Scheduling real-time tasks with nMPRA architecture for embedded applications

Ionel ZAGAN, Vasile Gheorghiţă GĂITAN

***Abstract*—The main feature of real-time embedded systems is to ensure determinism and predictability of tasks execution. The purpose of this paper is to analyze and describe the appropriate schedulability methods used for nMPRA (Multi Pipeline Register Architecture) architecture in real-time environments. Taking into consideration the requirements of hard real-time systems, the hardware scheduler engine (nHSE) must guarantee a constant scheduling frequency, providing hardware based isolation and timing predictability for hard real-time tasks.**

***Index Terms*—predictability, real-time systems, nMPRA architecture**

I. INTRODUCERE

ABORDAREA acestui subiect din punct de vedere ştiinţific şi tehnologic se datorează evoluţiilor spectaculoase din domeniul sistemelor înglobate şi aria de aplicabilitate care poate fi automotive, avionics sau robotică.

Limitările abordărilor actuale sunt date de arhitectura procesorului, memoriei, subsistemului I/O, a limbajelor de nivel înalt şi a compilatoarelor acestora precum şi un răspuns impredictibil la întreruperile asincrone.

Comportamentul impredictibil pentru cele mai multe sisteme de operare în timp real (SOTR) comerciale este dat de execuţia aceleaşi instrucţiuni într-un număr variabil de cicli. Arhitecturile obţinute trebuie să asigure planificări predictibile chiar dacă firele de execuţie vor fi încărcate până la limită. Astfel, indiferent de încărcarea unui fir de execuţie toate task-urile vor fi planificate.

În ceea ce priveşte planificatoarele de timp real, majoritatea implementărilor au în vedere obţinerea unei arhitecturi deterministe în timp şi izolarea contextelor pentru thread-urile de tip hard. Elementele de dificultate ale acestor obiective apar la nivelul SOTR, prin înglobarea funcţionalităţii acestuia în hardware şi evidenţierea îmbunătăţiri performanţelor prin programe de test adecvate.

Răspunsul nedeterminist la întreruperile externe asincrone se datorează faptului că pentru cele mai multe SOTR comerciale, execuţia aceleiaşi instrucţiuni se finalizează într-un număr variabil de cicli datorită hazardurilor şi nu numai.

Această lucrare descrie şi compară abordările actuale care au contribuit la reducerea segmentării execuţiei programelor, prezentând metode eficiente pentru minimizarea costurilor de planificare necesare pentru nHSE prin eliminarea întreruperilor inutile.

Întrebarea dacă sistemele preemptive sunt mai bune decât sistemele non-preventive a fost abordată pentru o lungă perioadă de timp. În literatura de specialitate au fost furnizate multe soluţii parţiale, unele probleme fiind încă obiectul discuţiilor. Fiecare dintre aceste soluţii are avantaje şi dezavantaje, în funcţie de predictibilitatea şi eficienţa sistemului pentru care au fost implementate.

Schimbarea contextelor este un factor cheie în algoritmii de planificare de timp real, deoarece permite sistemului de operare să aloce imediat procesorul task-urilor cu prioritate mai mare. În sistemele full-preemptive, execuţia task-ului curent poate fi întrerupt în orice moment de un alt task cu prioritate mai mare. Execuţia task-ului întrerupt se va relua doar atunci când nu mai există task-uri cu prioritate mai mare pregătite pentru execuţie. În unele implementări schimbarea contextelor poate fi complet interzisă pentru a evita interferenţele imprevizibile dintre task-uri dar şi pentru îmbunătăţirea predictibilităţii sistemului. Pentru unele sisteme de timp real, planificatorul preemptiv poate fi dezactivat doar pentru anumite intervale de timp în timpul execuţiei secţiunilor critice, cum ar fi ISR (Interrupt Service Routine).

Un dezavantaj principal al implementărilor non-preventive este acela că introduce un factor suplimentar de blocare pentru task-urile cu prioritate mare, dar cu toate acestea, există mai multe avantaje importante atunci când se adoptă acest tip de planificare.

Atunci când realizăm o analiză a sistemelor de operare, inclusiv a planificatorului acestuia, trebuie luate în considerare următoarele aspecte [1]:

- în multe situaţii practice, precum planificarea I/O sau comunicarea utilizând mediile partajate, orice întrerupere este imposibil sau foarte greu de acceptat. Aceasta deoarece, suspendarea task-ului curent va determina o creştere a efectului de *cache miss* şi o influenţă negativă asupra mecanismului *pre-fetch,* implicând un WCET impredictibil;
- în planificarea non-preemptivă, problemele introduse de excluderea mutuală sunt neînsemnate deoarece, prin natura

Manuscript received December 2, 2015.

Ionel ZAGAN is with the Computers Department, Ştefan Cel Mare University of Suceava, 13 University street, Suceava, Romania (e-mail: zagan@eed.usv.ro).

Vasile Gheorghiţă GĂITAN is with the Computers Department, Ştefan Cel Mare University of Suceava, 13 University street, Suceava, Romania (e-mail: gaitan@eed.usv.ro)

algoritmului de planificare se garantează accesul exclusiv la resursele partajate. În planificarea preemptivă însă, pentru a garanta accesul la resursele partajate evitând fenomenul de inversare a priorităților, este necesar implementarea unor protocoale complexe specifice mecanismelor de control al resurselor partajate;

- în sistemele de timp real hard cu planificare non-preemptivă efectul de jitter este minim pentru toate task-urile din sistem, simplificându-se astfel tehnicile de control pentru compensarea și diminuarea efectelor negative datorate întârzierilor;
- execuția non-preemptivă permite folosirea tehnicilor de partajare a stivei pentru a salva spațiu de memorie din sistemele înglobate de mici dimensiuni.

Planificatoarele preemptive introduc fluctuații pentru timpii de execuție a task-urilor, degradând astfel predictibilitatea sistemului. În procesul de proiectare al acestor tipuri de planificatoare trebuie avut în vedere existența unor costuri introduse de:

- planificare – reprezintă timpul consumat de algoritmul de planificare;
- pipeline – însumează timpul datorat ciclilor de ceas pierduți de instrucțiunile care au fost deja extrase și decodificate, deoarece pe banda de asamblare trebuie introduse instrucțiunile noului task, timpul necesar introducerii noului task pe banda de asamblare și timpul datorat refacerii benzii de asamblare pentru task-ul întrerupt în momentul în care acesta își reia execuția;
- cache-related – reprezintă timpul necesar pentru a încărca liniile cache pierdute în momentul schimbării contextului;
- bus-related – reprezintă timpul introdus de operațiile de acces la memoria RAM, datorate efectului de *cache miss*.

Însumarea tuturor acestor timpi, sau doar o parte din aceștia, reprezintă *Architecture related cost*, acest cost fiind caracterizat de o variație semnificativă în funcție de punctele în care are loc schimbarea de context.

În analiza algoritmilor de planificare trebuie să se țină cont de aspecte precum, complexitatea implementării, eficacitatea schemei de planificare și predictibilitatea în estimarea coeficientului *Architecture related cost*.

II. Planificarea preemptive și non-preemptive cu algoritmul Deadline Monotonic

În această lucrare, se va nota cu n numărul task-urilor periodice sau sporadice ce se vor planifica pe un singur procesor. Fiecare task τ_i este caracterizat de un WCET (Worst Case Execution Time) notat cu C_i, un deadline D_i și o perioadă de execuție T_i. Este fixat un model de deadline, care obligă ca D_i să fie mai mic sau egal decât T_i.

În ceea ce privește planificarea, fiecărui task τ_i îi este asignată o prioritate P_i, folosită pentru a selecta care dintre task-urile gata de execuție poate fi planificat. O valoare mai mare pentru P_i reprezintă o prioritate mai mare a task-ului respectiv, acestea fiind ordonate în ordine descrescătoare, astfel încât: $\forall i \mid 1 \le i < n : P_i > P_{i+1}$.

Considerăm că timpii de activare sunt fixați la momentul proiectării iar timpul de execuție al task-urilor trebuie să fie mai mic sau egal decât valoarea C_i.

Pentru a exemplifica aceste notații, propunem un set de trei task-uri cu parametrii relativi acestora din Tabelul I.

TABEL I
PARAMETRII UNUI SET DE TASK-URI

	T_i	C_i	D_i
τ_1	7	2	6
τ_2	12	3	10
τ_3	22	7	17

În Fig. 1 este reprezentată planificarea preemptivă a setului de task-uri din Tabelul I prin intermediul algoritmului Deadline Monotonic. Se poate observa că planificarea acestui set de task-uri nu este una optimă, deoarece task-ul τ_3 nu își atinge termenul limită.

Prin folosirea metodei de planificare non-preemptive se elimină total schimbările de context, și implicit se micșorează coeficientul *Architecture related cost*. O analiză de fezabilitate pentru un set de task-uri non-preemptiv este mai dificil de realizat deoarece este necesară o analiză pe o perioadă mai lungă de timp.

Bril și alții [2] au demonstrat că în planificarea non-preemptivă, WCET-ul unui task nu este obligatoriu să apară în prima perioadă de execuție.

Deoarece execuția task-urilor cu prioritate mare este întârziată, apare o anomalie de planificare denumită *self-pushing phenomenon* conform căreia nu se îndeplinesc deadline-urile impuse. De aceea, este necesară o analiză pe o perioadă mai lungă de execuție, denumită L_i (*Level-i Active Period* definit în [2]), cel puțin până când task-ul τ_i cu prioritatea P_i își termină execuția.

Yao, Buttazzo, și Bertogna au arătat în [3] că analiza task-urilor non-preemptive poate fi redusă la un singură perioadă, respectând următoarele condiții:

- setul de task-uri este fezabil cu planificarea preemptive;
- deadline-urile relative sunt mai mici sau egale cu perioadele.

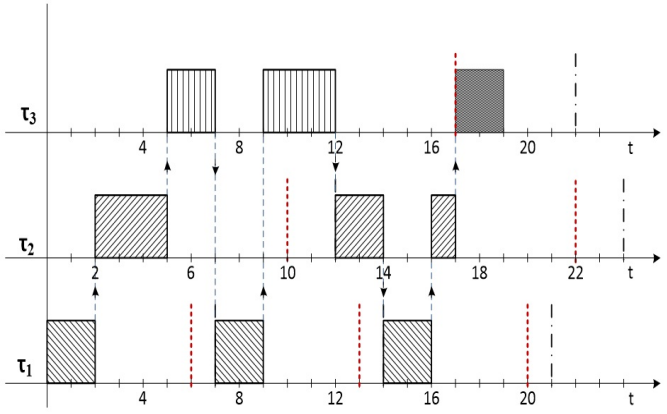

Fig. 1. Planificarea preemptive a task-urilor din Tabelul I cu algoritmul Deadline Monotonic

III. MODELUL PREEMPTION THRESHOLDS

Această idee de planificare a fost propusă pentru prima dată de Wang și Saksena în [4]. Conform aceastei abordări, fiecărui task τ_i îi este atribuită o prioritate normală P_i și un prag de preemțiune $\theta_i \geq P_i$ astfel încât, un task poate dezactiva sistemul preemptive până la un anumit prag de preemțiune θ_i. Așadar, o schimbare de context poate avea loc doar dacă prioritatea noului task P_j este mai mare decât pragul de preemțiune θ_i al task-ului aflat în execuție τ_i. Această metodă de planificare este un compromis între planificarea full preemptivă și cea full non-preemptivă. Este firesc deoarece, dacă fiecare prag de prioritate este considerat ca fiind egal cu prioritatea nominală a task-ului, planificatorul se comportă precum algoritmul full preemptiv, în schimb dacă toate pragurile de prioritate sunt setate ca prioritatea maximă din sistem planificatorul devine unul non-preemptive.

Wang și Saksena au demonstrat că prin setarea corespunzătoare a pragurilor de prioritate se poate obține o eficiență bună a schemei de planificare și implicit un grad mai mare de utilizare a procesorului [1].

IV. MODELUL DEFERRED PREEMPTIONS

Această metodă implică stabilirea pentru fiecare task τ_i a celui mai lung interval q_i care poate fi executat non-preemptive.

În funcție de câte intervale non-preemptive sunt implementate, acest model poate fi implementat în două variante diferite. Primul model se numește *Floating* și implică, definirea de către programator a regiunilor non-preemptive prin intermediul unor primitive introduse în codul task-ului pentru dezactivarea și activarea întreruperilor. Modelul *Activation-triggered* este caracterizat de faptul că regiunile non-preemptive sunt declanșate de apariția unui task cu prioritate mai mare. În ambele cazuri, intervalele non-preemptive nu sunt cunoscute în momentul proiectării, și pentru motive de analiză se consideră cele mai defavorabile cazuri.

În exemplul din Fig. 2. se ilustrează planificarea Deadline Monotonic aplicând metoda Deferred Preemptions pentru același set de task-uri din Tabelul I.

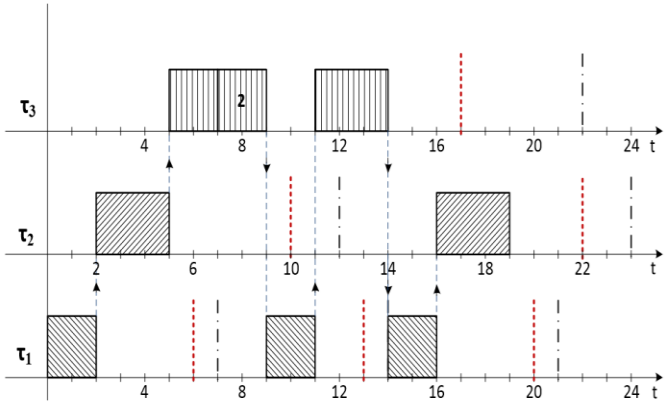

Fig. 2. Planificarea Deadline Monotonic cu modelul *deferred preemptions* pentru setul de task-uri din Tabelul I

Considerând că $q_3 = 2$, planificarea setului de task-uri este fezabilă, observându-se perioada non-preemptive până la schimbarea contextului în favoarea unui task mai prioritar.

V. MODELUL TASK SPLITTING

Această metodă a fost prezentată în detaliu de Burns în [5], numindu-se și *Cooperative scheduling* deoarece task-urile cooperează pentru a oferi puncte de preemțiune adecvate, garantând o planificare optimă.

Conform acestei abordări, un task τ_i se execută în modul non-preemptive iar preempțiunile sunt permise numai în anumite puncte predefinite, numite *preemption points*. Pentru aceasta, task-ul τ_i este împărțit într-un număr de m_i segmente non-preemptive prin intermediul unor algoritmi bine definiți, obținându-se în final $m_i - 1$ puncte de preempțiune.

Dacă un task cu prioritate mai mare ajunge în starea pregătit pentru execuție între două puncte de preempțiune, întreruperea task-ului curent se va face la următorul punct de preemțiune [5].

Pentru a exemplifica acest model am considerat același set de task-uri din Tabelul I, presupunând că τ_3 este împărțit în două subjob-uri de 5 respectiv 2 unități. Planificarea produsă de Deadline Monotoic cu metoda *task splitting* este fezabilă, fiind ilustrată în Fig. 3.

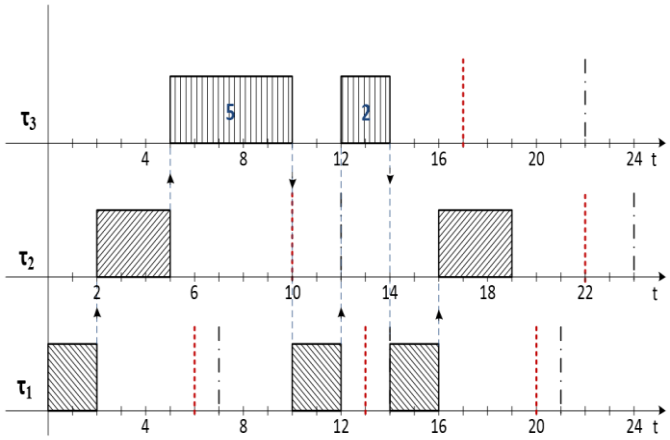

Fig. 3. Planificarea Deadline Monotonic cu modelul *task splitting* pentru setul de task-uri din Tabelul I

În situația în care un set de task-uri nu este planificabil în modul non-preemptiv, există câteva moduri de a împărți job-urile în subjob-uri pentru a obține o planificare fezabilă, dacă aceasta există.

Așadar, este foarte important să se identifice cele mai bune locații pentru fiecare task τ_i unde să fie introduse punctele de preemțiune, deoarece există secțiuni precum operații I/O, secțiuni critice sau bucle scurte, unde nu se dorește întreruperea task-ului.

După o analiză de fezabilitate pentru un set de task-uri, trebuie să rezulte o schemă de planificare optimă cu costuri de preemțiune minime.

VI. Îmbunătățirea Performanțelor Arhitecturii nMPRA prin reducerea Jitter-ului

Arhitectura nMPRA propusă de Găitan și alții în [6] este o implementare scalabilă, ușor extensibilă. Procesorul utilizează o linie de asamblare cu cinci etaje pentru îmbunătățirea performanțelor de calcul. Acest lucru permite rularea a până la cinci instrucțiuni în banda de asamblare, pentru ca raportul de execuție să fie în final de o instrucțiune/ciclu de ceas [7].

Implementarea în hardware a schemelor de planificare și reducerea semnificativă a timpilor de comutare au condus la obținerea unei arhitecturi competitive și inovatoare. Folosind o schemă de planificare de tip round-robin, arhitectura nMPRA introduce un jitter de maxim trei cicluri de ceas, fiind o implementare deterministă datorită planificatorului hardware nHSE [8]. Întârzierea maximă a planificatorului nHSE la frecvența de lucru de 10MHz este de 340ns.

Aceste teste practice efectuate în scopul validării și implementării procesorului nMPRA au fost realizate folosind un FPGA Virtex6 pe o placa de referință ML605 produsă de Xilinx.

VII. Concluzii

În urma acestor studii de fezabilitate pentru determinarea schemei de planificare ideale corespunzătoare unui sistem de timp real hard, putem trage următoarele concluzii.

Modelul de planificare *Preemption Thresholds* poate reduce numărul schimbărilor de context, cu toate că există dezavantajul costului de preemțiune ce nu poate fi ușor de estimat deoarece nu se poate calcula cu exactitate numărul de schimbări de context pentru fiecare task.

Folosind modelul de planificare *deferred preemptions* numărul de schimbări de context pot fi estimate cu exactitate, doar că punctele în care au loc acestea nu pot fi știute off-line.

Modelul de planificare cooperativ *Task splitting* este cel mai predictibil mecanism pentru estimarea costului de preemțiune, deoarece se poate estima cu exactitate atât numărul cât și punctele în care au loc schimbările de context. Implementarea acestei scheme de planificare se face prin inserarea unor funcții sistem în codul sursă, lucru ce implică mărirea jitter-ului prin inserarea unui timp adițional.

Bibliografie

[1] Giorgio C. Buttazzo, ”Hard Real-Time Computing Systems” - Predictable Scheduling Algorithms and Applications, Third edition, 2011.

[2] R. J. Bril, J. J. Lukkien, andW. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. Real-Time System, 42(1-3):63–119, 2009.

[3] G. Yao, G. C. Buttazzo, and M. Bertogna. Feasibility analysis under fixed priority scheduling with fixed preemption points. In Proc. of the 16th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA’10), pages 71–80, Macau, SAR, China, August 23-25, 2010.

[4] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In Proc. of the 6th IEEE Int. Conference on Real-Time Computing Systems and Applications (RTCSA’99), pages 328–335, Hong Kong, China, December 13-15, 1999.

[5] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. S. Son, editor, Advances in Real-Time Systems, pages 225–248, 1994.

[6] Gaitan, V.G.; Gaitan, N.C.; Ungurean, I., "CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers," in Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.23, no.9, pp.1661-1674, Sept. 2015.

[7] E. Dodiu and V.G. Gaitan, “Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and theory of operation,“ 2012 IEEE EIT International Conference on Electro-Information Technology, Indianapolis, IN, USA, 6-8 May 2012, ISBN: 978-1-4673-0818-2, ISSN: 2154-0373.

[8] Gaitan, Nicoleta Cristina; Zagan, Ionel and Gaitan, Vasile Gheorghita, “Predictable CPU Architecture Designed for Small Real-Time Application - Concept and Theory of Operation” International Journal of Advanced Computer Science and Applications(IJACSA), 6(4), 2015, DOI: 10.14569/IJACSA. 2015. 060406, U.S ISSN: 2156-5570(Online), pp. 47-52.