大型文件传输项目总结

可实现对超过4G大小的单个文件进行操作

在头文件之前 加入 宏定义(一定要之前)

#define \_LARGEFILE64\_SOURCE

#define \_LARGEFILE\_SOURCE

#define \_FILE\_OFFSET\_BITS 64

Open函数 或上 O\_LARGEFILE标志

Sleek() 函数 改成 sleek64()

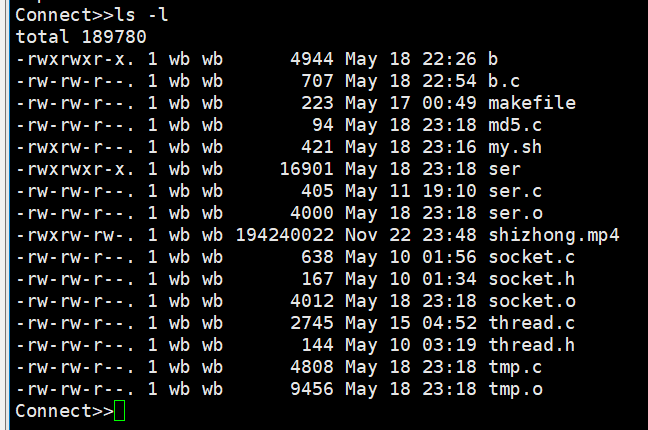
S所有涉及到 size的地方 都将类型改成 long long 类型

基本命令可实现带参数的ls , rm , mv.

Ls 命令 及ls -l







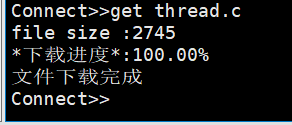
Rm命令:



Mv命令:



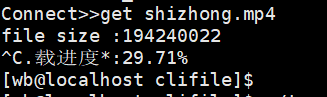
下载 :



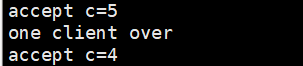
断点续传:

“shizhong.mp4”是一个近200M的视频

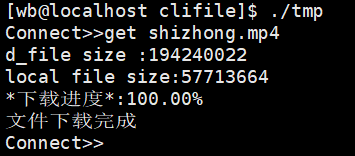
当客户端下载时:



突然中止后 , 服务器显示:

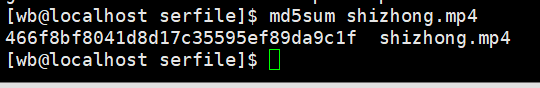


重新连接后继续下载:

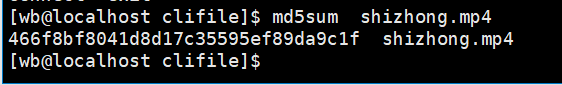


这里显示了当前本地已经下载的文件大小 , 最后根据MD5sum 进行对照 , 发现数值时一致的

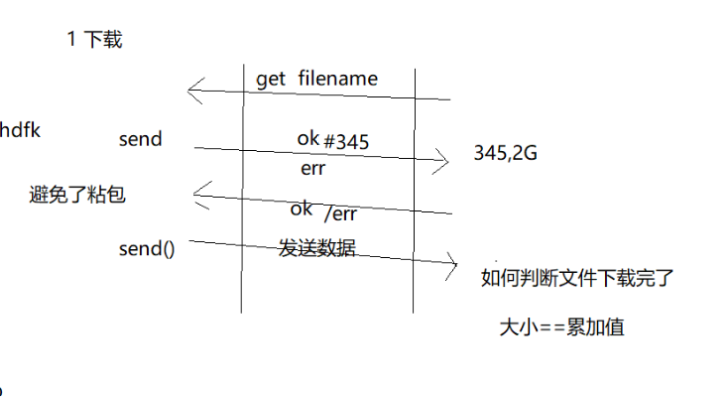
这是服务器端的显示:



这是客户端的显示:



断点续传的基本思路:



服务器端 :

//断点续传

//对第二次收到客户端发送的消息进行甄别

//如果后面含有文件的偏移量 ,

//则说明是上次断开的文件 , 需要进行断点续传

//按照客户端发送的偏移量进行偏移

//然后开始传输数据

客户端 :

//断点续传

//先判断本地是否此有文件

//无同名文件 , 则为普通下载

//有同名文件文件

//判断其大小 是否和服务器端相同

//相同 , 则为普通下载

//不同 , 则说明是上一次下载中断的 , 需要进行断点续传

//第二次发送消息 , 附加文件的偏移量

//将本地文件光标移到最后 , 等待接收服务器端发送的文件

在断点续传中遇到的问题:

//问题:客户端在传输过中 , 中断传输 , 会造成服务器端的终止

//原因: 在send的时候 , 突然中止 , 底层抛出一个SIGPIPE信号 ,

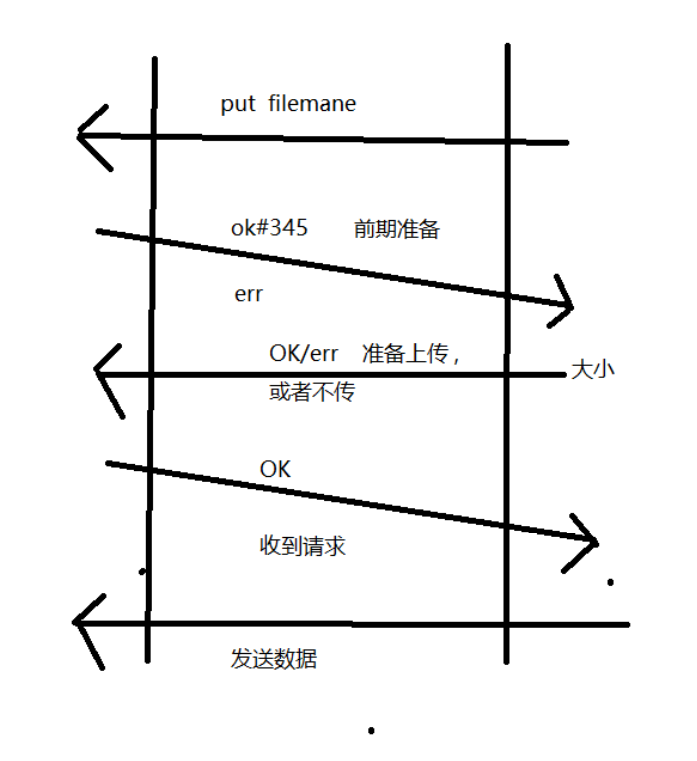
//而SIGPIPE的默认处理方式是终端进程

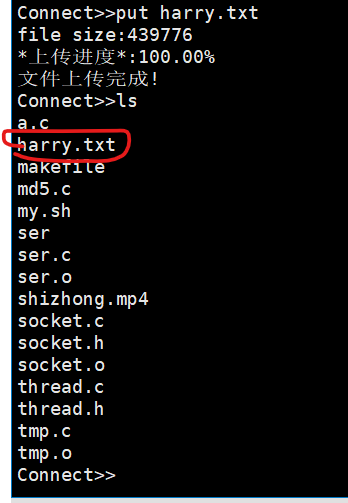
//解决方法 , 使用signal(SIGPIPE,SIG\_IGN); 屏蔽信号

//头文件#include<signal.h>

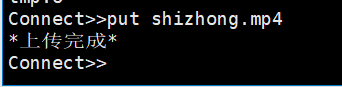
上传:

思路:





秒传:



服务器端:

//秒传

//在第一次收到客户端的消息时 , 如果同名 ,

//则计算文件的MD5值 ,

//然后发送给客户端

//客户端比较本地文件MD5值 ,

//根据客户端返回的结果 ,

//如果是"same#file#" , 则确认秒传

//如果不是 , 则为普通传输

//如果相同 , 客户端将发送一个"same#file#"

客户端:

//秒传

//对服务器发送回的消息进行甄别 ,

//如果含有MD5值 , 则需要计算本地文件的MD5值 ,

//并与服务器端的相比较 ,

//如果不是 , 则为普通上传

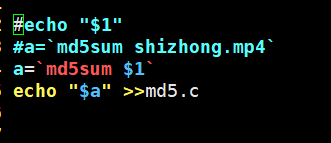
//如果相同 , 则为秒传 ,

//客户端将发送一个"same#file#" , 确认为相同文件

遇到的问题:

MD5值得计算 , 原本是项通过脚本得出计算值 , 返回给代码 , 然后发送到客户端进行比对 , 但在实际实现当中出现了问题 , 在C向脚本传参 , 脚本带参计算MD5 , 使用awk进行分割 , 再将结果返回给C .可能是学的不精 , 使用起来很是费劲 .尝试了好久都不能实现 , 所以决定转换思路 .使用system()函数 , 执行脚本命令 , 将脚本计算出的MD5值写入文件 , C端再读取文件 , 用strtok()进行分割 .这样拿出计算值 发送给客户端 , 让其对照.

以下是my.sh截图



秒传的主要时间就是消耗在MD5的计算上 , 还有文件的打开,读取以及关闭