Gestão do tempo

Este módulo trata dos aspectos de programação relacionados à gestão do tempo em um ambiente UNIX. Os seguintes tópicos são abordados:

- referências de tempo
- tempo absoluto
- funções de calendário
- o relógio do sistema
- temporizadores
- esperas

Antes de iniciar, é importante definir corretamente alguns termos relacionados a tempo:

- **Tempo absoluto** (*calendar time*) é um instante preciso na referência de tempo universal, como 18/10/2004, 14:00:00.000 GMT-3. Uma data é parte do tempo absoluto.
- **Intervalo** é uma parte contínua de tempo entre dois tempos absolutos. Um exemplo seria o intervalo de 13:30:00 a 17:50:00 do dia 18/10/2004.
- **Tempo decorrido** (*elapsed time*) é o tamanho de um intervalo. No exemplo acima, o tempo decorrido seria de 4:20:00.
- A **duração** (*amount of time*) é a soma dos tempos decorridos em todos os intervalos considerados. Por exemplo, a duração de nosso curso é de 60 horas, distribuídas entre vários intervalos.
- O **período** é o tempo decorrido no intervalo entre dois eventos, considerado sobretudo quando esses eventos são parte de uma seqüência de eventos repetitivos.
- O **tempo de CPU** (*CPU time*) é similar ao tempo absoluto, mas considerado para cada processo em particular.
- O **tempo de processamento** (*processor time*) é a duração de uso da CPU por um ou mais processos.

Relógios no Linux

Um sistema Linux possui dois relógios: o **relógio de hardware** e o **relógio do sistema**.

O *relógio de hardware* avança sem interferência do sistema operacional, mesmo quando a máquina está desligada. Sua resolução faz parte da especificação do hardware. Ele também é conhecido como relógio da BIOS, relógio CMOS ou RTC (*Real-Time Clock*). No Linux, pode-se manipular o RTC através do comando `hwclock`.

O *relógio do sistema* é um valor de tempo mantido pelo núcleo do sistema operacional. Ele é inicializado com o valor do RTC durante a carga do sistema e mantido atualizado através de interrupções de tempo geradas pelo hardware. O RTC não é consultado frequentemente durante a vida ativa do sistema.

Tempo absoluto

Várias funções permitem consultar/ajustar o tempo absoluto do sistema ou manipular variáveis contendo informações de tempo absoluto. Várias granularidades de tempo são possíveis, do segundo ao micro-segundo. Em todas as funções a seguir, o ajuste do tempo absoluto do sistema só pode ser feito por processos do administrador.

Tempo absoluto simples

```
#include <time.h>
time_t time (time_t *result)
int stime (time_t *newtime)
```

Estas funções permitem consultar ou ajustar o tempo absoluto do sistema com uma granularidade de segundos. O tipo `time_t` pode representar o tempo absoluto ou tempos decorridos. Quando usado para representar o tempo absoluto, indica o número de segundos decorridos desde 01/01/1970 00:00:00 UTC (esse instante é conhecido como *epoch*). Fusos horários não são considerados.

```
#include <time.h>
double difftime (time_t time1, time_t time0)
```

Retorna o número de segundos decorridos entre os dois tempos absolutos informados. Essa é a única forma portável de fazer esse cálculo, pois a implementação do tipo `time_t` pode mudar entre sistemas.

Um pequeno exemplo do uso das funções `time` e `difftime`:

```
#include <time.h>

int main (int argc, char * argv[])
{
   time_t antes, depois ;

   antes  = time (0) ;
   sleep (5) ;
   depois = time (0) ;

   printf ("Entre %d e %d se passaram %f segundos\n",
           antes, depois, difftime (depois, antes)) ;
}
```

Tempo absoluto preciso

As funções abaixo indicadas permitem obter/ajustar tempos absolutos com mais resolução que as anteriores:

```
#include <sys/time.h>
int gettimeofday (struct timeval *tp, struct timezone *tzp)
int settimeofday (const struct timeval *tp, const struct timezone *tzp)
```

Estas funções permitem informar ou ajustar o tempo absoluto do sistema (desde *epoch*). O tempo é representado pela estrutura `struct timeval` descrita a seguir:

```
#include <sys/time.h>
struct timeval
{
   long int tv_sec  ; // segundo decorridos
   long int tv_usec ; // micro-segundos decorridos (valor mínimo depende do
sistema)
}
```

O fuso horário local é informado através do `struct timezone *tzp` (nos sistemas atuais esse ponteiro deve ser nulo, pois essa é uma característica obsoleta do UNIX 4.3 BSD mantida nas funções para preservar a compatibilidade).

```
#include <sys/time.h>
int adjtime (const struct timeval *delta, struct timeval *olddelta)
```

Ajustar abruptamente o relógio do sistema gera descontinuidades que podem interferir no funcionamento de vários processos. Esta função permite acelerar ou retardar o relógio do sistema gradualmente para fazer o ajuste indicado por `delta` (que pode ser positivo ou negativo), garantindo que o tempo do sistema não sofra descontinuidades.

Um pequeno exemplo do uso da função `gettimeofday`:

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

int main (int argc, char * argv[])
{
   struct timeval antes, depois ;
   float delta ;

   gettimeofday (&antes, 0) ;
   sleep (2) ;
   gettimeofday (&depois, 0) ;

   delta = (depois.tv_sec + depois.tv_usec/1000000.0) -
           (antes.tv_sec  + antes.tv_usec /1000000.0) ;
   printf ("Diferenca de %f segundos\n", delta) ;
}
```

Datas

Os tempos reportados pelas funções acima são úteis para computações, mas não são significativos para seres humanos. Portanto, são de pouca utilidade em relatórios, mensagens de erro, diálogos, etc. Para obter valores de tempo significativos aos humanos é necessário quebrar os resultados das funções anteriores em *dia/mês/ano* e *hora:minuto:segundo* (relativos a um fuso horário). As funções abaixo permitem fazer essa conversão:

```
#include <time.h>
struct tm * localtime (const time_t *time)
```

Converte o tempo informato em time em uma estrutura `struct tm` que indica o tempo equivalente, expresso no fuso horário local do sistema (a variável global `tzname` recebe informações sobre o fuso horário local). Retorna `null` se a conversão não for possível. O formato da estrutura retornada está descrito a seguir:

```
#include <time.h>
struct tm
{
   int tm_sec ;   segundos (0..59)
   int tm_min ;   minutos (0..59)
   int tm_hour ;  horas (0..23)
   int tm_mday ;  dia do mês (1..31)
   int tm_mon ;   mês do ano (0..11)
   int tm_year ;  ano (com dois dígitos, contado a partir de 1900)
   int tm_wday ;  dia da semana a partir de domingo (0..6)
   int tm_yday ;  dia do ano (0..365)
   int tm_isdst ; é horário de verão nessa data ? (+ : sim, 0 : não, - : informação
não disponível).
```

```
}
```

```
#include <time.h>
struct tm * gmtime (const time_t *time)
```

Função similar a `localtime`, mas o tempo obtido é expresso com relação ao UTC (*Coordinated Universal Time*, antes conhecido como GMT - *Greenwich Mean Time*) ao invés de usar o fuso horário local.

```
#include <time.h>
time_t mktime (struct tm *brokentime)
```

Esta função converte uma data no formato `struct tm` ao formato `time_t` usado na função `time`. Ela também completa a estrutura `tm`, preenchendo os campos *dia da semana* e *dia do ano*, com base nos demais campos. O valor -1 é retornado em caso de erro.

```
#include <time.h>
char * asctime (const struct tm *brokentime)
char * ctime   (const time_t *time)
```

Convertem a data descrita em `brokentime` ou `time` em uma string no formato padrão "`Mon Oct 18 14:17:09 2004\n`". A função `strftime` permite um maior controle sobre o formato da string gerada; conversões no sentido contrário (de string para data interna) são oferecidas pelas funções `strptime` e `getdate` (vide manual da GLibC).

Um pequeno exemplo do uso da função `localtime`:

```
#include <time.h>

int main (int argc, char * argv[])
{
   time_t      ct ;
   struct tm  *lt ;

   ct = time (NULL) ;
   lt = localtime (&ct) ;

   printf ("%02d/%02d/%04d %02d:%02d:%02d\n",
           lt->tm_mday, lt->tm_mon, lt->tm_year + 1900,
           lt->tm_hour, lt->tm_min, lt->tm_sec) ;
}
```

O relógio do sistema

Algumas funções permitem a um processo conhecer seu uso de tempo de processador:

```
#include <time.h>
clock_t clock(void)
```

Informa o tempo de CPU usado pelo processo, em pulsos de relógio (*clock ticks*). Para conhecer o tempo em segundos, o valor informado deve ser dividido pela constante CLOCKS_PER_SEC (definida na norma POSIX como 1.000.000, independentemente do sistema).

Importante: o valor absoluto de tempo retornado pela função `clock` não tem significado isolado. Portanto essa função somente deve ser usada para medir intervalos, como mostra o exemplo a seguir:

```
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
... /* do the work */
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

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

Esta função retorna informações detalhadas sobre o uso de tempo de processador pelo processo corrente e seus descendentes em `buffer`. O valor de retorno é idêntico ao da função `clock` (ou -1 em caso de erro).

A estrutura `tms` usada como parâmetro tem o seguinte formato:

```
#include <sys/times.h>
struct tms
{
   clock_t tms_utime ;  tempo de processador usado pelo programa em modo usuário
   clock_t tms_stime ;  tempo de processador usado pelo programa em modo núcleo
   clock_t tms_cutime ; soma de tms_utime e tms_cutime para todos os filhos já
terminados
   clock_t tms_cstime ; idem, para tms_stime e tms_cstime
}
```

Todos os tempos são informados em pulsos de relógio (*clock ticks*); ao contrário da função `clock`, os valores aqui informados são valores reais e podem ser usados diretamente.

Temporizadores

As funções `alarm` e `setitimer` fornecem mecanismos para um processo interromper a si próprio em algum instante no futuro. Elas permitem armar temporizadores que, ao disparar, geram sinais para o processo. Cada processo tem à sua disposição três temporizadores independentes:

- `ITIMER_REAL` : um temporizador de tempo real, que mede o tempo decorrido; ele gera um sinal `SIGALRM` ao disparar.
- `ITIMER_VIRTUAL` : um temporizador de tempo virtual, que mede o tempo de processador gasto pelo processo; ele gera um `SIGVTALRM` ao disparar.
- `ITIMER_PROF` : um temporizador de profiling, que mede o tempo usado pelo processo em modo usuário e modo núcleo; ele gera um sinal `SIGPROF` ao disparar.

O programador deve definir um tratador (*handler*) para o sinal gerado pelo temporizador **antes de armá-lo**. A ação default dos sinais de temporizadores é abortar o processo, caso um tratador não tenha sido previamente definido.

```
#include <sys/time.h>
int setitimer (int which, struct itimerval *new, struct itimerval *old)
```

Arma o temporizador indicado por wich de acordo com o prazo definido em new. Os valores de wich podem ser `ITIMER_REAL`, `ITIMER_VIRTUAL` ou `ITIMER_PROF`. Se `old` for indicado, retorna informação sobre o valor anterior do prazo.

```
#include <sys/time.h>
int getitimer (int which, struct itimerval *old)
```

Retorna informação sobre o temporizador indicado em `wich` na estrutura `old`.

A estrutura itimerval usada pela funções acima tem o seguinte formato (a estrutura `timeval` foi descrita anteriormente):

```
#include <sys/time.h>
struct itimerval
{
   struct timeval it_value ; Prazo para o primeiro disparo do temporizador
   struct timeval it_interval ; Período entre disparos sucessivos, ou zero (só um
disparo)
}
```

```
#include <unistd.h>
unsigned int alarm (unsigned int seconds)
```

Esta função é uma forma mais simples (e antiga) de armar o temporizador `ITIMER_REAL`.

Um pequeno exemplo do uso de temporizadores:

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

// tratador do sinal SIGALRM (disparo do timer)
void disparo_timer (int signum)
{
   printf ("Recebi o sinal %d\n", signum) ;
}

int main ()
{
   struct itimerval timer ;
   struct sigaction action ;

   // define a ação para o sinal SIGALRM
   action.sa_handler = disparo_timer ;
   sigemptyset (&action.sa_mask);
   action.sa_flags = 0;
   sigaction (SIGALRM, &action, 0);

   // ajusta valores do temporizador
   timer.it_interval.tv_usec = 0;  // disparos sucessivos, micro-segundos
   timer.it_interval.tv_sec = 1;   // disparos sucessivos, segundos
   timer.it_value.tv_usec = 0;     // primeiro disparo, micro-segundos
   timer.it_value.tv_sec = 3 ;     // primeiro disparo, segundos

   // arma o temporizador
   setitimer (ITIMER_REAL, &timer, NULL) ;

   // laço vazio
   while (1)
     sleep (1);
```

```
}
```

Sleeping

A função `sleep` oferece uma forma simples de fazer um processo "dormir" por alguns segundos. Ela aguarda o tempo definido em seconds ou a chegada de um sinal (o que ocorrer antes):

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds)
```

O processo será "acordado" (retornará da chamada) mais cedo se receber um sinal externo. Para "tirar uma soneca" sem ser acordado por sinais, o adequado é usar a chamada de I/O `select` (definindo o *time-out* desejado e um conjunto vazio de descritores de arquivos a monitorar).

```
#include <time.h>
int nanosleep (const struct timespec *requested_time, struct timespec *remaining)
```

Esta função permite definir períodos de espera inferiores a um segundo. A duração da espera é definida pela estrutura `requested_time`, mas pode ser arredondada para um múltiplo da resolução mínima do relógio do sistema. A estrutura `remaining` informa o tempo de espera que sobrou, caso o processo seja acordado antes da hora por um sinal.

As estruturas `requested_time` e `remaining` têm o seguinte formato:

```
#include <time.h>
struct timespec
{
   long int tv_sec ;  número de segundos
   long int tv_nsec ; número de nano-segundos.
}
```

Atividades

- Construa uma função `elapsed()` que gere a seguinte saída na tela a cada invocação:

```
Se passaram 3.029384 segundos desde o início do programa
```

- Construa um mecanismo baseado em temporizador que informe na tela a seguinte saída, uma vez por minuto:

```
Hoje é 18/10 e agora são 15:30
```

- O utilitário UNIX `time` permite medir o tempo de execução de um comando qualquer. Ele funciona da seguinte forma:

```
$ time ls -R /usr
...
0.580u 1.070s 0:31.66 5.2%      0+0k 0+0io 182pf+0w
```

Ao terminar o comando, são informados o tempo de execução do processo em modo usuário, em modo núcleo, o tempo total decorrido na execução e algumas outras informações sobre entrada/saída. Construa uma função que, ao ser chamada no final do programa, reporte essas mesmas informações de tempo.