Măsurarea TIMPULUI

Mihai Scorţaru

Mulţi programatori doresc să măsoare sau să controleze durata unei operaţiuni 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 timpul de execuţie se apropie de sfârşit. În acest articol vă vom prezenta câteva modalităţi de a utiliza ceasul sistem al calculatorului.

Preliminarii

Comportamentul multor programe depinde de un anumit moment de timp. Astfel, un "ceas deşteptător" virtual va putea suna la ora stabilită, un satelit artificial va putea fi lansat atunci când trebuie etc.

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 robotica industrială sau din lumea burselor, dar pentru cititorii ***GInfo*** cel mai potrivit exemplu este cel al programelor care rezolvă diferite probleme date la concursurile de programare. De foarte multe ori, atunci când concurenţii nu găsesc un algoritm eficient pentru rezolvarea unei probleme, ei preferă să utilizeze diverse metode pentru a testa diferite soluţii posibile şi a o alege pe cea mai bună. Programele vor încerca să găsească soluţii noi până când timpul de execuţie admis se va apropia de sfârşit şi apoi vor furniza cea mai bună soluţie 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*.

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

Subprograme de tip GetTime

Cea mai simplă astfel de metodă este folosirea funcţiilor care returnează, într-o anumită formă, timpul din momentul în care sunt apelate. În *Pascal* avem procedura `GetTime` a *unit*-ului `DOS`, iar în C++ funcţia `gettime()` declarată în fişierul *header* **`DOS.H`**.

Pentru *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 destul de clar semnificaţiile valorilor pe care le vor primi după apel.

Pentru C++ apelul folosit va fi de tipul `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
};
```

Semnificaţiile valorilor pe care le vor primi cele patru câmpuri după apelul funcţiei apar în comentarii.

Pentru a măsura exact timpul care a trecut de la începerea execuţiei programului, prima instrucţiune executată va trebui să fie un apel al unei proceduri/funcţii de tip `gettime`, urmată de salvarea în variabile auxiliare a valorilor obţinute.

În momentul în care dorim să ştim cât timp a trecut de la începerea execuţiei programului, vom realiza un nou apel şi vom calcula timpul care a trecut între primul şi ultimul apel, efectuând câteva operaţii simple.

Varianta Pascal

Listing GETTIME.PAS

```
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;
  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.
```

Varianta C++

Listing GETTIME.CPP

```
#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++);
  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);
 }
```

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 și la scrierea codului pentru calcularea efectivă a diferenței în ore, minute, secunde și sutimi.

Subprograme de tip SetTime

Pierderea de timp datorată calculării efective a diferenței în ore, minute, secunde și sutimi poate fi evitată setând la început ceasul sistem la ora `00:00`, apoi pentru citirea timpului scurs între momentul setării ceasului pe `0` și 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 fișierul *header* **`DOS.H`**. Apelul trebuie să fie de tipul `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 și sutimi.

Varianta Pascal

Listing SETTIME.PAS

```
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;
  Gettime(ore,minute,secunde,sutimi);
  Writeln('Timp de executie: ',Zerouri(ore),
        ' ore ', Zerouri(minute),' minute ',
                      Zerouri(secunde),'.',
                Zerouri(sutimi),' secunde.')
End.
```

Varianta C++

Listing SETTIME.CPP

```
#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++);
  gettime(&t);
  printf("\nTimp de executie: %02d ore %02d \
       minute %02d.%02d secunde\n",t.ti_hour,
                t.ti_min,t.ti_sec,t.ti_hund);
}
```

Această metodă este puțin mai eficientă decât cea anterioară, dar necesită modificarea ceasului sistem, operație care nu este recomandabilă.

Capturarea întreruperii 08h

Această metodă este mai rapidă și 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. Totuși, 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 și vechea rutină deoarece, în caz contrar, timpul sistemului se va opri și acest lucru ar putea avea efecte nedorite.

În continuare vom prezenta un exemplu care folosește 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 (`10·18.2=182`) și 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 fișierul *header* **`DOS.H`**.

Varianta Pascal

Listing INTR8.PAS

```
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.
```

Varianta C++

Listing INTR8.CPP

```
#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);
}
```

Și această metodă are unele dezavantaje deoarece implică scrierea 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 efecturea operațiilor pe care trebuie să le realizeze programul.

Locația de memorie $0000:$046C

Ultima variantă pe care o vom prezenta elimină și 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 și se află la adresa `$0000:$046C`.

Pentru a utiliza această metodă vom păstra valoarea stocată la această locație în momentul începerii execuției și 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 același lucru, putem folosi *macro*-ul `MK_FP` definit în cadrul fișierului *header* **DOS.H**. Acest *macro* creează un pointer din părțile sale componente (segment și deplasament). În cazul nostru segmentul va avea valoarea `0`, iar deplasamanetul va avea valoarea `0x46C`.

Varianta Pascal

Listing ABSOLUTE.PAS

```
var i,t:Longint;
    time:Longint absolute $0:$46C;

Begin
  t:=time;
  i:=0;
  while (time-t<182) do Inc(i)
End.
```

Varianta C++

Listing MKFP.CPP

```
#include <dos.h>
void main(void){
  long i=0,t,*time=(long*)MK_FP(0,0x46C);
  t=*time;
  while (*time-t<182) i++;
}
```

Chiar și această metodă are un mic dezavantaj; ea nu va da rezultatele așteptate 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 așteptărilor.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa skortzy@xnet.ro.

M/F?

Multe persoane s-au întrebat dacă numeroasele instrumente pe care le folosim zi de zi trebuie considerate a fi de genul feminin sau de genul masculin.

Unele obiecte au primit deja un gen. De exemplu, serialul Star Trek a impus genul feminin pentru navele stelare. Se pune acum problema genului calculatoarelor.

Un grup de oameni de știință (format doar din bărbați) au ajuns la concluzia că trebuie să considerăm calculatoarele ca fiind de genul feminin și au dat cinci argumente în favoarea acestui fapt.

- Nimeni, cu excepția Creatorului lor, nu le înțelege logica internă.
- Limbajul nativ pe care îl folosesc pentru comunicarea cu alte calculatoare nu mai poate fi înțeles de nimeni altcineva.
- Mesajul "Bad command or file name" este la fel de informativ ca și mesajul "Dacă nu știi de ce sunt supărată pe tine, atunci cu siguranță îți voi spune eu!".
- Chiar și cele mai mici greșeli pe care le facem sunt stocate în memoria permanentă pentru a putea fi regăsite ulterior.
- Imediat după ce te atașezi de unul, îți dai seama că cheltui jumătate din venituri cumpărând accesorii pentru el.

Un alt grup (format numai din femei) au ajuns la concluzia că genul calculatoarelor trebuie să fie masculin și au adus și ele cinci argumente în favoarea acestei opinii.

- Au o mulțime de date, dar nu au nici o idee despre ceea ce se întâmplă în jur.
- Ar trebui să te ajute să îți rezolvi problemele dar, în cea mai mare parte a timpului, constituie adevărata problemă.
- Imediat după ce te atașezi de unul observi că, dacă ai mai fi așteptat puțin, ai fi putut obține un model mai bun.
- Pentru a le atrage atenția trebuie neapărat aprinse.
- Descărcări mari de energie la fac inutilizabile pentru restul nopții.