## 资源分析：

master main线程

STDIN\_FILENO：用于测试时的控制台输入。

"?file%d" 能打开文件则触发main线程instant配置事件，记录文件id到全局变量。

"/file%d:file%d" 能打开头尾两个文件则触发main线程waited配置事件，记录头尾文件id到全局变量。

g\_iMainEventFd：用于监听下配置事件与sync发来的通讯事件。

MASTER\_MAIN\_EVENT\_KEYIN\_INSTANT 用文件id打开并读取文件，发送到instant链表，触发sync线程instant配置事件。

MASTER\_MAIN\_EVENT\_KEYIN\_WAITED 用头尾文件id循环打开并读取文件，循环发送到waited链表，触发sync线程waited配置事件。

MASTER\_MAIN\_EVENT\_SLAVE\_RESTART

MASTER\_MAIN\_EVENT\_CHECKALIVE\_TIMER

g\_pstInstantList：用于main线程与sync线程间传输instant配置与重传。

由于main线程需要写入，所以由main线程创建。

只通过main线程逐个往表尾填充数据，由事件通知sync线程从new节点取新数据（后移新数据的标志）并发送（一般只在表尾取最后一个节点，但考虑到可能事件通知失败，所以还是设置一个new的节点标志与newSize节点个数，也就是说一次可能发送多个）。

如果对端回复instant数据包OK，则根据uiDataID查找并删除节点；如果对端回复instant数据包error，则根据uiDataID查找并重发该节点（如cSendTimers超过3次则删除节点，重发后cFindTimers置零，cSendTimers自加）。需要特别说明的是，在查找过程中，从表头开始查找，途中经过的节点的cFindTimers都需要自加（说明在这个节点后面发送的节点都已经收到了回复），而后面在遍历链表时则根据每个节点的cFindTimers参数（暂定大于1）选择是否要重发。

如果触发g\_iInstantTimerFd定时器，则从表头开始遍历链表，如果cFindTimers超过阈值（暂定大于1）则说明该节点超时未回，重发此节点（重发后cFindTimers置零，cSendTimers自加，如cSendTimers超过3次则删除节点）；为避免表尾节点不回导致有节点cFindTimers参数无法超过阈值，则在定时遍历链表时，如果该节点cFindTimers不超过阈值则自加，以便在下次遍历时进行重发。

如果模块重启，则清空链表。

g\_pstWaitedList：用于main线程与sync线程间传输waited配置与重传。

由于main线程需要写入，所以由main线程创建。

只通过main线程逐个往表尾填充数据，填充多个完毕后通过事件通知sync线程，查看uiMsgLen是否超过定量阈值，是则申请uiMsgLen长度的内存，从头遍历链表，如果节点发送次数cSendTimers达3次则删除节点，减小uiMsgLen值；如果发送次数小于3次则将节点数据放入消息包，并增加发送次数cSendTimers，最后整体发送，重置g\_iWaiteTimerFd定时器。

如果触发g\_iWaiteTimerFd定时器，满足定时阈值，申请uiMsgLen长度的内存，从头遍历链表，如果节点发送次数cSendTimers达3次则删除节点，减小uiMsgLen值；如果发送次数小于3次则将节点数据放入消息包，并增加发送次数cSendTimers，最后整体发送，重置g\_iWaiteTimerFd定时器。

如果对端回复成功的waited节点ID，则查找并删除该节点。如果不回复则等待定时定量阈值的触发。（极端情况下，对端一直不回复waited成功消息，虽然节点发送多次后会被删除，但会有很多次的无意义发送）

如果模块重启，则清空链表。

master sync线程

g\_iSyncSockFd：

接收到消息包，重置定时器g\_iCheckaliveTimerFd，读取消息包，二进制形式打印消息包，解析消息包（需要详细说明）。

如有发送消息包，则重置定时器g\_iKeepaliveTimerFd。

g\_iSyncEventFd：

MASTER\_SYNC\_EVENT\_NEWCFG\_INSTANT

MASTER\_SYNC\_EVENT\_NEWCFG\_WAITED

g\_iLoginSynAckTimerFd：此定时器用于登录阶段三次握手的第二阶段。

收到login SYN消息包后，记录g\_cSlaveSpecifyID，修改g\_cMasterSyncStatus为STATUS\_LOGIN，记录g\_cLoginRspSeq以循环发送SYN ACK消息包，开启此定时器。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_LOGIN则发送SYN ACK消息包。重置此定时器。

收到login ACK消息包后，修改g\_cMasterSyncStatus为STATUS\_NEWCFG，关闭此定时器。

g\_iKeepaliveTimerFd：此定时器用于长时间未发送消息的保活之用。

由于每次发送都会重置此定时器，无需自主开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG则发送keepalive消息包。重置此定时器。

g\_iCheckaliveTimerFd：此定时器用于长时间未接收到消息的重启之用。

由于每次接收都会重置此定时器，无需自主开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG则说明通讯故障，向main线程发送MASTER\_EVENT\_CHECKALIVE\_TIMER事件（自行重启，g\_cMasterSpecifyID、g\_cMasterSyncStatus、链表）。重置此定时器。

g\_iInstantTimerFd：此定时器用于instant链表中的已发送过但未成功的配置消息的重发。

由于只要instant链表中有配置则需要定时重发，所以此定时器从g\_cMasterSyncStatus设置为STATUS\_ NEWCFG时开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG则从头遍历instant链表，如果该节点发送次数超过3次，则删除该节点；如果该节点查找次数达1（说明后面发送的配置在此条配置之前已经收到回复），则发送此条配置，并清零查找次数、累加发送次数。重置此定时器。

g\_iWaitedTimerFd：此定时器用于waited链表中的配置消息的定时重发。

此定时器从g\_cMasterSyncStatus设置为STATUS\_ NEWCFG时开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG且waited链表不为空，则申请总长度内存。从头遍历waited链表，如果该节点发送次数超过3次，则删除该节点，并减小消息包长度和；打包并发送waited消息包。重置此定时器。

slave main线程

g\_iMainEventFd：用于监听事件。

++++++

g\_ pstInstantList：用于。

++++++

g\_ pstWaitedList：用于。

++++++

slave sync线程

g\_iSyncSockFd：

接收到消息包，重置定时器g\_iCheckaliveTimerFd，读取消息包，二进制形式打印消息包，解析消息包（需要详细说明）。

如有发送消息包，则重置定时器g\_iKeepaliveTimerFd。

g\_iSyncEventFd：

MASTER\_SYNC\_EVENT\_NEWCFG\_INSTANT

MASTER\_SYNC\_EVENT\_NEWCFG\_WAITED

g\_iLoginSynAckTimerFd：此定时器用于登录阶段三次握手的第二阶段。

收到login SYN消息包后，记录g\_cSlaveSpecifyID，修改g\_cMasterSyncStatus为STATUS\_LOGIN，记录g\_cLoginRspSeq以循环发送SYN ACK消息包，开启此定时器。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_LOGIN则发送SYN ACK消息包。重置此定时器。

收到login ACK消息包后，修改g\_cMasterSyncStatus为STATUS\_NEWCFG，关闭此定时器。

g\_iKeepaliveTimerFd：此定时器用于长时间未发送消息的保活之用。

由于每次发送都会重置此定时器，无需自主开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG则发送keepalive消息包。重置此定时器。

g\_iCheckaliveTimerFd：此定时器用于长时间未接收到消息的重启之用。

由于每次接收都会重置此定时器，无需自主开启；此定时器不关闭。

触发此定时器后，如果g\_cMasterSyncStatus为STATUS\_ NEWCFG则说明通讯故障，向main线程发送MASTER\_EVENT\_CHECKALIVE\_TIMER事件（自行重启，g\_cMasterSpecifyID、g\_cMasterSyncStatus、链表）。重置此定时器。

## 逻辑分析：

master main线程开始：

log\_init

reliable\_sync\_init

g\_pstInstantList = instantList\_create;

g\_pstWaitedList = waitedList\_create;

g\_iMainEpollFd = epoll\_create;

g\_iMainEventFd = event\_init;

g\_iSyncEventFd = event\_init;

tool\_add\_event\_to\_epoll(g\_iMainEpollFd, g\_iMainEventFd);

tool\_add\_event\_to\_epoll(g\_iMainEpollFd, STDIN\_FILENO);

pthread\_create master\_sync\_thread

epoll\_wait(g\_iMainEpollFd,);

\_epoll\_stdin();

read STDIN\_FILENO

sscanf "?file%d"

open pcFilenameInstant

event\_set MASTER\_MAIN\_EVENT\_KEYIN\_INSTANT

sscanf "/file%d:file%d"

open pcFilenameBegin open pcFilenameEnd

event\_set MASTER\_MAIN\_EVENT\_KEYIN\_WAITED

\_epoll\_mainEvent();

event\_get g\_iMainEventFd

MASTER\_MAIN\_EVENT\_KEYIN\_INSTANT

open pcFilenameInstant

read iFileInstantFd

reliable\_sync\_send

event\_set MASTER\_SYNC\_EVENT\_NEWCFG\_INSTANT

MASTER\_MAIN\_EVENT\_KEYIN\_WAITED

for g\_iFileNumWaitedBegin g\_iFileNumWaitedEnd

open pcFilenameWaited

read iFileWaitedFd

reliable\_sync\_send

event\_set MASTER\_SYNC\_EVENT\_NEWCFG\_WAITED

MASTER\_MAIN\_EVENT\_SLAVE\_RESTART

++++++

MASTER\_MAIN\_EVENT\_CHECKALIVE\_TIMER

++++++

master sync线程开始 master\_sync\_thread：

master\_sync\_init

g\_iSyncEpollFd = epoll\_create

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iSyncEventFd)

g\_iSyncSockFd = socket\_init()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iSyncSockFd)

g\_iLoginSynAckTimerFd = timer\_create()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iLoginSynAckTimerFd)

g\_cMasterSpecifyID = (char)(rand() % 0x100)

g\_iKeepaliveTimerFd = timer\_create()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iKeepaliveTimerFd)

g\_iCheckaliveTimerFd = timer\_create()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iCheckaliveTimerFd)

g\_iInstantTimerFd = timer\_create()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iInstantTimerFd)

g\_iWaitedTimerFd = timer\_create()

tool\_add\_event\_to\_epoll(g\_iSyncEpollFd, g\_iWaitedTimerFd)

epoll\_wait g\_iSyncEpollFd

\_epoll\_syncSocket()

recvFromSlaveSync

timer\_start g\_iCheckaliveTimerFd

read

log\_hex

handle\_sync\_msg

handle\_one\_msg

if STATUS\_INIT || STATUS\_LOGIN

sync\_login

if cSynFlag STATUS\_INIT

if (g\_cSlaveSpecifyID == 0)

g\_cSlaveSpecifyID = req->cSpecifyID

else if(g\_cSlaveSpecifyID != req->cSpecifyID)

event\_set MASTER\_EVENT\_SLAVE\_RESTART

g\_cSlaveSpecifyID = req->cSpecifyID

timer\_start g\_iLoginSynAckTimerFd

g\_cMasterSyncStatus = STATUS\_LOGIN

g\_cLoginRspSeq = req->msgHeader.cSeq

if cAckFlag STATUS\_LOGIN

timer\_stop g\_iLoginSynAckTimerFd

g\_cMasterSyncStatus = STATUS\_NEWCFG

if STATUS\_NEWCFG

sync\_login

同上

sync\_newCfgInstant

pstNode = instantList\_find

if NEWCFG\_RESULT\_SUCCEED

instantList\_delete pstNode

else

if pstNode->iSendTimers >= 3

instantList\_delete pstNode

else

alloc\_master\_newCfgInstantReq

sendToSlaveSync

pstNode->iFindTimers = 0

pstNode->iSendTimers++

sync\_newCfgWaited

for i < ntohl(rsp->msgHeader.iLength) / sizeof(unsigned int)

waitedList\_findAndDelete ntohl(\*piSucceedID)

piSucceedID++

sync\_keepAlive

if(g\_cSlaveSpecifyID != req->cSpecifyID)

event\_setEventFlags MASTER\_EVENT\_SLAVE\_RESTART

g\_cSlaveSpecifyID = req->cSpecifyID

\_epoll\_syncEvent()

\_epoll\_loginSynAckTimer ()

\_epoll\_keepaliveTimer()

\_epoll\_checkaliveTimer()

\_epoll\_instantTimer()

\_epoll\_waitedTimer()