<https://www.cnblogs.com/aga-j/archive/2011/09/18/2180609.html>

**目标：**

  修改上一篇的select模式下的server，让它使用多线程来处理客户端请求（多进程的模式已经在上篇中加了注释）。

**思路：**

  （1）服务器

          我们已经在之前的客户端模型多个并发用户的过程中使用过多线程的技术了（其中还涉及到多线程利用条件变量进行线程同步），在这里我们可以很轻松的在上篇文章代码中加入线程部分代码。

//for thread

                   int \*lptr;

                   pthread\_t pid;

                   //for thread

                   for(i=0;i<=maxi;i++)

                   {

                            if( (sockfd=client[i]) <0)

                            continue;

                            if(FD\_ISSET(sockfd,&rset))

                            {

                                     //thread client

                                     lptr=(int\*)malloc(sizeof(int));

                                     \*lptr=sockfd;

                                     int errerr=pthread\_create(&pid,NULL,threadPerClient,lptr);

                                     if(errerr!=0)

                                     printf("err %s",strerror(errno));

                                     FD\_CLR(sockfd,&allset);                                //短连接后关闭socket，服务器发不过来

                                     client[i]=-1;

                                     printf("can read : %d,%d,%d\n",i,sockfd,nready);

                                     if(--nready<=0)

                                     break;

                                     //thread client

                            }

                   }

         但是！！！！真的那么轻松就可以加入吗？ 为什么我不直接在pthread\_create中传入sockfd，而是要再new一个整型数然后赋值给它再传递过去呢？

答案很明显，如果直接使用sockfd的指针，那么在下次sockfd被client[i]赋值时，sockfd的值变了！而这时线程的实际操作的正是这个sockfd（不只因为线程共享变量，同时还因为传递的是指针）！所以我们复制一个新值，让线程自己处理对应的fd。

这样就完了？很明显不会那么简单，让我们看看线程处理函数就知道。

void\* threadPerClient(void \*arg)

{

         int connfd=\*((int\*)arg);

         free(arg);                             //防止内存泄露

         pthread\_detach(pthread\_self());

         printf("client from %d(socket num)\n",connfd);

         str\_echo(connfd);

         close( connfd );

         return NULL;

}

看到注释了吗？“防止内存泄露“！！这是很容易发生的事，在刚才的代码中new一个新对象，那么我们就必须在使用后释放。除此以外我们还注意到这里调用了pthread\_detach函数，因为线程分为joinable和unjoinable两种。

如果线程是joinable状态，当线程函数自己返回退出时或pthread\_exit时都不会释放线程所占用堆栈和线程描述符（总计8K多）。只有当你调用了pthread\_join之后这些资源才会被释放。若是unjoinable状态的线程，这些资源在线程函数退出时或pthread\_exit时自动会被释放。  
    unjoinable属性可以在pthread\_create时指定，或在线程创建后在线程中pthread\_detach自己, 如：pthread\_detach(pthread\_self())，将状态改为unjoinable状态，确保资源的释放。或者将线程置为 joinable,然后适时调用pthread\_join.  
（其实就类似于进程退出时父亲进程的wait）

（2）客户端

         无需修改

**代码：**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include<sys/types.h>  
 2 #include<sys/socket.h>  
 3 #include<strings.h>  
 4 #include<arpa/inet.h>  
 5 #include<unistd.h>  
 6 #include<stdlib.h>  
 7 #include<stdio.h>  
 8 #include<string.h>  
 9 #include<errno.h>  
 10 #include<signal.h>  
 11 #include<sys/wait.h>  
 12 #include<sys/time.h>  
 13 #include<pthread.h>  
 14   
 15 #define LISTEN\_PORT 84  
 16   
 17 //服务器处理  
 18 void str\_echo(int sockfd) // 服务器收到客户端的消息后的响应  
 19 {  
 20 ssize\_t n;  
 21 char line[512];  
 22   
 23 printf("ready to read\n");  
 24   
 25 while( (n=read(sockfd,line,512))>0 ) //阻塞IO版本  
 26 {  
 27 line[n]='\0';  
 28 printf("Client Diary: %s\n",line);  
 29   
 30 char msgBack[512];  
 31 snprintf(msgBack,sizeof(msgBack),"recv: %s\n",line);  
 32 write(sockfd,msgBack,strlen(msgBack));  
 33 bzero(&line,sizeof(line));  
 34 }  
 35 printf("end read\n");  
 36 }  
 37   
 38 //子进程结束的信号处理  
 39 float timeuse;  
 40 void sig\_child(int signo) //父进程对子进程结束的信号处理  
 41 {  
 42 pid\_t pid;  
 43 int stat;  
 44   
 45 struct timeval tpstart,tpend;  
 46 //float timeuse;  
 47 gettimeofday(&tpstart,NULL);  
 48   
 49 while( (pid=waitpid(-1,&stat,WNOHANG))>0)  
 50 printf("child %d terminated\n",pid);  
 51   
 52 gettimeofday(&tpend,NULL);  
 53   
 54 timeuse+=1000000\*(tpend.tv\_sec-tpstart.tv\_sec)+tpend.tv\_usec-tpstart.tv\_usec;  
 55 //timeuse/=1000000;  
 56 printf("Use Time:%f\n",timeuse/1000000);  
 57 return;  
 58 }  
 59   
 60   
 61   
 62 //使用线程代替进程来处理客户请求  
 63 void\* threadPerClient(void \*arg)  
 64 {  
 65 int connfd=\*((int\*)arg);  
 66 free(arg); //防止内存泄露  
 67 pthread\_detach(pthread\_self());  
 68   
 69 printf("client from %d(socket num)\n",connfd);  
 70 str\_echo(connfd);  
 71   
 72 close( connfd );  
 73 return NULL;  
 74 }  
 75   
 76   
 77 int main(int argc, char \*\*argv)  
 78 {  
 79 int listenfd, connfd;  
 80 pid\_t childpid;  
 81 socklen\_t chilen;  
 82   
 83 struct sockaddr\_in chiaddr,servaddr;  
 84   
 85 //for select  
 86 int i,maxi,maxfd,sockfd;  
 87 int nready,client[FD\_SETSIZE];  
 88 ssize\_t n;  
 89 fd\_set rset,allset;  
 90 //for select  
 91   
 92 listenfd=socket(AF\_INET,SOCK\_STREAM,0);  
 93 if(listenfd==-1)  
 94 {  
 95 printf("socket established error: %s\n",(char\*)strerror(errno));   
 96 }  
 97   
 98 bzero(&servaddr,sizeof(servaddr));  
 99 servaddr.sin\_family=AF\_INET;  
100 servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);  
101 servaddr.sin\_port=htons(LISTEN\_PORT);  
102   
103 int bindc=bind(listenfd,(struct sockaddr \*)&servaddr,sizeof(servaddr));  
104 if(bindc==-1)  
105 {  
106 printf("bind error: %s\n",strerror(errno));  
107 }  
108   
109 listen(listenfd,SOMAXCONN); //limit是SOMAXCONN  
110   
111 //initial "select" elements  
112 maxfd=listenfd; //新增listenfd，所以更新当前的最大fd  
113 maxi=-1;  
114 for(i=0;i<FD\_SETSIZE;i++)  
115 client[i] = -1;  
116 FD\_ZERO(&allset);  
117 FD\_SET(listenfd,&allset);   
118 //end initial  
119   
120 int cliconn=0;  
121 signal(SIGCHLD,sig\_child);  
122 for(;;)  
123 {  
124 rset=allset;  
125 nready=select(maxfd+1,&rset,NULL,NULL,NULL); //一开始select监听的是监听口  
126 //如果有timeout设置，那么每次select之前都要再重新设置一下timeout的值  
127 //因为select会修改timeout的值。  
128 if(FD\_ISSET(listenfd,&rset))  
129 {  
130 chilen=sizeof(chiaddr);  
131   
132 connfd=accept(listenfd,(struct sockaddr\*)&chiaddr,&chilen);  
133 //阻塞在accept，直到三次握手成功了才返回  
134 if(connfd==-1)  
135 printf("accept client error: %s\n",strerror(errno));  
136 else   
137 printf("client connected:%d\n",++cliconn);  
138   
139 for(i=0;i<FD\_SETSIZE;i++)  
140 {  
141 if (client[i]<0)  
142 {  
143 client[i]=connfd; //找一个最小的插进入  
144 break;  
145 }  
146 }  
147 if(i==FD\_SETSIZE)  
148 {  
149 printf("too many clients\n");  
150 exit(0);  
151 }  
152 FD\_SET(connfd,&allset); //新加入的描述符，还没判断是否可以或者写，所以后面使用rset而不是allset  
153   
154   
155   
156 if(connfd>maxfd)  
157 maxfd=connfd;  
158 if(i>maxi)  
159 maxi=i;  
160 if(--nready<=0)  
161 continue;  
162 }  
163   
164 //for thread  
165 int \*lptr;  
166 pthread\_t pid;  
167 //for thread  
168   
169 for(i=0;i<=maxi;i++)  
170 {  
171 if( (sockfd=client[i]) <0)  
172 continue;  
173 if(FD\_ISSET(sockfd,&rset))  
174 {  
175 //thread client  
176 lptr=(int\*)malloc(sizeof(int));  
177 \*lptr=sockfd;  
178 //pthread\_t pid;  
179   
180 int errerr=pthread\_create(&pid,NULL,threadPerClient,lptr);  
181 if(errerr!=0)  
182 printf("err %s",strerror(errno));  
183   
184 FD\_CLR(sockfd,&allset); //短连接后关闭socket，服务器发不过来  
185 client[i]=-1;  
186   
187 printf("can read : %d,%d,%d\n",i,sockfd,nready);  
188 if(--nready<=0)  
189 break;  
190 //thread client  
191   
192 }  
193 }  
194   
195 }  
196 }