四 川 大 学 计 算 机 学 院、软 件 学 院

实　验　报　告

学号：2023141460321 姓名：孙谦昊 专业：计算机科学与技术　班级：行政七班　第　6　周

课程名称	网络编程	实验课时	5-7 节
实验项目	重现 TCP 半连接/全连接队列满场景	实验时间	2025.04.03
实验目的	通过编写客户端/服务端程序，再现 TCP 内核中半连接队列或全连接队列被占满后，新连接请求失败的现象，加深对 TCP 三路握手和队列管理的理解。		
实验环境	Visual Studio Code		

实验内容（算法、程序、步骤和方法）

借助 Python 语言，分别实现客户端进程和服务器端进程。实现的过程中，在客户端增添尝试短时间内快速创建大量连接到服务端的请求。并在最后记录并打印出连接成功和连接失败的数量与比例。

在服务端程序首先明确指定监听端口和 backlog（队列长度）大小，并在 accept 连接后，借助 time.sleep()函数故意延迟或暂停对连接的 accept 调用，从而人为造成全连接队列堆积。

客户端代码为：

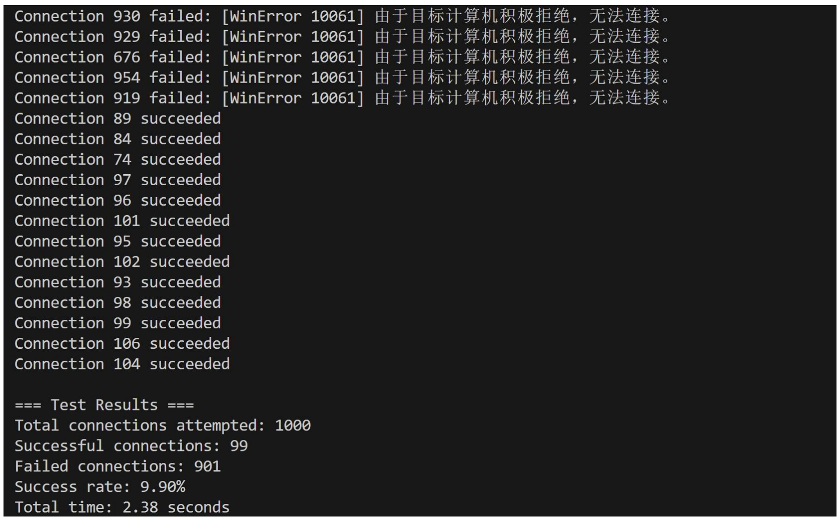
```
import socket
import time
from threading import Thread

host_name = ″127.0.0.1″
port_number = 12001
total_connections = 1000  # 尝试的总连接数
success_count = 0
fail_count = 0

def attempt_connection(connection_id):
    global success_count, fail_count
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.settimeout(1)  # 设置超时时间
        client_socket.connect((host_name, port_number))

        # 简单发送一个消息确认连接正常工作
        client_socket.send(f"Client
{connection_id}".encode())
        response = client_socket.recv(1024).decode()

        success_count += 1
        print(f"Connection {connection_id} succeeded")
        client_socket.close()
    except Exception as e:
        fail_count += 1
        print(f"Connection {connection_id} failed:
{str(e)}")

def main():
    start_time = time.time()
    threads = []

    # 创建并启动所有连接线程
    for i in range(total_connections):
        t = Thread(target=attempt_connection, args=(i,))
        threads.append(t)
        t.start()

    # 等待所有线程完成
    for t in threads:
        t.join()

    # 统计结果
    elapsed_time = time.time() - start_time
    success_rate = (success_count / total_connections) * 100

    print("\n=== Test Results ===")
    print(f"Total connections attempted:
{total_connections}")
    print(f"Successful connections: {success_count}")
    print(f"Failed connections: {fail_count}")
    print(f"Success rate: {success_rate:.2f}%")
    print(f"Total time: {elapsed_time:.2f} seconds")
```

```
if __name__ == ″__main__″:
    main()
```

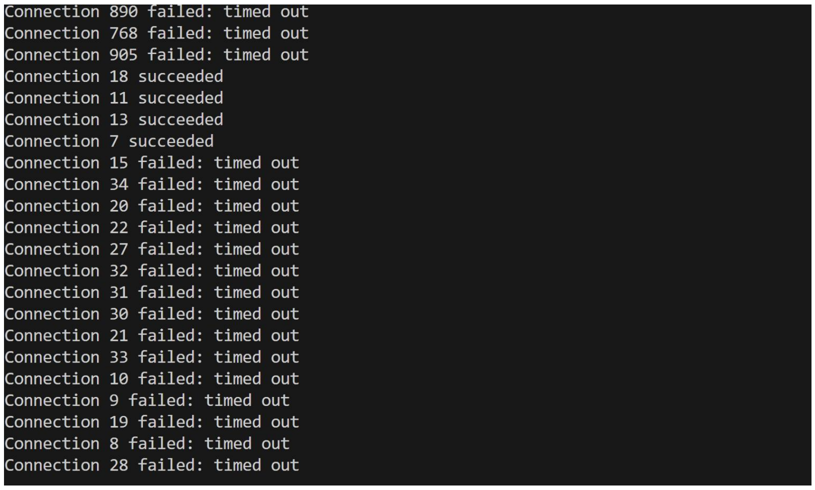
服务器端代码为：

```
import socket
import time

port_number = 12001
listen_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)  # 允许地址重用
listen_socket.bind((’’, port_number))
listen_socket.listen(5)  # 设置 backlog 为 5
print(f″Server is listening on port {port_number} with
backlog 5″)

try:
    while True:
        # 故意延迟 accept 调用
        time.sleep(0.1)  # 延迟 10 秒才接受新连接

        client_socket, addr = listen_socket.accept()
        print(f″Accepted connection from {addr}″)

        msg_from_client =
client_socket.recv(1024).decode()
        client_socket.send(f″Server echoes back:
{msg_from_client}″.encode())
        client_socket.close()
except Exception as e:
    print(e)
finally:
    listen_socket.close()
```

数据记录和计算

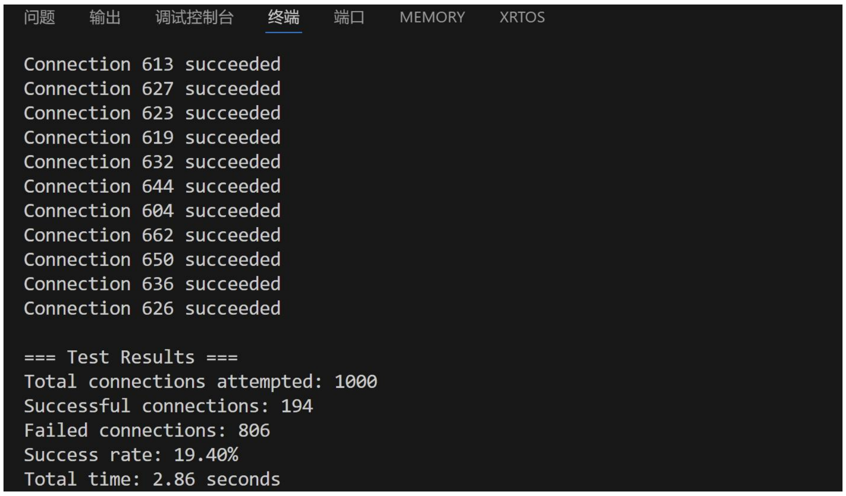


```
Connection 930 failed: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
Connection 929 failed: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
Connection 676 failed: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
Connection 954 failed: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
Connection 919 failed: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
Connection 89 succeeded
Connection 84 succeeded
Connection 74 succeeded
Connection 97 succeeded
Connection 96 succeeded
Connection 101 succeeded
Connection 95 succeeded
Connection 102 succeeded
Connection 93 succeeded
Connection 98 succeeded
Connection 99 succeeded
Connection 106 succeeded
Connection 104 succeeded

=== Test Results ===
Total connections attempted: 1000
Successful connections: 99
Failed connections: 901
Success rate: 9.90%
Total time: 2.38 seconds
```



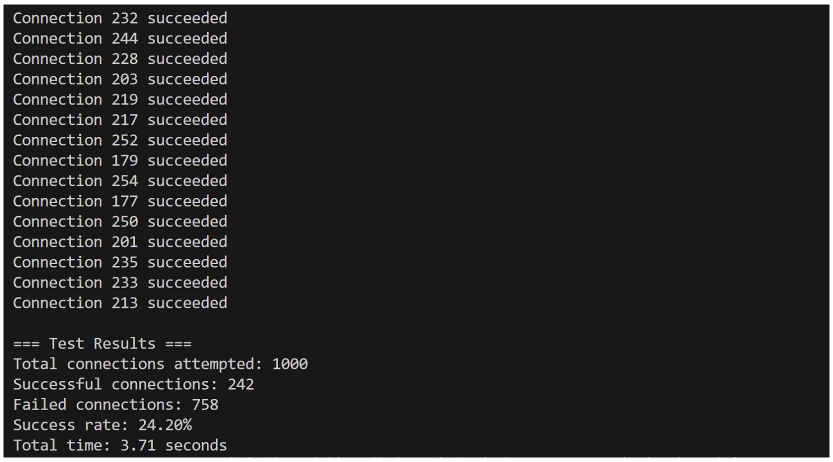
```
Connection 890 failed: timed out
Connection 768 failed: timed out
Connection 905 failed: timed out
Connection 18 succeeded
Connection 11 succeeded
Connection 13 succeeded
Connection 7 succeeded
Connection 15 failed: timed out
Connection 34 failed: timed out
Connection 20 failed: timed out
Connection 22 failed: timed out
Connection 27 failed: timed out
Connection 32 failed: timed out
Connection 31 failed: timed out
Connection 30 failed: timed out
Connection 21 failed: timed out
Connection 33 failed: timed out
Connection 10 failed: timed out
Connection 9 failed: timed out
Connection 19 failed: timed out
Connection 8 failed: timed out
Connection 28 failed: timed out
```

设置客户端超时时间为 5s,每两次请求之间的时间间隔为无限接近于 0s，请求总次数为 1000 次；服务器端 backlog 为 5，accept 调用延迟时间为 0.01s 中的设置下，由上图的结果可知成功完成全连接的请求数为 99 次，未完成全连接的请求数为 901 次，成功率为 9.9%。且可以观察到 backlog 满的情况下服务器端会主动拒绝新的请求访问。



```
Connection 613 succeeded
Connection 627 succeeded
Connection 623 succeeded
Connection 619 succeeded
Connection 632 succeeded
Connection 644 succeeded
Connection 604 succeeded
Connection 662 succeeded
Connection 650 succeeded
Connection 636 succeeded
Connection 626 succeeded

=== Test Results ===
Total connections attempted: 1000
Successful connections: 194
Failed connections: 806
Success rate: 19.40%
Total time: 2.86 seconds
```

设置客户端超时时间为 5s,每两次请求之间的时间间隔为无限接近于 0s，请求总次数为 1000 次；服务器端 backlog 为 50，accept 调用延迟时间为 0.01s 中的设置下，由上图的结果可知成功完成全连接的请求数为 194 次，未完成全连接的请求数为 806 次，成功率为 19.4%。且可以观察到 backlog 满的情况下服务器端会主动拒绝新的请求访问。



```
Connection 232 succeeded
Connection 244 succeeded
Connection 228 succeeded
Connection 203 succeeded
Connection 219 succeeded
Connection 217 succeeded
Connection 252 succeeded
Connection 179 succeeded
Connection 254 succeeded
Connection 177 succeeded
Connection 250 succeeded
Connection 201 succeeded
Connection 235 succeeded
Connection 233 succeeded
Connection 213 succeeded

=== Test Results ===
Total connections attempted: 1000
Successful connections: 242
Failed connections: 758
Success rate: 24.20%
Total time: 3.71 seconds
```

设置客户端超时时间为 5s,每两次请求之间的时间间隔为无限接近于 0s，请求总次数为 1000 次；服务器端 backlog 为 50，accept 调用延迟时间为 0.01s 中的设置下，由上图的结果可知成功完成全连接的请求数为 242 次，未完成全连接的请求数为 758 次，成功率为 24.2%。且可以观察到 backlog 满的情况下服务器端会主动拒绝新的请求访问。

结　论 （结　果）	从实验上述的实验结果中可以看到,服务器的 backlog 大小会直接影响其在高并发场景下的连接处理能力。 在半连接队列满时客户端会因超时显示重传失败（timed out）。

小　　结	通过编写客户端/服务端程序，再现了 TCP 内核中半连接队列或全连接队列被占满后，新连接请求失败的现象，进一步加深对 TCP 三路握手和队列管理的理解。
指导老师评　　议	成绩评定：　　　　　　　　　　　　指导教师签名：