实验 5 进程调度算法模拟-RR

一、预备知识

1、pause()函数

pause()函数是用于将调用进程挂起直至捕捉到信号为止。这个函数很常用，通常可以用于判断信号是否已到。

例如：

```
/* alarm_pause.c */
#include <stdio.h>
#include <stdlib.h>
int main()
{
 /*调用 alarm 定时器函数*/
int ret = alarm(5) ;
pause();

printf("I have been waken up.\n"); /* 此语句不会被执行 */
}
```

批注 [s1]: alarm()也称为闹钟函数，它可以在进程中设置一个定时器，当定时器指定的时间到时，它就向进程发送 SIGALRM 信号。

批注 [U2]: 令进程暂停，否则程序运行完 pirntf 后就会直接退出，则无法显示出 alarm(5)的效果

运行结果为：

```
[sl@localhost ~]$ ./pause
Alarm clock
```

因为SIGALRM默认的系统动作为终止该进程，因此程序在打印信息之前，就会被结束了。如果把pause()去掉，程序执行printf直接结束。

2、SIGKILL、SIGSTOP和SIGCONT

SIGKILL	该信号用来立即结束程序的运行，并且不能被阻塞、处理或忽略	终止
SIGSTOP	该信号用于暂停一个进程，且不能被阻塞、处理或忽略	暂停进程
SIGCONT	恢复暂停的进程	恢复进程

例如：在子进程中通过raise()给自己发送SIGSTOP信号，再给自己发送SIGKILL信号。由于前面SIGSTOP信号令子进程暂停了，所以后面的代码无法执行，子进程不可能给自己发送SIGKILL信号。父进程休眠5秒后，给子进程发送SIGCONT信号；子进程接到这个信号后从暂停状态恢复运行，接着运行后面的代码，给自己发送SIGKILL信号，子进程结束运行。

```
/* kill_stop.c */
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
int main()
{
pid_t pid;
int ret;
```

```
 /* 创建一子进程 */
 if ((pid = fork()) < 0)
 {
printf("Fork error\n");
 exit(1);
 }
 if (pid == 0)
 {
 /* 在子进程中使用 raise() 函数发出 SIGSTOP 信号, 使子进程暂停 */
printf("Child(pid : %d) will be stopped\n", getpid());
 raise(SIGSTOP);
printf("Child(pid : %d) has woken up\n", getpid());
raise(SIGKILL);
printf("Child(pid : %d) is over\n", getpid());
 exit(0);
 }
 else
 {
 /* 在父进程中调用 kill() 函数发出SIGCONT信号，让子进程继续执行 */
sleep(5);
kill(pid, SIGCONT);
printf("Parent has woken child up(pid : %d)\n",pid);
 }
 exit(0);
 }
```

3、随机数

在C语言中，rand()函数可以用来产生随机数，但是这不是真正意义上的随机数，是一个伪随机数，是根据一个数（我们可以称它为种子）为基准以某个递推公式推算出来的一系数，当这系列数很大的时候，就符合正态公布，从而相当于产生了随机数，但这不是真正的随机数，当计算机正常开机后，这个种子的值是固定的，为了改变这个种子的值，C提供了srand()函数，它的原形是void srand(int a)。

srand是种下随机种子数，你每回种下的种子不一样，用rand得到的随机数就不一样。为了每回种下一个不一样的种子，可以选用time(0)，time(0)是得到当前时时间值（因为每时每刻时间是不一样的了）。是随机数常用的方法。

```
srand(time(0)) ; //先设置种子，用1970.1.1至今的秒数，初始化随机数种子。
rand();          //然后产生随机数
```

例如：

```
/* rand_srand.c */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>          //使用当前时钟做种子
void main( void )
```

```
{int i;
     srand( time(0) );      //初始化随机数
    for( i = 0; i < 10;i++ )       //打印出10个1-100之间的随机数
    printf( " %d\n", rand()%100);
}
```

4、时间和定时器

在Linux系统下，获取毫秒级的系统时间，使用毫秒级的定时器，修改文件访问时间等，用sys/time.h下的方法会很方便。sys/time.h下的常用的数据结构和方法如下。

（1）数据结构：

timeval: 存储时间格式（time value）

```
struct timeval {
   long tv_sec; //秒
   long tv_usec; //毫秒
   }
```

itimerval:定时器结构体

```
struct itimerval{
   struct timerval it_value;      //从设定定时器开始计算，到第一次定时器生效的时间
   struct timerval it_interval;   //每两次定时器生效的时间间隔
   };
```

【注意】

- 如果 it_value 被设为 0，系统无视 it_interval 的值并且终止 timer，所以可以通过这种方式来使 timer 失效
- 如果 it_interval 被设为 0，则 timer 在执行一次之后失效

（2）使用方法：

设置定时器：int setitimer(int, const struct itimerval *, struct itimerval *);

第一个参数：定时器类型。我们一般使用ITIMER_REAL（实时定时器），不管进程在何种模式下运行（甚至在进程被挂起时），它总在计数。定时到达，向进程发送SIGALRM信号。

第二个参数：指定的定时器

第三个参数：调用setitimer之前的旧定时器，没有的话就为NULL。

（3）举例：

用setitimer设定的timer时间到的时候，会发出SIGALRM信号，通过signal(SIGALRM, sayHello)方法指定在收到信号时的动作。

按照代码中的设定，从setitimer方法被调用开始算起，经过5秒钟第一次执行sayHello方法，之后每3秒钟执行一次sayHello。

通过while(true)死循环保证进程不会终止，如果没有这个死循环，timer只会在5秒钟后响应一次。

```
/*timer.c */
#include <sys/time.h>
#include <stdio.h>
```

```
#include <signal.h>
#include <string.h>
#include <unistd.h>
void sayHello(int);
int main()
{
    signal(SIGALRM, sayHello);
    struct itimerval timer;
    memset(&timer, 0, sizeof(timer));
    timer.it_value.tv_sec = 5;
    timer.it_value.tv_usec = 0;
    timer.it_interval.tv_sec = 3;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, NULL);
while(1)
{
    pause();
}
    return 0;
}

void sayHello(int temp)
{
    printf("hello world!\n");
}
```

二、练习：模拟时间片轮转算法

1、用到的数据结构

```
/* PCB */
struct PCB
{
       pid_t pid;//进程 PID
       int state; //状态信息，1 表示正在运行，0 表示暂停,-1 表示结束
       unsigned long runned_time;//已运行时间
       unsigned long need_running_time;//剩余运行时间
};
/* PCB集合 */
struct PCB pcb[TOTAL];  //PCB 集合
```

2、算法思路

算法实现分主函数（main）和分派函数（Dispatch）。

（1）其中主函数（main）的核心功能为：

实现 6 个子进程的创建和子进程 PCB 的初始化。对子进程 PCB 初始化时，状态设为 0，运行时间由随机数产生。子进程创建后，就通过信号 SIGSTOP 让它处于暂停状态，当被分派

函数（Dispatch）选中后，才能继续执行，输出子进程 *x* 正在执行的信息。
同时要在主程序里设置定时器，定时器的时间间隔为时间片长度，时间片到，就调用分派函数（Dispatch）重新选派程序。
（2）分派函数的核心功能：
将正在执行的子进程暂停，状态变为 0，修改已运行时间和剩余运行时间。
如果该子进程剩余时间小于或等于 0，说明执行完毕，PCB 状态改为-1，结束该子进程。
重新选择下一个子进程，状态变为 1，输出该子进程的已运行时间和剩余运行时间，让该子进程恢复运行。
当所有子进程都结束后，则父程序结束。