Προγραμματισμός συστημάτων UNIX/POSIX

Χρονομέτρηση

Β. Δημακόπουλος

Χρονομέτρηση (I)

- Πρώτα από όλα, **τι ακριβώς θέλουμε να μετρήσουμε;**
  - Μας ενδιαφέρει ο χρόνος που ο επεξεργαστής αφιέρωσε σε ένα πρόγραμμα ή σε ένα συγκεκριμένο τμήμα του;
    - ***Χρόνος επεξεργαστή (χρόνος καθαρών υπολογισμών).***
  - Ή μας ενδιαφέρει πόση ώρα περνάει από ένα συγκεκριμένο γεγονός;
    - ***Πραγματικός χρόνος που παρήλθε.***

- Στην πρώτη περίπτωση, μας ενδιαφέρει μόνο πόσο χρόνο χρειάστηκε ο επεξεργαστής για καθαρούς υπολογισμούς, όχι πόσος χρόνος πέρασε από την ώρα που ξεκίνησε η χρονομέτρηση.
  - Ο χρόνος είναι *ανεξάρτητος του φόρτου του συστήματος – θα είναι πάντα ο ίδιος,* ακόμα και όταν εκτελούνται πολλές διεργασίες "ταυτόχρονα".
  - Μετρά "χρόνο επεξεργαστή" (CPU time) όχι πραγματικό χρόνο που παρήλθε (real/elapsed time).
  - Ίσως η πιο χρήσιμη περίπτωση όταν ενδιαφερόμαστε για εκτίμηση της επίδοσης του κώδικα.
- Στη δεύτερη περίπτωση μας ενδιαφέρει πόσος *πραγματικός* χρόνος παρήλθε.
  - *Εξαρτάται από τον φόρτο του συστήματος*! Όταν εκτελείται μόνο η εφαρμογή μας, η χρονομέτρηση θα είναι ακριβής, όταν εκτελούνται κι άλλες θα έχει διακυμάνσεις. Η χρονομέτρηση πρέπει να γίνεται σε ελεγχόμενο περιβάλλον.
  - Είναι χρήσιμο όταν μας ενδιαφέρει η συνολική συμπεριφορά, σε πραγματικές συνθήκες.

Χρονομέτρηση (II)

- **Τι τάξη μεγέθους είναι οι χρόνοι που μας ενδιαφέρουν;**
  - Είναι χρόνοι της τάξης των λεπτών/ωρών/ημερών κλπ ;
    - Εδώ δεν μπαίνει θέμα ακρίβειας.
  - Είναι χρόνοι τάξης δευτερολέπτων / δεκάτων ή εκατοστών του δευτερολέπτου;
    - Κι εδώ η ακρίβεια δεν είναι (συνήθως) ουσιαστικό θέμα, οι περισσότεροι μηχανισμοί χρονομέτρησης είναι εφαρμόσιμοι.
  - Είναι χρόνοι της τάξης του msec / μsec / nsec?
    - Εδώ απαιτείται χρονομέτρηση με κατάλληλη υποστήριξη από το υλικό. Οι χρονομετρητές πρέπει να έχουν αυξημένη ακρίβεια και ανάλυση (resolution).
  - Στην τελευταία περίπτωση δεν υπάρχουν πάντα λύσεις τυποποιημένες που δουλεύουν σε όλα τα συστήματα.
  - Για τις υπόλοιπες περιπτώσεις μπορούν να χρησιμοποιηθούν αρκετοί και διαφορετικοί μηχανισμοί.

Χρονομέτρηση (III)

- **Τι μηχανισμούς χρονομέτρησης διαθέτουμε;**
  - Μετρούν χρόνο ("wall-clock" timers);
  - Μετρούν διαστήματα (interval timers);

- Στην πρώτη περίπτωση, οι χρόνοι που παίρνουμε είναι «έτοιμοι» για χρήση.
- Στη δεύτερη περίπτωση, θα πρέπει να ξέρουμε τη *διάρκεια του κάθε διαστήματος* προκειμένου να βρούμε τον πραγματικό χρόνο:
  - Π.χ. ρολόγια που αυξάνουν έναν μετρητή ανά τακτά διαστήματα.
  - Π.χ. μετρητές υψηλής ακρίβειας της CPU που μετρούν το πλήθος των παλμών του ρολογιού (clock cycles): *θα πρέπει να γνωρίζουμε τη συχνότητα του επεξεργαστή* για να βρούμε ποια είναι η διάρκεια του κάθε παλμού και άρα σε πόσο χρόνο αντιστοιχούν οι παλμοί που μετρήσαμε.

Δύο τρόποι χρονομέτρησης

1. Από το τερματικό
   - Εντολή time: μετρά την εκτέλεση ενός ολόκληρου προγράμματος

```
$ time ./a.out
…
real    0m5.316s
user    0m2.304s
sys     0m0.004s
```

Real: πραγματικός χρόνος που παρήλθε (εμπεριέχει ότι καθυστερήσεις υπήρχαν, π.χ. αναμονή να πληκτρολογήσει κάτι ο χρήστης)

User: χρόνος καθαρών υπολογισμών (CPU time) χωρίς τις κλήσεις συστήματος

Sys: χρόνος καθαρών υπολογισμών (CPU time) που δαπανήθηκαν σε κλήσεις συστήματος

2. Προγραμματιστικά
   - Για χρονομέτρηση ενός τμήματος του κώδικά μας:

```
<timing_call 1>    /* Record time t1 */
  <κώδικας>        /* The code we want to measure */
<timing_call 2>    /* Record time t2 */
```

   - *Η διαφορά των δύο χρόνων* t2, t1 θα μας δώσει την επιθυμητή μέτρηση

Χρονομέτρηση με την clock()

- Η **`clock()`** μετρά καθαρό χρόνο εκτέλεσης (CPU time)
  - Συνήθως ακρίβεια *εκατοστού του δευτερολέπτου*

- Η `clock()` είναι interval-based και επιστρέφει τον χρόνο που αφιέρωσε η CPU από τη στιγμή που ξεκίνησε το πρόγραμμά σας, μετρημένο σε «*κύκλους*» (clocks).
  - Για να βρείτε το χρόνο σε δευτερόλεπτα θα πρέπει να διαιρέσετε με τη σταθερά **`CLOCKS_PER_SEC`**.
  - Προσοχή στη διαίρεση γιατί και το `CLOCKS_PER_SEC` και η τιμή επιστροφής της `clock()` είναι ακέραιοι.

- Θα πρέπει να κάνετε `#include <time.h>`.

- Προσοχή: επειδή η `clock()` επιστρέφει ακέραιο, αν το πρόγραμμά σας χρειάζεται πολλή ώρα να τρέξει υπάρχει κίνδυνος να μηδενιστεί ο χρονομετρητής και να λάβετε λάθος μετρήσεις. Π.χ. στο solaris αναφέρεται ότι ο χρονομετρητής μηδενίζεται και μετρά πάλι από την αρχή κάθε 36 λεπτά καθαρού χρόνου εκτέλεσης!

```
$ man clock
```

Παράδειγμα χρονομέτρησης με την clock()

```
#include <stdio.h>
#include <time.h> /* Για την clock() */

int main() {
  double t1, t2;  /* Για αποφυγή ακέραιας διαίρεσης */
  int i, sum = 0;

  t1 = (double) clock(); /* επιστρέφει clock_t (συνήθως int ή long) */
  for (i = 0; i < 100000000; i++)
    sum += i;
  t2 = (double) clock();

  printf("Added 10000000 numbers in %lf sec (CPU time).\n",
         (t2 - t1) / CLOCKS_PER_SEC );
  return 0;
}
```

Χρονομέτρηση με την times()

- Η **times**() μετρά και πραγματικό χρόνο αλλά και καθαρό χρόνο εκτέλεσης (CPU time).
  - Ο πραγματικός χρόνος που επιστρέφει είναι το χρονικό διάστημα που παρήλθε από κάποιο απροσδιόριστο σημείο στο παρελθόν (π.χ. system boot time).
  - `#include <sys/times.h>`.

- Και η `times()` είναι interval-based. Όμως επιστρέφει χρόνους μετρημένους σε «*χτύπους ρολογιού*» (clock ticks).
  - Για να βρείτε το χρόνο σε δευτερόλεπτα θα πρέπει να διαιρέσετε με το πλήθος των χτύπων ρολογιού ανά δευτερόλεπτο, το οποίο το βρίσκεται μόνο προγραμματιστικά ως εξής:

```
ticspersec = sysconf(_SC_CLK_TCK);    /* unistd.h */
```

  - Προσοχή πάλι στις διαιρέσεις γιατί και οι χτύποι ανά δευτερόλεπτο και η τιμή επιστροφής της `times()` είναι ακέραιοι.

- Επιστρέφει τον *πραγματικό χρόνο που παρήλθε.*
- Παίρνει ως παράμετρο ένα **struct tms** από όπου μπορούμε να μάθουμε για τους καθαρούς χρόνους εκτέλεσης:

```
struct tms {
  clock_t tms_utime, tms_stime     /* for me */
  clock_t tms_cutime, tms_cstime;  /* for my child processes */
};
```

```
$ man -s 3 times
```

Παράδειγμα χρονομέτρησης με την times()

```
#include <stdio.h>
#include <sys/times.h>        /* Για την times() */
#include <unistd.h>           /* Για την sysconf() */

int main() {
  double t1, t2, cpu_time;    /* Για αποφυγή ακεραίας διαίρεσης */
  struct tms tb1, tb2;        /* Το χρειάζεται η times() */
  long   ticspersec;
  int    i, sum = 0;

  t1 = (double) times(&tb1); /* Η times() επιστρέφει (long) int */
  for (i = 0; i < 100000000; i++)
    sum += i;
  t2 = (double) times(&tb2);

  cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                       (tb1.tms_utime + tb1.tms_stime));
  ticspersec = sysconf(_SC_CLK_TCK);     /* # clock ticks / sec */

  printf("Real time: %lf sec; CPU time: %lf sec.\n",
         (t2 - t1) / ticspersec, cpu_time / ticspersec);
  return 0;
}
```

Χρονομέτρηση με την gettimeofday()

- Η **gettimeofday()** μετρά τον *πραγματικό* χρόνο που παρήλθε…
  - … από την 1/1/1970, ώρα 00:00 (το λεγόμενο «**Epoch**»).
  - `#include <sys/time.h>  /* Όχι το sys/times.h !! */`
  - Πολύ συχνή η χρήση της στην πράξη.

- Η `gettimeofday()` επιστρέφει χρόνο (wall-clock time).
  - Υλοποιείται συνήθως (όχι πάντα) με αρκετά μεγάλη ανάλυση (της τάξης του 1μsec).

- Παίρνει δύο παραμέτρους, με τη δεύτερη συνήθως NULL. Η πρώτη παράμετρος είναι δείκτης σε ένα **struct timeval** με τα εξής πεδία:

```
struct timeval {
  time_t tv_sec;          /* seconds */
  unsigned int tv_usec;   /* microseconds */
};
```

Το time_t ήταν μέχρι πρότινος ένας ακέραιος 32bit. Σε 68 χρόνια γίνεται overflow!

-- βλ. “year 2038 problem”

```
$ man gettimeofday
```

Παράδειγμα χρονομέτρησης με την gettimeofday()

```
#include <stdio.h>
#include <sys/time.h>       /* Για την gettimeofday() */

int main() {
  struct timeval tv1, tv2;
  int    i, sum = 0;
  double t;

  gettimeofday(&tv1, NULL);
  for (i = 0; i < 100000000; i++)
    sum += i;
  gettimeofday(&tv2, NULL);

  t = (tv2.tv_sec - tv1.tv_sec) +         /* seconds */
      (tv2.tv_usec - tv1.tv_usec)*1.0E-6; /* convert μsec to sec */

  printf("real time: %lf sec.\n", t);
  return 0;
}
```

Τεχνική: εύρεση της ανάλυσης της gettimeofday()

- Πώς μπορώ να βρω τι ανάλυση (resolution) έχει η `gettimeofday();`
  - Δηλαδή, ποιος είναι ο μικρότερος χρόνος που μπορεί να μετρήσει;

```
struct timeval tv1, tv2;
int resolution;

gettimeofday(&tv1, NULL);
do {
  gettimeofday(&tv2, NULL);
}
while (tv1.tv_usec == tv2.tv_usec);     /* Μέχρι να αλλάξει! */

resolution = tv2.tv_usec - tv1.tv_usec; /* Σε μsec */
```

(θεωρώντας ότι ο χρόνος για την κλήση της gettimeofday() είναι αμελητέος σε σχέση με την ανάλυση)

Χρονομέτρηση με την clock_gettime()

- Η **clock_gettime()** είναι η πλέον σύγχρονη κλήση:
  - Μετρά με βάση κάποιο από τα παρεχόμενα ***ρολόγια***.
  - Σε όλα τα συστήματα POSIX εγγυημένα υπάρχει ένα ρολόι που μετρά πραγματικό χρόνο (**CLOCK_REALTIME**).
  - Διάφορα συστήματα παρέχουν επιπλέον ρολόγια.
    - Π.χ. στο Solaris υπήρχε το CLOCK_HIGHRES (πραγματικού χρόνου με υπερυψηλή ανάλυση)
    - Σε πρόσφατες εκδόσεις του Linux υπάρχει το CLOCK_PROCESS_CPUTIME_ID (για χρόνους καθαρών υπολογισμών στη CPU).
  - `#include <time.h>`.

- Κλήση:

```
clock_gettime(clockid_t clk, struct timespec *tp);

 struct timespec {
    time_t tv_sec;        /* seconds */
    long   tv_nsec;       /* nanoseconds */
 };
```

(στο tv_nsec μπορούμε να βάλουμε από 0 μέχρι 999.999.999).

- Επιπλέον ευκολία:

```
clock_getres(clockid_t clk, struct timespec *tp);
```

  - Στο tp λαμβάνουμε την ανάλυση (resolution) του ρολογιού clk.

```
$ man clock_gettime
```

Παράδειγμα χρονομέτρησης με την clock_gettime()

```
#include <stdio.h>
#include <time.h>       /* Για την clock_gettime() */

int main() {
  struct timespec ts1, ts2;
  int    i, sum = 0;
  double t;

  clock_gettime(CLOCK_REALTIME, &ts1);
  for (i = 0; i < 100000000; i++)
    sum += i;
  clock_gettime(CLOCK_REALTIME, &ts2);

  t = (ts2.tv_sec - ts1.tv_sec) +          /* seconds */
      (ts2.tv_nsec - ts1.tv_nsec)*1.0E-9;  /* convert nsec to sec */
  printf("real time: %lf sec.\n", t);

  clock_getres(CLOCK_REALTIME, &ts1);
  printf("clock resolution: %lf nsec.\n", ts1.tv_sec*1.0E9 + ts1.tv_nsec);
  return 0;
}
```