This HM receives through a UNIX socket a command, here identified as **`0x00000101 (tag 0xD7E3D316D0D88C47)`**. This command results immediately in a **`vioHmStartVMs`** call, which in turn results in a tentative start monitoring action for the `vmraix1` VM. We also note that the associated hexadecimal tag value is the same as in the related health library log records. But, even if the HA monitoring is enabled at KSYS, the actual VM monitoring does not start and remains marked as `STARTING` at the HSDB level because the VM Agent *has not yet been discovered* by our HM instance. This is expected because we did not install the VM Agent fileset on the `vmraix1` VM, so there is no appropriate counterpart endpoint there to interact with the HM instance.

Install the fileset and see what happens. As described “HM startup sequence” on page 159, the running HM instances are already broadcasting continuously `Hello` messages, waiting for the agents on the managed VMs to respond. A complete HM to VMM handshake flow is detailed in “HM-VMM communication establishment”.

HM-VMM communication establishment

We covered in “VM Agent activation” on page 164 the blind startup sequence of the VMM daemon for the case of an isolated VM when VM monitoring is not enabled at the KSYS level and the VM does not have the HA client adapters created. We saw in that case how VMM enters a steady state loop where it is looking continuously for the HA client adapters on which to listen for `Hello` messages.

Here we present the case where the VMM agent is deployed on a VM enabled for monitoring and discovered by KSYS after enablement, as described in 2.3.1, “Discovery of a VM that is enabled for monitoring” on page 203. The focus here is on the handshake protocol between the VMM and the HMs, but we are also interested in the HSDB updates about the VM status. The running HMs on the VIOS side are broadcasting repeatedly `Hello` messages and waiting for the possible agents on the managed VMs to respond, as described in “HM activation” on page 150.

In Example 2-190, we filter the output of the **`procstack`** command on our test VM, `vmraix1`, so that we can easily identify later the thread type for each TID.

Example 2-190 VMM threads for the HM to VMM communication establishment scenario

```
# uname -uL
IBM,02212424W 5 vmraix1
[vmraix1:root:/var/ksys/log/hmvmmhs:] lssrc -s ksys_vmm
Subsystem         Group            PID          Status
 ksys_vmm                          10682746     active
# procstack 10682746|egrep "tid|main|threadMain"
---------- tid# 17563989 (pthread ID:      1) ----------
0x10007e9c  main(??, ??) + 0xf1c
---------- tid# 19202453 (pthread ID:   1286) ----------
0x100e6110  UnixSocket::threadMain()(??) + 0x50
---------- tid# 20513041 (pthread ID:   1029) ----------
0x102cc564  VmMonitor::threadMain()(??) + 0x3c4
---------- tid# 27918627 (pthread ID:   2828) ----------
0x1007185c  HvncpSender::threadMain()(??) + 0x8fc
---------- tid# 23724513 (pthread ID:   2571) ----------
0x10083e24  HvncpReceiver::threadMain()(??) + 0x6a4
---------- tid# 17826247 (pthread ID:   2314) ----------
0x1008ee44  VlanCommunication::threadMain()(??) + 0x44
---------- tid# 17760557 (pthread ID:   2057) ----------
0x1007185c  HvncpSender::threadMain()(??) + 0x8fc
---------- tid# 24183173 (pthread ID:   1800) ----------
```