1. 引言
2. 编写目的:

涉及两部分: 扫描仪网络层 与 UI层间的通信

扫描仪网络层 与 服务器间的通信

1. 协议参考文档:

UI与NET通信协议.pdf

扫描仪与服务器间的TCP通讯协议.pdf

1. 总体设计:
2. 需求规定:
3. UI的需求:

每次发送命令至网络层, 不允许界面发生阻塞; 所以网络层采取异步设计;

1. 与服务器间数据包的需求:
2. 去除每次发送的重包(依据流水号, 只有当接收到的数据包流水号大于等于本地缓存流水号时, 才属于正常数据包);
3. 错误数据包(数据包的长度不正确)
4. 数据包验证码错误后, 需要重新发送该数据包;
5. UI的命令至网络层的要求:
6. 注: 发送命令至网络层(NET层)需注意: 必须等上一条命令返回后, 才可以发送下一条命令.
7. 运行环境:

Ubuntu 12.04 32位操作系统

1. 基本设计概念和处理流程
2. 采用的技术: 多线程开发, 线程间的同步;
3. 涉及六条线程, 一条进程:
   1. 处理命令线程
   2. 发送数据至服务器线程
   3. 从服务器接收数据线程
   4. 心跳线程
   5. 解包线程
   6. 检测超时接收线程
   7. DHCP进程
4. 接口设计:
5. 涉及UI的接口设计

初始化接口: int init\_client\_net(char \* filePath);

param : filePath 网络的配置文件路径 (net.ini)

retval: 0: 成功; -1: 失败;

从UI界面接收命令: int recv\_ui\_data(const char \*news);

param: news : 接收UI命令; 格式 cmd~content~

retval: 0: 成功; -1: 失败;

向UI界面返回处理信息: int send\_ui\_data(char \*\*buf);

param: \*buf : 返回的命令处理信息; 格式 cmd~content~

retval: 0: 成功; -1: 失败;

释放net 服务层分配的内存int free\_memory\_net(char \*buf);

param: buf : 返回处理信息的内存.

retval: 0: 成功; -1: 失败;

销毁net 服务; int destroy\_client\_net();

retval: 0: 成功; -1: 失败;

1. 程序设计思路:

1. 命令内容:

1. 普通业务命令(登录, 登出….)

2. 上传文件(主要是图片)

3. 下载文件(模板和数据库表)

1. 接收普通命令的处理思路:
   * 1. 从recv\_ui\_data()接收UI命令, 立即缓存之队列1, 并通知处理命令线程, 然后该函数返回;
     2. 命令线程接收到信号通知后, 去队列1 中获取命令, 进行命令处理方法选择, 进行组数据包, 然后放入队列2, 缓存至发送数据链表, 然后通知发送数据线程, 同时等待成功处理信号;
     3. 发送数据线程接收到信号通知后,去队列2 中取出发送数据, 进行发送, 成功后, 继续去轮询, 监听信号;
     4. 接收线程接收应答后, 将内容写入队列3 中, 然后通知解包线程, 继续去轮询接收数据;
     5. 解包线程接收到 信号通知后, 去队列3 中取数据, 组成完整的数据包, 然后判断该数据包是否属于正确的数据包, 然后选择该数据包的处理方法, 将接收到的数据包转义后放入队列4中,然后删除发送链表中的发送数据包, 通知主线程给UI进程回复, 通知处理命令线程成功处理;
     6. 主线程接收到信号通知后, 去队列4 中取出数据, 进行处理方法选择, 解析数据包内容, 然后组织返回UI进程的信息, 通过send\_ui\_data()返回给UI;
     7. UI进程处理完网络层信息后, 需要将应答数据内存返回给网络, 调用free\_memory\_net();
     8. 当需要销毁网络服务时, 调用destroy\_client\_net();
2. 上传图片命令的处理:
   * 1. 该命令不同的地方在于: 处理命令线程和解包线程;
     2. 处理命令线程: 接收到上传图片命令后, 按轮次发送数据包, 每轮发送数据包的个数最大为50个, 每轮数据包发送前, 先组装一个测试网络情况的数据包, 放入队列2中, 然后将数据包放入发送链表中, (此数据包不带任何文件数据, 只含有报头信息), 组包方法为: send\_perRounc\_begin();等待该数据包的应答信号, 然后组织本轮发送的数据包, 放入队列2中, 并将这些数据包放入发送链表中, 等待成功处理信号通知;
     3. 解包线程, 接收到信号通知后, 选择上传图片的处理方法, 处理行为: 解析该数据包是否为需要重新发送的数据包, 如果是, 去发送链表中找到该数据包, 重新放入队列2, 发送信号, 通知发送线程; 如果不需要重新发送, 判断该数据包是否需要给UI回应, 如果不需要, 直接丢弃, 发送信号给组包线程, 继续处理接下来的数据包, 如果需要, 将该数据包放入队列4, 删除发送链表中的所有数据包, 通知主线程, 同时通知组包线程退出该命令的处理;
3. 下载文件的命令:
   * 1. 该命令的不同地方在于: 解包线程
     2. 解包线程的处理: 选择该命令的处理方法: 然后判断该数据包是否为最后一包数据, 否, 将该数据包放入队列 4, 通知主线程; 是, 将该数据包放入队列4后, 删除发送链表中的数据, 然后通知主线程;
4. 检测超时接收线程工作方法:
   * 1. 定时轮询,
     2. 判断标识符, 客户端此时是否出错, 出错, 继续下次定时轮询;
     3. 未出错, 判断发送链表中数据的个数: 当为0 时, 继续下次轮询;
     4. 为个数1, 判断 数据包是否超时, 超时, 重新发送, 否, 继续下次轮询;
     5. 为个数n, 判断 最后发送的一包数据是否超时, 超时, 重新发送所有的数据包, 否, 继续下次轮询;
5. 处理命令线程:
   * 1. 接收到信号通知后, 去队列中去命令, 然后选择执行的方法, 进行处理, 组织数据包, 将处理结果通知发送线程(网络命令)并等待成功应答信号通知, 或将结果通知主线程(本地命令); 然后该方法, 继续轮询, 等待信号;
6. 发送数据包线程:
   * 1. 收到信号通知后, 去队列中取出发送数据包, 进行发送, 成功,缓存该发送成功的时间和数据包地址;
7. 接收数据线程:
   * 1. 轮询阻塞套接字去接收数据, 将接收的数据放入队列, 去通知解包线程;
8. 解包线程:
   * 1. 接收到信号通知, 去队列中取出数据, 组成完整数据包, 进行数据包正确性的判断(去除重包, 错包, 对于校验码错的数据包, 需要重新发送该请求), 然后选择该数据包所对应的执行方法, 成功执行后, 需要通知主线程的数据包, 将内容放入队列, 去通知主线程, 否则, 退出该方法, 继续轮询等待信号通知;
9. 主线程: 主要涉及两个方法: int recv\_ui\_data(const char \*news); int send\_ui\_data(char \*\*buf):
   * 1. recv\_ui\_data(); 该方法负责从UI进程接收命令, 进行处理;
     2. send\_ui\_data(); 该方法负责向UI进程返回处理命令的结果, UI进程只需去轮询该进程, 即可;
10. DHCP进程, 只需初始化后, 就不需要去关注了, 会自动去申请IP地址;