- WinMain是WFreeRDP-Client的入口, 主要进行了4个主要操作逻辑

1. 初始化 RDP_CLIENT_ENTRY_POINTS 对象, 此对象主要绑定了客户端启动关闭等回调函数。

```
ZeroMemory(&clientEntryPoints, sizeof(RDP_CLIENT_ENTRY_POINTS));
clientEntryPoints.Size = sizeof(RDP_CLIENT_ENTRY_POINTS);
clientEntryPoints.Version = RDP_CLIENT_INTERFACE_VERSION;
RdpClientEntry(&clientEntryPoints);
```

2. 初始化客户端需要的环境对象, 主要是rdp对象

```
context = freerdp_client_context_new(&clientEntryPoints);
```

3. 解析cmd命令,给setting对象各个成员赋值

```
    int size = WideCharToMultiByte(CP_UTF8, 0, args[i], -1, NULL, 0, NULL, NULL);
    argv[i] = calloc(size, sizeof(char));

    if (!argv[i])
        goto out;

    if (WideCharToMultiByte(CP_UTF8, 0, args[i], -1, argv[i], size, NULL,
                            NULL) != size)
        goto out;
}

settings = context->settings;
wfc = (wfContext*) context;

if (!settings || !wfc)
    goto out;

status = freerdp_client_settings_parse_command_line(settings, argc, argv,
         FALSE);

if (status)
{
    freerdp_client_settings_command_line_status_print(settings, status, argc, argv);
    goto out;
}
```

4. 启动客户端, 并等待客户端关闭

```
if (freerdp_client_start(context) != 0)
    goto out;

thread = freerdp_client_get_thread(context);

if (thread)
{
    if (WaitForSingleObject(thread, INFINITE) == WAIT_OBJECT_0)
    {
        GetExitCodeThread(thread, &dwExitCode);
        ret = dwExitCode;
    }
}

if (freerdp_client_stop(context) != 0)
    goto out;
```

- RdpClientEntry(...)主要绑定了一些回调, 如客户端启动回调, 客户端停止回调, 连接前回调, 链接后回调等。这些回调。在 instance->PostConnect = wf_post_connect;中绑定了特别多的回调, 这些是用来连接恒功以后更新图像用的。

wfreerdp_client_new主要绑定了连接前回调,连接后回调, 用户账号密码输入回调, 其中连接前回调主要做了两件事, 为setting赋值, 主要参数是屏幕分辨率和初始化channels绑定channels相关的各个阶段的回调

```
int RdpClientEntry(RDP_CLIENT_ENTRY_POINTS* pEntryPoints)
{
    pEntryPoints->Version = 1;
    pEntryPoints->Size = sizeof(RDP_CLIENT_ENTRY_POINTS_V1);
    pEntryPoints->GlobalInit = wfreerdp_client_global_init;
    pEntryPoints->GlobalUninit = wfreerdp_client_global_uninit;
    pEntryPoints->ContextSize = sizeof(wfContext);
    pEntryPoints->ClientNew = wfreerdp_client_new;
    pEntryPoints->ClientFree = wfreerdp_client_free;
    pEntryPoints->ClientStart = wfreerdp_client_start;
    pEntryPoints->ClientStop = wfreerdp_client_stop;
    return 0;
}
```

- wfreerdp_client_start主要做了两件事

```
    wfc->wndClass.lpfnWndProc = wf_event_proc;
    wfc->wndClass.cbClsExtra = 0;
    wfc->wndClass.cbWndExtra = 0;
    wfc->wndClass.hCursor = wfc->cursor;
    wfc->wndClass.hbrBackground = (HBRUSH) GetStockObject(BLACK_BRUSH);
    wfc->wndClass.lpszMenuName = NULL;
    wfc->wndClass.lpszClassName = wfc->wndClassName;
    wfc->wndClass.hInstance = hInstance;
    wfc->wndClass.hIcon = wfc->icon;
    wfc->wndClass.hIconSm = wfc->icon;
    RegisterClassEx(&(wfc->wndClass));
    wfc->keyboardThread = CreateThread(NULL, 0, wf_keyboard_thread, (void*) wfc, 0,
                                       &wfc->keyboardThreadId);

    if (!wfc->keyboardThread)
        return -1;

    wfc->thread = CreateThread(NULL, 0, wf_client_thread, (void*) instance, 0,
                               &wfc->mainThreadId);

    if (!wfc->thread)
        return -1;
```

1. 创建keyboard线程, 这个线程会将截获用户的键盘输入, 发送到服务器, 注意是通过wf_ll_kbd_proc()函数处理的。

```
    HHOOK hook_handle;
    wfc = (wfContext*) lpParam;
    assert(NULL != wfc);
    hook_handle = SetWindowsHookEx(WH_KEYBOARD_LL, wf_ll_kbd_proc, wfc->hInstance,
                                   0);

    if (hook_handle)
    {
```

```
if (wParam == WM_KEYDOWN)
{
    DEBUG_KBD("Pause, sent as Ctrl+NumLock");
    freerdp_input_send_keyboard_event_ex(input, TRUE, RDP_SCANCODE_LCONTROL);
    freerdp_input_send_keyboard_event_ex(input, TRUE, RDP_SCANCODE_NUMLOCK);
    freerdp_input_send_keyboard_event_ex(input, FALSE, RDP_SCANCODE_LCONTROL);
    freerdp_input_send_keyboard_event_ex(input, FALSE, RDP_SCANCODE_NUMLOCK);
}
```

2. 创建client线程这个线程是目前最重要的线程, 主要链接了FeeRDP服务器

```
if (!freerdp_connect(instance))
    goto end;
```

创建线程, 处理来自服务器的输入事件, 包括鼠标键盘还有焦点变换等

```
if (async_input)
{
    if (!(input_thread = CreateThread(NULL, 0, wf_input_thread,
                                      instance, 0, NULL)))
    {
        WLog_ERR(TAG, "Failed to create async input thread.");
        goto disconnect;
    }
}
```

循环中不断发送客户端的焦点到服务器

```
nCount = 0;

if (freerdp_focus_required(instance))
{
    wf_event_focus_in(wfc);
    wf_event_focus_in(wfc);
}
```

从消息队列中获取windows事件,并分发。这里并没有找到消息的来源。

```
    {
        DWORD tmp = freerdp_get_event_handles(context, &handles[nCount], 64 - nCount);

        if (tmp == 0)
        {
            WLog_ERR(TAG, "freerdp_get_event_handles failed");
            break;
        }

        nCount += tmp;
    }

    if (MsgWaitForMultipleObjects(nCount, handles, FALSE, 1000,
                                  QS_ALLINPUT) == WAIT_FAILED)
    {
        WLog_ERR(TAG, "wfreerdp_run: WaitForMultipleObjects failed: 0x%08lX",
                 GetLastError());
        break;
    }
```

- freerdp_connect相当复杂的一个函数, 主要处理FreeRDP的链接操作。

1. 调用PreConnect回调, 用来设置,setting主要获取了本机的屏幕分辨率。

```
IFCALLRET(instance->PreConnect, status, instance);
instance->ConnectionCallbackState = CLIENT_STATE_PRECONNECT_PASSED;
```

2. 初始化channels
3. 设置键盘布局
4. 调用真正的链接函数

5. 判断是否使用RemoteFx,并执行相关逻辑

6. 绑定pointer相关的回调, 绑定到update对象之中

```
if (status)
    status2 = freerdp_channels_pre_connect(instance->context->channels,
                                           instance);
```