Exercicio: Implementar 3 tarefas periodicas no GNU-Linux

Descrição do problema:

Tarefas periódicas como o nome indica são tarefas que são executadas em tempos conhecidos. Por exemplo, uma tarefa periodica com o período 2 segundos é executada a cada 2 segundos.

Uma ideia de uma tarefa periodica seria:

```
while(1)
{
      Executa_funcao();
      dorme_por_tempo_fixo(T);
}
```

Neste exemplo, temos um laco infinito que chama a funcao “Executa_funcao” e depois dorme por um tempo T. Dessa forma essa tarefa seria executada a cada T unidades de tempo. Infelizmente esse exemplo só serve para dar uma ideia do problema a ser tratado. Antes, convem examinar as interfaces da API que existem para obter o tempo e como fazer com que um processo durma por um dado tempo T.

Tambem deve ser estudado os problemas de precisao das funcoes para fazer um processo dormir.

Funções relacionadas ao tempo

```
unsigned int sleep(unsigned int seconds);
```

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
struct timeval {
   time_t tv_sec; /* seconds */
   suseconds_t tv_usec; /* microseconds */};
struct timezone {
   int tz_minuteswest; /* minutes W of Greenwich */
   int tz_dsttime; /* type of dst correction */};
```

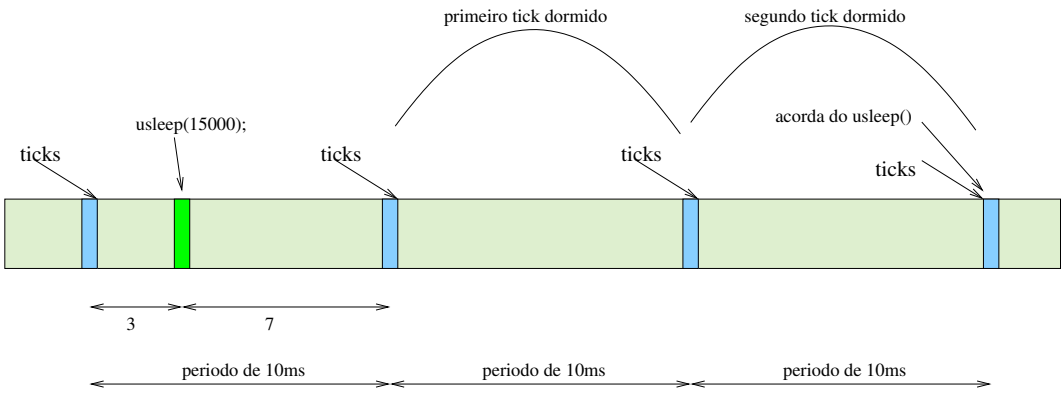
```
int nanosleep(const struct timespec *req, struct timespec *rem);
struct timespec
{
   time_t tv_sec; /* seconds */
   long tv_nsec; /* nanoseconds */
};
```

```
void usleep(unsigned long usec);
```

Imprecisao no GNU-Linux

As funções de marcação de tempo, tais como usleep() e nanosleep() deveriam ter precisa de micro-segundo($10^{-6}s$) e nano-segundo ($10^{-9}s$). Infelizmente no GNU-Linux a precisao destas funções está ligada ao sistemad e marcação de tempo, os ticks de relogio do sistema. Nos computatores intel-compativeis, esses ticks ocorrem a cada $10ms$.

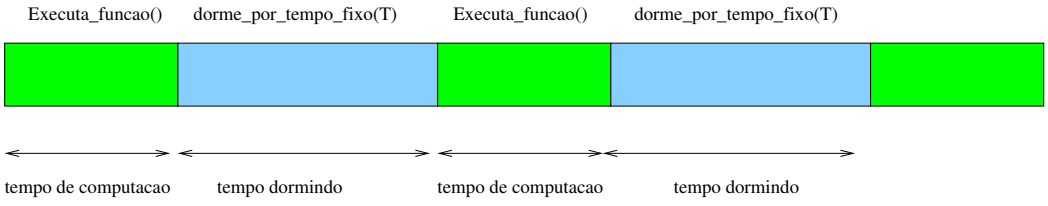
Se queremos que um processo durma por $15ms$, o sistema entende que ele deve dormir por no mínimo $15ms$ e não exatamente. Como toda a contagem de tempo funciona com os ticks de $10ms$, o tal processo durmirá por 2 ticks. Além disso, o processo não iniciará a executar a função usleep num tempo coincidente com o tick do relogio. Se a chamada a usleep() ocorrer $7ms$ antes do tick do relogio, o processo dormira os 2 ticks necessários + os $7ms$ resultando em $27ms$.



Questão da tarefa periódica

No exemplo da tarefa periódica, o tempo de computação da função "Executa_funcao()" é considerado constante. Numa situação real, diversos processos estão concorrendo pela CPU e estão ocorrendo chaveamentos entre estes processos, bloqueios por causa de recursos, situações em que o processador executa com interrupções desabilitadas, etc.

Todo isso faz com que o tempo de computação da função não seja constante e para comprensar o tempo de dormir também deve ser variável.



Exemplo de tarefa periódica

Depois de conhecer um pouco da peculiaridade de ligar com tarefas periódicas no GNU-Linux, vamos examinar um código em C de uma tarefa periódica. Esse código esta em `http://www.lcmi.ufsc.br/~romulo/discipli/cad-ii2/exemplo-periodica.c`. Para maiores informações sobre tarefas periódicas veja em `http://www.lcmi.ufsc.br/~romulo/reltec/linux.pdf`.

```
#include <stdio.h>
#include <sys/time.h>
#include <sched.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
#include <math.h>

#define DEZNA6          1000000

#define MAX_ATIVACOES   1000

#define PERIODO         200000

long inicio_periodo[MAX_ATIVACOES+1];
long sleep_pedido[MAX_ATIVACOES+1];
long iniciou[MAX_ATIVACOES+1];
long terminou[MAX_ATIVACOES+1];


/* Retorna o instante atual em relação a base,
   em microsegundos */

long agora(long base){
     struct timeval  tvA;
     long  agora_real;

     gettimeofday( &tvA, NULL);
     agora_real = ((tvA.tv_sec * DEZNA6) + tvA.tv_usec );
     return agora_real - base;
}


void tarefa(char *arg){
     long base; /* Inicio dos tempos para a tarefa */
     int a; /* Numero da ativação atual */

     struct timespec ts; /* Variável para o nanosleep */
     struct timespec tsr; /* Variável para o nanosleep */
     struct timeval tv; /* Variável para ler a hora */
```

```
        int i; /* Usado pela falsa aplicação */
        double sum = 0.0; /* Usado pela falsa aplicação */
        float avg; /* Usado pela falsa aplicação */
        long t1[1000]; /* Usado pela falsa aplicação */
        long t2[1000]; /* Usado pela falsa aplicação */


        /* Dorme 1 segundo para sincronizar com interrupções do timer */
        sleep(1);

        /* Determina o instante zero da tarefa */
        gettimeofday(&tv,NULL);
        base = (tv.tv_sec * DEZNA6) + tv.tv_usec;

        /* Executa número limitado de ativações */
        a = 1;
        while( a <= MAX_ATIVACOES ) {
             inicio_periodo[a] = a * PERIODO;

             sleep_pedido[a-1] = (a * PERIODO) - agora(base);

             ts.tv_sec = 0;
             ts.tv_nsec = sleep_pedido[a-1] * 1000; /* em nanosegundos */
             tsr.tv_sec=0;
             tsr.tv_nsec=0;
             nanosleep(&ts, &tsr);

             iniciou[a] = agora(base);


             for (i=0;i<1000;i++){ /* Algoritmo */
                  sum=(((t1[i]/131.7)/131.45)); /* da        */
                  avg=((sum/t2[i])-sum); /* aplicação */
             }

             terminou[a] = agora(base);
             a++;
        }
}
```

```
int main(void){

     pthread_t t0;
     int i;

     pthread_create( &t0, NULL, (void *)tarefa, "0");

     pthread_join(t0, NULL);

     /* Mostra as medidas */

     printf("sleep_pedido=%ld\n", sleep_pedido[0] );

     for(i=1; i <= MAX_ATIVACOES; ++i)
          printf("Chegada=%7ld, inicio=%7ld, conclusao=%7ld, sleep_pedido=%7l
               inicio_periodo[i], iniciou[i], terminou[i], sleep_pedido[i] );

     for(i=1; i <= MAX_ATIVACOES; ++i)
          printf("Liberacao=%7ld, Resposta=%7ld, Conclusao-inicio=%7ld\n",
               iniciou[i] - inicio_periodo[i], terminou[i] - inicio_periodo[i
               terminou[i] - iniciou[i] );

     return 0;
}
```