NAME

hv_vss - Hyper-V Volume Shadow Copy Service API

SYNOPSIS

```
#include <dev/hyperv/hv_snapshot.h>

#define VSS_SUCCESS             0x00000000
#define VSS_FAIL        0x00000001

enum hv_vss_op_t {
        HV_VSS_NONE = 0,
        HV_VSS_CHECK,
        HV_VSS_FREEZE,
        HV_VSS_THAW,
        HV_VSS_COUNT
};

struct hv_vss_opt_msg {
        uint32_t  opt;          /* operation */
        uint32_t  status;       /* 0 for success, 1 for error */
        uint64_t  msgid;        /* an ID used to identify the transaction */
        uint8_t         reserved[48];   /* reserved values are all zeroes */
};
```

DESCRIPTION

The freeze or thaw functionality of application is important to guarantee the application consistent backup. On windows platform, VSS is defined to do live backup. But for VM guest running on Hyper-V, the corresponding VSS is not defined yet. For example, a running database server instance, it knows when the applications' freeze/thaw should start or finish. But it is not aware of the freeze/thaw notification from Hyper-V host. The **hv_vss** is designed to notify application freeze/thaw request. Thus, it plays a role of broker to forward the freeze/thaw command from Hyper-V host to userland application if it registered VSS service on FreeBSD VM, and sends the result back to Hyper-V host.

Generally, hv_vss_daemon(8) takes the responsibility to freeze/thaw UFS file system, and it is automatically launched after system boots. When Hyper-V host wants to take a snapshot of the FreeBSD VM, it will first send VSS capability check to FreeBSD VM. The **hv_vss** received the request and forward the request to userland application if it is registered. Only after **hv_vss** received the VSS_SUCCESS response from application, the hv_vss_daemon(8) will be informed to check whether file system freeze/thaw is supported. Any error occurs during this period, **hv_vss** will inform Hyper-V host that VSS is not supported. In addition, there is a default timeout limit before sending response to

Hyper-V host. If the total response time from application and hv_vss_daemon(8) exceeds this value, timeout will occurs and VSS unsupported is responsed to Hyper-V host.

After Hyper-V host confirmed the FreeBSD VM supports VSS, it will send freeze request to VM, and **hv_vss** will first forward it to application. After application finished freezing, it should inform **hv_vss** and file system level freezing will be triggered by hv_vss_daemon(8). After all freezing on both application and hv_vss_daemon(8) were finished, the **hv_vss** will inform Hyper-V host that freezing is done. Of course, there is a timeout limit as same as VSS capability is set to make sure freezing on FreeBSD VM is not hang. If there is any error occurs or timeout happened, the freezing is failed on Hyper-V side.

Hyper-V host will send thaw request after taking the snapshot, typically, this period is very short in order not to block the running application. **hv_vss** firstly thaw the file system by notifying hv_vss_daemon(8), then notifies user registered application. There is also a timeout check before sending response to Hyper-V host.

All the default timeout limit used in VSS capability check, freeze or thaw is the same. It is 15 seconds currently.

NOTES

hv_vss only support UFS currently. If any of file system partition is non UFS, the VSS capability check will fail. If application does not register VSS, **hv_vss** only support backup for file system level consistent. The device should be closed before it was opened again. If you want to simultaneously open "/dev/hv_appvss_dev" two or more times, an error (-1) will be returned, and errno was set.

If hv_vss_daemon(8) was killed after system boots, the VSS functionality will not work.

EXAMPLES

The following is a complete example which does nothing except for waiting 2 seconds when receiving those notifications from **hv_vss**

```
#include <string.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/param.h>
#include <sys/ucred.h>
#include <sys/mount.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
```

```
#include <poll.h>
#include <stdint.h>
#include <syslog.h>
#include <errno.h>
#include <err.h>
#include <fcntl.h>
#include <ufs/ffs/fs.h>
#include <paths.h>
#include <sys/ioccom.h>
#include <dev/hyperv/hv_snapshot.h>

#define UNDEF_FREEZE_THAW	(0)
#define FREEZE			(1)
#define THAW			(2)
#define CHECK			(3)

#define	VSS_LOG(priority, format, args...) do	{				\
		if (is_debugging == 1) {				\
			if (is_daemon == 1)				\
				syslog(priority, format, ## args);	\
			else						\
				printf(format, ## args);		\
		} else {						\
			if (priority < LOG_DEBUG) {			\
				if (is_daemon == 1)			\
					syslog(priority, format, ## args);	\
				else					\
					printf(format, ## args);	\
			}						\
		}							\
	} while(0)

#define CHECK_TIMEOUT		1
#define CHECK_FAIL		2
#define FREEZE_TIMEOUT		1
#define FREEZE_FAIL		2
#define THAW_TIMEOUT		1
#define THAW_FAIL		2

static int is_daemon	= 1;
```

```
static int is_debugging     = 0;
static int simu_opt_waiting = 2; // seconds

#define GENERIC_OPT(TIMEOUT, FAIL)                                      \
        do {                                                            \
                sleep(simu_opt_waiting);                                \
                if (opt == CHECK_TIMEOUT) {                             \
                        sleep(simu_opt_waiting * 10);                   \
                        VSS_LOG(LOG_INFO, "%s timeout simulation\n",    \
                            __func__);                                  \
                        return (0);                                     \
                } else if (opt == CHECK_FAIL) {                         \
                        VSS_LOG(LOG_INFO, "%s failure simulation\n",    \
                            __func__);                                  \
                        return (CHECK_FAIL);                            \
                } else {                                                \
                        VSS_LOG(LOG_INFO, "%s success simulation\n",    \
                            __func__);                                  \
                        return (0);                                     \
                }                                                       \
        } while (0)

static int
check(int opt)
{
        GENERIC_OPT(CHECK_TIMEOUT, CHECK_FAIL);
}

static int
freeze(int opt)
{
        GENERIC_OPT(FREEZE_TIMEOUT, FREEZE_FAIL);
}

static int
thaw(int opt)
{
        GENERIC_OPT(THAW_TIMEOUT, THAW_FAIL);
}
```

```
static void usage(const char* cmd) {
        fprintf(stderr,
          "%s -f <0|1|2>: simulate app freeze."
          " 0: successful, 1: freeze timeout, 2: freeze failed\n"
          " -c <0|1|2>: simulate vss feature check"
          " -t <0|1|2>: simulate app thaw."
          " 0: successful, 1: freeze timeout, 2: freeze failed\n"
          " -d : enable debug mode\n"
          " -n : run this tool under non-daemon mode\n", cmd);
}

int
main(int argc, char* argv[]) {
        int ch, freezesimuop = 0, thawsimuop = 0, checksimuop = 0, fd, r, error;
        uint32_t op;
        struct pollfd app_vss_fd[1];
        struct hv_vss_opt_msg  userdata;

        while ((ch = getopt(argc, argv, "f:c:t:dnh")) != -1) {
                switch (ch) {
                case 'f':
                        /* Run as regular process for debugging purpose. */
                        freezesimuop = (int)strtol(optarg, NULL, 10);
                        break;
                case 't':
                        thawsimuop = (int)strtol(optarg, NULL, 10);
                        break;
                case 'c':
                        checksimuop = (int)strtol(optarg, NULL, 10);
                        break;
                case 'd':
                        is_debugging = 1;
                        break;
                case 'n':
                        is_daemon = 0;
                        break;
                case 'h':
                default:
                        usage(argv[0]);
                        exit(0);
```

```
		}
	}

	openlog("APPVSS", 0, LOG_USER);
	/* Become daemon first. */
	if (is_daemon == 1)
		daemon(1, 0);
	else
		VSS_LOG(LOG_DEBUG, "Run as regular process.\n");

	VSS_LOG(LOG_INFO, "HV_VSS starting; pid is: %d\n", getpid());

	fd = open(VSS_DEV(APP_VSS_DEV_NAME), O_RDWR);
	if (fd < 0) {
		VSS_LOG(LOG_ERR, "Fail to open %s, error: %d %s\n",
		    VSS_DEV(APP_VSS_DEV_NAME), errno, strerror(errno));
		exit(EXIT_FAILURE);
	}
	app_vss_fd[0].fd     = fd;
	app_vss_fd[0].events = POLLIN | POLLRDNORM;

	while (1) {
		r = poll(app_vss_fd, 1, INFTIM);

		VSS_LOG(LOG_DEBUG, "poll returned r = %d, revent = 0x%x\n",
		    r, app_vss_fd[0].revents);

		if (r == 0 || (r < 0 && errno == EAGAIN) ||
		    (r < 0 && errno == EINTR)) {
			/* Nothing to read */
			continue;
		}

		if (r < 0) {
			/*
			 * For poll return failure other than EAGAIN,
			 * we want to exit.
			 */
			VSS_LOG(LOG_ERR, "Poll failed.\n");
			perror("poll");
```

```
		exit(EIO);
}

/* Read from character device */
error = ioctl(fd, IOCHVVSSREAD, &userdata);
if (error < 0) {
		VSS_LOG(LOG_ERR, "Read failed.\n");
		perror("pread");
		exit(EIO);
}

if (userdata.status != 0) {
		VSS_LOG(LOG_ERR, "data read error\n");
		continue;
}

op = userdata.opt;

switch (op) {
case HV_VSS_CHECK:
		error = check(checksimuop);
		break;
case HV_VSS_FREEZE:
		error = freeze(freezesimuop);
		break;
case HV_VSS_THAW:
		error = thaw(thawsimuop);
		break;
default:
		VSS_LOG(LOG_ERR, "Illegal operation: %d\n", op);
		error = VSS_FAIL;
}
if (error)
		userdata.status = VSS_FAIL;
else
		userdata.status = VSS_SUCCESS;
error = ioctl(fd, IOCHVVSSWRITE, &userdata);
if (error != 0) {
		VSS_LOG(LOG_ERR, "Fail to write to device\n");
		exit(EXIT_FAILURE);
```

```
                        } else {
                                VSS_LOG(LOG_INFO, "Send response %d for %s to kernel\n",
                                  userdata.status, op == HV_VSS_FREEZE ? "Freeze" :
                                  (op == HV_VSS_THAW ? "Thaw" : "Check"));
                        }
                }
                return 0;
        }
```

SEE ALSO

hv_utils(4), hv_vss_daemon(8)

HISTORY

The daemon was introduced in October 2016 and developed by Microsoft Corp.

AUTHORS

FreeBSD support for **hv_vss** was first added by Microsoft BSD Integration Services Team <*bsdic@microsoft.com*>.