SISTEM DE CALCUL IN TIMP REAL

SCTR

-SZOKE ENIKO -

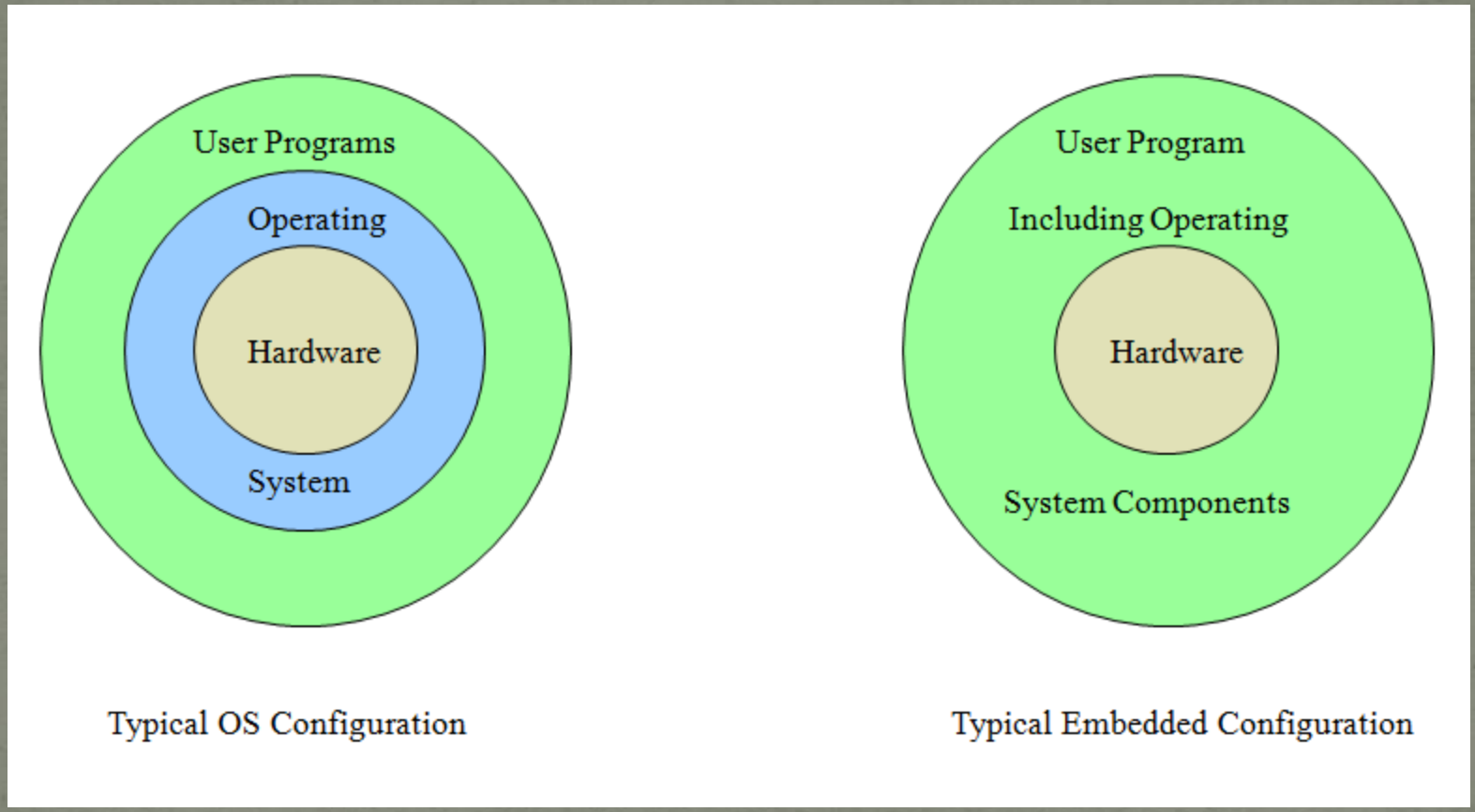
Curs 7

Cuprins

Sistem de operare SO

- Un **sistem de operare** (SO) este un program care realizează gestionarea resurselor hardware şi software ale unui calculator.
- Un SO realizează sarcini de bază cum ar fi:
  - controlul şi alocarea memoriei
  - gestionarea apelurilor către funcțiile sistem
  - controlul dispozitivelor de intrare şi ieşire
  - gestionarea comunicațiilor
  - manipularea fişierelor

Sistem de operare in timp real



- In miezul structurii se afla SC din pct. de v. fizic - **hardwarul.**
- Urmeaza **programul monitor** sau **firmwarul** - programul care preia comanda calculatorului atunci cand el este pornit, testeaza daca exista configuratia minima necesara ca sistemul sa poata functiona si sa predea comanda la SO. Acest program se afla in memorie interna si este de regula inscriptionat intr-o memorie de tip ROM.

  Firmwarul este specific fiecarei familii de SC, respectiv fiecarei familii de procesoare.
- **SO** este incarcat din memoria externa a calculatorului si reprezinta mediul care permite utilizatorului sa vizualizeze, sa incarce sau sa execute alte tipuri de programe.

- SO este interfata dintre SC si operatorul uman.
- - user friendly (operatorul sa cunoasca cat mai putin din structura interna a unui SC dar sa-l foloseasca eficient.)

- **Programele aplicative** sunt programe scrise de utilizatorii SC prin intermediul limbajelor de programare.
- **Programe utilitare** sunt produse de catre firmele de software si au rolul fie sa usureze munca utilizatorului (editoare de texte, grafice, medii de informare capacitatii hard - Norton Commandersau) sau compilatoare de limbaje de programare sau fie sa deserveasca interese pentru anumite aplicatii ale SC (ascultare muzica).

- Sistemele de operare proiectate pentru aplicatii real-time sunt folosite în general pentru computere de tip **"embedded"** (înglobate, precum telefoane mobile, roboti industriali sau echipamente de cercetare ştiintifică).
- Numim sistem în timp real încorporat un sistem în timp real înglobat într-un aparat, altfel spus caracterizat prin faptul că hardware-ul şi implicit şi software-ul sunt încorporate în aparatul însuşi.

Sistem de operare

În unele cazuri pune la dispoziția utilizatorului o interfață grafică (WindowsOS, MacOS).

Constituie o platformă pentru alte programe sistem sau pentru programele de aplicații.

Clasificare:

după tehnologie:

- tip Unix
- altele (DOS, Windows)

după tipul de licență:

- contra cost
- gratuită

după stadiul de utilizare:

- ieşite din uz (CP-M, DOS)
- în uz (Linux, Windows)

Sistem de operare

după tipul de utilizare:
- uz general (GPOS) (Linux, Windows)
- numai pentru calculatoare tip desktop (MS-DOS, MacOS)
- numai pentru calculatoare de tip mainframe (VM)
- pentru sisteme de timp real (RTOS)
- pentru sisteme încapsulate (embedded)

după scop:
- profesionale
- pentru cercetare
- hobby

Caracteristicile SOTR

- **Marimea timpului de raspuns** - intervalul de timp intre lansarea unei cereri de serviciu si raspunsul la acesta de catre SO.
- Timpul de raspuns = timp de asteptare + timp de executie.

- **Paralelismul** - posibilitatea de a opera simultan cu mai multe cereri de serviciu.
- **Partajarea si protectia** - arata nivelul la care utilizatorii au posibilitatea sa foloseasca in comun informatiile existente in SC si nivelul la care pot sa comunice si sa protejeze reciproc informatiile.
- **Generalitatea, flexibilitatea si extensibilitatea** - arata gradul in care un SO poate fi utilizat si adaptat pt. o aplicatie specifica.

Caracteristicile SOTR

- **Fiabilitate si disponibilitate** - proprietatea unui sistem de a se defecta cat mai rar si de a evita goluri in functionare (blocaje).
- **Transparenta si vizibilitate** - arata gradul in care SO permite utilizatorului sa obtina informatii despre cum lucreaza el.

6.1. Terminologie:

Functiile SO sunt:

- Gestionarea resurselor SC:
  - Gestionarea unitatii centrale de calcul
  - Gestionarea memoriilor sau a unitatilor periferice
- Controla comunicarea si paralelismul dintre programe sau secvente de programe.

Resurse: Totalitatea disponibilitatilor cu care poate opera SOTR, disponibilitati oferite utilizatorului.

Resurse reale (fizice)

Resurse virtuale

Terminologie:

Resurse reale (fizice): disponibilitatile hardware existente intr-un SC.

- CPU
- Memoriile interne si externe
- Busurile de adrese si de date
- Ceasul de timp real
- Unitatile periferice

Resurse virtuale - apar in cazul in care cantitatea resurselor reale de un anumit tip este mai mica decat suma cererilor de resurse venite din partea utilizatorilor prin intermediul SOTR. In aceste conditii SO poate crea fiecarui utilizator iluzia ca poseda o resursa proprie, chiar daca in realitate exista un singur exemplar. Se utilizeaza la memorii si periferice virtuale.

Terminologie:

Actiunea prin care SOTR este sesizat ca utilizatorul sau programele aplicative au nevoie de resurse se numeste **cerere de resurse.**

Totalitatea cererilor de resurse care trebuiesc satisfacute la un moment dat se **numeste incarcarea sistemului.**

Prin **alocare** se intelege totalitatea actiunilor prin care SOTR reuseste sa satisfaca cererile de resurse.

Dispozitivul care efectueaza alocarea este **alocatorul de resurse**. Acest alocator de resurse lucreaza dupa niste strategii foarte clar definite si programate.

Terminologie:

Un SC lucreaza cu informatii:

- **programe**
- **date**

Prelucrarea informatiilor:

- Secventele de program sunt executate la nivel de instructiune
- Se lucreaza cu date numai daca ele sunt solicitate.

Programe:

Task

Rutina

Subrutina

Subrutine re-entrante

Date:

Date dedicate

Date comune

Terminologie:

Taskurile reprezinta componentele elementare de program care sunt activate de SOTR si executate de catre CPU pe baza unor criterii de prioritate.

Rutinele sunt unitatile de program pentru tratarea rapida a unor evenimente. Ele sunt activate de SOTR la detectarea producerii evenimentului respectiv tot dupa anumite criterii de prioritate.

Subrutinele sunt portiuni de programe activate direct de catre taskuri:

- **subrutine dedicate** care sunt activate de catre un singur task
- **subrutine comune** care pot fi activate de catre mai multe taskuri dar nu pot fi intrerupte
- **subrutine re-entrante** care pot fi apelate de catre mai multe taskuri aparent simultan, fiind intreruptibile.

Terminologie:

Date dedicate: care pot fi apelate si utilizate de catre un singur task. Adresa lor este cunoscuta in acest caz doar de instructiuni din acel task.

Date comune: care pot fi apelate din mai multe taskuri si servesc comunicarii intre elementele de program. Adresele acestor date trebuiesc cunoscute de catre toate taskurile care le utilizeaza.

Planificarea taskurilor se referă la găsirea de soluții fiabile pentru asignarea procesorului, pentru fiecare task în parte, asfel încât să nu existe suprapuneri ale execuției lor pe durata operării sistemului.

Procedura vs. functie vs. rutina

- O **procedură** reprezintă numele unui bloc dintr-un cod, la fel ca o subrutină, dar cu facilități suplimentare. De exemplu, poate accepta parametri, ce pot fi intrări, ieşiri sau de trecere (pass-through).
- Astfel, cuvântul cheie PROCEDURE există doar în câteva limbaje de programare şi a dispărut din multe altele. Limbajul C nu îl foloseşte, iar limbajele Modula şi Pascal au atât PROCEDURE cat si FUNCTION

- O **funcție** este considerată în anumite limbaje ca fiind o procedură care returnează o valoare. Totuşi, este de obicei termenul folosit pentru a referi orice bloc cu cod ce poartă un nume. De notat că limbajele bazate pe C folosesc exclusiv cuvântul de cod *function*.
-

Procedura, functie, rutina

- **Funcțiile** acceptă parametri, returnează valori şi sunt ținute de obicei separat de codul programului principal. Multe limbaje de programare au o funcție specială (în C, funcția *main*) desemnată ca punct de intrare a unui program.

- O **subrutină** este o secvență de cod executată la cerere, separată de fluxul principal al programului. Interpretorul va sări la acel cod, îl va executa şi va returna programului principal. In limbajul de asamblare se utilizeaza o comparație, salt (jump) şi returnare (return), completate cu offset-uri, adrese de memorie şi/sau etichete.
-

Procedura, functie, rutina

- **Subrutinele** sunt considerate de mulți ca fiind greu de întreținut, dificil de citit şi utilizat, rezervate pentru uz atunci când nimic altceva nu merge (precum instrucțiunea GOTO.

  O secvență dintr-un cod, separată de corpul principal, ce îndeplineşte o operațiune discretă de exemplu, un document se poate referi la aşa-numitele subrutine de tratare a fişierelor.

Notiunea de task

- Un **task** este un program sau o parte a unui program care reunește activități independente sau autonome, apte de a fi rulate simultan cu altele.
- De exemplu, un program de procesare a textului și un program de calcul tabelar care rulează în același timp pe un sistem desktop sunt, fiecare dintre ele, task-uri. Acestea oferă posibilitatea de a face un calcul şi de a edita un text simultan.

  Dar procesorul de text poate, de exemplu, să permită editarea unui document şi tipărirea unui al doilea document, ambele acțiuni în același timp, posibil, al doilea în background.

Notiunea de task

- Din punctul de vedere al implementarii, **taskurile TR** reprezinta cele mai mici unitati de prelucrare ce pot fi planificate. Întregul sistem consta dintr-un **set de n taskuri** {ti}i=1,n, care coopereaza pentru îndeplinirea cerintelor de implementare. Fiecare task **ti** are asociat un set de proprietati:

  ti(ai, ri, ti,max, di, pi, li, vi(t), Ri)

  unde:

- **ai** - momentul aparitiei taskului **ti** pe un anumit procesor - este fie momentul crearii, fie cel al transferarii de pe un alt procesor. Dupa acest moment, taskul va fi luat în considerare de catre planificator.
- **ri** - momentul la care taskul poate sa-si înceapa executia, este pregatit (ready); unele taskuri sunt dependente de altele, neputând fi lansate în executie pâna la terminarea primelor; aceste taskuri sunt legate prin relatii de precedenta (îsi comunica un rezultat la sfârsitul prelucrarii) sau de cauzalitate; uneori momentul **ri** coincide cu **ai**

Notiunea de task

- **ti,max** - timpul de executie pentru cazul cel mai defavorabil (worst case execution time - WCET)
- **di** - momentul, termenul limita la care taskul îsi poate termina executia - deadline
- **pi** - perioada de activare, în cazul în care **ti** este un task periodic
- **li** - prioritatea (urgenta, criticalitatea) taskului **ti**
- **vi(t)** - o functie valoare, optionala, care descrie importanta pentru sistem ca taskul **ti** sa-si termine executia înainte de deadline
- **Ri** - setul de resurse necesar executiei taskului.

Notiunea de task

- Cunoasterea apriorica a parametrilor taskurilor nu este doar dificil de realizat, dar implica si o limita superioara în cerinta de resurse.
- Planificarea realizata pe baza acestor valori, asigura însa fiabilitatea sistemului. În timpul executiei, parametrii oscileaza între o valoare minima si cea maxima calculata, un mare numar de resurse ramânând nefolosite pentru un procent destul de mare de timp.
- Pe baza parametrilor **ri**, **ti**,**max**, **di**, planificatorul determina urmatorii parametri:
- **si** - momentul la care taskului **ti** i se aloca (scheduled) resursele **Ri** necesare executiei
- **ci** - momentul terminarii executiei (completed); acest moment nu corespunde neaparat momentului **si+ti**,**max**, pentru ca taskul poate fi suspendat temporar din executie de catre un task mai prioritar

Prioritate

- Nu toate taskurile au aceeasi importanta pentru un sistem. Importanta fiecarui task se ilustreaza print-un parametru al taskului, cel de **prioritate**. Prioritatile se definesc de catre utilizator sau sistem, functie de evolutia sistemului ele se pot modifica, fiind **dinamice** sau pot ramâne constante, caz în care se numesc **statice**. Taskurile pot fi **clasificate** în trei categorii in functie de prioritate:

- **taskuri TR critice** - *hard real-time tasks* - îndeplinirea constrângerilor lor temporale este esentiala pentru sistem, depasirea deadlines neputând fi tolerata si ducând la efecte catastrofale; au prioritatea cea mai mare; acestor taskuri în multe sisteme, li se rezerva static resursele, aceasta fiind o garantie necesara, dar nu suficienta pentru a trata toate situatiile dinamice ce ar putea aparea; în general, numarul taskurilor critice este foarte redus comparativ cu numarul total de taskuri ale sistemului

Notiunea de task

- **tri** - timpul de raspuns al taskului, perioada dintre momentul când taskul poate fi activat - **ri** si cel când își termina executia - **ci**, deci **tri=ci-ri**
- **tli** - timpul de latenta al unui task, adica timpul cu care un task poate fi întârziat, fara a depasi deadline, deci **tli=di-ri-ti,max**
- **li** - laxitatea taskului - pentru îndeplinirea constrângerii de timp **di**, deadline, taskul trebuie sa-si înceapa executia cel mai târziu la momentul **li=di-ti,max**
- **ui** - rata de utilizare a procesorului, **ui=ti,max/pi**.
- Ansamblul acestor parametri trebuie sa verifice relatiile:**ai<ri<si<ci-ti,max<di-ti,max**.
- În unele sisteme TR, termenul limita, deadline, este modelat printr-o functie de timp, numita functie valoare. Functia valoare **vi(t)** descrie utilitatea pentru sistem a terminarii taskului la momentul **t**. Functia **vi(t)** are valoarea maxima daca **ti** își termina executia înainte de deadline **di**.

- **taskuri TR esentiale** - *soft real-time tasks* - îndeplinirea constrângerilor lor temporale este importanta pentru sistem, dar depasirea deadlines poate fi tolerata
- **taskuri neesentiale** - *non-real-time tasks* - nu sunt direct asociate activitatilor TR ale aplicatiei; au prioritatea cea mai scazuta, planificarea lor pentru executie se realizeaza dupa executia celor din primele doua categorii.

Periodicitate

Dupa modul de succesiune al momentelor de lansare în executie **si**, taskurile se clasifica în:

- **taskuri periodice** - un task **ti** este periodic, cu perioada **pi** daca este reexecutat dupa fiecare durata de timp egala cu **pi**
- **taskuri aperiodice** - timpii de activare nu sunt periodici, depinzând de un set de conditii de activare; de obicei, aceste taskuri sunt necritice
- **taskuri sporadice** - au o natura aperiodica, având de obicei constrângeri stricte (*hard deadlines*).
- Majoritatea taskurilor unui sistem sunt periodice.

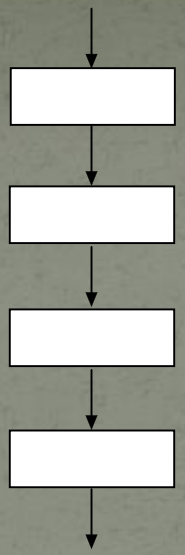
Algoritm

- un tipar de gândire pentru rezolvarea unei probleme intr-un numar finit de pasi prin care datele de intrare ale problemei sunt transformate in date de iesire.

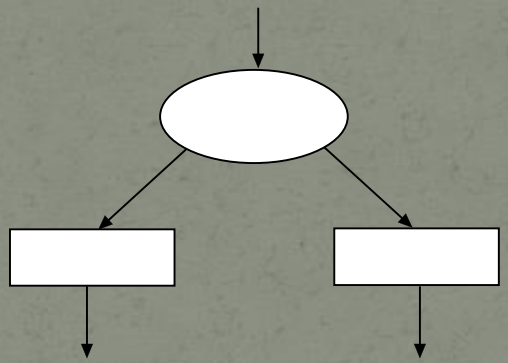
- Un algoritm este usor de reprezentat daca operatiile ce compun ratonamentul problemei sunt corect organizate in grupuri numite **structuri.**

- structuri **liniare**
- structuri **alternative**
- structuri **repetitive**

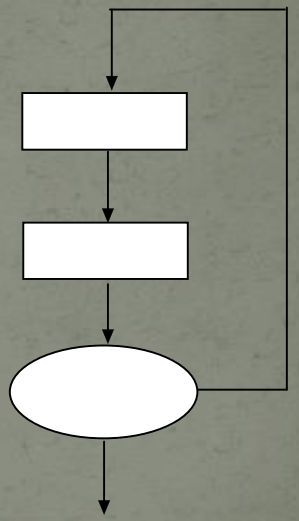
- structuri liniare - compuse din una sau mai multe operatii care se executa secvential de la prima la ultima



- structuri alternative - compuse dintr-un punct de decizie si ramificatii



- structuri repetitive - compuse din una sau mai multe operatii a caror repetitie este controlata printr-un punct de control



- Exemplu de algoritm

•Pasii calculul mediei artimetice a două numere sunt:

1 -start;

2 -citeşte primul număr;

3 -citeşte al doilea număr;

4 -calculează suma lor;

5 -împarte rezultatul la 2;

6 -afişează rezultatul calculat;

7 -stop.

Moduri de exprimare a algoritmilor

- **Scheme logice**
- diagrame de blocuri

- **Pseudocod**
- propozitii scurte
- cu cuvinte cheie predefinite
- exprimate in engleza sau romana

Pseudocod in limba romana:

Pseudocod	Semnificatie
inceput	inceputul/sfarsitul unui algoritm sau al unui bloc de operatii
sfarsit	
citeste	introducerea datelor de la tastatura
scrie	afisarea datelor pe ecran
daca A	
atunci	punct de control sau decizie luata in functie de conditia A
altfel	
sfarsit daca	
repeta	secvente repetitive conditionate
pana cand A	
cat timp A **executa**	
sfarsit cat timp	

Pseudocod in limba romana:

Pseudocod	Semnificatie
pentru contor =vi, vf **executa** **sfarsit pentru**	secventa repetitiva cu contor
←	atribuire
//	comentariu

Problema:
Scrieti un algoritm pentru determinarea proprietatii de numar par (pseudocod)

Problema:
Ce valori afiseaza urmatoarea secventa de operatii pentru x=3 si y=4

```
citeste x,y
    a <- x
    y <- x
    x <- a
scrie x,y
```

Problema:
Ce valori afiseaza urmatoarea secventa de operatii pentru x=6 si y=7

```
citeste x,y
    z <- x+y
    a <- z-x
    y <- x
    z <- y
    x <- a
scrie y,x
```

Problema:
Propune valori de intrare pentru x si y astfel incat urmatoarea secventa de operatii sa afiseze valorile 3 si 8

```
citeste x,y
    z <- x+y
    x <- z-y
    y <- z-x
scrie y,x
```

Problema:
Propune valori de intrare pentru x,y si z astfel incat urmatoarea secventa de operatii sa afiseze valorile 3,2 si 1

```
citeste x,y,z
    a <- x
    x <- z
    z <- a
    y <- y div 3
scrie x,y,z
```

Schemele logice

- Schemele logice sunt notații grafice formate din blocuri legate între ele prin săgeți
- O schemă logică descrie grafic paşii unui algoritm
- Totodată ea specifică prelucrările care se execută asupra datelor

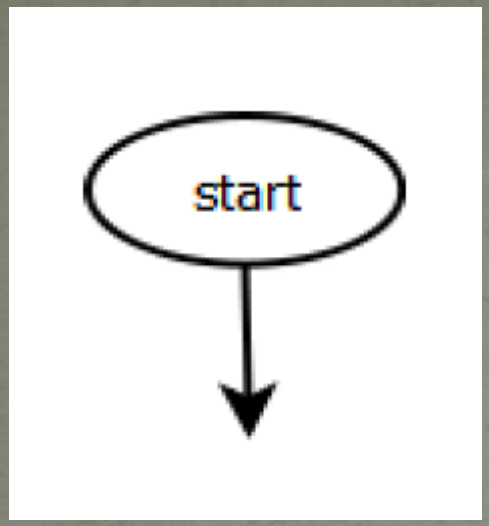
Operatii de baza - blocurile

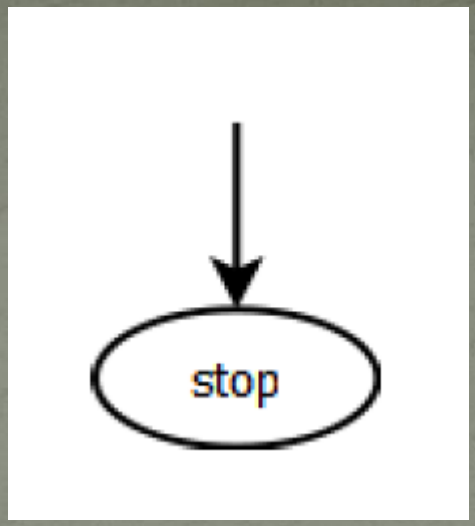
- Blocul de start
- Blocul de stop
- Blocul de citire
- Blocul de scriere
- Blocul de atribuire
- Blocul de decizie

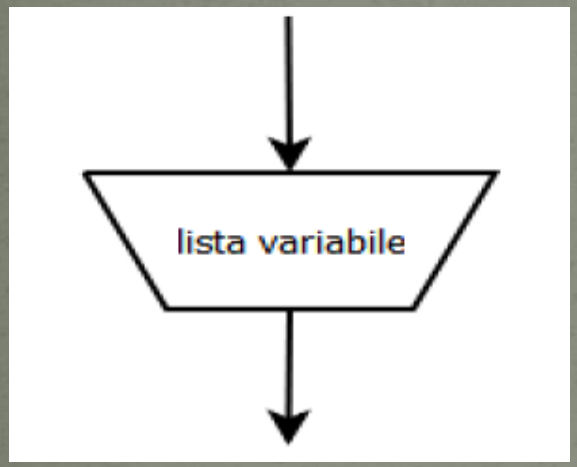
Orientarea blocurilor

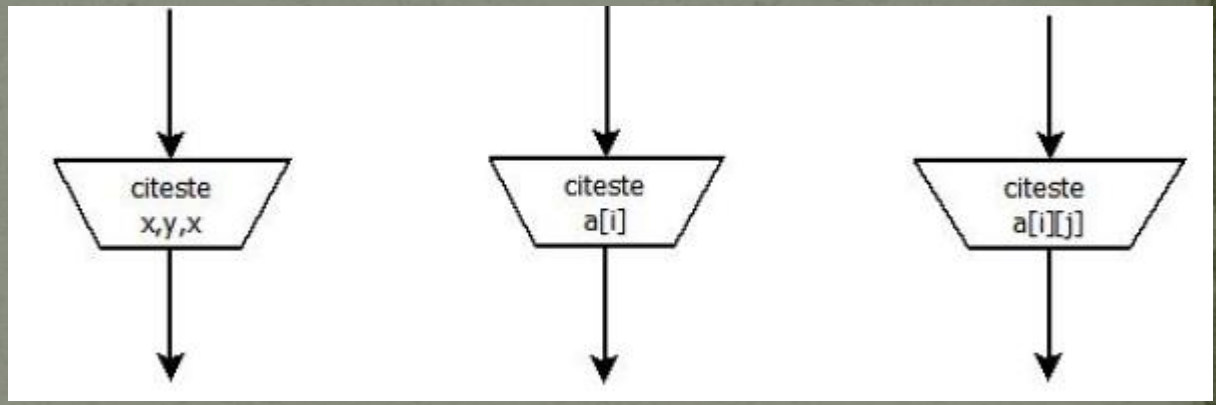
- “Curg” pe pagina de sus in jos
- Se dezvolta arborescent
- Sunt legate intre ele prin sageti directionale
- Conectori de pagina
  - leaga blocuri aflate pe aceeasi foaie de hartie
  - cerculete cu numere in ele
- Conectori intre pagini
  - leaga blocuri aflate pe foi de hartie diferite
  - sagetute pline cu numere in ele

LucidChart
yEd

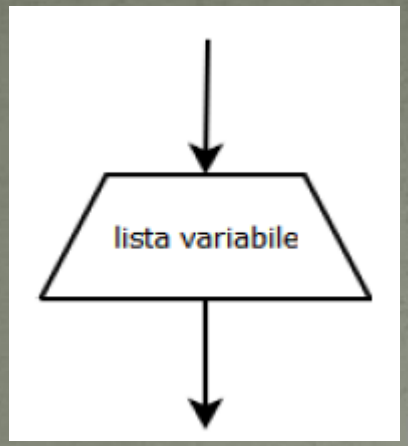


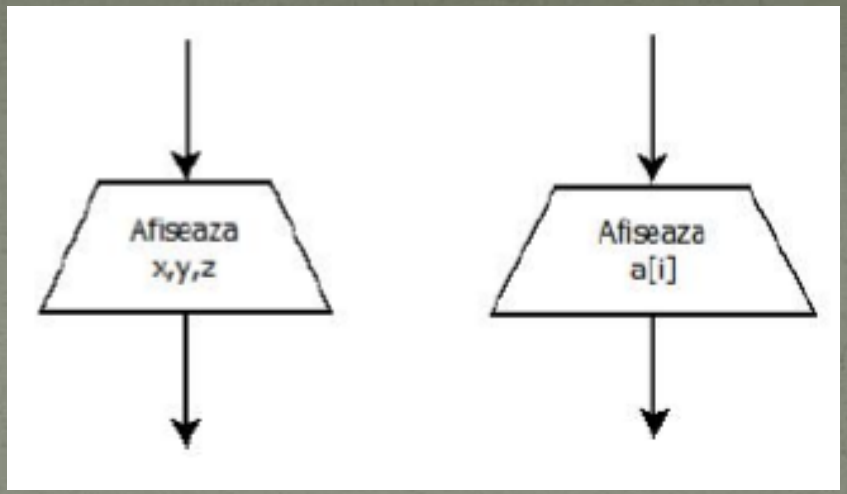






Bloc de citire

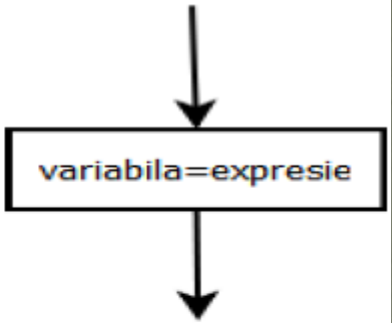




Bloc de scriere

De preferat:

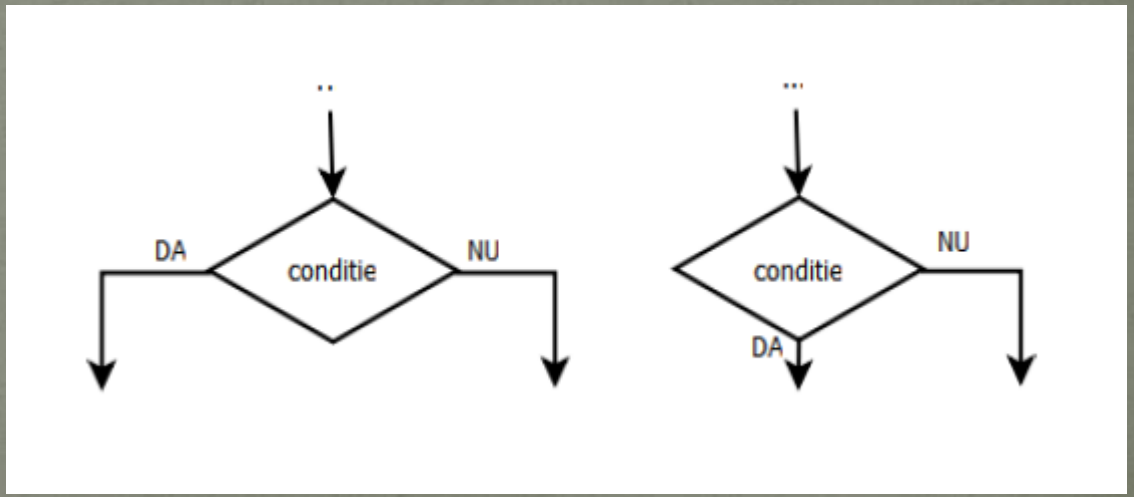
Blocul de atribuire



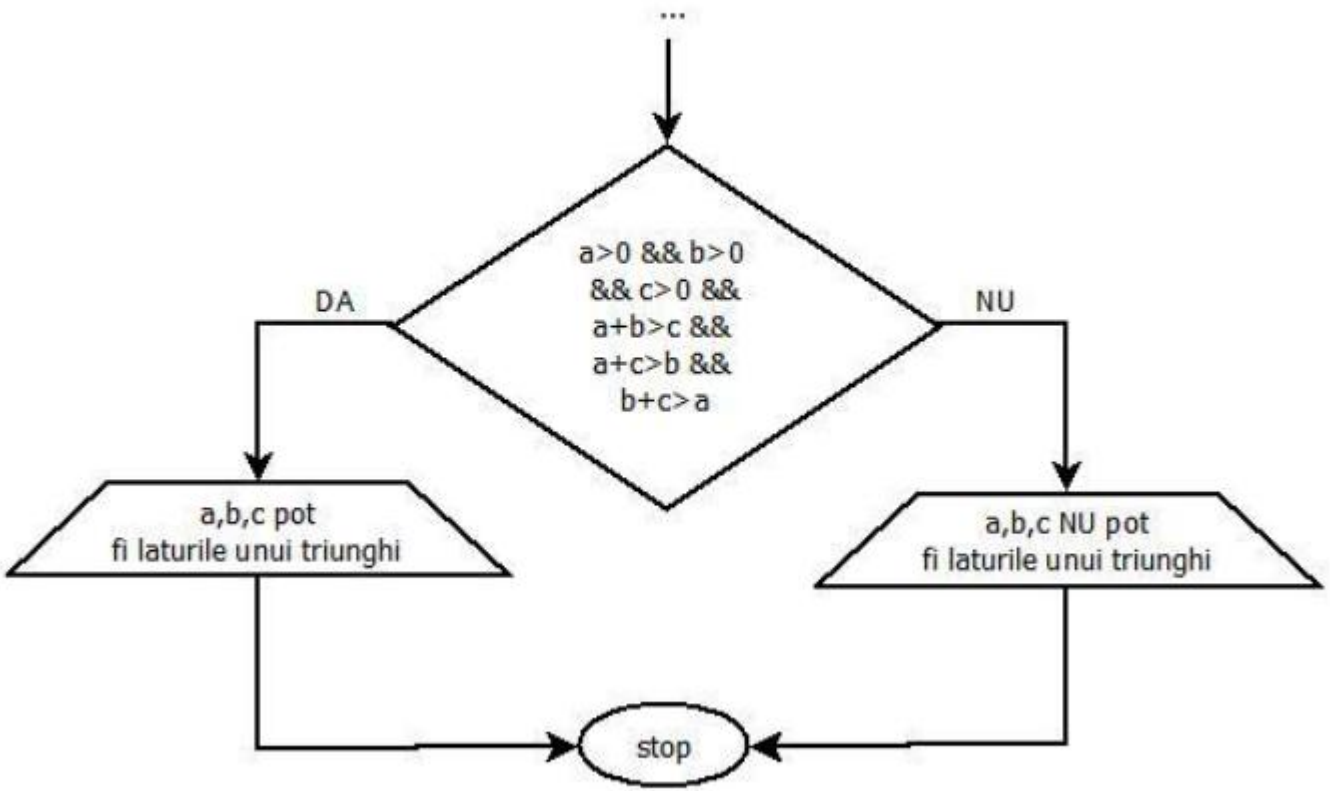
- are rolul de a da valori noi unor variabile
- valorile anterioare ale acelor variabile se vor pierde
- unele atribuiri se pot folosi de valorile “vechi” ale variabilei ce primeşte noua valoare

i++

Blocul de decizie



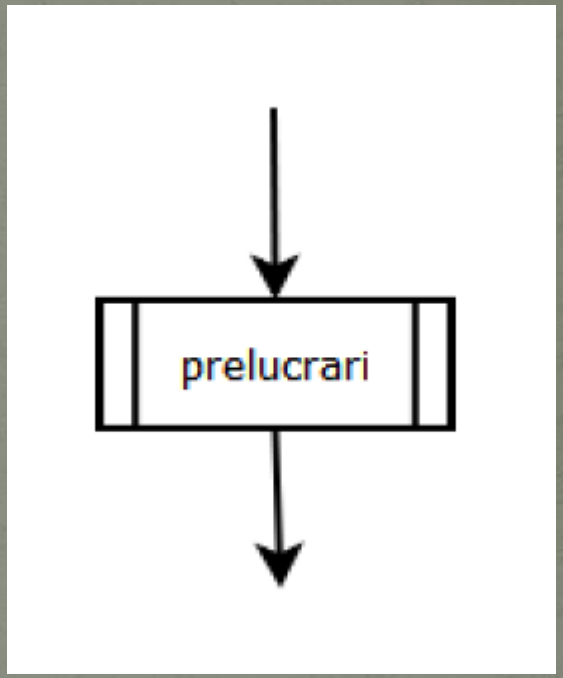
Exemple de utilizare



Structuri de control

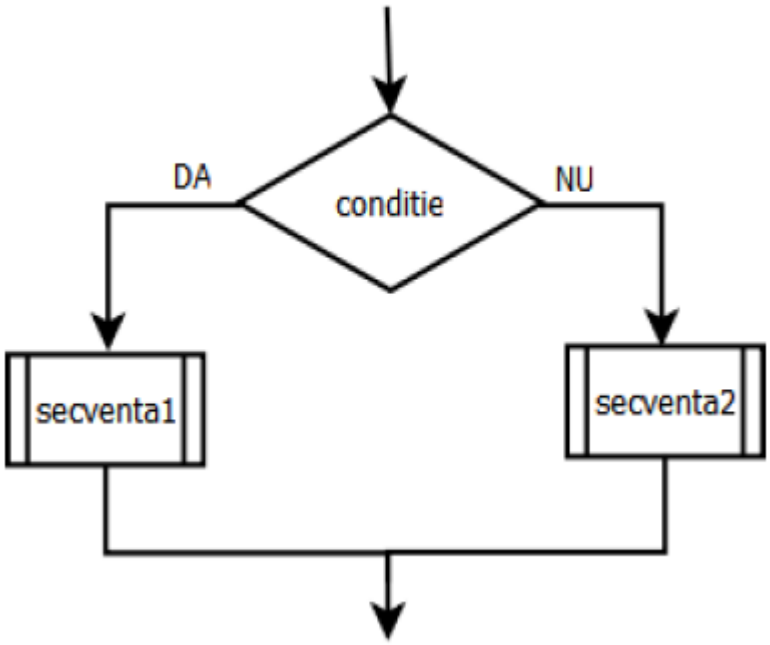
Secventa:

- cea mai simpla structura de control
- presupune executia unui sir ordonat de operatii de baza
- de exemplu o secventa poate cuprinde
  - o citire
  - doua atribuiri
  - o decizie



Selectia:

- are rolul de a selecta o secvență din două pentru execuție în funcție de valoarea condiției
- conține
•un bloc de selecție
•cele două secvențe ce se execută atunci cand condiția evaluată este adevărată respectiv falsă
- Diferența între un bloc de selecție şi o structură de selecție
•este aceea că o structură de selecție totdeauna va include un bloc de selecție
•adică blocul de selecție este parte componentă a structurii de selecție pe lângă cele două secvențe

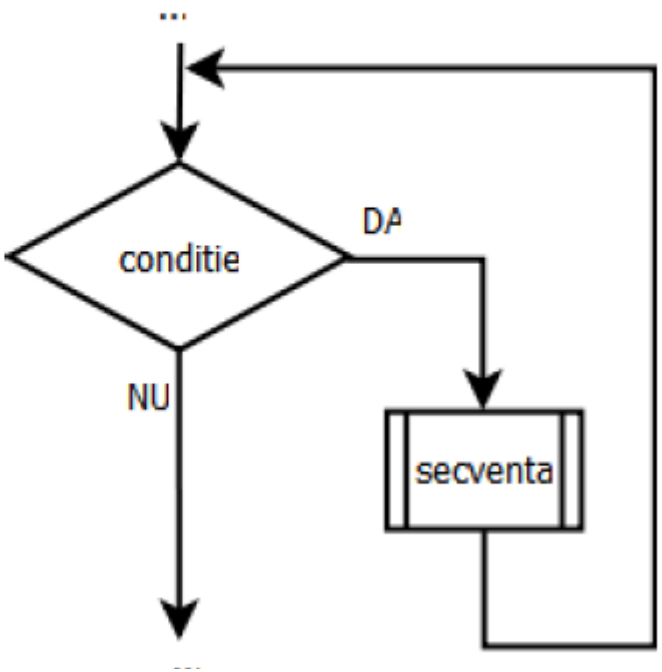



```
dacă condiție
{
        instrucțiuni1
}
altfel
{
        instrucțiuni2
}
```

Structuri repetitive

- - Structura cu număr necunoscut de repetiții cu test inițial (CÂT TIMP sau WHILE)
- - Structura cu număr necunoscut de repetiții cu test final (EXECUTA - CÂT TIMP sau DO-WHILE)
- - Structura cu număr cunoscut de repetiții (PENTRU sau FOR)

Ciclul cu test initial



Functionare

- cât timp condiția este adevărată
  - se va executa secvența în mod repetat

```
cât timp condiție
{
        instrucțiuni
}
```

Ciclul cu test intial

- Scopul unei astfel de construții este:
  - să avem o condiție adevărată când traversăm inițial blocul condițional
  - se execută secvența care are rolul de a modifica variabilele condiției
  - după un număr finit de treceri condiția trebuie să devină falsă
  - se blocheaza astfel execuția secvenței
  - se continua cu rularea schemei logice prin blocurile care ar urma mai jos

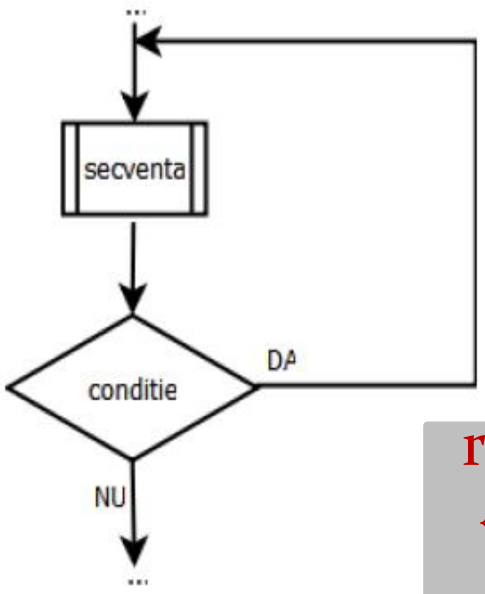
Ciclul cu test intial

- dacă condiția este falsă la prima trecere prin blocul decizional
  - atunci secvența nu va ajunge să fie executată niciodată

- dacă condiția este tot timpul adevărată
  - algoritmul se va bloca într-o buclă infinită, fără a mai putea ajunge vreodată la blocul de stop

Ciclul cu test final

executa
{
instructiune
}
pana cand conditie



repeta
{
instructiune
}
pana cand conditie

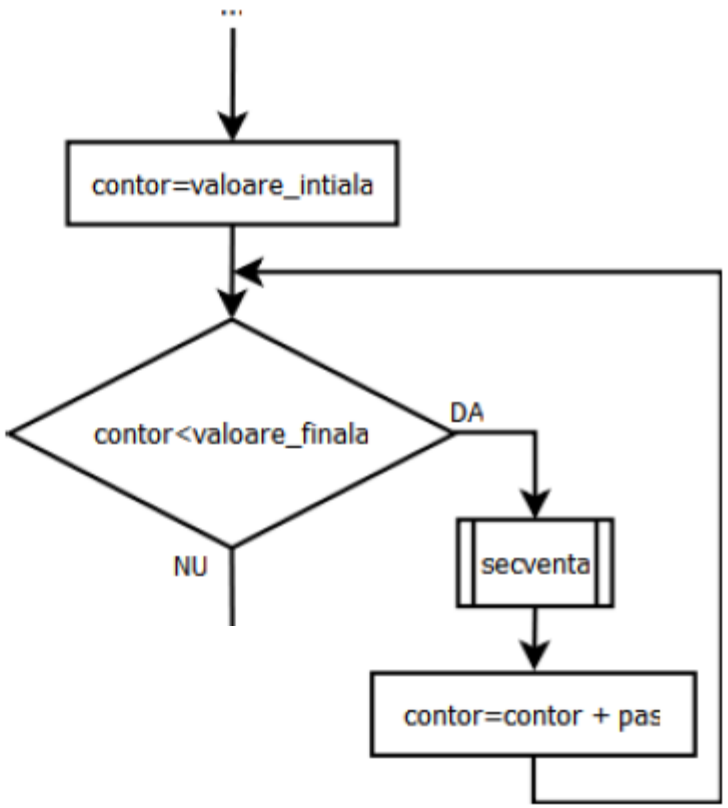
- se execută secvența
- după aceea se evaluează condiția
- dacă condiția este adevărată se merge din nou și se execută secvența

Ciclul cu test final

- Se poate observa că această structură va permite execuția secvenței cel puțin o dată indiferent de ce valoarea logică conține condiția
- nu este strict necesară
- orice algoritm se poate concepe folosind numai structuri de control ciclu cu test inițial
- în practică sunt situații când structura cu test final oferă o soluție mai elegantă din punct de vedere al concepției algoritmului

Ciclu cu contor

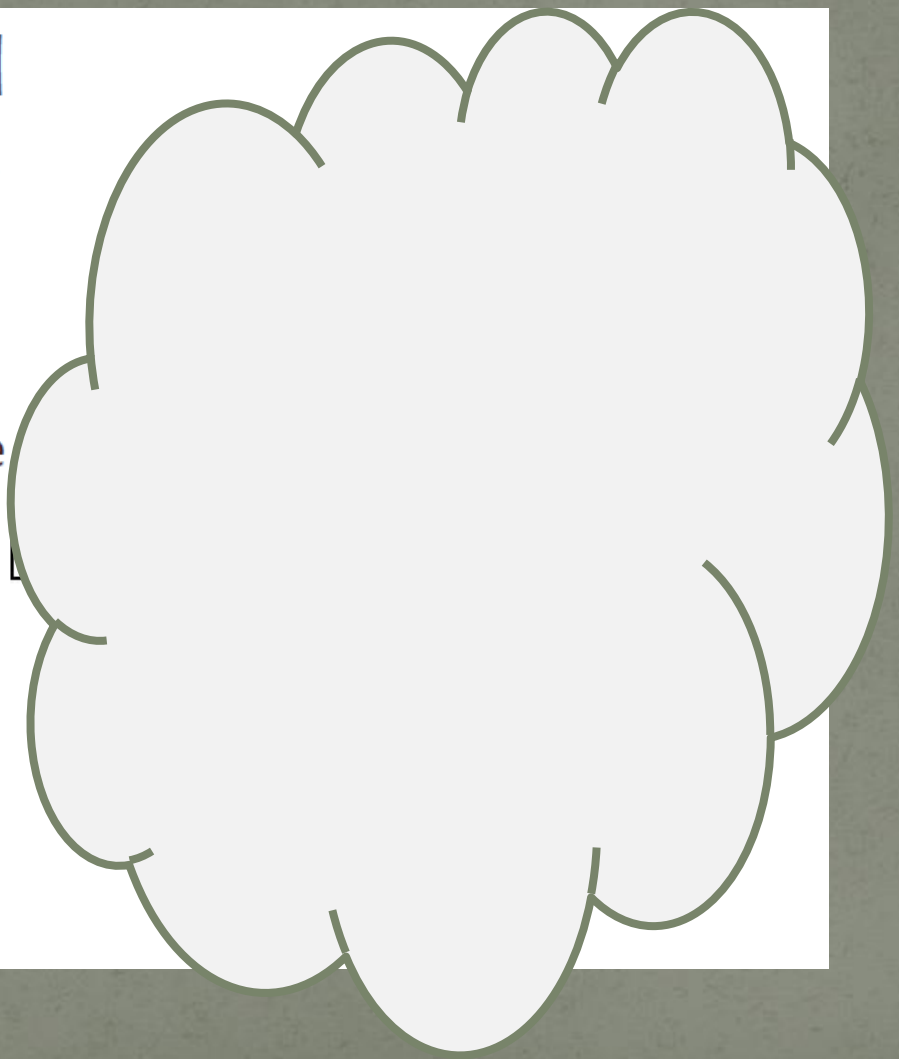
```
pentru contor
 {
executa
        instructiune
}
```



- se bazează pe un contor care
  - primeşte o valoare inițială
  - parcurge toate valorile unui interval continu
  - până se atinge o valoare finală
  - se executa la fiecare pas o aceeaşi secvență

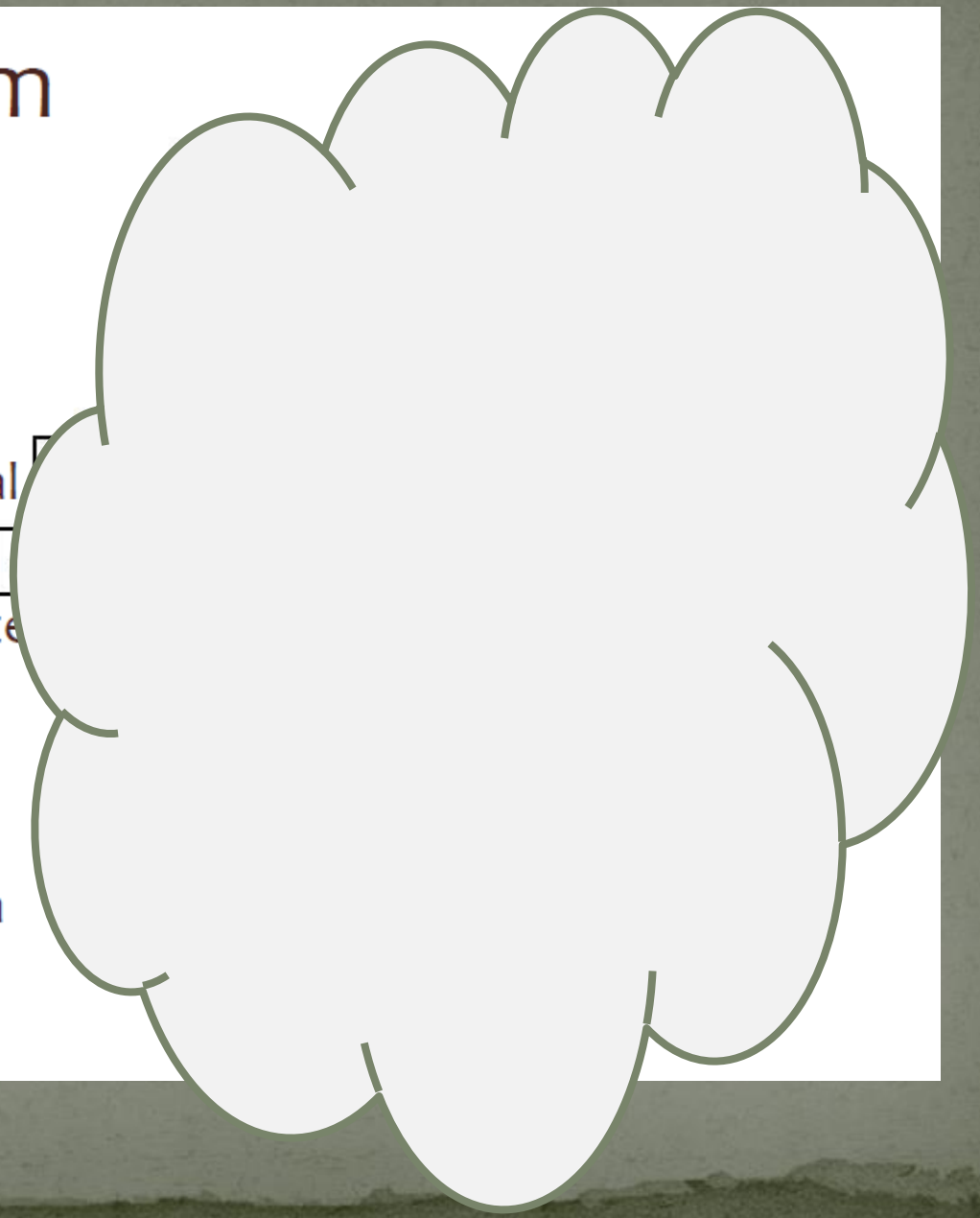
Functia modul

- Citim un numar de la tastatura
- Daca numarul este pozitiv
  - Atunci modulul este egal cu numarul
  - Altfel modulul este egal cu numarul negat
- Afisam pe ecran modulul numarului



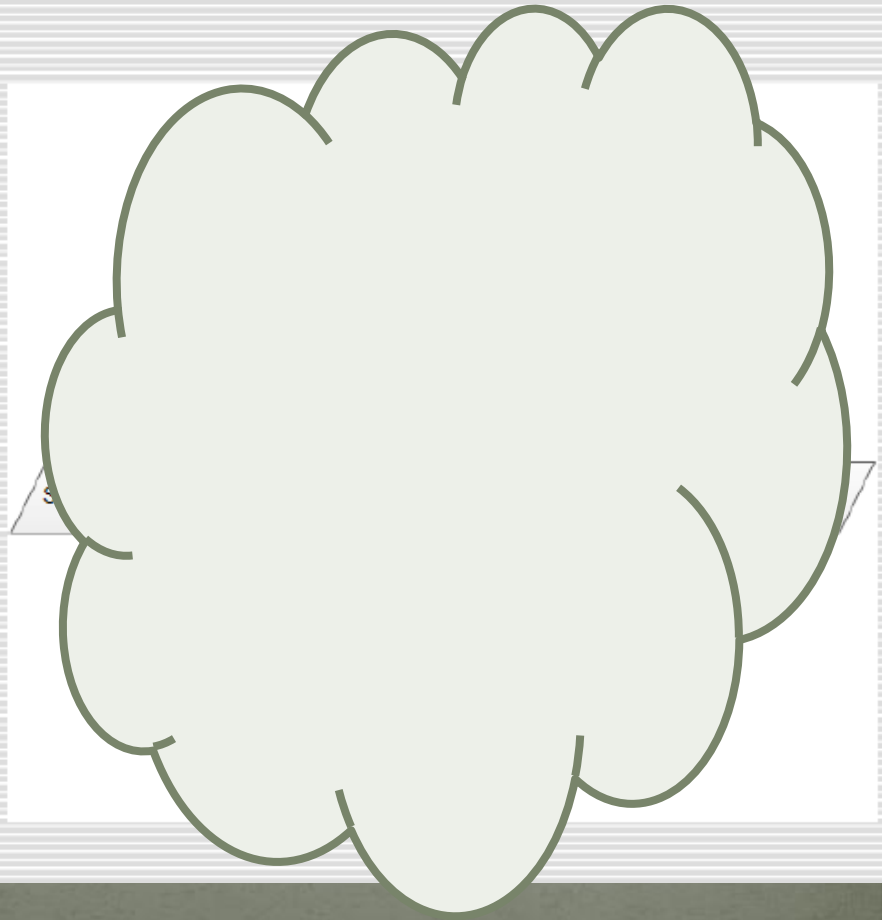
Functia signum

- Citim un numar de la tastatura
- Daca numarul este pozitiv
  - atunci signum este egal cu 1
  - altfel daca numarul este nul
    - atunci signum este zero
    - altfel signum este -1
- Afisam pe ecran functia signum



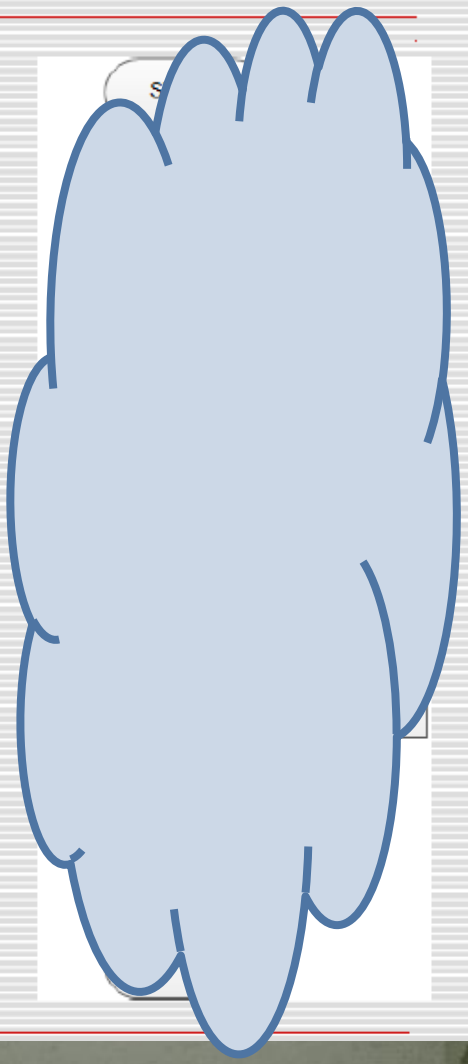
Maximul a trei numere (varianta 1)

- O primă variantă de rezolvare a problemei găsirii celui mai mare dintre 3 numere citite a, b și c este algoritmul reprezentat prin schema logică alăturată
- Se compară mai întâi primele 2 numere (a și b) și se continuă, în funcție de valoarea de adevăr a expresiei a>b pe ramura corespunzătoare: se compară cel mai mare dintre a și b cu c



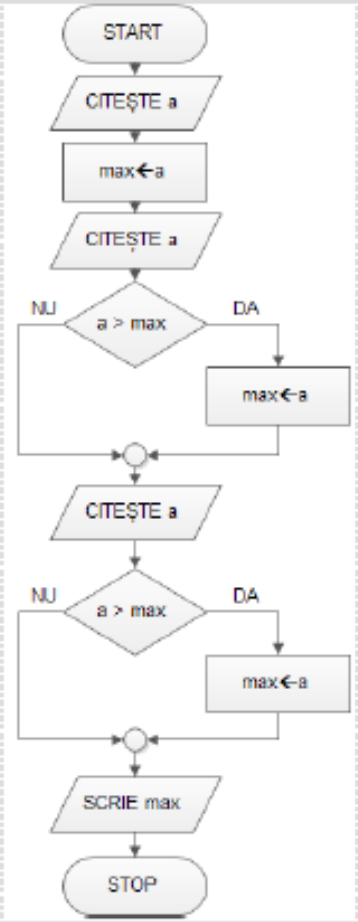
Maximul a trei numere (varianta 2)

- O altă variantă de rezolvare este cea din schema alăturată;
- Se folosește o variabilă suplimentară max, în care se păstrează cel mai mare număr de până la momentul respectiv;
- Avantajul cel mai important este reducerea riscurilor unei erori de programare;
- În locul unui bloc de decizie complex, care avea la rândul său subordonate alte două blocuri, schema alăturate folosește două blocuri de decizie simple;
- Acesta e un principiu pe care îl vom aplica și în viitor: preferăm să rezolvăm mai multe subprobleme simple în locul uneia complicate.



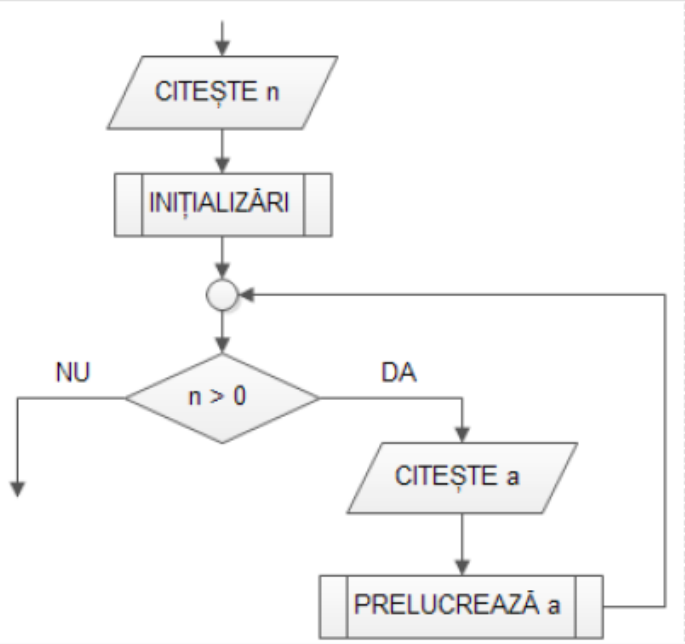
Maximul a trei numere (varianta 2’)

- Schema logică alăturată seamănă foarte bine cu cea anterioară;
- De data aceasta folosim însă o singură variabilă a, în care reținem pe rând valorile tuturor celor 3 numere citite (citirea datelor de intrare se face în 3 pași);
- Principiul de funcționare este asemănător cu cel de la Ping Pong (“învingătorul rîmâne la masă”);
- Variabila max corespunde jucătorului care se află deja la masă, care este cel mai bun dintre cei care au jucat până acum;
- Pe rând vin alți jucători (CITEȘTE a) care se confruntă cu cel aflat deja la masă;
- Dacă îl înving (a>max), îi iau locul (max ← a);
- Un avantaj important al acestei versiuni este acela că poate fi ușor adaptat pentru a calcula maximul a oricâte numere citite.



Forma generală a unui algoritm care prelucrează n numere citite

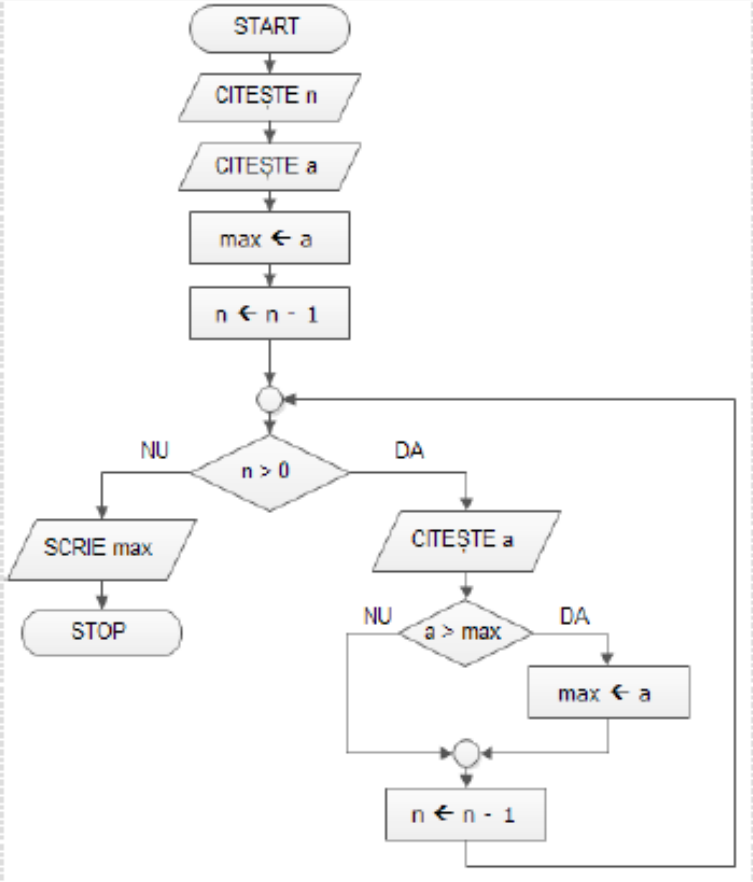
- Schema alăturată corespunde unui algoritm general de prelucrare a unui șir de n numere;
- Mai precis, se citește mai întâi n, reprezentând lungimea șirului (numărul elementelor acestuia), iar apoi, în cadrul buclei repetitive, câte un element, a, al șirului, care este prelucrat imediat după citire (prelucrarea diferă de la o problemă la alta).



- -Decrementare

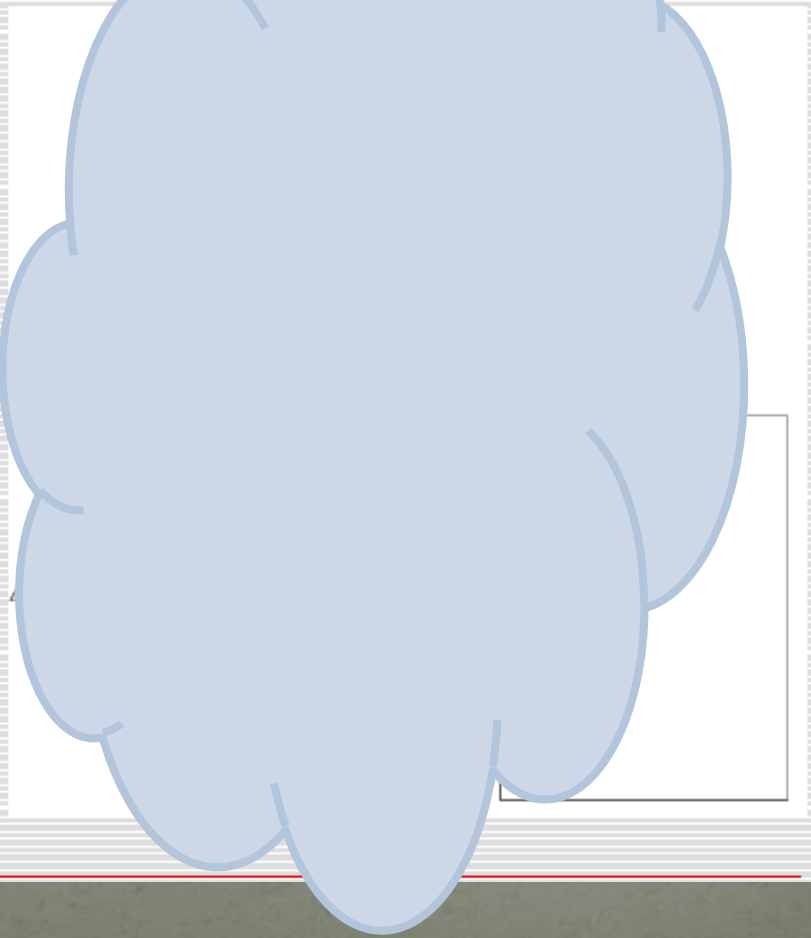
Algoritm pentru calculul maximului a n numere (n oarecare, citit)

- Algoritmul alăturat citește n și apoi n numere, care sunt reținute pe rând în variabila a și determină maximul lor;
- De fapt, pe parcursul rulării algoritmului semnificația valorii reținute de n este "câte numere mai avem de citit și prelucrat";
- După citirea și prelucrarea fiecărui număr, n scade cu 1;
- Principiul de funcționare este din nou "învingătorul rămâne la masă";
- Algoritmul combină ideile slide-urilor precedente: respectă șablonul pentru prelucrarea a n numere, iar fiecare număr este tratat la fel ca în cazul determinării maximului a 3 numere (e comparat cu maximul de până la acel moment și în cazul în care e mai mare ca acesta, îi ia locul).



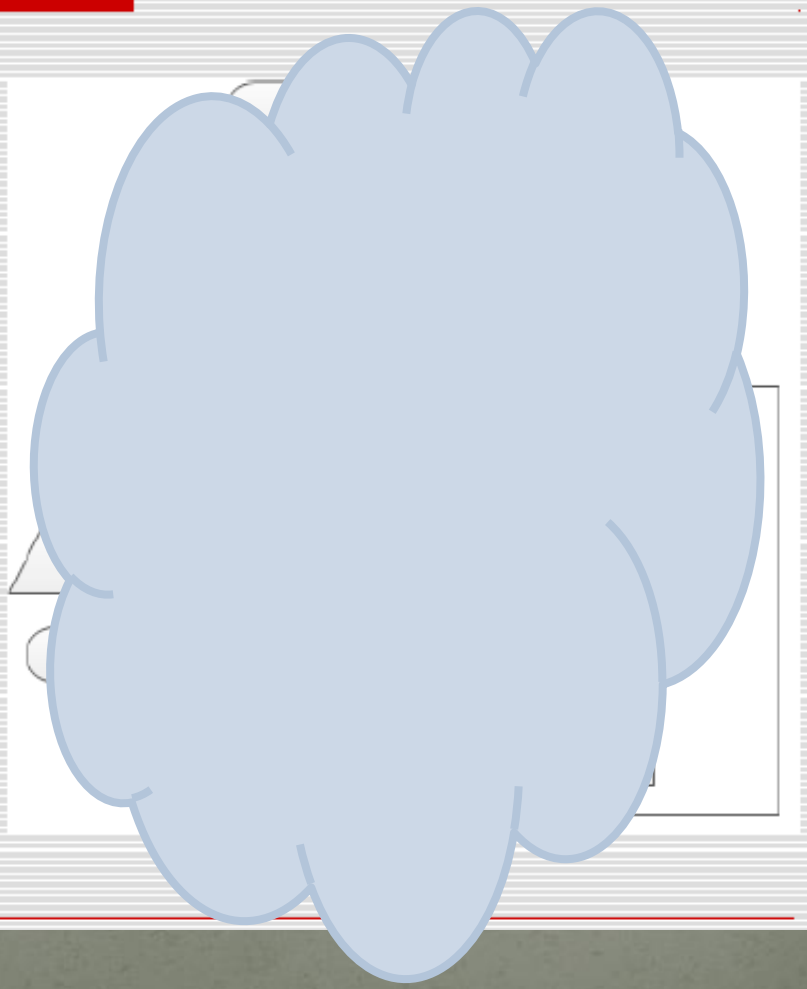
Algoritm pentru calculul sumei a n numere (n oarecare, citit)

- Un alt exemplu de prelucrare a unui șir de n numere (cu n citit înainte) este calculul sumei elementelor șirului;
- Schema logică alăturată diferă de cea pentru determinarea maximului doar în ceea ce privește prelucrarea termenului curent a.



Algoritm pentru calculul sumei a n numere – varianta 2

- O variantă mai simplă de rezolvare a problemei anterioare se bazează pe faptul că 0 este element neutru pentru adunare;
- Dacă inițializăm s cu 0, atunci putem introduce prelucrarea primului element al șirului în bucla repetitivă, ca în schema alăturată.



- Sa se calculeze factorialul unui numar natural n introdus de la tastatura.

- Sa se afiseze multimea divizorilor proprii ai unui numar natural n, introdus de la tastatura(n<1000).

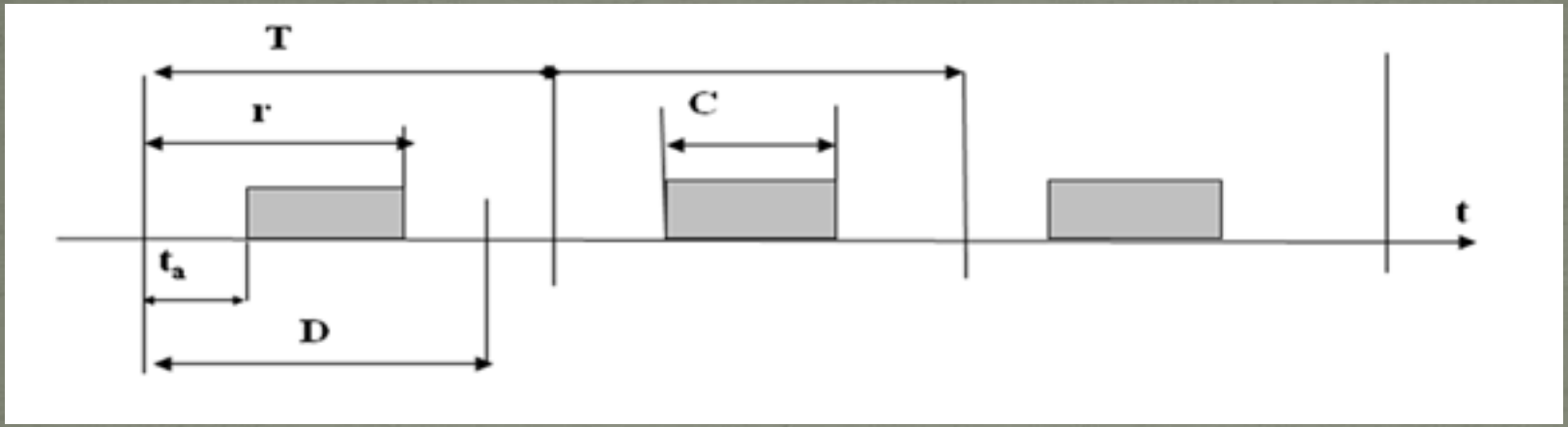
continuare Curs 7

- **Task** - unitate elementara de program

- **Planificarea taskurilor** se referă la găsirea de soluții fiabile pentru asignarea procesorului, pentru fiecare task în parte, asfel încât să nu existe suprapuneri ale execuției lor pe durata operării sistemului.

- Taskuri **periodice**
- Taskuri **neperiodice**

Fiecare task este format din:



T – perioada de repetiţie

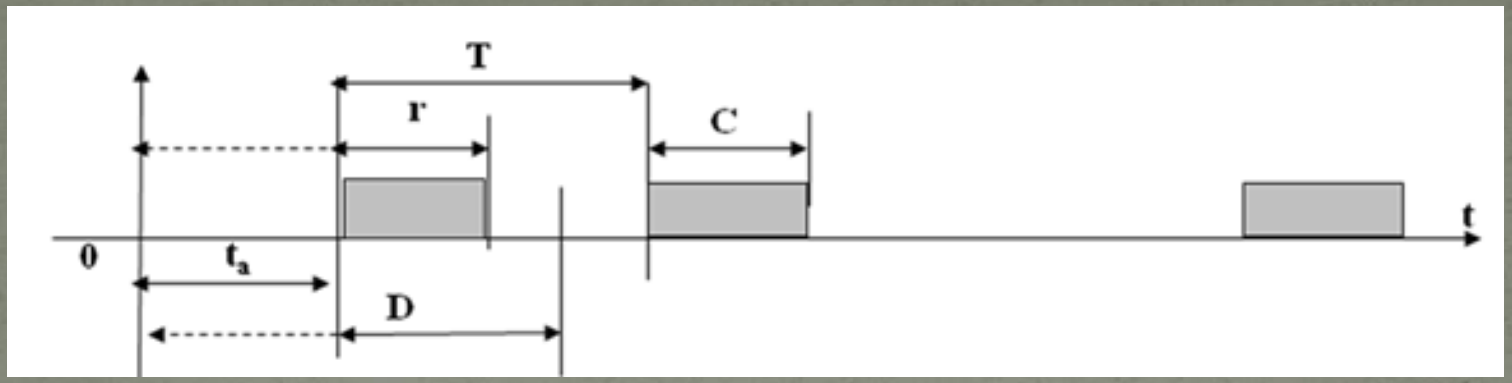
D – timpul limită maxim (deadline) - timpul până la care execuţia taskului trebuie să se încheie

t_a – timp de apariţie – determină momentul în care taskul este disponibil pentru execuţie

C – timp de execuţie / calcul – durata maximă a taskului

r – timp de răspuns – timpul în care execuţia taskului se încheie

Task periodic



T – perioada minimă de repetiţie (opţional)

D – timpul limită maxim (deadline) - timpul până la care execuţia taskului trebuie să se încheie

t_a – timp de apariţie – determină momentul în care taskul este disponibil pentru execuţie

C – timp de execuţie/calcul – durata maximă a taskului

r – timp de răspuns – timpul în care execuţia taskului se încheie

Task aperiodic

Abordarea Multi-Tasking

- O bucla de control: un singur “task”
- Foreground/background: doua task-uri
- Generalizare: task-uri multiple
  - Numite si procese, thread-uri
  - Fiecare proces se executa in paralel
    - Procesele interactioneaza simultan cu elementele externe
      - Monitorizeaza senzorii, controleaza efectoarele, trateaza , IO etc.
    - Creeaza iluzia paralelismului
  - Cerinte
    - Planificarea proceselor
    - Partajarea datelor intre procese concurente

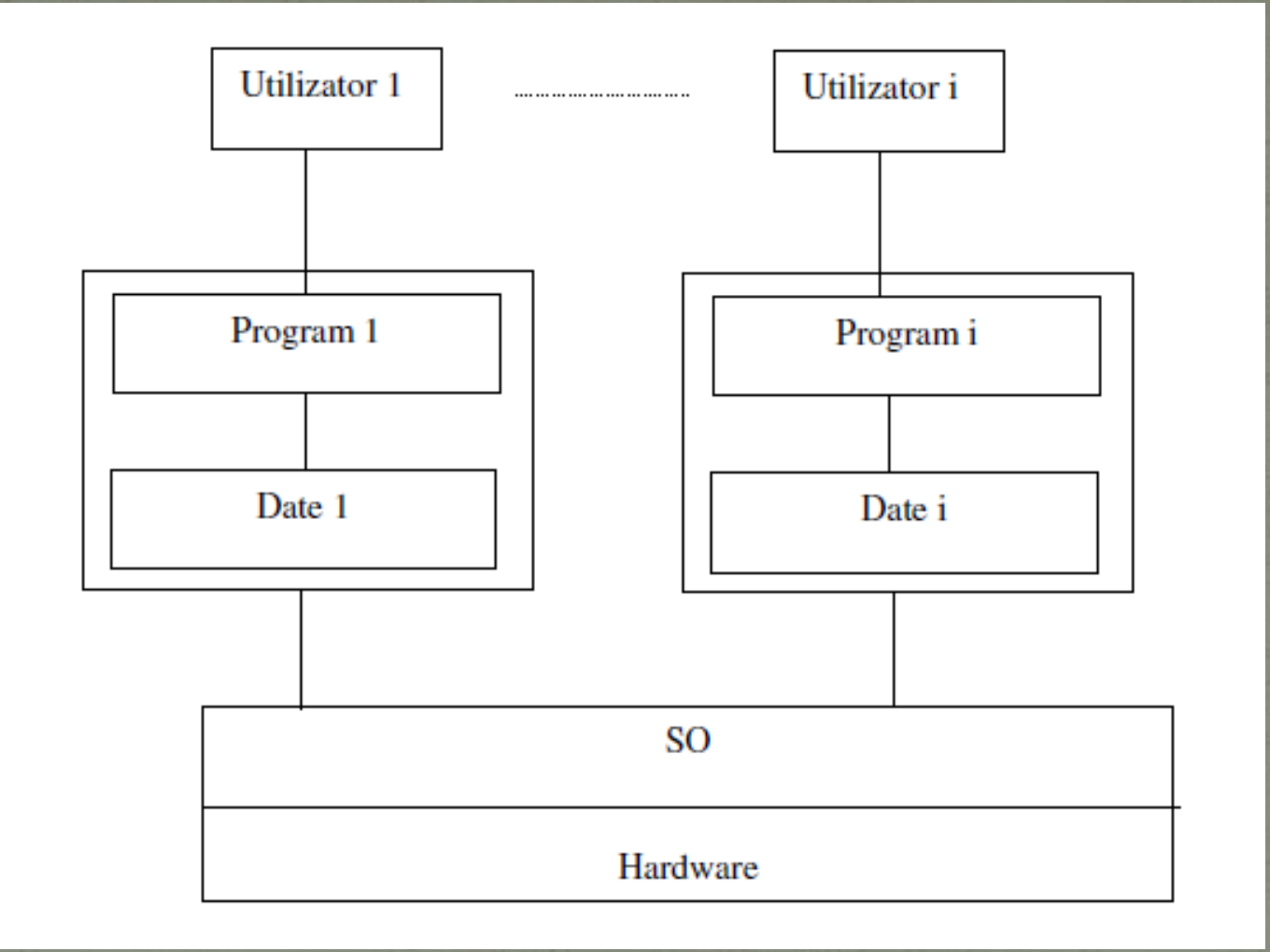
Multitasking

- Ce inseamna Multitasking?
  - Procese separate impart acelasi procesor (sau procesoare)
  - Fiecare task se executa in contextul propriu
  - Detine procesorul pentru cuanta curenta de timp
  - Are variabile proprii
  - Poate fi intrerupt
- Mai multe task-uri pot interactiona si functiona ca un program unitar.

Sistem de operare in timp real multitasking - SOTRM

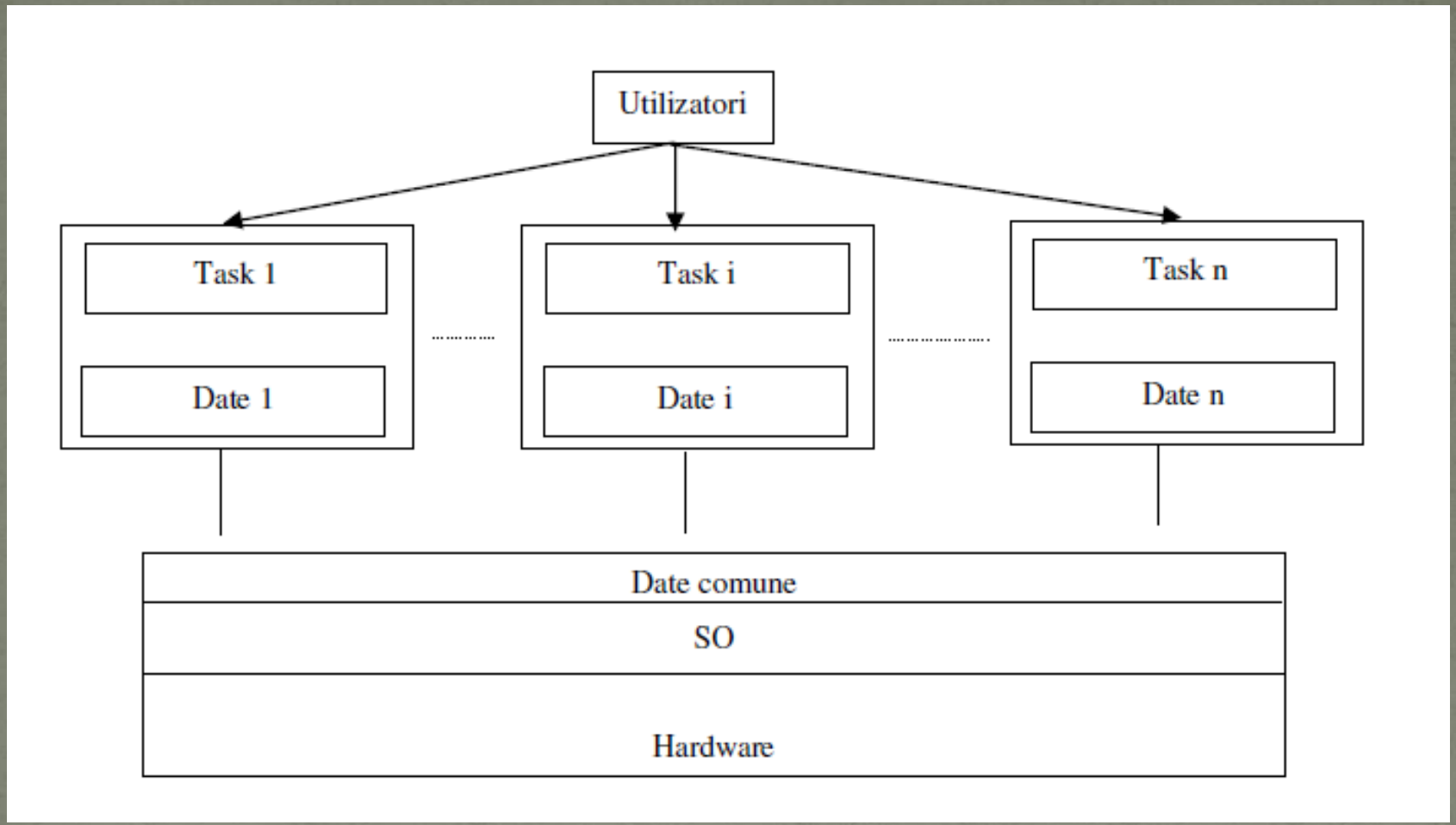
- O metoda uzuala de a structura o aplicatie timp - real este aceea de a realiza un numar de task-uri cooperante care se executa concurent. În mod uzual, proiectarea si implementarea aplicaiei timp -real se poate face mai usor daca sistemul de operare (SO) utilizat suporta prelucrari **multitasking**.
- Este cunoscut ca SO traditionale se bazeaza pe aceea ca toate aplicatiile sau task-urile din sistem sunt executate **concurent** pe un singur calculator 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 aplicatiilor timp - real raportul performanta/cost determina utilizarea unui sistem cu un singur procesor si cu SO adecvat pentru executia concurenta a task-urilor.
- Confuzie între proprietatile **multiuser** si **multitasking** ale SO?
- Proprietatea multiuser a SO asigura ca fiecare utilizator executa propria aplicatie ca si când ar avea la dispozitie toate resursele calculatorului:
- Fiecare program (aplicatie) ruleaza în propriul mediu, protejat fata de celelalte aplicatii.


Utilizator 1
Utilizator i
Program 1
Program i
Date 1
Date i
SO
Hardware

Multiuser

- Proprietatea multitasking a SO se refera la faptul ca acesta gestioneaza mai multe task-uri care coopereaza în cadrul unei aplicatii, în vederea realizarii functiilor pentru care aceasta a fost proiectata. Cooperarea presupune ca task-urile comunica între ele prin mesaje si ca partajeaza date comune.



Multitasking

Avem două tipuri de multitasking:

- **multitasking noncooperativ,**
- **multitasking cooperativ.**

Multitasking-ul non-cooperativ - situaţia în care task-urile sunt programe de execuţie separate.

Task-urile care sunt programe de execuţie separate, sunt numite **procese** în acest caz, multitasking-ul non-cooperativ se numeste **multiprocessing.**

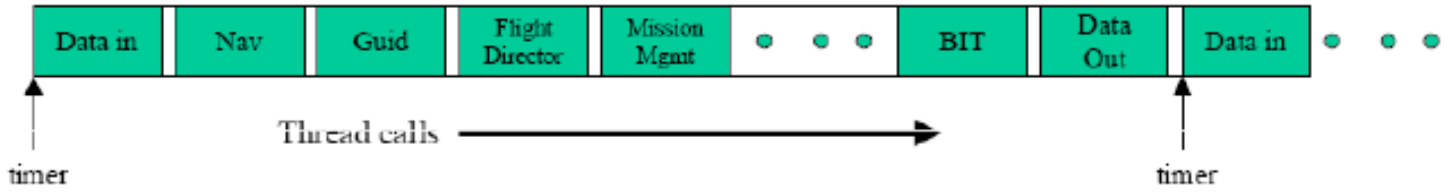
Multitasking-ul cooperativ - situaţia în care task-urile sunt părţi ale unui program sau, cu alte cuvinte, părţi ale unui proces. Task-urile care sunt părţi ale unui program sunt numite **fire de execuţie**, iar în acest caz, pentru multitasking-ul cooperativ se foloseşte noţiunea de **multithreading.**

Multitasking

- Cate task-uri impart acelasi procesor?
  - Sisteme cu executie ciclica
  - Sisteme round-robin
  - Sisteme preemptive

Sisteme cu executie ciclica

- Foloseste o planificare statica pentru a ordona toate firele de executie



- Pro:
  - Usor de implementat (folosite in sisteme critice si de mentinere a vietii)
- Contra:
  - Nu sunt foarte eficiente d.p.d.v. al folosirii CPU
  - Nu permit un timp de raspuns optim (intotdeauna, unele sarcini au prioritate mai mare)

Siteme Round-Robin

- Procesele se executa secvential pana la finalizarea tuturor
- De multe ori in conjunctie cu o schema de executie ciclica
- Fiecarui task ii este alocata o cuanta de timp.
- Exista un timer de sistem care genereza o intrerupere la expirarea fiecarei cuante de timp
- Task-ul se executa pana la finalizare sau pana la terminarea cuantei de timp alocate lui.
- Contextul este salvat sau refacut la fiecare iesire/intrare a procesului intr-o cuanta de executie.

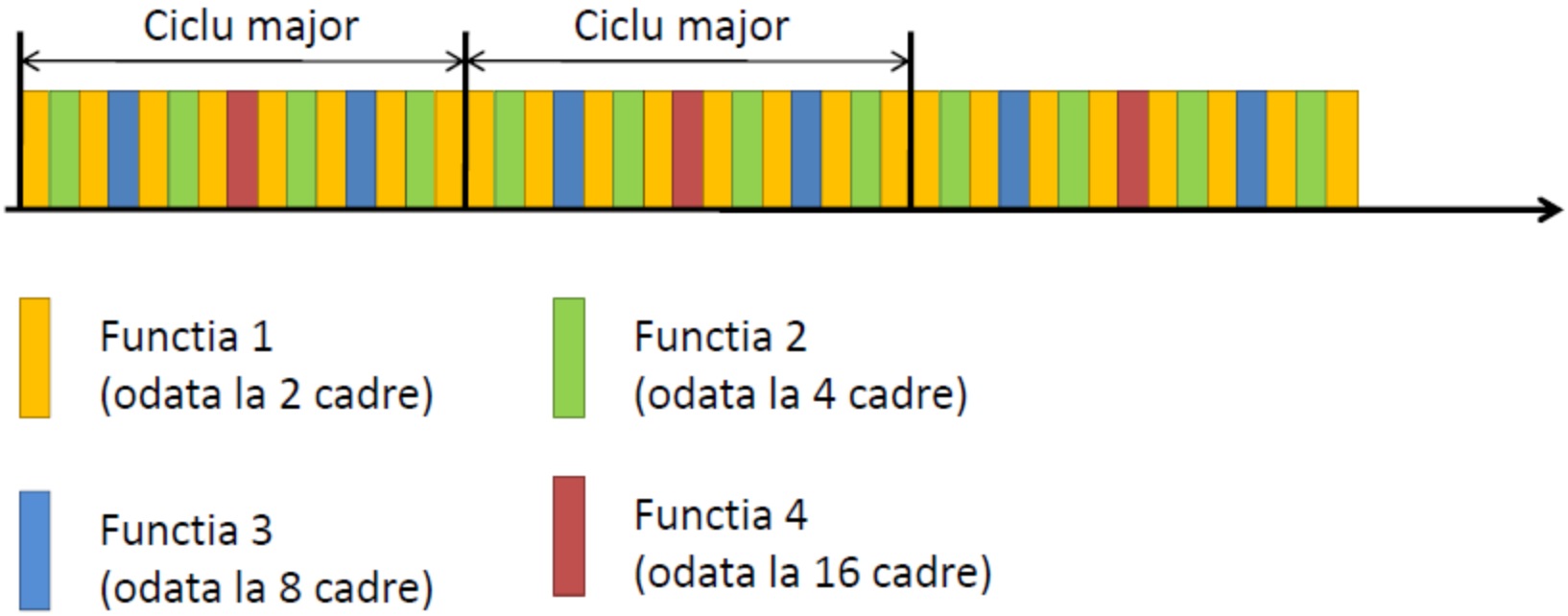
Planificarea Statica

- Aplicabila la task-urile periodice
- Se construieste o tabela si fiecare proces are alocata o cuanta de timp
- Prezivibil dar inflexibil
    - Tabela este total refacuta cand un singur proces isi modifica timpul de executie

- Timpul este impartit in cicli minori (o cuanta) si un timer declanseaza executia procesului planificat pentru cuanta respectiva de timp.
- Un set de cicli minori constituie un ciclu major care se repeta countinuu.
- Operatiile sunt implementate ca niste proceduri si sunt incluse in liste de executie pentru fiecare ciclu minor
- La inceputul unui ciclu minor timerul apeleaza in ordine fiecare procedura din lista respectiva.
- Fara preemptare: operatiile lungi trebuie “sparte” pentru a putea incape intr-un ciclu minor.

Exemplu

- Patru functii care se executa la 50, 25, 12.5 si 6.25Hz (20,40,80 si 160ms) pot fi planificate intr-o executie ciclica cu un ciclu minor de 10ms:



Sisteme preemptive

- Un task cu prioritate mai mare poate sa preempteze pe un al doilea, daca acesta din urma este in executie in cuanta curenta de timp
- Prioritatile alocate fiecarui task sunt bazate pe urgenta executiei fiecarui task in parte
- Prioritatile pot sa fie fixe sau dinamice

Preemptare

- Non-preemptive: procesul, odata pornit nu se opreste decat dupa ce si-a terminat executia
  - » Ex.: N task-uri, fiecare task j apelat la un interval T_j are nevoie de un timp C_j de executie
    atunci: $T_j \geq C_1 + C_2 + \ldots + C_N$ in cel mai rau caz (toate celelalate N-1 task-uri sunt si ele gata)
- Preemptive: un proces poate fi oprit pentru rularea altui proces
  - Complica implementarea
  - Dar putem face mai bine planificare

De ce avem nevoie de planificare?

- **Planificarea:** Dacă procesele nu se vor executa după o anumită planificare se va ajunge la o congestie a resurselor.
- Planificarea este produsă de planificator (scheduler).
- **Planificatorul (Scheduler)** – este modulul care implementează algoritmii de planificare.
- **Planificare validă –** atunci când toate taskurile își ating deadline-urile.

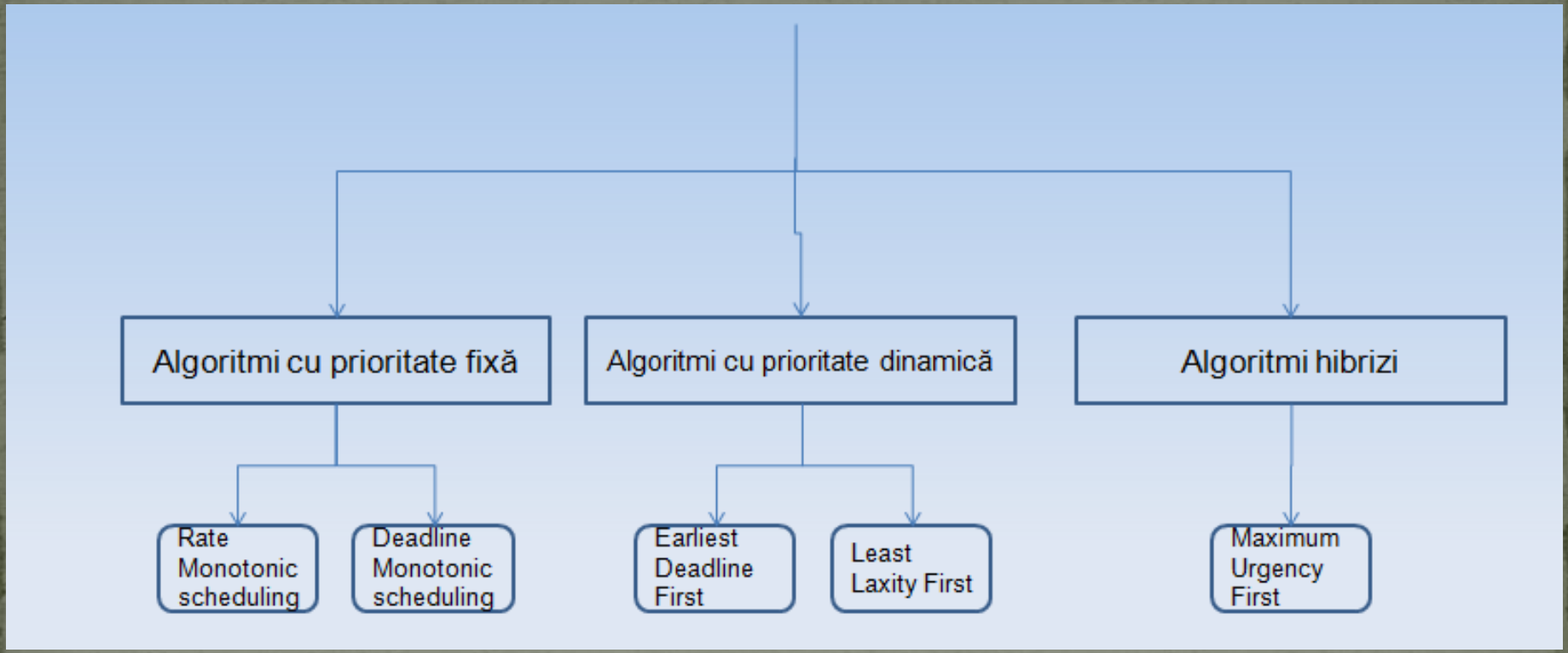
Algoritmii de planificare se pot clasifica:

-după numărul de procesoare sistem:
- în **algoritmi monoprocesor** și
- **algoritmi multiprocessor;**

-după momentul de generare al planificării:
- în **algoritmi statici** (planificare generată offline)
- **algoritmi dinamici** (planificare generată online, în timpul operării sistemului);

-în funcție de acceptarea sau nu a întreruperilor:
- **algoritmi preemptivi** (se admite întreruperea task-ului curent de către un alt task cu prioritate mai mare)
- **algoritmi nonpreemptivi** (nu se admit întreruperi ale taskurilor).

Mecanismele de planificare implementează tehnicile menționate.
Planificatorul folosit pentru planificarea proceselor RT este o unitate de program care controlează lansarea în executie.


Algoritmi cu prioritate fixă
Algoritmi cu prioritate dinamică
Algoritmi hibrizi
Rate Monotonic scheduling
Deadline Monotonic scheduling
Earliest Deadline First
Least Laxity First
Maximum Urgency First

Evaluarea performantelor algoritmilor de planificare

- Cazul static: planificare off-line care asigura ca toate deadline-urile sunt satisfacute
- Cazul dinamic: nici o garantie a priori ca termenul limita va fi satisfacut

Evaluarea performantelor algoritmilor de planificare

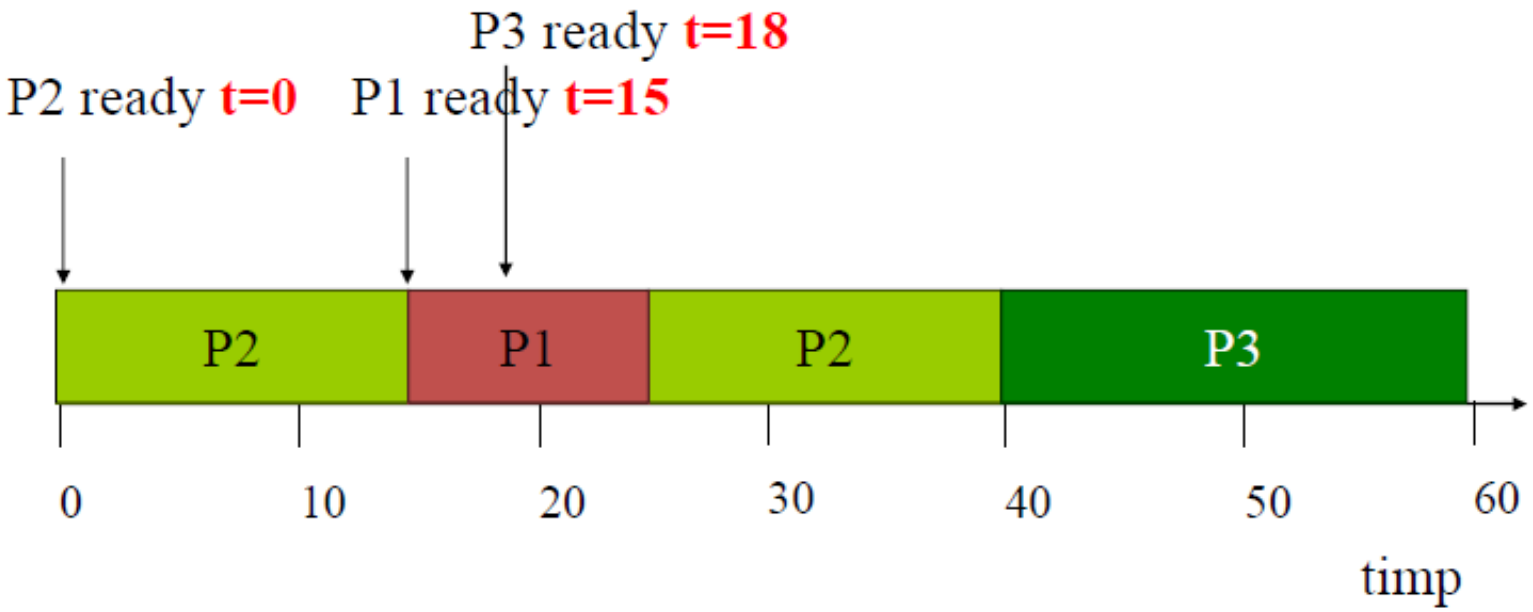
- Cazul static: planificare off-line care asigura ca toate deadline-urile sunt satisfacute
  - Metrica secundara:
    - » Maximizarea numarului mediu de sosiri devreme
    - » Minimizarea numarului mediu de intarzieri
- Cazul dinamic: nici o garantie a priori ca termenul limita va fi satisfacut
  - metrica:
    - » Maximizarea numarului de procese care satisfac termenul limita

Planificarea bazata pe prioritati

- Fiecarui proces i se atribuie o prioritate (static sau dinamic)
  - Atribuirea prioritatilor se face tinandu-se cont de constrangerile de timp
  - Prioritatea statica: atragatoare pentru un sistem simplu (ieftin si nu necesita recalculari)
- La un moment de timp, ruleaza sarcina cu cea mai mare prioritate
  - Daca ruleaza un proces cu prioritate mica, si soseste un altul cu prioritate mai mare, primul proces este preemptat si
- Atribuirea unor prioritati corespunzatoare permite tratarea anumitor situatii in timp real.

Exemplu

- P1: prioritate 1, timp executie 10
- P2: prioritate 2, timp executie 30
- P3: prioritate 3, timp executie 20



Algoritmul **RATE MONOTONIC (RM)**

-cel mai celebru algoritm de planificare;

-se foloseşte pentru planificarea taskurilor periodice;

-priorităţile se alocă în raport cu perioada de repetiţie a taskurilor: taskul cu perioada cea mai mică are prioritatea maximă;

-este un algoritm preemptiv, adică un task mai puţin prioritar poate fi întrerupt în orice moment de un task mai prioritar;

Algoritmul **Early Deadline First (EDF)**

- Algoritm cu prioritate dinamică

- Priorităţile sunt asignate în funcţie de deadline:

  - procesul cu deadline-ul cel mai devreme, are prioritate mai mare;

  - procesul cu deadline-ul mai tarziu, are prioritate mai mică;

Acest algoritm îmbunătăţeşte gradul de utilizare a procesorului în comparaţie cu metoda RM. De asemenea poate trata atât taskuri periodice cât şi taskuri aperiodice (sporadice). Taskurile se consideră preemptibile

Algoritmul **Least Laxity First (LLF)**

- Algoritmul acordă prioritate maximă taskului care are timpul disponibil (laxity time) minim; acest timp se calculează ca diferența între timpul limită (deadline) şi timpul de execuție al taskului; este o măsură a duratei pe care un task o poate petrece în aşteptare. Acest algoritm îmbunătățeşte probabilitatea de succes în comparație cu algoritmul EDF.

Algoritmul First-Come First-Served

- primul sosit primul servit – presupune organizarea unei cozi de aşteptare pentru taskurile ce urmează a fi executate; taskurile vor fi executate în ordinea sosirii, fără să se permită întreruperea taskului în execuție.

- Este de fapt varianta adaptată la unitățile de disc a algoritmului First-In, First-Out (FIFO).

- A spune ca minte B.
- B spune ca minte C
- C spune ca minte atat A cat si B

Cine spune adevarul?