|  |  |
| --- | --- |
| 时间戳 | 时间戳表示SVMK启动以来的时间值，每次启动时会对该值进行清0。 |
| tick | tick是系统的基本时间单位，是定时等待时间、时间窗口时间、的基本时间度量单位 |
| 核心态 | 在DeltaUnity中核心态是系统特权级状态，程序在此状态下可以执行特权级操作 |
| 用户态 | 在DeltaUnity中用户态是系统的非特权级状态，程序在此状态下只可以执行用户级操作 |
| 核心分区 | 运行在核心态的分区 |
| 应用分区 | 运行在用户态的分区 |
| IDLE分区 | 运行在核心态，是系统的空闲分区 |
| 用户空间 | 用户态程序可以访问的空间 |
| 核心空间 | 核心态程序可以访问的空间 |

/\*

分区：appVM，kerVM，idlVM

<NAME,ID>

ID

状态：休眠、就绪、运行、等待、挂起、挂等

分区管理：

\*/

Machine APPVMADT{

appVM;

kerVM;

idlVM;

curVM//当前分区

appVMLogicAddress;//应用分区的逻辑地址范围

canAccessInterfaceSet;// 分区所配特权接口集合

exceptionVMSet;//产生异常的分区集合

schedulingTable;//时间调度表

SSTWakenUpType;//时间调度表唤醒类型，包含基本、定时、时间窗或时间调度表启动唤醒类型

//获取应用分区的ID号

1RK.2.1.T\_VMK\_ReturnCode vmkGetVMID(

T\_UBYTE \*ubpName,

T\_UWORD \*uwpVMID

){

If(\*ubpName==NULL){

\*uwpVMID=curVM<ID>;

return VMK\_OK;

}

if(NoValidName(ubpName)){

return VMK\_INVALID\_ADDRESS;

}

if(vm.<NAME:ubpName>∈appVM){

return VMK\_INVALID\_NAME;

}

\*uwpVMID=vm.<ID>;

return VMK\_OK;

}

A0=ω(vm'0, uwpVMID'0){1RK.2.1.T\_VMK\_ReturnCode vmkGetVMID(

T\_UBYTE \*ubpName,

T\_UWORD \*uwpVMID})}

A1=ω(vm'0, uwpVMID'0){

If(\*ubpName==NULL){

\*uwpVMID=curVM<ID>;

return VMK\_OK;}}{Q1}

A2= ubpName==NULL→ω(vm'0, uwpVMID'0){ uwpVMID=curVM<ID>;

return VMK\_OK;}{Q1}

A3=ω(vm'0, uwpVMID'0){ uwpVMID=curVM<ID>;return VMK\_OK;}

→ uwpVMID'1== curVM<ID>∧vmkGetVMID( )= VMK\_OK

A4=\* ubpName==NULL→ uwpVMID'1== curVM<ID>∧vmkGetVMID( )= VMK\_OK

A5=\* ubpName!=NULL→ω(vm'0, uwpVMID'0){Q1}

A6=ω(vm'0, uwpVMID'0){Q1}

→ω(vm'0){if(NoValidName(ubpName)){

return VMK\_INVALID\_ADDRESS;}}{Q2}

A7= ω(vm'0, uwpVMID'0){if(NoValidName(ubpName)){return VMK\_INVALID\_ADDRESS;}}{Q2}

→( (NoValidName(ubpName) →ω(vm'0, uwpVMID'0){ return VMK\_INVALID\_ADDRESS;})

A8=\* ubpName!=NULL

→( (NoValidName(ubpName) →ω(vm'0, uwpVMID'0){ return VMK\_INVALID\_ADDRESS;})

A9=ω(vm'0, uwpVMID'0){ return VMK\_INVALID\_ADDRESS;})

→ vmkGetVMID()=VMK\_INVALID\_ADDRESS

A10= \*ubpName!=NULL∧NoValidName(ubpName) → vmkGetVMID()=VMK\_INVALID\_ADDRESS

A11= ω(vm'0, uwpVMID'0){if(NoValidName(ubpName)){return VMK\_INVALID\_ADDRESS;}}{Q2}

→( !(NoValidName(ubpName) →ω(vm'0, uwpVMID'0){Q2})

A12=\* ubpName!=NULL∧ !(NoValidName(ubpName)

→ω(vm'0, uwpVMID'0){Q2})

A13= \*ubpName!=NULL∧ !(NoValidName(ubpName)

→ω(vm'0, uwpVMID'0){ if(vm.<NAME:ubpName>∈appVM){

return VMK\_INVALID\_NAME;}}){Q3}

A14=ω(vm'0, uwpVMID'0){ if(vm.<NAME:ubpName>∉appVM){

return VMK\_INVALID\_NAME;}}){Q3}

→ vm'1.<NAME:ubpName>∉appVM→ω(vm'1, uwpVMID'0){ return VMK\_INVALID\_NAME }

A15=ω(vm'1, uwpVMID'0){ return VMK\_INVALID\_NAME }

→ vmkGetVMID= VMK\_INVALID\_NAME

A16= ubpName!=NULL∧ !(NoValidName(ubpName) ∧ vm'1.<NAME:ubpName>∉appVM

→ vmkGetVMID= VMK\_INVALID\_NAME

A17=ω(vm'0, uwpVMID'0){ if(vm.<NAME:ubpName>∉appVM){

return VMK\_INVALID\_NAME;}}){Q3}

→ vm'1.<NAME:ubpName>∈appVM→ω(vm'1, uwpVMID'0){ Q3}

A18=\* ubpName!=NULL∧!(NoValidName(ubpName)

A19=ω(vm'1, uwpVMID'0){Q3}

→\*uwpVMID==vm.<ID>∧ω(vm'1, uwpVMID'1){return VMK\_OK;}

A20=ω(vm'1, uwpVMID'1){return VMK\_OK;}

→ vmkGetVMID( )= VMK\_OK

A21=\* ubpName!=NULL∧!(NoValidName(ubpName)∧ vm'1.<NAME:ubpName>∈appVM

→\* uwpVMID'1==vm.<ID>∧ vmkGetVMID( )= VMK\_OK

功能结果表达式：

A4=\* ubpName==NULL→ \*uwpVMID'1== curVM<ID>∧vmkGetVMID( )= VMK\_OK

A10=\* ubpName!=NULL∧NoValidName(ubpName) → vmkGetVMID()=VMK\_INVALID\_ADDRESS

A16=\* ubpName!=NULL∧!(NoValidName(ubpName) ∧vm'1.<NAME:ubpName>∉appVM

→ vmkGetVMID= VMK\_INVALID\_NAME

### RK.2.1 vmkSyscallGetVMID

T\_VMK\_ReturnCode vmkSyscallGetVMID(

T\_UBYTE \* name,

T\_UWORD \* vmID

){

if(!checkValidAddress(vmID)){

return VMK\_INVALID\_ADDRESS;

}

If(\*name==NULL){

\* vmID =curVM<ID>;

return VMK\_OK;

}

if(!checkValidAddress(name)){

return VMK\_INVALID\_ADDRESS;

}

vm.<NAME:\*name>∈appVM;

if(vm==NULL){

return VMK\_INVALID\_NAME;

}

\* vmID =vm.<ID>;

return VMK\_OK;

}

### RK.2.2 vmkGetVMID

T\_VMK\_ReturnCode vmkInGetVMID(

T\_UBYTE \*name,

T\_UWORD \*vmID

){

if(\*vmID==NULL)

return VMK\_INVALID\_ADDRESS;

else {

if(\*name ==NULL){

if(curVM∈idleVM){ //IDLE虚拟机执行此接口

return VMK\_INVALID\_NAME;

}

else if(curVM∈appVM∪kerVM){ //核心虚拟机或应用虚拟机执行此接口

\*vmID =curVMID;

return VMK\_OK;

}

}

else{

vm.<NAME:ubpName>∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_NAME;

}

\*uwpVMID=vm.<ID>;

return VMK\_OK;.

}

}

}

### RK.2.3 vmkSyscallGetVMInfo

T\_VMK\_ReturnCode vmkSyscallGetVMInfo (

T\_UWORD vmID,

T\_VMK\_VMBasicInfo \* buf

){

if(\*buf ==NULL)

return VMK\_INVALID\_ADDRESS;

else{

if(!checkValidAddress(buf)){

return VMK\_INVALID\_ADDRESS;

}

if(vmID ==VMK\_NULL\_VID){

\* buf = curVM< INFO >;

return VMK\_OK;

}

else{

vm.<ID: vmID >∈appVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

\* buf = vm.<INFO>;

return VMK\_OK;

}

}

}

### RK.2.4 vmkGetVMInfo

获取指定虚拟机的信息，虚拟机包括核心虚拟机和应用虚拟机。

T\_VMK\_ReturnCode vmkGetVMInfo(

T\_UWORD vmID,

T\_VMK\_VMBasicInfo \*buf

) {

if(\*buf ==NULL)

return VMK\_INVALID\_ADDRESS;

else{

if(vmID ==VMK\_NULL\_VID){

if(curVM∈idleVM) //IDLE虚拟机执行此接口

return VMK\_INVALID\_ID;

else if(curVM∈appVM∪kerVM) { //核心虚拟机或应用虚拟机执行此接口

\* buf = curVM< INFO >;

return VMK\_OK;

}

}

else{

vm.<ID: vmID >∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

\* buf = vm.<INFO>;

return VMK\_OK;

}

}

}

### RK.2.5 vmkSyscallGetVMName

T\_VMK\_ReturnCode vmkSyscallGetVMName (

T\_UWORD vmID,

T\_UBYTE \* name

){

if(!checkValidAddress(name)){// 33个字节地址空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

If(uwVMID ==VMK\_NULL\_VID){

\* name = curVM. < NAME >;

return VMK\_OK;

}

else{

vm.<ID: vmID >∈appVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

\* name =vm.<NAME>;

return VMK\_OK;

}

}

### RK.2.6 vmkGetVMName

T\_VMK\_ReturnCode vmkGetVMName(

T\_UWORD vmID,

T\_UBYTE \*name

){

If(\*name==NULL)

return VMK\_INVALID\_ADDRES;

else{

If(vmID ==VMK\_NULL\_VID){

if(curVM∈idleVM) //IDLE虚拟机执行此接口

return VMK\_INVALID\_ID;

else if(curVM∈appVM∪kerVM) { //核心虚拟机或应用虚拟机执行此接口

\* name = curVM< NAME >;

return VMK\_OK;

}

}

else{

vm.<ID: vmID >∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

\* name =vm.<NAME>;

return VMK\_OK;

}

}

}

### RK.2.7 vmkSyscallGetVMPhysicalAddress

//获取逻辑地址对应的物理地址

Set vmSpace;// 虚拟机空间内存块表

T\_VMK\_ReturnCode vmkSyscallGetVMPhysicalAddress(

T\_UWORD vmID,

T\_UWORD logicalAddress,

T\_VOID\* phyAdd

){

vm.<ID:uwVMID>∈appVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

if(!checkValidAddress(phyAdd)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

if(!checkMappingAddress(logicalAddress,vmID)){// 地址在<vmID>指定的应用虚拟机空间中未被映射，应返回VMK\_FAIL

return VMK\_ FAIL;

}

\* phyAdd= computePhyAdd(vmID,logicalAddress );

Return VMK\_OK

}

### RK.2.8 vmkGetVMPhysicalAddress

T\_VMK\_ReturnCode vmkGetVMPhysicalAddress(

T\_UWORD vmID,

T\_UWORD logicalAddress,

T\_VOID \*phyAdd

) {

vm.<ID:uwVMID>∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

If(\*phyAdd==NULL)

Return VMK\_INVALID\_ADDRESS;

if(!checkMappingAddress(logicalAddress,vmID)){// <logicalAddress>表示的地址在<vmID>指定的虚拟机空间中未被映射

return VMK\_ FAIL;

}

\* phyAdd= computePhyAdd(vmID,logicalAddress );

Return VMK\_OK;

}

### RK.2.9 vmkSyscallGetVMStatus

curAppVM//当前应用虚拟机

T\_VMK\_ReturnCode vmkSyscallGetVMStatus(

T\_UWORD vmID,

T\_VMK\_VMStatus \*status

){

if(!checkValidAddress(status)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

If(vmID ==VMK\_NULL\_VID){

\* status = curAppVM.<STATUS>;

return VMK\_OK;

}

else{

vm.<ID:uwVMID>∈appVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}

else{

\* status = vm.< STATUS >;

return VMK\_OK;

}

}

}

### RK.2.10 vmkGetVMStatus

T\_VMK\_ReturnCode vmkGetVMStatus(

T\_UWORD vmID,

T\_VMK\_VMStatus \*status

){

if(\*status==NULL){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

If(vmID ==VMK\_NULL\_VID){

If(curVM∈idleVM){

\* status = curVM.<STATUS>;

return VMK\_INVALID\_ID;

}else if(curVM∈idleVM∪kerVM){

\* status = curVM.<STATUS>;

return VMK\_ OK;

}

}

else{

vm.<ID:uwVMID>∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

}else{

\* status = vm.< STATUS >;

return VMK\_OK;

}

}

}

### RK.2.11 vmkSyscallGetExceptionVM

exceptionAppVMSet //管理分区中存放产生异常应用分区的队列

T\_VMK\_ReturnCode vmkSyscallGetExceptionVM(

T\_UWORD \*vmID

){

if(!checkValidAddress(vmID)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

vm∈exceptionAppVMSet;//应用队列中取出一个元素

if(vm==NULL){

return VMK\_INVALID\_STATE;

}else{

if(vm. < STATUS >== VMK\_VM\_DEBUGING){

return VMK\_INVALID\_USER;

}else{

If(curVM.isNotManagePartition(vm)){// 调用此接口的虚拟机不是该应用虚拟机///的管理虚拟机

Return VMK\_INVALID\_USER;

}else{

\*vmID=vm.<ID>;

Return VMK\_OK;

}

}

}

}

### RK.2.12 vmkGetExceptionVM

T\_VMK\_ReturnCode vmkGetExceptionVM(

T\_UWORD \*vmID

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(vmID==NULL){

Return VMK\_INVALID\_ADDRESS;

}

vm∈exceptionAppVMSet;//应用队列中取出一个元素

if(vm==NULL){

return VMK\_INVALID\_STATE;

}else{

if(vm. < STATUS >== VMK\_VM\_DEBUGING){

if(curVM.isSupportDebugging()==TRUE&&curVM∈kerVM){

\*vmID=vm.<ID>;

Return VMK\_OK;

}else{

Return VMK\_INVALID\_USER;

}

}

}

}

### RK.2.13 vmkSyscallGetTTSID

curTTS//当前时间调度表

TTSset//时间调度表集合

T\_VMK\_ReturnCode vmkSyscallGetTTSID(

T\_UBYTE \*ttsName,

T\_UWORD \*ttsID

){

if(!checkValidAddress(ttsID)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

If(\*ttsName==NULL){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\*ttsID= curTTS.<ID>;

Return VMK\_OK;

}

}else{

if(!checkValidAddress(ttsName)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

tts.<NAME: \*ttsName >∈ TTSset;

if(tts==NULL){

return VMK\_INVALID\_NAME;

}else{

\* ttsID = tts.<ID>;

return VMK\_OK;

}

}

}

### RK.2.14 vmkGetTTSID

T\_VMK\_ReturnCode vmkGetTTSID(

T\_UBYTE \*ttsName,

T\_UWORD \*ttsID

){

if(\*ttsID ==NULL)

Return VMK\_INVALID\_ ADDRESS;

If(\*ttsName==NULL){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\*ttsID= curTTS.<ID>;

Return VMK\_OK;

}

} else{

tts.<NAME: \*ttsName >∈ TTSset;

if(tts==NULL){

return VMK\_INVALID\_NAME;

}else{

\* ttsID = tts.<ID>;

return VMK\_OK;

}

}

}

### RK.2.15 vmkSyscallGetTTSInfo

T\_VMK\_ReturnCode vmkSyscallGetTTSInfo(

T\_UWORD ttsID,

T\_VMK\_TTSBasicInfo \*buf

){

if(!checkValidAddress(buf)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL){

return VMK\_INVALID\_ ID;

}

if(ttsID==VMK\_NULL\_TTS){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\*buf=curTTS.< TTSBasicInfo >;

return VMK\_ OK ;

}

}else{

if(tts!=NULL){

\*buf=tts.< TTSBasicInfo >;

return VMK\_ OK ;

}

}

}

### RK.2.16 vmkGetTTSInfo

T\_VMK\_ReturnCode vmkGetTTSInfo(

T\_UWORD ttsID,

T\_VMK\_TTSBasicInfo \*buf

){

if(\*buf ==NULL){

return VMK\_INVALID\_ ADDRESS;

}

if(ttsID==VMK\_NULL\_TTS){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\*buf=curTTS.< TTSBasicInfo >;

return VMK\_ OK ;

}

}else{

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL)

return VMK\_INVALID\_ID

else{

\*buf=tts.< TTSBasicInfo >;

return VMK\_ OK ;

}

}

}

### RK.2.17 vmkSyscallGetTTSName

T\_VMK\_ReturnCode vmkSyscallGetTTSName(

T\_UWORD ttsID,

T\_UBYTE \*ttsName

){

if(!checkValidAddress(ttsName)){//空间没有被映射或地址空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

}

if(ttsID==VMK\_NULL\_TTS){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\* ttsName=curTTS.< NAME >;

return VMK\_ OK ;

}

}else{

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL)

return VMK\_INVALID\_ID

else{

\* ttsName =tts.< NAME >;

return VMK\_ OK ;

}

}

}

### RK.2.18 vmkGetTTSName

T\_VMK\_ReturnCode vmkGetTTSName(

T\_UWORD ttsID,

T\_UBYTE \*ttsName

){

if(\*ttsName ==NULL)

Return VMK\_INVALID\_ ADDRES;

if(ttsID==VMK\_NULL\_TTS){

if(curTTS==NULL)

Return VMK\_INVALID\_STATE;

else{

\* ttsName=curTTS.< NAME >;

return VMK\_ OK ;

}

}else{

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL)

return VMK\_INVALID\_ID

else{

\* ttsName =tts.< NAME >;

return VMK\_ OK ;

}

}

}

### RK.2.19 vmkGetVMCount

T\_VMK\_ReturnCode vmkGetVMCount(

T\_UWORD \*vmCount

){

If(\*uwpVMCount==NULL){

returnVMK\_INVALID\_ADDRESS;

}

\*uwpVMCount=getSize(appVM);

returnVMK\_OK;

}

### RK.2.20 vmkSyscallHaltVM

interruptProcessSet//虚拟外部中断或虚拟服务中断集合

T\_VMK\_ReturnCode vmkSyscallHaltVM(

T\_VMK\_WakeupType wakeupType,

T\_UWORD ticks

){

If(wakeupType!= VMK\_WAKEUP\_BASIC || tWakeupType != VMK\_WAKEUP\_TIME || tWakeupType!= VMK\_WAKEUP\_TTS\_START || tWakeupType!= VMK\_WAKEUP\_TTS\_WINDOW)

Return VMK\_INVALID\_TYPE;

If(tWakeupType == VMK\_WAKEUP\_TIME){// 唤醒类型为定时唤醒类型时

If(ticks ==0){

Return VMK\_INVALID\_TIME;

}else{

Waiting(ticks);// <ticks>等待时间完毕

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}

}

if(tWakeupType== VMK\_WAKEUP\_TTS\_START){// 唤醒类型为时间调度表调度启动唤醒类型

If( curVM. VMBasicInfo < schedType >!= VMK\_SCHED\_PRIORITY){// 当前分区的调度类型不是优先级调度

Return VMK\_INVALID\_TYPE;

}else{

curVM. VMBasicInfo < schedType >= VMK\_SCHED\_TIME\_TABLE;

waiting(wakeupWindows)// 当前应用虚拟机的时间窗口来临时

curVM.callRunning();//运行当前应用虚拟机

return VMK\_OK;

}

}

If(tWakeupType== VMK\_WAKEUP\_TTS\_WINDOW){// 唤醒类型为时间窗唤醒类型

If( curVM. VMBasicInfo < schedType >!= VMK\_SCHED\_TIME\_TABLE){// 当前分区的调度类型不是时间调度表调度

Return VMK\_INVALID\_TYPE;

}

CurVM.giveupCPU();

waiting(wakeupWindows)// 当前应用虚拟机的下一个时间窗口来临时

curVM.callRunning();//运行当前应用虚拟机

return VMK\_OK;

}

If(tWakeupType == VMK\_WAKEUP\_BASIC){// 唤醒类型为基本唤醒类型时

if(ticks==0){

if(curVM.interruptProcessSet!=NULL)

Return VMK\_OK;

else{

curVM.VMStatus=VMK\_VM\_WAITING;// 应将当前应用虚拟机的状态变为等待态

waiting(interruptProcess)// 当有虚拟外部中断或虚拟服务中断投递到此虚拟机时

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}

}else{

if(curVM.interruptProcessSet!=NULL)

Return VMK\_OK;

else{

curVM.VMStatus=VMK\_VM\_WAITING;// 应将当前应用虚拟机的状态变为等待态

if(hasInterruptProcess(curVM,ticks)){// 在<ticks>指定的等待时间内有虚拟外部中断或虚拟服务中断投递到此虚拟机时

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}else{

awakenVM(curVM);// 在<ticks>等待时间完毕后，应唤醒当前应用虚拟机

return VMK\_OK;

}

}

}

### RK.2.21 vmkHaltVM

T\_VMK\_ReturnCode vmkHaltVM(

T\_VMK\_WakeupType wakeupType,

T\_UWORD ticks

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(curVM∈ idlVM){

Return VMK\_INVALID\_USER;

}

If(wakeupType!= VMK\_WAKEUP\_BASIC || tWakeupType != VMK\_WAKEUP\_TIME)

Return VMK\_INVALID\_TYPE;

If(tWakeupType != VMK\_WAKEUP\_TIME){

If(ticks==0)

Return VMK\_INVALID\_TIME;

Else{

Waiting(ticks);// <ticks>等待时间完毕

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}

}

If(tWakeupType != VMK\_WAKEUP\_BASIC){

if(ticks==0){

if(curVM.interruptProcessSet!=NULL)

Return VMK\_OK;

else{

curVM.VMStatus=VMK\_VM\_WAITING;// 应将当前应用虚拟机的状态变为等待态

waiting(interruptProcess)// 当有虚拟外部中断或虚拟服务中断投递到此虚拟机时

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}

}else{

if(curVM.interruptProcessSet!=NULL)

Return VMK\_OK;

else{

curVM.VMStatus=VMK\_VM\_WAITING;// 应将当前应用虚拟机的状态变为等待态

if(hasInterruptProcess(curVM,ticks)){// 在<ticks>指定的等待时间内有虚拟外部中断或虚拟服务中断投递到此虚拟机时

awakenVM(curVM);// 唤醒当前应用虚拟机

return VMK\_OK;

}else{

awakenVM(curVM);// 在<ticks>等待时间完毕后，应唤醒当前应用虚拟机

return VMK\_OK;

}

}

}

}

}

### RK.2.22 vmkSyscallStartVM

curInterface//当前被调用的接口

canAccessInterfaceSet//虚拟机分配的特权接口集合

T\_VMK\_ReturnCode vmkSyscallStartVM(

T\_UWORD vmID,

T\_VMK\_SchedType type

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是应用分区，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(type !=VMK\_SCHED\_DEFAULT|| type !=VMK\_SCHED\_TIME\_TABLE|| type !=VMK\_SCHED\_PRIORITY)// 如果<type>不是默认调度类型、优先级调度类型和时间调度表调度类型

Return VMK\_INVALID\_TYPE;

If(vm. VMStatus!= VMK\_VM\_DORMANT)// 如果<vmID>指定的应用虚拟机的状态不是休眠态

Return VMK\_INVALID\_STATE;

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<vmID>指定的应用虚拟机的状态为休眠态，并且<type>为优先级调度类型或时间调度表类型

Set(curVM. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

If(type ==VMK\_SCHED\_ DEFAULT){// 如果<vmID>指定的应用虚拟机的状态为休眠态，并且<type>为默认调度类型

Set(curVM. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

### RK.2.23 vmkStartVM

T\_VMK\_ReturnCode vmkStartVM(

T\_UWORD vmID,

T\_VMK\_SchedType type,

BOOL isSuspended

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

vm.<ID: uwpVMID>∈appVM∪kerVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(type !=VMK\_SCHED\_DEFAULT|| type !=VMK\_SCHED\_TIME\_TABLE|| type !=VMK\_SCHED\_PRIORITY)// 如果<type>不是默认调度类型、优先级调度类型和时间调度表调度类型

Return VMK\_INVALID\_TYPE;

If(vm∈appVM&& type ==VMK\_SCHED\_TIME\_TABLE)// 如果<vmID>指定的虚拟机为核心虚拟机，并且<type>为时间调度表调度类型，应返回VMK\_INVALID\_TYPE

Return VMK\_INVALID\_TYPE;

If(vm. VMStatus!= VMK\_VM\_DORMANT)如果<vmID>指定的核心虚拟机或用户虚拟机的状态不是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(type !=VMK\_SCHED\_DEFAULT){

If(isSuspended==FALSE){

Set(curVM. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(curVM. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

return VMK\_INVALID\_OK;

}

}else{

If(isSuspended==FALSE){

Set(curVM. VMBasicInfo < schedType >, last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(curVM. VMBasicInfo < schedType >, last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

return VMK\_INVALID\_OK;

}

}

}

### RK.2.24 vmkSyscallStopVM

T\_VMK\_ReturnCode vmkSyscallStopVM(

T\_UWORD vmID

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_RUNNING){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){//如果<vmID>指定的应用虚拟机的状态为挂起态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){//如果<vmID>指定的应用虚拟机的状态为挂等态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

}

### RK.2.25 vmkStopVM

T\_VMK\_ReturnCode vmkStopVM(

T\_UWORD vmID

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

vm.<ID: uwpVMID>∈appVM∪kerVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_RUNNING){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){//如果<vmID>指定的应用虚拟机的状态为挂起态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){//如果<vmID>指定的应用虚拟机的状态为挂等态

vm.VMStatus= VMK\_VM\_ DORMANT;// 状态变为休眠态

Return VMK\_OK;

}

}

### RK.2.26 vmkSyscallResetVM

T\_VMK\_ReturnCode vmkSyscallResetVM(

T\_UWORD vmID,

T\_VMK\_SchedType type

){

If(curInterface ∉ curVM.canAccessInterfaceSet)){ // 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

If(type !=VMK\_SCHED\_DEFAULT|| type !=VMK\_SCHED\_TIME\_TABLE|| type !=VMK\_SCHED\_PRIORITY)// 如果<type>不是默认调度类型、优先级调度类型和时间调度表调度类型

Return VMK\_INVALID\_TYPE;

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vmID==VMK\_NULL\_VID){

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ RUNNING){//如果<vmID>指定的应用虚拟机的状态为运行态

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){//如果<vmID>指定的应用虚拟机的状态为挂起态

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){//如果<vmID>指定的应用虚拟机的状态为挂等态

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

}

### RK.2.27 vmkResetVM

T\_VMK\_ReturnCode vmkResetVM(

T\_UWORD vmID,

T\_VMK\_SchedType type

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(type !=VMK\_SCHED\_DEFAULT|| type !=VMK\_SCHED\_TIME\_TABLE|| type !=VMK\_SCHED\_PRIORITY)// 如果<type>不是默认调度类型、优先级调度类型和时间调度表调度类型

Return VMK\_INVALID\_TYPE;

vm.<ID: uwpVMID>∈appVM∪kerVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(vm∈kerVM&& type ==VMK\_SCHED\_TIME\_TABLE)// 如果<vmID>指定的虚拟机为核心虚拟机，并且<type>为时间调度表调度类型，应返回VMK\_INVALID\_TYPE

Return VMK\_INVALID\_TYPE;

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vmID==VMK\_NULL\_VID){

If(type ==VMK\_SCHED\_TIME\_TABLE|| type ==VMK\_SCHED\_PRIORITY){// 如果<type>是优先级调度类型或时间调度表调度类型

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else if(type ==VMK\_ SCHED\_DEFAULT){

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

If(type !=VMK\_ SCHED\_DEFAULT){// 如果<type>不是默认调度

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ RUNNING){//如果<vmID>指定的应用虚拟机的状态为运行态

If(type !=VMK\_ SCHED\_DEFAULT){// 如果<type>不是默认调度

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

If(type !=VMK\_ SCHED\_DEFAULT){// 如果<type>不是默认调度

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){//如果<vmID>指定的应用虚拟机的状态为挂起态

If(type !=VMK\_ SCHED\_DEFAULT){// 如果<type>不是默认调度

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){//如果<vmID>指定的应用虚拟机的状态为挂等态

If(type !=VMK\_ SCHED\_DEFAULT){// 如果<type>不是默认调度

Set(vm. VMBasicInfo < schedType >,type);//将该应用虚拟机的调度类型变为<type>

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}else{

Set(vm. VMBasicInfo < schedType >,last\_SchedType\_type);//将该应用虚拟机的调度类型变为最近一次运行的调度类型

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_INVALID\_OK;

}

}

}

### RK.2.28 vmkSyscallSuspendVM

T\_VMK\_ReturnCode vmkSyscallSuspendVM(

T\_UWORD vmID

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 根据当前应用虚拟机所配特权接口判断当前虚拟机没有执行此接口的权限，应返回VMK\_INVALID\_USER

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的应用虚拟机不存在

return VMK\_INVALID\_ID;

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND)//如果<vmID>指定的用户虚拟机的状态是挂起态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING)//如果<vmID>指定的用户虚拟机的状态是挂等态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_RUNNING){//如果<vmID>指定的应用虚拟机的状态为运行态

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

vm.VMStatus= VMK\_VM\_ SUSPEND\_and\_ WAITING;// 状态变为挂等态

Return VMK\_OK;

}

}

### RK.2.29 vmkSuspendVM

T\_VMK\_ReturnCode vmkSuspendVM(

T\_UWORD vmID

){

vm.<ID: uwpVMID>∈appVM∪kerVM;

if(vm==NULL){// 如果不存在<vmID>指定的核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(vm. VMStatus== VMK\_VM\_DORMANT)//如果<vmID>指定的用户虚拟机的状态是休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND)//如果<vmID>指定的用户虚拟机的状态是挂起态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING)//如果<vmID>指定的用户虚拟机的状态是挂等态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY){//如果<vmID>指定的应用虚拟机的状态为就绪态

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_RUNNING){//如果<vmID>指定的应用虚拟机的状态为运行态

vm.VMStatus= VMK\_VM\_ SUSPEND;// 状态变为挂起态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_WAITING){//如果<vmID>指定的应用虚拟机的状态为等待态

vm.VMStatus= VMK\_VM\_ SUSPEND\_and\_ WAITING;// 状态变为挂等态

Return VMK\_OK;

}

}

### RK.2.30 vmkSyscallResumeVM

T\_VMK\_ReturnCode vmkSyscallResumeVM(

T\_UWORD vmID

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 根据当前应用虚拟机所配特权接口判断当前虚拟机没有执行此接口的权限，应返回VMK\_INVALID\_USER

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的应用虚拟机不存在, 应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

If(vm. VMStatus== VMK\_VM\_DORMANT)// 如果<vmID>指定应用虚拟机的状态为休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY)// 如果<vmID>指定应用虚拟机的状态为休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ RUNNING)// 如果<vmID>指定应用虚拟机的状态为休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ WAITING)// 如果<vmID>指定应用虚拟机的状态为休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. isSuspendByKerException())//如果<vmID>指定应用虚拟机是被核心级异常处理程序挂起的，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){// 如果<vmID>指定应用虚拟机的状态为挂等态，应将该应用虚拟机的状态变为等待态，并返回VMK\_OK

vm.VMStatus= VMK\_VM\_WAITING;// 状态变为等待态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){// 如果<vmID>指定应用虚拟机的状态为挂起态，应将该应用虚拟机的状态变为就绪态，并返回VMK\_OK

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

Return VMK\_OK;

}

}

### RK.2.31 vmkResumeVM

T\_VMK\_ReturnCode vmkResumeVM(

T\_UWORD vmID

){

vm.<ID: uwpVMID>∈appVM∪kerVM;

if(vm==NULL){// 如果不存在<vmID>指定的核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

If(vm. VMStatus== VMK\_VM\_DORMANT)// 如果<vmID>指定应用虚拟机的状态为休眠态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ READY)// 如果<vmID>指定应用虚拟机的状态为就绪态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ RUNNING)// 如果<vmID>指定应用虚拟机的状态为运行态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ WAITING)// 如果<vmID>指定应用虚拟机的状态为等待态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. isSuspendByKerException())//如果<vmID>指定应用虚拟机是被核心级异常处理程序挂起的，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

If(vm. VMStatus== VMK\_VM\_ SUSPEND\_and\_WAITING){// 如果<vmID>指定应用虚拟机的状态为挂等态，应将该应用虚拟机的状态变为等待态，并返回VMK\_OK

vm.VMStatus= VMK\_VM\_WAITING;// 状态变为等待态

Return VMK\_OK;

}

If(vm. VMStatus== VMK\_VM\_ SUSPEND){// 如果<vmID>指定应用虚拟机的状态为挂起态，应将该应用虚拟机的状态变为就绪态，并返回VMK\_OK

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

Return VMK\_OK;

}

}

### RK.2.32 vmkSyscallStartTTS

TTSset//时间调度表集合

Timer.timeConfigType()//系统定时器中断的时间间隔配置

T\_VMK\_ReturnCode vmkSyscallStartTTS(

T\_UWORD ttsID

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 根据当前应用虚拟机所配特权接口判断当前虚拟机没有执行此接口的权限，应返回VMK\_INVALID\_USER

Return VMK\_INVALID\_USER;

}

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL){// 如果不存在<ttsID>指定的时间调度表，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ ID;

}

If(tts. TTSStatus!= VMK\_TTS\_STOPED){// 如果<ttsID>指定的时间调度表的状态不为停止态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

}else{

If(Timer.timeConfigType()!=one-shot){// 系统定时器中断的时间间隔配置为非one-shot方式

If(tts.getStartWatingTime()==0){

vm=tts.getFirstWindowVM();//取得时间调度表第一个时间窗口所指虚拟机

If(vm. VMBasicInfo < schedType >== VMK\_SCHED\_PRIORITY){

tts. TTSStatus= VMK\_TTS\_ ABORT; //应将该时间调度表变为放弃态

Return VMK\_ OK;

}else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus!= VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态不为就绪态

tts. TTSStatus= VMK\_TTS\_WAITING\_STATUS;// 将该时间调度表变为等待态，并返回VMK\_OK

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus== VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态为就绪态

tts. TTSStatus= VMK\_TTS\_ RUNNING;// 将该时间调度表变为运行态，并返回VMK\_OK

Return VMK\_ OK;

}else if(tts.firstWindowStartTime!=0){// 时间调度表第一个时间窗口的起始时间不为0

tts. TTSStatus= VMK\_TTS\_ IDLE;// 将该时间调度表变为空闲态，并返回VMK\_OK

Return VMK\_ OK;

}

}else{

wait(tts.getStartWatingTime());//等待指定的时间后

vmkSyscallStartTTS();//进行时间调度表的启动

Return VMK\_ OK;

}

}else{//系统定时器中断的时间间隔配置为one-shot方式

vm=tts.getFirstWindowVM();//取得时间调度表第一个时间窗口所指虚拟机

If(vm. VMBasicInfo < schedType >== VMK\_SCHED\_PRIORITY){

tts. TTSStatus= VMK\_TTS\_ ABORT; //应将该时间调度表变为放弃态

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus!= VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态不为就绪态

tts. TTSStatus= VMK\_TTS\_WAITING\_STATUS;// 将该时间调度表变为等待态，并返回VMK\_OK

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus== VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态为就绪态

tts. TTSStatus= VMK\_TTS\_ RUNNING;// 将该时间调度表变为运行态，并返回VMK\_OK

Return VMK\_ OK;

}else if(tts.firstWindowStartTime!=0){// 时间调度表第一个时间窗口的起始时间不为0

tts. TTSStatus= VMK\_TTS\_ IDLE;// 将该时间调度表变为空闲态，并返回VMK\_OK

Return VMK\_ OK;

}

}

}

}

### RK.2.33 vmkStartTTS

T\_VMK\_ReturnCode vmkStartTTS(

T\_UWORD ttsID

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

tts.<ID: ttsID >∈ TTSset;

if(tts==NULL){// 如果不存在<ttsID>指定的时间调度表，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ ID;

}

If(tts. TTSStatus!= VMK\_TTS\_STOPED){// 如果<ttsID>指定的时间调度表的状态不为停止态，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

} else{

If(Timer.timeConfigType()!=one-shot){// 系统定时器中断的时间间隔配置为非one-shot方式

If(tts.getStartWatingTime()==0){

vm=tts.getFirstWindowVM();//取得时间调度表第一个时间窗口所指虚拟机

If(vm. VMBasicInfo < schedType >== VMK\_SCHED\_PRIORITY){

tts. TTSStatus= VMK\_TTS\_ ABORT; //应将该时间调度表变为放弃态

Return VMK\_ OK;

}else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus!= VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态不为就绪态

tts. TTSStatus= VMK\_TTS\_WAITING\_STATUS;// 将该时间调度表变为等待态，并返回VMK\_OK

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus== VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态为就绪态

tts. TTSStatus= VMK\_TTS\_ RUNNING;// 将该时间调度表变为运行态，并返回VMK\_OK

Return VMK\_ OK;

}else if(tts.firstWindowStartTime!=0){// 时间调度表第一个时间窗口的起始时间不为0

tts. TTSStatus= VMK\_TTS\_ IDLE;// 将该时间调度表变为空闲态，并返回VMK\_OK

Return VMK\_ OK;

}

}else{

wait(tts.getStartWatingTime());//等待指定的时间后

vmkSyscallStartTTS();//进行时间调度表的启动

Return VMK\_ OK;

}

}else{//系统定时器中断的时间间隔配置为one-shot方式

vm=tts.getFirstWindowVM();//取得时间调度表第一个时间窗口所指虚拟机

If(vm. VMBasicInfo < schedType >== VMK\_SCHED\_PRIORITY){

tts. TTSStatus= VMK\_TTS\_ ABORT; //应将该时间调度表变为放弃态

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus!= VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态不为就绪态

tts. TTSStatus= VMK\_TTS\_WAITING\_STATUS;// 将该时间调度表变为等待态，并返回VMK\_OK

Return VMK\_ OK;

} else if(vm. VMBasicInfo < schedType >== VMK\_SCHED\_TIME\_TABLE && vm. VMStatus== VMK\_VM\_ READY){// 时间调度表第一个时间窗口所指虚拟机调度方式为时间调度表调度及虚拟机状态为就绪态

tts. TTSStatus= VMK\_TTS\_ RUNNING;// 将该时间调度表变为运行态，并返回VMK\_OK

Return VMK\_ OK;

}else if(tts.firstWindowStartTime!=0){// 时间调度表第一个时间窗口的起始时间不为0

tts. TTSStatus= VMK\_TTS\_ IDLE;// 将该时间调度表变为空闲态，并返回VMK\_OK

Return VMK\_ OK;

}

}

}

}

### RK.2.34 vmkSyscallStopTTS

T\_VMK\_ReturnCode vmkSyscallStopTTS(

T\_VOID

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 根据当前应用虚拟机所配特权接口判断当前虚拟机没有执行此接口的权限，应返回VMK\_INVALID\_USER

Return VMK\_INVALID\_USER;

}

If(curTTS==NULL){

return VMK\_INVALID\_ STATE;

}else{

tts. TTSStatus!= VMK\_TTS\_STOPED//如果当前时间调度表存在，应将当前时间调度表的状态变为停止态，并返回VMK\_OK

return VMK\_OK;

}

}

### RK.2.35 vmkStopTTS

T\_VMK\_ReturnCode vmkStopTTS(

T\_VOID

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(curTTS==NULL){

return VMK\_INVALID\_ STATE;

}else{

tts. TTSStatus!= VMK\_TTS\_STOPED//如果当前时间调度表存在，应将当前时间调度表的状态变为停止态，并返回VMK\_OK

return VMK\_OK;

}

}

### RK.2.36 vmkSyscallSetVMPageAttribute

T\_VMK\_ReturnCode vmkSyscallSetVMPageAttribute(

T\_UWORD logicalAddr,

T\_UWORD size,

T\_UWORD attr

){

if(!checkValidAddress(logicalAddr)){// 如果<logicalAddr>指向的逻辑地址空间没有被映射或者指向的逻辑地址空间是用户态程序不可写访问的

return VMK\_INVALID\_ADDRESS;

}

if(size==0)

return VMK\_INVALID\_SIZE;

if(!addressOrderBy(logicalAddr,4K) || size%4096!=0)

return VMK\_INVALID\_ALIGNED;

if(attr!= MM\_UREAD\_ONLY)

return VMK\_FAIL;

set(curVM. VMBasicInfo< vmSpace >,attr);// 设置当前应用虚拟机的内存页访问属性为只读属性，并返回VMK\_OK

return VMK\_OK;

}

### RK.2.37 vmkSetVMPageAttribute

T\_VMK\_ReturnCode vmkSetVMPageAttribute(

T\_UWORD vmID,

T\_UWORD logicalAddr,

T\_UWORD size,

T\_UWORD attr

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

if(size==0)

return VMK\_INVALID\_SIZE;

if(!addressOrderBy(logicalAddr,4K) || size%4096!=0)

return VMK\_INVALID\_ALIGNED;

if(attr!= MM\_UREAD\_ONLY|| attr!= MM\_SREAD\_ONLY)

return VMK\_FAIL;

if(vmID ==VMK\_NULL\_VID){

if(!checkValidAddress(logicalAddr,curVM))// <logicalAddr>指定的逻辑地址在当前虚拟机的地址空间没有被映射，应返回VMK\_FAIL

return VMK\_FAIL;

vm=curVM;

}else{

vm.<ID: vmID >∈appVM∪kerVM;

if(vm==NULL){// 如果<vmID>不为VMK\_NULL\_VID，并且不存在<vmID>指定的核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

if(!checkValidAddress(logicalAddr,vm))// <logicalAddr>指定的逻辑地址在<vmID>指定虚拟机的地址空间没有被映射，应返回VMK\_FAIL

return VMK\_FAIL;

}

set(vm. VMBasicInfo< vmSpace >,attr);// 设置当前应用虚拟机的内存页访问属性为只读属性，并返回VMK\_OK

return VMK\_OK;

}

### RK.2.38 vmkSetVMStackProtect

T\_VMK\_ReturnCode vmkSetVMStackProtect(

T\_UWORD vmID,

T\_UWORD guardSize

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

if(vmID ==VMK\_NULL\_VID){

if(guardSize>curVM.vStackSize)// 如果<guardSize>大于虚拟机系统栈可用的大小

return VMK\_INVALID\_SIZE;

if(hasBeenProtected(curVM. vStack))// 如果需要设置的系统栈空间已经被保护

return VMK\_INVALID\_ STATE;

if(guardSize%4096!=0)

return VMK\_INVALID\_ ALIGNED;

if(!checkValidAddress(curVM. vStack)// 如果需要设置的栈空间是虚拟机正在使用的系统栈空间

return VMK\_INVALID\_ STATE;

setProtectedStack(curVM);// 应设置当前核心虚拟机或者IDLE虚拟机的系统栈保护，并返回VMK\_OK

return VMK\_ OK;

}else{//<vmID>不为VMK\_NULL\_VID

vm.<ID: vmID >∈appVM∪kerVM;

if(vm==NULL){// 如果<vmID>不为VMK\_NULL\_VID，并且不存在<vmID>指定的核心虚拟机和应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

if(guardSize>curVM.vStackSize)// 如果<guardSize>大于虚拟机系统栈可用的大小

return VMK\_INVALID\_SIZE;

if(hasBeenProtected(curVM. vStack))// 如果需要设置的系统栈空间已经被保护

return VMK\_INVALID\_ STATE;

if(guardSize%4096!=0)

return VMK\_INVALID\_ ALIGNED;

if(!checkValidAddress(curVM. vStack)// 如果需要设置的栈空间是虚拟机正在使用的系统栈空间

return VMK\_INVALID\_ STATE;

setProtectedStack(vm);// 应设置<vmID>指定的虚拟机系统栈保护，并返回VMK\_OK。

return VMK\_ OK;

}

}

### RK.4.1 vmkSyscallInitializeVint

T\_VMK\_ReturnCode vmkSyscallInitializeVint(

T\_VMK\_VMVintEnMask \*vintEnMask,

T\_VMK\_VINT\_HANDLER handler

){

if(!checkValidAddress(vintEnMask)// <vintEnMask>指向的虚拟中断屏蔽码空间没有被映射, 指向的虚拟中断屏蔽码空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

if(!checkValidAddress(handler)// < handler >指向的虚拟中断屏蔽码空间没有被映射, 指向的虚拟中断屏蔽码空间是用户态程序不可写访问

return VMK\_INVALID\_ADDRESS;

if(curVM. VMVintEnMask < isInitialized >==TRUE)// 如果当前应用虚拟机的虚拟中断已经初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

curVM. VMVintEnMask < exIntEnMask >=\* vintEnMask; //根据<vintEnMask>安装当前应用虚拟机的虚拟中断屏蔽码。

curVM.installVintAddress(handler);// 根据<handler>安装当前应用虚拟机的虚拟中断入口程序，并返回VMK\_OK

return VMK\_ OK;

}

### RK.4.2 vmkInitializeVint

T\_VMK\_ReturnCode vmkInitializeVint(

T\_VMK\_VMVintEnMask \*vintEnMask,

T\_VMK\_VINT\_HANDLER handler

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(vintEnMask==NULL|| handler==NULL){

Return VMK\_INVALID\_ADDRESS;

}

if(curVM. VMVintEnMask < isInitialized >==TRUE)// 如果当前应用虚拟机的虚拟中断已经初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

curVM. VMVintEnMask < exIntEnMask >=\* vintEnMask; //根据<vintEnMask>安装当前应用虚拟机的虚拟中断屏蔽码。

curVM.installVintAddress(handler);// 根据<handler>安装当前应用虚拟机的虚拟中断入口程序，并返回VMK\_OK

return VMK\_ OK;

}

### RK.4.3 vmkSyscallDisablePIC

interruptMappingSet;//中断映射表

T\_VMK\_ReturnCode vmkSyscallDisablePIC(

T\_UWORD vintNum

){

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

if(vinEnBeenMasked(vintNum,curVM. VMVintEnMask < exIntEnMask >))//如果虚拟外部中断号<vintNum>对应的外部中断被当前应用虚拟机屏蔽，应返回VMK\_MASKED

return VMK\_ MASKED;

hardintNum.<ID: vintNum >∈ interruptMappingSet;// 中断映射表将虚拟中断号<vintNum>转换成硬件中断号，根据虚拟中断号vintNum在interruptMappingSet中查找对应的硬件中断号

if(!valid(hardintNum))// 无法将虚拟外部中断号映射到硬件中断号，或者转换后的硬件中断号大于等于硬件中断个数，应返回VMK\_INVALID\_STATE

return VMK\_ INVALID\_STATE;

if(outOfRang(hardintNum))// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止当前应用虚拟机指定的虚拟外部中断，并返回VMK\_OK。

return VMK\_ OK ;

}

### RK.4.4 vmkDisablePIC

T\_VMK\_ReturnCode vmkDisablePIC(

T\_UWORD vintNum

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

if(curVM. VMVintEnMask < isInitialized >==FALSE)// 如果当前应用虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

if(vinEnBeenMasked(vintNum,curVM. VMVintEnMask < exIntEnMask >))//如果虚拟外部中断号<vintNum>对应的外部中断被当前应用虚拟机屏蔽，应返回VMK\_MASKED

return VMK\_ MASKED;

hardintNum.<ID: vintNum >∈ interruptMappingSet;// 中断映射表将虚拟中断号<vintNum>转换成硬件中断号，根据虚拟中断号vintNum在interruptMappingSet中查找对应的硬件中断号

if(!valid(hardintNum))// 无法将虚拟外部中断号映射到硬件中断号，或者转换后的硬件中断号大于等于硬件中断个数，应返回VMK\_INVALID\_STATE

return VMK\_ INVALID\_STATE;

if(outOfRang(hardintNum))// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止当前应用虚拟机指定的虚拟外部中断，并返回VMK\_OK。

return VMK\_ OK ;

}

### RK.4.5 vmkSyscallEnablePIC

T\_VMK\_ReturnCode vmkSyscallEnablePIC(

T\_UWORD vintNum

){

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

if(vinEnBeenMasked(vintNum,curVM. VMVintEnMask < exIntEnMask >))//如果虚拟外部中断号<vintNum>对应的外部中断被当前应用虚拟机屏蔽，应返回VMK\_MASKED

return VMK\_ MASKED;

hardintNum.<ID: vintNum >∈ interruptMappingSet;// 中断映射表将虚拟中断号<vintNum>转换成硬件中断号，根据虚拟中断号vintNum在interruptMappingSet中查找对应的硬件中断号

if(!valid(hardintNum))// 无法将虚拟外部中断号映射到硬件中断号，或者转换后的硬件中断号大于等于硬件中断个数，应返回VMK\_INVALID\_STATE

return VMK\_ INVALID\_STATE;

if(outOfRang(hardintNum))// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

curVM. VMVintEnMask < isGlobalEnMask >=1;// 使能当前应用虚拟机指定的虚拟外部中断，并返回VMK\_OK。

return VMK\_ OK ;

}

### RK.4.6 vmkEnablePIC

T\_VMK\_ReturnCode vmkEnablePIC(

T\_UWORD vintNum

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

if(vinEnBeenMasked(vintNum,curVM. VMVintEnMask < exIntEnMask >))//如果虚拟外部中断号<vintNum>对应的外部中断被当前应用虚拟机屏蔽，应返回VMK\_MASKED

return VMK\_ MASKED;

hardintNum.<ID: vintNum >∈ interruptMappingSet;// 中断映射表将虚拟中断号<vintNum>转换成硬件中断号，根据虚拟中断号vintNum在interruptMappingSet中查找对应的硬件中断号

if(!valid(hardintNum))// 无法将虚拟外部中断号映射到硬件中断号，或者转换后的硬件中断号大于等于硬件中断个数，应返回VMK\_INVALID\_STATE

return VMK\_ INVALID\_STATE;

if(outOfRang(hardintNum))// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

curVM. VMVintEnMask < isGlobalEnMask >=1;// 使能当前应用虚拟机指定的虚拟外部中断，并返回VMK\_OK。

return VMK\_ OK ;

}

### RK.4.7 vmkSyscallSendServiceInt

T\_VMK\_ReturnCode vmkSyscallSendServiceInt(

T\_UWORD vmID,

T\_UWORD serviceIntNum

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是不是应用虚拟机，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

If(vm. VMStatus!= VMK\_VM\_DORMANT)// 如果<vmID>指定的应用虚拟机尚未被启动，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

if(vm. VMBasicInfo < schedType >!= VMK\_WAKEUP\_BASIC){// 如果<vmID>指定应用虚拟机的唤醒类型不是基本唤醒类型

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

return VMK\_ OK ;

}else{//是基本唤醒类型

If(!hasTimer()&&vm.VMStatus!= VMK\_VM\_ SUSPEND){// 是非定时并且虚拟机没有被挂起

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_ OK ;

}

If(hasTimer()&&vm.VMStatus!= VMK\_VM\_ SUSPEND){// 是定时并且虚拟机没有被挂起

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_ OK ;

}

}

}

### RK.4.8 vmkSendServiceInt

T\_VMK\_ReturnCode vmkSendServiceInt(

T\_UWORD vmID,

T\_UWORD serviceIntNum

){

if(vintNum <=0||vintNum>=31)// 如果<vintNum>不在虚拟外部中断号范围内，应返回VMK\_INVALID\_INDEX

return VMK\_INVALID\_INDEX;

vm.<ID:uwVMID>∈appVM∪kerVM;

if(vm==NULL){

return VMK\_INVALID\_ID;

If(vm. VMStatus!= VMK\_VM\_DORMANT)// 如果<vmID>指定的应用虚拟机尚未被启动，应返回VMK\_INVALID\_STATE

Return VMK\_INVALID\_STATE;

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_INVALID\_STATE。

return VMK\_INVALID\_STATE;

If(vm. VMBasicInfo < schedType >!= VMK\_WAKEUP\_BASIC){// 如果<vmID>指定核心虚拟机或应用虚拟机的唤醒类型不是基本唤醒类型

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

return VMK\_ OK ;

} else{//是基本唤醒类型

If(!hasTimer()&&vm.VMStatus!= VMK\_VM\_ SUSPEND){// 是非定时并且虚拟机没有被挂起

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_ OK ;

}

If(hasTimer()&&vm.VMStatus!= VMK\_VM\_ SUSPEND){// 是定时并且虚拟机没有被挂起

vm.sendPIC(serviceIntNum);// 应向<vmID>指定的应用虚拟机发送服务中断

vm.VMStatus= VMK\_VM\_READY;// 状态变为就绪态

return VMK\_ OK ;

}

}

}

### RK.4.9 vmkSyscallAckException

T\_VOID vmkSyscallAckException(

T\_VOID

)

{

}

### RK.4.10 vmkGetVint

vinitSet//需要投递的虚拟中断集合

vexceptionSet//虚拟异常中断集合

vtickSet//虚拟tick中断集合

vexnInitSet//虚拟外部中断集合

vserviceSet//虚拟服务中断集合

vint//虚拟中断

T\_VMK\_ReturnCode vmkGetVint(

T\_UWORD \*vintType,

T\_UWORD \*vintNum,

T\_UWORD \*entry

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(vintType==NULL|| vintNum ==NULL|| entry ==NULL){

Return VMK\_ INVALID\_ADDRESS;

}

if(curVM. VMVintEnMask < isInitialized>==FALSE)// 如果当前虚拟机的虚拟中断尚未初始化，应返回VMK\_ FAIL。

return VMK\_ FAIL;

if(vinitSet ==NULL)// 如果没有虚拟中断(异常、tick中断、外部中断、服务中断)需要投递，应返回VMK\_FAIL

return VMK\_ FAIL;

if(vexceptionSet!=NULL){// 如果有虚拟异常需要投递

vint ∈vexceptionSet;// 应获取需要投递的虚拟异常

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止虚拟中断

return VMK\_ OK ;

}

if(vtickSet!=NULL){// 如果有虚拟tick中断需要投递

vint ∈ vtickSet;// 应获取需要投递的tick中断

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止虚拟中断

return VMK\_ OK ;

}

if(vexnInitSet!=NULL){// 如果有虚拟外部中断需要投递

vint ∈ vexnInitSet;// 应获取需要投递的外部中断

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止虚拟中断

return VMK\_ OK ;

}

if(vserviceSet!=NULL){// 如果有虚拟服务中断需要投递

vint ∈ vserviceSet;// 应获取需要投递的服务中断

curVM. VMVintEnMask < isGlobalEnMask >=0;// 禁止虚拟中断

return VMK\_ OK ;

}

}

### RK.4.11 vmkSetIsrStackProtect

vintStack//中断栈

T\_VMK\_ReturnCode vmkSetIsrStackProtect(

T\_UWORD guardSize

){

If(calledFormInterruptProcess()){

Return VMK\_CALLED\_FROM\_ISR;

}

If(guardSize> curVM .vintStackSize())//如果<guardSize>大于中断栈可用的大小

Return VMK\_INVALID\_SIZE;

if(hasBeenProtected(curVM. vintStack))// 如果需要设置的系统栈空间已经被保护

return VMK\_INVALID\_ STATE;

if(guardSize%4096!=0)

return VMK\_INVALID\_ ALIGNED;

setProtectedStack(curVM. vintStack);// 应设置当前核心虚拟机或者IDLE虚拟机的系统栈保护，并返回VMK\_OK

return VMK\_ OK;

}

### RK.5.1 vmkSyscallEveWriteData

T\_VMK\_ReturnCode vmkSyscallEveWriteData(

T\_UDWORD\* time,

T\_UWORD type,

T\_VOID\* data,

T\_UWORD dataLen

){

If(curInterface ∉ curVM.canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

If(dataLen==0)// 如果<dataLen>表示的事件长度为0，应返回VMK\_INVALID\_SIZE

Return VMK\_INVALID\_SIZE;

If(haveExistedType(type))//type>为已存在的事件类型

{

Return VMK\_INVALID\_TYPE;

}

if(!checkValidAddress(time))// 如果<time>指向的存放事件发生时间的空间没有被映射，或空间是用户态程序不可读访问,应返回VMK\_FAIL

return VMK\_INVALID\_ADDRESS;

if(!checkValidAddress(data))// 如果< data >指向的存放事件发生时间的空间没有被映射，或空间是用户态程序不可读访问,应返回VMK\_FAIL

return VMK\_INVALID\_ADDRESS;

if(bufferLeftSize< dataLen)// 如果缓冲剩余空间小于<dataLen>时，应返回VMK\_INVALID\_SIZE

return VMK\_INVALID\_SIZE;

else{// 如果缓冲剩余空间不小于<dataLen>，应从缓冲区的写指针位置开始写数据，并返回VMK\_OK。

writeData(buffer,data);

return VMK\_OK;

}

}

### RK.5.2 vmkEveWriteData

T\_VMK\_ReturnCode vmkEveWriteData(

T\_UDWORD\* time,

T\_UWORD type,

T\_VOID\* data,

T\_UWORD dataLen

){

If(dataLen==0)// 如果<dataLen>表示的事件长度为0，应返回VMK\_INVALID\_SIZE

Return VMK\_INVALID\_SIZE;

If(haveExistedType(type))//type>为已存在的事件类型

{

Return VMK\_INVALID\_TYPE;

}

If(time==NULL||data==NULL)

Return VMK\_INVALID\_ADDRESS;

if(bufferLeftSize< dataLen)// 如果缓冲剩余空间小于<dataLen>时，应返回VMK\_INVALID\_SIZE

return VMK\_INVALID\_SIZE;

else{// 如果缓冲剩余空间不小于<dataLen>，应从缓冲区的写指针位置开始写数据，并返回VMK\_OK。

writeData(buffer,data);

return VMK\_OK;

}

}

### RK.5.3 vmkInitEvent

T\_VOID vmkInitEvent(

T\_VMK\_ConfigEvent \*eventConfig

){

}

### RK.6.1 vmkSyscallGetCPUTime

T\_VMK\_ReturnCode vmkSyscallGetCPUTime(

T\_UDWORD \*time

){

if(!checkValidAddress(time))// 如果<time>指向的存放事件发生时间的空间没有被映射，或空间是用户态程序不可读访问,应返回

return VMK\_INVALID\_ADDRESS;

\*time=curVM. time;//获取当前时间戳的值，并返回VMK\_OK

Return VMK\_OK;

}

### RK.6.2 vmkGetCPUTime

T\_VMK\_ReturnCode vmkGetCPUTime(

T\_UDWORD \*time

){

if(time==NULL)//

return VMK\_INVALID\_ADDRESS;

\*time=curVM. time;//获取当前时间戳的值，并返回VMK\_OK

Return VMK\_OK;

}

### RK.6.3 vmkSyscallGetKernelInfo

T\_VMK\_ReturnCode vmkSyscallGetKernelInfo(

T\_VMK\_VMKBasicInfo \*buf

){

if(!checkValidAddress(buf))// 如果< buf >指向的存放事件发生时间的空间没有被映射，或空间是用户态程序不可读访问,应返回

return VMK\_INVALID\_ADDRESS;

\* buf =curVM. VMKBasicInfo;获取内核基本信息，并返回VMK\_OK

Return VMK\_OK;

}

### RK.6.4 vmkGetKernelInfo

T\_VMK\_ReturnCode vmkGetKernelInfo(

T\_VMK\_VMKBasicInfo \*buf

){

If(buf==NULL)

return VMK\_INVALID\_ADDRESS;

\* buf =curVM. VMKBasicInfo;获取内核基本信息，并返回VMK\_OK

Return VMK\_OK;

}

### RK.6.5 vmkSyscallGetSystemTicks

T\_VMK\_ReturnCode vmkSyscallGetSystemTicks(

T\_UWORD\* ticks

){

if(!checkValidAddress(ticks))// 如果< ticks >指向的存放事件发生时间的空间没有被映射，或空间是用户态程序不可读访问,应返回

return VMK\_INVALID\_ADDRESS;

\*time=curVM. VMKBasicInfo< vmkTicks >;//获取系统自启动以来的tick数，并返回VMK\_OK

Return VMK\_OK;

}

### RK.6.6 vmkGetSystemTicks

T\_VMK\_ReturnCode vmkGetSystemTicks(

T\_UWORD\* ticks

){

if(ticks ==NULL)//

return VMK\_INVALID\_ADDRESS;

\*time=curVM. VMKBasicInfo< vmkTicks >;//获取系统自启动以来的tick数，并返回VMK\_OK

Return VMK\_OK;

}

//停止时间调度表

RK.2.38 T\_VMK\_ReturnCode vmkStopTTS(

T\_VOID

){

If(curVM.accessInterface∉ canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

If(vm.schdulingTableIsNotOn()){//如果当前没有启动时间调度表调度

Return VMK\_INVALID\_ STATE;

}

//如果当前启动了时间调度表调度，那么依次执行以下操作：

Vm.setCurSchedulingTableStatus(wating);// 设置被停止的当前时间调度表的状态为等待主时间帧；

Vm.stopCurScheduling();

setCurSchedulingTableWindowStatus(IDLE);// 设置时间调度表调度的当前时间窗口分区为IDLE分区；

callScheduling();//请求分区调度

return VMK\_OK;

}

//启动时间调度表

RK.2.39 T\_VMK\_ReturnCode vmkStartTTS(

T\_UWORD uwTTSID

){

If(curVM.accessInterface∉ canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

schedulingSST.<ID: uwpVMID>∈ schedulingTable //指定的时间调度表;

if(schedulingSST ==NULL){// 如果不存在<uwTTSID>指定的时间调度表，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

if(schedulingSST .isOnStarting()){//如果<uwTTSID>指定的时间调度表已经启动，应返回VMK\_INVALID\_STATE

return VMK\_INVALID\_ID STATE;

}else{

If(schedulingSST.timeManageStatus()==one-shot){

Vm.stopCurSchedulingTable();//停止当前时间调度表

schedulingSST.setCurSchedulingTableStatus(wating);// 设置<uwTTSID>指定的时间调度表的状态为等待主时间帧；

Vm.setCurSchedulingTableWindowStatus(IDLE);// 设置时间调度表调度的当前时间窗口分区为IDLE分区；

Vm.clearTimer();//清空定时器中断响应的间隔时间；

schedulingSST.setCurSchedulingTable(curSchedulingTable);// 设置<uwTTSID>指定的时间调度表为当前时间调度表；

Vm.setEnableTimerWindow();//使能定时器中断响应以当前分区时间窗口大小的间隔时间响应中断

Vm.timerOperation();//进行时间通知，包括系统tick时间、时间调度表窗口时间、时间调度表主帧时间

Vm.callScheduling();//请求分区调度

return VMK\_OK;

}else{

Vm.stopCurSchedulingTable();//停止当前时间调度表

schedulingSST.setCurSchedulingTableStatus(wating);// 设置<uwTTSID>指定的时间调度表的状态为等待主时间帧；

setCurSchedulingTableWindowStatus(IDLE);// 设置时间调度表调度的当前时间窗口分区为IDLE分区；

Vm.setCurTimerWindowType(watingTimer);// 设置当前时间窗口通知类型为等待主时间帧；

schedulingSST.setCurSchedulingTable(curSchedulingTable);// 设置<uwTTSID>指定的时间调度表为当前时间调度表；

Vm.timerOperation();/进行时间调度表主帧的时间通知；

Vm.callScheduling();//请求分区调度

return VMK\_OK;

}

}

}

RK.6.1 T\_VMK\_ReturnCode vmkInitializeVint(

T\_VMK\_VMVintEnMask \*tpVIntEnMask,

T\_VMK\_VINT\_HANDLER tpHandler

){

If(checkValidInterruptAdd(tpVIntEnMask)){// 指向的虚拟中断屏蔽码空间没有被映射、指向的虚拟中断屏蔽码空间不可读写访问

Return VMK\_INVALID\_ADDRESS;

}

If(checkValidInterruptAdd(tpHandler)){// 表示的地址没有被映射、表示的地址不可访问

Return VMK\_INVALID\_ADDRESS;

}

If(vm.tpInterruptIsInitialed()){//如果当前分区的虚拟中断已经初始化

Return VMK\_INVALID\_STATE;

}

If(checkNotValidAddress(tpHandler)){// 如果<tpHandler>表示的地址未按4字节对齐，应返回VMK\_INVALID\_ALIGNED

Return VMK\_INVALID\_ALIGNED;

}

Vm.instalInterruptEntryandMask(tpVIntEnMask, tpHandler);// 根据<tpVIntEnMask>和<tpHandler>安装当前分区的虚拟中断入口程序和屏蔽码

Vm.setTPInterruptStatus(TRUE);// 设置当前分区的虚拟中断初始化标识为TRUE

Return VMK\_OK;

}

PICset//虚拟中断号范围集合

PICHardwareset//实际的中断号范围集合

PICNotAllowed//当前分区屏蔽的外部中断集合

PICMapSet//中断映射表集合

RK.6.2 T\_VMK\_ReturnCode vmkDisablePIC(

T\_UWORD uwVIntNum

){

If(uwVIntNum∉PICset){// 如果<uwVIntNum>不在虚拟中断号范围内，应返回VMK\_INVALID\_INDEX

Return VMK\_INVALID\_INDEX;

}

Pic.<ID: uwVIntNum >∈PICNotAllowed;

If(pic!=NULL){// 如果虚拟中断号<uwVIntNum>对应的外部中断被当前分区屏蔽，应返回VMK\_MASKED

Return VMK\_MASKED;

}

picHardNum.<ID: uwVIntNum>∈PIDMapSet; 根据用户所配置的中断映射表将虚拟中断号<uwVIntNum>转换成硬件中断号

if(checkNoValid(picHardNum)){//如果无法将虚拟中断号映射到硬件中断号或者转换后的硬件中断号大于等于硬件中断个数

return VMK\_INVALID\_STATE;

}

If(picHardNum∉ PICHardwareset){// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内

Return VMK\_INVALID\_INDEX;

}

If(vm.disablePIDSuccessed()){//如果成功禁止当前分区的指定虚拟外部中断

Return VMK\_OK;

}

}

RK.6.3 T\_VMK\_ReturnCode vmkEnablePIC(

T\_UWORD uwVIntNum

){

If(uwVIntNum∉PICset){// 如果<uwVIntNum>不在虚拟中断号范围内，应返回VMK\_INVALID\_INDEX

Return VMK\_INVALID\_INDEX;

}

Pic.<ID: uwVIntNum >∈PIDNotAllowed;

If(pic!=NULL){// 如果虚拟中断号<uwVIntNum>对应的外部中断被当前分区屏蔽，应返回VMK\_MASKED

Return VMK\_MASKED;

}

picHardNum.<ID: uwVIntNum>∈PIDMapSet; 根据用户所配置的中断映射表将虚拟中断号<uwVIntNum>转换成硬件中断号

if(changeInValid()){//如果无法将虚拟中断号映射到硬件中断号或者转换后的硬件中断号大于等于硬件中断个数

return VMK\_INVALID\_STATE;

}

If(picHardNum∉ PICHardwareset){// 如果用户所配置中断映射表中硬件中断号不在实际的中断号范围内

Return VMK\_INVALID\_INDEX;

}

If(enablePIDSuccessed()){//如果成功禁止当前分区的指定虚拟外部中断

Return VMK\_OK;

}

}

waitingTimeQueue/差分时间等待队列

RK.6.4 T\_VMK\_ReturnCode vmkSendServiceInt(

T\_UWORD uwVMID,

T\_UWORD uwServiceIntNum

){

If(vmkStopVM ∉ canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

vm.<ID: uwpVMID>∈appVM;

if(vm==NULL){// 如果<uwVMID>表示的分区不是应用分区，应返回VMK\_INVALID\_ID

return VMK\_INVALID\_ID;

}

If(uwVIntNum∉PICset){// 如果<uwVIntNum>不在虚拟中断号范围内，应返回VMK\_INVALID\_INDEX

Return VMK\_INVALID\_INDEX;

}

If(vm.isNotOnStarting()){//如果<uwVMID>指定的分区尚未被启动

Return VMK\_INVALID\_STATE

}

If(vm.tpInterruptIsNotInitialed()){//如果当前分区的虚拟中断已经初始化

Return VMK\_INVALID\_STATE;

}

If(vm.awakenType()!=VMK\_WAKEUP\_BASIC){// 定的分区的唤醒类型不是VMK\_WAKEUP\_BASIC

sendPIC(vm);// 向<uwVMID>指定的分区发送服务请求中断

return VMK\_OK;

}else{

If(waitingTime==0){// 并且等待时间为0

sendPIC(vm);// 向<uwVMID>指定的分区发送服务请求中断

awaken(vm);// 唤醒该分区

if(vm.isNotHungup()){//如果该分区没有被挂起，应设置该分区为就绪态

vm.setCurStatus(READY);

}

callScheduling();//请求分区调度

return VMK\_OK;

}

If(waitingTime<=ENDTIME){// 并且等待时间还未超时

sendPIC(vm);// 向<uwVMID>指定的分区发送服务请求中断

waitingTimeQueue =waitingTimeQueue-vm;//将该分区从差分时间等待队列上移除

awaken(vm);// 唤醒该分区

if(vm.isNotHungup()){//如果该分区没有被挂起，应设置该分区为就绪态

vm.setCurStatus(READY);

}

callScheduling();//请求分区调度

return VMK\_OK;

}

}

}

RK.6.5 T\_VMK\_ReturnCode vmkAckException(

T\_VOID

){

}

leftBufferSize\\剩余缓冲空间

RK.8.1 T\_VMK\_ReturnCode vmkEveWriteData(

T\_UDWORD\* dwTime,

T\_UWORD uwType,

T\_VOID\* vpData,

T\_UWORD uwDataLen

){

If(vmkStopVM ∉ canAccessInterfaceSet){// 当前分区是否有权限调用本接口

Return VMK\_INVALID\_USER;

}

If(uwDataLen==0){

Return VMK\_INVALID\_SIZE;

}

If(notValidCheck(dwTime){

Return VMK\_INVALID\_ADDRESS;

}

If(notValidCheck(vpData){

Return VMK\_INVALID\_ADDRESS;

}

If(uwType<6){

Return VMK\_INVALID\_TYPE;

}

If(leftBufferSize< uwDataLen){

Return VMK\_INVALID\_SIZE;

}else{

writeDataInBuffer();

return VMK\_OK;

}

}

RK.9.1 T\_VMK\_ReturnCode vmkGetCPUTime(

T\_UDWORD \*udwpTime

){

If(notValidCheck(dwTime){// 如果<udwpTime>指向的地址空间没有被映射、地址空间不可读写访问

Return VMK\_INVALID\_ADDRESS;

}

If(successGetCurTime()){

Return VMK\_OK;

}

}

RK.9.2 T\_VMK\_ReturnCode vmkGetKernelInfo(

T\_VMK\_VMKBasicInfo \*tpBuf

){

If(notValidCheck(tpBuf){// 如果< tpBuf >指向的地址空间没有被映射、地址空间不可读写访问

Return VMK\_INVALID\_ADDRESS;

}

If(successGetKernelInfo()){

Return VMK\_OK;

}

}

RK.9.3 T\_VMK\_ReturnCode vmkGetSystemTicks(

T\_UWORD\* ticks

){

If(notValidCheck(ticks){// 如果< ticks >指向的地址空间没有被映射、地址空间不可读写访问

Return VMK\_INVALID\_ADDRESS;

}

If(successGetTicksInfo()){

Return VMK\_OK;

}

}

}

endOf APPVMADT

∀∃∧∨¬→↔⊕⇔⇒≠╞├⊂⊆⊄∈∉∅φ∩∪∪∩Γ≈≡≥≤⊙∴∴∵∷∈≠≡≌±÷×⊥∞∘•◁αβωε

Αα [alpha]

Ββ [beta]

Γγ [gamma]

Δδ [delta]

Εε [epsilon]

Ζζ [zeta]

Ηη [eta]

Θθ [theta]

Ιι [iota]

Κκ [kappa]

Λλ [lamda]

Μμ [mu]

Νν [nu]

Ξξ [xiksai]

Οο [omicron]

Ππ [pi]

Ρρ [rhorou]

Σσς [sigma]

Ττ [tau]

Υυ [upsilon]

Φφ [phi]

Χχ [chikai]

Ψψ [psipsai]

Ωω [omega]