#### 第八章 线程编程

1. 什么是线程？线程与进程有什么相同和不同的地方？

线程：

线程是操作系统中进行资源调度的最小单位，被包含在进程中，是进程中的实际运行单位。

线程与进程不同：

线程的单位小于进程，一个进程可包含多个线程；

线程是资源调度的独立单位，进程的资源分配的独立单位；

线程占用资源比进程少；

线程的切换CPU的开销、延时小于进程切换CPU

2. 线程之间的通信方式有哪些？

使用全局变量或使用消息机制通信

3. 怎样结束一个线程？

线程正常执行结束并返回；

线程被另一个线程结束；

线程自行调用pthread\_exit()

4. 线程之间的同步方法有哪些？

互斥量、信号量、条件变量

5. 编写一个工程，包括3个文件，主程序main.c,线程一thread1.c，线程二thread2.c，其中main.c创建另外两个线程，线程一每隔1秒钟，读取系统的当前时间并显示出来，线程二计算pi的值，每隔1秒钟把计算结果显示出来。

//main.c

#include"threads\_def.h"

#include<pthread.h>

int main(void)

{

pthread\_t thread\_1, thread\_2;

void \*ret = NULL;

pthread\_create(&thread\_1, NULL, thread1, 0);

pthread\_create(&thread\_2, NULL, thread2, 0);

pthread\_join(thread\_1, &ret);

pthread\_join(thread\_1, &ret);

return 0;

}

//thread1.c

#include"threads\_def.h"

#include<time.h>

#include<stdio.h>

#include<unistd.h>

void\* thread1(void \*data)

{

time\_t t = 0;

struct tm \*LocalTime = NULL;

while (1)

{

time(&t);

//get local time

LocalTime = localtime(&t);

printf("LocalTime: %s", asctime(LocalTime));

sleep(1);

}

}

//thread2.c

#include"threads\_def.h"

#include<stdio.h>

#include<unistd.h>

#define MAX\_ACCU 500

void\* thread2(void \*data)

{

int R[MAX\_ACCU] = { 2 };//R(1) = 2.00...

int Sum[MAX\_ACCU] = { 2 };//Sum(1) = 2.00...

long i, j, t, d;

int plus;

//Calc

for (i = 1; i < MAX\_ACCU; i++)

{

//R(n)\*(n)

plus = 0;

for (j = MAX\_ACCU - 1; j >= 0; j--)

{

t = R[j] \* i + plus;

R[j] = t % 10;

plus = t / 10;

}

//R(n+1) = R(n)\*n / (2\*n+1)

d = 2 \* i + 1;

plus = 0;

for (j = 0; j < MAX\_ACCU; j++)

{

t = R[j] + plus \* 10;

R[j] = t / d;

plus = t % d;

}

//Sum(n+1) = Sum(n) + R(n+1)

plus = 0;

for (j = MAX\_ACCU - 1; j >= 0; j--)

{

t = Sum[j] + R[j] + plus;

Sum[j] = t % 10;

plus = t / 10;

}

}

while (1)

{

//Output

printf("%d.", Sum[0]);

for (i = 1; i < MAX\_ACCU; i++)

{

printf("%d", Sum[i]);

}

printf("\n");

sleep(1);

}

}

//threads\_def.h

//Get time and show

void\* thread1(void \*data);

//Calc PI and show

void\* thread2(void \*data);

//Makefile

CC=gcc

Thread:main.o thread1.o thread2.o

$(CC) -lpthread -o Thread main.o thread1.o thread2.o

thread1.o:main.o threads\_def.h

$(CC) -c main.c

thread1.o:thread1.c threads\_def.h

$(CC) -c thread1.c

thread2.o:thread2.c threads\_def.h

$(CC) -c thread2.c

clean:

rm Thread \*.o

