主要做了三件事, 将context妆化为窗口context, 启动键盘截获线程, 启动客户端线程。看似简单实则复杂的很, 尤其是客户端线程。

1.将context妆化为窗口context, 标识没看懂是怎么转化的, 猜测可能直接申请了wfContext这么大的内存。然后给窗口context赋值。

```
HWND hWndParent;
HINSTANCE hInstance;
wfContext* wfc = (wfContext*) context;
freerdp* instance = context->instance;
hInstance = GetModuleHandle(NULL);
hWndParent = (HWND) instance->settings->ParentWindowId;
instance->settings->EmbeddedWindow = (hWndParent) ? TRUE : FALSE;
wfc->hWndParent = hWndParent;
wfc->hInstance = hInstance;
wfc->cursor = LoadCursor(NULL, IDC_ARROW);
wfc->icon = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_ICON1));
wfc->wndClassName = _tcsdup(_T("FreeRDP"));
wfc->wndClass.cbSize = sizeof(WNDCLASSEX);
wfc->wndClass.style = CS_HREDRAW | CS_VREDRAW;
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));
```

2.这个简单, 注册了一个hook接口, 然后不断的通过TranslateMessage和DispatchMessage分发消息, 注册的hook接口wf_ll_kbd_proc主要就是判断一些特别的案件, 做一下过滤处理, 然后通过freerdp_input_send_keyboard_event_ex发送到服务端。最后通过CallNextHookEx调用下一个hook

```
wfc->keyboardThread = CreateThread(NULL, 0, wf_keyboard_thread, (void*) wfc, 0,
                                   &wfc->keyboardThreadId);
```

3. wf_client_thread这个是最复杂的, 说复杂只是因为调用层次比较复杂, 其内部结构并不复杂。

- 复杂点, 这个主要就是调用freerdp的连接接口

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

- 如果采用异步输入, 需要再创建一个input线程, 最终调用proxy中的对应回调。这里的回调在input_register_client_callbacks中绑定。如果是异步, 会把input对象成员的回调替换为input_message_*系列, 并且将与安排input对象绑定的

input_send_*event系列赋值给proxy。input_message_*系列会将鼠标键盘事件转化为message缓存到input_queue中, 并在wf_input_thread线程函数中通过freerdp_message_queue_process_message异步发送鼠标键盘事件。间接调用的proxy中的KeyboardEvent等回调对象。

```
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;
    }
}
```

- while循环不断的判断客户端窗口有无事件产生, 如果产生则进行数去抓取。每次循环都要获取一次焦点

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

获取transport input等事件队列中的事件数量

```
{
    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;
}
```

等待这些事件触发, 一秒内不处罚执行下一次while循环

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

获取数据的检测函数, 这个就是获取服务器数据的接口

```
{
    if (!freerdp_check_event_handles(context))
    {
        if (client_auto_reconnect(instance))
            continue;

        WLog_ERR(TAG, "Failed to check FreeRDP file descriptor");
        break;
    }
}
```

判断是否需要断开链接

```
if (freerdp_shall_disconnect(instance))
    break;
```

windows事件循环, 将时间分发到FreeRDP-Client窗口, 然后再分发到系统

```
while (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
{
    msg_ret = GetMessage(&msg, NULL, 0, 0);

    if (instance->settings->EmbeddedWindow)
    {
        if ((msg.message == WM_SETFOCUS) && (msg.lParam == 1))
        {
            PostMessage(wfc->hwnd, WM_SETFOCUS, 0, 0);
        }
        else if ((msg.message == WM_KILLFOCUS) && (msg.lParam == 1))
        {
            PostMessage(wfc->hwnd, WM_KILLFOCUS, 0, 0);
        }
    }

    if (msg.message == WM_SIZE)
    {
        width = LOWORD(msg.lParam);
        height = HIWORD(msg.lParam);
        SetWindowPos(wfc->hwnd, HWND_TOP, 0, 0, width, height, SWP_FRAMECHANGED);
    }

    if ((msg_ret == 0) || (msg_ret == -1))
    {
        quit_msg = TRUE;
        break;
    }
```

跳出while循环以后, 如果启用异步input则处理完input消息队列中的数据

```c
/* cleanup */
if (async_input)
{
    wMessageQueue* input_queue;
    input_queue = freerdp_get_message_queue(instance, FREERDP_INPUT_MESSAGE_QUEUE);

    if (MessageQueue_PostQuit(input_queue, 0))
        WaitForSingleObject(input_thread, INFINITE);
}
```