TCP并发程序设计和C/S结构程序的开发

71118415 叶宏庭

东南大学软件学院

Email: 213182964@seu.edu.cn

May 15, 2021

1 实验目的

掌握TCP并发程序设计和C/S结构程序的开发，具体来说就是针对date程序，实现多进程版的并发程序，完成同样的date程序功能。

2 实验环境

2.1 操作系统:

Ubuntu 20.04

2.2 辅助软件:

CTEX(用于编写tex报告)

3 实验内容

3.1 阅读date源程序:

在进行修改前，先完整阅读date源程序，了解socket的工作原理，通信机制。（详细代码请见附带code文件夹）

3.1.1 客户端程序datetimec.c

指定服务器的地址与端口号:

```
memset( &servaddr , 0 , sizeof( servaddr ) );
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons( 13 );
```

与服务器建立连接:

```
if( connect( sockfd , (struct sockaddr *)&servaddr , sizeof( servaddr ) ) < 0 )
{
        printf( "connect error\n" );
        exit( 1 );
}
```

读取服务器发来的内容：

```
while( ( n = read( sockfd , recvline , MAXLINE ) ) > 0 )  {
                recvline[ n ] = 0;
                if( fputs( recvline , stdout ) == EOF ) {
                        printf( "fputs error\n" );
                        exit( 1 );
                }
        }
```

3.1.2 服务端程序datetimes.c

监听指定端口：

```
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl( INADDR_ANY );
servaddr.sin_port = htons( 13 );

bind( listenfd , (struct sockaddr *)&servaddr , sizeof( servaddr ) );
listen( listenfd , 1024 );
```

处理客户端请求：

```
for( ; ; )
{
    connfd = accept( listenfd , (struct sockaddr *)NULL , NULL );
    ticks = time( NULL );
    snprintf( buff , sizeof( buff ) , "%.24s\r\n" , ctime( &ticks ) );
    write( connfd , buff , strlen( buff ) );
    close( connfd );
}
```

4 实验结果与分析

4.1 修改后的并发程序:

修改后的date程序，完成多并发要求。（详细代码请见附带code文件夹）

4.1.1 客户端程序:

因为并发是针对服务端的设计，所以客户端程序无需修改。

4.1.2 服务端程序:

服务端程序为了实现多并发的要求，所以采用fork()函数来完成具体设计。只需修改原有的处理请求部分代码，其余部分无需修改。

fork函数使用：

```
for( ; ; )
{
    connfd = accept( listenfd , (struct sockaddr *)NULL , NULL );
    if((pid = fork()) == 0){
        close(listenfd);
        time_t ticks;
        time(&ticks );
        snprintf( buff , sizeof( buff ) , "%.24s\r\n" , ctime(&ticks) );
        printf("%s\n", buff);
        write( connfd , buff , strlen( buff ) );
        close( connfd );
        printf("%s\n", "Into sleep!!");
        sleep(2);
        printf("%s\n", "Out sleep and exit!!");
        exit(0);
    }
    close(connfd);
}
```

首先采用fork()函数调用，来创建子进程完成请求的处理任务，在子进程中关闭监听，获取服务器的系统时间，采用write()函数完成系统时间的网络传输。

为了体现本程序的并发性，这里采用了sleep() 函数来实现并发的可视化。在一个进程结束任务处理后，我们让进程休眠2s后再退出，再加上两句printf 的输出，我们就能明显看到并发程序的运行效果。（如下图所示）

```
[root@ocp net_program]# ./server
Sat May 15 19:44:12 2021

Into sleep!!
Sat May 15 19:44:13 2021

Into sleep!!
Out sleep and exit!!
Out sleep and exit!!
```

从输出的结果中我们可以看出，在第一个进程还在sleep中时，已经有别的进程为第二个用户处理了请求，最两个进程也是先后从sleep状态中醒来并且退出。

从结果中，我们可以看出，这个程序完整的实现了TCP的并发设计。