```
/*********************************************************************************************/
/* clock()、time()、clock_gettime()和 gettimeofday()函数的用法和区别                          */
/* 转自 http://blog.sina.com.cn/s/blog_790f5ae10100rwd3.html                                  */
//*********************************************************************************************/

#ifdef ANSI_CLOCK
/*********************************************************************************************
一)ANSI clock 函数

1)概述:
clock 函数的返回值类型是 clock_t, 它除以 CLOCKS_PER_SEC 来得出时间, 一般用两次 clock 函数来计算进程自身运行的时间.
ANSI clock 有三个问题:
1) 如果超过一个小时, 将要导致溢出.
2) 函数 clock 没有考虑 CPU 被子进程使用的情况.
3) 也不能区分用户空间和内核空间.
→ 所以 clock 函数在 Linux 系统上变得没有意义.
2)测试
编写 test1.c 程序,测试采用 clock 函数的输出与 time 程序的区别.
*********************************************************************************************/

//vi test1.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void )
{
	long i=8L;
	lock_t start, finish;
	double  duration;

	printf( "Time to do %ld empty loops is ", i );
	start = clock();
	while (--i)
		system("cd");
	finish = clock();
	duration = (double)(finish - start) / CLOCKS_PER_SEC;

	printf( "%f seconds\n", duration );
	return 0;
}

/*
Time to do 1000 empty loops is 0.180000 seconds
real	0m3.492s
user	0m0.512s
sys	0m2.972s
3)总结:
(1)程序调用 system("cd");，这里主要是系统模式子进程的消耗,test1 程序不能体现这一点.
(2)0.180000 seconds 秒的消耗是两次 clock()函数调用除以 CLOCKS_PER_SEC.
(3)clock()函数返回值是一个相对时间，而不是绝对时间.
(4)CLOCKS_PER_SEC 是系统定义的宏，由 GNU 标准库定义为 1000000.
*/
#endif

#ifdef TIME_FUNC
/*********************************************************************************************

二)times()时间函数
1)概述:
```

原型如下：
clock_t times(struct tms *buf);
tms 结构体如下:
strace tms
{
	clock_t tms_utime;
	clock_t tms_stime;
	clock_t tms_cutime;
	clock_t tms_cstime;
}
注释:
tms_utime 记录的是进程执行用户代码的时间.
tms_stime 记录的是进程执行内核代码的时间.
tms_cutime 记录的是子进程执行用户代码的时间.
tms_cstime 记录的是子进程执行内核代码的时间.

2)测试:

```
*****************************************************************************************************/
//vi test2.c
#include <sys/times.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
static void do_cmd(char *);
static void pr_times(clock_t, struct tms *, struct tms *);

int main(int argc, char *argv[])
{
	int i;
	char *c[5] = {"dd", "if=/dev/zero", "f=/dev/null", "bs=1M", "count=10000"};
	argv = c;

	for(i=1; argv[i]!=NULL; i++)
		do_cmd(argv[i]);

	exit(1);
}

static void do_cmd(char *cmd)
{
	struct tms tmsstart, tmsend;
	clock_t start, end;
	int status;
	if((start=times(&tmsstart))== -1)
		puts("times error");
	if((status=system(cmd))<0)
		puts("system error");
	if((end=times(&tmsend))== -1)
		puts("times error");
	pr_times(end-start, &tmsstart, &tmsend);
	exit(0);
}

static void pr_times(clock_t real, struct tms *tmsstart, struct tms *tmsend)
{
	static long clktck=0;
	if(0 == clktck)
		if((clktck=sysconf(_SC_CLK_TCK))<0)
			puts("sysconf err");
```

```
    printf("real:%7.2f\n", real/(double)clktck);
    printf("user-cpu:%7.2f\n", (tmsend->tms_utime - tmsstart->tms_utime)/(double)clktck);
    printf("system-cpu:%7.2f\n", (tmsend->tms_stime - tmsstart->tms_stime)/(double)clktck);
    printf("child-user-cpu:%7.2f\n", (tmsend->tms_cutime - tmsstart->tms_cutime)/(double)clktck);
    printf("child-system-cpu:%7.2f\n", (tmsend->tms_cstime - tmsstart->tms_cstime)/(double)clktck);
}
/*
编译:
gcc test2.c -o test2
测试这个程序:
time ./test2 "dd if=/dev/zero f=/dev/null bs=1M count=10000"
10000+0 records in
10000+0 records out
10485760000 bytes (10 GB) copied, 4.93028 s, 2.1 GB/s
real:   4.94
user-cpu:   0.00
system-cpu:   0.00
child-user-cpu:   0.01
child-system-cpu:   4.82
real    0m4.943s
user    0m0.016s
sys     0m4.828s

3)总结:
(1)通过这个测试,系统的 time 程序与 test2 程序输出基本一致了.
(2)(double)clktck 是通过 clktck=sysconf(_SC_CLK_TCK)来取的,也就是要得到 user-cpu 所占用的时间,就要用
(tmsend->tms_utime - tmsstart->tms_utime)/(double)clktck);
(3)clock_t times(struct tms *buf);返回值是过去一段时间内时钟嘀嗒的次数.
(4)times()函数返回值也是一个相对时间.
*/
#endif

#ifdef CLOCK_GETTIME
/**********************************************************************************************
三)实时函数 clock_gettime
在 POSIX1003.1 中增添了这个函数,它的原型如下:
int clock_gettime(clockid_t clk_id, struct timespec *tp);
它有以下的特点:
1)它也有一个时间结构体:timespec ,timespec 计算时间次数的单位是十亿分之一秒.
strace timespec
{
    time_t tv_sec;
    long tv_nsec;
}
2)clockid_t 是确定哪个时钟类型.
CLOCK_REALTIME: 标准 POSIX 实时时钟
CLOCK_MONOTONIC: POSIX 时钟,以恒定速率运行;不会复位和调整,它的取值和 CLOCK_REALTIME 是一样的.
CLOCK_PROCESS_CPUTIME_ID 和 CLOCK_THREAD_CPUTIME_ID 是 CPU 中的硬件计时器中实现的.

3)测试:
**********************************************************************************************/
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#define MILLION 1000000

int main(void)
{
    long int loop = 1000;
    struct timespec tpstart;
    struct timespec tpend;
    long timedif;
```

```
    clock_gettime(CLOCK_MONOTONIC, &tpstart);
    while (--loop){
        system("cd");
    }
    clock_gettime(CLOCK_MONOTONIC, &tpend);
    timedif = MILLION*(tpend.tv_sec-tpstart.tv_sec)+(tpend.tv_nsec-tpstart.tv_nsec)/1000;
    fprintf(stdout, "it took %ld microseconds\n", timedif);
    return 0;
}
/*
编译:
gcc test3.c -lrt -o test3
计算时间:
time ./test3
it took 3463843 microseconds
real    0m3.467s
user    0m0.512s
sys     0m2.936s
*/
#endif

#ifdef GETTIME_OF_DAY
/*************************************************************************************************
四)时间函数 gettimeofday()
1)概述:
gettimeofday()可以获得当前系统的时间,是一个绝对值
原型如下：
int gettimeofday ( struct timeval * tv , struct timezone * tz )
timeval 结型体的原型如下:
struct timeval
{
        time_t     tv_sec;
        suseconds_t tv_usec;
};
所以它可以精确到微秒
*************************************************************************************************/
//测试:
#include <sys/time.h>
#include <stdio.h>
#include <unistd.h>
int
main()
{
    int i=10000000;
    struct timeval tvs,tve;
    gettimeofday(&tvs,NULL);
    while (--i);
        sleep(1);  /*1s*/
    gettimeofday(&tve,NULL);
    double span = tve.tv_sec-tvs.tv_sec + (tve.tv_usec-tvs.tv_usec)/1000000.0;
    printf("time: %.12f\n",span);
    return 0;
}
/*
gcc test5.c
./a.out
time: 0.041239000000
*/
#endif

/*************************************************************************************************
五)四种时间函数的比较
```

1)精确度比较:
以下是各种精确度的类型转换:
1 秒=1000 毫秒(ms), 1 毫秒=1/1000 秒(s);
1 秒=1000000 微秒(μ s), 1 微秒=1/1000000 秒(s);
1 秒=1000000000 纳秒(ns),1 纳秒=1/1000000000 秒(s);

2)
clock()函数的精确度是 10 毫秒(ms)
times()函数的精确度是 10 毫秒(ms)
gettimofday()函数的精确度是微秒(μ s)
clock_gettime()函数的计量单位为十亿分之一，也就是纳秒(ns)

3)测试 4 种函数的精确度:
vi test4.c

```
**********************************************************************************************************/
#include   <stdio.h>
#include   <stdlib.h>
#include   <unistd.h>
#include   <time.h>
#include   <sys/times.h>
#include   <sys/time.h>
#define    WAIT()     for(i=0;i<298765432;i++)
#define    MILLION    1000000

int main(int argc, char *argv[])
{
   int i;
   long ttt;
   clock_t s,e;
   struct tms aaa;


   s=clock();
   WAIT();
   e=clock();
   printf("clock time : %.12f\n",(e-s)/(double)CLOCKS_PER_SEC);

   long tps = sysconf(_SC_CLK_TCK);
   s=times(&aaa);
   WAIT();
   e=times(&aaa);
   printf("times time : %.12f\n",(e-s)/(double)tps);

   struct timeval tvs,tve;
   gettimeofday(&tvs,NULL);
   WAIT();
   gettimeofday(&tve,NULL);
   double span = tve.tv_sec-tvs.tv_sec + (tve.tv_usec-tvs.tv_usec)/1000000.0;
   printf("gettimeofday time: %.12f\n",span);

   struct timespec tpstart;
   struct timespec tpend;
   clock_gettime(CLOCK_REALTIME, &tpstart);
   WAIT();
   clock_gettime(CLOCK_REALTIME, &tpend);
   double timedif = (tpend.tv_sec-tpstart.tv_sec)+(tpend.tv_nsec-tpstart.tv_nsec)/1000000000.0;
   printf("clock_gettime time: %.12f\n", timedif);
   return EXIT_SUCCESS;
}
/*
```

```
gcc -lrt test4.c -o test4
debian:/tmp# ./test4
clock time : 1.190000000000
times time : 1.180000000000
gettimeofday time: 1.186477000000
clock_gettime time: 1.179271718000
```

六)内核时钟

默认的 Linux 时钟周期是 100HZ,而现在最新的内核时钟周期默认为 250HZ.
如何得到内核的时钟周期呢?

```
grep ^CONFIG_HZ /boot/config-2.6.26-1-xen-amd64
CONFIG_HZ_250=y
CONFIG_HZ=250
```

结果就是 250HZ.
而用 sysconf(_SC_CLK_TCK);得到的却是 100HZ
例如:

```
#include   <stdio.h>
#include   <stdlib.h>
#include   <unistd.h>
#include   <time.h>
#include   <sys/times.h>
#include   <sys/time.h>
int
main ( int argc, char *argv[] )
{
   long tps = sysconf(_SC_CLK_TCK);
   printf("%ld\n", tps);

   return EXIT_SUCCESS;
}
```

为什么得到的是不同的值呢?
因为 sysconf(_SC_CLK_TCK)和 CONFIG_HZ 所代表的意义是不同的.
sysconf(_SC_CLK_TCK)是 GNU 标准库的 clock_t 频率.
它的定义位置在:/usr/include/asm/param.h
例如:

```
#ifndef HZ
#define HZ 100
#endif
```

最后总结一下内核时间:

内核的标准时间是 jiffy,一 jiffy 就是一个内部时钟周期,而内部时钟周期是由 250HZ 的频率所产生中的,也就是一个时钟滴答,间隔时间是 4 毫秒(ms).
也就是说:
1 个 jiffy=1 个内部时钟周期=250HZ=1 个时钟滴答=4 毫秒
每经过一个时钟滴答就会调用一次时钟中断处理程序，处理程序用 jiffy 来累计时钟滴答数,每发生一次时钟中断就增 1.
而每个中断之后,系统通过调度程序跟据时间片选择是否要进程继续运行,或让进程进入就绪状态.
最后需要说明的是每个操作系统的时钟滴答频率都是不一样的,LINUX 可以选择(100,250,1000)HZ,而 DOS 的频率是 55HZ.

七)为应用程序计时

用 time 程序可以监视任何命令或脚本占用 CPU 的情况.
1)bash 内置命令 time
例如:

```
time sleep 1
real    0m1.016s
user    0m0.000s
sys     0m0.004s
```

2)/usr/bin/time 的一般命令行
例如:

```
\time sleep 1
0.00user 0.00system 0:01.01elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (1major+176minor)pagefaults 0swaps
```

注:
在命令前加上斜杠可以绕过内部命令.

/usr/bin/time 还可以加上-v 看到更具体的输出:
\time -v sleep 1

```
Command being timed: "sleep 1"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 0%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 0
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 178
Voluntary context switches: 2
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

这里的输出更多来源于结构体 rusage.
最后，我们看到 real time 大于 user time 和 sys time 的总和，这说明进程不是在系统调用中阻塞,就是得不到运行的机会.
而 sleep()的运用，也说明了这一点.
*/