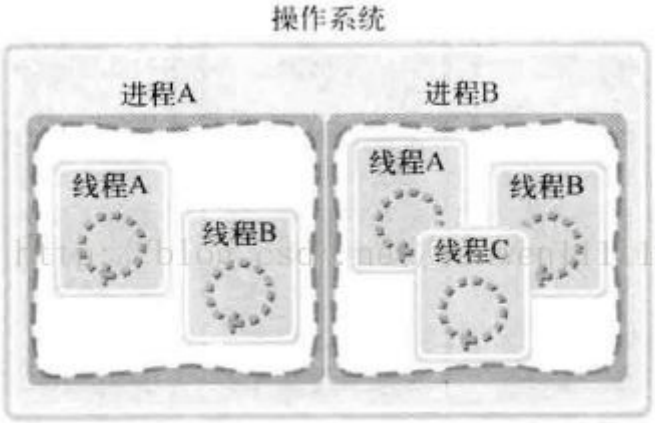
TCP/IP网络编程(十八)

笔记本: 网络编程
创建时间: 2018/11/16 11:07 **更新时间:** 2018/11/21 18:24
作者: xiangkang94@outlook.com
标签: 第十八章(多线程服务器端的实现)

- Web服务器的发展迫使UNIX系列操作系统开始重视线程，Web服务器进程需要向多个客户端提供服务，因此逐渐舍弃进程，而用效率更高的线程
- 多进程模型的缺点
  - 创建进程需要大量的开销
  - 进程间通信需要用到IPC技术
  - 经常发生上下文切换，时间长
- 多线程的优点
  - 线程的创建和上下文切换比进程简单且快
  - 线程间的通信无需特殊技术
- 进程和线程之间的比较
  - 每个进程都有自己的数据区、heap、stack，进程间相互独立。线程间共享数据区和heap（线程拥有自己的栈）

  - 图18-2 线程的内存结构
  - 进程：在操作系统构成单独执行流的单位; 线程：在进程构成单独执行流的单位



  - 图18-3 操作系统、进程、线程之间的关系
- 线程创建

```
#include <pthread.h>
int pthread_create(pthread_t * restrict thread, const pthread_attr_t restrict attr, void *(*start_routine)
(void *), void * restrict arg)
//restrict用来修饰指针不能被别的指针引用
//成功返回0，失败其他值
//thread：保存新创建进程ID的变量地址值
//attr：用于传递线程属性的参数，NULL表示默认
//start_routine：线程的执行流函数指针
//arg：第三个参数函数指针的参数信息
```

- 线程等待

```
int pthread_join(pthread_t thread, void **status)
//成功返回0，失败其他值
//thread：等待的线程ID
//status：保存线程的main函数返回值的指针变量地址值
```

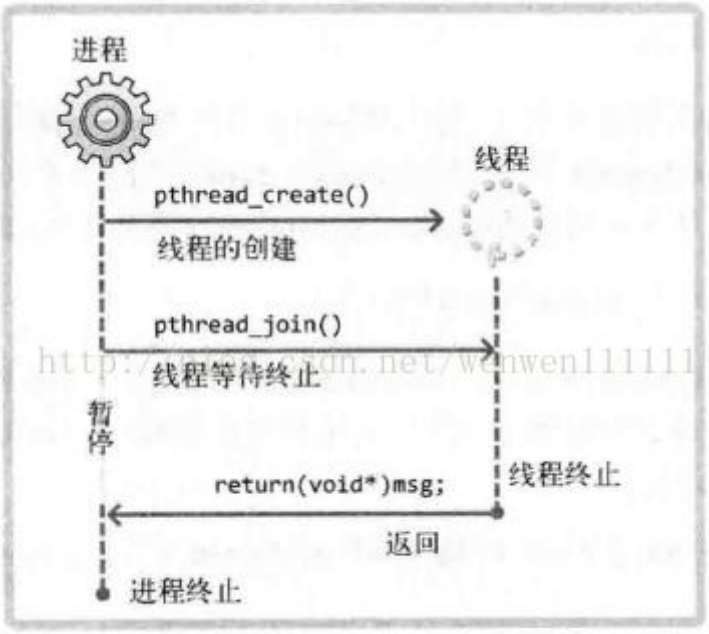
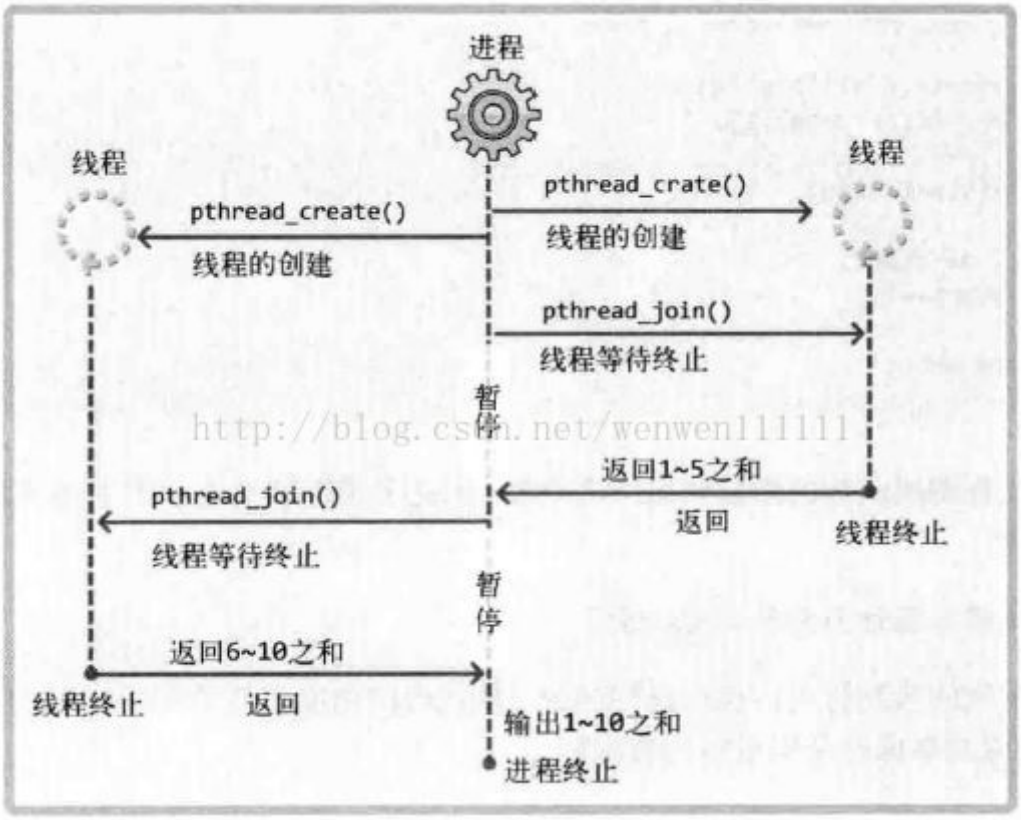


图18-6　调用pthread_join函数

  - //调用join函数的进程(或线程)将进入等待状态，直到第一个参数为ID的线程终止为止。
- 可在临界区内调用的函数
  - 临界区：多个线程同时调用函数时可能产生问题，这类函数内部存在临界区（共同访问的那块代码）。临界区中至少存在一条这类代码。
  - 线程安全函数：被多个线程同时调用时不会引发问题；线程安全函数的名称以_r后缀 //与临界区无关，安全函数也可能有临界区
  - 非线程安全函数：被同时调用时会引发问题
  - 编译使用_r的函数方法：声明头文件前定义_REENTRANT，也可以 gcc -D_REENTRANT mythread.c -o mthread -lpthread
- sample

```
#include <stdio.h>
#include <pthread.h>
void * thread_summation(void *arg);

int sum = 0;
int main(int argc, char *argv[]){
    pthread_t id_t1,id_t2;
    int range1[] = {1,5};
    int range2[] = {6,10};

    pthread_create(&id_t1,NULL,thread_summation,(void *)range1);
    pthread_create(&id_t2,NULL,thread_summation,(void *)range2);
```

```
    pthread_join(id_t1,NULL);  //调用join之后会等待id_t1线程返回
    pthread_join(id_t2,NULL);  //id_t1线程返回之后等待id_t2线程返回

    printf("sum = %d\n",sum);
    return 0;
}

void * thread_summation(void *arg){
    int start = ((int *)arg)[0];
    int end = ((int *)arg)[1];

    while(start <= end){
        sum += start;
        start++;
    }
    return NULL;
}
```



//由于上述例子数据较小,虽然存在临界区问题，但是效果很难显现，下列实例可能会引发问题

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

#define NUM_THREAD  100
void * thread_inc(void *arg);
void * thread_des(void *arg);
long long num = 0;  //bit64
int main(int argc, char *argv[]){
    pthread_t thread_id[NUM_THREAD];
    int i;
    for(i=0;i<NUM_THREAD;i++){
        if(i%2)
            pthread_create(&thread_id[i],NULL,thread_inc,NULL);
        else
            pthread_create(&thread_id[i],NULL,thread_des,NULL);
    }
    for(i=0;i<NUM_THREAD;i++)
        pthread_join(thread_id[i],NULL);
```

```
    printf("result = %lld\n",num);  //预期结果应该是0，一般的线程对全局sum做++,一半--
    return 0;
}

void * thread_inc(void *arg){
    int i;
    for(i=0;i<50000000;i++){
        num += 1;
    }
    return NULL;
}

void * thread_des(void *arg){
    int i;
    for(i=0;i<50000000;i++){
        num -= 1;
    }
    return NULL;
}
```

- 线程存在的问题和临界区
  - 多线程访问同一变量是问题：参考上例中的sum
  - 临界区的定义：函数内同时运行多个线程时引起问题的多条语句构成的代码块
- 线程同步
  - 互斥量(加锁) 未展开
  - 信号量 未展开
- 摧毁线程的方法
  - 调用pthread_join函数 //线程一直等待，**阻塞**
  - 调用phread_detach函数 //通常使用这种方式摧毁线程，**不会引起线程终止或者进入阻塞状态，且不能在调用该函数后再调用join**