并发服务器 II—— 多线程

2012-10

课程内容

- POSIX 线程库 Pthreads 介绍
- POSIX pthreads 库提供的基本线程的操作
- 线程的属性
- 线程互斥和同步
- 使用 GDB 调试线程以及线程的调优
- 多线程网络服务器模型
- 试验题目

多进程服务器模型回顾

Unix 下的大多数网络服务器程序都是这么编写的，即父进程接受连接，派生子进程，子进程处理与客户的交互。

虽然这种模型很多年来使用得很好，但是 fork 时有一些问题：

- fork 是昂贵的。内存映像要从父进程拷贝到子进程，所有描述字要在子进程中复制等等。目前有的 Unix 实现使用一种叫做写时拷贝（copy – on – write）的技术，可避免父进程数据空间向子进程的拷贝。尽管有这种优化技术， fork 仍然是昂贵的；
- fork 子进程后，需要用进程间通信（IPC）在父子进程之间传递信息。Fork 之前的信息容易传递，因为子进程从一开始就有父进程数据空间及所有描述字的拷贝。但是从子进程返回信息给父进程需要做更多的工作。 ；

线程模型的提出

- 线程有助于解决这两个问题。线程有时被称为轻权进程（lightweight process），因为线程比进程“轻权”，一般来说，创建一个线程要比创建一个进程快 10 ～ 100 倍。
- 一个进程中的所有线程共享相同的全局内存，这使得线程很容易共享信息，但是这种简易性也带来了同步问题。
- 一个进程中的所有线程不仅共享全局变量，而且共享：进程指令、大多数数据、打开的文件（如描述字）、信号处理程序和信号处置、当前工作目录、用户 ID 和组 ID 。
- 但是每个线程有自己的线程 ID 、寄存器集合（包括程序计数器和栈指针）、栈（用于存放局部变量和返回地址）、 error 、信号掩码、优先级。
- 程序的编译

  gcc –o test test.c -lpthread

线程的创建

```
#include <pthread.h>
   int  pthread_create(pthread_t  *  thread,
                  pthread_attr_t * attr,
                  void *(*start_routine)(void *),
                  void * arg
                        );
```

线程的退出

- 显示的调用 pthread_exit() 结束线程执行
  - void pthread_exit(void *retval);
- 让线程处理程序返回
- 使用 pthread_cancel() 函数终止其他线程的执行
  - int pthread_cancel(pthread_t thread);

等待线程结束

使用 pthread_join() 函数等待被创建的线程结束
pthread_join() 函数会挂起创建线程的线程的执行
直到等待到想要等待的子线程
函数原型 ：

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

线程的分离

联合线程与分离线程的差别

主线程可以不断地创建子线程

子线程本身自己有自我回收内存资源的能力

函数原型：

int pthread_detach(pthread_t th);

pthread_detach() 和 pthread_join() 一般情况下不能同时使用

线程的属性

属性名	意义
detachstate	**选择被创建的线程是处于可加入的状态还是分离状态**
schedpolicy	**为被创建的线程选择调度策略。**
schedparam	**为被创建的线程选择调度参数。**
inheritsched	**选择对新创建的线程的调度策略和调度参数是否被 schedpolicy 和 schedparam 属性决定或者是通过父线程继承而得到的**
scope	**为选择被创建的线程调度竞争范围。**

同步操作中的互斥访问—— mutex

- Mutex ：互斥设备 (MUTual Exclusion device)
- mutex 有如下特性：
  - 原子性：对 mutex 的加锁和解锁操作是原子的
  - 单一性：拥有 mutex 的线程除非释放 mutex ，否则其他线程不能拥有此 mutex
  - 非忙等待：等待 mutex 的线程处于等待状态，直到要等待的 mutex 处于未加锁状态，这时操作系统负责唤醒等待此 mutex 的线程

POSIX 线程库中与 mutex 相关函数

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
    pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

使用 GDB 调试线程以及线程的调优

- GDB 不仅可以调试单进程程序，也可以调试多进程、多线程程序
- 使用 info thread 来查看当前系统中的线程信息
- 通过 thread 命令可以切换线程

线程并发服务器

步骤：

1. 初始化套接字

   sockfd=socket();

   bind(sockfd, 。。。)

2. 侦听套接字

   listen(sockfd,...)

3. 创建线程，线程使用连接套接字进行实际通讯

与进程模型的差别

线程间是共享进程的存储空间，子进程是复制父进程的资源
所以对于多线程而言：

1. 在线程内不要关闭侦听套接字；（考虑后果）；
2. 传递的参数采用形参或是针对线程传递专用参数；

```
如传连接套接字
int connectd;
connectd=accetp();
pthread_create(&thread,NULL,(void*)command,
(void*)connectd);
```

或

```
int * connectd ;
connectd = (int *)malloc(sieof(int));
pthread_create(&thread,NULL,(void*)command,
(void*)connectd);
```

3、缓冲区在线程的堆栈分配；

试验题目 1 ：

修改远程控制程序服务器程序，将其从循环模式或多进程模式修改为多线程模式

多线程端口扫描程序

实现一个多线程端口扫描程序：

要求：

1. 能同时扫描 5 个 IP 地址；
2. 针对每个 iP 地址，开设 100 个线程对其进行扫描；
3. 如果端口打开，使用函数 getservbyport 获取其服务名，在屏幕上打印： IP port servername, 如果是未知服务，则屏幕显示： ip port unkonown

思路（1）

```
struct port_seg
{
        int dest;
        unsigned min_port;
        unsigned max_port;
};
typedef struc port_seg port_segment;
```

思路（2）

主线程：

1、从控制端接受用户的输入；

2、根据用户的输入结果，根据 IP 地址创建 100 线程进行扫描

- 创建 100 个线程描述符

```
pidthread_t * thread;
thread = ( pthread_t * )malloc( THREAD_NUM * sizeof(pthread_t) );
```

- 为每个线程创建扫描的分工

```
for ( int j = 0; j < argc - 1; ++j ) {
    for ( int i = 0; i < THREAD_NUM; ++i ) {
        port_segment port;
        port.dest = dest_ip[ j ];
        port.min_port = i * SEG_LEN + 1;

        /* the last segment */
        if ( i == (THREAD_NUM - 1) )
            port.max_port = MAX_PORT;
        else
            port.max_port = port.min_port + SEG_LEN - 1;
.
```

思路（3）

3、创建线程，开始扫描

```
if ( pthread_create(&thread[i], NULL, scan, (void *)&port) != 0 )
        {
            perror( "pthread_create failed\n" );
            free(thread);
            exit(-2);
   }
```

4、释放创建的资源

```
free(pthread);
```

线程程序