Linux网络编程：9. 服务器模型

学习过《软件工程》吧？软件工程可是每一个程序员“必修”的课程啊。如果你没有学习过，建议你去看一看。在这一章里面，我们一起来从软件工程的角度学习网络编程的思想。在我们写程序之前，我们都应该从软件工程的角度规划好我们的软件，这样我们开发软件的效率才会高。在网络程序里面，一般的来说都是许多客户机对应一个服务器。为了处理客户机的请求，对服务端的程序就提出了特殊的要求。我们学习一下目前最常用的服务器模型。

循环服务器：循环服务器在同一个时刻只可以响应一个客户端的请求。
并发服务器：并发服务器在同一个时刻可以响应多个客户端的请求。

9.1 循环服务器：UDP服务器

UDP循环服务器的实现非常简单：UDP服务器每次从套接字上读取一个客户端的请求，处理，然后将结果返回给客户机。

可以用下面的算法来实现：

```
socket(...);
bind(...);
while(1)
{
    recvfrom(...);
    process(...);
    sendto(...);
}
```

因为UDP是非面向连接的，没有一个客户端可以老是占住服务端。只要处理过程不是死循环，服务器对于每一个客户机的请求总是能够满足。

9.2 循环服务器：TCP服务器

TCP循环服务器的实现也不难：TCP服务器接受一个客户端的连接，然后处理，完成了这个客户的所有请求后，断开连接。

算法如下：

```
socket(...);
bind(...);
listen(...);
while(1)
{
    accept(...);
    while(1)
    {
      read(...);
      process(...);
      write(...);
    }
    close(...);
}
```

TCP循环服务器一次只能处理一个客户端的请求。只有在这个客户的所有请求都满足后，服务器才可以继续后面的请求。这样如果有一个客户端占住服务器不放时，其它的客户机都不能工作了。因此，TCP服务器一般很少用循环服务器模型的。

9.3 并发服务器：TCP服务器

为了弥补循环TCP服务器的缺陷，人们又想出了并发服务器的模型。并发服务器的思想是每一个客户机的请求并不由服务器直接处理，而是服务器创建一个子进程来处理。

算法如下：

```
socket(...);
```

```
bind(...);
listen(...);
while(1)
{
      accept(...);
      if(fork(..)==0)
      {
       while(1)
       {
          read(...);
          process(...);
          write(...);
       }
       close(...);
       exit(...);
      }
      close(...);
}
```

TCP并发服务器可以解决TCP循环服务器客户机独占服务器的情况。不过也同时带来了一个不小的问题。为了响应客户机的请求，服务器要创建子进程来处理。而创建子进程是一种非常消耗资源的操作。

9.4 并发服务器：多路复用I/O

为了解决创建子进程带来的系统资源消耗，人们又想出了多路复用I/O模型。

首先介绍一个函数 select：

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
        fd_set *except fds, struct timeval *timeout)
void FD_SET(int fd, fd_set *fdset)
void FD_CLR(int fd, fd_set *fdset)
void FD_ZERO(fd_set *fdset)
int FD_ISSET(int fd, fd_set *fdset)
```

一般的来说当我们在向文件读写时，进程有可能在读写出阻塞，直到一定的条件满足。比如我们从一个套接字读数据时，可能缓冲区里面没有数据可读（通信的对方还没有发送数据过来），这个时候我们的读调用就会等待（阻塞）直到有数据可读。如果我们不希望阻塞，我们的一个选择是用 select 系统调用。只要我们设置好 select 的各个参数，那么当文件可以读写的时候 select 回“通知”我们，说可以读写了。

readfds：所有要读的文件文件描述符的集合。
writefds：所有要的写文件文件描述符的集合。
exceptfds：其他的服要向我们通知的文件描述符。
timeout：超时设置。
nfds：所有我们监控的文件描述符中最大的那一个加1。

在我们调用 select 时进程会一直阻塞直到以下的一种情况发生：
1) 有文件可以读；
2) 有文件可以写；
3) 超时所设置的时间到。

为了设置文件描述符我们要使用几个宏。
FD_SET：将 fd 加入到 fdset。
FD_CLR：将 fd 从 fdset 里面清除。
FD_ZERO：从 fdset 中清除所有的文件描述符。
FD_ISSET：判断 fd 是否在 fdset 集合中。

使用 select 的一个例子：

```
int use_select(int *readfd, int n)
```

```
{
  fd_set my_readfd;
  int maxfd;
  int i;

  maxfd=readfd[0];
  for(i=1;i<n;i++)
     if(readfd[i]>maxfd) maxfd=readfd[i];
  while(1)
  {
     /* 将所有的文件描述符加入 */
     FD_ZERO(&my_readfd);
     for(i=0;ilt;n;i++)
        FD_SET(readfd[i],*my_readfd);
     /* 进程阻塞 */
     select(maxfd+1,& my_readfd,NULL,NULL,NULL);
     /* 有东西可以读了 */
     for(i=0;ilt;n;i++)
        if(FD_ISSET(readfd[i],&my_readfd))
        {
           /* 原来是我可以读了 */
           we_read(readfd[i]);
        }
  }
}
```

使用select后我们的服务器程序就变成了：

```
初始化(socket,bind,listen);
while(1)
{
   设置监听读写文件描述符(FD_*);
   调用select;
   如果是倾听套接字就绪，说明一个新的连接请求建立
   {
      建立连接(accept);
      加入到监听文件描述符中去;
   }
   否则说明是一个已经连接过的描述符
   {
      进行操作(read或者write);
   }
}
```

多路复用I/O可以解决资源限制的问题。着模型实际上是将UDP循环模型用在了TCP上面。这也就带来了一些问题。如由于服务器依次处理客户的请求，所以可能会导致有的客户会等待很久。

9.5 并发服务器：UDP服务器

人们把并发的概念用于UDP就得到了并发UDP服务器模型。并发UDP服务器模型其实是简单的。和并发的TCP服务器模型一样是创建一个子进程来处理的算法和并发的TCP模型一样。

除非服务器在处理客户端的请求所用的时间比较长以外，人们实际上很少用这种模型。

9.6 一个并发TCP服务器实例

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>
```

```
#define MY_PORT 8888

int main(int argc, char **argv)
{
  int listen_fd, accept_fd;
  struct sockaddr_in client_addr;
  int n;

  if((listen_fd=socket(AF_INET, SOCK_STREAM, 0))<0)
  {
      printf("Socket Error:%s\n\a",strerror(errno));
      exit(1);
  }

  bzero(&client_addr,sizeof(struct sockaddr_in));
  client_addr.sin_family=AF_INET;
  client_addr.sin_port=htons(MY_PORT);
  client_addr.sin_addr.s_addr=htonl(INADDR_ANY);
  n=1;
  /* 如果服务器终止后,服务器可以第二次快速启动而不用等待一段时间 */
  setsockopt(listen_fd,SOL_SOCKET,SO_REUSEADDR,&n,sizeof(int));
  if(bind(listen_fd,(struct sockaddr *)&client_addr,sizeof(client_addr))<0)
  {
      printf("Bind Error:%s\n\a",strerror(errno));
      exit(1);
  }
  listen(listen_fd,5);
  while(1)
  {
      accept_fd=accept(listen_fd,NULL,NULL);
      if((accept_fd<0)&&(errno==EINTR))
          continue;
      else if(accept_fd<0)
      {
          printf("Accept Error:%s\n\a",strerror(errno));
          continue;
      }
      if((n=fork())==0)
      {
          /* 子进程处理客户端的连接 */
          char buffer[1024];

          close(listen_fd);
          n=read(accept_fd,buffer,1024);
          write(accept_fd,buffer,n);
          close(accept_fd);
          exit(0);
      }
      else if(n<0)
          printf("Fork Error:%s\n\a",strerror(errno));
      close(accept_fd);
  }
}
```

你可以用我们前面写客户端程序来调试着程序，或者是用来telnet调试。