Linux 内核实验报告

实验题目：实验三-内核的定时机制实验

学号：　200900301236　　　　　（辅修号：　）
日期：2012.4.28　　班级：09 软 1 姓名：王添枝
Email：tzwang2012@163.com

实验目的：练习怎样编写调用内核的时间测量功能为应用程序测量和精确定时。理解 linux 内核的定时机制及其数据结构以及怎样从用户空间去访问内核空间的时间数据。

硬件环境：
软件环境：ubuntu10.10
　　　　　linux 内核:2.6.35.13
　　　　　gcc:4.4.5

实验步骤：

一、实验原理：

在每个用户进程的主要执行动作前设定时间，主要动作完成后获取时间，如此得到了主要动作的执行时间，系统总共有三种类型的时间 ITIMER_REAL,ITIMER_VIRTUAL,ITIMER_PROF. 三种类型的数据分别记录了程序执行的实际时间、程序执行的虚拟时间、程序执行的 CPU 时间。

二、程序设计

1. 问题 A

设置一个 ITIMER_REAL 型定时器，每秒产生一个信号，信号处理函数输出计数器的值，并将计数器的值加 1。

2. 问题 B

设置一个 ITIMER_REAL 型定时器，每一微秒产生一个信号，信号处理函数从 gettimeofday()得到当前时间并输出。

3. 问题 C

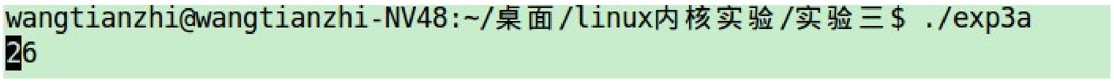
父进程创建两个子进程，在执行程序时带有三个参数，分别表示 fibonacci 的第几项，子进程一计算第一个参数的，子进程计算第二个参数的，父进程计算第三个参数的。每个进程都有 ITIMER_REAL,ITIMER_VIRTUAL,ITIMER_PROF 三种定时器。

4. 问题 D

设置 ITIMER_REAL,ITIMER_VIRTUAL,ITIMER_PROF 三种定时器各一个，分别用来报告每 1.25 秒时间内发生了多少次中断。信号处理函数从/proc/stat 中读出中断次数并计算 1.25 秒内产生的中断次数，然后输出。

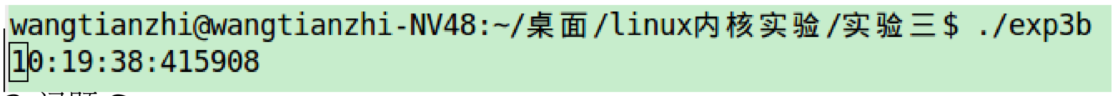
三、调试记录

1. 问题 A



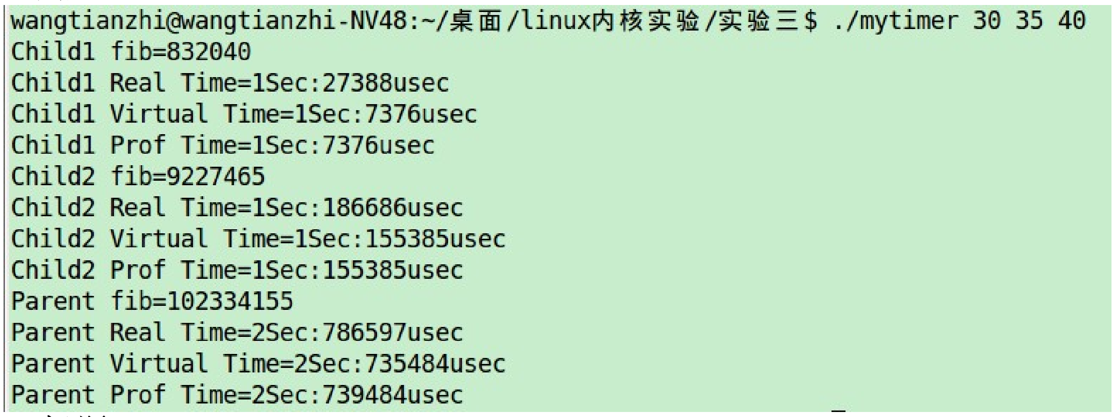
```
wangtianzhi@wangtianzhi-NV48:~/桌面/linux内核实验/实验三$ ./exp3a
26
```

2.问题 B



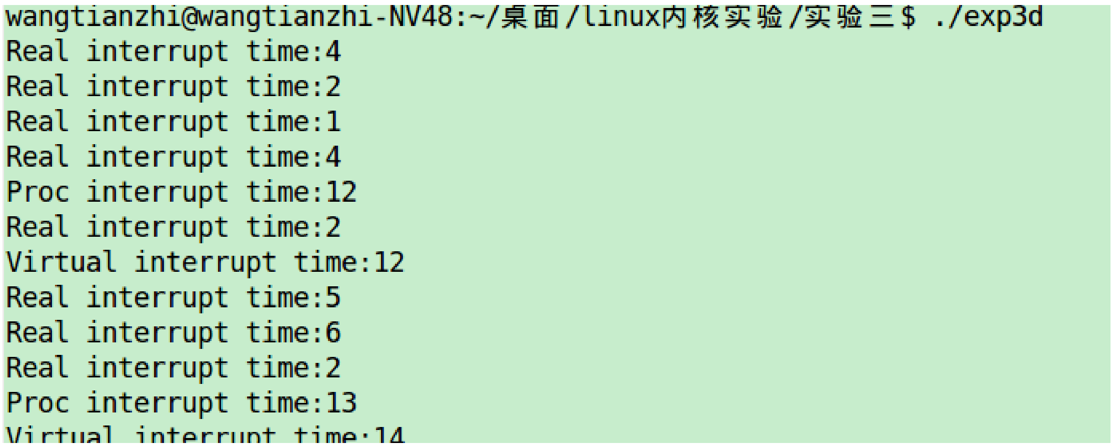
```
wangtianzhi@wangtianzhi-NV48:~/桌面/linux内核实验/实验三$ ./exp3b
10:19:38:415908
```

3.问题 C



```
wangtianzhi@wangtianzhi-NV48:~/桌面/linux内核实验/实验三$ ./mytimer 30 35 40
Child1 fib=832040
Child1 Real Time=1Sec:27388usec
Child1 Virtual Time=1Sec:7376usec
Child1 Prof Time=1Sec:7376usec
Child2 fib=9227465
Child2 Real Time=1Sec:186686usec
Child2 Virtual Time=1Sec:155385usec
Child2 Prof Time=1Sec:155385usec
Parent fib=102334155
Parent Real Time=2Sec:786597usec
Parent Virtual Time=2Sec:735484usec
Parent Prof Time=2Sec:739484usec
```

4. 问题 D



```
wangtianzhi@wangtianzhi-NV48:~/桌面/linux内核实验/实验三$ ./exp3d
Real interrupt time:4
Real interrupt time:2
Real interrupt time:1
Real interrupt time:4
Proc interrupt time:12
Real interrupt time:2
Virtual interrupt time:12
Real interrupt time:5
Real interrupt time:6
Real interrupt time:2
Proc interrupt time:13
Virtual interrupt time:14
```

四、结论分析与体会

本次实验并不是很难，重复性工作较多。通过本次实验，对内核的定时机制有了一定的了解，知道如何从用户空间去访问内核空间的时间数据。

程序完整源码

1. 问题 A

```
/*************************************************************
copyright :(C) 2012 by wangtianzhi
Function  : 内核的定时机制实验 - 问题 A
              使用 ITIMER_REAL 型定时器实现一个 gettimeofday(),将它设置为每秒产生一个信号,
并计算已经经过的秒数。
*************************************************************/
#include <sys/time.h>
#include <stdio.h>
#include <signal.h>
```

```c
static void sighandle(int);
static int second = 0;

int main(){
    struct itimerval v;
    signal(SIGALRM,sighandle);
    v.it_interval.tv_sec = 1;
    v.it_interval.tv_usec = 0;
    v.it_value.tv_sec = 1;
    v.it_value.tv_usec = 0;
    setitimer(ITIMER_REAL, &v, NULL);
    for(;;);
}
static void sighandle(int s){
    second++;
    printf("%d\r",second);
    fflush(stdout);
}
```

2. 问题 B

```c
/************************************************************
File      : exp3b.c
copyright :(C) 2012 by wangtianzhi
Function  : 内核的定时机制实验 - 问题 B
            使用问题 A 实现的 gettimeofday()实现一个精确到微秒级的“壁钟”
************************************************************/
#include <sys/time.h>
#include <stdio.h>
#include <signal.h>

static void sighandle(int);
//static int second = 0;
struct timeval now;
int today;
int main(){
    struct itimerval v;
    signal(SIGALRM,sighandle);
    v.it_interval.tv_sec = 0;
    v.it_interval.tv_usec = 1;
    v.it_value.tv_sec = 0;
    v.it_value.tv_usec = 1;
    setitimer(ITIMER_REAL, &v, NULL);
    for(;;);
}
```

```
static void sighandle(int s){
    //second++;
    gettimeofday(&now, NULL);
    today = now.tv_sec%(3600*24);
    //printf("%d\r",today);
        printf("%02d:%02d:%02d:%ld\r",today/3600,(today%3600)/60,(today
%3600)%60,now.tv_usec);
//  printf("%s\r",ctime(&(now.tv_sec)));
   //printf("%li\r",now.tv_sec);
    //printf("%li\r",now.tv_usec);
    //printf("%d\r",second);
    fflush(stdout);
}
```

3.问题 C

```
/
************************************************************
*****
                  mytimer.c  -  description
                                ------------------
copyright           : (C) 2012 by wangtianzhi
Function            : 测试并发进程执行中的各种时间。
                      给定 3 个菲波纳奇项数值可选在 36-45 之间
************************************************************
******/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>

static void psig_real(void);//父进程的 3 个定时中断处理函数原
型
static void psig_virtual(void);
static void psig_prof(void);

static void c1sig_real(void);//子进程 1 的 3 个定时中断处理函
数原型
static void c1sig_virtual(void);
static void c1sig_prof(void);

static void c2sig_real(void);//子进程 2 的 3 个定时中断处理函
数原型
static void c2sig_virtual(void);
static void c2sig_prof(void);
```

```
long unsigned int fibonacci(unsigned int n);

//记录 3 种定时的秒数的变量
static long p_real_secs=0,c1_real_secs=0,c2_real_secs=0;
static                                                    long
p_virtual_secs=0,c1_virtual_secs=0,c2_virtual_secs=0;
static long p_prof_secs=0,c1_prof_secs=0,c2_prof_secs=0;

//记录 3 种定时的毫秒秒数的结构变量
static struct itimerval p_realt,c1_realt,c2_realt;
static struct itimerval p_virtt,c1_virtt,c2_virtt;
static struct itimerval p_proft,c1_proft,c2_proft;

int main(int argc,char **argv)
{
   long unsigned fib=0;
   int pid1,pid2;
   unsigned int fibarg;
   int status;
   int i;
   if(argc < 4) {
      printf("Usage: testsig arg1 arg2 arg3\n");
      return 1;
   }

   //父进程设置 3 种定时处理入口
   signal(SIGALRM,psig_real);
   signal(SIGVTALRM,psig_virtual);
   signal(SIGPROF,psig_prof);

   //初始化父进程 3 种时间定时器
   p_realt.it_interval.tv_sec = 9;
   p_realt.it_interval.tv_usec = 999999;
   p_realt.it_value.tv_sec = 9;
   p_realt.it_value.tv_usec = 999999;
   setitimer(ITIMER_REAL,&p_realt,NULL);


   p_virtt.it_interval.tv_sec = 9;
   p_virtt.it_interval.tv_usec = 999999;
   p_virtt.it_value.tv_sec = 9;
   p_virtt.it_value.tv_usec = 999999;
   setitimer(ITIMER_VIRTUAL,&p_virtt,NULL);
   p_proft.it_interval.tv_sec = 9;
   p_proft.it_interval.tv_usec = 999999;
   p_proft.it_value.tv_sec = 9;
   p_proft.it_value.tv_usec = 999999;
   setitimer(ITIMER_PROF,&p_proft,NULL);

   pid1 = fork();
   if(pid1==0){
      //子进程 1 设置 3 种定时处理入口
      signal(SIGALRM,c1sig_real);
      signal(SIGVTALRM,c1sig_virtual);
       signal(SIGPROF,c1sig_prof);

       //初始化子进程 1 的 3 种时间定时器
       c1_realt.it_interval.tv_sec = 9;
       c1_realt.it_interval.tv_usec = 999999;

       c1_realt.it_value.tv_sec = 9;
       c1_realt.it_value.tv_usec = 999999;
       setitimer(ITIMER_REAL,&c1_realt,NULL);

       c1_virtt.it_interval.tv_sec = 9;
       c1_virtt.it_interval.tv_usec = 999999;

       c1_virtt.it_value.tv_sec = 9;
       c1_virtt.it_value.tv_usec = 999999;
       setitimer(ITIMER_VIRTUAL,&c1_virtt,NULL);

       c1_proft.it_interval.tv_sec = 9;
       c1_proft.it_interval.tv_usec = 999999;

       c1_proft.it_value.tv_sec = 9;
       c1_proft.it_value.tv_usec = 999999;
       setitimer(ITIMER_PROF,&c1_proft,NULL);

       //子进程 1 开始计算 fib
      fib = fibonacci(atoi(argv[1]));
      //打印子进程 1 所花费的 3 种时间值
      getitimer(ITIMER_REAL,&c1_realt);
        printf("Child1  fib=%ld\nChild1  Real  Time=%ldSec:
%ldusec\n",fib,c1_real_secs+                                10
-c1_realt.it_value.tv_sec,1000000                            -
c1_realt.it_value.tv_usec);
```

```
      getitimer(ITIMER_VIRTUAL,&c1_virtt);
                 printf("Child1     Virtual     Time=%ldSec:
%ldusec\n",c1_virtual_secs+                               10
-c1_virtt.it_value.tv_sec,1000000                          -
c1_virtt.it_value.tv_usec);
      getitimer(ITIMER_PROF,&c1_proft);
                   printf("Child1      Prof      Time=%ldSec:
%ldusec\n",c1_prof_secs+                                  10
-c1_proft.it_value.tv_sec,1000000                          -
c1_proft.it_value.tv_usec);
   }
   else if((pid2=fork()) == 0){
      //子进程 2 设置 3 种定时中段入口
         signal(SIGALRM,c2sig_real);
      signal(SIGVTALRM,c2sig_virtual);
       signal(SIGPROF,c2sig_prof);

       //初始化子进程 2 的 3 种时间定时器
       c2_realt.it_interval.tv_sec = 9;
       c2_realt.it_interval.tv_usec = 999999;

       c2_realt.it_value.tv_sec = 9;
       c2_realt.it_value.tv_usec = 999999;
       setitimer(ITIMER_REAL,&c2_realt,NULL);

       c2_virtt.it_interval.tv_sec = 9;
       c2_virtt.it_interval.tv_usec = 999999;

       c2_virtt.it_value.tv_sec = 9;
       c2_virtt.it_value.tv_usec = 999999;
       setitimer(ITIMER_VIRTUAL,&c2_virtt,NULL);

       c2_proft.it_interval.tv_sec = 9;
       c2_proft.it_interval.tv_usec = 999999;

       c2_proft.it_value.tv_sec = 9;
       c2_proft.it_value.tv_usec = 999999;
       setitimer(ITIMER_PROF,&c2_proft,NULL);

       //子进程 2 开始计算 fib
      fib = fibonacci(atoi(argv[2]));
      //打印子进程 2 所花费的 3 种时间值
      getitimer(ITIMER_REAL,&c2_realt);
         printf("Child2  fib=%ld\nChild2  Real  Time=%ldSec:
```

```
%ldusec\n",fib,c2_real_secs+                          10
-c2_realt.it_value.tv_sec,1000000                          -
c2_realt.it_value.tv_usec);
      getitimer(ITIMER_VIRTUAL,&c2_virtt);
                 printf("Child2     Virtual     Time=%ldSec:
%ldusec\n",c2_virtual_secs+                          10
-c2_virtt.it_value.tv_sec,1000000                          -
c2_virtt.it_value.tv_usec);
      getitimer(ITIMER_PROF,&c2_proft);
                   printf("Child2      Prof      Time=%ldSec:
%ldusec\n",c2_prof_secs+                          10
-c2_proft.it_value.tv_sec,1000000                          -
c2_proft.it_value.tv_usec);
   }
   else  {
      //父进程开始计算fib
       fib = fibonacci(atoi(argv[3]));
      //打印父进程所花费的 3种时间值
      getitimer(ITIMER_REAL,&p_realt);
         printf("Parent  fib=%ld\nParent  Real  Time=%ldSec:
%ldusec\n",fib,p_real_secs+                          10
-p_realt.it_value.tv_sec,1000000                          -
p_realt.it_value.tv_usec);
      getitimer(ITIMER_VIRTUAL,&p_virtt);
                 printf("Parent     Virtual     Time=%ldSec:
%ldusec\n",p_virtual_secs+                          10
-p_virtt.it_value.tv_sec,1000000                          -
p_virtt.it_value.tv_usec);
      getitimer(ITIMER_PROF,&p_proft);
                   printf("Parent      Prof      Time=%ldSec:
%ldusec\n",p_prof_secs+ 10 -p_proft.it_value.tv_sec,1000000
- p_proft.it_value.tv_usec);

       //等待子进程结束
      //第一个参数：
      //          pid<-1 等待进程组识别码为 pid 绝对值的任何
子进程。
      //        pid=-1 等待任何子进程,相当于 wait()。
      //        pid=0 等待进程组识别码与目前进程相同的任何子进
程。
      //        pid>0 等待任何子进程识别码为 pid 的子进程。
       //第二个参数存放子进程的状态结束符
      waitpid(0,&status,0);
      waitpid(0,&status,0);
   }
}

//父进程的 3 个定时中断处理函数
static void psig_real(){
   p_real_secs += 10;
   //p_realt.it_interval.tv_sec = 9;
   //p_realt.it_interval.tv_usec = 999999;
   //p_realt.it_value.tv_sec = 9;
   //p_realt.it_value.tv_usec = 999999;
   //setitimer(ITIMER_REAL,&p_realt,NULL);
}
static void psig_virtual(){
   p_virtual_secs += 10;
}
static void psig_prof(){
   p_prof_secs += 10;
}

//子进程 1 的 3 个定时中断处理函数
static void c1sig_real(){
   c1_real_secs += 10;
}
static void c1sig_virtual(){
   c1_virtual_secs += 10;
}
static void c1sig_prof(){
   c1_prof_secs += 10;
}

//子进程 2 的 3 个定时中断处理函数原型
static void c2sig_real(){
   c2_real_secs += 10;
}
static void c2sig_virtual(){
   c2_virtual_secs += 10;
}
static void c2sig_prof(){
   c2_prof_secs += 10;
}

//fib 的递归计算函数
long unsigned int fibonacci(unsigned int n)
{
   if(n == 0)return 0;
   else if(n == 1 || n == 2)return 1;
   else return fibonacci(n-1)+fibonacci(n-2);
}
```

4.问题 D

```
/
***********************************************************
*****
File:exp3d.c
Copyright : (C)2012 by wangtianzhi
Function  : 编写一个定时报告中断次数的程序，它可以连续报告三种定时
         类型每 1.25 毫秒的时间间隔内，系统发生了多少次中断。
  ***********************************************************
*******/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>

//中断处理函数
static void sig_real(void);
static void sig_virtual(void);
static void sig_prof(void);

unsigned long real_stime,virtual_stime,prof_stime;//开始时中断数
unsigned long real_otime,virtual_otime,prof_otime;//结束时中断数

//记录 3 种定时的毫秒秒数的结构变量
static struct itimerval realt,virtt,proft;
FILE *thisProcFile1,*thisProcFile2,*thisProcFile3;
char str_real[502],str_virtual[502],str_prof[502];
char sreal[20],svirtual[20],sprof[20];
int main(){
   thisProcFile1 = fopen("/proc/stat","r");
   while(fgets(str_real,502,thisProcFile1)!=NULL){
      sscanf(str_real,"%s%ld",sreal,&real_stime);
      if(strcmp(sreal,"intr")==0){
         close(thisProcFile1);
      // rewind(thisProcFile1);//将内部文件指针移到文件开头
         break;
      }
   }
// close(thisProcFile);
   virtual_stime=real_stime;
   prof_stime=real_stime;
   //设置 3 种定时处理入口
   signal(SIGALRM,sig_real);
   signal(SIGVTALRM,sig_virtual);
    signal(SIGPROF,sig_prof);

   //初始化 3 种时间定时器
   realt.it_interval.tv_sec = 0;
   realt.it_interval.tv_usec = 1250;
    realt.it_value.tv_sec = 0;
   realt.it_value.tv_usec = 1250;

   virtt.it_interval.tv_sec = 0;
   virtt.it_interval.tv_usec = 1250;
   virtt.it_value.tv_sec = 0;
    virtt.it_value.tv_usec = 1250;

   proft.it_interval.tv_sec = 0;
   proft.it_interval.tv_usec = 1250;
   proft.it_value.tv_sec = 0;
    proft.it_value.tv_usec = 1250;

   setitimer(ITIMER_REAL,&realt,NULL);
   setitimer(ITIMER_VIRTUAL,&virtt,NULL);
   setitimer(ITIMER_PROF,&proft,NULL);
   for(;;);
   return 0;
}

static void sig_real(){
   thisProcFile1 = fopen("/proc/stat","r");
   while(fgets(str_real,502,thisProcFile1)!=NULL){
      sscanf(str_real,"%s%ld",sreal,&real_otime);
      if(strcmp(sreal,"intr")==0){
           close(thisProcFile1);
           //rewind(thisProcFile1);
         break;
      }
   }
// printf("real_otime%ld\n",real_otime);
```

```
   printf("Real      interrupt      time:%ld\n",real_otime-
real_stime);
   real_stime = real_otime;

}
static void sig_virtual(){
   thisProcFile2 = fopen("/proc/stat","r");
   while(fgets(str_virtual,502,thisProcFile2)!=NULL){
      sscanf(str_virtual,"%s%ld",svirtual,&virtual_otime);
      if(strcmp(svirtual,"intr")==0){
          // rewind(thisProcFile2);
         close(thisProcFile2);
         break;
      }
   }
// close(thisProcFile);
// printf("virtual_otime:%ld\n",virtual_otime);
   printf("Virtual   interrupt   time:%ld\n",virtual_otime-
virtual_stime);
   virtual_stime = virtual_otime;
}
static void sig_prof(){
   thisProcFile3 = fopen("/proc/stat","r");
   while(fgets(str_prof,502,thisProcFile3)!=NULL){
      sscanf(str_prof,"%s%ld",sprof,&prof_otime);
      if(strcmp(sprof,"intr")==0){
            close(thisProcFile3);
         // rewind(thisProcFile3);
         break;
      }
   }
// close(thisProcFile);
// printf("prof_otime:%ld\n",prof_otime);
   printf("Proc    interrupt    time:%ld\n",prof_otime    -
prof_stime);
   prof_stime = prof_otime;
}
```

参考材料

linux 操作系统内核实习
linux 内核设计与实现