《网络程序设计》实践课

实验报告

实验名称：端口扫描实验

实验类型：验证型

指导教师：贾浩

专业班级：

姓　　名：

学　　号：

实验评分

序号	实验分数构成	分数
1	实验结果(20 分)	
2	实验过程(40 分) （包括实验指导教师提问回答情况）	
3	实验报告书写规范（20 分）	
4	实验思考（20 分）	
实验总分		

一、实验目的

* 了解端口扫描的基本概念和工作原理

二、实验设计

开发环境：Visual Studio 2022
1、编写一个利用全连接的端口扫描程序，能显示目标主机的端口开放情况。
2、要求能在命令行输入要扫描的目标主机和端口范围。比如：scan *.*.*.* nnnn-mmmm。

三、实验过程（包含实验结果）

TCP 三次握手由 connect 函数完成，我们只需要遍历端口号进行连接即可

```
// 扫描端口
for (i = startPort; i <= endPort; i++) {
    targetAddr.sin_port = htons(i);
    if (connect(sock, (struct sockaddr*)&targetAddr, sizeof(targetAddr)) == 0) {
        printf("端口 %d: 开放\n", i);
        closesocket(sock);
        sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    }
}
```

打开命令提示符（CMD），导航到生成的可执行文件所在的目录运行，输入 CNetinterfaceInfo.exe 127.0.0.1 1-1000，扫描 1-1000 端口。

```
E:\作业\网络编程\C\CNetinterfaceInfo\Debug>CNetinterfaceInfo.exe 127.0.0.1 1-1000
扫描 127.0.0.1 的端口 1-1000...
端口 22: 开放
端口 80: 开放
端口 135: 开放
```

我们可以验证一下，使用 netstat -ano 验证对比（由于全连接扫描速度很慢，200 端口左右就停止运行了），验证发现正确。

```
PS C:\Users\pyp> netstat -ano

活动连接

  协议  本地地址          外部地址        状态           PID
  TCP    0.0.0.0:22             0.0.0.0:0              LISTENING       7380
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       7380
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1960
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
```

四、讨论与分析（实验思考中问题理解）

问题 1 扫描特别慢

```
E:\作业\网络编程\C\CNetinterfaceInfo\Debug>CNetinterfaceInfo.exe 127.0.0.1 1-1000
扫描 127.0.0.1 的端口 1-1000...
端口 22: 开放
```

解决方案： 扫描速度较慢，因为需要完成完整的 TCP 三次握手，后面实验思考 2 进行了改进。

实验思考 1：阐述全连接扫描的原理。

通过完成 TCP 协议的三次握手过程来实现。

扫描程序向目的端口发送 SYN 包，如果目标端口开放，目标主机会回应一个 SYN+ACK 包。扫描程序收到 SYN+ACK 包后，发送一个 ACK 包，完成三次握手。如果三次握手成功，说明目标端口是开放的。如果目标端口未开放，目标主机会发送一个 RST 包，表示连接被拒绝。

实验思考 2：你的程序是否考虑了扫描效率？如没有考虑你准备如何改进？

程序逐个端口进行扫描，每个端口都需要完成三次握手导致扫描速度慢，且如果目标端口未开放，connect 函数会等待较长时间才返回失败，扫描效率低。

可以使用**多线程进行扫描**

```
for (int i = 0; i < MAX_THREADS; i++) {
    int currentStart = startPort + i * portsPerThread;
    int currentEnd = (i == MAX_THREADS - 1) ? endPort : currentStart + portsPerThread - 1;
    struct ScanParams* params = (struct ScanParams*)malloc(sizeof(struct ScanParams));
    params->targetIP = targetIP;
    params->startPort = currentStart;
    params->endPort = currentEnd;

    _beginthread(scanPortRange, 0, params);
}
```

```
E:\作业\网络编程\C\CNetinterfaceInfo\Debug>CNetinterfaceInfo.exe 127.0.0.1 1-1000
扫描 127.0.0.1 的端口 1-1000...
请按任意键继续. . . 端口 22: 开放
端口 445: 开放
端口 135: 开放
端口 80: 开放
```

五、实验者自评（从实验设计、实验过程、对实验知识点的理解上给出客观公正的自我评价）

通过本次实验，我对全连接扫描的原理有了清晰的理解，明白了其是通过完成 TCP 三次握手来判断端口是否开放。在面对扫描效率低的问题时，我能分析出原因并提出使用多线程进行改进的思路，这表明我对实验知识点有一定的掌握和应用能力。但在实际编程中，还需要进一步将理论知识转化为实际的优化代码，提高自己的编程实践能力。

六、 附录：关键代码（给出适当注释，可读性高）

```
void scanPortRange(void* param) {
    struct ScanParams* params = (struct ScanParams*)param;
    WSADATA wsaData;
    SOCKET sock;
    struct sockaddr_in targetAddr;
    WSAStartup(MAKEWORD(2, 2), &wsaData);

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock == INVALID_SOCKET) {
        printf("socket() failed: %d\n", WSAGetLastError());
        WSACleanup();
        return;
    }

    targetAddr.sin_family = AF_INET;
    targetAddr.sin_addr.s_addr = inet_addr(params->targetIP);

    for (int i = params->startPort; i <= params->endPort; i++) {
        targetAddr.sin_port = htons(i);
        if (connect(sock, (struct sockaddr*)&targetAddr, sizeof(targetAddr)) == 0) {
            printf("端口 %d: 开放\n", i);
            closesocket(sock);
            sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        }
    }

    closesocket(sock);
    WSACleanup();
    free(params);
    _endthread();
}

int main(int argc, char* argv[]) {
    int portsPerThread = totalPorts / MAX_THREADS;
    int remainder = totalPorts % MAX_THREADS;

    for (int i = 0; i < MAX_THREADS; i++) {
        int currentStart = startPort + i * portsPerThread;
        int currentEnd = (i == MAX_THREADS - 1) ? endPort : currentStart + portsPerThread -
1;
        struct ScanParams* params = (struct ScanParams*)malloc(sizeof(struct ScanParams));
        _beginthread(scanPortRange, 0, params);
```

```
    }

    // 等待所有线程完成
    system("pause");
    return 0;
}
```