

1 实验目的

- 1) 了解线程的原理
- 2) 编写并运行一个多线程程序

2 实验内容

写一个多线程的程序，生成 Fibonacci 序列。($fib_0=0$; $fib_1=1$; $fib_n=fib_{n-1}+fib_{n-2}$)。

要求 1：用户运行程序时在命令行输入一个数 N，然后程序创建一个独立新的线程来输出连续 N 个 Fibonacci 数。

要求 2：采用两种 thread 来实现(pthread 或者 Windows thread 或者 Java thread)。

3 实验过程概述

本实验分为相关原理学习，程序设计，程序运行结果等阶段。本试验选用 windows thread 和 pthread。

4 相关原理介绍

Pthread

Pthread 是一套通用的线程库，它广泛的被各种 Unix 所支持，是由 POSIX 提出的。

基本接口介绍¹：

1. pthread_create

```
#include <pthread.h>
```

```
int pthread_create(pthread_t * thread, pthread_attr_t * attr, void *  
(*start_routine)(void *), void * arg);
```

第一个参数是一个 pthread_t 型的指针用于保存线程 id. 以后对该线程的操作都要用 id 来标示.

第二个参数是一个 pthread_attr_t 的指针用于说明要创建的线程的属性, 使用 NULL, 表示要使用缺省的属性.

第三个参数指明了线程的入口, 是一个只有一个(void *)参数的函数.

第四个参数指明了要传到线程的参数.

2. pthread_exit

```
#include <pthread.h>
```

```
void pthread_exit(void *retval);
```

pthread_exit 结束调用线程的执行.所有通过 pthread_cleanup_push 设置的清除句柄将会被反序执行(后进先出)。)。

最后调用线程被终止。

retval 是这个线程结束的返回值, 可以通过在别的线程中调用 pthread_join 来获取这个值。没有返回值。

3. pthread_join

```
#include <pthread.h>
```

```
int pthread_join(pthread_t th, void **thread_return);
```

挂载一个正在执行的线程直到该线程通过调用 `pthread_exit` 或者 `cancelled` 结束。

如果 `thread_return` 不为空，则线程 `th` 的返回值会保存到 `thread_return` 所指向的区域。

`th` 的返回值是它给 `pthread_exit` 的参数，或者是 `pthread_cancelled` 如果是被 `cancelled` 的。

被依附的线程 `th` 必须是 `joinable` 状态。一定不能是 `detached` 通过使用 `pthread_detach` 或者 `pthread_create` 中使用 `pthread_create_detached` 属性。

当一个 `joinable` 线程结束时，他的资源(线程描述符和堆栈)不会被释放直到另一个线程对它执行 `pthread_join` 操作。

如果成功，返回值存储在 `thread_return` 中，并返回 0，否则返回错误码：

ESRCH:找不到指定线程

EINVAL:线程 `th` 是 `detached` 或者已经存在另一个线程在等待线程 `th` 结束

EDEADLK:`th` 的参数引用它自己(即线程不能 `join` 自身)

注意事项：编译的时候，需要加上编译参数 “-pthread”。

Windows thread^{II}

在 Windows 的多线程编程中，创建线程的函数主要有 `CreateThread`。

1、创建线程

可以通过调用 `CreateThread` 函数来创建一个线程，函数原型如下：

```
HANDLE CreateThread(
```

```

LPSECURITY_ATTRIBUTES lpsa,

DWORD cbStack,

LPTHREAD_START_ROUTINE lpStartAddr,

LPVOID lpvThreadParam,

DWORD fdwCreate,

LPDWORD lpIDThread

);

```

参数 lpStartAddr 是线程开始的地址，也就是新创建的线程开始执行的地方，一般将一个函数（线程函数）的地址传递给该参数。

参数 lpvThreadParam 是传递给一中所说的线程函数的参数。

2、线程函数

线程函数的类型并无限制，返回值可以是任意类型。由于创建线程的时候，只传入了一个 LPVOID 类型的参数。定义的线程函数最好也只包含一个 LPVOID 类型的参数，如果需要多个参数，可以传递一个结构体指针，然后强制转换。

5 实验程序设计

以 Pthread 为例

```
#include<stdio.h>
```

```
#include"pthread.h"
```

```
int N; //线程间共享的数据
```

```

void* fib(){ //线程函数

    int i;

    int fib_a=0;

    int fib_b=1;

    int sum;

    printf("%d %d",fib_a,fib_b);

    for (i=0;i<N-2;i++){

        sum=fib_a+fib_b;

        fib_a=fib_b;

        fib_b=sum;

        printf(" %d",sum);

    }

    pthread_exit(0); //线程结束

}

```

```

int main(){

    char *number;

    printf("please input the number N\n");

    scanf("%d",&N);

    pthread_t tid;

    pthread_attr_t attr;

    pthread_attr_init(&attr);

```

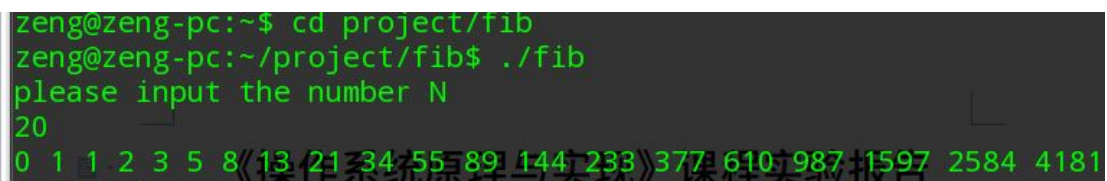
```
pthread_create(&tid,&attr,fib,NULL); 创建线程

pthread_join(tid,NULL);

}
```

6 实验结果

在终端输出 20 个斐波拉契数列。



```
zeng@zeng-pc:~$ cd project/fib
zeng@zeng-pc:~/project/fib$ ./fib
please input the number N
20
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

7 附录

Windows thread 程序代码：

```
#include<stdio.h>

#include<windows.h>

int N;

DWORD WINAPI fib(LPVOID param){

    int i;

    int fib_a=0;

    int fib_b=1;

    int sum;

    printf("%d %d",fib_a,fib_b);

    for (int i=0;i<N-2;i++){

        sum=fib_a+fib_b;

        fib_a=fib_b;
```

```

        fib_b=sum;

        printf(" %d",sum);

    }

    return DWORD();

}

int main(){

    printf("please input the number N\n");

    scanf("%d",&N);

    HANDLE ThreadHandle;

    ThreadHandle=CreateThread(NULL,0,fib,NULL,0,NULL);

    if(ThreadHandle!=NULL){

        WaitForSingleObject(ThreadHandle,INFINITE);

        CloseHandle(ThreadHandle);

        printf("\nthe windows thread finished");

    }

    return 0;

}

```

ⁱ Pthread 学习: <http://www.cppblog.com/saha/articles/189802.html>

ⁱⁱ Windows thread:<http://blog.csdn.net/njuitjf/article/details/5315047>