficiente pentru a evalua efectul încărcării procesorului în situația constrângerilor de timp. Pentru această cauză s-a introdus noțiunea de *relaxare a procesorului* şi reprezintă timpul maxim cât procesorul poate să ramână în idle dupa eticheta de timp *t* fara a cauza ratarea unui deadline. *LP(t)* variază în funcție de valoarea lui *t*. Pentru toate valorile lui *t, LP(t)≥0* [7], [9]

B. ALGORITMI DE PLANIFICARE

Planificarea unui set de task-uri presupune execuția acestora cu respectarea constrangerilor de timp pentru :

-toate task-urile din sistem când sistemul funcționează în modul nominal

-cel puțin task-urile vitale de care depinde funcționarea în condiții sigure a procesului [8].

Algoritmul de planificare este cel care asignează taskurile procesorului si oferă o listă a task-urilor ce sunt programate pentru execuție.

Planificarea On-line şi Off-line

Acest tip de planificare crează o schiță completă de execuție cu toti parametrii task-urilor. Aceasta schiță este cunoscută dinaintea execuției şi acest lucru determină o abordare rigidă în care toți parametrii sistemului sunt cunoscuți de la început [5], [7].Sistemul este foarte greu adaptabil în timpul funcționării de aceea o bună alternativă este planificarea On-line, ce permite selectarea în orice moment a task-urilor în funcție de evenimente. Acest tip de planificare este specifică task-urilor aperiodice.

Planificarea Preemptivă sau non-preemptivă

În planificarea preemptivă orice task poate fi întrerupt de un task mai prioritar, dacă acel task a fost activat ca urmare a sosirii unui eveniment extern pentru care el stătea în aşteptare. În cadrul planificarii non-preemptive task-urile nu pot fi întrerupte în această manieră. Avantajul acestui tip de panificare este că ea exclude accesul concurent la resursele critice ale sistemului în schimb are şi unele dezavantaje în ce priveşte timpii superiori de execuție pentru unii algoritmi.

Planificarea centralizată sau distribuită

Planificarea este centralizată dacă este facută pe o arhitectură centralizată sau pe un nod care înregistrează parametrii tuturor taskurilor nodurilor. Planificarea în sistemele multi procesor este distribuită daca nodurile dintr-un sistem au cel puțin un mecanism de planificare locală cu strategii de cooperare între noduri. Existența acestor strategii poate face posibilă migrarea task-urilor între noduri .

III. CARACTERISTICI DE PERFORMANȚĂ ALE SISTEMELOR BAZATE PE LINUX RTAI

Sistemele embedded realizate cu ajutorul sistemului de operare Linux nativ nu sunt de timp real datorita jitter-ului mare pe care îl au task-urile. În vederea îmbunătățirii performanțelor şi asigurarea unui suport pentru aplicațiile de timp real extensia de timp real RTAI este o opțiune demnă de luat în calcul [4].

RTAI-ul a fost construit ca o parte integrantă a Linux-ului şi poate fi rulat atat ca modul kernel cât şi ca proces în spațiul utilizator. Prin intermediul acestuia se pot desemna în sistem task-uri care să ruleze în spațiul de adrese kernel cu o prioritate mai mare decat însuşi kernelul.

A. DESCRIEREA ECHIPAMENTULUI DE TEST

Pentru realizarea testelor propuse am folosit urmatorul suport hardware şi software:

HARDWARE

1) Placă de test TS 7300 produsă de Technologic Systems cu urmatoarele componente:
   - Procesor Cyrrus Logic EP9301 (arhitectura ARM 920T)
   - Periferice: porturi IO , port USB, JTAG , CAN, UART
   - Ceas de timp real
   - Circuit FPGA Altera Cyclone II programabil direct din Linux
   - Compact Flash
   - Port de expansiune PC104
2) Sursă de alimentare
3) Switch Ethernet cu 5 porturi
4) Programator pentru CF
5) Osciloscop TEKTRONIX TDS2024B

SOFTWARE

Pentru placa:
1) Sistem de operare Linux Debian
2) Kernel 2.4.26
3) Adeos Patch de la Technologic Systems
4) RTAI versiunea 3.2
5) Compilator gcc 3.3.4 pentru ARM
6) Librariile Glibc 2.3.2 pentru ARM

Pentru dezvoltare:
1) Sistem de dezvoltare Linux Red Hat 9.0
2) Cross compiler pentru ARM
3) Compilator gcc 3.2.2 20030222(Red Hat Linux 3.2.2-5)

B. PREGĂTIREA MEDIULUI DE TEST

Pregătirea mediului de test presupune o serie de etape care sunt descrise în continuare. Ordinea în care ele sunt scrise este importantă.

Compilarea unui crosscompiler

Aceasta etapă presupune descărcarea crosscompilatorului crosstool-0.28. Pentru folosirea acestuia este necesară ştergerea variabilei LD_LIBRARY_PATH iar apoi în fisierul demo-arm.sh se debifează rândul “cat arm.dat gcc-3.3.4-glibc-2.3.2.dat sh all.sh -notest”. Se lansează în compilare folosind comanda “./demo-arm.sh”.

Instalarea pachetului modutils-build

Instalarea pachetului modutils build se face conform urmatoarelor proceduri:
- se dezinstalează pachetul folosind comanda “tar –jxvfmodutils-2.4.26.tar.bz2”
- se configurează scriptul de instalare
- „../modutils-2.4.26/configure –prefix=/home/user/… --target=arm-unknown-linux-gnu”
- se exportă variabile de sistem necesare
- “Export PATH=/opt/crosstool/arm-unknown-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin:$PATH”
- se instalează ruland make apoi make install

Aplicarea patch-ului ADEOS

Patch-ul ADEOS este o secțiune de cod de nivel jos care are rolul de a face legătura între Linux şi platforma hardware pe care el rulează în cazul nostru microcontrolerul EP9301.

Aplicarea acestui patch se face astfel:

„tar –zxvf linux24-ts8-kernelsource.tar.gz”

„cd linux24”

„patch –p1 <cale-catre-adeos/ts72xx_ts8_adeos.patch”

Compilarea kernelului

Folosind noul crosscompiler obținut la una din etapele

anterioare se compilează kernelul care acum are inclus în el patch-ul ADEOS. Folosind acest crosscompiler nucleul va fi compilat pentru arhitectura ARM.

„tar –zxvf tskernel-2.4.26-ts11-src.tar.gz”

Se modifică variabila CROSS_COMPILE=arm-unknown-linux-gnu- în fisierul makefile. Se setează variabila depmod

„Export PATH=/opt/crosstool/arm-unknown-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin:$PATH”

„Make ts7200_config”

„Make oldconfig”

„Make dep”

„Make vmlinux”

„Make modules”

„Make modules_install INSTALL_MOD_PATH=/home/user/...”

Aceasta ultimă comandă crează şi modulele care vor fi folosite ulterior de kernel.

Realizarea unui SD card bootabil

Etapa presupune partiționarea unui SD card folosind utilitarul fdisk din Linux, în trei partiții. Prima partiție conține un sistem de fișiere custom care va fi folosit doar în procesul de boot-are a sistemului, partiția a doua este cea pe care kernelul va fi rezident iar cea de-a treia partiție va conține totalitatea serviciilor oferite de distribuția Debian Sarge. Din momentul în care această etapă este încheiată se poate proceda la punerea sub tensiune a sistemului. În mod normal acesta trebuie să booteze până la prompt-ul de login.

C. *REZULTATE EXPERIMENTALE*

În această secțiune sunt prezentate rezultatele cazurilor de test realizate. Primul dintre acestea se referă la testarea jitterului sistemului.

A. JITTER TEST CASE

Prin jitter se înțelege întarzierea maxima pe care un task o poate avea fața de perioada sa nominală în cazul task-urilor periodice sau față de durata așteptată pentru cele neperiodice.

Testul de jitter cazul I.

Acest test evidentiază jitterul task-ului de test când procesorul este idle în cea mai mare parte a timpului.

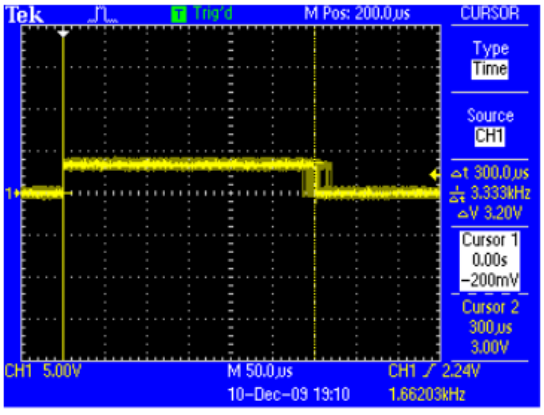

Fig. 3. Jitter-ul cand procesorul este în idle.

Fig. 3 prezintă jitterul task-ului de test. El este vizibil în partea dreaptă a formei de undă dreptunghiulare si are contrasul puțin mai scazut decât restul semnalului. Sincronizarea osciloscopului se realizează pe frontul crescător al semnalului iar măsurarea jitterului se face pe frontul căzător.

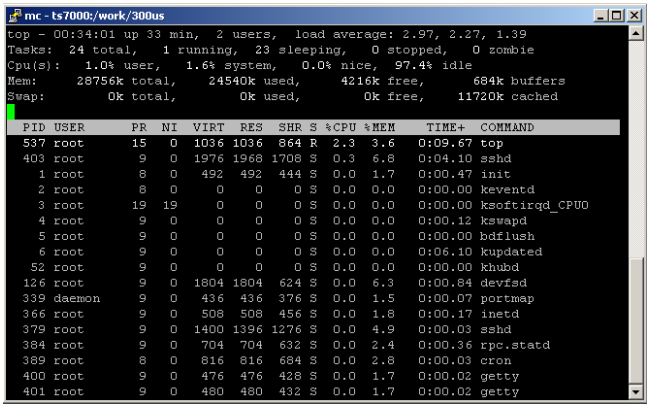

Fig. 4. Gradul de încărcare al procesorului penru cazul I.

În mod uzual când sistemul are un factor de încărcare mic al procesorului jitter-ul este şi el mic.

Testul de jitter cazul II

Acest test evidențiază jitter-ul task-ului atunci când procesorul este încărcat foarte tare.

Încărcarea procesorului am realizat-o rulând comanda „ping –f 192.168.0.14” care trimite flood spre IP-ul specificat ca parametru.

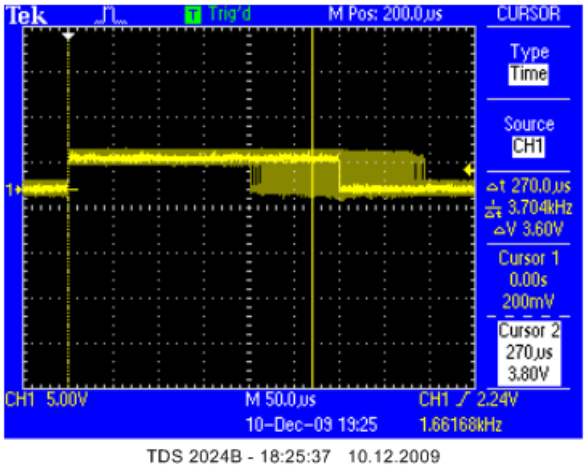

Fig.5. Jitter-ul task-ului de test pentru o încărcare maximă a procesorului.

TABEL 1. REZULTATELE CAZURILOR DE TEST PENTRU SISTEMUL PROPUS.

Test case	Min	Max
I	2us	30us
II	2us	105us

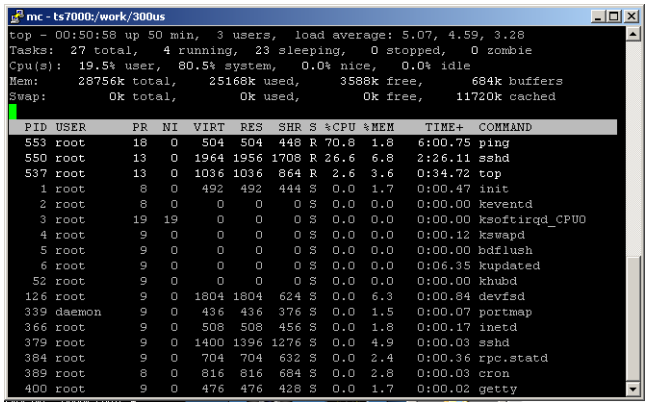

Fig. 6. Gradul de încărcare al procesorului pentru cazul II.

B. PREEMPTIVE TEST CASE

Acest caz este realizat pentru a observa dacă mediul RTAI este preemptiv.

Cazul I

În acest test sunt configurate doua task-uri pe două priorități diferite astfel: Task-ul mai prioritar generează semnalul rectangular reprezentat pe osciloscop cu albastru iar task-ul mai puțin prioritar este reprezentat cu galben. Ele sunt asignate la 2 pini de la conectorul DIO al sistemului. Fig7. Ilustrează formele de undă generate de cele doua task-uri.

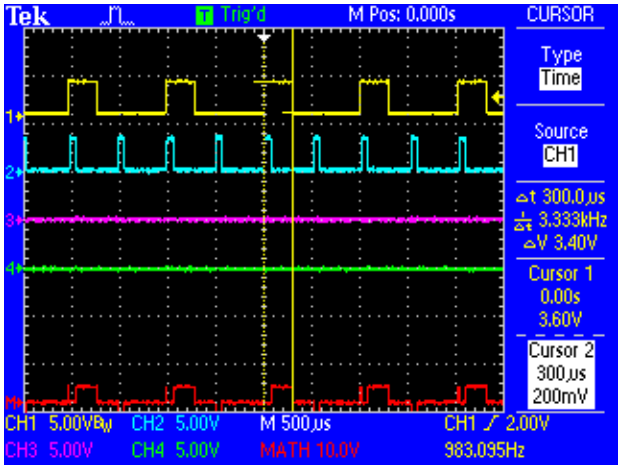

Fig. 7. Semnale rectangulare produse de doua task-uri cu priorități diferite (albastru-prioritate mare, galben prioritate mică).

Cazul II

În acest caz prioritățile sistemului se schimbă pentru a observa comportamentul preemptiv al sistemului. Aşa cum se observă din Fig. 8 task-ul al doilea este întarziat pană după terminarea celui cu prioritate mai mare.

IV. CONCLUZII

Aşa cum am menționat în partea de început a lucrării sistemele de operare de nivel înalt nu au cele mai bune caracteristici pentru a fi folosite în aplicațiile hard real-time. Cu toate acestea însă am demonstrat în acest articol că sistemul propus poate obține performanțe relativ bune comparativ cu alte sisteme similare de aceeaşi categorie. In mod uzual un dispozitiv embedded bazat pe Windows Mobile nu merge mai jos de 1ms. Rezultatele obținute dovedesc ca această limită este depaşită în cazul arhitecturii propuse spre testare, ordinul de mărime în ce priveşte jitterul fiind cu o unitate mai mic decât în cazul sistemului Windows. Având în vedere că valorile jitterului nu sunt dintre cele mai mulțumitoare putem concluziona că acest sistem nu poate fi folosit la orice cerință de timp real. Pentru sistemele soft real-time poate fi folosit însa nu şi pentru cele hard real-time.

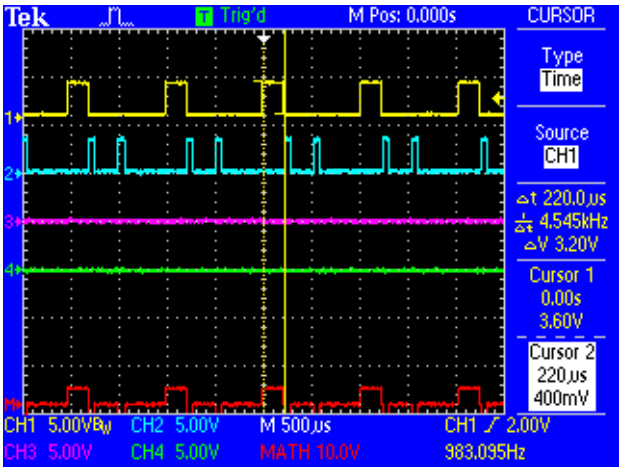

Fig. 8. Semnale rectangulare produse de doua task-uri cu prioritati diferite ce ilustreaza comportamentul preemptiv al sistemului.

REFERINȚE

[1] Francis Cottet, Joëlle Delacroix, Claude Kaiser, Zoubir Mammeri, "Scheduling In Real-Time Systems," John Wiley & Sons Ltd, England 2002 ,p19-96

[2] Guoyin Zhang, Luyuan Chen, Aihong Yao, "Study and Comparison of the RTHAL-Based and ADEOS-Based RTAI Real-time Solutions for Linux", First International Multi-Symposiums on Computer and Computational Sciences 2006

[3] Chiandone, M.; Cleva, S.; Sulligoi, G , "PC-based feedback acceleration control using Linux RTAI", 13th European Conference on Power Electronics and Applications, 2009

[4] Dozio, L.; Mantegazza, P., "Real time distributed control systems using RTAI", Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing 2003.

[5] Li Jun; Yang Fumin; Lu Yansheng, "A feasible schedulability analysis for fault-tolerant hard real-time systems", 10th IEEE International Engineering of Complex Computer Systems, 2005.

[6] Chang-Gun Lee; Joosun Hahn; Yang-Min Seo; Sang Lyul Min; Rhan Ha; Seongsoo Hong; Chang Yun Park; Minsuk Lee; Chong Sang Kim, "Enhanced analysis of cache-related preemption delay in fixed-priority preemptive scheduling", The 18th IEEE Real-Time Systems Symposium 1997.

[7] Ho, I.; Jin-Cherng Lin, "A method of test cases generation for real-time systems" , First International Symposium on Object-Oriented Real-time Distributed Computing, 1998

[8] Atif, Y.; Hamidzadeh, B.," A scalable scheduling algorithm for real-time distributed systems", 18th International Conference on Distributed Computing Systems 1998.

[9] Peter Altenbernd, "Deadline- Monotonic Software Scheduling for the CO-Synthesis of Parallel Hard Real-Time Systems", IEEE Transactions on Computers ,2004

[10] Ching-Chih Han; Kwei-Jay Lin; Chao-Ju Hou," Distance-constrained scheduling and its applications to real-time systems", IEEE Transactions on Computers , Volume 45, July 1996 p814 - 826.