实验五 内核的定时机制实验

## 实验目的

本实验是练习**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**。通过这个实验我们可以进一步理解 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。

## 实验内容

A 在用户态编写一个程序，该程序设定一个定时器，在时间到期的时候做出某种可观察的响应（方法不限）。

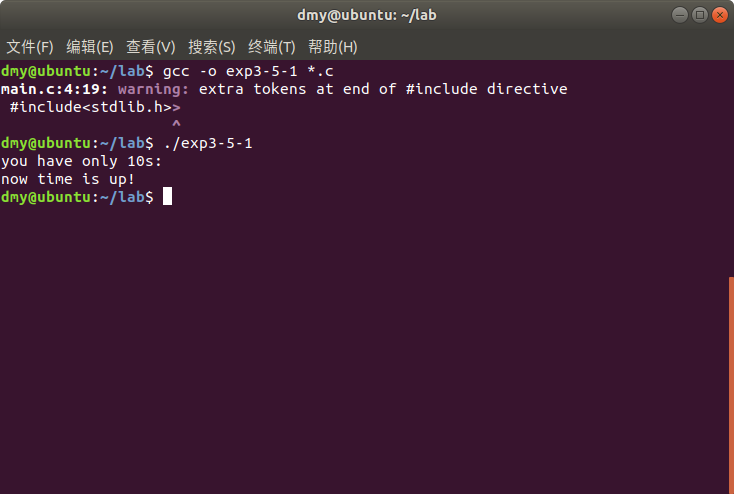
B 用定时器 ITIMER\_REAL实现gettimeofday的功能。使其一秒钟产生一个信号，计算已经过的秒数。

设计思路：设置定时器 ITIMER\_REAL 间隔为一秒钟。并为计时到时设定信号处理程序，即singal(SIGALRM,)，使其输出当前所记时间。

**C** 记录一个进程运行时所占用的real time, cpu time, user time, kernel time。

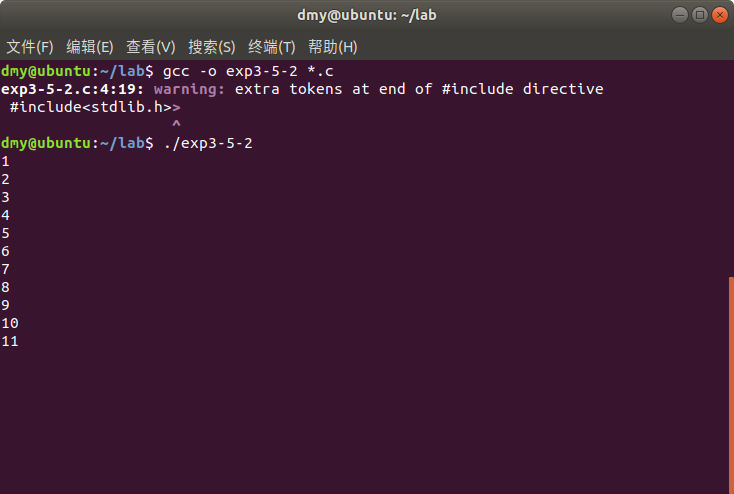
## 实验步骤

1. 定时器及其相应



|  |
| --- |
| #include<stdio.h>  #include<time.h>  #include<sys/time.h>  #include<stdlib.h>>  #include<signal.h>  int num = 0;  void rwaltimer\_set()  {  struct itimerval itv, oldtv;  itv.it\_interval.tv\_sec = 1;  itv.it\_interval.tv\_usec = 0;  itv.it\_value.tv\_sec = 1;  itv.it\_value.tv\_usec = 0;  setitimer(ITIMER\_REAL,&itv, &oldtv);  }  void sigalrm\_handler(int sig)  {  num++;  }  int main(int argc, char\* argv[])  {  signal(SIGALRM, sigalrm\_handler);  rwaltimer\_set();  puts("you have only 10s:");  while (num < 10);  puts("now time is up!");  return 0;  } |

1. 计算经过的秒数，根据第一个程序进行修改即可



|  |
| --- |
| #include<stdio.h>  #include<time.h>  #include<sys/time.h>  #include<stdlib.h>>  #include<signal.h>  int num = 0;  void rwaltimer\_set()  {  struct itimerval itv, oldtv;  itv.it\_interval.tv\_sec = 1;  itv.it\_interval.tv\_usec = 0;  itv.it\_value.tv\_sec = 1;  itv.it\_value.tv\_usec = 0;  setitimer(ITIMER\_REAL,&itv, &oldtv);  }  void sigalrm\_handler(int sig)  {  fflush(stdout);  printf("%d\n", ++num);  fflush(stdout);  }  int main(int argc, char\* argv[])  {  signal(SIGALRM, sigalrm\_handler);  rwaltimer\_set();  for (;;);  return 0;  } |

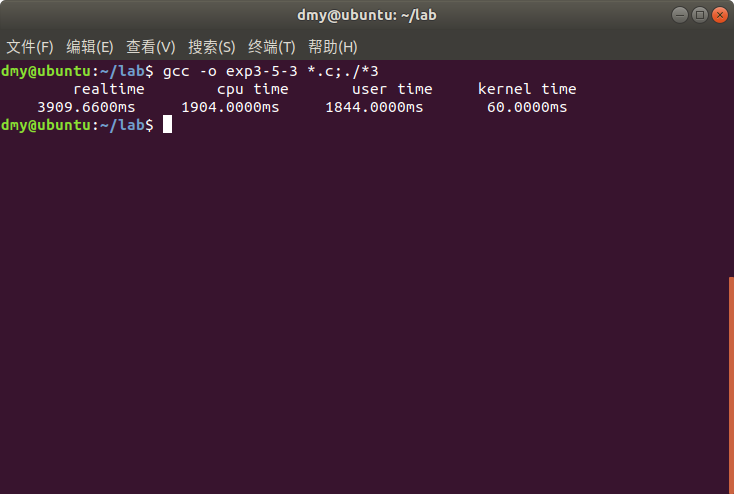
1. 统计运行时间

这里用分解质因数模拟CPU的使用

用fork模拟进入内核的调用

Sleep模拟IO阻塞

用原时间减去运行后定时器剩余时间得到总耗时



|  |
| --- |
| #include<stdio.h>  #include<time.h>  #include<sys/time.h>  #include<stdlib.h>  #include<signal.h>  #include<unistd.h>  static int realtime, virttime, proctime;  //0 ITIMER\_REAL 1 ITIMER\_VIRTUAL 2 ITIMER\_PROF  int tottime = 100;  void rwaltimer\_set()  {  struct itimerval itv, oldtv;  itv.it\_interval.tv\_sec = tottime;  itv.it\_interval.tv\_usec = 0;  itv.it\_value.tv\_sec = tottime;  itv.it\_value.tv\_usec = 0;  setitimer(ITIMER\_REAL, &itv, &oldtv);  setitimer(ITIMER\_VIRTUAL, &itv, &oldtv);  setitimer(ITIMER\_PROF, &itv, &oldtv);  }  void sigalrm\_handler(int sig)  {  fflush(stdout);  switch (sig)  {  case SIGALRM:realtime++; break;  case SIGVTALRM:virttime++; break;  case SIGPROF:proctime++; break;  default:  break;  }  fflush(stdout);  }  int main(int argc, char\* argv[])  {  int pid = 0;  signal(SIGALRM, sigalrm\_handler);  signal(SIGVTALRM, sigalrm\_handler);  signal(SIGPROF, sigalrm\_handler);  rwaltimer\_set();  for (int i = 2; i % (2 << 30 - 1); i++);  for (int i = 0; i < 1000; i++)  if (!fork())  exit(0);  sleep(2);  printf("%16s%16s%16s%16s\n","realtime","cpu time","user time","kernel time");  struct itimerval t1, t2, t3;  getitimer(ITIMER\_REAL, &t1);  getitimer(ITIMER\_VIRTUAL, &t2);  getitimer(ITIMER\_PROF, &t3);  double ans1 = t1.it\_value.tv\_sec \* 1000 + t1.it\_value.tv\_usec / 1000.0;  ans1 = tottime \* 1000 - ans1;  double ans2 = t2.it\_value.tv\_sec \* 1000 + t2.it\_value.tv\_usec / 1000.0;  ans2 = tottime \* 1000 - ans2;  double ans3 = t3.it\_value.tv\_sec \* 1000 + t3.it\_value.tv\_usec / 1000.0;  ans3 = tottime \* 1000 - ans3;  printf("%13.4fms %13.4fms %13.4fms %13.4fms\n",ans1,ans3,ans2,ans3-ans2);  return 0;  } |