Tema **5 SISTEME DE OPERARE TIMP - REAL MULTITASKING**

5.1 Introducere

O metoda uzuala de a structura o *aplicație timp - real* este aceea de a realiza un numar de task-uri cooperante care se executa concurent (pentru *task* vezi si 6.1). În mod uzual, proiectarea si implementarea aplicației timp - real se poate face mai usor daca *sistemul de operare (SO) utilizat suport*a *prelucr*ari *multitasking*.

Este cunoscut ca SO tradiționale se bazeaza pe aceea ca toate aplicațiile sau task-urile din sistem sunt executate concurent pe o singura UC cu un singur procesor. În contrast, procesarea paralela presupune ca mai multe procesoare sa lucreze în paralel, fiecare dintre ele executând concurent task-uri. Sistemele cu procesare paralela au un cost ridicat si în majoritatea aplicațiilor timp - real raportul performanța/cost determina utilizarea unui sistem cu un singur procesor si cu SO adecvat pentru execuția concurenta a task-urilor.

Pentru începatori, o confuzie poate aparea între proprietațile *multiuser* si *multitasking* ale SO. Proprietatea *multiuser* a SO asigura ca fiecare utilizator executa propria aplicație ca si când ar avea la dispoziție toate resursele calculatorului:

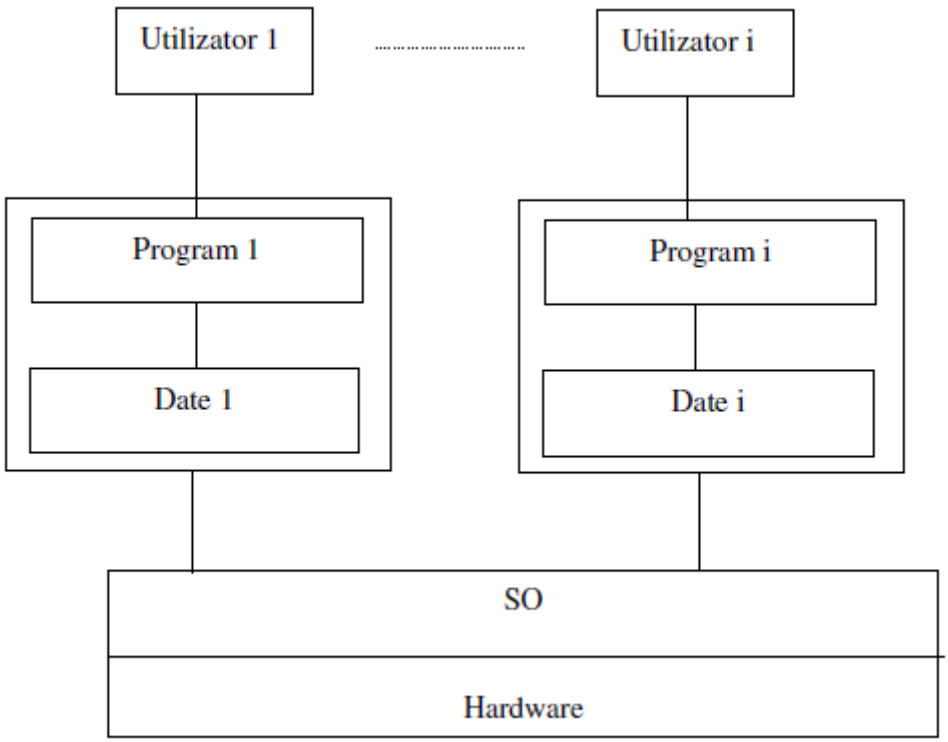


Fig. 5.1-1 Execuția multiuser

Fiecare program (aplicație) ruleaza în propriul mediu, protejat fața de celelalte aplicații. Proprietatea *multitasking* a SO se refera la faptul ca acesta gestioneaza mai multe task-uri care coopereaza *în cadrul unei aplicații*, în vederea realizarii funcțiilor pentru care aceasta a fost proiectata. Cooperarea presupune ca task-urile comunica între ele prin mesaje si ca partajeaza date comune.

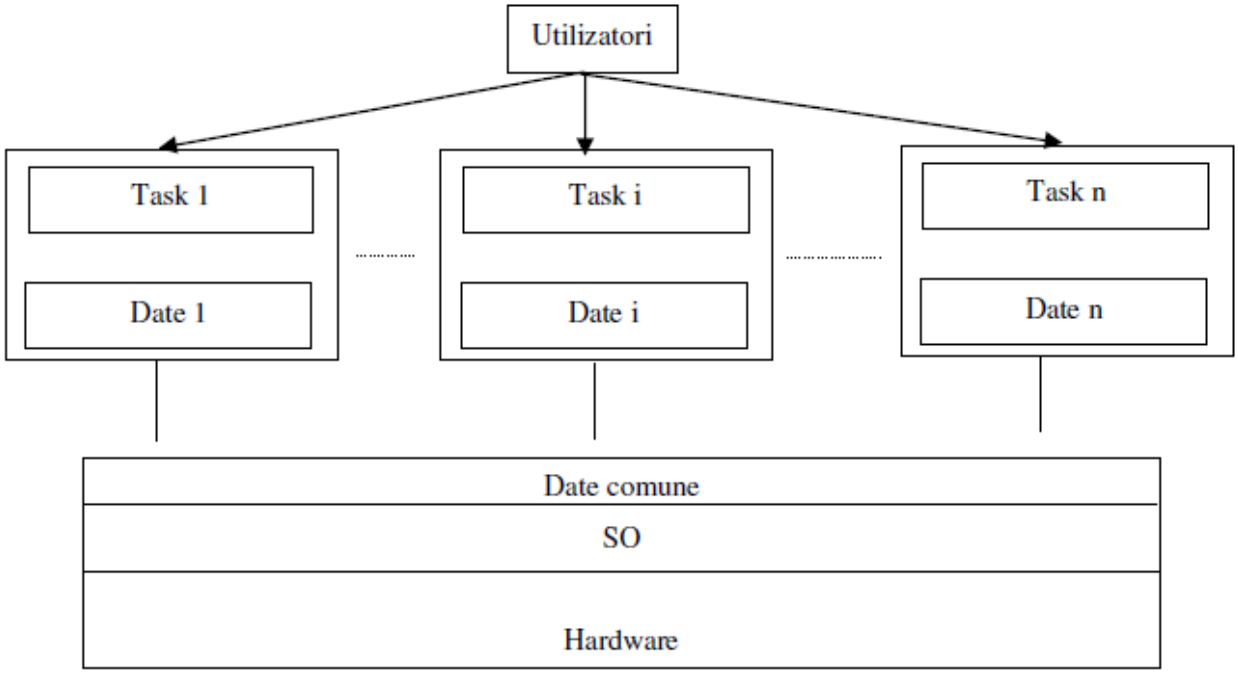


Fig. 5.1-2 – Execuția multitasking

În aplicațiile multitasking comunicarea între task-uri si partajarea datelor se face prin intermediul unor primitive iar SO previne accesele nepermise la datele partajate. De asemenea, se asigura protecția datelor private taskurilor. (De notat însa ca interferențele voite sau care sunt recunoscute ca voite de SO nu pot fi prevenite – SO presupune ca task-urile coopereaza).

Uzual, în sistemele timp - real încorporate (Embedded Real-Time Systems), *aplicația timp - real* care ruleaza pe suportul hardware este alcatuita din *Sistemul de Operare Timp - Real Multitasking*

(SOTRM) si task-urile care implementeaza funcțiile aplicației sau funcții de sistem. Principalele atribuții ale SOTRM se refera la:
Alocarea resurselor: O cerința importanta la un SO este alocarea resurselor calculatorului la diverse task-uri. În cadrul aplicațiilor timp - real procedura de alocare este mai complicata pentru ca unele dintre activitați sunt critice în raport cu timpul. Prin urmare, trebuie sa existe modalitați si discipline foarte riguroase de asignare a prioritaților la task-uri si de alocare a timpului UC în concordanța cu o anumita schema de prioritate.

Comunicare între task-uri: Un task poate utiliza servicii furnizate de catre alte task-uri si de asemenea serviciile oferite de el pot fi utilizate de alte task-uri. Prin urmare, SO trebuie sa dispuna de mecanisme pentru a permite task-urilor sa comunice între ele (*zone de memorie partajat*a*, mesaje, cutii postale etc)*. În plus, task-urile pot fi activate de catre evenimente externe, SO trebuind astfel sa sesizeze evenimentele din mediul extern (de exemplu prin *utilizarea întreruperilor*) si sa transfere tratarea lor catre task-urile asociate.

Gestiunea secțiunilor critice: Este frecvent întâlnita situația în care mai multe task-uri trebuie sa acceseze resurse comune, atât hardware cât si software. De aceea, SO trebuie sa aiba posibilitatea de a preveni ca 2 sau mai multe task-uri sa utilizeze în acelasi timp aceste resurse.
În concluzie, SOTRM trebuie sa gestioneze *partajarea resurselor*, *comunicarea*, *sincronizarea* si sa *planifice* si *s*a *dispecerizeze* execuția task-urilor conform cerințelor de timp real ale acestora (vezi si capitolul 6). Funcțiile SOTRM pot fi repartizate astfel:
 gestiunea task-urilor (planificare, dispecerizare);
 comunicarea si sincronizarea între task-uri;
 tratarea evenimentelor externe (uzual, întreruperi externe);
 gestiunea memoriei;
 partajarea codului;
 partajarea dispozitivelor;
 partajarea datelor.
Suplimentar, unele SOTRM au funcții de comunicații de date, gestiune fisiere sau conțin programe utilitare pentru mentenanța, depanare, etc.
Exista SOTRM care coabiteaza cu un SO gazda de uz general precum MS-DOS, Windows, Linux etc. si utilizeaza ca suport pentru dispozitivele de I/E, gestiunea de fisiere si programe utilitare mijloacele existente în sistemul de operare gazda. În literatura de specialitate aceste arhitecturi de SO sunt referite ca arhitecturi de tip *"kernel dual"*.
În continuarea capitolului 5 se prezinta noțiuni introductive legate de planificarea si dispecerizarea task-urilor iar în capitolul 6 sunt prezentați algoritmii clasici de planificare a task-urilor în prezența constrângerilor de timp rigide.

5.2 Gestiunea task-urilor în aplicații timp - real

Gestiunea task-urilor este facuta de catre doua module: *planificatorul* de task-uri, care gestioneaza starile taskurilor si listele asociate acestora si *dispecerul* care este responsabil de alocarea UC la task-ul care intra în execuție. Aceste module constituie *nucleul* (*kernelul)* SO. De asemenea, pe lânga nucleu exista si task-uri sistem care ruleaza dupa regulile task-urilor utilizator si asigura servicii la nivel de sistem.

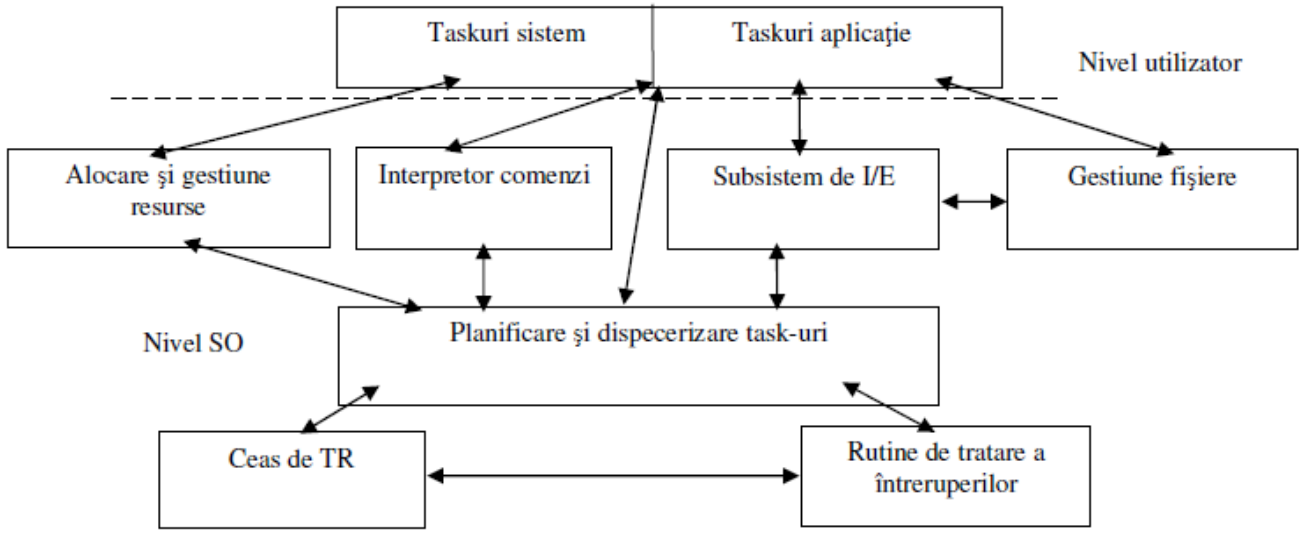


Fig. 5.2-1 Structura generala a unei aplicații timp – real

5.2.1 Tranziția starilor si descriptori de task

Pe un calculator cu un singur procesor numai un task poate rula la un moment de timp dat si din acest motiv celelalte task-uri trebuie sa fie în alta stare decât *activ* (în execuție). O diagrama tipica de tranziție a starilor este prezentata în figura 5.2.1-1.

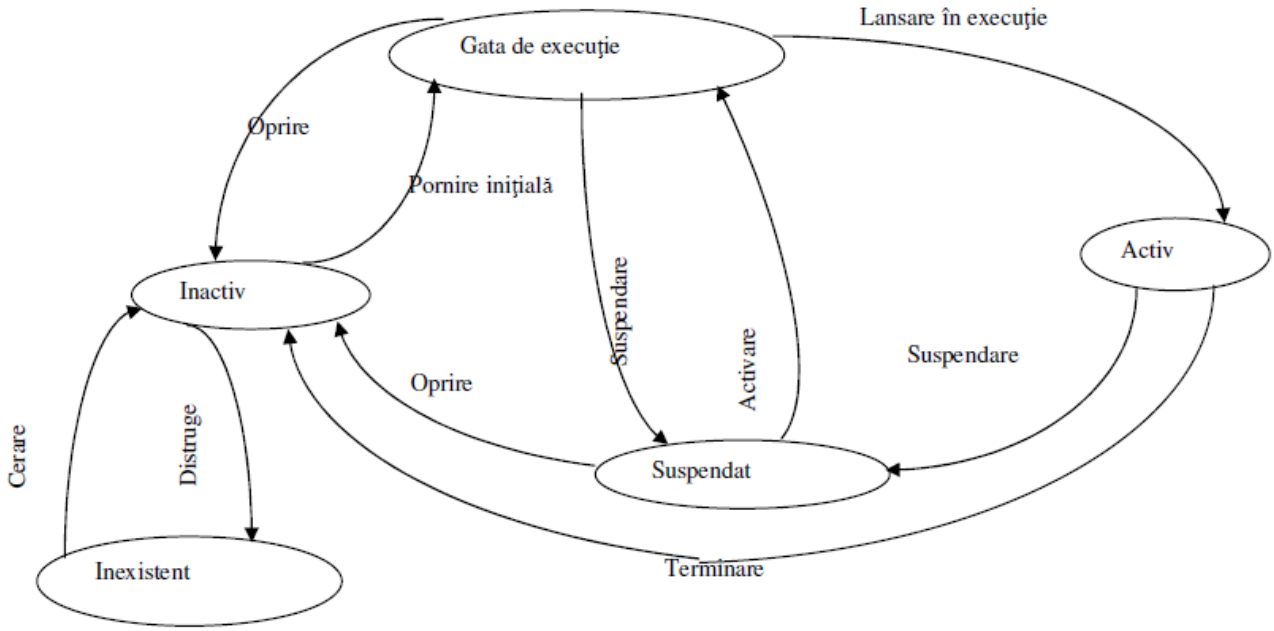


Fig. 5.2.1-1 – Tranziții de stari ale taskurilor

1. În starea *activ* (sau în execuție engl. *running*) este un singur task care deține controlul procesorului la un moment de timp dat. Acesta are în mod normal cea mai mare prioritate dintre task-urile care erau gata de execuție în momentul dispecerizarii.
2. În starea *gata de execuție* (engl. *ready*) pot fi mai multe task-uri. Resursele cerute de catre task trebuie sa fie disponibile pentru ca acesta sa poata fi trecut în starea gata de execuție.
3. În starea *suspendat*, un task poate fi în asteptare (engl. *waiting*), blocat (engl. *locked out*) respectiv întârziat (engl. *delayed*). Execuția oricarui task aflat în aceasta stare a fost suspendata anterior deoarece task-ul respectiv solicita resurse care nu erau disponibile sau task-ul asteapta apariția unui eveniment sau pentru ca task-ul asteapta sa treaca un interval de timp.
4. Starea *inactiv* (engl. *idle*). SOTRM este constient de existența acestui task, dar task-ul nu are alocata prioritate si nu este înca gata de execuție.

5. Starea inexistent (engl. *non-existing*). SOTRM nu este constient de existența acestui task, desi el poate fi rezident în memoria calculatorului. Starile task-ului pot fi modificate prin acțiuni ale SOTRM. De exemplu, daca o resursa devine disponibila / nedisponibila SOTRM modifica starea task-ului care are nevoie de resursa respectiva. De asemenea, starile unor task-uri pot fi modificate la inițiativa altor task-uri ale aplicației.

5.2.2 Descriptori de task

Din prezentarea referitoare la starile task-urilor si din cea referitoare la mecanismul întreruperilor (capitolul 4) este evidenta necesitatea ca SOTRM sa pastreze informații de stare despre fiecare task din sistem. Aceste informații sunt memorate în *descriptori de task* (DT). Un DT este o zona de memorie asociata unui task care conține cel puțin urmatoarele informații:

- identificator de task (ID);
- prioritatea task-ului;
- starea curenta;
- zona de memorare a mediului volatil.

Daca SOTRM permite ca task-ul sa fie transferat între memoria interna si memoria externa, în DT trebuie de asemenea pastrate informații referitoare la localizarea imaginii task-ului în memoria externa si dimensiunea memoriei interne necesara. De asemenea, în DT se pot memora informații referitore la blocurile de control pentru fisierele utilizate etc.
În mod uzual, descriptorii sunt memorați într-un tablou. Pentru gestiunea task-urilor bazata pe prioritați, fiecare descriptor conține câmpuri utilizate de catre planificator si dispecer pentru înlanțuirea task-urilor, dupa prioritate, în liste asociate starilor. Acest lucru este ilustrat în figura 5.2.1 – 2.

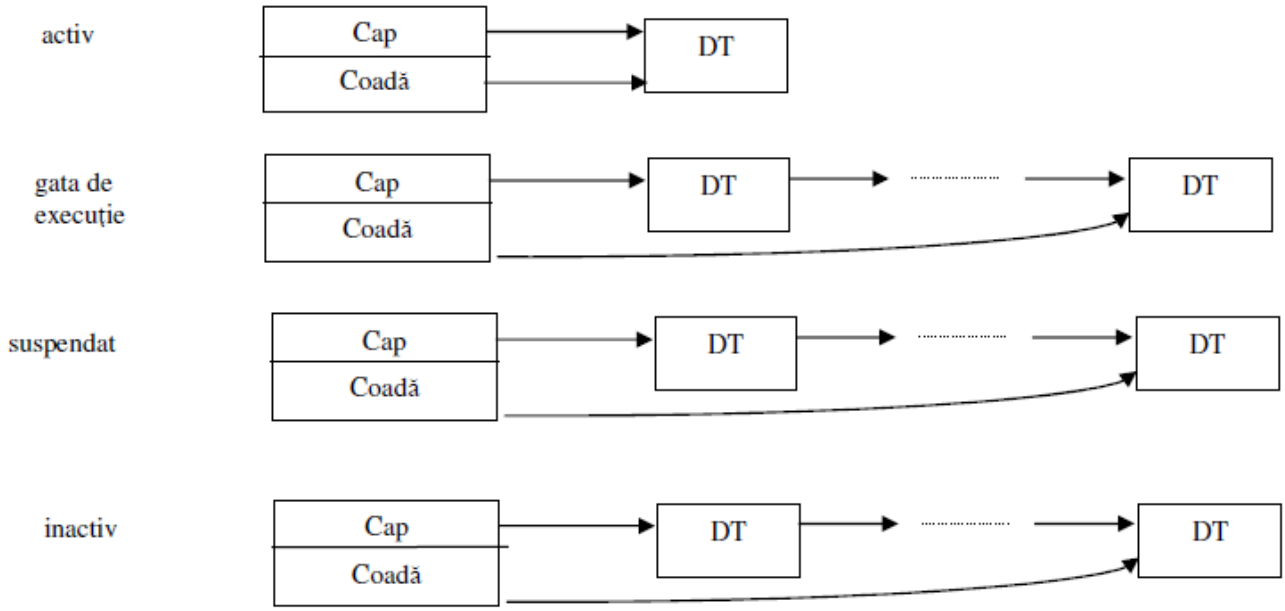


Fig. 5.2.1 – 2 Structura listelor de task-uri conform starilor

5.2.3 Dispecerizarea si planificarea task-urilor

Funcția principala a segmentului din SO care se ocupa de planificarea si dispecerizarea task-urilor este aceea de alocare a timpului procesorului la diverse task-uri, care pot fi *periodice*, *sporadice* sau *aperiodice* (vezi si capitolul 6).

Planificatorul se ocupa de deciziile de alocare de nivel înalt precum cele legate de funcționarea dispozitivelor de I/E. El recepționeaza informația despre disponibilitatea resurselor din sistem si trece task-urile din starea *suspendat* în starea *gata de execuție*. De asemenea, poate avea ca sarcina sa urmareasca daca task-urile de prioritate mai mica au asteptat prea mult si sa le schimbe prioritatea pentru a se asigura ca si ele vor fi lansate în execuție.
Dispecerul de task-uri este numit si *planificator de "nivel scazut"* atunci când este apelat comuta task-urile salvând mediul volatil de execuție al task-ului curent si lansând în execuție taskul gata de

execuție care are cea mai mare prioritate. De asemenea dispecerul ia decizii imediate de comutare a task-urilor declansat de catre evenimente externe (de exemplu ca raspuns la întreruperea de la un dispozitiv I/E, de la ceasul de timp real etc).

5.2.4 Niveluri de prioritate

Alocarea prioritaților task-urilor în aplicațiile de TR se face fie *pre-execuție*, în faza de proiectare, fie *dinamic*, în timpul execuției. Prioritatea alocata este în funcție de criteriile luate în considerare de catre algoritmul de planificare (vezi si capitolul 6): perioada de lansare în execuție, timpul limita de raspuns etc. Task-urile periodice sunt trecute în starea gata de execuție la începutul perioadei de timp asociata fiecarui task. Task-urile sporadice sau aperiodice sunt trecute în starea gata de execuție la apariția unor evenimente care trebuie tratate.
Un eveniment poate fi generat:
- de fenomene care se petrec în mediul extern;
- de trecerea unui interval de timp;
- de prelucrari interne aplicației.

Un criteriu de la care se porneste în gestiunea prioritaților este arhitectura aplicației timp - real. Cel mai frecvent aplicația este structurata pe mai multe niveluri: rutine de tratare întreruperi, task-uri periodice (bazate pe ceas) cu constrângeri de timp rigide si task-uri de baza, fara constrângeri rigide de timp. Prioritațile pot fi grupate pe 3 niveluri mai largi, ca în figura 5.2.3-1:

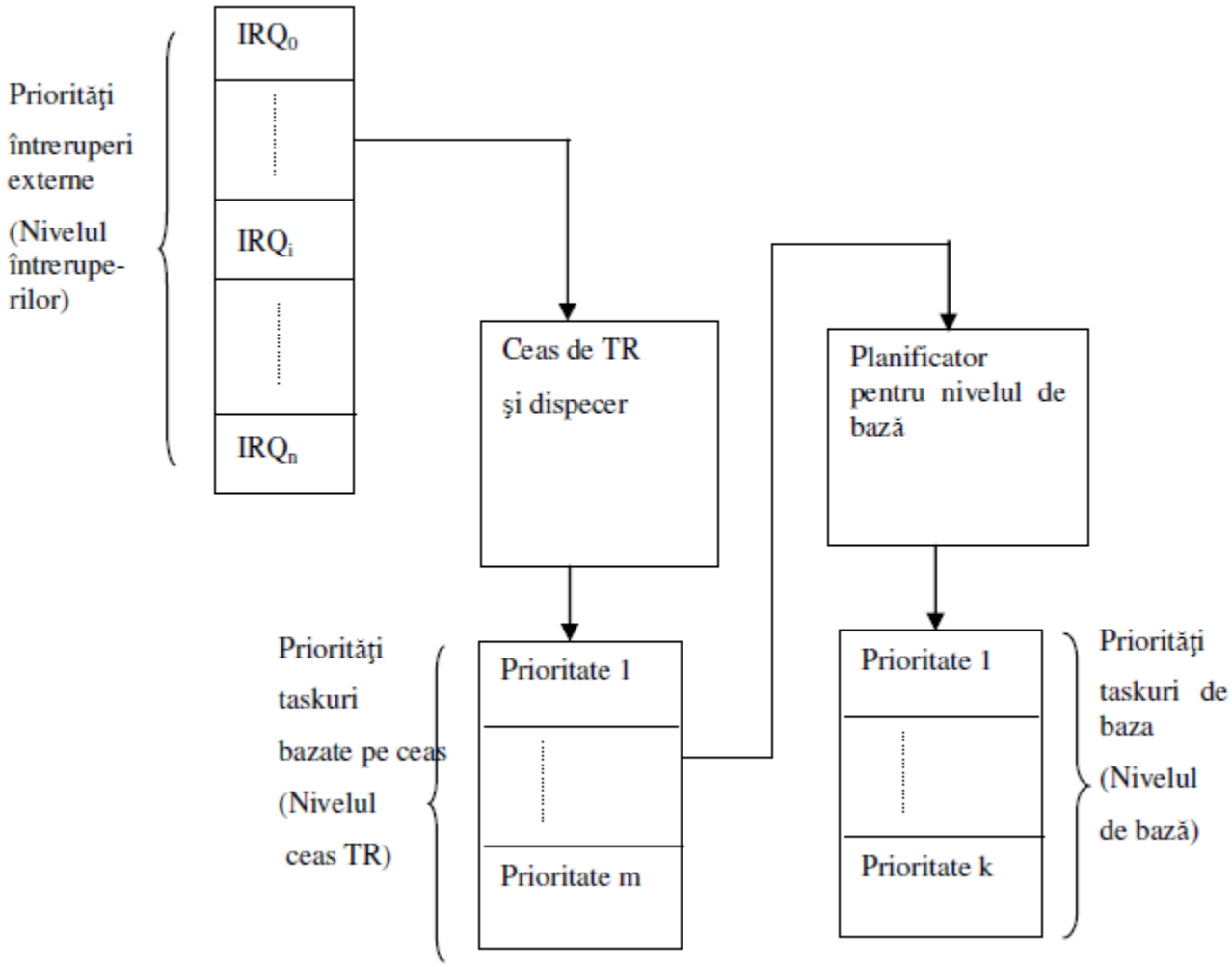


Fig. 5.2.3-1 Niveluri de prioritati în SOTRM

Nivelul întreruperilor (cel mai prioritar): La acest nivel sunt rutinele de serviciu pentru tratarea întreruperilor pentru task-urile si dispozitivele care cer un timp de raspuns foarte rapid (uzual de ordinul microsecundelor sau milisecundelor). Una dintre aceste rutine va trata întreruperea de la ceasul de timp real si va declansa dispecerizarea task-urilor.

Nivelul de ceas: Conține task-urile periodice, bazate pe ceas. La acest nivel sunt task-urile care cer prelucrare repetata (periodic), precum task-urile de achiziție si control procese. În unele aplicații, task-ul cel mai puțin prioritar la acest nivel este planificatorul pentru task-urile la nivelul de baza.
*Nivelul de baz*a – task-urile de prioritate mai scazuta, care nu au de respectat timpi limita de prelucrare sau care accepta anumite plaje de eroare de timp. Ele pot avea toate aceeasi prioritate sau li se pot aloca priorități diferite.
Prin urmare, arhitectura unei aplicații timp - real bazata pe nivelurile de prioritate prezentate mai sus poate fi modelata pe nivelele:
 Rutine de tratare a evenimentelor externe (întreruperi externe), cu execuție foarte rapida.
 Drivere de control dispozitive, control în proces, achiziție date, cu execuție preponderant periodica.
 Task-uri de prelucrare, memorare pe suport magnetic, vizualizare pe display etc. pentru care constrângerile de timp real sunt mai relaxate.

Existența nivelului de întreruperi si a nivelului task-urilor bazate pe ceas implica utilizarea unei strategii de *planificare preemptiv*a. Aceasta se refera la faptul ca atunci când un task trece în starea gata de execuție si el are o prioritate mai mare decât task-ul în curs de execuție, atunci task-ul curent va fi trecut în starea gata de execuție iar task-ul cu prioritate mai mare va fi lansat în execuție. În contrast, o strategie nepreemptiva de planificare presupune ca orice task în curs de execuție, indiferent de prioritate, nu este suspendat decât daca într-o forma sau alta el cedeaza controlul procesorului sau daca timpul alocat pentru execuția sa a expirat.
*Exemplu: planificare preemptiv*a *vs. planificare nepreemptiv*a
Fie un sistem care recepționeaza un semnal de întrerupere de alarma de la un proces iar ca raspuns la alarma trebuie sa activeze un task care controleaza un dispozitiv de alertare (hupa). Acesta este un task de prioritate mare la nivelul de baza. Ca raspuns la întreruperea de alarmare, rutina de tratare a întreruperii va face ca task-ul de alarmare sa fie plasat în lista task-urilor gata de execuție. Dupa plasarea în lista, rutina de tratare poate lua una din urmatoarele decizii:
 reîntoarcere în task-ul întrerupt;
 salt la dispecer.
Daca controlul este dat task-ului întrerupt, atunci task-ul de alarmare nu va fi lansat în execuție pâna când nu se declanseaza replanificarea în sistem la intervalul de timp stabilit pentru aceasta sau pâna când task-ul curent un cedeaza controlul. Aceasta este o strategie nepreemptiva.
Daca însa din rutina de tratare se face salt la dispecer iar task-ul de alertare are prioritatea cea mai mare dintre task-urile existente în lista gata de execuție, acesta va fi imediat lansat în execuție dupa ce task-ul întrerupt este trecut de catre dispecer în starea gata de execuție.

5.2.4.1 Nivelul întreruperilor

La 4.2 s-a aratat ca o întrerupere poate forța alocarea unității centrale la o rutina de tratare, iar sistemul nu are control asupra timpului cât dureaza execuția acesteia. Deoarece acest mod de alocare a unității centrale este asincron, este necesar ca timpul afectat prelucrarilor în cadrul rutinelor de tratare sa fie redus la minim. Se obisnuieste ca în cadrul acestor rutine sa se memoreze informațiile strict necesare asociate evenimentelor si sa se paseze aceste informații catre un task care le va prelucra ulterior. Task-ul respectiv va rula pe un nivel de prioritate mai scazuta (la nivel de ceas sau la nivel de baza).
Rutinele de tratare trebuie sa asigure mijloacele de comutare a task-urilor în primul rând salvând mediul volátil al acestora (registrii, stiva, starea coprocesorului matematic, etc). La terminarea sa, fie restaureaza mediul volatil salvat la început si da controlul task-ului întrerupt, *fie face salt la dispecer*, lasând mediul volatil salvat.

Regasirea informației despre mediul volatil se va face ulterior prin intermediul descriptorului de task. În cadrul nivelului întreruperilor pot fi mai multe priorități în funcție de masina. Uneori partea hardware rezolva problema prioritaților, alteori SO trebuie sa asigure ca întreruperile cu prioritate mai mica sa nu întrerupa pe cele de prioritate mai mare.

5.2.4.2 Nivelul de ceas

Unul dintre nivelurile de întrerupere este destinat pentru întreruperea de ceas de timp real. Rutina de tratare asociata este lansata în execuție la un anumit interval care în mod uzual este determinat de rata

de activare a task-ului care necesita cea mai mare frecvența de apelare. Valorile tipice sunt între 20 200 msec.

Vom denumi în continuare fiecare întrerupere de ceas „*tick*”, care este cel mai mic interval de timp cunoscut de sistem. Funcțiile de baza ale rutinei de tratare a întreruperii de la ceas sunt legate de obicei de actualizarea orei si datei sistem si de transferul controlului catre dispecer. Acesta din urma, va selecta care task se va lansa la un tick particular din sistem. Task-urile la nivelul de ceas sunt de doua tipuri:

- Periodice (ciclice) – care cer o sincronizare precisa cu mediul extern;
- întârziate (cu asteptare) – care necesita întârzieri fixe între lansari succesive în execuție sau întârzierea

activitații lor pe o perioada de timp data.

5.2.4.2.1 Task-uri periodice

Pentru aceste task-uri se specifica: *perioada de execuție, timpul (durata) de execuție, timpul de lansare în execuție în raport cu începutul perioadei* si *limita de timp la care trebuie s*a-si *termine execuția*. Task-urile de prioritate mai mica în cadrul nivelului de ceas se vor executa cu o anumita instabilitate în timp, pentru ca asteapta terminarea execuției task-urilor cu prioritate mai mare.

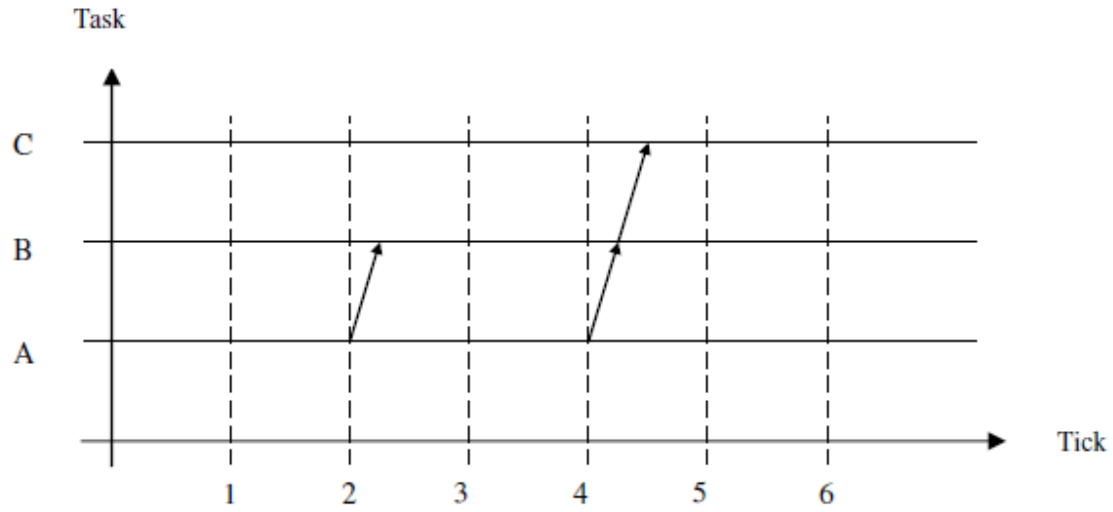


Figura 5.2.3.2.-1 Diagrama de activare a task-urilor, ordinea priorităţilor A, B, C

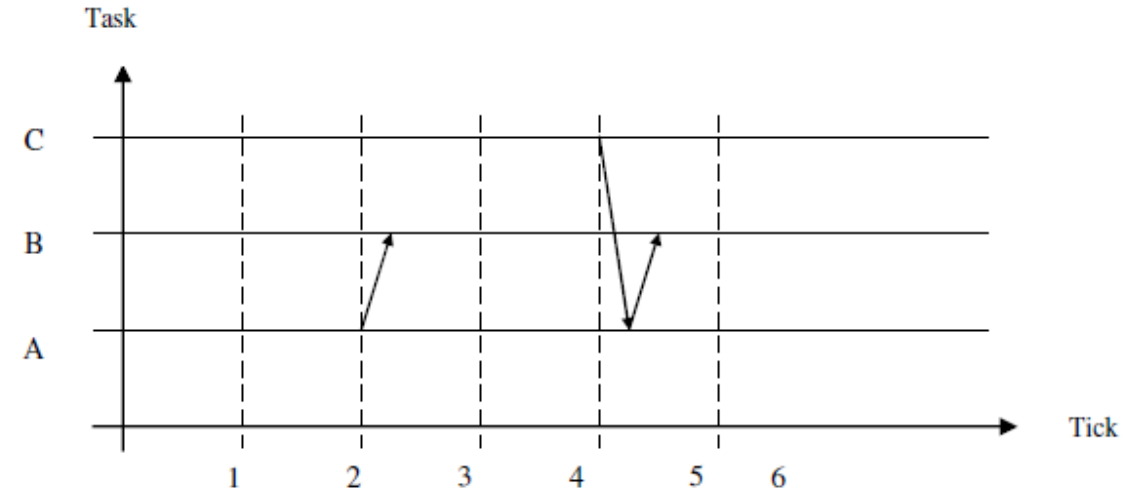


Figura 5.2.3.2.-2 - Diagrama de activare a task-urilor, ordinea priorităţilor C, A, B

5.2.4.2.2 Task-uri întârziate

Aceste task-uri îsi întârzie execuția pentru perioade fixe de timp, de exemplu pentru a permite completarea unor evenimente externe (exemplu: asteapta 100 msec pentru închiderea unui contactor apoi starteaza prelucrarile). Aceste task-uri utilizeaza informația de la ceasul de timp real, dar de cele mai multe ori se executa la nivel de baza. Când un task cere o întârziere el trece din starea în *execuție* în s*uspendat* si ramâne asa pâna când trece perioada de întârziere.

O metoda de implementare a funcției de întârziere este de a realiza o lista de descriptori de task-uri, ordonata dupa numarul de tick-uri pentru întârziere. Primul task din lista este cel care trebuie sa astepte cel mai puțin fața de tick-ul curent. Când un task solicita o întârziere, descriptorul de task este plasat în lista de task-uri în asteptare (*delay*). Inserarea în lista delay se face calculând numarul de tick-uri necesare din momentul apelului solicitarii. Controlul listei *delay* se face de catre un task care realizeaza planificarea la nivel de baza. Acesta este activat periodic si controleaza lista task-urilor întârziate pentru a vedea daca un task urmeaza a fi lansat.

Daca da, este trecut din lista task-urilor întârziate în lista celor gata de execuție.
Multe sisteme de operare nu dispun de algoritmi de operare periodica iar utilizatorul trebuie sa implementeze un timer suficient de precis (utilizând de exemplu funcții sistem sau programând ceasul si capturând întreruperile de la acesta) pe baza caruia sa apeleze periodic planificatorul la nivel de baza.

5.2.4.3 Nivelul de baza

Task-urile la nivelul de baza au constrângeri de timp flexibile. Felul în care task-urile la nivel de baza sunt planificate variaza de la un sistem la altul. Un algoritm de planificare poate fi Round-Robin: fiecare task din lista de task-uri gata de execuție (*ready*) este selectat si i se permite execuția pâna când task-ul ce se executa se autosuspenda sau pâna când se starteaza planificatorul de la nivelul de baza. Prioritatea poate fi fixa sau alocata dinamic. Alocarea dinamica a prioritaților se poate face fie de catre un planificator de nivel înalt, fie de catre task-uri, ad-hoc.

5.2.5 Apelul la dispecer

Dispecerul are doua condiții de intrare:
1. Întrerupere de la ceasul de timp real sau orice întrerupere care semnaleaza completarea unei cereri de I/E.
2. Suspendarea unui task datorata solicitarii unei întârzieri sau completarii ori cererii unui transfer de I/E. Întotdeauna se lanseaza în execuție task-ul cu cea mai mare prioritate. În primul caz (condiția 1), daca task-ul întrerupt nu va fi continuat la revenirea din întrerupere (alt task este mai prioritar), este trecut în starea gata de execuție. În al doilea caz (condiția 2) cautarea se starteaza de la task-ul urmator cu prioritate imediat mai mica care este în lista ready, deoarece nu poate fi în lista ready un task de prioritate mai mare ca a task-ului curent.

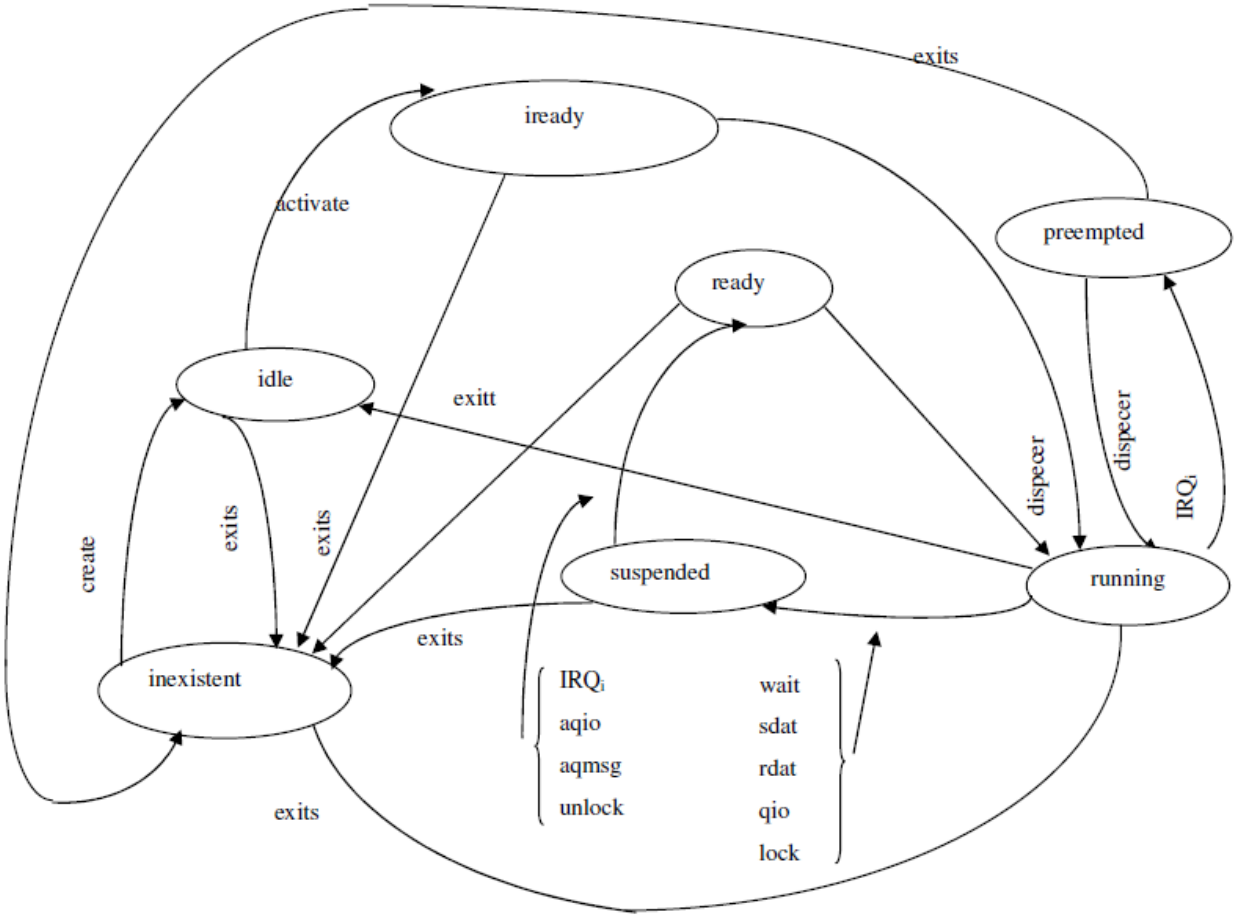
În figura 5.2.4.-1 este prezentata o schema logica a modulului de dispecerizare. Se presupune ca fiecare task are prioritate distincta (nu sunt mai multe task-uri cu aceeasi prioritate).

5.2.6.1 Starile task-urilor

Un task poate fi într-una dintre urmatoarele stari:
 Nonexistent: task-ul nu are descriptor, REX nu stie de existența sa.
 Idle: task-ul are descriptor, REX stie de existența sa, dar task-ul nu a fost activat explicit si nu este luat în considerare la planificare.
 Iready: este gata de lansare în execuție pentru prima oara (tranziție din starea idle).
 Ready: gata de execuție.
 Preempted: este gata de reluarea execuției dupa o întrerupere a acesteia la apariția unor evenimente a caror tratare activeaza task-uri cu prioritate mai mare.
 Suspended: motivul suspendarii poate fi:
ț asteptare mesaj de la alt task;

ț asteptare raspuns de la un task în urma emiterii unui mesaj;
ț blocat pe resursa critica;
ț asteptare evenimente externe;
ț asteptare pe interval de timp.
 Running: task-ul este în curs de execuție.

Diagrama de tranziție a starilor si directivele care concura la trecerea dintr-o stare în alta se prezinta în figura 5.2.5-1:



Fif. 5.2.5 -1 Automatul de tranziție a starilor la SOTRM REX