**实验2.2 内核定时器 实验报告**

1. **实验目的**

学习掌握内核定时器的实现原理和方法，建立一种用户空间机制来测量多线程程序的执行时间。

1. **实验内容**
2. 实现一个定时器，一秒钟产生一个信号，计算已经经过的秒数。
3. 记录一个进程运行时所占用的real time、cpu time、user time、kernel time。
4. 编写一个主程序产生两个子进程，分别递归计算N = 20/30/36的斐波那契数列，分别对三个进程计算相应的real time、cpu time、user time、kernel time。
5. **实验设计原理**
6. 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。
7. 任务开始前设置好定时器**ITIMER\_REAL，ITIMER\_VIRTUAL，ITIMER\_PROF**,即其相应的信号处理程序。在任务执行过程中内核定时器通过产生等间隔的信号来记录进程所需的各种时间参量，并在任务结束后打印出来。
8. 与B原理基本相同，不同的只是在任务开始前要分别设定好每个进程的定时器，而且其最终的实验结果也由相应进程自身打印出来。
9. **实验步骤**
10. **代码清单**

/\*\*\*\*part1.c\*\*\*\*\*/

//part 1

#include <sys/time.h>

#include <stdio.h>

#include <signal.h>

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("The counter is :%d\n",second);

//fflush(stdout);

}

1. **代码清单**

/\*\*\*\*\*\*\*\*\*\*\*\*part2.c\*\*\*\*\*\*\*\*\*\*/

//part2

#include <sys/time.h>

#include <stdio.h>

#include <signal.h>

static void sighandle(int);

static long realsecond = 0;

static long vtsecond = 0;

static long profsecond = 0;

static struct itimerval realt,virtt,proft;

int main(){

struct itimerval v;

int i,j;

long moresec,moremsec,t1,t2;

signal(SIGALRM,sighandle);

signal(SIGVTALRM,sighandle);

signal(SIGPROF,sighandle);

v.it\_interval.tv\_sec = 10;

v.it\_interval.tv\_usec = 0;

v.it\_value.tv\_sec = 10;

v.it\_value.tv\_usec = 0;

setitimer(ITIMER\_REAL,&v,NULL);

setitimer(ITIMER\_VIRTUAL,&v,NULL);

setitimer(ITIMER\_PROF,&v,NULL);

for(j= 0;j<1000;j++){

for(i= 0;i<500;i++){printf("\*\*\*\*\*\*\*\*\r");fflush(stdout);}

}

getitimer(ITIMER\_PROF,&proft);

getitimer(ITIMER\_REAL,&realt);

getitimer(ITIMER\_VIRTUAL,&virtt);

printf("\n");

moresec = 10 - realt.it\_value.tv\_sec;

moremsec = (1000000 - realt.it\_value.tv\_usec)/1000;

printf("realtime = %ld sec, %ld msec\n",realsecond+moresec,moremsec);

moresec = 10 - proft.it\_value.tv\_sec;

moremsec = (1000000 - proft.it\_value.tv\_usec)/1000;

printf("cputime = %ld sec, %ld msec\n",profsecond+moresec,moremsec);

moresec = 10 - virtt.it\_value.tv\_sec;

moremsec = (1000000 - virtt.it\_value.tv\_usec)/1000;

printf("usertime = %ld sec, %ld msec\n",vtsecond+moresec,moremsec);

t1 = (10 - proft.it\_value.tv\_sec)\*1000 + (1000000 - proft.it\_value.tv\_usec)/1000 + profsecond\*10000;

t2 = (10 - virtt.it\_value.tv\_sec)\*1000 + (1000000 - virtt.it\_value.tv\_usec)/1000 + vtsecond\*10000;

moresec = (t1 - t2)/1000;

moremsec = (t1 - t2) % 1000;

printf("kerneltime = %ld sec, %ld msec\n",moresec,moremsec);

fflush(stdout);

}

static void sighandle(int s)

{

switch(s){

case SIGALRM:realsecond+=10;break;

case SIGVTALRM:vtsecond+=10;break;

case SIGPROF:profsecond+=10;break;

default :break;

}

}

1. **代码清单**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*part3.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//part3

#include <sys/time.h>

#include <stdio.h>

#include <signal.h>

#include <unistd.h>

static void c1\_sighandle(int s);

static void c2\_sighandle(int s);

static void p\_sighandle(int s);

static long p\_realt\_secs = 0,c1\_realt\_secs = 0,c2\_realt\_secs = 0;

static long p\_virtt\_secs = 0,c1\_virtt\_secs = 0,c2\_virtt\_secs = 0;

static long p\_proft\_secs = 0,c1\_proft\_secs = 0,c2\_proft\_secs = 0;

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;

static struct itimerval ini\_value;

int main(){

int fib = 0;

int pid1,pid2;

int status;

long moresec,moremsec,t1,t2;

pid1 = fork();

if (pid1 == 0){//c1

//set c1 signal handle

signal(SIGALRM,c1\_sighandle);

signal(SIGVTALRM,c1\_sighandle);

signal(SIGPROF,c1\_sighandle);

ini\_value.it\_interval.tv\_sec = 10;

ini\_value.it\_interval.tv\_usec = 0;

ini\_value.it\_value.tv\_sec = 10;

ini\_value.it\_value.tv\_usec = 0;

//set c1 timer

setitimer(ITIMER\_REAL,&ini\_value,NULL);

setitimer(ITIMER\_VIRTUAL,&ini\_value,NULL);

setitimer(ITIMER\_PROF,&ini\_value,NULL);

fib = fibonacci(20);

//get timer of c1 and print

getitimer(ITIMER\_REAL,&c1\_realt);

getitimer(ITIMER\_VIRTUAL,&c1\_virtt);

getitimer(ITIMER\_PROF,&c1\_proft);

printf("\n");

moresec = 10 - c1\_realt.it\_value.tv\_sec;

moremsec = (1000000 - c1\_realt.it\_value.tv\_usec)/1000;

printf("c1fib(20)=%ld\nrealtime=%ldsec,%ldmsec\n",fib,c1\_realt\_secs+moresec,moremsec);

moresec = 10 - c1\_proft.it\_value.tv\_sec;

moremsec = (1000000 - c1\_proft.it\_value.tv\_usec)/1000;

printf("cputime = %ld sec, %ld msec\n",c1\_proft\_secs+moresec,moremsec);

moresec = 10 - c1\_virtt.it\_value.tv\_sec;

moremsec = (1000000 - c1\_virtt.it\_value.tv\_usec)/1000;

printf("usertime = %ld sec, %ld msec\n",c1\_virtt\_secs+moresec,moremsec);

t1=(10-c1\_proft.it\_value.tv\_sec)\*1000+(1000000-c1\_proft.it\_value.tv\_usec)/1000 + c1\_proft\_secs\*10000;

t2=(10-c1\_virtt.it\_value.tv\_sec)\*1000+(1000000-c1\_virtt.it\_value.tv\_usec)/1000 + c1\_virtt\_secs\*10000;

moresec = (t1 - t2)/1000;

moremsec = (t1 - t2) % 1000;

printf("kerneltime = %ld sec, %ld msec\n",moresec,moremsec);

fflush(stdout);

exit(0);

}//end c1

else{

pid2 = fork();

if (pid2 == 0){//c2

//set c2 signal handle

signal(SIGALRM,c2\_sighandle);

signal(SIGVTALRM,c2\_sighandle);

signal(SIGPROF,c2\_sighandle);

ini\_value.it\_interval.tv\_sec = 10;

ini\_value.it\_interval.tv\_usec = 0;

ini\_value.it\_value.tv\_sec =10;

ini\_value.it\_value.tv\_usec = 0;

//set c2 timer

setitimer(ITIMER\_REAL,&ini\_value,NULL);

setitimer(ITIMER\_VIRTUAL,&ini\_value,NULL);

setitimer(ITIMER\_PROF,&ini\_value,NULL);

fib = fibonacci(30);

//get timer of c2 and print

getitimer(ITIMER\_PROF,&c2\_proft);

getitimer(ITIMER\_REAL,&c2\_realt);

getitimer(ITIMER\_VIRTUAL,&c2\_virtt);

printf("\n");

moresec = 10 - c2\_realt.it\_value.tv\_sec;

moremsec = (1000000 - c2\_realt.it\_value.tv\_usec)/1000;

printf("c2fib(30)=%ld\nrealtime=%ldsec,%ldmsec\n",fib,c2\_realt\_secs+moresec,moremsec);

moresec = 10 - c2\_proft.it\_value.tv\_sec;

moremsec = (1000000 - c2\_proft.it\_value.tv\_usec)/1000;

printf("cputime=%ldsec,%ldmsec\n",c2\_proft\_secs+moresec,moremsec);

moresec = 10 - c2\_virtt.it\_value.tv\_sec;

moremsec = (1000000 - c2\_virtt.it\_value.tv\_usec)/1000;

printf("usertime=%ldsec,%ldmsec\n",c2\_virtt\_secs+moresec,moremsec);

t1=(10-c2\_proft.it\_value.tv\_sec)\*1000+(1000000-c2\_proft.it\_value.tv\_usec)/1000 + c2\_proft\_secs\*10000;

t2=(10-c2\_virtt.it\_value.tv\_sec)\*1000+(1000000-c2\_virtt.it\_value.tv\_usec)/1000 + c2\_virtt\_secs\*10000;

moresec = (t1 - t2)/1000;

moremsec = (t1 - t2) % 1000;

printf("kerneltime = %ld sec, %ld msec\n",moresec,moremsec);

fflush(stdout);

exit(0);

}//endc2

}

//parent

//setparent signal handle

signal(SIGALRM,p\_sighandle);

signal(SIGVTALRM,p\_sighandle);

signal(SIGPROF,p\_sighandle);

ini\_value.it\_interval.tv\_sec = 10;

ini\_value.it\_interval.tv\_usec = 0;

ini\_value.it\_value.tv\_sec = 10;

ini\_value.it\_value.tv\_usec = 0;

//set parent timer

setitimer(ITIMER\_REAL,&ini\_value,NULL);

setitimer(ITIMER\_VIRTUAL,&ini\_value,NULL);

setitimer(ITIMER\_PROF,&ini\_value,NULL);

fib = fibonacci(36);

getitimer(ITIMER\_PROF,&p\_proft);

getitimer(ITIMER\_REAL,&p\_realt);

getitimer(ITIMER\_VIRTUAL,&p\_virtt);

printf("\n");

moresec = 10 - p\_realt.it\_value.tv\_sec;

moremsec = (1000000 - p\_realt.it\_value.tv\_usec)/1000;

printf("pfib(36)=%ld\nrealtime=%ldsec,%ldmsec\n",fib,p\_realt\_secs+moresec,moremsec);

moresec = 10 - p\_proft.it\_value.tv\_sec;

moremsec = (1000000 - p\_proft.it\_value.tv\_usec)/1000;

printf("cputime = %ld sec, %ld msec\n",p\_proft\_secs+moresec,moremsec);

moresec = 10 - p\_virtt.it\_value.tv\_sec;

moremsec = (1000000 - p\_virtt.it\_value.tv\_usec)/1000;

printf("usertime = %ld sec, %ld msec\n",p\_virtt\_secs+moresec,moremsec);

t1= (10 - p\_proft.it\_value.tv\_sec)\*1000 + (1000000 - p\_proft.it\_value.tv\_usec)/1000 + p\_proft\_secs\*10000;

t2 = (10 - p\_virtt.it\_value.tv\_sec)\*1000 + (1000000 - p\_virtt.it\_value.tv\_usec)/1000 + p\_virtt\_secs\*10000;

moresec = (t1 - t2)/1000;

moremsec = (t1 - t2) % 1000;

printf("kerneltime = %ld sec, %ld msec\n",moresec,moremsec);

fflush(stdout);

//wait c1,c2 terminal

wait(&status);

wait(&status);

return 0;

}//end main

int fibonacci(int n)

{

if( n == 0 ) return 0;

else if( n == 1 || n == 2) return 1;

else return(fibonacci(n-1)+fibonacci(n-2) );

}

static void c1\_sighandle(int s)

{

switch(s){

case SIGALRM:c1\_realt\_secs+=10;break;

case SIGVTALRM:c1\_virtt\_secs+=10;break;

case SIGPROF:c1\_proft\_secs+=10;break;

default :break;

}

}

static void c2\_sighandle(int s)

{

switch(s){

case SIGALRM:c2\_realt\_secs+=10;break;

case SIGVTALRM:c2\_virtt\_secs+=10;break;

case SIGPROF:c2\_proft\_secs+=10;break;

default :break;

}

}

static void p\_sighandle(int s)

{

switch(s){

case SIGALRM:p\_realt\_secs+=10;break;

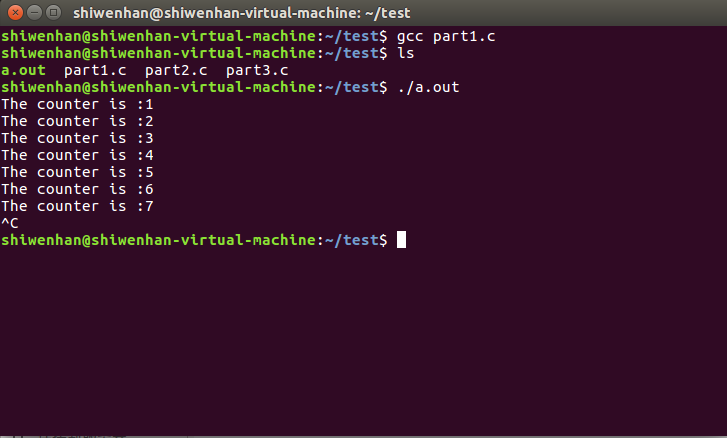
case SIGVTALRM:p\_virtt\_secs+=10;break;

case SIGPROF:p\_proft\_secs+=10;break;

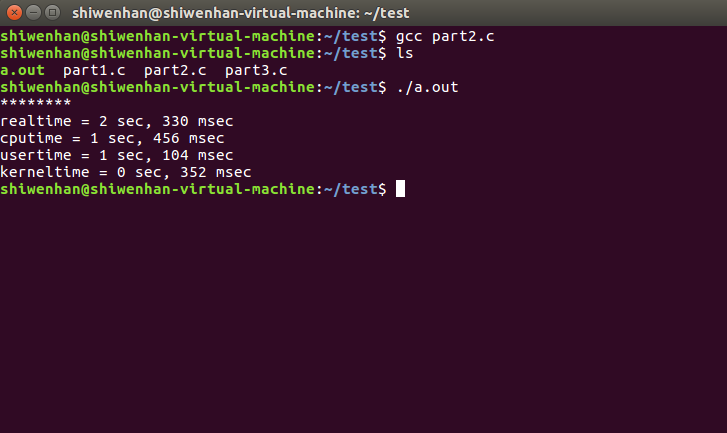
default :break;

}

}

1. **实验结果及分析**
2. **实验结果截图如下：**

其每1s输出一个定时信号，printf函数负责将这个结果打印至屏幕上。

1. **实验结果截图如下：**

可以看出运行本次进程所需要的各个时间为：

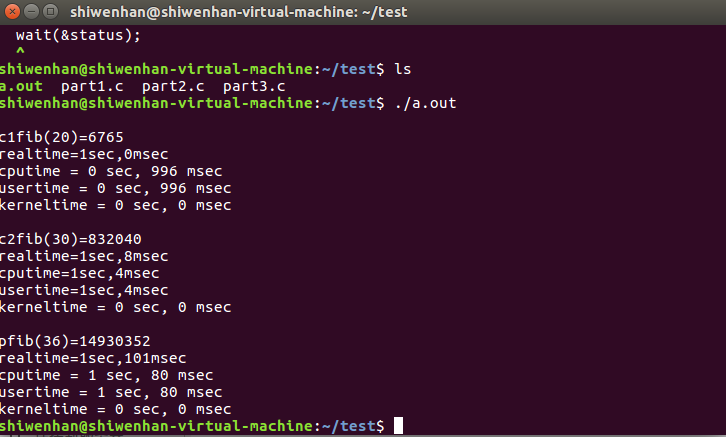
realtime（实际使用时间）：2.330ms

cputime（CPU占用时间）:1.456ms

usertime（在用户模式下的时间）：1.104s

kerneltime（在内核模式下的时间）：0.352s

可见用户态加上内核态占用的时间就是总的CPU占用时间。而realtime又是什么呢？可以理解为一个进程从开始到运行结束的时间，这个时间与运行环境有很大的关系，同时也包括了其他进程使用的时间片和进程阻塞时间（比如等待IO完成），类似于进程的生命周期。因此realtime要大于cputime也就不足为奇了。

1. **实验结果截图如下：**

可以看出三个进程所用的时间依次递增，这是很显然的因为计算数值越大的斐波那契数、需要的递归嵌套深度就越大。

1. **人员任务分配**

本次实验由小组成员

史文翰

No.2014211218

Cla.2014211304

独自完成。