TrustZone感知的机密虚拟机众智项目一期详细设计说明书

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 修订版本 | 修改描述 | 作者 |
| 2023-10-24 | V1.0 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. 整体设计 3](#_Toc6705)

[1.1. 总体思路 3](#_Toc2596)

[1.2. 软件第三方依赖说明 3](#_Toc5986)

[1.3. 软件（方案）约束说明 4](#_Toc32470)

[1.3.1. 硬件环境要求 4](#_Toc26535)

[1.3.2. 操作系统要求 4](#_Toc10650)

[2. vtzdriver模块设计与实现 4](#_Toc8382)

[2.1. vtzdriver模块内同vtz\_proxy通信的接口与数据结构 8](#_Toc11726)

[3. qemu模块设计与实现 15](#_Toc29398)

[3.1. qemu模块数据结构 16](#_Toc21510)

[4. vtz\_proxy 设计与实现 17](#_Toc209)

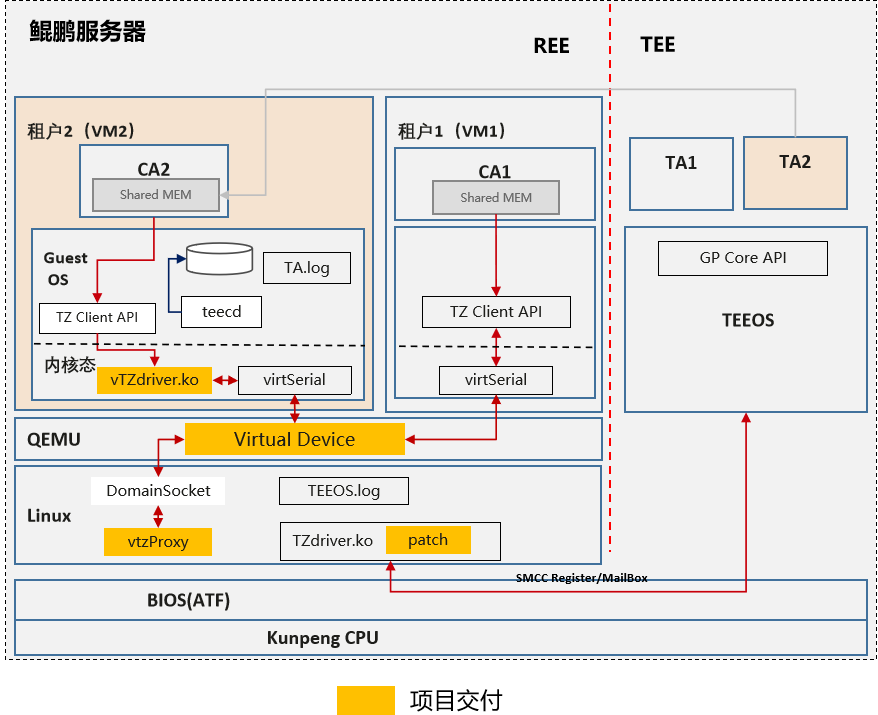
[4.1. vtz\_proxy模块同vtzdriver通信的接口与数据结构 18](#_Toc9674)

[5. tzdriver(patch)设计与实现 24](#_Toc8063)

1. 整体设计
   1. 总体思路

本项目拟借助qemu虚拟串口 virtserial、充分利用内存拷贝与内存共享，构建TrustZone感知的机密虚拟机，其整体架构如图 1所示。构建vtzdriver，提供与tzdriver相同的接口供上层itrustee\_client（libteec、teecd、tlogcat）调用。利用qemu提供的virtserial，在VM侧创建字符设备，在Host侧创建socket，连通VM与Host。vtz\_proxy接受识别由vtzdriver转发的tzdriver调用，识别后调用tzdriver对应接口。调用结果由vtz\_proxy、qemu、vtzdriver返回给上层应用。从而实现在VM中使用TEE的体验与本地Host上无差异。

为充分利用内存共享，本项目支持由VM内的CA发起的内存共享的操作，以及基于此机制的零切换方案（secGear Switchless）的使用。



**图 1 TrustZone感知的机密虚拟机项目总体架构**

* 1. 软件第三方依赖说明

编译、安装vtzdriver、vtz\_proxy时所需第三方依赖：

无

* 1. 软件（方案）约束说明
     1. 硬件环境要求

Host端硬件环境要求运行的泰山服务器已经预置了TrustZone特性，即预装iTrustee 机密OS（CCOS）以及配套的BMC、BIOS固件。

* + 1. 操作系统要求

Host端操作系统要求如表1所示。

**表 1 Host操作系统要求**

| 项目 | 版本 | 获取方式 |
| --- | --- | --- |
| openEuler OS | 20.03-LTS-SP1-everything-aarch64或更新版本  22.03 LTS | [https://repo.openeuler.org/openEuler-20.03-LTS-SP1/ISO/aarch64/](https://repo.openeuler.org/openEuler-20.03-LTS-SP1/ISO/aarch64/" \o " )  https://repo.openeuler.org/openEuler-22.03-LTS/ |

VM端操作系统要求Linux kernel版本4.19~5.10。

1. vtzdriver模块设计与实现

vtzdriver的使用者为itrustee\_client中的teecd、tlogcat程序与libteec库，提供与tzdriver相同的接口，使得上层应用最大化感知不到替换。vtzdriver的实现不涉及内核代码的修改，采用ko的方式在内核提供功能。vtzdriver实现了与tzdriver相同的接口，表2中列出了具体接口。

**表 2 vtzdriver接口**

|  |  |  |  |
| --- | --- | --- | --- |
| 模块/设备节点 | 对应文件 | 接口 | 描述 |
| /dev/tc\_ns\_client | vtzf.c |  | 1. 用于处理CA发过来的消息或命令 2. 用于teecd创建fd，做鉴权 |
|  | TC\_NS\_CLIENT\_IOCTL\_SES\_OPEN\_REQ TC\_NS\_CLIENT\_IOCTL\_SES\_CLOSE\_REQ TC\_NS\_CLIENT\_IOCTL\_SEND\_CMD\_REQ | CA 打开session，关闭session，发送命令 |
|  | TC\_NS\_CLIENT\_IOCTL\_CANCEL\_CMD\_REQ | 暂不支持 |
|  | TC\_NS\_CLIENT\_IOCTL\_LOGIN | teecd 设置鉴权信息 |
|  | TC\_NS\_CLIENT\_IOCTL\_UPDATE\_TA\_CRL | 更新cms证书(未使能) |
|  | TC\_NS\_CLIENT\_IOCTL\_REGISTER\_AGENT | CA注册agent,同时创建agent buffer |
|  | TC\_NS\_CLIENT\_IOCTL\_UNREGISTER\_AGENT | CA解注册agent，销毁agent buffer |
|  | TC\_NS\_CLIENT\_IOCTL\_SEND\_EVENT\_RESPONSE | 当agent代理服务处理完任务后，给gtask返回消息 |
|  | TC\_NS\_CLIENT\_IOCTL\_WAIT\_EVENT | CA中agent代理等待gtask发过来的消息 |
|  | TC\_NS\_CLIENT\_IOCTL\_LOAD\_APP\_REQ | CA或agent代理加载TA、动态库等 |
|  | TC\_NS\_CLIENT\_IOCTL\_TUI\_EVENT | tui agent处理消息（未使能） |
|  |  |  |  |
| /dev/tc\_private | vtzf.c |  | 1. 用于agent代理服务 2. 用于后加载服务或驱动 3. 用于同步时间 4. 用于设置native ca的身份 |
|  | TC\_NS\_CLIENT\_IOCTL\_GET\_TEE\_VERSION | 获取gp.tee.api\_level的版本号 |
|  | TC\_NS\_CLIENT\_IOCTL\_GET\_TEE\_INFO | 获取tee信息 |
|  | TC\_NS\_CLIENT\_IOCTL\_SET\_NATIVECA\_IDENTITY | 设置native ca的身份信息 |
|  | TC\_NS\_CLIENT\_IOCTL\_LATEINIT | teecd后加载动态驱动、动态服务 |
|  | TC\_NS\_CLIENT\_IOCTL\_SYC\_SYS\_TIME | teecd同步系统时间 |
|  | TC\_NS\_CLIENT\_IOCTL\_REGISTER\_AGENT | teecd注册agent,同时创建agent buffer |
|  | TC\_NS\_CLIENT\_IOCTL\_UNREGISTER\_AGENT | teecd解注册agent，销毁agent buffer |
|  | TC\_NS\_CLIENT\_IOCTL\_SEND\_EVENT\_RESPONSE | 当agent代理服务处理完任务后，给gtask返回消息 |
|  | TC\_NS\_CLIENT\_IOCTL\_WAIT\_EVENT | teecd中agent代理等待gtask发过来的消息 |
|  | TC\_NS\_CLIENT\_IOCTL\_LOAD\_APP\_REQ | teecd中agent代理加载TA、动态库等 |
|  |  |  |  |
| /dev/teelog |  |  |  |
| tlogger.c |  | 1. 用于日志模块的读和设置操作 |
|  | TEELOGGER\_GET\_VERSION | 获取teeos版本信息字符串 |
|  | TEELOGGER\_SET\_READERPOS\_CUR | tlogger设置读位置到最新 |
|  | TEELOGGER\_SET\_TLOGCAT\_STAT | tlogger设置日志文件转储 |
|  | TEELOGGER\_GET\_TLOGCAT\_STAT | tlogger获取当前是否转储文件 |
|  | TEELOGGER\_GET\_LOG\_POOL | 获取日志pool（未使能） |
|  | TEELOGGER\_LOG\_POOL\_APPEND | 追加日志pool（未使能） |
|  | TEELOGGER\_GET\_TEE\_INFO | tlogger获取tee信息 |
|  |  |  |  |

vtzdriver在加载时会打开串口设备/dev/virtio-ports/vtzf\_serialport，并创建两个内核线程vtz\_wr\_thread\_0、vtz\_rd\_thread\_0分别用于从串口发送和接收数据。上层应用对vtzdriver的调用均会执行send\_to\_proxy()。由send\_to\_proxy()创建结构write\_data与event\_data，将待发送的数据包拷贝到write\_data，并将write\_data加入到发送队列，将event\_data加入到接收链表，再通过wake\_up\_wr\_thread()调用唤醒内核线程vtz\_wr\_thread\_0。vtz\_wr\_thread\_0从发送队列取出write\_data后向串口依次发送，并唤醒vtz\_rd\_thread\_0。vtz\_rd\_thread\_0被唤醒后从串口读取数据，并根据数据包首字段packet\_size拆分数据获取完整的RSP数据包，遍历接收链表取出对应event\_data，填充RSP数据后唤醒用户线程。

每一个发送的数据包都有一个唯一的seq\_num对应，由全局变量g\_seq\_num\_xxx提供，以互斥的方式每次获取seq\_num后，g\_seq\_num\_xxx会递增2。通过seq\_num确保vtzdriver与vtz\_proxy每次通信的发送数据与接收数据是一一对应的。为了将CA的数据包和agent的数据包隔离开来，将0x0~0xFFFF的seq\_num保留给agent使用，将大于0xFFFF的seq\_num给CA使用。

在switchless场景下，需要在vtzdriver获取到user\_buf用户空间buf全部的PAGE，并将所有PAGE的物理地址gpa(经过qemu层地址转换为hva)发送给vtz\_proxy。为减少qemu层地址转换以及tzdriver获取PAGE的开销，需要对PAGES做分片处理。具体规则是按照获取的pages数组的顺序，PAGE对应的gpa若连续则划分为一个page\_block。page\_block保存块内首个PAGE的gpa以及块内PAGE的数量。通过这种方式可以将数万个PAGE转换为数十个page\_blocks。考虑到虚拟串口一次性最大可发送32KB的数据，仅在switchless场景下待发送的数据包可能大于32KB，需要分片处理，将待发送的数据包拆分成多个片段后依次发送。分片依据struct\_packet\_cmd\_send\_cmd数据包携带的page\_blocks数量，默认设置BLOCK\_MTU 为1024个page\_block。由于串口是串行发送且唯一指定了vtz\_wr\_thread\_0写串口，可以保证分片的接收顺序与发送顺序一致。为便于组装Packet，首个分片除携带MTU个page\_blocks外，在头部还具有完整的struct\_packet\_cmd\_send\_cmd控制信息。其余分片均不包含控制信息，仅携带page\_blocks。

VM内agent注册时会首先由vtzdriver分配空间并映射到用户空间，记buf1，随后向vtz\_proxy发送注册agent的命令包，由vtz\_proxy调用tzdriver相应接口完成注册，记buf2。buf1与buf2的映射关系由vtz\_proxy维护，两个buf间的数据拷贝由tzdriver完成。

VM内CA通过libteec调用mmap，vtzdriver分配空间并映射后会向vtz\_proxy发送命令包同样进行mmap操作。

vtzdriver在处理来自上层CA的调用TC\_NS\_CLIENT\_IOCTL\_SEND\_CMD\_REQ时，需要针对不同的TEEC\_ParamType类型作出对应处理。当TEEC\_ParamType为TEEC\_VALUE\_INPUT、TEEC\_VALUE\_OUTPUT、TEEC\_VALUE\_INOUT时，由于上层传递的是地址，故从用户空间读取值后，向vtx\_proxy进行值传递；当TEEC\_ParamType为TEEC\_MEMREF\_TEMP\_INPUT、TEEC\_MEMREF\_TEMP\_OUTPUT、TEEC\_MEMREF\_TEMP\_INOUT时，根据从用户空间获取的buffer\_size，采用kzalloc分配对应大小空间，并将数据从用户空间拷贝至内核，buffer采用地址传递经qemu层地址转换，buffer\_size采用值传递；当TEEC\_ParamType为TEEC\_MEMREF\_PARTIAL\_INPUT、TEEC\_MEMREF\_PARTIAL\_OUTPUT、TEEC\_MEMREF\_PARTIAL\_INOUT时，由于用户buffer是事先采用mmap获取的，故遍历shared\_mem链表找出对应kernel地址，buffer采用地址传递经qemu层地址转换，buffer\_size值传递；当TEEC\_ParamType为TEEC\_MEMREF\_SHARED\_INOUT时，使用switchless零拷贝特性，不能进行任何形式的拷贝，而是通过内核接口get\_user\_pages()获取PAGES，再调用page\_to\_phys()获取物理地址。

* 1. vtzdriver模块内同vtz\_proxy通信的接口与数据结构

**（1）**vtzdriver**模块内接口**

vtzdriver模块内部接口如表3 所示，接口功能均为与vtz\_proxy进行通信，将数据发送至vtz\_proxy并接收返回的数据。

**表 3** vtzdriver**模块内部接口**

|  |  |
| --- | --- |
| **接口** | **功能** |
| static int open\_tzdriver(  struct vtzf\_dev\_file \*dev\_file,  uint32\_t flag) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int close\_tzdriver(  struct vtzf\_dev\_file \*dev\_file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int proxy\_mmap(  struct vtzf\_dev\_file \*dev\_file,  void \* user\_buffer,  uint32\_t buffer\_size,  uint32\_t pgoff,  uint8\_t unmap) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_open\_session(  struct vtzf\_dev\_file \*dev\_file,  struct tc\_ns\_client\_context \*clicontext) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_close\_session(  struct vtzf\_dev\_file \*dev\_file,  void \_\_user \*argp) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_send\_cmd(  struct vtzf\_dev\_file \*dev\_file,  struct tc\_ns\_client\_context \*context) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_send\_cancel\_cmd(  struct vtzf\_dev\_file \*dev\_file,  void \*argp) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_client\_login\_func(  struct vtzf\_dev\_file \*dev\_file,  const void \_\_user \*buffer) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_client\_login\_func(  struct vtzf\_dev\_file \*dev\_file,  const void \_\_user \*buffer) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_get\_tee\_version(  struct vtzf\_dev\_file \*dev\_file,  void \_\_user \*argp) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_late\_init(  unsigned long arg) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int sync\_system\_time\_from\_user(  struct vtzf\_dev\_file \*dev\_file,  const struct tc\_ns\_client\_time \*user\_time) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int ioctl\_register\_agent(  struct vtzf\_dev\_file \*dev\_file,  void \_\_user \*argp) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_unregister\_agent(  struct vtzf\_dev\_file \* dev\_file,  unsigned int agent\_id) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int send\_wait\_event(  struct vtzf\_dev\_file \*dev\_file,  unsigned int agent\_id) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int send\_event\_response(  struct vtzf\_dev\_file \*dev\_file,  unsigned int agent\_id) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int tc\_ns\_load\_secfile(  struct vtzf\_dev\_file \*dev\_file,  struct load\_secfile\_ioctl\_struct \*ioctlArg) | 将参数传递给vtz\_proxy并接收相应的返回 |
| int tc\_ns\_get\_tee\_info(  int ptzfd,  void \_\_user \*argp,  bool flag) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int open\_tzdriver\_tlogger(  struct tlogger\_reader \*dev\_file,  uint32\_t flag) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int close\_tzdriver\_tlogger(  struct tlogger\_reader \*dev\_file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int get\_log\_from\_host(  struct tlogger\_reader \*dev\_file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int get\_teeos\_version(  struct tlogger\_reader \*dev\_file,  uint32\_t cmd,  unsigned long arg) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int set\_reader\_cur\_pos(  const struct file \*file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int set\_tlogcat\_f\_stat(  const struct file \*file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| static int get\_tlogcat\_f\_stat(  const struct file \*file) | 将参数传递给vtz\_proxy并接收相应的返回 |
| int send\_to\_proxy(  void \* wrt\_buf,  size\_t size\_wrt\_buf,  void \* rd\_buf,  size\_t size\_rd\_buf,  uint32\_t seq\_num) | 将参数传递给vtz\_proxy并接收相应的返回 |

（2）vtzdriver模块内数据结构

vtzdriver模块内与vtz\_proxy通信相关的数据结构均在comm\_structs.h中定义，数据结构命名形如struct\_packet\_cmd\_xxxx表示发送给vtz\_proxy的数据包，形如struct\_packet\_rsp\_xxxx表示从vtz\_proxy接收的返回值。如表4 所示。

**表 4** vtzdriver**模块内数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  } struct\_packet\_cmd\_general; | CMD包通用控制字段，packet\_size表示数据包大小，cmd是vtzdriver与vtz\_proxy通信的命令识别码，seq\_num序列号用于区分一次通信 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_general; | RSP包通用控制字段，packet\_size表示数据包大小，seq\_num序列号用于区分一次通信 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  uint32\_t vmid;  uint32\_t flag;  } struct\_packet\_cmd\_open\_tzd; | vmid字段用于在qemu层获取pid后填充，flag区别打开的设备 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  int32\_t ptzfd;  } struct\_packet\_rsp\_open\_tzd; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_close\_tzd; | ptzfd指定vtz\_proxy关闭的fd |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_close\_tzd; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_getteever; | 数据结构用于发送命令给vtz\_proxy获取tee版本 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  uint32\_t tee\_ver;  } struct\_packet\_rsp\_getteever; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  bool istlog;  } struct\_packet\_cmd\_getteeinfo; | 数据结构用于发送命令给vtz\_proxy获取tee信息 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct tc\_ns\_tee\_info info;  } struct\_packet\_rsp\_getteeinfo; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  void \*phyaddr;  struct agent\_ioctl\_args args;  } struct\_packet\_cmd\_regagent; | 数据结构用于发送命令给vtz\_proxy注册agent |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct agent\_ioctl\_args args;  } struct\_packet\_rsp\_regagent; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_unregagent; | 数据结构用于发送命令给vtz\_proxy注销agent |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_unregagent; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint32\_t agent\_id;  } struct\_packet\_cmd\_event; | 数据结构用于发送命令给vtz\_proxy等待gtask发过来的消息 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  uint32\_t index;  } struct\_packet\_cmd\_lateinit; | 数据结构用于发送命令给vtz\_proxy teecd后加载动态驱动、动态服务 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_lateinit; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  struct tc\_ns\_client\_time tcNsTime;  } struct\_packet\_cmd\_synctime; | 数据结构用于发送命令给vtz\_proxy同步系统时间 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_synctime; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint8\_t cert\_buffer[];  } struct\_packet\_cmd\_login; | 数据结构用于发送命令给vtz\_proxy设置鉴权信息 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_login\_non; | 数据结构用于发送命令给vtz\_proxy设置鉴权信息（无buffer情况） |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_login; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  struct load\_secfile\_ioctl\_struct ioctlArg;  } struct\_packet\_cmd\_load\_sec; | 数据结构用于发送命令给vtz\_proxy加载TA、动态库 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct load\_secfile\_ioctl\_struct ioctlArg;  } struct\_packet\_rsp\_load\_sec; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  struct tc\_ns\_client\_context cliContext;  } struct\_packet\_cmd\_session; | 数据结构用于发送命令给vtz\_proxy打开或关闭session |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct tc\_ns\_client\_context cliContext;  } struct\_packet\_rsp\_session; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  int32\_t err\_flag;  int32\_t is\_fragment;  uint32\_t fragment\_block\_num;  uint32\_t vm\_page\_size;  uint64\_t block\_addrs[TEE\_PARAM\_NUM];  uint32\_t block\_size[TEE\_PARAM\_NUM];  unsigned long long addrs[TEE\_PARAM\_NUM];  struct tc\_ns\_client\_context cliContext;  } struct\_packet\_cmd\_send\_cmd; | 数据结构用于发送命令给vtz\_proxy发送命令，is\_fragment、fragment\_block\_num、vm\_page\_size、block\_addrs、block\_size与switchless功能有关，用于传递page |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct tc\_ns\_client\_context cliContext;  } struct\_packet\_rsp\_send\_cmd; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  struct tc\_ns\_client\_context cliContext;  pid\_t pid;  } struct\_packet\_cmd\_cancel\_cmd; | 数据结构用于发送命令给vtz\_proxy |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct tc\_ns\_client\_context cliContext;  } struct\_packet\_rsp\_cancel\_cmd; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint64\_t buffer;  uint32\_t size;  uint32\_t offset;  } struct\_packet\_cmd\_mmap; | 数据结构用于发送命令给vtz\_proxy进行mmap |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_mmap; | 数据结构中包含了vtz\_proxy的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  } struct\_packet\_cmd\_nothing; | 数据结构用于发送命令给vtz\_proxy，与tzdriver无关 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_nothing; | 数据结构中包含了vtz\_proxy的返回值 |

1. qemu模块设计与实现

为了实现host对VM地址空间的访问，需要将VM内部的地址转换为Host侧的虚拟地址。qemu维护了gpa到hva的映射，可以通过接口gpa2hva()将gpa转换为hva，故只需要将gpa传递到qemu层，由qemu完成gpa到hva的转化。gpa2hva()是static函数，需修改qemu源码将其导出。对于拷贝的方式，gpa的获取可以由VM内核函数virt\_to\_phys()完成。对于switchless零拷贝的方式，可以通过get\_user\_pages()获取VM用户buf地址所有的PAGE，并将页面锁定使其驻留在VM的物理页面上，随后通过page\_to\_phys()获取gpa。

要实现地址的转换还需要截获VM到host的数据流，qemu serialport由VM--->Host的数据流可在源文件./qemu/hw/char/virtio-console.c的回调函数static ssize\_t flush\_buf(VirtIOSerialPort \*port, const uint8\_t \*buf, ssize\_t len)中截获。

由于pid在host侧的唯一性，故使用qemu的pid作为VM的vmid以区别不同虚拟机以及容器。vmid仅在传送open tzdriver的命令包时在qemu层通过getpid()获取。

VTZF\_LOAD\_SEC、VTZF\_FS\_REGISTER\_AGENT、VTZF\_OPEN\_SESSION、VTZF\_SEND\_CMD涉及地址转换。

* 1. qemu模块数据结构

qemu模块内数据结构用于获取pid、地址转换。

**表 5** qemu**模块内数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  uint32\_t vmid;  uint32\_t flag;  } struct\_packet\_cmd\_open\_tzd; | 在qemu层获取pid并填充到vmid字段 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  void \*vmaddr;  struct AgentIoctlArgs args;  } struct\_packet\_cmd\_regagent; | 将VM内agent buf 地址转为hva，以便tzdriver访问 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  int32\_t cpu\_index;  struct SecLoadIoctlStruct ioctlArg;  } struct\_packet\_cmd\_load\_sec; | 将VM内 buf 地址转为hva，以便tzdriver访问 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  int32\_t cpu\_index;  TC\_NS\_ClientContext cliContext;  } struct\_packet\_cmd\_session; | 将VM内 buf 地址转为hva，以便tzdriver访问 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  int32\_t err\_flag;  int32\_t is\_fragment;  uint32\_t fragment\_block\_num;  uint32\_t vm\_page\_size;;  uint64\_t block\_addrs[TEEC\_PARAM\_NUM];/  uint32\_t block\_size[TEEC\_PARAM\_NUM];  unsigned long long addrs[TEEC\_PARAM\_NUM];  TC\_NS\_ClientContext cliContext;  } struct\_packet\_cmd\_send\_cmd; | 将VM内 buf 地址转为hva，以便tzdriver访问 |
| typedef struct {  uint64\_t phy\_addr;  uint32\_t page\_num;  uint32\_t frag\_flag;  }struct\_page\_block; | 分片block\_pages地址转换 |

1. vtz\_proxy 设计与实现

vtz\_proxy与vtzdriver的通信在host侧以socket的方式进行。vtz\_proxy在初始化时会为每一个串口（在此场景下socket与虚拟串口是一一对应，后续不加区分两者）创建对应结构struct serial\_port\_file。该结构包含一个互斥锁用于写串口互斥、一个rd\_buf读缓冲区、一个opend标志、一个struct vm\_file\*，struct vm\_file是对VM的抽象。

由于VM可能重启、关机，导致socket连接会断开，因此需要使用access检查socket文件状态，若连接已关闭但socket文件存在且可读写，则建立socket连接，并将fd加入struct pollfd g\_pollfd[SERIAL\_PORT\_NUM]方便监听POLLIN事件；若连接曾打开但socket文件不存在，则需断开socket连接，从g\_pollfd中移除，并重置serial\_port\_file。

vtz\_proxy采用线程池的方式处理来自vtzdriver的命令请求。默认创建128个任务处理线程。当使用poll()监听到POLLIN事件后，则从对应的串口中读取数据到rd\_buf中，根据数据包首字段packet\_size拆分出完整数据包packet\_item，调用thread\_pool\_submit提交给任务队列。当任务处理线程接收到数据包后，会根据cmd字段调用tzdriver对应接口，并沿原串口写回返回值。

* 1. vtz\_proxy模块同vtzdriver通信的接口与数据结构

1. vtz\_proxy**模块内接口**

**表 6** vtz\_proxy**模块内接口**

|  |  |
| --- | --- |
| **接口** | **功能** |
| static void open\_tzdriver(  struct\_packet\_cmd\_open\_tzd \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void close\_tzdriver(  struct\_packet\_cmd\_close\_tzd \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void log\_in\_NonHidl(  struct\_packet\_cmd\_login\_non \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void log\_in(  struct\_packet\_cmd\_login \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void get\_tee\_ver(  struct\_packet\_cmd\_getteever \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void get\_tee\_info(  struct\_packet\_cmd\_getteeinfo \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void SyncSysTime(  struct\_packet\_cmd\_synctime \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void open\_session(  struct\_packet\_cmd\_session \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void close\_session(  struct\_packet\_cmd\_session \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void send\_cmd(  struct\_packet\_cmd\_send\_cmd \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void load\_sec\_file(  struct\_packet\_cmd\_load\_sec \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void vtz\_mmap(  struct\_packet\_cmd\_mmap \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void vtz\_nothing(  struct\_packet\_cmd\_nothing \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 将返回值发送给vtzdriver |
| void register\_agent(  struct\_packet\_cmd\_regagent \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| void wait\_event(  struct\_packet\_cmd\_event \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| void sent\_event\_response(  struct\_packet\_cmd\_event \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void tlog\_get\_teever(  struct\_packet\_cmd\_get\_ver \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void tlog\_set\_reader\_cur(  struct\_packet\_cmd\_set\_reader\_cur \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void tlog\_set\_stat(  struct\_packet\_cmd\_set\_tlogcat\_stat \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void tlog\_get\_stat(  struct\_packet\_cmd\_get\_tlogcat\_stat \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |
| static void tlog\_get\_log(  struct\_packet\_cmd\_get\_log \*packet\_cmd,  struct serial\_port\_file \*serial\_port) | 调用tzdriver对应接口并将返回值发送给vtzdriver |

1. vtz\_proxy**模块内数据结构**

**表 7** vtz\_proxy**模块内数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_general; | CMD包通用控制字段，packet\_size表示数据包大小，cmd是vtzdriver与vtz\_proxy通信的命令识别码，seq\_num序列号用于区分一次通信 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  uint32\_t vmid;  uint32\_t flag;  } struct\_packet\_cmd\_open\_tzd; | RSP包通用控制字段，packet\_size表示数据包大小，seq\_num序列号用于区分一次通信 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  int32\_t ptzfd;  } struct\_packet\_rsp\_open\_tzd; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_close\_tzd; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_close\_tzd; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_getteever; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  uint32\_t tee\_ver;  } struct\_packet\_rsp\_getteever; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  bool istlog;  } struct\_packet\_cmd\_getteeinfo; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  TC\_NS\_TEE\_Info info;  } struct\_packet\_rsp\_getteeinfo; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  void \*vmaddr;  struct AgentIoctlArgs args;  } struct\_packet\_cmd\_regagent; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct AgentIoctlArgs args;  } struct\_packet\_rsp\_regagent; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint32\_t agent\_id;  } struct\_packet\_cmd\_event; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  uint32\_t index;  } struct\_packet\_cmd\_lateinit; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_lateinit; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  TC\_NS\_Time tcNsTime;  } struct\_packet\_cmd\_synctime; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_synctime; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint8\_t cert\_buffer[CERT\_BUF\_MAX\_SIZE];  } struct\_packet\_cmd\_login; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  } struct\_packet\_cmd\_login\_non; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_login; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  struct SecLoadIoctlStruct ioctlArg;  } struct\_packet\_cmd\_load\_sec; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  struct SecLoadIoctlStruct ioctlArg;  } struct\_packet\_rsp\_load\_sec; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  TC\_NS\_ClientContext cliContext;  } struct\_packet\_cmd\_session; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  TC\_NS\_ClientContext cliContext;  } struct\_packet\_rsp\_session; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  int32\_t err\_flag;  int32\_t is\_fragment;  uint32\_t fragment\_block\_num;  uint32\_t vm\_page\_size;  uint64\_t block\_addrs[4];//qemu and proxy don't use  uint32\_t block\_size[4];  unsigned long long addrs[4]; //used by ref mem mmap  TC\_NS\_ClientContext cliContext;  } struct\_packet\_cmd\_send\_cmd; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  TC\_NS\_ClientContext cliContext;  } struct\_packet\_rsp\_send\_cmd; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  \_\_s32 cpu\_index;  TC\_NS\_ClientContext cliContext;  pid\_t pid;  } struct\_packet\_cmd\_cancel\_cmd; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  TC\_NS\_ClientContext cliContext;  } struct\_packet\_rsp\_cancel\_cmd; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  int32\_t ptzfd;  uint64\_t buffer;  uint32\_t size;  uint32\_t offset;  } struct\_packet\_cmd\_mmap; | 数据结构包含调用tzdriver接口的参数 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_mmap; | 数据结构包含传递给vtzdriver的返回值 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t cmd;  uint32\_t seq\_num;  } struct\_packet\_cmd\_nothing; | 数据结构与tzdriver无关 |
| typedef struct {  uint32\_t packet\_size;  uint32\_t seq\_num;  uint32\_t ret;  } struct\_packet\_rsp\_nothing; | 数据结构包含传递给vtzdriver的返回值 |

1. tzdriver(patch)设计与实现

由于pid在host侧的唯一性，故使用qemu的pid作为VM的vmid/nsid以区别不同虚拟机以及容器。vmid/nsid仅在传送open tzdriver的命令包时在qemu层通过getpid()获取。

由于vtz\_proxy与VM（qemu）是不同的进程，vtz\_proxy无法直接读写VM（qemu）的地址空间，需要借助tzdriver在内核态读写VM。一个进程读写另一个进程的地址空间可以在内核中调用access\_process\_vm()实现。

int access\_process\_vm(struct task\_struct \*tsk, unsigned long addr, void \*buf, int len, unsigned int gup\_flags) 需要获取进程task\_struct，而task\_struct可以通过pid获取。为减少开销和避免对tzdriver接口参数的调整，在struct tc\_ns\_dev\_file中新增了三个字段 uint32\_t vmpid、 bool isVM、 uint32\_t vm\_page\_size。vmpid存储VM的pid，isVM标识VM，vm\_page\_size用于解决switchless场景下VM和host PAGE\_SIZE大小不一致的页面映射。新增一个接口TC\_NS\_CLIENT\_IOCTL\_SET\_VM\_FLAG用于设置vmpid、isVM。

为了更方便地在tzdriver中读写VM，新增read\_from\_VMclient()与write\_to\_VMclient()，两者功能同read\_from\_client()与write\_to\_client()类似，只不过读写对象是VM。

针对switchless场景VM与Host PAGE\_SIZE不一致的情况，新增fill\_vm\_shared\_mem\_info\_block()用于获取多个虚拟地址的PAGES。