6 ALGORITMI DE PLANIFICARE A TASK-URILOR ÎN SCTR CU CONSTRÂNGERI DE TIMP RIGIDE

6.1 Noțiuni introductive

Problema planificarii task-urilor în prezența constrângerilor de timp rigide poate fi formulata astfel:

Fiind dat un set de task-uri precum si *constrângerile de precedenț*a*, cerințe de resurse* si *constrângerile de timp asociate lor, trebuie g*asit*a o soluție pentru planificarea task-urilor pe un calculator dat.*

Task-uri

În mod formal, *task*-urile sunt entitați de programare care consuma resurse (timp procesor, memorie, date de
intrare etc.) pentru a obține unul sau mai multe rezultate (iesiri).

Un task poate sa fie *periodic (ciclic)*, *sporadic*, sau *aperiodic*. Un task *Ti* este *periodic* daca este lansat periodic, de exemplu la intervale de *Pi* secunde. *Pi* este numita *perioada taskului Ti.* Constrângerea de periodicitate necesita ca task-ul sa se execute în mod obligatoriu si o singura data la fiecare perioada; nu se cere ca timpul de execuție al task-ului sa aiba aceeasi durata cu perioada. Un task *Ti* este *sporadic dac*a nu este periodic, dar poate fi invocat la intervale de timp neregulate. Task-urile sporadice sunt caracterizate printr-o limita superioara a frecvenței la care ele pot fi invocate. Acest lucru este de obicei specificat prin cerința ca invocarile succesive ale unui task sporadic *Ti* sa fie separate în timp de cel puțin *t(i)* secunde. Taskurile sporadice sunt uneori trecute în categoria task-urilor aperiodice. Totusi, task-urile *aperiodice* sunt acele taskuri care nu sunt periodice si care nuau o limita superioara a frecvenței de invocare.

*Constrângeri de precedenț*a

Între task-uri pot exista *constrângeri de precedenț*a, pentru a specifica daca unele trebuie sa fie executate înaintea altora. De exemplu daca iesirea taskului *Ti* trebuie folosita ca intrare de taskul *Tj*, atunci taskul *Tj* esteconstrâns sa fie precedat de taskul *Ti*. Constrângerile de precedența ale unui set de task-uri pot fi reprezentate prin *graful de precedenț*a.

Un exemplu de graf de precedența este în figura 6-1 Sagețile indica precedența între task-uri.

Se noteaza mulțimea taskurilor care preced taskul *T* prin ←*(T)*, ←*(T)* indicând care taskuri trebuie sa-si termine execuția înaintea declansarii task-ului *T*.

Exemplul

În figura 6-1 avem

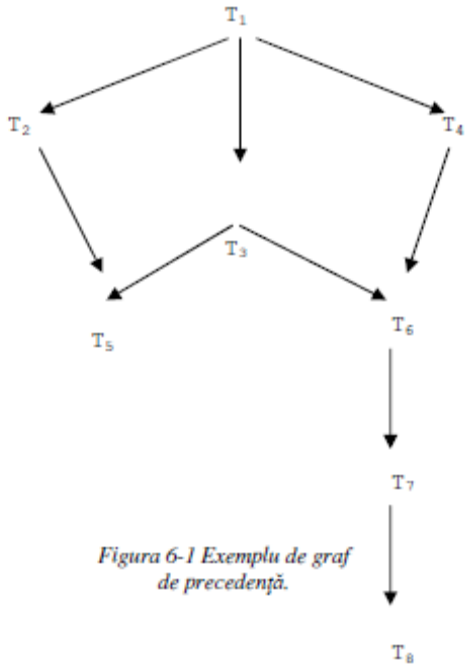

Figura 6-1 Exemplu de graf de precedență.

←(1)=∅
←(2)={1}
←(3)={1}
←(4)={1}
←(5)={1,2,3}
←(6)={1,3,4}
←(7)={1,3,4,6}
←(8)={1,3,4,6,7}

De asemenea se poate scrie $i \leftarrow j$ pentru a indica faptul ca taskul *Ti* precede taskul *Tj*, acest lucru fiind echivalent cu $j \rightarrow i$ (*Tj* este precedat de *Ti*).
Operatorul de precedența este tranzitiv, adica daca
$i \leftarrow j$ *si* $j \leftarrow k \Rightarrow i \leftarrow k$

Prin urmare, pentru comoditatea reprezentarii, se pot specifica numai stramosii imediat anteriori în mulțimea precedenta; de exemplu, se poate scrie ← (5)={2,3} deoarece ← (2)={1}.
În unele cazuri, operatorii →si ←sunt folosiți pentru a nota care task are prioritate mai mare: $i \rightarrow j$ poate sa însemne ca *Ti* are prioritate mai mare decât *Tj*.

Resurse

Fiecare task are cerințe de resurse: procesor, memorie, acces la magistrale de I/E etc. Resursele pot fi exclusive, atunci când un singur task le poate deține, sau neexclusive. În plus, o resursa poate fi exclusiva pentru anumite operații si neexclusiva pentru altele. Exemplu: pentru operații neatomice de scriere/citire a unor obiecte de memorie, operația de citire este neexclusiva, pe când operația de scriere este exclusiva.

Constrângeri de timp

Timpul lansarii unui task (în engl. *release time*) este momentul de timp începând de la care task-ul își poate începe execuția - toata informația necesara pentru a începe execuția task-ului este disponibila. Pentru task-urile periodice, timpul lansarii task-ului *Ti* este un deplasament fața de momentul de start al perioadei *Pi*.

*Timpul limit*a al unui task (în engl. *deadline*) este momentul de timp pâna la care task-ul trebuie sa-si termine execuția. Constrângerile pentru timpul limita pot sa fie rigide sau flexibile, depinzând de natura task-ului corespunzator.
*Timpul limit*a *relativ* (în engl. *relative deadline*) al unui task este timpul limita absolut minus timpul lansarii. Astfel, daca task-ul *Ti* are timpul limita relativ *di* si este lansat la momentul de timp *t*, trebuie sa fie executat pâna la momentul de timp *t* + *di*.

*Timpul limit*a *absolut* este timpul pâna la care execuția task-ului trebuie sa fie completa. În acest exemplu, timpul limita absolut al lui *Ti* este *t* + *di*.

Planificare
Planificarea task-urilor se spune ca este realizabila daca toate task-urile sunt startate dupa timpul lansarii si se termina înainte de timpul limita. O mulțime de task-uri este A-realizabila atunci când daca i se aplica un algoritm de planificare A rezulta o planificare realizabila. Cel mai mare volum de munca pentru planificarea timp - real se adreseaza obținerii de planificari realizabile.
Planificare poate fi *pre-execuție* (off-line), sau *dinamic*a (on-line). Planificarea off-line se face în avans fața de execuție, specificând când se executa task-urile periodice si care sunt ferestrele de timp pentru task-urile sporadice/aperiodice în eventualitatea ca ele sunt invocate. Planificarea on-line se face în timpul execuției, pe masura ce task-urile ajung în sistem. Algoritmii folosiți pentru planificarea on-line trebuie sa fie suficient de rapizi pentru a permite respectarea constrângerilor de timp de catre task-uri.

Prioritațile de execuție relative ale task-urilor sunt în funcție de natura lor si de starea curenta a procesului controlat. Exista astfel algoritmi care presupun ca prioritatea task-urilor nu se schimba în timpul execuției; acestia sunt numiți *algoritmi de prioritate statica*. În contrast, *algoritmii de prioritate dinamic*a presupun ca prioritatea se schimba în timp. Cei mai cunoscuți algoritmi de prioritate statica si de prioritate dinamica sunt algoritmul de „ritm monoton" (în engl. Rate Monotonic algorithm - **RM**) si respectiv algoritmul „cel mai apropiat timp limita primul planificat" (in engl. Earliest Deadline First - **EDF**).

Planificarea poate sa fie *preemptiv*a sau *nepreemptiv*a. Planificare este preemptiva daca task-ul care deține controlul procesorului poate fi întrerupt pentru planificarea altor task-uri gata de execuție care au prioritate mai mare (si reluat mai târziu). Planificarea este nepreemptiva daca task-ul care deține controlul procesorului nu poate fi întrerupt, planificarea fiind restartata atunci când task-ul cedeaza controlul procesorului – se termina, este blocat pe o resursa etc.

În continuare vor fi prezentați si analizați algoritmii RM si EDF. Scopul acestor algoritmi este respectarea timpului limita pentru fiecare task. Pentru cei doi algoritmi se fac urmatoarele presupuneri:
1. Task-urile nu au secțiuni nepreemptive si costul pentru preempțiune este neglijabil.
2. Numai cerințele de procesare sunt semnificative; memoria, I/E si alte cerințe de resurse sunt neglijabile.
3. Toate task-urile sunt independente; nu sunt constrângeri de precedența.

Notații

Notațiile utilizate în acestă tema vor fi:

n Numarul de task-uri din mulțimea task-urilor.
$\boldsymbol{e_i}$ Timpul de execuție al task-ului *Ti (cazul cel mai defavorabil).*
$\boldsymbol{P_i}$ Perioada task-ului *Ti*, daca el este periodic.
$\boldsymbol{I_i}$ a *k*-a perioada a task-ului periodic *Ti* începe la timpul *Ii + (k-1) Pi* , unde *Ii* este denumit *fazarea* taskului *Ti.*
$\boldsymbol{d_i}$ Timpul limita relativ al task-ului *Ti* (relativ la timpul lansarii).
$\boldsymbol{D_i}$ Timpul limita absolut al task-ului *Ti*.

r_i Timpul lansarii task-ului *Ti*.
$h_T(t)$ Suma timpilor de execuție ale iterațiilor task-urilor din mulțimea task-urilor T, care au timpul limita absolut nu mai târziu de *t*.

6.2 Algoritmul de planificare RM

Algoritmul RM este unul dintre cei mai studiați si utilizați în practica. Este o schema uniprocesor, preemptiva si cu prioritate statica.

Prezentarea algoritmului

Fața de presupunerile 1 – 3 de mai sus, se mai fac presupunerile:
1. Toate task-urile din setul de task-uri sunt periodice.
2. Timpul limita relativ al unui task este egal cu perioada sa.

Presupunerea 5 simplifica analiza algoritmului RM deoarece ea presupune ca poate fi cel mult o iterație a oricarui task activ în orice moment.

Prioritatea task-ului este inversa în raport cu perioada sa; daca perioada task-ului *Ti* este mai mica decât perioada *Tj* atunci *Ti* are prioritate mai mare decât *Tj*.

Task-urile cu prioritate mai mare pot preempta task-urile cu prioritate mai mica.

Exemplul
Fie trei task-uri A, B, C cu perioadele $P_A = 2$, $P_B = 6$, $P_C = 10$. Timpii de execuție sunt $e_A = 0.5$, $e_B = 2.0$, $e_C = 1.75$, iar $I_A = 0$, $I_B = 1$, $I_C = 3$. Deoarece $P_A < P_B < P_C$, T_A are cea mai mare prioritate. La fiecare relansare el preempteaza task-ul aflat în execuție. Similar, task-ul *TC* nu se poate executa atât timp cât T_A sau T_B sunt în execuție.
Pentru a determina daca un set de task-uri periodice independente este planificabil de algoritmul RM.
Teorema
Un set de n task-uri periodice independente planificate prin algoritmul RM își respecta întotdeauna timpii limita pentru toți timpii de lansare a task-urilor daca

$$\frac{e_1}{P_1} + \frac{e_2}{P_2} + \ldots + \frac{e_n}{P_n} \leq n(2^{1/n} - 1)$$

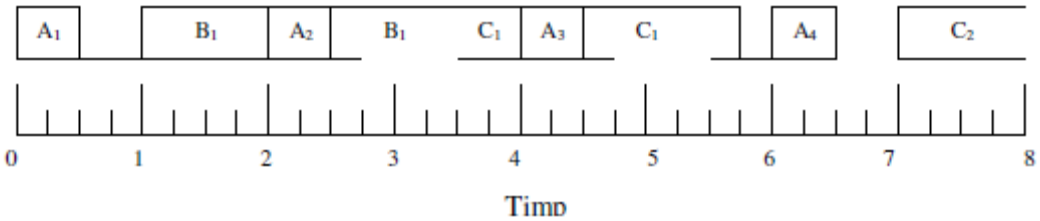

Figura 6-2 Exemplu de algoritm RM; *Xj* denota a *j*-a iterație a task-ului T_X.

e_i / P_i este *utilizarea* resursei de catre task-ul *Ti.* Limita utilizarii, $n(2^{1/n}-1)$, converge rapid catre $ln2 = 0.69$ când *n* este suficient de mare.

Limitarea data mai sus este pesimista, deoarece se presupune setul de task-uri în cazul cel mai defavorabil, puțin probabil de întâlnit în practica.

Condiția din teorema este suficienta dar nu si necesara. Altfel spus, pot fi seturi de task-uri cu o utilizare mai mare decât $n(2^{1/n} –1)$ si care pot fi planificate de algoritmul RM. Timpul ramas poate fi utilizat pentru task-uri de fond de prioritate mai mica. Limita $n(2^{1/n} –1)$ este prezentata în figura 6-3.
Se încearca în continuare sa se determine *condiția necesara* si *suficienta* pentru planificabilitatea RM . În acest sens, pentru a determina daca un set de task-uri cu utilizare mai mare decât limita

din teorema 1 poate sa-si respecte timpii limita, se poate utiliza un test de *planificabilitate exacta* bazat pe teorema *zonei critice*.

Teorema

Pentru un set de task-uri periodice independente, daca un task Ti îsi respecta primul sau timp limita di ≤ Ti, când toate task-urile de prioritate mai mare sunt startate la acelasi timp, atunci Ti îsi poate respecta toți timpii limita viitori, pentru orice timpi de lansare.

Este important de notat ca teorema 2 se poate aplica la orice tip de asignare a prioritaților, nu numai pentru asignarea asociata cu algoritmul RM.

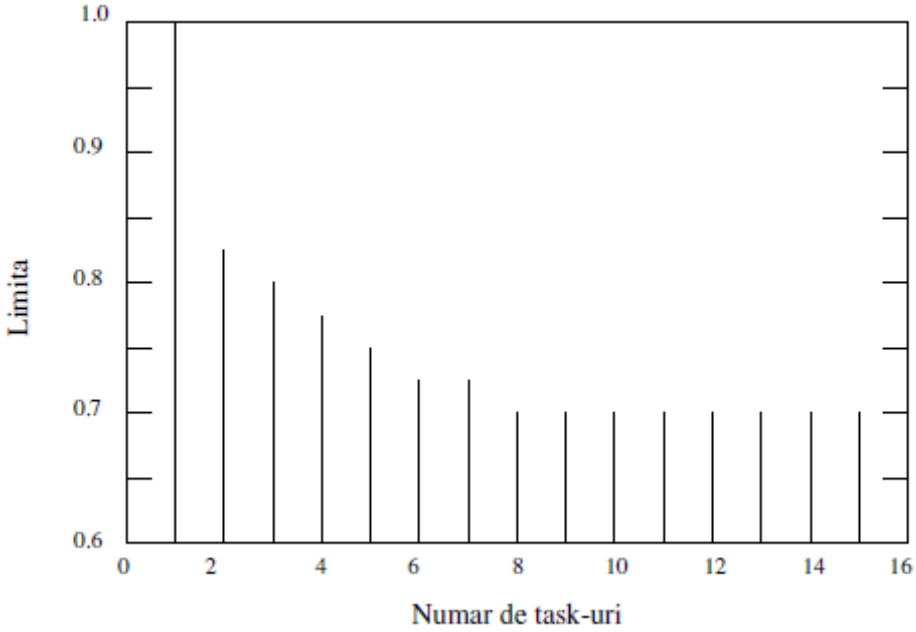

Figura 6-3 Limita de utilizare pentru algoritmul RM.

Pentru demonstrarea teoremei 2, reluam exemplul 6-2 pentru trei task-uri. Se presupune ca fazarile task-urilor sunt toate 0 (prima iterație a fiecarui task este la timpul 0). Se considera pentru început prima iterație. *T1* care este cel mai prioritar se starteaza primul si nu poate fi întârziat de nici un alt task din sistem.

Deoarece la lansarea task-ului *T1* procesorul întrerupe orice alta activitate, singura condiție pentru planificarea fezabila a *T1* este $e1 \leq P1$. Este clar ca aceasta este o condiție necesara si suficienta.

Task-ul *T2* va fi executat cu succes daca pentru prima sa iterație exista timp suficient pentru execuție în intervalul *[0,P2]*. Fie *t* momentul la care *T2* îsi termina execuția. Numarul total de iterații ale task-ului *T1* care a fost reluat în intervalul *[0,t]* este $[t/P_1]$. Pentru ca *T2* sa se termine la *t* fiecare dintre iterațiile lui *T1* reluata în
intervalul *[0,t]* trebuie sa fie completata si în plus trebuie sa fie un timp *e2* suficient pentru execuția *T2*. Altfel spus trebuie sa fie satisfacuta condiția:

$$t = \left\lceil \frac{t}{P_1} \right\rceil e_1 + e_2$$

Prin urmare, este suficient sa se gaseasca un timp $t \in [0,P2]$ care sa satisfaca condiția de mai sus. Întrebarea practica este cum se poate verifica ca acest *t* exista. Deoarece fiecare interval are un numar infinit de puncte, nu se poate verifica condiția pentru fiecare posibil *t*. Soluția poate fi gasita daca se ține cont de faptul ca $[t/P_1]$ reprezinta de câte ori task-ul *T1* a sosit în intervalul *[0,t]* (la multiplii de *P1* cu salturi de *e1*). Din acest motiv, $[t/P_1]e1$ reprezinta *necesarul de timp de execuție* pentru *T1* în intervalul *[0,t]*. Prin urmare, daca se gaseste un întreg *k* astfel încât $kP1 \geq ke1+e2$ si $kP1 \leq P2$, atunci s-a gasit condiția necesara si suficienta pentru ca *T2* sa fie

planificabil cu algoritmul RM. Astfel, este nevoie sa se verifice numai daca $t \geq [t/P_1]e1 + e2$ pentru valori ale lui *t* multiplu de *P1*, astfel ca $t \leq P2$. Deoarece exista un numar finit de multipli pentru *P1* care sunt mai mici sau egali cu *P2,* verificarea este finita.
În final se considera task-ul *T3*. Din nou este suficient de aratat ca prima iterație se termina înainte de *P3*. Daca *T3* își termina execuția la *t*, cu aceleasi argumente ca pentru *T2* avem:

$$t = \left\lceil \frac{t}{P_1} \right\rceil e_1 + \left\lceil \frac{t}{P_2} \right\rceil e_2 + e_3$$

T3 este planificabil daca exista un timp $t \in [0,P3]$ care sa satisfaca condiția de mai sus. Dar partea dreapta a ecuației de mai sus are salturi numai la multiplii de *P1* si *P2*. Este asadar suficient sa se verifice daca inegalitatea:

$$t \geq \left\lceil \frac{t}{P_1} \right\rceil e_1 + \left\lceil \frac{t}{P_2} \right\rceil e_2 + e_3$$

este satisfacuta pentru *t* multiplu de *P1* si/sau *P2*, astfel ca $t \leq P3$.
Se poate prezenta acum condiția necesara si suficienta pentru planificabilitatea RM. Pentru aceasta, se fac urmatoarele notații suplimentare:

$$W_i(t) = \sum_{j=1}^{i} e_j \left\lceil \frac{t}{P_j} \right\rceil$$

$$L_i(t) = \frac{W_i(t)}{t}$$

$$L_i = \min_{0 < t < P_i} L_i(t)$$

$$L = \max\{L_i\}$$

$W_i(t)$ este „cantitatea" totala de procesare efectuata de task-urile *T1 , T2 ..... Ti,* inițiate în intervalul *[0,t]*. Daca toate task-urile sunt lansate la momentul 0, atunci task-ul *Ti* va fi completat sub algoritmul RM la momentul *t'*, astfel încât $W_i(t') = t'$ (daca acest *t'* exista).
Pentru condiția necesara si suficienta pentru planificabilitate, se poate enunța teorema urmatoare:

Teorema
Dat fiind un set de n task-uri periodice (cu P1 ≤P2 ≤.... ≤Pn), task-ul Ti poate fi fezabil planificat utilizând
algoritmul RM daca si numai daca Li ≤1.

Demonstrație: Daca *Li ≤1*, atunci exista $t \in [0,Pi]$ a.i. *Wi(t)/t ≤1*, altfel spus *Wi(t) ≤t.* Deoarece *Ii* = 0 pentru orice *i = 1,...,n Wi(t) ≤t* implica faptul ca la timpul *t* cerințele de calcul ale task-urilor *T1* pâna la *Ti* au fost respectate. Deoarece *t ≤Pi* task-ul *Ti* își respecta timpul limita. Invers, daca *Wi(t) > t* pentru orice $t \in [0,Pi]$, atunci nu va fi suficient timp de execuție pentru task-ul Ti înainte de timpul limita, *Pi.*

Metoda de verificare practica a planificabilității
La întrebarea practica referitoare la modul cum se verifica daca *Wi(t) ≤t*, se poate raspunde usor prin examinarea ecuației

$$W_i(t) = \sum_{j=1}^{i} e_j \left\lceil \frac{t}{P_j} \right\rceil$$

Se vede ca *Wi(t)* este constant, exceptând un numar finit de puncte când task-urile sunt relansate. Va fi deci necesar numai sa se calculeze *Wi(t)* la momentele:

$$\tau_i = \{ lP_j \mid j=1,\ldots,i;\ l=1,\ldots, \lfloor Pi/Pj \rfloor \}$$

Atunci, sunt doua condiții de planificare RM:

RM1. Daca $min_{t \in \tau i}\ W_i(t) \leq t$, task-ul *Ti* este RM-planificabil

RM2. Daca $max_{i \in \{1,..,n\}}\ \{min_{t \in \tau i}\ W_i\ (t)/t\} \leq 1$ pentru $i \in [1,\ldots,n],\ t \in \tau_i$ atunci în totalitate, setul *T* este RM planificabil.

Exemplul

Se considera un set de patru task-uri unde:

i	e_i	P_i
1	20	100
2	30	150
3	80	210
4	100	400

Atunci:

$\tau_1 = [100]$

$\tau_2 = [100, 150]$

$\tau_3 = [100, 150, 200, 210]$

$\tau_4 = [100, 150, 200, 210, 300, 400]$

Sa calculam planificabilitatea RM pentru fiecare task. Figura 6-4 conține diagrame ale *Wi(t)* pentru i = 1, 2, 3, 4. Task-ul *Ti* este RM planificabil daca oricare parte a diagramei *Wi(t)* se plaseaza sub linia *Wi(t) = t*.

În termeni algebrici se poate scrie:

- Task-ul T1 este RM planificabil daca e1 ≤100
- Task-ul T2 este RM planificabil daca

e1 + e2 ≤100 sau

2e1 + e2 ≤150

- Task-ul T3 este RM planificabil daca

e1 + e2 + e3 ≤100 sau

2e1 + e2 + e3 ≤150 sau

2e1 + 2e2 + e3 ≤200 sau

3e1 + 2e2 + e3 ≤210

- Task-ul T4 este RM planificabil daca

e1 + e2 + e3 + e4 ≤100 sau

2e1 + e2 + e3 + e4 ≤150 sau

2e1 + 2e2 + e3 + e4 ≤200 sau

3e1 + 2e2 + e3 + e4 ≤210 sau

3e1 + 2e2 + 2e3 + e4 ≤300 sau

4e1 + 3e2 + 2e3 + e4 ≤400

Extinderea algoritmului de planificare RM pentru includerea task-urilor Sporadice

Algoritmul RM se adreseaza sistemelor în care toate task-urile sunt periodice. În realitate se întâlnesc rar astfel de sisteme timp - real, cel mai des fiind întâlnite cele care au atât task-uri periodice cât si task-uri sporadice sau aperiodice. Prin urmare, analiza trebuie extinsa pentru determinarea influenței execuției acestora asupra încarcarii sistemului si respectarii limitelor de timp.

Task-urile sporadice sunt lansate neregulat, ca raspuns la evenimente care apar independent de execuție. Deoarece task-urile sporadice nu au asociate perioade, trebuie sa se determine rata maxima la care ele pot fi lansate. Astfel spus, trebuie sa existe un interval de timp minim între lansarile succesive ale task-urilor sporadice. În caz contrar, nu exista o limita maxima pentru timpul de încarcare a sistemului de catre task-urile sporadice si va fi imposibil de garantat ca se respecta timpii limita.

O modalitate de includere a task-urilor sporadice este de a le considera pur si simplu ca task-uri periodice, cu o perioada egala cu intervalul de timp minim posibil între doua lansari succesive. Pentru aceasta, se pot imagina mai multe metode.

Metoda 1

O metoda simpla de a încorpora în sistem task-uri sporadice este aceea de a defini un task periodic fictiv, de cea mai mare prioritate, la care sa se gaseasca o perioada de execuție fictiva. În intervalul de timp planificat pentru execuția acestui task, procesorul este disponibil pentru a executa orice task sporadic gata de execuție. În rest, timpul va fi afectat task-urilor periodice. Task-urile sporadice nu vor fi lansate în afara intervalului planificat.

Metoda 2

Planificarea task-urilor sporadice se poate realiza si prin algoritmul *„server suspendat”* (în engl. Deferred Server - **DS**). Prin algoritmul DS, prezentat în [LSS87], ori de câte ori procesorul este planificat pentru task-uri sporadice si nu gaseste astfel de task-uri gata de execuție, planificatorul lanseaza în execuție task-uri periodice, în ordinea prioritații. Totusi, daca un task sporadic trece în starea gata de execuție, el preempteaza task-ul periodic curent si poate ocupa un timp total egal cu timpul alocat pentru task-urile sporadice.

Criteriul de planificabilitate pentru algoritmul DS se poate obține din algoritmul de baza RM. Atunci când timpul limita relativ al tuturor task-urilor este egal cu perioada lor si *Us* este utilizarea procesorului alocata pentru task-urile sporadice, se poate arata ca este posibila planificarea task-urilor periodice daca utilizarea totala *U* (incluzând si contribuția task-urilor sporadice) satisface urmatoarea limita:

$$U \leq \begin{cases} U_s \text{ daca } & U_s \geq 0.5 \\ 1 - U_s \text{ daca } & U_s \leq 0.5 \end{cases}$$

6.3 Algoritmul de planificare EDF

Conform algoritmului EDF, procesorul executa întotdeauna task-ul cu cel mai apropiat timp limita absolut. EDF este un algoritm de planificare cu *prioritate dinamic*a; prioritațile task-urilor nu sunt fixate preexecuție, fiind stabilite în timpul execuției, în funcție de apropierea timpilor limita absoluți fața de momentul planificarii.

Exemplul

Se considera urmatorul set de task-uri (aperiodice):

Task	Timp de sosire	Timp de execuţie	Timp limită absolut
T_1	0	10	30
T_2	4	3	10
T_3	5	10	25

Când *T1* soseste, este singurul task care asteapta sa se lanseze si îsi începe execuţia imediat. *T2* soseste la timpul *4*; din *D2 < D1*, el are o prioritate mai mare decât *T1* si îl preempteaza. *T3* soseste la momentul *5*; deoarece *D3 < D2*, el are o prioritate mai mica decât *T2* si trebuie sa astepte ca *T2* sa se termine. Când *T2* se termina (la timpul *7*), *T3* se starteaza (deoarece el are o prioritate mai mare decât *T1*), *T3* se executa pâna în momentul *17*, dupa care *T1* se poate relua si executa complet.

Pentru tratarea algoritmului EDF se fac toate presupunerile care s-au facut pentru algoritmul RM, exceptând faptul ca task-urile nu trebuie sa fie periodice.

EDF este un algoritm optimal de planificare uniprocesor. Altfel spus, daca EDF nu poate planifica fezabil o mulţime de task-uri pe un procesor, nu exista nici un alt algoritm de planificare care ar putea face aceasta.

Daca toate task-urile sunt periodice si au timpii limita relativi egali cu perioada lor, testul pentru planificabilitatea mulţimii de task-uri este simplu:

Daca utilizarea totala a unei mulţimi de task-uri nu este mai mare decât 1, mulţimea de task-uri poate fi planificata fezabil pe un singur procesor, cu algoritmul EDF.

Dupa cum s-a aratat anterior, testul de planificabilitate nu este simplu pentru cazul când timpii limita relativi nu sunt egali cu perioada; în astfel de cazuri, trebuie dezvoltata o planificare utilizând algoritmul EDF, pentru a vedea daca se respecta toţi timpii limita pentru un interval de timp dat. Se prezinta în continuare un test de planificabilitate EDF în acest caz.

Se definesc

$$u = \sum_{i=1}^{n} \frac{e_i}{P_i}, \; d_{max} = max_{1 \le i \le n} \{d_i\} \text{ şi } P = cmmc(P_1,\ldots,P_n),$$

unde *cmmc* este cel mai mic multiplu comun.

Fie $h_T(t)$ suma timpilor de execuţie ai tuturor task-urilor din mulţimea *T,* a caror limita de timp absoluta este mai mica decât *t*.

Teorema: O mulţime de n task-uri nu este fezabila EDF daca si numai daca:

- $u > 1$

sau

- *există* $t < \min\left\{P + d_{max}, \frac{u}{1-u} \max_{1 \le i \le n}(P_i - d_i)\right\}$

astfel încât $h_T(t) > t$.

6.4 Analiza comparativa a algoritmilor RM si EDF

Planificarea task-urilor de timp real astfel încât sa se garanteze ca toate task-urile îsi respecta timpul limita este o parte importanta a oricarui SOTR. Cele mai utilizate tehnici de planificare în timp real sunt cele periodice, în care planificarea este stabilita pre-execuţie, fiind exprimata printr-o tabela de planificare care are numarul de intrari egal cu *cmmmc* al perioadelor task-urilor. Aceasta elimina deciziile de planificare în timpul execuţiei si minimizeaza

supraîncarcarea (planificatorul este declansat la fiecare tick si planifica taskurile conform tabelei), dar introduce câteva probleme precum:

- tabela de planificare trebuie calculata off-line, adesea manual si este dificil de modificat atunci când caracteristicile task-urilor se modifica în timpul proiectarii;
- task-urilor aperiodice de prioritate mare este foarte posibil sa li se aloce timp în mod necorespunzator, deoarece apariția lor nu poate fi anticipata pre-execuție;
- plaja mare de perioade (în aceeasi aplicație sunt task-uri cu perioade de 1 x ms, 10 x ms, 100 x ms etc.), des întâlnita în aplicațiile MCP duce la tabele de dimensiuni foarte mari, inacceptabile uneori pentru memoria sistemelor utilizate pentru aplicații încorporate.

Alternativa la aceste tipuri de planificatoare este folosirea de planificatoare bazate pe prioritate precum RM si EDF. Utilizarea lor implica însa încarcarea suplimentara a procesorului, prin doua componente: *supraîncarcarea execuției* (datorita execuției codului planificatorului) si *supraîncarcarea de planificare* (limitele teoretice ale numarului de task-uri planificabile cu un anumit algoritm). Împreuna, aceste supraîncarcari limiteaza timpul procesor utilizat efectiv pentru calculele în timp real.

6.5 Studiu de caz: algoritm mixt de planificare RM-EDF

Utilizând planificarea mixta, se elimina supraîncarcarea de planificare, dar supraîncarcarea execuției ramâne. Aceasta depinde de lista în care este task-ul care urmeaza a fi blocat/deblocat (pentru lista *PD* supraîncarcarea este mai mare decât pentru lista *PF*). Se pot considera urmatoarele cazuri:

1) Blocarea unui task din *PD*: Δtb este $O(1)$ (acelasi ca pentru EDF). În cazul cel mai defavorabil, Δts apare când exista cel puțin un task gata de execuție în lista *PD*, necesitând parcurgerea acesteia pentru a selecta urmatorul task. Astfel, Δts este $O(x)$. Parcurgerea se efectueaza numai daca *Ce* este diferit de 0.

2) Deblocarea unui task din *PD*: Δtd este $O(1)$. Deoarece la sfârsitul acestei operații exista cel puțin un task gata de execuție în lista *PD*, aceasta va fi parcursa pentru selectarea urmatorului task pentru execuție. Astfel, Δts este $O(x)$.

3) Blocarea unui task din *PF*: *tb* este acelasi ca pentru RM, dar pentru un numar mai mic de task-uri în lista, astfel Δtb este $O(n-x)$. Deoarece în execuție este task-ul din lista *PF* care se blocheaza, înseamna ca nu exista task-uri gata de execuție în lista *PD* (acestea au prioritatea mai mare decât a oricarui task din *PF*), si aceasta nu va fi parcursa pentru selecție. Planificatorul selecteaza pentru execuție task-ul indicat de *firstțready*, astfel ca Δts este $O(1)$ (acelasi ca pentru RM).

4) Deblocarea unui task din *PF*: Δtd este $O(1)$ (acelasi ca pentru RM). Este posibil ca în lista *PD* sa existe task-uri gata de execuție, daca *Ce* este diferit de 0, prin urmare pentru cel mai defavorabil caz Δts poate fi $O(x)$.

Din aceasta analiza, rezulta pentru operațiile de blocare/deblocare o supraîncarcare totala a execuției planificatorului de $\Delta tb + 2\Delta ts + \Delta td$. Pentru task-urile din lista *PD*, supraîncarcarea este $O(1) + 2O(x) + O(1) = 2O(x)$, ceea ce este echivalent cu parcurgerea de doua ori a unei liste de lungime *x*. Pentru task-urile din lista *PF*, supraîncarcarea este $O(n-x) + O(1) + O(1) + O(x) = O(n)+O(2)$, ceea ce este echivalent practic cu parcurgerea unei liste de lungime *n*, o singura data. Prin urmare, supraîncarcarea este mai mica decât pentru EDF (deoarece se parcurge de doua ori o lista de lungime *x* comparativ cu parcurgerea de doua ori a uneia de lungime *n*, iar $x < n$) si cu puțin mai mare decât RM (o lista de lungime *n* parcursa o singura data).