## 实验9　Linux线程（Thread)

**1. 实验目的**

了解Linux中的线程相关机制。

1. **相关知识**

Linux线程相关的初级实验习题包括理解和应用进程与线程的概念、创建和管理线程、线程同步以及线程间通信等。以下是一些初级的实验习题：

### 习题一：理解并创建线程

* 什么是进程？什么是线程？它们之间有什么区别和联系？
* 在Linux系统中，如何使用pthread库创建一个线程？请写出基本的代码结构。
* 编写一个程序，使用pthread\_create()函数创建一个线程，该线程简单地打印“Hello from Thread!”。
* 推荐程序命名：thread-9-1.c

下面的C程序示例，它使用pthread\_create()函数创建一个线程，该线程简单地打印“Hello from Thread!”。

**c**复制代码

#include <stdio.h>#include <stdlib.h>#include <pthread.h>

*// 线程函数*void\* threadFunction(void\* arg) {

printf("Hello from Thread!\n");

return NULL;

}

int main() {

pthread\_t thread; *// 定义线程标识符*

int result;

*// 创建线程*

result = pthread\_create(&thread, NULL, threadFunction, NULL);

if (result != 0) {

fprintf(stderr, "Error creating thread\n");

return 1;

}

*// 等待线程结束*

result = pthread\_join(thread, NULL);

if (result != 0) {

fprintf(stderr, "Error joining thread\n");

return 2;

}

return 0;

}

### 代码说明：

****包含头文件****：

* + #include <stdio.h>: 用于标准输入输出函数，如printf。
  + #include <stdlib.h>: 用于标准库函数，如exit。
  + #include <pthread.h>: 用于POSIX线程（pthread）库函数。

****线程函数****：

* + void\* threadFunction(void\* arg): 这是线程执行的函数。在这个例子中，它只是打印一条消息并返回NULL。

****主函数****：

* + pthread\_t thread;: 定义一个线程标识符变量。
  + pthread\_create(&thread, NULL, threadFunction, NULL);: 创建一个新线程，执行threadFunction。如果创建失败，返回非零值。
  + pthread\_join(thread, NULL);: 等待线程结束。如果等待失败，返回非零值。

### 编译和运行：

在Linux系统上，可以使用以下命令编译和运行这个程序：

**sh**复制代码

gcc -o thread\_example thread\_example.c -lpthread

./thread\_example

运行后，你应该会看到输出：

复制代码

Hello from Thread!

这表明线程已经成功创建并执行了它的任务。

### 习题二：线程的同步与互斥

* 什么是竞态条件（Race Condition）？为什么在多线程编程中需要避免？
* 在Linux系统中，如何使用互斥锁（mutex）来保护共享资源？请写出基本的代码结构。
* 编写一个程序，使用两个线程分别增加和减少一个全局变量的值。使用互斥锁来确保对该变量的访问是线程安全的。
* 推荐程序命名thread-9-2.c

**cpp**复制代码

#include <iostream>#include <pthread.h>

*// 全局变量和互斥锁*int global\_var = 0;pthread\_mutex\_t mutex;

*// 增加全局变量的线程函数*void\* increment(void\* arg) {

for (int i = 0; i < 100000; ++i) {

pthread\_mutex\_lock(&mutex);

++global\_var;

pthread\_mutex\_unlock(&mutex);

}

return nullptr;

}

*// 减少全局变量的线程函数*void\* decrement(void\* arg) {

for (int i = 0; i < 100000; ++i) {

pthread\_mutex\_lock(&mutex);

--global\_var;

pthread\_mutex\_unlock(&mutex);

}

return nullptr;

}

int main() {

pthread\_t inc\_thread, dec\_thread;

*// 初始化互斥锁*

pthread\_mutex\_init(&mutex, nullptr);

*// 创建线程*

pthread\_create(&inc\_thread, nullptr, increment, nullptr);

pthread\_create(&dec\_thread, nullptr, decrement, nullptr);

*// 等待线程完成*

pthread\_join(inc\_thread, nullptr);

pthread\_join(dec\_thread, nullptr);

*// 销毁互斥锁*

pthread\_mutex\_destroy(&mutex);

*// 输出最终结果*

std::cout << "Final value of global\_var: " << global\_var << std::endl;

return 0;

}

这个程序创建了两个线程，一个用于增加全局变量global\_var的值，另一个用于减少它的值。通过使用互斥锁mutex来确保对global\_var的访问是线程安全的。最终，主线程等待两个子线程完成后，输出global\_var的最终值。

**4. 实验思考与扩充**