ANSI C Standart: < *time.h* >

Tüm işletim sistemlerinde zaman alan taşınabilir kodun yolunun olmadığına dair şikayetler duyuyorum.Örneğin Unix üzerinde günlük zaman alma kullanılır.fakat Windows üzerinde günlük Çağrı için zaman alma yoktur. Orada, zamanı ölçmek için saat gibi bir çağrı kullanmak gerekir.

Aslında, bu sadece tarihsel nedenlerden kaynaklanan bir karışıklık. ve bütün bir toplumun o M firması ile ilgili bir küçümsemediği şey. Burada, ANSI C'87'nin tanımlanmış olan, zamanı ölçmek için de facto standartlarının yolları hakkında biraz konuşalım.

ANSI C'yi desteklemeyi iddia eden herhangi bir sistem üzerinde, bir <time.h> olmalıdır.ANSI C standardı C-kütüphanesi işlevleri şunlardır.

```
clock_t      clock(void);
     time_t       time(time_t * tp);
     double       difftime(time_t time2, time_t time1);
     time_t       mktime(struct tm * tp);

     char *       ctime(const time_t *clock);
     struct tm *  localtime(const time_t *clock);
     struct tm *  gmtime(const time_t *clock);
     char *       asctime(const struct tm *tm);
     size_t       strftime(const char *s, size_t  maxsize,  const  char
                         *format, const struct tm *timeptr);
```

Structm okunabilir biçim içine zaman bölmede kullanılır.

```
struct tm {
     int   tm_sec;    /* seconds after the minute - [0, 61] */
                      /* for leap seconds */
     int   tm_min;    /* minutes after the hour - [0, 59] */
     int   tm_hour;   /* hour since midnight - [0, 23] */
     int   tm_mday;   /* day of the month - [1, 31] */
     int   tm_mon;    /* months since January - [0, 11] */
     int   tm_year;   /* years since 1900 */
     int   tm_wday;   /* days since Sunday - [0, 6] */
     int   tm_yday;   /* days since January 1 - [0, 365] */
     int   tm_isdst;  /* daylight saving time tag */
}
```

Bu standart'da, zaman dönüş değeri (gerçekten sadece büyük bir aritmetik sayıdır time_t) takvim zamanı 00:00:00, 1 Ocak 1970 (evrensel ortak saat) gelen saniye sayısı ile ölçülür.Daha doğrusu bu süreçte program başladığında saat,saat keneleri(parçaları)sayısı haline dönüşür.

Localtime ve gmtime gibi çağrılar, yerel saat dilimi veya Greenwich saati (çoğunlukla tarihi önemi olan) ya da bir tm yapısının içine takvim süresi dönüştürür.

Kesinlikle diziler bir yapı veya bir time_t daha iyi olduğundan Insan geçerli zamanı anlamak için, dizileri kullanır.. Bu amaçla, iki işlev çağrıları tasarlanmıştır. ctime ve asctime bir takvim zaman ya da bir struct tm alır ve formda Saat biçiminde bir diziye dönüştürür:

```
Sun Jan 3 15:14:13 1998\n\0
```

strftime her türlü printf denetim önerileri ile biçimlendirme zamanı sunan bir işlevdir.

Unix günün zaman alması

Doğrudan günün saatini almak kullanarak,ölçme zamanı geri geliyor, bu,Unix üzerinde bir sistem çağrısıdır. En özet şudur:

```
#include < sys/time.h >

   int
   gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

Yapılar sys/time.h içinde tanımlanmış tp ve stp tarafından işaret edilir.

```
struct timeval {
            time_t       tv_sec;   /* seconds since Jan. 1, 1970 */
            suseconds_t  tv_usec;  /* and microseconds */
    };
```

Duvar Saati Zamanı vs İşlem Süresi

Bugüne kadar, biz takvim süresi kategorisi altına düşen tüm zamanlama ölçümleri hakkında konuştuk, veya Epoch, 00:00:00 1 Ocak 1970, eşgüdümlü evrensel saat (çünkü sistem tarafından tutulan saniye sayısıdır duvar saati süresi var UTC). UTC de Greenwich Saati olarak da bilinir.

Tüm bilinen süreçler hakkından sonra,işlem süresi(cpu time) olarak da adlandırılan süreç zamanını(process time) konuşalım.Süreç zamanı merkezi işlemcinin diğer kaynaklar tarafından ne kadar başvurulduğunu ifade eden bir kavramdır.Takvim zamanın da olduğu gibi süreç zamanı da zaman keneleri(parçaları)içinde tarihsel olarak saniyede 50, 60 ya da 100 parçalar ile ölçülür.

Her bir işlem için, sistem üç değer korur:

- Saat süresi (duvar saatini)
- Kullanıcı CPU süresi
- Sistem CPU süresi

Unix üzerinde, işlem süresi çağrılarak toplanır.

```
#include < sys/times.h >
  clock_t times (struct tms * buf);
```

Tms yapısı aşağıdaki yapıya sahiptir.

```
struct tms {
    clock_t tms_utime;  /* user CPU time */
    clock_t tms_stime;  /* system CPU time */
    clock_t tms_cutime; /* user CPU time, terminated children */
    clock_t tms_cstime; /* system CPU time, terminated children */
  }
```

Not:dönüş değeri zamanı geçmişte bir keyfi noktadan ölçülen duvar saati zamanıdır.bir mutlak değeri kullanmaz,fakat oldukça görecelidir. İşte o kod segmentin duvar saati yürütme zamanını almak için programda iki noktada elde edilen iki dönüş değerleri arasındaki farkın alınması budur.En iyi olarak ANSI c standardın da duvar saati süresini ölçmek için time.h kullanılabilir fakat işlem süresinde kullanılamaz.

U/stime-geçerli sürecin kullanıcı alanı veya sistem çağrıları, sırasıyla harcadığı cpu zamanı ölçer. CU/stime wait\ olan geçerli işlemler tüm yavru(processleri) işlemleri için aynı miktarları ölçer ' ed.

Her bir saniyede ne kadar parçanın olduğunu öğrenmek için,kullanılır.

```
#include < unistd.h >
  long clktck = 0;
  clktck = sysconf(_SC_CLK_TCK);
```

En aşağıdaki koda bakalım.

```
/* timetest.c */
#include < stdio.h >
#include < stdlib.h >

#include < time.h >
#include < sys/time.h >
#include < sys/times.h >
#include < unistd.h >

typedef struct {
  clock_t        clock_val;
  time_t         time_val;
```

```
  struct timeval gettime_val;
  struct tms     times_val;
} all_times;

void get_time(all_times * t)
{
  t->clock_val = clock();
  t->time_val = time(NULL);
  gettimeofday(&(t->gettime_val), NULL);
  times(&(t->times_val));
}

void print_duration(all_times * t1, all_times * t2)
{
  float elapsed;
  long clktck, nticks;

  nticks = t2->clock_val - t1->clock_val;
  elapsed = (float)nticks/CLOCKS_PER_SEC;
  fprintf(stderr,"clock: %f sec\n", elapsed);

  nticks = t2->time_val - t1->time_val;
  fprintf(stderr,"time: %d sec\n", nticks);

  elapsed = t2->gettime_val.tv_sec - t1->gettime_val.tv_sec;
  elapsed += (t2->gettime_val.tv_usec - t1->gettime_val.tv_usec)/1e6;
  fprintf(stderr,"gettimeofday: %f sec\n", elapsed);

  clktck = sysconf(_SC_CLK_TCK);
  nticks = t2->times_val.tms_utime - t1->times_val.tms_utime;
  elapsed = (float)nticks/clktck;
  fprintf(stderr,"times: %f sec user time\n", elapsed);

  nticks = t2->times_val.tms_stime - t1->times_val.tms_stime;
  elapsed = (float)nticks/clktck;
  fprintf(stderr,"times: %f sec system time\n", elapsed);
}

long simplecat()
{
  char c;
  int i;
  long cnt = 0;

  i = fread(&c, 1, 1, stdin);
  while(i > 0) {
    fwrite(&c, 1, 1, stdout);
    i = fread(&c, 1, 1, stdin);
    cnt ++;
  }

  return cnt;
}

int main(void)
{
  int totalbytes;

  all_times a1, a2;

  get_time(&a1);
```

```
  totalbytes = simplecat();

  get_time(&a2);
  print_duration(&a1, &a2);

  fprintf(stderr, "total of %f MiB\n", totalbytes/1e6);

  return 0;
}
```

Yukarıdaki kod çıkışı iki senaryoda oldukça farklı görünüyor.Hadi bunun nedenlerini araştıralım ☺

```
UNIX> timetest < gigantic.jpeg > /dev/null
clock: 2.140000 sec
time: 2 sec
gettimeofday: 2.193709 sec
times: 2.120000 sec user time
times: 0.020000 sec system time
total of 59.134079 MiB
UNIX> timetest
1
1
2
2
3
3
4
4
5
5
6
6
clock: 0.000000 sec
time: 8 sec
gettimeofday: 7.487184 sec
times: 0.000000 sec user time
times: 0.000000 sec system time
total of 0.000012 MiB
```