****

**操作系统及安全**

**实 验 报 告**

**实验单元：** 多线程控制

**班 级：** 4211090901

**学 号：**

**姓 名：**

# 实验目的

1. 认识程线程与进程的关系；
2. 掌握pthread库的基本使用。
3. 能够基于pthread编写多线程系统应用。

# 实验内容

1. 编程实现多线程模拟龟兔赛跑的程序，显示效果自行设计。

要求：定义速度，定义距离

代码：

// 线程

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

//定义结构体 速度和距离

typedef struct run

{

    int speed;

    int distance;

}run;

void\* **turtle**(void\* args){

    run \*runInfo;

    runInfo = (run\*)args;

    for (int i = 0; i < runInfo->distance; i+=runInfo->speed)

    {

**printf**("乌龟正在跑步，已经跑了%d,一共有%d\n",i,runInfo->distance);

**sleep**(1);

    }

}

void\* **rabbit**(void\* args){

    run \*runInfo;

    runInfo = (run\*)args;

    for (int i = 0; i < runInfo->distance; i+=runInfo->speed)

    {

**printf**("兔子正在跑步，已经跑了%d,一共有%d\n",i,runInfo->distance);

**sleep**(1);

    }

}

int **main**(int argc, char const \*argv[])

{

    run runTurtle = {5,30};

    run runRubbit = {10,50};

    pthread\_t pthread[2];

    // 创建进程

    int pthread0 = **pthread\_create**(&pthread[0],**NULL**,**turtle**,(void\*)&runTurtle);

    int pthread1 = **pthread\_create**(&pthread[1],**NULL**,**rabbit**,(void\*)&runRubbit);

    // 检查进程是否创建成功

    if(!pthread0 && !pthread1){

**printf**("线程号1:%ld,线程号2:%ld\n",pthread[0],pthread[1]);

    }else{

**printf**("进程创建失败");

        return -1;

    }

    // 等待进程运行完成

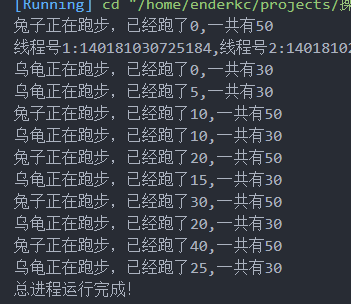
**pthread\_join**(pthread[0],**NULL**);

**pthread\_join**(pthread[1],**NULL**);

**printf**("总进程运行完成！\n");

    return 0;

}

运行截图：

1. 编程实现基于互斥量的多线程共享资源访问，自行设计共享资源，可以是一个数组，一个结构体，一个文件，等等。（读、修改、插入、删除）

给予互斥量的 栈的访问

给予互斥量的 链表的访问

代码：

/\*

    给予互斥量数组的访问

\*/

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/syscall.h>

// 共享资源

int a[10] = {0,1,2,3,4,5,6,7,8,9};

// 信号量

pthread\_mutex\_t mutex;

// 线程函数操作

/\*

    根据参数下标，读取数组中对应的数据

\*/

void\* **myRead**(void\* arg){

    //自己分离自己，不用主线程回收它的资源

**pthread\_detach**(**pthread\_self**());

    int n = \*(int\*)arg;

    if (n <=10 && n >= 0){

        int t = a[n];

**printf**("共享资源中第%d个值为%d\n",n,t);

    }else{

**perror**("读取的数据超出数组范围\n");

    }

}

/\*

    读取数组中的所有数据

\*/

void\* **readAll**(void\* arg){

    //自己分离自己，不用主线程回收它的资源

**pthread\_detach**(**pthread\_self**());

    for (int i = 0; i < 10; i++)

    {

**printf**("%d  ",a[i]);

    }

**printf**("  数组输出完毕\n");

}

/\*

    删除数组中某一个元素

\*/

void\* **delete**(void\* arg){

    //自己分离自己，不用主线程回收它的资源

**pthread\_detach**(**pthread\_self**());

    int n = \*(int\*)arg;

    if(n > 9){

**perror**("删除数据超出范围\n");

        return (void\*)-1;

    }

**pthread\_mutex\_lock**(&mutex); // 加锁

    a[n] = 0;

    // 数组后面的元素向前移动一位

    for (int i = n; i < 9; i++)

    {

        int tmp = a[i];

        a[i] = a[i+1];

        a[i+1] = tmp;

    }

**pthread\_mutex\_unlock**(&mutex); // 解锁

    // 展示删除完之后的样子

    pthread\_t t;

**printf**("删除完成:");

**readAll**(**NULL**);

}

/\*

    项数组中插入

    接收两个参数：

        第一个参数：插入的位置

        第二个参数：插入的参数

\*/

void\* **insert**(void\* args){

    //自己分离自己，不用主线程回收它的资源

**pthread\_detach**(**pthread\_self**());

    int \*arr = (int\*)args;

    if(arr[0] > 9){

**perror**("插入数据超出范围\n");

        return (void\*)-1;

    }

**pthread\_mutex\_lock**(&mutex); // 加锁

    for (int i = 9; i > arr[0]; i--)

    {

        a[i] = a[i-1];

    }

    a[arr[0]] = arr[1];

**pthread\_mutex\_unlock**(&mutex); // 解锁

    // 展示删除完之后的样子

    pthread\_t t;

**printf**("插入完成:");

**readAll**(**NULL**);

}

/\*

    pthread\_join用于等待一个线程的结束，

    也就是主线程中要是加了这段代码，

    就会在加代码的位置卡主，直到这个线程执行完毕才往下走。

\*/

int **main**(int argc, char const \*argv[])

{

**printf**("========BEGIN==========\n");

    // 线程号

    pthread\_t tid[4];

    // 初始化信号量

**pthread\_mutex\_init**(&mutex,**NULL**);

    // 创建线程(读取)

    int readData = 5;

    int deleteData = 3;

    int insertData[] = {4,7};

    int ret1 = **pthread\_create**(&tid[0],**NULL**,**myRead**,(void\*)&readData);

    int ret2 = **pthread\_create**(&tid[1],**NULL**,**delete**,(void\*)&deleteData);

    int ret3 = **pthread\_create**(&tid[2],**NULL**,**insert**,(void\*)&insertData);

    int ret4 = **pthread\_create**(&tid[3],**NULL**,**readAll**,**NULL**);

**pthread\_mutex\_destroy**(&mutex);  // 销毁互斥锁

    if(ret1 && ret2 && ret3 && ret4){

**perror**("创建进程失败\n");   // 用于打印错误信息

        return -1;

    }

    for (int i = 0; i < 4; i++)

    {

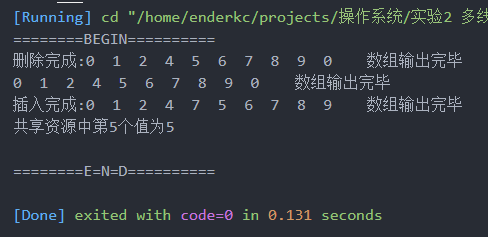
**pthread\_join**(tid[i],**NULL**);

    }

**printf**("\n========E=N=D==========\n");

    return 0;

}

运行截图：

# 实验总结

**完成该实验用到了哪些知识点（罗列）**

**完成实验过程中，自己在技术、实践能力，解决问题方面的收获**

* segment fault 原因 ：
  + 内存访问越界
  + 多线程程序使用了线程不安全的函数。
  + 多线程读写的数据未加锁保护。
  + 非法指针
  + 堆栈溢出.
* pthread\_join用于等待一个线程的结束，也就是主线程中要是加了这段代码，就会在加代码的位置卡主，直到这个线程执行完毕才往下走。
* perror("创建进程失败\n"); // 用于打印错误信息
* //自己分离自己，不用主线程回收它的资源 pthread\_detach(pthread\_self());

**遇到的问题及具体解决方案。**

* 不会分析具体程序的运行过程，解决方案：找相关博客，ChatGPT