Măsurarea timpului

Capitolul 12

- **Preliminarii**
- **Preluarea timpului**
- **Setarea timpului**
- **Întreruperea 08h**
- **Locația $0000:$046C**
- **Rezumat**
- **Implementări sugerate**

Mulți programatori doresc să măsoare sau să controleze durata unei operațiuni realizate de calculator. Chiar şi la concursurile de programare, timpul alocat rulării este limitat, motiv pentru care ar fi bine ca programul să "ştie" când se apropie timpul de execuție de sfârşit. În cadrul acestui capitol vă vom prezenta câteva modalități de a utiliza ceasul sistem al calculatorului.

12.1. Preliminarii

Comportamentul multor programe depinde de un anumit moment de timp. Din aceste motive, calculatoarele trebuie să ştie în orice moment "cât e ceasul". Unele programe trebuie să ştie cât timp a trecut de la începerea rulării. Ne putem imagina oricâte exemple din cele mai diverse domenii, dar pentru elevi cel mai potrivit exemplu este cel al programelor care rezolvă diferite probleme date la concursurile de programare. De foarte multe ori, atunci când concurenții nu găsesc un algoritm eficient pentru rezolvarea unei probleme, ei preferă să utilizeze diverse metode pentru a testa diferite soluții posibile şi a o alege pe cea mai bună, în speranța că aceasta este şi cea corectă. Programele vor încerca să găsească soluții noi până în momentul în care timpul de execuție admis se va apropia de sfârşit şi apoi vor furniza cea mai bună soluție găsită până în acel moment.

Pentru a realiza acest lucru, este folosită (prin diferite procedee) întreruperea 08h a sistemului de operare MS DOS, numită şi întreruperea timer-ului. Vom exemplifica aceste procedee pentru compilatoarele Borland Pascal 7.0 şi Borland C++ 3.1.

Operația folosită pentru exemplificarea măsurării timpului scurs este executarea unui ciclu vid având două miliarde de iterații.

Deşi măsurarea timpului este utilă pentru marea majoritate a problemelor, ea constituie un instrument deosebit de important în momentul în care avem de-a face cu o problemă NP-completă, deoarece pentru aceste probleme este foarte puțin probabil să existe un algoritm de rezolvare cu timp de execuție redus.

După cum se va vedea în cadrul capitolelor următoare, o parte a abordărilor care pot fi folosite în rezolvarea problemelor NP-complete se bazează pe generarea de soluții noi până în momentul în care timpul afectat execuției expiră.

12.2. Preluarea timpului

Cea mai evidentă metodă de măsurare a timpului este folosirea rutinelor care returnează într-o anumită formă, timpul la momentul în care sunt apelate. În Pascal avem procedura `GetTime` a unit-ului `DOS`, iar în C++ funcția `gettime()` declarată în fişierul *header* `DOS.H`.

Pentru limbajul Pascal va trebui să folosim un apel de tipul `GetTime(Ore, Minute, Secunde,Sutimi)`, unde `Ore`, `Minute`, `Secunde` şi `Sutimi` sunt variabile de tip `Word` sau un tip compatibil cu acesta. Denumirile pe care le-am ales pentru cele patru variabile sugerează clar semnificațiile valorilor pe care le vor primi după apel.

Pentru C++ apelul folosit va fi de forma `gettime(&t)`, unde `t` este o structură de tip `time`. Structura `time` este definită în fişierul *header* `DOS.H` astfel:

```
struct time{
   unsigned char ti_min;  // minute
   unsigned char ti_hour; // ore
   unsigned char ti_hund; // sutimi de secunda
   unsigned char ti_sec; // secunde
};
```

Semnificațiile valorilor pe care le vor primi cele patru câmpuri după apelul funcției apar în comentarii.

Pentru a măsura exact timpul care a trecut de la începerea execuției programului, prima instrucțiune executată va trebui să fie un apel al unei proceduri/funcții de tip `gettime`, urmată de salvarea în variabile auxiliare a valorilor obținute. În momentul în care dorim să ştim cât timp a trecut de la începerea execuției programului, vom realiza un nou apel şi vom calcula timpul care a trecut între primul şi ultimul apel, efectuând câteva operații simple.

12.2.1. Varianta Pascal

În cele ce urmează vom prezenta un scurt program *Pascal* care ilustrează modul în care poate fi utilizată procedura `GetTime`.

```
uses Dos;
var ore_start,minute_start,secunde_start,sutimi_start,
    ore_final,minute_final,secunde_final,sutimi_final,
    ore,minute,secunde,sutimi:Word;
    i:Longint;

function Zerouri(nr:Word):string;
var s:string;
begin
  Str(nr,s);
  if Length(s)=1 then
    s:='0'+s;
  Zerouri:=s
end;

Begin
  Gettime(ore_start,minute_start,secunde_start,sutimi_start);
  for i:=1 to 200000000 do; { ciclu vid }
  Gettime(ore_final,minute_final,secunde_final,sutimi_final);
  sutimi:=100+sutimi_final-sutimi_start;
  Dec(secunde_final,1-sutimi div 100);
  sutimi:=sutimi mod 100;
  secunde:=60+secunde_final-secunde_start;
  Dec(minute_final,1-secunde div 60);
  secunde:=secunde mod 60;
  minute:=60+minute_final-minute_start;
  Dec(ore_final,1-minute div 60);
  minute:=minute mod 60;
  ore:=(24+ore_final-ore_start) mod 24;
  Writeln('Timp de executie: ',
          Zerouri(ore),' ore ',
          Zerouri(minute),' minute ',
          Zerouri(secunde),'.',Zerouri(sutimi),' secunde.')
End.
```

12.2.2. Varianta C++

În cele ce urmează vom prezenta un scurt program *C++* care ilustrează modul în care poate fi utilizată funcția `gettime`.

```
#include <stdio.h>
#include <dos.h>

void main(void){
  struct time t,tstart;
  int ore,minute,secunde,sutimi;
```

```
  gettime(&tstart);
  for (long i=0;i<2000000000;i++);                    { ciclu vid }
  gettime(&t);
  sutimi=100+t.ti_hund-tstart.ti_hund;
  t.ti_sec-=1-sutimi/100;
  sutimi%=100;
  secunde=60+t.ti_sec-tstart.ti_sec;
  t.ti_min-=1-secunde/60;
  secunde%=60;
  minute=60+t.ti_min-tstart.ti_min;
  t.ti_hour-=1-minute/60;
  minute%=60;
  ore=(24+t.ti_hour-tstart.ti_hour)%24;
  printf("Timp de executie: %02d ore %02d \
          minute %02d.%02d secunde.\n",
          ore,minute,secunde,sutimi);
}
```

12.2.3. Dezavantaje

Această metodă este cea mai ineficientă, datorită timpului relativ mare necesar executării subprogramelor de acest tip. În plus, se pierde timp atât la execuție, cât şi la scrierea codului pentru calcularea efectivă a diferenței în ore, minute, secunde şi sutimi.

12.3. Setarea timpului

Pierderea de timp datorată calculării efective a diferenței în ore, minute, secunde şi sutimi poate fi evitată setând la început ceasul sistem la ora 00:00, apoi pentru citirea timpului scurs între momentul setării ceasului pe 0 şi momentul dorit, vom apela subprogramele prezentate anterior.

Într-un program Pascal va trebui să folosim procedura `SetTime` a unit-ului `DOS` prin apelul `SetTime`(0,0,0,0). Într-un program C++ vom folosi funcția `settime()`, declarată în fişierul *header* `DOS.H`. Apelul trebuie să fie de forma `settime(&t)`, unde `t` este o structură de tip `time`. Câmpurile variabilei `t` trebuie să aibă toate valoarea 0.

Dacă am setat la început ceasul sistem la ora 00:00, atunci un apel `GetTime` ne va da informații referitoare la timpul scurs de la începerea execuției. Nu mai este nevoie de efectuarea de operații suplimentare pentru determinarea timpului în ore, minute, secunde şi sutimi.

12.3.1. Varianta Pascal

În cele ce urmează vom prezenta un scurt program *Pascal* care ilustrează modul în care poate fi utilizată procedura `SetTime`.

```
uses Dos;
var ore,minute,secunde,sutimi:Word;
    i:Longint;

function Zerouri(nr:Word):string;
var s:string;
begin
  Str(nr,s);
  if Length(s)=1 then
    s:='0'+s;
  Zerouri:=s
end;

Begin
  Settime(0,0,0,0);
  for i:=1 to 2000000000 do; { ciclu vid }
  Gettime(ore,minute,secunde,sutimi);
  Writeln('Timp de executie: ',
          Zerouri(ore),' ore ',
          Zerouri(minute),' minute ',
          Zerouri(secunde),'.',Zerouri(sutimi),' secunde.')
End.
```

12.3.2. Varianta C++

În cele ce urmează vom prezenta un scurt program *C++* care ilustrează modul în care poate fi utilizată funcția `settime`.

```
#include <stdio.h>
#include <dos.h>

void main(void){
  struct time t={0,0,0,0};
  settime(&t);
  for (long i=0;i<2000000000;i++);                    { ciclu vid }
  gettime(&t);
  printf("Timp de executie: %02d ore %02d \
          minute %02d.%02d secunde.\n",
          t.ti_hour,t.ti_min,t.ti_sec,t.ti_hund);
}
```

12.3.3. Dezavantaje

Această metodă este puțin mai eficientă decât cea anterioară, dar necesită modificarea ceasului sistem, operație care nu este recomandabil. De fapt, la marea majoritate a

concursurilor de programare o astfel de operație este interzisă prin regulamente, iar concurenții care încearcă modificarea ceasului sistem pot fi descalificați.

12.4. Întreruperea 08h

Această metodă este mai rapidă şi nu are efecte secundare nedorite. Timer-ul este o rutină care este apelată cu o frecvență de aproximativ 18,2 ori pe secundă, adică o dată la aproximativ 55 de milisecunde. Operația efectuată de această rutină este incrementarea ceasului sistem cu 55 ms.

Totuşi, prin capturarea întreruperii, putem adăuga propriul nostru cod care să se execute în momentul în care este apelată rutina corespunzătoare întreruperii. Trebuie doar să fim atenți ca rutina scrisă de noi să apeleze şi vechea rutină deoarece, în caz contrar, timpul sistemului se va opri şi acest lucru ar putea avea efecte nedorite.

În continuare vom prezenta un exemplu care foloseşte această metodă pentru a măsura timpul. Să presupunem că dorim să executăm un ciclu timp de 10 secunde; pentru aceasta vom inițializa o variabilă `time` cu valoarea 182 ($10 \cdot 18.2 = 182$) şi vom decrementa această valoare cu o unitate la fiecare execuție a întreruperii `08h`. Ciclul se va întrerupe în momentul în care variabila `time` va avea valoarea 0.

Într-un program *Pascal* vom folosi procedura `GetIntVec` pentru a captura o întrerupere; această procedură este declarată astfel:

```
GetIntVec(IntNo:Byte;var Vector:Pointer),
```

unde `IntNo` indică numărul întreruperii, iar `Vector` va indica adresa la care era păstrată rutina care este executată în momentul apelării întreruperii. Pentru a executa o anumită rutină în momentul apelării întreruperii vom folosi procedura `SetIntVec` a cărei declarație este:

```
SetIntVec(IntNo:Byte;Vector:Pointer),
```

unde `IntNo` indică numărul întreruperii, iar `Vector` indică adresa la care este păstrată rutina care va fi executată în momentul apelării întreruperii. Aceste două proceduri sunt definite în unit-ul `DOS`.

Într-un program C++ vom folosi funcția `getvect()` pentru a captura o întrerupere; antetul acestei funcții este:

```
void interrupt (*getvect(int interruptno))(),
```

unde `interruptno` indică numărul întreruperii. Funcția va returna un *pointer* către rutina care este executată în momentul apelării întreruperii. Pentru a executa o altă rutină în momentul apelării întreruperii vom folosi funcția `setvect()` al cărei antet este

```
void setvect(int interruptno,void interrupt (*isr)()),
```

unde `interruptno` indică numărul întreruperii, iar `isr` indică adresa la care este păstrată funcția care va fi executată în momentul apelării întreruperii. Aceste două funcții sunt declarate în fişierul *header* `DOS.H`.

12.4.1. Varianta Pascal

În cele ce urmează vom prezenta un scurt program *Pascal* care ilustrează modul în care pot fi utilizate procedurile `GetIntVec` şi `SetIntVec` pentru măsurarea timpului.

```
uses Dos;
var i,time:Longint;
    OldTimer:procedure;

procedure MyTimer;interrupt;
begin
  Dec(time);
  inline ($9C);
  OldTimer
end;

Begin
  time:=182;
  i:=0;
  GetIntVec(8,@OldTimer);
  SetIntVec(8,@MyTimer);
  while time>0 do Inc(i);
  SetIntVec(8,@OldTimer)
End.
```

12.4.2. Varianta C++

În cele ce urmează vom prezenta un scurt program *C++* care ilustrează modul în care pot fi utilizate procedurile `getvect` şi `setvect` pentru măsurarea timpului.

```
#include <dos.h>

long i,time=182;

void interrupt (*OldTimer)(...);

void interrupt MyTimer(...){
  time--;
  OldTimer();
}

void main(void){
  OldTimer=getvect(8);
  setvect(8,MyTimer);
  while(time)
    i++;
  setvect(8,OldTimer);
}
```

12.4.3. Dezavantaje

Bineînțeles, dezavantajul principal al acestei metode este necesitatea scrierii de cod suplimentar. În plus, sunt efectuate operații suplimentare cum ar fi decrementarea variabilei `time` sau apelurile rutinei vechi de tratare a *timer*-ului. Acestea reduc puțin timpul care este folosit pentru efectuarea operațiilor pe care trebuie să le realizeze programul.

12.5. Locația $0000:$046C

Ultima variantă pe care o vom prezenta elimină şi aceste ultime deficiențe. Ea se bazează pe accesarea directă a unei locații de memorie în care este păstrat numărul de apeluri ale timer-ului începând cu ora 00:00.

Numărul de tacți (apeluri ale timer-ului) care au trecut de la miezul nopții este reprezentat pe patru bytes şi se află la adresa $0000:$046C.

Pentru a utiliza această metodă vom păstra valoarea stocată la această locație în momentul începerii execuției şi apoi, pentru a măsura intervalul de timp scurs, vom scădea valoarea inițială din valoarea curentă de la această locație. Diferența va indica numărul de tacți care au trecut de la începerea execuției programului.

În Pascal, pentru a accesa direct o locație de memorie putem folosi clauza **`absolute`**.

În C++, pentru a realiza acelaşi lucru, putem folosi o mulțime de metode. Cea mai simplă dintre ele este utilizarea unui pointer care va referi locația respectivă.

12.5.1. Varianta Pascal

În cele ce urmează vom prezenta un scurt program *Pascal* care ilustrează modul în care poate fi accesată locația `$0000:$046C` şi cum poate fi aceasta utilizată pentru măsurarea timpului.

```
var i,t:Longint;
    time:Longint absolute $0:$46C;
Begin
  t:=time;
  i:=0;
  while (time-t<182) do
    Inc(i)
End.
```

12.5.2. Varianta C++

În cele ce urmează vom prezenta un scurt program *C++* care ilustrează modul în care poate fi accesată locația `$0000:$046C` şi cum poate fi aceasta utilizată pentru măsurarea timpului.

```
void main(void){
  long far *time = (long far *)0x46c;
  long t = *time;
  while (*time - t < 182);
}
```

12.5.3. Dezavantaje

Chiar şi această metodă are mici dezavantaje; ea nu va da rezultatele aşteptate în cazul în care ceasul sistem va deveni 00:00 în timpul execuției programului. Apariția acestei situații este destul de improbabilă în cazul în care se lucrează cu intervale de timp de ordinul secundelor sau minutelor, dar ea ar putea apărea dacă intervalele de timp măsurate sunt mai mari. În acest caz s-ar putea ca programele să nu mai funcționeze conform aşteptărilor. Cu toate acestea, aceasta este cea mai des utilizată metodă la concursurile de programare, deoarece riscul ca ea să nu funcționeze corect este foarte mic, este uşor de implementat, iar numărul operațiilor suplimentare efectuate este minim.

12.6. Rezumat

În cadrul acestui capitol am prezentat mai multe metode care pot fi utilizate pentru măsurarea timpului. Am început cu cea mai intuitivă metodă şi am arătat că aceasta este ineficientă. În continuare am prezentat alte trei metode (din ce în ce mai puțin intuitive, dar din ce în ce mai eficiente) şi am prezentat îmbunătățirile pe care le aduce fiecare, dar şi principalele dezavantaje.

În final, am concluzionat că pentru concursurile de programare, de obicei, este recomandată utilizarea ultimei metode prezentate.

12.7. Implementări sugerate

Pentru a vă obişnui cu modul în care puteți măsura timpul de execuție al programelor dumneavoastră vă sugerăm să încercați să utilizați metoda în următoarele situații:

1. compararea duratelor de execuție ale algoritmilor de sortare rapidă şi sortare prin selecție pentru şiruri cu peste 10000 de elemente;
2. rezolvarea problemei comis-voiajorului prin metoda backtracking, întreruperea execuției programului după două secunde şi afişarea celei mai bune soluții determinate până în acel moment;
3. măsurarea duratei de execuție a unor secțiuni critice ale unor programe.
4. măsurarea timpilor de execuție pentru diferite implementări ale algoritmilor cunoscuți.