Introducere în problematica sistemelor NESECVENȚIALE

Anca Vasilescu, Oana Georgescu

În primul episod au fost prezentate principalele probleme specifice sistemelor nesecvențiale. În acest număr vom aborda câteva exemple mai complexe de sisteme paralele, concurente sau distribuite. Acest episod este structurat în trei părți: prezentarea sistemelor nesecvențiale din perspectivă cronologică, clasificarea sistemelor concurente dată de Bacon și cerințe pentru realizarea aplicațiilor concurente.

Primele "simptome" de paralelism și concurență

Problematica programării concurente își are originea la începutul anilor '80. Dar, înainte de aceasta, se poate vorbi despre aspecte concurente la nivelul sistemului de operare, mai ales după apariția calculatoarelor paralele și a rețelelor de calculatoare.

Cronologic vorbind, primele "simptome" de concurență s-au manifestat la începutul anilor '50, după apariția sistemelor seriale cu ***multiprogramare***. Acestea au introdus câteva noutăți importante:

- tehnica *polling* (prin care procesorul poate sonda periodic starea perifericelor);
- lucrul cu întreruperi;
- canalul de intrare/ieșire (un procesor specializat pe efectuarea operațiilor de intrare/ieșire) care poate lucra în paralel cu procesorul central.

Mecanismele de mai sus au permis introducerea unor tehnici noi de exploatare eficientă a unității centrale de prelucrare. Dintre acestea, s-au remarcat utilizarea zonelor de memorie de tip tampon (***buffers***) și dezvoltarea tehnicii ***spooling***. *Buffer*-ele de memorie aveau rolul de a regla diferența de viteză între perifericele lente și procesoarele rapide. Tehnica ***spooling*** (*Simultaneous Peripheral Operation On-Line*) se referă, pe de o parte, la posibilitatea citirii în avans a programelor și, pe de altă parte, la posibilitatea afișării asincrone a rezultatelor. Cele două direcții evidențiau un aspect nou de paralelism, concretizat în posibilitatea suprapunerii în timp a execuției operațiilor de intrare/ieșire și a calculelor aritmetice și logice.

Ca o extensie a sistemelor cu multiprogramare s-au impus sistemele ***interactive*** și sistemele ***multitasking***. Principalele caracteristici specifice sistemelor interactive sunt:

- tehnica de deservire a cererilor utilizatorilor în timp partajat (*time-sharing*);
- redirectarea;
- legarea în *pipe*.

Următoarea etapă a fost dezvoltarea ***sistemelor multiprocesor*** și a ***sistemelor multicalculator***. Un sistem multiprocesor rezultă din interconectarea a două sau mai multe procesoare cu memorii și echipament de intrare/ieșire comune.

Într-un sistem multicalculator, sunt interconectate calculatoare prin linii de comunicare și formează o ***rețea de calculatoare***. Într-o rețea avem mai multe calculatoare autonome (independente) care pot sau nu să comunice fiecare cu fiecare. Comparativ cu organizarea unei rețele de calculatoare, un sistem multiprocesor este controlat de un același sistem de operare, care asigură interconexiunea între procesoare și *toate* componentele care concură la realizarea sarcinilor.

Dacă două sau mai multe calculatoare sau procesoare cooperează într-o manieră oarecare, atunci putem spune că totalitatea resurselor lor formează un ***sistem distribuit***. De aici, sistemele ***multicalculator***, ***rețelele de calculatoare*** și sistemele ***multiprocesor*** apar ca fiind cazuri particulare de sisteme distribuite. Din punct de vedere cronologic, rețelele locale (***LAN*** - *Local Area Network*) au fost considerate ca fiind primele organizări fizice pentru sisteme cu prelucrare distribuită.

Există numeroase asemănări între sistemele multiprocesor și sistemele multicalculator. Cele mai multe asemănări rezultă din faptul că ambele tipuri de sisteme suportă operații efectuate paralel și/sau concurent.

La începutul acestui mileniu, în opinia oamenilor de știință, cele trei direcții fundamentale de cercetare care vor prezenta interes vor fi: microprocesoarele, inteligența artificială și prelucrarea distribuită. Din această cauză, se preconizează o orientare în special spre arhitecturile ***non von Neumann***, mai noi și mai puțin studiate… Acestea sunt modele de sisteme de calcul care nu respectă principiile *von Neumann* [6], de exemplu sistemele expert (care se bazează și pe inferențe logice, nu exclusiv pe calcule aritmetice) și sistemele paralele (care nu respectă principiul secvențialității).

Clasificarea sistemelor concurente dată de Bacon

J. Bacon definește foarte simplu conceptul de ***concurență***. ***Concurent*** înseamnă ***în același timp***. [1]

Un sistem concurent trebuie să gestioneze activități separate, dar care se desfășoară simultan. Mai exact, vom considera că două activități sunt concurente dacă, la un moment dat, fiecare activitate se află într-un punct situat între punctul de start și punctul final al desfășurării sale (ambele activități sunt în desfășurare).

Astfel, activitățile concurente pot fi privite din două puncte de vedere și anume: cel al structurii interne a sistemului, rezultând ***sisteme inerent concurente***, sau din cel al aplicațiilor care stau la baza funcționării sistemului respectiv, rezultând ***aplicații potențial concurente***. În cele ce urmează vom lua în discuție caracteristicile acestor entități delimitate de *Bacon*.

Un sistem este ***inerent concurent*** dacă gestionează activități care se pot întâmpla simultan în exteriorul sistemului de calcul, de exemplu utilizatori care doresc să execute programe sau clienți care vor să execute tranzacții.

O posibilitate de obținere a concurenței este determinarea soluției prin aplicarea unui algoritm concurent de divizare a problemei în părți care, la rândul lor, pot fi rezolvate prin activități desfășurate în paralel. Această variantă este recomandată pentru aplicațiile cu volum mare de calcule și de date și în care este importantă cerința de obținere a rezultatelor în timp real. Aplicațiile de acest gen sunt ***aplicații potențial concurente***. Abordarea concurentă îmbunătățește rezolvarea secvențială cel puțin din două puncte de vedere:

- un același răspuns poate fi obținut la același preț folosind un număr mai mare de calculatoare ieftine;
- un model mai performant (mai scump) de realizare a sistemului fizic poate conduce la rafinarea rezultatului problemei.

Sisteme inerent concurente

După cum am definit anterior, sistemele care trebuie să gestioneze simultan activități care se desfășoară în mediul exterior lor sunt cunoscute ca sisteme inerent concurente. Din această categorie fac parte:

- sistemele în timp real;
- sistemele de gestiune a bazelor de date și de prelucrare a tranzacțiilor;
- sistemele de operare.

Este de preferat ca, în fiecare caz, implementarea sistemului să fie distribuită pe mai multe sisteme de calcul (*host*-uri, *noduri*, *site*-uri) decât să se abordeze o metodologie centralizată de organizare și gestionare.

Sisteme în timp real

Principala caracteristică a sistemelor în timp real constă în faptul că funcționarea lor trebuie să țină cont de anumite restricții de timp impuse de mediul înconjurător sistemului respectiv. Importanța restricțiilor temporale impuse împarte sistemele în timp real în:

- sisteme puternic dependente de timp;
- sisteme slab dependente de timp.

Din alt punct de vedere, sistemele în timp real pot fi: ***statice*** sau ***dinamice***. Într-un sistem static, analiza activităților pe care sistemul trebuie să le execute poate fi făcută în etapa de proiectare a sistemului. Într-un sistem dinamic, cererile asupra sistemului pot fi adresate la orice moment de timp, neregulat și imprevizibil. Mai mult, un sistem dinamic trebuie să poată răspunde acestor cereri în timp real, respectând parametrii de performanță impuși de clientul care a adresat cererea.

Cele mai reprezentative exemple de sisteme în timp real sunt cele proiectate pentru ***controlul proceselor***. Câteva exemple sunt sistemele de monitorizare a pacienților unei unități sanitare, sistemele de monitorizare a zborurilor, a motoarelor vehiculelor, sistemele de control al traficului aerian, *controler*-ele pentru roboți, sistemele de procesare și vizualizare de imagini în timp real etc.

În astfel de sisteme de control, la nivel de proces (*process control*), scenariul comun de funcționare se bazează pe faptul că datele sunt colectate (prin testarea sistemelor care generează acele date) și apoi analizate prin algoritmi specifici. Atât colectarea, cât și analiza acestor date, sunt evenimente periodice și previzibile. Totuși, această activitate periodică de colectare și analiză trebuie să poată răspunde unor evenimente neprevăzute. Astfel, se remarcă două tipuri de activități: de monitorizare și de control. Monitorizarea implică colectarea datelor, în timp ce controlul are ca efect corelarea periodică și eficientă a datelor, atât cu scopul funcționării sistemului, cât și cu cererea clientului, care așteaptă un răspuns din partea sistemului.

Sistemele de control la nivel de proces sunt sisteme în timp real, puternic dependente de timp și statice.

Proiectarea sistemelor în timp real trebuie să prevadă situațiile limită în care sistemul însuși trebuie să decidă importanța evenimentelor neprevăzute care apar. Astfel, sistemul trebuie să poată decide la un moment dat, dacă este mai important să monitorizeze datele, astfel încât rezultatele să aibă o precizie maximă sau este de preferat di-

minuarea preciziei și furnizarea răspunsului într-un timp minim posibil, eventual chiar în timp real.

Caracterul concurent al acestor sisteme rezultă, cel mai adesea, tocmai din faptul că mai multe calculatoare mici sunt corelate, astfel încât să controleze un proces industrial sau experimental de dimensiune și/sau de importanță mare.

Nu toate sistemele în timp real sunt sisteme distribuite de tip multicalculator. De exemplu, o aplicație simplă, un robot, un motor de vehicul pot *încapsula* un singur calculator de control, pentru care partea de software va asigura singură atât monitorizarea, cât și controlul întregului sistem. Mai mult, aplicațiile sunt distribuite dacă sunt scrise astfel încât să permită desfășurarea simultană a mai multor activități distincte.

O altă clasă de sisteme în timp real sunt ***sistemele multimedia***. Spunem despre un sistem că este multimedia dacă are posibilitatea să îmbogățească afișarea clasică prin text și grafică cu imagini în mișcare și/sau voce. Dacă presupunem că o stație de lucru are un sistem de navigare bazat pe ferestre, unele ferestre pot fi folosite pentru video. Multe aplicații, cum ar fi: telefonia-video, poșta-video sau videoconferințele, trebuie să afișeze în timp real o persoană care vorbește. În acest caz, partea video (imaginea) și partea audio (vocea) trebuie să se sincronizeze astfel încât, pe ecranul participanților la comunicare, acestea să *ajungă* simultan (vocea trebuie sincronizată în special cu mișcarea buzelor vorbitorului filmat).

Videoconferințele și telefonia-video funcționează în timp real. Dacă sistemul nu memorează nici un fel de date, acestea sunt pierdute după încheierea transmisiei. Poșta-video presupune transmiterea de mesaje electronice care pot conține text, combinat cu voce și cu imagini în mișcare. Pentru aceasta componentele trebuie memorate și transmise la stația care va recepționa mesajul.

După ce procesul de citire a mesajului este declanșat, sistemul trebuie să respecte anumite cerințe specifice caracterului de sistem în timp real cu privire la transmiterea sincronizată a componentelor și la minimizarea ratei de întârziere. Probleme de sincronizare în sistemele multimedia sunt dezbătute în [1].

Stațiile multimedia sunt considerate sisteme în timp real cu atât mai mult cu cât, într-o transmisie de telefonie-video, de exemplu, datele alterate pot fi retransmise numai dacă intervine explicit o voce umană, care să ceară repetarea lor. De aici rezultă că un sistem multimedia este unul slab dependent de timp.

Dacă stațiile care participă la sistemul multimedia sunt și distribuite geografic, atunci sistemul concurent devine și distribuit.

Sistemele multimedia își au originea în sistemele cu timp partajat (*time-sharing*). Diferența este una cantitativă deoarece un sistem multimedia solicită o putere de prelucrare mai mare, lățime de bandă de rețea, capacitate de memorare mai mare și aplicații *software* care să poată valorifica aceste facilități.

Dacă se compară un astfel de sistem slab dependent de timp cu unul de control, se remarcă o diferență calitativă deoarece, într-un sistem de control este mult mai restrictivă cerința pentru ca răspunsul primit să fie în timp real, un sistem de control fiind unul puternic dependent de timp.

În concluzie, cerințele care se impun pentru realizarea unui sistem în timp real ar putea fi:

- să suporte activități independente; dintre acestea, unele pot fi periodice (cum este colectarea de date) sau imprevizibile (cum este cerința de a răspunde la un semnal de tip alarmă);
- să respecte cerințele fiecăreia dintre activitățile în desfășurare;
- să aibă în vedere posibilitatea ca unele activități să poată coopera pentru satisfacerea unor scopuri comune.

Sisteme de gestiune a bazelor de date și de prelucrare a tranzacțiilor

Aplicațiile de baze de date vizează un volum mare de date, cum ar fi datele stocate pe medii permanente. Acțiunea specifică unui utilizator de baze de date constă în interogări asupra datelor din acea bază de date. Spre deosebire de un utilizator, proprietarul bazei de date se preocupă de întreținerea bazei de date, astfel încât datele conținute să respecte caracteristicile unei baze de date consistente, actuale, sigure, corecte.

Prin sistem de gestiune a bazelor de date (***DBMS*** - *DataBase Management System*) ne referim la un sistem de date proiectat să interacționeze cu utilizatorii săi și să organizeze citirile și scrierile de date în sistem, astfel încât să răspundă optim cerințelor utilizatorilor. Un sistem *DBMS* este unul concurent deoarece trebuie să satisfacă simultan cerințele mai multor utilizatori.

În cele ce urmează, prin ***tranzacție*** denumim generic o cerere a unui client adresată bazei de date, care are proprietatea de a nu altera consistența bazei de date. Cel mai simplu exemplu de sistem de prelucrare a tranzacțiilor este serviciul oferit de un automat bancar (***ATM*** - *Automatic Teller Machine*). Considerente legate de proiectarea și organizarea sistemelor de tip *DBMS* se pot găsi în [2]. O problemă importantă care apare în *DBMS* constă în soluționarea conflictelor generate la nivelul cererilor (interogărilor) adresate bazei de date. Accesul concurent (simultan al mai multor clienți la baza de date) este o soluție viabilă atât timp cât se reușește să se scadă timpul de răspuns al *DBMS* la cererile utilizatorilor. Mai mult, din punctul de vedere al bazei de date, tranzacțiile care doar citesc date se pot executa în paralel, indiferent de numărul acestora (vezi problema cititor-scriitor din episodul anterior).

Pentru realizarea unui sistem concurent de gestiune a bazelor de date și de prelucrare a tranzacțiilor se impun câteva cerințe importante cum ar fi:

- să suporte activități independente;
- să asigure posibilitatea ca activități diferite să poată accesa și actualiza date comune, dar numai în limitele păstrării consistenței și corectitudinii bazei de date.

• să asigure depozitarea rezultatelor tranzacțiilor în zone de securitate maximă înainte ca utilizatorul să fie anunțat că tranzacția s-a încheiat cu succes.

Sisteme de operare distribuite

În acest context este important să se deosebească un calculator destinat unui utilizator de un calculator *multiuser*. În prima categorie intră orice sistem de calcul de la calculatoarele personale mici și relativ ieftine până la stațiile de lucru cuprinse în rețele de calculatoare. Sistemele *multiuser* sunt formate din mai multe minicalculatoare sau microcalculatoare performante, destinate fiecare unui număr mic de utilizatori, sau din mai multe sisteme multiprocesor de tip *mainframe* sau supercalculatoare. Fiecare dintre aceste structuri se bazează pe configurații *hardware* specifice, în concordanță cu caracterul concurent al aplicațiilor pe care le vor suporta.

Într-un sistem *singleuser*, tastatura și ecranul utilizatorului sunt "împachetate" împreună cu memoria și procesorul/procesoarele sistemului, formând astfel un singur sistem de calcul. Utilizatorii unui sistem *multiuser* accesează sistemul prin intermediul unui terminal. Terminalele sunt separate de memoria principală, de procesoarele centrale și de perifericele partajate în sistem. Funcțiile unui terminal (stație) sunt, de obicei, orientate pe anumite tipuri de activități, de exemplu un terminal poate dezvolta o aplicație de grafică sau un sistem de navigare bazat pe ferestre. Pentru executarea acestor funcțiuni, un terminal poate avea propria memorie internă și propriile procesoare. Se remarcă faptul că terminalul clasic, format din monitor și tastatură, este completat cu componente care îl transformă într-un calculator cu destinație prestabilită (*special-purpose computer*). Un sistem *multiuser* poate fi unul distribuit dacă are un număr mare de terminale, chiar situate la distanță față de un calculator central și care sunt gestionate în grup prin terminale speciale.

Un sistem de operare de interes general rulează pe calculatoare personale, pe stații de lucru *singleuser* sau pe sisteme partajate. Scopul unui astfel de sistem de aplicații este să execute programele utilizatorilor și să aloce acestor aplicații resursele *hardware* solicitate (memorie, discuri, procesoare etc.).

Pricipalele caracteristici ale unui sistem concurent sunt bine reprezentate atât în sistemele *singleuser*, cât și în sistemele *multiuser* moderne. Putem da două argumente pentru a susține această afirmație: pe de o parte, faptul că perifericele tind spre o structură care le apropie de organizarea pe niveluri a procesoarelor și, pe de altă parte, sistemul de operare, direct răspunzător de starea perifericelor, poate să execute programe utilizator cât timp perifericele execută diverse operații de intrare/ieșire.

Mai mult, într-un sistem *singleuser*, cât timp se execută o activitate de lungă durată (de exemplu, paginarea unui document foarte mare), utilizatorul poate lansa în paralel execuția altor *task*-uri (de exemplu, citirea ultimelor mesaje sosite prin *e-mail*). Pentru aceasta, sistemul de operare trebuie să poată suporta simultan astfel de activități distincte, concurente, prin rularea de aplicații diferite simultan cu gestionarea diferitelor componente din sistem.

Caracterul de sistem concurent este mai puternic la nivelul sistemelor *multiuser*, prin executarea simultană a mai multor aplicații utilizator, prin paralelizarea desfășurării *task*-urilor perifericelor simultan cu satisfacerea comenzilor curente ale utilizatorilor sistemului.

Potrivit clasificării anterioare a sistemelor concurente, putem spune că sistemele de operare gestionează evenimente care sunt mai degrabă neregulate și imprevizibile decât periodice, iar procesul de încărcare a datelor de intrare/ieșire este unul dinamic.

În plus față de considerațiile făcute la nivelul unui sistem *singleuser*, în cazul unuia *multiuser* sistemul de operare trebuie:

• să gestioneze partajarea resurselor sistemului între diverși utilizatori, astfel încât să le ofere tuturor aceeași calitate a serviciilor;
• să răspundă tuturor conflictelor care pot să apară între cererile utilizatorilor asupra resurselor sistemului (memorii, timp de prelucrare, spațiu rezervat fișierelor).

Cererile adresate sistemului de operare apar în mod dinamic și, unele, pot implica o serie de componente adiacente sistemului, care trebuie să fie gestionate tot de sistemul de operare.

Dacă sistemul *multiuser* este unul distribuit, atunci se adaugă noi valențe sistemului de operare. De exemplu, apare posibilitatea ca mai mulți utilizatori să partajeze același sistem de fișiere. Sistemul de operare trebuie să poată gestiona centralizat fișierele comune, trebuie să aibă prevăzute mecanisme specifice de protecție față de erorile de disc sau căderile *software*. Pentru toate aceste motive este de preferat ca sistemul de fișiere să fie furnizat ca un serviciu de rețea (*network-based service*). De asemenea, ar trebui ca un număr rezonabil de *server*-e de fișiere să poată furniza suficient spațiu de memorare și capacitate de prelucrare pentru toate calculatoarele din sistemul distribuit și nu numai pentru sistemele mici. Aceste *server*-e partajate trebuie să răspundă cererilor simultane ale diferiților clienți, deci sunt sisteme concurente.

Sistemele de operare ale calculatoarelor care fac parte din sisteme distribuite trebuie să conțină *software* pentru comunicare. La nivelul cel mai scăzut, gestionarea unei conexiuni de rețea este o acțiune similară cu gestionarea unui periferic de viteză foarte mare. Diferențele apar, mai ales, datorită faptului că datele care sunt lansate în rețea pot fi expediate de oricare dintre stațiile distribuite și nu numai de la un singur periferic.

Calculatoarele pot trimite în sistem date sau mesaje de mare viteză, ceea ce face ca, simultan, pe canalele de comunicație să se transmită secvențe de date de la diferite surse. La nivelurile înalte ale gestionării comunicării în sistemele distribuite se pune problema corectitudinii și completitudinii datelor transmise. Gestionarea acestor situații este un argument în favoarea afirmației că însuși subsistemul de

comunicare din cadrul unui sistem de operare distribuit este un sistem concurent.

În concluzie, principalele cerințe pentru realizarea unui sistem de operare distribuit ar fi:

- să suporte activități independente la nivelul aplicației (deci în exteriorul sistemului de operare), fie mai multe activități ale unui singur utilizator, fie diferite activități ale mai multor utilizatori; cererile pe care aplicațiile le adresează sistemului de operare sunt dinamice și neprevizibile;
- să suporte activități diferite în interiorul aplicațiilor sale de tip: cereri simultane de la mai mulți clienți ai sistemului și diverse *task*-uri de gestionare a componentelor distribuite în sistem;
- să permită diverselor activități să coopereze pentru obținerea unor scopuri comune;
- să gestioneze conflictele care apar la alocarea resurselor din sistem;
- să asigure păstrarea corectitudinii datelor din sistem în urma operațiilor de intrare/ieșire asupra acelorași zone de memorie;
- să asigure corectitudinea execuției unei activități, chiar dacă aceasta a fost descompusă în mai multe *task*-uri care au fost executate în paralel sau concurent cu *task*-urile altor activități.

Aplicații potențial concurente

Față de sistemele inerent concurente (concurente prin structura internă a componentelor lor), sistemele care funcționează pe baza aplicațiilor potențial concurente nu sunt sisteme concurente prin structura lor, ci dobândesc caracteristici de sisteme concurente prin natura aplicațiilor pe care le execută.

Caracterul concurent al unei aplicații poate să rezulte din unul dintre următoarele aspecte:

- există un volum mare de date care trebuie prelucrate;
- există un volum mare de calcule care trebuie efectuate;
- se impune cerința ca rezultatul aplicației să fie furnizat în timp real;
- echipamentele *hardware* permit o eventuală execuție în paralel a activităților concurente.

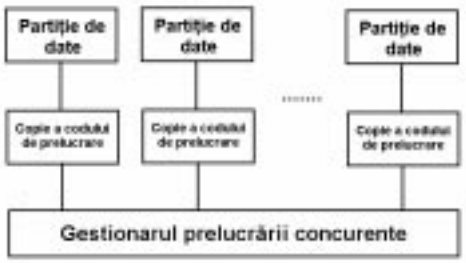
În cele ce urmează vom aborda câteva tehnici de obținere a unei soluții concurente pentru o problemă dată. Exemple concrete de utilizare și implementare a acestor metode se găsesc în [1], [5].

Replicarea codului. Partiționarea datelor

Replicarea (multiplicarea, copierea) codului și partiționarea (împărțirea) datelor sunt două metode prin care un algoritm secvențial poate primi caracteristici de algoritm concurent.

Prin partiționare, volumul mare de date, pe care trebuie să le prelucreze aplicația concurentă, se împarte în secvențe distincte, fiecare secvență fiind ulterior prelucrată de un alt proces/procesor. Programul de prelucrare poate să fie același sau se poate ca partiții diferite să fie prelucrate de programe diferite. În cazul în care programul de prelucrare este unic, se poate prevedea ca fiecare prelucrare să se execute pe baza unei copii a programului respectiv, adică să se aplice o replicare a codului programului de prelucrare.

În figura următoare este reprezentată situația în care soluția concurentă presupune atât partiționarea datelor, cât și replicarea codului.



Replicarea codului. Partiționarea datelor

Într-o astfel de abordare, dacă toate componentele trebuie să funcționeze până la terminarea *task*-urilor proprii, atunci timpul total de funcționare este egal cu timpul de inițializare a execuției paralele la care se adaugă timpul maxim necesar componentei celei mai costisitoare de timp și timpul de sincronizare a rezultatelor componentelor.

Un timp optim se obține dacă împărțirea pe componente se poate face astfel încât componentele să solicite timpi relativ egali de funcționare (astfel, timpii de așteptare sunt reduși la valori minime). Această situație este cea care maximizează și concurența pentru execuția algoritmului respectiv.

Această metodă statică de partiționare este potrivită pentru probleme de găsire a unor clase particulare de puncte pentru funcții date (rădăcini, puncte de întoarcere).

Față de situația descrisă anterior, pentru unele probleme este important ca, după încheierea execuției uneia dintre componente, algoritmul să decidă încheierea execuției și pentru toate celelalte componente. În acest caz, se schimbă considerabil timpul total de execuție a algoritmului. Pentru aceeași clasă de probleme matematice, această soluție este potrivită atunci când este suficient să se găsească o singură soluție a problemei.

Prelucrarea datelor în pipeline

O altă abordare simplă a concurenței este prelucrarea în *pipeline*. Aceasta presupune că ansamblul activităților care trebuie aplicate datelor poate fi împărțit în faze de prelucrare distincte, astfel încât diferite secvențe de date să treacă în flux de la o fază la alta.

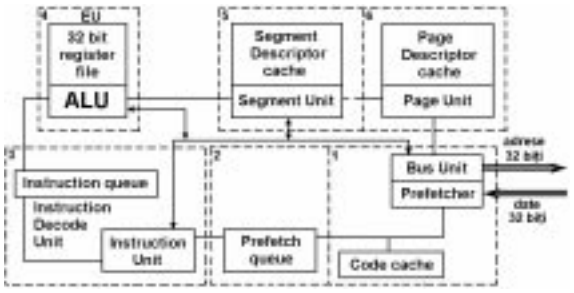
Un exemplu clasic de prelucrare în *pipeline* este realizarea unui compilator pentru care fazele de lucru sunt: analiza lexicală, analiza semantică, verificarea tipurilor de date, generarea de cod ș.a.m.d. Aici, imediat ce un modul a trecut de analiza lexicală este trimis în faza de analiză semantică și automat un al doilea modul este încărcat pentru analiza lexicală.

În acest model este important faptul că diverse activități sunt simultan în execuție în diferite faze și trebuie să se sincronizeze între ele: să aștepte ca noile date să devină disponibile sau, din contră, să aștepte ca faza următoare să-și termine procesul asupra datelor curente.

În arhitectura procesoarelor s-au dezvoltat și tehnici de procesare (prelucrare) paralelă. Astfel, dacă la 8086 erau doar două unități funcționale care lucrau în paralel (*Execution Unit* și *Bus Interface Unit*), structura procesoarelor 286 conține patru unități funcționale care lucrează în paralel, în timp ce procesorul 386 conține șase astfel de unități care lucrează în paralel. Acestea sunt:

- *Bus Interface Unit* (***BIU***) - unitatea de comunicare cu magistrala;
- *Code Prefetch Unit* - unitatea de citire în avans a instrucțiunilor;
- *Instruction Decode Unit* - unitatea de decodificare a instrucțiunilor;
- *Execution Unit* (***EU***) - unitatea de execuție a instrucțiunilor;
- *Segment Unit* - unitatea de translatare a adreselor logice (translatează adresele logice în adrese liniare și realizează verificările de protecție);
- *Paging Unit* - unitatea de translatare a adreselor liniare (translatează adresele liniare în adrese fizice și realizează verificările de protecție legate de mecanismul de paginare a memoriei); această unitate conține o memorie *cache* cu informațiile de lucru pentru cel mult 32 de pagini, cele mai recent accesate.

Acest model poate fi reprezentat schematic astfel [4]:



Structura internă a unui microprocesor Intel

Paralelismul execuției s-a dezvoltat prin expandarea unităților de decodificare a instrucțiunilor și de execuție într-o bandă de asamblare (*pipeline*) cu cinci niveluri. Unitatea corespunzătoare fiecărui nivel operează în paralel cu celelalte, putându-se executa, în acest mod, cinci operații în paralel. Nivelul L1 al memoriei *cache* a procesorului 486 are rolul de a mări procentul instrucțiunilor care se pot executa la viteză maximă. Odată cu utilizarea memoriei *cache* pe două niveluri, *pipeline*-ul memoriei a devenit și mai complex. Performanțele procesoarelor *Pentium 4* se datorează și unui *pipeline* intern cu 20 de stagii.

Arhitectura de tip *pipeline* a structurii interne a microprocesoarelor permite accelerarea fazelor de extragere, decodificare și execuție a instrucțiunilor programului intern al procesorului. Toate microprocesoarele moderne au organizarea internă de tip *pipeline* [3]. La o unitate centrală modernă se pot întâlni organizări diverse de tip *pipeline*, de exemplu:

- ***pipeline artimetic*** la nivelul circuitelor aritmetico-logice; acest model se referă la separarea fazelor în execuția operațiilor de adunare, înmulțire, împărțire și extragere a rădăcinii pătrate; astfel, pentru adunare și scădere în virgulă mobilă se identifică următoarele patru suboperații, care pot fi realizate pe blocuri separate: compararea exponenților, denormalizarea mantisei pentru operandul mai mic, operația de adunare/scădere a mantiselor, normalizarea rezultatului;
- ***pipeline la nivelul execuției instrucțiunilor*** presupune executarea diferitelor faze ale interpretării instrucțiunilor în blocuri diferite (extargere, decodificare, citire operanzi, calcul, scriere rezultate);
- ***pipeline la nivelul controlului programului*** se referă la conectarea liniară a mai multor procesoare pentru prelucrarea succesivă a unui flux de date. Aici apare un paralelism funcțional, la nivel arhitectural;
- ***pipeline la nivelul memoriei*** care separă operațiile de prezentare a adresei, decodificare adresă, depunere dată.

Algoritmi cu structură arborescentă a spațiului de soluții

Mulți algoritmi pot fi reprezentați într-o structură arborescentă a spațiului lor de soluții. În aceste cazuri, oportunitatea procesării paralele este imediată, dacă ramuri diferite ale arborelui de reprezentare sunt independente. Aplicațiile potrivite pentru acest model sunt problemele de căutare.

Cazul cel mai nefavorabil (traversarea întregului spațiu de soluții) este atins numai dacă problema nu admite soluție.

Altfel, algoritmul trebuie să decidă încheierea activităților ramurilor în momentul în care pe una din ramuri a fost găsită o soluție.

Acest model este, de asemenea, potrivit pentru abordarea problemelor de recunoaștere a formelor. Fiecare ramură care trebuie executată în paralel va putea fi traversată în mod repetat, deoarece există alternative diferite pentru atingerea unei soluții (cazuri nedeterministe).

Mai mult, aici fiecare ramură poate avea asociată o probabilitate de a conduce la soluție. În acest caz, algoritmul trebuie să vizeze și selectarea combinației de ramuri care conduc la cea mai bună probabilitate de obținere a unei soluții.

O structură arborescentă de abordare introduce noi aspecte de paralelism față de caracteristicile simple, statice, de partiționare liniară descrise anterior.

O problemă nouă, care se ridică aici, este cum să se construiască dinamic (în timpul rezolvării problemei) un număr rezonabil de activități paralele și cum să se păstreze pentru fiecare evidența proceselor proprii și a procesoarelor implicate.

Partajarea datelor

În sistemele de gestiune a tranzacțiilor se impune accesul diferitelor tranzacții la aceleași date comune. Există cazuri particulare de sisteme în care datele nu pot fi partiționate (într-un sistem de rezervare a biletelor nu este eficient să se distribuie pentru vânzare locuri diferite la case de bilete diferite). Caracterul concurent al unor astfel de aplicații poate fi atins, nu prin simpla partiționare a datelor, ci prin partajarea datelor între numeroasele procese din sistem. O situație specifică pentru această abordare este următoarea: întreaga sarcină de efectuat este împărțită în activități și este reținută într-o zonă comună de memorie ca o structură de date care poate fi folosită în comun de diferite entități. Fiecare activitate citește o lucrare din structura comună de date, o marchează ca fiind preluată pentru execuție și o trimite activității specifice în vederea execuției.

O problemă ar fi evitarea deteriorării datelor datorită accesului concurent la zona comună.

Cerințe pentru realizarea aplicațiilor concurente

Din considerentele anterioare rezultă câteva abordări simple ale caracterului concurent la nivelul algoritmilor. Acestea pun accent pe partiționarea datelor static sau dinamic, respectiv pe partiționarea codului într-o structură de tip *pipeline*. În general, acestea sunt opțiunile programatorului care are de abordat un sistem potențial concurent. Pentru sistemele inerent concurente, soluția cea mai potrivită folosește partajarea datelor, respectiv partajarea memoriei comune.

La nivelul aplicațiilor, se pot enunța câteva cerințe prin care să se impună caracterul concurent al soluției problemei de rezolvat. Astfel, aplicațiile trebuie:

- să suporte activități individuale;
- să poată gestiona aceste activități de o manieră similară; aceste activități trebuie să poată fi create la cerere și întrerupte când nu mai sunt necesare;
- să sincronizeze diferite activități care se desfășoară simultan; să asigure posibilitatea transferului de date între activitățile simultane;
- să se adapteze fiecărei situații particulare de tip: activitățile pot partaja date sau, din contră, trebuie impuse restricții de acces comun la anumite date.

Aplicațiile trebuie scrise într-un limbaj de programare care, împreună cu sistemul de operare gazdă, are facilități de implementare a condițiilor de mai sus.

Bibliografie

[1] Bacon Jean, *Concurrent Systems. Operating Systems, Database and Distributed Systems: an Integrated Approach*, Addison-Wesley, 1998;

[2] Dollinger Robert, *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj Napoca, 2000;

[3] Grigoraș Dan, *Calculul paralel. De la sisteme la programarea aplicațiilor*, Editura Computer Libris Agora, Cluj Napoca, 2000;

[4] Lungu Vasile, *Procesoare Intel. Programare în limbaj de asamblare*, Editura Teora, București, 2000;

[5] Quinn M.J., *Parallel Computing - Theory and Practice*, McGraw-Hill 1994;

[6] von Neumann John, *The Computer and the Brain*, Yale University Press, New Haven;

D-na Anca Vasilescu este asistent universitar la Universitatea Transilvania din Brașov și poate fi contactată prin e-mail la adresa vasilex@info.unitbv.ro. *D-na Oana Georgescu este asistent universitar la Universitatea Sextil Pușcariu din Brașov și poate fi contactată prin e-mail la adresa* o.georgescu@info.unitbv.ro.