Universitatea Politehnica București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Tema de curs

Sisteme de Operare Avansate

PLANIFICATORUL DE PROCESE IN REAL-TIME LINUX

Coordonator:
Conf. dr. ing. Ştefan Stăncescu
Master IISC, anul I

Student:
Jerca Constantin

2016 - 2017

Cuprins

1. INTRODUCERE

Sistemele în timp real au evoluat de-a lungul ultimelor decenii, într-o manieră relativ calmă. Performanţa a crescut, se poate spune, în mod dramatic, dar principalele paradigme au fost destul de stabile de la mijlocul anilor `80. Acest lucru se schimbă în ultimii ani. Marea schimbare aparută în domeniul embedded este folosirea mai multor procesoare, ce duce în destul de multe cazuri la o reevaluare a metodelor vechi si a modului de a gândi în timp real.

Un sistem de calcul în timp real este capabil să execute foarte fiabil programe cu cerințe de sincronizare foarte specifice, lucru important pentru multe proiecte științifice și de inginerie. Componenta cheie necesară pentru a construi un sistem în timp real este un sistem de operareîn timp real; alte componente hardware și software de piese care alcătuiesc un întreg sistem în timp real sunt discutate în secțiunea următoare.

Cu toate că componenta principală necesară pentru crearea unui sistem în timp real este RTOS, sunt necesare mai multe piese de software și hardware pentru a construi un sistem în timp real de la început până la sfârșit.

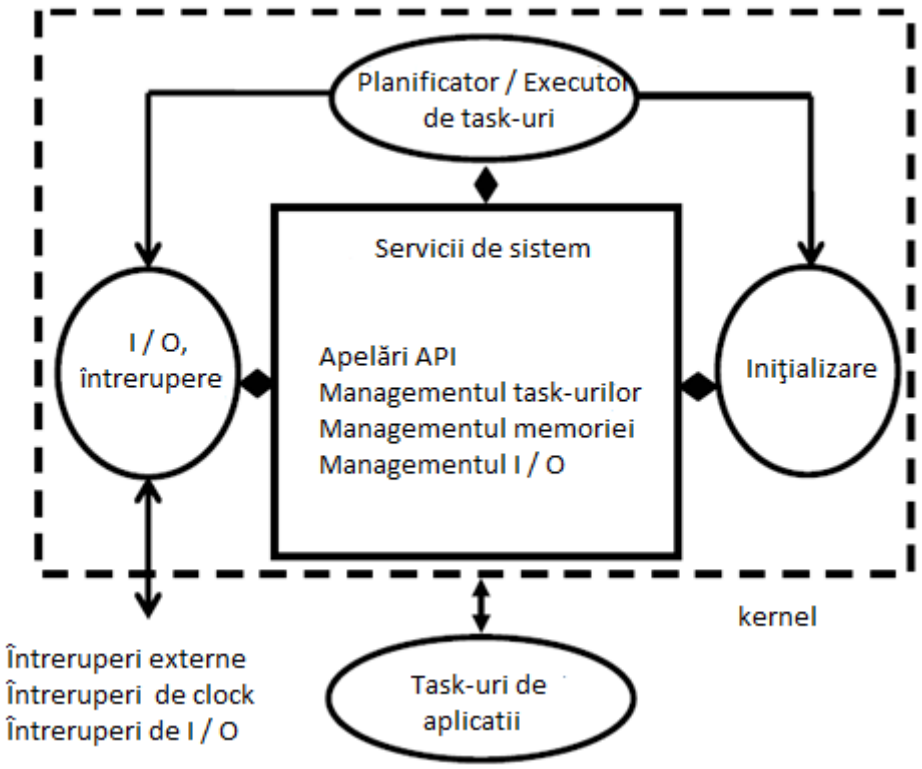

Figura 1.1 Structura unui kernel RTOS [1]

Linux are o poziție puternică în toate tipurile de sisteme embedded, variind de la produse electronice de consum la dispozitive de rețea și o gamă largă de aplicații industriale, inclusiv sistemele legate de securitate. Resursele tehnologice potrivite pentru disponibilitate ridicată, în timp real, și siguranța sistemelor critice au fost în continuă expansiune și îmbunătățire. La baza acestei evoluții este disponibilitatea sistemelor de operare stabile cu proprietăți de încredere în timp real. Extinderea și îmbunătățirea proprietăților în timp real Open Source RTOS este un efort de cercetare și dezvoltare continuă.

2. PLANIFICATORUL DE PROCESE

Pentru un set dat de lucru, problema generală de planificare solicită o ordine în funcție de care procesele urmează să fie executate astfel încât diferitele constrângeri sunt îndeplinite. De obicei, un proces în timp real se caracterizează prin timpul de execuție, timpul de gata, limita de timp și cerinţele privind resursele. Executarea unui proces poate sau nu poate fi întreruptă (planificarea preemptivă sau non-preemptivă). Peste setul de procese, există o relație de prioritate care limitează ordinea de execuție. Executarea unui proces nu poate începe până la executarea tuturor predecesorilor săi (în funcție de relația de prioritate) este finalizată. Sistemul pe care procesele urmează să fie executate este caracterizat prin cantitatea de resurse disponibile.

Practic, problema de planificare este de a stabili un program pentru executarea proceselor, astfel încât toate acestea sunt finalizate înainte de termenul limită de ansamblu.

Având în vedere un sistem în timp real, abordarea de planificare corespunzătoare ar trebui să fie proiectată în funcție de proprietățile sistemului și sarcinile care apar în ea.

RTOS-urile furnizează funcții pentru a crea, inițializa și activa task-uri noi. Ele oferă funcții pentru a aduna informații cu privire la activitățile existente în sistem, pentru numirea task-ului, verificarea stării unuitask dat, opțiuni pentru executarea task-ului, cum ar fi utilizarea de co-procesor, modele specifice de memorie, precum și stergerea task-ului. Ștergerea necesită adesea măsuri speciale de precauție, în special în ceea ce privește semafoarele, pentru task-urile de memorie partajată.

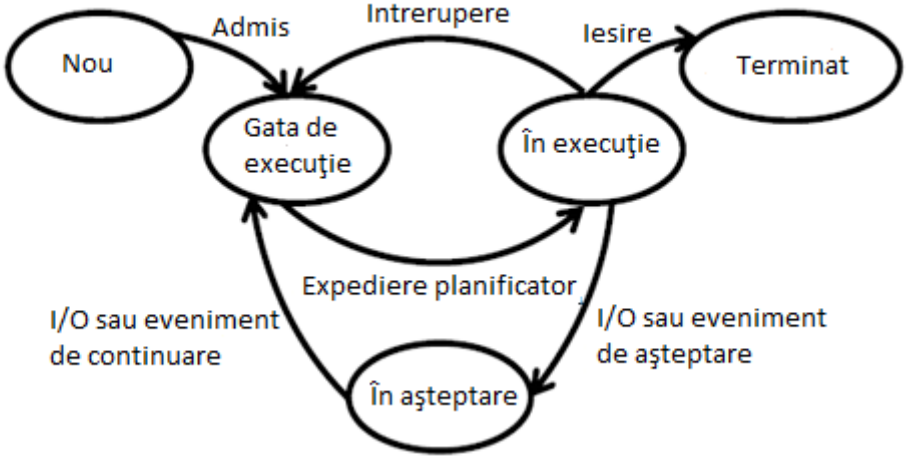

Figura 2.1 Diferitele stări in care un task poate fi în timpul ciclului său de viață de execuție în cadrul unui RTOS Task State Transitions [1]

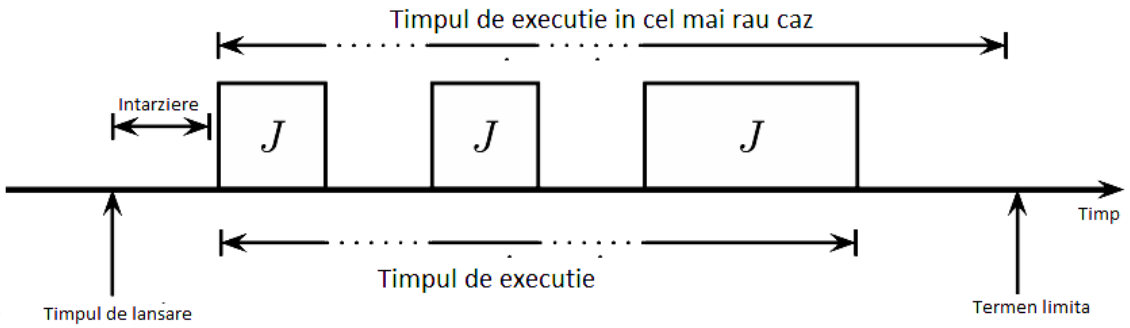

Figura 2.2 Proprietățile de timp ale unui proces în timp real

Un RTOS ar trebui să fie în măsură să identifice în mod dinamic task-ul cu cel mai vechi deadline. Pentru a se ocupa de termenele limită, planificatorul poate converti termenele limită la niveluri de prioritate, care sunt utilizate pentru alocarea resurselor. Această tehnică este folosită dtorită lipsei unei soluții mai bune și dă eroari mai puține.

Atunci când se utilizează planificarea task-urilor prioritizate, RTOS trebuie să aibă un număr suficient de nivele de prioritate pentru implementarea efectivă. Inversiunea priorităţilor are loc atunci când un task cu prioritate mai mare trebuie să aștepte un task cu prioritate mai mică pentru a elibera o resursă și la rândul său, task-ul prioritar inferior este în așteptarea unui task cu prioritate medie. Două rezolvări care se ocupă cu inversiunea priorităţilor, și anume moștenirea prioritară și protocolul de moştenire a priorităţii, au nevoie de un nivel de prioritati suficient.

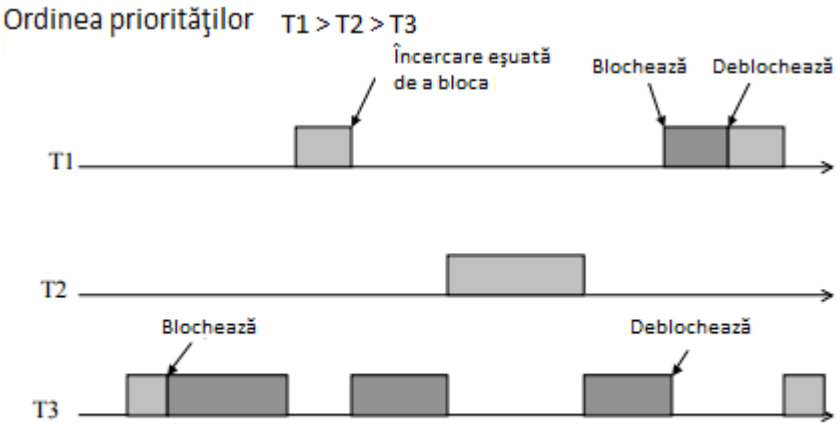

Figura 2.3.1 Problema inversării de prioritate:
Procesul T2 cauzează întârzierea procesului T1 cu o prioritate mai mare [3]

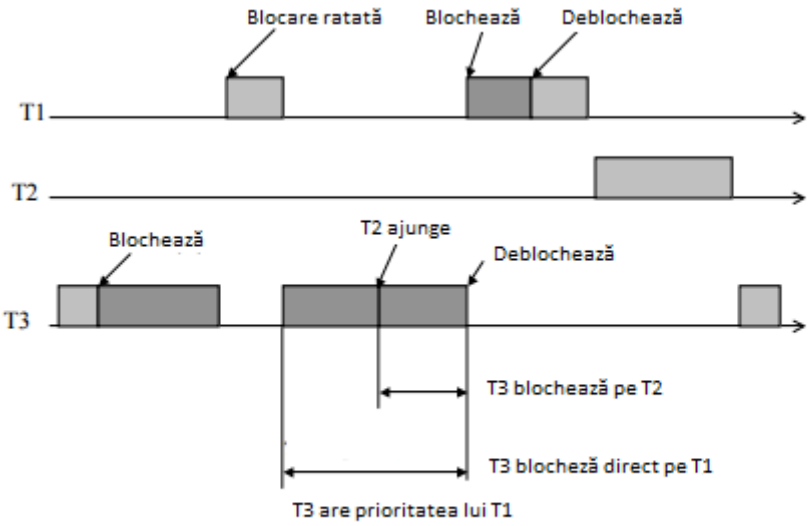

Figura 2.3.2 Problema inversării de prioritate:
Protocolul de moştenire a priorităţii [3]

Având în vedere un sistem în timp real, un proces care urmează să înceapă executarea poate solicita să primească informațiile furnizate de către un alt proces al sistemului. Prin urmare, executarea unui proces trebuie începută după terminarea execuției celuilalt proces. Acesta este conceptul de dependență. Procesele dependente folosesc memoria partajată sau datele de comunicare pentru a transfera informațiile generate de un alt proces. În timp ce decidem despre planificarea unui sistem în timp real care conține unele procese dependente, ar trebui să luam în considerare ordinea timpului de pornire și de terminare a proceselor.

3. PLANIFICATORUL DE PROCESE IN REAL-TIME LINUX

Planificatorul este componenta de nucleu care decide care proces va fi următorul executat de către CPU. Fiecare proces are o politică de planificare asociată și o prioritate de planificare statică, *sched_priority*. Planificatorul ia decizii bazate pe cunoașterea politicii de planificare și de prioritatea statică a tuturor proceselor de pe sistem.

Pentru procesele programate sub una dintre politicile de planificare normale **(OTHER, IDLE, BATCH)**, prioritatea nu este utilizată în luarea deciziilor de planificare (aceasta trebuie să fie specificată ca 0). Aceste politici nu pot fi considerate în evaluarea real time. [5]

Procese programate sub una dintre politicile în timp real **(FIFO, RR)** au o valoare de prioritate în intervalul de la 1 (scăzut) la 99 (ridicat). După cum arată şi numărul, fire de execuţie în timp real au întotdeauna prioritate mai mare decât firele normale. POSIX.1 necesită o implementare pentru a sprijini doar un minim de 32 nivele de prioritate distincte pentru politicile în timp real. Programele portabile ar trebui să utilizeze *sched_get_priority_min(2)* și *sched_get_priority_max (2),* pentru a găsi gama de priorități acceptate pentru o anumită politică. [5]

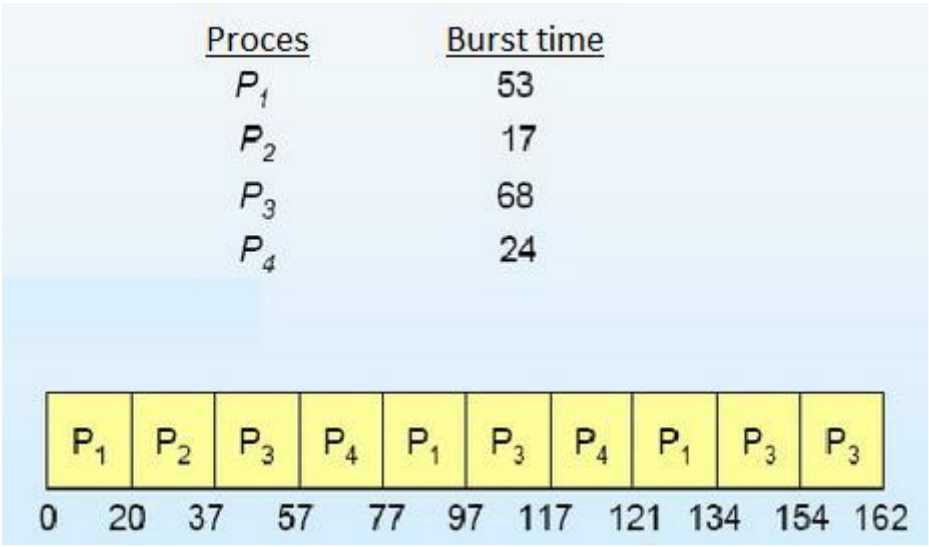

Figura 3.1 Exemplu de planificare cu algoritmul RR [2]

Conceptual, planificatorul păstrează o listă de procese executabile pentru fiecare posibilă valoare a priorităţii. Pentru a determina care fir de execuţie rulează următorul, planificatorul caută în lista valoarea nevidă cu cea mai mare prioritate statică și selectează firul de execuţie de la capul acestei liste.

Politica de planificare a unui fir de execuţie determină unde va fi introdus acesta în lista de fire de execuţie cu prioritate statică egală și modul în care se va deplasa în interiorul acestei liste.

Toată planificarea este preemptivă: în cazul în care un fir de execuţie cu o prioritate mai mare statică devine gata pentru a rula, firul care rulează în prezent va fi preempted și va reveni la lista de așteptare pentru nivelul de prioritate statică. Politica de planificare determină ordonarea numai în lista de fire de execuţie executabile cu prioritate statică egale.

4. LINUX REAL-TIME PREEMPT

De la versiunea de kernel 2.6.18 mai departe, Linux-ul devine treptat echipat cu capabilități în timp real, dintre care majoritatea sunt derivate din fostele patch-uri *realtime-preempt* dezvoltate de Ingo Molnar, Thomas Gleixner, Steven Rostedt, și alţii. Până când ele vor fi complet fuzionate în nucleul principal, acestea trebuie să fie instalate pentru a obține cele mai bune performanțe în timp real. [6]

Fără patch-uri și înainte de includerea lor deplină în nucleu principal, configurația nucleului oferă doar trei clase de preempțiune, care prevăd reducerea considerabilă a celui mai rău caz de latență de planificare:

CONFIG_PREEMPT_NONE ..... deloc
CONFIG_PREEMPT_VOLUNTARY ..... putin
CONFIG_PREEMPT_DESKTOP ..... considerabil

Cu patch-urile aplicate sau după includerea lor completă în nucleu principal, elementul de configurare suplimentar CONFIG_PREEMPT_RT devine disponibil. Dacă această opțiune este selectată, Linux este transformat într-un sistem de operare în timp real. Politicile de planificare FIFO și RR sunt apoi folosite pentru a rula un thread cu prioritate adevărată în timp real și o latență de planificare în cel mai rău caz minimă. [4]

Scopul patch-uri PREEMPT_RT este de a minimiza cantitatea de cod a nucleului care este nonpreempted, minimizând în același timp, cantitatea de cod care trebuie modificată în scopul de a oferi opţiunea de preempted. Secţiunile care sunt în mod normal preempted sunt secţiunile de tratare a întreruperilor sau cele de anulare a întreruperilor. Patch-ul PREEMPT_RT valorifică capacitățile SMP ale nucleului Linux pentru a adăuga acest atribut de preempţiune în plus, fără a necesita o rescriere completă a nucleului. Într-un sens, se poate gândi vag de preempţiune ca adăugarea unui nou procesor la sistem, și apoi utilizarea primitivelor de blocare normală pentru a se sincroniza cu orice acțiune întreprinsă de procesul preempted.

Patch-ul PREEMPT-RT convertește Linux într-un nucleu complet preempted. El realizează acest lucru făcând:

- Efectuarea primitivele de blocare din nucleu preemptibile (folosind spinlocks), prin reimplementarea cu rt_mutexes. [9]
- Secţiunile critice protejate prin spinlock_t și rwlock_t sunt acum preempted. Crearea de secțiuni de bază nonpreempted este încă posibilă cu raw_spinlock_t (aceleași API-uri, cum ar fi spinlock_t). [9]
- Punerea în aplicare a moștenirilor prioritare pentru spinlock-uri și semafoare în nucleu. [9]
- Conversia tratării întreruperilor din contextul nucleului în contextul firelor de execuţie: patch-ul PREEMPT-RT tratează întreruperile soft în

contextul firelor de execuţie, care este reprezentat de un *task_struct* ca un proces comun în spațiu utilizatorului. Cu toate acestea este de asemenea posibil să se înregistreze o întrerupere în contextul nucleului. [9]

Aproape toate rutinele de tratare a întreruperilor rulează în contextul unui proces în mediul PREEMPT_RT. Cu toate că orice întrerupere poate fi marcată SA_NODELAY pentru a face ca acesta să ruleze în contextul întreruperii, numai întreruperile fpu_irq, irq0, irq2 și lpptest au SA_NODELAY specificat. Dintre acestea, numai irq0 (întreruperea timer-ului per-CPU) este utilizată în mod normal. fpu_irq este pentru întreruperile de coprocesor în virgulă mobilă și lpptest este utilizată pentru benchmarking-ul latențelor întreruperilor. Rețineți că temporizatoarele software (add_timer () și friends) nu se execută în contextul întreruperilor hardware, în schimb, se execută în contextul proceselor și sunt pe deplin prioritare. [7]

Pentru a se reduce perioada de latenţă, există câteva modificări în PREEMPT_RT al căror scop principal este de a reduce sau de a intrerupe programarea latenta.

Prima astfel de schimbare implică hardware-ul x86 MMX / SSE. Acest hardware este manipulat în nucleu cu preempted dezactivat, iar acest lucru înseamnă că, uneori, trebuie să se aștepte până când precedente instrucțiuni MMX / SSE se execută complet. Unele instrucțiuni MMX / SSE nu sunt nicio problemă, dar altele durează foarte mult, de aceea PREEMPT_RT refuză să le folosească pe cele lente. [7]

A doua modificare se aplică la tabloul de repartiţie al variabilelelor per-CPU, ca o alternativă la dezactivarea anterioară a întreruperilor.

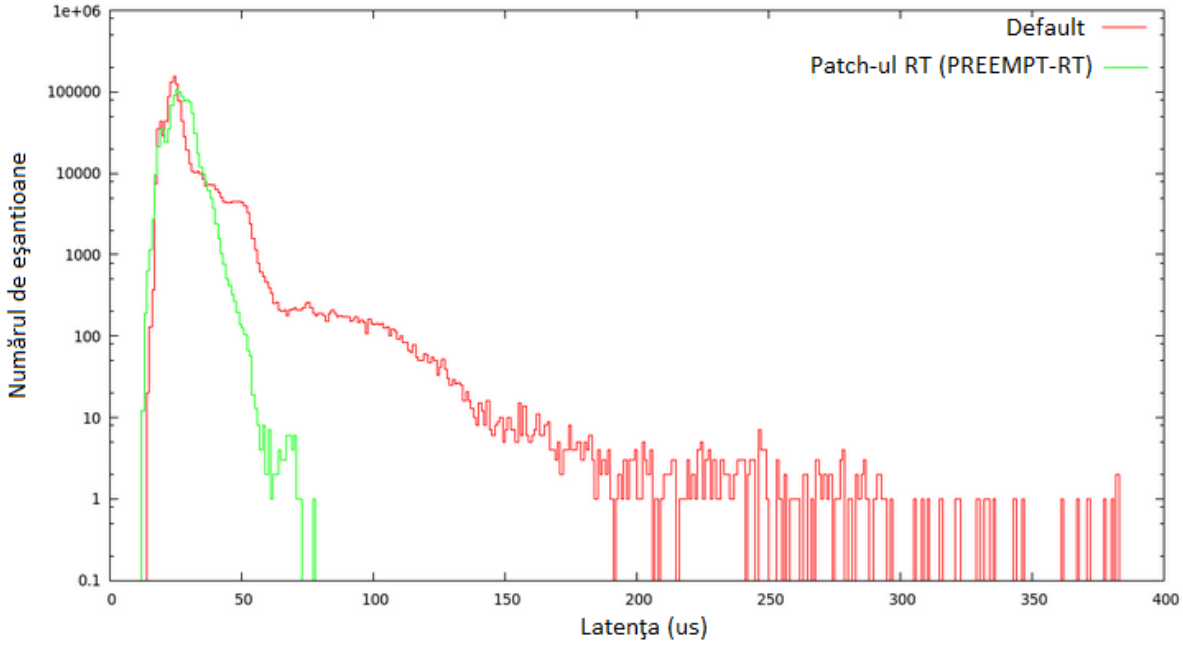

Figura 4.1 Influenţa numărului de eşantioane asupra latenţei
în cazul Linux preemted default şi cazul patch-ului PREEMPT-RT [8]

5. FUNCTII API ALE PLANIFICATORULUI REAL-TIME LINUX

sched_setscheduler (2)	Setează algoritmul de planificare și parametrii firului de execuţie specificat.
sched_getscheduler(2)	Returnează algoritmul de planificare a firului de execuţie specificat.
sched_setparam(2)	Setați parametrii de planificare a firului de execuţie specificat.
sched_getparam(2)	Returnează parametrii de planificare a firului de execuţie specificat.
sched_get_priority_max(2)	Returnează prioritatea maximă disponibilă în algoritmul de planificare specificat.
sched_get_priority_min(2)	Returnează prioritatea minimă disponibilă în algoritmul de planificare specificat.
sched_rr_get_interval(2)	Returnează valoarea cuantei de timp utilizată pentru firele de execuţie care sunt planificate cu algoritmul RR.
sched_yield(2)	Cauzează apelantul să renunțe la CPU, astfel încât un alt fir să fie executat.
sched_setaffinity(2)	(Linux specific) Setează afinitatea de procesor a unui fir de execuţie specific.
sched_getaffinity(2)	(Linux specific) Returnează afinitatea de procesor a unui fir de execuţie specific.
sched_setattr(2)	Setați politica de planificare și parametrii unui fir de execuţie specificat. Acest (Linux specific) apel de sistem oferă un superset al funcționalității sched_setscheduler (2) și sched_setparam (2).
sched_getattr(2)	Returnează politica de planificare și parametrii unui fir de execuţie specificat. Acest (Linux specific) apel de sistem oferă un superset al funcționalității sched_setscheduler (2) și sched_setparam (2).

Tabel 5.1 API-ul planificatorului Linux [4]

6. BIBLIOGRAFIE

[1] Liu, Jane W.S., Real-time systems, Upper Saddle River, NJ: Prentice Hall, 2000
[2] Kevin Churnetski, Real-time scheduling algorithms, task visualization, Rochester Institute of Technology, 2006
[3] Insup Lee, OS Overview – Real-Time Scheduling, CSE 480/CIS 700, University of Pennsylvania, 2006
[4] Michael Kerrisk, man7.org, http://man7.org/linux/man- pages/man7/sched.7.html , accesat la data de 1.09.2017
[5] Embedded Market Study, UBM Tech Electronics, 2014 http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf
accesat la data 1.09.2017
[6] Luotao Fu, Robert Schwebel, Kernel Development Group, https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO accesat la data 1.09.2017
[7] Paul McKenney, "A realtime preemption overview", LWN.net, August 10, 2005
[8] EMLID, Raspberry PI Real-Time Kernel, APM, NEWS, TUTORIALS, https://emlid.com/raspberry-pi-real-time-kernel/ accesat la data de 30.06.2016
[9] Theodore Ts'o, Darren Hart, Jekacur, https://rt.wiki.kernel.org/ , accesat la data de 1.09.2017