统计一段时间内操作系统缺页次数

实验目的

学习虚拟内存的基本原理和 Linux 虚拟内存管理技术。
深入理解、掌握 Linux 的按需调页过程。

实验内容

统计从当前时刻起，一段时间内操作系统发生的缺页中断次数。

实验提示

一、原理

在 Linux 系统的/proc 文件系统中有一个记录系统当前基本状况的文件 stat。该文件中有一节是关于中断次数的。这一节中记录了从系统启动后到当前时刻发生的系统中断的总次数以及各类中断的分别发生的次数。这一节以关键字 intr 开头，紧接着的一项是系统发生中断的总次数，之后依次是 0 号中断发生的次数，1 号中断发生的次数……其中缺页中断是第 14 号中断，也就是在关键字 intr 之后的第 16 项。如图 1 是查看 stat 文件的终端显示结果，可以看到系统已经发生过 1313551 次缺页中断。

```
[jhr@linux /proc]$ less stat
cpu  2379645 0 2644679 27996304
disk 1029569 213590 0 0
disk_rio 556382 123698 0 0
disk_wio 473187 89892 0 0
disk_rblk 4449910 989524 0 0
disk_wblk 3785424 719136 0 0
page 2504416 1951299
swap 4818 3725
intr 68969288 33020628 2 0 0 0 0 3 0 1 0 0 34635100 0 1 1313551 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
ctxt 46494837
btime 1019754086
processes 205568
stat (END)
```

图 1　stat 文件内容

实验可以利用 stat 文件提供的数据在一段时间的开始时刻和结束时刻分别读取缺页中断发生的次数，然后作一个简单的减法操作，就可以得出这段时间内发生缺页中断的次数。由于 stat 文件的数据是由系统动态更新的，过去时刻的数据是无法采集到的，所以这里的开始时刻最早也只能是当前时刻，实验中采用的统计时间段就是从当前时刻开始的一段时间。

二、示例

编写程序文件 pfintr.c

```
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FILENAME "/proc/stat"     /*指定文件操作的对象*/
#define DEFAULTTIME 5             /*设定缺省的统计时间段长度为 5 秒*/

static void sig_handler(int signo);
int get_page_fault(void);
int readfile(char *data);

int exit_flag=0;
int page_fault;

int main(int argc,char **argv)
{
        struct itimerval v;
        int cacl_time;

        if(signal(SIGALRM,sig_handler) == SIG_ERR)
        {
                printf("Unable to create handler for SIGALRM\n");
                return -1;
        }
        if(argc <= 2)
        page_fault = get_page_fault();

        /*初始化 timer_real*/
        if(argc < 2)
        {
                printf("Use default time!\n");
                cacl_time = DEFAULTTIME;
```

```
        }
        else if(argc == 2)
        {
                printf("Use user's time\n");
                cacl_time = atoi(argv[1]);
        }
        else if(argc > 2)
        {
                 printf("Usage:mypage [time]\n");
                 return 0;
        }
        v.it_interval.tv_sec = cacl_time;
        v.it_interval.tv_usec = 0;
        v.it_value.tv_sec = cacl_time;
        v.it_value.tv_usec = 0;
        setitimer(ITIMER_REAL,&v,NULL);

        while(!exit_flag)
                ;
        printf("In %d seconds,system calls %d page fault!\n",cacl_time,page_fault);
        return 0;
}

static void sig_handler(int signo)
{
         if(signo == SIGALRM)   /*当 ITIMER_REAL 为 0 时，这个信号被发出*/
         {
                 page_fault = page_fault-get_page_fault();
                 exit_flag = 1;
         }
}

/*该函数通过调用文件操作函数 readfile，得到当前系统的缺页中断次数*/
int get_page_fault(void)
{
   char d[50];
   int retval;

        /*读取缺页中断次数*/
        retval = readfile(d);
        if(retval<0)
        {
                printf("read data from file failed!\n");
                exit(0);
        }
        printf("Now the number of page fault is %s\n",d);
        return atoi(d);
}
```

```
/*该函数对/proc/stat 文件内容进行读操作，读取指定项的值*/
int readfile(char *data)
{
  int fd;
  int seekcount = 0;
        int retval = 0;
        int i = 0;
        int count = 0;
        char c,string[50];

        fd = open(FILENAME,O_RDONLY);
        if(fd < 0)
        {
                printf("Open file /proc/stat failed!\n");
                return -1;
        }

        /*查找 stat 文件中的关键字 intr */
        do{
                 i = 0;
                 do{
                        lseek(fd,seekcount,SEEK_SET);
                        retval = read(fd,&c,sizeof(char));
                        if(retval < 0)
                        {
                                printf("read file error!\n");
                                return retval;
                        }
                        seekcount += sizeof(char);

                        if(c == ' ' || c == '\n')
                        {
                                string[i] = 0;
                                break;
                        }
                        if((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c>= 'A' && c <=
'Z'))
                                 string[i++] = c;
                   }while(1);
           }while(strcmp("intr",string));

           printf("find intr!\n");

       /*读取缺页次数*/
           do{
                   lseek(fd,seekcount,SEEK_SET);
                   retval = read(fd,&c,sizeof(char));
                   if(retval < 0)
                   {
```

```
                printf("read file error!\n");
                return retval;
            }
            seekcount += sizeof(char);
            if(c == ' ' || c == '\n')
            {
                string[i] = 0;
              i = 0;
                count++;
            }
            if((c >= '0' && c <= '9') || (c >= 'a' && c<= 'z') || (c >= 'A' && c<= 'Z'))
string[i++] = c;

    }while(count != 16);

        close(fd);
        strcpy(data,string);
        return 0;
}
```

执行编译后的可执行文件就可以统计出一段时间内缺页中断的次数了。如果系统在指定的时间段内不执行任何用户任务的话，得出的缺页次数往往会是 0 次。为了使实验效果明显，可以在同一系统的另一终端下执行一个大任务，比如 cat 一个大文件，这样在统计的终端下就可以看到一定次数的缺页中断。