free_bird816

浏览: 56708 次

性别:

来自: 济南

最近访客 更多访客>>

sun_2008 niejun0205

jerome_yi 平江夜弹

文章分类

- 全部博客 (185)
- delphi (18)
- 数据库 (38)
- dotnet (12)
- javascript (7)
- java (31)
- vb (3)
- asp (1)
- linux & VOIP & callcenter (27)
- 生活 (4)
- 计算机其它 (25)
- perl (1)
- 网站架构 ecshop (8)
- 网络 (3)
- 项目管理 (2)

OSIP协议栈使用入门（续一：纯协议栈逻辑分析）（转）

博客分类： linux & VOIP & callcenter

网络协议 数据结构 Socket 网络应用 嵌入式

很长时间之前，简单粗略地看了下*Osip,eXosip,ortp*等并快速“封装”了一个*Windows*下的基于*VC6*的*MFC*的*SIP*软电话（全部源代码*VC6*工程文件及*Lib*库可在本*Blog*共享文件夹找到），由于时间限制，只能是一知半解地纯“应用”式地分析了一下*osip,eXosip*等开发库的代码，作为兴趣爱好者参考了解下*SIP*电话工作原理还可以，但作为商用产品开发参考则还是太浅显了些：）

最近扩展嵌入式*Linux*平台上的*SIP*功能模块（基于*OSIP*），由于使用的*Osip*不包括*Call Transfer*相关字段*(Refer,Notify*等*)*的解析和状态机控制（最近的*Osip*版本是否有扩展未查看）不能支持呼叫转接，需要手工扩展，有机会对*Osip*的主要事务状态机、解析库等部分稍有了些较深入的了解，结合*SIP RFC*总结分享如下。

（注：下文假设阅读者已经大概了解SIP协议的简单呼叫流程，会使用Ethereal等抓包工具分析SIP消息结构，对C语言的指针、链表、内存控制及状态机等概念有足够的认识。）

要应用Osip到我们的程序中去，首先要看官方文档，文档中对Osip协议栈提供的各个功能部件如何使用都有比较详细的描述，但未进行整体性的分析，某些中文的指导文档也都停留在对其简单的翻译，不能为不熟悉该协议栈使用的用户快速参考使用，本文档不按照Osip的代码进行按功能分块说明，而是根据实际使用时的代码使用顺序来对主要逻辑流程进行分析，并适当对流程中使用到的功能部件进行说明，具体更详细的功能说明或疑问可直接查看官方文档对应部分的解释或直接查看功能函数源代码即可解决。

- 准备工作

  先认识几个结构体：osip_t，osip_message_t，osip_dialog_t，osip_transaction_t；

  osip_t是一个全局变量，所有要使用Osip协议栈的事务处理能力的程序都要第一步就初始化它（相对应于只使用osipparser库进行SIP消息字段解析的应用来说，如果只使用parser库到自己的程序中，想必对SIP协议栈已经很熟悉了，不需再往下看了^_^），它内部主要是定义了Osip协议栈的四个主要事务链表、消息实际发送函数及状态机各状态事件下的回调函数等；

  osip_message_t是SIP消息的C语言结构体存储空间，收到SIP消息解析后存在该结构中方便程序使用接收到的消息中的指定的字段，发送消息前为方便设置要发送的字段值，将要发送的内容存在该结构中等发送时转为字符串；

  osip_dialog_t则是SIP RFC中的dialog或叫call leg的定义，它标识了uac和uas的一对关系，并一直保持到会话(session)结束，一个完整的dialog主要包括from,to,callid,fromtag,totag,state等（可查看源码），其中fromtag,totag,callid在一个dialog成功建立后才完整，体现在SIP消息中，就是From、To的tag，Call-id字段的值相同时，这些消息是属于它们对应的一个Dialog的，例如将要发起invite时，只有fromtag,callid填充有值，在收到to远端的响应时，收到totag填充到dialog中，建立成功一个dialog，后继的逻辑均是使用这个dialog进行处理（如transaction事务处理），state表示本dialog的状态，与transaction的state有很大的关联，共用由Enum结构state_t定义；

  osip_transaction_t则是RFC中的事务的定义，它表示的是一个会话的某个Dialog之间的某一次消息发送及其完整的响应，例如invite-100-180-200-ack这是一个完整的事务，bye-200这也是一个完整的事务，体现在SIP消息中，就是Via中的branch的值相同表示属于一个事务的消息（当然，事务是在Dialog中的，所以From、To的tag，Call-id值也是相同的），事务对于UAC,UAS的终端类型不同及消息的不同，分为四类，

- osgi (0)
- SOA&集群&云 (4)

社区版块
- 我的资讯 (0)
- 我的论坛 (12)
- 我的问答 (0)

存档分类
- 2012-07 (2)
- 2012-04 (2)
- 2012-03 (4)
- 更多存档...

最新评论

fox-idea：学习
用PC作数据库服务器的容灾问题

xwei78：话说这个下载地址不太好找啊，我去oracle网站转了好一会才找 ...
oracle10g官方客户端配置

前面说的invite的事务，主叫uac中会关联一个ict事务，被叫uas会关联一个ist事务，而除了invite之外，都归类定义主叫nict，被叫nist，在Osip中，它是靠有限状态机来实现的上述四种事务（osip_fsm_type_t中定义）的，它的主要属性值有callid,transactionid，分别来标识dialog和transaction，其中还有一个时间戳birth_time标识事务创建时间，可由超时处理函数用来判断和决定超时情况下的事务的进行和销毁，而它的state属性是非常重要的，根据上述的事务类型不同，其值也不同，它是前面提到的状态机的“状态”，在实际状态机的逻辑执行中是一个关键值；

- Osip初始化

提到osip的初始化，可能大家都看过官方文档里第一页的代码，首先就是osip_init(&osip)初始化了全局的osip_t结构体，然后对它的回调函数进行设置，很多人估计就是一看到这密密麻麻的一页多的call_back设置被吓到了，但结合前面分析的三个结构体的含义，这里的含义就很清晰了：

osip_t中有一个cb_send_message函数指针，它是Osip最终与外界网络交互的接口，它的参数有(

osip_transaction_t * trn,　/*本消息所属的事务*/

　　osip_message_t * sipmsg, /*待发送的消息结构体*/

　　char *dest_socket_str,　/*目标地址*/

　　int32_t dest_port,　　/*目标端口*/

　　int32_t send_sock) /*用来发送消息的socket*/

其中trn传入主要是为了方便获取事务的上下文数据，它有一个void指针your_instance，可以用来传入更多数据方便发送消息时参考，例如将该事务所属的dialog指针传入；

而sipmsg则是我们要发送的SIP消息的C结构体，使用osip_message_to_str将其按RFC文档格式转换为一个字符串（osip中的parser模块的主要功能），再通过任意你自己的网络数据发送函数使用send_sock发送给dest_socket_str和dest_port指定的目标，当然，要记得使用osip_free释放刚才发送出去的字符串占用的内存，Osip中很多osipparser提供的消息解析处理函数都是动态内存分配的，使用完毕后需要及时释放；

使用osip_set_cb_send_message成功设置回调函数，我们的SIP消息就有了出口了，下面继续分析（当然，了解到了上面的流程，也可以手工指定了）。

下面的回调函数分为三类，分别是普通事务消息（osip_message_callback_type_t中定义）的处理回调函数、事务销毁事件（osip_kill_callback_type_t中定义）的清理回调函数以及事务执行过程中的错误事件（osip_transport_error_callback_type_t中定义）处理回调函数：

先说简单的，事务销毁事件，事务正常结束（成功完成状态机流程）或由超时处理函数强制终结等情况下均调用了这些回调函数，一般就是释放事务结构体，为ICT,NICT,IST,NIST各设置或共用一个回调函数均可，只要正确释放不再使用的内存即可；

错误处理函数则是在整个状态机执行过程中发生的任何错误的出口，一般用来安插log函数方便调试，也可以直接设为空函数；

而最关键的就是正常消息的处理回调函数了，其量是非常大的，但仔细分下类，也和上面的回调函数一样，也是分为四类，我们可有根据实际程序的需要来进行设置，例如，SIP电话机就不需要处理OSIP_NIST_REGISTER_RECEIVED这个SIP注册服务器才需要处理的Register消息事件了，精简一下，如果只是要做一个只需要实现主叫功能且不考虑错误情况的UAC的Demo软电话程序，则只需要设置如下几个事件的回调函数：

OSIP_ICT_INVITE_SENT 发出Invite开始呼叫

OSIP_ICT_STATUS_1XX_RECEIVED 收到180

OSIP_ICT_STATUS_2XX_RECEIVED 收到200

OSIP_ICT_ACK_SENT 发出ack确定呼叫

OSIP_NICT_BYE_SENT 发出bye结束呼叫

OSIP_NICT_STATUS_2XX_RECEIVED 收到200确认结束呼叫

OSIP_NIST_BYE_RECEIVED 收到bye结束呼叫

OSIP_NIST_STATUS_2XX_SENT 发出 200确定结束呼叫

而要增加接受呼叫的被叫UAS功能，则只需要增加如下事件：

OSIP_IST_INVITE_RECEIVED 收到invite开始呼叫
OSIP_IST_STATUS_1XX_SENT 发出180
OSIP_IST_STATUS_2XX_SENT 发出200
OSIP_IST_ACK_RECEIVED 收到ack确认呼叫
具体的函数定义，则直接参
考osip_message_cb_t，osip_kill_transaction_cb_t，osip_transport_error_cb_t即可，回调函数的设置同上可以手工设置，也可以使用Osip提供的对应的osip_set_xxx_callback函数；

- 发出SIP消息
  要发送SIP消息，从上面的分析可知有几个必要的条件，osip_messag_t结构的待发送消息，osip_dialog_t结构体的dialog以及osip_transaction_t的事务；
  首先osip_malloc新分配一个dialog，使用osip_to_init，osip_to_parse，osip_to_free这类parser函数功能函数按RFC设置call-id,from,to,local_cseq等必要字段（原则是：后面生成实际SIP消息结构体要用到的字段就需要设置），使用osip_message_init初始化一个sipmsg，根据dialog来填充该结构体（不同的消息填充的数据是不同的，没有捷径可走，只能看RFC根据需要填充字段），如果要给SIP消息添加Body例如SDP段，需要使用osip_message_set_body，osip_message_set_content_type函数，设置的值是纯文本，如果是SDP，Osip有提供简单的解析和生成便捷函数例如sdp_message_to_str,sdp_message_a_attribute_add，但只是简单的字符操作，要填充合法的字段需要自己参考SDP的RFC文档，同样没捷径可走。
  现在我们有了两个必要条件了，还有最后一个也是最关键的部件，就是事务的创建和触发，
  int osip_transaction_init(
  osip_transaction_t ** transaction, /*返回的事务结构体指针*/
  osip_fsm_type_t ctx_type, /*事务类型ICT/NICT/IST/NIST*/
  osip_t * osip, /*前文说的全局变量*/
  osip_message_t * request) /*前面生成的sipmsg*/
  创建了一个新的事务，并自动根据事务类型、dialog和sipmsg进行了初始化，最重要的是它使用了__osip_add_ict等函数，将本事务插入到全局的osip_t结构体的全局FIFO链表中去了，不同的事务类型对应不同的FIFO，由前文可知，本类函数有四个，FIFO也有四个，对应ICT,NICT,IST,NIST，注意这个这里使用osip_transaction_set_out_socket把发送sip消息的socket接口配给该事务，方便自动调用前面设置的发送消息回调函数使用它自动发送消息；
  前文提到了transaction里的state作为状态机的“状态”，要执行状态机，就需要有“事件”来触发，事件结构体osip_event_t需要使用osip_new_outgoing_sipmessage来对sipmsg进行探测生成，设置正确的事件值，省却了我们手工设置的工作，它调用evt_set_type_outgoing_sipmessage来设置“事件”type_t，并将sipmsg挂到事件结构体的sip属性值上，有了根据消息分析出的事件后，使用osip_fifo_add(trn->transactionff, ev)将事件插入到事务的事件FIFO中，即transactionff属性；

  有了上面的发送消息的必要条件了，消息是如何实际出发的呢？上面提到了，SIP消息的发送和响应是一个事务，不能隔离开来，即消息的发送需要事务状态机来控制，我们上面设置了状态机的状态和事件，要触发它，就是要执行状态机了：
  osip_ict_execute
  osip_nict_execute
  osip_ist_execute
  osip_nist_execute
  分别用来遍历前面提到的四个事务FIFO，取出事务，再依次取出事务内的事件FIFO上的事件，使用osip_transaction_execute依次执行（有兴趣的可以更深一步去查看，可以看到它最终就是调用了我们前面设置的消息回调函数，至于具体调用哪个，这就是OSIP协议栈内部帮我们做的大量的工作了^_^）；
  如果某个事务不能正常终结怎么办呢？例如发出了Invite没有收到任何响应，按RFC定义，不同的事务有不同的超时时间，osip_timers_ict[nict|ist|nist]_execute这些函数就是来根据取出的事务的时间戳与当前时间取差

后与规定的超时时间比对，如果超时，就自动设置了超时“事件”并将事务“状态”设为终结，使用前面设定的消息超时事件回调函数处理即可（如果设置了）；

如果网络质量不稳定，经常丢失消息，需要使用osip_retransmissions_execute函数来自动重发消息而不是等待超时；

为了即时响应SIP消息的处理推动状态机，上述的九个函数需要不停执行，可以将它放入单独线程中。

- 收到SIP消息

  有了前面的发送SIP消息的理解，接收消息的处理就方便理解了，收到SIP消息，使用osip_parse进行解析，得到一个osip_message_t的sipmsg，使用evt_set_type_incoming_sipmessage得到事务的“事件”，并同上将sipmsg挂到事件结构体的sip字段，随后立即使用osip_find_transaction_and_add_event来根据“事件”查找事务（有兴趣可以深入看一下，事务的查找是通过SIP消息Via中的branch来匹配的），否则新建事务，然后推动状态机执行。

- 状态机内部逻辑

  弄清了上面的状态机的大概逻辑，设置正确完备的回调函数，就可以正确使用Osip来进行工作了，如果要进一步深入Osip，比如要扩展Osip的状态机处理自定义的消息字段和实现新的事务逻辑来生成新业务时，就需要对状态机的内部逻辑有一定的了解；

  前面一再强调，Osip内部的几个重要的数据结构osip_message_t,osip_dialog_t,osip_transaction_t，其中面向用户的主要是前后两个，而中间的dialog则很多时候是在状态机内部使用的，例如：收到消息，解析到sipmsg中，查找transaction并进行驱动，随后找到它关联的dialog（或者新生成）解析填充要发送的消息结构体sipmsg，再次根据dialog和sipmsg查找或生成transaction。

  如果要扩展Osip，要做工作主要有：

  扩展osip_message_t，增加要解析的字段或消息头，并参考原Osip函数生成对应的SIP字符串生成和解析函数；

  扩展osip_dialog_t，增加新的属性，对应osip_message_t的新增内容；

  扩展状态机的事件和状态类型，设置对应的回调函数，并关联新增事件和状态类型到osip_message_t的解析函数或osip_dialog_t的初始化函数中，而osip_transaction_t大多数时候不需要扩展，只要在对应的事务类型（大多数时候是NICT、NIST）处理逻辑中，增加对新增事件和状态类型的判断和调用回调函数的逻辑即可。

转http://mbstudio.spaces.live.com/blog/cns!C898C3C40396DC11!2860.entry

游戏外挂(转) | oSIP协议栈(及eXoSIP,Ortp等)使用入门（ ...

2010-02-03 08:54 | 浏览 1562 | 评论(0) | 分类:编程语言 | 相关推荐 MORE

评论

发表评论

您还没有登录,请您登录后再发表评论