Lab3报告

1.运行时间

shm未完成，所以这里只有PIPE和Socket的对比

1.1测试参数为：ANNA_KARENINA.txt cat had her > cyclebin

```
quin@ubuntu:~/projects/qu_lab3$ time ./socketTest.out ANNA_KARENINA.txt cat had her > cyclebin

real    0m34.540s
user    0m0.024s
sys     0m0.024s
quin@ubuntu:~/projects/qu_lab3$ time ./pipeTest.out ANNA_KARENINA.txt cat had her > cyclebin

real    0m47.614s
user    0m0.012s
sys     0m0.044s
```

1.2： 测试参数为：big.txt cat had her > cyclebin

```
quin@ubuntu:~/projects/qu_lab3$ time ./socketTest.out big.txt cat had her > cyclebin

real    3m24.141s
user    0m0.060s
sys     0m0.148s
quin@ubuntu:~/projects/qu_lab3$ time ./pipeTest.out big.txt cat had her > cyclebin

real    4m2.643s
user    0m0.084s
sys     0m0.120s
```

总结：pipe和socket模式都是按行读取，发送给处理函数，可以看出socket模式耗时明显更多一些。这是因为pipe只需要从进行磁盘-内存-CPU的拷贝，而socket(使用INET)还需要通过网卡发送接受数据，有额外的耗时。

2.代码设计

2.0 共有部分

由于实验的三部分很相似，适合ISA的模型，我为他们设计了一个父类。每一个部分，都通过继承这个父类，重写examword()方法来实现。

```
class FileExamine
    {
    public:
        FileExamine() = default;
        FileExamine(string FilePath);
        virtual ~FileExamine() = default;
        virtual void examword(string Word) {};
        static bool checkWord(string Sentence, string Word);
        void readyForNextExam();
    protected:
        ifstream readfrom;
    };
```

生成子类的工厂：

```
FileExamine* create(FileExamineType Type, string FilePath)
        {
            switch (Type) {
            case PIPE:
                return new FileExamine_pipe_impl(FilePath);
                break;
            case SOCKET:
                return new FileExamine_socket_impl(FilePath);
                break;
            case SHM:
                return new FileExamine_shm_impl(FilePath);
                break;
            }
        }
```

2.1 PIPE

```
if (pid == 0)
        {//child
            close(fd[1]);
            while (1)
            {
                //read from pipe , use find() to check if the line contain our intersting word
            }
            _exit(0);
        }
        else if (pid > 0)
        {//father
            close(fd[0]);
            while (!readfrom.eof())
            {
                //write a line of file to child
            }
        }
```

2.2 SOCKET

socket操作较为复杂，容易出错，我封装了Server类和Client类，这两个类只在FileExamine_socket_impl中可见。socke的申请，连接等操作都被封装到类里，这样也更容易避免了忘记关闭连接的错误。

```
if (pid == 0)
        {//child
            server.wait();
            server.checkForWord(Word);
            _exit(0);
```

```
        }
        else if (pid > 0)
        {//father
            Client client = Client();
            client.connectTo(server);
            string aLine;
            while (!readfrom.eof())
            {
                //write a line of file to child
                getline(readfrom, aLine);
                client.writeTo(aLine);
            }
            client.closeCon();
        }
```

2.3 SHM

shm部分未完成。

3.遇到问题

这次实验里我遇到的最复杂的一个问题是这样的：

在SOCKET模式里，我封装了Server类和Client类。在一开始的设计中，我考虑到可能会对一个文件进行多次查询，Server类被设为静态全局的，而每次查询时我们构造一个Client类，连接到服务器。Client的构造和析构非常简单

```
Client() {
    socket_id = socket(**AF_INET**, SOCK_STREAM, 0);
}
~Client() {
    close(socket_id);
}
```

如下的代码会失败，错误为BAD FILE DESCRIPTOR.这个错误在去掉前面定义，但未使用的client后不再出现。我没有查找到相关的资料，基于这个现象，我有一个猜想，我认为这是由于client2先被构造出来，得到一个有效的socket。但马上client1被析构，系统回收client1的socket，并发现client1的socket未被使用，进行了某些特殊的操作导致client2的socket被无效化。而client1的socket如果使用过，就会被正常回收，client2的socket仍然有效。

```
//client1
Client client;
fork()
...
//in father
{
    //client2
    client = Client();
    //the function will try connect() to the server
    client.connectTo(server);
}
```

4.Linux下的IPC机制

Linux提供了大量IPC机制，可以分为六种

1. 管道(pipe & fifp)
2. 信号(signal)
3. 消息队列(Message)
4. 共享内存(SHM)
5. 信号量(semaphore)
6. 套接字(socket)

每种IPC有特定的应用场景，例如当父子进程通信，最简单的方式一般就是管道；信号常用于需要实现软中断机制的时候；SHM可以用于零拷贝，在需要进行大量IO时节约时间；套接字一般用于网络传输，或两个不同的应用间传输信息(例如本地服务应用)；信号量多用于同步机制，说明是否有足够的资源供某进程使用。

5.Map-Reduce模型和Handoop

Map-Reduce模型是用于分布式计算的，同时间可能有多个Mapper在不同的进程/机器上工作，每个在总数据的子集上运行同样的算法，并将中间结果发给Reducer，Reducer收集来自所有Mapper的数据，并将其总结为最终结果。

Handoop是JAVA中的一个分布式系统，提供了一套坚固的底层架构，可以让开发人员快速地开发一套集群系统。Handoop有很多好的特性，高容错，高吞吐量，对开发人员友好。Handoop可能不是最好的分布式框架，但他足够好而且容易搭建，使用。在这个大数据越来越重要的时代，Handoop也越来越流行。