FTP Report

生11 刘云 2021011932

1 Server

1.1 基本结构

Server启动时, 持续存在一个监听进程, 监听指定端口的TCP连接. 当有client连接至server的指定端口后, `fork` 新进程用于处理该client的所有操作. 当该client结束后, 对应的进程也结束.

在每个处理client的主进程中, 通过control socket与client联络. 主进程中维护一个消息循环, 阻塞监听control socket传来的client的指令. 根据每条指令执行不同的server操作, 并通过control socket给予回复. 对于 `RETR` , `STOR` , `LIST` 指令, 会在主进程外额外 `fork` 开辟数据传输进程DTP. Server确保主进程仅维护control socket, DTP仅维护data socket.

在处理需要DTP的指令时, 首先根据 `PORT` 或 `PASV` 建立与client的数据传输TCP连接. 当数据传输的TCP连接建立成功后, 在主进程中 `fork` 子进程专一维护数据传输相关的功能, 如文件读取, 文件发送, 文件写入等. 主进程在此时专一维护control socket. 在本次实现中, 主进程于DTP通过管道相连, 用于处理数据传输过程中control socket传来的 `ABOR` 指令.

本实验实现了数据传输中响应 `ABOR` 指令. 当数据传输中收到 `ABOR` 指令后, 主进程向DTP(数据传输的子进程)发送 `SIGTERM` 消息. DTP中重写了 `SIGTERM` 的响应, 会将开启的socket接口, 文件接口, 管道接口等关闭后结束数据传输, 并结束DTP进程. 主进程会在此时通过control socket向client发送回复, 来完成 `ABOR` 的要求.

部署server时, 需要提前将server IP写入 `server.c` 中.

1.2 实现功能

1.2.1 基本指令

- `USER` , `PASS` , `QUIT` , `SYST` , `TYPE` , `PORT` , `PASV` .
- `RETR` , `STOR` , `LIST` : 通过DTP进程传递, 支持 `ABOR` 和 `QUIT` 中断
- `PWD` : 显示相对root的路径.
- `CWD` : 具有路经检查, 避免进入root外的路径. 在root目录下返回上级目录会停留在root目录并返回client错误.
- `MKD` , `RMD` : 具有路经检查, 避免进入root外的路径.

1.2.2 额外指令

- `ABOR` : 支持 `RETR` , `STOR` , `LIST` 指令传输中的中断.
- `SIZE` : 可以返回指定文件大小. 搭配Linux内建的ftp客户端, 可以实现文件传输进度的显示.
- `REST` : 设定文件传输指针位置. 结合SIZE指令或本地文件属性, 可以实现文件续传.
- `NLST` : 提供NLST格式的文件列表. 搭配Linux内建的ftp客户端, 可以实现 `get` 指令下Tab键的快捷提示.

1.2.3 断点续传

`SIZE` 和 `REST` 的支持可以配合对应的client实现断点续传.

1.2.4 超时检测

在PASV模式下, server会开放特定端口阻塞等待client连接. 如果客户端此时已经断开连接, 则server会陷入卡死. 本实验中实现的server在TCP accept过程中加入超时检测, 当超过1分钟没有客户端连接后, 将退出当前的操作.

Server在主进程阻塞监听client消息时, 也存在超时检测机制. 若超出设定时间没有收到client的指令, 则认为client已经断开连接, server会安全关闭服务该client的进程.

具体的超时时间以宏定义方式记录在 `server.h` 中.

1.2.5 断开检测

Server在阻塞接受消息时, 会监控client的连接状态. 若client连接断开, 则关闭当前所有开启的接口, 并安全关闭若存在的DTP进程.

1.2.6 路径安全检查

在禁止 `../` 结构的路径外, server会通过 `realpath` 函数展开相对路径, 并于root路径比对. 如果超出root的范围, 则禁止该操作.
在 `MKD`, `STOR` 等涉及新建文件的操作, server会首先检查文件名外的路径是否合法, 随后再允许文件的创建.

1.2.7 错误陷阱

- 检查文件是否存在.
- 检查不支持指令.
- 检查指令参数.
- `fork`, `poll`, `pipe`, `stat` 等指令的错误处理, 并发送错误回复.
- 进程结束前关闭开放的socket, file, pipe等接口以及存在的子进程.
- 检查 `REST` 指令生命周期. 若 `REST` 指令后未跟随 `RETR` 或 `STOR`, 以及文件传输后, 剥夺 `REST` 设定.

2 Client

2.1 基本结构

Client为C语言编写的命令行程序. 核心实现思路是, 尽量直接将用户输入的指令直接发送给server, 仅在必要时进行预处理和结果响应.

开启client主进程时, 根据命令行参数指定的IP和port连接. 当连接成功后, client进入消息循环, 阻塞监听用户键盘输入, 并将每一次输入发送给server. 其中 `PORT`, `PASV`, `RETR`, `STOR`, `LIST`, `REST` 等指令需要client进行必要操作, 并针对回复中的code对client进行设置.

其中 `PORT` 需要client监听特定端口. `PASV` 需要记录server IP和端口. `RETR`, `STOR`, `LIST` 需要开启子进程完成与server的连接和数据传输. `REST` 需要记录指针移动位置. 以上指令均需要核对server回复中的code来判断client的设置是否批准.

当执行 `RETR`, `STOR` 和 `LIST` 时, 主进程会开辟DTP用户数据传输. 主进程专一维护control socket, DTP专一维护data socket.
本实验中client实现了数据传输的人工中断. 当执行三个指令时, client主进程监听键盘的 `ctrl+C` 操作. 当主进程监听到事件后, 首先向DTP传递 `SIGTERM` 消息. DTP中 `SIGTERM` 消息响应被重写, 收到消息后会搁置正在执行的数据传输或接收, 并监听与主进程的管道接口. 此时主进程向server发送 `ABOR` 请求. 若 `ABOR` 请求批准, 则主进程通过管道通知DTP安全结束进程. 若 `ABOR` 请求不批准, 则DTP回到中断位置重新接收数据.

2.2 实现功能

2.2.1 基本指令

- `USER`, `PASS`, `QUIT`, `SYST`, `TYPE`, `MKD`, `CWD`, `PWD`, `RMD`: 直接向server发送消息.
- `RETR`, `STOR`, `LIST`: 通过DTP进程传递, 支持用户 `ctrl+C` 中断, 并向server发送 `ABOR` 消息. 另外 `RETR` 和 `STOR` 都默认从client启动所在目录寻找待处理文件, 即两个指令会自动获取路径中的文件名(去除路径)作为待操作的本地文件.
- `PORT`, `PASV`: 监听特定端口或记录相应参数后, 向server发送消息.

2.2.2 额外指令

- `ABOR`: 在收到用户中断操作后, 向server发送 `ABOR` 请求, 并根据回应处理DTP进程.
- `REST`: 设定文件传输指针位置. 结合server端 `SIZE` 指令或本地文件属性, 可以实现文件续传. `REST` 指令后必须跟随 `RETR` 或 `STOR`, 否则 `REST` 指令将会失效. 在 `RETR` 和 `STOR` 后, `REST` 指令也会立刻失效.
- `SIZE`: 获取server中文件大小.
- `get`: 对基本FTP指令的包装, 实现断点下载文件的功能. 指令包装了获取本地文件大小, `SIZE`, `PASV`, `REST`, `RETR` 等基本指令, 可以优先执行断点续传, 若无法执行断点续传, 则正常传输文件, 并实现传输进度条. 指令格式 `get <remote_path> <loacal_path>`.
- `put`: 对基本FTP指令的包装, 实现断点上传文件的功能. 功能与 `get` 类似. 指令格式 `put <loacal_path> <remote_path>`.

2.2.3 断点续传

Client支持基本的FTP指令完成手动的断点续传. 在得知本地文件大小或通过 `SIZE` 指令获取server中文件大小后, 可以通过 `REST` 指令协调client与server的文件指针, 实现 `RETR` 和 `STOR` 的断点续传.

Client也额外实现了 `get` 和 `put` 指令, 提供更加方便的用户接口. `get` 和 `put` 会尝试获取本地文件大小和通过 `SIZE` 获取server对应文件的大小. 若失败, 则直接进入常规的传输. 若成功, 则通过 `REST` 指令尝试初始化断点续传. 若 `REST` 成功, 则进行 `RETR` 或 `STOR` 启动断点续传, 并根据文件大小展示传输进度.

2.2.4 中断操作

Client在数据传输过程中监听用户 `ctrl+C` 操作, 并发送 `ABOR` 指令给server实现传输中断.

2.2.5 超时检测

类似server的处理方式, 在PORT模式下, client会开放特定端口阻塞等待server连接. 当超过1分钟没有server连接后, 将退出当前的操作.

Client在主进程阻塞监听server回复时, 若超出设定时间没有收到server的回复, 则认为server已经断开连接, client会安全关闭程序.

具体的超时时间以宏定义方式记录在 `client.h` 中.

2.2.6 错误陷阱

- 检查回复: 虽然client不直接检查请求格式, 但是client会检查server的回复, 来更新client的状态. `PORT` 中, 若server拒绝, client会拆除已开放的socket. `REST` 中, client会在server同意后, 再调整client文件指针. `SIZE` 中, client会在server同意后, 再记录文件大小, 以防止部分server不支持 `SIZE` 指令. 在自定指令 `get` 和 `put` 中, 会根据server回复, 选择断点重传或普通传输.
- 检查文件是否存在.
- 检查不支持指令.
- 检查端口可用状态.
- 检查client自定的 `get` 和 `put` 参数.
- `fork` , `poll` , `pipe` , `stat` 等指令的错误处理.
- 进程结束前关闭开放的socket, file, pipe等接口以及存在的子进程.
- 检查 `REST` 指令生命周期. 若 `REST` 指令后未跟随 `RETR` 或 `STOR` , 以及文件传输后, 剥夺 `REST` 设定.

2.2.7 接口优化

FTP基本指令中的 `RETR` 和 `STOR` 参数指向的是server中的文件地址. Client支持用户直接输入最基本FTP指令. 在直接使用 `RETR` 和 `STOR` 时, client会自动获取参数的文件名, 并尝试在client当前目录下打开该文件, 上传或收取远程文件. 在代码实现中, `RETR` 和 `STOR` 会直接使用当前打开的本地文件接口. 因此其他用户可以方便的在本代码基础上实现更复杂已用的传输指令. 如本实验中实现的 `get` 和 `put` 可以同时指定本地文件和远程文件.

2.2.8 PORT参数缺省

为简便调试和使用, `PORT` 指令支持缺省参数. 此时client会随机选择尝试端口范围内的端口. 得到可用端口后, client代替用户补全指令中IP和端口, 并发送给server. 此功能需要提前将client IP写入 `client.c` 中.

3 感受

通过本实验我了解了FTP大致框架. 我原本以为FTP最困难的是文件传输等具体的过程, 但实际上FTP关注的是client和server的沟通协调. 我也通过尝试Linux内建的ftp客户端了解到, 常见的客户端会对基本的FTP指令进行封装, 类似于 `RETR` 和 `get` 的关系. 本实验最困难的应该是程序结构的设计, 以及如何管理父子进程, 另外也需要对FTP协议相对熟悉, 例如需要发送几次消息接收几次消息. 错误陷阱设计也比较困难, 因为涉及到client和server的协调问题, 不可以仅处理好自己即可. 最后感谢邹冠升同学, 黎叙锐同学对我本次实验提供的帮助.