电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

互联网程序设计报告

题目 **基于io_uring的事件处理引擎**

学科专业 计算机科学与技术

学　　号 202221080520

作者姓名 王夏兵

课程老师 聂晓文

目录

第一章 需求分析

1.1 背景及简介

1.1.1 IO模型介绍

（1）基于文件描述符的阻塞式IO：使用Linux提供的基于文件流的系统调用，程序调用函数时会进入sleep状态，知道IO完成；

（2）非阻塞式IO：如select函数、poll函数、epoll函数使用的是非阻塞IO，调用函数时不会阻塞，而是立即返回，得到文件符列表；

1.1.2 io_uring简介

io_uring是Linux内核的一个异步I/O接口以及一种高效的IO框架，可以在Linux内核中使用。它提供了一种新的异步IO模型，可以在不需要额外的内存分配和系统调用的情况下管理IO操作。io_uring是共享内存中的一对环形缓冲器，被用作用户空间和内核之间的队列。io_uring 的基本逻辑与 linux-aio 是类似的：提供两个接口，一个将 I/O 请求提交到内核，一个从内核接收完成事件,这两种队列介绍如下，如图1.1。

（1）提交队列（SQ）：用户空间进程使用提交队列来向内核发送异步I/O请求；

（2）完成队列（CQ）：内核使用完成队列将异步I/O操作的结果发回给用户空间；

这两个队列都是单生产者、单消费者，大小是 2 的幂次提供无锁接口，内部使用 内存屏障做同步。

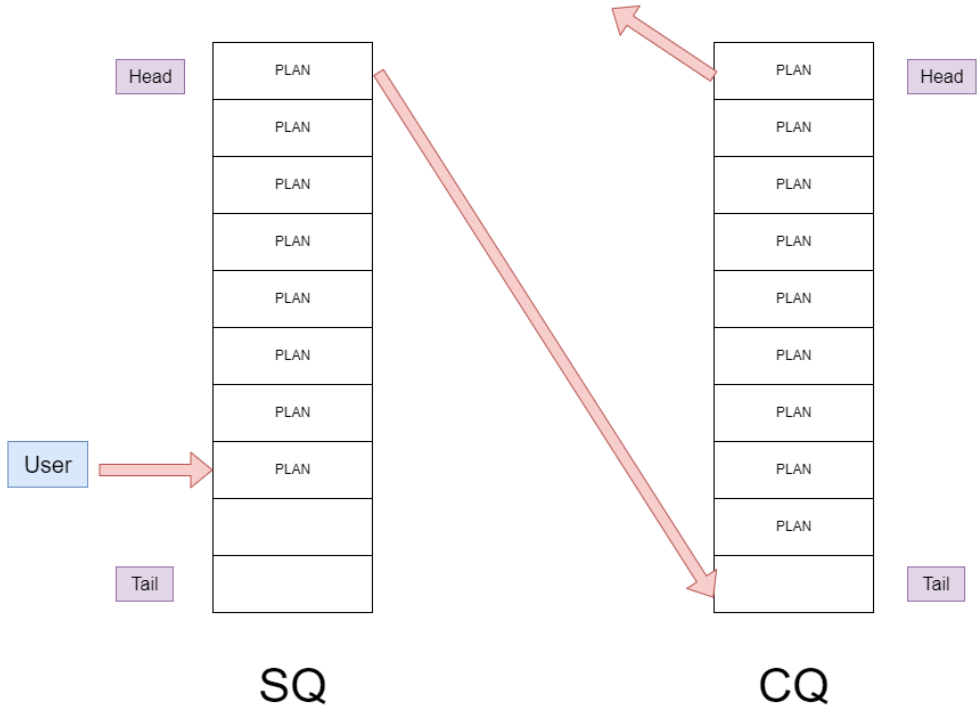

图1.1 io_uring队列

io_uring有着高效率低延迟的优点，其统一了多种IO操作，它支持各种IO操作，包括文件IO操作、网络IO操作等，提高了用户的开发效率，增加了程序的可扩展性。

1.2 主要API

（1）io_uring_submit：提交请求；

（2）io_uring_sqe_set_data：设置sq队列;

（3）io_uring_wait_cqe：等待事件到达;

（4）io_uring_prep_write：执行写回操作;

1.3系统要求

（1） 模仿muduo中epoll的事件处理方案，采用io_uring实现一个事件循环；

（2） 安装liburing，阅读io_uring_setup、io_uring_enter手册；

（3） 采用io_uring实现一个事件分发器，角色相当于epoll；

（4） 实现两个channel，FileChannel、TcpChannel做功能测试、性能测试；

第二章 总体设计

2.1 FileChannel

FileChannel利用io_uring实现了文件的读写功能。该部分实现了文件流操作，能对文件进行基本的读写IO操作，测试代码见第四章。代码框架如图2.1所示。

```
class FileChannel
{
    private:
    io_uring_params m_sInitParams;
    io_uring m_sIoUring;
    int fd;

    public:
    FileChannel(std::string file_path);
    bool read();
    bool write(const std::string data);
    ~FileChannel(){}

};
```

图2.1 FileChannel

2.2 TcpChannel

TcpChannel实现了封装实现了网络通信相关功能。实现了客户端信息的读取，整体结构如图2.2所示。其中，run函数实现了服务端对客服端请求处理的主要逻辑。处理流程如下：

（1） 客户端发送连接建立请求，建立连接；

（2） 服务端在sq队列中初始化一个事件用于处理之后的请求，并将事件设为自定义的read类型；

（3） 客户端发送数据；

（4） 服务端接收数据并将其放入cq队列，等待处理；

（5） 从cq队列取出数据进行处理，并将数据修改为自定义的write类型；

（6） 将处理完的数据提交到sq队列，等待处理；

（7） 数据从sq队列被送入到cq队列，处理数据，向客户端发送回送信息；

（8） 客户端继续等待下一个事件；

```cpp
class TcpChannel
{
    private:
        std::string m_sIp;
        ConnectionData* m_cSocketData;
        ThreadPool m_cThreadPool;
        int m_nPort;
        int m_nSocketFd;
        struct io_uring m_cRing;
        //存储接收的数据
        std::unordered_map<int, ConnectionData *> m_umConnections;

        //初始化信息
        void startAccept();
        void startRead(int client_fd);
        void startWrite(int client_fd);
        //设置回送信息
        void setEchoMessage(ConnectionData *data,const std::string str);
        //处理事件
        void processEvent(io_uring_cqe *cqe);
        void processAccept(int res);
        void processRead(int client_fd, int res);
        void processWrite(int client_fd, int res);
        //回收内存
        void clear(int client_fd);

    public:
        TcpChannel();
        TcpChannel(const std::string ip, const int port, int thread_num);
        void startIouring();
        void run();
        ~TcpChannel();
};
```

图2.2 TcpChannel

整个流程的逻辑如图2.3和图2.4所示。

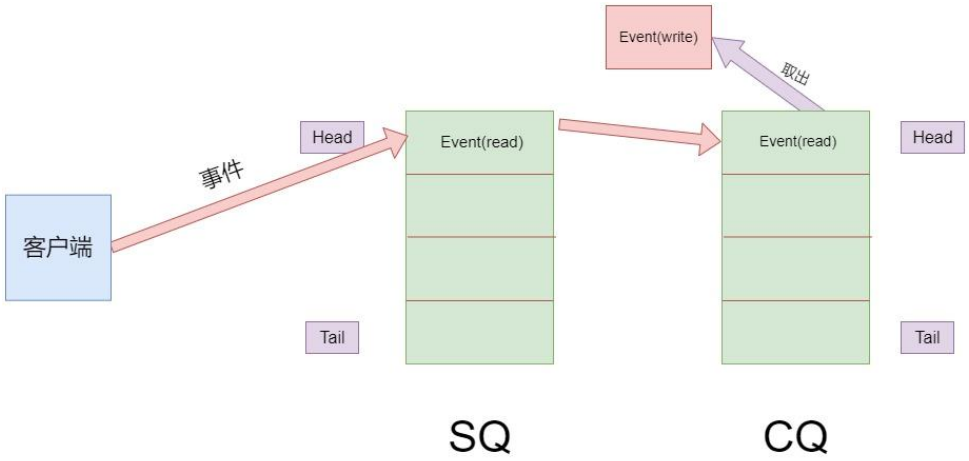

图2.3 处理客户端请求

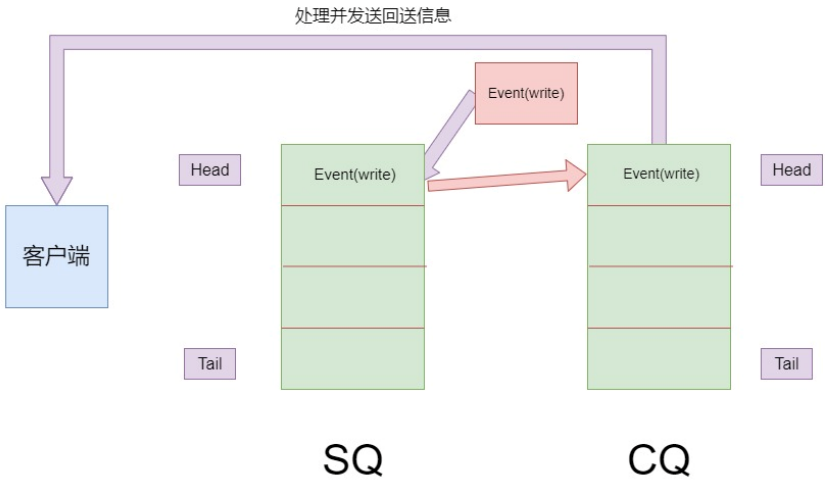
处理并发送回送信息
Event(write)
Head
Event(write)
Event(write)
Head
客户端
Tail
Tail
SQ
CQ

图2.4 处理客户端请求

第三章 详细设计与实现

3.1 主要API介绍

3.1.1 FileChannel主要API

FileChannel类设计如如图2.1，实现主要功能的函数：

（1） read：对文件进行读操作；

（2） write：对文件进行写操作；

使用FileChannel构造对象时，传入文件名作为参数。之后便可以使用io_uring相关API实现文件的零拷贝来进行读写操作

3.1.2 TcpChannel主要API

TcpChannel类设计如图2.2，实现了事件的循环处理机制，主要功能的函数：

（1） startAccept：初始化队列信息；

（2） processEvent：处理不同的事件；

（3） setEchoMessage：设置发送给客户端的信息；

（4） clear：回收内存；

（5） run：启动服务端；

通过调用run函数来启动服务端的服务，初始化相关信息，等待客户端发送消息。io_uring是基于事件驱动机制的，当客户端发送事件后，通过调用processEvent进行处理，事件经过了read->write并且服务端向客户端回送请求，一个事件完整的生命周期就结束了，服务端继续等待客户端发来的事件。当有多个客户端请求时，每一个客户端对应一个描述符分别送入队列中进行处理，并且通过不同的文件描述符对不同的客户端进行回复。当客户端断开连接时，调用clear函数，清理为这个客户端接收数据而申请的内存。

第四章 测试

4.1 开发环境

表4-1展示了本系统的开发环境。

表4-1 配置信息

配置项	配置参数
操作系统	Ubuntu 22.04.1 LTS
CPU型号	Intel® Core™ i5-4590 CPU @ 3.30GHz
内存	8G
程序运行环境	C++11

4.2 功能测试

4.2.1 FileChannel测试

测试文件的读写功能，测试代码如图4.1，结果写入文件。

```
void FileTest()
{
    FileChannel fileChannel("./test.txt");
    fileChannel.write("qwertyuisdafsghjkuykuukukuukuozcxxzvvbnmb,");

}
```

图4.1 FileChannel测试

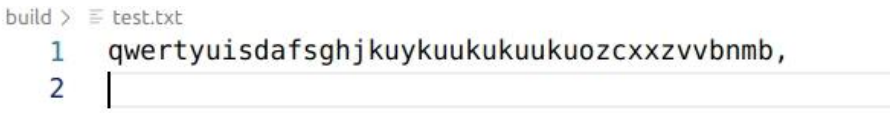

图4.2 FileChannel测试结果

4.2.2 TcpChannel测试

采用telnet进行功能测试。图4.3为服务端测试信息，图4.4为客户端测试信息，服务器IP为本地机器，端口为3737。客户端连续发送了三个信息，服务器接收正常，并正常传输回送数据。

```
wangxiabing@wangxiabing-Ubuntu22:~/conputerprograms/homework/io_uring_channels-master/build$ ./test
socket_fd:3
receive data:asfasdf

receive data:asdfasfasdf

receive data:1122222
```

图4.3 TcpChannel服务端

```
wangxiabing@wangxiabing-Ubuntu22:~/conputerprograms/homework/io_uring_channels-master$ telnet 127.0.0.1 3737
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
asfasdf
received: asfasdf
asdfasfasdf
received: asdfasfasdf
1122222
received: 1122222
```

图4.4 TcpChannel客户端

4.3 压力测试

对TcpChannel进行压力测试，利用第三方测试软件检测资源情况。发送的数据统一为128字节大小，测试结果如表4-2所示。

表4-2 压力测试

线程数	发送数据量	CPU 平均占用率	总耗时(ms)	单数据耗时(ms)
8	8000	11.8%	3	0.00037
8	80000	13.4%	33	0.00041
8	160000	14.1%	63	0.00039
8	320000	21.0%	163.2	0.00051

4.4 实验总结

整个实验完成了实验基本要求，功能测试通过。在压力测试上，测试了多线程的压力情况，但是未使用线程池进行测试，可能会有进一步的测试结果，这也是本次实验的不足之处。

附录

本次实验及实验报告由本人独立完成，具体工程代码详见附件。